UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

CENTRO DE FÍSICA APLICADA Y TECNOLOGÍA AVANZADA

IMPLEMENTACIÓN DE COMPUTADORA DE A BORDO PARA PROTOTIPO DE ROBOT MÓVIL TIPO ROVER

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciado en Tecnología

PRESENTA:
Irving Fernando Galindo Ramírez

DIRECTOR DE TESIS:
Dr. Rafael Guadalupe Chávez Moreno

Querétaro, Qro., 2019

Resumen

En la actualidad, existen diferentes agencias espaciales que van explorando cada vez más lugares del sistema solar, Marte es uno de los destinos de exploración con más misiones dentro del sistema solar; esto se debe a su relativa cercanía a la tierra y a la evidencia que sugiere que en algún momento pudo haber habido vida en ese planeta. Los rovers son la principal herramienta usada para la exploración y el análisis de la superficie marciana.

Un rover está compuesto por diferentes clases de sistemas, uno de ellos es la computadora de a bordo, que se encarga de procesar las señales dentro del rover, coordinar el funcionamiento de los subsistemas y llevar a cabo la comunicación con la estación terrena.

En este trabajo se presentan los pasos realizados para la implementación de la computadora de a bordo: el análisis de sus requerimientos para seleccionar sus componentes, el proceso de diseño de los comandos utilizados para el control de los subsistemas del rover, las diferentes etapas en el desarrollo del programa principal, el proceso para realizar la adquisición de datos de los sensores, el desarrollo de la interfaz de usuario para la estación terrena, el proceso de manufactura de la placa de circuito impreso y las pruebas realizadas para validar el funcionamiento de la computadora de a bordo.

Este trabajo forma parte del proyecto “Diseño y construcción de un prototipo de robot móvil semiautónomo para exploración de terrenos similares a la superficie marciana”, ese proyecto busca producir capital humano calificado en el área espacial, mediante el desarrollo de un prototipo de rover, que eventualmente pueda participar en la URC.

Reconocimientos

A la Unidad de Alta Tecnología y al Laboratorio Nacional de Ingeniería Espacial y Automotriz (LN-INGEA CONACyT 232660), por brindarme acceso a sus instalaciones y al equipo utilizado durante el desarrollo del proyecto.

A mi tutor, el Dr. Rafael G. Chávez Moreno, por todo lo que me enseñó a lo largo de mi formación académica, además del apoyo que me brindó durante la realización de este proyecto.

Al CONCyTEQ, a la Universidad Politécnica de Santa Rosa Jauregui y al M. en I. Javier Ceballos Olivares, por el apoyo brindado para la obtención de recursos a través del Programa de apoyo a nuevos talentos científicos y tecnológicos 2018.

A mis sinodales el Dr. Carlos Romo Fuentes, el Dr. Ángel Luis Rodríguez Morales, el Dr. José Alberto Ramírez Aguilar, el Dr. Jorge Alfredo Ferrer Pérez y el Dr. Rafael G. Chávez Moreno, por las pláticas que hemos tenido y que me han hecho reflexionar sobre las opciones que tengo disponibles para la siguiente etapa de mi vida.

A los integrantes del departamento aeroespacial de la UAT, por los esfuerzos que realizan día a día para apoyar en el desarrollo y el fortalecimiento del sector espacial del país.

A los profesores de la Licenciatura en Tecnología, por haber participado en mi formación.

Agradecimientos

A mis papás, por apoyarme incondicionalmente a lo largo de mi vida, por el cariño que me han brindado, por creer en mí a pesar de los tropiezos que he tenido, por todo lo que me han enseñado, por todo el cariño que les tengo.

A mis hermanos y mi familia, por el tiempo que compartimos mientras crecíamos, por soportar mi temperamento, por los buenos momentos, por los malos momentos, por todas las lecciones de vida que he aprendido a su lado.

A mi sobrino, por brindarme una razón extra para esforzarme y seguir adelante.

A mis amigos, por las desveladas para estudiar, por las desveladas para festejar, por las incontables horas que hemos pasado juntos y que me han ayudado a convertirme en la persona que soy.

Índice general

Índice de tablas

Índice de figuras

Lista de abreviaciones

OBC	On Board Computer
URC	University Rover Challenge
COTS	Commercial Off-The-Shelf
SBC	Single Board Computer
FODA	Fortalezas Oportunidades Debilidades y Amenazas
UNAM	Universidad Nacional Autónoma de México
UAT	Unidad de Alta Tecnología
UPSRJ	Universidad Politécnica de Santa Rosa Jáuregui
LN-INGEA	Laboratorio Nacional de Ingeniería Espacial y Automotriz
CONACyT	Consejo Nacional de Ciencia y Tecnología
CONCyTEQ	Consejo de Ciencia y Tecnología del Estado de Querétaro
NASA	National Aeronautics and Space Administration
MER	Mars Exploration Rover
MDRS	Mars Desert Research Station
CPU	Central Processing Unit
RAM	Random Access Memory
DRAM	Dynamic Random Access Memory
SRAM	Static Random Access Memory
PROM	Programmable Read-Only Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPIO	General Purpose Input/Output
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
GPU	Graphics Processing Unit

AV	Audio y Video
HDMI	High-Definition Multimedia Interface
USB	Universal Serial Bus
RTOS	Real-Time Operating System
RTS	Real-Time System
SRTS	Soft Real-Time System
HRTS	Hard Real-Time System
POO	Programación Orientada a Objetos
PVM	Python Virtual Machine
SPI	Serial Peripheral Interface
SCLK	Serial Clock
MOSI	Master Output/Slave Input
MISO	Master Input/Slave Output
CS	Chip Select
SS	Slave Select
I2C	Inter-Integrated Circuit
MSB	Most Significant Bit
GPS	Global Positioning System
MEMS	Microelectromechanical Systems
LAN	Local Area Network
LRC	Longitudinal Redundancy Check
SSH	Secure Shell
SDLC	Software Development Life Cycle
PNV	Plan de Navegación
PCB	Printed Circuit Board

1 Introducción

En este trabajo se presenta la implementación de una computadora de a bordo (OBC por sus siglas en inglés), la cual, forma parte del proyecto “diseño y construcción de un prototipo de robot móvil semiautónomo para exploración de terrenos similares a la superficie marciana” que se está desarrollando como un esfuerzo en conjunto entre la Unidad de Alta Tecnología de la Facultad de Ingeniería de la UNAM y la Universidad Politécnica de Santa Rosa Jauregui. Uno de los objetivos de esta colaboración es fomentar la formación de capital humano calificado en las áreas requeridas para el desarrollo del prototipo de rover y, en particular, en el área espacial; otro de los objetivos a mediano plazo es llevar el rover a participar en la URC.

Un rover está compuesto de varias clases de subsistemas, entre esos se encuentra la computadora de a bordo que es la que se encarga de procesar la información proveniente de los subsistemas que integran al rover, coordinar la distribución de información entre sistemas, aplicar los criterios de control necesarios para ejecutar las tareas asignadas, además de encargarse de la comunicación entre el rover y la estación terrena.

La implementación de la OBC se realizó con el uso de componentes COTS que están al alcance del público en general, con esto se busca fortalecer la idea de que no es necesario tener un presupuesto inmenso para desarrollar este tipo de proyectos. Para la elección de los componentes, se tomaron en cuenta los lineamientos de la URC.

En este trabajo se presenta el proceso de análisis de los requerimientos de la OBC para la elección de una computadora de placa única (SBC por sus siglas en inglés) y los demás componentes, el proceso de diseño de los comandos utilizados por la OBC para el control de los subsistemas del rover, el desarrollo del código (con el lenguaje de programación Python) del programa principal de la OBC, el proceso para realizar la adquisición de datos de los sensores, el desarrollo de la interfaz de usuario para la estación terrena (mediante el uso de LabVIEW), el proceso de manufactura de la placa de circuito impreso y las diversas pruebas realizadas para validar el funcionamiento de la OBC.

2 Objetivos

2.1 General

Implementar una computadora de a bordo que permita recibir comandos de manera inalámbrica desde una estación terrena y controlar el funcionamiento de los subsistemas simulados de un prototipo de robot móvil tipo rover orientado a la exploración de un escenario análogo a Marte. Mediante el análisis de los requerimientos de la computadora de a bordo, se establecerán sus componentes y sus funciones, además, se ejecutarán pruebas para evaluar su correcto funcionamiento.

2.2 Específicos

- Realizar el análisis de los requerimientos de la OBC.
- Elegir una computadora de placa única que sea adecuada para la OBC.
- Elegir los sensores que permitan obtener el estado de operación.
- Identificar e implementar un protocolo de comunicación idóneo para la transferencia de información entre la OBC y los subsistemas simulados del rover.
- Desarrollar una interfaz de usuario para la estación terrena.
- Implementar un sistema de comunicación inalámbrica con la estación terrena.
- Diseñar y ejecutar las pruebas que permitan evaluar el desempeño de la OBC.

2.3 Hipótesis

Es posible implementar una computadora de a bordo de bajo costo, mediante el uso de componentes que se encuentran al alcance del público en general (COTS).

2.4 Motivación

El sector espacial va cobrando cada vez más relevancia para el país ya que realiza aportes en diversos sectores de la sociedad, como el estudio del cambio climático, la prevención de desastres naturales, el apoyo a la seguridad nacional y la ampliación de los enlaces de comunicación satelital que brindan a la población servicios que simplifican sus actividades diarias, además de también contribuir en la formación de empleos.

El sector espacial mexicano está integrado por actores provenientes de la triple hélice: industria, academia y gobierno. Es aquí donde se encuentra el mayor

motivante para llevar a cabo el proyecto, el fomento de la formación de capital humano capacitado en el área espacial, no solo de los miembros que participan en el proyecto del rover, sino de la comunidad universitaria en general. Lo anterior se basa en la idea de que los estudiantes, al darse cuenta que no es necesario tener una formación especializada en esta área, encontrarán menos intimidante el integrarse a proyectos de esta índole, lo cual propiciará su especialización y apoyará al crecimiento del sector espacial del país; ese es uno de los puntos importantes mencionados en el análisis FODA realizado en el Plan de Órbita 2.0 [1], que explica que, en el país existe una cantidad considerable de recursos humanos, solo hace falta que se especialicen para poder apoyar a la academia y a la industria con el fortalecimiento del sector espacial de México.

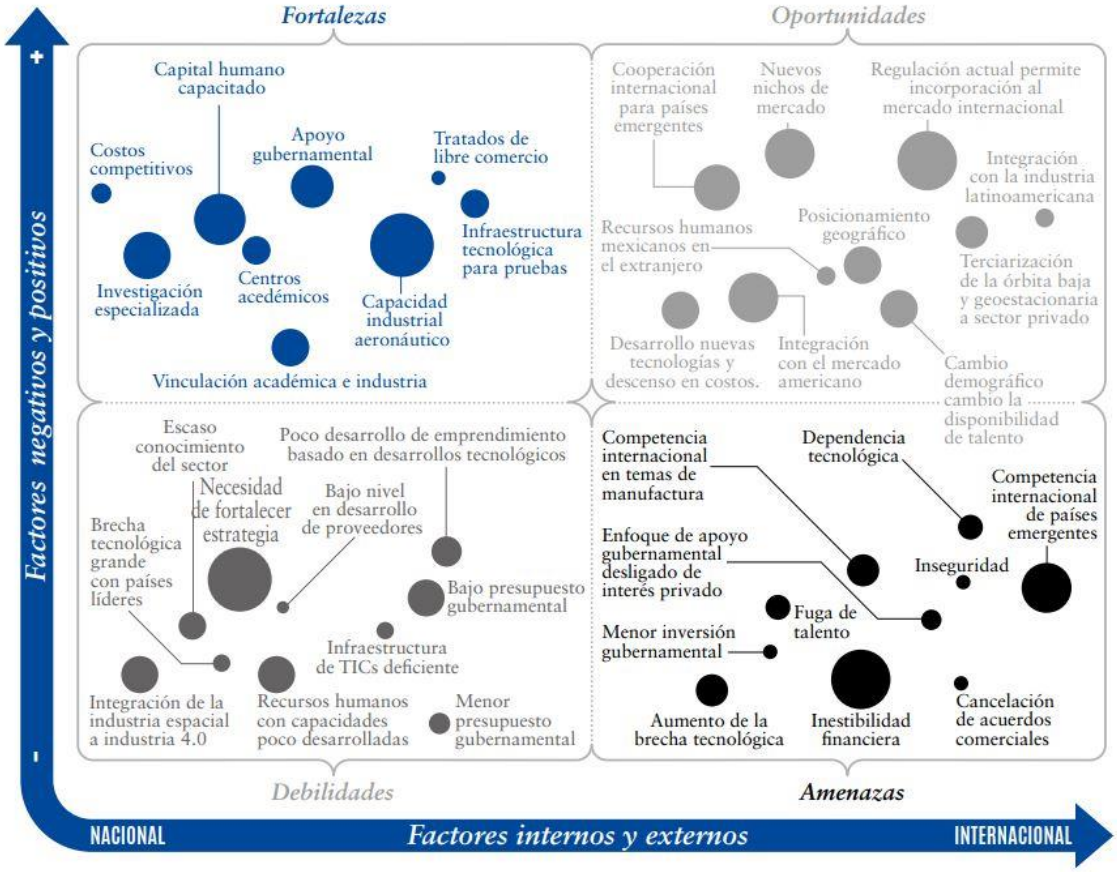


Figura 2.1 Representación gráfica del análisis FODA presente en el Plan de Órbita 2.0 [Crédito Agencia Espacial Mexicana]

3 Antecedentes

3.1 Exploración espacial

Existen diversas maneras de explorar el espacio, una de las más populares es la astronomía, sin embargo, se centrará la atención en la exploración espacial mediante misiones en las que se entra en contacto físico con el cuerpo celeste en estudio. La exploración y el estudio del espacio han ayudado a comprender mejor las leyes que rigen el universo, realizar avances tecnológicos, crear nuevas industrias y muchos otros aspectos que han beneficiado a la humanidad.

Un tema muy debatido en la exploración espacial es la viabilidad de la participación de humanos directamente involucrados en misiones espaciales. Por una parte, la experiencia que ha dejado la exploración espacial muestra que la presencia de humanos en estas misiones es de gran ayuda. La habilidad para reaccionar a escenarios inesperados, para hacer arreglos a equipos robóticos y la versatilidad para realizar una gran variedad de tareas, son características que han permitido que muchas de las misiones espaciales sean un éxito.

Por el otro lado, es bien sabido que la presencia de seres humanos en vehículos espaciales causa que el diseño sea mucho más complejo, forzando rigurosas medidas de seguridad con las que se debe contar, causando que el costo de la misión se eleve mucho. Además, conforme avanza la tecnología los equipos robóticos se van haciendo más y más convenientes para misiones de exploración espacial lejanas a la tierra. Con esto y los recientes éxitos de la NASA, como el descubrimiento un planeta con características parecidas a la tierra *Proxima B* o la llegada de la sonda New Horizons a Plutón, se puede ver el interés de una parte de la comunidad científica por continuar la investigación y el desarrollo de vehículos robóticos de exploración espacial. Hay varios tipos de robots involucrados en las misiones espaciales, pero los de interés particular para este trabajo son los rovers planetarios.

3.2 Rover

Un rover planetario se puede ver como una nave espacial ya que cuenta con muchas de las mismas funciones y subsistemas, solo que este está limitado por las restricciones que le impone el ambiente planetario al que estaría expuesto. Las restricciones de diseño típicas son: la masa, el suministro de energía, los recursos computacionales y la complejidad para la comunicación con el rover. Además de las restricciones del ambiente del planeta, también se toman en cuenta las tareas que realizará el rover y la carga científica con la que cargará (instrumentos con los que realiza pruebas como análisis de muestras). Una distribución de masas típica en rovers se comprende de 30% chasis, 20% carga científica y 50% subsistemas. Hay

4 tipos de rovers planetarios: A.- Vehículos autónomos (Sojourner); B.- Vehículos teleoperados (Lunokhod); C.- Vehículos operados por humanos sin presurizar (Apollo Lunar Rover Vehicle); D.- Vehículos operados por humanos presurizados (aún no han tenido misiones) [2].

3.2.1 Lunokhod 1 y 2

El primer rover en explorar satisfactoriamente otro cuerpo planetario fue el Lunokhod 1, era parte de la misión Luna 17 de la unión soviética, que fue lanzada el 10 de noviembre de 1970 y aterrizó en la Luna el 17 de ese mismo mes. Dos años más tarde el lander Luna 21 aterrizó, llevaba una versión mejorada del rover anterior, el Lunokhod 2 que tenía cámaras de mejor resolución y un mejor sistema de control.

Éste rover era teleoperado desde la Tierra, tenía un chasis con 8 ruedas, era alimentado por energía solar durante el día y por la noche hacía uso de un radioisótopo de polonio-210 para mantenerse caliente ya que las temperaturas a las que se podía enfrentar en la superficie lunar llegaban hasta los -150 °C [3].

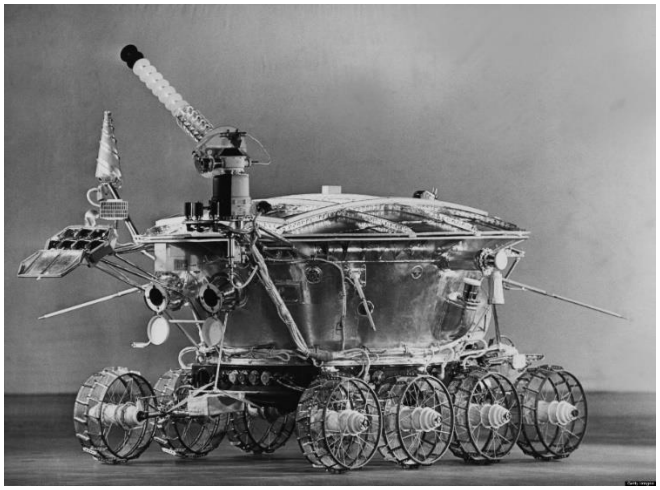


Figura 3.1 Fotografía del rover Lunokhod [Crédito NASA]

3.2.2 Sojourner

En el año de 1977 el lander Mars Pathfinder aterrizó en Marte cargando, entre otros instrumentos de estudio, el micro-rover Sojourner. Contaba con una suspensión tipo rocker-bogie con 6 ruedas que le proporcionaba una gran estabilidad y capacidad para superar obstáculos grandes, sus ruedas estaban hechas de aluminio. Estaba equipado con dos cámaras en blanco y negro, una cámara a color y un espectrómetro de rayos X [4]. Contaba con cierto grado de autonomía al girar, evitando obstáculos con el uso de sus sensores. El Sojourner recorrió un total de 106 m en un radio de 10 m alrededor del lander Pathfinder.

Figura 3.2 Fotografía del rover Sojourner en Marte [Crédito NASA]

3.2.3 Spirit y Opportunity

Ambos rovers fueron parte de la misión Mars Exploration Rovers (MER), aterrizaron en Marte y comenzaron operaciones en el 2004, fueron desplegados en dos regiones dónde se cree que podría haber rastros de la presencia de agua, la misión objetivo de los rovers era la búsqueda y caracterización de rocas o suelos cercanos a sus sitios de aterrizaje para descubrir más sobre el pasado del planeta donde pudieron haber existido condiciones habitables.

Ambos rovers tenían una masa de 174 kg y contaban con un chasis de seis ruedas con suspensión tipo rocker-bogie. Cada rueda era impulsada independientemente y las de las esquinas eran capaces de girar permitiendo así cambiar de orientación sin necesidad de avanzar. Contaban con instrumentos científicos que les permitían realizar estudios geológicos y observaciones atmosféricas.

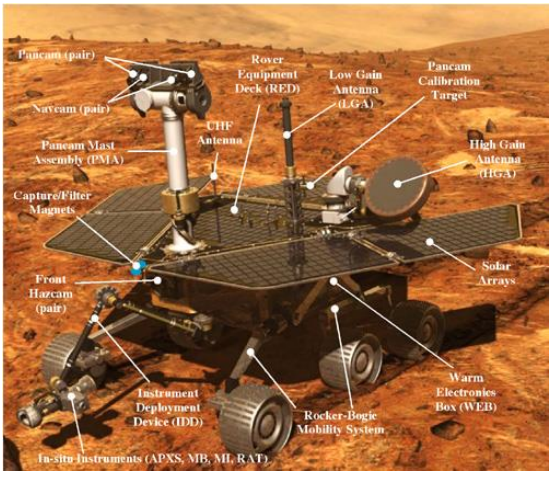


Figura 3.3 Herramientas científicas presentes en el rover Spirit [Crédito NASA]

3.2.4 Curiosity

Siendo parte de la misión Mars Science Laboratory, el Curiosity es el rover más grande y capaz que se ha enviado al espacio hasta el momento. Llegó a Marte el 5 de agosto de 2012, su aterrizaje se realizó con un método completamente nuevo. La nave espacial en la que viajaba descendió con ayuda de un paracaídas, en los segundos finales antes del aterrizaje el sistema encendió sus cohetes para permanecer suspendido a unos metros de la superficie de Marte mientras el rover bajaba y aterrizaba sobre sus ruedas, finalmente la nave se estrelló a una distancia segura para evitar dañar al rover.

El Curiosity está cuenta con una suspensión tipo rocker-bogie y es capaz de sobrepasar obstáculos de hasta 75 cm de altura. Avanza a una velocidad promedio de 30 m/h. Lleva un radioisótopo de plutonio que le proporciona energía eléctrica, lo que le permite alimentar una mayor cantidad de instrumentos científicos sin importar la cantidad de luz solar o la temporada en la que se encuentre. La misión de éste rover era responder la pregunta ¿Alguna vez tuvo Marte las condiciones ambientales necesarias para mantener vivos a microbios?, no mucho tiempo después de iniciar su misión sus herramientas científicas encontraron evidencia química y mineral de ambientes habitables que existieron en el pasado de Marte [5].

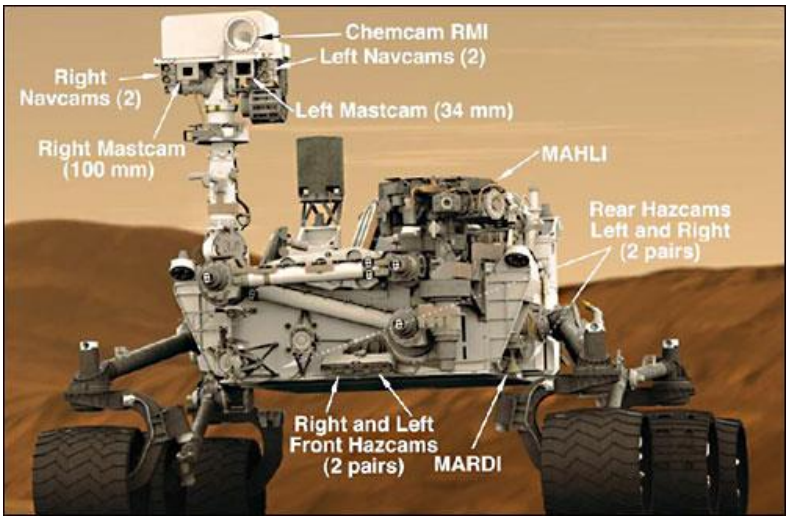


Figura 3.4 Herramientas científicas presentes en el rover Curiosity [Crédito NASA]

3.3 Computadoras de a bordo en rovers

La computadora de a bordo de un rover se encarga de realizar la comunicación entre la estación de control y el rover, procesar esa información junto con la proveniente de los subsistemas y usarla para coordinar el funcionamiento de los subsistemas para ejecutar las tareas asignadas. Para cumplir con su función, la OBC de un rover planetario necesita una cantidad significativa de recursos computacionales. Tradicionalmente suelen usarse procesadores de 32 bits, que

sean resistentes a dosis altas de radiación (una dosis acumulativa de 25 krad suele ser suficiente para la mayoría de las misiones [6]), su velocidad de procesamiento debe ser suficiente para ejecutar procesos de control y navegación en tiempo real, debe poder ejecutar un sistema operativo en tiempo real (RTEMS, Linux o VxWorks son los más populares), y debe tener memoria no volátil para almacenar la información relevante para su misión.

Las OBCs de los rovers que se encuentran en marte cuentan con las siguientes características:

- Sojourner
  Microprocesador Intel 80C85 a 2 MHz capaz de ejecutar 100 KIPS (cien mil instrucciones por segundo) con arquitectura de 8 bits, con 16 kB de memoria de arranque PROM (memoria programable de solo lectura), con 176 kB de EEPROM (PROM eléctricamente borrable) para almacenamiento de programas y 512 kB de memoria RAM (memoria de acceso aleatorio) [7].
- Spirit y Opportunity
  Procesador RAD6000 a 20 MHz capaz de ejecutar 22 MIPS con arquitectura de 32 bits, con 128 kB de memoria DRAM (RAM dinámica), 11 MB de EEPROM y 256 MB de memoria flash [7].
- Curiosity
  Procesador central BAE RAD750 hasta 200 MHz con arquitectura de la familia PowerPC 750 capaz de ejecutar 400 MIPS, 256 MB de DRAM, 256 kB de EEPROM y 2 GB de memoria flash [8].

A diferencia de los rovers en marte, las OBCs de los rovers diseñados para competencias en la tierra no necesitan microprocesadores con la misma tolerancia a la radiación, por eso es posible usar componentes con más poder de procesamiento y con un costo mucho menor, en la *Tabla 3-1* se encuentran ejemplos de los componentes usados como OBC en competencias terrestres:

Tabla 3-1 Características los sistemas usados como OBC en rovers que han participado en la URC

Rover	Sistema	Especificaciones
York University Rover Team	ALIX.2D2 [9]	CPU: AMD Geode LX800, 500 MHz. DRAM: 256 MB
Oregon State Mars Rover	Personalizado [10]	CPU: 2 x ATxmega256A3, 32 MHz. Flash: 256 KB. EEPROM: 4 KB. SRAM: 16 KB
Mars Rover Manipal	Raspberry Pi 3 [11]	CPU: 4 x ARM Cortex-A53, 1.2GHz. RAM: 1 GB
Raptors	sbRIO [12]	CPU: Dual-core ARM Cortex A9, 667 MHz. RAM: 512 MB. Flash 512 MB

3.4 University Rover Challenge

La university rover challenge (URC) es una competencia de robótica para equipos de estudiantes universitarios de diferentes partes del mundo. La URC se celebra anualmente en una zona del desierto de Utah, conocida como Mars Desert Research Station, en la que se presentan ciertas características similares a las que se pueden encontrar en Marte. La *Tabla 3-2* permite visualizar algunas de las diferencias entre la Tierra y Marte.

Tabla 3-2 Propiedades físicas en la Tierra y Marte (para ejemplificar algunas de las diferencias entre estos) [13] [14]

Planeta	**Gravedad (m/s^2)**	**Presión (hPa)**	**Temperatura media y rango (K)**	**Energía solar (W/m^2)**	**Velocidad del viento (m/s)**	**Composición atmosférica**
Tierra	9.78	1013	288, 283 - 293	1380	0 a 100	78% N_2, 21% O_2, H_2O, Ar, CO_2, Ne, He, CH_4, H_2
Marte	3.71	6.3	210, 184 - 242	595	2 a 30	96% CO_2, 1.9% Ar, 1.8% N_2, O_2, CO, H_2O, NO, Ne, Kr

El objetivo de la URC es motivar a los estudiantes a desarrollar habilidades en robótica, mejorar el estado del arte en rovers y aprender a trabajar con un equipo multidisciplinario en el que colaboran científicos e ingenieros.

Cada año se van cambiando algunos detalles de las tareas de las que se compone la competencia para fomentar la flexibilidad en el diseño de los rovers y mejorar sus capacidades. Los rovers deben realizar tareas teleoperadas o autónomas que simulan la asistencia que deben brindar a astronautas en Marte. Las tareas teleoperadas se llevan a cabo desde estaciones de control sin vista directa al rover, el operador debe controlar al rover con el video y la información de los sensores. En las tareas autónomas el rover debe realizar la toma de decisiones para completar los objetivos, para esto requiere evasión de obstáculos y planeación de rutas.

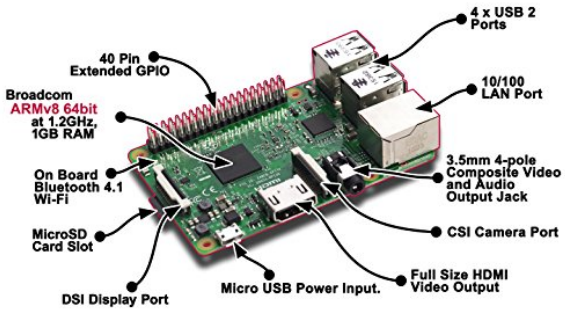
Existe una lista completa de las reglas de la competencia [15], pero en términos generales las reglas son que el rover no debe pesar más de 50 Kg (70 Kg con todas las piezas que se usarán en las diferentes tareas), debe caber en una plataforma cuadrada de 1.2 m por 1.2 m, el costo del rover no debe sobrepasar los 18,000 dólares estadounidenses y la frecuencia que podrá usar en el rover para comunicación debe ser 900 MHz o 2.4GHz.

4 Marco teórico

4.1 Raspberry Pi

La Raspberry Pi posee características que la convierten en candidata ideal para la OBC: tiene un gran poder de procesamiento (4 x ARM Cortex-A53 de 64 bits a 1.4GHz), los GPIOs con los que cuenta facilitan la implementación de protocolos de comunicación serial (UART, I2C y SPI) y es compatible con sistemas operativos que usan núcleos capaces de brindar comportamiento en tiempo real, Linux.

La Raspberry Pi es una computadora contenida completamente en una placa del tamaño de una tarjeta de crédito, tiene un precio accesible y consume poca energía en relación a una computadora de escritorio promedio. Fue desarrollada en el Reino Unido por la Raspberry Pi Foundation con el objetivo de apoyar a la enseñanza de materias relacionadas con la computación dentro de escuelas a niveles básicos, sin embargo, rápidamente creció una gran comunidad centrada en el uso de esta computadora para el desarrollo de proyectos de electrónica y robótica. La gran popularidad de la Raspberry Pi para proyectos de esa naturaleza se debe principalmente a tres factores: la disponibilidad de entradas y salidas de propósito general (GPIO), la diversidad de sistemas operativos con los que cuenta y los lenguajes de programación en los que se puede desarrollar.



Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figura 4.1 Características de la Raspberry Pi 3 modelo B (con explicación de GPIOs) [Crédito Element14.com]

Las entradas y salidas de la Raspberry le permiten usar protocolos de comunicación como UART, SPI e I2C lo que facilita la interacción con sensores y actuadores para el control de diversos sistemas, aunado a esto, la diversidad de sistemas operativos y de lenguajes de programación también favorece a que se unan diferentes comunidades de desarrolladores y gracias a esto se cuenta con una gran cantidad

de información para guiarse durante el desarrollo de proyectos lo que hace a la Raspberry Pi una plataforma adecuada para la computadora de a bordo.

La Raspberry Pi 3 modelo B+ cuenta con las siguientes especificaciones [16]:

- **CPU:** Cuatro núcleos ARM Cortex-A53 de 64 bits a 1.4GHz
- **GPU:** VideoCore IV a 400MHz
- **RAM:** 1GB de memoria LPDDR2
- **Redes:** Ethernet Gigabit, WiFi de doble banda 802.11ac, Bluetooth 4.2 y Bluetooth de baja energía.
- **Salidas de audio/video:** Jack analógico de AV de 3.5 mm, puerto HDMI 1.3
- **Puertos periféricos:** 4 puertos USB 2.0, interfaz serial para cámara, interfaz serial para pantalla.
- **Almacenamiento:** Un puerto para microSD de hasta 512GB
- **Alimentación:** 5 volts a 2.5 amperes por puerto micro USB
- **Extra:** 40 pines de GPIOs, PoE (alimentación por ethernet) mediante hardware adicional

4.1.1 Sistema operativo: Raspbian

Un sistema operativo es el software que maneja el hardware presente en una computadora, provee una base para la ejecución de programas y actúa como un intermediario entre el usuario de la computadora y recursos computacionales con los que cuenta. En general, el hardware incluye la CPU, memoria RAM, dispositivos de entrada/salida, memoria para almacenamiento de información y debe proveer mecanismos apropiados para asegurar la correcta operación de la computadora. Una responsabilidad fundamental del sistema operativo es asignar correctamente los recursos del sistema a los programas para que puedan ejecutar correctamente sus tareas [17]. A continuación, se muestra una representación abstracta de como el sistema operativo actúa mediando la interacción del usuario y el hardware de la computadora.

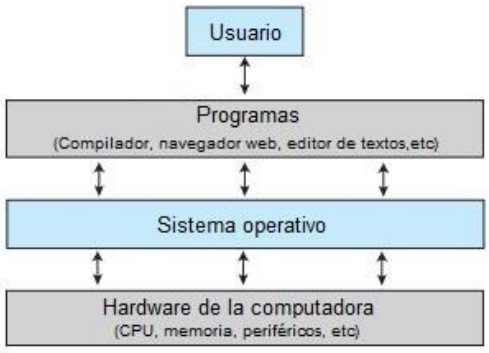


Figura 4.2 Vista abstracta de los componentes de un sistema computacional [17]

Raspbian es uno de los sistemas operativos más populares para la Raspberry Pi en la actualidad. Raspbian es un sistema operativo de código abierto que está basado en Debian, ha sido modificado específicamente para su uso con la Raspberry, comparte casi todas las características de Debian, incluyendo su gran repositorio de paquetes de software. El núcleo de Linux y la mayoría del software de Raspbian está licenciado bajo la licencia GPLv2 que permite su uso libre [18].

Raspbian le brinda un gran potencial para el desarrollo de proyectos a la Raspberry Pi con su amplia variedad de software y su compatibilidad con los lenguajes de programación más populares, a pesar de esto, existen discusiones sobre la capacidad de Raspbian para ofrecer las capacidades de un sistema operativo en tiempo real.

Los sistemas operativos en tiempo real (RTOS) son usados con sistemas en tiempo real (RTS), cuando existen requerimientos temporales estrictos para el correcto funcionamiento del sistema, por ejemplo, algunos sistemas espaciales que controlan experimentos científicos o sistemas industriales de control. Un RTS tiene restricciones temporales fijas y bien definidas, el procesamiento de información debe ser realizado durante el tiempo definido por dichas restricciones, estos sistemas funcionan correctamente solo si se realizan las tareas dentro de los tiempos previstos. De acuerdo a la manera de programar la ejecución de los procesos en la CPU (CPU Scheduling), se suele distinguir entre sistemas en tiempo real flexibles (SRTS) y sistemas en tiempo real estrictos (HRTS). Los SRTS no proveen garantía de cuando se ejecutará el proceso crítico (el proceso con las restricciones temporales), solo garantizan que el proceso tendrá una prioridad más alta que los procesos no críticos (que no están sujetos a restricciones temporales). Los HRTS tienen requerimientos más estrictos y los procesos deben ser realizados en los tiempos definidos [17]. Ya que Linux cuenta con la capacidad de asignar prioridades a los procesos que se ejecutan en la CPU y como Raspbian implementa núcleos basados en Linux, se puede tratar una Raspberry Pi con Raspbian como un SRTS.

4.2 Lenguajes de programación

Un lenguaje de programación es un lenguaje formal, que comprende un conjunto de instrucciones que producen varios tipos de salidas. Los lenguajes de programación proveen abstracciones, principios de organización y estructuras de control que permiten a los programadores escribir buenos programas.

Un lenguaje de programación es usado para escribir programas computacionales, los cuales involucran a una computadora llevando a cabo algún tipo de cómputo, algoritmo y/o el control de dispositivos externos como impresoras, memorias o robots. Normalmente los lenguajes sirven para comunicarnos entre personas, los lenguajes de programación permiten a los humanos comunicar instrucciones a las máquinas.

La mayoría de los lenguajes de programación comparten algunos bloques de construcción que sirven para la descripción de información y los procesos o transformaciones que se les aplican, estos bloques son: Sintaxis, Semántica, Sistema de tipos y Bibliotecas estándar.

Usualmente se clasifican a los lenguajes de programación en cuatro paradigmas principales: Imperativo, Lógico, Funcional y Orientado a Objetos. En un lenguaje Imperativo, se escribe un algoritmo, así como el orden de la ejecución de sus pasos. En un lenguaje lógico, se declaran un conjunto de expresiones y reglas lógicas, después el sistema de implementación del lenguaje debe usar dichas afirmaciones y reglas para deducir los resultados deseados. El paradigma funcional trata a las expresiones como funciones matemáticas, dichas funciones aceptan argumentos y regresan una solución única. La programación orientada a objetos es un paradigma en el cual se describen objetos vistos como entidades separadas que tienen sus propias variables y estados que pueden ser modificados por métodos específicos del objeto. Estos objetos están organizados en clases, de las cuales heredan métodos y variables. El paradigma orientado a objetos provee beneficios importantes para la reusabilidad de código y su extensibilidad [19].

4.2.1 Python

Python es un lenguaje de programación muy poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y un enfoque simple pero efectivo a la programación orientada a objetos. La intuitiva sintaxis de Python y su tipado dinámico (lo que permite asignar diferentes tipos de valores a una variable, a diferencia de lenguajes como C o Java donde se tiene que declarar el tipo de la variable), junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas. El intérprete de Python y la extensa biblioteca estándar están disponibles de forma gratuita en el sitio web de Python. El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ [20].

Python es un lenguaje de programación multiparadigma, lo que significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. También es un lenguaje de programación interpretado, lo que significa que no necesita ser compilado, lo que le permite utilizarse en diferentes sistemas operativos como Windows o Linux, aunque se suele tomar esto como una característica negativa ya que, al tratarse de un lenguaje interpretado, Python es más lento que un código compilado, a pesar de ello, actualmente las diferencias en velocidad son pequeñas y cada vez se reduce más esta brecha gracias al incremento de la capacidad de procesamiento de las nuevas generaciones de CPUs, además del acceso al cómputo en la nube. Muchas veces tiene más valor

reducir los tiempos de desarrollo, calidad del código y la facilidad para darle mantenimiento, que es precisamente a lo que se enfoca Python.

4.2.1.1 Intérprete de Python

Cuando se escribe un programa de Python el intérprete se encarga de leerlo y mandar a ejecutar las instrucciones que contiene. El intérprete es una capa de software lógico entre el código y el hardware de la computadora. Al ejecutar un código el intérprete tiene que realizar dos pasos: Traduce el código original a un *Código de bytes* (Byte code) y luego lo envía a la *Máquina Virtual de Python* (PVM por sus siglas en inglés).

El código de bytes es simplemente una representación de bajo nivel, independiente de la plataforma, del código original (Source). Cuando se tiene el código de bytes, se envía a la máquina virtual de Python, que es simplemente un bucle de código que itera a través de las instrucciones en el código de bytes, una por una, para llevarlas a cabo.

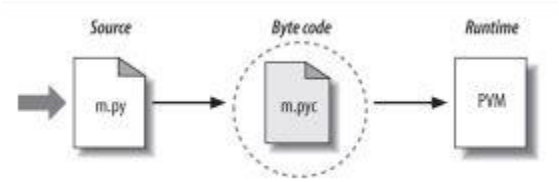


Figura 4.3 Pasos que sigue el Intérprete de Python para ejecutar un programa [21]

4.2.1.2 Tipos de datos y operaciones

Lo más básico de la programación es realizar operaciones con datos, en Python los datos se manejan como objetos y las operaciones que se le pueden realizar, dependen del tipo de objetos con los que se esté trabajando. Se puede pensar en los objetos como piezas de memoria con valores y conjuntos de operaciones asociadas. Hay conceptos más complejos a cerca de los objetos que se verán más adelante, en la *Tabla 4-1* se muestran los tipos de objetos que vienen por default en Python:

Tabla 4-1 Tipos de objetos encontrados por default en Python [22]

Tipo de objeto	Ejemplos de inicialización	Ejemplos de operaciones

Números	`1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()`	`>>> 123 + 222`
		`345`
		`>>> 2 ** 10`
		`1024`

Cadenas/Strings	`'spam', "Bob's", b'a\x01c', u'sp\xc4m'`	`>>> S = 'String'`
		`>>> len(S)`
		`6`
		`>>> S[0]`
		`'S'`
		`>>> S + '/Cadena'`

		‘String/Cadena’

Listas	[1, [2, 'tres'], 4.5], list(range(10))	>>> L = [85, ‘Ls’, 3.2]
		>>> len(L)
		3
		>>> L.append(‘OBC’)
		>>> L
		[85, ‘Ls’, 3.2, ‘OBC’]

Diccionarios	{'comida': 'spam', 'sabor': 'yum'}, dict(horas=10)	>>> D = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
		>>> D[‘food’]
		‘Spam’

Tuplas	(1, 'spam', 4, 'U'), tuple('spam'), namedtuple	>>> T = (1, 2, 3, 4)
		>>> len(T)
		4
		>>> T.count(4)
		1

Archivos	open('huevo.txt'), open(r'C:\jamon.bin', 'wb')	>>> f = open('data.txt', 'w')
		>>> f.write('Hello\n')
		6
		>>> f.close()

Conjuntos	set('abc'), {'a', 'b', 'c'}	>>> X = set('spam')
		>>> Y = {'h', 'a', 'm'}
		>>> X & Y
		{'m', 'a'}
		>>> X | Y
		{'m', 'h', 'a', 'p', 's'}

Otros tipos importantes	Booleanos, None. Types	>>> 1 > 2, 1 < 2 (False, True) None <type 'list'>

Los números pueden ser enteros, flotantes, complejos, decimales con precisión fija, racionales con numerador y denominador, etc. Soportan las operaciones matemáticas normales, suma (+), resta (-), multiplicación (*), división (/), exponenciación (**). Y si se requieren más funciones, Python viene equipado con el módulo *math* que se puede importar para tener acceso a sus funciones.

Las cadenas se usan para almacenar información textual y colecciones arbitrarias de bytes (como el contenido de un archivo de imagen). Son un ejemplo de lo que en Python se maneja como secuencias, una colección de objetos ordenados por posición. Se les pueden aplicar operaciones de secuencias, concatenación, repetición y otros métodos específicos del objeto String. Las cadenas son inmutables, las operaciones realizadas sobre ellas no cambian su valor, la única manera de cambiar la cadena original es sobrescribirla completamente con otro valor.

El objeto “lista” es la secuencia más general en Python. Son colecciones de objetos (que pueden ser de diferentes tipos) ordenados por posición, sin tamaño fijo. Son mutables, lo que significa que pueden ser modificadas por asignación de posiciones y por varios métodos específicos de las listas. Proveen una forma flexible de

representar colecciones diversas (listas de archivos en un folder, empleados en una compañía y muchas otras).

4.2.1.3 Sentencias y sintaxis

Cada sentencia en Python tiene su propio rol y su propia sintaxis, aunque varias de ellas comparten cierto patrón en su sintaxis. En un programa estas sentencias pueden realizar acciones, repetir tareas, realizar elecciones, construir estructuras más grandes y muchas cosas más. A continuación, se presenta una tabla que resume el conjunto de sentencias de Python:

Tabla 4-2 Sentencias comunes en Python [23]

Sentencia	**Rol**	**Ejemplo**
Asignación	Crear referencias	`a, b = 'good', 'bad'`
Llamadas y otras expresiones	Correr funciones	`log.write("spam, ham")`
Llamadas a print	Imprimir objetos	`print('The Killer', joke)`
if/elif/else	Seleccionar acciones	`if "python" in text:` `  print(text)`
for/else	Iteración	`for x in mylist:` `  print(x)`
while/else	Bucles generales	`while X > Y:` `  print('hello')`
pass	Marcador de posición vacío	`while True:` `  pass`
break	Salir de bucles	`while True:` `  if exittest(): break`
continue	Continuar bucle	`while True:` `  if skiptest(): continue`
def	Funciones y métodos	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`
return	Resultados de funciones	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
yield	Funciones generadoras	`def gen(n):` `  for i in n: yield i*2`

global	Espacios de nombres	`x = 'old'`
		`def function():`
		`  global x, y; x = 'new'`

nonlocal	Espacios de nombres (Python 3.X)	`def outer():`
		`  x = 'old'`
		`  def function():`
		`  nonlocal x; x = 'new'`

import	Acceso a módulos	`import sys`
from	Acceso a atributos	`from sys import stdin`

class	Construcción de objetos	`class Subclass(Superclass):`
		`  staticData = []`
		`  def method(self): pass`

try/except/ finally	Atrapar excepciones	`try:` `  action()`

		except: print('action error')
raise	Alzar excepciones	raise EndSearch(location)
assert	Chequeo para debuguear	assert X > Y, 'X too small'
with/as	Gestores de contexto (Python 3.X, 2.6+)	with open('data') as myfile: process(myfile)
del	Eliminar referencias	del data[k] del data[i:j] del obj.attr

Para explicar la sintaxis, se observa la *Tabla 4-3* en la cual se encuentra la comparación de la sentencia **`if`** implementada en los lenguajes C y Python.

Tabla 4-3 Comparación de la sentencia if en C y en Python

C	Python

if (x > y) {
x = 1;
y = 2;
}
	if x > y:
	x = 1
	y = 2

Lo primero que se nota es que la sintaxis de Python es más simple, lo cual lo hace más fácil de comprender.

En concreto, son cuatro los aspectos a recordar de la sintaxis de Python: Los paréntesis son opcionales, el fin de línea es el fin de la sentencia, se agregan los dos puntos (:) al final de la cabecera y el fin del sangrado representa el final del bloque de código anidado dentro de una sentencia (de aquí en adelante se usará el término indentado o indentación para referirse a la sangría).

Todas las sentencias en Python que tienen otras sentencias anidadas dentro de ellas, siguen el mismo patrón de la sentencia de cabecera que termina en dos puntos, seguido de un bloque de código anidado (la lista de sentencias) debajo de la sentencia de cabecera e indentado como sigue:

```
Sentencia de cabecera:
      Sentencia 1
      Sentencia N
```

Se separan las sentencias con un salto de línea y el fin de la indentación significa el fin del bloque de código anidado.

El uso de la indentación como base de la sintaxis del lenguaje, obliga a los programadores a producir un código uniforme, regular y fácilmente comprensible. Lo que se traduce en códigos consistentes y fáciles de interpretar. Además de evitar problemas con los incontables debates y preferencias de indentación de los programadores.

4.2.1.4 Funciones

En términos simples, una función es una herramienta que agrupa un conjunto de sentencias para que puedan ser ejecutadas más de una vez en un programa, en otras palabras, una serie de procedimientos empaquetados que se invocan al llamar un nombre. Las funciones también pueden computar un resultado y permiten especificar parámetros que sirven como entradas de la función que pueden variar cada vez que se ejecute el código. Además de esto, las funciones sirven para evitar tener múltiples copias redundantes del código de una operación, reduciendo el trabajo futuro radicalmente: si la operación debe ser modificada posteriormente, solo se debe actualizar una sección de código y no las múltiples copias que hubiera habido. A continuación, se muestra una tabla de sentencias relacionadas a funciones en Python:

Tabla 4-4 Sentencias y expresiones relacionadas con las funciones [24]

Sentencia o expresión	**Ejemplos**
Expresiones de llamada	`myfunc('spam', 'eggs', meat=ham, *rest)`
def	`def printer(message):` `  print('Hello ' + message)`
return	`def adder(a, b=1, *c):` `  return a + b + c[0]`

global	`x = 'old'`
	`def changer():`
	`  global x; x = 'new'`

nonlocal (3.X)	`def outer():`
	`  x = 'old'`
	`  def changer():`
	`  nonlocal x; x = 'new'`

yield	`def squares(x):` `  for i in range(x): yield i ** 2`
lambda	`funcs = [lambda x: x**2, lambda x: x**3]`

Las funciones se construyen con la sentencia *def*, que crea un objeto y lo asigna a un nombre. Su formato general es el siguiente:

```
def name(arg1, arg2, … , argN):
      sentencias
      return value
```

Además de las posibilidades que ofrecen las funciones en muchos lenguajes de programación, en Python se pueden usar algunas características especiales para los argumentos de entrada: normales, por palabra clave, por default y número arbitrario.

4.2.1.5 Módulos

Los módulos empaquetan código del programa e información para ser reusados, proveen un espacio independiente para minimizar la colisión de nombres asignados

a los diferentes elementos dentro de los programas. Los módulos corresponden a los archivos del programa, cada archivo es un módulo, y los módulos importan otros módulos para usar los nombres definidos en estos. También pueden corresponder a extensiones codificadas en lenguajes externos como C, Java o C# [25]. Los módulos son procesados con dos sentencias diferentes:

`import`: Permite al programador importar un módulo completo

`from`: Permite al programador importar nombres particulares de un módulo

4.2.1.6 Clases y programación orientada a objetos (POO)

Las clases son una manera de agrupar: información que está relacionada de algún modo y funciones que actúan sobre esa información. En otros términos, las clases sirven para reflejar objetos reales en el dominio de un programa.

Cuando se crea un objeto de una clase, los valores que se almacenan en ese objeto son llamados atributos, y las funciones asociadas con el objeto se conocen como métodos. También existen los atributos de clase que un valor que se comparte entre todos los objetos de esa clase.

Para ejemplificar la creación de clases y objetos de esa clase se empieza por suponer que se quiere representar una persona que tiene nombre, apellidos y puede comer. La forma en la que se define en Python:

```
class Persona(object):
      especie = “Humano”                        # Atributo de clase
      def __init__(self, nombre, apellidos):
            self.nombre = nombre                # Atributo de objeto
            self.apellidos = apellidos
      def comer(self, alimento):                # Método de objeto
            print self. nombre + " está comiendo " + alimento
>>> pepe = Persona(“José”, “Galindo López”)
```

La sentencia *class* permite crear la clase “Persona” y necesita de un constructor *__init__* para poder crear objetos de esa clase.

Las clases son importantes para el enfoque a la programación orientada a objetos (POO). En enfoque de POO se vuelve más valioso conforme aumenta el tamaño y la complejidad del programa. También le agrega al programa una capa extra de organización y modularidad, lo que hace más fácil entender el código y también facilita su modificación por otros programadores [26].

Uno de los conceptos más importantes en la POO es la Herencia. La herencia es una forma de organizar objetos en una jerarquía del objeto más general al más específico. Se suele decir que una clase es una subclase de la clase de la que

hereda, o que la otra clase es la superclase. Una subclase puede heredar atributos y/o métodos de su superclase. La herencia sirve para representar objetos que tienen algunas diferencias y algunas similitudes en la forma en la que funcionan. Se pueden poner todas las características que se comparten en una superclase y de ahí definir las subclases con sus características únicas.

Para ejemplificar, se puede definir a un estudiante como un tipo de persona, que cumple con las características generales de una persona pero que, además, tiene características únicas. Usando la clase "Persona", presente más arriba, se construye la clase "Estudiante" de la siguiente manera:

```
class Estudiante(Persona):
      def __init__(self, nombre, apellidos, promedio):
            self.promedio = promedio
            self.materias = []
            super(Estudiante, self).__init__(nombre, apellidos)
      def agregar_materia(self, materia):
            self.materias.append(materia)
>>> juan = Estudiante("Juan", "Martínez Ramírez", 8.5)
```

Como la clase "Estudiante" es subclase de "Persona", el objeto "juan" hereda los atributos y métodos de la clase "Persona": *nombre, apellidos* y *comer*. Al ejecutar el código y llamar al método *comer* del objeto "juan", se obtiene lo siguiente:

```
>>> juan.comer("Torta")
Juan está comiendo Torta
```

4.2.1.7 Excepciones

Las excepciones son estructuras que permiten lidiar con escenarios en los que se presenta un error para el que se requiere realizar algún procedimiento especial para evitar que el programa deje de funcionar. Cuando ocurre una excepción se salta al código de un manejador de excepciones, la sentencia *try*, ese código debe poder lidiar de manera apropiada con el error y regresar al flujo normal del programa. Este protocolo provee una manera coherente para responder a eventos inusuales [27].

Las excepciones se usan de la siguiente manera:

```
try:
      Bloque de código
except Tipo de error:
      Código para lidiar con el error
```

Los tipos de errores más comunes de son: IndexError, TypeError y ValueError.

4.3 Protocolos de comunicación

Un protocolo de comunicación es un conjunto de reglas que deben ser seguidas rigurosamente para lograr la comunicación entre dos o más sistemas. Son necesarios para asegurar que la interacción entre dichos sistemas sea exitosa. Las reglas, regulaciones, la sintaxis y la semántica son definidas en el protocolo. La transmisión de información se puede llevar a cabo por medio de fluctuaciones de alguna propiedad física, por ejemplo, el voltaje.

Existen diversos tipos de protocolos de comunicación, pero se enfoca la atención en los protocolos de comunicación más usados con sistemas embebidos y microcontroladores, los protocolos de comunicación serial.

La comunicación serial es una forma de intercambio de información entre sistemas digitales en el cual se transmiten bits secuencialmente, un bit a la vez por línea, mediante un bus (un bus es un conjunto de conductores paralelos en los que se varía un voltaje). Dos de las principales ventajas del uso de este tipo de comunicación son: 1) El bajo número de líneas conductoras que se necesitan para su implementación 2) Muchos microcontroladores y sensores cuentan con soporte para las interfaces seriales más populares [28]. La principal desventaja es su reducida velocidad de transmisión de datos. Para comprender las principales interfaces de comunicación serial hay que revisar algunos conceptos básicos:

Tipo de comunicación:

- **Simplex**: La comunicación se realiza enteramente en una dirección, sin posibilidad de cambiar la dirección
- **Half Duplex**: La transmisión se puede realizar en ambas direcciones, pero no simultáneamente.
- **Full Duplex**: La transmisión se puede realizar en ambas direcciones simultáneamente.
- **Síncrona**: La comunicación serial síncrona usa una señal de reloj compartida para marcar la presencia de los bits de la información. Se necesitan al menos dos líneas: una para el reloj y otra para los datos. El dispositivo maestro, es el encargado de generar la señal de reloj, mientras que los demás dispositivos, los esclavos, operan de acuerdo al reloj.
- **Asíncrona**: En la comunicación serial asíncrona, la transmisión de datos se realiza a una velocidad previamente establecida. Para marcar el inicio de la transmisión se usa un bit de inicio y para marcar el final se utiliza un bit de parada.

Las principales interfaces de comunicación serial usadas en sistemas embebidos son: UART, SPI, I2C y CAN.

4.3.1 UART

La UART (Universal Asynchronous Receiver/Transmitter) no es un protocolo de comunicación, sino una interfaz presente en la mayoría de los microcontroladores. El principal objetivo de la interfaz UART es el intercambio de información serial con otro dispositivo. Ya que es comunicación asíncrona, los dispositivos que se comunican deben tener la misma configuración de velocidad de transmisión (baudrate). La siguiente figura muestra cómo se conecta.

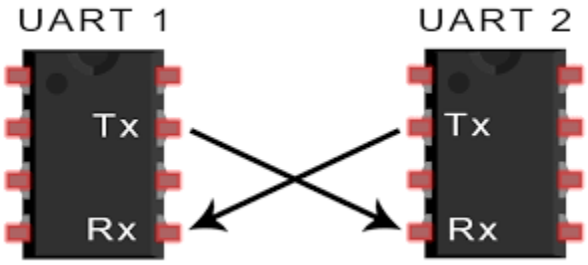


Figura 4.4 Conexión entre dos interfaces UART [Crédito circuitbasics.com]

Las líneas usadas son:

- **TX:** Es la línea usada para enviar información.
- **RX:** Es la línea usada para recibir información.

La información enviada mediante la interfaz UART está organizada en paquetes. Cada paquete tiene: un bit de inicio, el marco de datos de 5 a 9 bits (puede cambiar dependiendo de las capacidades de la UART), un bit opcional de paridad y uno o dos bits de fin.

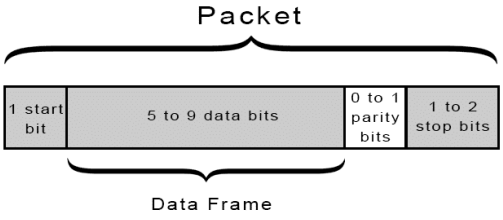


Figura 4.5 Estructura de un paquete de transmision con UART [Crédito circuitbasics.com]

Las líneas TX de cada UART suelen permanecer en un nivel de voltaje alto mientras no hay transmisión. Para comenzar una transmisión, el transmisor cambia la línea TX de alto a bajo durante un ciclo del reloj, cuando el receptor detecta la transición de alto a bajo, comienza a leer los bits en su línea RX.

El marco de datos puede tener una longitud de 5 a 8 bits (si se usa el bit de paridad, hasta 9 cuando no se usa el bit de paridad), en la mayoría de los casos la información es enviada con el bit menos significativo primero [29].

El bit de paridad se usa para verificar la integridad de la información transmitida. Cuando el marco de datos tiene una cantidad par de unos, el bit de paridad debe ser 0, cuando es una cantidad impar de unos, el bit de paridad debe ser 1. De esta manera, la UART que está recibiendo la información puede verificar si se corrompió la información o no.

El bit de fin se presenta cuando la UART transmisora cambia la línea TX de bajo a alto durante dos o más ciclos de reloj.

La velocidad de transmisión más común es de 9600 baudios, pero también se usan: 19200, 38400, 57600 y 115200.

Las principales ventajas de usar la interfaz UART son: Solo usa dos líneas para la transmisión de información, no es necesario tener una señal de reloj, puede tener un bit de paridad para ayudar a verificar la integridad del mensaje y se pueden usar otras configuraciones de la estructura del paquete (solamente si ambos dispositivos están configurados igual). La principal desventaja es que no es posible usar múltiples dispositivos en el mismo bus.

4.3.2 SPI

El protocolo de comunicación SPI (Serial Peripheral Interface) emplea un esquema de intercambio de datos entre un dispositivo maestro y uno o más esclavos, el maestro escoge al esclavo con el que se comunicará por medio de la línea SS, ambos dispositivos pueden enviar información al mismo tiempo. La siguiente figura ejemplifica la configuración más simple del bus SPI.

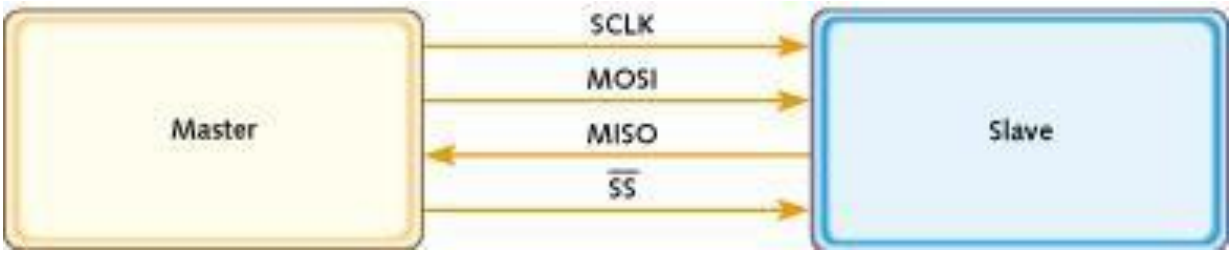


Figura 4.6 Conexión entre un maestro y un esclavo para comunicación SPI [Crédito National Instruments]

Las líneas típicas para SPI son:

- **Reloj (SCLK):** Es la línea donde el maestro produce la señal de reloj. Las demás señales en la transmisión dependen de los bordes del reloj.

- **Master Output/Slave Input (MOSI):** Es la línea donde el maestro pasa información hacia el esclavo.
- **Master Input/Slave Output (MISO):** Es la línea donde los esclavos pasan información hacia el maestro.
- **Chip Select (CS) o Slave Select (SS):** Son el conjunto de líneas con las que el maestro selecciona al esclavo con el que se comunicará.

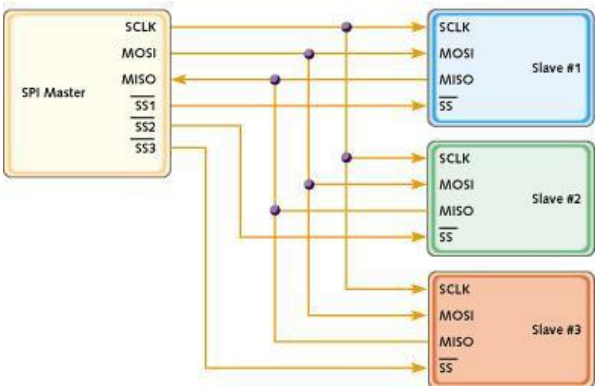


Figura 4.7 Dispositivo maestro conectado a tres esclavos para comunicación SPI [Crédito National Instruments]

Los pasos básicos de la transmisión de datos en SPI son:

1. El maestro activa un esclavo cambiando el estado de la línea **SS**, que usualmente implica pasar de voltaje “alto” a “bajo”.
2. El maestro genera la señal de reloj.
3. El maestro envía datos por la línea MOSI. El esclavo lee los bits.
4. Si se necesita una respuesta el esclavo envía datos por la línea MISO. El maestro lee los bits.

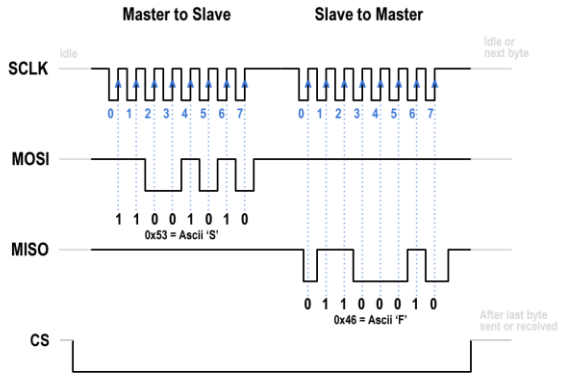


Figura 4.8 Ejemplo de una transmisión de datos con SPI [Crédito Electricimp.com]

Sus principales ventajas son: La simplicidad para implementar y las altas velocidades de transferencia de datos. Sus principales desventajas son: Cada

esclavo adicional requiere una línea extra del maestro, no tiene mecanismo para verificar si se recibió la información y no tiene estándares oficiales.

Para más información sobre la comunicación con SPI referirse a [30].

4.3.3 I2C

El protocolo de comunicación I2C o IIC (Inter-Integrated Circuit por sus siglas en inglés) se usa para la comunicación entre un maestro (o múltiples maestros) y uno o más esclavos de forma bidireccional. Un esclavo no puede transmitir datos a menos que el maestro lo instruya. Cada dispositivo en el bus debe tener una dirección única. Los dispositivos con interfaz I2C pueden tener uno o varios registros donde la información es almacenada, escrita o leída. La *Figura 4.9* muestra cómo se pueden conectar múltiples dispositivos con solo 2 líneas.

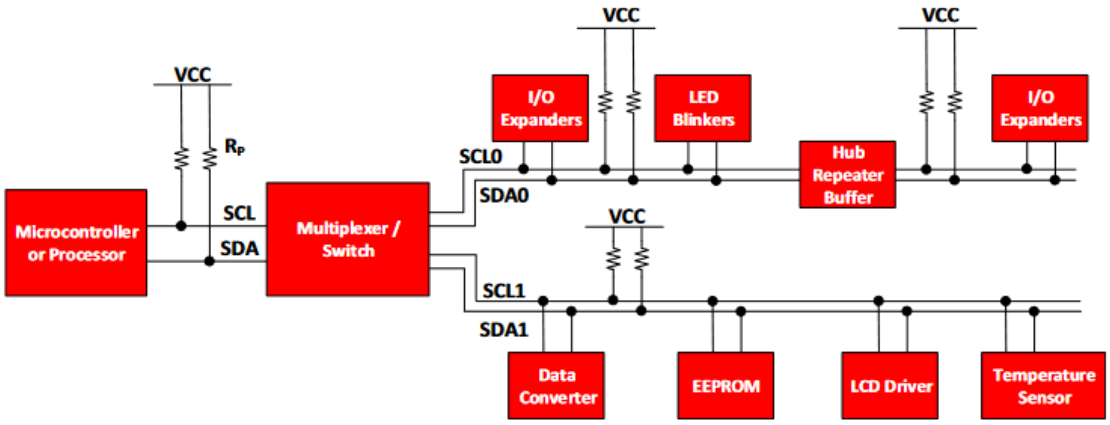


Figura 4.9 Ejemplo de conexión de varios dispositivos en el bus I2C [Crédito Texas Instruments]

Las líneas usadas en la comunicación I2C son:

- **SCL:** Es la línea donde el maestro genera la señal de reloj. Debe estar conectada a VCC con una resistencia “Pull-up”.
- **SDA:** Es la línea donde se intercambian los datos. Debe estar conectada a VCC con una resistencia “Pull-up”.

El procedimiento general para que un maestro se comunique con un esclavo es el siguiente:

1. Si el maestro quiere enviar datos al esclavo:
    - Envía una condición START y la dirección del esclavo.
    - Envía los datos al esclavo.
    - Termina la transmisión con una condición de STOP.
2. Si el maestro quiere leer datos del esclavo:
    - Envía una condición de START y la dirección del esclavo.

- Envía la dirección del registro del esclavo de donde quiere leer los datos.
- Recibe los datos del esclavo.
- Termina la transmisión con una condición de STOP.

Las condiciones START y STOP son condiciones únicas, que no pueden ocurrir de ninguna manera excepto que el maestro las produzca intencionalmente. La condición START se presenta cuando hay una transición de “alto a bajo” en la línea SDA mientras SCL esté “alto”. La condición STOP se presenta cuando hay una transición de “bajo a alto” en la línea SDA mientras SCL esté “alto”. A continuación, se ejemplifica lo anterior:

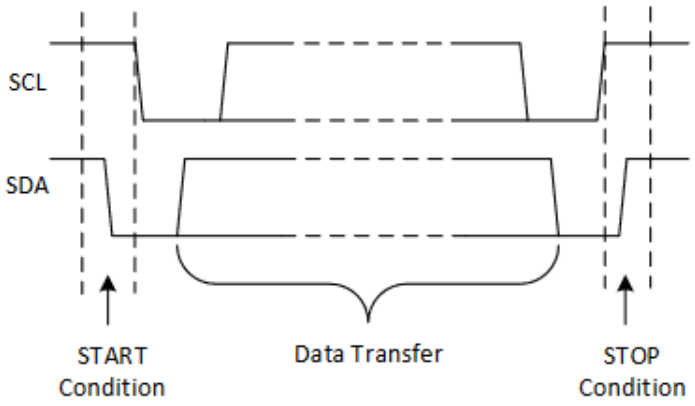


Figura 4.10 Condiciones de START y STOP en la comunicación I2C [Crédito Texas Instruments]

Cada bit es transferido durante cada pulso del reloj. La transmisión se realiza por bloques de un byte y un bit de reconocimiento (ACK), que sirve para distinguir si el último byte enviado fue recibido exitosamente o no. Cada byte transferido puede conformarse de: a) La dirección del esclavo y un bit de lectura o escritura (la dirección usualmente es de 7 bits, aunque pueden usarse direcciones de 10 bits [31]), b) La dirección de un registro, c) Datos leídos o escritos. Los datos son transferidos con el bit más significativo (MSB) primero. Cualquier cantidad de bytes pueden ser transmitidos entre una condición de START y STOP. Los datos en la línea SDA debe mantenerse estable durante la fase en “alto” del reloj, ya que esos cambios de estado corresponden a las condiciones START y STOP. La *Figura 4.11* muestra un ejemplo de la transferencia de un byte:

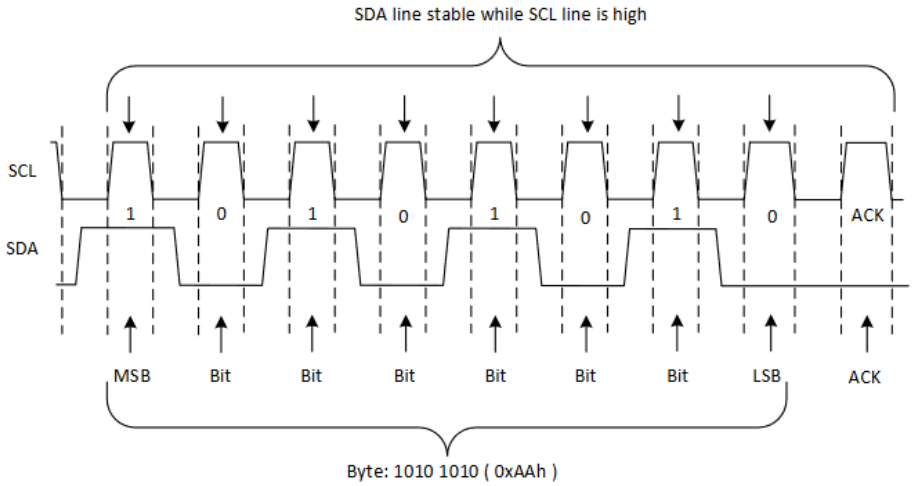


Figura 4.11 Ejemplo de la transferencia de un byte de datos con I2C [Crédito Texas Instruments]

Las principales ventajas de la comunicación I2C son: Provee buena comunicación entre dispositivos que se comunican constantemente y solo se requieren dos líneas para realizar la comunicación, sin importar la cantidad de dispositivos siendo controlados. Su principal desventaja es que la velocidad de transferencia es baja.

Para más información sobre I2C referirse a [32].

4.4 GPS

El sistema de posicionamiento global (GPS por sus siglas in inglés) es un sistema de localización que se usa frecuentemente para la navegación. El GPS se compone de tres elementos: los satélites en órbita, las estaciones terrestres de seguimiento y control, y los receptores GPS se los usuarios.

El objetivo del GPS es averiguar la posición de un usuario que tiene un receptor GPS, la posición está definida por su latitud, longitud y altitud. Para esto, se debe medir la distancia del usuario a un mínimo de 4 satélites, a este proceso se le conoce como “Trilateración”. La medición de la distancia entre el usuario y un satélite se obtiene multiplicando el tiempo de vuelo de la señal emitida por el satélite por su velocidad de propagación. El tiempo de vuelo es el tiempo que tarda en viajar la señal del satélite al usuario [33], para medir esta diferencia de tiempo se necesita que los relojes de ambos sistemas estén sincronizados.

Si los relojes en los satélites y los de los receptores tuvieran la misma precisión sólo se necesitarían 3 satélites para averiguar la posición del usuario, pero debido a la discrepancia en precisión entre los relojes en los satélites y los de los receptores GPS, se debe tener un cuarto satélite para ayudar con la corrección de tiempo y distancia.

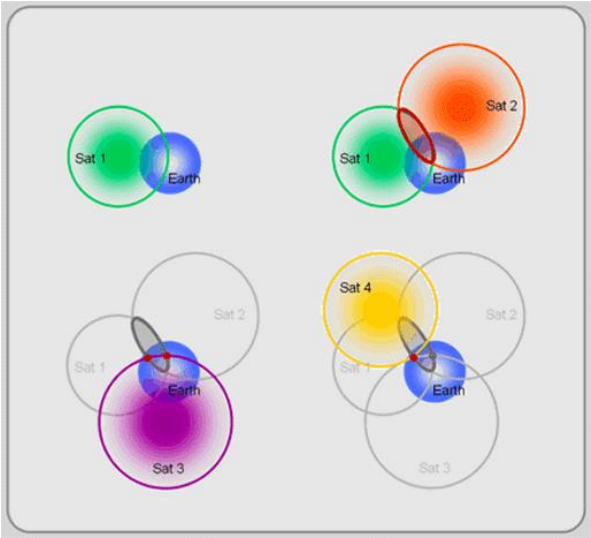


Figura 4.12 Esquema del cálculo de la posición de un receptor GPS por medio de trilateración [Crédito Giscommons.org]

Muchos de los receptores GPS utilizan el estándar NMEA para mostrar la información que obtienen. El formato NMEA se basa en líneas de datos separados por comas, por ejemplo:

```
$GPGLL,4916.45,N,12311.12,W,225444,A,*1D
```

La línea anterior contiene lo siguiente:

- GLL: Identificador de sentencia
  - Posición geográfica, latitud y longitud.
- 4916.46,N: Información de latitud
  - 49 grados, 16.45 minutos, Norte
- 12311.12,W: Información de longitud
  - 123 grados, 11.12 minutos, Oeste
- 225444: Hora en la que se fijó (se fija cuando puede ver 4 o más satélites)
  - 22:54:44 UTC
- A: Validez de la información
  - A (valido), V (invalido)
- *1D: Suma de control (checksum)

4.5 Acelerómetro, Giroscopio y Magnetómetro

Los acelerómetros, giroscopios y magnetómetros actualmente utilizan la tecnología MEMS (sistemas micro electromecánicos por sus siglas en inglés). Los acelerómetros son dispositivos que miden fuerzas de aceleración, ya sea estática (la fuerza de la gravedad) o dinámica (al mover el acelerómetro). Suelen ser usados para determinar el ángulo de inclinación de un dispositivo con respecto a la superficie terrestre y para identificar la manera en la que se mueve un dispositivo.

Existen diferentes maneras de hacer que un acelerómetro funcione, las más comunes usan: 1) El efecto piezoeléctrico, contienen estructuras microscópicas que, al ser estresadas por fuerzas de aceleración, producen voltajes proporcionales a la aceleración y 2) Cambios en capacitancia, donde se encuentran dos estructuras cercanas con una capacitancia establecida, de tal forma que, al presentarse aceleraciones que causen que se muevan las estructuras, haya un cambio de capacitancia proporcional a la aceleración [34].

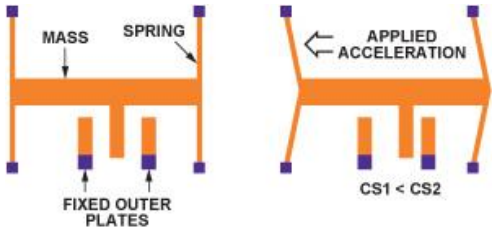


Figura 4.13 Estructura de un acelerómetro que mide cambios de capacitancia proporcionales a la aceleración aplicada [Crédito Analog.com]

Los giroscopios son dispositivos que miden velocidad angular. Suelen usarse para el seguimiento del movimiento de un dispositivo, en cámaras digitales para la estabilización de imágenes, en drones para ayudar en el sistema de navegación, entre muchas otras.

La mayoría de los giroscopios utilizan cambios en capacitancia para poder hacer mediciones y tienen una estructura parecida a la mostrada en la ilustración anterior. Los cambios en capacitancia se presentan cuando dos masas son sometidas a una velocidad angular, el efecto Coriolis ejerce una fuerza en direcciones opuestas sobre las masas, causando un movimiento que genera el cambio de capacitancia [35].

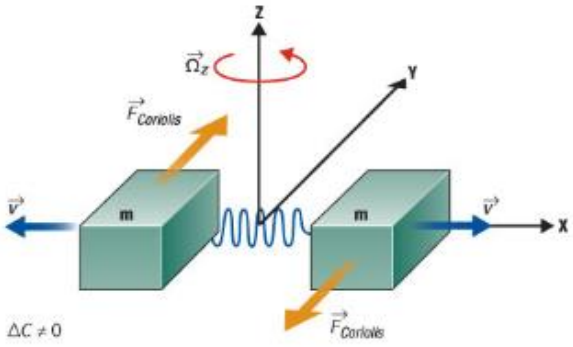


Figura 4.14 Representación de la funcionalidad de un giroscopio sometido a una velocidad angular [Crédito Electroiq.com]

Los magnetómetros miden la fuerza de campos magnéticos. Suelen usarse para monitorear los efectos del viento solar en el campo magnético terrestre, detectar lugares con firmas magnéticas o para saber la orientación de un dispositivo.

Principalmente se usan dos formas para medir los campos magnéticos: 1) Con el efecto Hall, aplicando una corriente a una placa conductora y exponiéndola a un campo magnético, se perturba el flujo de electrones provocando que se concentren de un lado de la placa generando un voltaje proporcional a la fuerza del campo magnético, 2) Con el efecto magnetoresistivo, con el uso de materiales que cambian su resistencia con la presencia de campos magnéticos.

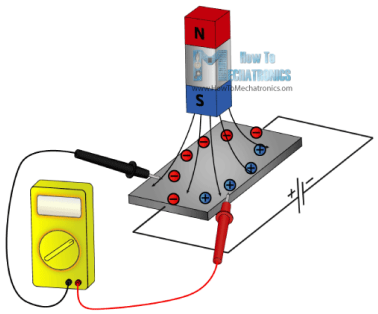


Figura 4.15 Representación del efecto Hall sobre una placa conductora [Crédito Howtomechatronics.com]

4.6 Sensor de presión y temperatura

Un sensor de presión como el BMP180 utiliza tecnología piezoresistiva. Esta tecnología se basa en el uso de materiales que cambian su resistencia cuando son sometidos a compresión o deformación, el material piezoresistivo se acomoda sobre un diafragma de tal manera que se pueda detectar la presión [36].

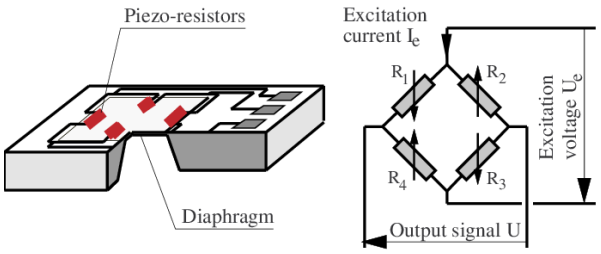


Figura 4.16 Ejemplo de sensor de presión piezoresistivo [Crédito Researchgate.net]

Las lecturas de temperatura se obtienen midiendo la resistencia de un componente previamente caracterizado (la variación en la resistencia es proporcional al cambio de temperatura), el componente resistivo se calibra para cierto rango de temperatura con el fin de obtener una mejor precisión en las lecturas.

4.7 Tarjeta microSD

Una tarjeta SD (Secure Digital) es una pequeña tarjeta que permite guardar información en dispositivos portátiles. Las tarjetas SD se diferencian por su medida,

su capacidad de almacenamiento y la velocidad con la que se transmiten los datos. La microSD es el formato más pequeño de las tarjetas SD.

Las tarjetas SD son un tipo de memoria flash (memoria de estado sólido, no es volátil y puede ser reprogramada eléctricamente), con componentes NAND (que puede ser leído o escrito por bloques). Cada tarjeta tiene un microcontrolador que permite la transferencia de información entre la tarjeta y otro dispositivo.

Gracias a su pequeño tamaño y alta capacidad de almacenamiento, las tarjetas microSD son usadas con una variedad amplia de dispositivos que tienen una ranura especial para la microSD: teléfonos inteligentes, tabletas, computadoras y dispositivos para desarrollo de proyectos electrónicos, entre ellos la Raspberry Pi. Existen otras maneras de conectar una tarjeta microSD a un dispositivo que no posee la entrada adecuada: Lectores de tarjetas especiales, adaptadores USB o por medio de SPI.

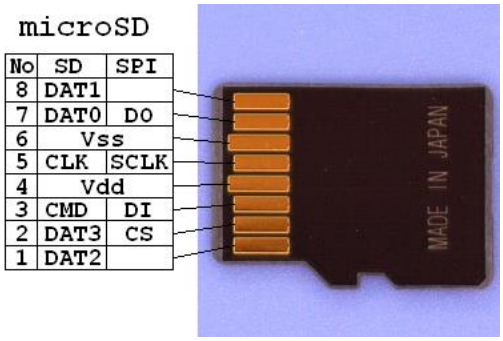


Figura 4.17 Explicación de la función de los pines en una tarjeta microSD [Crédito Elm-chan.org]

La comunicación con la SD se puede realizar en uno de dos modos: el modo SD o el modo SPI. Por default, la SD opera en el modo SD, no obstante, se puede cambiar la operación a modo SPI. Para esto, se debe iniciar una secuencia de energización (Con DO y CS en 1 lógico, generar al menos 74 ciclos de reloj en SCLK) y enviar el comando CMD0 por DI (con CS en 0). Lo hace que la SD se resetee y se ponga en modo SPI.

Después de entrar en modo SPI se deben leer las especificaciones de la tarjeta, inicializarla y, por último, realizar la transferencia de datos [37].

4.8 Módulo XBee

Los módulos XBee son pequeños dispositivos que transmiten y reciben datos inalámbricamente usando señales de radio (la mayoría opera a 2.4 GHz, pero hay algunos que operan a 900 MHz), tienen un grupo de protocolos de comunicación para diferentes arreglos de transmisión inalámbrica de datos, además existen distintos modelos que mantienen el posicionamiento de los pines para poder

intercambiar módulos fácilmente y aumentar la capacidad de actualización de proyectos electrónicos.

El módulo XBee es un transmisor inalámbrico muy popular debido a que envía y recibe información por medio de un puerto serial, lo que le brinda una amplia compatibilidad con diferentes grupos de microcontroladores y computadoras. Además, se puede tener una red de varios módulos comunicándose entre sí o simplemente tener un par de ellos intercambiando información. Se pueden usar para una variedad de tareas desde implementar domótica en una casa hasta controlar inalámbricamente un robot [38].

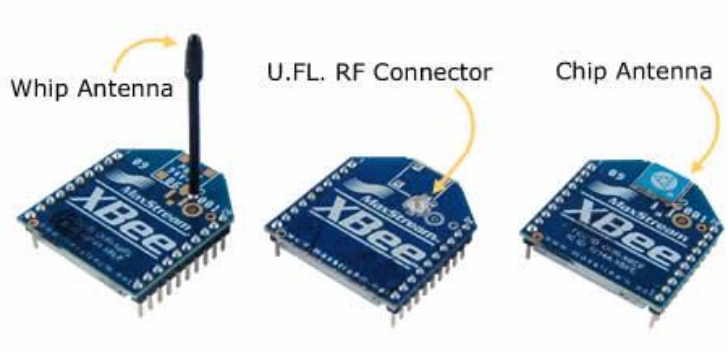


Figura 4.18 Diferentes tipos de antena para el módulo XBee [Crédito XBee.cl]

5 Metodología

La computadora de a bordo es uno de los sistemas que se están desarrollando en el proyecto “Diseño y construcción de un prototipo de robot móvil semiautónomo para exploración de terrenos similares a la superficie marciana”, en el cual se busca diseñar y construir un prototipo de rover, este proyecto está siendo desarrollado por un grupo de trabajo conformado por integrantes de la Universidad Nacional Autónoma de México y de la Universidad Politécnica de Santa Rosa Jauregui. Se obtuvieron los recursos, con el fin de adquirir los materiales para el desarrollo del rover, mediante el *Programa de apoyo a nuevos talentos científicos y tecnológicos 2018*, programa financiado por el CONCYTEQ.

El desarrollo del rover se dividió en seis áreas, con el fin de equilibrar la carga de trabajo y aprovechar las habilidades de los integrantes del equipo. La división quedó definida de la siguiente manera:

1. Sistema de la computadora de a bordo
2. Sistema del brazo robótico
3. Sistema de la visión computacional
4. Sistema de potencia
5. Sistema de propulsión
6. Diseño estructural del rover

Cada división tiene sus objetivos y retos específicos, pero la meta final de todos es la construcción del prototipo de rover para hacer pruebas, realizar mejoras y eventualmente llevarlo a competir a la URC.

La OBC, por su naturaleza, tendrá que interactuar con los demás sistemas del rover, es por eso que, el primer paso fue reunirse y platicar con los integrantes responsables del desarrollo de cada sistema para tener una mejor idea de cómo interactuaría la OBC con los sistemas. Además, en estas reuniones se creó el material que posteriormente se presentaría en el proceso de selección de los proyectos que serían apoyados por el programa *Nuevos Talentos 2018.*

5.1 Explicación y análisis de los sistemas del rover

De las seis áreas del rover que están en desarrollo, cinco sistemas son de naturaleza electrónica y a estos se les refiere en este trabajo como: computadora de a bordo, brazo robótico, visión, potencia y propulsión. Con eso clarificado, a continuación, se hace una breve explicación de las funciones de cada sistema y, posteriormente, se condensa la información en la *Tabla 5-1*.

La computadora de a bordo se encarga de coordinar y controlar los demás sistemas del rover, recibir y transmitir información a la estación terrena, monitorear las condiciones de operación del rover y ejecutar las acciones necesarias para completar las tareas asignadas.

El sistema del brazo robótico se encargará de diseñar y construir un brazo robótico con cuatro grados de libertad: tres articulaciones y un efector final. El sistema debe ser capaz de: posicionar el brazo en un punto deseado, reportar la posición actual del efector final, mover cada articulación independientemente, activar su efector final, detener su movimiento por si se presenta un imprevisto y regresar a una posición de espera con un menor consumo energético.

El sistema de visión se encargará de implementar los algoritmos de visión computacional con un Kinect. El sistema debe analizar las imágenes para: discernir si se puede avanzar (o hay un obstáculo) y calcular una serie de pasos que el rover puede seguir para esquivar un obstáculo que se encuentre enfrente (obtener un plan de navegación).

El sistema de potencia será el responsable de manejar la alimentación de todos los sistemas del rover. Este sistema debe: alimentar o cortar la alimentación de un sistema especificado, reportar el nivel de carga de las baterías, reportar la relación de carga contra descarga y medir la temperatura de operación de las baterías.

El sistema de propulsión será el responsable de controlar las 6 ruedas de rover (4 con capacidad de cambiar de orientación y 2 que solo podrán avanzar), brindándole la movilidad. El sistema podrá cambiar y reportar la orientación de las 4 ruedas de sus extremos, tendrá un dispositivo para medir la distancia recorrida y contará con sensores para medir el consumo energético de cada motor.

Al analizar las propuestas de las funciones de cada sistema se obtuvo la siguiente información sobre las posibles capacidades y limitaciones de cada sistema:

Tabla 5-1 Parámetros relevantes para la interacción de los sistemas y la computadora de a bordo

Sistema	**Funciones/Capacidades**	**Observaciones**
Brazo robótico	- Posicionar el brazo en un punto dado - Detectar posición actual - Rotar articulaciones independientemente - Activar efector final - Detener operación - Regresar a posición default	- No se proporcionaron límites en el rango de los movimientos que será capaz de realizar - Aún falta determinar que parámetros son relevantes para revisar la integridad del brazo

Visión	- Reconocer si hay obstáculos frente al rover - Obtención de plan de navegación	- No se especifica una distancia de seguridad para realizar maniobras - No se especifica que tanto tiempo tomará calcular el plan de navegación - Aún falta determinar que parámetros son relevantes para revisar la integridad del sistema - Se asume que podrá calcular un plan de navegación: Con visión directa de un objetivo o asumiendo que la posición objetivo está justo enfrente (detrás de los obstáculos que pueda haber). Con el plan de navegación conteniendo los pasos necesarios para rodear obstáculos y avanzar hacia el objetivo
Potencia	- Activar o desactivar un sistema - Reportar los parámetros de integridad del sistema	- Se debe tener una alimentación aislada de la alimentación de los motores
Propulsión	- Reportar y cambiar la orientación de 4 ruedas - Medir la distancia recorrida - Avanzar una distancia a un ángulo proporcionado	- No está establecida la velocidad a la que avanzará el rover, o si se va a poder cambiar la velocidad - El rover será capaz de girar sobre su propio eje - No se especifica si las ruedas podrán cambiar su sentido de giro, se hace la suposición que sí.

5.2 Análisis de requerimientos de la computadora de a bordo

Tomando en cuenta las funciones que se han descrito, se establece que la computadora de a bordo debe:

- Poder controlar los subsistemas del rover (visión, brazo robótico, propulsión y potencia).
- Tener la capacidad de transmitir información inalámbricamente a la estación terrena.

- Contar con dispositivos que brinden un panorama general del estado de operación del rover (sensores, un dispositivo para localización y algún tipo de almacenamiento extraíble)

Existen varias soluciones para realizar la transmisión de información de manera inalámbrica, sin embargo, al considerar los lineamientos de la URC, se encuentra la limitación a bandas de 900 MHz o 2.4 GHz [15]. Lo que sugiere dos opciones: usar una red de área local (LAN por sus siglas en inglés) o usar un módulo XBee. La red de área local permitiría tener altas velocidades de transferencia, pero con un mayor consumo energético. Por el otro lado, el módulo XBee es muy simple de usar, hay un extenso repertorio de guías que explican cómo usarlo, su precio no es muy elevado, no ocupa mucho espacio y su consumo energético es bajo, aunque su velocidad de transferencia es relativamente baja.

Para la selección de los demás dispositivos del rover, primero se tuvo que definir qué protocolo de comunicación se usaría para el control de los subsistemas del rover. Al tener definido el protocolo, se seleccionaron los dispositivos para obtener el estado de operación, basándose en la compatibilidad con el protocolo y la computadora de placa única con la que se desarrollaó la computadora de a bordo.

5.2.1 Control de subsistemas del rover

Para el control de los subsistemas del rover se propuso usar una serie de caracteres como los comandos de control, cuya estructura contenga: la procedencia del mensaje, el identificador del comando, un conjunto de valores separados por comas y al final una suma de control (checksum) para verificar la integridad del mensaje.

$$\sim PP{:}CMD\_CMD,X_1,X_2,\ldots,X_n{*}SC$$

Donde:

Tabla 5-2 Explicación de estructura de los comandos para la comunicación entre la OBC y los subsistemas[1]

~	Carácter de inicio de mensaje
PP	Procedencia del mensaje, puede ser: CA, VI, PO, PR o BR
:	Carácter que separa la procedencia y el comando
CMD_CMD	Comando (más adelante se encuentra la lista de comandos)
,	Carácter para separación de datos
$X_1 \ldots X_n$	Datos relevantes para el comando
*	Carácter de fin de mensaje

[1] Los caracteres se manejan en código ASCII y cuando se transmiten, se envían por bytes; por ejemplo, una 'A' se enviaría como 1000001_2 (65_{10} en código ASCII). Consultar *Tabla del código ASCII*

SC	Suma de control, se usa una comprobación longitudinal de redundancia (LRC por sus siglas en inglés) aplicando un XOR sobre los caracteres entre '~' y '*'

Se decidió que los comandos fueran una secuencia de caracteres en código ASCII para que fueran menos vulnerables a errores por cambios en bits. Por ejemplo, si un comando fuera de un byte (en vez de 7) sería más probable que algunos cambios de bits pasaran la verificación de la suma de control y causaran algún problema con el funcionamiento de la computadora.

Teniendo definida la estructura general y utilizando los parámetros de la *Tabla 5-1*, se hizo la propuesta de los comandos para cada sistema. A continuación, se presenta la lista de comandos para cada sistema, la descripción de su función y la respuesta esperada. Para facilitar la visualización:

- Se usa el carácter '**>**' para representar el comando enviado desde la computadora de a bordo y '**<**' para representar la respuesta del subsistema.
- Los posibles valores de los datos en los comandos se especifican entre paréntesis, por ejemplo, si el dato solo puede ser 'a', 'b' o 'c' se pone (a/b/c) o si se espera que el valor esté en un rango entre o y 100 se pone (0-100).
- Todos los subsistemas comparten la misma respuesta si detectan un error en la suma de control o si el comando recibido no coincide con ninguno de los de su sistema: **< ERR_CMD**.

Visión

Caracteres de procedencia: VI

Comandos:

- **> CLC_PNV**
  Pide iniciar el cálculo del plan de navegación.
- **< CLC_PNV,N**
  Regresa el estimado de tiempo (en segundos) que tarda en calcular el plan de navegación.
- **> CAN_MOV,D**
  Pregunta si es seguro que el rover avance D centímetros.
- **< CAN_MOV,(0/1)**
  Responde 1 si es seguro que avance o 0 si no es seguro (hay un obstáculo).
- **> GET_PNV**
  Solicita el plan de navegación.
- **< GET_PNV,A_1 D_1,A_2 D_2,…,A_N D_N**
  Regresa la lista de ángulos y distancias del plan de navegación, si regresa una lista vacía (sin ningún valor) significa que aún no está listo el cálculo.
- **> GET_HSK**

Solicita el estado de operación del sistema (Housekeeping).

- **< GET_HSK**
  Regresa el estado de operación, aún no se define lo que contendría.

Ejemplo:

~CA:GET_PNV*41

Potencia

Caracteres de procedencia: PO

Comandos:

- **> AOD_SIS,(0/1),(VI/PR/BR)**
  Activar (1) o Desactivar (0) el sistema especificado: VI, PR o BR (solo se puede activar o desactivar un sistema a la vez).
- **< AOD_SIS**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> GET_HSK**
  Solicita el estado de operación del sistema.
- **< GET_HSK,(0-100),(0.0 a 10.0),T,(0/1) (0/1) (0/1)**
  Regresa: El porcentaje de carga de las baterías, la relación carga y descarga (por ejemplo, 0.5 significa que se descarga dos veces más rápido de lo que se carga), la temperatura de las baterías en grados centígrados y el estado de los sistemas VI, PR, BR (en ese orden. 0 inactivo, 1 activo).

Ejemplo:

~CA:AOD_SIS,0,VI*4B

Propulsión

Caracteres de procedencia: PR

Comandos:

- **> GET_POS**
  Solicita la orientación (en grados) de las llantas con respecto al frente del rover.
- **< GET_POS,(0 a 90),(D/I),(0 a 90),(D/I),(0 a 90),(D/I),(0 a 90),(D/I)**
  Regresa la orientación de las cuatro ruedas exteriores y la dirección (Derecha o Izquierda). El ordenamiento de las ruedas se puede observar en la *Imagen del orden de las ruedas*.
- **> RST_CNT**
  Pide reiniciar contador de distancia.
- **< RST_CNT**

Regresa el comando recibido para indicar que se recibió correctamente.

- **> GET_DST**
  Solicita la distancia recorrida.
- **< GET_DST,D**
  Regresa la distancia recorrida (en centímetros).
- **> ROT_FWL,(0 a 90),(D/I)**
  Pide girar las llantas frontales de 0 a 90 grados hacia la derecha (D) o izquierda (I), los grados deben ser enteros (sin punto decimal).
- **< ROT_FWL,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> ROT_BWL,(0 a 90),(D/I)**
  Pide girar las llantas traseras de 0 a 90 grados hacia la derecha (D) o izquierda (I) , los grados deben ser enteros (sin punto decimal).
- **< ROT_BWL,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> MOV_RVR,(-999 a 999),(-180 a 180)**
  Pide avanzar de -999 a 999 centímetros, girando en un rango de -180 a 180 grados (los grados se cuentan de enfrente del rover: negativo es hacia la izquierda y positivo es hacia la derecha), ambas cantidades deben ser enteras (sin punto decimal).
- **< MOV_RVR,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> STP_NOW**
  Pide detener todas las acciones.
- **< STP_NOW**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> GET_HSK**
  Solicita el estado de operación del sistema.
- **< GET_HSK,C,CM,CM,CM,CM,CM,CM,CM,CM,CM,CM**
  Regresa el estado de operación: La corriente consumida por todos los motores y la corriente consumida por cada motor en amperes.

Ejemplo:

~CA:GET_DST*72

Brazo Robótico

Caracteres de procedencia: BR

Comandos:

- **> GET_POS**
  Solicita la posición del brazo.
- **< GET_POS,X,Y,Z,A**

Regresa la posición del brazo y el ángulo al que se encuentra su efector final.

- **> SET_POS,X,Y,Z,A**
  Pide posicionar brazo en el punto X, Y, Z y con el efector final en un ángulo A.
- **< SET_POS,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> ROT_ART,(1/2/3),G**
  Pide rotar la articulación (1, 2 o 3), G grados (-90 a 90). El orden de las articulaciones se puede observar en la *Imagen de las articulaciones del brazo robótico*.
- **< ROT_ART,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> ACT_EFN,(0/1)**
  Pide cerrar (1) o abrir (0) el efector final.
- **< ACT_EFN,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> STP_NOW**
  Pide detener todas las acciones.
- **< STP_NOW**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> RST_NOW**
  Pide regresar a la posición default.
- **< RST_NOW,T**
  Regresa el estimado de tiempo que tarda en realizar la acción.
- **> GET_HSK**
  Solicita el estado de operación del sistema.
- **< GET_HSK**
  Regresa el estado de operación, aún no se define lo que contendría.

Ejemplo:

~CA:ROT_ART,0,45*58

Ya que los comandos no son muy largos, máximo unos 50 bytes, no se necesita un protocolo de comunicación con mucha velocidad de transmisión. Es por esa razón que se optó por el uso de: I2C para la comunicación con los subsistemas del rover (además de la ventaja que provee al solo requerir 2 cables) y el módulo XBee para la comunicación con la estación terrena.

Con todo lo anterior definido, se procedió a asignar las direcciones de los sistemas del rover: computadora de a bordo (08_{16}), visión (09_{16}), potencia ($0A_{16}$), propulsión ($0B_{16}$) y brazo robótico ($0C_{16}$). A continuación, se muestra el esquema general de comunicación.

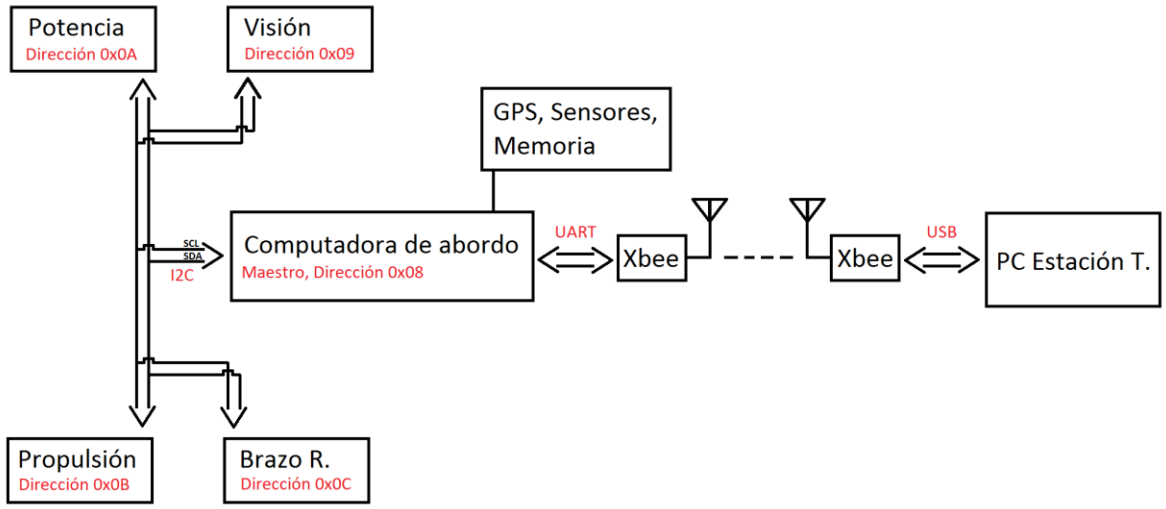


Figura 5.1 Esquema general de comunicación de la OBC: con los dispositivos del rover y con la estación terrena

5.2.2 Propuesta de materiales

Teniendo definida la forma en la que se interactúa con el resto de los sistemas del rover y los protocolos de comunicación a usar, el siguiente paso fue elegir los componentes que conforman a la computadora de a bordo:

Tabla 5-3 Componentes que se planean usar en la OBC

Dispositivo	**Descripción de función**	**Justificación**
Raspberry Pi 3 modelo B+	Plataforma en la que se implementa la computadora de a bordo	- Es un sistema que cuenta con una gran velocidad de operación. - Cuenta con una gran comunidad desarrollando software y hardware que facilita la implementación de proyectos electrónicos. - Es compatible con el protocolo I2C. - Es compatible con el protocolo SPI. - Posee 40 GPIOS. - Tiene dos chips de UART (aunque un está ocupado por el Bluetooth).
MicroSD de 16 GB	Memoria para la Raspberry Pi	- Es necesaria para el sistema operativo de la Raspberry Pi.

MicroSD de 16 GB	Memoria de almacenamiento extraíble	- No ocupa mucho espacio al conectar. - Es compatible con el protocolo de comunicación SPI.
Adaptador microSD	Permite conectar otra tarjeta microSD a la Raspberry Pi	- Facilita el conectar una segunda tarjeta microSD. - Trae los pines para comunicarse por medio de SPI.
BMP180	Sensor de temperatura y presión	- Es compatible con el protocolo de comunicación I2C.
LSM9DS0	Sensor: Acelerómetro, Giroscopio y Magnetómetro	- Es compatible con el protocolo de comunicación I2C.
Módulo GY-NEO6MV2	Módulo GPS para saber la posición del rover	- Compatible con la UART de la Raspberry Pi.
Módulo XBee S1 802.15.4	Permite transmitir información inalámbricamente	- Fácil de usar. - Bajo consumo energético. - Compatible con la UART de la Raspberry Pi.

5.3 Instalación del sistema operativo en la Raspberry Pi

Se escogió el sistema operativo Raspbian ya que, al utilizar un núcleo de Linux, cuenta con la capacidad de asignar prioridades a los procesos que se ejecutan en la CPU, lo que lo hace un sistema operativo en tiempo real. Otro punto positivo es que cuenta con una extensa cantidad de información en cuanto a cómo resolver problemas que se puedan presentar.

Para instalar Raspbian se siguieron los siguientes pasos:

- Descargar la imagen más reciente de Raspbian de la página: https://www.raspberrypi.org/downloads/raspbian/
- Descargar e instalar el programa Etcher (https://www.balena.io/etcher/).
- Insertar la tarjeta microSD a la computadora.
- Abrir el programa Etcher y se seleccionar el archivo .img o .zip que se descargó en el primer paso.
- Seleccionar la tarjeta microSD en la que se desea instalar Raspbian.
- Revisar las selecciones y hacer click en “Flash”.

Una herramienta muy útil para poder trabajar con la Raspberry de forma “headless” (sin tenerla conectada a un monitor, mouse o teclado) es el SSH (Secure Shell), que permite el acceso a la línea de comandos de forma remota desde otro dispositivo en la misma red. Para activar SSH en una Raspberry “headless”, se necesita otro dispositivo para crear un archivo llamado “ssh” en la partición “boot” de la tarjeta microSD (el archivo no debe tener ninguna extensión), finalmente al encender la Raspberry se podrá utilizar la herramienta SSH. Para más formas de activar SSH ver: https://www.raspberrypi.org/documentation/remote-access/ssh/README.md

5.4 Desarrollo del programa para la comunicación con los subsistemas

Antes de comenzar con el desarrollo del programa para la comunicación con los subsistemas del rover, fue necesario activar la interfaz I2C de la Raspberry Pi e instalar las herramientas para usar I2C con Python.

Activando la interfaz I2C

En la terminal, ejecutar el comando **`sudo raspi-config`** y seleccionar **`Interfacing Options S`**eleccionar la opción **`I2C`** y `activarla`.

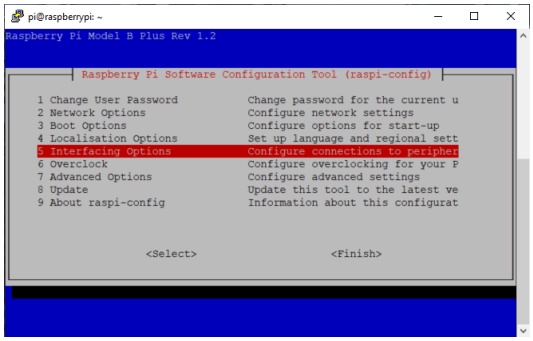


*Figura 5.2 Activación de la interfaz I2C en la Raspberry: Ejecutando **sudo raspi-config** con PuTTY*

Instalación de herramientas para usar I2C con Python

En la terminal, ejecutar los comandos **`sudo apt-get install -y python-smbus`** y **`sudo apt-get install -y i2c-tools`**. Al terminar de ejecutar los comandos anteriores, se ejecuta **`sudo i2cdetect -y 1`** para verificar su función.

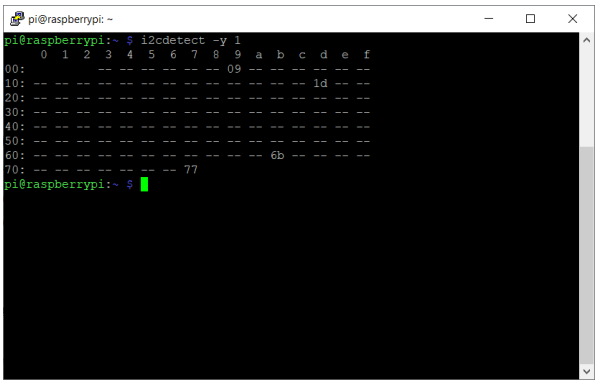


*Figura 5.3 Muestra de las direcciones de los dispositivos conectados al bus I2C de la Raspberry después de ejecutar el comando **i2cdetect -y 1***

Después de realizar lo anterior, se procede a desarrollar el *código para las primeras pruebas con I2C*, en las cuales se llevaba a cabo una comunicación I2C simple entre la Raspberry Pi y un Arduino Leonardo, con la Raspberry operando como el maestro y el Arduino como el esclavo. La prueba consiste en hacer que el usuario elija uno de tres comandos disponibles, se envía el comando por I2C, el esclavo reconoce el comando y realiza la tarea asignada a dicho comando (en la *Figura 5.4* se observa cómo se hace la conexión), con esto se busca familiarizarse con las características básicas del uso de la interfaz I2C. Es importante aclarar que, ya que la Raspberry Pi opera como el maestro y las líneas de la interfaz I2C (SDA y SCL) operan en modo "open-drain", no es necesario usar cambiadores de nivel de voltaje (la Raspberry y el Arduino operan con diferentes voltajes, pero esta configuración asegura que el nivel lógico sea de 3.3 V).

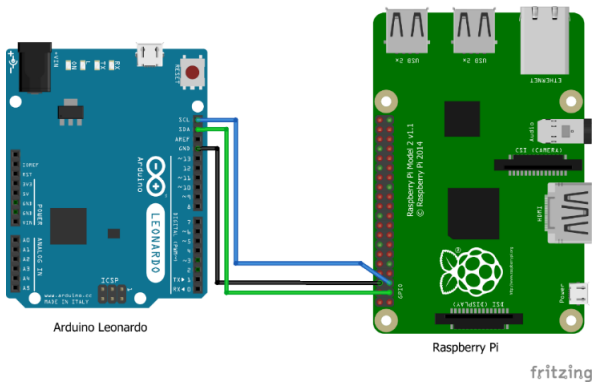


Figura 5.4 Diagrama de conexión entre el Arduino Leonardo y la Raspberry Pi

Tabla 5-4 Conexión del Arduino Leonardo y la Raspberry Pi

Arduino Leonardo	Raspberry Pi
PD0 - SCL	Pin 5 - SCL
PD1 - SDA	Pin 3 - SDA
GND	Pin 6 - GND

Con el conocimiento básico de cómo se usa la interfaz I2C para intercambiar información con un esclavo, se procedió a realizar el diseño de la estructura del código, que describiría los sistemas del rover, mediante el paradigma de programación orientada a objetos (para fomentar la reutilización del código y facilitar su continuo desarrollo).

```
class Subsistema(object):
    init = '~'
    origin = "CA"
    end = '*'
    address = 0x00
    cmd_list = []

    def __init__(self, direccion):...

    def get_comando(self, cmd):...

    def decode_resp(self, resp):...
```

Figura 5.5 Código en Python de la estructura de una clase que describe un subsistema del rover

En la Figura 5.5, se pueden observar las características que comparten todos los subsistemas del rover: Los símbolos propuestos en la estructura de los comandos para el control de los sistemas del rover (‘~’, ‘*’ y ‘CA’), la dirección del subsistema, una lista de comandos y algunos métodos para obtener los comandos o decodificar las respuestas.

A partir de esa clase base, se desarrolló el código para cada subsistema, heredando las características comunes e implementando las características únicas de cada subsistema.

```python
class Vision(Subsistema):
    # Al modificar la lista de comandos (cmd_list), también se deben
    # modificar las constantes que de la línea de abajo
    cmd_list = ["CLC_PNV", "CAN_MOV", "GET_PNV", "GET_HSK"]
    clc_pnv, can_mov, get_pnv, get_hsk = range(len(cmd_list))
    avanzar_seguro = True
    plan_de_navegacion = []  # [[A1, D1], [A2, D2], ..., [Aj, Dj]]
    pnv_angulo, pnv_distancia = 0, 1
    housekeeping = []  # Aún no se define su estructura

    def __init__(self, direccion):...

    def decode_resp(self, resp):...

    def getcmd_clc_pnv(self):...

    def dec_clc_pnv(self, info):...

    def getcmd_can_mov(self, distancia):...

    def dec_can_mov(self, info):...

    def getcmd_get_pnv(self):...

    def dec_get_pnv(self, info):...

    def getcmd_get_hsk(self):...

    def dec_get_hsk(self, info):...
```

Figura 5.6 Código colapsado de la clase del subsistema de visión

En la *Figura 5.6* se observa la estructura de la clase del subsistema de visión, a continuación, se explica brevemente lo que representa cada elemento:

- **`cmd_list`**: La lista de comandos con los que se puede comunicar con el sistema.
- **`clc_pnv, can_mov, get_pnv, get_hsk`**: Variables para hacer más claro el código.
- **`sleep_times`**: La lista de tiempos que se tardaría el sistema de visión en ejecutar los comandos.
- **`avanzar_seguro`**: Parámetro que almacena la respuesta del comando **`CAN_MOV`**.
- **`plan_de_navegacion`**: Lista que almacena el plan de navegación obtenido con el comando **`GET_PNV`**.
- **`x, y`**: Variables para hacer más claro el código.
- **`housekeeping`**: Parámetro para almacenar el estado de operación del sistema de visión, ya que aún no se define que contendría se comentó '#???'.

- **`def getcmd…/dec…`**: Son el conjunto de métodos que sirven para obtener los comandos con el formato establecido y para descifrar las respuestas obtenidas.

Los demás sistemas siguen una estructura análoga al del sistema de visión, por eso que no se muestra el código de esas clases. No obstante, hay cuatro puntos que merecen ser explicados más a fondo ya que influyen de manera importante en el proceso de desarrollo del código.

```
def __init__(self, direccion):
    # type: (int) -> None
    """Inicializador del objeto Vision.

    Args:
        direccion: La dirección del sistema de visión.

    Returns:
        Nada.
    """
    super(Vision, self).__init__(direccion)
    if ARDUINO_SIMULATION:
        self.origin = "VI"
```

Figura 5.7 Código para ejemplificar el estilo de documentación de los métodos

El primer punto corresponde a la manera de documentar las funciones o métodos. Se usa el estilo 484 de las *Propuestas de Mejora de Python* (PEP), que establece una estructura para describir el funcionamiento del método o función. Para ejemplificar lo anterior, en la *Figura 5.7* se pueden observar las dos partes de la estructura para documentar un método: la primera parte es una línea que empieza con '**`# type:`**' en la que se especifica el tipo de datos de entrada (entre paréntesis) y el tipo de dato de salida (separados por los caracteres '**`->`**'); la segunda parte está contenida en las líneas entre los caracteres '**"""**', en esta parte se explica lo que hace la función, cuáles son los argumentos de entrada y qué datos regresa.

El segundo punto trata sobre la adición de las dos últimas líneas de código que se ven en la *Figura 5.7*, lo que modifica la procedencia de los comandos. Esto se hizo ya que fue necesario usar un Arduino que simulara las respuestas de los subsistemas (el desarrollo de los subsistemas aún se encontraba en una etapa temprana) y no había otra forma de discernir hacia que subsistema iban dirigidos los comandos. Esta característica se puede desactivar, una vez se comiencen las pruebas con los subsistemas reales del rover, asignando un valor **`False`** a la variable **`ARDUINO_SIMULATION`** en el código. En vez de que los caracteres de procedencia siempre sean '**CA**' (como se establece en la sección *5.2.1*), la modificación de la variable '**`origin`**' le da la información al Arduino para obtener el destino de los comandos que recibe, en la figura de ejemplo se puede ver que se cambia a '**VI**' que corresponde a un mensaje dirigido al sistema de visión.

El tercer punto sirve para explicar la manera en la que la OBC recupera las respuestas de los subsistemas, **el tercer y cuarto punto deben ser tomados en cuenta al momento de diseñar el módulo de comunicación de los**

subsistemas. Al realizar algunas pruebas, se encontró un error con la comunicación I2C, cuando la Raspberry Pi intentaba pedir más de un byte de información de respuesta al Arduino, se presentaba un error (*Figura 5.8*). Luego de un tiempo de investigación, se concluyó que se debía a los problemas que tiene la Raspberry Pi cuando algún dispositivo realiza *clock stretching* [39]. Esto ocurre cuando un esclavo necesita más tiempo para responder al maestro, por lo que el esclavo baja la línea SCL a tierra hasta que está listo para responder. Al identificar este problema se decidió que, para pedir información a los subsistemas, la computadora de a bordo debe recuperar la respuesta un byte a la vez y el primer byte debe corresponder al número de bytes del resto de la respuesta. Por ejemplo, suponiendo que la computadora de a bordo envió el comando **`~BR:GET_POS*6F`** (con la procedencia BR por lo discutido en el segundo punto) y el Arduino generó una respuesta **`~BR:GET_POS,52,17,81,43*60`**, al momento en que la computadora realice la consulta del primer byte, el Arduino regresa el número **26** ,que es el número de bytes que contiene la respuesta; con esta información, la computadora de a bordo realizará ese número de consultas para obtener la respuesta completa. Ya que los comandos no son muy largos (máximo 50 bytes) y que la velocidad del bus I2C del Raspberry Pi es de 50 Kbit/s, no se espera que este cambio repercuta perceptiblemente en el intercambio de información entre la computadora de a bordo y los subsistemas.

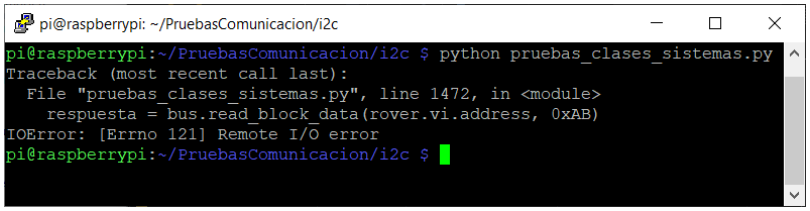


Figura 5.8 Error en la comunicación I2C por clock stretching

El cuarto punto se relaciona a la estructura con la que se envían los comandos, ya que existe una limitación con la biblioteca utilizada para la comunicación I2C. La limitación solo permite enviar mensajes en bloques de máximo 32 bytes, por lo que, si el comando que envía la OBC es de más de 32 caracteres, entonces se enviará en dos o más partes; los subsistemas deben encargarse de concatenar los bloques que reciban y verificar que cumplan con la estructura definida en la sección *5.2.1*.

5.4.1 Simulador de los subsistemas

Como se ha mencionado previamente, para simular la respuesta de los subsistemas se utilizó un Arduino Leonardo. Para instalar el IDE de Arduino: Se descargó el instalador de la página: https://www.arduino.cc/en/Main/Software, se ejecutó el instalador y se siguieron los pasos para terminar la instalación.

Para que el Arduino produjera respuestas, simulando a los subsistemas del rover, debe que ser capaz de: 1) Realizar comunicación con I2C, 2) Reconocer los comandos recibidos, y 3) Generar una respuesta siguiendo el formato establecido.

Para **implementar la comunicación con I2C** se utilizó la biblioteca Wire, la cual viene incluida en la instalación de Arduino. Con esta biblioteca se obtuvo la funcionalidad de recibir y enviar comandos por el bus I2C.

En la parte de **reconocer los comandos recibidos** se desarrolló un código que se encargaba de:

a) Checar la integridad del comando: En este paso se comprueba que el comando recibido cumpla con el formato predefinido y pase la suma de control. Si se encuentra algún error se reporta a la computadora de a bordo modificando la respuesta con el comando **`ERR_CMD`** y saltando a la parte de enviar respuesta.
b) Dirigir el comando al sistema para el que iba dirigido: Para esto se utilizaron los caracteres de procedencia del comando para verificar a qué subsistema iba dirigido el comando (VI, PO, PR o BR) y ejecutar el siguiente paso, si los caracteres no correspondían a ninguno los subsistemas se modifica la respuesta con el comando **`ERR_CMD`** y salta a la parte de enviar respuesta.
c) Ejecutar el comando: Aquí se compara el comando con la lista de comandos de dicho subsistema: i) Si no se encuentra en la lista, se modifica la respuesta con el comando **`ERR_CMD`** y se envía. ii) Si se encuentra en la lista, se pasa a la parte del código encargada de generar la respuesta.

En la parte de **generar una respuesta parecida a la que producirían los subsistemas del rover** el código se encarga de construir la respuesta del subsistema y enviarla. Se analizó cada uno de los comandos y se desarrolló el código que generaría una respuesta similar a la que se esperaría de cada comando, para algunos comandos la respuesta es la misma cadena de caracteres, para la mayoría, la respuesta es una serie de cantidades aleatorias acotadas a los parámetros esperados para el rover y para unos se usan los datos del comando para hacer algún cálculo simple para generar la respuesta.

```
}else if(strcmp(cmd, "GET_PNV") == 0){
  randm = random(3,6); // Puntos en el plan de navegación
  c2 = 0;
  for(x=0;x<randm;x++){
    if (random(0,10) < 5){
      resp[c+c2+(x*6)] = '-';
      c2++;
    }
    resp[c+c2+(x*6)] = char(random(1,10)+choffs);
    resp[c+1+c2+(x*6)] = char(random(1,10)+choffs);
    resp[c+2+c2+(x*6)] = ' ';
    resp[c+3+c2+(x*6)] = char(random(1,10)+choffs);
    resp[c+4+c2+(x*6)] = char(random(1,10)+choffs);
    resp[c+5+c2+(x*6)] = ',';
  }
  c += (x * 6) + c2 - 1;
}else if(strcmp(cmd, "GET_HSK") == 0){
  resp[c]='O';
  resp[c+1]='K';
  c = c + 2;
}else{
  // Hubo error en el comando
  badCmd[BAD_CMD] = true;
  resp[4]='E';
  resp[5]='R';
  resp[6]='R';
  resp[7]='_';
  resp[8]='C';
  resp[9]='M';
  resp[10]='D';
  c--; // Debería ser 11
}
```

Figura 5.9 Una parte del código del Arduino, esta parte ayuda con el reconocimiento de los comandos dirigidos al subsistema de visión

Como ejemplo, en *Figura 5.9* se observa una parte del código que se encarga de procesar el comando recibido y realizar cierta acción dependiendo de si reconoció el comando: **GET_PNV**, **GET_HSK** o cuando no coincide con ninguno de los comandos. Para el primer caso, la respuesta se genera obteniendo un número aleatorio de puntos que contendrá el plan de navegación (de 3 a 5 puntos) y se procede a generar valores aleatorios para esos puntos. Para el segundo caso, como aún no se define qué es lo que contendrá el Housekeeping del subsistema de visión, la respuesta solo contiene 'OK'. En el tercer caso, si el comando recibido no coincide con ninguno de los comandos de ese subsistema, se procede a modificar la respuesta con el valor del comando **ERR_CMD** para enviarlo y reportar a la computadora de a bordo que hubo un problema en la comunicación para volver a intentarla.

5.5 Adquisición de las mediciones de los sensores

Para obtener parámetros relevantes para la navegación y las condiciones de operación del rover, la computadora de a bordo recaba información de:

- Un sensor de temperatura.

  Sirve para determinar la temperatura en los alrededores del rover, esta temperatura puede ser relevante para la operación de los diferentes sistemas que conforman el rover, sin embargo, aún no están definidos los rangos de temperatura de operación de los sistemas.

- Un acelerómetro, giroscopio y magnetómetro

Servirá para auxiliar en la navegación, por el momento solo se planea usar para obtener la orientación del rover y así poder avanzar hacia la ubicación objetivo, aunque existan obstáculos bloqueando la vista directa al objetivo.

- Un módulo GPS

Servirá para determinar la ubicación del rover y auxiliar en la navegación, se prevé que la ubicación objetivo del rover se señale con una coordenada para que no sea necesario tener una línea de visión sin obstrucciones entre el rover y el objetivo.

5.5.1 Lecturas de temperatura y presión

Al principio se usó el sensor DHT11 para obtener la información de la temperatura. Para poder usar este sensor, se instaló su biblioteca ejecutando los siguientes comandos en la terminal:

- Se instala git: **`sudo apt-get install git-core`**
- Para poder agregar la biblioteca a Python: **`sudo apt-get install build-essential python-dev`**
- Se clona el repositorio git de la biblioteca de Adafruit para el sensor DHT11: **`git clone https://github.com/adafruit/Adafruit_Python_DHT.git`**
- Se ingresa a la carpeta del repositorio clonado: **`cd Adafruit_Python_DHT`**
- Se instala la biblioteca: **`sudo python setup.py install`**

Habiendo hecho esto, se procedió a desarrollar el *código para la prueba de obtención de temperatura*, estas pruebas consistieron mostrar la lectura de temperatura y humedad cada 3 segundos hasta terminar la ejecución del programa (presionando **`CTRL+C`**). A continuación, se especifica cómo se realizó la conexión.

Tabla 5-5 Conexión del sensor DHT11 a la Raspberry Pi

Sensor DHT11	Raspberry Pi
Pin 1 - DATA	Pin 7 - GPIO 4
Pin 2 - VDD	Pin 1 - 3.3 V
Pin 3 - GND	Pin 9 - GND

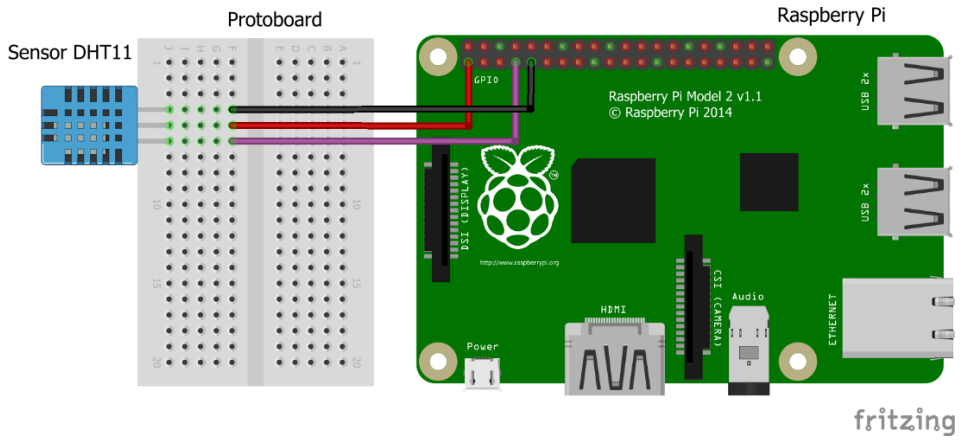


Figura 5.10 Diagrama de conexión del sensor DHT11 a la Raspberry Pi

Al conseguir el sensor BMP180, se procedió a instalar su biblioteca ejecutando los siguientes comandos en la terminal:

- Se clona el repositorio git de la biblioteca de Adafruit para el sensor BMP180: **`git clone https://github.com/adafruit/Adafruit_Python_BMP.git`**
- Se ingresa a la carpeta del repositorio clonado: **`cd Adafruit_Python_BMP`**
- Se instala la biblioteca: **`sudo python setup.py install`**

Se desarrolló un segundo *código para la prueba de obtención de temperatura*, con el cual se mostraba, solo una vez, las mediciones de: temperatura, presión, altitud y presión a nivel del mar. A continuación, se especifica cómo se realizó la conexión.

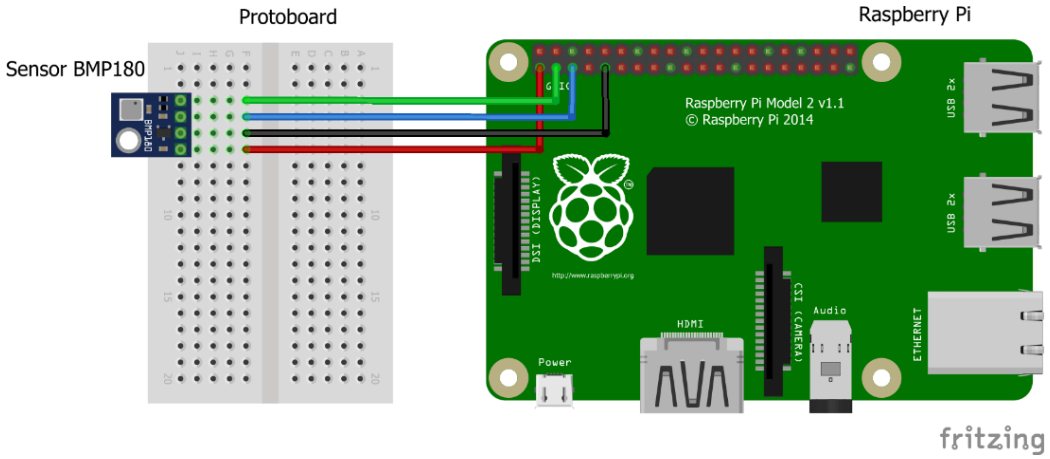


Figura 5.11 Diagrama de conexión del sensor BMP180 a la Raspberry Pi

Tabla 5-6 Conexión del sensor BMP180 a la Raspberry Pi

Sensor BMP180	Raspberry Pi
VIN	Pin 1 - 3.3 V
GND	Pin 9 - GND
SCL	Pin 5 - SCL

SDA	Pin 3 - SDA

5.5.2 Lecturas del acelerómetro, giroscopio y magnetómetro

Al principio, se realizaron pruebas con el sensor MPU6050, este sensor integra un acelerómetro y un giroscopio. Se usó el código que se encuentra en la página: https://github.com/Tijndagamer/mpu6050/blob/master/mpu6050/mpu6050.py, y se desarrolló el *código para la prueba con el MPU6050*, que consistió en repetir el muestreo de la aceleración y velocidad angular a las que era sometido el sensor. A continuación, se especifica cómo se realizó la conexión.

Tabla 5-7 Conexión del sensor MPU6050 a la Raspberry Pi

Sensor MPU6050	Raspberry Pi
VCC	Pin 1 - 3.3 V
GND	Pin 9 - GND
SCL	Pin 5 - SCL
SDA	Pin 3 - SDA

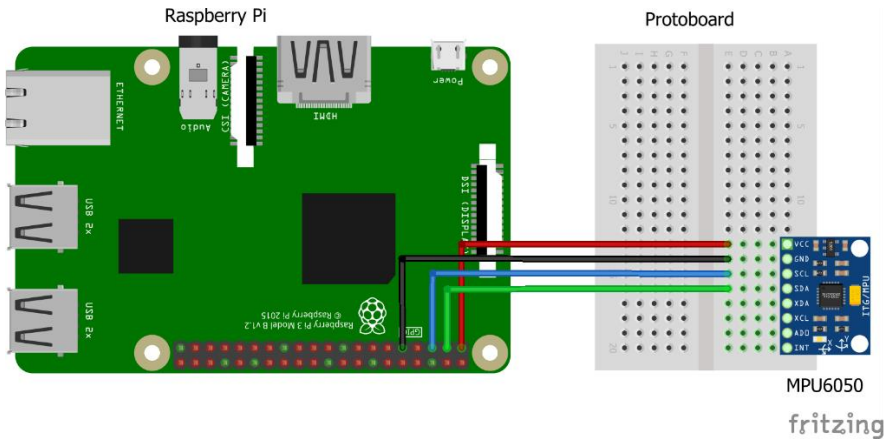


Figura 5.12 Diagrama de conexión del sensor MPU6050 a la Raspberry Pi

Posteriormente se utilizó el sensor LSM9DS0, que cuenta con acelerómetro, giroscopio y magnetómetro. Se optó por utilizar el código de la biblioteca de la página: https://github.com/jckw/Adafruit_LSM9DS0/blob/master/Adafruit_LSM9DS0/LSM9DS0.py, se escribió el *código para la prueba con el sensor LSM9DS0*, esta prueba consiste en mostrar las mediciones de la aceleración, la velocidad angular, la fuerza del campo magnético, el "pitch", el "roll" y su orientación compensada (la explicación de cómo se obtienen se encuentra en [40]); cada que se presionaba una tecla se actualizaban las mediciones. A continuación, se especifica cómo se realizó la conexión.

Tabla 5-8 Conexión del sensor LSM9DS0 a la Raspberry Pi

Sensor LSM9DS0	Raspberry Pi
3V3	Pin 1 - 3.3 V
GND	Pin 9 - GND
SCL	Pin 5 - SCL
SDA	Pin 3 - SDA

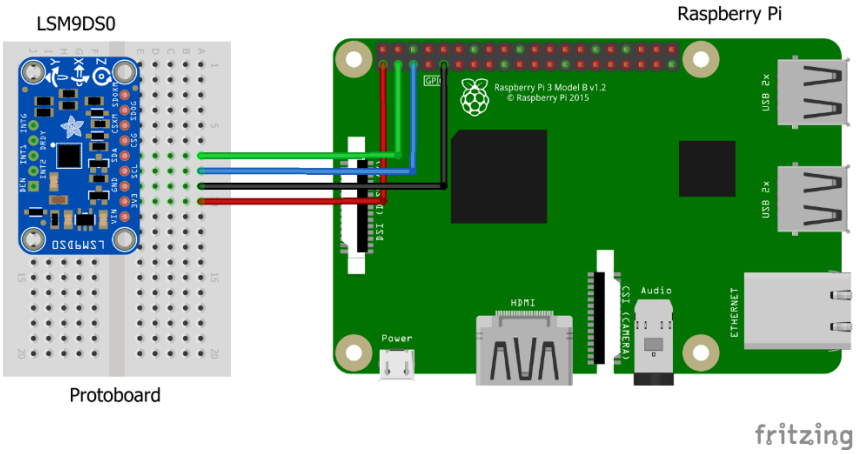


Figura 5.13 Diagrama de conexión del sensor LSM9DS0 a la Raspberry Pi

5.5.3 Lecturas de posición con el GPS

Para poder interactuar con el módulo GY-NEO6MV2, primero se debe desactivar el inicio de sesión de la terminal por el puerto serial de la Raspberry Pi, para esto se debe hacer lo siguiente:

- Se abre la herramienta de configuración de la Raspberry Pi ejecutando el comando **`sudo raspi-config`** en la terminal.
- Se selecciona la opción "**Interfacing Options**".
- Se selecciona la opción "**Serial**".
- Seleccionar "**No**" para desactivar el acceso a la consola mediante el puerto serial.
- Seleccionar "**Si**" para activar el puerto serial.

Para verificar el correcto funcionamiento, se procede a conectar el módulo GPS de la siguiente manera:

Tabla 5-9 Conexión del módulo GPS a la Raspberry Pi

Módulo GY-NEO6MV2	Raspberry Pi
VCC	Pin 1 - 3.3 V
RX	Pin 8 - TXD
TX	Pin 10 - RXD

GND	Pin 9 - GND

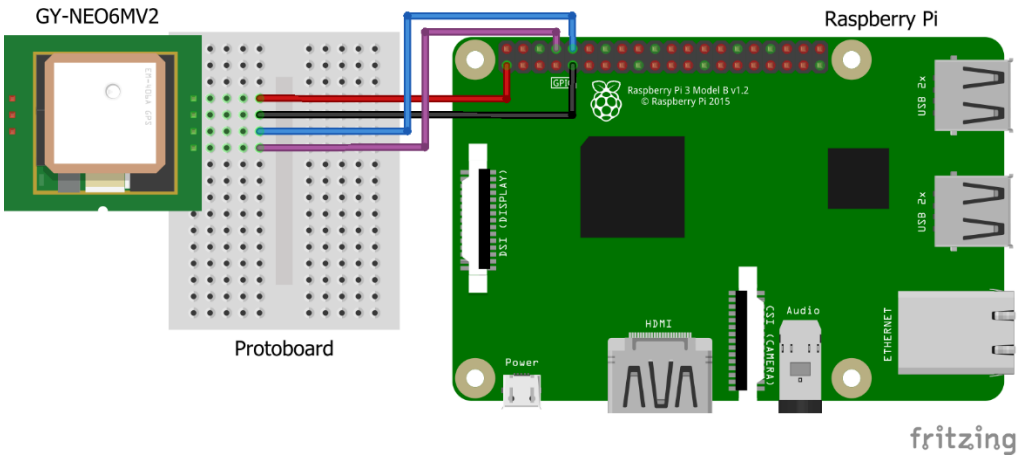


Figura 5.14 Diagrama de conexión del módulo GPS a la Raspberry Pi

Una vez realizada la conexión, al ejecutar el comando **`cat /dev/ttyS0`** en la terminal, se debe poder ver la información proveniente del módulo GPS (si se usara un modelo anterior a la Raspberry Pi 3 el comando que se debe usar es **`cat /dev/ttyAMA0`**). Después de comprobar que funcionara el módulo GPS, se procedió a desarrollar un *código de prueba del GPS* que consiste en leer los comandos NMEA con Python.

Sim embargo, de esta manera el módulo GPS proporciona mucha información que (para esta implementación) no se utiliza, por esto, se buscó en la hoja de datos del *GPS6MV2 (GPS)* la manera de desactivar los mensajes innecesarios y se hizo otro *código para desactivar los comandos NMEA* de tal forma que solo regresara los comandos GPGLL que contienen la información de la latitud, longitud y su validez.

5.6 Implementación del almacenamiento extraíble

Ya que la Raspberry Pi solo cuenta con una entrada para memorias microSD y que desde ésta se ejecuta el sistema operativo, se decidió utilizar la interfaz SPI para tener disponible una segunda entrada para el almacenamiento extraíble. En etapas tempranas del proyecto se pensaba simplemente usar uno de los cuatro puertos USB disponibles en la Raspberry Pi, pero se optó por el uso de una segunda tarjeta microSD ya que ocupa menos espacio, además de que Raspbian cuenta con maneras relativamente sencillas de implementarlo (usando árboles de dispositivos, que describen el hardware y la manera de utilizarlo).

El primer paso fue instalar la herramienta para construir el archivo de árbol de dispositivos, esto se hace ejecutando el comando **`sudo apt install device-tree-compiler`** desde la terminal. Posteriormente, se crea el *archivo para crear el árbol de dispositivos* y se compila con el comando **`dtc -@ -I dts -O dtb -o`**

`mmc_spi.dtbo mmc_spi.dts` ("mmc_spi.dts" es el nombre del archivo mencionado anteriormente y "mmc_spi.dtbo" es el nombre del archivo ya compilado). Luego se copia el archivo compilado al folder en donde el sistema operativo lo puede usar, para esto se ejecuta el comando **`sudo cp mmc_spi.dtbo /boot/overlays/`**. Para permitir que el archivo compilado sea ejecutado automáticamente al iniciar, se debe editar el archivo de configuración de inicio con el comando **`sudo nano /boot/config.txt`** y agregando al final del archivo lo siguiente: **dtoverlay=mmc_spi**

Se conectó la microSD a la Raspberry Pi con ayuda de un adaptador, las conexiones son las siguientes:

Tabla 5-10 Conexión de la tarjeta microSD a la Raspberry Pi

Adaptador microSD	Raspberry Pi
3V	Pin 1 - 3.3 V
GND	Pin 25 - GND
CLK	Pin 23 - SPI_CLK
DO	Pin 21 - SPI_MISO
DI	Pin 19 - SPI_MOSI
CS	Pin 24 - SPI_CE0

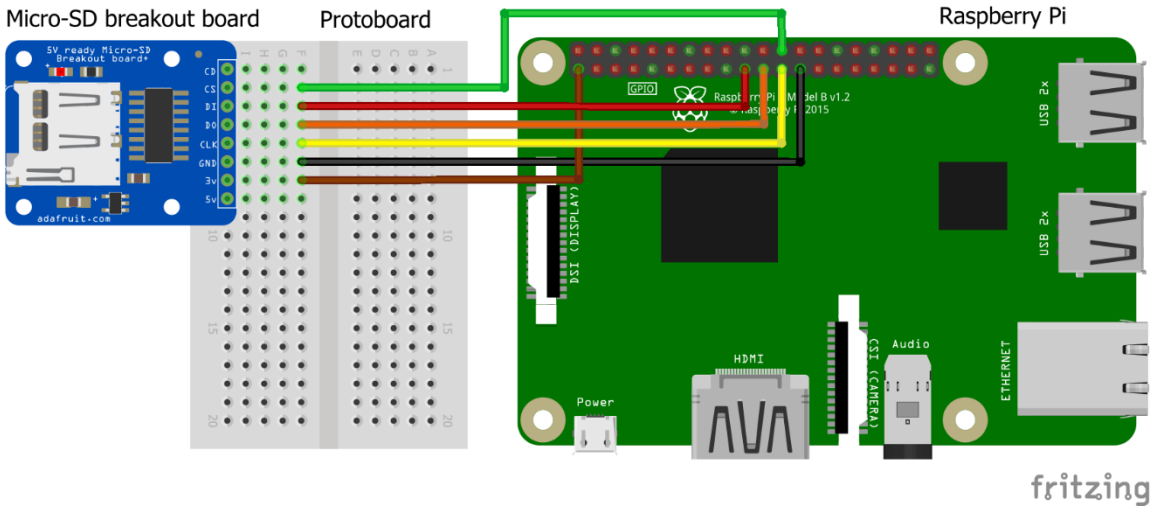


Figura 5.15 Diagrama de conexión de la tarjeta microSD a la Raspberry Pi

Con eso hecho, al reiniciar la Raspberry el sistema busca automáticamente las tarjetas SD conectadas en el puerto SPI. Se procedió a revisar que el sistema mostrara la tarjeta SD usando el comando **`ls /dev/mmc*`** y se formateó la tarjeta (teniendo precaución de no formatear la microSD que contiene el sistema operativo, que normalmente se encuentra como mmcblk0).

Para hacer que la memoria se monte automáticamente al iniciar el sistema operativo, se crea la carpeta donde se montará la memoria con el comando **`sudo mkdir /media`** y se ejecuta el comando **`sudo blkid`** para obtener la información

de los dispositivos, se encuentra el nombre de la memoria y de anota la ruta donde se encuentra (en este caso: **/dev/mmcblk2p1**). Luego se procede a editar el archivo de la tabla de archivos del sistema con el comando **`sudo nano /etc/fstab`** (hay que tener cuidado al editar este archivo ya que es importante para el sistema operativo) y se le agrega la siguiente línea: **/dev/mmcblk2p1 /home/pi/media/microSD2 vfat defaults,auto,user,rw,nofail,x-systemd.device-timeout=1 0 0**

Al realizar pruebas para verificar el correcto funcionamiento de la memoria extraíble, se presentó un problema al desmontar la memoria, extraerla, copiar los archivos a otra computadora, insertar la memoria e intentar montarla; al intentar montarla, después de haberla extraído e insertado nuevamente, se producía un error que impedía que se montara de manera exitosa. Después de hacer más pruebas se identificó que el problema era que la tabla de particiones se corrompía al extraer la memoria. La solución a este problema consiste en hacer una copia de la tabla de particiones y, al detectar el problema, se usa esa copia para reconstruir la tabla de particiones. Para implementar la solución se siguen los siguientes pasos: 1) Hacer una copia de la tabla de particiones ejecutando el comando **`sudo sfdisk -d /dev/mmcblk2 > mmcblk2-backup.txt`** 2) Usar el comando **`sudo parted -l`** y buscar en la respuesta la palabra “unknown” que indica una tabla de particiones corrupta 3) Si la tabla de particiones presenta el error, se procede a reconstruirla usando la copia con el comando **`sudo sfdisk -d /dev/mmcblk2 > mmcblk2-backup.txt`** 4) Montar la memoria con el comando **`sudo mount -t vfat ~/media/microSD2`**

Finalmente, se desarrolló un *conjunto de scripts para montar y desmontar la microSD al presionar un botón*. A continuación, se explica el funcionamiento de los scripts presentes en la liga anterior:

El archivo **listen-for-mount.py** se encarga de detectar un cambio de flanco en el GPIO 23 y ejecutar el siguiente paso (el script **umsd.sh)**. Para poder ejecutar este archivo al iniciar el sistema, se debe mover a una carpeta específica (**`sudo mv listen-for-mount.py /usr/local/bin/`**) y hacerlo ejecutable (**`sudo chmod +x /usr/local/bin/listen-for-mount.py`**).

El archivo **listen-for-mount.sh** se encarga de iniciar o detener el servicio para el script anterior, así como éste, se debe mover a un directorio específico (**`sudo mv listen-for-mount.sh /etc/init.d/`**), se debe hacer ejecutable (**`sudo chmod +x /usr/local/bin/listen-for-mount.sh`**) y se registra como script de inicio (**`sudo update-rc.d listen-for-mount.sh defaults`**).

El archivo **umsd.sh** se encarga de montar o desmontar la memoria microSD y mandar a llamar un último script **turn-LED.py.** Ese último script se encarga de encender o apagar un LED que se conectó con una resistencia al GPIO 25 y su otra terminal a GND.

También se aprovechó para escribir los *scripts para apagar la Raspberry cuando se presiona un botón.* El procedimiento para hacer que funcionen es el mismo que se siguió para los archivos **listen-for-mount.py** y **listen-for-mount.sh,** lo único que cambia es el puerto donde se encuentra el botón (el GPIO 18) y el comando que se ejecuta al presionar el botón (**`shutdown -h now`**).

5.7 Desarrollo del programa para la comunicación con la estación terrena

Al comenzar el desarrollo de la comunicación con la estación terrena, aún no se había adquirido el módulo XBee, pero se tenía un par de módulos RF que operan a 433 MHz, por lo que se decidió probar si podrían servir para la comunicación. Para esto, se instalaron dos bibliotecas (wiringPi y 433Utils) ejecutando los siguientes comandos en la terminal:

- Para clonar e instalar la biblioteca wiringPi del repositorio de Github: **`git clone git://git.drogon.net/wiringPi && cd wiringPi &&./build`**
- Clonando la biblioteca 433Utils desde Github: **`git clone --recursive git://github.com/ninjablocks/433Utils.git`**
- Entrando en la carpeta donde están los archivos: **`cd 433Utils/RPi_utils`**
- Compilando la biblioteca: **`make all`**

La biblioteca 433Utils hace uso de wiringPi para utilizar puertos comunes y transferir información como si estuviera usando la interfaz serial (bitbanging) y así poder enviar mensajes con los módulos RF; esta biblioteca cuenta con algunos sketches para Arduino que ejemplifican su uso con estos módulos RF. Se cargó uno de los sketches en el Arduino y se procedió a enviar comandos desde la Raspberry. No se avanzó más con las pruebas con estos módulos ya que la comunicación era muy inestable, por lo que se prefirió esperar a adquirir el módulo XBee. A continuación, se muestra el diagrama de conexión utilizado.

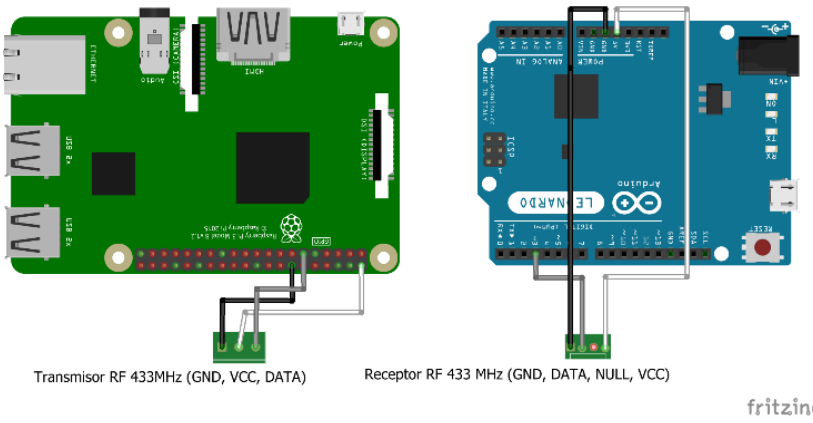


Figura 5.16 Diagrama de conexión del transmisor a la Raspberry Pi y el receptor al Arduino Leonardo

Tabla 5-11 Conexión del transmisor a la Raspberry Pi y el receptor al Arduino Leonardo

Transmisor 433 MHz	Raspberry Pi	Receptor 433 MHz	Arduino Leonardo
GND	Pin 14 - GND	GND	GND
VCC	Pin 2 - 5 V	DATA	Pin 3
DATA	Pin 11 – GPIO 17	NULL	--
--	--	VCC	5 V

Una vez que se consiguieron los módulos XBee, se hicieron unas pruebas sencillas de enviar y recibir mensajes simples como “Hola Arduino”. En estas pruebas, la estación terrena utilizaba un Arduino para controlar el módulo XBee y así enviar o recibir mensajes con ayuda del monitor serial; mientras que la Raspberry utilizaba un módulo XBee con conector USB. A continuación, se muestra el diagrama de conexión.

Tabla 5-12 Conexión del módulo XBee al Arduino Leonardo

Módulo XBee	Arduino Leonardo
GND	GND
VCC	5 V
DOUT	Pin 11 - RX
DIN	Pin 10 - TX

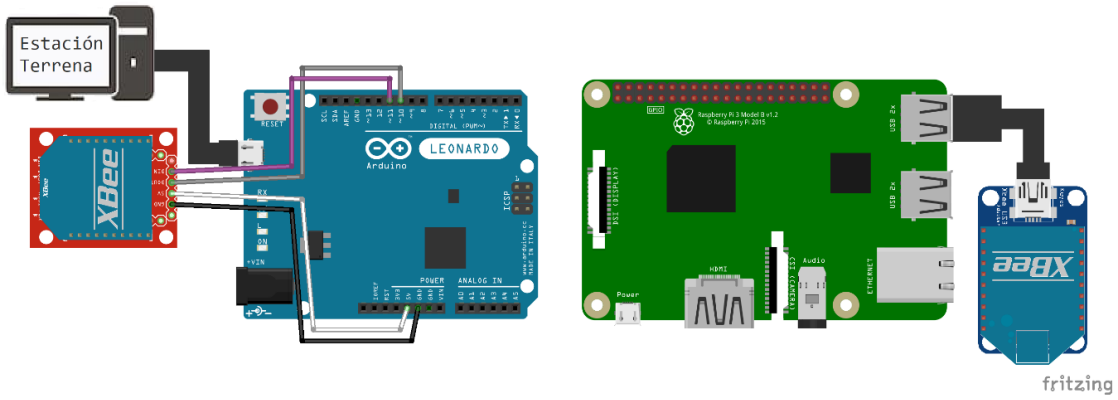


Figura 5.17 Diagrama de conexión de los módulos XBee para la comunicación con la estación terrena, primera configuración

Se observó que no era buena idea utilizar esta configuración ya que la estación terrena necesitaría un Arduino, por lo cual se decidió que la estación terrena utilizaría el módulo XBee conectado por USB; mientras que la Raspberry utilizaría su UART para enviar y recibir mensajes. Se le hicieron algunas modificaciones al *código para enviar y recibir mensajes con el módulo XBee* y se corroboró su correcto funcionamiento conectándolo como se muestra a continuación.

Tabla 5-13 Conexión del módulo XBee a la UART de la Raspberry Pi

Módulo XBee	Raspberry Pi

GND	Pin 20 - GND
VCC	Pin 1 - 3.3 V
DOUT	Pin 10 - RXD
DIN	Pin 8 - TXD

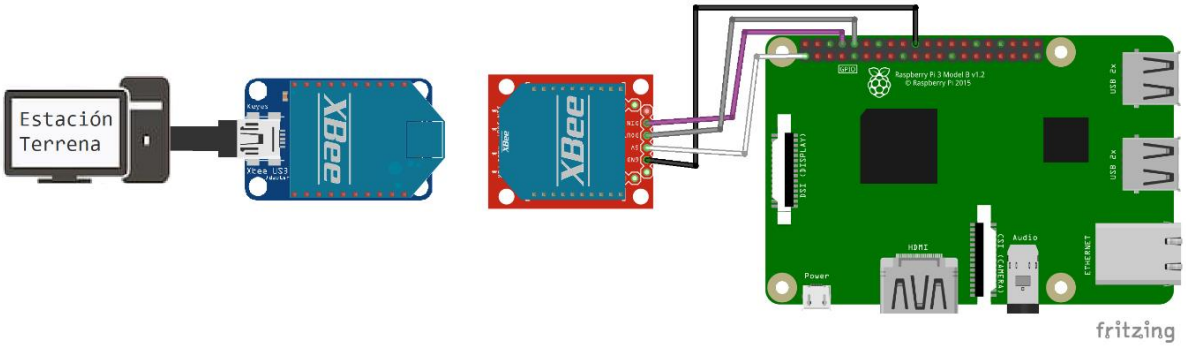


Figura 5.18 Diagrama de conexión de los módulos XBee para la comunicación con la estación terrena, configuración final

Para este punto se desarrolló una *primera versión de la estación terrena* en LabVIEW, de esta forma se pudo comenzar a probar el intercambio de mensajes entre la Raspberry y la Estación Terrena.

5.8 Integración de los componentes y diseño del algoritmo de control

Para este punto ya se cuenta con todos los componentes de la computadora de a bordo funcionando por separado, por lo que se procede a integrar los códigos en un solo programa con el que sea posible realizar pruebas y corroborar que todo funcione bien en conjunto, no obstante, se tuvieron que modificar algunos aspectos:

Cambio del puerto del GPS.

Al agregar el XBee para la comunicación con la estación terrena, se tuvo que cambiar en donde se conectaría el módulo GPS. Ya que la UART de la Raspberry Pi está siendo ocupada por el XBee, se investigó alguna forma alternativa de conectar y usar el GPS; después de analizar algunas alternativas, se decidió configurar una UART virtual. Para esto, se utilizó la biblioteca PIGPIO que se instala siguiendo los siguientes pasos y ejecutando los comandos en la terminal:

1. Remover cualquier folder de PIGPIO que pudiera haber, no es necesario si es la primera vez que se instala: **`rm pigpio.tar`** y luego **`sudo rm -rf PIGPIO`**
2. Descargar la versión más reciente de la biblioteca de la página web: **`wget abyz.me.uk/rpi/pigpio/pigpio.tar`**
3. Extraer los archivos: **`tar xf pigpio.tar`**
4. Entrar a la carpeta y preparar la instalación: **`cd PIGPIO`** y luego **`make`**

5. Instalar y esperar a que termine: **`sudo make install`**

Para verificar que la biblioteca esté funcionando, se pueden ejecutar lo siguientes comandos: **`sudo pigpiod`** para iniciar el Daemon y **`sudo ./x_pigpio`** para checar funcionalidad (esta prueba usa el GPIO 25, por lo que se debe quitar lo que esté conectado). Además, se registra el Daemon para que se ejecute al iniciar Raspbian con el comando: **`sudo systemctl enable pigpiod`**

Teniendo la biblioteca instalada y funcionando, se procedió a desarrollar un *conjunto de códigos de prueba para usar el GPS con una UART virtual* con lo que se hace lo mismo que las pruebas descritas en la sección *5.5.3* pero usando el GPIO 27 como RX y el GPIO 17 como TX. Se conectó de la siguiente manera:

Tabla 5-14 Conexión del módulo GPS a la Raspberry Pi (UART virtual)

Módulo GY-NEO6MV2	Raspberry Pi
VCC	Pin 17 - 3.3 V
RX	Pin 11 - GPIO 17
TX	Pin 13 - GPIO 27
GND	Pin 9 - GND

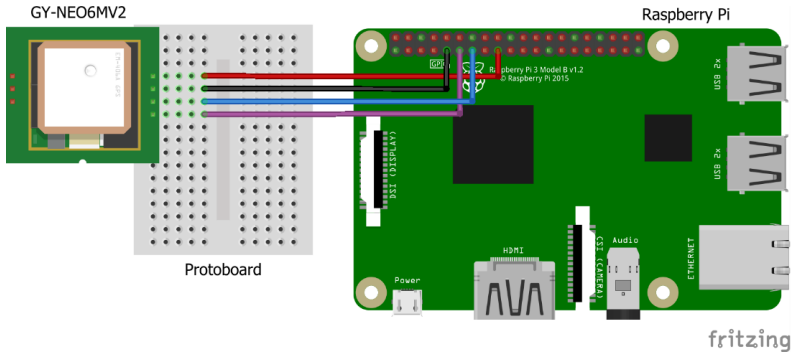


Figura 5.19 Diagrama de conexión del módulo GPS a la Raspberry Pi (usando una UART virtual)

Adición de un logger.

Se agregó el manejo de un archivo en el que se recopila información sobre la ejecución del programa principal de la OBC. En cada entrada del archivo se registra lo siguiente:

- La fecha y hora de la entrada.
- El tipo de entrada:
  - INFO: Identifica la correcta ejecución de una función. Por ahora muestra cada que se logra realizar una comunicación exitosa.
  - WARNING: Identifica la presencia de problemas de formato, tanto de los comandos de los subsistemas, como del mal uso de las funciones disponibles.

    - ERROR: Identifica la presencia de errores que detendrían la ejecución del programa principal. Sirve para ayudar al desarrollador a identificar y corregir problemas dentro del código.
- Mensaje de la entrada: Contiene el nombre de la sección del código que produce la entrada y la información que permite analizar el contexto por el que se produjo la entrada

El *código del logger* permite agregar entradas a un archivo de log que se copia a la memoria extraíble. Se modificó el script **umsd.sh** (encontrado en *B.8*) para que, al momento de presionar el botón para desmontar la memoria, copie el archivo de log a la memoria y después continúe con el desmontado de la microSD, de esta manera siempre que se extrae la memoria se cuenta con la última versión del archivo de log.

Mejora del Arduino simulando los sistemas.

Se agregaron variables para mejorar el simulador de los subsistemas y permitirle registrar el efecto de algunos comandos recibidos como: la distancia avanzada, la posición del brazo robótico y poder reportar si están encendidos o apagados los sistemas de visión, propulsión o brazo robótico.

Correcciones a la estación terrena.

Al hacer pruebas con la interfaz que se tenía, se detectó que había un error en el envío de información, el problema era que la interfaz necesitaba recibir un mensaje justo antes de poder enviar una respuesta. Después de analizar el VI se detectó el error y se determinó que sería mejor usar máquinas de estado para controlar mejor el flujo de la interfaz. En esta *mejora a la interfaz de la estación terrena* se implementaron las máquinas de estado y se agregaron los elementos de la interfaz para poder ingresar los valores relevantes para los comandos de los que se hablarán más abajo. El desarrollo y las pruebas hasta este punto, fueron realizadas con todos los componentes conectados por medio de una protoborad de la siguiente forma:

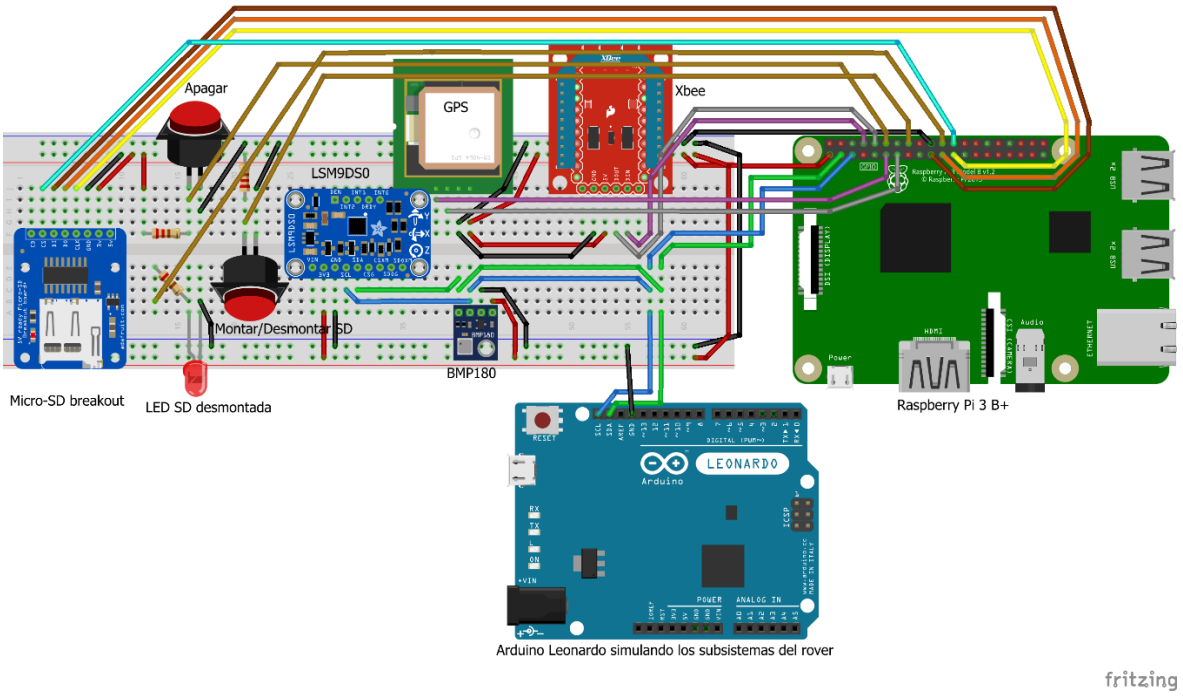


Figura 5.20 Diagrama de conexión de los elementos de la OBC (en una protoboard)

Estando en este punto, se definieron los comandos que tendría disponible la estación terrena (siguiendo los mismos principios descritos en la sección *5.2.1*). A continuación, se listan y se describen brevemente los comandos:

Estación Terrena

Caracteres de procedencia: ET

Comandos:

- **> ENT_MAN**
  Pasa el rover a modo manual. Los comandos manuales (los que comienzan con PRP y BRR) no regresan respuesta para evitar interrupciones.
- **< ENT_MAN**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> PRP_AVZ,(E/A)**
  Hace que el rover avance (E) o retroceda (A) la distancia de avance estándar, que por default es de un centímetro.
- **> PRP_GIR,(D/I)**
  Hace que el rover gire hacia la derecha (D) o izquierda (I) un número estándar de grados, por default es un grado.
- **> BRR_ART,(3/2/1/-1/-2/-3)**
  Hace que el rover gire la articulación (3, 2 o 1) del brazo robótico un número estándar de grados, el signo indica el sentido de giro.
- **> BRR_EFF,(0/1)**

Hace que el rover abra (0) o cierre (1) su efector final.

- **> ENT_SAM**
  Pasa el rover a modo semiautónomo. Todos los comandos después de este comando se consideran comandos semiautónomos.
- **< ENT_SAM**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> STP_RVR**
  Hace que el rover detenga todas las acciones que esté ejecutando. **Aún no se implementa esta función**.
- **< STP_RVR**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> CHN_PWR,(0/1/2/3/4)**
  Cambia el estado de operación del rover. **Aún no se implementa esta función.** Se proponen las siguientes instrucciones para el rover: (0) Apagar, (1) Encender, (2) Reiniciar, (3) Hibernar y (4) Reprogramar.
- **< CHN_PWR**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> MOD_PAR,(0/1/2/3/4/5)**
  Modifica algunos parámetros importantes del rover: (0) Posición objetivo, (1) Distancia estándar de avance, (2) Grados estándar de giro, (3) Velocidad estándar*, (4) Nivel de batería baja y (5) Tiempo de espera para recargar batería. *Aún no se define si será posible modificar la velocidad del rover.
- **< MOD_PAR**
  Regresa el comando recibido para indicar que se recibió correctamente.
- **> EST_RVR**
  Pide el estado de operación del rover: El estado de operación de cada subsistema, las lecturas de los sensores, la ubicación del rover, etc.
- **< EST_RVR,…**
  Regresa el estado de operación del rover.
- **> AVZ_PSO**
  Hace que el rover ejecute los pasos necesarios para llegar a la posición objetivo.
- **< AVZ_PSO**
  Regresa el comando recibido para indicar que se recibió correctamente o si hubo algún problema al ejecutar la acción lo reporta.
- **> GET_PNV**
  Hace que el rover ejecute los pasos necesarios para que el rover obtenga el plan de navegación y lo envíe a la estación terrena.
- **< GET_PNV,PNV**
  Regresa el plan de navegación (PNV) o si hubo algún problema al intentar obtenerlo.

Posteriormente se hizo el desarrollo de los algoritmos que usa la computadora de a bordo para controlar los subsistemas del rover y llevar a cabo las tareas asignadas desde la estación terrena.

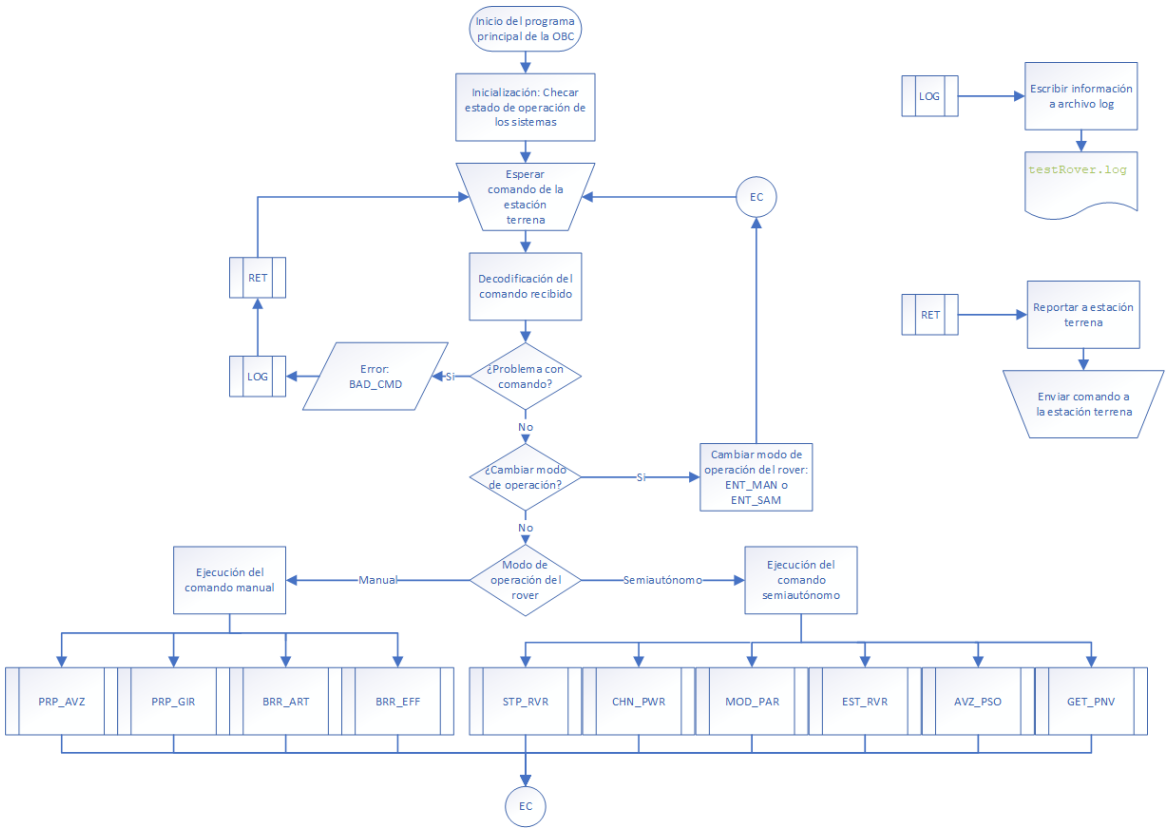


Figura 5.21 Diagrama de flujo general. En este, se espera a recibir un comando de la estación terrena y se ejecuta.

La *Figura 5.21* corresponde al algoritmo general de control, se encarga de seguir la secuencia de inicialización del rover, esperar a recibir comandos de la estación terrena, decodificar los comandos recibidos y ejecutarlos. Hay 12 comandos disponibles, 2 de ellos (ENT_MAN y ENT_SAM) se encargan de cambiar el modo de operación del rover: navegación manual o navegación semiautónoma; de los 10 comandos restantes, 2 aún no se implementan ya que aún falta definir las capacidades y limitaciones finales de los subsistemas. Teniendo el algoritmo general, se continuó con el desarrollo de los algoritmos correspondientes a los 8 comandos restantes. A continuación, se muestran los diagramas de flujo de estos.

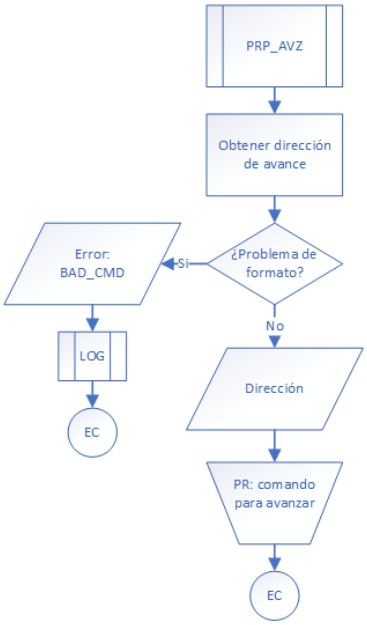


Figura 5.22 Diagrama de flujo del comando PRP_AVZ, ejecuta la orden de avanzar en la dirección especificada

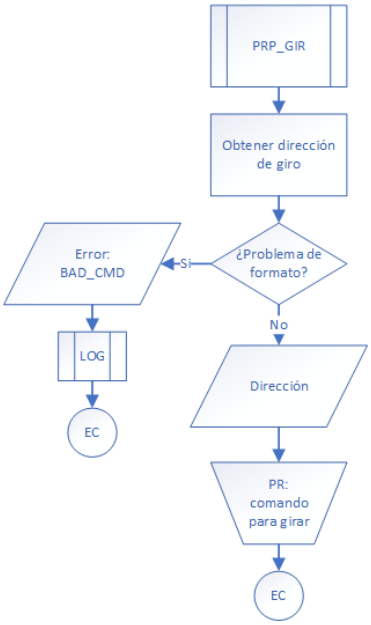


Figura 5.23 Diagrama de flujo del comando PRP_GIR, ejecuta la orden de girar en la dirección especificada

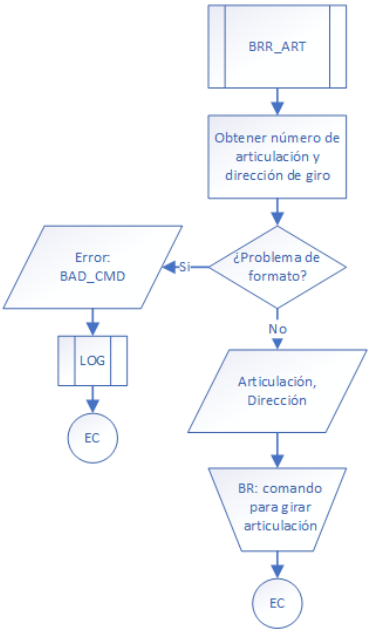


Figura 5.24 Diagrama de flujo del comando BRR_ART, ejecuta la orden de rotar una articulación en la dirección especificada

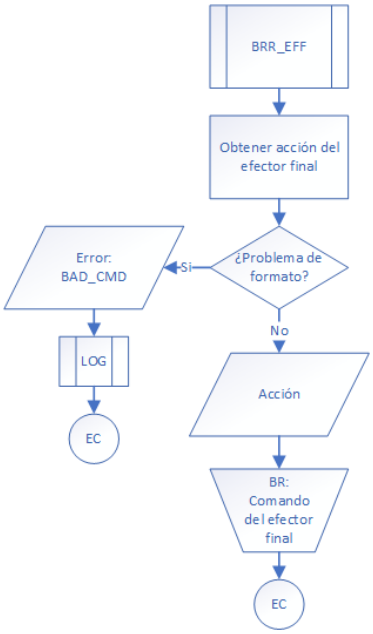


Figura 5.25 Diagrama de flujo del comando BRR_EFF, ejecuta la orden de accionar el efector final

Las cuatro figuras anteriores muestran los algoritmos de los comandos que se ejecutan solo cuando el rover está en modo manual, estos se encargan de hacer que el rover avance o retroceda, gire sobre su propio eje a la derecha o izquierda, mueva alguna articulación del brazo robótico y cierre o abra su efector final. Los cuatro comandos restantes se ejecutan solo cuando el rover está en modo de operación semiautónoma.

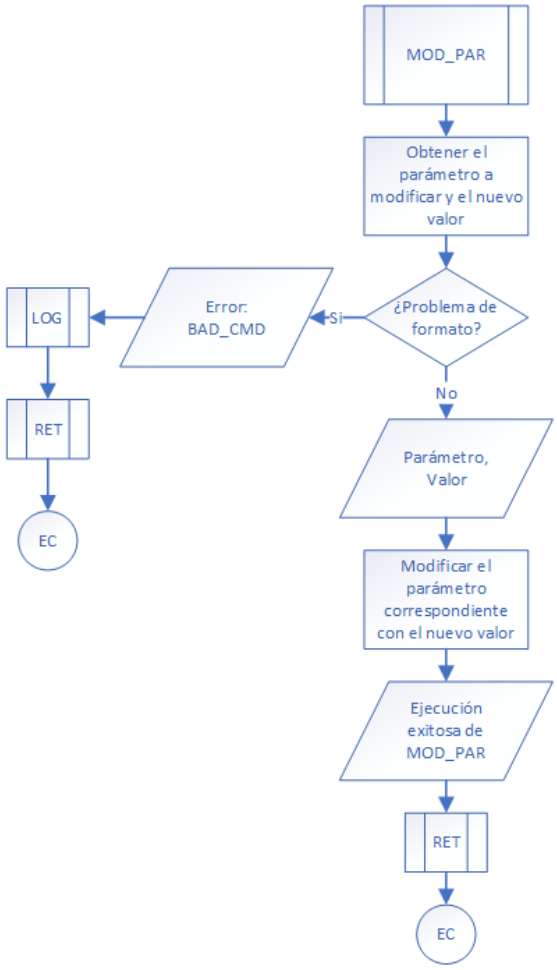


Figura 5.26 Diagrama de flujo del comando MOD_PAR, ejecuta la orden de modificar alguno de los parámetros modificables de la OBC

En la *Figura 5.26* se observa el algoritmo del comando MOD_PAR, este comando se encarga de modificar algunos parámetros del rover: La posición objetivo, la distancia estándar de avance, el número de grados estándar de giro, la velocidad estándar, el nivel de batería baja y el tiempo de espera para recargar batería.

En la *Figura 5.27* se observa el algoritmo del comando EST_RVR, este comando se encarga de recabar la información del estado de operación de los sistemas del rover y sus sensores, para después enviar esa información a la estación terrena.

En la *Figura 5.28* se observa el algoritmo del comando AVZ_PSO, este comando es el que se encarga de hacer que el rover avance a la posición objetivo de manera semiautónoma. La computadora de a bordo compara la posición actual con la posición objetivo, checa el estado de los sistemas, ejecuta las acciones necesarias para adquirir el plan de navegación (PNV) y lleva ejecuta cada paso del PNV siempre corroborando que sea seguro avanzar. Antes de que se ejecute el paso del PNV, se corrobora que haya suficiente batería, si el nivel de carga es menor a un nivel preestablecido, se apagan los sistemas y se espera durante cierto tiempo para

que se cargue la batería, terminando esa espera vuelve a checar el nivel de batería y si todo está bien, continua con la ejecución. Cada que se termina la ejecución de un paso del PNV, se comprueba si se ha llegado a la posición objetivo, si aún no se llega y no quedan pasos en el PNV, se pide uno nuevo y se sigue repitiendo hasta que se asegure de llegar a la posición objetivo. Por ahora, se piensa comprobar si se ha llegado a la posición objetivo utilizando el GPS, sin embargo, es posible que se cambie de estrategia.

En la *Figura 5.29* se observa el algoritmo del comando GET_PNV, este comando se encarga de obtener el plan de navegación y enviárselo a la estación terrena. Para obtener el PNV lo primero que se hace es checar que no se esté en la posición objetivo, si ya se está ahí, se envía un plan vacío. Por el otro lado, si aún no se encuentra el rover en la posición objetivo, se checa el nivel de la pila y se asegura de tener suficiente para obtener el PNV. Luego, checa si están encendidos los sistemas de visión y propulsión, si están apagados los enciende; después se gira el rover hasta tenerlo orientado hacia la posición objetivo, se pide al sistema de visión iniciar el cálculo del PNV y, después de esperar el tiempo necesario para calcularlo, se obtiene el PNV y se envía a la estación terrena.

Al finalizar el desarrollo de los algoritmos, se tradujeron esos algoritmos a código en lenguaje Python y se realizaron pruebas. No se adjuntan los códigos finales ni del Arduino Leonardo simulando las respuestas de los subsistemas, ni de la computadora de a bordo ya que ocuparían demasiado espacio en los Apéndices.

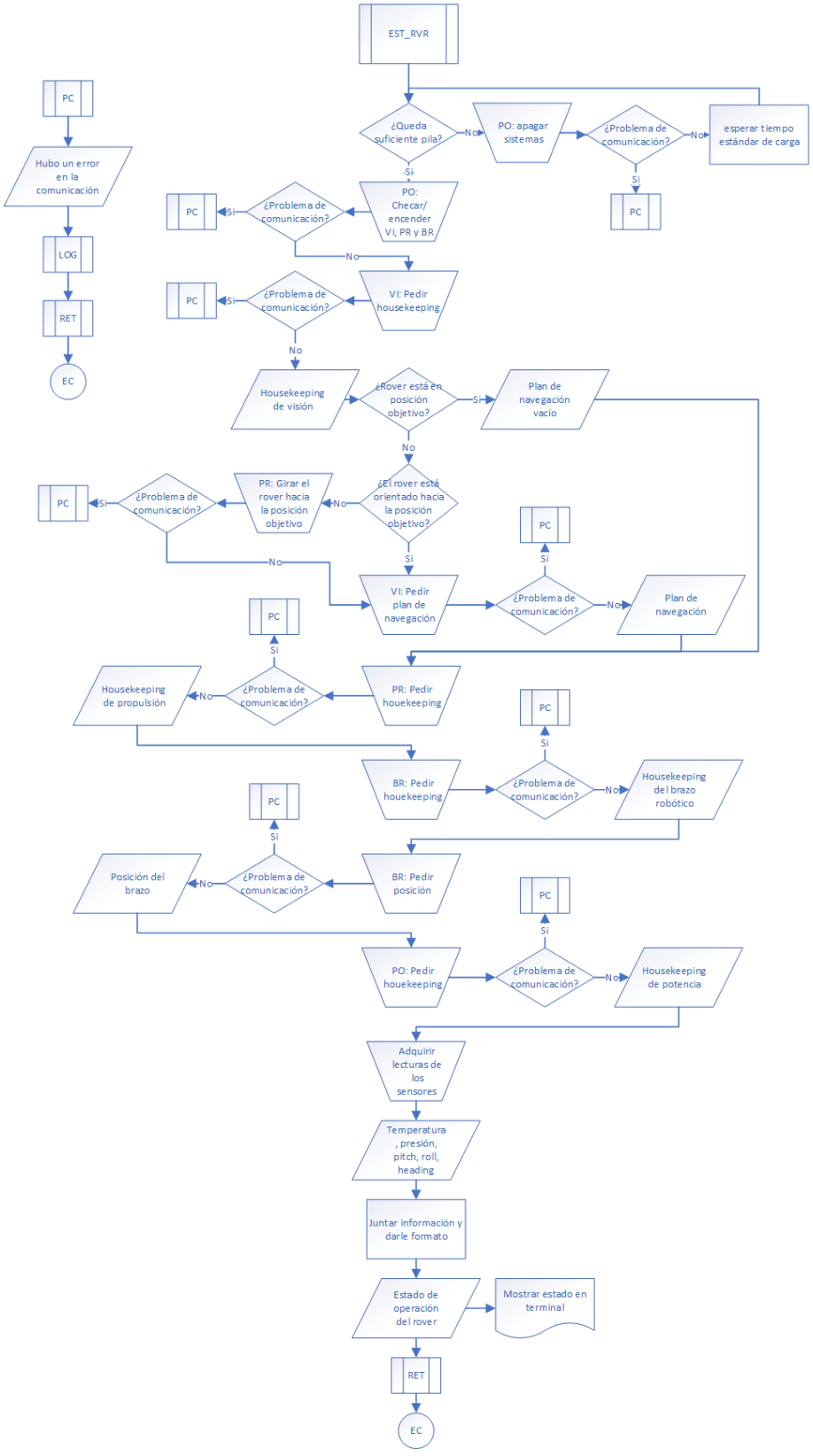


Figura 5.27 Diagrama de flujo del comando EST_RVR, ejecuta la orden de reportar a la estación terrena el estado de operación del rover

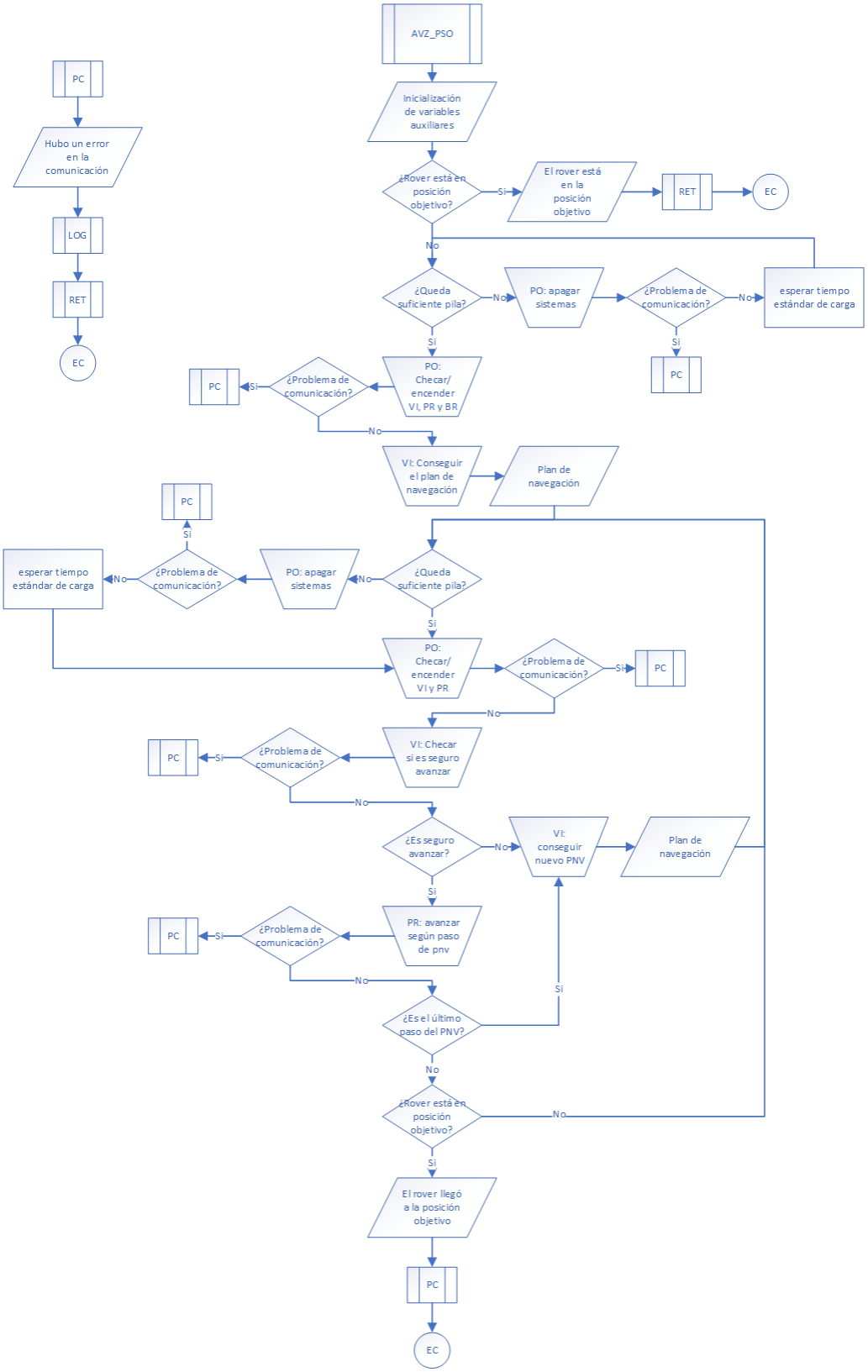


Figura 5.28 Diagrama de flujo del comando AVZ_PSO, ejecuta la orden de avanzar de manera semiautónoma hasta la posición objetivo

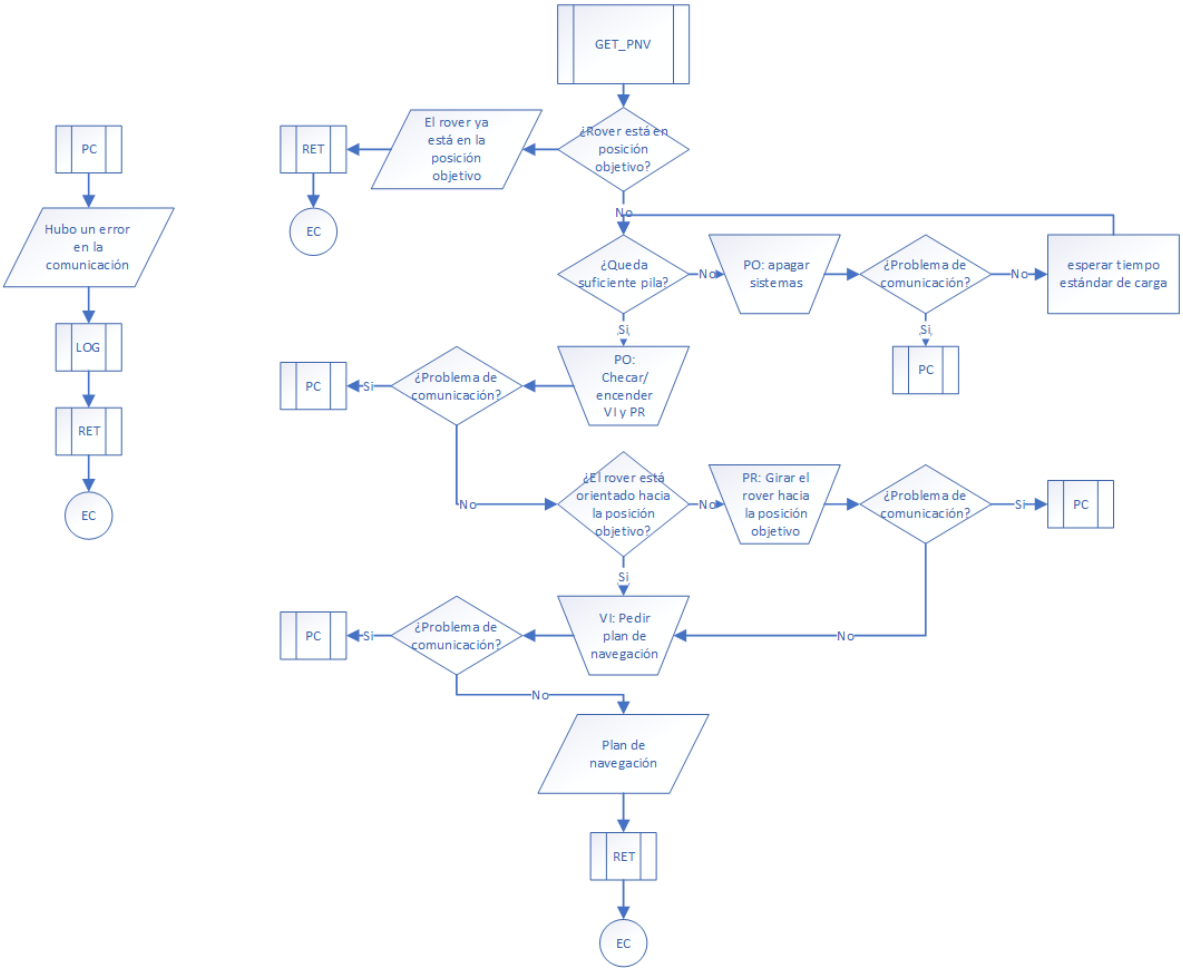


Figura 5.29 Diagrama de flujo del comando GET_PNV, ejecuta la orden de reportar a la estación terrena el plan de navegación actual del rover

5.9 Manufactura de la placa de circuito impreso para la OBC

Para manufacturar la placa de circuito impreso de la OBC, se utilizó una placa de cobre de 10 cm x 15 cm y se tomaron esas dimensiones como base para el diseño de la PCB. En la *Figura 5.30* se observa el esténcil de la PCB, en la *Figura 5.31* se observa la máscara de componentes (silkscreen en inglés) que sirve como guía al momento de colocar los componentes. Ambas figuras fueron obtenidas mediante el diseño del esquemático de la OBC con el uso del software Eagle.

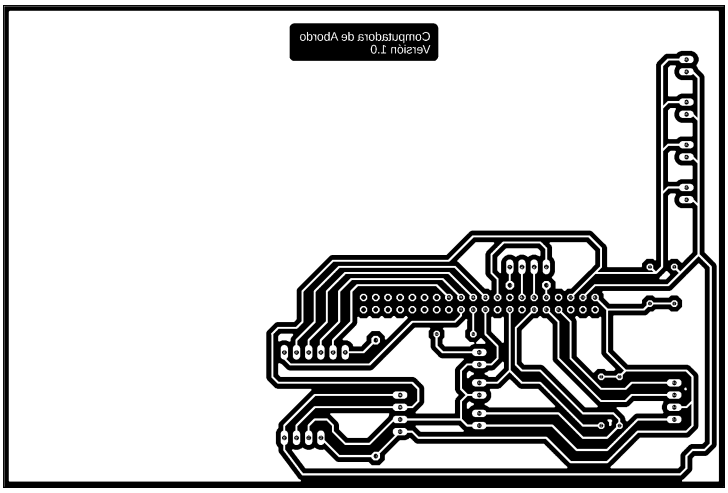


Figura 5.30 Negativo de la plantilla correspondiente a la PCB de la OBC

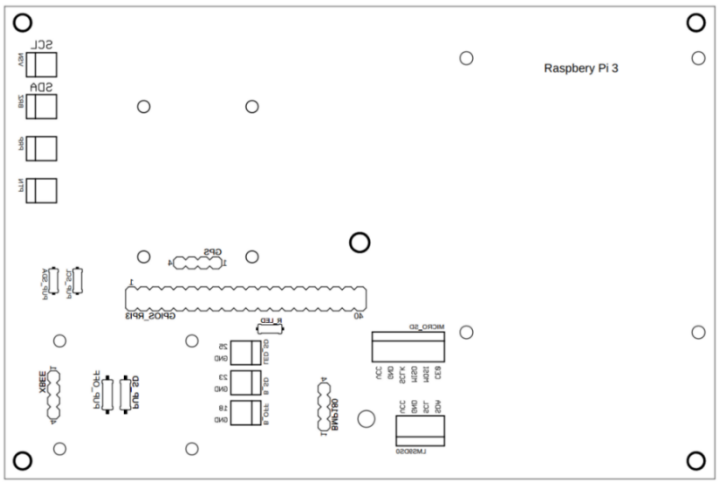


Figura 5.31 Máscara de componentes de la OBC, sirve como guía al momento de colocar los componentes.

La manufactura del circuito impreso se realizó en el laboratorio HiL & SiL en la Unidad de Alta Tecnología de la UNAM. En este proceso se usaron los siguientes equipos:

- Laminadora de film seco RLM419 P
- Insoladora doble cara con vacío HELLAS
- Dremel
- Cautín

Se utilizaron los siguientes materiales:

- Materiales comunes: Lija de agua (grano fino), guantes, estopa, thinner, agua, recipientes, palillos chinos, cepillo de dientes
- Hoja de acetato
- Película seca fotosensible (Ordyl Alpha 940)

- Bicarbonato de sodio
- Cloruro férrico
- Estaño líquido
- Placa de cobre de 10 cm x 15 cm
- Soldadura
- Componentes electrónicos:
    - 1 header macho doble de 20 pines
    - 4 headers hembra de 4 pines
    - 1 header hembra de 6 pines
    - 1 juego de conector JST serie PH de 4 pines
    - 1 juego de conector JST serie PH de 6 pines
    - 7 juegos de conectores JST serie PH a 90° de 2 pines
    - 1 led rojo
    - 1 resistencia de 330 Ω
    - 4 resistencias de 4.7 kΩ
    - 2 botones interruptores de diferente color
    - 1.5 m de cable plano de 20 vías

Teniendo listo el esquemático de la OBC, los equipos y el material; se realizaron los siguientes pasos:

1. Se imprimió en la hoja de acetato el esténcil de la PCB (el patrón de la PCB con color invertido y con el modo espejo aplicado), tal y como lo muestra la *Figura 5.30*. Se hace de esta forma ya que el proceso de manufactura con el uso de la película fotosensible así lo requiere.
2. Con los guantes puestos, se procedió a lijar la placa de cobre (debajo de un chorro de agua) cuidando que quedase lo más lisa posible. Una vez se aseguró que estuviera limpia, se secó la placa y se llevó al cuarto obscuro.
3. En el cuarto obscuro, se preparó la laminadora: Se estableció la temperatura del rodillo a 105°C, con una presión de 1 y una velocidad de 0.5 m/min.
4. Mientras se esperaba a que se calentara la laminadora, se cortó una sección de película fotosensible de tal forma que ocupara un área un poco mayor al área ocupada por el esquemático. Se retiró la capa protectora de la cara que haría contacto con la placa de cobre y se colocó la película fotosensible sobre la placa de cobre de tal forma que cubriese la superficie deseada.
5. Con la laminadora lista, se introdujo la placa de cobre previamente preparada y se usó el rodillo manual para asegurar que el rodillo caliente tuviera buen contacto. Se repitió este paso para eliminar la mayor cantidad posible de burbujas de aire. En la *Figura 5.32* se observa este paso.
    a. El completar satisfactoriamente este paso llevó varios intentos ya que no se lograron eliminar todas burbujas de aire entre la placa de cobre y la película fotosensible, el mejor resultado se puede observar en la *Figura 5.36*.

6. Posteriormente, se alineó el esténcil con la placa laminada y se usó cinta adhesiva para fijarlo. Se colocó la placa en la insoladora, se encendió el vacío para evitar que se moviera, se encendió la iluminación UV y se programó el tiempo de exposición.
    a. Debido a que no se conocía el tiempo de exposición óptimo para que curara perfectamente la película fotosensible, se repitieron cuatro veces los pasos 5 y 6 con 3, 2.5, 2 y 1.5 minutos de exposición, con el último produciendo los mejores resultados.
7. Se retiró la segunda capa protectora de la película fotosensible y se procedió a revelar la placa sumergiéndola en una solución líquida de: una cucharadita de bicarbonato de sodio por cada vaso de agua, este proceso se muestra en la *Figura 5.34*.
    a. Fue necesario usar un cepillo de dientes para ayudar a retirar la película, lo cual provocó daños a algunas vías, se puede observar esto en la *Figura 5.35*. Al revisar la hoja de datos de la película fotosensible se descubrió el problema, para este paso se debió usar carbonato de sodio o carbonato de potasio.
8. Teniendo la placa revelada, se sumergió en un recipiente con cloruro férrico para atacar la capa de cobre expuesta. Con ayuda de unos palillos chinos se continuó moviendo la placa hasta que se disolvió la capa de cobre expuesta. Una vez terminado esto, se retiró la placa y se limpió.
9. Se retiró la película que recubría el cobre usando una estopa sumergida en thinner, se frotó vigorosamente la placa hasta remover completamente la película protectora y se le dio un baño en estaño líquido para retrasar la oxidación del cobre.
    a. No se recomienda usar thinner ya que no es muy efectivo para retirar la película y se pueden dañar las vías (como se observa en la *Figura 5.37*), para evitar esto, se debió sumergir la placa en una solución de hidróxido de sodio.
10. Se procedió a hacer las perforaciones necesarias con el Dremel, se colocaron los componentes y se soldaron. La *Figura 5.38* y la *Figura 5.39* muestran la PCB con los componentes soldados.
11. Se unieron los conectores hembra JST a sus respectivos componentes: El conector de 4 pines se unió con una tira de 4 vías de cable plano al sensor LSM9DS0 y el otro conector, el de 6 pines, se unió al adaptador de tarjetas microSD con una tira de 6 vías de cable plano.
12. Finalmente, se colocaron los sensores y los módulos, para completar la PCB de la computadora de a bordo, el resultado de la manufactura del circuito impreso y el posicionamiento de los componentes se puede apreciar en la *Figura 5.40*.

Figura 5.32 Placa de cobre con película fotosensible en la laminadora

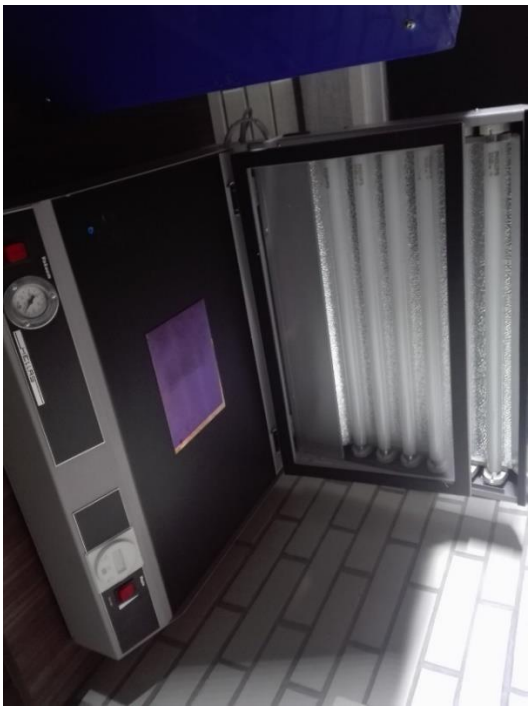


Figura 5.33 Placa laminada en la insoladora

Figura 5.34 Proceso de revelado de la placa

Figura 5.35 Vías afectadas al revelar la placa

Figura 5.36 Placa laminada seleccionada, aún hay burbujas de aire pero no afectan tantas vías

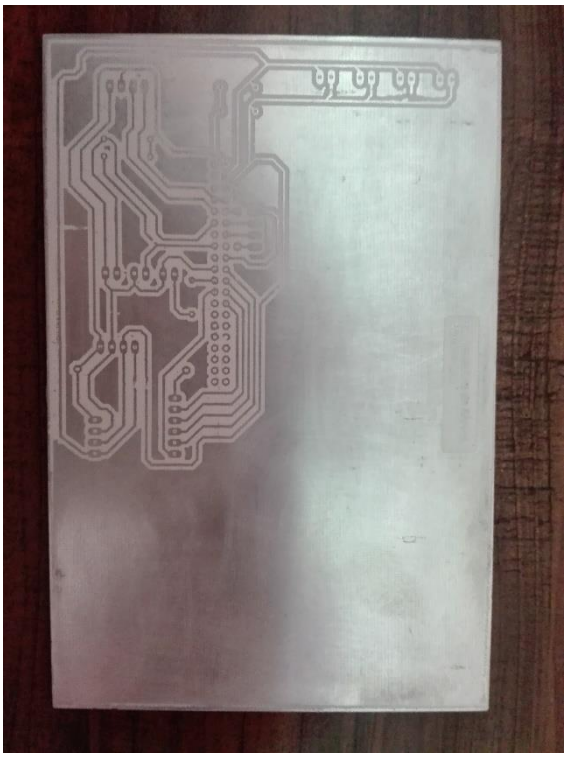


Figura 5.37 PCB después del ataque químico y recien estañada

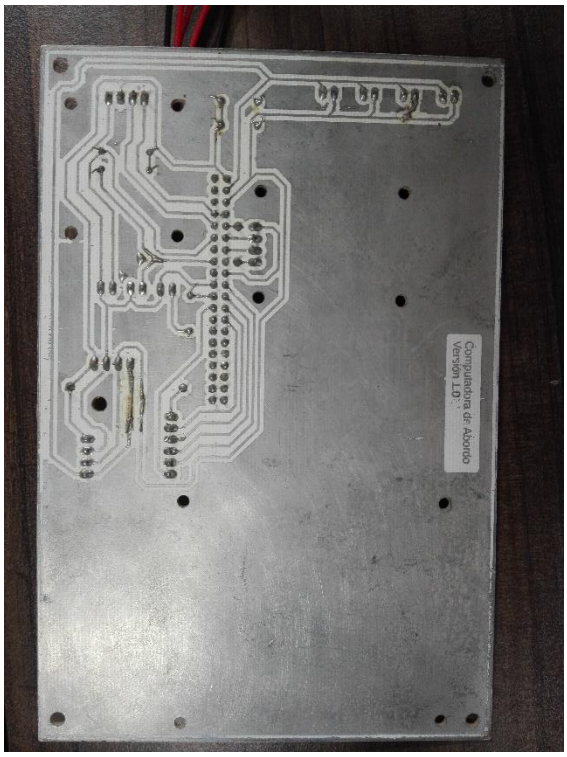


Figura 5.38 Circuito impreso terminado, vista de abajo

Figura 5.39 Circuito impreso terminado, vista de arriba

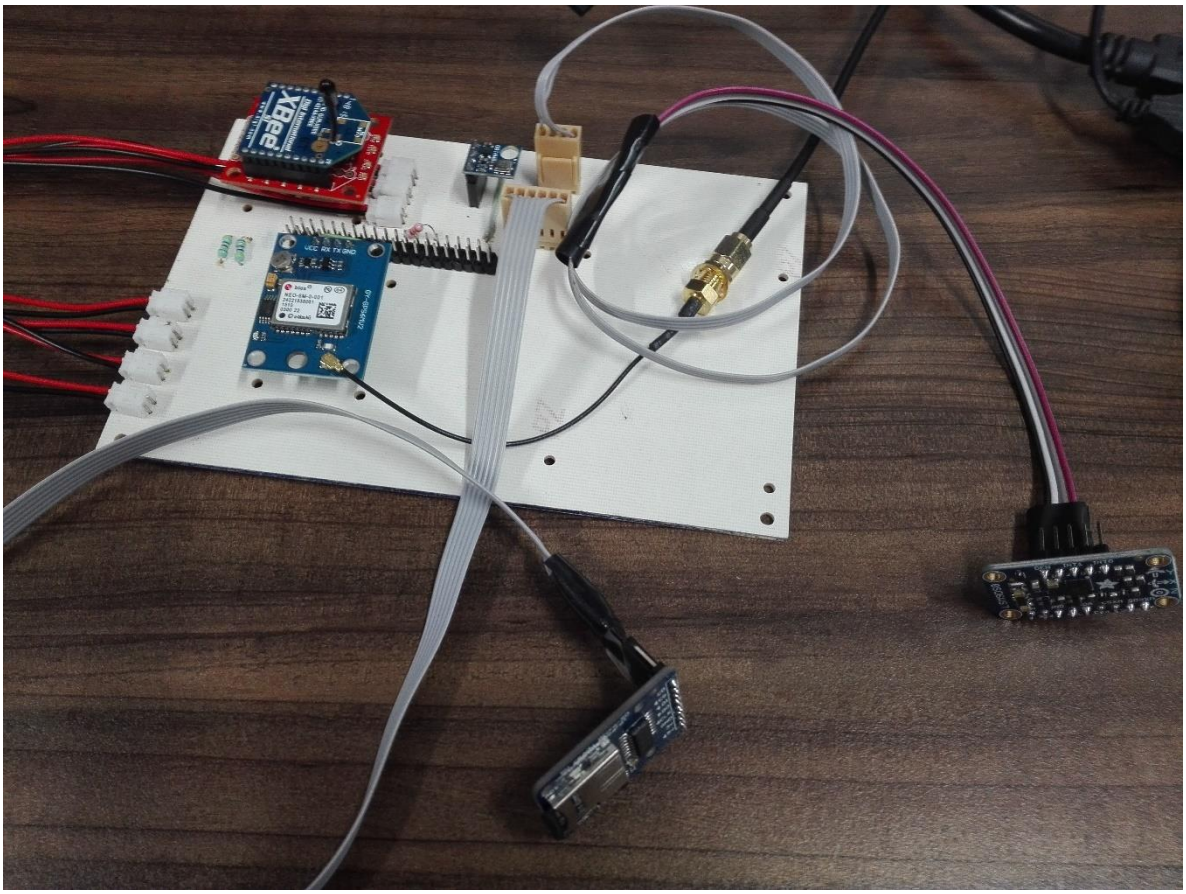


Figura 5.40 PCB de la computadora de a bordo con los componentes conectados

6 Pruebas

En esta sección se muestran, de manera condensada, las pruebas descritas en la sección anterior y las pruebas finales que se hicieron con la computadora de a bordo.

6.1 Pruebas de comunicación con los subsistemas

En la sección *5.4* se describe el proceso de desarrollo de los programas utilizados en estas pruebas y la instalación de las bibliotecas necesarias. Para realizar las pruebas, se utilizó el siguiente material:

- 3 cables jumper.
- 1 Arduino Leonardo con su respectiva fuente de alimentación.
- 1 Raspberry Pi 2 B+ con su respectiva fuente de alimentación.

Prueba 1. Comunicación I2C básica.

El objetivo de esta prueba era realizar una transmisión básica de información mediante comunicación I2C para familiarizarse con los aspectos básicos del uso de la interfaz I2C de la Raspberry Pi.

Para realizarla, se siguieron los siguientes pasos:

1. Se instalaron las bibliotecas necesarias siguiendo los pasos de la sección *5.4*.
2. Se utilizaron los cables jumper para conectar la Raspberry Pi y el Arduino Leonardo de acuerdo a la *Tabla 5-4*. La *Figura 6.1* muestra la conexión para estas pruebas.
3. Se cargó el sketch de la sección *B.1* en el Arduino.
4. Se conectaron los dispositivos a su respectiva fuente de alimentación.
5. Se ejecutó el programa de la sección *B.1* (el escrito en lenguaje Python) en la Raspberry.

Resultado 1.

Tal y como se esperaba, la comunicación entre la Raspberry Pi y el Arduino fue un éxito, en la *Figura 6.2* se observa el intercambio de mensajes entre estos dispositivos.

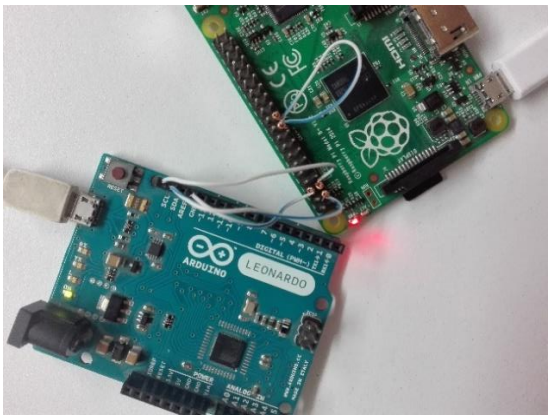


Figura 6.1 Conexión de las interfaces I2C del Arduino Leonardo y la Raspberry Pi

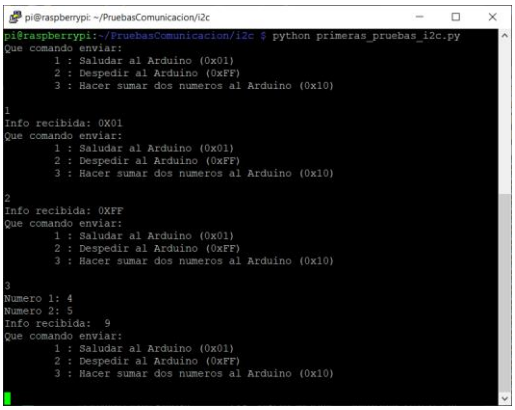


Figura 6.2 Intercambio de mensajes en pruebas básicas de comunicación I2C

Prueba 2. Comunicación entre OBC y subsistemas.

El objetivo de la segunda prueba fue desarrollar y comprobar la funcionalidad del programa que describe los subsistemas del rover como objetos y que permite interactuar con estos a través de los comandos de control y la extracción de información de sus respuestas. Ya se han mostrado algunos fragmentos relevantes de los programas usados en estas pruebas, no se anexa el código completo debido a su dimensión.

Para realizar las pruebas, se siguieron los siguientes pasos:

1. Se utilizaron los cables jumper para conectar la Raspberry Pi y el Arduino Leonardo de acuerdo a la *Tabla 5-4*.
2. Se cargó el sketch, que se encarga de simular las respuestas de los subsistemas, al Arduino.
3. Se conectó el Arduino y la Raspberry a su respectiva fuente de alimentación.
4. Se ejecutó el programa descrito en la sección *5.4* en la terminal de la Raspberry.

Resultado 2.

Al final de las pruebas se confirmó la correcta función del programa que utiliza clases para describir las propiedades y las funciones de los subsistemas del rover. La utilización de clases provee una manera sencilla de programar las interacciones entre la computadora de a bordo y los subsistemas. En la *Figura 6.3* se puede observar una versión contraída del código y un ejemplo de cuáles serían los pasos a seguir para obtener el plan de navegación del sistema de visión.

Para hacer uso de este código y continuar con su desarrollo, se necesitan tener en cuenta cuatro aspectos importantes: i) Se debe seguir el estilo PEP 484 para agregar documentación al código. ii) El valor de la variable **`ARDUINO_SIMULATION`** sirve para definir si las pruebas se harán: con los sistemas reales (False) o con el

Arduino simulando a los sistemas (True). iii) La respuesta de un subsistema debe ser enviado un byte a la vez, con el primer byte siendo el número de bytes de la respuesta, sin contar el primer byte. iv) Los subsistemas deben poder concatenar el comando de la OBC (que lo envía en bloques de 32 bytes), revisar que cumpla con el formato correcto y responder de acuerdo al comando recibido.

Figura 6.3 Vista colapsada del código desarrollado para la comunicación entre la OBC y los subsistemas

6.2 Pruebas de adquisición de temperatura

En la sección *5.5.1* se describieron las dos pruebas y los pasos para instalar las bibliotecas usadas. Para realizar estas pruebas, se utilizó lo siguiente:

- 4 cables jumper.
- 1 sensor DHT11.
- 1 sensor BMP180.
- 1 Raspberry Pi 2 B+ con su respectiva fuente de alimentación.
- 1 protoboard y adaptador de GPIOs de Raspberry Pi a protoboard.

Prueba 1. Sensor DHT11.

El objetivo de esta prueba fue adquirir las lecturas de temperatura y humedad con el sensor digital DHT11, usando las bibliotecas disponibles.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se instalaron las bibliotecas necesarias siguiendo los pasos mencionados en la sección *5.5.1*.

2. Se conectó el sensor DHT11 a la Raspberry Pi siguiendo lo mostrado en la *Tabla 5-5*. De los cables que se ven en la *Figura 6.4*: El blanco es GND, el naranja es VCC y el morado es DATA.
3. Se conectó la Raspberry a su fuente de alimentación.
4. Se ejecutó el programa mostrado en la sección *B.2*, correspondiente al sensor DHT11, en la Raspberry.

Resultado 1.

En la *Figura 6.6* se observa que la obtención de lecturas de temperatura y humedad con el sensor DHT11 fue un éxito. A pesar de ello, no se piensa utilizar este sensor con la computadora de a bordo, ya que necesita un bus especial para obtener las lecturas de este sensor.

Figura 6.4 Conexión del sensor DHT11

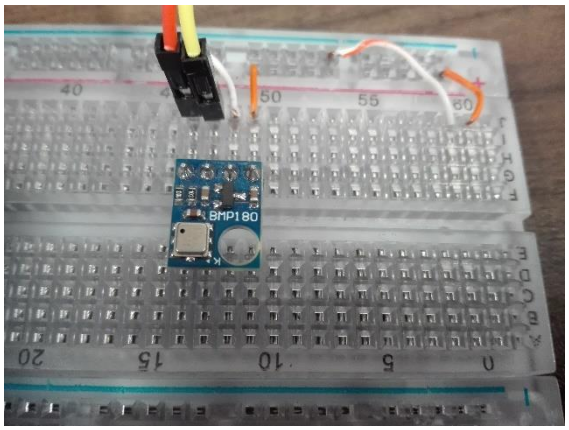


Figura 6.5 Conexión del sensor BMP180

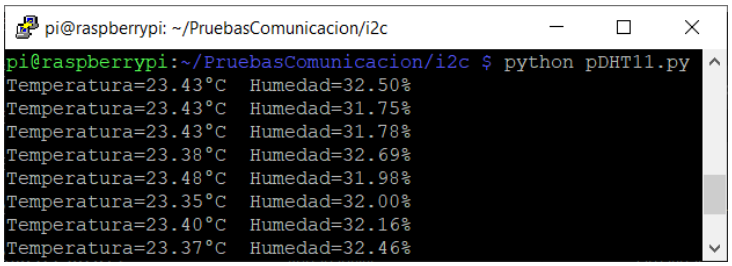


Figura 6.6 Ejecución de la prueba con el sensor DHT11

Prueba 2. Sensor BMP180.

El objetivo de la segunda prueba fue aprender a usar la biblioteca para la adquisición de las lecturas de temperatura y presión atmosférica del sensor BMP180. Este es el sensor que utiliza la OBC para medir su temperatura de operación en el rover.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se instalaron las bibliotecas descritas en la sección *5.5.1*.

2. Se conectó el BMP180 a la Raspberry Pi de acuerdo a la *Tabla 5-6*. De los cables que se ven en la *Figura 6.5*: El blanco es GND, el naranja es VCC, el jumper amarillo es SCL y el jumper naranja es SDA.
3. Se conectó la Raspberry a su fuente de alimentación.
4. Se ejecutó el programa de la sección *B.2*, correspondiente al sensor BMP180, en la Raspberry.

Resultado 2.

En la respuesta mostrada en la *Figura 6.7* se puede apreciar que el sensor tiene un problema al medir la presión atmosférica, pero la lectura de la temperatura era correcta. Ya que no se planea usar la medición de la presión en la OBC, se decidió continuar usando este sensor.

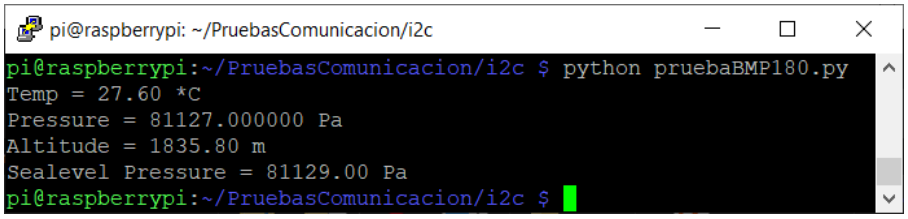


Figura 6.7 Error en la lectura de la presión con el sensor BMP180

6.3 Pruebas de adquisición de lecturas del acelerómetro, giroscopio y magnetómetro

En la sección *5.5.2* se describieron las dos pruebas y los pasos seguidos para obtener el código de las bibliotecas de los sensores. Para realizar estas pruebas, se utilizó lo siguiente:

- 4 cables jumper.
- 1 sensor MPU6050.
- 1 sensor LSM9DS0.
- 1 Raspberry Pi 3 con su respectiva fuente de alimentación.
- 1 protoboard y adaptador de GPIOs de Raspberry Pi a protoboard.
- 1 celular con una aplicación de brújula.

Prueba 1. Sensor MPU6050.

El objetivo de esta prueba fue adquirir las lecturas de aceleración y velocidad angular con el sensor MPU6050, haciendo uso del código de la biblioteca del sensor.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se obtuvo el código para utilizar el sensor.

2. Se conectó el sensor MPU6050 a la Raspberry Pi de acuerdo a la *Tabla 5-7*. De los cables que se ven en la *Figura 6.8*: El blanco es GND, el naranja es VCC, el jumper amarillo es SCL y el jumper naranja es SDA.
3. Se conectó la Raspberry a su fuente de alimentación.
4. Se ejecutó el programa de la sección *B.3* en la terminal de la Raspberry.

Resultado 1.

Fue exitosa la adquisición de las mediciones de la aceleración y la velocidad angular a las que fue expuesto el sensor MPU6050, no obstante, se planea utilizar la información de un magnetómetro para obtener la orientación del rover, por esto, no se usa este sensor en la computadora de a bordo.

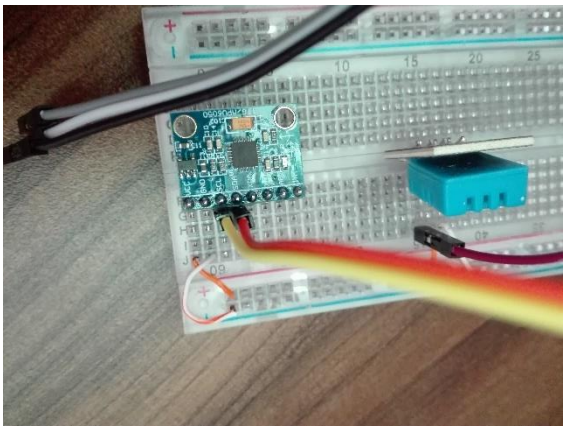


Figura 6.8 Conexión del sensor MPU6050

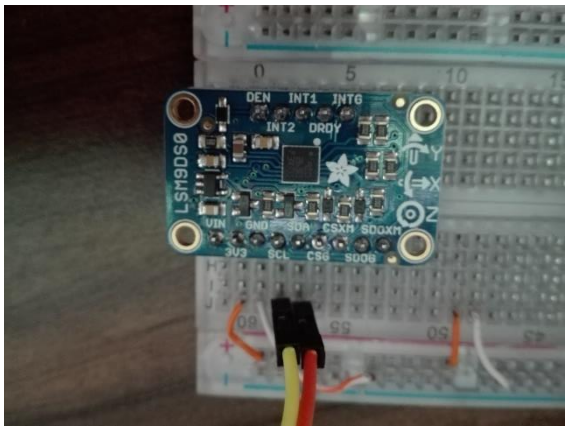


Figura 6.9 Conexión del sensor LSM9DS0

Prueba 2. Sensor LSM9DS0.

El objetivo esta segunda prueba fue familiarizarse con la biblioteca para adquirir las mediciones del sensor LSM9DS0. Este es el sensor que usa la OBC para calcular la orientación del rover utilizando las mediciones de la intensidad del campo magnético y la aceleración. La orientación se utiliza para hacer que el rover esté dirigido hacia la posición objetivo al momento de calcular el plan de navegación.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se obtuvo el código para utilizar el sensor.
2. Se conectó el sensor LSM9DS0 a la Raspberry Pi de acuerdo a la *Tabla 5-8*. De los cables que se ven en la *Figura 6.9*: El blanco es GND, el naranja es VCC, el jumper amarillo es SCL y el jumper naranja es SDA.
3. Se conectó la Raspberry a su fuente de alimentación.
4. Se ejecutó el programa de la sección *B.4* en la Raspberry.
5. Se tomaron cuatro medidas sujetando el sensor paralelo al suelo y orientado de la siguiente manera (la orientación fue aproximada, para revisarla, solo se usó una aplicación de brújula en un celular):
    a. Con el sensor orientado hacia el norte.

b. Con el sensor orientado hacia el oeste.
c. Con el sensor orientado hacia el sur.
d. Con el sensor orientado hacia el este.

Resultado 2.

En la *Figura 6.10*, se puede ver que el cálculo de la orientación (Heading) parece coincidir con las orientaciones aproximadas al momento de hacer las mediciones. Se decidió usar este sensor y el método para calcular la orientación, en la computadora de a bordo, tomando en cuenta que en un futuro sería necesario calibrar el sensor.

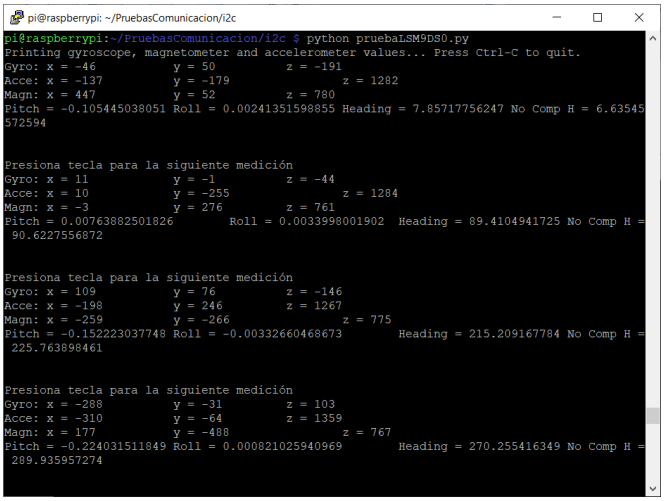


Figura 6.10 Lecturas del sensor LSM9DS0

6.4 Pruebas de adquisición de posición con el módulo GPS

En las secciones *5.5.3* y *5.8* se describe el proceso para preparar las pruebas con el módulo GPS, sin embargo, ya que el funcionamiento y la conexión en ambas pruebas es similar, solo se muestra el proceso de la implementación con la UART virtual. Para llevar a cabo esta prueba se utilizó lo siguiente:

- 4 jumpers macho-hembra.
- 1 módulo GY-NEO6MV2.
- 1 extensión de antena GPS de 3.3 V y adaptador de conector MMCX a SMA.
- 1 Raspberry Pi 3 con su respectiva fuente de alimentación.
- 1 protoboard y adaptador de GPIOs de Raspberry Pi a protoboard.

El objetivo de esta prueba fue aprender a adquirir y seleccionar las lecturas de la posición del módulo GPS, utilizando una UART virtual. Por el momento, la OBC solo utiliza la posición para reportarla a la estación terrena. Se planea que en un futuro sea utilizada para auxiliar a la navegación en modo semiautónomo, aunque aún no se define exactamente como se hará.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se habilitó la UART virtual, siguiendo los pasos descritos en la sección *5.8*.
2. Se conectó el módulo GPS a la Raspberry Pi de acuerdo a la *Tabla 5-14*. De los cables que se ven en la *Figura 6.11*: El jumper negro es GND, el jumper morado es VCC, el jumper blanco es TX, el jumper gris es RX y el cable negro que se ve en el otro extremo del módulo, es el adaptador de MMCX a SMA, se conectó a la extensión de antena.
3. Se conectó la Raspberry a su fuente de alimentación.
4. Se ejecutó el primer programa de la sección *B.11*, en la Raspberry.
    a. Se observó que había información que no le sería útil a la OBC. Se desarrolló un programa para desactivar la información innecesaria.
5. Se ejecutó el segundo programa de la sección *B.11*, en la Raspberry.

Resultado.

Al ejecutar el programa del punto 4, se observó que el GPS regresaba información que no se planea utilizar con la OBC (*Figura 6.12*), por el momento solo se considera usar los mensajes GPGLL.

Después de ejecutar el programa del punto 5, el módulo GPS dejó de enviar todos los mensajes y solo continuó enviando los mensajes GPGLL (como se muestra en la *Figura 6.13*). Normalmente sería suficiente con configurar el módulo GPS una sola vez, pero al parecer la batería del módulo GPS no sirve, por lo que se agregó el código del segundo programa a la secuencia de inicialización del programa principal de la OBC.

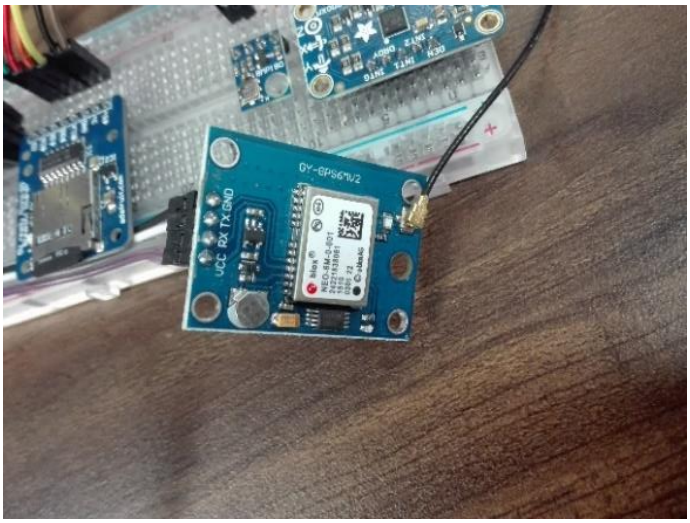


Figura 6.11 Conexión del módulo GPS

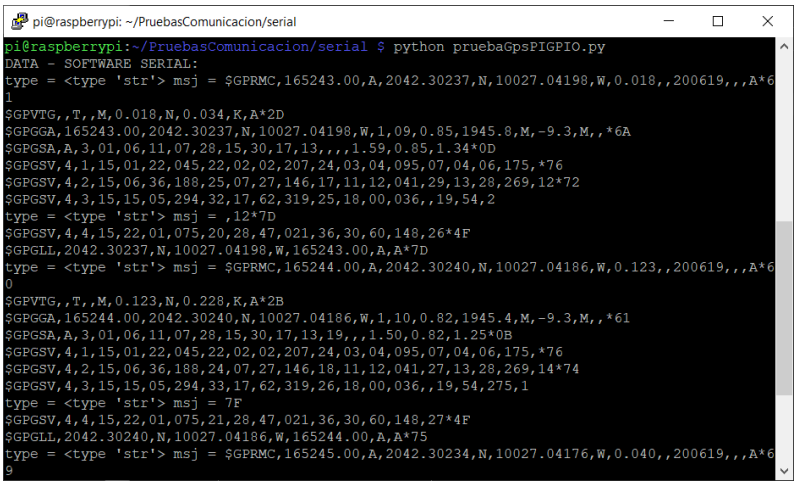


Figura 6.12 Ejecución del código para obtener la posición del GPS

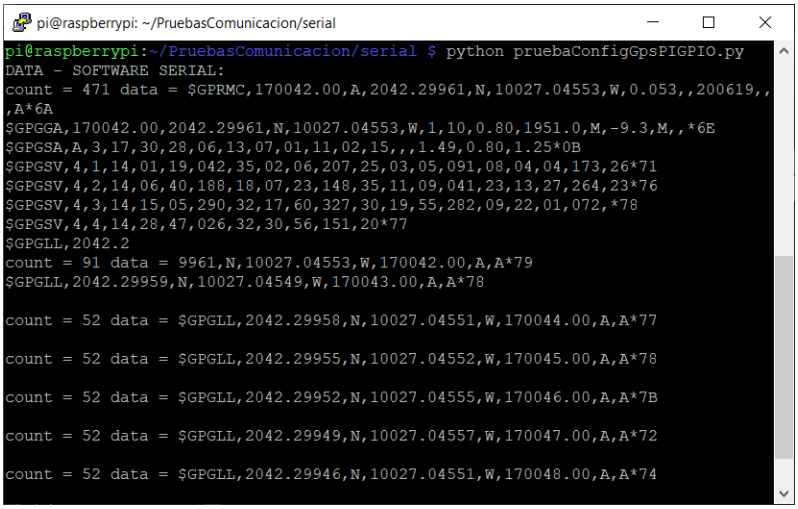


Figura 6.13 Ejecución del código que desactiva el reporte de la información inecesaria del GPS

6.5 Pruebas con la tarjeta microSD

En la sección *5.6* se describe el proceso para implementar el almacenamiento extraíble y las pruebas que se hicieron. Para llevar a cabo estas pruebas, se utilizó lo siguiente:

- 6 jumpers macho-macho.
- 1 resistencia de 220 Ω.
- 1 LED rojo.
- 2 botones.
- 1 adaptador para tarjetas microSD.
- 1 tarjeta microSD de 16 GB.
- 1 Raspberry Pi 3 con su respectiva fuente de alimentación.
- 1 protoboard y adaptador de GPIOs de Raspberry Pi a protoboard.

El objetivo de esta prueba fue habilitar un segundo puerto para la segunda memoria microSD, esto permitió tener fácil acceso al archivo donde se encuentra la información relacionada a la ejecución del programa principal de la OBC, sin necesidad de tener disponible una conexión SSH.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se siguieron los pasos descritos en la sección *5.6* para poder usar la memoria con la interfaz SPI.
2. Se conectó el adaptador a la Raspberry Pi de acuerdo a la *Tabla 5-10*. De los cables que se ven en la *Figura 6.14*: El jumper negro es GND, el jumper café es VCC, el jumper amarillo es CLK, el jumper naranja es DO, el jumper rojo es DI y el jumper verde es CS.
3. Se conectó la Raspberry a su fuente de alimentación.
4. Se hicieron pruebas de montado/desmontado de la memoria:
    a. Desmontar y montar la memoria sin extraerla.
    b. Desmontar la memoria, extraerla, volver a insertarla y montarla.
5. Se desarrollaron scripts para:
    a. Montar y desmontar la memoria al presionar un botón, prendiendo un LED mientras esté desmontada
    b. Apagar la Raspberry Pi al presionar un botón.
6. Se hicieron pruebas para verificar el funcionamiento de los botones y del LED que muestra el estado de la memoria.

Resultado.

En la prueba del punto 4.b se presentó el problema en el que se corrompía la tabla de particiones. Se solucionó obteniendo una copia funcional de la tabla de particiones y, al detectar el problema, usando dicha copia para reconstruir la tabla de particiones corrupta. El único inconveniente de esta solución es que el proceso de reconstruir la tabla de particiones demora unos segundos, no obstante, es fácil identificar que ese proceso terminó ya que se cuenta con un LED que indica el estado de la memoria (mientras está desmontada y no se puede usar se enciende, cuando está montada y lista para usarse se apaga). Los desarrollos realizados durante estas pruebas dan acceso a una manera sencilla de extraer el archivo que contiene la información de la ejecución del programa principal de la OBC. Además, se aprovechó la practica en el desarrollo de scripts para implementar un botón que se usa para apagar la Raspberry Pi de manera segura, sin necesidad de tener una sesión SSH abierta.

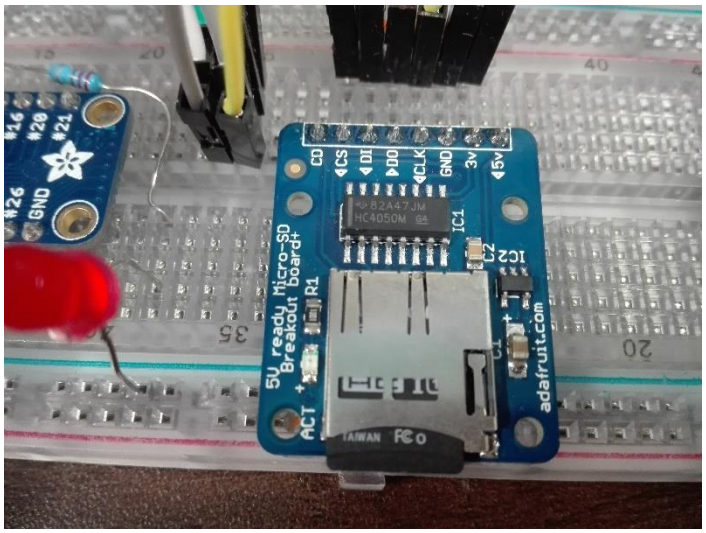


Figura 6.14 Conexión del adaptador de tarjetas microSD

6.6 Pruebas de comunicación con la estación terrena

En la sección *5.7* se describe el proceso para preparar las tres pruebas de comunicación con la estación terrena. Para llevar a cabo estas pruebas se utilizó lo siguiente:

- 10 jumpers macho-hembra.
- 1 par receptor-transmisor RF de 433 MHz.
- 1 módulo XBee con conexión serial.
- 1 módulo XBee con conexión USB.
- 1 Raspberry Pi 3 con su respectiva fuente de alimentación.
- 1 Arduino Leonardo conectado a una computadora.
- 1 protoboard y adaptador de GPIOs de Raspberry Pi a protoboard.

Prueba 1. Par RF de 433 MHz.

El objetivo de esta prueba fue intentar lograr establecer una comunicación entre la Raspberry Pi y la estación terrena, usando un par receptor-transmisor que opera a una frecuencia de 433 MHz.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se siguieron los pasos descritos en la sección *5.7* para instalar las bibliotecas usadas con el par RF.
2. Se conectó el transmisor a la Raspberry Pi y el receptor al Arduino Leonardo, de acuerdo a la *Tabla 5-11*. De los cables que se ven en la *Figura 6.15*: El jumper negro es GND, el jumper blanco es VCC y el jumper gris es DATA. De los cables que se ven en la *Figura 6.16*: El jumper negro es GND, el jumper blanco es VCC y el jumper gris es DATA.

3. Se conectó la Raspberry a su fuente de alimentación y se conectó el Arduino Leonardo a la computadora (estación terrena).
4. Se hizo un programa muy simple que enviaba un comando 10 veces y se ejecutó en la Raspberry.
5. En la estación terrena, se usó el Arduino para leer lo que recibía el receptor y mostrarlo utilizando el monitor serial del software de Arduino.

Resultado 1.

Observando los comandos mostrados en el monitor serial, fue evidente que había un problema en la transmisión de información, la estación terrena recibía aproximadamente uno de los diez comandos enviados. Esto se presentaba con el par RF separado unos 30 centímetros, cuando se aumentaba la distancia, se terminaban perdiendo los 10 comandos enviados. Ya que no se pensaba usar este par RF para la comunicación, no se investigaron las causas del error en la comunicación y se dejaron las pruebas así.

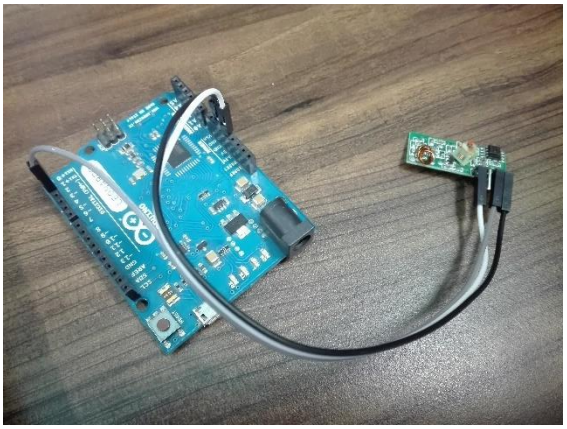


Figura 6.15 Conexión del receptor RF

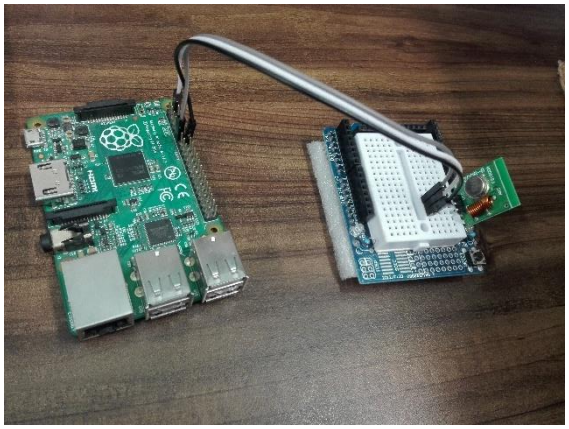


Figura 6.16 Conexión de transmisor RF

Prueba 2. Módulo XBee conectado a la OBC mediante USB.

El objetivo de la segunda prueba fue el mismo que el de la primera prueba, lograr establecer una comunicación entre la Raspberry Pi y la estación terrena, pero esta vez, utilizando un par de módulos XBee que operan en la banda de los 2.4 GHz.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se conectó el módulo XBee con el puerto serial al Arduino Leonardo y el módulo XBee con el puerto USB a la Raspberry, de acuerdo a la *Tabla 5-12*. La *Figura 6.17* muestra la conexión al Arduino Leonardo, mientras que la *Figura 6.18* muestra la conexión de los pines del módulo XBee.
2. Se conectó la Raspberry a su fuente de alimentación y se conectó el Arduino Leonardo a la estación terrena.
3. Se hizo un programa en Python que esperaba a recibir un número de la estación terrena y le regresaba el doble del número recibido.

4. Se hizo un sketch que esperaba que el usuario ingresara un número, lo enviaba a la Raspberry, esperaba el resultado y lo mostraba en el monitor serial.
5. Se cargó el sketch al Arduino y se ejecutó el programa de Python en la Raspberry.
    a. Se enviaron con el Arduino los siguientes números: 14, 15, 16 y 17.

Resultado 2.

En la *Figura 6.19* se ve que la comunicación entre los módulos XBee fue exitosa, no hubo problemas de comunicación y no se perdió información. Es posible usar esta configuración para implementar la comunicación, sin embargo, es mejor intercambiar la posición de los módulos XBee, para que la estación terrena no dependa del Arduino.

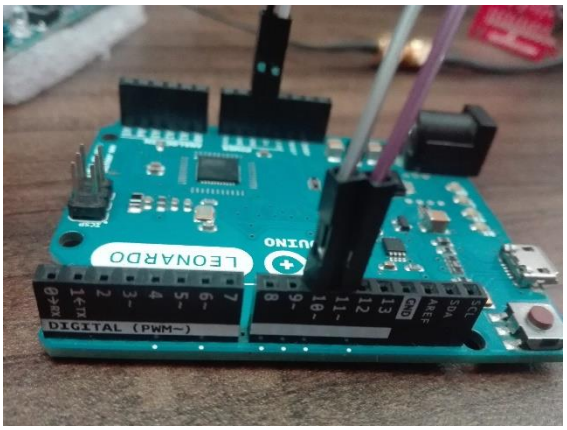


Figura 6.17 Conexión del módulo XBee al Arduino Leonardo (pines del Arduino)

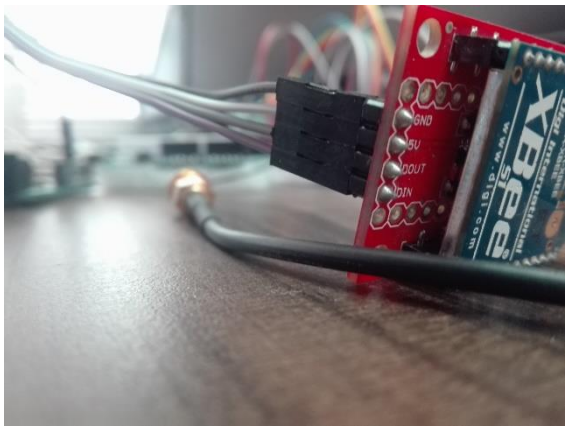


Figura 6.18 Conexión del módulo XBee al Arduino Leonardo (pines del XBee)

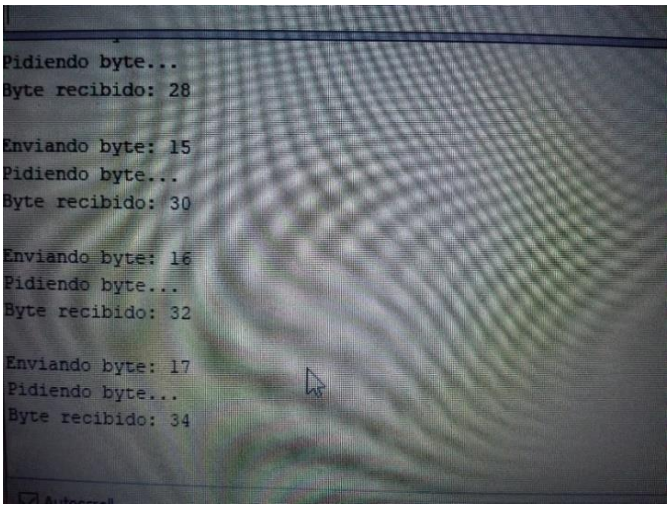


Figura 6.19 Foto del monitor serial del Arduino Leonardo al ejecutar la segunda prueba

Prueba 3. Módulo XBee conectado a la estación terrena mediante USB.

El objetivo de la tercera prueba fue verificar que se pudiera intercambiar el orden de los módulos XBee, ya que es mejor que la estación terrena utilice el módulo XBee con un puerto USB. Además, se empezó con el desarrollo de la interfaz de usuario de la estación terrena, para lo cual se utilizó LabVIEW.

Para realizar la prueba, se siguieron los siguientes pasos:

1. Se conectaron los módulos de acuerdo a la *Tabla 5-13*. La *Figura 6.20* y la *Figura 6.21* muestran la conexión de los módulos XBee.
2. Se conectó la Raspberry a su fuente de alimentación.
3. Se abrió el programa de Arduino, se seleccionó el puerto serial correspondiente al módulo XBee y se abrió el monitor serial (para poder ver y enviar información).
4. Se ejecutó el programa de la sección *B.10* en la Raspberry, este programa enviaba el mensaje “Hola estación terrena x” cada dos segundos, con el valor de x incrementando de 1 hasta 10, adicionalmente, si recibe algún mensaje lo muestra en la terminal.
5. Se usó el monitor serial para enviar los siguientes mensajes a la Raspberry: “Hola arduino leonardo” y “Prueba”.
    a. Después de comprobar que la comunicación fuera exitosa, se siguió con el siguiente punto.
6. Se desarrolló la interfaz de usuario para la estación terrena usando LabVIEW, el archivo VI corresponde al de la sección *B.12*.
7. Se ejecutó el programa del paso 4 y se utilizó la nueva interfaz para enviar los mensajes: “Comando enviado!”, “Prueba!!1” y “Hola Raspberry Pi”.

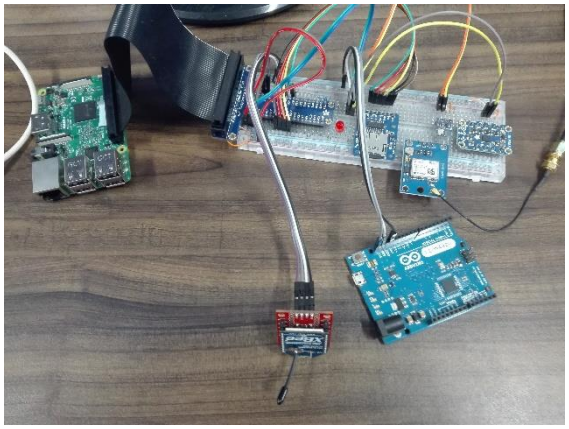


Figura 6.20 Conexión del módulo XBee a la UART de la Raspberry Pi

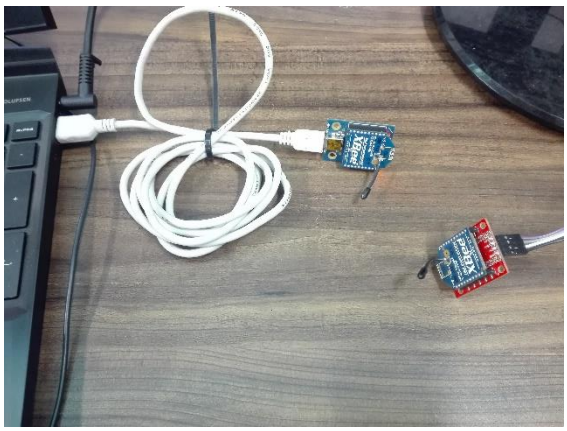


Figura 6.21 Conexión del módulo XBee a la estación terrena

Resultado 3.

De los pasos 4 y 5, se pueden observar los mensajes recibidos en la OBC en la *Figura 6.22* y los mensajes recibidos en la estación terrena en la *Figura 6.23*.

La *Figura 6.24* muestra los mensajes recibidos en la OBC y en la *Figura 6.25* se puede observar el intercambio de mensajes entre la OBC y la estación terrena.

Al finalizar estas pruebas, se logró verificar que es posible utilizar los módulos XBee en esta configuración. Se desarrolló el código que permite que la OBC reciba mensajes de la estación terrena y también pueda enviarlos. También se hizo una primera versión de la interfaz de usuario para la estación terrena.

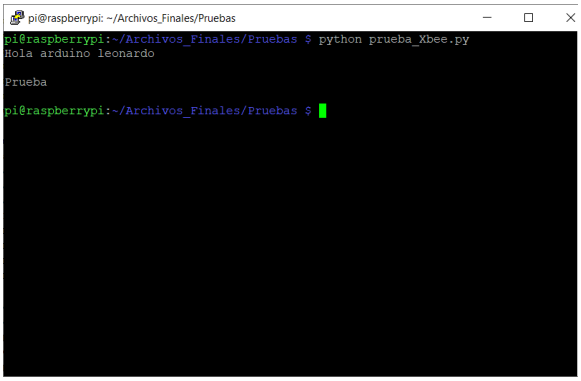


Figura 6.22 Primera ejecución del programa para enviar y recibir mensajes con el módulo XBee

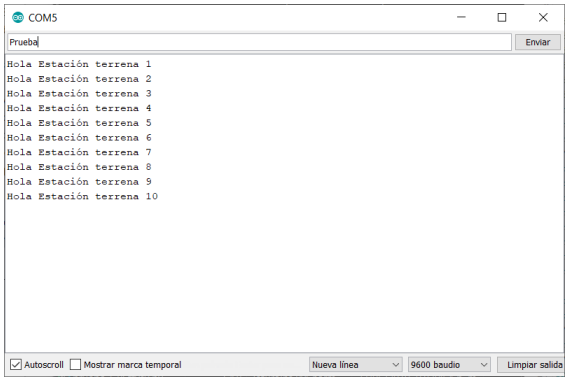


Figura 6.23 Mensajes recibidos con el módulo XBee, mostrados en el monitor serial del Arduino

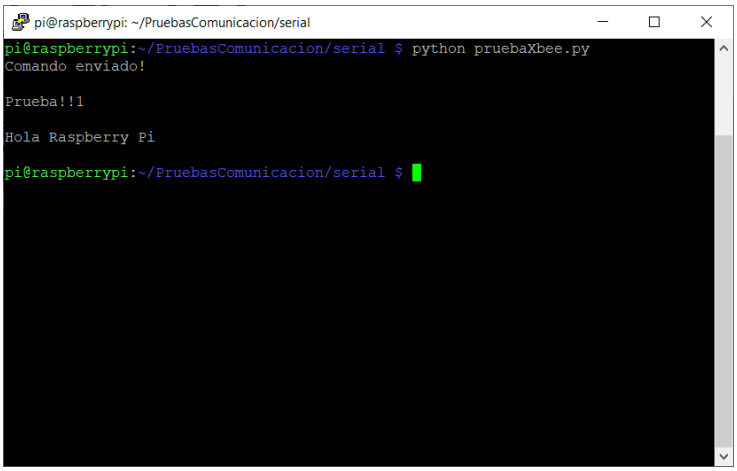


Figura 6.24 Segunda ejecución del programa en la Raspberry

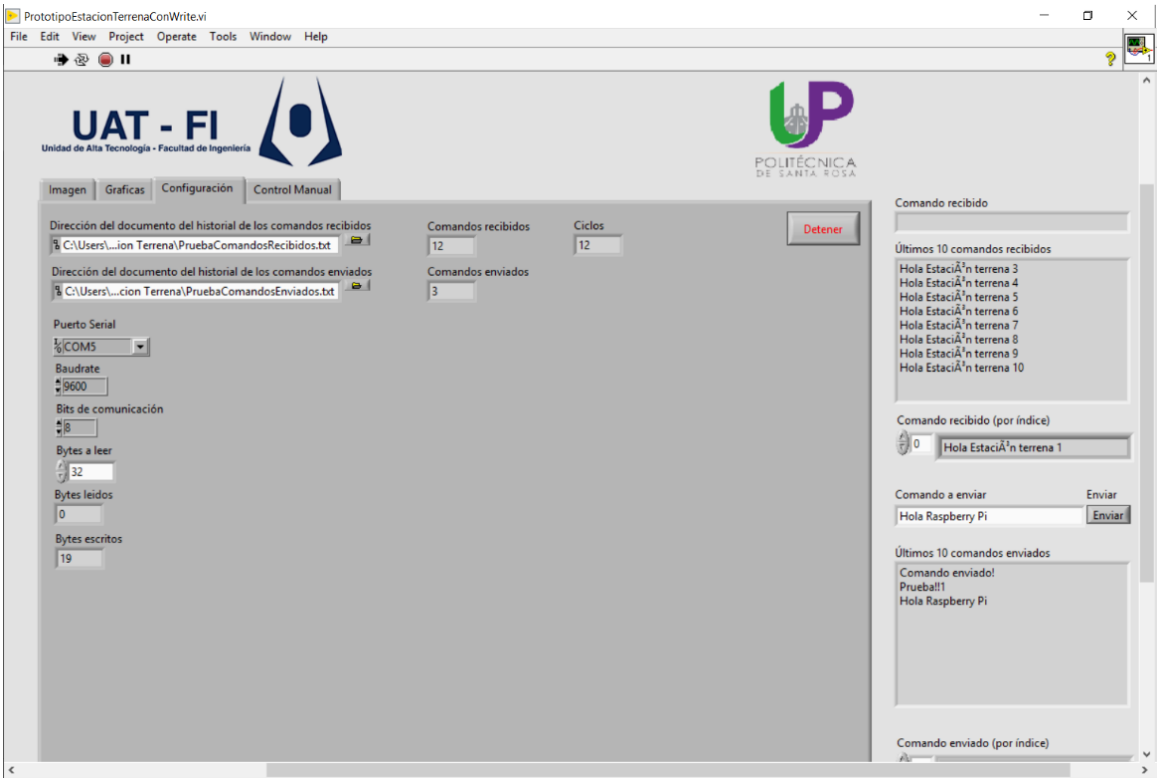


Figura 6.25 Interfaz de usuario de la estación terrena intercambiando información con la Raspberry Pi

6.7 Pruebas finales de la OBC en protoboard.

En la sección *5.8* se describieron las modificaciones realizadas para poder integrar todos los componentes de la OBC. Se hicieron diversas pruebas conforme se iba integrando cada parte al programa principal y se corrigieron los problemas que se fueron presentando, en esta sección solo se presentarán los aspectos más relevantes. Para llevar a cabo estas pruebas se utilizó lo siguiente:

- 1 tira de cables jumper macho-hembra y macho-macho.
- 1 sensor BMP180.
- 1 sensor LSM9DS0.
- 1 módulo GY-NEO6MV2.
- 1 adaptador para tarjetas microSD.
- 1 tarjeta microSD de 16 GB.
- 3 resistencias, una de 220 Ω y dos de 4.7 kΩ.
- 1 LED rojo.
- 2 botones.
- 2 módulos XBee, uno con conexión serial y otro con USB.
- 1 Raspberry Pi 3 B+ con su respectiva fuente de alimentación.
- 1 Arduino Leonardo.
- 1 computadora, desempeña el papel de la estación terrena.

- 1 protoboard y adaptador de GPIOs de Raspberry Pi a protoboard.

Los objetivos de estas pruebas fueron: llevar a cabo la integración los componentes de la OBC logrando su correcta operación y verificar que se ejecutara el código de acuerdo a los comandos recibidos de la estación terrena, revisando que se siguieran los pasos definidos en los algoritmos de control diseñados.

Para realizar las pruebas, se siguieron los siguientes pasos:

1. Se hicieron los cambios descritos en la sección *5.8* y se procedió a conectar los componentes como se muestra en la *Figura 5.20*.
    a. Se cambió la manera de conectar el GPS.
    b. Se agregó un archivo de log para tener un registro de la información de la ejecución del programa principal de la OBC.
    c. Se agregaron algunas características a la simulación de los subsistemas del rover.
2. Se definieron los comandos que tendría disponible la estación terrena para controlar al rover y se agregaron dichos comandos a la interfaz de la estación terrena.
    a. Se modificó el programa de la OBC para reconocer los comandos de la estación terrena y se hicieron pruebas.
3. Se diseñaron los algoritmos que definen qué es lo que debe hacer la OBC para llevar a cabo las tareas asignadas desde la estación terrena.
    a. Con base en estos algoritmos se hizo el desarrollo del código y se agregó al programa principal de la OBC.
4. Se revisó que los botones operaran correctamente:
    a. Botón que apaga la Raspberry Pi
        i. Se presionó el botón y se corroboró que la OBC se apagara tal y como estaba contemplado.
    b. Botón que monta y desmonta la tarjeta microSD
        i. Se presionó el botón para desmontar la microSD y se corroboró que se encendiera el LED que indica que la microSD está desmontada.
        ii. Se removió la microSD para revisar el contenido en otra computadora.
        iii. Se volvió a insertar la microSD en la OBC y se presionó el botón para montar la microSD, revisando que no se presentara ningún problema.
5. Se revisó la interacción entre la estación terrena y la OBC:
    a. Se inició la interfaz de la estación terrena y el programa principal de la OBC.
    b. Se envió cada comando disponible y se corroboró que la ejecución del código reflejara lo presente en los algoritmos diseñados.

Resultado.

Al realizar el paso 2.a se detectó un problema con la interfaz de la estación terrena, el problema consistía en que solo era posible enviar un mensaje después de recibir un mensaje. Se procedió a rediseñar la interfaz y a corroborar que funcionara adecuadamente, en la *Figura 6.27* se puede observar la nueva interfaz en la que se agregaron botones para facilitar el envío de comandos.

Conforme se fueron ejecutando los comandos de la interfaz de la estación terrena, se presentaron algunas diferencias entre lo que ejecutaba el código y lo diseñado en los algoritmos. Se hicieron las correcciones necesarias al código y se corroboró que hubiera una interacción correcta.

Para este punto, la OBC es capaz de: Obtener mediciones de temperatura, calcular la orientación del rover (a partir de las mediciones de aceleración e intensidad de campo magnético), obtener la ubicación geográfica de rover, producir un archivo con la información de la ejecución del programa principal, copiar el archivo de log a una memoria microSD para facilitar la depuración del código, intercambiar mensajes con la estación terrena y ejecutar los pasos necesarios para que el rover complete las tareas asignadas por la estación terrena.

Se cuenta con una interfaz para la estación terrena con la que se puede: configurar los archivos donde se guardará el historial de todos los comandos enviados y recibidos, seleccionar el puerto al que está conectado el módulo XBee, cambiar el número de bytes a leer y enviar comandos a la OBC para asignarle tareas.

El siguiente paso fue implementar lo que se ha hecho con la OBC pero ahora conectándola mediante la placa de circuito impreso y realizar las pruebas necesarias para verificar que funcionara de la misma manera.

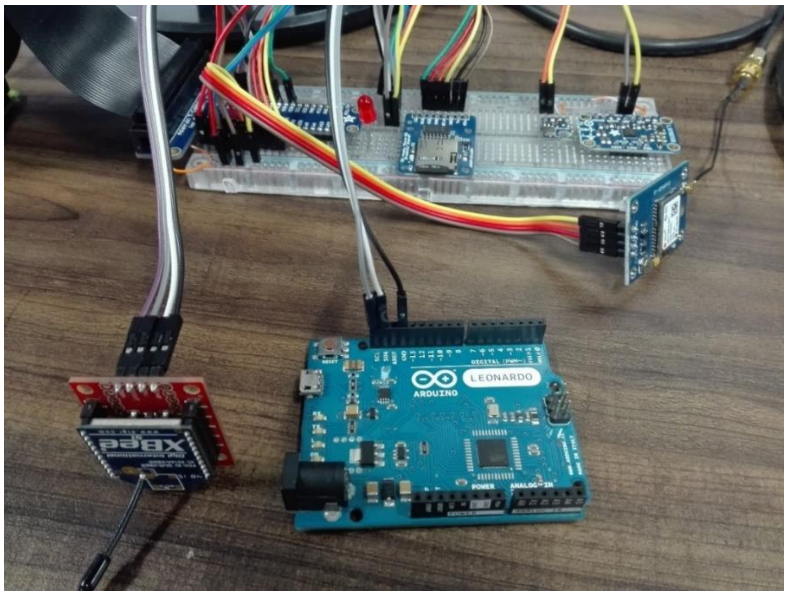


Figura 6.26 Conexión de la computadora de a bordo en una protoboard

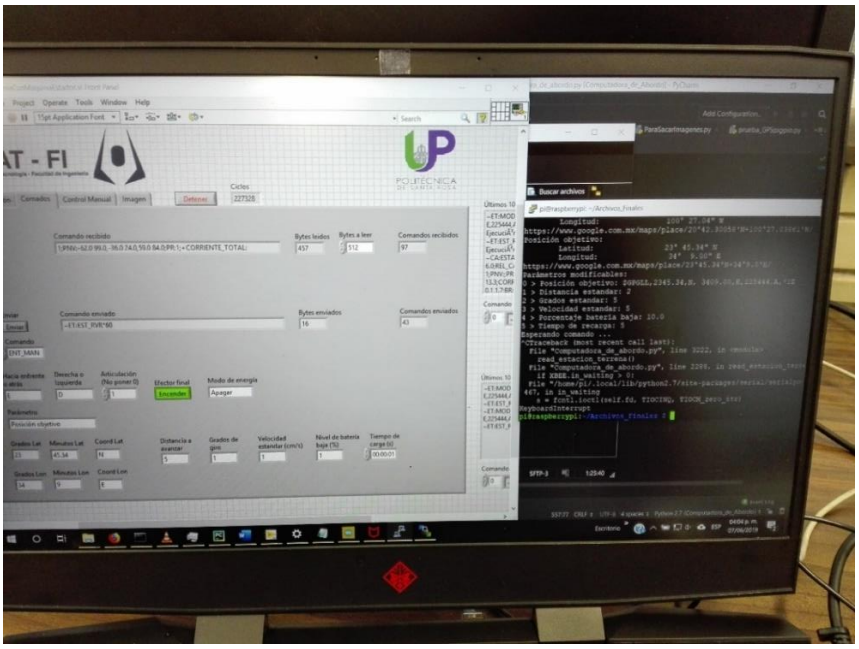


Figura 6.27 Interacción de la computadora de a bordo y la estación terrena

6.8 Pruebas de la OBC en PCB

En la sección *5.9* se describió el proceso para manufacturar la PCB de la OBC, el siguiente paso fue conectar la OBC mediante la PCB y realizar pruebas para corroborar que el funcionamiento fuera el mismo.

Al intentar verificar el correcto funcionamiento de la OBC conectada mediante la PCB, se detectaron problemas en la comunicación I2C que se presentaban al

alimentar el Arduino mediante la computadora de la estación terrena, estos problemas no se presentaban si la alimentación provenía directamente desde la Raspberry Pi. Fue por esta razón que se decidieron realizar las dos siguientes pruebas: 1) Prueba con osciloscopio, para determinar las causas del problema y 2) Prueba de emisiones electromagnéticas, para corroborar que el diseño de la PCB no produjera emisiones que pudieran interferir con el funcionamiento de los subsistemas en el rover.

6.8.1 Prueba con osciloscopio

El objetivo de esta prueba fue determinar las causas que hacían que se presentaran problemas de comunicación al alimentar al Arduino con la computadora de la estación terrena, esto se realizó mediante el análisis de los oscilogramas que mostraban las tramas de las líneas del bus I2C en diferentes condiciones, más adelante se explican dichas condiciones. Estas pruebas se llevaron a cabo en el laboratorio HiL & SiL de la Unidad de Alta Tecnología del campus Juriquilla de la UNAM. Para realizarlas, se utilizó lo siguiente:

- 1 computadora de a bordo en PCB.
- 1 computadora de a bordo en protoboard.
- 1 fuente de alimentación para la computadora de a bordo.
- 1 Arduino Leonardo con su respectivo cable de alimentación.
- 1 computadora con la interfaz de la estación terrena.
- 1 multímetro.
- 1 osciloscopio.
- 2 puntas de prueba para osciloscopio.
- 1 cautín.
- 3 conectores JST hembra serie PH a 90° de 2 pines.
- 1.5 m de cable plano de 20 vías.
- 2 puntas de prueba para osciloscopio.

Las pruebas se realizaron siguiendo los siguientes pasos:

1. Lo primero que se hizo fue corregir una omisión que causaba que la Raspberry Pi se apagara inmediatamente.
    I. Se unieron los 3 conectores JST de 2 pines a los dos botones y al LED, con una tira de 6 líneas de cable plano. Luego, se conectaron con sus respectivos conectores macho siguiendo la guía de la máscara de componentes, en la *Figura 6.28* se observan estos componentes ya conectados.
    II. Se volvió a conectar la Raspberry Pi y se comprobó que el problema estuviera resuelto.

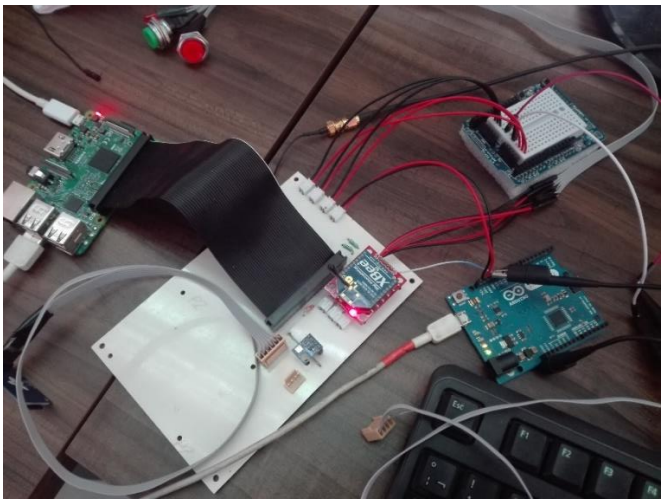


Figura 6.28 Conexión de la computadora de a bordo en la PCB

2. Se conectó la computadora de a bordo de tal manera que fuera rápido alternar la conexión entre protoboard y PCB. Esa conexión, y la del osciloscopio, se puede observar en la *Figura 6.29*.

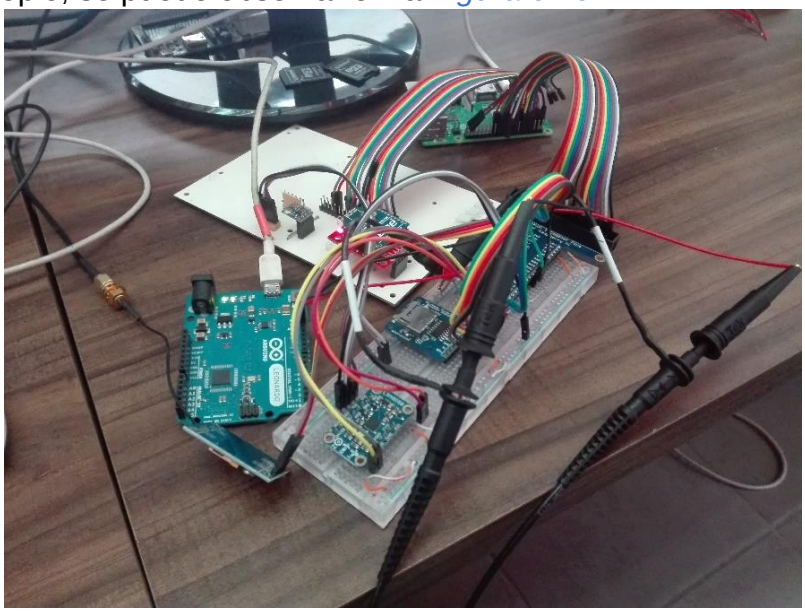


Figura 6.29 Conexión para las pruebas con el osciloscopio

   a. En el punto 3 se usarán los siguientes términos para referirse a la ejecución de los siguientes comandos y programas:
      i. **`i2cdetect -y 1`**: es el comando para averiguar las direcciones de los dispositivos que están conectados al bus I2C.
      ii. **`pruebaBMP180`**: es el programa que se encuentra en la sección *B.2* correspondiente al sensor BMP180.
      iii. **`pruebaLSM9DS0`**: es el programa que se encuentra en la sección *B.4*.

iv. **testBMP180**: es una versión modificada de **pruebaBMP180**, en la que repite 50 veces el intento de comunicación, se detiene si la comunicación es exitosa y reporta la cantidad de intentos.
v. **testLSM9DS0**: es una versión modificada de **pruebaLSM9DS0**, en la que repite 500 veces un intento de comunicación, se detiene si la comunicación es exitosa y reporta la cantidad de intentos
vi. **testArduino**: es un programa en el que repite 50 veces un intento de comunicación con el simulador de subsistemas, se detiene si la comunicación es exitosa y reporta la cantidad de intentos.

3. Se procedió a obtener los oscilogramas correspondientes a lo siguiente:
   a. Funcionamiento en la protoboard:
      i. Ejecutar el comando **i2cdetect -y 1**
      ii. Ejecutar el programa **pruebaBMP180**
      iii. Ejecutar el programa **pruebaLSM9DS0**
   b. Funcionamiento en la PCB:
      i. Ejecutar el comando **i2cdetect -y 1**
      ii. Ejecutar el programa **pruebaBMP180**
      iii. Ejecutar el programa **pruebaLSM9DS0**
      iv. Ejecutar el comando **i2cdetect -y 1**
         1) Se observó un comportamiento de capacitor en las líneas de I2C, se decidió desconectar el sensor LSM9DS0 y repetir los dos primeros puntos.
      v. Ejecutar el comando **i2cdetect -y 1** con el sensor LSM9DS0 desconectado.
      vi. Ejecutar el comando **pruebaBMP180** con el sensor LSM9DS0 desconectado.
   c. Funcionamiento en la PCB, con el sensor LSM9DS0 conectado, pero con versiones modificadas de los programas **pruebaBMP180** y **pruebaLSM9DS0** para ver si alcanzaban a responder después de cierto número de intentos:
      i. Ejecutar el programa **testBMP180**.
      ii. Ejecutar el comando **i2cdetect -y 1**.
      iii. Ejecutar el programa **testLSM9DS0**.
      iv. Ejecutar el programa **testArduino**.

Resultados.

Las figuras que se encuentran más adelante corresponden a los oscilogramas que se obtuvieron al realizar las pruebas anteriormente descritas, los cuales muestran las tramas de las líneas SDA en amarillo y de SCL en azul, del bus de comunicación I2C.

La *Figura 6.30* muestra las tramas correspondientes a la ejecución de: (*i*) el comando **`i2cdetect -y 1`**, (*ii*) el programa **`pruebaBMP180`** y (*iii*) el programa **`pruebaLSM9DS0`**. En estas, se observa un funcionamiento correcto y no hay errores de comunicación, por eso se toman como imágenes de referencia.

En la *Figura 6.31* se observan las tramas correspondientes a la ejecución de los comandos y programas descritos en el punto *3.b*. El inciso (*iv*) corresponde a la ejecución del comando **`i2cdetect -y 1`**, todos los dispositivos conectados al bus I2C respondieron, al comparar (*iv*) y (*i*) no se nota una gran diferencia, solo hay un poco de distorsión de la trama de SDA. El inciso (*v*) muestra la ejecución del programa **`pruebaBMP180`**, la comunicación con el sensor no fue exitosa, al comparar (*v*) y (*ii*) se observa una gran diferencia en la forma de las tramas, además, se aprecia un ligero comportamiento capacitivo. El inciso (*vi*) muestra la ejecución del programa **`pruebaLSM9DS0`**, nuevamente falla la comunicación, las tramas están muy distorsionadas y se presenta nuevamente el comportamiento de un capacitor descargándose; se decidió volver a ejecutar el comando **`i2cdetect -y 1`** pero reduciendo la resolución temporal para observar toda la comunicación, el inciso (*vii*) muestra claramente el efecto capacitivo. Ya que el sensor LSM9DS0 agrega la mayor longitud a las líneas SDA y SCL, se decidió desconectarlo y volver a ejecutar los dos primeros puntos del punto *3.b*. En los incisos (*viii*) y (*ix*) se puede observar una leve mejora en el comportamiento de las tramas de las vías SDA y SCL.

Al revisar las imágenes obtenidas de las pruebas donde se ejecuta el comando **`i2cdetect -y 1`** se observó que las señales tendían a estabilizarse conforme más intentos de comunicación se hacían, fue por esto que se decidió modificar los programas originales para repetir el intento de comunicación hasta tener éxito o hasta que exceder una cantidad predeterminada de intentos. Después de correr el programa **`testBMP180`** varias veces, se estableció que se necesitaban alrededor de 30 intentos para obtener una comunicación exitosa con el sensor BMP180. Para este punto se decidió investigar estrategias para contrarrestar los efectos del aumento de capacitancia en el bus I2C. Se encontró que suelen tomarse dos medidas: 1) Poner resistencias "*pull-up*" cerca de los dispositivos para mejorar el tiempo de respuesta y 2) Disminuir la velocidad de la comunicación.

Al diseñar el esquemático para la PCB, se tuvo en cuenta un escenario en el que se necesitara agregar resistencias "*pull-up*", por lo que se tenían disponibles las conexiones para estas y se contaba con resistencias de 4.7 kΩ. Antes de agregar las resistencias, se comprobó que fuera seguro usarlas: Los pines de la interfaz I2C de la Raspberry Pi cuentan con resistencias "*pull-up*" internas de 1.8 kΩ, por lo que la corriente que suministran es de 1.83 mA; Al agregar en paralelo una resistencia de 4.7 kΩ se estaría reduciendo la resistencia a aproximadamente 1.3 kΩ, lo que haría que los pines suministraran hasta 2.53 mA, lo cual está dentro del rango de operación estándar de los pines de la Raspberry Pi.

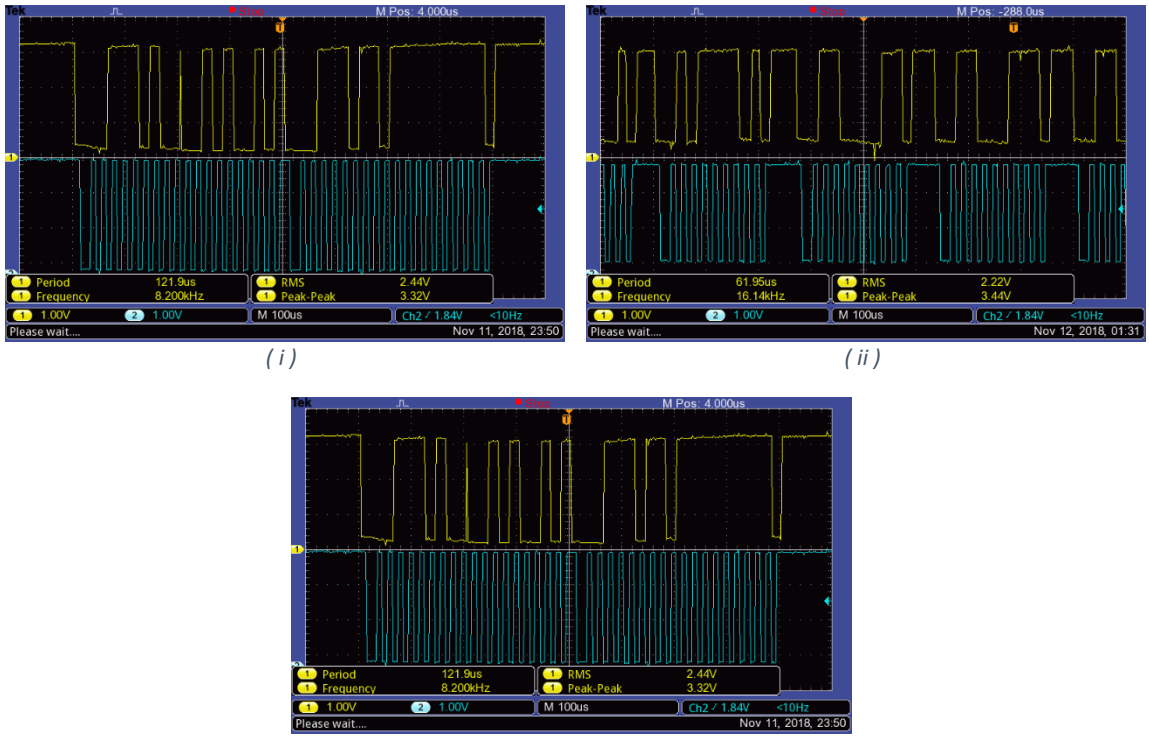
Para modificar la velocidad de la comunicación I2C se tuvo que editar el archivo de configuración, para esto se hizo lo siguiente:

- Ejecutar en la terminal el comando **`sudo nano /boot/config.txt`**
- Buscar la línea donde aparece el texto **dtparam=i2c_arm=on** y agregar lo siguiente **,i2c_arm_baudrate=50000**
- Guardar los cambios y reiniciar la Raspberry Pi.

De esta manera se disminuyó la frecuencia de SCL de 100 kHz a 50 kHz, esto no afectó negativamente el funcionamiento de la OBC ya que los comandos de control son relativamente cortos. Por ejemplo, para transferir 100 caracteres a 50 kHz se necesitan aproximadamente 32 ms, tomando en cuenta que la OBC recibe las respuestas un byte a la vez, cada solicitud mediante I2C agrega el tiempo equivalente a un carácter extra (por la estructura de los mensajes I2C).

Con esos cambios realizados, se ejecutaron los programas descritos en el punto *3.c*, los resultados se observan en la *Figura 6.32*. Después de ejecutar el programa **`testBMP180`** unas cuatro veces, se observó que el número de intentos para completar la comunicación disminuyó a aproximadamente 16 que es casi la mitad en comparación a la ejecución antes de los cambios; el inciso (*x*) muestra las tramas con las que la comunicación fue un éxito, pero se siguen presentando inconsistencias en la forma de las tramas. El inciso (*xi*) muestra la ejecución del comando **`i2cdetect -y 1`**, se aprecia un comportamiento extraño, pero no se investiga más. A pesar los cambios, la comunicación con el sensor LSM9DS0 seguía fallando después de 500 intentos, el inciso (*xii*) muestra la trama de algunos de estos intentos y los problemas que tiene el sensor para producir respuestas adecuadas (la línea SDA la maneja el sensor cuando envía respuestas). El programa **`testArduino`** logró completar la transferencia después de solo 2 intentos, a pesar de que es el que involucra la mayor cantidad de bytes (razón por la cual se excluyó de los puntos *3.a* y *3.b*), en el inciso (*xiii*) se observan las tramas de esa transferencia.

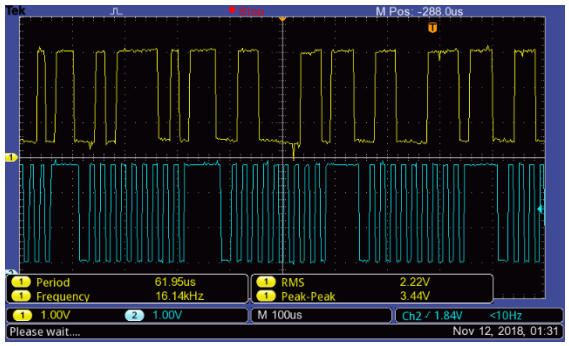
Ya que es necesario para la OBC que el sensor LSM9DS0 tenga la flexibilidad de ubicarse en el eje de rotación del rover, no es posible cambiar la manera de conectar el sensor. En otras palabras, es necesario que en una siguiente etapa del proyecto se modifique el diseño de la PCB de tal forma que se disminuya la sensibilidad de la OBC a los efectos que produce el ruido que introducen las fuentes de alimentación y el funcionamiento de otros sistemas cercanos.



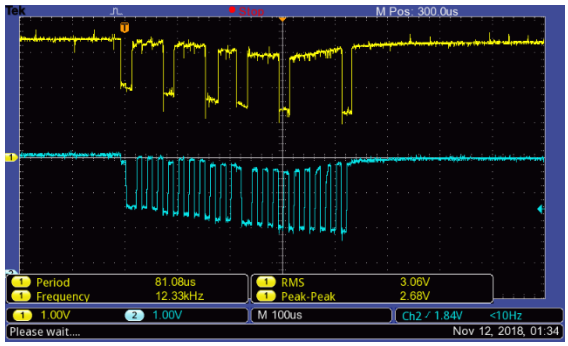
(i) *(ii)*

(iii)

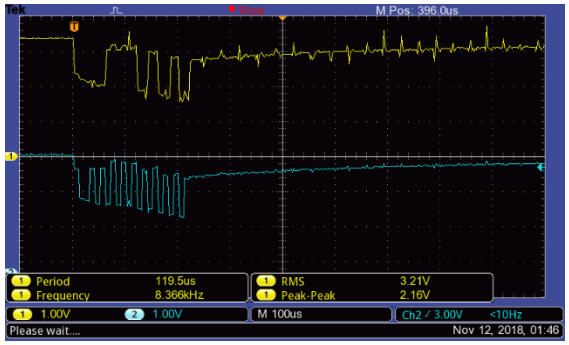
Figura 6.30 Tramas de las líneas SDA (amarillo) y SCL (azul). Las imágenes corresponden a lo siguiente: (i) Ejecución del comando i2cdetect -y 1 (ii) Ejecución del programa "pruebaBMP180" (iii) Ejecución del programa "pruebaLSM9DS0"



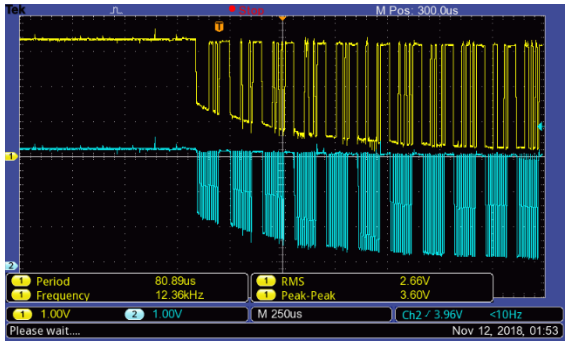
(iv)



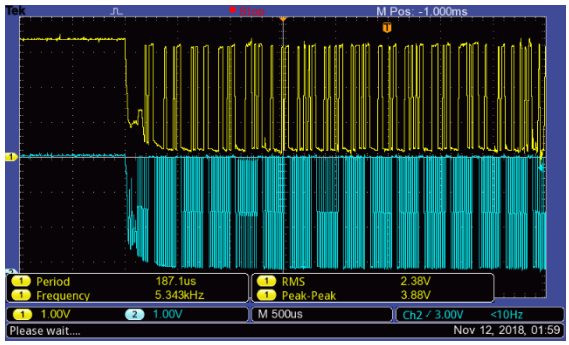
(v)



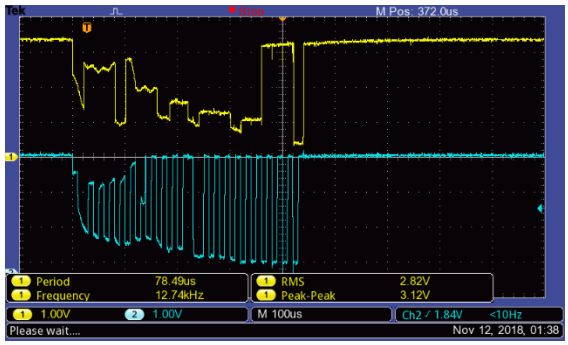
(vi)



(vii)

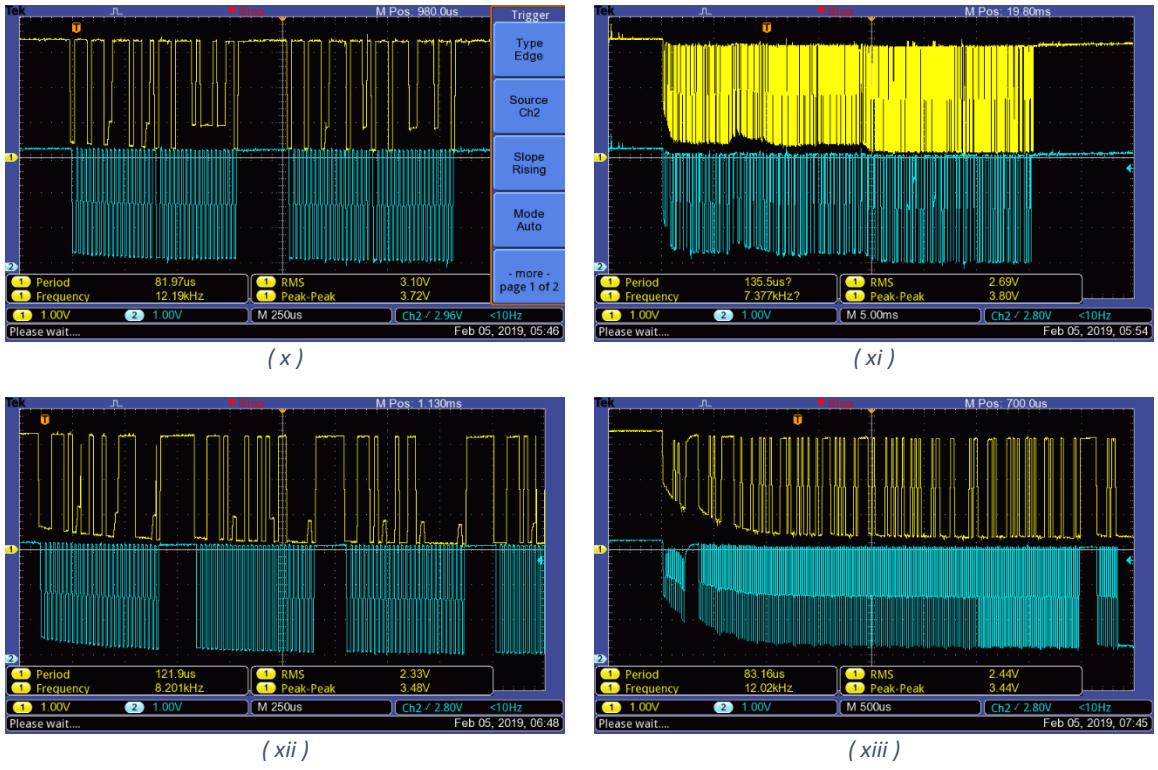


(viii)



(ix)

Figura 6.31 Tramas de las líneas SDA (amarillo) y SCL (azul). Las imágenes corresponden a lo siguiente: (iv) Ejecución del comando i2cdetect -y 1 (v) Ejecución del programa "pruebaBMP180" (vi) Ejecución del programa "pruebaLSM9DS0" (vii) Ejecución del comando i2cdetect -y 1 (viii) Ejecución del comando i2cdetect -y 1 con el sensor LSM9DS0 desconectado (ix) Ejecución del programa "pruebaBMP180" con el sensor LSM9DS0 desconectado



(x) *(xi)*

(xii) *(xiii)*

Figura 6.32 Tramas de las líneas SDA (amarillo) y SCL (azul). Las imágenes corresponden a lo siguiente: (x) Ejecución del programa "testBMP180" (xi) Ejecución del comando i2cdetect -y 1 (xii) Ejecución del programa "testLSM9DS0" (xiii) Ejecución del programa "testArduino"

6.8.2 Escaneo de emisiones electromagnéticas

El objetivo de esta prueba fue averiguar si el diseño de la tarjeta PCB provocaba la generación de emisiones electromagnéticas que pudieran interferir con el funcionamiento de los demás sistemas del rover y, de esta manera, saber si será necesario tomarlo en cuenta para la segunda versión de la PCB que se propone diseñar en una siguiente etapa del proyecto. Adicionalmente, se realizó otra prueba para observar la magnitud de las emisiones generadas por el WiFi de la Raspberry Pi.

Estas pruebas se llevaron a cabo en el "cuarto limpio" de la Unidad de Alta Tecnología del campus Juriquilla de la UNAM. Para realizarlas, se utilizó lo siguiente:

- 1 equipo EMSCAN EHX+ con su fuente de alimentación.
- 1 computadora con el software necesario instalado y un cable de ethernet.
- 1 par de pulseras electroestáticas.
- 1 bata de laboratorio.
- 1 mesa con aterrizaje a tierra.

- 1 tapete antiestático.
- 1 pliego de tela para protección contra emisiones electromagnéticas.

Condiciones con las se realizaron las pruebas:

- La computadora de a bordo siempre estuvo encendida y siendo alimentada con una fuente de 5 VDC con una corriente máxima de 2.5 A.
- En el cuarto había una red WiFi utilizada para ejecutar programas en la Rapberry Pi.
- El Arduino Leonardo estaba conectado a un puerto USB de la Raspberry Pi.
- El filtro de aire estuvo encendido durante todas las pruebas.

Prueba 1. Medición de frecuencias cercanas a 50 kHz.

El objetivo de esta prueba es comprobar el nivel de emisión de ondas electromagnéticas en frecuencias cercanas a la velocidad de transmisión con la que se lleva a cabo la comunicación I2C.

Las mediciones comparten los siguientes parámetros:

- Las celdas que realizaron la medición se muestran en la *Figura 6.33*.
- La computadora de a bordo y sus componentes fueron colocados como se muestra en la *Figura 6.34*.
- Son mediciones espaciales que muestran la amplitud de las emisiones medidas a cierta frecuencia.
- RBW: 30.5 kHz.
- ERX Control Level: Nivel 1 --- 7.5 mm.
- Impedancia de 50 Ohms, Z = 50 Ω.

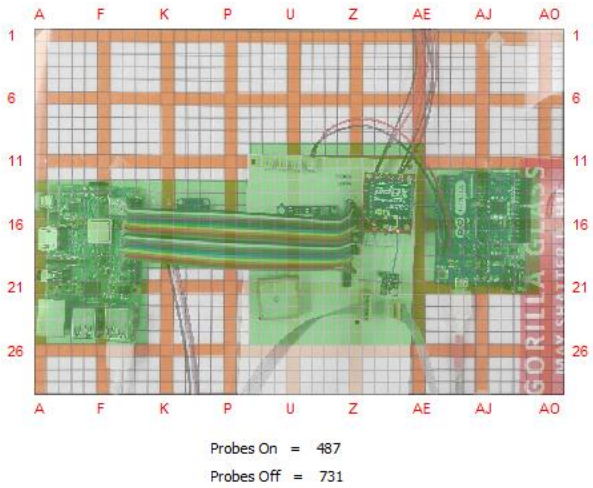


Figura 6.33 Celdas activas para la medición de emisiones (resaltadas en verde)

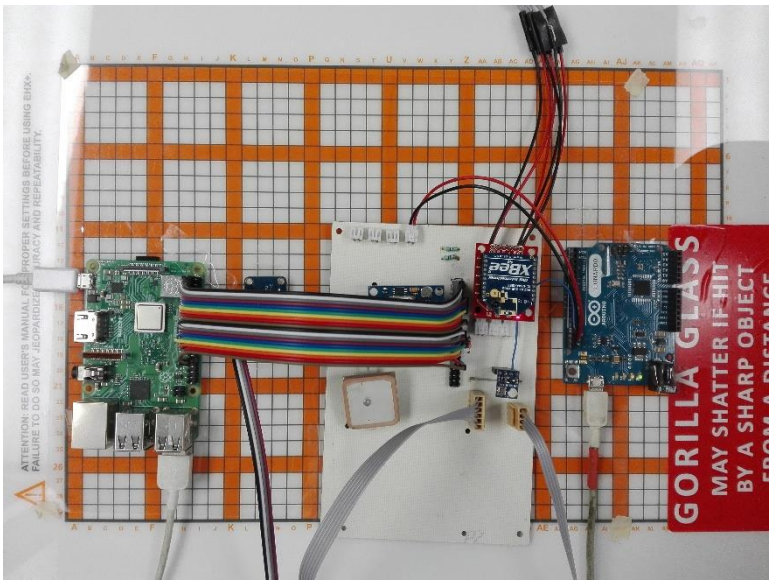


Figura 6.34 Posicionamiento de los componentes de la computadora de a bordo

Se hicieron las mediciones intercalando 4 modos de operación:

A. **Stand-by:** El sistema está encendido, pero sin ejecutar ningún programa.
B. **Programa LSM:** Se ejecuta continuamente la comunicación I2C con el sensor LSM9D0S.
C. **Programa BMP:** Se ejecuta continuamente la comunicación I2C con el sensor BMP180.
D. **Programa ARD:** Se ejecuta continuamente la comunicación I2C con el Arduino Leonardo.

Las pruebas se realizaron siguiendo los siguientes pasos:

1. Se configura la frecuencia central en la que se realizará la medición.
2. Se hace un escaneo espacial en el modo Stand-by (**A**).
3. Se hace un escaneo espacial ejecutando el programa LSM (**B**).
4. Se hace un escaneo espacial en el modo Stand-by (**A**).
5. Se hace un escaneo espacial ejecutando el programa BMP (**C**).
6. Se hace un escaneo espacial en el modo Stand-by (**A**).
7. Se hace un escaneo espacial ejecutando el programa ARD (**D**).
8. Se obtiene una comparación espacial con el resultado del punto **3** como la Test View y el resultado del punto **2** como la Reference View**.**
9. Se obtiene una comparación espacial con el resultado del punto **5** como la Test View y el resultado del punto **4** como la Reference View**.**
10. Se obtiene una comparación espacial con el resultado del punto **7** como la Test View y el resultado del punto **6** como la Reference View**.**
11. Se repiten los pasos del **2** al **10**, para tener dos conjuntos de mediciones.
12. Se repiten los pasos del **1** al **11,** variando la frecuencia central con: 30 kHz, 40 kHz, 50 kHz, 60 kHz y 70 kHz.

Observaciones de los resultados.

Con la frecuencia central de 30 kHz, la emisión más fuerte se muestra en la *Figura 6.35* y corresponde a la medición de la segunda repetición cuando la Raspberry estaba operando en el modo Stand-by, con:

- Amplitud mínima: -14.63 dBuV, -121.63 dBm, 0.688x10^{-3}pW
- Amplitud máxima: -10.48 dBuV, -117.48 dBm, 1.8x10^{-3}pW

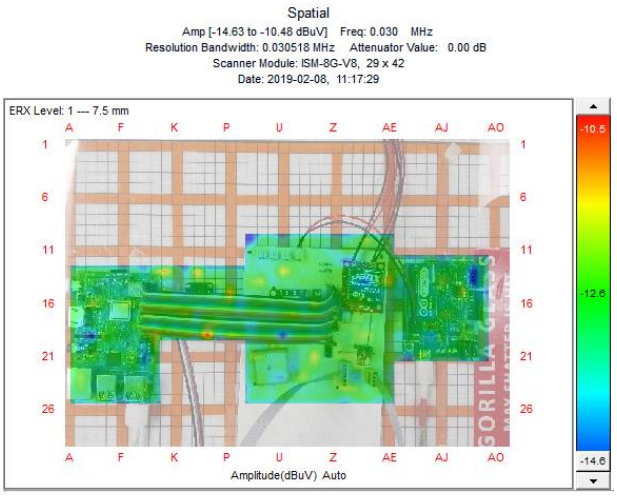


Figura 6.35 Emisión electromagnética en el modo Stand-by, midiendo en 30 kHz

Con la frecuencia central de 40 kHz, la emisión más fuerte se muestra en la *Figura 6.36* y corresponde a la medición de la segunda repetición cuando la Raspberry estaba ejecutando el programa LSM, con:

- Amplitud mínima: -14.63 dBuV, -121.63 dBm, 0.688x10^{-3}pW
- Amplitud máxima: -10.48 dBuV, -117.48 dBm, 1.8x10^{-3}pW

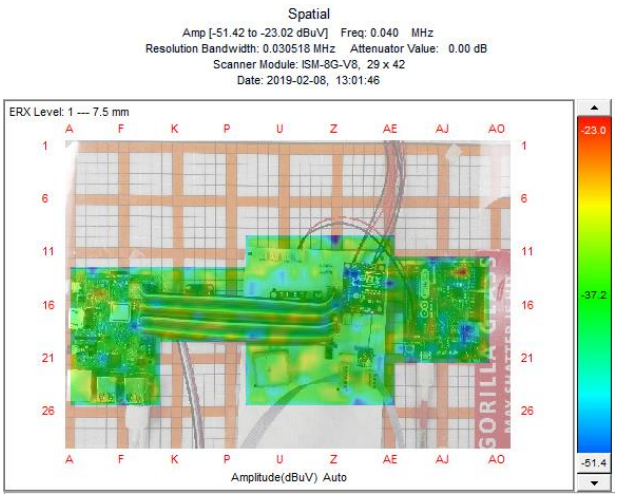


Figura 6.36 Emisión electromagnética ejecutando el programa LSM, midiendo en 40 kHz

Con la frecuencia central de 50 kHz, la emisión más fuerte se muestra en la *Figura 6.37* y corresponde a la medición de la segunda repetición cuando la Raspberry estaba ejecutando el programa BMP, con:

- Amplitud mínima: -14.63 dBuV, -121.63 dBm, 0.688x10^{-3}pW
- Amplitud máxima: -10.48 dBuV, -117.48 dBm, 1.8x10^{-3}pW

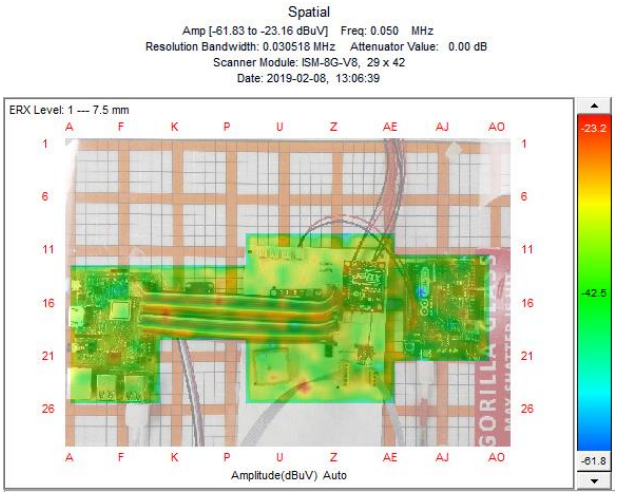


Figura 6.37 Emisión electromagnética ejecutando el programa BMP, midiendo en 50 kHz

Con la frecuencia central de 60 kHz, la emisión más fuerte se muestra en la *Figura 6.38* y corresponde a la medición de la tercera repetición cuando la Raspberry estaba operando en el modo Stand-by, con:

- Amplitud mínima: -14.63 dBuV, -121.63 dBm, 0.688x10^{-3}pW
- Amplitud máxima: -10.48 dBuV, -117.48 dBm, 1.8x10^{-3}pW

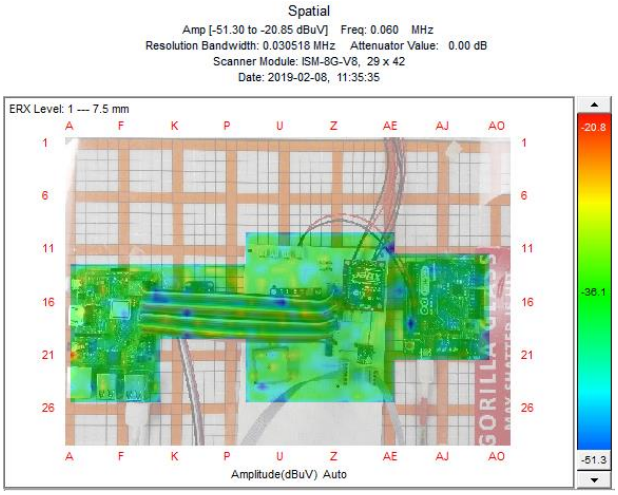


Figura 6.38 Emisión electromagnética en el modo Stand-by, midiendo en 60 kHz

Con la frecuencia central de 70 kHz, la emisión más fuerte se muestra en la *Figura 6.39* y corresponde a la medición de la primera repetición cuando la Raspberry estaba ejecutando el programa ARD, con:

- Amplitud mínima: -14.63 dBuV, -121.63 dBm, 0.688x10^{-3}pW
- Amplitud máxima: -10.48 dBuV, -117.48 dBm, 1.8x10^{-3}pW

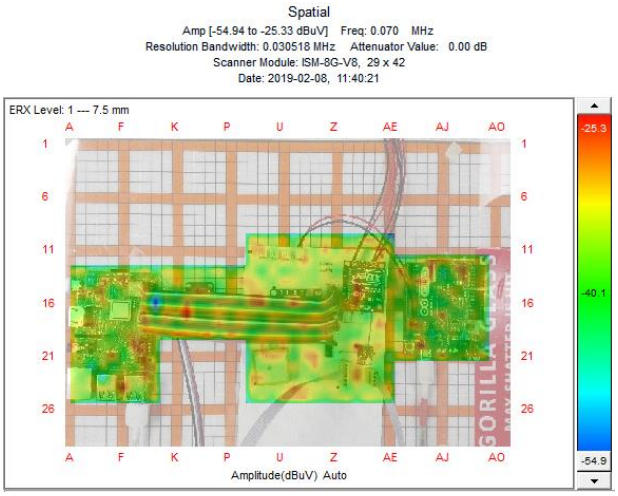


Figura 6.39 Emisión electromagnética ejecutando el programa ARD, midiendo en 70 kHz

La potencia de las emisiones es muy baja en todas las pruebas, la más alta fue de 1.8x10^{-3}pW. Por lo que se concluye que las emisiones generadas por el PCB de la computadora de a bordo no comprometen ni al funcionamiento de los sistemas cercanos a ésta ni a la comunicación con los subsistemas.

Prueba 2. Medición de potencia del WiFi.

El objetivo de esta prueba fue observar las emisiones generadas por la transmisión de información a través de WiFi.

Las mediciones comparten los siguientes parámetros:

- Las celdas que realizaron la medición se muestran en la *Figura 6.33*.
- La computadora de a bordo y sus componentes fueron colocados como se muestra en la *Figura 6.34*.
- Fue un escaneo espectral/espacial, es una medición espacial derivada del escaneo espectral, muestra la amplitud de la emisión medida para una frecuencia seleccionada en la gráfica espectral.
- ERX Control Level: Nivel 1 --- 7.5 mm.
- Impedancia de 50 Ohms, Z = 50 Ω.

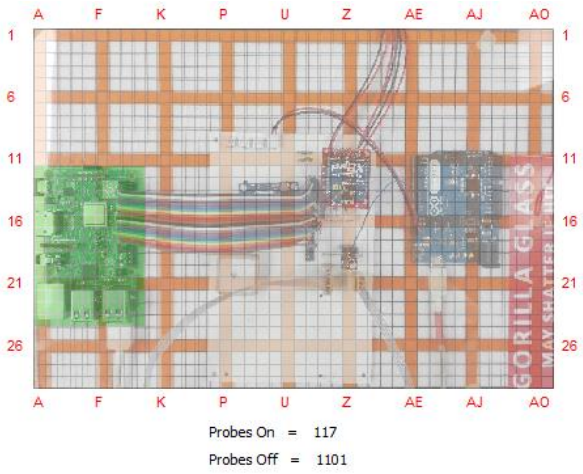


Figura 6.40 Celdas activas resaltadas en verde

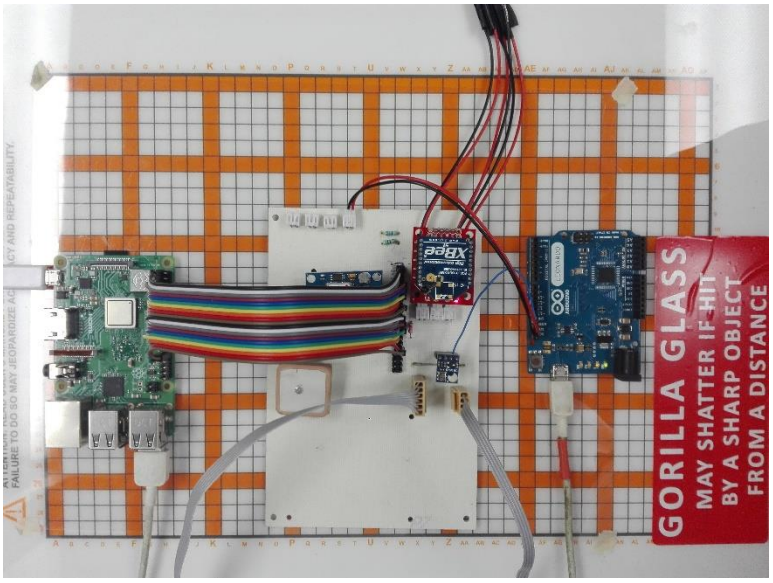


Figura 6.41 Posicionamiento de los componentes de la computadora de a bordo

La prueba se realizó siguiendo los siguientes pasos:

1. Se configura la frecuencia central en la que se realizará la medición.
    a. La frecuencia central fue de 2400 MHz.
2. Se configura el ancho de banda para la medición.
    a. El ancho de banda abarcó de 2300 MHz a 2525 MHz.
3. Se hace un escaneo espectral/espacial con la Raspberry transmitiendo información por WiFi. Se obtiene la medición espectral.
4. En la medición espectral anterior, se selecciona la frecuencia donde se observe una mayor amplitud de la emisión.

Observaciones del resultado.

En la *Figura 6.42* se puede apreciar un pico en la amplitud de las emisiones alrededor de los 2457 MHz, frecuencia que está contenida en el canal 11 de la banda de los 2.4 GHz. Además, al seleccionar esta frecuencia, se obtuvo la medición espacial derivada del escaneo espectral, la cual muestra la intensidad de las emisiones mapeadas sobre el espacio que ocupa la Raspberry Pi. La *Figura 6.43* muestra que la localización del pico más intenso de las emisiones, corresponde a la localización del módulo WiFi de la Rapberry Pi 3 B+. Por lo tanto, es posible que las emisiones medidas correspondan al intercambio de información por WiFi, sin embargo, se debe tener en cuenta que la amplitud máxima corresponde a una emisión muy débil (5.2 dBuV, -101.2 dBm, 66.2x10^{-3}pW).

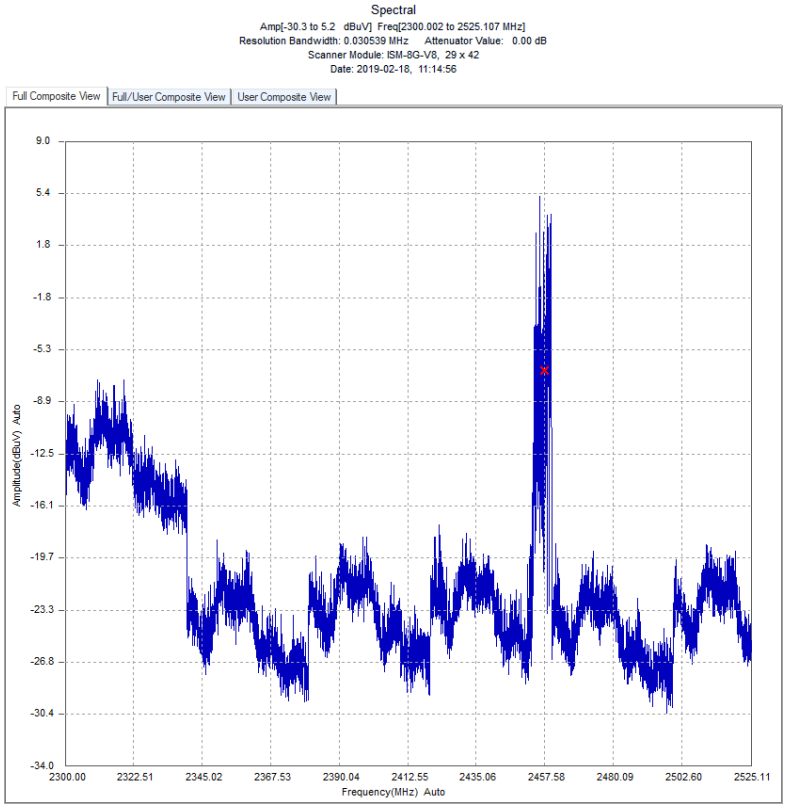


Figura 6.42 Escaneo espectral de la prueba

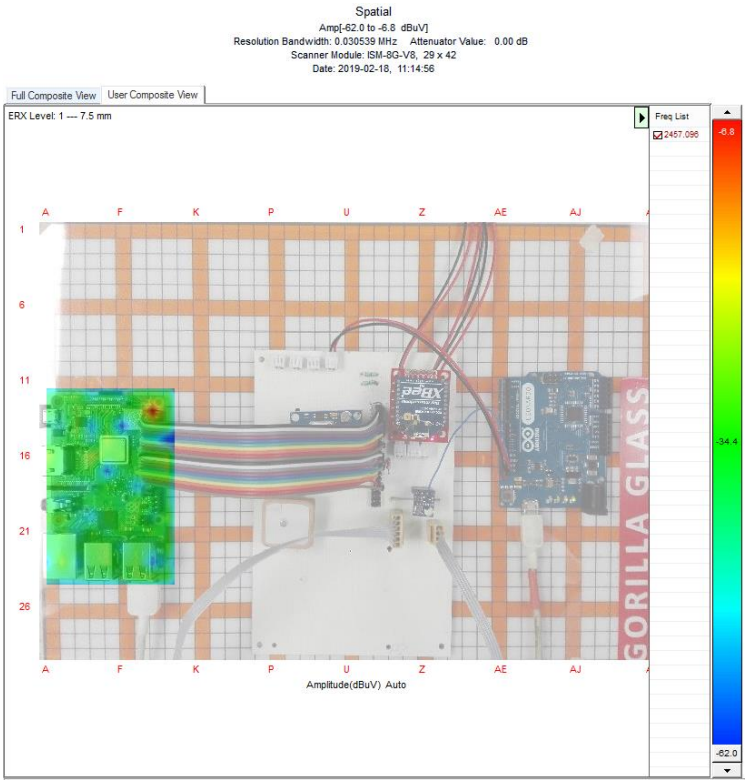


Figura 6.43 Emisión medida en la frecuencia de 2457 MHz

6.9 Pruebas finales de la computadora de a bordo y la estación terrena

Teniendo la computadora de a bordo funcionando correctamente en la PCB, se decidió realizar las pruebas finales. Para llevar a cabo estas pruebas, se utilizó lo siguiente:

- Computadora de a bordo en la PCB.
- Computadora con la interfaz de la estación terrena.
- Las respectivas fuentes de alimentación.

El objetivo de estas pruebas fue corroborar que ya no hubiera problemas con el funcionamiento de la OBC en la PCB.

Las pruebas se realizaron siguiendo los siguientes pasos:

1. Se hicieron mejoras para la estación terrena y se obtuvo la *versión final de la interfaz*, en la cual se modificó la posición y la funcionalidad de los elementos de la interfaz gráfica de acuerdo al comando que el usuario quiera enviar.
    a. Además, se agregó un comando especial “DEB_BUG” para tener acceso, desde la estación terrena, a los comandos que usa la computadora de a bordo para controlar los subsistemas (para

comprobar el correcto funcionamiento de todos los comandos: de estación terrena a computadora de a bordo y de computadora de a bordo a subsistemas). Así mismo, se agregó el reconocimiento y la ejecución de estos comandos al programa principal de la OBC.

2. Se conectó la estación terrena y la computadora de a bordo como se muestra en la *Figura 6.44*.
3. Se revisó que los componentes de la OBC funcionaran.
4. Se enviaron, desde la estación terrena, todos los comandos de la OBC a los subsistemas y se verificó su funcionamiento.
5. Se envió un comando de la estación terrena y se observó la interacción de la OBC con los subsistemas.
    a. Se repitió este paso hasta haber probado cada comando disponible y se haber corroborado que la ejecución del código reflejara lo presente en los algoritmos diseñados.

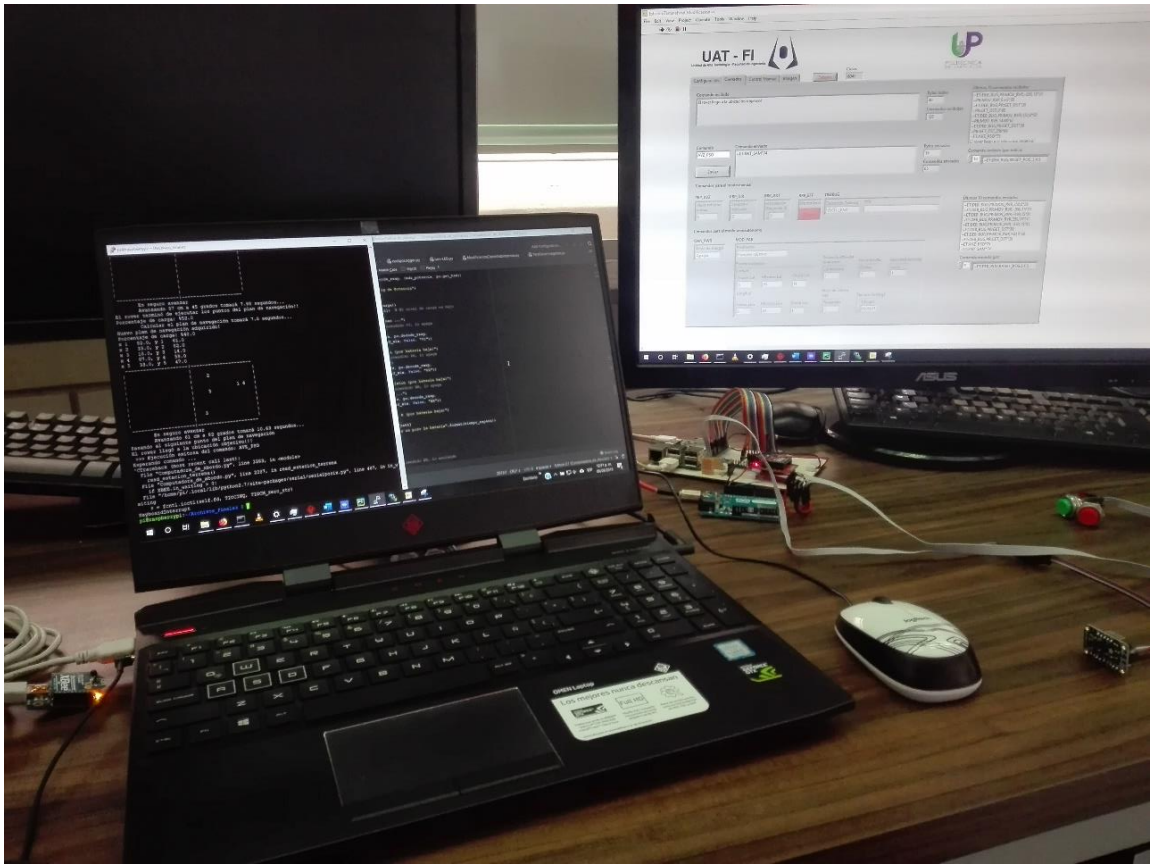


Figura 6.44 Banco de pruebas usado para las pruebas finales

Resultados.

Cuando se conectó la estación terrena y la computadora de a bordo para empezar a hacer las pruebas, se hizo una inspección de inicio para verificar que todos los componentes funcionaran correctamente. Al intentar desmontar la tarjeta microSD

no funcionaba como debería, después de revisar se encontró que, en el conector, el cable de tierra se desconectó, se hizo la reparación que se muestra en la *Figura 6.45* y se corroboró su funcionamiento. Además, se aprovechó la ocasión para corregir otras de las conexiones, en la *Figura 6.46* se puede observar esto.

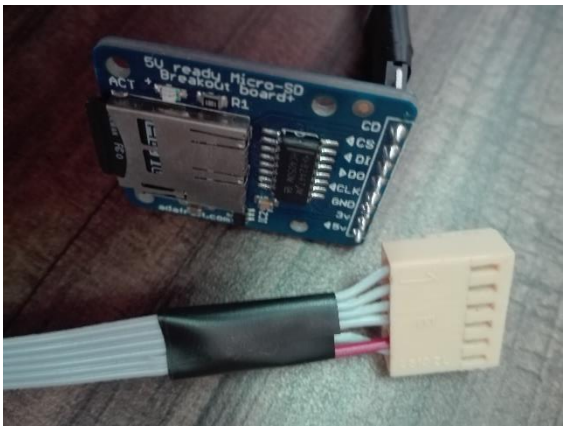


Figura 6.45 Reparación al conector de la tarjeta microSD, el cable morado es la tierra

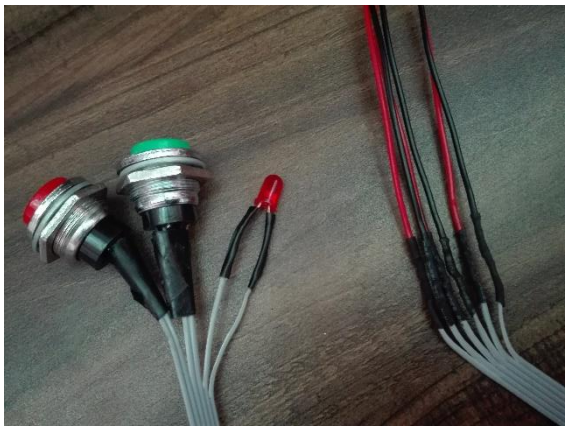


Figura 6.46 Reconexión de componentes usando thermofit en vez de usar cinta de aislar

Al ejecutar los comandos “DEB_BUG”, se identificaron algunos errores en el simulador de Arduino. Se corrigió: una función que se encarga de obtener el valor numérico de los parámetros de los comandos, el formato de la respuesta de 4 comandos y el funcionamiento de la variable que guarda la distancia recorrida por el rover.

Al ejecutar los demás comandos de la estación terrena, se comparó lo que ejecutaba la computadora de a bordo con los algoritmos que se habían establecido. Se identificó la ausencia de información para supervisar lo que estaba haciendo la OCB. Se procedió a agregar puntos en los que se muestra en la terminal lo que se está ejecutando. También se hicieron modificaciones a las rutinas que se siguen cuando se reciben algunos comandos:

- **ENT_MAN**: Se agregaron los pasos para asegurarse que los sistemas de propulsión y brazo robótico se enciendan al recibir este comando. Si hubiera algún error al encenderlos, se reporta a la estación terrena.

- **Comandos manuales**: Se agregó una función de envío de comandos solo para comandos manuales, esta función envía inmediatamente los comandos y no pide confirmación de recepción. La función robusta que se usa con los demás comandos tiene métodos para reintentar la comunicación si falla, pide la respuesta y revisa que el formato de la respuesta sea correcto, lo cual agrega tiempo a la transmisión y no se requiere con los comandos manuales.

- **GET_PNV**: Se agregaron algunos puntos donde se revisa que existan las condiciones necesarias para poder obtener el plan te navegación (que haya

suficiente pila, que los sistemas de visión y propulsión estén encendidos o se puedan encender). Se aplicaron los cambios necesarios para obtener el plan de navegación también en **AVZ_PSO**.

Al terminar las modificaciones previamente mencionadas, se volvó a probar el funcionamiento de la computadora de a bordo y se determinó que los comandos se ejecutaban como estaba previsto en los algoritmos diseñados.

7 Resultados

Computadora de a bordo

En la *Figura 7.1* se puede observar una fotografía de la computadora de a bordo del rover, a lo largo de su desarrollo se obtuvieron dos mecanismos que pueden ayudar en una siguiente etapa del proyecto:

1. Se estableció una estructura que funciona como guía para el diseño de los comandos de comunicación entre los sistemas del rover, la cual se describe en la sección *5.2.1.*
2. Se definió un esquema para agregar nuevas funcionalidades al código de los subsistemas del rover y para formular nuevos sistemas de ser necesario, la descripción de éste se encuentra en la sección *5.4.*

A continuación, se presentan las características de la computadora de a bordo, a manera de resumen:

- Puede obtener la temperatura de operación de la OBC con el sensor BMP180 que está ubicado en la PCB.
- Puede obtener la orientación respecto al norte magnético con el sensor LSM9DS0, cuenta con un cable de unos 40 cm para poder ubicarlo en el centro de gravedad del rover o en otro punto brindándole algo de flexibilidad.
- Puede obtener la ubicación geográfica del rover con el módulo GY-NEO6MV2, para el cual se cuenta con una antena compatible y un cable de 5 metros para brindar flexibilidad al momento de posicionarla en le rover.
- Dispone con dos botones, uno para apagar la OBC y el otro para desmontar la tarjeta microSD. Estos botones tienen un empaquetado y están conectados con cables de 60 cm, con esas características se pueden montar en diferentes puntos de la estructura del rover para tener fácil acceso a estos.
- Dispone de un LED que muestra el estado de la microSD, si está apagado significa que la microSD está montada, se enciende cuando la microSD está desmontada. El LED está conectado con un cable de 60 cm, lo cual le permite posicionarlo en diferentes puntos de la estructura del rover.
- Tiene un adaptador para una memoria microSD secundaria en la cual, al desmontarla, se copia un archivo de logs con los registros de operación y errores del programa principal. Este adaptador está conectado a la PCB con un cable de 60 cm para poder montarlo en diferentes puntos de la estructura del rover y tener fácil acceso a éste.
- Cuenta con un repertorio de comandos con los que puede controlar los 4 subsistemas que integrarían el rover: visión, potencia, propulsión y brazo

robótico. Estos comandos se envían por un bus de solo 2 vías usando el protocolo de comunicación serial I2C.

- El programa principal de la OBC fue desarrollado teniendo en mente uso de: 1) El paradigma de la programación orientada a objetos, se usan clases para describir los subsistemas, sus comandos y la manera de decodificar sus respuestas, 2) La guía de estilo PEP 484, que es un conjunto de pautas que sirven para explicar cómo funcionan los métodos y las funciones.
- Puede comunicarse inalámbricamente con la estación terrena y ejecutar los comandos recibidos, con el uso del módulo XBee que está ubicado en la PCB.
- El programa principal le permite seguir los algoritmos diseñados para brindarle al rover los modos de operación manual o navegación semiautónoma.

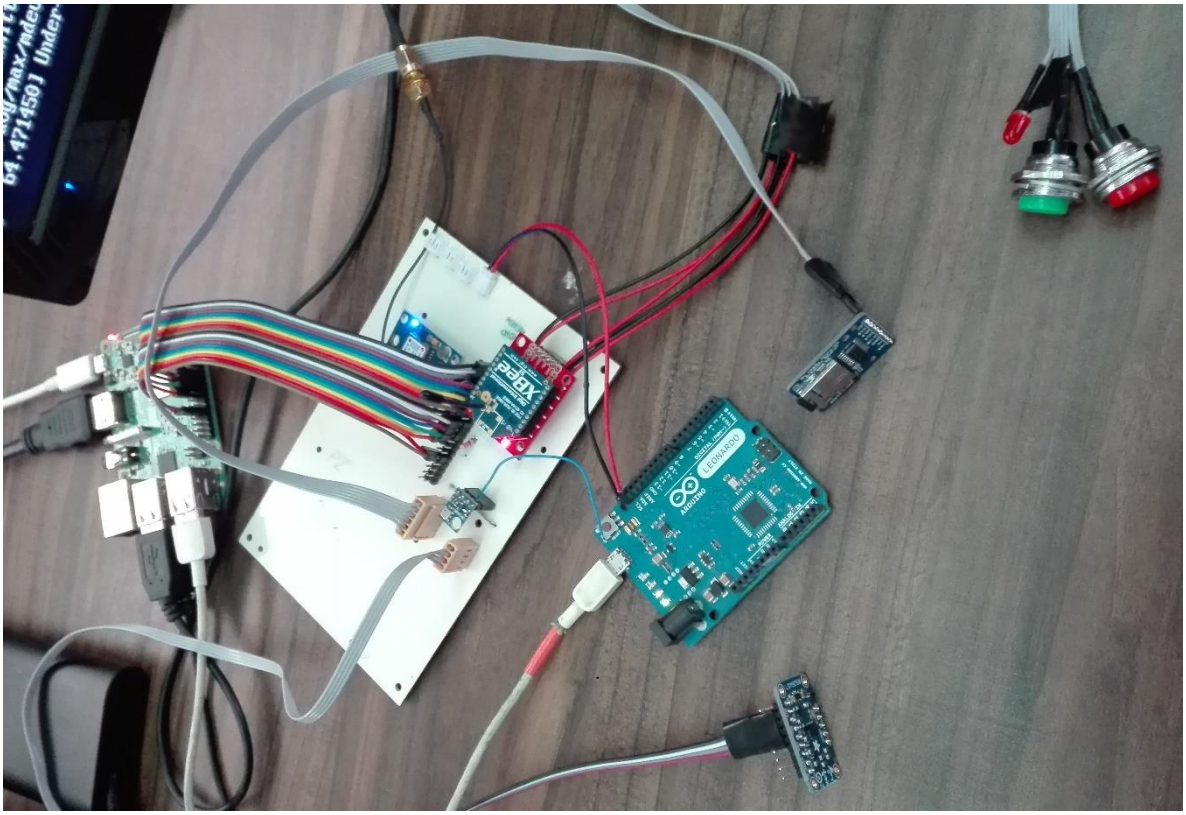


Figura 7.1 Computadora de a bordo

Estación terrena

La computadora de la estación terrena emplea un módulo XBee para comunicarse inalámbricamente con la computadora de a bordo. La estación terrena cuenta con una interfaz de usuario desarrollada con el software LabVIEW, esta interfaz es de gran ayuda para el usuario ya que le facilita: configurar los parámetros relacionados al manejo del módulo XBee, establecer la localización de los archivos en los que se

registra el historial de comunicación, visualizar la información recibida y controlar el contenido de los comandos que se le envían a la OBC. A continuación, se presenta la interfaz de usuario de la estación terrena y se explica su funcionalidad.

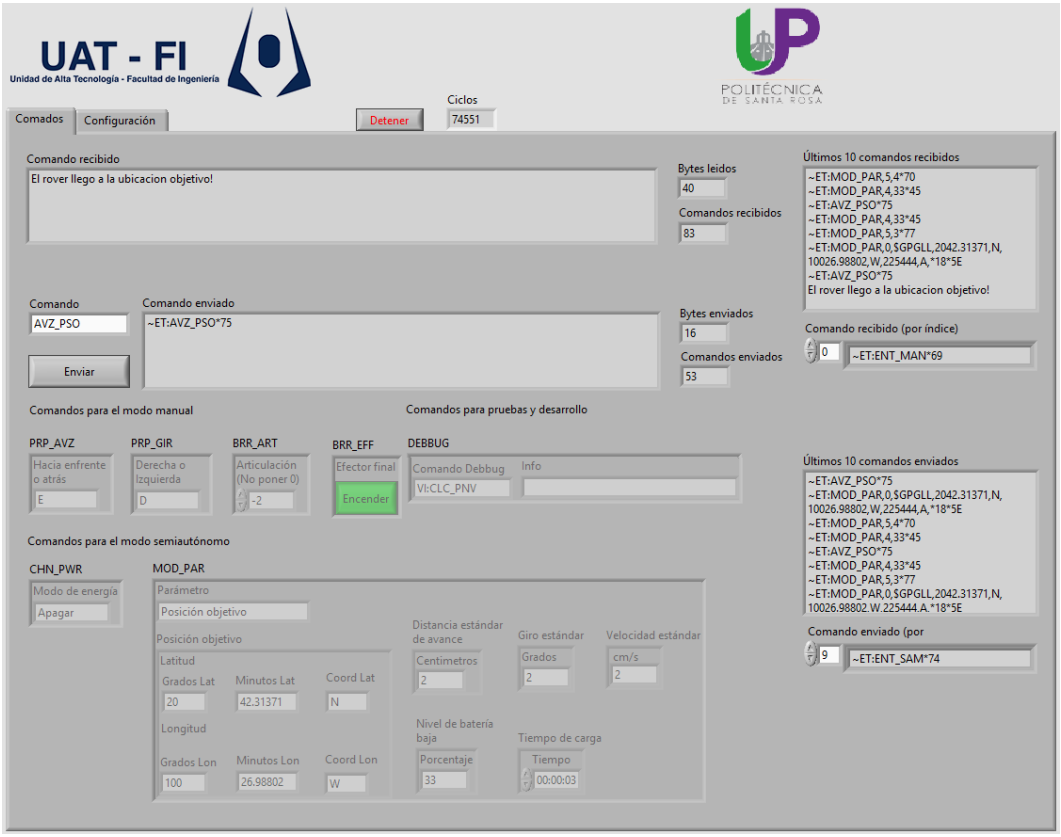


Figura 7.2 Interfaz de la estación terrena: pestaña de Comandos

En la *Figura 7.2* se puede observar la interfaz en la pestaña de *Comandos*, donde se encuentran los siguientes controles:

- **Comando recibido**: Muestra el comando recibido más recientemente.
- **Bytes leídos**: Muestra la cantidad de bytes del comando recibido.
- **Comandos recibidos**: Muestra el número total de comandos recibidos en la sesión actual.
- **Últimos 10 comandos recibidos**: Muestra una lista de los últimos 10 comandos recibidos.
- **Comando recibido (por índice)**: Muestra un comando recibido en la sesión actual, se pueden buscar los comandos recibidos mediante su índice (que va de 0 hasta lo que muestre **Comandos recibidos**).
- **Comando**: Al presionarlo, se puede seleccionar el comando que se desea enviar, no se envía nada hasta no presionar el botón “Enviar”. Dependiendo

del comando seleccionado se habilitan los controles relacionados con ese comando y se deshabilitan los demás elementos.

- **Botón "Enviar"**: Al presionarlo se envía el comando con la información que esté disponible.
- **Comando enviado**: Muestra el comando enviado más recientemente.
- **Bytes enviados**: Muestra el número de bytes del comando enviado.
- **Comandos enviados**: Muestra el número total de comandos enviados en la sesión actual.
- **Últimos 10 comandos enviados**: Muestra una lista de los últimos 10 comandos enviados.
- **Comando enviado (por índice)**: Muestra un comando enviado en la sesión actual, se pueden buscar los comandos enviados mediante su índice (que va de 0 hasta lo que muestre **Comandos enviados**).
- **Comandos para pruebas y desarrollo**:
    - **DEB_BUG**: En esta sección se encuentran los controles para pruebas y desarrollo de comandos de la computadora de a bordo.
        - **Comando debbug**: Al presionar se puede seleccionar el comando de la computadora de a bordo que se requiere ejecutar. "VI", "PO", "PR" y "BR" le indica a la computadora de a bordo a qué subsistema va dirigido el comando.
        - **Info**: Aquí se ingresan los parámetros que se agregaran al comando. Si se selecciona en **Comando debbug** los comandos especiales "VI:", "PO:", "PR:" o "BR:" se puede ingresar algún comando nuevo que no esté contemplado aún, por ejemplo, suponiendo que se está desarrollando el comando *SAY_HII* del sistema del brazo robótico. Se seleccionaría el comando "BR:" y en **Info** se agregaría el texto "SAY_HII".
- **Comandos para el modo manual**:
    - **PRP_AVZ**: En esta sección se encuentra el control del parámetro para seleccionar la dirección de avance del rover.
        - **Hacia enfrente o atrás**: Al presionar se selecciona hacia donde se avanza: "E" enfrente y "A" atrás.
    - **PRP_GIR**: En esta sección se encuentra el control del parámetro para seleccionar la dirección de giro.
        - **Derecha o Izquierda**: Al presionar se selecciona el sentido de giro: "D" derecha e "I" izquierda.
    - **BRR_ART**: En esta sección se encuentra el control del parámetro para seleccionar la articulación, del brazo robótico, a rotar y el sentido en el que rotará.
        - **Articulación**: Se puede seleccionar un valor de 1 a 3 que corresponde a el número de articulación, el signo indica si se gira hacia un lado o hacia el otro (Aún no está bien definido el sentido dependiendo de que articulación se gire).

    - **BRR_EFF**: En esta sección se encuentra el botón para encender o apagar el efector final del brazo robótico.
        - **Efector final**: Es el botón que define si se quiere encender o apagar el efector final. Al estar en “Encender” y verde se manda a encender, al estar en “Apagar” y rojo se manda a apagar.
- **Comandos para el modo semiautónomo**:
    - **CHN_PWR**: En esta sección se encuentra el control del parámetro para seleccionar el estado de energía del rover. Este comando aún no está implementado.
        - **Modo de energía**: Se puede seleccionar el modo de energía al que cambiará el rover.
    - **MOD_PAR**: En esta sección se encuentran los controles de los parámetros a cambiar en el rover.
        - **Parámetro**: Al presionar se puede seleccionar el parámetro a cambiar. Dependiendo del parámetro que se selecciones se habilitan los controles relevantes.
            - **Posición objetivo**: En esta subsección se encuentran los controles con los que se modificará la posición objetivo del rover.
                - **Latitud y longitud**: Los controles pueden ser modificados de acuerdo a la nueva posición objetivo: Grados, minutos y coordenada.
        - **Distancia estándar de avance**: Modifica la distancia mínima que avanzará el rover (en centímetros).
        - **Giro estándar**: Modifica la mínima cantidad de grados a los que girará el rover, sobre su propio eje.
        - **Velocidad estándar**: Modifica la velocidad estándar del rover.
        - **Nivel de batería baja**: Modifica el porcentaje de la pila en el cual se detecta un nivel de batería baja y se ejecutan planes de ahorro de energía.
        - **Tiempo de carga**: Modifica el tiempo que esperará el rover, en el evento de que se detecte un nivel de batería baja, para recargar su pila.
- **Botón “Detener”**: Presionar el botón detiene la ejecución de la interfaz de la estación terrena.

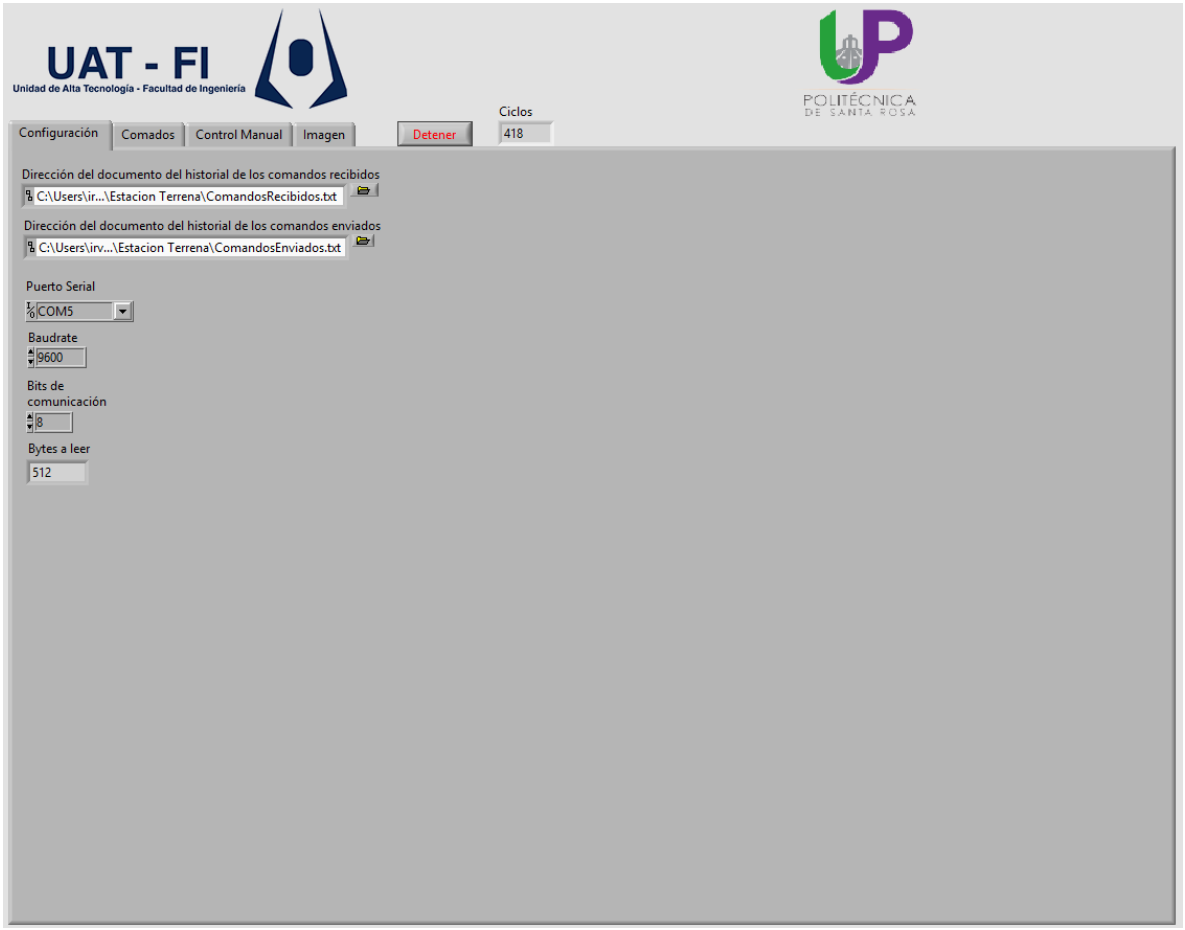


Figura 7.3 Interfaz de la estación terrena: pestaña de Configuración

En la *Figura 7.3* se puede observar la interfaz en la pestaña de *Comandos*, donde se encuentran los siguientes controles:

- Las direcciones donde se encuentran los archivos donde se guardan los historiales completos de los comandos recibidos y los comandos enviados. En cada línea del registro se encuentra: el comando, la fecha y la hora en la que se recibió o envió.
- El puerto serial en el que se conecta el módulo XBee.
- La tasa de transferencia (baudrate). El valor por default es de 9600. No se recomienda modificar este parámetro a menos de que también se haga en el módulo XBee de la computadora de a bordo.
- Los bits de comunicación. El valor por default es de 8. Solo cambiar si también se modifica en el módulo XBee de la computadora de a bordo.
- El número de bytes a leer. El valor por default es 512. Si se modifica el tamaño de las respuestas de la computadora de a bordo se puede actualizar este valor, la respuesta más larga corresponde al comando EST_RVR que regresa el estado de operación del rover.

Simulador con Arduino

Se desarrolló un simulador que cuenta con las siguientes características:

- Recibe y envía comandos mediante su interfaz I2C.
- Procesa el comando recibido haciendo lo siguiente:
  - Revisa para cual subsistema va dirigido el comando.
  - Revisa la integridad del comando, si tiene un formato correcto o es un comando válido para ese subsistema.
  - Extrae la información del comando.
- Genera y envía una respuesta con la que simula la interacción de un subsistema del rover, el contenido de la respuesta depende del comando recibido.

Exposición del proyecto

La OBC fue presentada, como parte del proyecto “Diseño y construcción de un prototipo de robot móvil semiautónomo para exploración de terrenos similares a la superficie marciana”, en:

- La 32ª Exposición de Ciencia y Tecnología del Estado de Querétaro (EXPOCYTEQ 2018) bajo la temática “Desastres naturales: Terremotos y Huracanes” y con sede en el Centro Educativo y Cultural del Estado de Querétaro (CECEQ).
- La exposición del Programa Nuevos Talentos Científicos y Tecnológicos 2018, con sede en la Universidad Politécnica de Santa Rosa Jauregi.
- El Encuentro de Jóvenes Talentos Querétaro 2018 Concurso de Prototipos Científicos y Tecnológicos, con sede en la Universidad Politécnica de Querétaro.

8 Conclusiones

Se pudo realizar la implementación de la computadora de a bordo usando componentes COTS, las pruebas realizadas indican que es capaz de comunicarse inalámbricamente con la estación terrena y ejecutar las acciones necesarias para completar las tareas que se le asignan. El desempeño de la OBC es bueno, pero no perfecto, existen oportunidades de mejora.

La placa de circuito impreso es susceptible al ruido introducido por alimentar al Arduino con una computadora externa, es necesario que en una siguiente etapa del proyecto se modifique el diseño de la PCB para evitar los problemas que se pudieran presentar al realizar pruebas con los sistemas reales del rover.

La elección de los componentes de la OBC fue adecuada, no obstante, los problemas que presenta la Raspberry Pi con la comunicación I2C al tener un dispositivo que realice *clock stretching*, provocan un ligero aumento en la complejidad del diseño del módulo de comunicación de los subsistemas.

Al finalizar el trabajo, se identificó una deficiencia en el uso de metodologías orientadas al desarrollo de proyectos del área espacial, en su lugar, se emplearon estrategias de desarrollo de software comunes como el análisis de requerimientos, el diseño de pequeños bloques de código y la constante ejecución de pruebas. A pesar de que se cumplieron los objetivos establecidos, se considera imperativo que, en una siguiente fase, se utilicen metodologías más robustas.

9 Trabajos futuros

Debido a que el desarrollo del prototipo de rover aún se encuentra en la etapa inicial, se deja pendiente para trabajos futuros lo siguiente:

- Calibración de los sensores: Es necesario calibrar los sensores una vez se haya definido la ubicación final de la computadora de a bordo y sus sensores dentro del chasis del rover.
- Mejoras a la interfaz de usuario de la estación terrena: Actualmente se presenta la información proveniente de la OBC tal y como fue recibida, es necesario desempaquetar la información y presentarla de una manera más amigable para el usuario. Mejorar la sección de comandos para la navegación en modo manual, de tal forma que sea más amigable para el usuario.
- Volver a establecer las capacidades y limitantes de los sistemas del rover: Es necesario que se defina lo que puede hacer cada sistema del rover, para verificar si los algoritmos actuales de la OBC son compatibles o si es necesario volver a diseñarlos.
- Mejoras al programa principal de la OBC:
    - Desarrollar un script para que cuando se encienda la Raspberry se ejecute inmediatamente el programa de la OBC.
    - Separar el programa en diferentes bibliotecas para dejar solo el código que sigue los algoritmos en el programa principal, de esta manera se facilitaría la lectura del código, lo cual mejoraría la experiencia de desarrollo.
    - Hacer que la recepción de comandos provenientes de la estación terrena y la ejecución de las tareas se realice de manera paralela, hasta ahora se realiza de manera secuencial.
- Desarrollar pruebas usando el Robotic Operating System y evaluar la factibilidad de implementar la OBC empleando las herramientas que provee ROS, el cual es compatible con la Raspberry Pi.

Bibliografía

[1] A. E. Mexicana, «Plan de Órbita 2.0 mapa de ruta del sector especial mexicano,» [En línea]. Available: http://www.promexico.mx/documentos/mapas-de-ruta/plan-orbita-2.0.pdf. [Último acceso: 5 Octubre 2018].

[2] A. Ellery, «Why Rovers?,» de *Planetary Rovers: Robotic exploration of the solar system*, Springer, 2016, pp. 1-4.

[3] E. Howell, «Lunokhod 1: 1st Successful Lunar Rover,» Space, 20 Diciembre 2016. [En línea]. Available: https://www.space.com/35090-lunokhod-1.html. [Último acceso: 28 Septiembre 2018].

[4] N. J. P. Laboratory, «NASA Facts | Mars Pathfinder,» [En línea]. Available: https://www.jpl.nasa.gov/news/fact_sheets/mpf.pdf. [Último acceso: 28 Septiembre 2018].

[5] N. Mars, «Mars Science Laboratory,» [En línea]. Available: https://mars.nasa.gov/programmissions/missions/present/msl/. [Último acceso: 4 Octubre 2018].

[6] B. Wilcox y . R. Jones, «MUSES-CN Nanorover Mission and Related Technology,» *Proceedings IEEE Conference,* pp. 287-295, 2000.

[7] A. Ellery, «Rover sensorimotor control systems,» de *Planetary Rovers, robotic exploration of the solar system*, Chichister, UK, Springer, 2016, pp. 146-151.

[8] T. d. J. M. Sanguino, «50 years of rovers for planetary exploration: A retrospective review for future directions,» *Robotics and Autonomous Systems,* vol. 94, pp. 172-185, 2017.

[9] M. Post, «LESSONS LEARNED FROM THE YORK UNIVERSITY ROVER TEAM (YURT) AT THE UNIVERSITYROVER COMPETITION 2008,» IAC-09-E.1.5.3, Toronto, Ontario (Canada), 2008.

[10] J. A. Grimes, «2011 Oregon State Mars Rover Design Report,» Oregon State University Robotics Club, Corvallis, Oregon, USA, 2011.

[11] M. R. Manipal, «URC 2017 | Mars Rover Manipal,» 2017. [En línea]. Available: http://marsrovermanipal.com/urc17.html. [Último acceso: 29 Noviembre 2018].

[12] K. Andrzejczak, «Safety systems in rover controller,» World Scientific News, Lodz University of Technology, Lodz, Poland, 2017.

[13] R. M. Haberle, «Understanding Mars and Its Atmosphere,» de *The Atmosphere and Climate of Mars*, Cambridge University Press, 2017, pp. 8, 16.

[14] G. Tuthill, «Planetary Fact Sheets,» NASA/MSU-Bozeman CERES Project, [En línea]. Available: http://btc.montana.edu/ceres/malcolm/cd/html/orbitsfacts.html. [Último acceso: 15 Febrero 2019].

[15] T. M. Society, «Requirements and guidelines,» University Rover Challenge 2018, [En línea]. Available: http://urc.marssociety.org/home/requirements-guidelines. [Último acceso: 25 Septiembre 2018].

[16] G. Halfacree, «Raspberry Pi Beginner's Guide,» 2018. [En línea]. Available: https://www.raspberrypi.org/magpi-issues/Beginners_Guide_v1.pdf. [Último acceso: 12 Marzo 2019].

[17] A. Silberschatz, «What operating systems do, Real-Time Embedded Systems, Real-Time CPU Scheduling, Real-Time Scheduling,,» de *Operating system concepts*, John Wiley & Sons, Inc., 2018, pp. 3-4, 45-46, 227, 792.

[18] W. Harrington, «Raspbian, Getting started with Raspbian,» de *Learning Raspbian*, Birmingham, UK, Packt Publishing, 2015, pp. 10-12, 17-20.

[19] D. o. E. E. a. C. S. University of Central Florida, «Major Programming Paradigms,» [En línea]. Available: http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html. [Último acceso: 19 Marzo 2019].

[20] G. v. Rossum, «El tutorial de Python,» Septiembre 2009. [En línea]. Available: http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf. [Último acceso: 19 Marzo 2019].

[21] M. Lutz, «How Python Runs Programs,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 27-32.

[22] M. Lutz, «Types and Operations,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 93-278.

[23] M. Lutz, «Statements and Syntax,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 319-328.

[24] M. Lutz, «Function Basics,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 473-535.

[25] M. Lutz, «Modules: The Big Picture,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 669-689.

[26] M. Lutz, «Classes and OOP,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 783-868.

[27] M. Lutz, «Exception Basics,» de *Learning Python*, California, O'Reilly Media, 2013, pp. 1081-1085.

[28] A. P. Godse, «Serial Communication,» de *Microprocessor Techniques*, Technical Publications Pune, 2008, pp. 427-430.

[29] C. Basics, «Basics of UART communication,» [En línea]. Available: http://www.circuitbasics.com/basics-uart-communication/. [Último acceso: 27 Marzo 2019].

[30] Anusha, «Basics of Serial Peripheral Interface (SPI),» Electronicshub, 20 Junio 2017. [En línea]. Available: https://www.electronicshub.org/basics-serial-peripheral-interface-spi/. [Último acceso: 27 Marzo 2019].

[31] I. BUS, «10 Bit Adressing,» I2C BUS, [En línea]. Available: https://www.i2c-bus.org/addressing/10-bit-addressing/. [Último acceso: 27 Marzo 2019].

[32] J. Valdez, «Understandingthe I2C Bus,» Junio 2015. [En línea]. Available: http://www.ti.com/lit/an/slva704/slva704.pdf. [Último acceso: 27 Marzo 2019].

[33] A. Pozo Ruz, «Sistema GPSdescripción, análisis de errores, aplicaciones y futuro,» *Mundo electrónico,* nº 306, pp. 54-59, 2000.

[34] Dimension Engineering LLC, «A beginner's guide to accelerometers,» [En línea]. Available: https://www.dimensionengineering.com/info/accelerometers. [Último acceso: 1 Abril 2019].

[35] J. Esfandyari, «Introduction to MEMS gyroscopes,» 15 Noviembre 2010. [En línea]. Available: https://electroiq.com/2010/11/introduction-to-mems-gyroscopes/. [Último acceso: 1 Abril 2019].

[36] «MEMS Presssure Sensors: The designer engineer's guide,» Avnet Abacus, 2019. [En línea]. Available: https://www.avnet.com/wps/portal/abacus/solutions/technologies/sensors/pressure-sensors/core-technologies/mems/. [Último acceso: 27 10 2019].

[37] Elm-chan.org, «How to Use MMC/SDC,» 18 Febrero 2018. [En línea]. Available: http://elm-chan.org/docs/mmc/mmc_e.html. [Último acceso: 1 Abril 2019].

[38] Xbee.cl, «¿Qué es Xbee?,» [En línea]. Available: https://xbee.cl/que-es-xbee/. [Último acceso: 28 Marzo 2019].

[39] Cletech Ltd. & Co. KG, «Raspberry Pi I2C clock-stretching bug,» Advamation.com, 17 Agosto 2013. [En línea]. Available: http://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html. [Último acceso: 6 Mayo 2019].

[40] st.com, «Using LSM303DLH for a tilt compensated electronic compass,» Agosto 2010. [En línea]. Available: https://www.pololu.com/file/0J434/LSM303DLH-compass-app-note.pdf. [Último acceso: 22 Mayo 2019].

Apéndices

A Material de apoyo

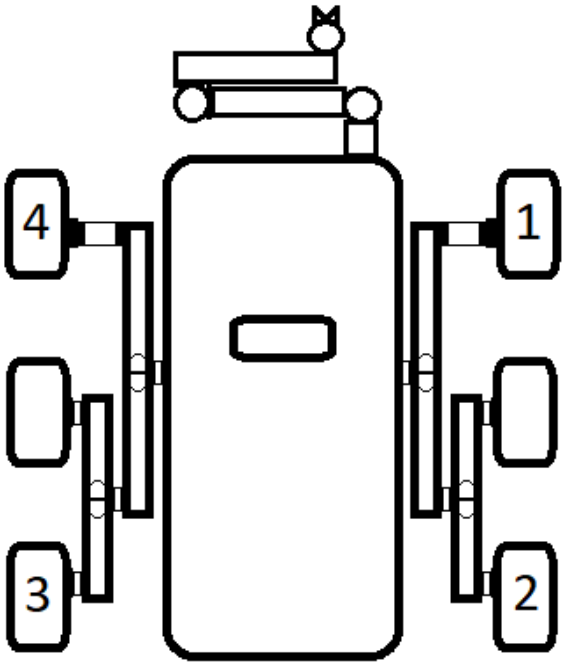
A.1 Tabla del código ASCII

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

La imagen se obtuvo del sitio: https://ascii.cl/es/

A.2 Imagen del orden de las ruedas



A.3 Imagen de las articulaciones del brazo robótico

B Código de los programas desarrollados

B.1 código para las primeras pruebas con I2C

El código de la Raspberry (en Python)

```python
# coding=utf-8
'''
    Pines del bus I2C de la Raspberry
GPIO 2 -> SDA
GPIO 3 -> SCL
'''
import smbus
import time

bus = smbus.SMBus(1)
# Dirección del esclavo
address = 0x04
# Instrucciones disponibles
saludo = 0x01
despedida = 0xff
pideDatos = 0x10

def writeCommand(cmd, nums = [0x00, 0]):
    if cmd != pideDatos:
        bus.write_byte(address, cmd)
    else:
        bus.write_i2c_block_data(address, cmd, nums)
    return -1

def readResponse():
    number = bus.read_byte(address)
    return number

while True:
    print("Que comando enviar:")
    print("\t1 : Saludar al Arduino (0x01)")
    print("\t2 : Despedir al Arduino (0xFF)")
    print("\t3 : Hacer sumar dos numeros al Arduino (0x10)")
    opc = input("\n")
    if opc == 1:
        writeCommand(saludo)
    elif opc == 2:
        writeCommand(despedida)
    else:
        digs = [0, 0]
        for x in range(2):
            digs[x] = input("Numero %d: " % (x + 1))
        writeCommand(pideDatos, digs)
    time.sleep(0.1)

    number  = readResponse()
    if opc in [1, 2]:
        print "Info recibida: " + format(number, '#04X')
    else:
        print "Info recibida: ", number
```

El código de Arduino

```arduino
#include <Wire.h>

#define SLAVE_ADDRESS 0x04
int number = 0;
int state = 0;
int modifier = 5;
unsigned int saludar = 0x01;
unsigned int despedir = 0x10;
unsigned int sumar = 0xff;

void setup() {
  Serial.begin(9600); // Para el monitor serial
  // Inicializando el Arduino como esclavo
  Wire.begin(SLAVE_ADDRESS);

  // Definiendo callbacks para la comunicación i2c
  Wire.onReceive(receiveData);
  Wire.onRequest(sendData);

  Serial.println("Listo");
}

void loop() {
  delay(100);
}

// Callback para recibir datos
void receiveData(int byteCount){
  int c = 0;
  int temp;
  while(Wire.available()) {
    number = Wire.read();
    Serial.print("Comando recibido: ");
    Serial.println(number, HEX);
    if(c == 1){temp = number;}
    c ++;
  }
  if(c > 1){
    number = number + temp;
  }
}

// Callback para enviar datos
void sendData(){
  Wire.write(number);
}
```

B.2 código para la prueba de obtención de temperatura

Código usado con el sensor DHT11:

```
# coding=utf-8
import sys
import time
import Adafruit_DHT

# Configuracion del tipo de sensor DHT
sensor = Adafruit_DHT.DHT11

# GPIO al cual estará conectado (pin 7)
gpio = 4

try:
    # Se repite el bucle hasta que haya una interrupción del teclado
    while True:
        # Obtiene la humedad y la temperatura del sensor
        humedad, temperatura = Adafruit_DHT.read_retry(sensor, gpio)

        # Imprime en la consola las variables temperatura y humedad con un decimal
        print('Temperatura={0:0.2f}°C  Humedad={1:0.2f}%'.format(temperatura, humedad))

        # Duerme 3 segundos
        time.sleep(3)
# Se ejecuta en caso de que falle alguna instruccion dentro del try o se presente una interrupción del teclado
except Exception, e:
    print str(e)
```

Código usado con el sensor BMP180:

```
# coding=utf-8
# Se importa la librería que funciona para los sensores: BMP085 y BMP180
import Adafruit_BMP.BMP085 as BMP085
# Crea un objeto que contendrá la información del sensor BMP180
bmp180 = BMP085.BMP085()

# Imprime la temperatura, la presión, la altitud y la presión al nivel del mar
print 'Temp = {0:0.2f} *C'.format(bmp180.read_temperature())
print 'Pressure = {0:0.2f} Pa'.format(bmp180.read_pressure())
print 'Altitude = {0:0.2f} m'.format(bmp180.read_altitude())
print 'Sealevel Pressure = {0:0.2f} Pa'.format(bmp180.read_sealevel_pressure())
```

B.3 código para la prueba con el MPU6050

```python
# coding=utf-8
import smbus
import time

"""This program handles the communication over I2C
between a Raspberry Pi and a MPU-6050 Gyroscope / Accelerometer combo.
Made by: MrTijn/Tijndagamer
Released under the MIT License
Copyright (c) 2015, 2016, 2017 MrTijn/Tijndagamer
"""

class mpu6050:
    # Global Variables
    GRAVITIY_MS2 = 9.80665
    address = None
    bus = None
    # Scale Modifiers
    ACCEL_SCALE_MODIFIER_2G = 16384.0
    ACCEL_SCALE_MODIFIER_4G = 8192.0
    ACCEL_SCALE_MODIFIER_8G = 4096.0
    ACCEL_SCALE_MODIFIER_16G = 2048.0
    GYRO_SCALE_MODIFIER_250DEG = 131.0
    GYRO_SCALE_MODIFIER_500DEG = 65.5
    GYRO_SCALE_MODIFIER_1000DEG = 32.8
    GYRO_SCALE_MODIFIER_2000DEG = 16.4
    # Pre-defined ranges
    ACCEL_RANGE_2G = 0x00
    ACCEL_RANGE_4G = 0x08
    ACCEL_RANGE_8G = 0x10
    ACCEL_RANGE_16G = 0x18
    GYRO_RANGE_250DEG = 0x00
    GYRO_RANGE_500DEG = 0x08
    GYRO_RANGE_1000DEG = 0x10
    GYRO_RANGE_2000DEG = 0x18
    # MPU-6050 Registers
    PWR_MGMT_1 = 0x6B
    PWR_MGMT_2 = 0x6C
    ACCEL_XOUT0 = 0x3B
    ACCEL_YOUT0 = 0x3D
    ACCEL_ZOUT0 = 0x3F
    TEMP_OUT0 = 0x41
    GYRO_XOUT0 = 0x43
    GYRO_YOUT0 = 0x45
    GYRO_ZOUT0 = 0x47
    ACCEL_CONFIG = 0x1C
    GYRO_CONFIG = 0x1B

    def __init__(self, address, bus=1):...

    def read_i2c_word(self, register):...

    def get_temp(self):...

    def set_accel_range(self, accel_range):...

    def read_accel_range(self, raw = False):...

    def get_accel_data(self, g = False):...

    def set_gyro_range(self, gyro_range):...

    def read_gyro_range(self, raw = False):...

    def get_gyro_data(self):...

    def get_all_data(self):...

if __name__ == "__main__":
    mpu = mpu6050(0x68)
    print("Durmiendo 2 segundos para que se calibre")
    time.sleep(2)
    while True:
        print("Temp: {:5.2f}°C".format(mpu.get_temp()))
        accel_data = mpu.get_accel_data()
        print("\nAceleración:")
        print("\tX: {:6.2f} m/s^2".format(accel_data['x']))
        print("\tY: {:6.2f} m/s^2".format(accel_data['y']))
        print("\tZ: {:6.2f} m/s^2".format(accel_data['z']))
        gyro_data = mpu.get_gyro_data()
        print("\nVelocidad angular:")
        print("\tX: {:6.2f} deg/s".format(gyro_data['x']))
        print("\tY: {:6.2f} deg/s".format(gyro_data['y']))
        print("\tZ: {:6.2f} deg/s".format(gyro_data['z']))
        time.sleep(1)
```

B.4 código para la prueba con el sensor LSM9DS0

```
# coding=utf-8

# The MIT License (MIT)
#
# Copyright (c) 2017, Jack Weatherilt
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
#  1. Redistributions of source code must retain the above copyright notice,
#     this list of conditions and the following disclaimer.
#
#  2. Redistributions in binary form must reproduce the above copyright notice,
#     this list of conditions and the following disclaimer in the documentation
#     and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

import math
import time
import smbus

# The same address is used for both the magnetometer and accelerometer, but
# each has their own variable, to avoid confusion.
LSM9DS0_MAG_ADDRESS = 0x1D
LSM9DS0_ACCEL_ADDRESS = 0x1D
LSM9DS0_GYRO_ADDRESS = 0x6B

# LSM9DS0 gyrometer registers
LSM9DS0_WHO_AM_I_G = 0x0F
LSM9DS0_CTRL_REG1_G = 0x20
LSM9DS0_CTRL_REG3_G = 0x22
LSM9DS0_CTRL_REG4_G = 0x23
LSM9DS0_OUT_X_L_G = 0x28
LSM9DS0_OUT_X_H_G = 0x29
LSM9DS0_OUT_Y_L_G = 0x2A
LSM9DS0_OUT_Y_H_G = 0x2B
LSM9DS0_OUT_Z_L_G = 0x2C
LSM9DS0_OUT_Z_H_G = 0x2D

# 2D gyroscope low-high register tuple
LSM9DS0_OUT_XYZ_LH_G = ((LSM9DS0_OUT_X_L_G, LSM9DS0_OUT_X_H_G),
                        (LSM9DS0_OUT_Y_L_G, LSM9DS0_OUT_Y_H_G),
                        (LSM9DS0_OUT_Z_L_G, LSM9DS0_OUT_Z_H_G))

# LSM9DS0 temperature addresses
LSM9DS0_OUT_TEMP_L_XM = 0x05
LSM9DS0_OUT_TEMP_H_XM = 0x06

# Temperature low-high register tuple
LSM9DS0_OUT_TEMP_LH = (LSM9DS0_OUT_TEMP_L_XM, LSM9DS0_OUT_TEMP_H_XM)

# Magnetometer addresses
LSM9DS0_STATUS_REG_M = 0x07
LSM9DS0_OUT_X_L_M = 0x08
LSM9DS0_OUT_X_H_M = 0x09
LSM9DS0_OUT_Y_L_M = 0x0A
LSM9DS0_OUT_Y_H_M = 0x0B
LSM9DS0_OUT_Z_L_M = 0x0C
LSM9DS0_OUT_Z_H_M = 0x0D

# 2D magnetometer low-high register tuple
LSM9DS0_OUT_XYZ_LH_M = ((LSM9DS0_OUT_X_L_M, LSM9DS0_OUT_X_H_M),
                        (LSM9DS0_OUT_Y_L_M, LSM9DS0_OUT_Y_H_M),
                        (LSM9DS0_OUT_Z_L_M, LSM9DS0_OUT_Z_H_M))

# Shared (mag and accel) addresses
LSM9DS0_WHO_AM_I_XM = 0x0F
LSM9DS0_INT_CTRL_REG_M = 0x12
LSM9DS0_INT_SRC_REG_M = 0x13
LSM9DS0_CTRL_REG1_XM = 0x20
LSM9DS0_CTRL_REG2_XM = 0x21
LSM9DS0_CTRL_REG5_XM = 0x24
LSM9DS0_CTRL_REG6_XM = 0x25
LSM9DS0_CTRL_REG7_XM = 0x26

# Accelerometer addresses
LSM9DS0_OUT_X_L_A = 0x28
LSM9DS0_OUT_X_H_A = 0x29
LSM9DS0_OUT_Y_L_A = 0x2A
LSM9DS0_OUT_Y_H_A = 0x2B
LSM9DS0_OUT_Z_L_A = 0x2C
LSM9DS0_OUT_Z_H_A = 0x2D

# 2D accelerometer low-high register tuple
LSM9DS0_OUT_XYZ_LH_A = ((LSM9DS0_OUT_X_L_A, LSM9DS0_OUT_X_H_A),
                        (LSM9DS0_OUT_Y_L_A, LSM9DS0_OUT_Y_H_A),
                        (LSM9DS0_OUT_Z_L_A, LSM9DS0_OUT_Z_H_A))

# Various settings included in the Arduino library. I haven't used these,
# to keep to a default setting for simplicity, however, users can change
# the settings easily.
LSM9DS0_ACCELRANGE_2G = 0b000 << 3
LSM9DS0_ACCELRANGE_4G = 0b001 << 3
LSM9DS0_ACCELRANGE_6G = 0b010 << 3
LSM9DS0_ACCELRANGE_8G = 0b011 << 3
LSM9DS0_ACCELRANGE_16G = 0b100 << 3


LSM9DS0_ACCELDATARATE_POWERDOWN = 0b0000 << 4
LSM9DS0_ACCELDATARATE_3_125HZ = 0b0001 << 4
LSM9DS0_ACCELDATARATE_6_25HZ = 0b0010 << 4
LSM9DS0_ACCELDATARATE_12_5HZ = 0b0011 << 4
LSM9DS0_ACCELDATARATE_25HZ = 0b0100 << 4
LSM9DS0_ACCELDATARATE_50HZ = 0b0101 << 4
LSM9DS0_ACCELDATARATE_100HZ = 0b0110 << 4
LSM9DS0_ACCELDATARATE_200HZ = 0b0111 << 4
LSM9DS0_ACCELDATARATE_400HZ = 0b1000 << 4
LSM9DS0_ACCELDATARATE_800HZ = 0b1001 << 4
LSM9DS0_ACCELDATARATE_1600HZ = 0b1010 << 4

LSM9DS0_MAGGAIN_2GAUSS = 0b00 << 5
LSM9DS0_MAGGAIN_4GAUSS = 0b01 << 5
LSM9DS0_MAGGAIN_8GAUSS = 0b10 << 5
LSM9DS0_MAGGAIN_12GAUSS = 0b11 << 5

LSM9DS0_MAGDATARATE_3_125HZ = 0b000 << 2
LSM9DS0_MAGDATARATE_6_25HZ = 0b001 << 2
LSM9DS0_MAGDATARATE_12_5HZ = 0b010 << 2
LSM9DS0_MAGDATARATE_25HZ = 0b011 << 2
LSM9DS0_MAGDATARATE_50HZ = 0b100 << 2
LSM9DS0_MAGDATARATE_100HZ = 0b101 << 2

LSM9DS0_GYROSCALE_245DPS = 0b00 << 4
LSM9DS0_GYROSCALE_500DPS = 0b01 << 4
LSM9DS0_GYROSCALE_2000DPS = 0b10 << 4


class LSM9DS0(object):
    # Debug set to false for the moment. Change to find bugs
    def __init__(self, accel_address=LSM9DS0_ACCEL_ADDRESS,
                 mag_address=LSM9DS0_MAG_ADDRESS, gyro_address=LSM9DS0_GYRO_ADDRESS,
                 I2C=None, busnum=None):...

    def readLowHigh(self, i2c_device, lowhigh):...

    def readSensor(self, i2c_device, xyz_lh):...

    def readGyro(self):...

    def readMag(self):...

    def readAccel(self):...

    def read(self):...

    # The documentation on reading temperature is not very clear, and it appears
    # that the sensor does not provide an ambient temperature reading, with no
    # absolute value, instead measuring change in temp inside the chip
    def rawTemp(self):...


imu = LSM9DS0()
print("Printing gyroscope, magnetometer and accelerometer values... Press Ctrl-C to quit.")
while True:
    # Grab (x, y, z) readings for gyro, mag and accelerometer
    gyro, mag, accel = imu.read()

    # Unpack tuples
    gyro_x, gyro_y, gyro_z = gyro
    mag_x, mag_y, mag_z = mag
    acc_x, acc_y, acc_z = accel

    # Print sets of values in separate lines
    print("Gyro: x = {}\t\ty = {}\t\tz = {}".format(gyro_x, gyro_y, gyro_z))
    print("Acce: x = {}\t\ty = {}\t\tz = {}".format(acc_x, acc_y, acc_z))
    print("Magn: x = {}\t\ty = {}\t\tz = {}".format(mag_x, mag_y, mag_z))
    # Normalising the accelerometer data
    # Dividing variable (don't know why I use this)
    acc_norm_div = math.sqrt(acc_x ** 2 + acc_y ** 2 + acc_z ** 2)

    # Normalised values
    acc_x_norm = acc_x / acc_norm_div
    acc_y_norm = acc_y / acc_norm_div

    # Calc pitch and roll using trig
    pitch = math.asin(acc_x_norm)
    roll = - math.asin(math.radians(acc_y_norm / math.cos(pitch)))

    # Do some mathsy stuff to compensate for the tilt
    mag_x_comp = mag_x * math.cos(pitch) + mag_z * math.sin(pitch)
    mag_y_comp = mag_x * math.sin(roll) * math.sin(pitch) + mag_y * math.cos(roll) \
                 - mag_z * math.sin(roll) * math.cos(pitch)

    # Calculate the angle in degrees
    angle_deg = math.degrees(math.atan2(mag_y_comp, mag_x_comp))
    h = math.degrees(math.atan2(mag_y, mag_x))

    # Work out as a bearing
    if angle_deg < 0:
        angle_deg += 360
    if h < 0:
        h += 360
    print("Pitch = {}\tRoll = {}\tHeading = {}\tNo Comp H = {}".format(pitch, roll,
                                                                       angle_deg, h))
    print("\n")

    raw_input("Presiona tecla para la siguiente medición")
```

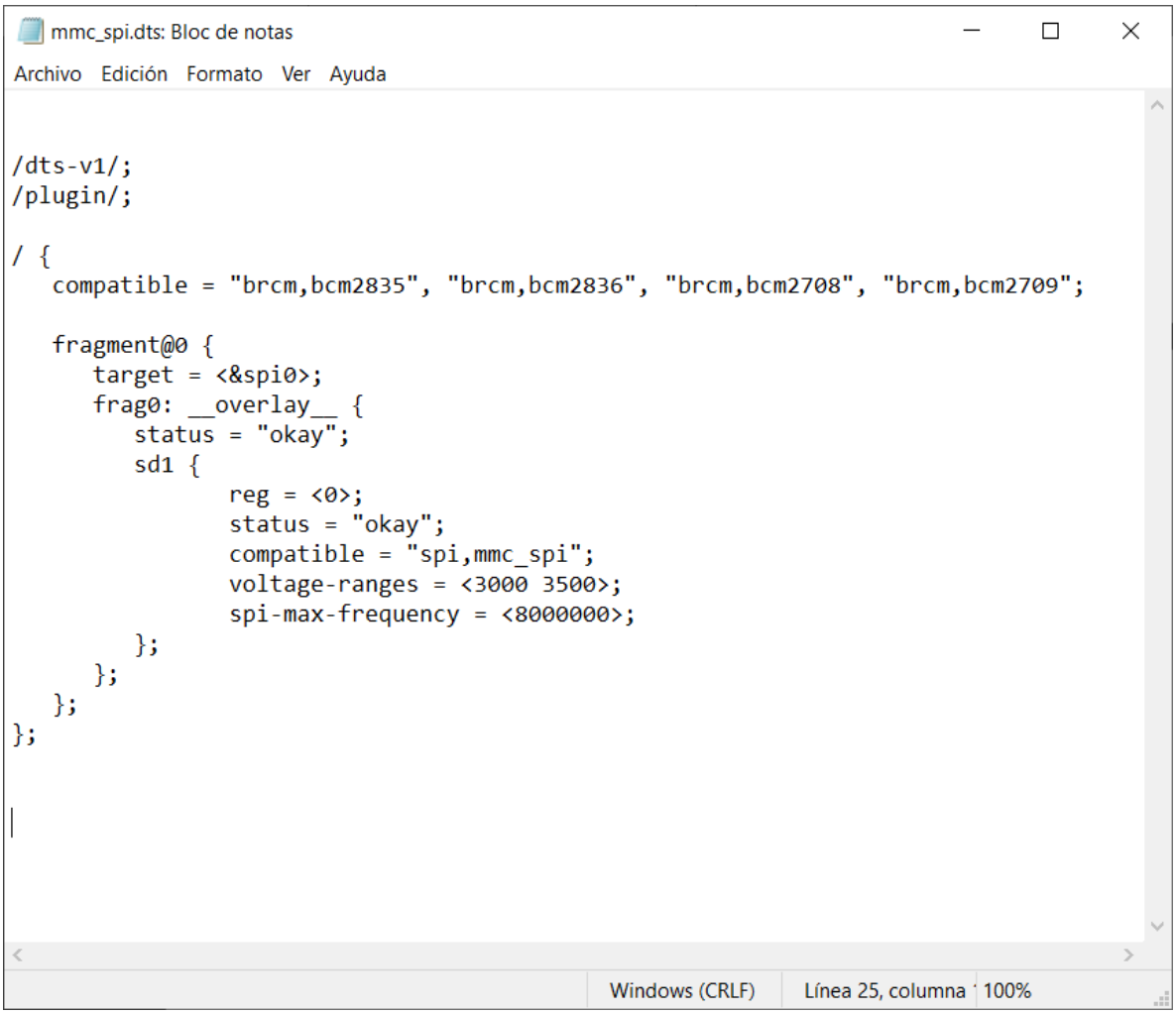
B.5 código de prueba del GPS

```python
# coding=utf-8
import serial

# El puerto serial de la Raspberry Pi 3 al que está conectado el GPS
port = "/dev/ttyS0"

# Objeto serial del GPS
gps = serial.Serial(port, baudrate = 9600, timeout = 0.5)

data = gps.readline()

print(data)

gps.close()

```

B.6 código para desactivar los comandos NMEA

```python
# coding=utf-8
import serial
import time

# El puerto serial de la Raspberry Pi 3 al que está conectado el GPS
port = "/dev/ttyS0"
delay = 0.1  # Para que tenga un delay entre escrituras

# Objeto serial del GPS
gps = serial.Serial(port, baudrate = 9600, timeout = 0.5)

print("gps port: {}".format(gps.name))

# Para deshabilitar: RMC,VTG,GGA,GSA,GSV,TXT
gps.write(b'$PUBX,40,RMC,0,0,0,0,0,0*47\r\n')
time.sleep(delay)
gps.write(b'$PUBX,40,VTG,0,0,0,0,0,0*5E\r\n')
time.sleep(delay)
gps.write(b'$PUBX,40,GGA,0,0,0,0,0,0*5A\r\n')
time.sleep(delay)
gps.write(b'$PUBX,40,GSA,0,0,0,0,0,0*4E\r\n')
time.sleep(delay)
gps.write(b'$PUBX,40,GSV,0,0,0,0,0,0*59\r\n')
time.sleep(delay)
#gps.write(b'$PUBX,40,TXT,0,0,0,0,0,0*43\r\n')
#time.sleep(delay)

#data = gps.readline()

#print(data)

gps.close()
print("gps open: {}".format(gps.is_open))
```

B.7 archivo para crear el árbol de dispositivos



Para más información acerca de cómo se hizo funcionar una segunda memoria microSD visitar la siguiente página:
https://ralimtek.com/raspberry%20pi/electronics/software/raspberry_pi_secondary_sd_card/

B.8 conjunto de scripts para montar y desmontar la microSD al presionar un botón

Script de Shell: listen-for-mount.sh

```
#! /bin/sh

### BEGIN INIT INFO
# Provides:          listen-for-mount.py
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
### END INIT INFO

# If you want a command to always run, put it here

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Starting listen-for-mount.py"
    /usr/local/bin/listen-for-mount.py &
    ;;
  stop)
    echo "Stopping listen-for-mount.py"
    pkill -f /usr/local/bin/listen-for-mount.py
    ;;
  *)
    echo "Usage: /etc/init.d/listen-for-mount.sh {start|stop}"
    exit 1
    ;;
esac

exit 0
```

Script de Python: listen-for-mount.py

```
#!/usr/bin/env python

import RPi.GPIO as GPIO
import subprocess
import time

script="/home/pi/Scripts/./umsd.sh"
ngpio = 23
t = 0.5

GPIO.setmode(GPIO.BCM)
GPIO.setup(ngpio, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    # Detiene la ejecucion del programa hasta que detecte un cambio de flanco a tierra
    GPIO.wait_for_edge(ngpio, GPIO.FALLING)
    #GPIO.wait_for_edge(ngpio, GPIO.RISING)
    time.sleep(t)
    subprocess.call([script], shell=False)

```

Script de Shell: umsd.sh

```
#!/bin/sh
# Versión de prueba del script que se encargaría de montar o desmontar la SD secundaria
MMCBKP="/home/pi/mmcblk2-backup.txt"
MNTPNT="/home/pi/media/microSD2"
MNTDEV="/dev/mmcblk2p1"
DEVICE="/dev/mmcblk2"
LOGDIR="/home/pi/Logs"
LGNAME="/testRover.log"
RETURN=0
PYSCRP="/home/pi/Scripts/turn-LED.py"
if [ $(sudo mount | grep -c $MNTPNT) != 1 ]; then  #No está montada la SD
  #echo "Montando la SD"
  if [ $(sudo parted -l | grep -c "unknown") -ne 0 ]; then  #No se puede montar normal
    #echo "Recuperando la tabla de particiones"
    sudo sfdisk -q $DEVICE < $MMCBKP
    if [ $? != 0 ]; then
      RETURN=1
    fi
  fi
  sudo mount -t vfat $MNTDEV
  if [ $? != 0 ]; then
    RETURN=1
  else
    #echo "Debug: Apagando LED de SD desmontada"
    python $PYSCRP 0
  fi
else  #Está montada la SD
  #echo "Desmontando la SD"
  sudo cp "${LOGDIR}$LGNAME" "${MNTPNT}$LGNAME"
  if [ $? != 0 ]; then
    RETURN=1
  fi
  sudo umount -t vfat $MNTDEV
  if [ $? != 0 ]; then
    RETURN=1
  else
    #echo "Debug: Prendiendo LED de SD desmontada"
    python $PYSCRP 1
  fi
fi
if [ $RETURN != 0 ];  then  #RETURN es 1, hubo un error
  echo "Hubo algún error"
fi
```

Script de Python: turn-LED.py

```
import RPi.GPIO as GPIO
import sys

gpioled = 25
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(gpioled, GPIO.OUT)
a = sys.argv[1]

if (a == "1" ):
    GPIO.output(gpioled, GPIO.HIGH)
else:
    GPIO.output(gpioled, GPIO.LOW)
```

B.9 scripts para apagar la Raspberry cuando se presiona un botón

Script de Shell: listen-for-shutdown.sh

```
#! /bin/sh

### BEGIN INIT INFO
# Provides:          listen-for-shutdown.py
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
### END INIT INFO

# If you want a command to always run, put it here

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Starting listen-for-shutdown.py"
    /usr/local/bin/listen-for-shutdown.py &
    ;;
  stop)
    echo "Stopping listen-for-shutdown.py"
    pkill -f /usr/local/bin/listen-for-shutdown.py
    ;;
  *)
    echo "Usage: /etc/init.d/listen-for-shutdown.sh {start|stop}"
    exit 1
    ;;
esac

exit 0
```

Script de Python: listen-for- shutdown.py

```
#!/usr/bin/env python


import RPi.GPIO as GPIO
import subprocess


GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.wait_for_edge(18, GPIO.FALLING)

subprocess.call(['shutdown', '-h', 'now'], shell=False)
```

B.10 código para enviar y recibir mensajes con el módulo XBee

```
# coding=utf-8
import serial
import time

# Envía 10 veces un mensaje, checa si hay algo que leer y lo imprime

port = "/dev/ttyS0"  # Puerto serial del Xbee

# Serial object
xbee = serial.Serial(port, baudrate=9600, timeout=0.5)
msj = "Hola Estación terrena "
for x in range(10):
    xbee.write(msj + str(x + 1) + '\n')
    if xbee.in_waiting > 0:
        data = xbee.read(xbee.in_waiting)
        print data
    time.sleep(2)

xbee.close()

```

B.11 conjunto de códigos de prueba para usar el GPS con una UART virtual

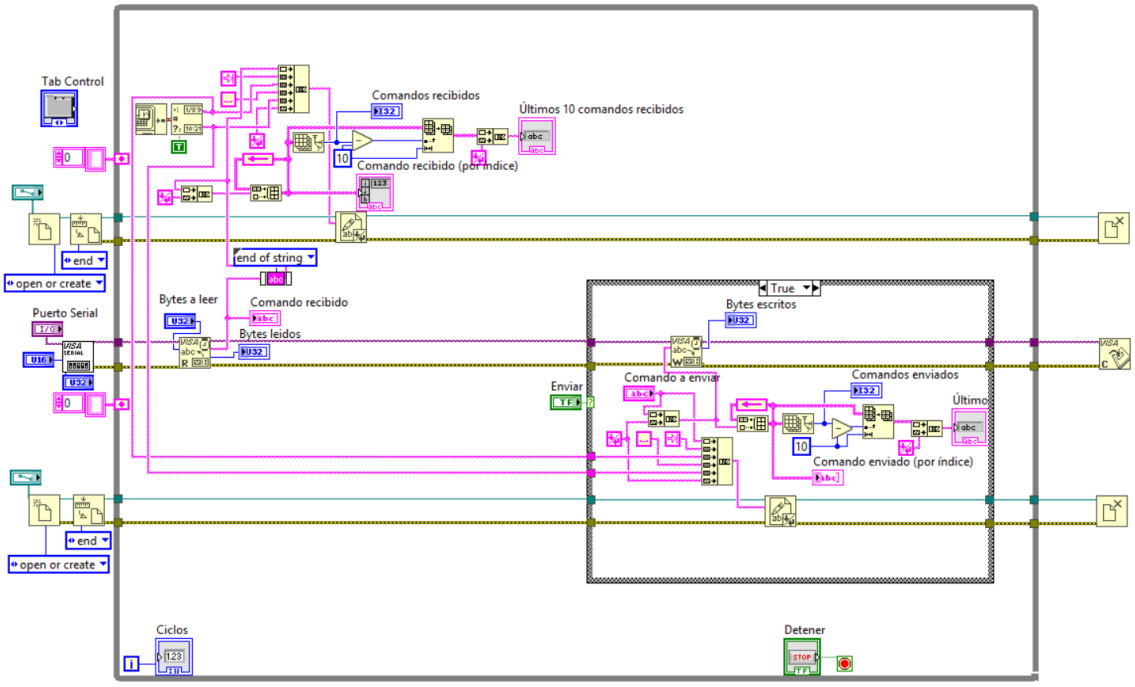
Este es el código de prueba equivalente al de la sección *B.5*

```
# coding=utf-8
import time
import pigpio

RX=27
TX=17

try:
    pi = pigpio.pi()
    pi.set_mode(RX, pigpio.INPUT)
    pi.set_mode(TX, pigpio.OUTPUT)
    pi.bb_serial_read_open(RX, 9600, 8)
    print "DATA - SOFTWARE SERIAL:"
    while True:
        (count, data) = pi.bb_serial_read(RX)
        if count:
            # Transforma de bytearray a str
            msj = ''.join(chr(i) for i in data)
            # se pasa msj[:-2] porque tiene '\r\n' al final
            print("type = {} msj = {}".format(type(msj), msj[:-2]))
        time.sleep(0.5)

except KeyboardInterrupt:
    pi.bb_serial_read_close(RX)
    pi.stop()
```

Este es el código de configuración de los mensajes NMEA equivalente al de la sección *B.6*

```
# coding=utf-8
import time
import pigpio

RX=27
TX=17

baud = 9600
bits = 8

try:
    pi = pigpio.pi()
    # GPIO 27 como salida
    pi.set_mode(TX, pigpio.OUTPUT)
    # Se apagan las exepciones para cerrar puerto serial
    # aunque no esté abierto el puerto
    pigpio.exceptions = False
    pi.bb_serial_read_close(RX)
    # Se vuelven a encender las excepciones
    pigpio.exceptions = True
    # GPIO 27 como entrada
    pi.set_mode(RX, pigpio.INPUT)
    pi.bb_serial_read_open(RX, baud, bits)
    # Crea una "waveform" representando la información
    pi.wave_clear()
    delay = 0.1
    config_cmds = ["$PUBX,40,RMC,0,0,0,0,0,0*47\r\n",
                   "$PUBX,40,VTG,0,0,0,0,0,0*5E\r\n",
                   "$PUBX,40,GGA,0,0,0,0,0,0*5A\r\n",
                   "$PUBX,40,GSA,0,0,0,0,0,0*4E\r\n",
                   "$PUBX,40,GSV,0,0,0,0,0,0*59\r\n",
                   "$PUBX,40,RMC,0,0,0,0,0,0*47\r\n"]
    for m in config_cmds:
        c_cmd = [i for i in m]
        pi.wave_add_serial(TX, baud, c_cmd, bb_bits=bits)
        wid=pi.wave_create()
        pi.wave_send_once(wid)    # transmit serial data
        pi.wave_delete(wid)
        while pi.wave_tx_busy(): # wait until all data sent
            pass
        time.sleep(delay)

    print "DATA - SOFTWARE SERIAL:"

    while True:
        (count, data) = pi.bb_serial_read(RX)
        if count:
            print("count = {} data = {}".format(count, data))
        time.sleep(1)

except KeyboardInterrupt:
    pi.bb_serial_read_close(RX)
    pi.stop()
```

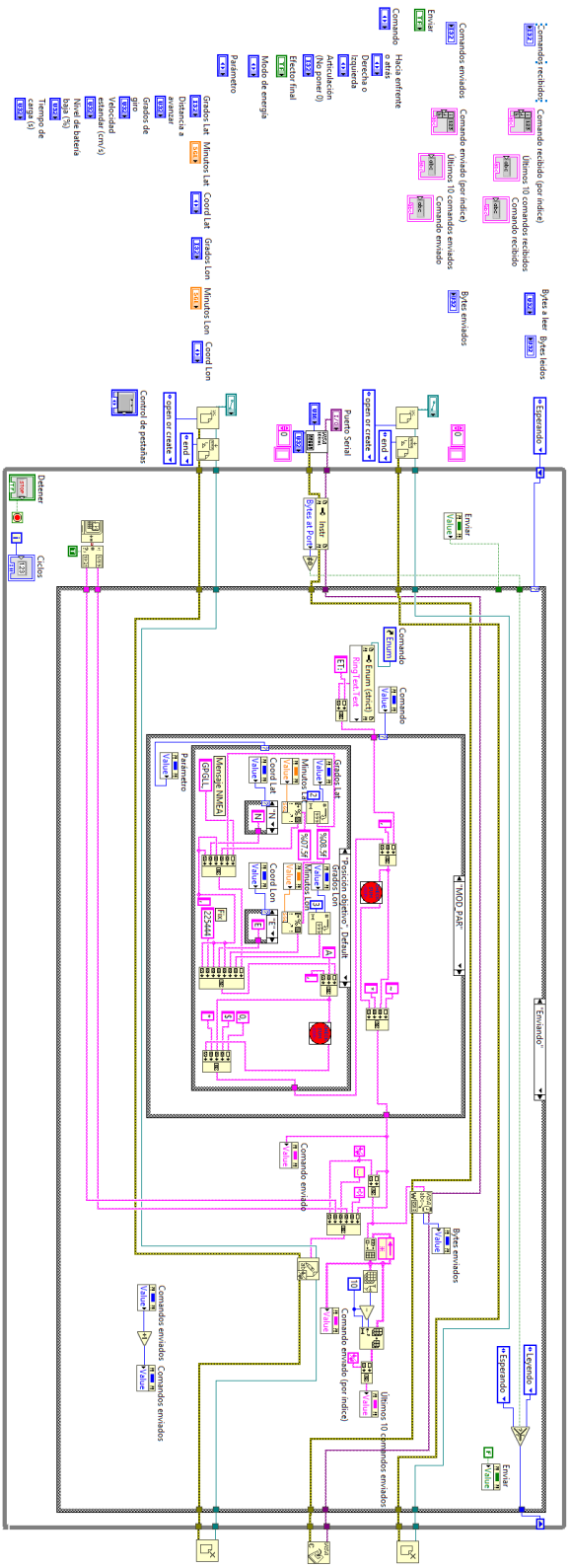
B.12 primera versión de la estación terrena



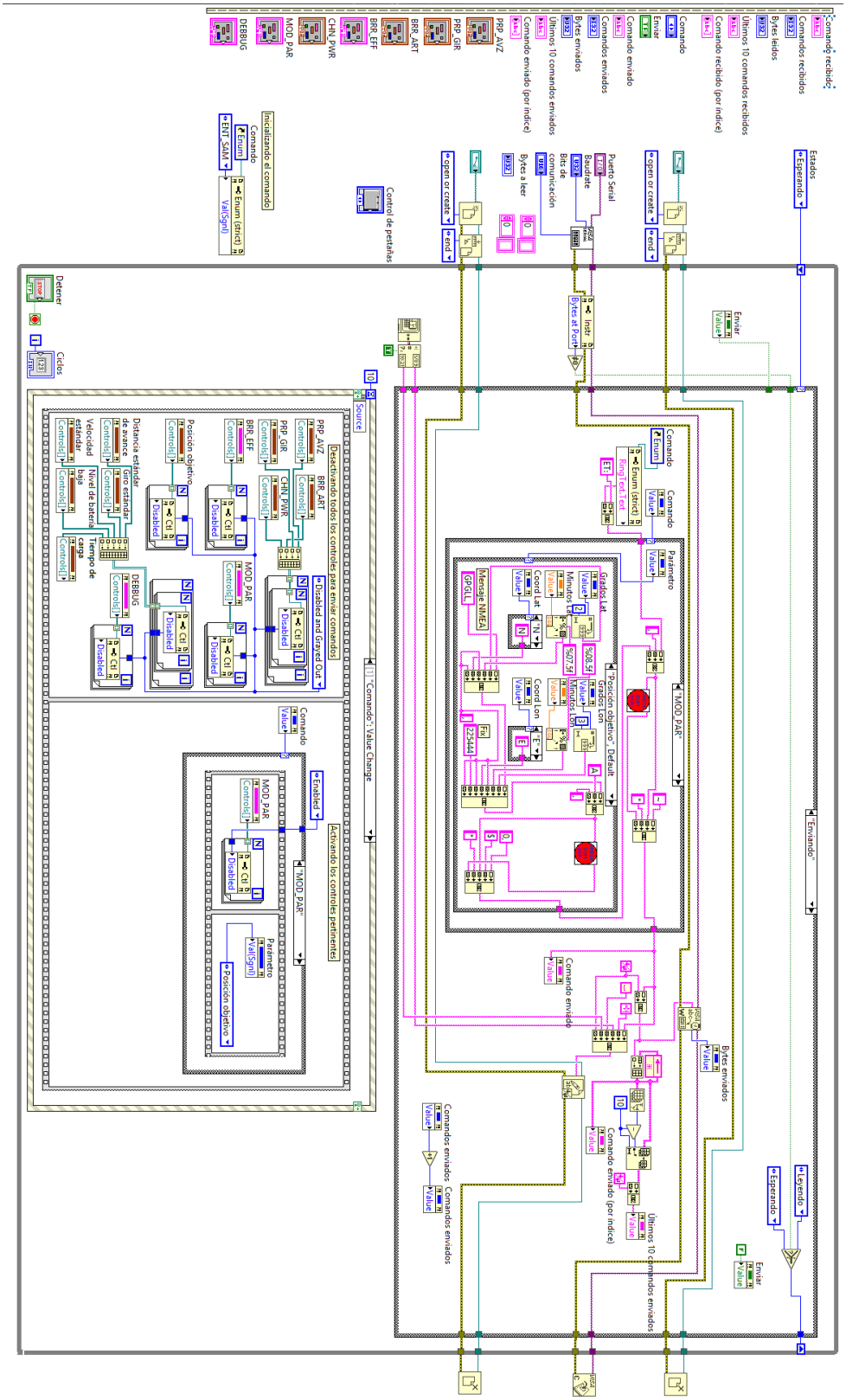
B.13 código del logger

```
# coding=utf-8

import logging  # Para loggear

# Configurando el logger
LOG_FILE = "/home/pi/Logs/testRover.log"
logger = logging.getLogger("PruebaRover")
handler = logging.FileHandler(LOG_FILE)
formatter = logging.Formatter("%(asctime)s %(levelname)-7s %(message)s")
handler.setFormatter(formatter)
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)

logger.info("Un mensaje de Log de nivel informativo")
logger.warning("Un mensaje de Log de nivel alerta")
logger.error("Un mensaje de Log de nivel error")
logger.debug("Un mensaje de Log de nivel debbug")
```

B.14 mejora a la interfaz de la estación terrena

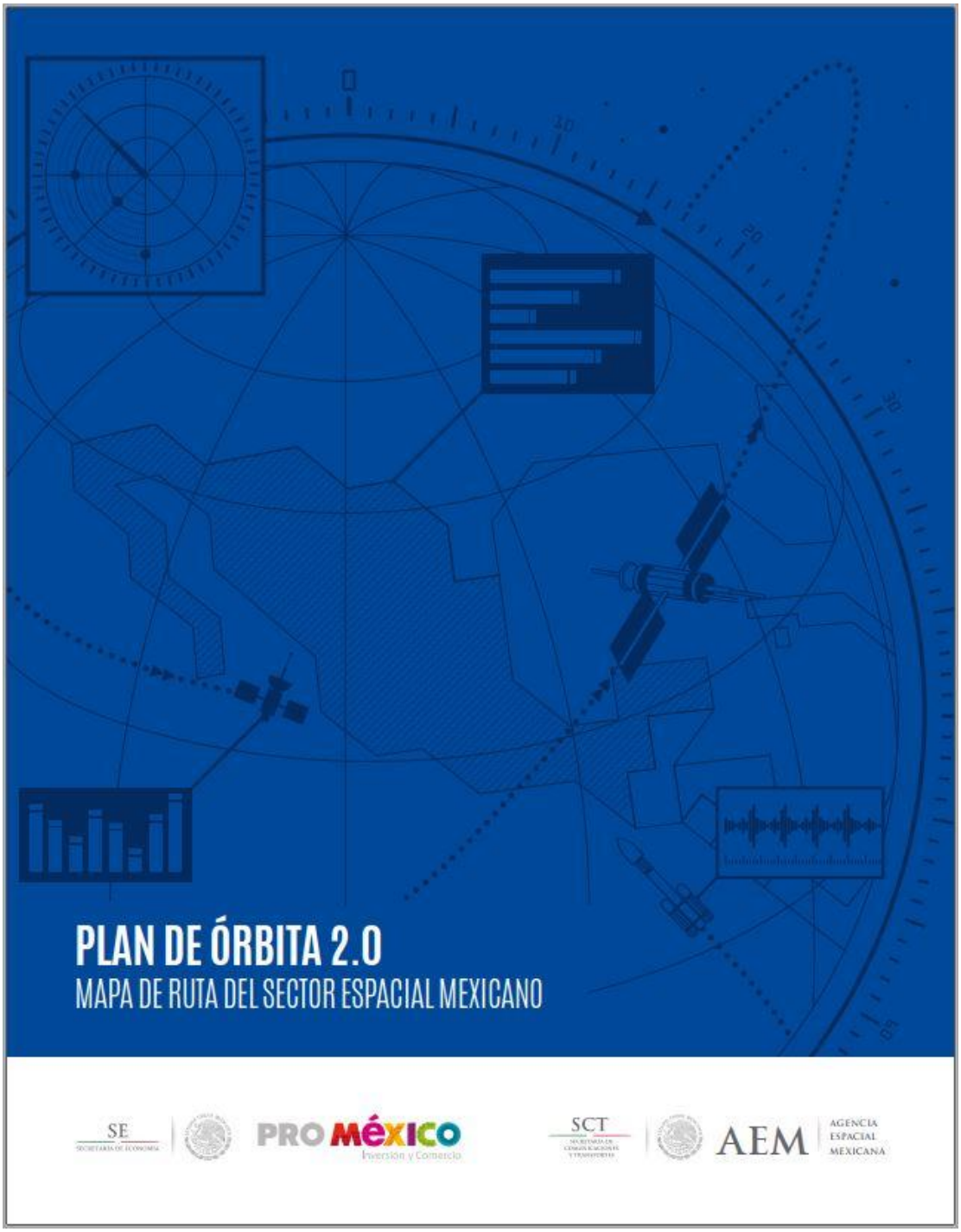


B.15 versión final de la interfaz



C Documentos importantes mencionados

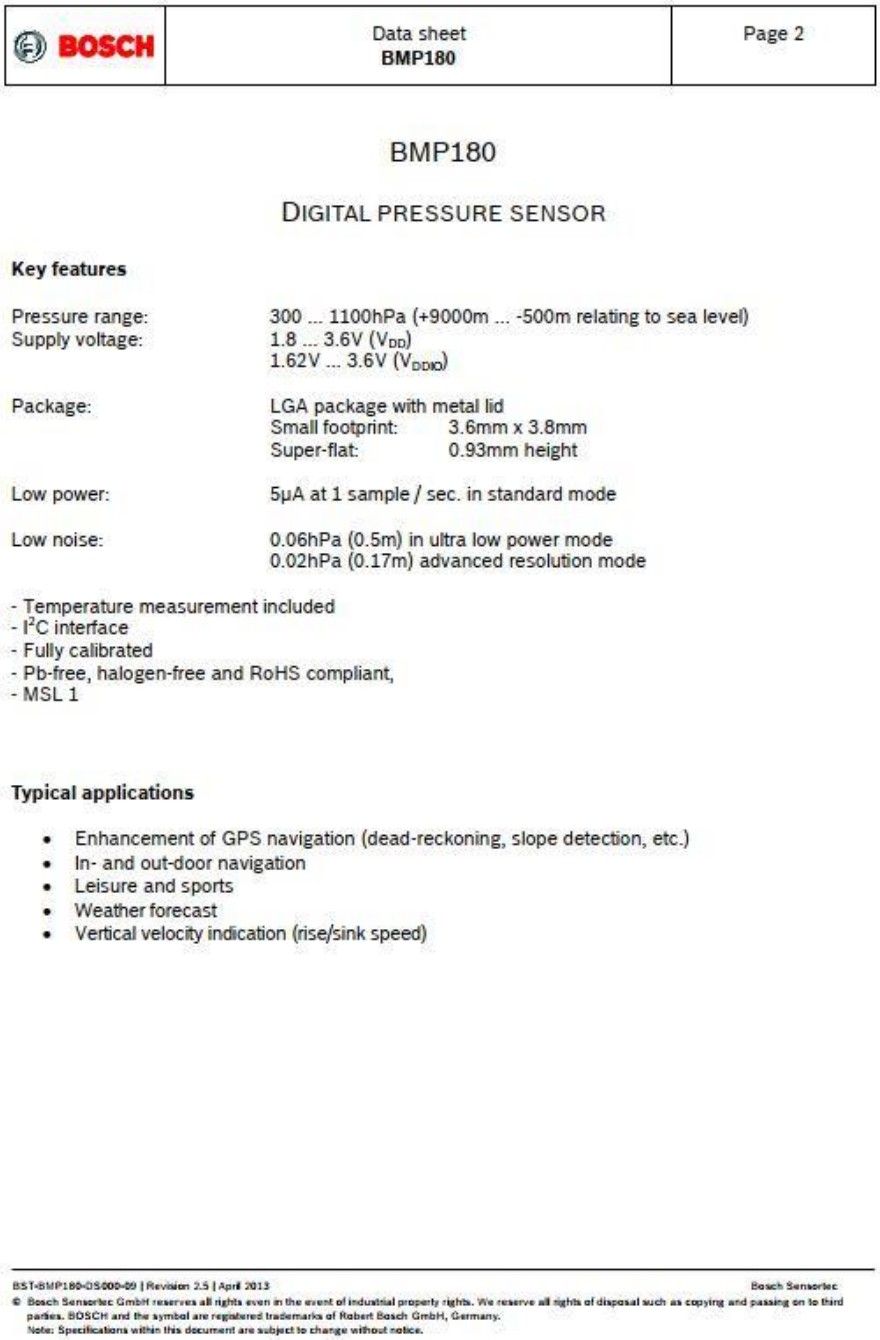
C.1 Plan de Órbita 2.0



La versión completa se puede encontrar en:
http://www.promexico.gob.mx/documentos/biblioteca/plan-orbita.pdf

Información relevante se puede encontrar en: https://www.gob.mx/aem

C.2 BMP180 (Sensor de presión y temperatura)



BOSCH	Data sheet **BMP180**	Page 2

BMP180

DIGITAL PRESSURE SENSOR

Key features

Pressure range:	300 ... 1100hPa (+9000m ... -500m relating to sea level)
Supply voltage:	1.8 ... 3.6V (V_{DD}) 1.62V ... 3.6V (V_{DDIO})
Package:	LGA package with metal lid Small footprint: 3.6mm x 3.8mm Super-flat: 0.93mm height
Low power:	5µA at 1 sample / sec. in standard mode
Low noise:	0.06hPa (0.5m) in ultra low power mode 0.02hPa (0.17m) advanced resolution mode

- Temperature measurement included
- I^2C interface
- Fully calibrated
- Pb-free, halogen-free and RoHS compliant,
- MSL 1

Typical applications

- Enhancement of GPS navigation (dead-reckoning, slope detection, etc.)
- In- and out-door navigation
- Leisure and sports
- Weather forecast
- Vertical velocity indication (rise/sink speed)

La versión completa se puede encontrar en: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP180-DS000.pdf

Documentación de apoyo se puede encontrar en: https://www.bosch-sensortec.com/bst/products/all_products/bmp180

C.3 LSM9DS0 (Acelerómetro, Giroscopio y Magnetómetro)

life.augmented

LSM9DS0

iNEMO inertial module:
3D accelerometer, 3D gyroscope, 3D magnetometer

Datasheet - production data

LGA-24 (4x4x1.0 mm)

Applications

- Indoor navigation
- Smart user interfaces
- Advanced gesture recognition
- Gaming and virtual reality input devices
- Display/map orientation and browsing

Features

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
- ±2/±4/±6/±8/±16 *g* linear acceleration full scale
- ±2/±4/±8/±12 gauss magnetic full scale
- ±245/±500/±2000 dps angular rate full scale
- 16-bit data output
- SPI / I^2C serial interfaces
- Analog supply voltage 2.4 V to 3.6 V
- Power-down mode / low-power mode
- Programmable interrupt generators
- Embedded self-test
- Embedded temperature sensor
- Embedded FIFO
- Position and motion detection functions
- Click/double-click recognition
- Intelligent power saving for handheld devices
- ECOPACK®, RoHS and "Green" compliant

Description

The LSM9DS0 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.

The LSM9DS0 has a linear acceleration full scale of ±2*g*/±4*g*/±6*g*/±8*g*/±16*g*, a magnetic field full scale of ±2/±4/±8/±12 gauss and an angular rate of ±245/±500/±2000 dps.

The LSM9DS0 includes an I^2C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.

The system can be configured to generate interrupt signals on dedicated pins and is capable of motion and magnetic field detection. Thresholds and timing of interrupt generators are programmable by the end user.

Magnetic, accelerometer and gyroscope sensing can be enabled or set in power-down mode separately for smart power management.

The LSM9DS0 is available in a plastic land grid array package (LGA) and it is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

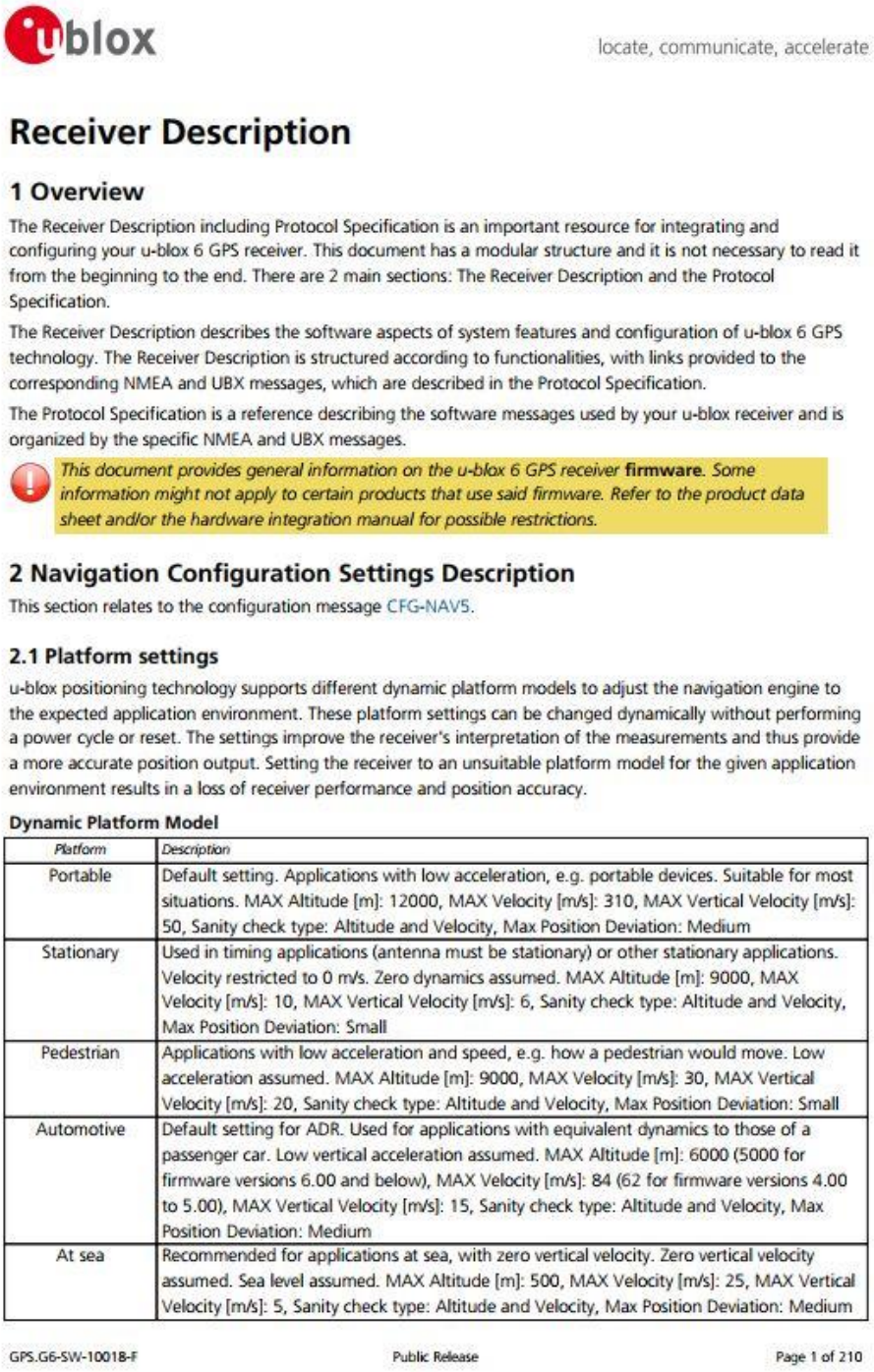
Table 1. Device summary

Part number	Temperature range [°C]	Package	Packing
LSM9DS0	-40 to +85	LGA-24	Tray
LSM9DS0TR	-40 to +85	LGA-24	Tape and reel

La versión completa se puede encontrar en: https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf

Documentación de apoyo se puede encontrar en: https://learn.adafruit.com/adafruit-lsm9ds0-accelerometer-gyro-magnetometer-9-dof-breakouts

C.4 GPS6MV2 (GPS)



u-blox

locate, communicate, accelerate

Receiver Description

1 Overview

The Receiver Description including Protocol Specification is an important resource for integrating and configuring your u-blox 6 GPS receiver. This document has a modular structure and it is not necessary to read it from the beginning to the end. There are 2 main sections: The Receiver Description and the Protocol Specification.

The Receiver Description describes the software aspects of system features and configuration of u-blox 6 GPS technology. The Receiver Description is structured according to functionalities, with links provided to the corresponding NMEA and UBX messages, which are described in the Protocol Specification.

The Protocol Specification is a reference describing the software messages used by your u-blox receiver and is organized by the specific NMEA and UBX messages.

This document provides general information on the u-blox 6 GPS receiver **firmware**. *Some information might not apply to certain products that use said firmware. Refer to the product data sheet and/or the hardware integration manual for possible restrictions.*

2 Navigation Configuration Settings Description

This section relates to the configuration message CFG-NAV5.

2.1 Platform settings

u-blox positioning technology supports different dynamic platform models to adjust the navigation engine to the expected application environment. These platform settings can be changed dynamically without performing a power cycle or reset. The settings improve the receiver's interpretation of the measurements and thus provide a more accurate position output. Setting the receiver to an unsuitable platform model for the given application environment results in a loss of receiver performance and position accuracy.

Dynamic Platform Model

Platform	*Description*
Portable	Default setting. Applications with low acceleration, e.g. portable devices. Suitable for most situations. MAX Altitude [m]: 12000, MAX Velocity [m/s]: 310, MAX Vertical Velocity [m/s]: 50, Sanity check type: Altitude and Velocity, Max Position Deviation: Medium
Stationary	Used in timing applications (antenna must be stationary) or other stationary applications. Velocity restricted to 0 m/s. Zero dynamics assumed. MAX Altitude [m]: 9000, MAX Velocity [m/s]: 10, MAX Vertical Velocity [m/s]: 6, Sanity check type: Altitude and Velocity, Max Position Deviation: Small
Pedestrian	Applications with low acceleration and speed, e.g. how a pedestrian would move. Low acceleration assumed. MAX Altitude [m]: 9000, MAX Velocity [m/s]: 30, MAX Vertical Velocity [m/s]: 20, Sanity check type: Altitude and Velocity, Max Position Deviation: Small
Automotive	Default setting for ADR. Used for applications with equivalent dynamics to those of a passenger car. Low vertical acceleration assumed. MAX Altitude [m]: 6000 (5000 for firmware versions 6.00 and below), MAX Velocity [m/s]: 84 (62 for firmware versions 4.00 to 5.00), MAX Vertical Velocity [m/s]: 15, Sanity check type: Altitude and Velocity, Max Position Deviation: Medium
At sea	Recommended for applications at sea, with zero vertical velocity. Zero vertical velocity assumed. Sea level assumed. MAX Altitude [m]: 500, MAX Velocity [m/s]: 25, MAX Vertical Velocity [m/s]: 5, Sanity check type: Altitude and Velocity, Max Position Deviation: Medium

GPS.G6-SW-10018-F Public Release Page 1 of 210

La versión completa se puede encontrar en: https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf

Documentación de apoyo se puede encontrar en: https://www.xarg.org/2016/06/neo6mv2-gps-module-with-raspberry-pi/

C.5 XBee S1 802.15.4

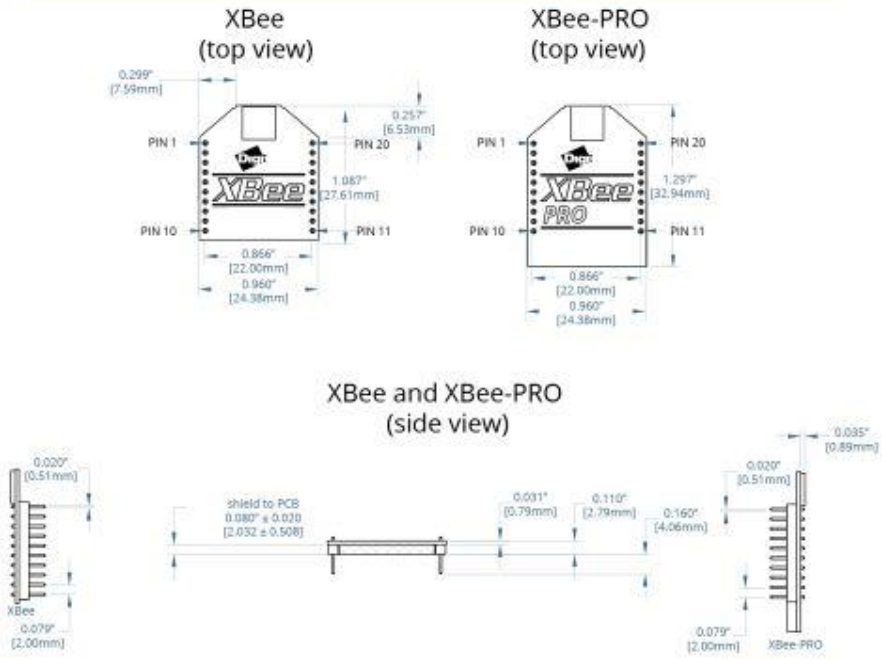
Antenna options

The ranges specified are typical for the integrated whip (1.5 dBi) and dipole (2.1 dBi) antennas. The printed circuit board (PCB) antenna option provides advantages in its form factor; however, it typically yields shorter range than the whip and dipole antenna options when transmitting outdoors. For more information, see XBee and XBee-PRO OEM RF Module Antenna Considerations Application Note.

XBee/XBee-PRO S1 802.15.4 (Legacy) Mechanical drawings

The following graphics show the mechanical drawings of the XBee / XBee-PRO OEM RF Modules. The XBee and XBee-PRO RF Modules are pin-for-pin compatible.

Note The antenna options not shown.



Mounting considerations

We design the through-hole module to mount into a receptacle so that you do not have to solder the module when you mount it to a board. The development kits may contain RS-232 and USB interface boards that use two 20-pin receptacles to receive modules.

The following illustration shows the module mounting into the receptacle on the RS-232 interface board.

La versión completa se puede encontrar en:
https://www.digi.com/resources/documentation/digidocs/PDFs/90000982.pdf

Otros temas de apoyo se pueden encontrar en:
https://www.digi.com/resources/documentation/digidocs/pdfs/90001456-13.pdf