UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño electrónico y arquitectura de programación del robot FinDER v3

TESIS

Que para obtener el título de

Ingeniero mecatrónico

P R E S E N T A

Mauro Alberto Rivero Espíndola

DIRECTOR DE TESIS

M.I. Yukihiro Minami Koyama

Ciudad Universitaria, Cd. Mx., 2019

Investigación realizada gracias al proyecto
UNAM-DGAPA-PAPIIT IT102518
“Robots móviles para el reconocimiento
e inspección de zonas con movilidad restringida”

CONTENIDO

Capítulo 1
Introducción

La presente tesis trata de proponer las herramientas necesarias para que una persona logre implementar una arquitectura de software para controlar un robot de manera teleoperada en un entorno de desastre, el cual puede ser simulado o real; en ambos casos, se tienen limitantes relacionadas principalmente a que, en un sitio de desastre, no se tiene control alguno sobre las condiciones del medio en que opera el robot como el terreno, cambios de temperatura, fugas de gas, líquidos inflamables, charcos de agua, presencia de lodo, arena o grava, deslizamientos, e incendios. Por ello, cuando se diseña un robot de rescate, este debe contar con robustez mecánica y en sus de programas de control, además de mecanismos que aseguren la integridad física del robot y del entorno.

Se listan los elementos utilizados en la capa de hardware y de software, es decir, los sensores, microcontroladores y algunos entornos de desarrollo de programación cómo Code Composer Studio, Energia y ROS. Se explica cómo conectar elementos de hardware como sensores y actuadores a un microcontrolador y de ahí a una computadora; y se describen los modelos y algunas prácticas de programación útiles para la implementación del control de un robot teleoperado en un entorno de desastre.

1.2 Antecedentes

La definición de robot es compleja debido a que la robótica, rama encargada de su estudio, es una rama de conocimiento en crecimiento y que cada día incluye nuevos campos de conocimiento en su desarrollo como ciencia **[1].**

Normalmente, al hablar de un robot, se debe acotar el tema de estudio, por ejemplo, al decir robot manipulador industrial, robot móvil o robot de exploración submarina, se está hablando de robots, pero además se hace uso de adjetivos que indican el caso de uso y especificaciones de cada robot.

Por tanto, conviene tener una definición que explique el término robot. Un robot se refiere a

una máquina o dispositivo móvil que cuenta con cierto grado de autonomía y que esté bajo control directo de una computadora.

El desarrollo de robots ha avanzado de manera constante de acuerdo a la definición anterior, y va en aumento desde la década de los sesenta con proyectos de investigación que dieron origen a robots móviles capaces de resolver tareas de planeación de rutas, control de sistemas remotos, sistemas expertos, entre otras. Uno de los robots que marcó la pauta en la forma en que se desarrolla la construcción de robots móviles, fue “Shakey” el cual surgió a partir de un proyecto para generar un autómata capaz de realizar misiones no triviales en un entorno real; el control de dicho autómata o máquina era mediante una computadora. Dicho autómata tenía la capacidad de sensado visual y táctil. **[2]**

El robot Shakey mostrado en la Figura 1.1, consta de una cámara de televisión capturando imágenes de su entorno, un sensor de telemetría láser para determinar la distancia a las paredes u otros objetos, y pulsadores (interruptores mecánicos) para detectar colisiones. Es oportuno también comentar que el entorno en que se experimentó con este robot fue controlado de principio a fin, y que el objetivo más allá de la construcción del robot, fue generar nuevas técnicas para la planeación y ejecución de acciones en un entorno real; lo que surgió fueron técnicas e ideas nuevas en temas de control robusto, visión artificial, aprendizaje y planeación; muchas de ellas aún se siguen utilizando.

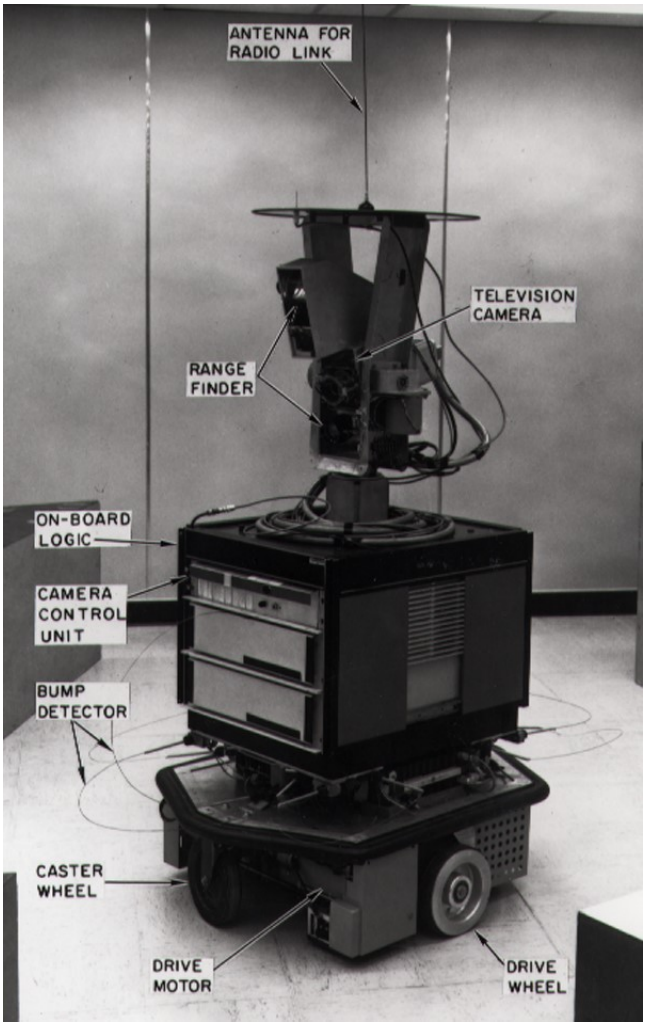

Figura 1.1 Fotografía del robot Shakey.

Una de esas ideas que se siguen utilizando es la forma en que organizaron los programas de control del robot. El robot Shakey constaba de una computadora principal DEC PDP-10. Entre la computadora y el vehículo móvil había una computadora de control de periféricos PDP-15 encargada de manejar las comunicaciones a bajo nivel con el hardware del robot; adicionalmente, un enlace de radio y de video. Los programas de la computadora PDP-10 estaban organizados en una jerarquía de tres capas. Los programas en el bajo nivel controlaban todos los motores y recibían toda la información sensorial. los programas en la capa intermedia supervisaban acciones primitivas, como mover el robot de una coordenada específica, o procesar la señal de video proveniente de la cámara. La planeación de acciones más complejas que requieren varias acciones primitivas, como ir de un cuarto a otro, estaban en la capa superior.

Esta jerarquía de tres capas mostrada en la Figura 1.2 es extensamente utilizada por su flexibilidad para el desarrollo de robots para la investigación en temas de planeación, razonamiento o aprendizaje en las capas de alto nivel, con sus correspondientes acciones primitivas en el nivel intermedio, dejando aparte la implementación de la comunicación con el hardware en la capa de bajo nivel para el control de los actuadores y la adquisición de información del entorno mediante los sensores.

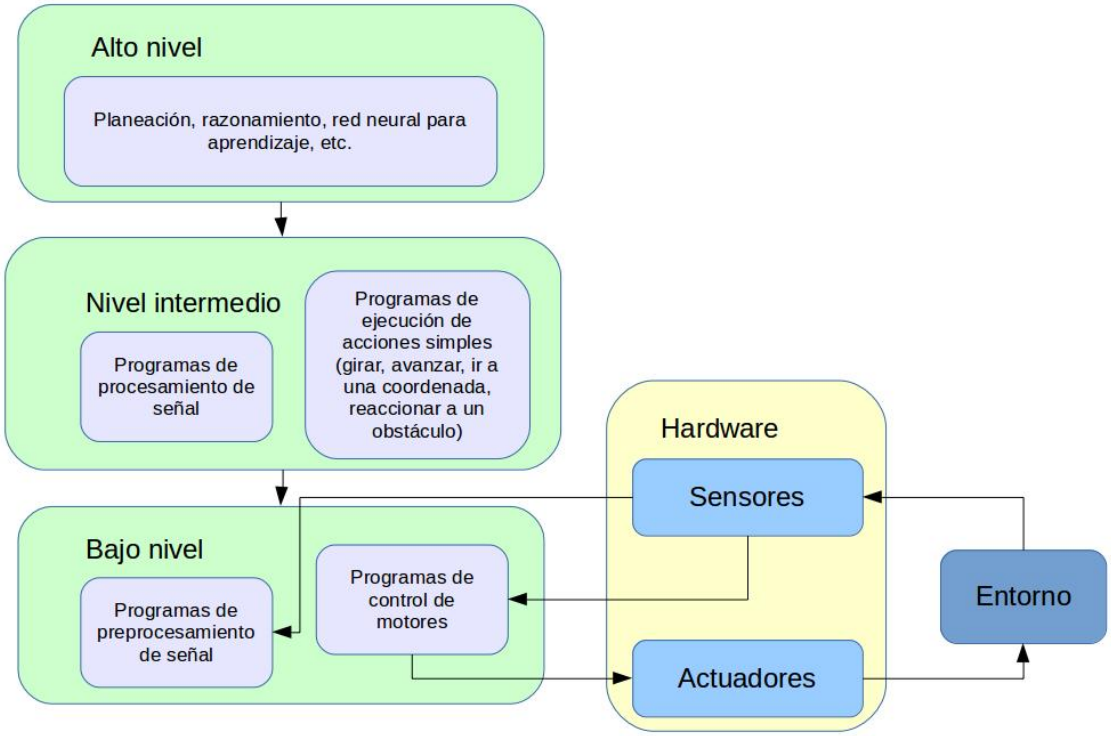

Figura 1.2 Jerarquía de tres capas de los programas del robot Shakey

1.1.1 El robot FinDER

El robot FinDER es un proyecto de diseño de robots para entornos de desastre llevado a cabo en la Facultad de Ingeniería de la UNAM; tiene como objetivo planteado servir como un agente de búsqueda en entornos de desastre, como un derrumbe o un sismo en una zona urbana, donde el terreno es escarpado, inaccesible y peligroso para un ser humano.

Lo que se busca en el robot FinDER es un comportamiento siempre respaldado por un operador remoto. Lo único que el robot va a percibir son las acciones de sus propios actuadores (por ejemplo, el desplazamiento de sus orugas o el desplazamiento de un brazo auxiliar), sin embargo, en la generación de acciones para actuar sobre el medio, siempre estará el operador remoto tomando decisiones, es decir, que el desarrollo de la capa de programas de nivel intermedio y alto no son aún necesarias, pero si deseables como desarrollo a largo plazo. El esquema del robot FinDER se muestra en la Figura 1.4.

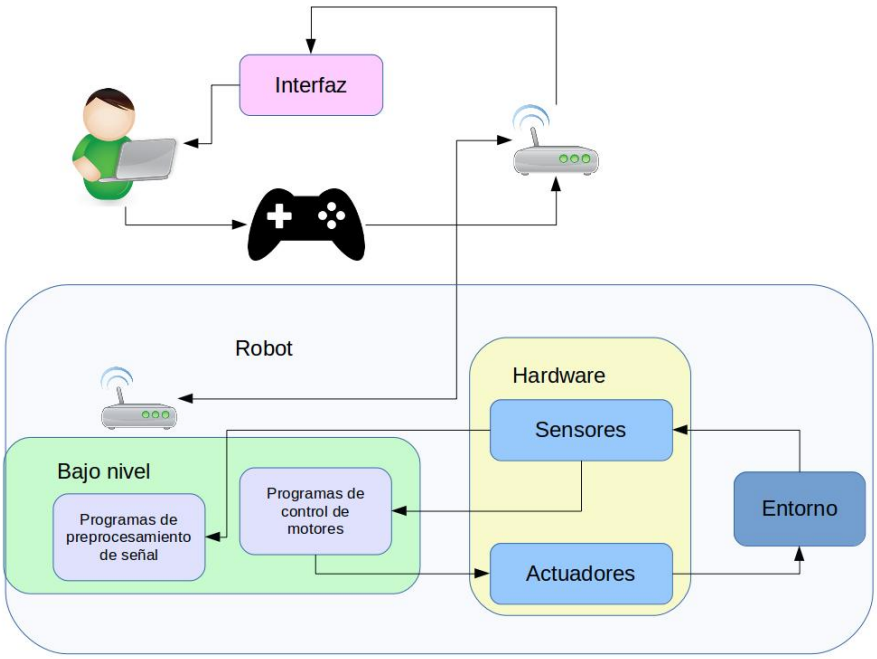

Figura 1.3 Estructura del robot FinDER como sistema

1.1.2 RoboCup

Una referencia sobre el entorno al que un robot de búsqueda debe enfrentarse lo proporciona la organización RoboCup para su competencia en la categoría Rescue **[3]**, RoboCup Rescue tiene como objetivo desarrollar y demostrar capacidades robóticas para responder ante situaciones de emergencia mediante una competencia anual que sirva para evaluar robots ante escenarios de desastre simulados.

De su *Rulebook o Libro de Reglas* que es la especificación de las pruebas que realizan en la competencia **[4]**, se pueden obtener algunas de las características de un entorno de desastre, o al menos una aproximación de los requerimientos de un robot que va a operar en un entorno de desastre.

El documento de la RoboCup Rescue sirve de referencia porque subdivide las tareas principales que un robot de rescate debe poder realizar; es decir, plantea pruebas que se clasifican por maniobrabilidad, movilidad, destreza y exploración, además de dar algunos lineamientos sobre la conFiguración de sensores que se ocupa en un robot de rescate.

Basados en las características de las arenas planteadas así como de escenarios reales, se plantea el entorno donde va a operar el robot, el cual es estocástico, es decir, de naturaleza cambiante. El robot estará sujeto a golpes, cambios bruscos de dirección, caídas, escaleras, pendientes, además de que el objetivo del robot es reconocer lo mejor posible el entorno y dar al teleoperador información actualizada periódicamente sobre el estado actual de sus sensores dedicados a la detección de víctimas, principalmente.

1.1.3 Microcontroladores

La arquitectura de conexión de elementos y organización de programas del robot Shakey mencionada al inicio del capítulo, tiene una parte no especificada en el robot FinDER hasta este momento, que es la comunicación con el hardware a bordo del robot. En el robot Shakey es la computadora de control de periféricos PDP-15. En el caso del FinDER, se hace uso de microcontroladores para hacer la comunicación con los sensores y electrónica de potencia del robot.

> *"Un microcontrolador es un circuito integrado programable que contiene todos los componentes necesarios para controlar el funcionamiento de una tarea determinada, como el control de una lavadora, un teclado de ordenador, una impresora, etc. Para esto, el microcontrolador utiliza muy pocos componentes asociados. Un sistema con un microcontrolador debe de contar con una memoria donde se almacena el programa que gobierna el funcionamiento del mismo que, una vez conFigurado y programado, sólo sirve para realizar la tarea asignada"**[5].***

Otra alternativa, es el uso de un FPGA (un FPGA es un arreglo de compuertas programables en el campo. Es un dispositivo programable con bloques de lógica cuya interconexión y funcionalidad puede ser programada en el momento), que es también una opción a tomar en cuenta cuando se diseña la parte de enlace entre la computadora y la electrónica de control del robot; la desventaja que presenta frente a un microcontrolador yace en la complejidad de la programación del FPGA, ya que requiere conocimientos de electrónica digital además de los conocimientos de programación, y que tiene un coste elevado respecto a los microcontroladores. No obstante, debe de tenerse en cuenta que el uso de FPGA ha estado aumentando gradualmente en los últimos años, y presenta también ventajas a considerar respecto a los microcontroladores y a los microprocesadores en general, ya que en la complejidad de programación antes mencionada, también yace libertad para optimizar procesos y realizar varias tareas de forma simultánea.

Por último, también se pueden considerar computadoras embebidas. Una computadora embebida contiene la funcionalidad de ambos, computadora y microcontrolador en una misma unidad, con lo cual el usuario no tiene que preocuparse en hacer las conexiones ni programar las rutinas de comunicación entre ambos ; la entrada es la interfaz con el usuario

y su salida son los periféricos de una computadora como el teclado, pantalla, mouse, además de un puerto GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General, es un pin genérico en un chip, cuyo comportamiento puede programar el usuario en tiempo de ejecución) y la capacidad de comunicación con otros dispositivos mediante I2C, protocolo serial, SPI, entre otros, dependiendo del fabricante. Si hay que mencionar algún aspecto negativo es que, al ser un sistema completo, si hay una falla de voltaje en el circuito, afectará al procesador de la computadora a bordo de la placa.

En la Figura 1.4 se muestra gráficamente los tres métodos mencionados para hacer la conexión entre un sistema físico y una computadora.

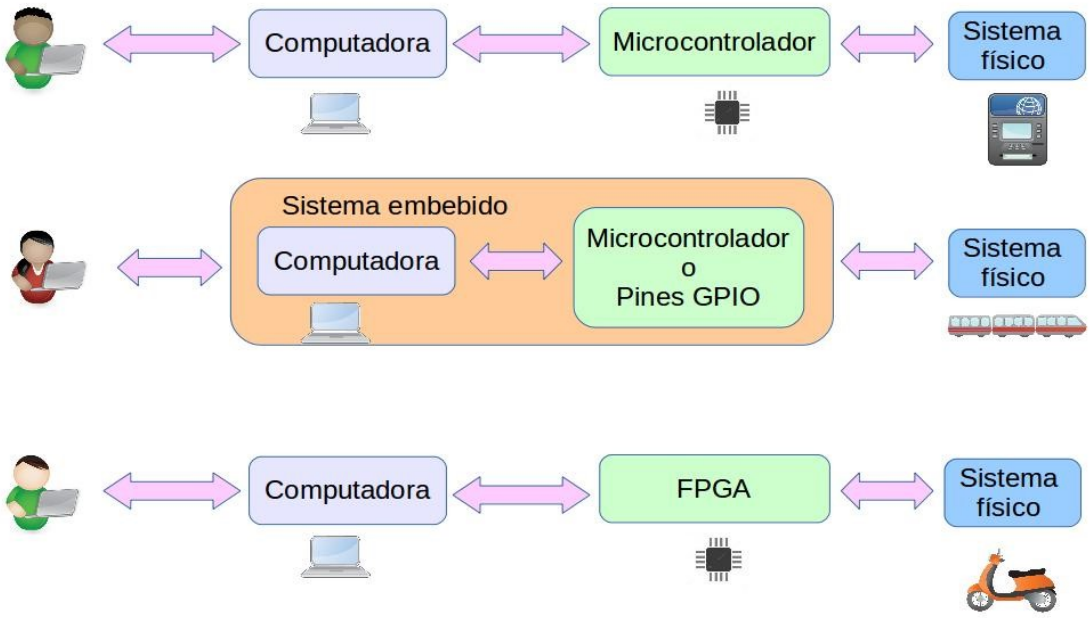

Figura 1.4 Conexión de un microcontrolador, una computadora embebida y un FPGA como interfaz entre un sistema físico y una computadora.

En la actualidad, existen varias empresas dedicadas a construir microcontroladores como Microchip Technologies, Texas Instruments, Atmel y ST Electronics. Normalmente cada empresa lanza un entorno de desarrollo específico para que el programador pueda dedicarse a hacer las rutinas de control de los periféricos y el tratamiento de datos mientras que los procedimientos asociados a la obtención de la información de los periféricos vienen normalmente en una API (Interfaz para la Programación de una Aplicación) en la cual solamente se llama a una función para realizar el control de un periférico.

Lo anterior ha permitido que surjan proyectos como Arduino, que consiste en un entorno de desarrollo de programación o IDE por sus siglas en inglés, junto con placas de desarrollo haciendo muy accesible la programación y aprendizaje del uso de microcontroladores. Tomando ventaja de las API de las empresas que manufacturan los microcontroladores y

tarjetas de desarrollo, y que generan bibliotecas genéricas que facilitan la implementación de los protocolos de transmisión de datos como UART, I2C,CAN, configuración de los pines como entrada/salida, ADC, PWM, entre otros.

1.1.4 Sistemas de control por realimentación

En la definición de robot, una parte menciona a una máquina que realiza una actividad con cierto grado de autonomía, y en la definición del entorno de operación del FinDER también se menciona que el robot debe tener sensores unidos a los elementos mecánicos que están siendo desplazados mediante un actuador.

Esto conduce a la definición de un sistema de control; cuando se tiene un sistema unido a una fuente de energía externa y se quiere que tenga un comportamiento deseado sin tener intervención humana, se deben utilizar dispositivos auxiliares que ayuden a lograr dicha meta. Una definición aceptable es *"Un sistema de control es todo aquel conjunto de dispositivos que actúan sobre otro sistema para modificar su comportamiento conforme a una referencia deseada*" **[6]**. Para terminar la definición cabe mencionar que se puede tener un sistema de control que logre su objetivo sin la necesidad de tener conocimiento alguno sobre el entorno y las variables físicas que influyen en el comportamiento del mismo, sin embargo cuando se miden estas variables físicas y se les permite influir en el desempeño del sistema de control se denomina un sistema de control por realimentación, ya que la variable física sensada influye ya sea para disminuir o aumentar el aspecto físico del sistema que representa en la operación del mismo.

En el caso de un robot como el FinDER o Shakey, se tienen motores unidos a ruedas, los cuales tienen como función mover al robot, y el comportamiento que se desea en ellos es que muevan al robot hacia adelante, hacia atrás, que hagan girar al robot sobre su eje o que se desplace en diagonal. Entonces, si se pueden enviar comandos de PWM (Pulse Width Modulation, es una señal de control que modifica el ciclo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga) a la electrónica de potencia que controla los motores se podrá tratar de lograr alguno de los comportamientos antes mencionados. En la práctica se tendrían problemas para lograrlo, sin embargo, como se observa en la Figura 1.5, al agregar información sobre el entorno, se puede influir en los comandos de PWM que se envían a los motores. Si esta información es la posición angular de las ruedas proporcionada por el sensor de posición angular en la rueda, y se obtiene la velocidad de desplazamiento midiendo el tiempo entre cada cambio de posición, los comandos de PWM se pueden ajustar conforme sea el comportamiento deseado del robot.

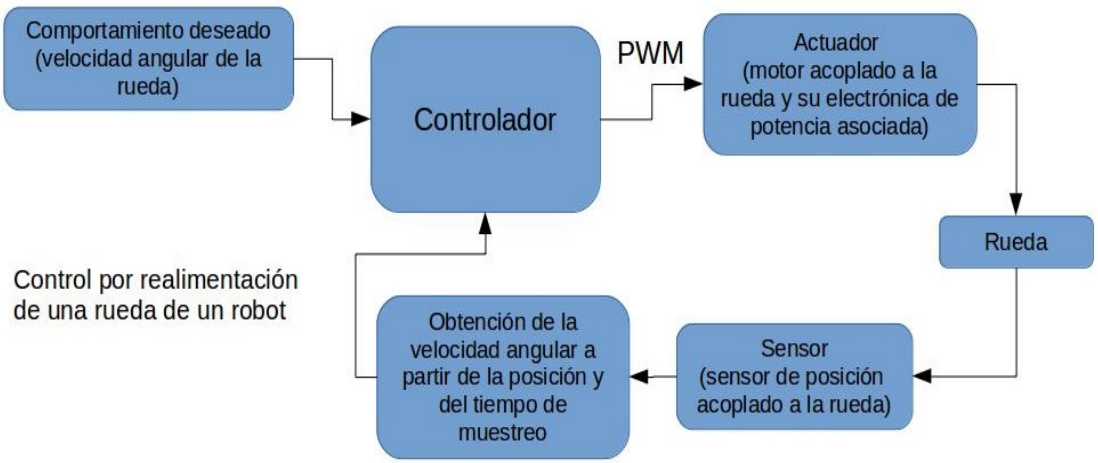

Figura 1.5 Ejemplo de un sistema de control por realimentación.

Un tipo de controlador comúnmente utilizado es el controlador PID (Proporcional, Integral, Derivativo), como se observa en la Figura 1.6, un controlador se alimenta del valor deseado (también conocido como objetivo de control y representado por la función r(t)) y de la realimentación proporcionada por el sistema físico (también conocido como planta) y da como salida una señal de PWM, que en términos más generales es la salida del controlador o señal de control, también definida por la función u(t).

El control PID como el nombre completo lo indica se conforma de tres componentes, la proporcional, la integral y la derivativa; en conjunto lo que tienen como objetivo es reducir a cero el error definido como la diferencia entre el objetivo de control y el valor de realimentación del sistema proporcionado por el sensor; la parte proporcional es la multiplicación del error por una constante durante la operación del sistema, la parte integral multiplica una constante por el valor acumulado del error de iteraciones pasadas, por lo tanto, va cambiando conforme el error tenga un valor diferente a cero, y tiene como efecto ayudar a que el error tienda rápidamente a cero; no obstante, al hacer muy grande la constante de la parte integral, provocará oscilaciones en el comportamiento del sistema. Por último, la parte derivativa es la multiplicación de una constante por la derivada del error; el signo de la derivada indica hacia donde tiende el error y su magnitud indica qué tan rápido crece o decrece, por tanto la parte derivativa lo que hace es anticiparse a un cambio rápido en la magnitud del error compensándolo.

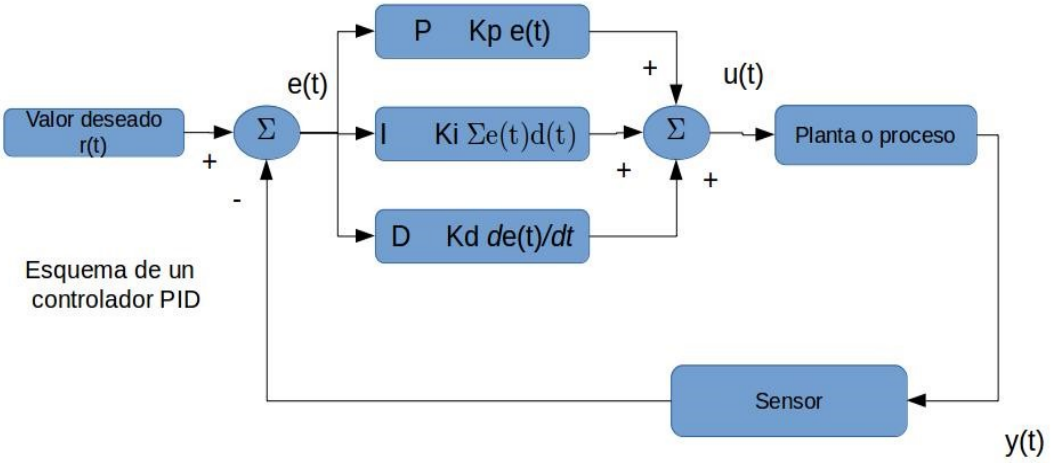

Figura 1.6 Esquema de un controlador PID en tiempo continuo.

Como puede observarse en la Figura 1.6, las funciones que definen al valor deseado r(t), al error e(t), a la señal de control u(t) y a la salida de la planta y(t), son funciones continuas en el tiempo; sin embargo, al hacer uso de una computadora para hacer el control de un sistema, dicho sistema debe cambiar a un sistema discreto como él mostrado en la Figura 1.7. Para ello conviene describir qué es una señal continua y una señal discreta. Una señal es una función que depende de alguna variable que tiene un significado físico. Una función tiene dominio y rango, si ambos son continuos se dice que la señal es continua o analógica; si el dominio es discreto y el rango analógico, la señal es discreta, y si ambos son discretos, la señal es digital.

Un sistema de tiempo discreto, es un sistema dinámico en el cual una o más variables pueden variar únicamente en ciertos instantes. Estos instantes, llamados de muestreo y que se indican por kT (k= 0,1, 2, …) pueden especificar el momento en el cual se realiza una medición física o el tiempo en el que se lee la memoria de la computadora.

El valor de T en la expresión kT es conocido como periodo de muestreo, y es el intervalo de tiempo en el que se obtiene una nueva señal de entrada. Los bloques que dicen A/D y D/A significan que en esa parte se hace una conversión de señal analógica a digital, y de digital a analógica. Una señal digital es fácilmente interpretable y utilizable por una computadora; por eso se hace uso de estos convertidores para pasar de una señal continua a discreta para que la computadora pueda procesar la información, y de digital a analógica para que la señal digital de control u(kT) generada por la computadora pase a la planta o proceso como una señal continua u(t). Esto puede verse en la Figura 1.7.

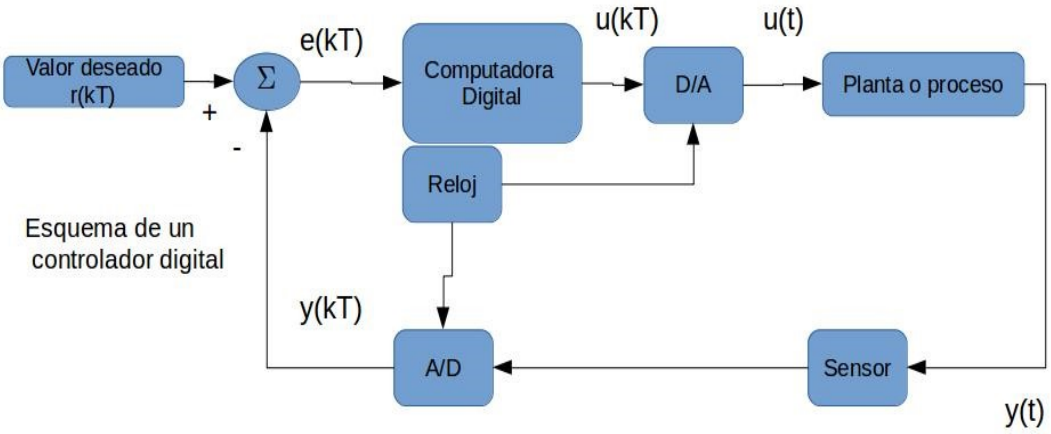

Figura 1.7 Esquema de un sistema de control digital. Si se compara al de la figura 1.6 se aprecia la aparición de elementos que permiten el funcionamiento del controlador en tiempo discreto

1.1.5 ROS

ROS (Robot Operating System) es un *framework* (una plataforma de desarrollo de programación) flexible para escribir programas para robots. Es un conjunto de herramientas, bibliotecas y convenciones que buscan simplificar la creación de comportamientos en robots robustos y complejos a través de una amplia variedad de plataformas robóticas.**[7]**

Este *framework* es modular y gracias a su amplia documentación y colaboración de miles de personas, consiste en una herramienta que facilita el desarrollo a nivel de programación en un robot.

A partir de su creación en el año 2007 como un esfuerzo por unificar y crear una plataforma de desarrollo de robots, ha sido probado en múltiples robots y proyectos de desarrollo e investigación en robótica. Por ello desde la segunda versión del robot FinDER hizo de esta plataforma de desarrollo, con la distribución Indigo.

1.2 Objetivos

El objetivo es tener un robot que funcione en un entorno de desastre de forma teleoperada, que sea seguro en su uso teniendo en cuenta sus restricciones mecánicas, es decir, que las partes móviles del mismo no causen una falla irreversible en el robot; asimismo, que el sistema eléctrico, a nivel de cableado, sea seguro en su utilización, ya que al ser un sistema teleoperado, una falla en el sistema eléctrico es un riesgo para las personas que estén alrededor del robot e incluso para el robot mismo.

1.3 Resumen por capítulos

En este capítulo se habla de los aspectos básicos de arquitectura de un robot, las capas de desarrollo existentes, se introduce al robot de búsqueda en entornos de desastre FinDER desarrollado en la Facultad de Ingeniería de la UNAM, se explican conceptos básicos que posteriormente tienen uso, como son los microcontroladores y tarjetas de desarrollo, la descripción de un sistema de control por realimentación, y se da una pequeña introducción a ROS, una plataforma de desarrollo tipo código abierto para proyectos de robótica.

En el Capítulo 2 *Análisis del problema,* se hace una descripción rigurosa de los sistemas mecánicos del robot FinDER v3, así como de la implementación de la capa de bajo nivel del robot FinDER v2, y a partir de ello se genera una definición precisa del problema a resolver, las limitaciones, restricciones, e identificación de aquellas variables útiles para hacer una búsqueda de soluciones. Con base en ello, se generaron matrices de decisión y se explica cómo se eligieron los componentes de la propuesta de solución.

En el Capítulo 3 *Diseño de las tarjetas impresas,* se explica cómo se diseñó la implementación física (a nivel de hardware, es decir sensores y módulos de potencia, cuando se habla de un sistema físico, entonces se trata de un mecanismo y su correspondiente actuador acoplado) de la solución propuesta en el Capítulo 2, es decir, el ordenamiento de sensores y módulos de potencia del robot, la implementación del programa que reside en la tarjeta de desarrollo encargada de ser el enlace entre la computadora y los sistemas físicos del robot.

En el Capítulo 4 *Software de bajo nivel,* se hace un análisis de los programas adaptados y desarrollados para la implementación del control mediante un teleoperador de los sistemas físicos del robot. Dichos programas se desarrollaron con lenguaje de programación C++, ya que es eficiente en términos de ejecución y muchos de los programas adaptados fueron desarrollados originalmente en Python.

En el Capítulo 5 *Pruebas y análisis de resultados*, se hace un contraste entre los resultados obtenidos y los criterios y restricciones planteados en el Capítulo 2, además de algunas muestras de los resultados logrados con los nodos de control por realimentación desarrollados en el Capítulo 4.

El Capítulo 6 *Conclusiones y trabajo a futuro* es dedicado a dar una conclusión del trabajo realizado y dar un panorama de los posibles caminos de desarrollo del proyecto mediante mejoras, observaciones y propuestas para mejorar el desempeño del robot.

Capítulo 2
Planteamiento de problema

El robot FinDER es un proyecto de robótica que ha ido evolucionando como se muestra en la Figura 2.1. Su método de tracción y el brazo manipulador han ido cambiando conforme se han encontrado soluciones a los problemas de diseño mecánico y electrónico, así como en los programas de control.

Figura 2.1 Fotografías que muestran el proceso iterativo del robot FinDER, de izquierda a derecha el FinDER v1, el FinDER v2 y el FinDER v3

En el caso del FinDER v3 (Figura 2.2), existe como antecedente el robot FinDER v2. A partir de la evaluación de desempeño del robot FinDER v2, se detectaron problemas mecánicos al probar el robot en entornos simulados de desastre como terrenos irregulares, escaleras y vados. Mediante la evaluación de su desempeño, se generaron las siguientes modificaciones que dieron paso a la versión v3.

El sistema de tracción y el chasis permanecieron intactos; el brazo manipulador se cambió casi completamente excepto por la base que se une al chasis; los brazos auxiliares también se cambiaron, además de las orugas y se agregó un sistema de suspensión para amortiguar el desplazamiento del robot en terrenos escabrosos.

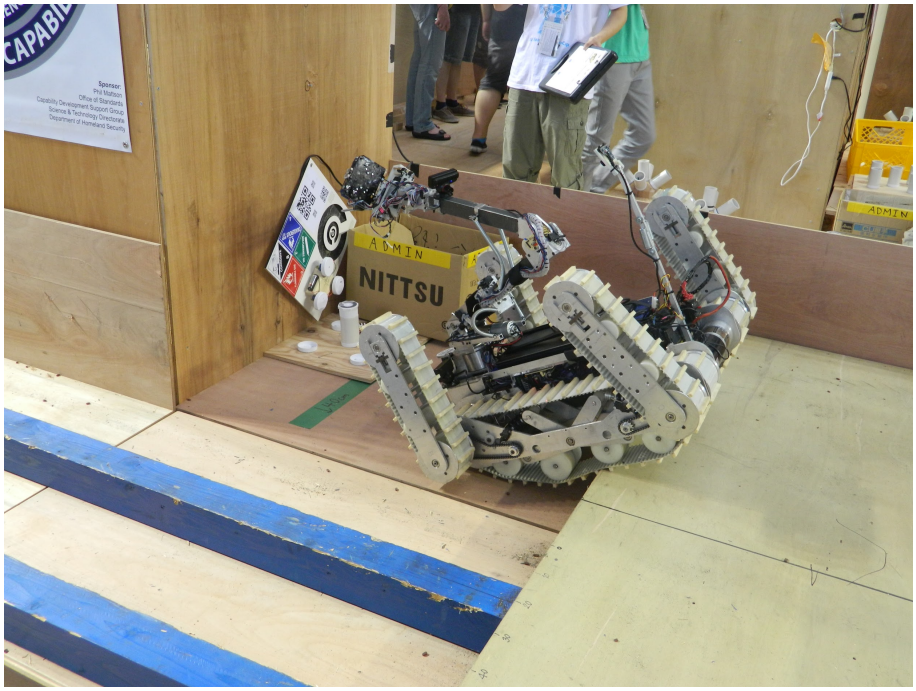

Figura 2.2 Robot FinDER v3 realizando pruebas de movilidad y destreza en la Robocup 2017

El robot FinDER v3, mostrado en la Figura 2.2, cuenta con dos orugas de tracción que permiten al robot desplazarse sobre terreno irregular, ayudado por cuatro brazos auxiliares o flippers que sirven como apoyo al enfrentarse a obstáculos como escalones y pendientes.

2.1 Descripción de los sistemas mecánicos del robot

En el sistema de tracción del FinDER v3 el movimiento de las orugas está dado mediante poleas dentadas, que están unidas al eje de un motor, la tensión del mecanismo polea-banda dentada se produce mediante un mecanismo de suspensión que aprovecha el peso del robot para mantener las condiciones necesarias para que, al desplazarse el motor, se desplace la banda dentada sin que afecten las condiciones del terreno; cabe destacar que la versión anterior carecía de un sistema de suspensión.

2.1.1 Sistema de tracción

En la Figura 2.3 se aprecia un diagrama de las partes involucradas en la tracción del robot. Los motores de tracción son motores AmpFlow E30-400 que permanecen de la versión anterior; los OSMC (Open Source Motor Controller es un módulo de control de motores, ver en Anexo A) son la etapa de potencia de la electrónica de control del motor. Además, se muestra dónde está colocado uno de los *encoders* de posición de una oruga de tracción.

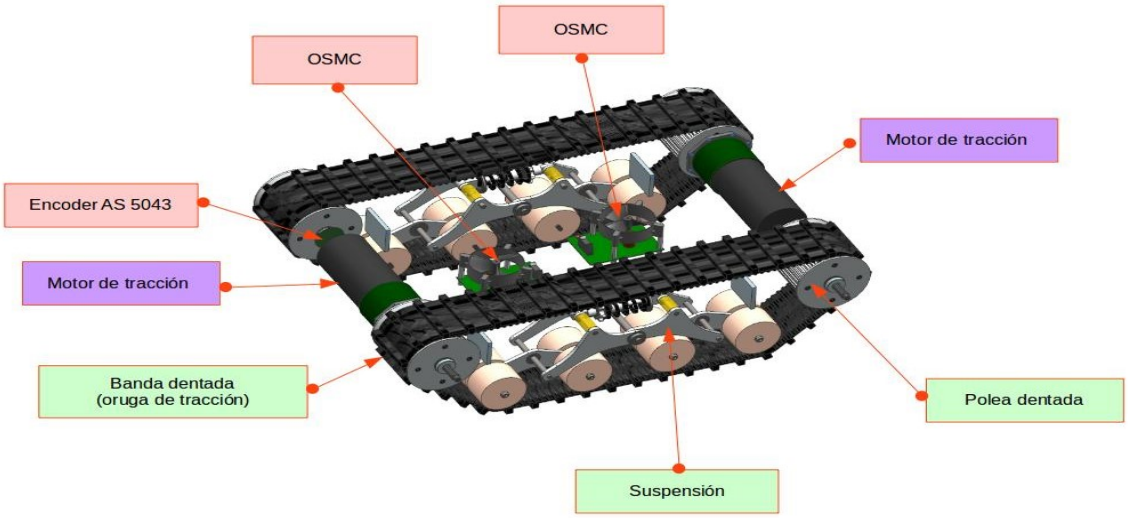

Figura 2.3 Elementos mecánicos del sistema de tracción del FinDER v3. Los motores principales de tracción son dos, en color negro, cada uno controlado por un módulo de potencia OSMC, en los ejes asociados al movimiento angular de las poleas de tracción se posicionaron sensores de posición angular AS5043.

En cada uno de los brazos auxiliares, en el robot Finder v2 se tenía un mecanismo catarina-cadena-catarina entre el motor-reductor y el brazo, sin embargo, dicha etapa de transmisión de potencia era insuficiente debido al peso del robot; por ello en el rediseño se decidió usar dos etapas de transmisión de potencia mecánica, la primera es un sinfín corona con auto bloqueo, la cual permite un control de posición preciso aún en condiciones de carga en la articulación. El *encoder* de posición angular AS5043 está acoplado al eje de la corona; después se implementó una segunda etapa de transmisión Catarina-cadena-catarina, que cumple la función de trasladar el par hasta el lugar donde está ubicado el brazo auxiliar para posicionarlo.

Además de este movimiento angular, cada brazo auxiliar cuenta con una oruga de tracción, la cual es accionada usando la misma potencia de los motores de tracción mediante el uso dos ejes colineales y sobre puestos mediante un rodamiento, el eje externo acoplado a la catarina que le proporciona un movimiento angular al brazo, y el eje interno acoplado a la polea dentada del brazo auxiliar y a la polea dentada del sistema de tracción principal del robot. En la Figura 2.4 se puede apreciar el sistema mecánico que compone un brazo auxiliar. Al adicionar la etapa de transmisión de sinfín- corona, se protege a los motores ya que el auto bloqueo de esta transmisión evita que el motor se active para mantener la posición del brazo auxiliar, esto es útil en situaciones donde el peso del robot puede ser soportado por solo un par de brazos auxiliares, por ejemplo.

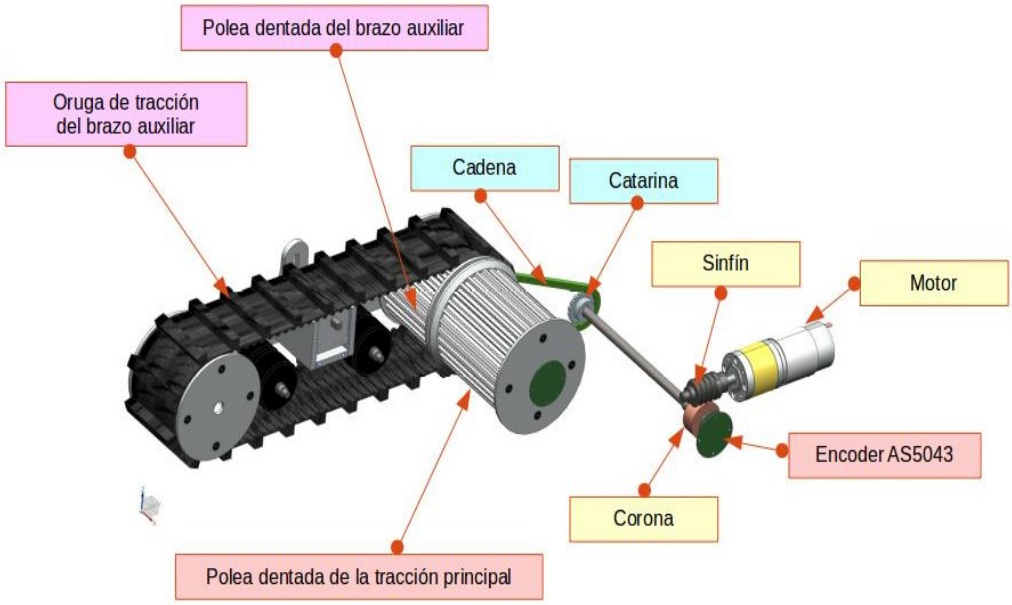

Figura 2.4 Elementos mecánicos de un brazo auxiliar del FinDER v3, el robot FinDER v3 cuenta con cuatro brazos auxiliares capaces de soportar el peso del robot gracias a una transmisión de potencia en dos etapas, primero un sinfín corona y a continuación una catarina cadena catarina.

2.1.2 Sistema de brazo manipulador y efector final

El brazo manipulador del robot FinDER v3 cuenta con seis grados de libertad, que son seis juntas rotacionales que permiten al operador establecer la posición y orientación del efector final.

En el efector final se tiene un servomotor acoplado a una pinza, un sensor térmico, uno de CO2 y una cámara de video, de manera que el brazo cumple ambas funciones, dar información al teleoperador del entorno mediante la cámara de video y las lecturas de los sensores, y manipular algún objeto del entorno que no exceda la capacidad de carga del efector final utilizando el servomotor.

Las tres primeras juntas rotacionales del brazo robótico tienen como sensor de posición angular un *encoder* AS5043, mientras que en las tres juntas de rotación del efector final se tienen motores con *encoder* de cuadratura integrado (ver Figura 2.5).

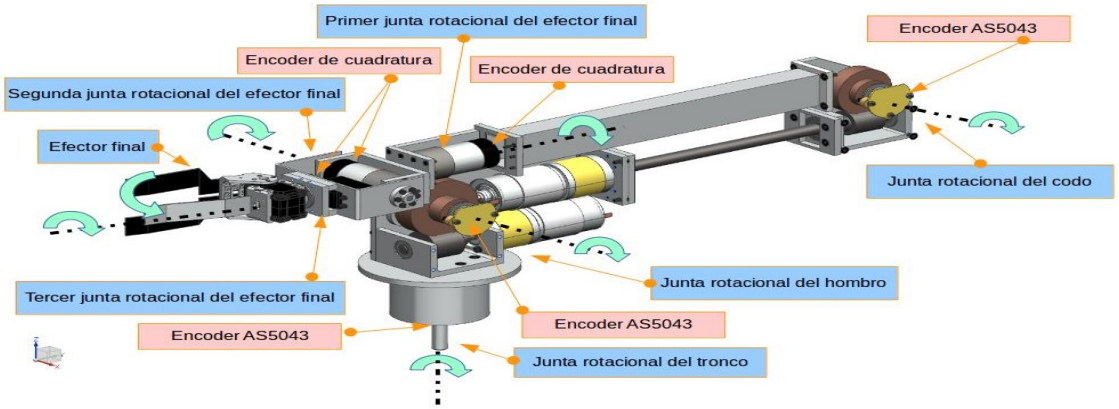

Figura 2.5 Elementos mecánicos de un brazo manipulador y efector final del FinDER v3. En total el brazo manipulador cuenta con seis grados de libertad, en la figura se muestran los tipos de transmisión utilizados en cada junta, así como la colocación de los sensores de posición angular AS5043

2.2 Capa de Hardware del robot Finder v2

Una vez realizado el rediseño mecánico del robot, se procedió a la revisión del sistema de control del robot, comenzando por la parte de electrónica de potencia y sensores, los microcontroladores utilizados y sus correspondientes programas; y la arquitectura de software en la computadora del robot que estaba implementada en la plataforma de desarrollo ROS. La capa de hardware está ilustrada en la Figura 2.6.

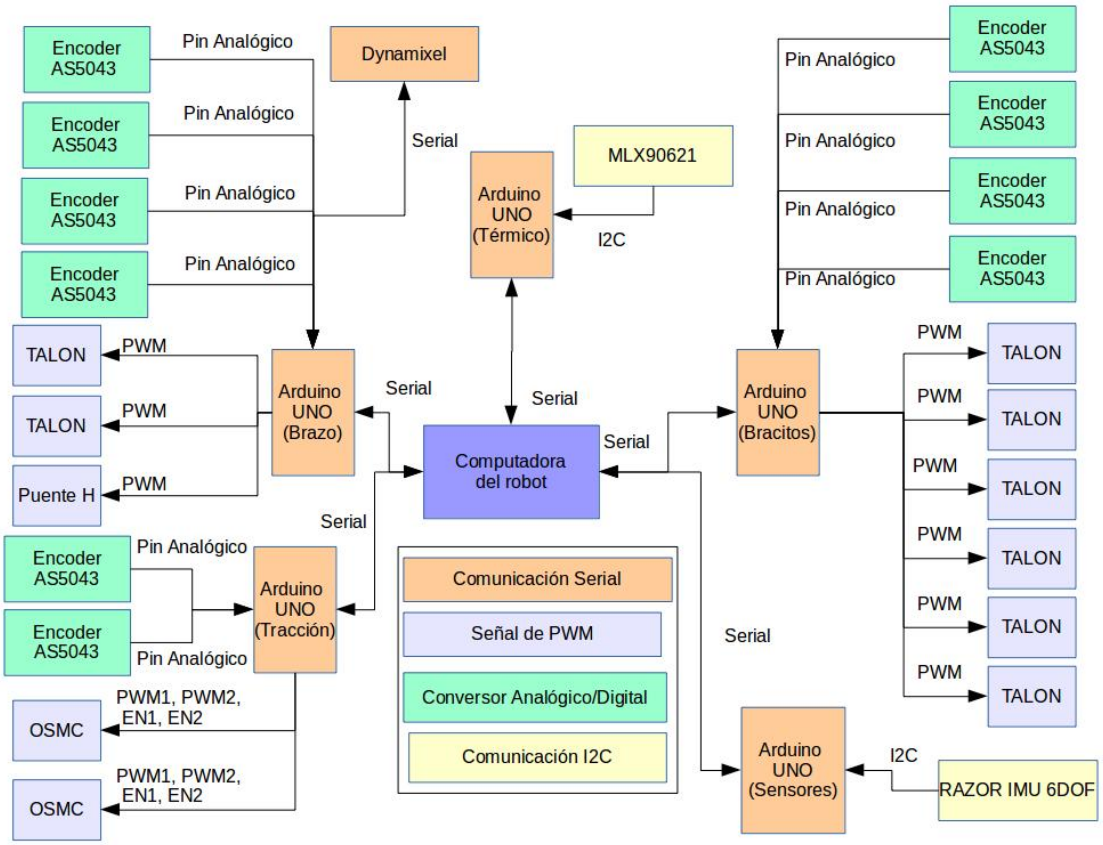

Figura 2.6 Esquema de conexión de los elementos de hardware del robot FinDER V2, la figura muestra la arquitectura heredada del robot FinDER v2 en cuanto a la distribución de sensores, módulos de potencia y microcontroladores.

2.2.1 Descripción de los sensores y módulos de potencia

Como puede observarse en la Figura 2.6, siguiendo el flujo a partir del bloque del centro en color azul que dice “Computadora del robot”, parten los microcontroladores que se comunican con los sensores y módulos de electrónica de potencia que manejan los motores del robot. En el robot FinDER v2 se hace uso del sensor de posición angular AS5043 (el funcionamiento a detalle del sensor está en el Anexo A), el cual tiene como posibles salidas de señal, una señal digital mediante SPI y una señal analógica, se hace uso de la señal analógica por facilidad de implementación, ya que se utilizan dos cables menos que si se usa la señal digital SPI.

En la parte de los módulos de electrónica de potencia se utilizan dos módulos OSMC para el control de los dos motores de tracción, cuatro módulos TALON SR (se puede revisar su funcionamiento a detalle en el anexo A) para los brazos auxiliares, y Otros cuatro módulos TALON SR y un puente H diseñado y construido ex profeso para el control del brazo manipulador de cinco grados de libertad del FinDER v2.

La tarjeta de desarrollo utilizada para la comunicación entre la computadora del robot y sus periféricos es la tarjeta de desarrollo Arduino UNO, que es fácil de utilizar gracias al entorno de desarrollo proporcionado por el fabricante, la extensa documentación y existencia de bibliotecas para la implementación de protocolos de comunicación.

2.2.2 Evaluación del desempeño y puntos a mejorar

Al evaluar el desempeño de esta implementación de enlace entre sensores y actuadores con la computadora principal, se notaron algunos puntos en que puede mejorar. El primero es que en esta implementación se hace uso de una API de comunicación serial que ofrece ROS llamada rosserial; esta API tiene como características el simplificar la complejidad de programación del protocolo de comunicación serial, ya que permite publicar y recibir tópicos directamente de los nodos de ROS que estén ejecutándose en la computadora central. No obstante, en esta simplificación sobreviene una carga de procesamiento adicional en el microcontrolador, por lo que se tiene que balancear los sensores y actuadores asignados a cada microcontrolador de manera que no se exceda el tiempo de muestreo en alguno. Si se mira detalladamente la Figura 2.6 se notará, por ejemplo, que la tarjeta designada como Arduino UNO (bracitos) y Arduino UNO (brazo), se complementan entre si para el control de actuadores y adquisición de señal de los sensores. La configuración recomendada sería la de la tarjeta Arduino UNO (tracción) donde el número de sensores coincide con el número de actuadores que controla.

Un segundo punto a mejorar son los sensores y actuadores. Se observó que faltan sensores que monitorizen el inicio y fin de recorrido de las juntas rotacionales del brazo manipulador y de los brazos auxiliares, y según el *Rulebook* del RoboCup, se debe tener incorporado un sensor de gas CO2 en el robot. Además, el sensor AS5043 tiene un modo de adquisición de datos mediante SSI el cual da una señal digital filtrada permitiendo reducir el tiempo de procesamiento de la señal analógica en el microcontrolador; el costo de este cambio, sería aumentar dos cables más para cumplir con el protocolo de comunicación digital SSI.

2.3 Formulación del problema

Teniendo en conocimiento la anterior arquitectura de electrónica y programación del robot y de las mejoras realizadas a los sistemas mecánicos como antecedentes, el problema que se plantea en su forma más general es la implementación de un sistema que conecte el sistema físico, es decir, el robot FinDER v3 y sus mecanismos de locomoción asociados con una computadora, la cual controlará dichos mecanismos de forma automática o con la intervención de un teleoperador.

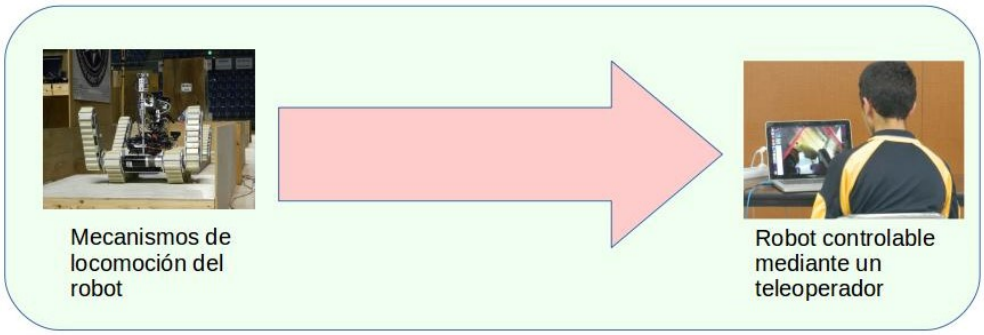

Figura 2.7 Formulación del problema. En términos generales se buscó generar un enlace entre los sistemas físicos del robot y el teleoperador que facilitara el uso del robot de forma remota.

A partir del enunciado anterior y de lo presentado en los dos apartados anteriores y en los Antecedentes, se realizó el siguiente análisis del problema: se identificaron las variables de entrada, de salida, y las variables de solución. Así como las restricciones de la implementación del problema.

Entrada: Mecanismos de locomoción del robot FinDER v3 y la arquitectura de electrónica y programación del robot FinDER v2

Variables de entrada: Limitaciones de las entradas

Número de sensores----------Tiempo de muestreo

Número de actuadores----------Duración de las baterías

Número de microcontroladores-----No exceder el 50% del uso del procesador de la computadora en la comunicación con ellos

API de comunicación serial micro-computadora----------Ninguna

Salida: Robot teleoperable

Variables de salida: Limitaciones de la salida

Acciones del robot----------Grado de cumplimiento de las tareas deseadas

Versatilidad----------Capacidad de acoplarse a software desarrollado por terceros

Confiabilidad----------El robot no realice acciones que lo dañen

Variables de solución

Microcontrolador (placa de desarrollo)

API de comunicación serial

Módulo de potencia

Sensores

Modularidad de la solución

Arquitectura de programación

Restricciones

Batería de 12V 18 Ah alimenta el sistema

El robot debe tener redundancia en cuanto a seguridad de operación ya que es teleoperado

Debido a las modificaciones de los sistemas mecánicos, el espacio para el hardware es menor

Se tiene restricción de costo, se tratará de reutilizar el mayor número de piezas de hardware y de código.

Criterios

Facilidad de programación

Consumo de CPU

Costo

Confiabilidad

Operabilidad

Figura 2.8 Análisis del problema. En el proceso de diseño, se identificaron las variables de solución, de entrada y de salida. Así como las restricciones propias del problema, y los criterios a tomar en cuenta para el desarrollo de la solución.

A partir de las variables de solución, se generaron diagramas con las posibles soluciones al alcance, en las Tablas 2.1 y 2.2 se muestran.

Tabla 2.1 Posibles soluciones para la selección del microcontrolador a utilizar

Sistema embebido	Raspberry pi B+ Beagle Bone Black
Arduino	Arduino Uno Arduino MEGA Arduino Leonardo Arduino DUE
Texas Instruments	MSP430 TivaCtm4123 TivaCtm4129
Teensy	Teensy 2.0 Teensy 3.6
Microchip	PIC16F1939 PIC18F4550

Tabla 2.2 Posibles soluciones para la API de comunicación serial

Rosserial	Software desarrollado por la comunidad de ROS para establecer comunicación serial con un microcontrolador y comunicarse directamente en el contexto de publicadores y suscriptores del entorno de ejecución de ROS
SerialGateway	Software desarrollado por la empresa qobticslab para su robot comercial Chefbot, es una implementación ligera de comunicación serial basada en dos programas, uno de adquisición de la información proveniente del microcontrolador, y otra que pasa dicha información al contexto de ejecución de ROS mediante publicadores y subscriptores
PySerial	Software open source desarrollado para comunicación serial utilizando Python como herramienta de desarrollo
Crear un programa a la medida en C++	Utilizando las bilbiotecas estándar de C/C++ hacer un programa que se comunique con el microcontrolador por el puerto serial, y convierta la información de entrada y salida al contexto de ejecución de ROS

Tabla 2.3 Posibles soluciones para los módulos de potencia/actuadores

Talon SR	60 A continuos
RoboClaw	2x7 A 2x15 A 2x30 A 2x45 A
Cytron	13 A
OSMC(Open Source Motro Controller)	160 A
Dynamixel AX1-2A ó AX-18A	0.9 A

Tabla 2.4 Posibles soluciones para los sensores a utilizar en el robot

Sensores de posición angular	AS5043 AS5040 AS5234 *Encoder* de cuadratura Sensores de final de carrera tipo interruptor
Sensores de imagen	Cámara RGB Cámara térmica Cámara de profundidad
Sensores de CO2	COZIR
Sensores de aceleración y giroscopio	Razor IMU 9DOF MPU6050
Sensor tipo LiDAR	Hokuyo URG-04LX

En el proceso de selección de soluciones, se consideró acotarlos en función del costo, de la facilidad de programación, de la facilidad de obtención o reutilización de elementos ya existentes en la versión anterior del robot. Se establecieron criterios de selección para cada una de las variables de solución descritas, y se hicieron matrices de decisión para llegar a la mejor solución de acuerdo a los criterios establecidos.

2.3.1 Sensores y módulos de potencia

En relación con los sensores y los módulos de potencia, previamente se había concluido que los sensores utilizados en la versión anterior tenían un desempeño aceptable para la operación del robot, por lo que únicamente dos restricciones se tomaron en cuenta; la primera, el costo de cambiar algún módulo o sensor; y segundo, la reducción de espacio para colocar los módulos de potencia debido a las modificaciones en el robot. También hay que resaltar que la selección del módulo de potencia depende también del motor que se desee controlar. En la Tabla 2.5 se muestran las características de los módulos de potencia utilizados en el robot.

Tabla 2.5 Matriz con las características relevantes de los módulos de potencia, así como la evaluación propuesta para su selección

Importancia	Criterio de selección/Solución	Talon SR	RoboClaw 2x30A	RoboClaw 2x15A	RoboClaw 2x7A	Cytron 13 A	OSMC
5	Costo	(50 USD)	(120 USD)	(90 USD)	(60 USD)	(15 USD)	(220 USD)
4	Corriente continua de salida	60A	30A	15A	7A	13A	160A
3	Espacio que ocupa	70x70x30 mm	74x52x17 mm	74x52x17 mm	48x42x17 mm	72x43x10 mm	110x110x80 mm
2	Facilidad de uso	9	10	10	10	9	9
1	Numero de canales	1	2	2	2	1	1

En lo relativo a los sensores, al evaluar su desempeño en el robot FinDER v2 se decidió continuar con los mismos sensores de imagen y el sensor LiDAR. En lo que respecta a los sensores de posición angular, se buscaron alternativas al sensor AS5043, y se encontraron los sensores AS5040 y AS5134, los cuales son parecidos en desempeño al sensor AS5043 para la obtención de la posición absoluta de una junta rotacional, aunque agregan una salida de señal de cuadratura que es útil en el caso de uso de una junta de rotación continua como el sistema de tracción. La desventaja que se encontró es que estos sensores no han sido probados en funcionamiento, además de no contar con un circuito para adaptarlos por lo que había que diseñar la PCB a medida y comprobar su correcto funcionamiento.

En el caso de los sensores de aceleración y posición angular, se comparó el desempeño de ambos respecto a su facilidad de uso y precisión, quedando mejor evaluada la IMU MPU6050.

Finalmente, en la Figura 2.9 se muestra la configuración elegida de sensores y módulos de potencia, y cómo se relacionan con los sistemas mecánicos del robot FinDER v3.

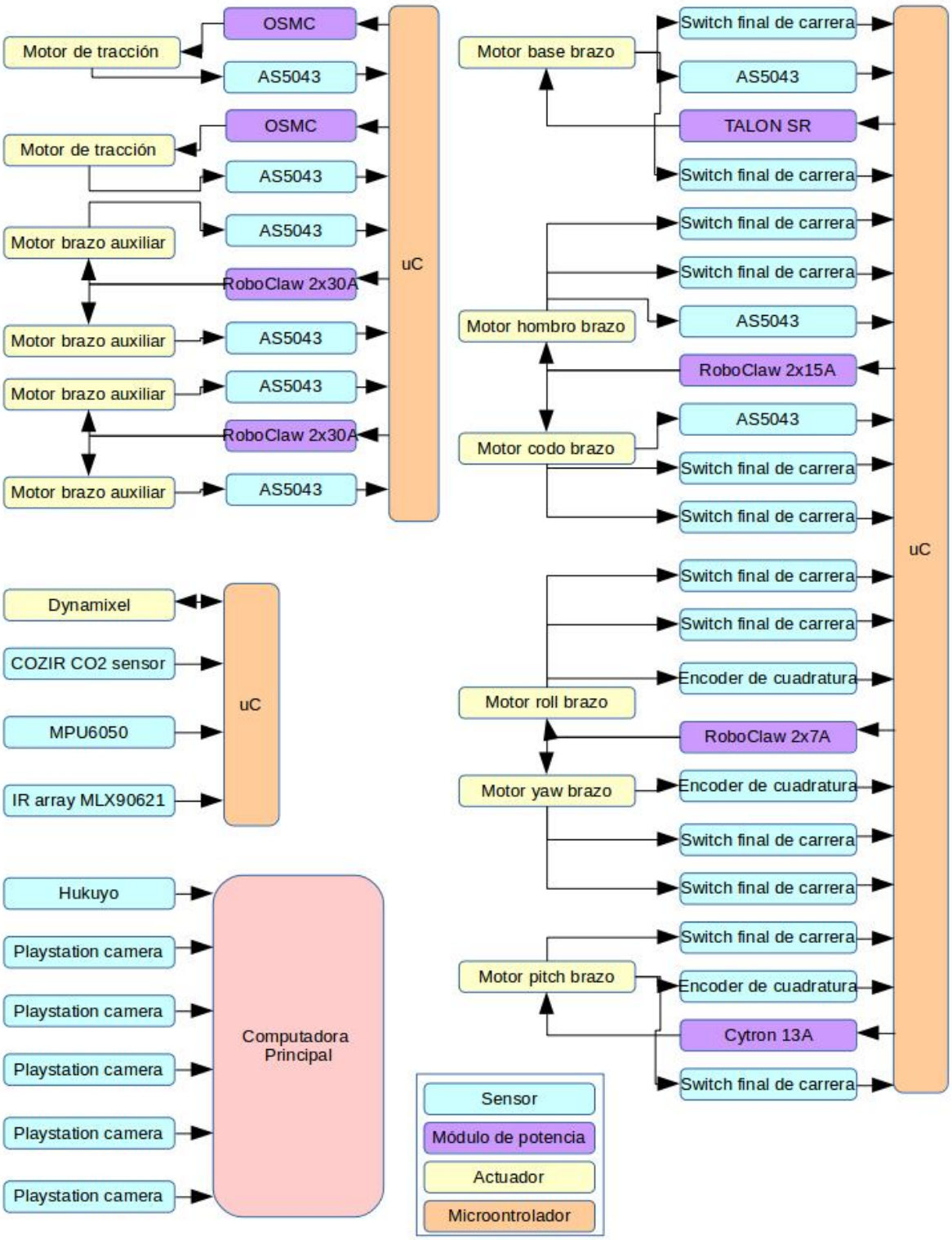

Figura 2.9 Configuración elegida de sensores y módulos de potencia de los motores del robot FinDER v3.

2.3.2 Microcontrolador y API de comunicación serial

Para la selección del microcontrolador se tomó en cuanta el diagrama propuesto en la Figura 2.9, donde se muestra claramente que un criterio importante de selección es el número de pines del microcontrolador y la velocidad del reloj del mismo, ya que a mayor velocidad de reloj se podrán tener más sensores y módulos de control en cada microcontrolador, manteniendo un tiempo de muestreo bajo. Asimismo, el criterio al que se le dio más importancia es la facilidad de programación de acuerdo a la cantidad y accesibilidad de documentación, evaluado en una escala del uno al diez. Los microcontroladores de Arduino, Texas Instruments y la Teensy tienen un IDE ampliamente documentado y una gran variedad de bibliotecas para conectar casi cualquier sensor o módulo de potencia además de que su programación es mediante el lenguaje de programación C++.

En el caso de sistemas embebidos el desarrollo de la parte de programación es mediante algún lenguaje que puede ser Python, C, C++ o algún otro, además de que se hace uso de bibliotecas proporcionadas por terceros para la comunicación con sensores y módulos de potencia.

Tabla 2.6 Matriz de selección del microcontrolador.

Importancia	Criterio de selección/Solución	Raspberry Pi 2B+	Beaglebone Black	Arduino UNO	Arduino Mega	Arduino DUE
4	Facilidad de programación	7	7	9	9	9
3	Costo	(60 USD) 4	(45 USD) 5	(22 USD) 7	(38.50USD) 6	(38.50USD) 6
2	Número de pines	(40 pines) 7	(65 pines) 8	(10 pines) 6	(54 pines) 8	(54 pines) 8
1	Velocidad del reloj	(1GHz) 9	(1GHz) 9	(16 MHz) 6	(16 MHz) 6	(84 MHz) 7
	Suma (Importancia * valoración)	63	68	75	76	77

Importancia	Criterio de selección/Solución	MSP430	Tivac tm4123	TivaC tm4129	Teensy 2.0
4	Facilidad de programación	9	9	9	9
3	Costo	(13 USD) 9	(13 USD) 9	(19 USD) 7	(24 USD) 7
2	Número de pines	(44 pines) 7	(43 pines) 7	(90 pines) 9	(46 pines) 7
1	Velocidad del reloj	(24 MHz) 6	(80MHz) 7	(120MHz) 8	(16MHz) 6
	Suma (Importancia * valoración)	83	84	83	77

Como puede observarse en la Tabla 2.6, el microcontrolador con mejor desempeño bajo los criterios seleccionados es de Texas Instruments TivaC tm4123, aunque por un poco margen sobre el MSP430 y el tm4129. No obstante, esta Tabla podría ser más grande y no asegura que no exista una tarjeta de desarrollo cuyas características sean más apropiadas. Sin embargo, al realizarla se limitó el campo de búsqueda a lo que se aceptaron más documentación y es de fácil obtención, así como a aquellos conocimientos que se tienen a la mano. El universo de tarjetas de desarrollo y de marcas que los comercializan es extenso, por lo que esta búsqueda de soluciones tuvo que restringirse.

En el caso de la API de comunicación serial, que es un conjunto de utilidades y herramientas que permiten implementar la comunicación entre dos sistemas. Por un lado, se tiene el microcontrolador y por otro lado la computadora central del robot. Con esto en mente, se puede describir brevemente cada una de las propuestas. En la Tabla 2.3 se muestran las propuestas de solución y los criterios de selección utilizados.

Rosserial es un paquete de ROS que permite simplificar la conexión de una placa de desarrollo a una computadora y comunicarse directamente con los nodos de ROS que estén siendo ejecutados en ella. Cuenta con comprobación de errores, y está ampliamente documentada.

La biblioteca SerialGateway es, por otro lado, una solución a la comunicación entre microcontrolador y computadora, no simplifica la comunicación con los nodos de ROS que se ejecutan en la computadora, pero ofrece flexibilidad en cuanto al tratamiento de los datos obtenidos del microcontrolador antes de publicarlos en la capa de ROS, por lo que permite disminuir la carga de procesamiento del microcontrolador.

El API de la RoboClaw, por su parte es un programa y una biblioteca de uso exclusivo con los módulos de potencia RoboClaw, ya que este módulo cuenta con un microcontrolador a bordo y funciones avanzadas de control de motores. Se hicieron pruebas de uso se verificó que dicha API no es confiable en cuanto a su utilización, ya que está aún en fase de desarrollo, no obstante, la API de comunicación serial con otros microcontroladores está bastante mejor documentada y probada, además de ser confiable.

Por último, se evaluó la posibilidad de hacer una API de comunicación serial utilizando el lenguaje de programación C++, ya que las soluciones antes mencionadas están desarrolladas en el lenguaje de programación Python, el cual es un lenguaje interpretado, y que por ello fomenta una mayor carga computacional en la CPU de la computadora. Se encontró que es mejor el desarrollo de este programa, no obstante, el esfuerzo de programación es grande y por ello no fue seleccionado, pero en opinión del autor, es la mejor manera de hacer la comunicación serial entre el microcontrolador y la computadora, ya que deja libre recursos de la computadora para otros procesos que hacen uso extensivo de ellos.

Tabla 2.7 Matriz de selección de la API de comunicación serial.

Importancia	Criterio de selección/solución	Rosserial	SerialGateway	Roboclaw API	Crear un nodo en C++
4	Confiabilidad	9	9	8	7
3	Facilidad de programación	9	8	9	7
2	Uso de la CPU	(14%) 7	(10%) 8	(20%) 6	(5%) 9
1	Flexibilidad de uso	7	9	6	9
	Suma (Importancia * valoración)	84	85	77	76

Capítulo 3
Diseño de las tarjetas impresas del robot

Basados en la Figura 2.9 y las Tablas 2.6 y 2.7 se realizaron dos iteraciones para la identificación de las posibles maneras de acomodar los sensores y módulos de potencia, utilizando la tarjeta de desarrollo TivaC tm4123 y la API de comunicación serial SerialGateway para el enlace entre los actuadores del robot FinDER v3 y los nodos de ROS encargados del control que se ejecutan en la computadora principal.

Se hizo uso del IDE Energia para hacer la programación de la tarjeta de desarrollo TivaC (a partir de ahora se referirá al microcontrolador seleccionado como TivaC por facilidad, la tarjeta de desarrollo TivaC tm4123), en la búsqueda de referencias para ello, la proporcionada por el libro "Mastering ROS for Robotic Programming"**[8]** así como por el repositorio de Github**[9]** asociado a los ejercicios del libro, sirvió de gran ayuda y referencia para generar la solución completa al problema.

Cuando en el proceso de prueba de funcionalidad de algún sensor se tuvieron problemas con el programa, se tomó en consideración el uso del entorno de desarrollo (IDE) Code Composer, que es un entorno de programación de microcontroladores de Texas Instruments donde se tienen funciones de monitorización de registros y tiempo de ejecución del programa que se está utilizando para la tarjeta de desarrollo TivaC. Gracias al uso de este IDE, fue posible encontrar problemas incluso en las bibliotecas de propósito general del IDE Energia.

En la parte del API de comunicación serial, quedó establecido el uso de la biblioteca SerialGateway en la computadora como un nodo de ROS escrito en Python y con la capacidad de obtener la información del microcontrolador mediante el protocolo de comunicación serial, así como enviar información a los módulos de potencia para el control de los actuadores del robot.

3.1 API de comunicación Serial

El funcionamiento de esta implementación es descrito a continuación. Basándose en la Figura 3.1, se observa que del lado del microcontrolador se tiene el uso de una biblioteca llamada Messenger, que mediante una abstracción de la comunicación serial creará un objeto que servirá como *handler* (administrador) de toda información que salga o entre por comunicación serial del microcontrolador mediante el periférico especificado de comunicación serial (usualmente es el 0, y está implementada la conversión de serial a USB embebido en la placa de desarrollo). Toda información recopilada por los sensores se enviará a este objeto tipo Messenger, el cual va a enviar la información ordenada y en un formato concreto. De igual manera, toda información recibida por comunicación serial en dicho periférico va a pasar por el objeto tipo Messenger, convirtiendo los datos de entrada a variables utilizables por las funciones de control de los módulos de potencia.

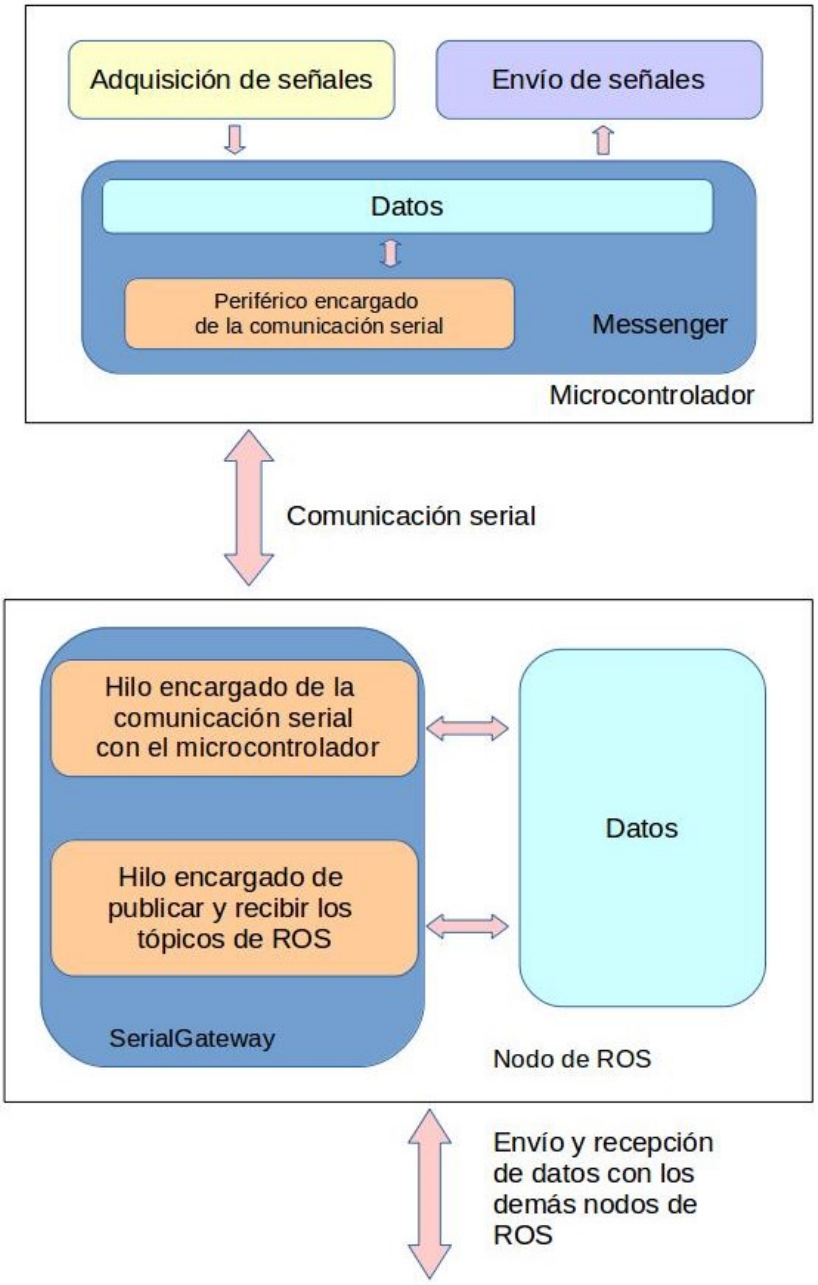

Figura 3.1 Diagrama funcional que describe el esquema de comunicación entre el microcontrolador con la computadora y los nodos de ROS mediante de la API de comunicación serial SerialGateway.

3.1.1 Descripción funcional

Del lado de la computadora, se tiene un nodo de ROS que está escrito con el lenguaje de programación Python y hace uso de la biblioteca SerialGateway; el uso de esta biblioteca está enfocado en hacer un hilo que va a ser ejecutado en segundo plano ocupándose de la comunicación serial, mientras que el hilo principal se encargará de mantener la comunicación con el resto de los nodos que reciben o envían información al microcontrolador apoyado en la API de ROS para Python llamada rospy, la cual permite transformar los datos utilizados dentro de un programa escrito en Python en publicadores,

subscriptores o servicios que se puedan comunicar con otros nodos que también utilicen alguna API de ROS, ya que ello permite el intercambio de información entre programas.
El hilo en segundo plano se encargará de mantener comunicación con el microcontrolador y existirá una transferencia de información entre ambos hilos, mediante las variables que estén asociadas a los valores que se requieren transportar entre el microcontrolador y los nodos de ROS. En resumen, la biblioteca SerialGateway facilita la transferencia de información entre los nodos de ROS que se encargan del lazo cerrado de control de actuadores, además de la presentación al teleoperador de la información pertinente para la operación del robot, y los microcontroladores que hacen la tarea de recolección de datos de los sensores y el envío de señales de control a los módulos de potencia de los actuadores.

3.1.2 Implementación en el microcontrolador

Una vez descrito el funcionamiento básico de la comunicación entre los sensores y módulos de potencia del robot y los nodos de ROS de control que se ejecutan en el robot, conviene describir el funcionamiento de los programas, tanto del microcontrolador como del nodo de ROS que se encarga de la comunicación microcontrolador-ROS. En la Figura 3.2 se muestra el diagrama de flujo del programa.

El programa del microcontrolador consta en forma general, de una estructura unificada, ya que el microcontrolador funciona como un medio para adquirir datos de los sensores y enviar señales de PWM a los puentes H que manejan la etapa de potencia de los actuadores. Con base en ello, en la Figura 3.3 se describe el funcionamiento del programa que está en cada uno de los microcontroladores TivaC que manejan los sistemas del robot. Aunque los sensores o el número de actuadores cambien, la secuencia que se sigue es la misma.

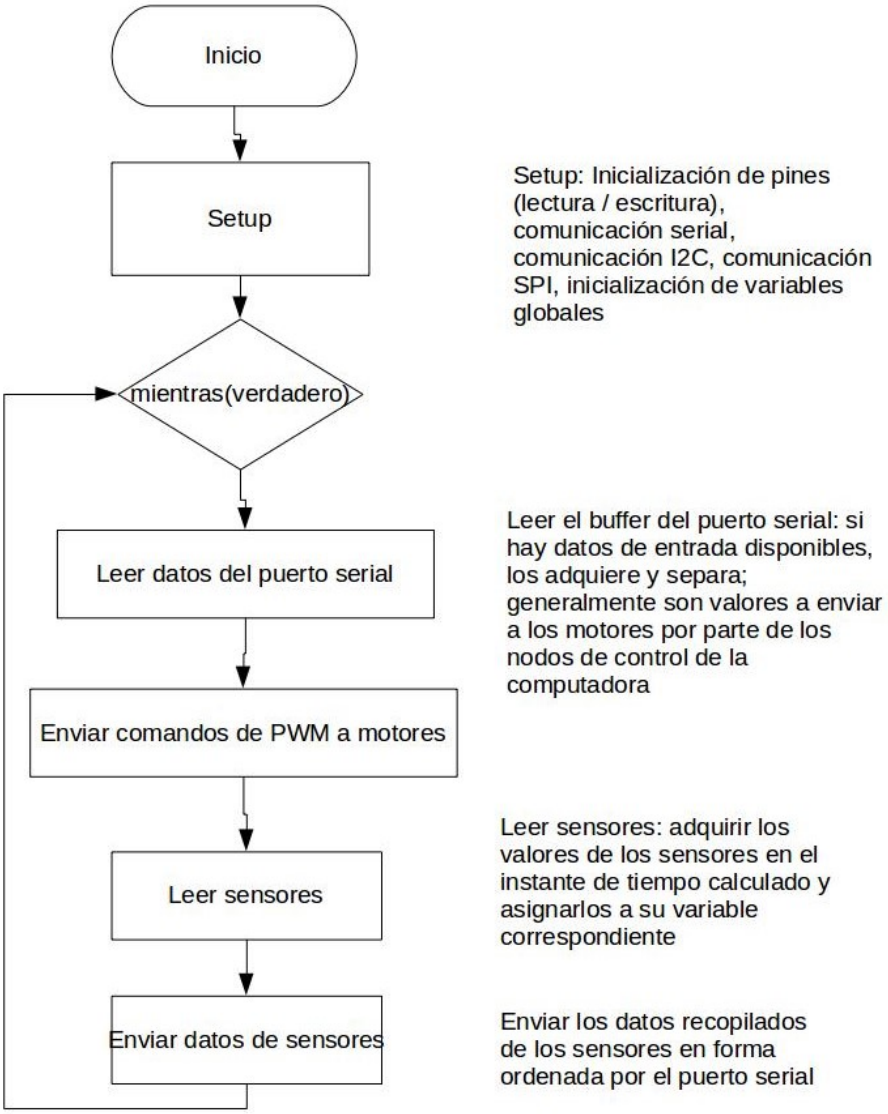

Figura 3.2 Diagrama de flujo general del programa utilizado en el microcontrolador.

3.1.3 Implementación en la computadora usando ROS

El funcionamiento a detalle del nodo de ROS que se encarga de conectar el microcontrolador con los nodos de control en la computadora del robot. Para ello, se observa en la Figura 3.3 en un diagrama de flujo del nodo, que, a simple vista, parece que es un proceso simple, ya que se ejecuta un ciclo infinito, y cuya condición para salir del ciclo es que ocurra un error en la ejecución del programa. La variable 'ROSInterruptException' sirve para este propósito. Y dentro del ciclo, se ejecutan dos procesos de fundamental importancia: el primero, es la ejecución de un método propio de la clase LaunchpadClass, asignado al objeto Launchpad, la clase LaunchpadClass se encarga de obtener la información obtenida en el hilo de comunicación serial, y convertirla y asignarla a variables que se utilizarán por los publicadores o subscriptores que se comunican con el nodo.

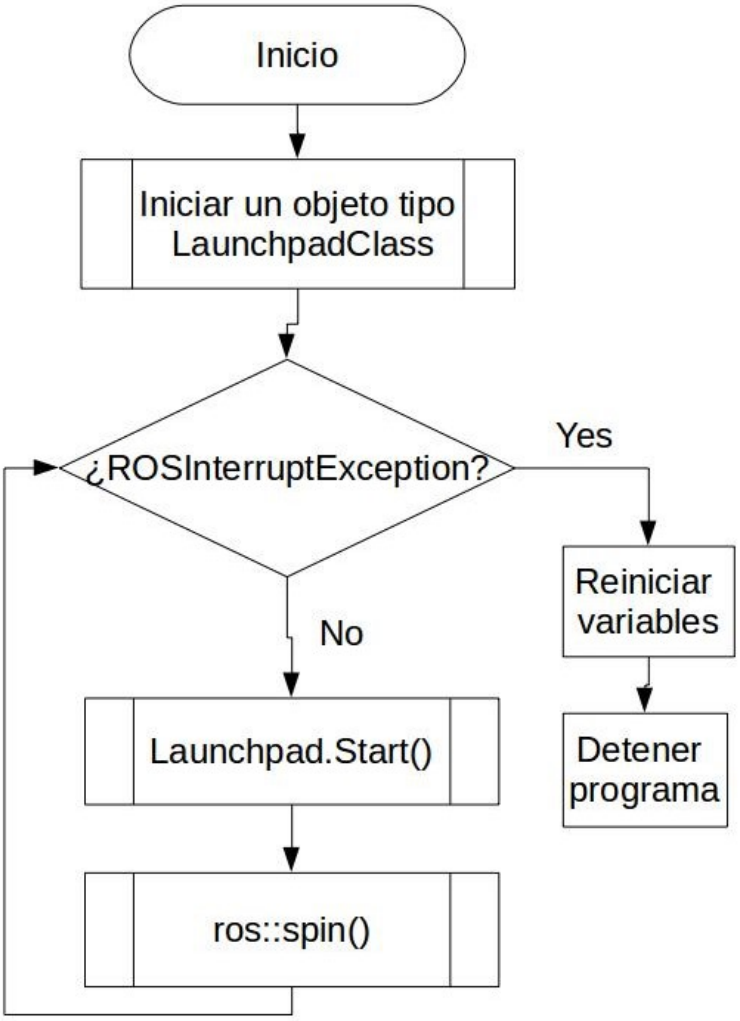

Figura 3.3 Diagrama de flujo del nodo de adquisición y envío de datos al microcontrolador.

El funcionamiento del objeto está más detallado en la Figura 3.4, donde se observa cómo es el funcionamiento interno del método **`Launchpad.start().`** Se observa que hay una entrada al objeto, en este caso la variable 'line', la cual es una variable de tipo cadena en el que se almacenan los datos recibidos por comunicación serial ya ordenados. Se pregunta si dicha variable está vacía, en caso verdadero, se sale de la función, y se sigue la ejecución del programa en la función **`ros::spin`** (ver Figura 3.3); en caso contrario, se discernirá qué tipo de dato es el que ha llegado del microcontrolador por el primer carácter de la cadena 'line'. En caso de ser la letra e, se asignarán los valores de los sensores que estén siendo adquiridos y enviados por el microcontrolador al nodo. En la Figura 3.4, se muestra el diagrama de flujo de un programa llamado **base_launchpad.py,** el cual adquiere la señal de los sensores de posición de las dos ruedas de tracción y si es los cuatro brazos auxiliares. Quiere decir que, si es la letra 'n' como indicador, la información contenida en la cadena 'line' será asignada a variables asociadas a los finales de carrera de los brazos auxiliares, y en caso de ser la letra 'i' quiere decir que la información proviene de la IMU (MPU6050).

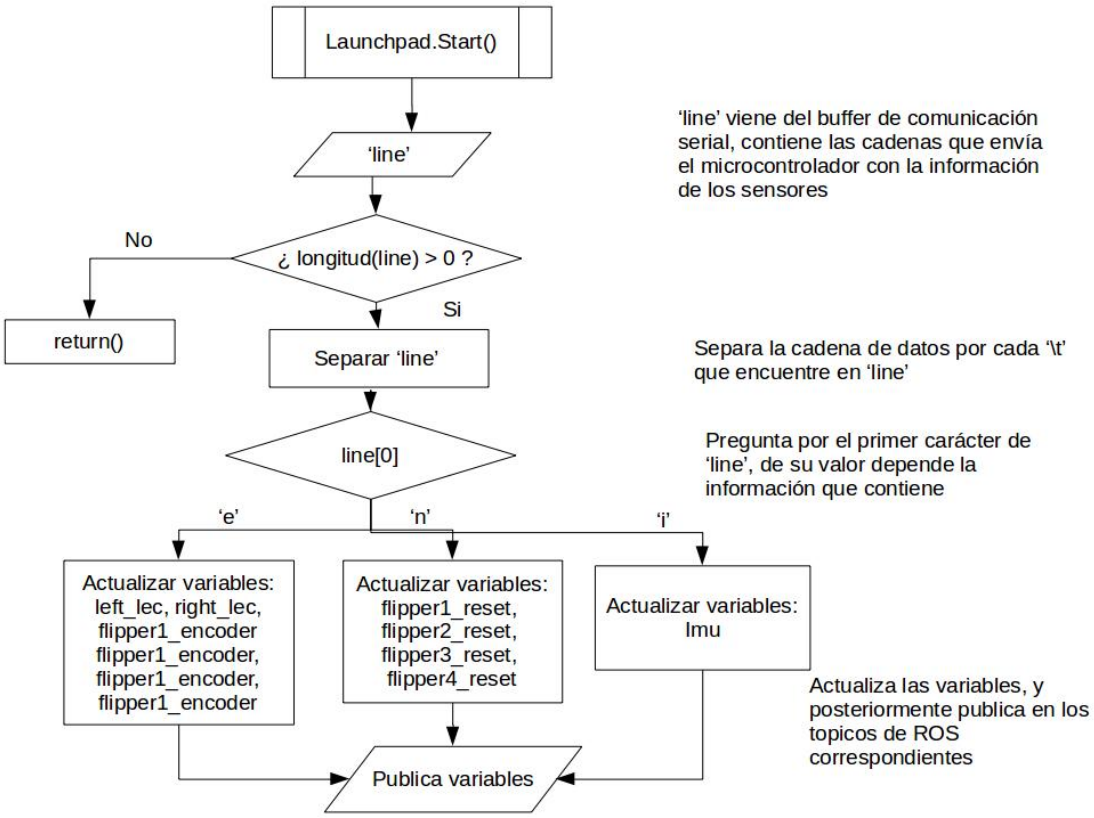

Figura 3.4 Diagrama de flujo de la función `Launchpad.start()` en el nodo base_launchpad.py.

Una vez asignadas las variables correspondientes de acuerdo al contenido de la variable '`line`', se publican las variables en sus respectivos publicadores, de manera que la información fluya hacia los demás nodos que se están ejecutando en el robot.

Hasta este punto, la información va del microcontrolador hacia los nodos de ROS; sin embargo, de los nodos de ROS también fluye información hacia el microcontrolador, de esto se encarga la función **`ros::spin()`** del programa.

La función **`ros::spin`** es la encargada de manejar la información que llega al nodo procedente de otros nodos de ROS en forma de tópicos, asignando a cada tópico de entrada al nodo una función que se encargará de procesar la información contenida en el mismo. Como se observa en la Figura 3.5 de izquierda a derecha, la información entra a las funciones (se les denomina comúnmente *callbacks* o funciones de retorno) que tomarán la información de entrada y la asignarán a variables propias de la clase Launchpad, de forma que esa información pasará de forma ordenada a las variables de tipo cadena '`speed_message"s"`' y a '`speed_message"f"`', conteniendo la información que los nodos de control de ROS quieren enviar a los módulos de potencia del robot a través del microcontrolador.

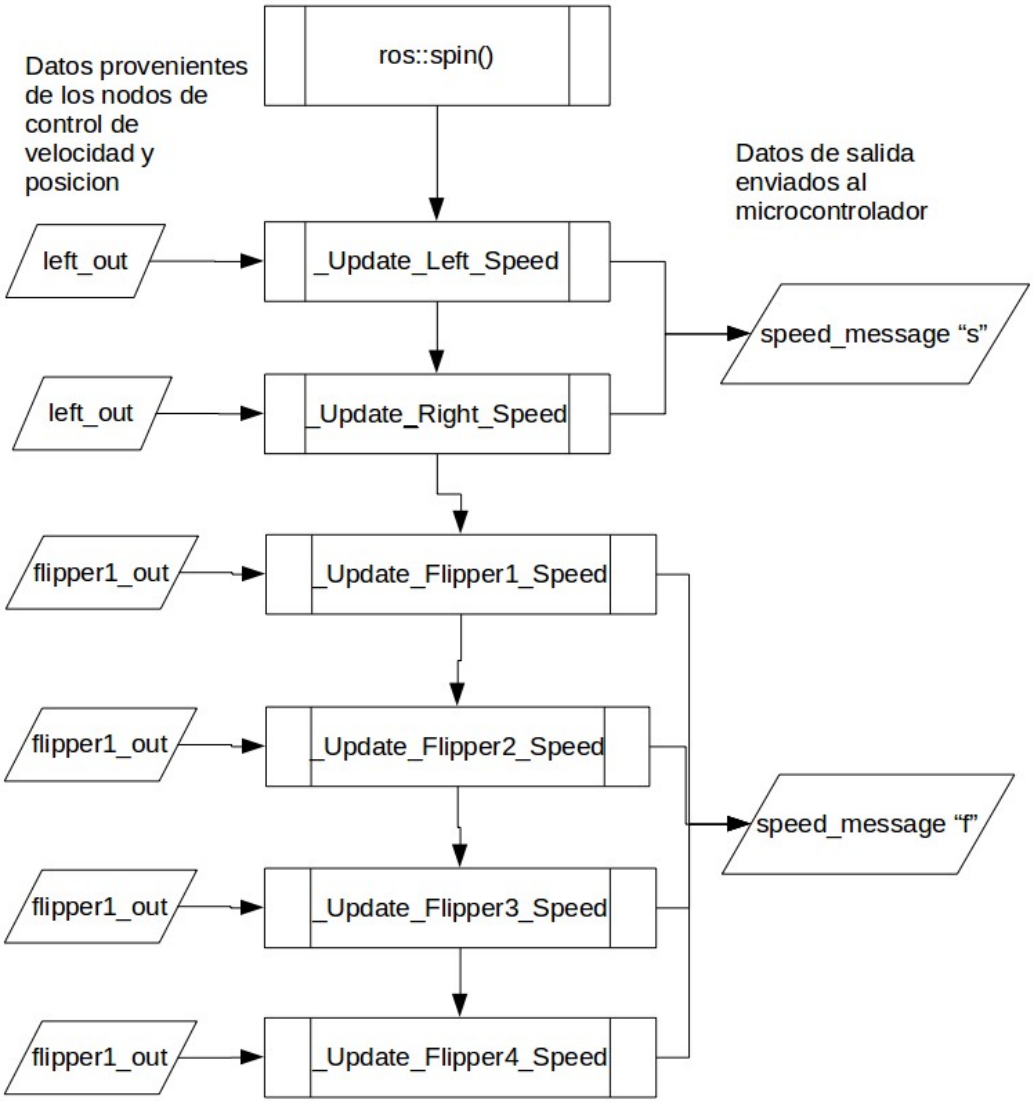

Figura 3.5 Función `ros::spin()` en el nodo base_launchpad.py.

Una vez descrito a detalle la implementación de la comunicación serial y realizadas las pruebas funcionales de los sensores y módulos de potencia del robot, se procedió a separar los mismos de acuerdo a los sistemas del robot que se van a controlar. El robot FinDER v3 se separó en tres subsistemas cuyo flujo de información pasa a través de un microcontrolador TivaC.

El primer subsistema es el de tracción, que comprende las dos orugas de tracción del robot, los cuatro brazos auxiliares, cada uno de estos con su módulo de potencia, sensor de posición angular y un sensor de final de carrera para cada brazo auxiliar, además de la IMU. El segundo subsistema, es el brazo manipulador, que contempla los seis motores de las seis juntas rotacionales del brazo, cada uno con tres sensores; el de posición angular, uno tipo interruptor de inicio de carrera y otro de final de carrera.

Y finalmente, un tercer subsistema, el efector final del brazo, el cual incluye un servomotor Dynamixel, y los sensores de monitoreo de dióxido de carbono CO2 COZIR y el arreglo de sensores infrarrojos MLX90621.

3.2 Método de diseño y prototipado de las tarjetas impresas

Tomando como base la Figura 2.11, se generaron tres bloques para controlar los sistemas de tracción y el brazo manipulador del robot. Tomando en cuenta los tiempos de adquisición de las señales de los sensores y la cantidad de pines de la tarjeta de desarrollo TivaC, se diseñaron en un proceso iterativo las tarjetas impresas de control de cada sistema. En la Figura 3.6 se muestran las placas de la primera iteración, donde se probaron los programas sencillos para la conexión de los sensores y módulos de potencia.

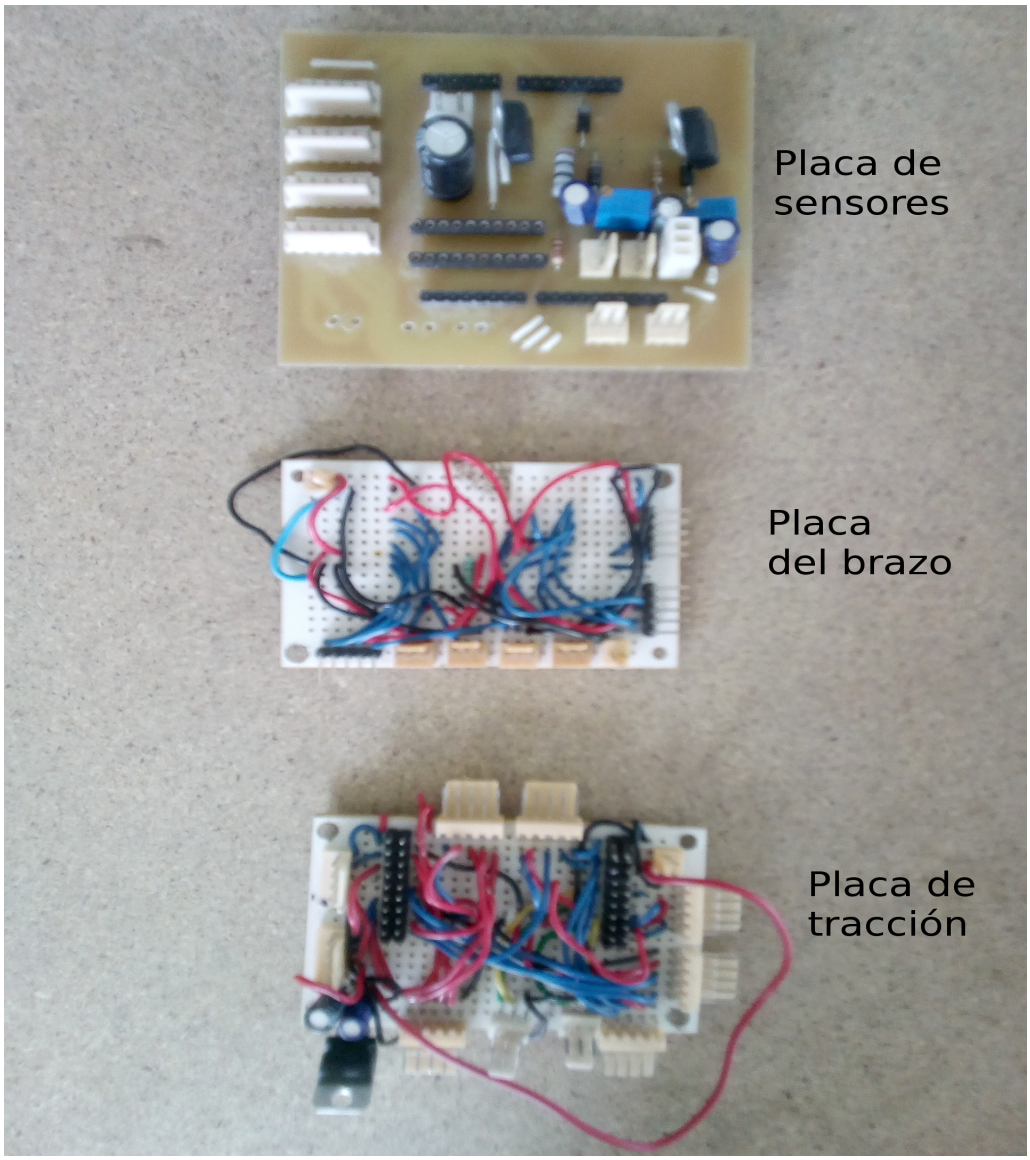

Figura 3.6 Tarjetas impresas de la primera iteración

Como se observa en la Figura 3.6, las primeras tarjetas impresas se hicieron con placas de cobre tipo *breadboard*, cableando manualmente y con conectores Molex para facilidad de corrección de errores en un procedimiento en el que cada una se probó utilizando una TivaC y se hicieron cables con conectores Molex para rapidez de prototipado. En esta etapa, como se mencionó anteriormente, se enfocó en encontrar errores de conexión o del funcionamiento de las bibliotecas de control de periféricos. No se agregaron todos los sensores de final de carrera ya que se consideró un gasto excesivo de tiempo y material; únicamente se hicieron pruebas unitarias de comunicación con los sensores y módulos de potencia.

En la segunda iteración se hizo un primer diseño de las tarjetas impresas utilizando software auxiliar, de la primera iteración se obtuvo información sobre los pines a utilizar del microcontrolador por cada elemento asociado a cada placa y se siguieron utilizando conectores tipo Molex para el cableado. En la figure 3.7 se muestra una fotografía de las tarjetas impresas resultado de este proceso.

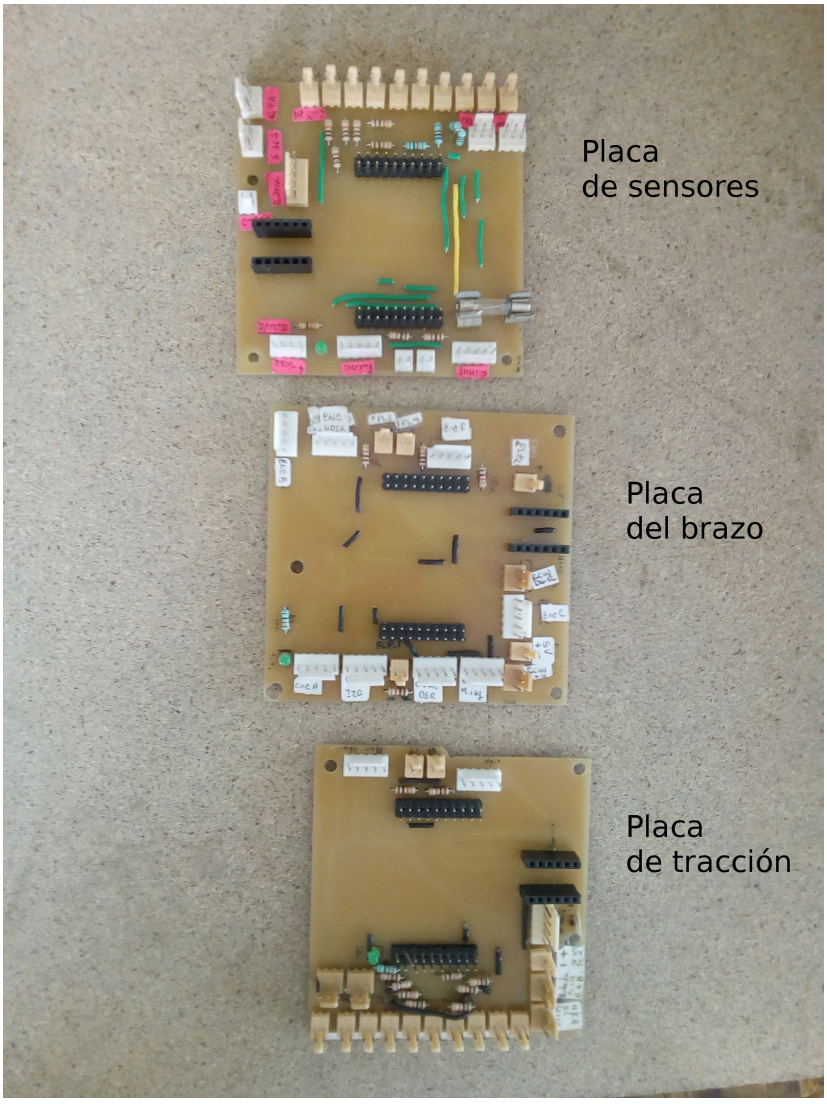

Figura 3.7 Tarjetas impresas de la segunda iteración.

Finalmente, en la tercera iteración se llegó a un esquema de conexión de los elementos del robot y se manufacturaron las tarjetas impresas que se encargan de facilitar el mantenimiento del robot.

Los detalles de esta implementación están en las Figuras 3.8 y 3.9, en las que se observa que se utilizaron conectores tipo BD9, DB25 y DB15 para facilitar la conexión de los elementos como sensores y módulos de potencia, ya que en esta iteración se puso también énfasis en hacer un diseño de fácil acceso a mantenimiento y revisión del estado de las placas. De manera que el cableado se hizo por módulos e influyó en el diseño de las placas. Los detalles del diseño y esquemáticos generados en la última iteración se encuentran en el Apéndice B.

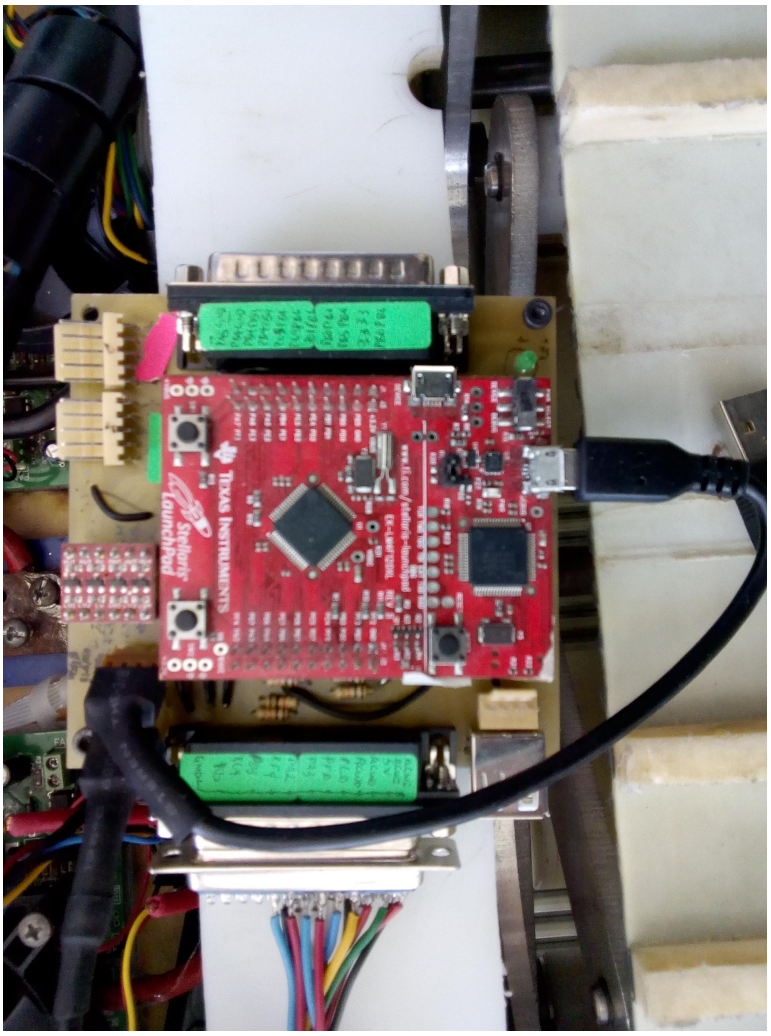

Figura 3.8 Placa de tracción final.

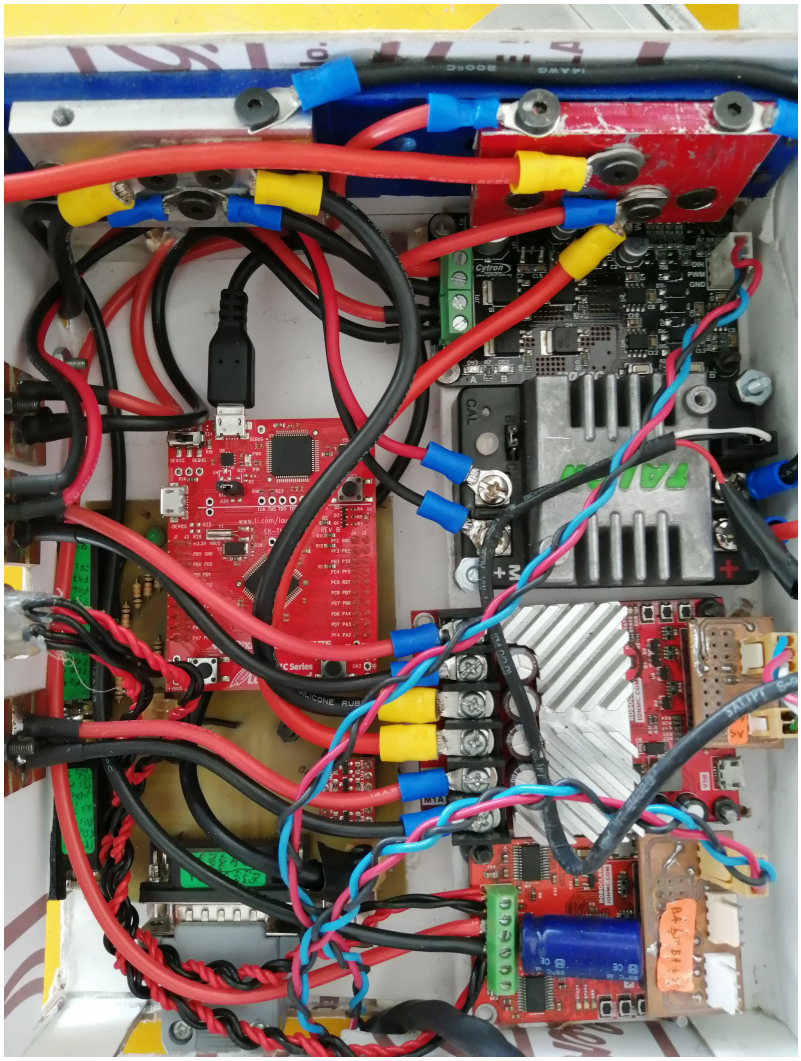

Figura 3.9 Placa del brazo final.

Capítulo 4
Software de control de bajo nivel

Una vez descrita la manera en que la información viaja entre los sensores, módulos de potencia y la computadora, y más específicamente entre los nodos de ROS, este capítulo se destinó a describir el funcionamiento de los programas o nodos que permiten que un teleoperador controle los mecanismos de tracción y manipulación del robot.

A partir de lo descrito en el capítulo anterior, se diseñaron tres tarjetas impresas para distribuir los sensores y módulos de potencia en los que se consideraron los tres subsistemas del robot: el sistema de tracción, el sistema del brazo manipulador y el sistema del efector final.

La Figura 4.1 muestra el diagrama de bloques simplificado del sistema de tracción del robot FinDER v3. Se debe leer de izquierda a derecha e indica que existe un teleoperador del sistema; dicho bloque está en color morado que significa la existencia física del elemento. En este caso, es el control de Xbox y el teleoperador que lo utiliza para controlar a distancia el robot.

Para describir el sistema de control que se desarrolló, se debe tener en cuenta que existen como antecedente los programas desarrollados para el robot FinDER v2, así como programas desarrollados por terceros en otros proyectos de programación de robots móviles, guías de desarrollo de programas y sugerencias en la amplia documentación existente para el desarrollo de programas en ROS. Partiendo de ello, gran parte de los programas descritos a continuación son una adaptación de dichos programas, de tal manera que dan al robot la funcionalidad deseada y, además, bajo el esquema propuesto, el desarrollo de tareas de nivel intermedio y alto mediante paquetes de ROS ya existentes es posible sin tener que hacer modificaciones.

El problema que se plantea en forma general es la teleoperación de los sistemas mecánicos del robot mediante un control de Xbox: para ello, ya existe un paquete de ROS que convierte la información de las palancas y botones del control a valores numéricos y booleanos. En la Figura 4.2 se muestra cómo es esta conversión. De la Figura 4.1 se tiene

la información de los sistemas que se van a controlar mediante cada placa de desarrollo TivaC, y en la sección anterior se describió el funcionamiento de la comunicación entre los bloques de TivaC y el nodo de adquisición y envío.

Para las tres tarjetas impresas desarrolladas, las entradas son las señales del control de Xbox, y las salidas, son las señales de control que se convierten en flujo de corriente en los motores mediante los módulos de potencia, y de regreso de los mecanismos del robot, se tienen las señales de los sensores de posición, sensores de aceleración, giroscopio y magnetómetro, sensor de dióxido de carbono y sensor térmico.

Partiendo del objetivo que es la teleoperación se identificaron los siguientes subproblemas:

- El usuario (teleoperador) debe de accionar alguna palanca o combinación de botones para dispar un comportamiento en algún actuador del robot.
- El sistema de tracción debe moverse con la cinemática de un robot de par diferencial, ya que con ello puede ser compatible por paquetes de navegación de ROS.
- El movimiento de todas las partes móviles debe estar realimentado, formando un controlador en lazo cerrado que permita un movimiento suave del actuador.
- La información proveniente de los sensores dedicados a la detección de víctimas debe llegar al teleoperador en forma entendible

4.1 Esquemas de conexión de nodos

Si se observan las Figuras 4.1 a 4.3 con detenimiento, corresponden con lo descrito en el capítulo anterior sobre la arquitectura de los sensores, actuadores, el programa del microcontrolador y el nodo de ROS encargado de adquirir y enviar datos entre el microcontrolador y los nodos de control del robot, que llegan finalmente al teleoperador. Los bloques en color azul son aquellos programas que se emplean para resolver los subproblemas recién planteados.

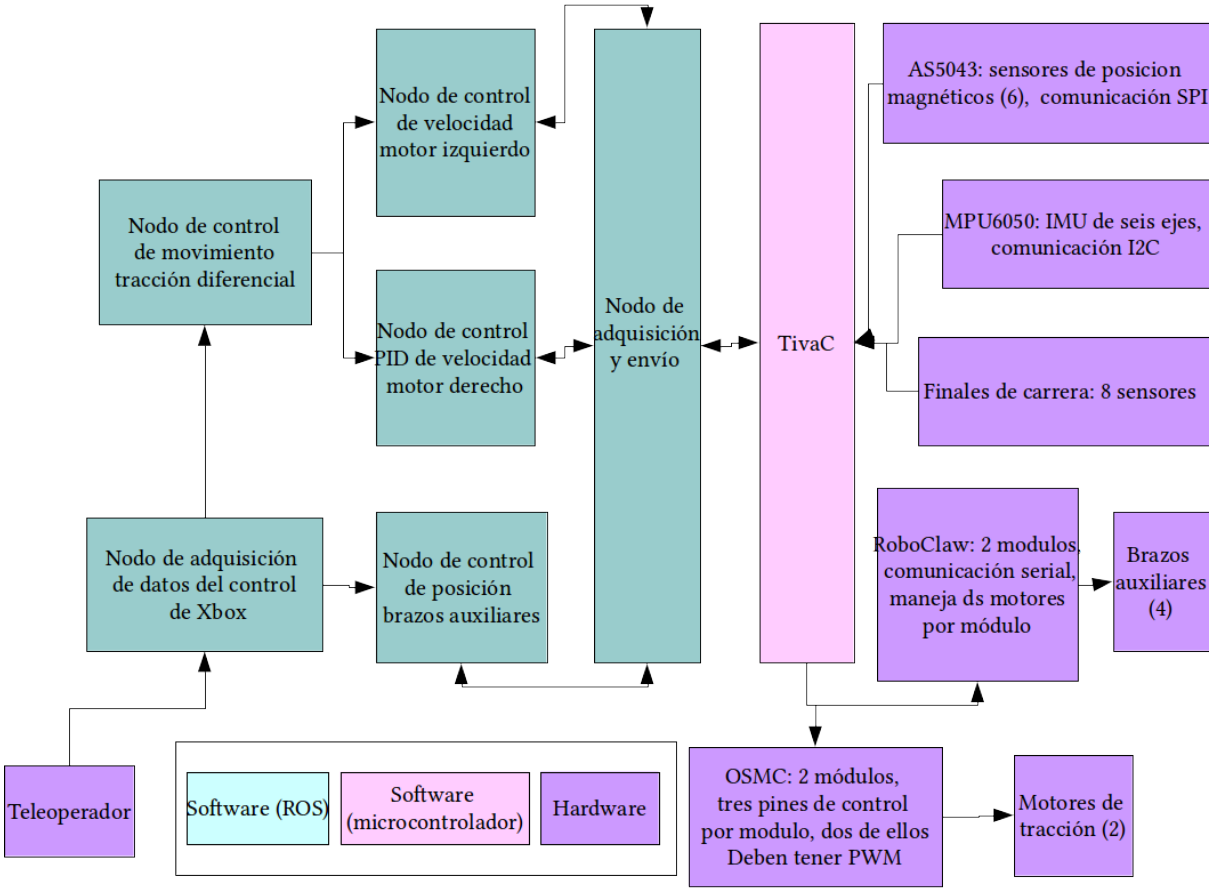

Figura 4.1 Esquema de los nodos de ROS conectados a la tarjeta impresa de tracción.

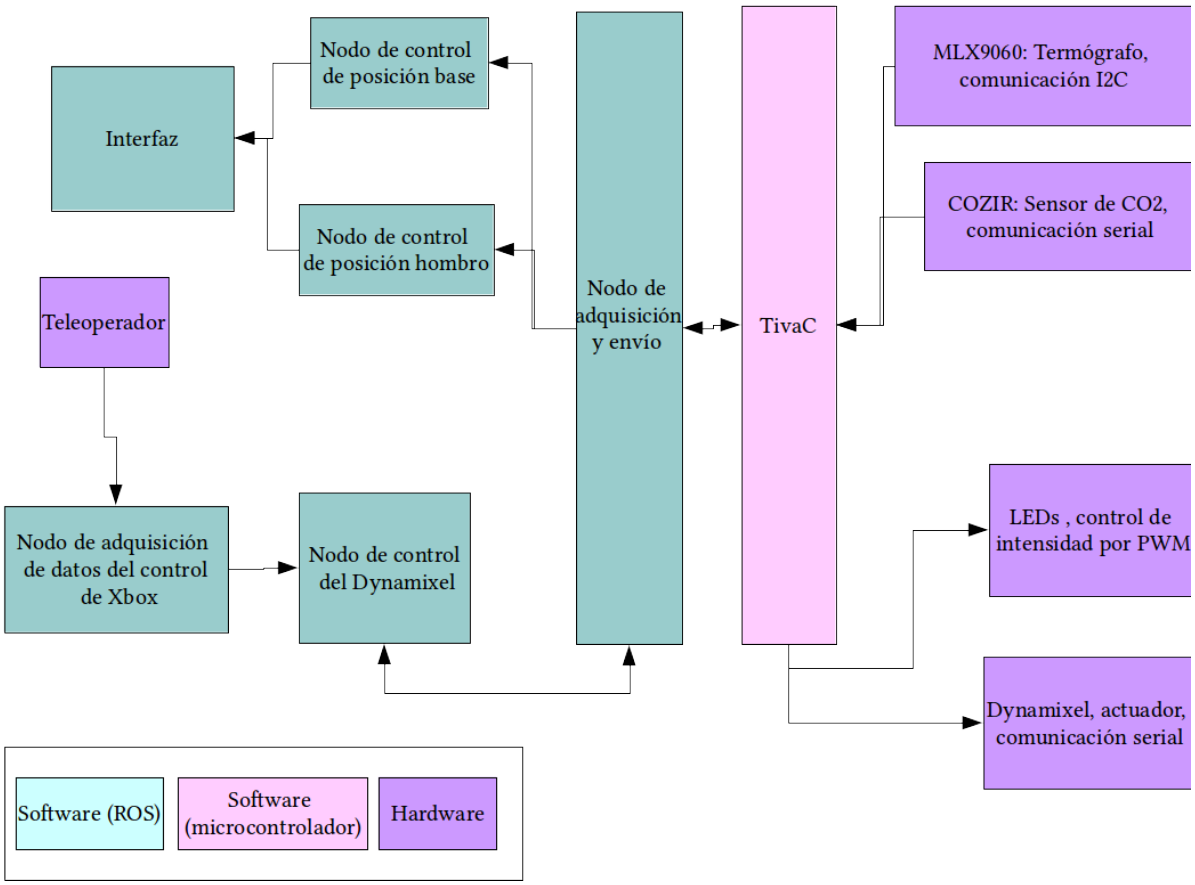

Figura 4.2 Esquema de los nodos de ROS conectados a la tarjeta impresa del efector final.

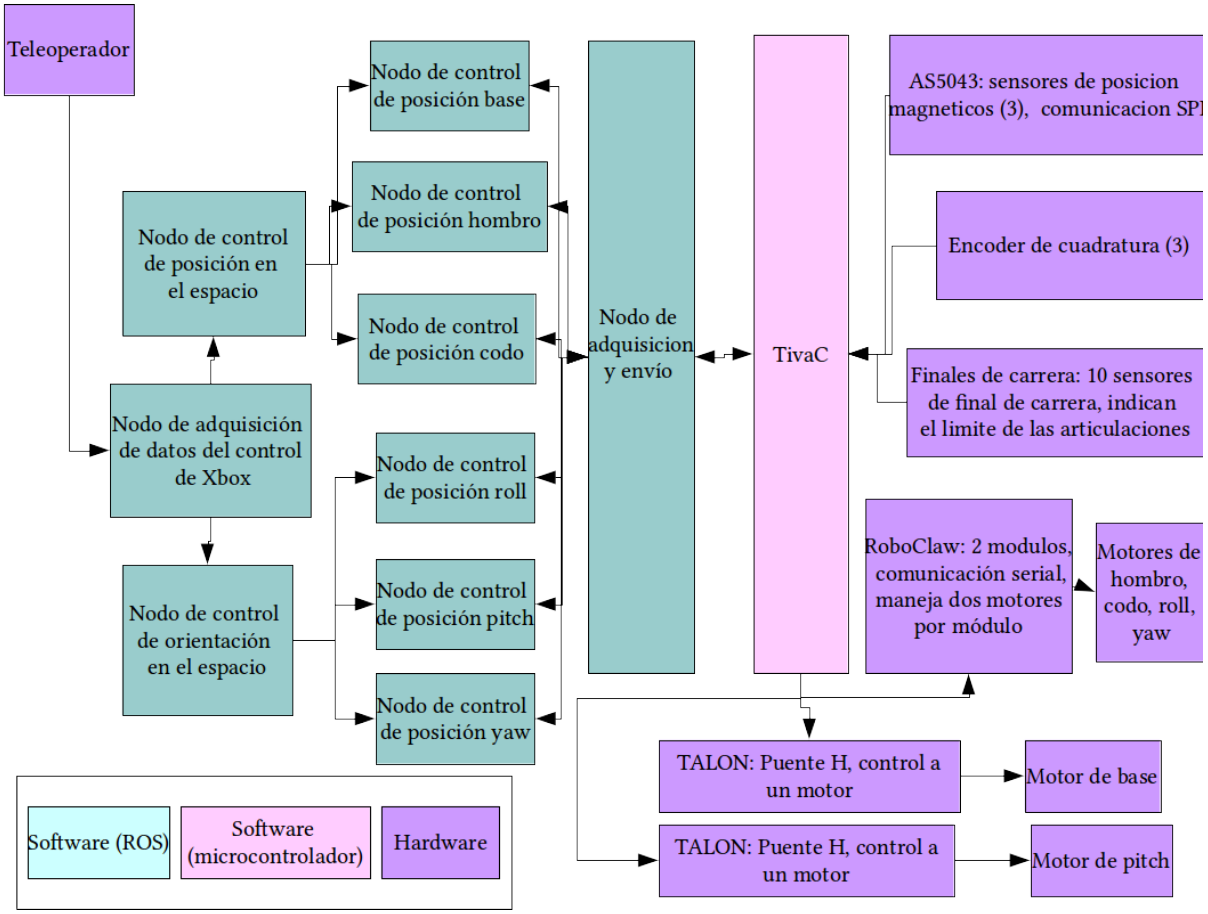

Figura 4.3 Esquema de los nodos de ROS conectados a la tarjeta impresa del brazo manipulador.

Para facilitar la implementación de los esquemas de las Figuras 4.1 a 4.3, existe ya un diagrama de control de sistemas robóticos que se ha generado a partir de buenas prácticas de desarrollo de un proyecto utilizando ROS (ver Figura 4.4). Existe un esquema de entradas y salidas ya desarrollado por la comunidad de ROS para realizar la conexión de un sistema robótico como es una base robótica móvil de tracción diferencial o un brazo robótico de n grados de libertad con ROS. Se puede ver en la Figura 4.4 cómo se ha organizado y separado los diferentes niveles de desarrollo de programación en el robot, partiendo del sistema físico, vemos como se conecta a la computadora del robot y en la capa de bajo nivel que denotan como “ros_control & friends” se tiene un esquema de cómo se controlan los sistemas de un robot. El bloque “RobotHW” engloba los nodos de control de lazo cerrado de los actuadores del robot, mientras que los bloques “base_controller” y “arm_controller” son aquellos que generan instrucciones para el movimiento coordinado de sistemas que se supone operen de dicha manera. Por ejemplo, el bloque que controla la base móvil del robot debe ser capaz de generar, por lo menos, dos comandos de velocidad (en la Figura 4.4, los círculos de color rojo indican controladores de velocidad) para que las dos ruedas de tracción de la base móvil se desplacen al mismo tiempo y con la misma velocidad, o de forma coordinada para generar un movimiento de giro. Asimismo, podría ser una base móvil con tres ruedas de tracción, en dichos casos, los comandos de salida serían tres, de tal forma que pueden generar el movimiento deseado de la base móvil. La misma funcionalidad se puede atribuir al bloque “arm_controller”.

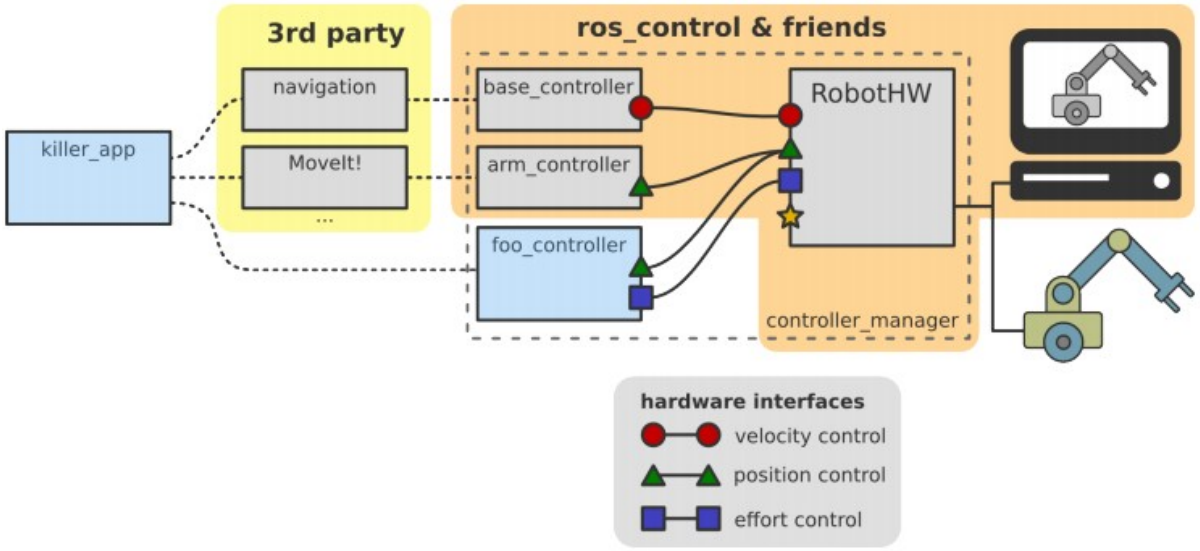

Figura 4.4 Esquema de conexión de elementos en ROS **[10]**.

Se puede observar que el bloque 3rd party (software de terceros) es el que envía comandos a los bloques antes mencionados. Este bloque engloba lo que son paquetes que generan comportamientos de alto nivel; el paquete *Navigation stack* de ROS es capaz de generar navegación autónoma de una base móvil robótica a partir de mapas estáticos de un lugar conocido. Y el paquete *MoveIt!* en el otro bloque se encarga de generar trayectorias y realizar las tareas complejas de anti-colisión y cinemática inversa para brazos manipuladores robóticos.

A partir de lo descrito anteriormente, se tiene una idea de cómo deben ser las entradas y salidas de los nodos intermedios entre el nodo joy que adquiere la señal del control de Xbox, y los controladores en lazo cerrado de los actuadores del robot. Los bloques "arm_controller" y "base_controller" deben ser implementados en el robot de tal forma que su entrada pueda provenir de los paquetes de *MoveIt!* o N*avigation stack* de ROS, así como del nodo joy que obtiene las señales que un teleoperador envía mediante los botones y palancas de un control tipo joystick, de tal manera que la arquitectura del robot pueda crecer mediante paquetes de terceros, como lo indica la Figura 4.4, manteniendo la funcionalidad de que un teleoperador pueda intervenir en la manipulación de los sistemas de base móvil y brazo manipulador del robot.

4.2 Teleoperación de la base de tracción

Para esto también se tomó en cuenta el código ya existente del robot FinDER v2 y el libro "Mastering ROS for Robotics Programming", en el cual se propone una solución a este esquema de conexión para un robot existente llamado Chefbot. En esta solución, mostrada en la Figura 4.6, se muestra el flujo de información a partir del nodo de teleoperación joy, de manera que el control de los sistemas de la base móvil, los brazos auxiliares y el brazo manipulador son controlados de tal forma que serán compatibles con paquetes de terceros, como *MoveIt! , Navigation Stack, Hector SLAM*, entre otros paquetes de gran uso que aprovechan la arquitectura descrita en la Figura 4.4.

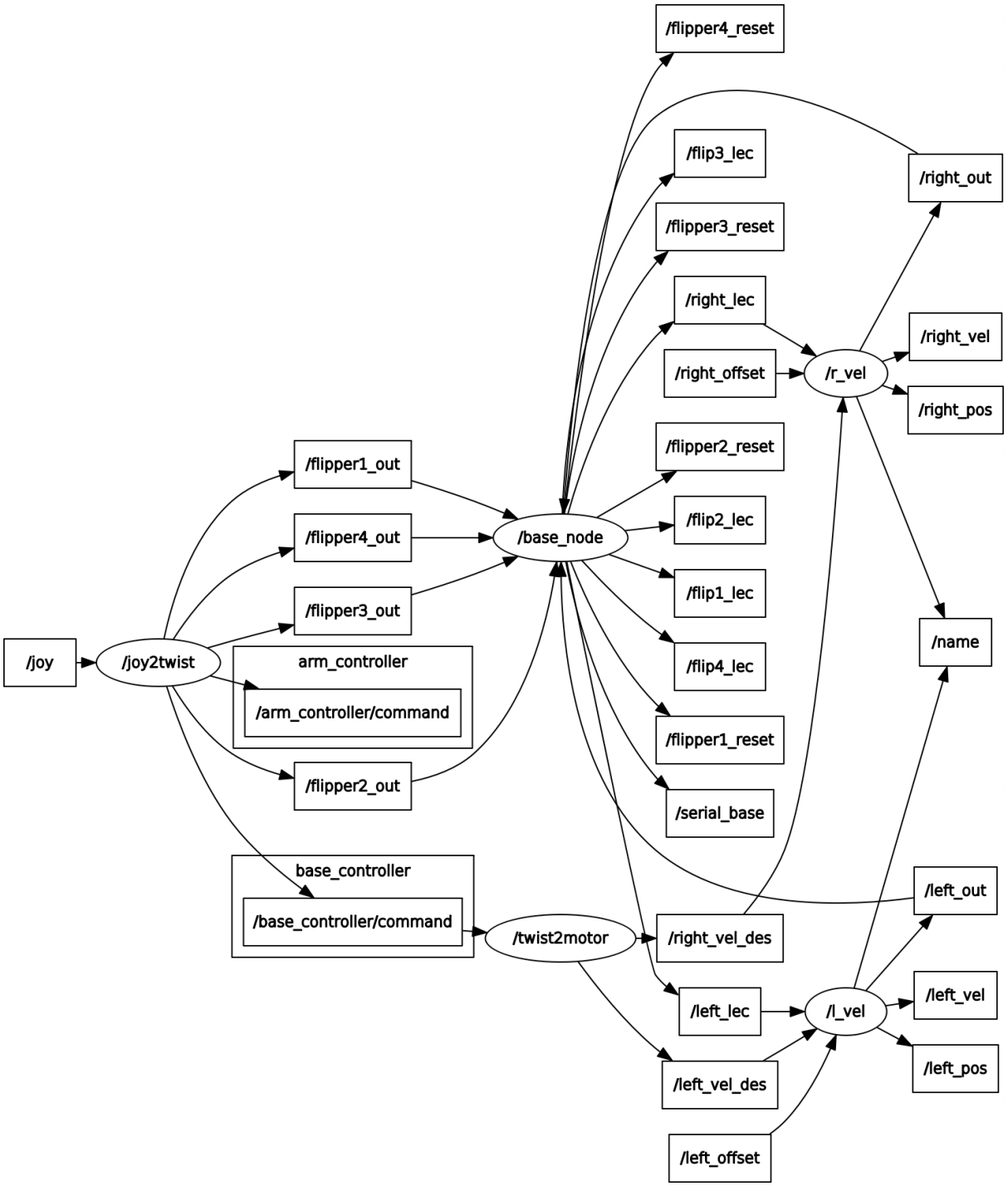

Figura 4.5 Conexión de nodos en ROS para el control de la base móvil.

La Figura 4.5 muestra los nodos de ROS, y los tópicos que se están publicando y recibiendo para hacer el control de la base móvil del robot FinDER v3. Los elementos dentro de una elipse denotan un nodo y los que están en un rectángulo son tópicos; las flechas indican en qué dirección la información de un tópico viaja. Por ejemplo, partiendo desde la izquierda, se tiene el nodo joy_node, el cual publica el tópico `/joy`, y se verifica que el nodo joy2twist se suscribe al tópico `/joy` y publica los tópicos: `/base_controller/command`, `/arm_controller/command`, `/flipper1_out`, `/flipper2_out`, `/flipper3_out` y `/flipper4_out`.

Se entiende que los primeros dos se encargan de controlar la base móvil y el brazo manipulador, mientras que los otros cuatro son tópicos para controlar los brazos auxiliares del robot de forma individual. Siguiendo el tópico `/base_controller/command`, llega a un nodo llamado twist2motor; dicho nodo se encarga de publicar los valores de velocidad deseada para las orugas de tracción del robot. Estos valores son `/left_vel_des` y `/right_vel_des`, que alimentan a los nodos l_vel y r_vel. Al observar en detalle los tópicos que publican y reciben estos nodos, se observa que son iguales, que reciben el valor deseado y los valores de los sensores de posición angular de las orugas del robot, así como que publican la velocidad del actuador y una señal de control `/right_out` y `/left_out`, que en el microcontrolador se traducirá en la modulación de la señal de PWM que se envía al módulo de potencia que controla dicho actuador.

Si se compara la Figura 4.5 y la Figura 4.1, podemos darnos cuenta que los bloques de color azul tienen un correspondiente nodo o nodos que se encargan de realizar la función descrita por los bloques mencionados. Por ejemplo, se tiene el nodo joy que corresponde al nodo de adquisición de datos del control de Xbox, los nodos joy2twist y twist2motor corresponden al bloque llamado nodo de control de movimiento tracción diferencial; los nodos l_vel y r_vel corresponden a los bloques nodo de control de velocidad motor izquierdo y nodo de control de velocidad motor derecho.

Como se puede observar en las Figuras 4.1, 4.2 y 4.3, el bloque que se encarga de la adquisición de la información del control que utiliza el teleoperador se conecta normalmente a otro bloque, antes de los bloques de control de los actuadores. En la Figura 4.1 dicho bloque se llama nodo de control de movimiento de tracción diferencial y otro que dice nodo de control de posición de brazos auxiliares. Ambos se caracterizan por estar entre el nodo de adquisición de datos del joystick y los nodos de control de los actuadores.

Asimismo, en la Figura 4.3 se tienen dos bloques, el nodo de control de posición en el espacio y el nodo de control de orientación en el espacio. También, en la Figura 4.2 se tiene nodo de control del Dynamixel. De todo esto se debe de entender que los datos que se obtienen del nodo joy deben de recibir un tratamiento y traducirse en comandos que sean útiles para el control de los sistemas del robot. Por ello, los bloques intermedios tienen importancia ya que en ellos se está haciendo una conversión de información, de un comportamiento deseado por el teleoperador a comandos de velocidad o posición, que son las entradas a los nodos de control de los actuadores del robot.

4.2.1 Nodo joy

En cada uno de los esquemas de teleoperación se tiene en cuenta un programa dedicado a la adquisición de datos del control de xbox; afortunadamente para este problema ya existe un paquete de ROS que se encarga de realizar esta tarea, dicho paquete se conoce como *joy* **[11]**. Este paquete, se encarga de adquirir la información de los dispositivos que el sistema operativo reconoce como tipo joystick y convierte dicha información a un tipo de dato compuesto de ROS. Publica dicha información en un mensaje tipo “Joy” el cual consta de tres partes, un encabezado que contiene un timestamp (información de fecha y hora) del momento en que ocurre un cambio en el estado del control, un arreglo de valores tipo

flotante (float32) que contiene los valores de los ejes de las palancas del control, y un arreglo de valores tipo entero (int32) que contiene los valores asociados a los botones del control. En las Tablas 4.1 y 4.2 se muestran los valores que se publican a partir de la manipulación del control de Xbox, y en la Figura 4.4 se muestran lo correspondientes botones, palancas y ejes de movimiento del control de Xbox.

Tabla 4.1. Índice de los ejes de los botones del control

Índice	Nombre del botón	Descripción
0	A	0 ó 1
1	B	0 ó 1
2	X	0 ó 1
3	Y	0 ó 1
4	LB	0 ó 1
5	RB	0 ó 1
6	Back	0 ó 1
7	Start	0 ó 1
8	Guide	0 ó 1
9	Botón palanca izquierda	0 ó 1
10	Botón palanca derecha	0 ó 1

Tabla 4.2. Índice de los ejes de las palancas del control

Índice	Nombre del eje en el control	Descripción
0	Eje izquierda/derecha palanca izquierda	-1 … 0 … 1 valores decimales
1	Eje arriba/abajo palanca izquierda	-1 … 0 … 1 valores decimales
2	Eje izquierda/derecha palanca derecha	-1 … 0 … 1 valores decimales
3	Eje arriba/abajo palanca derecha	-1 … 0 … 1 valores decimales
4	RT	0 … 1 valores decimales
5	LT	0 … 1valores decimales
6	D-Pad Izquierda/Derecha	-1 0 1 valores enteros
7	D-Pad Arriba/Abajo	-1 0 1 valores enteros

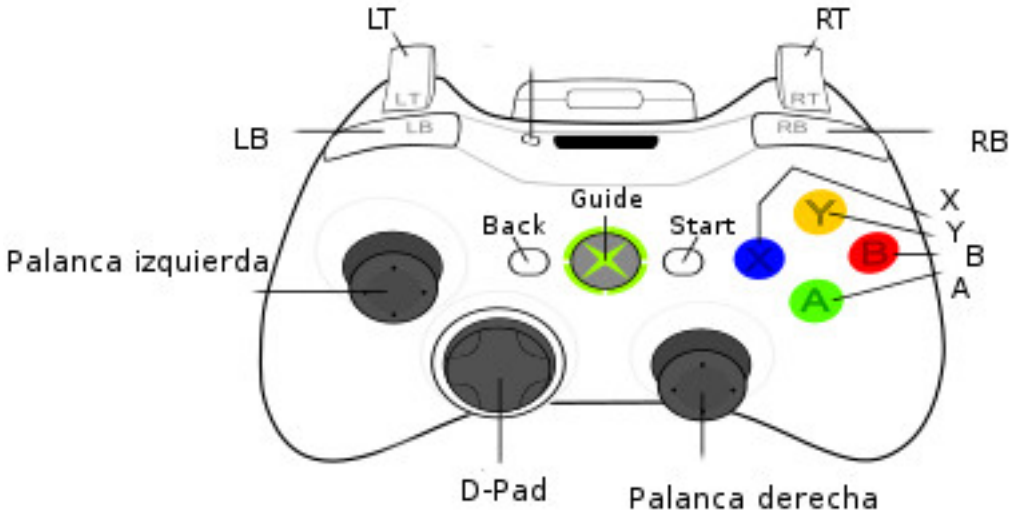

Figura 4.6 Botones y palancas (ejes) del control de xbox

4.2.2 Nodo joy2twist

El nodo joy2twist tiene como entrada el tópico que publica el nodo joy, y tiene como salidas los tópicos `cmd_vel_pub`, `cmd_arm_pub`, `cmd_vel_fl1`, `cmd_vel_fl2`, `cmd_vel_fl3` y `cmd_vel_fl4`. El tópico `cmd_vel_pub` es aquel que significa el avance de la base de tracción del robot en el espacio con una componente de velocidad de avance lineal y una componente de avance en rotación. El tópico `cmd_arm_pub` por su parte, indica la velocidad de movimiento del brazo manipulador en el espacio mediante desplazamientos en coordenadas euclidianas (ejes XYZ) y los respectivos giros sobre cada uno de los ejes para representar el movimiento del efector final. La decisión de hacerlo así es por un concepto conocido como desacople cinemático, utilizado en la obtención de la cinemática inversa de un brazo manipulador.

Una vez definidas las entradas y salidas del nodo, se hizo un análisis de los pasos a seguir para completar la tarea del nodo. Para ello se definieron los siguientes pasos:

- **Leer el tópico de entrada /joy**
- **Obtener la información relevante del tópico joy** y asignar variables, de manera que los valores de las palancas y botones del control se utilicen de acuerdo a lo planeado por el diseñador
- **Asignar los valores de las palancas o botones a los tópicos de salida.** Esto se puede hacer directamente pero no es recomendable, se pueden mapear los valores de las palancas a valores que al llegar al microcontrolador sean utilizables por el módulo de potencia que controla un actuador, o en caso de que el movimiento deseado sea una combinación de dos o más actuadores, se asigna a un tópico de tipo Twist, que pasará a otra etapa de procesamiento (cinemática inversa) antes de llegar al microcontrolador.
- **Publicar los tópicos de salida.**

En la Figura 4.6 se muestra el subscriptor al tópico tipo joy que genera el nodo de adquisición de datos del control de Xbox, y los publicadores son los dos mensajes tipo Twist que manejan la tracción, el brazo y los mensajes tipo Int16 que mueven los flippers.

```
//Entrada del nodo (suscriptores)
joy_sub=n.subscribe("/joy",10,&RobotDriver::joyCallback,this);

//Salida del nodo (publicadores)
  cmd_vel_pub=n.advertise<geometry_msgs::Twist>
                                    ("/base_controller/command",1);
cmd_arm_pub=n.advertise<geometry_msgs::Twist>("/arm_controller/comma
nd",1);

cmd_vel_fl1=n.advertise<std_msgs::Int16>("flipper1_out",1);
cmd_vel_fl2=n.advertise<std_msgs::Int16>("flipper2_out",1);
cmd_vel_fl3=n.advertise<std_msgs::Int16>("flipper3_out",1);
cmd_vel_fl4=n.advertise<std_msgs::Int16>("flipper4_out",1);
```

Figura 4.6 Entradas y salidas del nodo joy2twist.

Este nodo tiene dos partes importantes, una es la función asociada al subscriptor `joy_n`, llamada **`joyCallback( )`**, que se encarga de asignar los datos que envía el nodo joy a las variables internas del nodo que se ocupan para generar los datos de salida del nodo en los publicadores; esta función se encarga de hacer los dos primeros puntos de la lista de acciones del nodo.

La otra función importante es **`Update( )`**, la cual es la que realiza el tercer y cuarto paso de la lista; el procesamiento de la información del nodo joy y la transforma a los mensajes tipo Twist que van al sistema de tracción, al brazo y a los mensajes tipo Int16 que van a los brazos auxiliares. En el fragmento de código a continuación, se muestra cómo se convierten los datos de la palanca izquierda que corresponden a dos coordenadas en forma cartesiana en coordenadas polares.

```
//Conversión a coordenadas polares
temp_linear=sqrt(joy_h*joy_h+joy_v*joy_v);
temp_angular=atan2(joy_v,joy_h)*180/PI;
//Acotación del movimiento
if(temp_linear>0.2 && temp_angular>=0)
{base_cmd.angular.z=atan2(joy_v,joy_h)-PI/2;}
if(temp_linear>0.2 && temp_angular < 0)
{base_cmd.angular.z=atan2(joy_v,joy_h)+PI/2;}
if(temp_linear <=0.2)
{base_cmd.angular.z=atan2(joy_v,joy_h);}
if(temp_angular<0 )
{base_cmd.linear.x=-sqrt(joy_h*joy_h+joy_v*joy_v);}
if(temp_angular>=0)
```

```
{base_cmd.linear.x=sqrt(joy_h*joy_h+joy_v*joy_v);}
//Publicación del tópico
cmd_vel_pub.publish(base_cmd);
```

Figura 4.7 Función **`Update( )`** del nodo joy2twist.

Las primeras dos sentencias hacen la conversión a coordenadas polares, y aunque en teoría esto es lo único que un robot necesita para navegar, el resultado es que cuando se desea que gire el robot sobre su propio eje sin desplazarse delante o hacia atrás, no es posible por las ecuaciones que describen el desplazamiento (la componente de movimiento lineal vale joy_h). Por tanto, se acota el funcionamiento en esa región donde el teleoperador está moviendo la palanca completamente a la derecha o a la izquierda para que el robot gire sobre su propio eje sin desplazarse. Y la última sentencia es el publicador del mensaje tipo Twist `base_cmd`.

Para facilitar la programación del nodo una vez identificadas sus funciones principales, se creó una clase llamada RobotDriver, que es una abstracción de la capa de conversión de datos del nodo joy a tópicos que se encarguen de manejar sistemas robóticos en ROS. En la Figura 4.8 se muestra un diagrama UML que resume el funcionamiento de la clase.

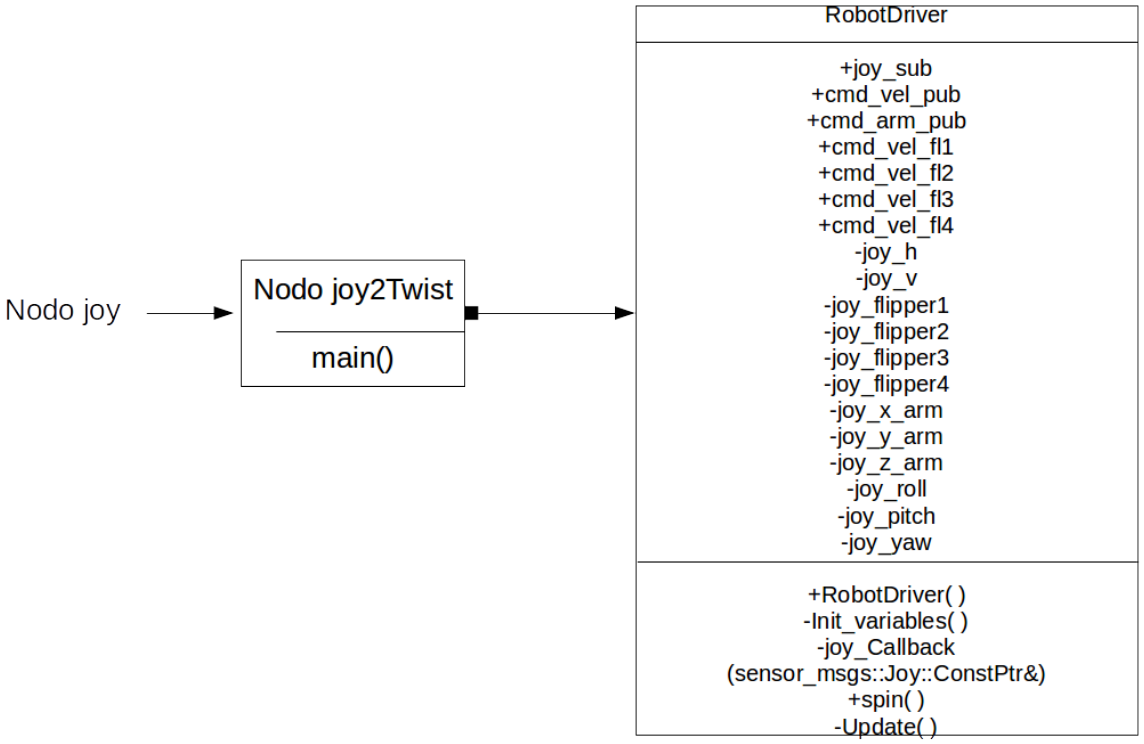

Figura 4.8 Diagrama UML de la clase RobotDriver.

La clase Twist2motor es una clase artificial, ya que en C++ la función **`main()`** es autónoma,

es decir, no es función miembro de ninguna clase y todos los programas deben contenerla, también, el cuadrado relleno que sale del bloque Twist2Motor indica que todo objeto de la clase RobotDriver pertenece y es creado por la clase Twist2Motor (es decir, que se crea en la función **`main( )`**).

En el bloque de RobotDriver, de arriba hacia abajo, primero es el nombre de la clase y el segundo sub bloque son las variables miembro de la clase; el + indica que son públicas y el – indica que son privadas. En este punto conviene explicar que las primeras seis variables son en realidad los publicadores de ROS, aunque en el código son variables privadas, y su función es publicar información hacia otros nodos, por lo que se decidió denotarlas como variables públicas. A continuación, se da una explicación de esta clase.

- Un suscriptor `joy_sub` se emplea para obtener el tópico `joy` proveniente del nodo `joy_node`; aquí se hace referencia a la función que toma dicho dato en el programa, que es la función miembro de la clase **`joy_Callback( )`**
- dos publicadores `cmd_vel_pub` y *`cmd_arm_pub`* de tipo geometry_msgs::Twist **[12]** para controlar la base móvil y el brazo manipulador respectivamente
- cuatro publicadores *`cmd_vel_fl1 - cmd_vel_fl4`* de tipo std_msgs::Int16 **[13]** para controlan los brazos auxiliares
- Dos variables tipo double `joy_h` y `joy_v` para mapear el movimiento de una palanca del control a coordenadas polares de tipo double
- Cuatro variables para el control de los cuatro brazos auxiliares del robot de tipo double (`joy_flipper1, joy_flipper2, joy_flipper3, joy_flipper4`)
- Tres variables de tipo double para el movimiento en el espacio cartesiano del brazo manipulador (`joy_roll, joy_pitch, joy_yaw`)
- Tres variables de tipo double para el movimiento de rotación sobre los ejes XYZ del efector final del brazo manipulador (`joy_roll, joy_pitch, joy_yaw`)
- Un constructor donde se inicializan los publicadores y suscriptores y que también llama a la función **`init_variables( )`**
- Una función **`init_variables( )`** para inicializar las variables privadas de la clase
- La función **`joy_Callback( ),`** que está asociada al suscriptor `joy_sub`, toma como entrada la referencia al apuntador del mensaje de entrada joy_msg
- Una función **`spin( )`**, que realiza la comprobación de errores de ROS y que llama a la función **`update( )`** para que en ella se publiquen los topicos de salida.
- Una función **`update( ),`** que se encarga de tomar los datos asignados en la función joy_Callback, y asignarlos a los publicadores `cmd_val_pub, cmd_arm_pub, cmd_vel_fl1, cmd_vel_fl2, cmd_vel_fl3, cmd_vel_fl4` gracias a las variables privadas de la clase.

Ahora que se ha descrito la clase creada, sus variables y funciones, en la Figura 4.9 se muestra un diagrama de flujo que muestra el funcionamiento del programa, se puede observar que se crea un objeto tipo RobotDriver, es decir, se crea un objeto tipo **`RobotDriver()`** y posteriormente se ejecuta la función spin() dentro de la cual lo primero que se hace es comprobar el correcto funcionamiento de ROS llamando a la función `ros::ok()`. En caso de que `ros::ok()` regrese un valor booleano verdadero, se ejecutan las tres siguientes funciones: `update()`, `ros::spinOnce()` y `ros::RateSleep()`.

La función `update()` que se describió anteriormente, asigna y publica los datos obtenidos del tópico asociado al joystick en las variables de la clase. La función `ros::spinOnce()` se encarga de que se llame implícitamente a la función `joy_Callback()`, ya que en el código solamente se declara ésta sin llamarla explícitamente.

Después, se ejecuta la función `ros::RateSleep()` que se encarga de actualizar las entradas y salidas del nodo respecto al servidor central de ROS (roscore) e intentar cumplir con la frecuencia de ejecución del nodo (`rate`).

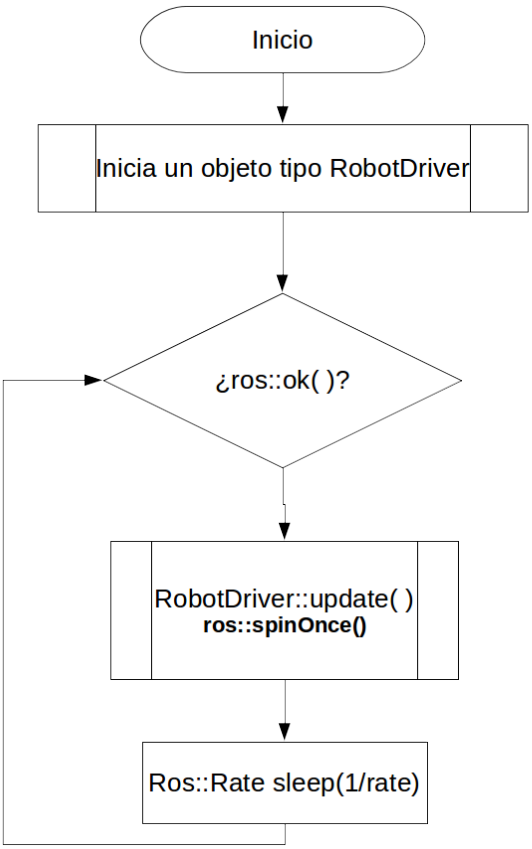

Figura 4.9 diagrama de flujo de ejecución del nodo joy2twist.

4.2.3 Nodo twist2motor

El nodo twist2motor se encarga de convertir el mensaje tipo Twist del nodo joy2twist a valores de velocidad deseada en las orugas del sistema de tracción del robot. El valor de velocidad deseada es una entrada al nodo de control PID de velocidad.

Este nodo contiene una peculiaridad y es que tiene dos frecuencias de operación: una, cuando no hay un valor de entrada, es decir, no hay un mensaje tipo Twist como entrada al nodo, caso para el cual la frecuencia de operación es de 10 Hz y no se publica nada; otra, cuando se reciba un nuevo valor de entrada al nodo, cambia la frecuencia de operación a 50 Hz y se mantiene durante cuatro ciclos de ejecución, durante los cuales se publica la velocidad deseada al nodo de control PID de velocidad.

La función que se activa cuando llega un valor de entrada, se encarga de convertir el mensaje contenido en el subscriptor de entrada al nodo a dos mensajes tipo Float32 que son, respectivamente, la velocidad deseada de la oruga derecha y la oruga izquierda, mediante las ecuaciones de cinemática de un robot de par diferencial, el fragmento de código de la Figura 4.10 es el que se utiliza

```
if( dr < -1.4 || dr > 1.4   )
{
//if dr>1.4 turn left
      right = ((dr * w /2));
      left = - (dr * w /2);

}
else
{
      right =map_vel*( ( 1.0 * dx ) + (dr * w /2));
      left = map_vel*(( 1.0 * dx ) - (dr * w /2));
}

      std_msgs::Float32 left_;
      std_msgs::Float32 right_;

      left_.data = left;
      right_.data = right;


      pub_lmotor.publish(left_);
      pub_rmotor.publish(right_);

      ticks_since_target += 1;
      ros::spinOnce();
```

Figura 4.10 Conversión de coordenadas polares del mansaje tipo Twist a las señales de control de los motores derecho e izquierdo.

Las ecuaciones que definen la cinemática de un robot de par diferencial son:

$$dx=(l+r)/2$$
$$dr=(r-l)/w$$

donde la entrada al sistema es dx y dr que son los valores del mensaje tipo Twist, que corresponden a las coordenadas polares de hacia dónde se quiere que se mueva el robot en función de hacia dónde esté moviendo la palanca izquierda del control de Xbox el teleoperador. La l y la r son el desplazamiento de la rueda izquierda y la rueda derecha respectivamente y w es la distancia entre los centros de las ruedas. Al despejar l y r del sistema de ecuaciones, se obtiene :

```
r=dx+dr*w/2
l=dx-dr*w/2
```

Cuando el valor de entrada en el mensaje Twist correspondiente al ángulo de desplazamiento (dr) es mayor a 1.4 o menor a 1.4 radianes, significa que el usuario quiere que el robot gire sobre su propio eje, sin avanzar linealmente. Por ello el código considera la condición anteriormente mencionada.

Para este nodo también se diseñó la clase TwisToMotors, y en la Figura 4.11 se observa el diagrama UML de la clase. la clase TwistToMotors es artificial ya que únicamente envuelve a la función **`main( ).`**

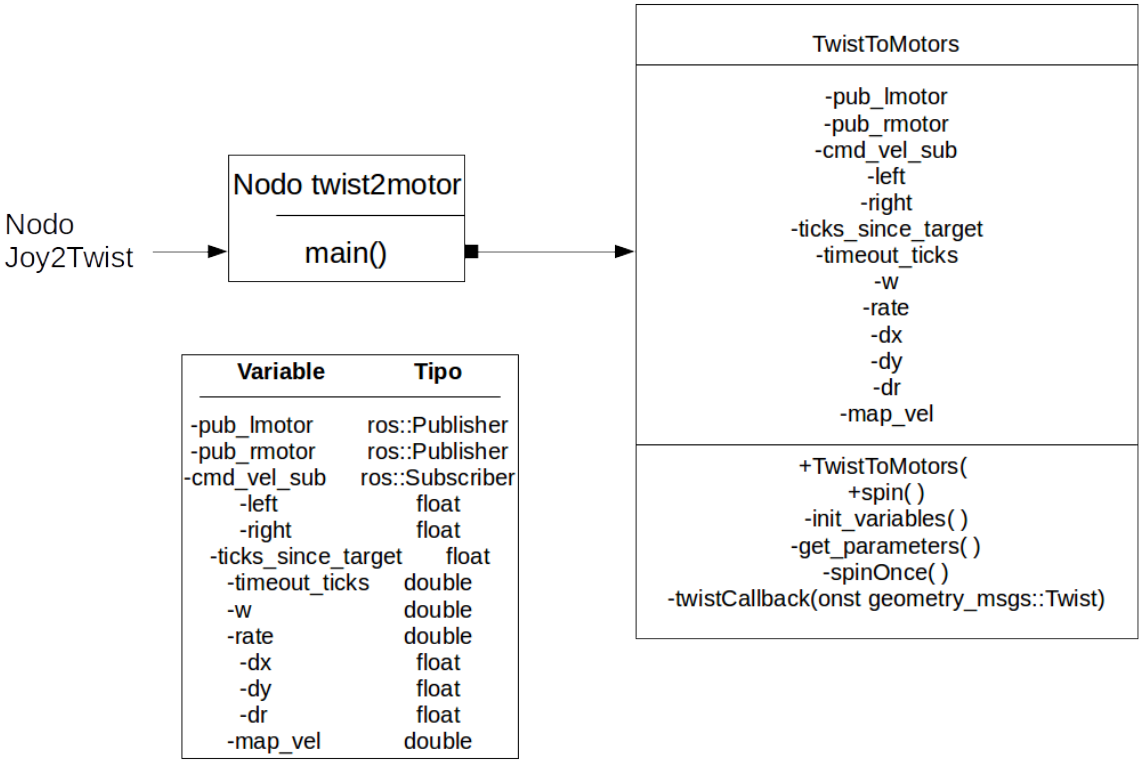

Figura 4.11 Diagrama UML de la clase TwistToMotors.

Se entiende de la Figura anterior que las variables `pub_lmotor` y `pub_rmotor` son tópicos que publican las velocidades de referencia para los controladores de velocidad de las orugas izquierda y derecha del robot, y la variable `cmd_vel_sub` es la que se suscribe a la salida del nodo joy2twist, ya que es un suscriptor, y se relaciona con la función miembro de la clase **`twistCalback(const geometry_msgs::Twist)`**. Las demás variables tienen un uso específico, y están asociadas a las ecuaciones de cinemática del robot, `'dx', 'dy', 'dr', 'left', 'right'` y `'map_vel'`.

Las variables ‘ticks_since_target’, ‘timeout_ticks’ y ‘rate’, por su parte, sirven para implementar las dos frecuencias de funcionamiento del programa. En la Figura 4.12 se muestra un diagrama de flujo del programa principal, donde se observa que estas variables implementan un contador que cambia la frecuencia de funcionamiento del nodo mientras está en ejecución.

De igual manera, en la Figura 4.13 se muestra el funcionamiento de las funciones miembro de la clase **spinOnce( )** y la función **twistCallback( )**. La función **spinOnce( )** es muy simple en su funcionamiento, toma los valores del mensaje tipo Twist que se asignan a las variables ‘dx’, ‘dy’ y ‘dr’, y las convierte a comandos de velocidad para las ruedas mediante las ecuaciones de cinemática de un robot de par diferencial, después publica esa información en los tópicos *pub_lmotor* y *pub_rmotor,* finalmente aumenta la frecuencia de ejecución del nodo a 50 Hz y en uno la variable ‘ticks_since_target’.

La función **twistCallback( )** por su parte, mediante el esquema de ejecución de nodos de ROS permite la ejecución de las funciones asociadas a suscriptores cuando se llama a la función **ros::spinOnce( )**. Y se encarga de asignar los valores del mensaje tipo Twist a las variables ‘dx’, ‘dy’ y ‘dr’.

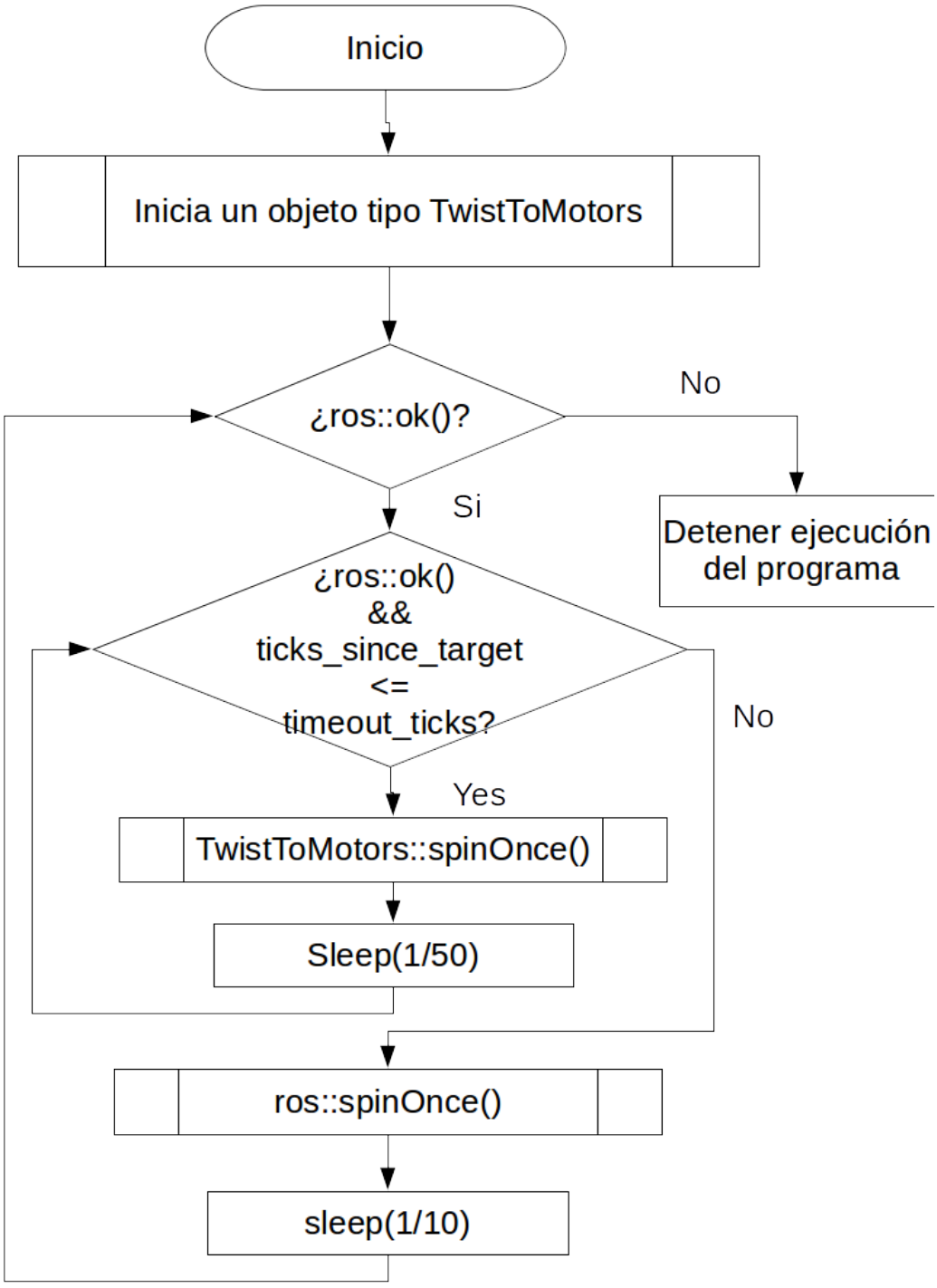

Figura 4.12 Diagrama de flujo del nodo TwistToMotors.

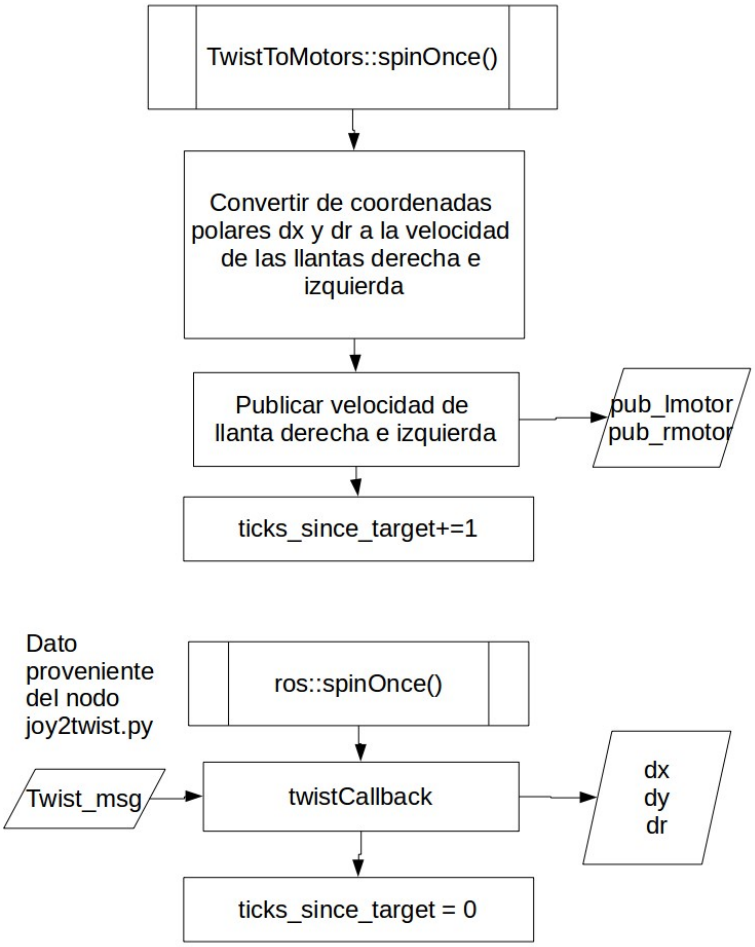

Figura 4.13 Diagrama de flujo de las funciones miembro de la clase TwistToMotors.

4.3 Esquema de control PID

De la Figura 4.5 se observa que luego del nodo twist2motor siguen nodos de control de velocidad de las orugas de tracción, y en el caso del brazo manipulador se necesitan nodos de control de posición angular. Para ello, se desarrolló un nodo genérico que sirve tanto para el control de posición como de velocidad, con miras a desarrollar un nodo que haga el control de posición y velocidad en cascada en un futuro.

4.3.1 Control de velocidad

El nodo de control PID de velocidad es utilizado en las orugas de tracción. Tiene como entradas la velocidad deseada de giro de la oruga y la posición angular del eje del motor, y como salidas tiene la velocidad angular y el comando a ingresar al nodo de envío y recepción de datos del microcontrolador que el módulo de potencia va a interpretar como PWM, de acuerdo a su especificación de uso. En la Figura 4.14 se muestra el diagrama de la clase diseñada para implementar el control PID de velocidad de una oruga de tracción;

puede observarse que es un programa con cierta complejidad respecto a los anteriores, así que se explicará detalladamente la función de las variables y funciones miembros de la clase, y posteriormente se apoyará dicha explicación con los diagramas de flujo de funcionamiento del programa y de las funciones miembro de la clase PID_control más importantes.

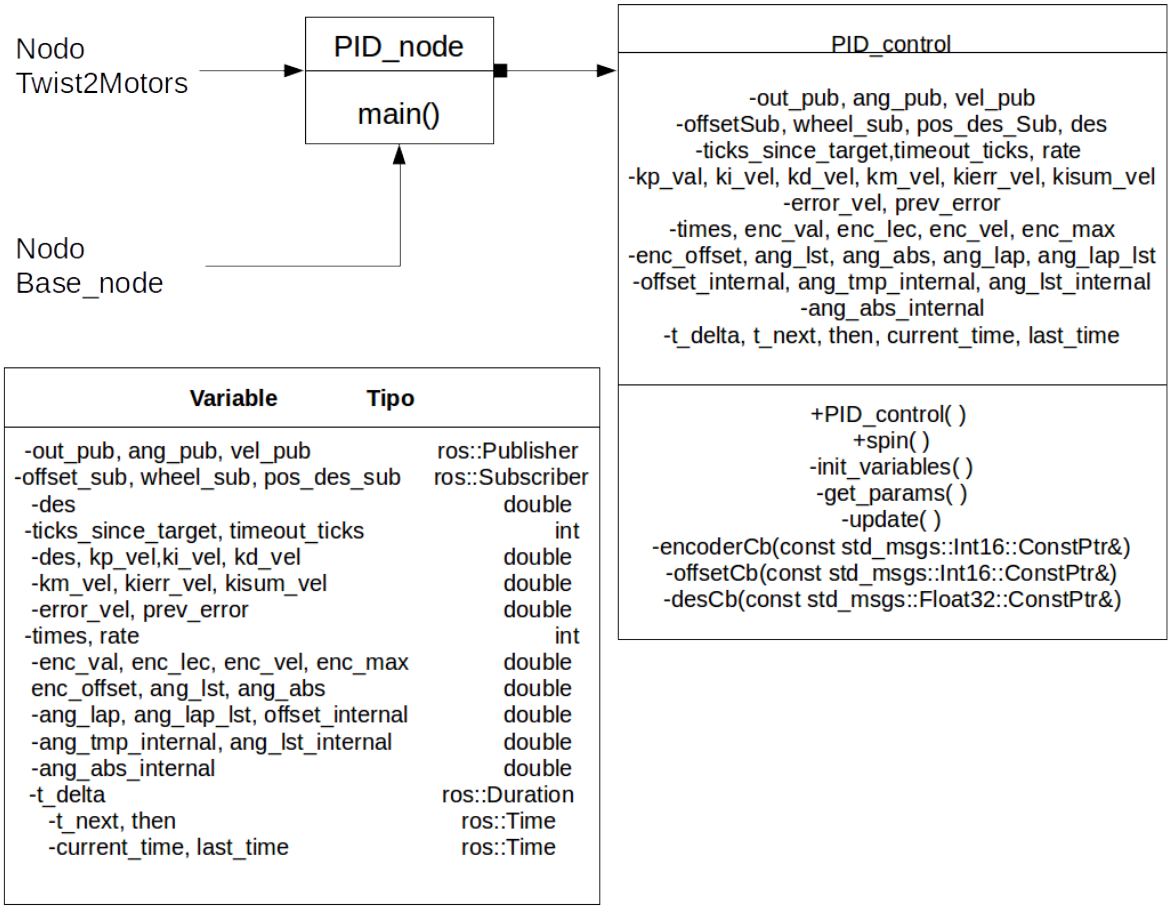

Figura 4.14 Diagrama UML de la clase PID_control

A continuación, se explicará el funcionamiento de la clase; como se menciona en el párrafo anterior, las entradas al programa vienen del nodo twist2motor que envía la señal de control de la oruga y del nodo base_node explicado en el capítulo anterior, que se encarga de enlazar ROS con el microcontrolador que está controlando los actuadores y obteniendo la información de los sensores del robot. Por ello los suscriptores son `offset_sub`, `wheel_sub` y `pos_des_sub`; `offset_sub` es un valor que se suma a la señal proveniente del `encoder` en caso de que sea un `encoder` absoluto, `wheel_sub` es el valor proveniente del `encoder`, es decir, del nodo node_base, y el suscriptor `pos_des_sub` es el que toma el tópico proveniente del nodo twist2motor con el valor de velocidad de referencia para el controlador.

Una vez descritas las entradas y las salidas, conviene describir las funciones principales (caso de uso) de la clase y cómo se relaciona con las funciones miembro de la clase, así como con las variables declaradas. De ser necesario, se sugiere revisar en la introducción el apartado 1.1.4 de control digital para entender mejor los siguientes pasos.

- Obtener la información del *encoder*, normalmente es una señal que indica la posición, utilizando el tiempo de ejecución, obtener diferenciales de tiempo y de distancia para construir una señal de velocidad
- Obtener la señal de referencia para el controlador, obtener del nodo twist2motor, la señal de control del actuador que normalmente se expresa en unidades de rad/s
- Calcular la señal de control, para ello se necesita la información de los dos pasos anteriores, además de tomar el tiempo de ejecución de cada ciclo y tener las diferenciales de tiempo y error para calcular correctamente la señal de control con la ecuación del controlador PID.

De acuerdo con los puntos anteriores y la Figura 4.14, se explica por qué existen muchas variables miembro de la clase, ya que gran parte de ellas son para la obtención de las señales del *encoder* (`'ang_lst'`, `'ang_abs'`, `'ang_lap'`, `'ang_lap_lst'`, `'offset_internal'`, `'ang_tmp_internal'`, `'ang_lst_internal'`, `'ang_abs_internal'`), la posterior construcción de la señal de velocidad a partir de ella (`'enc_vel'`, `'enc_lec'`, `'enc_vel'`, `'enc_max'`, `'enc_offset'`), las variables de cálculo de error, de valor deseado, y las constantes del controlador (`'des'`, `'kp_vel'`,`'ki_vel'`, `'kd_vel'`, `'km_vel'`, `'kierr_vel'`, `'kisum_vel'`, `'error_vel'`, `'prev_error'`) y las variables para calcular los diferenciales de tiempo de ejecución (`'t_delta'`, `'t_next'`, `'then'`, `'current_time'`, `'last_time'`). Además, las funciones **`encoderCb( )`** y **`offsetCb( )`** corresponden al primer paso, la función **`desCb( )`** al segundo paso, y la función **`update( )`** es la que se encarga de calcular la salida del controlador y publicarla, es decir el tercer paso.

En la Figura 4.15 se describe el funcionamiento principal del programa, y las Figuras 4.16 y 4.17 muestran las funciones miembro de la clase que son más importantes. La Figura 4.16 muestra la función **`update()`**, en la cual se lleva a cabo el control PID; las entradas al nodo, son la posición angular de la oruga proveniente del nodo de adquisición de datos con el microcontrolador que representa la realimentación del lazo de control y la velocidad deseada que proviene del nodo twist2motor que es el valor de referencia del control, y las salidas son la velocidad angular de la oruga y la señal de control; la velocidad angular de la oruga va al nodo del cálculo de la odometría del robot y la señal de control va al nodo de adquisición de datos del microcontrolador.

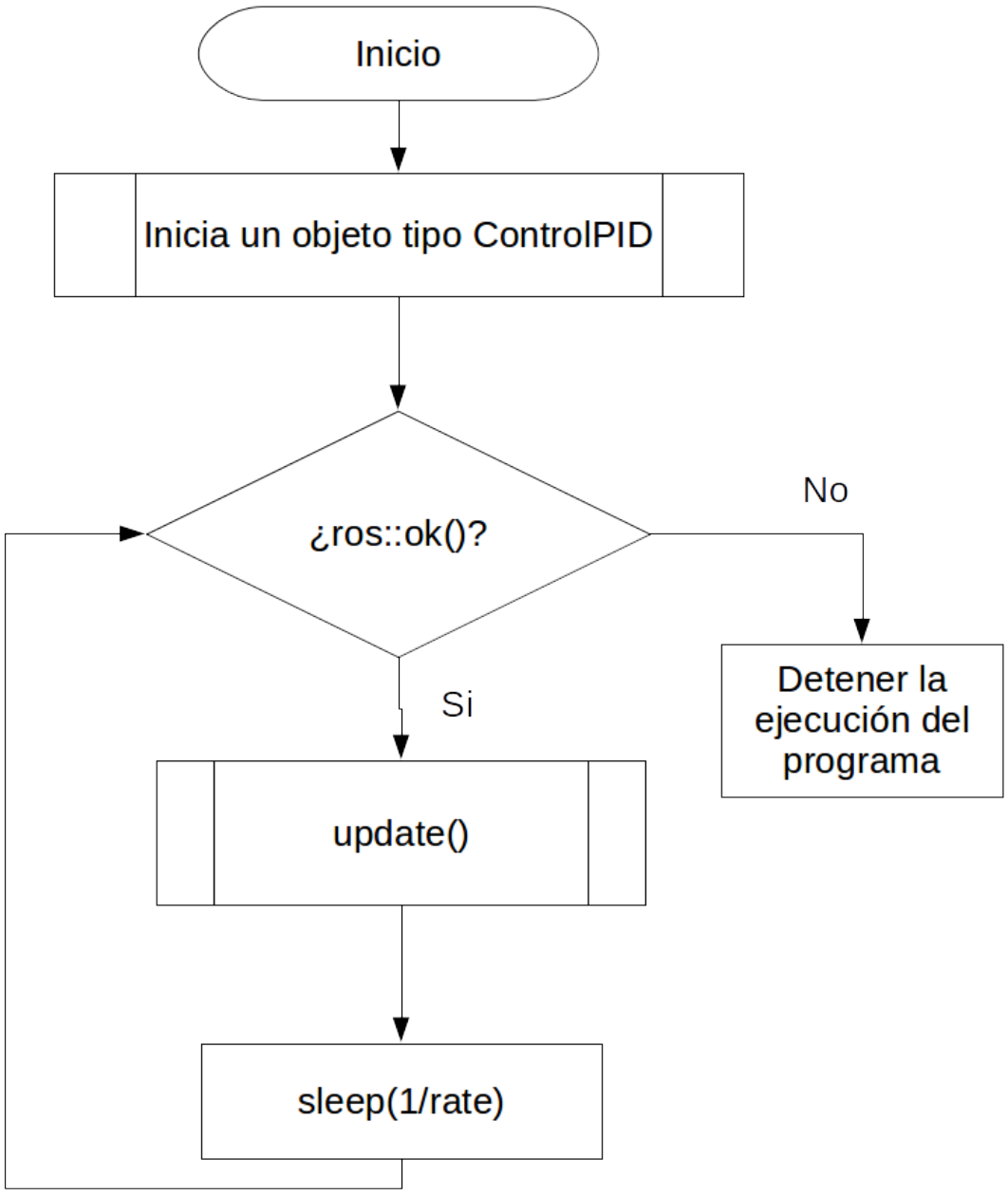

Figura 4.15 Diagrama de flujo de un nodo PID_control

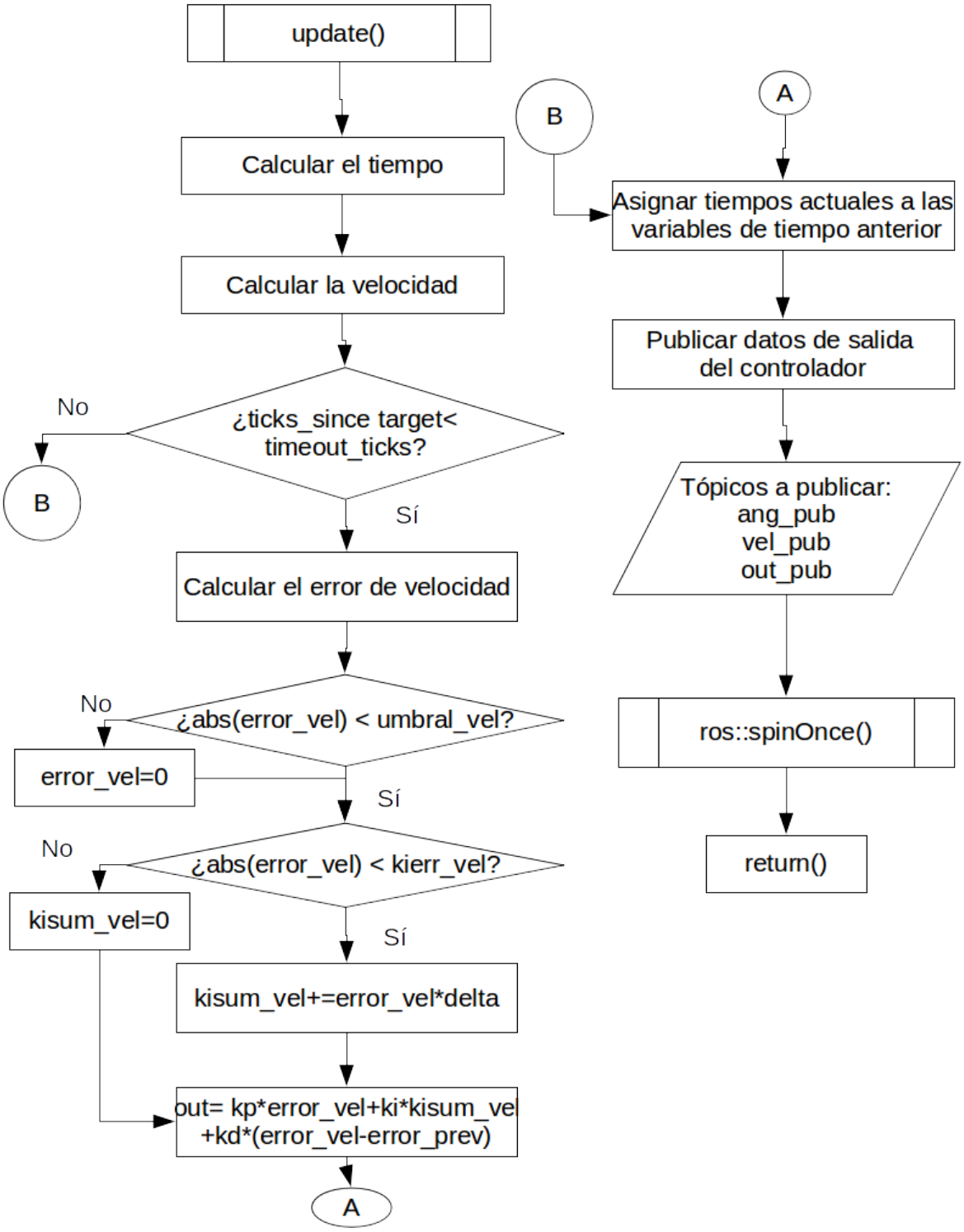

Figura 4.16 Diagrama de flujo de la función `PID_control::update()`.

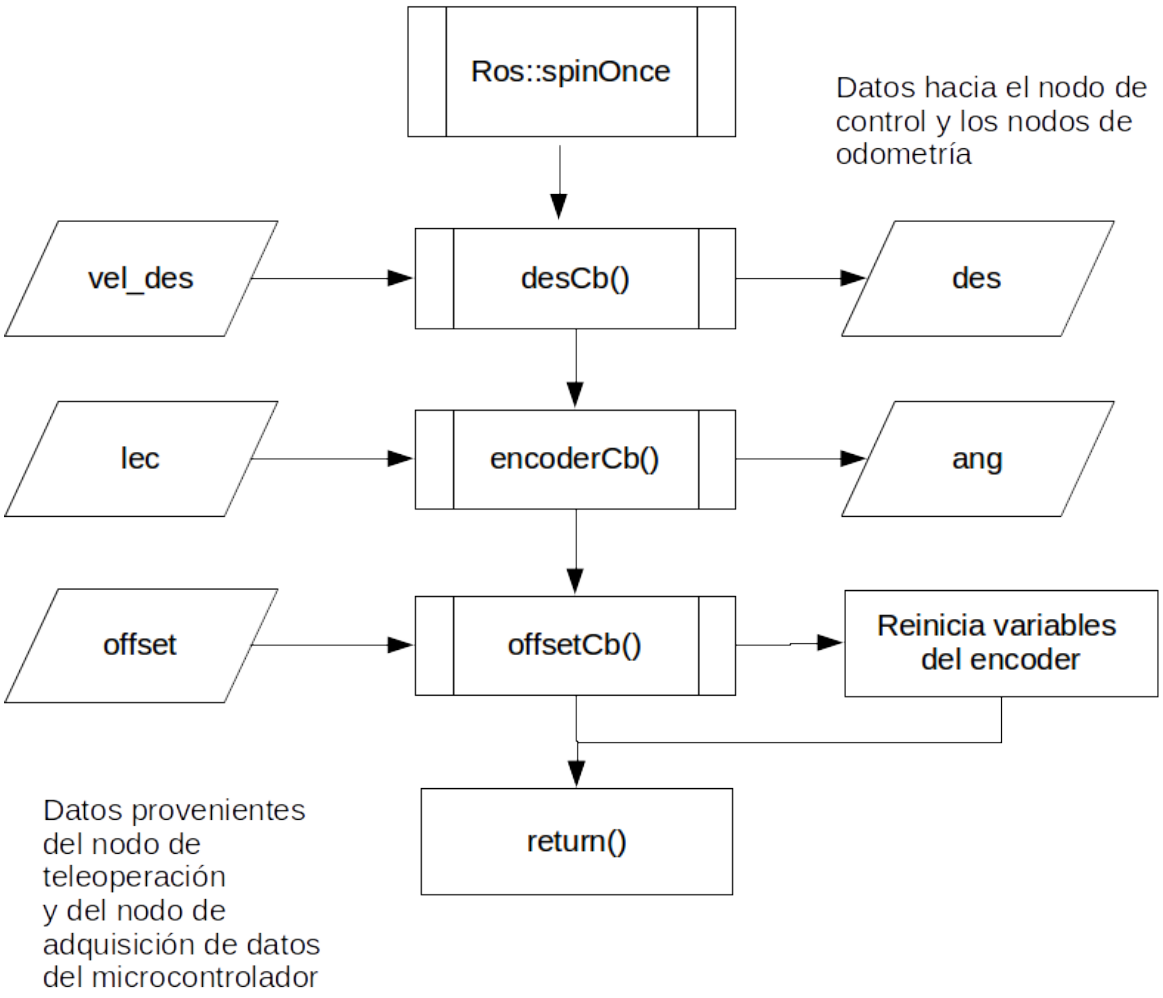

Figura 4.17 Diagrama de flujo de las funciones `PID_control::encoderCb()`, `PID_control::offsetCb()` y `PID_control::desCb()`.

4.3.2 Control de posición

Tomando como plantilla el nodo anterior, se desarrolló un nodo de control de posición para las articulaciones del brazo manipulador y los brazos auxiliares del robot. Se usó el mismo esquema de control, solamente cambió que en la obtención de la señal no se reconstruye la señal de velocidad y el error no se calculó respecto a la velocidad, sino a la posición. Con este esquema se tuvieron buenos resultados.

Capítulo 5
Pruebas y análisis de resultados

A partir de las especificaciones de la Figura 5.1, se tiene que la salida o solución del problema sería un robot teleoperable, capaz de ser controlado por un teleoperador y con un uso menor al 50% del procesador de la computadora a bordo del robot, además de tener confiabilidad en su uso, es decir, proteger al robot de colisiones con él mismo debido a retrasos en el envío de información visual al teleoperador. Además, se especifica que el nuevo modelo de programación del robot sea compatible con software de terceros y que sea modular.

Las pruebas realizadas se dividen en dos secciones: la primera, son las pruebas de la correcta implementación de las conexiones entre componentes de sensado y control de sistemas mecánicos, el diseño y construcción de la correspondiente tarjeta impresa y la adecuada programación de la rutina en el microcontrolador para obtener y enviar información mediante comunicación serial; la segunda, son las pruebas de los programas en la computadora en lo que se conoce como la capa de bajo nivel del software del robot. En esta parte, se hicieron pruebas de movimiento de las orugas de tracción, los brazos auxiliares, y el brazo manipulador.

Entrada: Mecanismos de locomoción del robot FinDER v3 y la arquitectura de electrónica y programación del robot FinDER v2

Variables de entrada: Limitaciones de las entradas

Número de sensores----------Tiempo de muestreo

Número de actuadores----------Duración de las baterías

Número de microcontroladores-----No exceder el 50% del uso del procesador de la computadora en la comunicación con ellos

API de comunicación serial micro-computadora----------Ninguna

Salida: Robot teleoperable

Variables de salida: Limitaciones de la salida

Acciones del robot----------Grado de cumplimiento de las tareas deseadas

Versatilidad----------Capacidad de acoplarse a software desarrollado por terceros

Confiabilidad----------El robot no realice acciones que lo dañen

Variables de solución

Microcontrolador (placa de desarrollo)

API de comunicación serial

Módulo de potencia

Sensores

Modularidad de la solución

Arquitectura de programación

Restricciones

Batería de 12V 18 Ah alimenta el sistema

El robot debe tener redundancia en cuanto a seguridad de operación ya que es teleoperado

Debido a las modificaciones de los sistemas mecánicos, el espacio para el hardware es menor

Se tiene restricción de costo, se tratará de reutilizar el mayor número de piezas de hardware y de código.

Criterios

Facilidad de programación

Consumo de CPU

Costo

Confiabilidad

Operabilidad

Figura 5.1 Análisis del problema

La primera parte de las pruebas se detalla en el Capítulo 3, mediante prueba y error se obtuvo el esquema del robot mostrado en la Figura 5.2 que resume la solución al problema de un robot modular, teleoperable y con un consumo bajo de recursos de la computadora central en la comunicación serial con los microcontroladores.

Además, para cumplir con el objetivo planteado de seguridad de la tele operación del robot, se adicionaron sensores de final de carrera en todas las juntas de revolución con posible colisión en su recorrido. Adicionalmente, se decidió en la última iteración agregar finales de carrera por software al menos en los actuadores con *encoder* de posición absoluta asociado para crear un sistema redundante en cuanto a su seguridad.

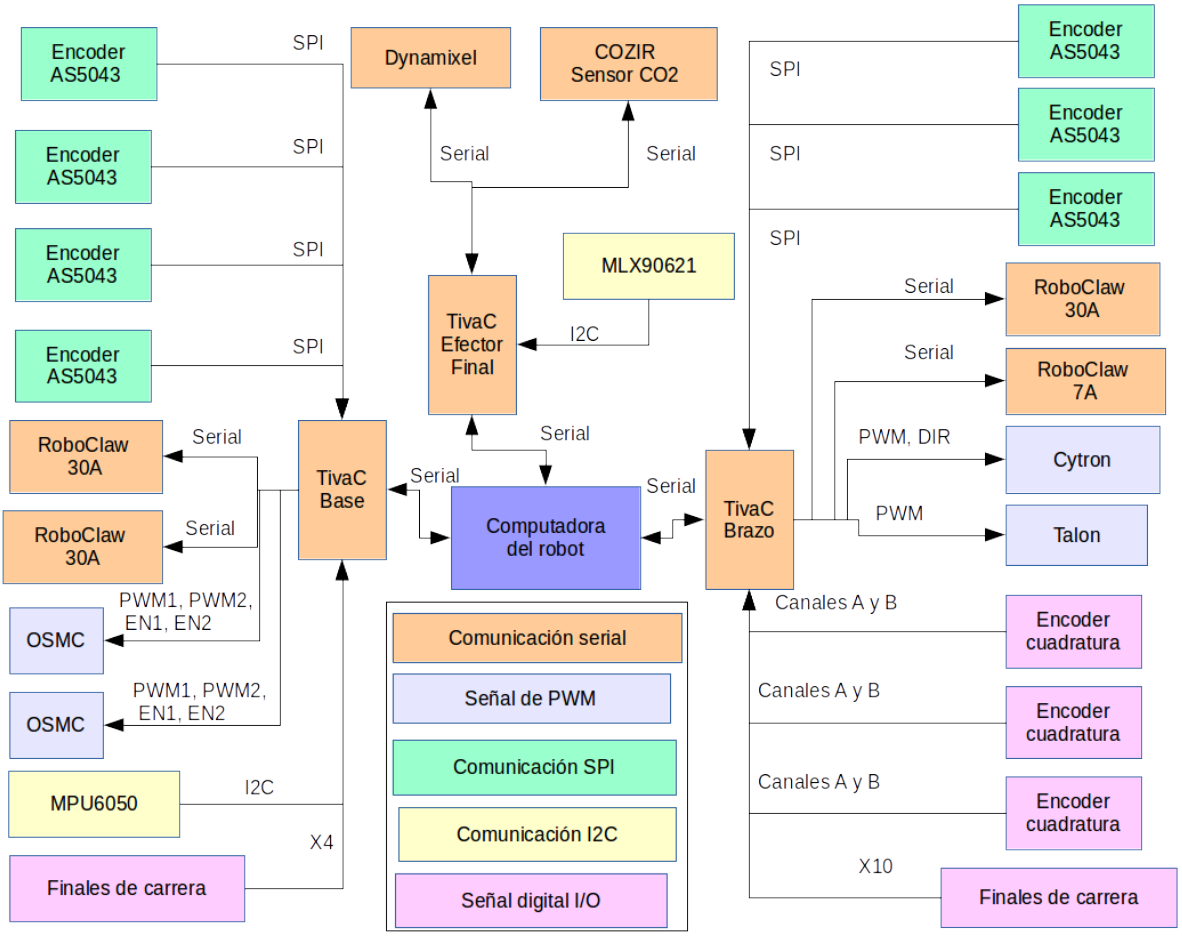

Figura 5.2 Esquema de conexión de los elementos de hardware del robot FinDER v3.

En cuanto al uso del procesador, se hizo uso de una utilidad del sistema operativo para graficar el uso del procesador mientras se ejecutaban los procesos en el robot, en la Figura 5.2 se muestra el uso de CPU y de memoria de la computadora de abordo por cada programa en ejecución mientras el robot está en operación, mediante una utilidad del sistema operativo conocida como *top*.

Al observar la Figura 5.3 desde arriba y de la izquierda hacia la derecha observamos que cada programa tiene asociado un *PID* (process id), el *USER* (usuario) asociado a dicho programa, PR es la prioridad de ejecución del proceso, VIRT es la cantidad de memoria virtual que utiliza, RES es la cantidad de memoria física que utiliza, SHR es la cantidad de memoria compartida, S es el estado del proceso. %CPU y %MEM indican qué porcentaje del procesador y memoria está utilizando el proceso. Los programas están ordenados empezando por aquel que más recursos consume y va decreciendo. El programa que más consume es python, el cual está asociado al nodo de comunicación serial; su consumo es de 22.6% de CPU y 0.9% de memoria. Los siguientes dos son los nodos de control PID de

las orugas de tracción del robot, que en conjunto usan 6% del CPU. Luego están los nodos twist2motor y joy2twist, cada uno usa 1% del CPU.

Se observa que, de todos estos programas, solamente uno se está ejecutando con el uso del intérprete de Python, mientras que los demás se ejecutan a bajo nivel ya que fueron programados en C++, y aunque las condiciones de frecuencia de ejecución son diferentes y que en el programa de comunicación serial con ROS se utilizan dos hilos de ejecución, se tiene la oportunidad de mejorar la eficiencia del nodo de comunicación serial realizando su codificación en C++.

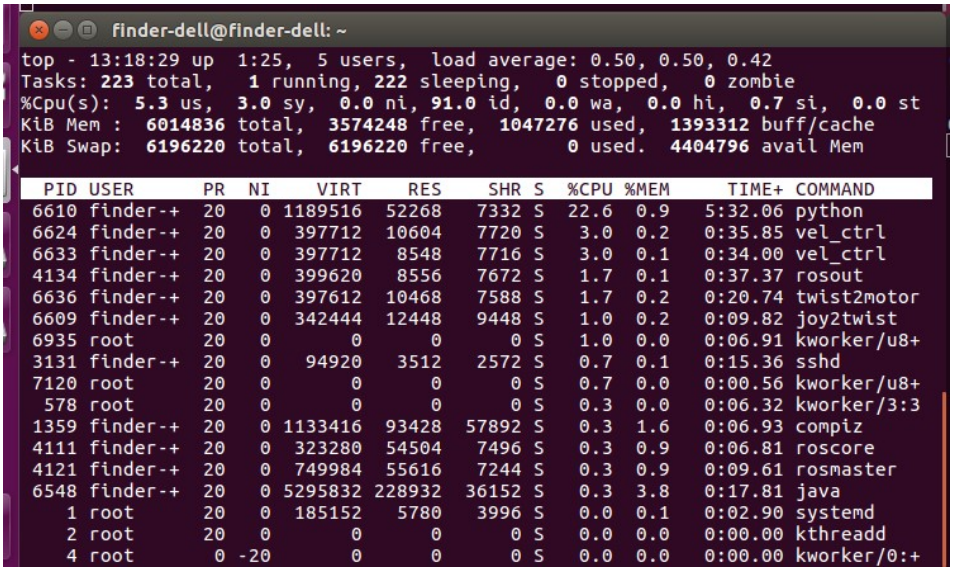

Figura 5.3 Utilidad que muestra el uso de recursos de cada programa en ejecución.

En la Figura 5.4, en otra utilidad conocida como gnome-system-monitor, se muestra de forma gráfica el uso de recursos como el procesador, memoria dinámica y transferencia de datos en red. La primera gráfica muestra el uso de los núcleos del procesador y su evolución en el tiempo, que está aproximadamente en 10% cuando se está utilizando únicamente la base de tracción del robot. Empleando el brazo manipulador y el efector final, puede llegar a usar el 30% de los recursos, lo cual es bastante bueno para los objetivos planteados.

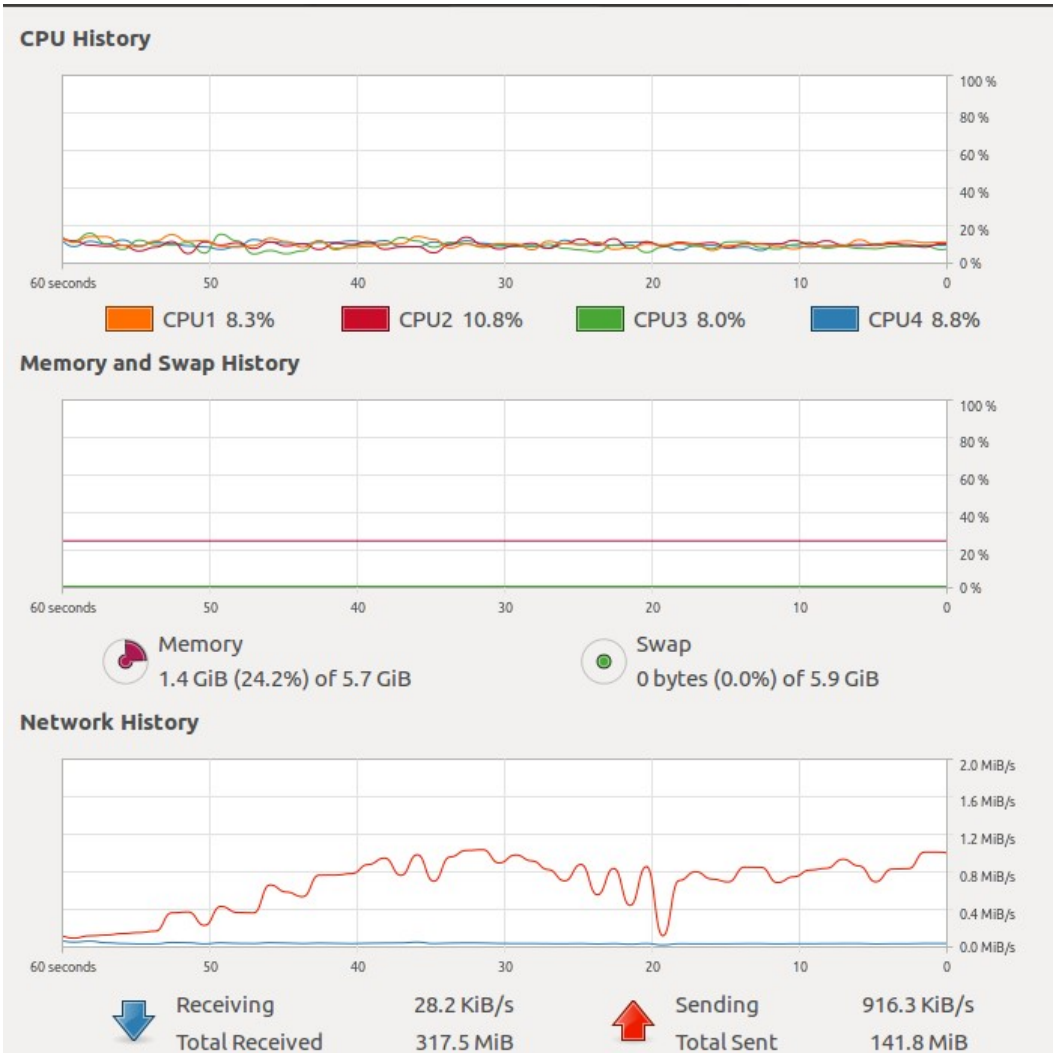

Figura 5.4 La utilidad gnome-system-monitor muestra de forma gráfica el uso de recursos.

Una manera de validar el correcto funcionamiento de los programas creados es mediante la calibración y prueba de los controladores PID de velocidad de las orugas de tracción. En las Figuras 5.5 a 5.8 se muestran las gráficas obtenidas en el procedimiento de puesta en marcha y obtención de las ganancias del controlador de velocidad. La Figura 5.5 tiene las variables left_out y right_out, que son las señales de control cuando se está moviendo la palanca del joystick hacia delante.

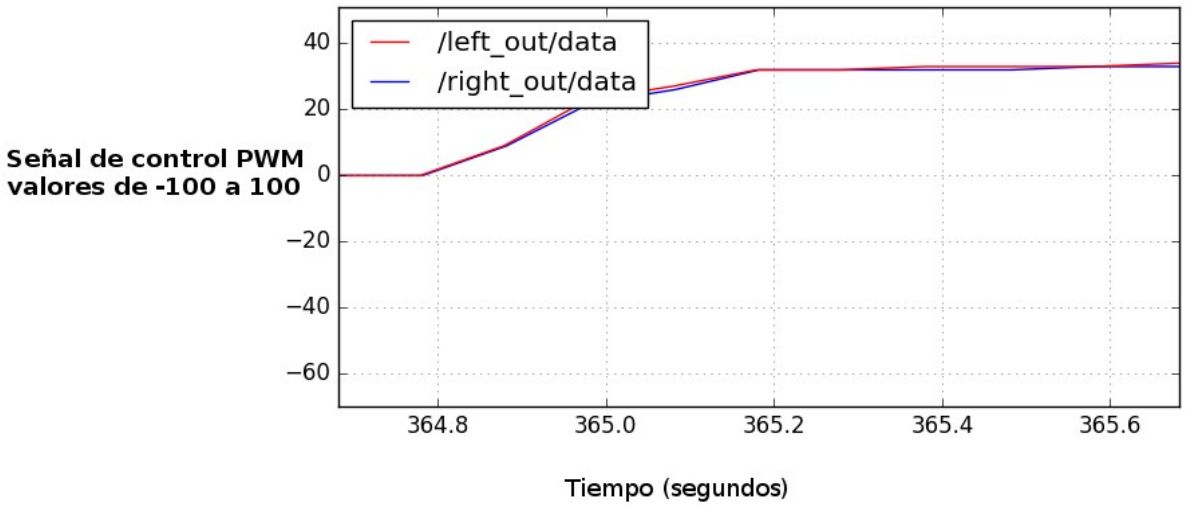

Figura 5.5 Gráfica de los valores de la señal de control de los controladores de velocidad sincronizados al enviar un comando de ir hacia delante.

La Figura 5.6 muestra el comportamiento de la señal deseada (/left_vel/data) contra la señal de realimentación (left_vel_des); se observa que el comportamiento de la señal reconstruida de velocidad a partir de la posición angular tiene algo de ruido, pero el desempeño se evaluó como bueno ya que el desplazamiento del robot es consistente con el movimiento de las palancas del control de xbox. No obstante, un ajuste de ganancias del controlador más exhaustivo puede mejorar en muchos aspectos el comportamiento de las gráficas. También, en la Figura 5.8 se muestra el comportamiento de la señal de control que se envía a los módulos de potencia; dicha señal es solamente representativa de la verdadera señal de control que debería ser una señal de voltaje, mapeada de 0 a 12 V. Sin embargo, se puede confiar en que esta gráfica representa el comportamiento de dicha señal. Se puede observar que el comportamiento del controlador es subamortiguado, primero tiene un sobrepaso y oscila hasta llegar a estabilizarse, y al final cuando el motor se detiene, existe un pico en sentido contrario donde el controlador ayuda al frenado de la oruga.

El mismo análisis puede hacerse al observar las Figuras 5.7 y 5.9, que son las mismas descritas en el párrafo anterior pero obtenidas de la oruga derecha.

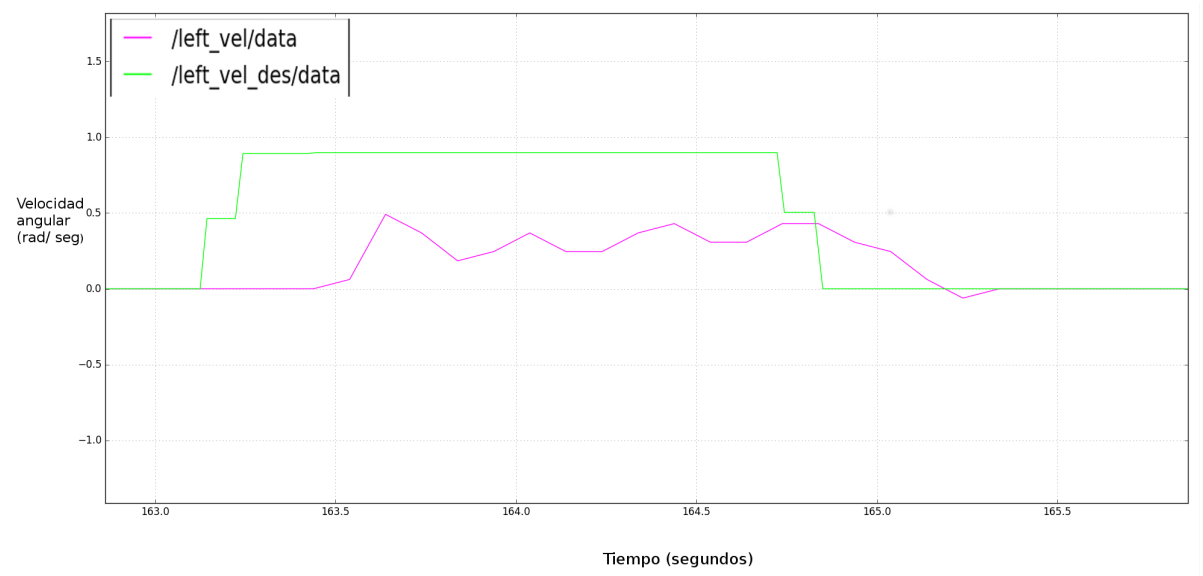

Figura 5.6 Gráfica de la señal deseada y de realimentación del controlador de velocidad de la oruga izquierda.

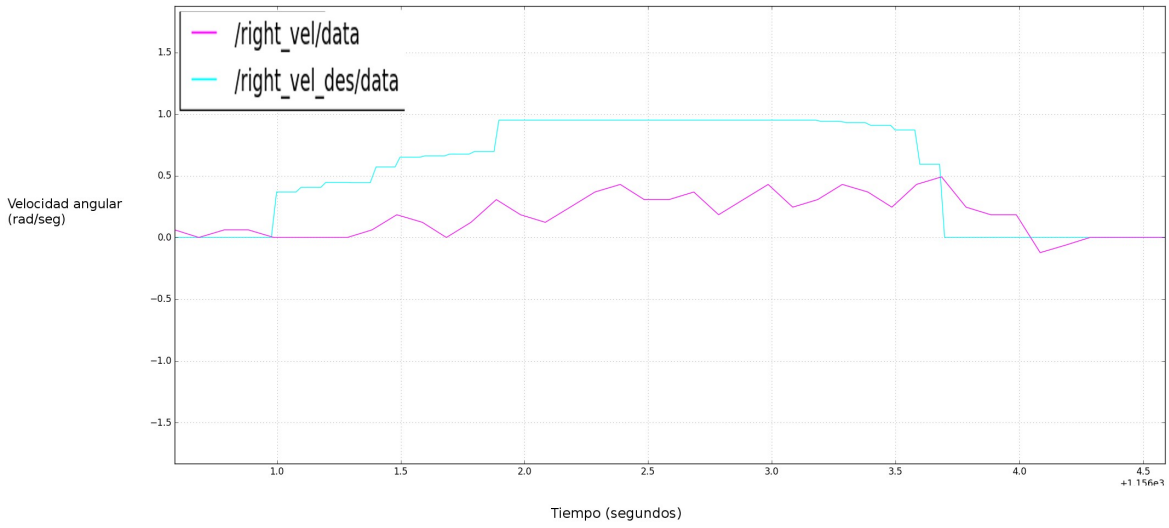

Figura 5.7 Gráfica de la señal deseada y de realimentación del controlador de velocidad de la oruga derecha.

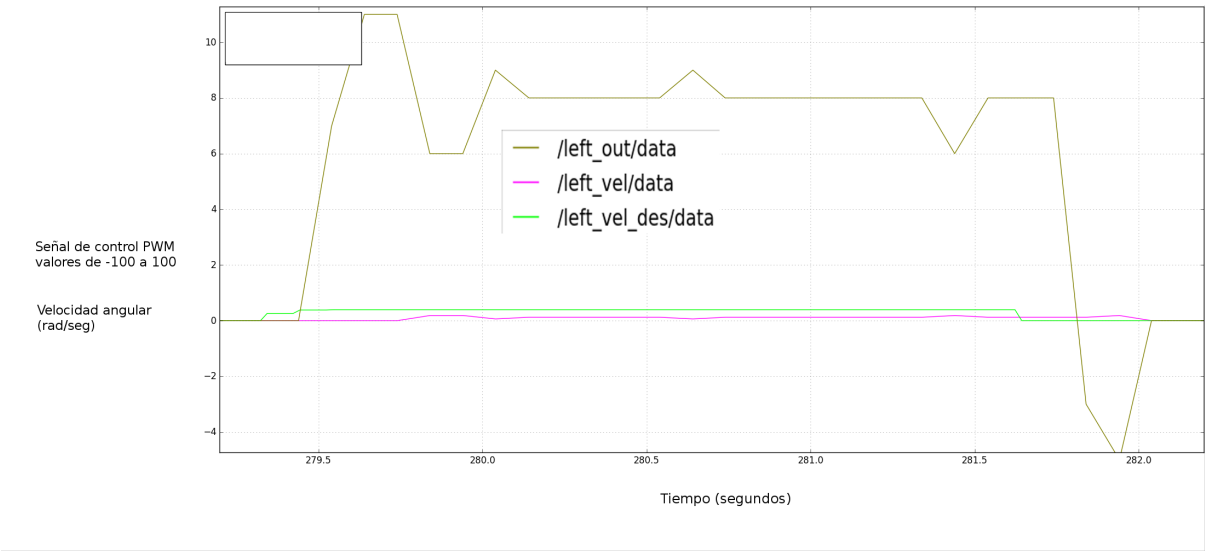

Figura 5.8 Gráfica de la señal de control, deseada y de realimentación del controlador de velocidad de la oruga izquierda.

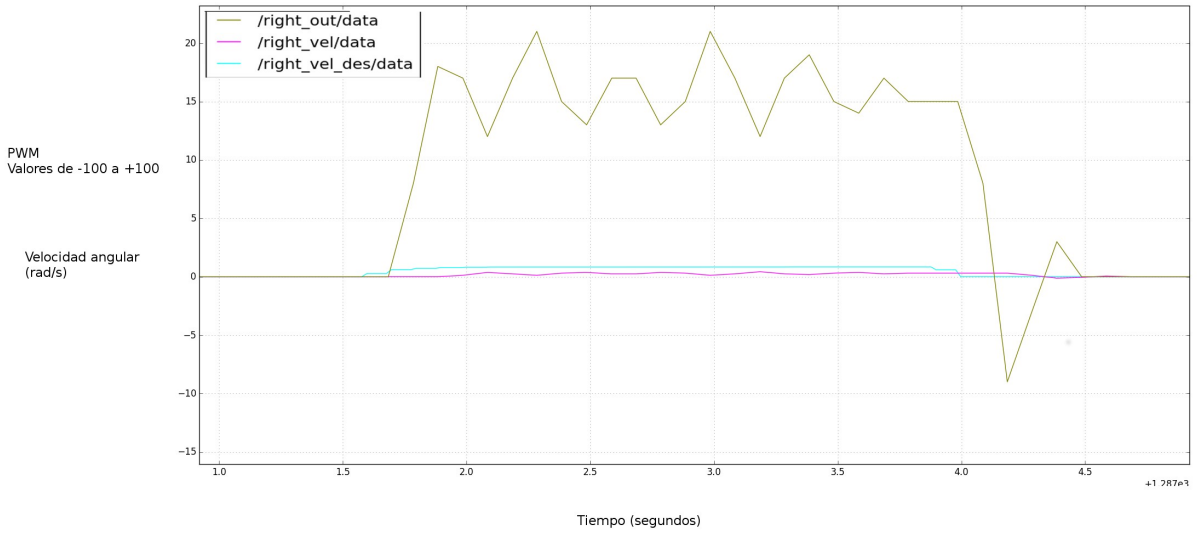

Figura 5.9 Gráfica de la señal de control, deseada y de realimentación del controlador de velocidad de la oruga derecha.

Capítulo 6
Conclusiones y trabajo a futuro

En la adaptación de la arquitectura del robot FinDER v2 al robot FinDER v3 se alcanzaron los objetivos en cuanto a la reducción del número de microcontroladores, a la implementación modular de la parte de tracción, brazo manipulador y efector final. Alguna prueba realizada se verificó que fue posible implementar una base móvil robótica que fue capaz de hacer un posterior desarrollo de alto nivel e incluso probar software de terceros. Una parte importante que no se menciona en el diseño realizado en este trabajo, es que el cableado y posicionamiento de las tarjetas impresas tuvo un papel importante en la correcta implementación de este sistema robótico.

En cuanto al uso del procesador, se llegó a un 35% de la capacidad de la computadora a bordo del robot, con lo que quedó probado que el uso de C++ es útil al momento de programar un sistema robótico en su capa de bajo nivel para dar mayor eficiencia al mismo; no obstante, el uso de Python en la implementación de la comunicación serial es un aspecto a mejorar (ver Figura 5.3), ya que haciendo la misma implementación utilizando C++, se puede lograr una notable mejora en eficiencia. Se sugiere que en una siguiente iteración se haga dicha implementación correctamente.

En cuanto a la confiabilidad del sistema, el movimiento individual de las juntas de revolución del robot quedó solucionado, utilizando switches mecánicos como final de carrera y estableciendo posiciones de tope en la ejecución de los programas de control de los módulos de potencia. El siguiente paso a tomar en este punto es evitar la colisión entre los eslabones del robot, por ejemplo, entre los brazos auxiliares del frente y los eslabones del bazo manipulador, así como entre los mismos eslabones. ¡Esto se puede resolver utilizando el paquete Move it!, el cual proporciona herramientas para el uso de alto nivel de un brazo manipulador.

Un aspecto interesante a trabajar también es el proponer e implementar nuevos modelos de control de los motores con base en lo realizado, un controlador más sofisticado que tome en cuenta el modelo del motor, tal como realimentación de estados o un controlador LQR, que pueden proponerse e implementarse teniendo como guía el presente trabajo. También se

pueden agregar elementos de no linealidad al sistema para mejorar el funcionamiento actual.

En cuanto a la evaluación del comportamiento del robot en un ambiente simulado se obtuvo una correcta evaluación de las capacidades físicas del robot, se sometió a tareas complejas como la navegación sobre terreno irregular, escaleras o barras paralelas, que ayudó a entender las limitaciones mecánicas del robot, así como de la teleoperación.

La navegación en terrenos irregulares y de difícil acceso implica que en una siguiente fase se realicen nodos de control de movimientos coordinados como rutinas para subir escaleras, movimiento en terrenos accidentados mediante un controlador que utilice como realimentación la señal del giroscopio, además de rutinas de control para los brazos auxiliares para que la base móvil se desplace en escaleras. Todo ello consistiría en crear la capa de nivel intermedio en la arquitectura del robot descrita en el Capítulo 1 (ver Figura 1.2).

El otro problema que se detectó está asociado a la teleoperación del robot ya que, en la transmisión de datos de las cámaras de vídeo del robot a la estación remota, existe un tiempo de retardo que en algunas condiciones de red puede ser de 30 segundos. Por consiguiente, también se propone como trabajo posterior hacer nodos que controlen el movimiento del robot de forma reactiva (comportamiento en función del LiDAR y de la información de las cámaras) y que apoyen a que la operación remota del robot sea segura.

Respecto a los componentes electrónicos y módulos de potencia que se utilizaron, el avance en la tecnología permite y obliga al desarrollador de robótica a modernizar muchos elementos como sensores, motores, o módulos de potencia, ya que continuamente salen al mercado nuevos dispositivos que pueden simplificar la labor de desarrollo. Por ello, seguramente en una siguiente mejora del robot se debe de hacer una extensa búsqueda de componentes que pueda ayudar a mejorar el desempeño o alguno de los aspectos aquí planteados, como el uso de recursos de la computadora central.

Finalmente, la arquitectura de bajo nivel a la que se llegó en este trabajo no es necesariamente la mejor; analizando el funcionamiento de la misma, se puede decir que es una arquitectura centralizada, donde se enfatiza la reducción del número de microcontroladores a utilizar y el papel que desempeñan es de un transmisor-receptor de información sin realizar un procesamiento de dichos datos, priorizando el aprovechamiento de la cantidad de pines y buses de comunicación que ofrece el mismo. Siguiendo el sentido de flujo de información, la computadora central lidia con el tratamiento de la información, tanto la que proviene de los sensores, como la que se envía a los módulos de potencia.

No obstante, hay otras arquitecturas que se pueden explorar y que se han utilizado exitosamente, por ejemplo, el del robot TR1 **[14],** donde se hace uso de varios microcontroladores de menor desempeño (Arduino nano), cada uno asociado a cada una de revolución del robot. En esta implementación, la cual se puede llamar distribuida, todos los microcontroladores están conectados en serie bajo un protocolo de comunicación I2C, que permite por su especificación tener un bus de comunicación con cuatro cables únicamente. Esto permite que el trabajo del microcontrolador bajo esta arquitectura sea realizar el tratamiento de la información (por ejemplo, el control de velocidad PID o el control de

posición PID), con lo que se logra bajar el uso de recursos de la computadora central.

Otra posible mejora que se propone bajo esta arquitectura distribuida es cambiar el protocolo de comunicación del bus al protocolo CAN, el cual es más robusto que el protocolo I2C en cuanto a la detección de errores, ya que está estandarizado por lo que es empleado en la industria.

Referencias

1 Robot Modeling and Control. Spong, Mark. Hutchinson, Seth. Vidyasagar, M. John Wiley and Sons. First Edition, pp 1-3

2 The Quest for Artificial Intelligence. A history of ideas and achievements. Nilsson, Nils J. . Cambridge University Press, 2010 pp. 213-216

3 http://wiki.robocup.org/Robot_League

4 http://wiki.robocup.org/images/3/36/2017-02_RoboCupRescueRulebook_lowres.pdf

5 Microcontrolador PIC16F84, Desarrollo de proyectos. Palacios Municio, Enrique. Remiro Domínguez, Fernando.Ra-Ma Editorial. Madrid, España, 2004.

6 https://es.wikipedia.org/wiki/Sistema_de_control

7 http://www.ros.org/about-ros/

8 https://github.com/StevenShiChina/books/blob/master/Mastering%20ROS%20for%20Robotics%20Programming.pdf

9 https://github.com/eigendreams/workspace

10 http://wiki.ros.org/joy

11 http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Twist.html

12 http://docs.ros.org/melodic/api/std_msgs/html/msg/Int16.html

13 http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf

14 https://slaterobots.com/tr2

15 http://www.ti.com/lit/ds/spms376e/spms376e.pdf

16 https://ams.com/as5043

17 https://www.melexis.com/en/product/mlx90621/far-infrared-sensor-array-high-speed-low-noise

18 http://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

19 http://www.robotis.us/dynamixel-ax-18a/

20 http://downloads.basicmicro.com/docs/roboclaw_user_manual.pdf

21 http://files.andymark.com/Talon_User_Manual_1_3.pdf

22 http://www.robotpower.com/products/osmc_info.html

23 https://www.cytron.io/c-motor-driver/p-10amp-5v-30v-dc-motor-driver-2-channels?src=category.discovery

24 http://www.ti.com/lit/ug/spmu298d/spmu298d.pdf
25http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-TM4C123G-LaunchPad/TM4C123G_LaunchPad_Workshop_Workbook.pdf

26 https://github.com/mavro10600/catkin_ws

Anexo A Sensores y Módulos de Potencia

Protocolos de comunicación

El microcontrolador TivaC tiene periféricos para comunicarse con otros dispositivos mediante varios protocolos de comunicación; tiene cuatro canales de comunicación SPI, seis de I2C y ocho de UART **[15]**. Los detalles de funcionamiento y modificaciones hechas para adaptar los sensores y microcontroladores están explicados en el Apéndice B. También, los detalles de los protocolos de comunicación utilizados están en el Anexo C. En la Figura 1.3, se listan los protocolos de comunicación utilizados y los elementos de hardware que usan dicho protocolo, así como su conexión.

A continuación, se listan los elementos utilizados:

-Sensor AS 5043 **[16]**
-MLX90621 16x4 IR **[17]**
-MPU6050 **[18]**
-Dynamixel AX18A **[19]**

Etapa de potencia

El control de motores de corriente directa se hace mediante circuitos diseñados específicamente para cumplir esa función, a estos elementos se les conoce con el nombre de etapa de potencia, comúnmente tienen como entrada una fuente de voltaje y una señal digital de control, la cual está compuesta generalmente por una señal de PWM la cual puede estar acompañada por señales digitales de habilitación (enable) y de dirección de giro (DIR); sin embargo, no es la única forma. Algunos pueden comunicarse mediante algún otro protocolo de comunicación como los vistos anteriormente en los sensores, y a la salida se conectan al motor que se quiere controlar. Para el sistema de tracción el robot FinDER v3 se tomaron elementos que ya existían en la versión anterior como son los dos OSMC que sirven para controlar los motores de tracción principal, se agregaron las tarjetas roboclaw de 2x30A y 2x7A para controlar la mayoría de los motores de los brazos auxiliares y del brazo manipulador del robot.

A continuación se listan los elementos utilizados:

-RoboClaw **[20]**
-TALON SR **[21]**
-OSMC **[22]**
-Cytron **[23]**

Apéndice FinDER v3. Detalles de la implementación de los
sensores del robot

Para asegurar un buen funcionamiento del robot, fue necesario obtener información de algunas condiciones específicas de los elementos de hardware del sistema. En este apartado se habla de los detalles de la implementación del sensor AS5043, el sensor IMU MPU6050 y el sensor de arreglo infrarrojo MLX90621. A continuación, se explican los cambios y mejoras realizados en las bibliotecas de comunicación para obtener una señal del sensor con buena velocidad y correcto funcionamiento.

Sensor de posición AS5043

Este sensor denominado 10 bits Sinchronous Serial Interface AS5043, es un arreglo de sensores de efecto Hall colocados en una disposicion circular alrededor del centro del dispositivo y da una salida en voltaje que representa el campo magnético perpendicular a la superficie del integrado.

Hay un diagrama de tiempos que explica la transmisión de datos, si el pin CSn pasa a activo bajo, el pin de DO pasa de su estado de alta impedancia a un uno lógico y la secuencia de lectura inicia. Después de un tiempo mínimo t(CLK FE) los datos son almacenados en el registro de corrimiento de salida en el primer flanco de bajada de CLK.

Cada subsecuente flanco de subida del reloj CLK envía un nuevo bit de datos, la palabra completa serial contiene 16 bits, los primeros 10 bits son la información angular D[0:9], los siguientes 6 bis contienen información del sistema acerca de la validez de los datos OCF, COF,LIN, Paridad y estatus del campo magnético (incremento, decremento fuera de rango).
Un nuevo muestreo puede iniciarse con un pulso en alto del pin CSn con una duración mínima de t(CSn). Y el término del envío de datos puede terminarse en cualquier momento poniendo el pin CSn en alto.

Pin de Mode Input

La posición angular absoluta es muestreada a una frecuencia de 10.4 kHz (t=96 us) en fast mode (modo de rápida frecuencia de transferencia) y a una frecuencia de 2.6 kHz (t=384 us) en slow mode (modo de baja frecuencia de transferencia).
Estos modos son seleccionados con el pin de Mode Input al inicializarse el sensor este pin activa o desactiva un filtro interno que reduce el ruido . Si se pone el pin en cero, se activa el filtro y el tiempo de muestreo baja a 2.6kHz y el tiempo de retraso de

propagación de la señal incrementa a 384 us Este modo de funcionamiento es recomendado para aplicaciones de alta precision y baja velocidad en donde el desplazamiento angular es menor a 360 grados.

Si se pone el pin en alto, se incrementa la velocidad de muestreo a 10.4 kHz y el retardo de propagación de la señal baja a 96 us . Se recomienda para aplicaciones de alta velocidad con desplazamientos de más de 360 grados o de rotación continua.

Tiempos:
t(DO) = 100 ns Tiempo entre el flanco de bajada de CSn y la activación de la salida
t(CLK FE)=500 ns Tiempo entre el flanco de caida de CSn y el primer flanco de bajada de CLK
T clk/2 = 500 ns Flanco de subida de CLK para lanzar un bit a cada tiempo
t(DO valid)=413 ns Máximo tiempo entre el flanco de levantamiento cd CLK y que la salida de datos sea válida
t(DO tristate)= 100 ns tiempo máximo después del cual el pin DO regresa a alta impedancia
t (CSn)=500 ns CSn=high Tiempo mínimo para inicia una lectura nueva
f(CLK)= 1 MHz frecuencia máxima de lectura de datos

Tomando en cuenta lo anteriormente descrito, se llevaron a cabo pruebas con el sensor. No obstante, dada la sencilla implementación del protocolo de comunicación SPI, se hizo manualmente una función que lee el sensor sin problemas, dado que en el momento de su implementación se desconocía el funcionamiento completo de la biblioteca SPI.h, y se prefirió tener una función que permitiera la conexión de varios sensores ne líneas separadas, aunque ello representaba un desperdicio de pines, dado que se trataba de un trabajo urgente, se hizo de esa manera en su momento y funcionó correctamente. Ahora, una vez estudiado a profundidad el protocolo de comunicación, y las funciones de la biblioteca SPI.h, se hizo una implementación elegante para utilizar los sensores. Además, que concluí que la decisión tomada en un inicio fue correcta, dado que dicha biblioteca debía modificarse para su correcto funcionamiento con el sensor as5043. Aunque en el diseño de proyecto para microcontroladores generalmente suele ser mal visto el hacer la interfaz manualmente para un periférico, o emularlo con funciones artificiales, que no usan los periféricos del microcontrolador, se debe tomar en cuenta lo siguiente:

-Se debían utilizar entre 3 a 6 sensores en cada microcontrolador

-Cada sensor usa tres pines del microcontrolador, además de VCC y VSS, ya que en el modo de lectura únicamente requiere de estos pines.

-Se tenía cierto temor a conectar mal algún sensor y pudiera afectar los pines de otro sensor o de la línea de comunicación, así que se decidió mantenerlos separados

-La función de emulación de la comunicación SPI es muy sencilla y se muestra a continuación:

```
    digitalWrite(csn, LOW);
  delayMicroseconds(4);
    unsigned int data = 0;
  digitalWrite(clk, LOW);
  delayMicroseconds(1);
  for (uint8_t k = 0; k < 16; k++)
    {
    digitalWrite(clk, HIGH);
    delayMicroseconds(1);
    data = (data << 1) | (digitalRead(d0) ? 0x0001 : 0x0000);
    digitalWrite(clk, LOW);
    delayMicroseconds(1);
     }
  digitalWrite(csn, HIGH);
  delayMicroseconds(1);
  digitalWrite(clk, HIGH);
```

En el uso del sensor as5043 con la biblioteca SPI.h, se encontró que debe de modificarse el archivo de la biblioteca para el correcto funcionamiento del sensor, esto es porque la esta biblioteca inicializa el periférico SSI (Synchronous Serial Interface) a una frecuencia de reloj de 4 MHz, y aunque se podría pensar en modificar dicha velocidad de transmisión usando la función setClockDivider() tal como lo recomienda la página de Sparkfun, se debe de tomar en cuenta que la Tivac funciona diferente en muchos sentidos y que en la linea 198 de la biblioteca SPI.cpp se tiene el siguiente segmento de código

```
#if defined(TARGET_IS_BLIZZARD_RB1)
      ROM_SSIConfigSetExpClk(SSIBASE,              ROM_SysCtlClockGet(),
SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 4000000, 8);
```

En la función anterior se tiene acceso al mismo registro que en la función setClockDivider(), para alcanzar la velocidad de transmisión del quinto argumento de la función SSIConfigSetExpClk(), entonces si dicho argumento es 4000000 quiere decir que modificarán el mismo registro que en setClockDivider() para alcanzar dicha velocidad de transmisión. Entonces, es en dicha línea donde hay que cambiar el 4000000 por 1000000 para el correcto funcionamiento del sensor. Se hizo un programa de prueba al respecto llamado spi_as5043 donde se puede hacer la comprobación de todo lo descrito hasta ahora.

El siguiente paso lógico es probar con dos o más sensores en la misma línea, cada uno con un pin de selección separado para hacer la selección del esclavo, y recordar dejar libre el pin PB7 que es el MOSI del bus SPI que se está ocupando.

Sensor IMU MPU6050

En la implementación del sensor MPU6050 con el microcontrolador TivaC se tuvieron dos problemas principalmente, uno asociado a la biblioteca Wire.h proveída por el proyecto de software libre Energia, en la cual se detectó un problema que impedía la correcta inicialización del bus I2C y evitaba la comunicación entre el microcontrolador y el sensor. Y el segundo problema se dio debido a la conFiguración del FIFO del MPU en la biblioteca MPU6050_6axis_motionapps20.h que provocaba una frecuencia de transmisión de datos de 20 hz del FIFO al microcontrolador y que provocaba que la salida de la IMU aparecía cada 50 ms. La documentación de ambas soluciones se incluye a continuación:

Para habilitar el modulo I2C, la biblioteca Wire.h hace uso del bus I2C el cual corresponde a los pines PA6 y PA7; sin embargo para habilitar alguna otra linea de I2C (la TivaC tiene 4) hay que modificar la biblioteca. Esencialmente el problema pasando el proyecto a Code Composer, entorno de desarrollo para microcontroladores de Texas Instruments que permite hacer depuración directamente en el microcontrolador, donde se tiene acceso a los registros del mismo y se puede observar que está sucediendo en los registros asociados al periférico I2C. Entonces, en la función ::begin() de la biblioteca Wire.h, se encontró el problema; había que poner un retardo después de habilitar el periférico antes de empezar a configurar los pines a utilizar como SDA y SCL. A continuación, se muestra el fragmento de código que se modificó:

```
void TwoWire::begin(void)
{

  if(i2cModule == NOT_ACTIVE) {
      i2cModule = BOOST_PACK_WIRE;
  }

  ROM_SysCtlPeripheralEnable(g_uli2cPeriph[i2cModule]);

  //Configure GPIO pins for I2C operation
  ROM_GPIOPinConfigure(g_uli2cConfig[i2cModule][0]);
  ROM_GPIOPinConfigure(g_uli2cConfig[i2cModule][1]);
  ROM_GPIOPinTypeI2C(g_uli2cBase[i2cModule],
g_uli2cSDAPins[i2cModule]);
....
```

La instrucción ROM_SysCtlPeripheralEnable(g_uli2cPeriph[i2cModule]); es la que habilita el periférico en el microcontrolador y, como puede observarse, inmediatamente después, comienza la configuración de los pines para operar como SDA y SCL, en Code Composer, cada vez que se habilita un periférico o un puerto se tiene acceso de

lectura escritura a los mismos, y se pueden monitorear los valores de los registros asociados. La documentación al respecto **[24]** indica que se debe de esperar cuando menos tres ciclos después de habilitar un periférico antes de comenzar a modificar los registros. Y esto se hace usando una instrucción que pregunta si el periférico está habilitado, y gastando ciclos de reloj hasta que se cumpla la condición, otra forma de hacerlo es mediante interrupciones, pero el procedimiento para dar de alta una interrupción y su función asociada es algo complicado en comparación **[25]** , entonces, la instrucción a usar después de habilitar el periférico es :

```
while (!ROM_SysCtlPeripheralReady(g_uli2cPeriph[i2cModule]))
{}
```

Que espera a que se habilite el periférico.

El otro problema que se presentó, es que el tiempo de actualización del FIFO del DMP estaba a 20 hz, que sumado a las demás operaciones realizadas en el microcontrolador, hacían que el tiempo de ejecución de cada ciclo llegara a tardar hasta 1 segundo, lo cual es inaceptable en un sistema que requiere un tiempo de ejecución mas rápido.

El problema se rastreó en la biblioteca del sensor MPU6050, siguiendo los pasos del setup del DMP en la MPU se logró encontrar en la biblioteca MPU6050_6axis_motionapps20.h en la función ::dmpInitialize(), en la línea de código mostrada a continuación, dónde se realiza una operación dónde se configura el funcionamiento del DMP en los registros de la MPU:

```
uint8_t test1 = writeProgDMPConFigurationSet(dmpConfig,
MPU6050_DMP_CONFIG_SIZE);
```

Aquí, en este punto, se encontró la razón de porque iba tan lenta la frecuencia de publicación de datos, en la línea 305 del arreglo dmpConfig define los valores a cargar en los registros de la mpu, un parámetro llamado fifo rate, está en 0x09, pero si lo cambiamos a 0x01 o a 0x00 se incrementa la frecuencia a 100hz y a 200 hz respectivamente.

Arreglo infrarrojo MLX90621

El programa ya existía para su implementación en un microcontrolador Arduino; sin embargo en mi opinión estaba muy mal hecho y la biblioteca que utilizaba para la comunicación I2C es arcaica y carecía de generalidad para su implementación en otro microcontrolador que no fuera de Atmel, entonces, hice la adaptación del programa para que utilizara la biblioteca Wire (usada para comunicación I2C) de tal manera que sea más general su uso en la TivaC, el funcionamiento estándar del protocolo de

comunicación I2C con el sensor tuvo que modificarse la biblioteca Wire para el correcto funcionamiento, se debe tener cuidado en el protocolo de comunicación con este sensor ya que es muy sensible y su funcionamiento es algo fuera de lo común en una implementación de este tipo.

Finalmente se creó una biblioteca para leer de forma sencilla el sensor, en la función de adquisición del vector de lecturas de la matriz de IR existe un problema debido a que la función Wire.requestFrom(int address,int length) solo almacena un buffer de 62 valores, cuando el vector deseado es de 128 valores, en la biblioteca no encontré una forma de modificarla para que entre una y otra lectura del buffer no se reiniciara la comunicación, que es el principal problema. Entonces, para solucionarlo, se agregaron tres funciones a la biblioteca para que hicieran la toma de lecturas de forma directa y más general:

En Wire.h:

```
uint8_t requestInit(uint8_t); //funcion para iniciar a recibir de forma fácil
uint8_t requestCont(uint8_t); //funcion para continuar la recepcion e forma fácil
uint8_t requestEnd(uint8_t); //funcion para continuar la recepcion e forma fácil
```

En Wire.cpp:

```
uint8_t TwoWire::requestInit(uint8_t address)
      {
            uint8_t dato;
            ROM_I2CMasterSlaveAddrSet(MASTER_BASE, address, true);
            ROM_I2CMasterControl(MASTER_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_START);
      while (ROM_I2CMasterBusy(MASTER_BASE))
      {}
            dato=ROM_I2CMasterDataGet(MASTER_BASE);

            return dato;
            }


            uint8_t TwoWire::requestCont(uint8_t address)
            {
                  uint8_t dato;
              ROM_I2CMasterSlaveAddrSet(MASTER_BASE, address, true);
```

```
              ROM_I2CMasterControl(MASTER_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_CONT);
      while (ROM_I2CMasterBusy(MASTER_BASE))
      {}
            dato=ROM_I2CMasterDataGet(MASTER_BASE);
            return dato;
            }


            uint8_t TwoWire::requestEnd(uint8_t address)
            {
                              uint8_t dato;
              ROM_I2CMasterSlaveAddrSet(MASTER_BASE, address, true);
              ROM_I2CMasterControl(MASTER_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
      while (ROM_I2CMasterBusy(MASTER_BASE))
      {}
            dato=ROM_I2CMasterDataGet(MASTER_BASE);

            return dato;
            }
```

Ambas soluciones vienen de depurar usando Code Composer , el código está en el repositorio**[26]**, se llama i2cmavro el proyecto y funciona bien, aunque deberían de agregarse funciones de error imitando la función Wire::requestFrom().

Apéndice B Esquemas de las tarjetas impresas del robot

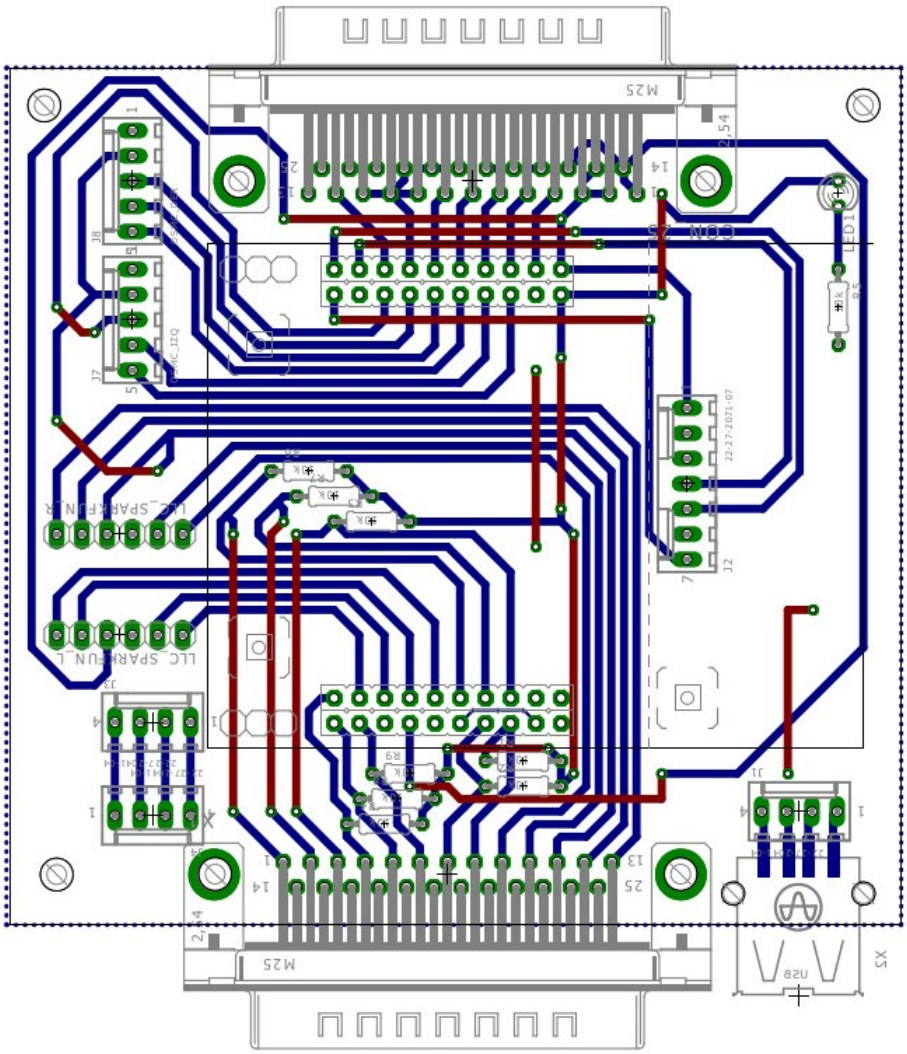

Figura B.1 Diseño de la tarjeta impresa para el módulo de tracción

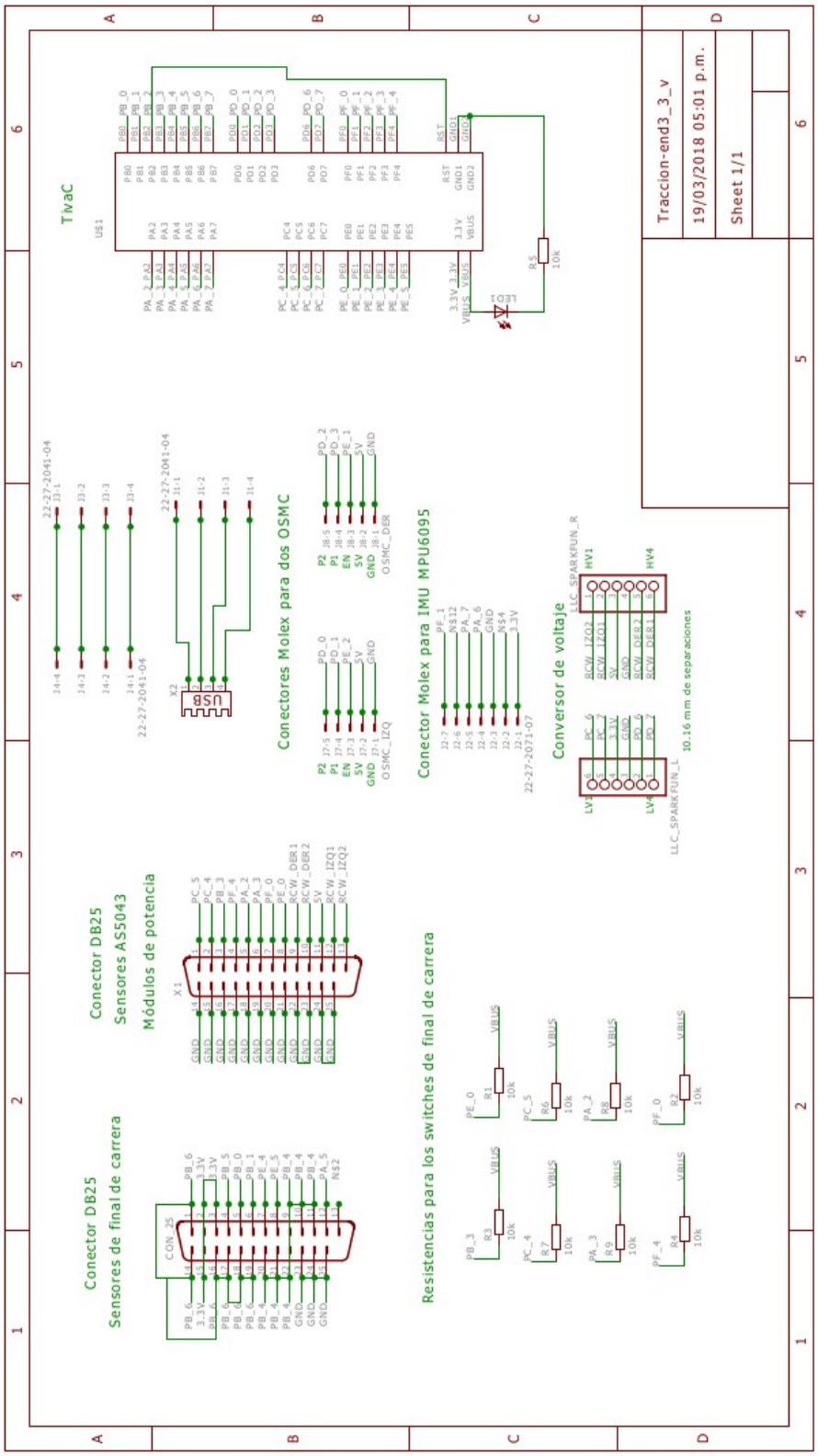

Figura B.2 Diseño esquemático de la tarjeta impresa para el módulo de tracción

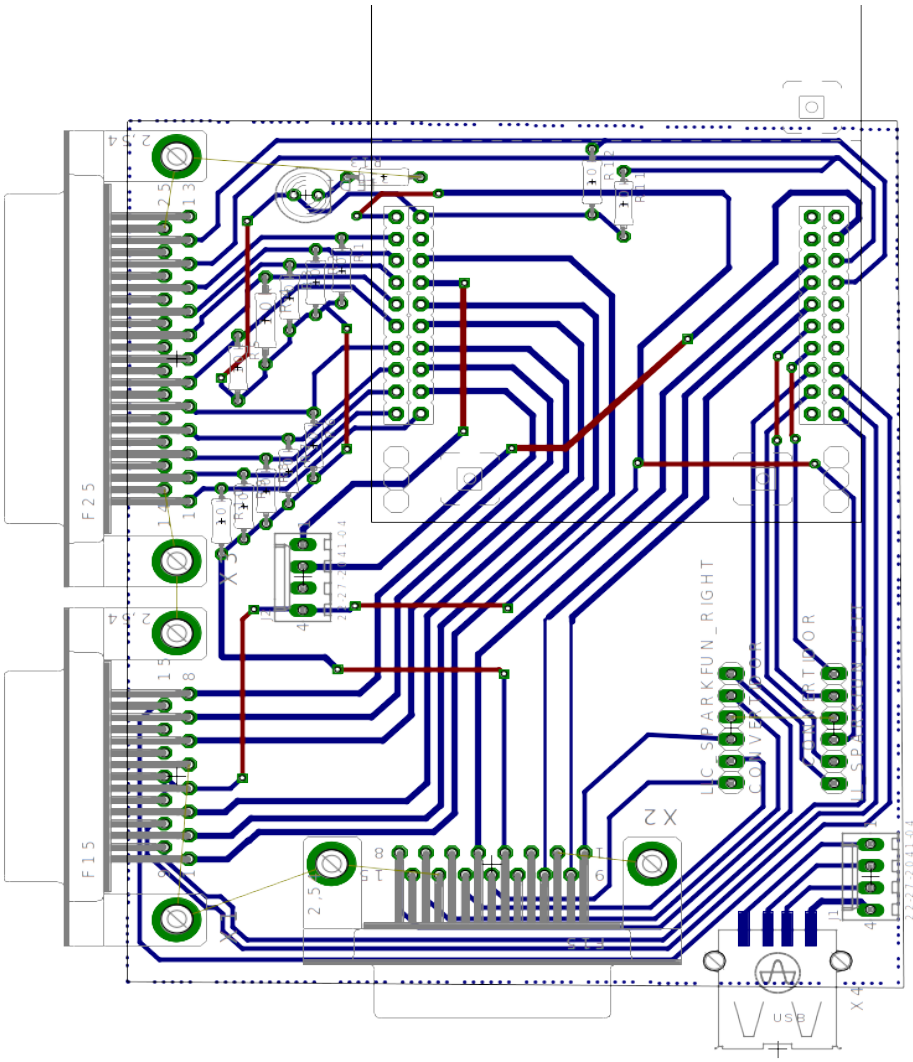

Figura B.3 Diseño de la tarjeta impresa para el módulo del brazo

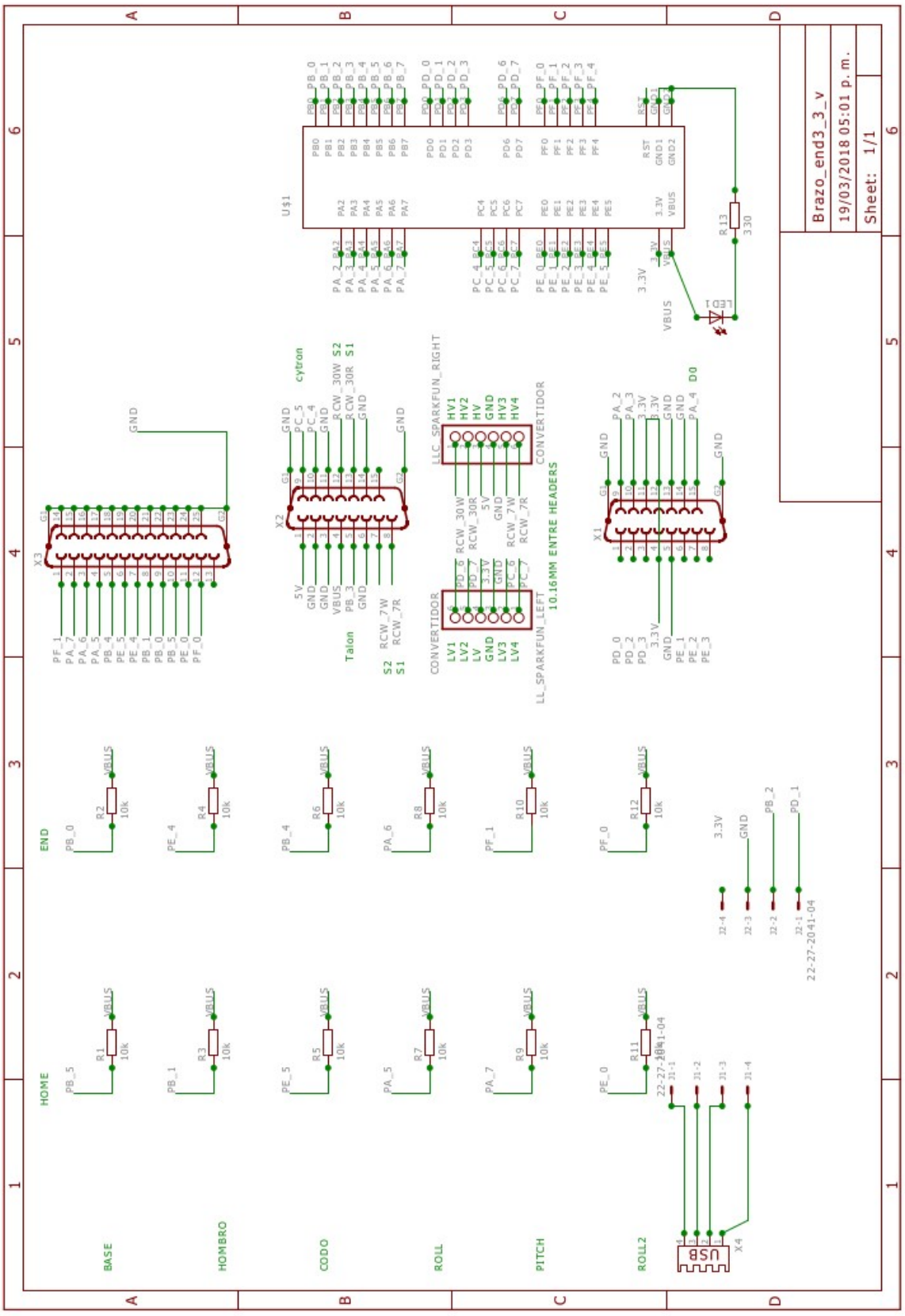

Figura B.4 Diseño esquemático de la tarjeta impresa para el brazo

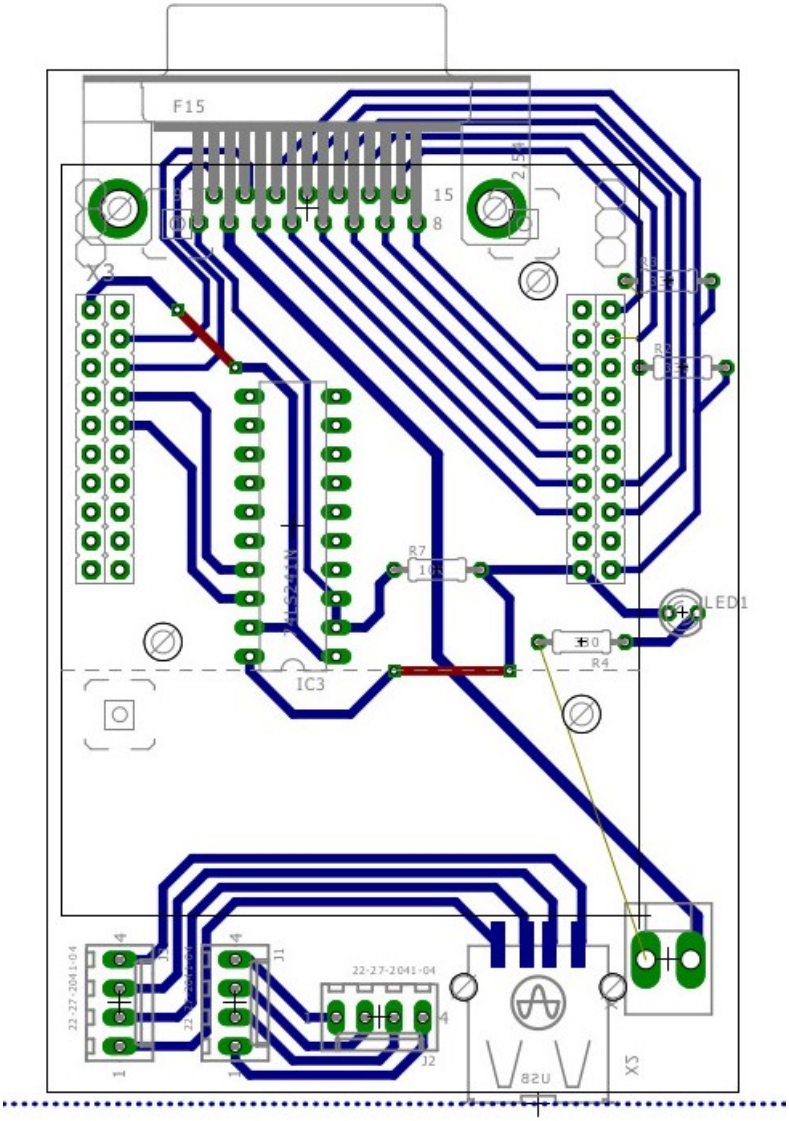

Figura B.5 Diseño de la tarjeta impresa para el módulo de sensores del efector final

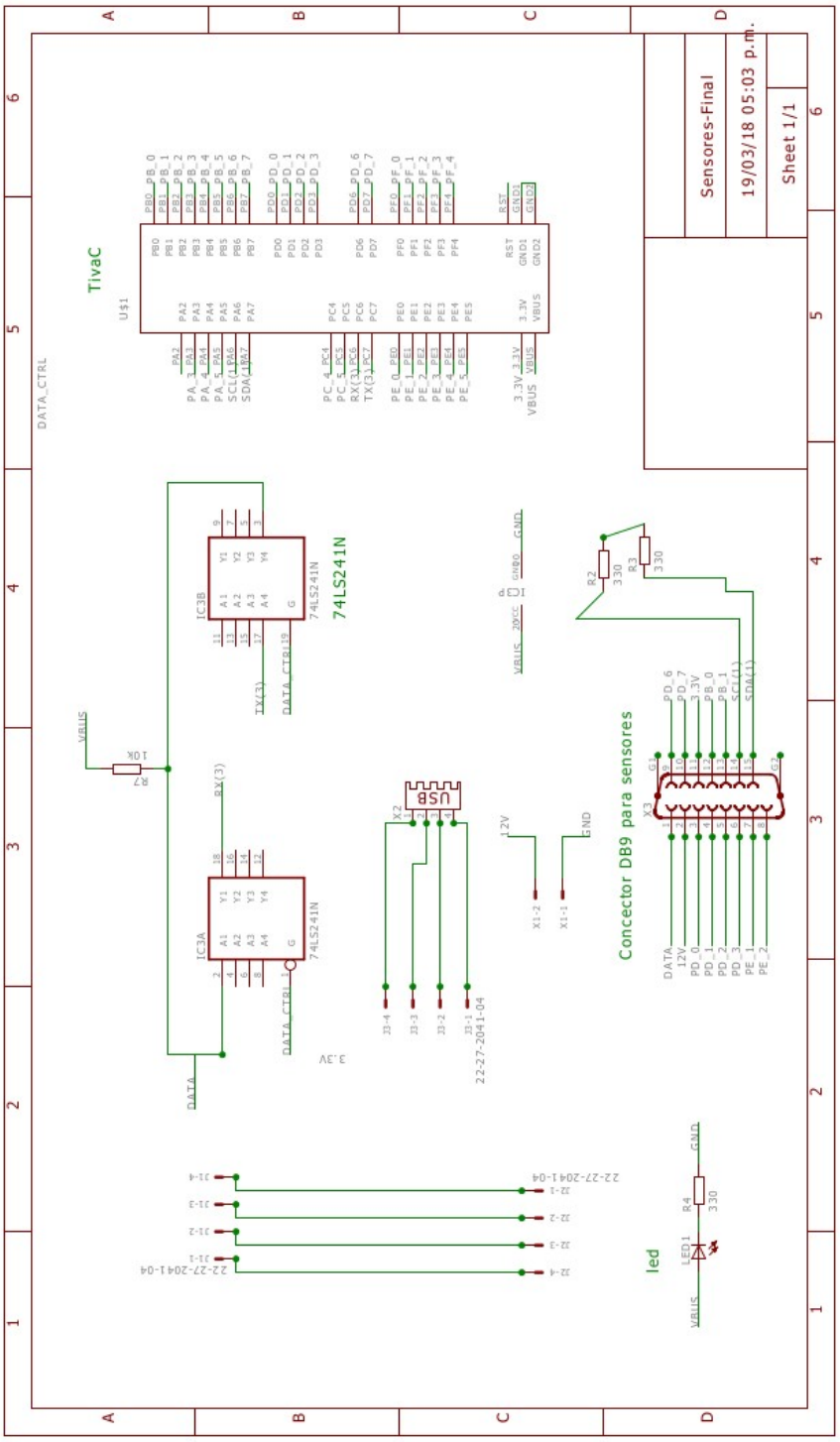

Figura B.6 Diseño esquemático de la tarjeta impresa para los sensores del efector final