ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo de algoritmos de control y movimiento de robots mediante LabVIEW Robotics.

AUTOR: Alejandro Alcaraz Salvago
DIRECTORES: Francisco José Ortiz Zaragoza
Pedro Javier Navarro Lorente

Septiembre / 2012

Autor	Alejandro Alcaraz Salvago
E-mail del Autor	Alejandro.alcaraz.salvago@gmail.com
Director(es)	Francisco José Ortiz Zaragoza Pedro Javier Navarro Lorente
Título del PFC	Desarrollo de algoritmos de control y movimiento de robots mediante LabVIEW Robotics.
Descriptores	

Resúmen

Este Proyecto Fin de Carrera tiene como objetivo implementar una serie de algoritmos para el control y movimiento autónomo de un robot bajo la herramienta de diseño y programación LabVIEW junto a su módulo Robotics.

A su vez, se introducirá el uso de RobotSim, herramienta que en conjunto con LabVIEW Robotics se emplea para la simulación en entornos 3D del código en un robot "virtual".

Se ha trabajado sobre el modelo de robot Robotics Starter Kit, de la compañía National Instruments, instalando el hardware adicional que se ha juzgado necesario para llevar a cabo los objetivos propuestos.

Finalmente, se ha desarrollado un algoritmo de navegación para el Starter Kit. Para su funcionamiento se parte de un mapa y posición conocidos, sobre los cuales el robot trazará una ruta hasta la posición "meta" que se le haya indicado, valiéndose de la información proveniente de sus sensores.

Titulación	Ingeniero de telecomunicación
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre - 2012

ÍNDICE

CAPÍTULO 1

OBJETIVOS DEL PROYECTO

1.1.- Introducción

En este capítulo plantearemos los objetivos del presente proyecto fin de carrera, además de describir los capítulos de los que estará compuesto.

En el ámbito académico el montaje y programación de robots, con fines tanto de investigación como de aprendizaje para los alumnos, son campos de gran aceptación y constante crecimiento. Siendo conscientes de ello, en National Instruments decidieron integrar en su entorno Labview diversas herramientas dirigidas a la programación en el campo de la robótica, ofreciendo como principal reclamo la facilidad de uso, debido a su particular entorno de desarrollo gráfico.

Este proyecto surge con la finalidad de comprobar el potencial de Labview en el campo de la robótica, estableciendo como meta final la elaboración de un algoritmo de navegación autónomo.

1.2.-Objetivos

Este Proyecto Fin de Carrera tiene como objetivo implementar una serie de algoritmos para el control y movimiento autónomo de un robot bajo la herramienta de diseño y programación Labview junto a su módulo Robotics.

A su vez, se introducirá el uso de RobotSim, herramienta que en conjunto con Labview Robotics se emplea para la simulación en entornos 3D del código en un robot "virtual".

Finalmente, se desarrollará un algoritmo de navegación para el modelo Labview Starter Kit 1.0, mostrado en la Figura 1.1. En éste, se partirá de un mapa y posición conocidos, sobre los cuales el robot trazará una ruta hasta la posición "meta" que se le haya indicado, valiéndose de la información proveniente de sus sensores.

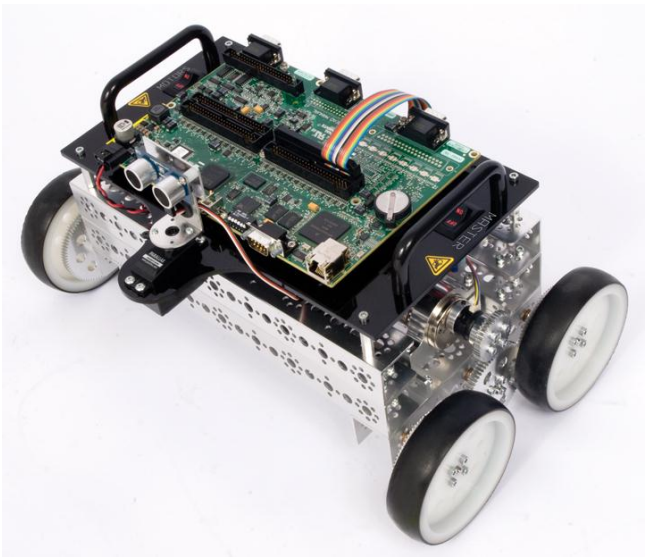

Fig. 1.1. Labview Robotics Starter Kit

El Starter Kit es un robot diferencial fabricado por National Instruments. Dispone de dos motores (izquierdo y derecho) provistos de sendos encoders, y un sensor de ultrasonidos montado en el frontal sobre un servo, el cual le permite rotar en un ángulo de 180º (simulando de esta forma la labor de un radar).

A esto le sumaremos:

- Una brújula digital, para conocer en todo momento la orientación.
- Dos interruptores de fin de carrera en las dos esquinas frontales, que avisarán al robot de posibles colisiones.
- Router Wifi instalado en la parte trasera, que permitirá la comunicación con el Starter Kit sin necesidad de cables.

Los objetivos a cumplir en el proyecto serán, en resumen:

1. Estudio del módulo LabVIEW Robotics.
2. Puesta en marcha del Starter Kit y programación básica con ejemplos del fabricante.
3. Implementación de nuevo hardware en el robot.
4. Desarrollo de aplicaciones con más posibilidades al emplear los nuevos recursos.
5. Depuración de programas y puesta en marcha de demostraciones

1.3.- Descripción de los capítulos del proyecto

El proyecto se estructurará en los siguientes capítulos:

1º Capítulo: Objetivos del Proyecto.

En este capítulo se describirán los objetivos que se buscan alcanzar con la realización del proyecto y los temas que se van a tratar en cada capítulo.

2º Capítulo: LabVIEW aplicado a la robótica.

Se realizará una breve presentación de la herramienta a emplear junto al módulo Robotics, incluyendo a continuación algunos programas de ejemplo descritos en detalle, para finalizar con la integración en LabView del simulador RobotSim.

3º Capítulo: LabVIEW Robotics Starter Kit.

Se describirá el robot empleado para la programación del algoritmo de navegación, *Labview Starter Kit "DaNI"*, así como las modificaciones que se le han introducido tanto a nivel hardware como software para dar soporte a los nuevos componentes.

4º Capítulo: Algoritmos de navegación.

A lo largo de este capítulo se desarrollarán dos algoritmos: un sencillo programa de control mediante teleoperación desde un PC remoto, y el algoritmo de navegación autónomo sobre un mapa conocido.

5º Capítulo: Pruebas de Campo.

Se describirán las pruebas realizadas a partir de los algoritmos de navegación, y los resultados ofrecidos.

6º Capítulo: Conclusiones y Futuros Proyectos.

Se analizará el grado de consecución de los objetivos y posibles proyectos y aplicaciones que puedan originarse a partir del presente.

CAPÍTULO 2

LABVIEW APLICADO A LA ROBÓTICA

2.1.- Introducción

LabVIEW es un lenguaje de programación visual que emplea iconos en lugar de líneas de código para crear aplicaciones. Fue creado por National Instruments en 1976, siendo la versión que emplearemos la publicada en 2010. La principal diferencia con lenguajes basados en texto es que, en lugar de que una serie de instrucciones determinen la ejecución del programa, es el flujo de datos el que la dicta, lo que viene a ser llamado dataflow programming.

En concreto, la ejecución viene supeditada a la estructura de un diagrama de bloques, el cual se crea a partir de la interconexión de una serie de funciones a través de cables, los cuales se encargan de propagar tanto variables como código tan pronto como se encuentren disponibles en las respectivas fuentes. Debido a la naturaleza de su estructura, permite ejecución en paralelo.

La metodología de trabajo de LabVIEW parte de la creación de programas llamados VIs (Virtual Instruments), los cuales se encuentran constituidos por tres elementos: un panel frontal, un diagrama de bloques, y un panel de conexiones.

El panel frontal es una interfaz de usuario en forma de tablero de instrumentos virtual, conteniendo las interfaces de entrada y salida de nuestro código, denominadas mandos e indicadores, con los cuales se puede interactuar durante la ejecución del programa. LabVIEW ofrece una serie de funciones específicas para tal cometido, contenidas en la paleta de herramientas Controls, accesible únicamente desde el panel frontal.

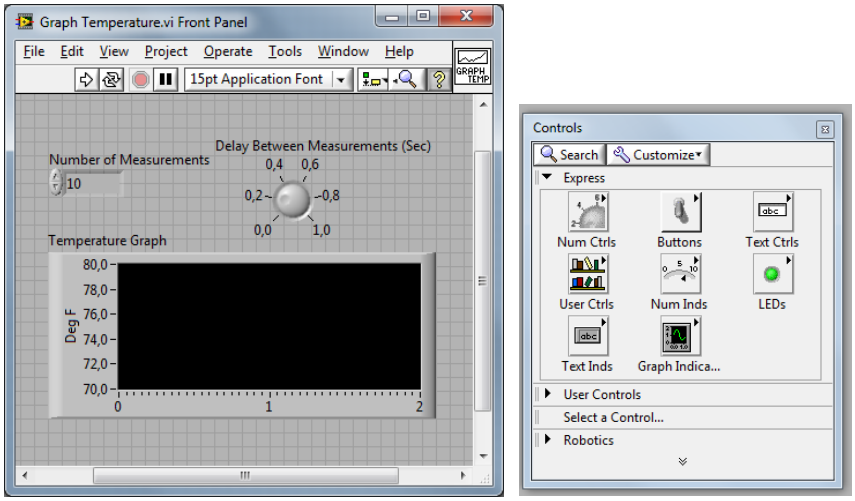

Figura 2.1. VI de ejemplo: Panel frontal (izquierda) y paleta de mandos (derecha)

La programación en sí se realiza sobre el diagrama de bloques, donde se introducen e interconectan los iconos gráficos que representan las funciones, y a partir de las cuales a su vez se obtienen los datos de entrada y salida del panel frontal. A través de la paleta de herramientas Functions podremos acceder a las librerías de funciones disponibles, así como buscar nuevas creadas por otros usuarios en el repositorio que mantiene National Instruments.

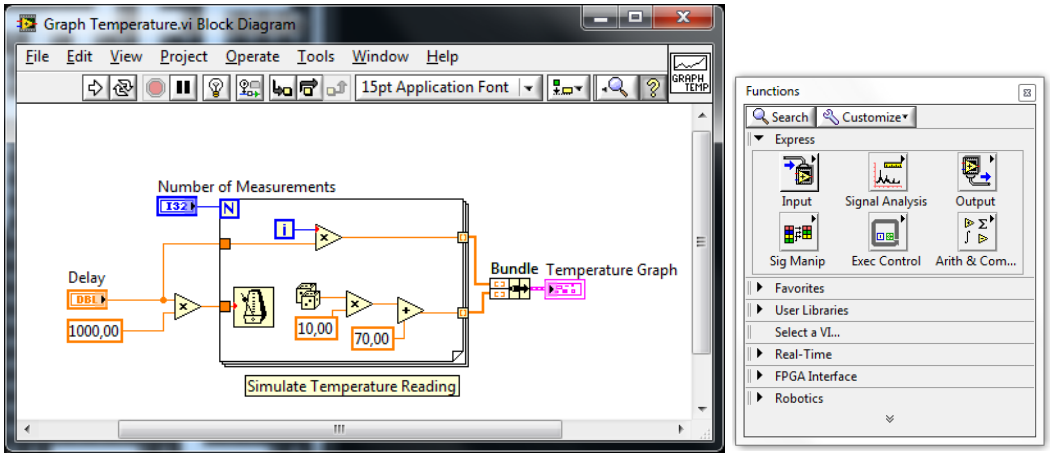

Figura 2.2. VI de ejemplo: Diagrama de bloques (izquierda) y panel de funciones (derecha)

El panel de conexiones se emplea en el caso de que se desee llamar el VI desde otro programa, definiendo tanto su representación gráfica (icono) como sus entradas y salidas, las cuales pueden personalizarse a partir de las que se hayan definido en el panel frontal del VI original.

El módulo LabVIEW Robotics se instala sobre la versión correspondiente de LabVIEW, expandiendo sus posibilidades en el campo de la robótica añadiendo funciones destinadas a facilitar la programación en dicho ámbito (desde algoritmos predefinidos a la interconexión con el hardware más extendido) y la herramienta Robotics Wizard, que permite crear y administrar un proyecto Robotics completo, ofreciendo soporte para diversas plataformas.

En este capítulo entraremos en detalle sobre dichas funcionalidades, realizaremos un ejemplo simple que nos ayudará a entender la filosofía a seguir a la hora de programar bajo esta plataforma, y presentaremos un simulador 3D para el que National Instruments ha añadido recientemente soporte en sus proyectos, Cogmation RobotSim.

2.2.- Requisitos de Software

Para trabajar con LabVIEW Robotics, es necesario un paquete de software específico. A continuación se listan los módulos recomendables para cubrir la mayor parte de posibilidades que ofrece Robotics:

- LabVIEW 2010
- LabVIEW Robotics 2010
- LabVIEW Control Design and Simulation
- LabVIEW FPGA
- LabVIEW MathScriptRT
- LabVIEW NI SoftMotion
- LabVIEW NI Vision Development
- LabVIEW Real-Time
- LabVIEW Statechart
- LabVIEW PID and Fuzzy Logic Toolkit
- System Identification Toolkit
- Instrument Drivers: NI-RIO 3.6.0
- Instrument Drivers: NI-VISA

Para la simulación existen diversas alternativas. En nuestro caso se ha testeado el paquete de Cogmation, que consiste en el siguiente software:

- RobotSim
- RobotBuilder

2.3.- Herramientas exclusivas de LabVIEW Robotics

El primer elemento a reseñar de las funcionalidades introducidas en el módulo Robotics, es la nueva librería de controladores y funciones específicamente diseñadas para la programación de robots. A continuación las describiremos pormenorizadamente, con el objetivo de dar una visión general de las posibilidades ofrecidas por la herramienta.

2.3.1.- Paleta de Mandos

A la paleta Controls en Labview se accede desde el Panel Frontal. En la versión 2010 se incluyeron únicamente dos indicadores orientados a la robótica, aunque las librerías existentes con anterioridad proveen de todos los recursos necesarios.

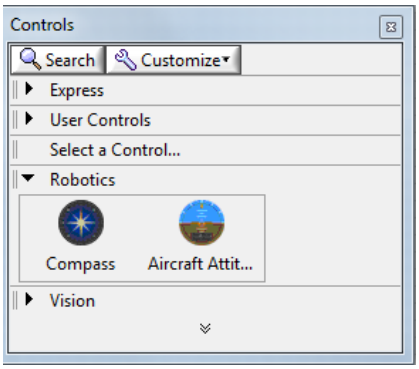

Figura 2.3. Contenido de la Paleta de Control Robotics

Estos indicadores se encuentran orientados a la navegación:

- Compass (Brújula): Permite representar la orientación actual de forma visual. Se le introducen los datos mediante una entrada de formato double.

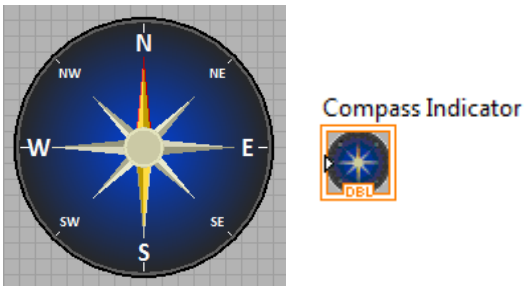

Figura 2.4. Instrumento brújula (izquierda) y su icono de función (derecha)

- Aircraft Attitude Indicator (Indicador de altitud para aeronaves): Reproduce la función del instrumento real. Los datos se introducen mediante un Cluster que contiene dos elementos, Roll y Pitch, ambos en formato double.

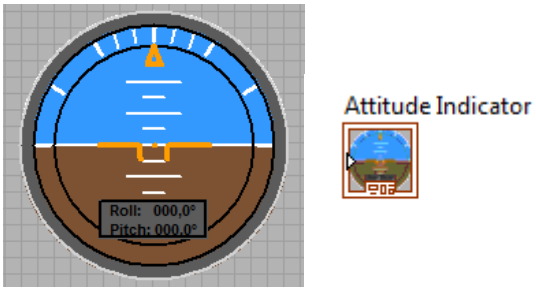

Figura 2.5. Instrumento AAI (izquierda) y su icono de función (derecha)

2.3.2.- Paleta de Funciones

La librería correspondiente a la paleta de funciones se presenta bastante completa, añadiendo ocho secciones que repasaremos a continuación.

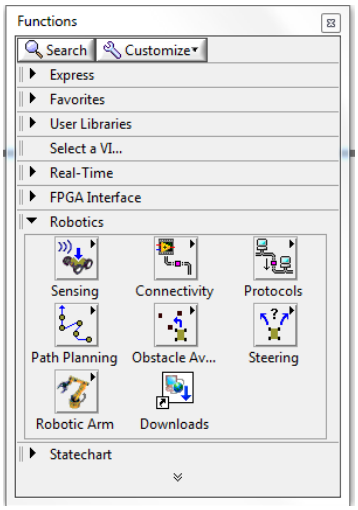

Figura 2.6. Contenido de la Paleta de Funciones Robotics

2.3.2.1.- Sensing

Funciones empleadas para configurar, controlar y recuperar datos desde instrumentos sensores. Cada tipo de sensor mostrado en la captura dispone a su vez de funciones exclusivas para modelos reales disponibles en el mercado, como puede verse en el caso del LIDAR. Es necesario, pues, tener definido previamente el hardware sobre el que se va a trabajar. Aparte de los incluidos por defecto, es posible buscar e instalar controladores de instrumentos adicionales mediante la paleta Instrument I/O, o el NI Instrument Driver Finder.

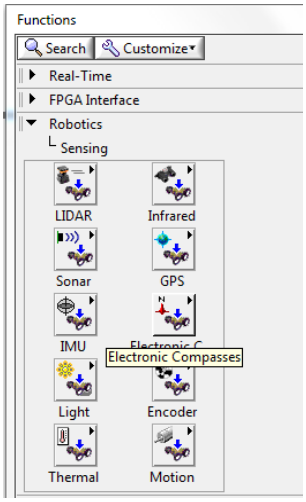

Figura 2.7. Contenido de la Paleta de Funciones Robotics/Sensing

A continuación se listan los modelos para los cuales Robotics ofrece por defecto drivers y módulos específicos para utilizarlos con LabView, clasificados por el tipo de sensor:

- LIDAR: Series de modelos: *Hokuyo URG*, *Sick LMS 2XX*, y *Velodyne HDL-64E S2*.
- Telémetro por infrarrojos: Modelos *Sharp IR GP2D12* y *Sharp IR GP2Y0D02YK*
- Telémetro Sonar: Modelos *Devantech SRF02*, *Devantech SRF05*, y *MaxSonar EZ1*.
- GPS: Modelos *Applanix POS LV*, *NavCom SF-2050*, y las series *Garming GPS* y *u-blox 5*.
- IMU: Serie *Crossbow NAV440*, modelos *MicroStrain 3DM-GX1*, *MicroStrain 3DM-GXx*, y *Ocean Server OS4000*.
- Brújula electrónica: Modelo *Devantech CMPS03*.
- Sensor de luz ambiente: Modelo *Vishay TEMT6000*.
- Encoder: Modelo *Maxon Encoder*.
- Térmico: Modelo *Devantech TPA81*.
- Movimiento: Modelos *Dynamixel Motor*, *Lynxmotion SSC32*, *TI MDL BDC24*.

2.3.2.2.- Connectivity

Funciones empleadas para comunicarse con otros softwares de robótica. A día de hoy, Robotics trae por defecto funciones destinadas a la conectividad de Skilligent, MobileRobots, y Cogmation RobotSim. En los ejemplos también es posible encontrar un proyecto que presenta conectividad con el simulador de Microsoft Robotics Developer Studio, aunque no tiene soporte oficial. Es posible que se añada en un futuro.

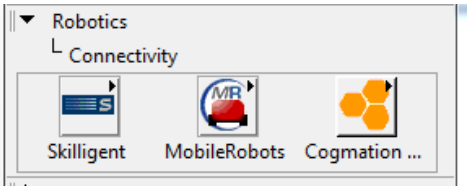

Figura 2.8. Contenido de la Paleta de Funciones Robotics/Connectivity

En nuestro caso, esta opción muestra su mayor relevancia en la posibilidad de emplear el simulador *RobotSim*, en el que entraremos en detalle más adelante.

2.3.2.3.- Protocols

Funciones empleadas para procesar datos formateados en protocolos de comunicación, como datos enviados por sensores. Dispone de funciones para NMEA, FPGA Device Communication, y jTK1.0.1.

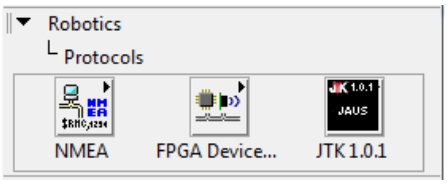

Figura 2.9. Contenido de la Paleta de Funciones Robotics/Protocols

2.3.2.4.- Path Planning

Funciones empleadas para calcular una trayectoria a un punto meta en un mapa que representa el entorno del robot. Nótese que se encuentran implementados los algoritmos de uso común de planificación de ruta A* y AD*., así como funciones para la generación y manejo de Occupancy Grid Maps y Directed Graph Maps. En el capítulo 4 se entrará en detalle con el algoritmo A*.

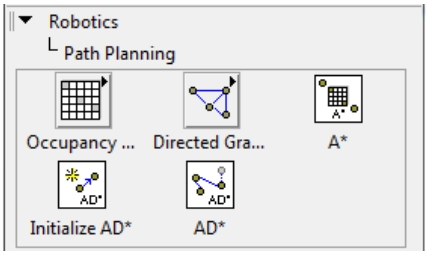

Figura 2.10. Contenido de la Paleta de Funciones Robotics/Path Planning

2.3.2.5.- Obstacle Avoidance

Implementan modelos de identificación de obstáculos para robots móviles. La función Simple VFH será empleada más adelante en este capítulo para elaborar un ejemplo práctico.

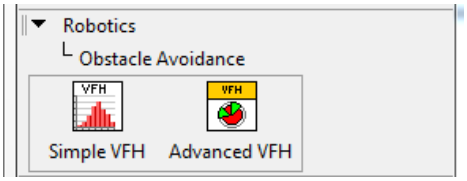

Figura 2.11. Contenido de la Paleta de Funciones Robotics/Obstacle Avoidance

2.3.2.6.- Steering

Funciones para crear e interactuar con los elementos que permiten la movilidad del robot. Permite desde la creación de las ruedas (Create Wheel), a los cálculos necesarios para traducir la información (Ej. El ángulo al que se desea girar) para que la interprete el motor y pueda ajustar su configuración de forma consecuente. Incluye también funciones genéricas para administrar la comunicación con el motor, obtener lecturas del mismo, etc.

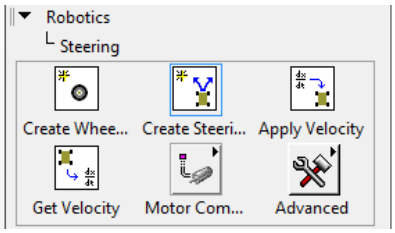

Figura 2.12. Contenido de la Paleta de Funciones Robotics/Steering

2.3.2.7.- Robotic Arm

Posibilita la creación y control de un brazo robot simulado. Realiza cálculos dinámicos y cinemáticos en un brazo, orientado a la generación de prototipos del brazo robot. Incluye: Serial Arm Definition, Kinematics, Dynamics, Homogeneous Transforms, Quaternions, Plots.

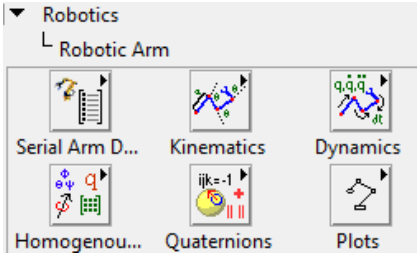

Figura 2.13. Contenido de la Paleta de Funciones Robotics/Robotic Arm

2.3.2.8.- Downloads

Enlaza a la sección NI LabVIEW Robotics Code Exchange de la web de National Instruments, el cual constituye un repositorio de código y artículos donde la comunidad de desarrolladores comparte recursos.

2.4.- Creación de un proyecto Robotics

Trabajar sobre un formato de proyecto en LabVIEW conlleva ciertas ventajas si se pretende acometer un trabajo de cierta complejidad. Para empezar, todos los VIs empleados para el proyecto, así como todos los archivos y subVIs necesarios para hacerlos funcionar, documentación, etc, se agruparán y organizarán automáticamente.

A su vez, los proyectos del módulo Robotics contienen todos los archivos necesarios para configurar la comunicación entre el hardware, crear builds del proyecto e implementarlas sobre la plataforma (en nuestro caso, el robot).

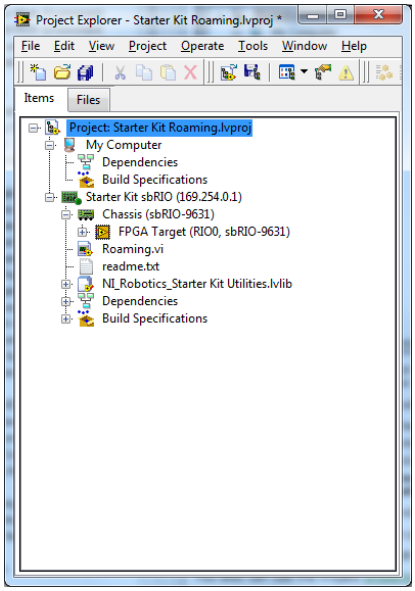

Figura 2.14. Ventana de un proyecto Robotics

En la figura 2.14 se presenta la estructura de un proyecto. En este caso, se ha definido una plataforma (Starter Kit) sobre la cual se encuentra montada una placa sbRIO-9631. LabVIEW se encarga de generar automáticamente los archivos necesarios para configurar el hardware que se vaya a emplear.

De la misma manera, los Vis se guardarán dentro del directorio de su plataforma correspondiente, junto a todos los archivos secundarios necesarios para su ejecución, los cuales se almacenarán en el directorio Dependencies. Pueden incluirse librerías de funciones específicas para la plataforma, como es el caso.

Build Specifications se emplea para configurar aplicaciones, compilando una versión autoejecutable del programa para ser almacenada en la plataforma y que ésta pueda iniciarla de forma autónoma sin dependencia del ordenador host.

2.4.1.- Robotics Project Wizard

LabVIEW 2010 incluye la aplicación Robotics Project Wizard, la cual permite crear de forma guiada un proyecto de tipo Robotics.

Para abrir el Robotics Wizard hacemos click en File→New y seleccionamos Project→Robotics Project.

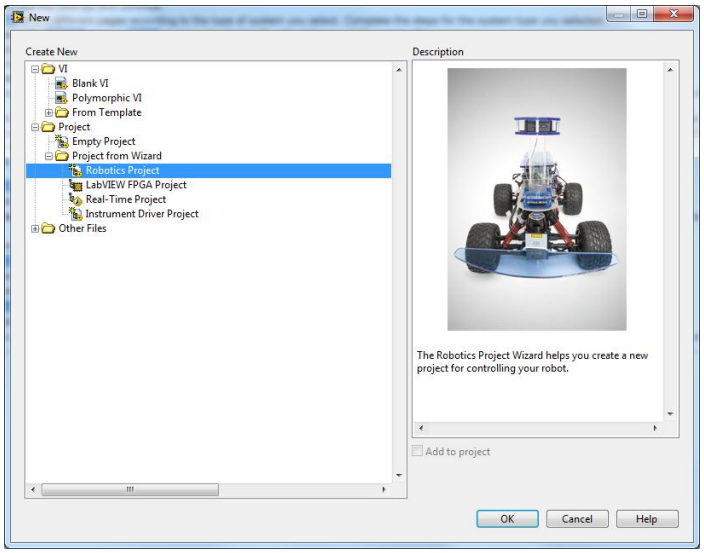

Figura 2.15. Creación de un proyecto Robotics

El gestor se encargará de administrar todos los archivos necesarios para nuestro proyecto, así como de adjuntar las librerías necesarias conforme añadamos funciones de módulos LabVIEW. Sin embargo, es necesario advertir que no hará lo mismo con los subIVs creados específicamente por el usuario, para los cuales se aconseja crear una subcarpeta dentro del proyecto con los archivos que planeemos añadir.

De esta forma, tendremos en un directorio todos los ficheros necesarios para administrar el robot, favoreciendo la portabilidad del proyecto si fuese necesaria.

Tras abrir el Robotics Project, se nos ofrecen distintos tipos de proyecto. El módulo de Robotics solo incluye por defecto el tipo Windows Platform, para habilitar el resto (Robotics Starter Kit, CompactRIO y Single-Board RIO) es necesario instalar los drivers NI-VISA Y NI-RIO.

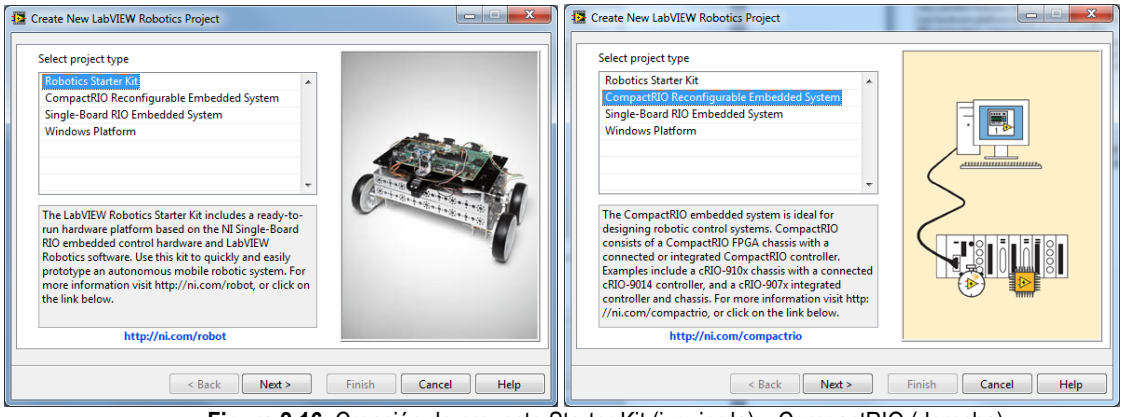

Figura 2.16. Creación de proyecto Starter Kit (izquierda) y CompactRIO (derecha)

En el caso del tipo CompactRIO Reconfigurable Embedded System, así como el de Single-Board RIO, se empieza por definir el equipo sobre el que vamos a trabajar. Si disponemos del mismo, basta con conectarlo a la red y utilizar la detección automática para recopilar su configuración.

Si no se dispone todavía del hardware, es posible definir la configuración del mismo de forma manual seleccionando la opción Create New System. Esto nos permitirá seleccionar el tipo de controlador Real-Time CompactRIO, el modelo de FPGA, y los módulos de hardware adicionales.

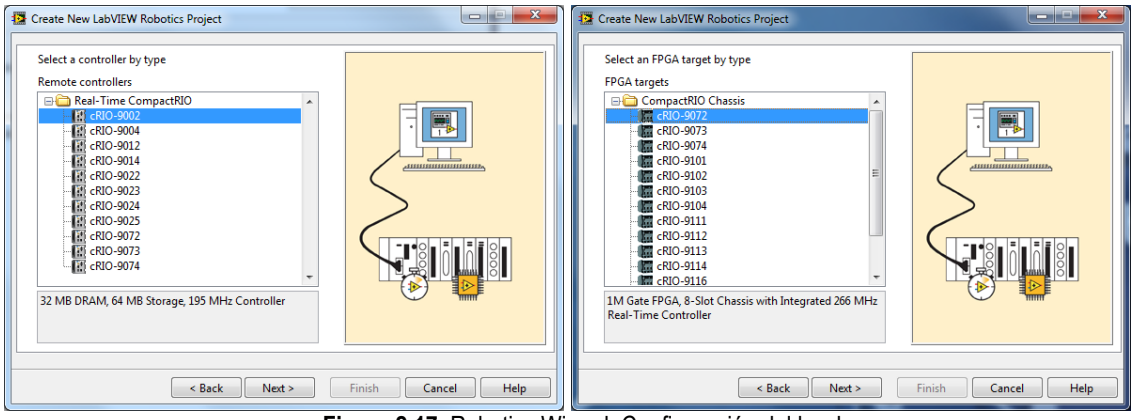

Figura 2.17. Robotics Wizard: Configuración del hardware

A continuación, seleccionamos de la lista los componentes del tipo sensores y actuadores que incorpora el hardware, para que el Wizard incluya en nuestro proyecto los drivers necesarios:

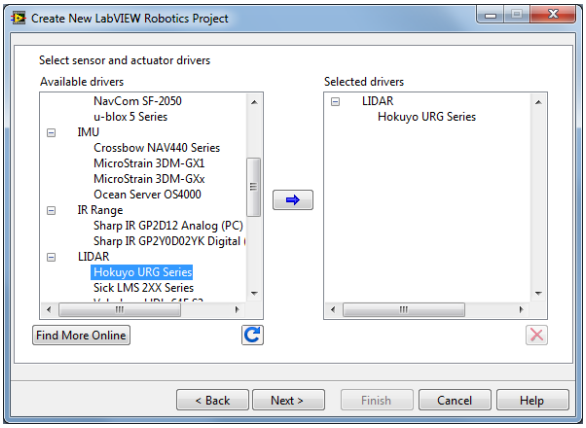

Figura 2.18. Robotics Wizard: Sensores

Finalmente, deberemos seleccionar el modelo de arquitectura que deseamos para nuestro robot. Estos no son más que simples plantillas sobre las que empezar a trabajar, pudiendo modificarlas totalmente dependiendo del objetivo y necesidades del proyecto. La plantilla constituirá el VI principal del proyecto, pero puede modificarse o cambiarse por otro si fuese necesario.

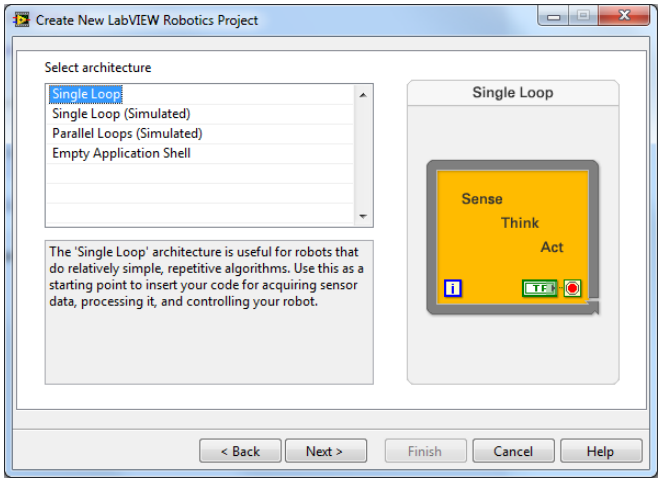

Figura 2.19. Robotics Wizard: Plantilla inicial

Cabe destacar que los Single y Parallel Loop (Simulated) presentan ejemplos en los que se simula por software el comportamiento de un sistema real simple.

Solo resta definir el nombre de nuestro proyecto y el directorio donde queremos guardarlo:

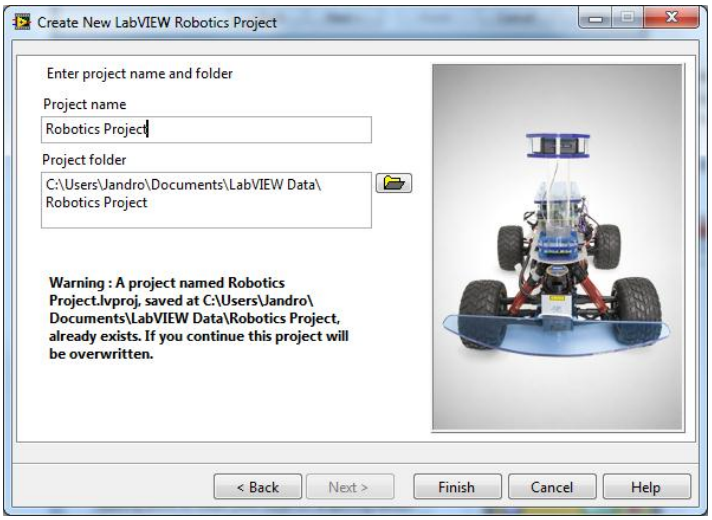

Figura 2.20. Robotics Wizard: Plantilla inicial

2.5.- Nuestro primer algoritmo

Para comenzar, se ha implementado un programa de navegación simple para un robot móvil (bajo Windows Platform → Single Control Loop), que emplea un dispositivo LIDAR como sensor para evitar obstáculos en su camino.

Nótese que para este primer ejemplo se ha optado por no definir el hardware específico de la plataforma, puesto que solo lo emplearemos para exponer la creación, estructura interna y programación de un robot en el entorno LabVIEW.

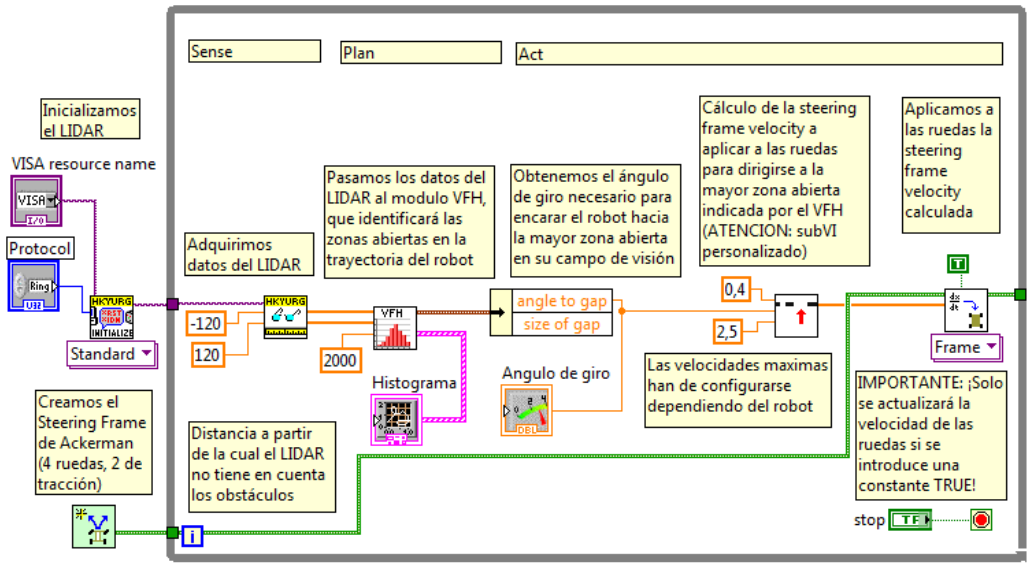

Figura 2.21. Diagrama de bloques

No existe ningún tipo de planificación de ruta. Está programado para errar sin objetivo fijo por el entorno. El LIDAR se ocupa de detectar los posibles obstáculos, pasándole la información al Vector Field Histogram, que indica en todo momento la dirección en la que se encuentra la zona libre de obstáculos más amplia en su camino. La planificación se reduce, pues, a elegir dicha dirección como nueva ruta.

Una vez que el robot dispone del ángulo necesario al que debe girar para establecer la nueva ruta, debe transmitir dicha información a los motores, para que lleven a cabo las acciones necesarias para corregir la trayectoria.

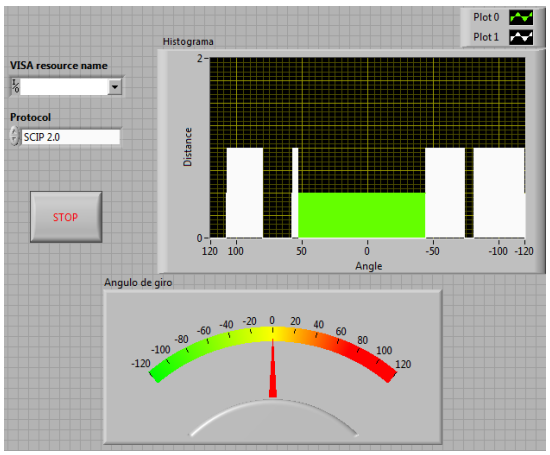

Figura 2.22. Panel frontal

Para hacer el ejemplo más gráfico, se han dispuesto indicadores en el panel frontal (Fig. 2.22), tanto para el histograma construido en tiempo real, como del ángulo de giro decidido.

A continuación se realiza un análisis del código paso a paso, describiendo las funciones empleadas y el uso que se le ha dado a sus entradas y salidas si fuese necesario.

2.5.1.- Lidar

El Lidar es un dispositivo sensor que permite determinar la distancia desde un emisor láser a un objeto o superficie, empleando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada.

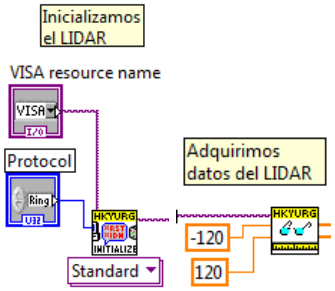

Figura 2.23. Diagrama de bloques: LIDAR

En este caso, se ha insertado un Lidar modelo Hokuyo URG (para el cual podemos encontrar funciones de acceso y lectura en Functions→Robotics→Sensing→LIDAR→Hokuyo URG Series). Para emplearlo es necesario invocar a la función Initialize, la cual establece la comunicación con el hardware. Es imprescindible, pues, indicar al menos el *VISA Resource Name* como entrada.

Nótese que Virtual Instrument Software Architecture (VISA) hace referencia a la librería de interfaz designada para controlar todo tipo de instrumentos, como GPIB, VXI, RS232, etc. Es un estándar diseñado para administrar los drivers para comunicarse con los instrumentos.

Al invocar Initialize podemos escoger, entre otros, el protocolo a emplear (SCIP 2.0 por defecto) y debido a su naturaleza polimórfica, si queremos que funcione como un driver Standard o en modo cRIO-USB.

Para adquirir datos, recurriremos a la función Acquire Data enlazando el VISA Resource Name con el Initialize anterior. Esta función debe encontrarse dentro del bucle, puesto que queremos que se encuentre funcionando de forma continua.

Acquire Data toma como inputs los ángulos entre los cuales va a efectuarse el barrido, tomando como referencia de 0º el centro del dispositivo. Desde dicho punto, Starter Point hace referencia a los puntos a la derecha del mismo, y End Point a la izquierda. En este caso hemos tomado el máximo de los valores posibles, rango (-120º, +120º) puesto que nos interesa hallar la ruta más apta.

El output de Acquire Data devuelve dos arrays, que contienen la magnitud y dirección de las mediciones tomadas. Esta información la pasaremos al Simple Vector Field Histogram.

2.5.2.- Simple Vector Field Histogram

El SVFH (accesible desde la paleta Functions→Robotics→Obstacle Avoidance→Simple VFH) es una función proporcionada por Robotics que identifica los obstáculos en la trayectoria, así como las zonas abiertas, convirtiéndose en una herramienta muy potente para la navegación.

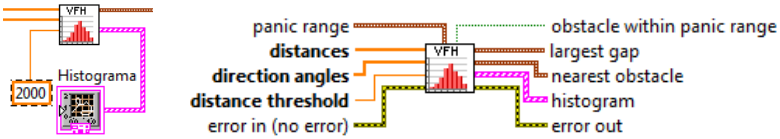

Figura 2.24. Diagrama de bloques: SVFH (izquierda) y su panel de conexiones (derecha)

En nuestro caso se emplea de la forma más simple, introduciendo los datos del entorno recogidos anteriormente por el LIDAR (magnitude→distances; direction→direction angles) y recogiendo el ángulo al que se encuentra la posible trayectoria que posee la mayor zona abierta (y su tamaño) respecto a la trayectoria actual. Estos datos se encuentran en el cluster largest gap.

Es necesario recalcar la importancia del input distance threshold, que fija la distancia a partir de la cual el SVFH no considera a los objetos en dicha dirección como obstáculos. Debe buscarse un equilibrio: si fuese demasiado pequeña, el robot no tendría tiempo de reaccionar. En cambio, una distancia demasiado grande implicaría que el robot pudiese considerar que no hay ninguna trayectoria posible. Esto último puede ejemplificarse en el caso de que el robot navegase por un pasillo, antes de llegar a una esquina con un giro a la derecha el robot la interpretaría como un camino cerrado, mientras que si distance threshold fuese lo suficientemente pequeña, para cuando interpretase la trayectoria recta como “final del pasillo” ya habría llegado a la esquina y el combo Lidar+SVFH registraría a su derecha la “zona abierta”.

Otras entradas y salidas de la función que no hemos empleado son:

- Panic range: define el rango al que el VFH identifica un obstáculo como un área del entorno a evitar. Es un cluster que incluye dos variables: panic threshold distance (distancia maxima a la que la salida obstacle within panic range se fija en TRUE) y panic threshold angle (especifica el ángulo máximo al que se fija obstacle within panic range como TRUE).

- Nearest obstacle: Cluster que contiene el ángulo y la distancia a la que se encuentra el obstáculo más cercano

Cabe destacar que histogram retorna el histograma elaborado (angulo x distancias), el cual podemos representar simplemente añadiendo su indicador correspondiente, pero ahí no acaba su potencial. Esta información podría ser empleada para interpretarse a más bajo nivel (aunque la función ya ofrece las salidas necesarias para las utilidades más inmediatas).

2.5.3.- SubVI: GoToLargestGap

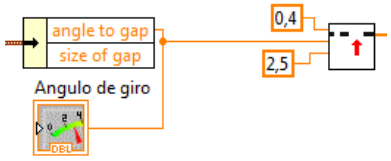

Figura 2.25. Diagrama de bloques: GoToLargestGap

Aplicando la función unbundle by name podemos acceder a los elementos del cluster largest gap que nos interesen. Nuestro algoritmo es simple: queremos girar hacia la zona abierta más grande, así que no nos importa tanto su tamaño (puesto que ya sabemos que es la zona más ancha) como el ángulo al que se encuentra respecto a la trayectoria actual (angle to gap).

Para traducir el ángulo de giro a la velocidad de las ruedas, se ha añadido el subVI denominado GoToLargestGap. Éste genera el array de una dimensión Steering Frame Velocity (x, y, theta) a partir de la velocidades máximas angular y recta.

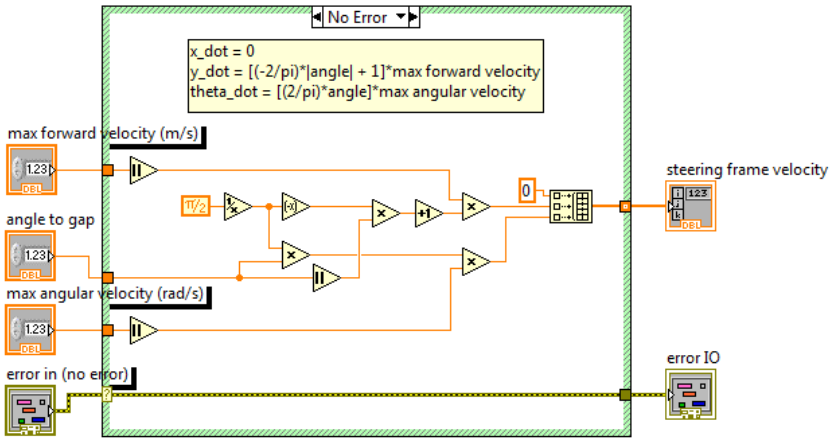

Figura 2.26. Diagrama de bloques del subVI GoToLargestGap

El array Steering Frame Velocity constituye la entrada necesaria para la función Apply Steering Frame Velocity To Wheels.

2.5.4.- SubVI: CreateAckermannSteeringFrame

Este subVI es un ejemplo de creación de un Steering Frame. Éste se lleva a cabo definiendo el conjunto de ruedas de las que dispone el robot detallando sus características mediante la función Create Wheel.vi (Robotics→Steering), reuniendo todas en un cluster, y pasándole la información a la función Creating Steering Frame.vi (la cual es polimórfica, permitiendo diferentes configuraciones).

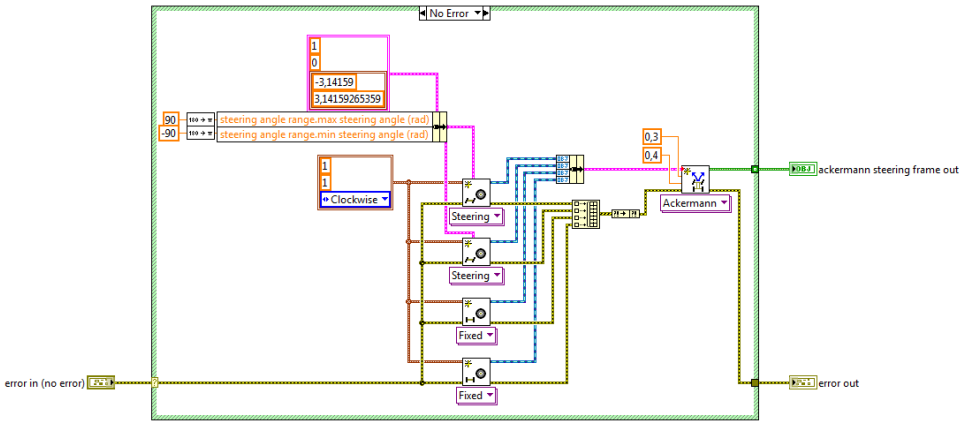

Figura 2.27. Diagrama de bloques del subVI CreateAckermannSteeringFrame

2.5.5.- Apply Steering Frame Velocity To Wheels

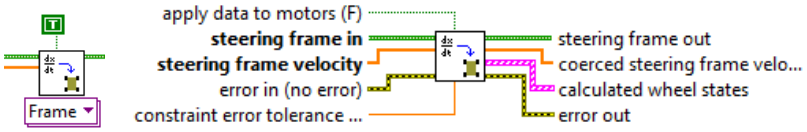

Figura 2.28. Diagrama de bloques: Apply Steering Frame Velocity (izquierda) y su panel de conexiones (derecha)

Función polimórfica que calcula el estado necesario de cada rueda para satisfacer el **Steering Frame Velocity**. (su otra instancia lo calcula a partir de la Arc Velocity).

- **Apply data to motors** permite actualizar automáticamente el estado de las ruedas si se fija en *True*.

- **Steering frame in** es la referencia al Steering Frame que se desea modificar.

- **Steering frame velocity** debe especificarse en la forma de un array que sigue la estructura [x_dot, y_dot, theta_dot], donde x_dot es la velocidad lateral (si la hubiese), y_dot es la velocidad hacia delante, y theta_dot es la velocidad angular en dirección contraria a las agujas del reloj. El subVI GoToLargestGap se encarga de calcularlo previamente.

2.6.- Simulación de un proyecto Robotics: RobotSim

RobotSim es un programa de la casa Cogmation que permite simular en un entorno 3D el comportamiento de un robot. A partir de la versión 2010 LabVIEW ofrece librerías de soporte para facilitar la comunicación entre el módulo Robotics y RobotSim, siendo una de las mejores opciones disponibles para tal efecto.

Se debe hacer hincapié en que RobotSim requiere la generación de un nuevo VI que contenga el código original a simular, sobre el cual se deben realizar modificaciones sustituyendo todas las funciones de comunicación directa con el hardware del robot por las específicas de comunicación con RobotSim. A efectos prácticos, RobotSim funciona como una nueva instancia del hardware.

Es decir, RobotSim precisa de su propia librería de funciones en LabView para interactuar con la simulación. De esta forma, deberemos modificar en consecuencia toda la programación de entradas y salidas para comunicarnos con el programa que controla al robot virtual.

Por lo tanto, la utilidad final de RobotSim pasa por testear los algoritmos que definen el comportamiento e "inteligencia" del robot. Todo lo referido al testeo de la comunicación y control del mismo a bajo nivel (Ej. programación de la FPGA) no es objetivo del programa.

RobotSim incluye varios modelos que emulan robots reales, aunque es posible crear otros empleando la herramienta de diseño y modelado Cogmation RobotBuilder. En nuestro caso, optaremos por el modelo incluido del Robotics Starter Kit.

A lo largo de esta sección introduciremos la simulación de proyectos Robotics mediante RobotSim, describiendo el entorno sobre el que se va a trabajar, y desarrollando un programa que sirva para ejemplificar la lógica a seguir a la hora de realizar las modificaciones necesarias sobre nuestros programas para compatibilizarlos con el simulador.

2.6.1.- El entorno RobotSim

Al abrir el programa (Fig. 2.27) se ofrece como plantilla un entorno 3D vacío con una malla que referencia al suelo. Aunque es posible crear un entorno de simulación desde cero, el programa incluye varias plantillas con entornos completos, listos para utilizarse, por lo que optaremos por abrir y editar uno de ellos.

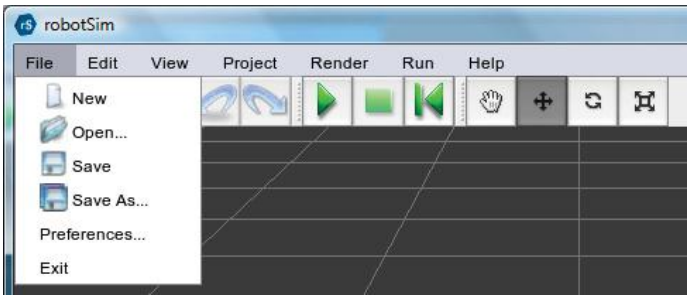

Figura 2.29. RobotSim: Ventana principal

Comencemos haciendo un repaso a las opciones básicas del programa que nos interesan, puesto que las relativas al modelado 3D no son el objetivo del presente proyecto. En concreto, veamos las posibilidades de configuración del entorno accesibles desde la pestaña View:

- La ventana de Objetos permite añadir elementos físicos al entorno, ya sean provenientes de una librería (muebles, columnas…) como creados desde cero a partir de formas geométricas básicas.

- La ventana de Robots, a su vez, permite añadir al entorno las instancias del robot virtual que deseemos. Nótese que deben haber sido previamente creados con la herramienta RobotBuilder para poder ser añadidos a un proyecto. Cada robot virtual lleva asociada una IP, que sirve como referencia para la comunicación con LabVIEW.

- Finalmente, la ventana de Propiedades (referenciada al objeto seleccionado previamente a su llamada) permite modificar parámetros básicos de cada elemento, desde tamaño a propiedades físicas.

Para la simulación que nos ocupa, se ha trabajado sobre la plantilla de un apartamento, añadiendo una columna en el centro de la habitación mediante la ventana de Objetos. En dicho entorno se ha situado una instancia del modelo 3D correspondiente al Starter Kit.

2.6.1.1.- Listado de Actuadores/Sensores en RobotSim

Una vez establecida la comunicación entre LabView y el robot designado en RobotSim, interactuaremos con él empleando una de las funciones de lectura/escritura, que precisa de tres datos de entrada: Robot Pointer. Object Name y Property Name.

- Robot Pointer indica el robot con el que nos estamos comunicando (ya que es posible simular mas de uno a la vez)

- Object Name hace referencia al nombre del sensor o actuador que tiene incorporado el robot con el que queremos comunicarnos.

- Property Name, a su vez, indica que propiedad de dicho sensor o actuador queremos leer o modificar.

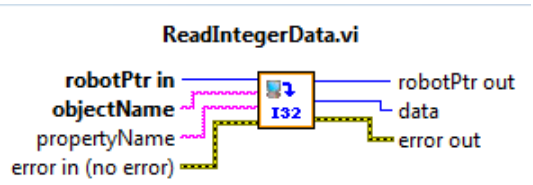

Figura 2.31. Función de lectura para RobotSim

Debido a las limitaciones actuales del software, RobotBuilder no permite listar directamente las referencias exactas (Object Name) de los actuadores y sensores de un robot. Los nombres empleados en el editor RobotBuilder para designar a los componentes no corresponden exactamente con su referencia de Objeto, de ahí la necesidad de conocer las referencias.

Como acabamos de ver, dichas referencias son las necesarias para interactuar con el robot desde LabView, por lo que esto representa un problema. Ocurre lo mismo con la lista de propiedades de cada objeto.

En futuras versiones de RobotSim/Builder está previsto subsanarlo, pero en la actualidad la única forma de obtener dichas listas es solicitarlas desde un VI.

Como necesitamos esta información previamente a la elaboración de la simulación, se revela como algo necesario disponer de un VI estándar que nos indique las referencias de actuadores y sensores del robot que vayamos a invocar. Se ha implementado una solución sencilla a dicho problema en el VI ObtenerActSensProp.

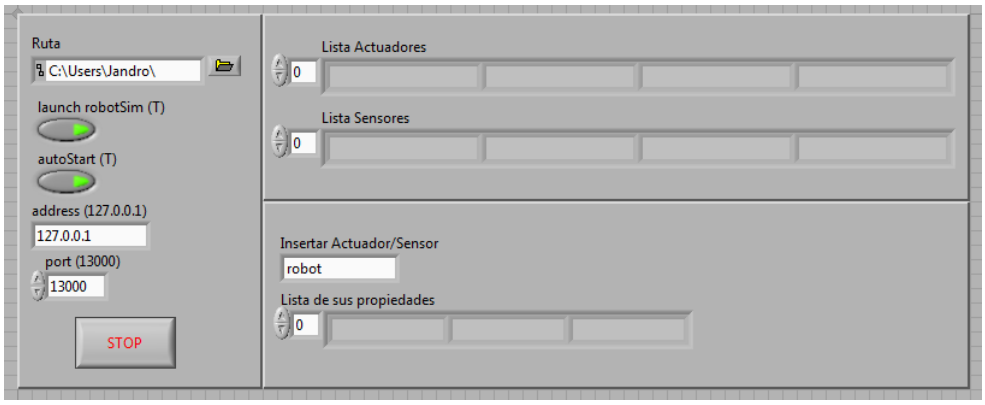

Figura 2.32. ObtenerActSensProp.VI: Panel frontal

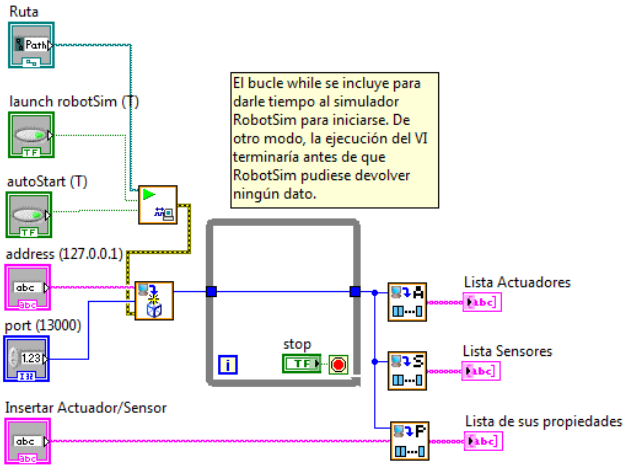

Figura 2.33. ObtenerActSensProp.VI: Diagrama de bloques

Para utilizar dicho VI, previamente debe crearse un entorno de simulación nuevo e importar dentro el robot del que se desea obtener el listado de referencias. La ruta a dicho entorno, en formato .rsim, se indicará en el cuadro de texto correspondiente del panel frontal.

A continuación se ejecuta el VI con el resto de valores por defecto, esperando a que se establezca la conexión con RobotSim. Los datos se recogen en los array Lista Actuadores y Lista Sensores. Si deseamos conocer las propiedades de alguno de ellos, bastará con introducirlo en el cuadro de texto "Insertar Actuador/Sensor" y reiniciar el VI

2.6.2.- Simulación del Starter Kit

A continuación se presenta un ejemplo en el cual se lleva a cabo la simulación de un robot programado en LabView a través de las funciones de conectividad con RobotSim que ofrece el paquete LabView Robotics.

Como ya se ha remarcado anteriormente, el código necesario para realizar simulaciones conlleva la sustitución de funciones específicas de entrada/salida al robot por otras orientadas a comunicarse con el simulador. En este caso, se ha programado empleando las funciones de simulación del NI Starter Kit, existiendo funciones equivalentes generalizadas a cualquier otro tipo de robot.

El simulador incluye un robot virtual creado en RobotBuilder que emula el Starter Kit real, tanto en aspecto como sensorización básica, aunque con el inconveniente de no existir módulos que permitan la lectura de los encoders.

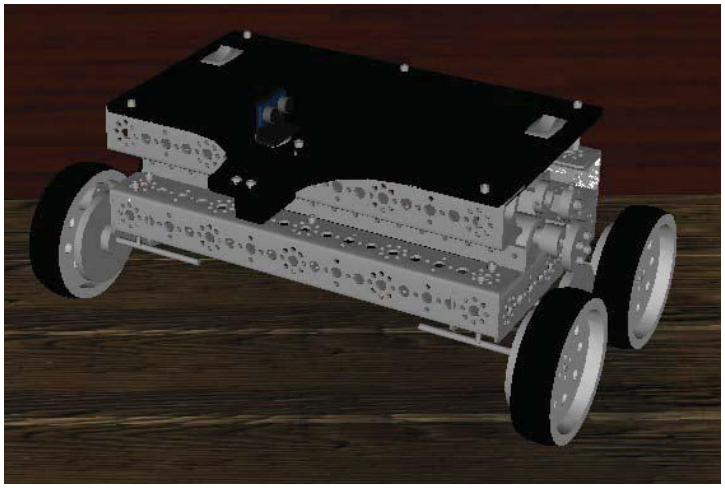

Figura 2.34. RobotSim: Starter Kit

En el capítulo 3 se profundizará en el Starter Kit. Por ahora, es suficiente con saber que se trata de un robot diferencial de cuatro ruedas que dispone de dos motores (izquierdo y derecho), y un sensor de ultrasonidos montado sobre un servo que le permite rotar su orientación.

Los paneles del VI principal presentan el siguiente aspecto:

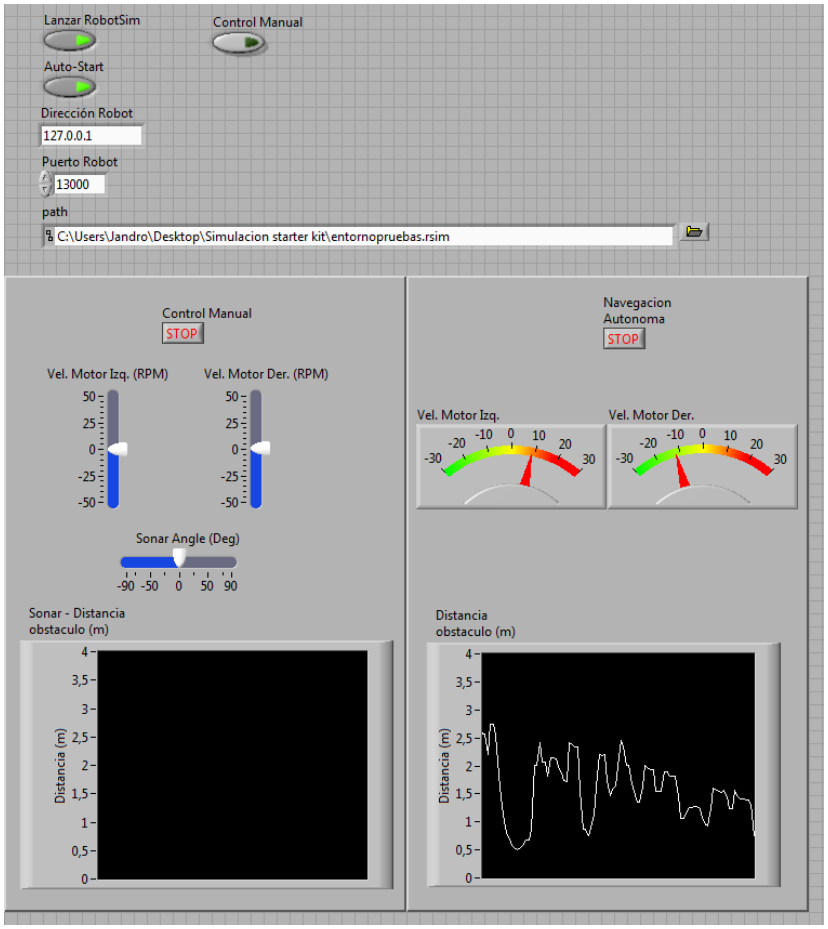

Figura 2.35. Panel frontal

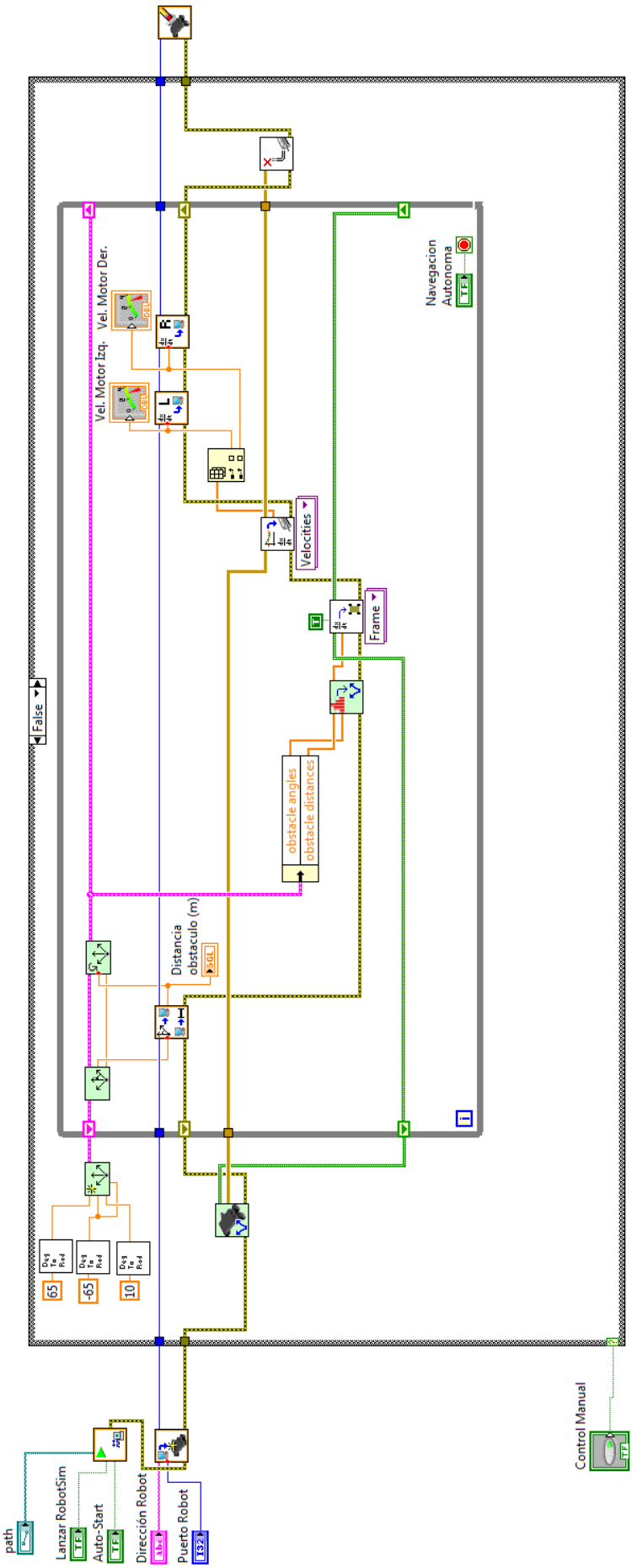

Figura 2.36. Diagrama de bloques: Navegación autónoma

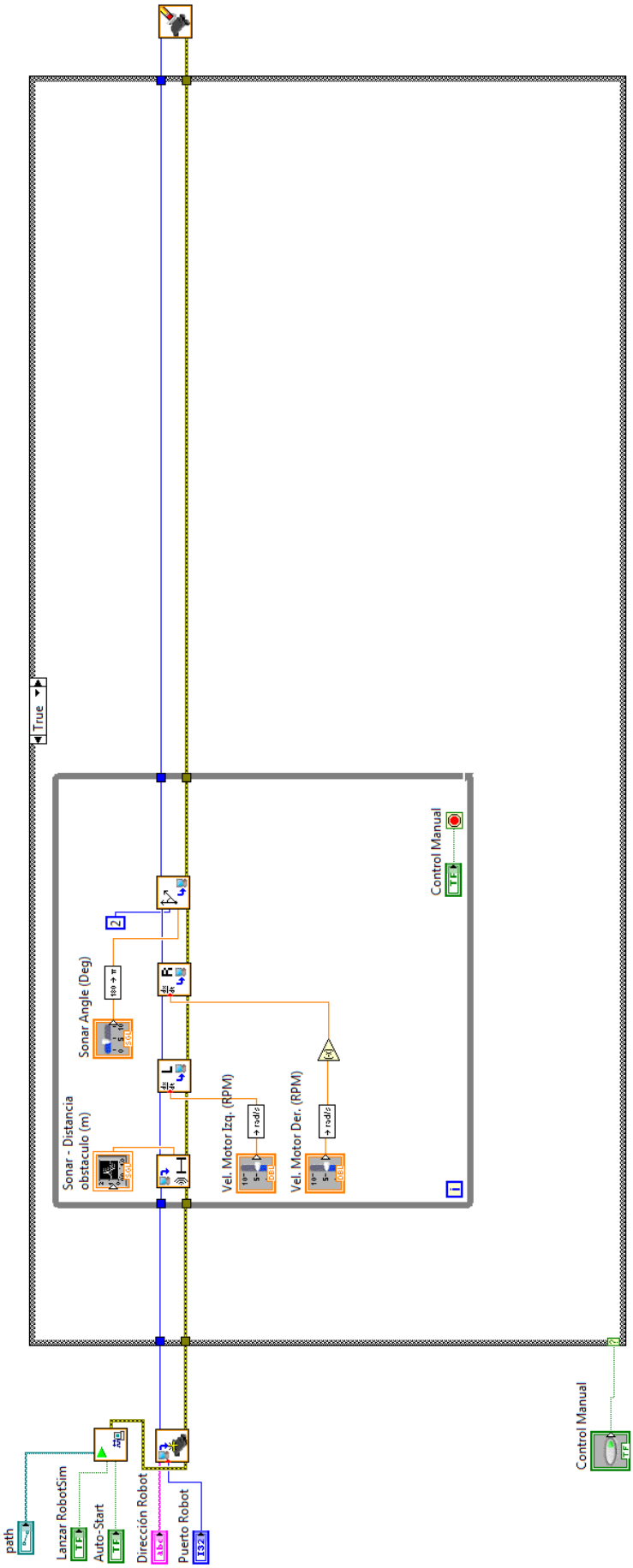

Figura 2.37. Diagrama de bloques: Control manual

2.6.2.1.- Descripción general

El código implementa dos comportamientos distintos del robot, seleccionables de forma previa a la ejecución del código a través del panel frontal:

- Control Manual (Botón Control Manual activado): Correspondiente al panel izquierdo. El usuario controla el movimiento del robot, empleando las barras que establecen las velocidades de los dos motores, y el ángulo en el que se orienta el sónar.
- Navegación autónoma (Botón Control Manual desactivado): correspondiente al panel derecho. El robot sigue un algoritmo de desplazamiento simple de forma autónoma.

Dicho algoritmo puede resumirse en que el robot se desplaza mientras emplea el sónar (montado sobre un servo que lo hace rotar cíclicamente) para encontrar los "huecos" en su trayectoria, desplazándose siempre hacia el mayor de ellos.

Esto se completa con la evaluación de una distancia umbral al obstáculo más cercano, a partir de la cual el robot retrocede. De esta forma se evita que choque al tratar de pasar por un hueco que no sea lo suficientemente ancho para él.

Con esta configuración el robot se limita a deambular sin rumbo fijo esquivando los obstáculos en su camino de forma bastante satisfactoria.

2.6.2.2- Inicialización y finalización de la simulación

Para inicializar la simulación, emplearemos la función robotSimLauncher.vi, localizada en la ruta Robotics→Connectivity→Cogmation Robotics→robotSim→StarterKit, desde donde podemos acceder a todas las funciones específicas para la simulación con RobotSim del NI Starter Kit.

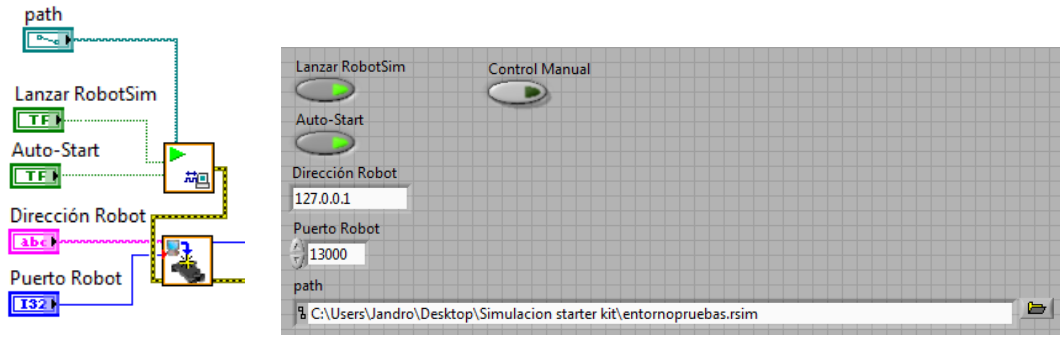

Figura 2.38. Código de inicialización (izquierda) y sus variables de entrada del panel frontal (derecha)

La simulación precisa de las siguientes entradas:

- Launch robotSim (Boolean): indica si se lanza la aplicación RobotSim.
- Auto-Start (Boolean): indica si se inicia la ejecución de la simulación automáticamente tras cargarse la aplicación.
- Path: dirección del archivo del proyecto de RobotSim que va a cargarse. Previamente ha de crearse un proyecto que contenga el robot a utilizar dentro del entorno de simulación deseado. En este caso tenemos un Starter Kit en un pequeño apartamento modificado a partir del proyecto por defecto.

El siguiente paso consiste en inicializar la comunicación con los robots contenidos en el proyecto cargado. Para cada robot que deseemos que intervenga en la simulación emplearemos la función CreateStarterKitRobot.vi, indicándole:

- Address: Dirección IP del robot (debe coincidir con la asignada al modelo en RobotSim)
- Port: Puerto de comunicación que vamos a emplear para conectarnos a la simulación.

Como salida, la función devolverá un puntero al robot con el que se ha realizado la conexión. Dicho puntero deberá ser empleado en las funciones destinadas a realizar algún tipo de comunicación con el robot (ya sea de input u output).

Para finalizar la comunicación con el simulador, deberemos llamar a la función DeleteStarterKitRobot, indicándole el robot con el que se finaliza la comunicación, mediante su puntero.

Figura 2.39. Función DeleteStarterKitRobot

Nótese que el botón Control Manual debe situarse en la posición deseada (si está desactivado el robot tiene comportamiento autónomo) antes de ejecutar el código. Dicho botón booleano controla los dos estados posibles de una estructura *case* que contiene las dos posibilidades de comportamiento del robot.

2.6.2.3.- Control Manual

El control manual se compone de un diagrama simple, encerrado en un bucle while. Nótese que todas las funciones de comunicación con el robot de este código son exclusivas de la simulación mediante Cogmation Robotsim, y a su vez son específicas del NI Starter Kit.

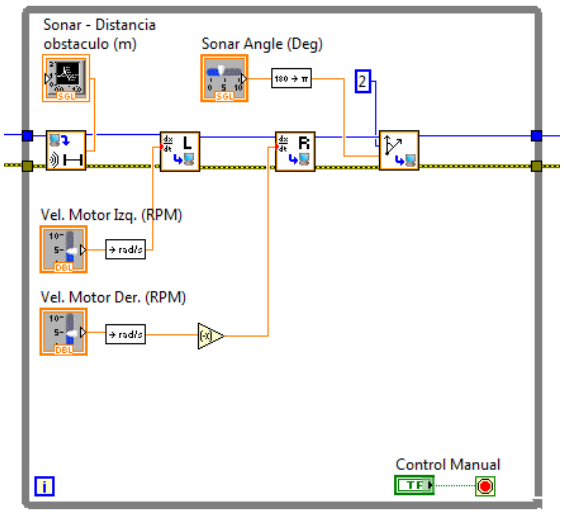

Figura 2.40. Control manual: Diagrama de bloques

Se inicia comprobando la lectura del sónar mediante la función GetSonarDistance. Proporcionándole el puntero del robot correspondiente, devuelve la distancia hacia el obstáculo que se encuentre en su trayectoria.

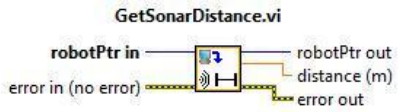

Figura 2.41. Función GetSonarDistance: Conexiones

La comunicación con los motores del robot simulado se lleva a cabo a través de las funciones SetLeftMotorVelocity y SetRightMotorVelocity. Basta con proporcionarle la velocidad deseada en rad/s y el puntero que hace referencia al robot al que se desea aplicar.

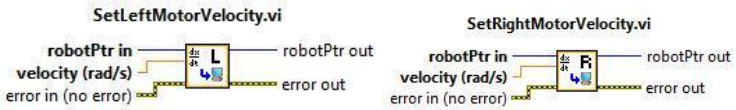

Figura 2.42. Funciones SetLeft/RightMotorVelocity: Conexiones

Para introducir las velocidades de los motores, se emplean botones del tipo Pointer Sliders, tomando como referencia RPM, velocidad que traduciremos a rad/s empleando un sencillo sub-vi:

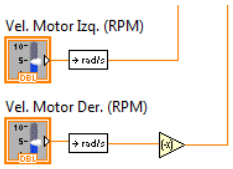

Figura 2.43. Conversión RPM a rad/s

Finalmente, se fija el ángulo al que el servo (sobre el que está montado el sensor de ultrasonidos) debe girar, mediante la función SetMotorAngle.vi. Dicho ángulo también se introduce de forma manual mediante otro Pointer Slider. La particularidad de esta función es que es genérica para todos los servos que incluye el robot, por lo que debe introducirse también un entero en el parámetro index que indique a cual de los servos debe aplicarse el giro:

- 0 → Motor izquierdo
- 1 → Motor derecho
- 2 →Servo motor (que controla el giro del sensor)

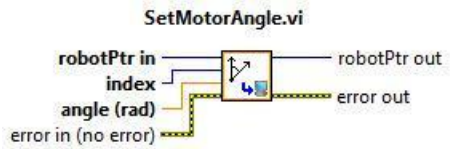

Figura 2.44. Función SetMotorAngle: Conexiones

En cada iteración del bucle se realizan las comprobaciones para actualizar la situación del robot. Se finaliza la ejecución mediante el botón STOP asociado a la condición de cierre del bucle.

2.6.2.4.- Navegación Autónoma

Para la simulación de la navegación autónoma vamos a emplear una librería de funciones (denominada Cogmation_Robot_Sim_Utilities.lvlib) proporcionada por NI, que contiene utilidades varias para agilizar la programación del Starter Kit.

En el diagrama de bloques expuesto anteriormente, pueden identificarse las funciones de dicha librería por el fondo verde de los iconos.

Esta librería resulta muy limitada en cuanto a contenido, pero su importancia radica en que dichas funciones pueden generalizarse a otros robots con ligeras modificaciones, e incluyen funciones orientadas a ser empleadas con una algoritmia básica, que puede servir de introducción a la programación basada en Labview Robotics.

- **Inicialización del vector de obstáculos**

La función Initialize ObstacleVectors crea el vector de obstáculos empleado para almacenar los datos recogidos por el sensor de ultrasonidos. Es necesario facilitarle el rango de ángulos sobre el que trabaja (mínimo y máximo) así como el inicial.

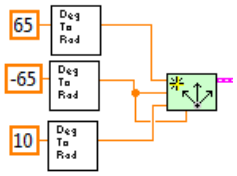

Figura 2.45. Llamamiento a la función InitializeObstacle

- **Actualización del sensor de ultrasonidos**

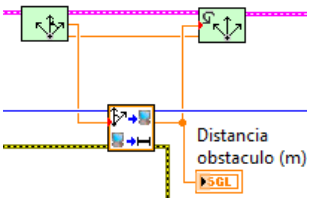

Figura 2.46. Empleo de la función Calculate Next Angle

Calculate Next Angle fija el siguiente ángulo al que el sensor debe desplazarse, lo cual se lleva a cabo mediante la función específica del Starter Kit UpdateSonar.vi, al cual le comunica dicho ángulo. Esta función precisa del RobotPtr (puntero que designa el robot con el cual se quiere establecer comunicación). Finalmente, arroja un valor double que indica la distancia al obstáculo en el ángulo requerido.

Dicho valor se almacena en el vector de obstáculos mediante la función Update Obstacle Vectors.

- **Inicialización del Steering Control**

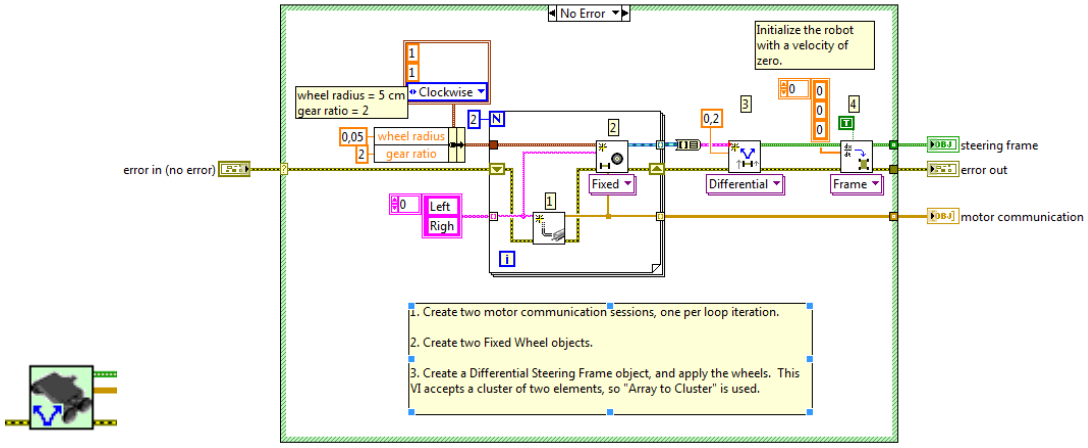

Figura 2.47. Icono de la función Steering Control (izquierda) y su diagrama de bloques (derecha)

Initialize Steering Control crea dos objetos rueda, estableciendo la comunicación motor con ambas. Aunque el Starter Kit tiene cuatro ruedas, presenta dos motores (izquierdo y derecho), cada uno de los cuales controla sus dos ruedas simultáneamente.

Posteriormente se inicializa el Steering Frame con estos datos y finalmente se aplica una velocidad inicial cero al vector que almacena el estado necesario de las ruedas.

- **Cálculo de la dirección**

Tomando el vector de obstáculos, le pasamos los datos de distancia a obstáculo respecto a su ángulo al VI Calculate Driving Direction.

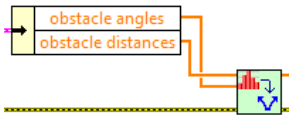

Figura 2.48. Llamamiento a Calculate Driving Direction

Dicha función contiene el siguiente algoritmo, que define el comportamiento del robot:

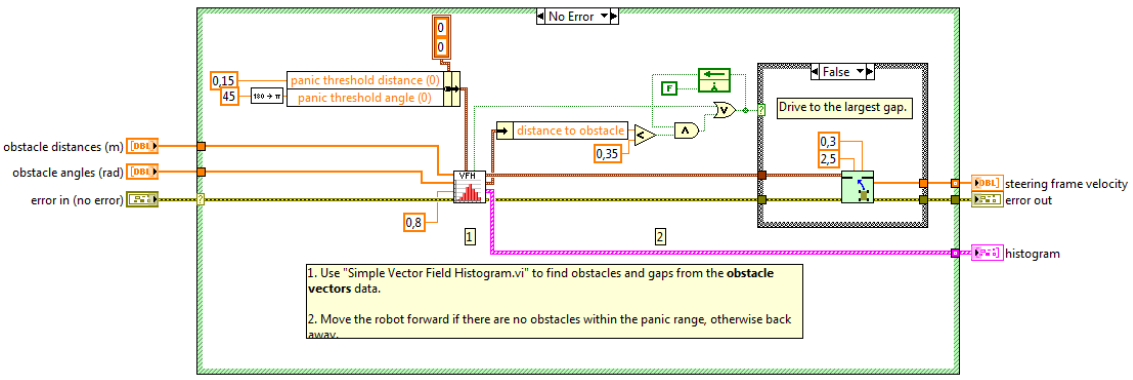

Figura 2.49. Calculate Driving Direction: Diagrama de bloques

Se comienza evaluando el hueco mayor en el campo de visión del robot mediante un Simple Vector Field Histogram, al cual se le define un panic range de 0.15m y un ángulo máximo de 45º (esto se refiere a la distancia a partir de la cual se evalúa en TRUE la salida que indica que uno de los obstáculos se encuentra en el rango de pánico, es decir, la distancia umbral a partir de la cual interpretamos que va a resultar un problema y debe obrarse para evitarlo).

Finalmente, se realiza una comprobación doble, tanto el estado de la salida obstacle within panic range del SVFH, como si el obstáculo más cercano se encuentra a una distancia menor de 0.35 metros. Si ninguna de ellas se cumple, se ejecuta el código correspondiente a la entrada FALSE de la estructura CASE (Drive To The Largest Gap). En el caso de que al menos una de ellas se cumpliese, se ejecutaría la parte del CASE correspondiente a la entrada TRUE (Drive Away From Nearest Obstacle).

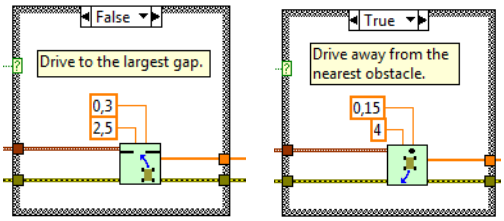

Figura 2.50. Posibilidades de la estructura CASE

- Drive to the largest gap: esta función realiza los cálculos necesarios para hallar las velocidades necesarias (lineal y angular) para dirigir el robot hacia el hueco más ancho existente en su campo de visión, las cuales se constituyen en la *steering frame velocity*. Nótese que se emplea una notación en la cual *x_dot* es la velocidad tangencial (cero en una configuración diferencial como la que nos ocupa) cuando lo habitual en otras fuentes es encontrar dicha velocidad designada como y_dot.

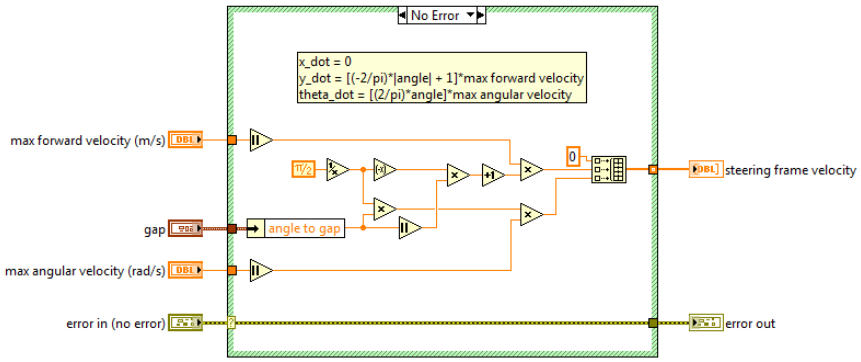

Figura 2.51. Drive to the largest gap: Diagrama de bloques

- Drive away from the nearest obstacle: En este caso, se toma como referencia el obstáculo más cercano y se dirige al robot en dirección contraria (esto acabará cuando volvamos a estar fuera de las distancias umbrales definidas anteriormente).
-

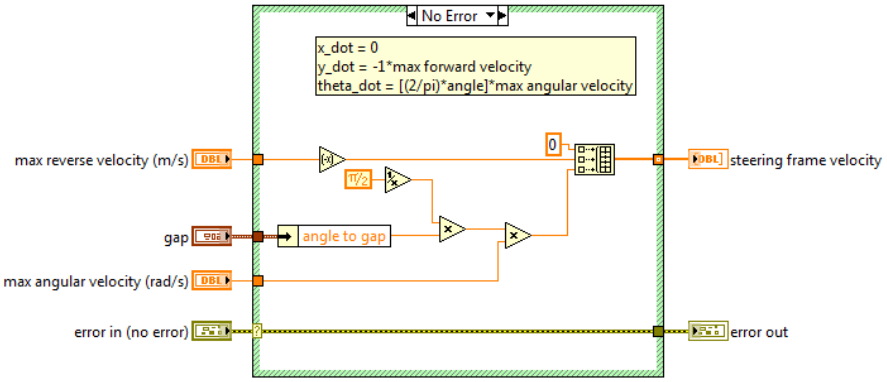

Figura 2.52. Drive away from the nearest obstacle: Diagrama de bloques

De esta forma se consigue una navegación autónoma simple pero efectiva. El robot deambulará sin rumbo fijo, evitando los obstáculos en su camino. Si llega al punto de no encontrar un camino viable, retrocederá para poder explorar otras alternativas.

A continuación, se aplicaría el cálculo del steering frame velocity, traduciendo dicha información al cálculo del estado necesario de las ruedas para cumplirla. Activando la variable booleana, esta información se actualiza en el motor communication.

- **Aplicación de la velocidad a los motores**

Mediante la función Get Motor Velocity Setpoints se obtiene, a partir del motor communication, las velocidades necesarias (en rad/s) para aplicar a los motores izquierdo y derecho, para cumplir las velocidades lineal y angular calculadas anteriormente.

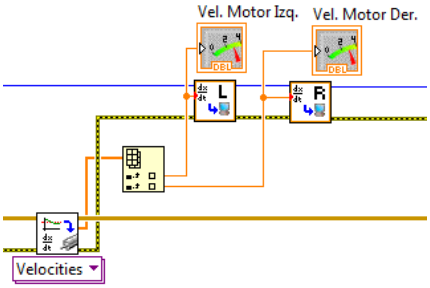

Figura 2.53. Aplicación de las velocidades

Finalmente, se cierra la comunicación motor.

2.6.2.5.- Ejecución de la simulación

La ejecución del programa se realiza de forma normal, haciendo click sobre el botón correspondiente, aunque es preciso que el entorno de simulación que se vaya a invocar esté abierto y activo en la ventana de RobotSim previamente a la ejecución del VI en LabVIEW para que la comunicación se establezca de forma satisfactoria. Así mismo, la IP introducida deberá coincidir con la de la instancia virtual del robot con el que se desee establecer comunicación, la cual puede consultarse desde RobotSim.

Como se ha detallado anteriormente, se ofrecen dos tipos de navegación: teledirigida y autónoma. En ambos casos la simulación responde como cabría esperar, permitiendo la primera manejar al robot virtual regulando de forma independiente la velocidad de sus motores desde el panel frontal del VI principal, mientras que la segunda nos permite observar el comportamiento del robot al operar sobre el algoritmo propuesto.

2.6.3.- Conclusiones

RobotSim ofrece una alternativa satisfactoria en cuanto a su funcionamiento, en tanto que cumple con su finalidad de ofrecer una simulación en un entorno virtual del comportamiento del robot físico.

Sin embargo, sufre de ciertas limitaciones debidas a su corto periodo de vida, representadas sobre todo en la falta de variedad de sensores simulados. También se echan en falta ciertas utilidades que se antojan necesarias, como alguna forma de comprobar la lista de actuadores y sensores de un robot directamente desde el programa (para lo cual se ha tenido que implementar un VI rudimentario en la sección 2.6.1.1).

El programa se encuentra constantemente en desarrollo, por lo que se espera que Cogmation sepa suplir las carencias iniciales. Dado que los problemas vienen dados sobre todo por la falta de contenido, y no por el funcionamiento en sí, es de esperar que nuevas versiones del programa lo completen y saquen a la luz el potencial de la herramienta.

CAPÍTULO 3

LABVIEW ROBOTICS STARTER KIT

3.1.- Introducción

La plataforma seleccionada para programar los algoritmos finales del presente proyecto es el modelo LabVIEW Robotics Starter Kit, de la casa National Instruments. Éste se promociona como un robot "listo para usar", enviándose completamente montado, lo cual ayuda enormemente a dar un punto de partida en el aprendizaje del mundo de la robótica. A su vez proporciona una plataforma abierta con capacidad para expandir sus funciones a través de nuevos módulos, sensores, y hardware en general.

Figura 3.1. LabVIEW Robotics Starter Kit

El Starter Kit está diseñado para funcionar bajo la ejecución de VIs, puesto que surgió con la intención de ofrecer un hardware a través del cual tanto principiantes como programadores más experimentados pudiesen sacar partido del nuevo módulo Robotics. Gracias a éste, pueden desarrollarse aplicaciones para el Starter Kit manteniéndose en la programación gráfica de alto nivel, sin necesidad de preocuparse por montar desde cero un robot y configurar su hardware.

De esta forma, cualquiera puede realizar un acercamiento al desarrollo centrándose en la programación, y a la vez también se permite un acercamiento a más bajo nivel accediendo directamente a la FPGA cuando sea necesario.

Por todo lo expuesto, el Starter Kit posee un notable potencial tanto como herramienta educativa en entornos académicos, como plataforma sobre la que poder testear nuevos algoritmos y aplicaciones. A lo largo del presente capítulo repasaremos su configuración, los principales componentes a tener en cuenta y su funcionamiento, además de detallar los añadidos y modificaciones realizadas.

3.2.- Visión general

El Starter Kit es un robot diferencial de cuatro ruedas que posee un controlador integrado NI Single-Board RIO, montado sobre una base Pitsco TETRIX. El controlador Single-Board RIO constituye una alternativa de bajo coste para la implementación de una plataforma NI CompactRIO. Este controlador, el cual funciona bajo tecnología LabVIEW Real-Time y LabVIEW FPGA, integra un procesador en tiempo real, una FPGA, y un sistema de entradas/salidas (analógicas y digitales), en una sola placa, además de permitir expansiones empleando módulos de la serie C de National Instruments.

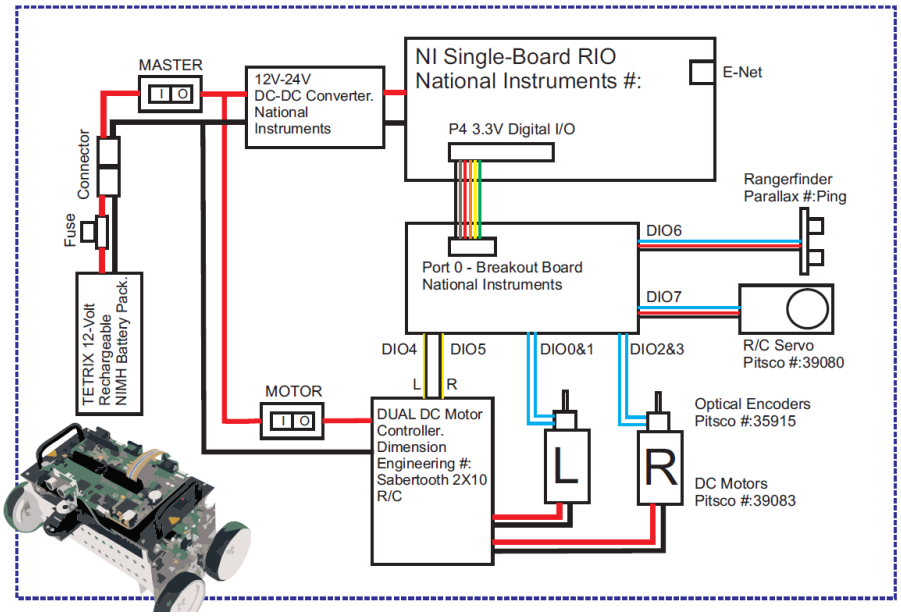

Figura 3.2. LabVIEW Robotics Starter Kit: Diagrama de bloques

Posee un par de motores DC (izquierdo y derecho) para controlar las ruedas. Éstos ofrecen una velocidad máxima de 152RPM, y vienen acompañados de sendos encoders para registrar el desplazamiento. Todo el conjunto se encuentra alimentado por una batería recargable de 12V, que le proporciona hasta 4 horas de autonomía.

Deben destacarse los interruptores MASTER y MOTOR situados en la parte superior. MASTER controla la alimentación de toda la plataforma, cumpliendo la labor de interruptor principal, mientras que MOTOR permite desactivar de forma independiente los motores. Esto resulta muy práctico a la hora de cargar, depurar y testear código para el que no necesitemos al robot en movimiento, asegurándonos que los motores están apagados, evitando algún posible percance si se activase por error un VI que comunicase con ellos.

3.3.- Sensores

En esta sección repasaremos los sensores preinstalados en el Starter Kit, así como los componentes extra que han sido incorporados para dotarle de nuevas capacidades necesarias para la elaboración de programas de mayor complejidad.

Cada componente vendrá precedido de una exposición sobre los principios de funcionamiento del mismo, a la cual le seguirá una descripción en detalle del hardware implementado. Finalmente se expondrá el código necesario para comunicarse con dicho hardware.

3.3.1.- Sensor de ultrasonido

3.3.1.1.- Principios teóricos

Un sensor de distancia por ultrasonido es un detector de proximidad que no precisa de contacto físico con el obstáculo para medir la distancia al mismo. Su funcionamiento se basa en generar una onda de sonido de alta frecuencia (muy por encima del rango de audición humano), y registrar la onda reflejada

que vuelve al sensor. Éste calcula el intervalo de tiempo entre que se emitió la señal y se recibió el eco para determinar la distancia a un objeto.

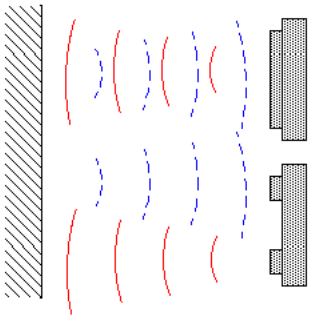

Figura 3.3. Sensor de ultrasonido de transductor único (arriba) y transmisión/recepción (abajo)

Para generar las señales se suele emplear un transductor piezoeléctrico que convierte energía eléctrica en sonido. De la misma forma, las señales recibidas se convierten en energía eléctrica para poder ser interpretadas. Algunos sensores, como el que nos ocupa, emplean dos sistemas independientes para la transmisión y recepción.

El poder detectar objetos a distancia y precisar su posición relativa resulta una cualidad muy provechosa en el campo de la robótica, por lo que estos sensores constituyen una herramienta utilizada con frecuencia para aplicaciones como la evasión de obstáculos, navegación y creación de mapas.

Sin embargo, a la hora de trabajar con un sensor de ultrasonido hay que tener en cuenta la existencia de una zona ciega entre la superficie del sensor y el alcance mínimo a partir del cual resulta fiable la detección de un objeto. Esta zona constituye siempre una de las características de diseño de un sensor, buscando su minimización en consonancia con la aplicación para la que esté destinado en un principio.

3.3.1.2.- Características del hardware

El Starter Kit incluye en la parte delantera un sensor de ultrasonido modelo Parallax PING))), centrado a una altura de 22cm (para evitar interferencias con ondas reflejadas en el suelo), y montado sobre un servo que le permite rotar en un rango de 180º en el plano horizontal. Según el fabricante, la medida de la distancia es fiable dentro del rango de 2 centímetros a 3 metros.

El sensor posee tres pines, los dos primeros correspondientes a la alimentación (la cual no debe exceder los +5V y 30mA), estando destinado el tercero al input/output.

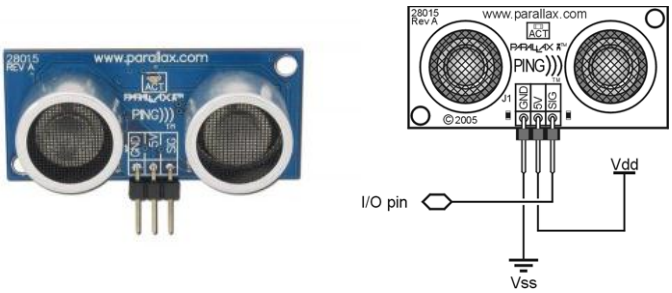

Figura 3.4. Parallax PING))) (izquierda) y su distribución de pines (derecha)

El funcionamiento se basa en lanzar una señal de activación a través del pin I/O del sensor (conectado a la placa sbRIO-9631 a través del puerto de entrada/salida digital referenciado como Port0/DIO6), el cual por defecto actúa como puerto de entrada. El sensor lanzará entonces una onda a 40kHZ y una velocidad de 344.4 metros por segundo, y esperará la respuesta del eco.

A partir de que se recibe la señal de activación, el pin I/O pasa a actuar como puerto de salida, comenzando a proveer un pulso que finalizará cuando se reciba el eco. Esto es, el ancho final del pulso resulta proporcional a la distancia al obstáculo detectado.

3.3.1.3.- Software

Robotics proporciona un VI correspondiente a la programación de la FPGA del Starter Kit, el cual incluye todo lo necesario para establecer comunicación con los componentes montados inicialmente en el robot, lo cual incluye, entre otros, el sensor Parallax PING))) y el servo sobre el que se encuentra montado.

De esta forma, se encuentran preprogramados en las librerías incluidas los segmentos de código empleados para obtener la lectura de la distancia al obstáculo y rotar el servo, con lo que bastará emplear la función Read/Write FPGA para realizar mediciones.

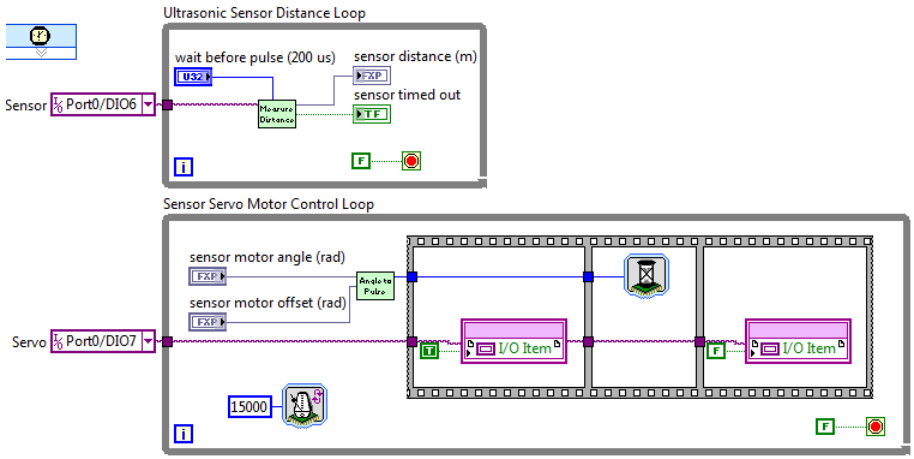

Figura 3.5. Segmento de código que controla el sensor de ultrasonido

3.3.2.- Encoder óptico en cuadratura

3.3.2.1.- Principios teóricos

Un encoder óptico monitorea la reflexión o interrupción de un haz de luz para detectar el movimiento de un disco segmentado giratorio conectado a un motor. Apuntando dos sensores distintos a diferentes lugares de la superficie del disco, el encoder será capaz de producir una salida en cuadratura (un par de señales desfasadas) que un microcontrolador podrá emplear para calcular el número de segmentos que se ha desplazado el disco.

Una vez que poseemos dicha información junto a las características físicas de nuestro robot (diámetro de las ruedas, longitud de su base, etc) es posible determinar con suficiente exactitud cuánto se ha desplazado físicamente cada rueda, y con ello realizar los cálculos necesarios para determinar el desplazamiento del robot en sí. Cubriremos este aspecto en más profundidad en el capítulo 4.

3.3.2.2.- Características del hardware

Cada uno de los dos motores del Starter Kit está provisto a su vez de un NI Encoder Kit de la casa Pitsco (Ref. #35915), el cual incluye un encoder de giro incremental, la base de montaje y el cableado.

Figura 3.6. NI Encoder kit

A la hora de hablar de un encoder uno de los principales atributos que deben conocerse es la resolución del mismo. En este caso es de 400PPR (Pulsos Por Revolución), o expresado en otros términos, 100CPR (Ciclos Por Revolución).

El NI Encoder Kit requiere de alimentación a 0-5V (provista desde la placa sbRIO-9631), y tiene dos pines de salida, A y B, correspondiendo a las dos señales desfasadas del encoder.

Las conexiones de las salidas de los Encoder Kit con la placa sbRIO-9631 se realizan a través de los pines digitales de entrada/salida, correspondiendo al encoder izquierdo los pines Port0/DIO0 (Phase A) y Port0DIO1 (Phase B), y al encoder derecho los siguientes, Port0/DIO2 (Phase A) y Port0/DIO3 (Phase B).

3.3.2.3.- Software

Al igual que ocurría con el sensor de ultrasonido, Robotics incluye en la librería específica del Starter Kit todo lo necesario para establecer una comunicación con el hardware. En concreto, en el VI Starter Kit FPGA el bucle denominado Encoder Loop se encarga de la lectura de los pulsos generados correspondientes a cada motor a través del subVI Increment Encoder Count, el cual incrementará o decrementará un contador en función de que el giro sea horario o antihorario.

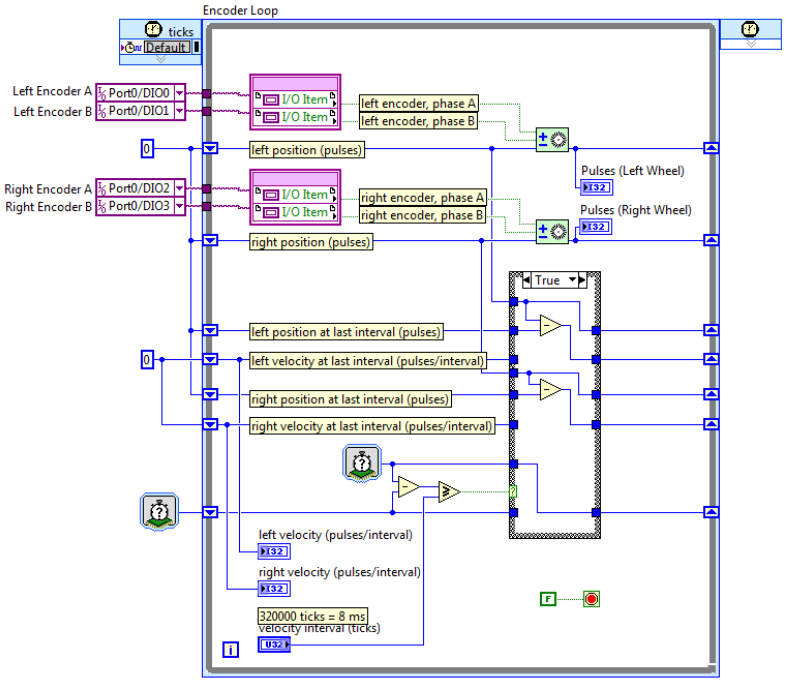

Figura 3.7. Segmento de código de la FPGA que controla los encoders

Para poder acceder directamente al número de pulsos en cada momento, se han agregado dos salidas, Pulses (Left Wheel) y Pulses (Right Wheel). De esta forma, será posible leer dicha información desde otro VI llamando a la función Read/Write FPGA, como veremos en el siguiente capítulo.

3.3.3.- Brújula digital

3.3.3.1.- Principios teóricos

El concepto de brújula digital se basa en el empleo de magnetómetros para evaluar la fuerza y dirección del campo magnético, a partir del cual es posible identificar la dirección y orientación del dispositivo. Constituyen un instrumento de navegación clave a la hora de diseñar sistemas de navegación autónoma, como pueda ser el caso de un robot.

Un aspecto que ha de tenerse en cuenta a la hora de trabajar con brújulas digitales es que, debido al empleo de magnetómetros, sus mediciones resultan extremadamente sensibles frente a interferencias magnéticas. La proximidad de materiales magnéticos, como ciertos metales, introduce errores insalvables en la medida, por lo que su montaje resulta un aspecto muy delicado, debiéndose buscar una configuración que mantenga cierta distancia con grandes superficies metálicas y motores.

3.3.3.2.- Características del hardware

Se ha decidido emplear el módulo CMPS09, una brújula digital compensada en inclinación. Ésta consiste en un acelerómetro en 3 ejes, un magnetómetro en 3 ejes y un procesador de 16 bits, encontrándose especialmente diseñada para compensar errores debidos a inclinaciones de la placa.

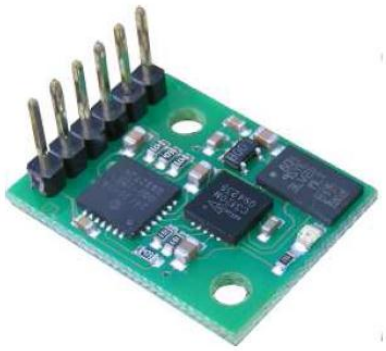

Figura 3.8. Brújula digital CMPS09

El módulo CMPS09 requiere de una alimentación entre 3.3 y 5V, con una corriente de salida de 25mA. Tiene tres modos de funcionamiento a escoger, de los cuales emplearemos el tercero: Modo PWM.

3.3.3.2.1.- Modo PWM: Descripción del funcionamiento

Conforme rote el módulo, será generado un pulso cuadrado, cuya duración en "alta" será proporcional al ángulo en el que se encuentre la brújula. En este modo, el ancho del pulso en "alta" puede variar desde 1ms (equivalente a una orientación de 0º) a 36.99ms (359.9º). Es decir, tiene una sensibilidad de 100us/º con 1ms de offset.
La señal pasa a "baja" durante un periodo de 65ms entre cada pulso, por lo que la duración de cada periodo es de 65ms más la propia duración variable del pulso, es decir, se encuentra entre 66 y 102ms.

Figura 3.9. Ejemplo de señal de salida correspondiente a una lectura de 190º

3.3.3.2.2.- Cableado

Para configurar el modo PWM basta con fijar el pin #3 (*Select PWM*) a la tierra de referencia. Los pines #1 y #6 corresponden a la alimentación, 3.3-5V y 0V (Tierra) respectivamente. El pin #2 da la salida del módulo, en este caso el pulso a medir. Los pines #4 (Calibración) y #5 (Para uso en fábrica) se dejan sin conectar.

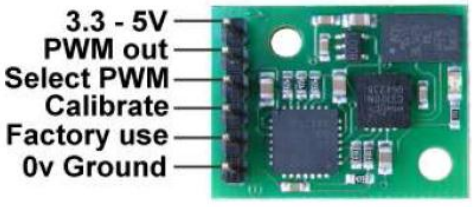

Figura 3.10. CMPS09: Conexionado

En nuestro caso, se ha decidido alimentar la brújula directamente a través de las salidas proporcionadas por la placa sbRIO-9631, puesto que proporcionan un voltaje adecuado (0-5V) y a su vez son parte del cableado necesario para recoger el pulso de salida del módulo.

En concreto se han empleado los pines #42 a #44, correspondientes al puerto P5, el cual proporciona conexiones digitales de entrada/salida.

CMPS09		sbRIO-9631 P5	
Pin	**Descripción**	**Pin**	**Descripción**
#1	0V Ground	#42	D GND
#4	Select PWM	#42	D GND
#5	PWM Out	#43	Port4/DIO5
#6	3.3-5V	#44	5V

Tabla 3.1. Tabla de conexionado del módulo CMPS09

3.3.3.2.3.- Montaje

Para el montaje físico de la brújula sobre el robot, se comenzó fabricando una plataforma metálica la cual se instaló sobre el lateral izquierdo del mismo. De esta forma, se intentó separar la brújula del bloque central del robot para evitar en la mayor medida posible las distorsiones del campo magnético creadas por la proximidad con los motores, la cual es una causa comúnmente reconocida de errores en brújulas digitales.

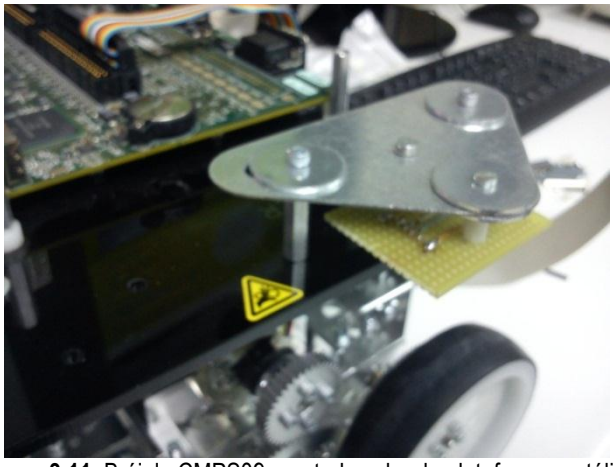

Figura 3.11. Brújula CMPS09 montada sobre la plataforma metálica

Sin embargo, las subsiguientes pruebas denotaron que la plataforma metálica era en sí misma una grave fuente de distorsión, haciendo inservible la brújula si se montaba en dicha configuración.

Finalmente se optó por realizar el montaje sobre un mástil de aluminio instalado en el lateral del Starter Kit. De esta forma las interferencias magnéticas se redujeron a un mínimo aceptable.

3.3.3.3.- Software

Es preciso realizar modificaciones en el VI que controla la FPGA para leer la información procedente del módulo CMPS09. Para leer la salida del modo PWM bastará con incluir un bucle Loop para leer el tiempo que transcurre en alta la señal procedente del pin#3 de la brújula, el cual se ha cableado a la sbRIO a través de la entrada digital Port4/DIO6.

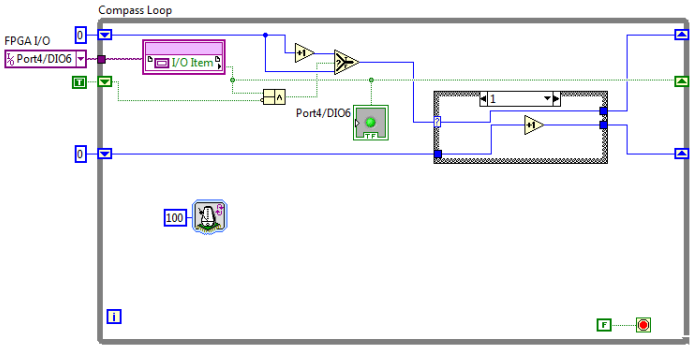

Figura 3.12. Segmento de código de la FPGA que controla el CMPS09

El bucle se encarga de iniciar una variable a cero e incrementarla en 1 en cada iteración (que tienen una duración de 100us). Cuando la señal de entrada pasa de alta a baja, se toma el valor de la variable, restándole el periodo correspondiente a la estancia en baja de cada período (65ms) mas el offset (1ms), siendo el resultado la orientación actual de la brújula, en grados, el cual se empleará para actualizar la variable Heading de la FPGA. Finalmente se reinicia el contador para medir la duración del siguiente período.

3.3.4.- Sensor de final de carrera

3.3.4.1.- Principios teóricos

Un interruptor final de carrera, también conocido como limit switch, es un tipo de sensor capaz de detectar la presencia de un objeto físico. En el caso que nos ocupa, son mecanismos que poseen algún tipo de accionador (palanca, botón, émbolo, etc) que es físicamente activado al hacer contacto con otro objeto.

Cuando un objeto hace contacto y ejerce la suficiente fuerza sobre el accionador del sensor, éste se mueve hasta su límite, donde cambia el estado de los contactos de su circuito interno, variando de esta forma la salida del sensor en función de las entradas y del estado del accionador en cada momento.

Algunas de sus aplicaciones básicas son:

- Detectar presencia/ausencia.
- Conteo.
- Detectar un rango de movimiento.
- Detectar posición.
- Abrir un circuito cuando se den circunstancias no seguras.

3.3.4.2.- Especificaciones del hardware

Se han instalado dos microinterruptores de rodillo modelo Omron V166-1C5 en los laterales delanteros del robot, a una distancia de 6cm del suelo, con la finalidad de que hagan las veces de sensores de contacto, y en el caso de que haya un choque con ellos, envíen una señal que pueda ser interpretada por el robot para abortar el movimiento que se estuviese llevando a cabo, para evitar daños del equipo.

Figura 3.13. Omron V166-1C5

Se ha creído necesario implementar este sistema de seguridad, aun a pesar de contar con el sensor de ultrasonido, puesto que éste solo es capaz de detectar obstáculos que se encuentren a su altura, por lo que un segundo par de sensores situados más a ras de suelo pueden resultar útiles.

El V166-1C5 es un interruptor que posee tres conexiones, NC, NO y COM, las cuales se detallan en la siguiente figura:

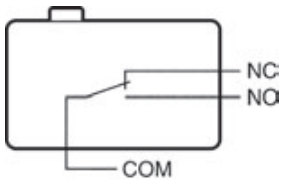

Figura 3.14. Diagrama del V166-1C5: Interruptor en OFF

Cuando el interruptor se encuentra en estado OFF, COM y NC están interconectadas. Al accionarse el interruptor (accionado por una palanca de rodillo mecánica) pasa al estado ON, por lo que pasa a cerrarse el circuito entre NO y COM, quedando NC sin conexión.

3.3.4.2.- Cableado

Se ha empleado el puerto 4 de la placa sbRIO para llevar a cabo el conexionado. Ambos interruptores se alimentan de forma que en estado OFF, mandan una señal de salida de 0V a los dos pines del puerto. Al activarse un interruptor, la señal de salida del mismo pasa a ser de 5V.

V166-1C5		sbRIO-9631 P5	
Pin	**Descripción**	**Pin**	**Descripción**
#1	NC	#50	D GND
#2	COM	#49	Port4/DIO8
#3	NO	#48	5V

Tabla 3.2. Conexiones del final de carrera derecho

V166-1C5		sbRIO-9631 P5	
Pin	**Descripción**	**Pin**	**Descripción**
#1	NC	#50	D GND
#2	COM	#47	Port4/DIO7
#3	NO	#48	5V

Tabla 3.3. Conexiones del final de carrera izquierdo

3.3.4.3.- Software

La implementación en el VI de la FPGA se limitará a un bucle temporal denominado Bumper Loop, que leerá la salida de los sensores de contacto y actualizará en consecuencia las variables booleanas Right/Left Bumper Activation, encontrándose cada salida en estado FALSE si la señal está en baja (0V) y pasando a TRUE en alta (5V).

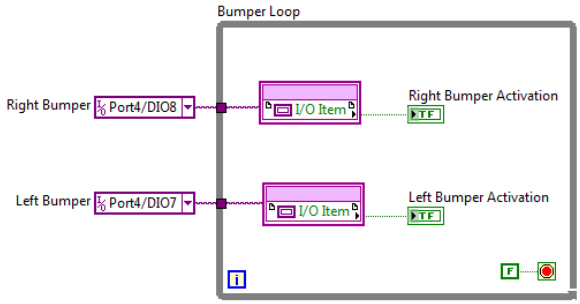

Figura 3.15. Segmento de código de la FPGA que controla los sensores final de carrera

3.4.- Router Inalámbrico

3.4.1.- Introducción

Un añadido interesante a las capacidades del Starter Kit pasa por ser capaces de comunicarnos con él sin necesidad del cable de red. En un robot móvil la necesidad de cableado físico para comunicarse con la plataforma desde un ordenador externo es un claro impedimento en su funcionamiento.

Si bien el robot en este caso nos permite cargar programas autoejecutables sin necesidad de conectividad con un PC, la capacidad de comunicarse en tiempo real de ejecución resulta deseable, cuando no necesaria, en una amplia variedad de aplicaciones.

La solución aportada para ello pasa por la instalación de un router inalámbrico en el Starter Kit, el cual se configurará como un punto de acceso Wifi. De esta forma, un terminal remoto podrá conectarse a la red y acceder al robot, siempre que conozca su IP.

3.4.2.- Montaje

La alimentación del router se sacará directamente de la batería que incorpora el Starter Kit. La configuración física de la plataforma no facilita el acceso a la misma, por lo que será necesario desmontar la base. No es un proceso complicado, pero debe operarse con precaución.

Para desmontar la base, deben tomarse ciertas precauciones. Los pasos a seguir serán:

1.- Asegurarse que los interruptores MASTER y MOTORS del robot están en posición OFF.
2.- Desconectar el cable de alimentación de la placa sbRIO-9631.
3.- Desconectar el cableado de ambos Encoder (conector de cuatro cables: rojo, azul, marrón y amarillo)
4.- Desatornillar y desmontar el sensor de ultrasonido.
5.- Desatornillar los 8 tornillos que unen las dos mitades de la base (2 tornillos en cada una de las esquinas)

Finalmente, podremos acceder al interior (Figura 3.18).

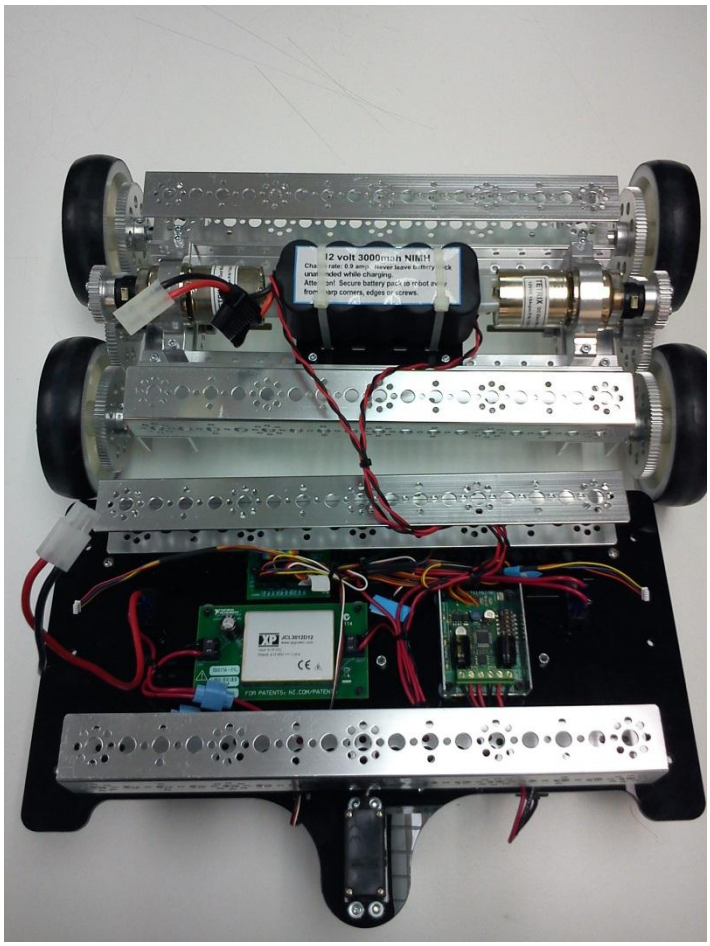

Figura 3.16. Starter Kit desmontado

Puede observarse la batería en la parte superior de la fotografía, mientras que en la parte inferior tenemos un conversor DC/DC modelo JCL3012D12, encargado, junto a otros integrados auxiliares, de extraer de los 12V de la batería diferentes voltajes, que emplean los dispositivos montados en el Starter Kit.

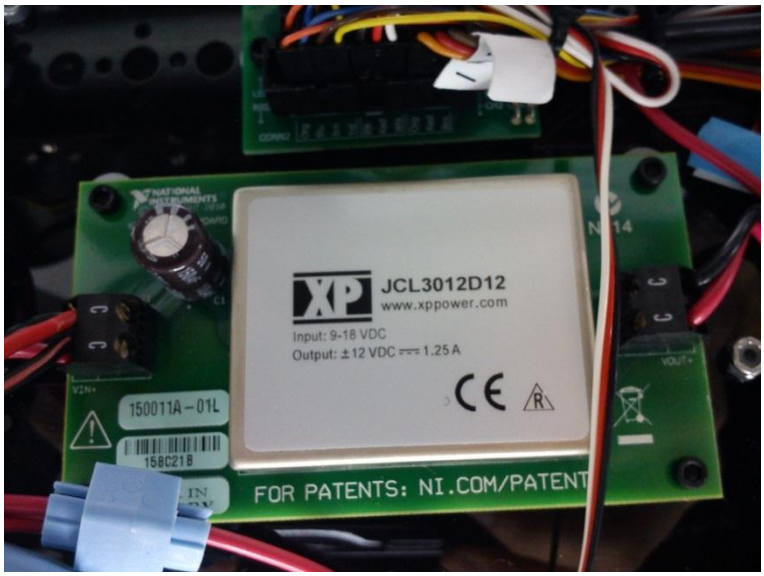

Figura 3.17. Conversor DC/DC JCL3012D12

En el caso del router, al requerir 12V de alimentación no precisamos de dicho conversor, pero aprovecharemos su conector de entrada (a la izquierda de la imagen) para empalmar dos cables con los que extraer los 12V. Finalmente, dichos cables se han soldado a un conector coaxial DC compatible con el puerto de alimentación del router.

Una vez tenemos nuestro cable de alimentación de 12V, solo resta seguir los pasos a la inversa para volver a montar el robot, y seleccionar dónde quiere colocarse el router. La parte trasera del cuerpo del robot ofrece el lugar más conveniente, al tener suficiente espacio y medios para fijarlo. En concreto, se han utilizado tornillos fijados a la base mediante tuercas para dotarla de dos puntos donde fijar el dispositivo, aprovechando que el mismo incluye en su parte inferior dos espacios para tal efecto.

Solo resta conectar el cable de alimentación al router y conectar uno de los puertos LAN al puerto Ethernet de la placa sbRIO.

3.4.3.- Configuración

Para configurar el router como un punto de acceso a través del cual nos conectaremos al Starter Kit, seguiremos los siguientes pasos:

1.- Se conecta el router a un PC vía cable Ethernet.

2.- Desde el navegador de internet, se accede a la página de configuración del router.

3.- Comprobamos su dirección IP y máscara de subred. En este caso la máscara es 255.255.255.0 y observamos que el router con el DHCP activado asigna dinámicamente IPs a los dispositivos conectados en el rango 192.168.2.100 a 192.168.2.199

Se aconseja desactivar el DHCP para evitar la posibilidad de que el router cambie la dirección IP del Starter Kit al conectarlo. Esta IP la fijaremos manualmente. Es preferible hacerla estática y conocida puesto que todos los proyectos a ejecutar, compilar y cargar en el Starter Kit deben conocerla para comunicarse con él.

4.- El Starter Kit se conecta mediante cable Ethernet a un PC con el software Measurement & Automation (MAX) instalado. A través del mismo se accede a la configuración de red de la placa sbRIO-9631 correspondiente, donde deberemos establecer una IP dentro del rango de la subnet del router (192.168.2.100 a .199), con la correspondiente máscara, y el Gateway correspondiente a la dirección IP del router dentro de la red (192.168.2.1). Guardamos los cambios y desconectamos el robot.

Con esto, la configuración está terminada. Basta con conectarse desde un PC a la red Wifi del router para tener acceso al Starter Kit.

CAPÍTULO 4

ALGORITMOS DE NAVEGACIÓN

4.1.- Introducción

En este capítulo se expondrán tres algoritmos de navegación que se han desarrollado e implementado para su funcionamiento en el Robotics Starter Kit:

- Algoritmo de navegación autónoma
- Algoritmo de navegación autónoma con evitación de obstáculos
- Teleoperación

Se detallará tanto su descripción como la implementación del código en LabVIEW.

4.2.- Algoritmo de navegación autónoma

4.2.1.- Introducción

El objetivo consiste en lograr que el robot, a partir de un mapa del entorno conocido previamente, sea capaz de desplazarse desde un punto A hasta un punto B de forma autónoma, esquivando los obstáculos en su camino.

4.2.2.- Especificaciones previas

- El robot se desplazará en línea recta, limitando su giro exclusivamente a cuando se encuentre en parada. De esta forma no solo se simplificará el algoritmo, sino que además los posibles fallos de odometría se mantienen estables y más fácilmente controlables.

- El robot solo girará en segmentos de 90º (giro a izquierda/derecha), los cuales se controlaran con la brújula digital.

- El mapa es conocido previamente, representado por una cuadricula (array de 2 dimensiones). Con el hardware disponible no es posible elaborar algoritmos de mapeo del entorno.

- Todo lo citado anteriormente conlleva que no importará la orientación real del robot (referenciada a la Tierra) sino la dirección inicial en referencia al mapa, ya que nos movemos siempre en giros de 90º dentro del mapa conocido, solo será necesario posicionar el robot en la orientación correcta al inicio del algoritmo.

4.2.3.- Estructura

El algoritmo se divide principalmente en dos fases: Path Planning (o cálculo de ruta) y Desplazamiento:

Path Planning

- Se utiliza el algoritmo A* para determinar la ruta a partir del mapa conocido. Este proceso se encuentra implementado como ejemplo en Labview Robotics.

- Debe recorrerse el array de coordenadas generado por el A*, convirtiendo los datos mediante el módulo “Hoja De Ruta”, el cual generará un array doble conteniendo las órdenes necesarias para alimentar el algoritmo de desplazamiento.

Desplazamiento

- El robot finalmente leerá los array resultado de “Hoja De Ruta” empleando un bucle. En cada ejecución del bucle el robot partirá de un estado de reposo, y ejecutará uno de los dos posibles movimientos indicados por la correspondiente posición del array, es decir: se desplazará X metros en línea recta, o girará 90º a izquierda/derecha.

- Al finalizar el recorrido de la Hoja De Ruta (estado “Fin”), indicando que se ha llegado a la meta designada, el robot dará la señal de STOP que terminará la ejecución del programa.

A continuación estudiaremos en detalle el funcionamiento de todos sus componentes.

4.2.3.1- Implementación

Para la programación de estos pasos, se ha optado por encapsular el código en una estructura Flat Sequence. Esta estructura permite dividir el código en cuadros, los cuales tienen la particularidad de que cada uno de ellos constituye a efectos prácticos un sub-diagrama, ejecutándose el conjunto de forma secuencial, de izquierda a derecha.

La ejecución secuencial permite controlar que ciertos módulos no empiecen a operar hasta que no se disponga de toda la información actualizada necesaria para sus entradas. Cada cuadro de la secuencia no empezará a ejecutarse hasta que su antecesor no haya terminado y dispuesto a su salida (entrada del siguiente) las variables actualizadas resultantes.

Siguiendo esta idea, la estructura Flat Sequence nos permite dividir el código en dos partes claramente diferenciadas, descritas anteriormente: Planificación de Ruta y Desplazamiento. De esta forma, nos aseguraremos que el robot no comenzará su rutina de movimiento hasta que no disponga del conjunto de ordenes completo necesario.

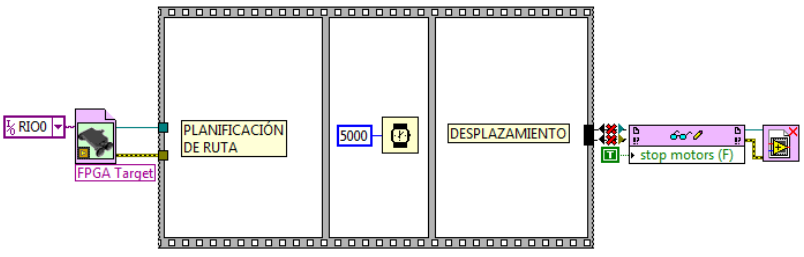

Figura 4.1. Estructura básica del algoritmo

En la Figura 4.1 puede observarse la estructura completa del algoritmo, en la que por motivos de espacio se ha omitido el código interno de cada cuadro, el cual detallaremos en profundidad en las siguientes secciones.

Se comienza con la inicialización de la referencia a la FPGA. A partir de ahí se entra en la primera parte de la Flat Sequence, donde se inicializará el mapa, ejecutando el algoritmo A* y se convertirán los datos mediante la llamada a hoja de Ruta. Las salidas de dicho VI se pasarán al siguiente cuadro, el cual integra una espera de 5 segundos antes de transmitir la salida de Hoja de Ruta al cuadro

encargado del desplazamiento. Una vez el robot ha llegado a la meta, se da la orden de parar los motores y se cierra la comunicación con la FPGA, dando fin a la ejecución del programa.

El motivo por el cual se incluye este periodo de espera entre ambos bloques, que podría parecer redundante, es que se ha observado que la inicialización de la brújula digital precisa de un tiempo levemente mayor que el resto del hardware. En las primeras pruebas se detectó esta particularidad, la cual provocaba que los primeros segundos en los que operaba el robot la brújula arrojaba una salida nula, por lo que en los casos en que un giro fuese la primera orden, provocaba un fallo fatal en el desplazamiento.

La inclusión de un periodo de espera de 5 segundos (el cual podría situarse en cualquier punto del código siempre que estuviese antes del cuadro de Desplazamiento) se mostró más que suficiente para permitir la correcta inicialización de la brújula.

4.2.4- Planificación de ruta: El algoritmo A*

LabVIEW Robotics ofrece diversas funciones para la planificación de ruta, entre las cuales se encuentra una implementación del algoritmo A*. Éste constituye una solución ampliamente usada en multitud de campos, desde el desarrollo de videojuegos a la navegación de vehículos autónomos.

El algoritmo de planificación de ruta A* emplea una búsqueda best-first por grafos, y encuentra un camino de menor coste desde un nodo inicial a un nodo meta dado un mapa. Conforme A* atraviesa el grafo busca el camino de mínimo coste conocido heurístico, a la vez que organiza una cola de prioridades para segmentos alternativos de trayectoria.

Como una de sus principales propiedades destaca el hecho de que, siempre que exista al menos una solución, A* encontrará la ruta mas eficiente.

4.2.4.1.- Descripción

Se emplea la fórmula

$$F(x) = G(x) + H(x)$$

Donde:

- G(x) es la distancia total desde el nodo inicial hasta el actual.
- H(x) es la función heurística que se emplea para estimar el coste del desplazamiento desde el nodo actual hasta la meta.
- F(x) será la aproximación actual del camino más corto a la meta.

Empezando por el nodo inicial, se crea una cola de prioridades de nodos a ser atravesados, conocida como lista abierta. Cuanto menor sea el coste estimado F(x) de un nodo, mayor será su prioridad dentro de la lista.

En cada paso del algoritmo, el nodo con el menor F(x) será eliminado de la cola, los valores F y H de sus vecinos actualizados, y dichos nodos serán añadidos a la cola. Cada nodo vecino será actualizado con su nodo inmediatamente antecesor. Los nodos que no precisen volver a ser evaluados serán movidos a la llamada lista cerrada. Este proceso continuará hasta alcanzar el nodo con el menor valor de F(x), conocido como meta.

Una vez alcanzada la meta, el valor F(x) de dicho nodo será la longitud del camino de menor coste entre el nodo inicial y meta. A su vez, leyendo la sucesión de nodos a la inversa se obtiene dicho camino.

4.2.4.2.- Ejemplo práctico

4.2.4.2.1.- Representación del área de búsqueda

Para ilustrar el concepto, veamos un ejemplo. Partiendo del mapa en la Figura 4.2, supongamos que el robot se encuentra en el punto A y debe calcular la ruta más eficiente al punto B.

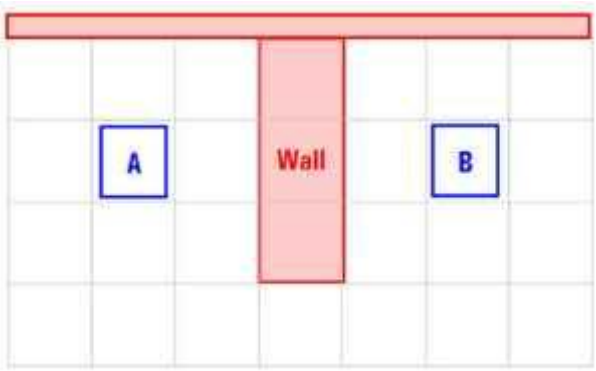

Figura 4.2. Mapa inicial del algoritmo A*

El primer paso a tomar es simplificar el área de búsqueda. En nuestro caso se ha optado por representarla por una cuadrícula, de forma que puede reducirse su expresión a un array de dos dimensiones. Cada elemento del array representa un cuadro de la cuadrícula, y su valor define si puede o no ser atravesado. Esto es, los elementos del array serán los nodos.

El camino se encuentra averiguando que cuadros se deberán tomar para ir del punto A al B, para a continuación atravesarlos yendo del punto medio de un cuadro al del siguiente. Comenzando desde el punto A, se evaluarán los nodos adjuntos y se decidirá el mejor paso, repitiendo el proceso hasta llegar a B.

4.2.4.2.2.- Comienzo de la búsqueda

Para lograrlo, el algoritmo A* forma dos listas: la lista abierta, que recoge los nodos que se consideran en cada momento potenciales candidatos para ser escogidos como el siguiente paso, y la lista cerrada, que contiene los nodos ya evaluados que no es preciso volver a considerar.

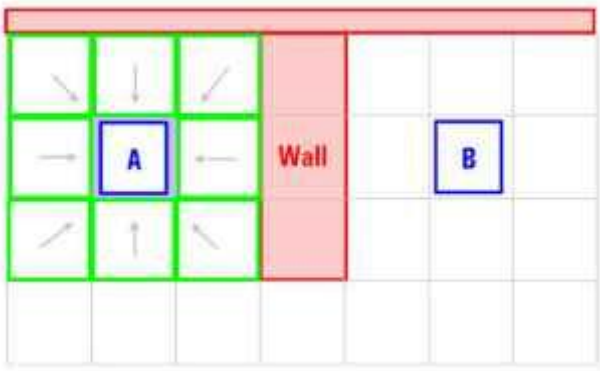

Figura 4.3. Nodos a evaluar en la primera iteración

El algoritmo se inicia incluyendo el nodo inicial en la lista cerrada, evaluando todos sus nodos adjuntos e incluyendo aquellos que sean atravesables en la lista abierta, ignorando el resto, y asignando a cada uno de ellos su nodo antecesor. En el ejemplo todos los nodos adjuntos al punto inicial A son atravesables), y su predecesor será A. En la Figura 4.3 vemos la representación de la primera iteración del algoritmo, donde las casillas verdes indican los nodos incluidos en la lista abierta, y las flechas en cada nodo de dicha lista indican su antecesor.

4.2.4.2.3.- Evaluación del coste

Nos basaremos en la anteriormente expuesta fórmula de F(x) para escoger el nodo adecuado en cada iteración. Concretamente, la trayectoria se generará recorriendo repetidamente la lista abierta y escogiendo el nodo de menor coste F(x).

G(x) se definía como el coste de moverse desde el nodo inicial hasta el nodo actual. En el ejemplo, se asigna un coste de valor 10 para cada movimiento de desplazamiento horizontal o vertical a un nodo adjunto, y un coste de 14 para un movimiento diagonal.

H(x) es el heurístico, y puede estimarse de muy variadas formas. LabVIEW emplea el método Manhattan, donde se calcula el número total de nodos atravesados horizontal y verticalmente para alcanzar el nodo meta B desde el nodo actual, ignorando el movimiento diagonal y cualquier obstáculo existente en el camino. Entonces se multiplica el total por 10, el coste de un movimiento horizontal o vertical.

4.2.4.2.4.- Cálculo del camino

En la Figura 4.4 puede verse el resultado de evaluar el coste de los nodos de la primera iteración. En el nodo resaltado de la derecha, puede observarse que G=10 porque solo está a un nodo de distancia del inicial A, y en dirección horizontal. Por otra parte, el heurístico H se calcula a partir de la distancia Manhattan que le separa de la meta B, en este caso 3 movimientos horizontales y por tanto H=30.

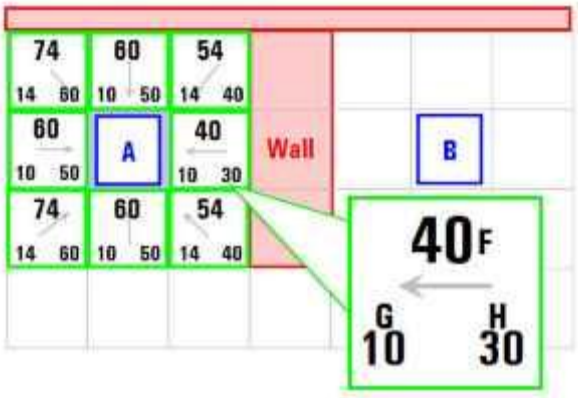

Figura 4.4. Costes de la primera iteración

En cambio, el nodo inmediatamente superior al que estamos evaluando, tiene G=14 al encontrarse un nodo en diagonal respecto al nodo A, y H=40 al encontrarse a 3 movimientos horizontales más 1 vertical del nodo B.

A continuación, se ha de escoger el nodo con menor coste F de todos los que se encuentran en la lista abierta y moverlo a la lista cerrada. Todos los nodos adyacentes a éste y que no pertenezcan a la lista cerrada deberán ser evaluados, y aquellos que sean atravesables añadidos a la lista abierta. Esta nueva evaluación incluye a los nodos de la lista abierta, dado que existe la posibilidad de que en futuras iteraciones sea escogido un mejor camino.

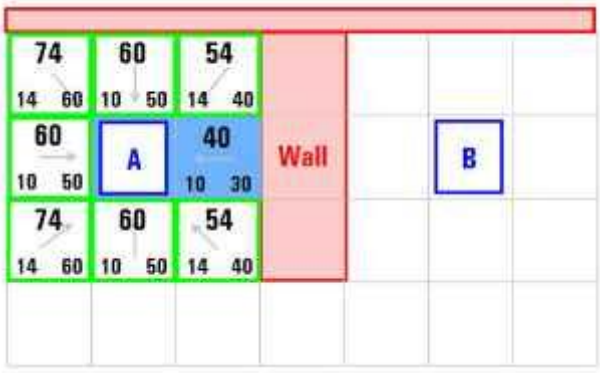

Figura 4.5. Elección del primer nodo

En la Figura 4.5 puede observarse que de los 9 nodos de la primera lista abierta, quedan 8 (resaltados en verde), mientras que el nodo de menor coste F se moverá a la lista cerrada (en azul, F=40). El siguiente paso será evaluar los nodos adyacentes al recién incluido en la lista cerrada: los que se encuentran inmediatamente a su derecha están categorizados como no atravesables, por lo que serán ignorados, al igual que lo será el de su izquierda (nodo A) al encontrarse incluido en la lista cerrada.

Los restantes 4 nodos adyacentes se encuentran ya en la lista abierta, así que será necesario comprobar si el camino a dichos nodos tendría menor coste si nos desplazásemos tomando como punto intermedio el nodo actual, empleando los valores G como referencia.

Tomando por ejemplo el nodo situado por debajo del actual (en azul), vemos que su coste G=14. Si quisiéramos llegar a dicho nodo a través del actual (cuyo coste es G=10), deberíamos desplazarnos un nodo en vertical, sumando 10 al coste que en total sería G=20. Al ser 14<20 es obvio que resultaría más óptimo desplazarnos directamente en diagonal desde A hasta dicho nodo, que tomar el camino intermedio por el nodo actual.

Al repetir este proceso para el resto de nodos de la lista abierta, puede comprobarse que no se encontrará ningún camino que disminuya su coste tomando el nodo actual como intermedio, por lo que no se modifica nada. Una vez finalizadas todas las evaluaciones, se pasa de nuevo a encontrar el mejor nodo siguiente en la lista abierta. En la Figura 4.6 se resalta dicho nodo.

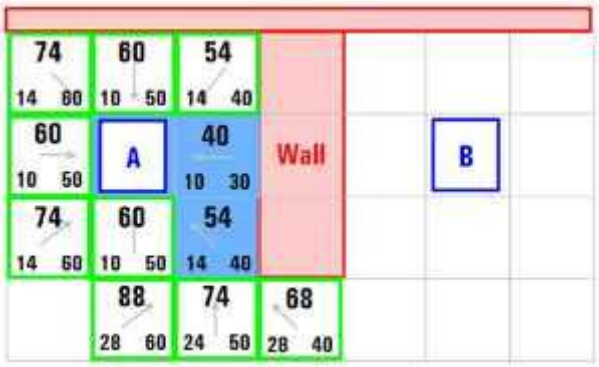

Figura 4.6. Elección del segundo nodo en la ruta

Para elaborar el camino completo, debe repetirse el proceso hasta que se añada el nodo meta B a la lista cerrada, dándose por finalizado el proceso de búsqueda (Figura 4.7).

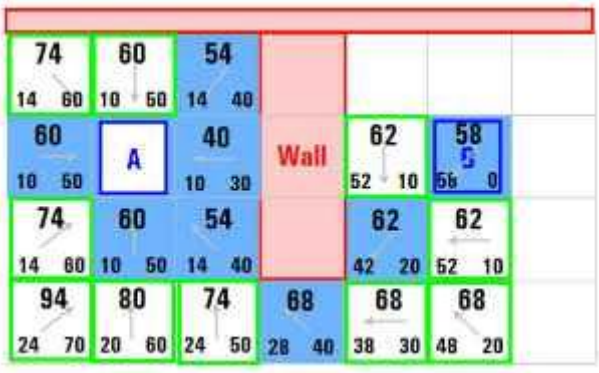

Figura 4.7. Algoritmo finalizado

Solo resta calcular la trayectoria, para lo cual tomaremos como punto de partida el nodo meta, siguiendo a su nodo antecedente. Se repetirá el proceso hasta encontrar el nodo inicial. El recorrido dictado por los nodos antecedentes será el camino de menor coste del punto A al punto B (Figura 4.8).

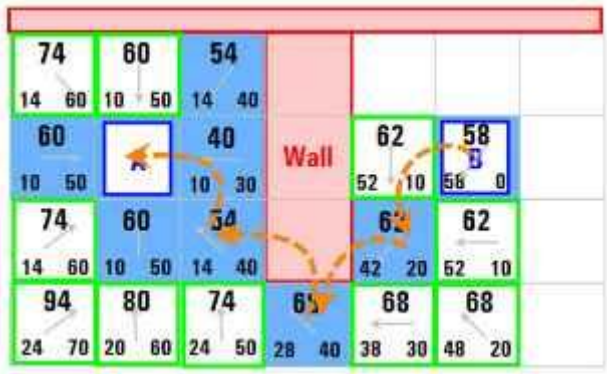

Figura 4.8. Cálculo de la ruta final

4.2.4.3.- Implementación

Robotics ofrece entre sus librerías varios algoritmos de Path Planning, entre los cuales se encuentra una función dedicada al cálculo del A* (denominada A Star Plan), lo cual facilita su implementación.

Sin embargo, el A* está programado inicialmente para calcular la ruta más óptima evaluando el movimiento en las ocho direcciones posibles en cada punto, incluyendo movimientos en diagonal. Dado que nuestro planteamiento inicial es que el robot se desplace en giros de 90º, el A* no cumple con dicho requisito.

Para lograr que el A* calcule un camino sin diagonales, basta con modificar la librería *NI_Robotics_OccupancyGridWorldMap.lvclass:GetNeighbours.vi* , en concreto editar el coste de movimiento para las casillas posicionadas diagonalmente respecto a la referencia actual. Fijaremos el coste de 50, el cual es lo suficientemente alto para que al evaluar el camino se descarten los movimientos en diagonal.

Nótese que se está modificando un VI perteneciente a una librería propia de LabVIEW, que se encuentra ligada a la instalación del programa. En el caso de actualizar la versión o se exportase el código a otro terminal, sería necesario volver a modificar el VI correspondiente de la librería antes indicada.

Una vez realizada esta modificación, podemos pasar a implementar el segmento de código que realiza la Planificación de Ruta (Figura 4.9)

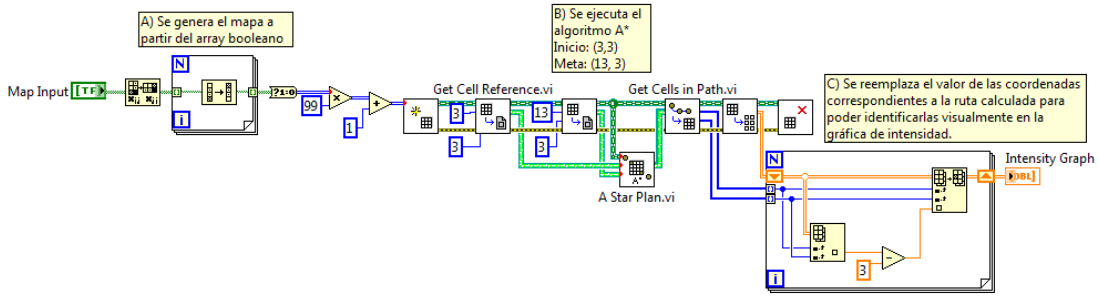

Figura 4.9. Código para la realización de la Planificación de Ruta

Se parte de un mapa de dos dimensiones, para el cual se utiliza un array 2D booleano como método de entrada de datos. Procesaremos dicho array convirtiendo sus valores booleanos en dos posibles costes para cada posición: 1 (coordenada atravesable) o 100 (no atravesable, léase una pared u obstáculo de cualquier tipo). Con este array alimentaremos la entrada de la función Create Occupancy Grid Map, el cual generará un mapa en la forma de una cuadrícula de celdas con variables asociadas.

Esta conversión de formato es requerida para trabajar con la función A Star Plan, y resulta especialmente útil al disponer LabVIEW de una serie de funciones específicas para trabajar con Occupancy Grids.

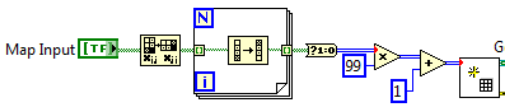

Figura 4.10. Inicialización del mapa

Una vez que tenemos el mapa creado, llega el momento de planificar la ruta en sí. Para ello necesitamos especificar los nodos inicio y final de ruta en nuestro mapa, para lo cual emplearemos la función Get Cell Reference. Llamaremos a dicha función un par de veces, para establecer ambos nodos. Dicha función precisa como entrada el mapa y las coordenadas del punto, dando como salida la referencia a la celda del mapa Occupancy Grid.

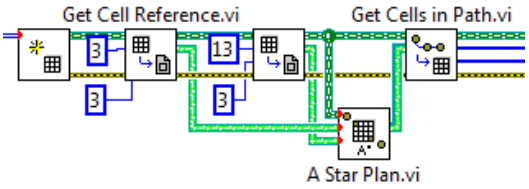

Figura 4.11. Ejecución del A*

Una vez que tenemos el mapa y las referencias a las celdas inicio y final, solo resta emplear dichos datos para invocar a la función A Star Plan, la cual se encargará de llevar a cabo el algoritmo de planificación de ruta. Para leer dicha información basta con tomar la salida de dicha función Path Reference como entrada de la función Get Cells in Path, la cual devolverá dos arrays de una dimensión que contienen respectivamente las coordenadas X e Y de toda la ruta. Estas coordenadas se encuentran ordenadas, siendo la primera el nodo inicial y la última el nodo meta.

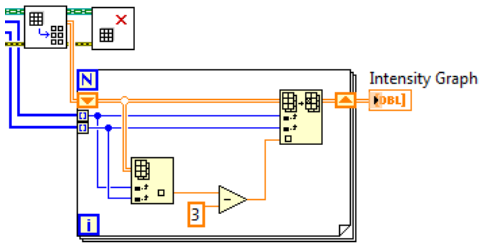

Figura 4.12. Representación gráfica del mapa con la ruta trazada

La planificación de ruta ha terminado, pero a continuación se realiza una pequeña modificación del mapa Occupancy Grid para mostrar en la interfaz gráfica el mapa con la ruta calculada. En este caso, en reducir en 3 el valor del coste de las coordenadas pertenecientes a la ruta, se representarán con un tono azul, mientras que las casillas atravesables serán blancas, y las no atravesables negras.

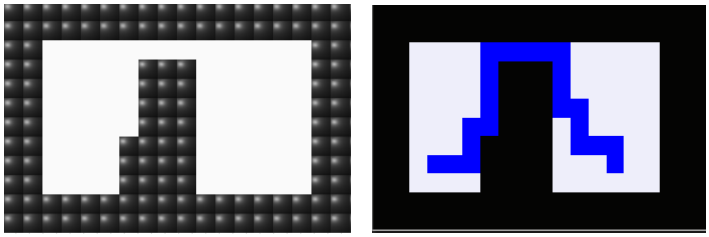

Figura 4.13. Mapa de entrada (izquierda) y gráfica con la ruta generada (derecha)

Como podemos ver en el ejemplo, se ha generado de forma satisfactoria una ruta siguiendo los requisitos impuestos por nuestro algoritmo de desplazamiento.

4.2.5.- Conversión de datos: Módulo Hoja de Ruta

4.2.5.1.- Descripción

Una vez que tenemos la ruta calculada mediante el algoritmo A*, es necesario convertir los datos a un formato que pueda manejar el módulo de desplazamiento. En este caso recogeremos la ruta en forma de un par de arrays de coordenadas X e Y, correspondientes a las salidas de la función Get Cells In Path. Dichos array representan la serie de coordenadas que componen la ruta calculada.

Este conjunto de coordenadas lo traduciremos en dos arrays de números enteros, los cuales contendrán las instrucciones necesarias para recorrer la ruta, de forma que solo deberemos alimentar el algoritmo de desplazamiento con dichas órdenes de forma secuencial.

Dichos arrays podrán contener cuatro tipos de órdenes. El array Tipo Movimiento indica la clase de movimiento que deberá realizar el robot a continuación, y su posición correspondiente del array Dato Movimiento especifica la variable necesaria en cada caso.

Tipo Movimiento	Dato Movimiento	Orden
0	X	Avanzar X*Escala metros
1	90	Girar a la derecha 90º
1	-90	Girar a la izquierda 90º
2	2	Fin (llegada a meta)

Tabla 4.1. Posibles tipos de órdenes generadas por Hoja de Ruta

De esta forma, el módulo Hoja de Ruta generará la lista de órdenes de forma previa al comienzo del algoritmo de desplazamiento.

4.2.5.2.- Implementación

Se ha creado un subVI denominado HojaDeRuta para encapsular el segmento de código necesario para la conversión de datos. Se constituye de tres entradas y tres salidas. Las entradas serán:

- X: Alimentado por la salida “X coordinates” de la función Get Cells in Path
- Y: Alimentado por la salida “Y coordinates” de la susodicha función.
- Orientación: Constante que indica la orientación inicial del robot.

En cuanto a las salidas:

- Tipo Movimiento: Array de una dimensión que indica la clase de movimiento a realizar, correspondiendo cada posición del array a una futura iteración del algoritmo de desplazamiento.
- Dato Movimiento: Array de una dimensión ligado a Tipo Movimiento, donde cada posición se corresponde a su homónima en el primer array, e incluye información necesaria para llevar a cabo cada acción.
- Array Orientacion: El cual almacena las diferentes orientaciones que deberá tomar el robot conforme recorre la ruta planificada. En un principio se añadió para comprobar el correcto funcionamiento de la conversión de datos, y depurar posibles errores.

Internamente, HojaDeRuta sigue una estructura basada en un bucle for que recorre una a una todas las coordenadas de la ruta. En cada iteración la coordenada actual es comparada con la anterior empleando Feedback Nodes (registros que almacenan el valor de la variable en la última iteración), en base a la cual se determinará que tipo de movimiento se ha realizado por última vez.

Una serie de estructuras Case anidadas completarán el proceso de identificar los datos relativos a dicho movimiento, y actualizarán los arrays Tipo y Dato Movimiento en consonancia.

4.2.5.2.1- Inicialización

La función comienza inicializando las variables y arrays necesarios, esto es, los correspondientes a los arrays de Orientaciones, Tipo y Dato Movimiento, así como variables auxiliares como la que indica en cada iteración el índice de los array a modificar, y una variable "Nº de casillas" que se encarga de almacenar y fijar la distancia para cada movimiento del tipo "Desplazamiento en línea recta".

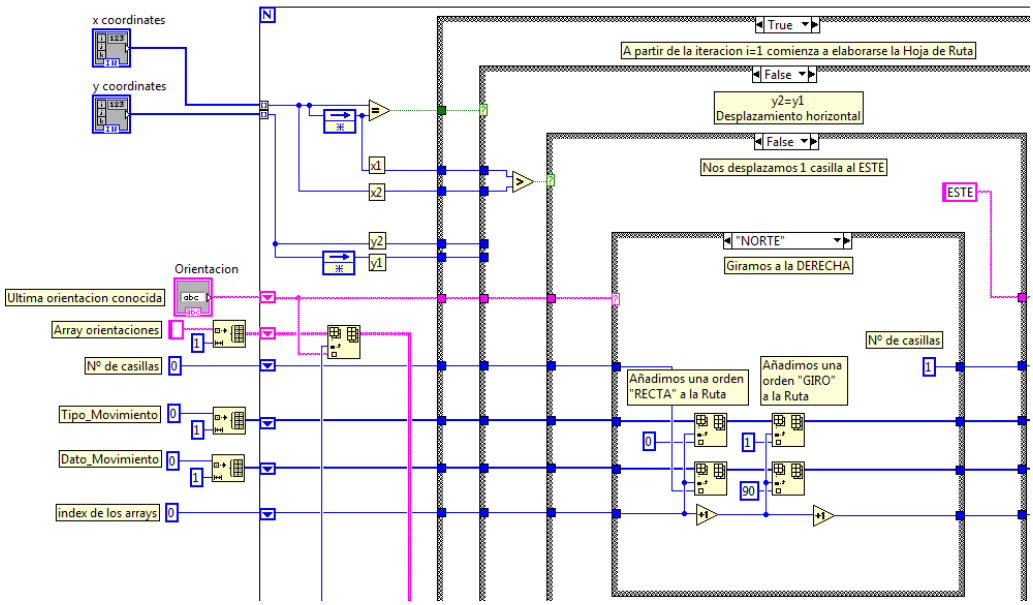

Figura 4.14. Inicialización de variables y uno de los posibles casos

4.2.5.2.2- Interpretación de la ruta

El primer Case se encarga de que no se realice ningún cálculo en la iteración inicial i=0, puesto que en el momento de partida no se dispone de ningún dato de la iteración inmediatamente anterior.

Una vez que el bucle llega a la segunda iteración, comienza la elaboración de la Hoja de Ruta. Para distinguir que tipo de movimiento se ha realizado, recurrimos a comparar las coordenadas (X,Y) actuales con las de la iteración anterior, de forma que:

- Si solo una de las coordenadas ha variado, significa que estamos en el estado "Avanzar en línea recta", por lo que se incrementa en 1 la variable "Nº de casillas". Nótese que en este estado no almacenamos todavía la información de la recta en el array.

- Si dos de las coordenadas han variado, se interpreta que el último movimiento ha sido un "Giro". A partir de la variación relativa de las coordenadas y de la última orientación del robot (recordar que partimos de una orientación inicial conocida) se calcula la dirección del giro y la nueva orientación del robot.

Finalmente, se almacenan dos tipos de movimiento seguidos: un “Desplazamiento en línea recta” correspondiente a la última recta antes del giro (donde pasaremos el “Nº de casillas” al array Datos) y a continuación almacenaremos el propio “Giro” (almacenando en Datos un entero, 90 o -90, en función de la dirección del mismo). La variable Nº de casillas es reiniciada a cero.

A destacar que todas las variables se transmiten de una iteración a la siguiente mediante el uso de Registros de Desplazamiento en el bucle.

Dada la cantidad de estructuras Case que han sido necesarias anidar para realizar este módulo, resulta excesivamente complejo mostrar la estructura completa del programa en imágenes. Por ello, a continuación se incluye el pseudocódigo desarrollado a priori para programar la lógica que sigue la función HojaDeRuta a la hora de evaluar la dirección de los giros.
Nótese que solo se muestra el primer caso implementado, siendo el resto variaciones del mismo en función de la orientación y coordenadas que hayan variado.

```
Int Aux = “NORTE”
Int casillas, j = 0;

For (i=0; i<long(hojaderuta); i++) {
        If (y2 == y1)
                If (x2 < x1)     ## Nos desplazamos una casilla al OESTE
                        Switch (Aux) {
                                case OESTE       ## Seguimos en línea recta
                                        casillas = casillas+1;
                                case NORTE       ## Giro a la izquierda
                                        Tipo_movimiento(j) = “RECTA”; # Representado por “0”
                                        Dato_movimiento(j) = casillas;
                                        Tipo_movimiento(j+1) = “GIRO” # “1”
                                        Dato_movimiento(j+1) = -90;
                                        Casillas = 1;
                                        j = j+2;
                                case SUR ## Giro a la derecha
                                        Tipo_movimiento(j) = “RECTA”;
                                        Dato_movimiento(j) = casillas;
                                        Tipo_movimiento(j+1) = “GIRO”
                                        Dato_movimiento(j+1) = 90;
                                        Casillas = 1;
                                        j = j+2;
                        }
                        Aux = “OESTE”;
                Else if ( x2 > x1) ## Nos desplazamos una casilla al ESTE
                        (...)

        Else If (x2 == x1)
                If (y2 < y1)     ## Nos desplazamos una casilla al NORTE
                        (...)

                Else if ( y2 > y1) ## Nos desplazamos una casilla al SUR
                        (...)
```

4.2.5.2.3- Finalización

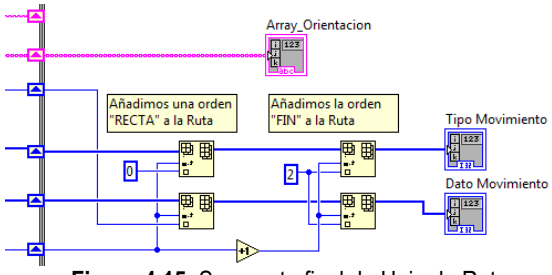

Figura 4.15. Segmento final de Hoja de Ruta

Tras recorrer todas las coordenadas de la ruta, el módulo finaliza añadiendo una última orden “Avanzar en línea recta” tomando el último valor de la variable “Nº de casillas”, y la orden “Fin”, que indica que se ha llegado al destino.

4.2.6.- Algoritmo de desplazamiento

4.2.6.1.- Descripción

Hasta este momento se ha calculado la ruta óptima y se han convertido los datos a una serie de órdenes. El paso siguiente es aplicar de forma secuencial el conjunto de órdenes contenidas en los arrays Tipo y Dato Movimiento al movimiento del robot, de forma que se desplace siguiendo la susodicha ruta.

El bloque de Desplazamiento se constituye principalmente de un bucle For al que se introducen los arrays Tipo y Dato a través de Auto-Indexed Tunnels. Al configurar de esta forma el bucle, significa que los datos de los arrays se introducirán de uno en uno y por orden. O dicho de otra forma, el bucle For se configura automáticamente para generar tantas iteraciones como posiciones tengan dichos arrays, procesando un par de entrada (Tipo,Dato) por iteración.

En cada iteración del bucle se lleva a cabo una comprobación a través de un Case para obrar en consonancia al tipo de movimiento requerido. A su vez, cada tipo de movimiento (en su correspondiente caso) contiene un bucle while con la condición de paro ligada a la finalización del movimiento. De esta forma, cada iteración del bucle For principal está ligada secuencialmente a una orden de la Hoja De Ruta.

Hay tres casos posibles, dependiendo del valor de la posición actual del array Tipo Movimiento:

- Caso “0”: Avance en línea recta
- Caso “1”: Giro.
- Caso “2”: Fin

El caso Fin se ha incluido por su utilidad en futuras modificaciones del algoritmo de navegación. En esta versión constituirá un código en blanco, puesto que el bucle For finalizará con los arrays Tipo/Dato Movimiento, y sabemos que el caso Fin se da en la ultima posición de dichos arrays.

4.2.6.2.- Caso Recta

4.2.6.2.1.- Descripción

Esta orden indica al robot que avance en línea recta una distancia especificada por el producto de la posición correspondiente del array Dato (que indica el número de casillas a desplazarse) por la Escala del mapa, que es una variable de entrada configurable previa a la ejecución, que indica a cuantos metros equivale cada cuadrícula del mapa.

Nótese que dentro de este caso hay un segundo caso anidado para prevenir de que si la variable Dato es nula, no se lleve a cabo ninguna acción y se pase a la siguiente iteración (u orden de movimiento).

4.2.6.2.2.- Implementación

Se comienza activando los motores y registrando las variables Init Pulses L e Init Pulses R, las cuales indican el valor de los pulsos del encoder de cada motor antes de comenzar el movimiento.

A continuación se inicia un bucle While, en el cual se fijan las velocidades de ambos motores a 8.4 radianes por segundo a través de una función Read/Write FPGA. Nótese que el motor derecho, al estar montado al contrario que el izquierdo, requiere ser configurado en sentido contrario al izquierdo para que las cuatro ruedas giren en la misma dirección, de ahí el valor negativo de la variable Right CCW Velocity Setpoint.

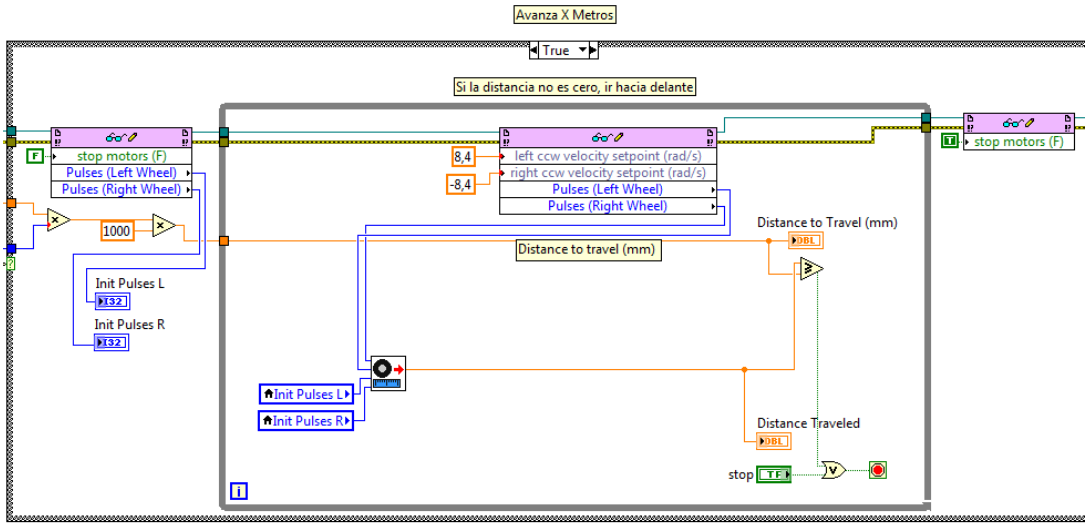

Figura 4.16. Caso "Avanzar en línea recta"

En cada iteración se irá calculando la distancia que se ha desplazado el robot mediante la odometría de las ruedas a partir de la variación del número de pulsos en los correspondientes encoders. Todo el cálculo de la odometría se ha encapsulado en un subVI a la que hemos denominado Odometry, a la que dedicaremos una sección a continuación.

La condición de finalización del bucle es que se mida una distancia desplazada igual o mayor a la distancia que se requería mover el robot, indicada por el producto de Dato Movimiento con la Escala.

Una vez que se sale del bucle, se activa la señal de parada de los motores a través de la función Read/Write FPGA, y se continúa hacia la siguiente iteración del bucle For principal.

4.2.6.2.3.- Odometría

La odometría es la técnica que estudia la estimación de la posición de un vehículo a partir del desplazamiento de sus ruedas. Resulta necesario recalcar que no estamos determinando su posición, sino estimándola a partir de los datos de sus encoders. La odometría introduce errores acumulativos en la medición del desplazamiento, cosa que deberá tenerse siempre presente.

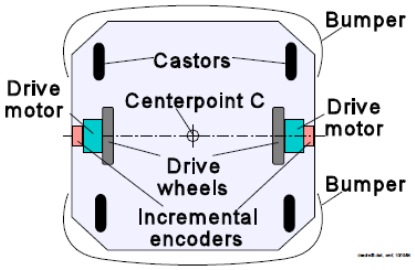

Figura 4.17. Configuración de un robot diferencial

En la Figura 4.17 puede observarse una configuración típica de un robot diferencial de cuatro ruedas, como es el caso que nos ocupa. Disponemos de encoders montados en los dos motores, que cuentan las revoluciones de las ruedas. El robot podrá así llevar a cabo ´la odometría empleando simples ecuaciones geométricas para calcular la posición relativa del vehículo erspecto a una posición inicial conocida.

4.2.6.2.3.1.- Descripción

Supongamos que, dado un intervalo de muestreo I, las ruedas izquierda y derecha muestran un incremento del número de pulsos de N_L y N_R respectivamente. Podemos determinar el factor:

$$c_m = \pi D_n / n C_e$$

Donde:

c_m: Factor de conversion de los pulsos del encoder a desplazamiento lineal de la rueda.
D_n: Diámetro nominal de la rueda (en mm)
C_e: Resolución del encoder (en Pulsos Por Revolución)
n: Relación de transmision del engranaje redactor entre el motor (donde se encuentra el encoder) y la rueda

A su vez, podemos calcular el incremento de distancia de cada rueda como

$$\Delta U_{L/R} = c_m N_{L/R}$$

Finalmente, calcularemos el desplazamiento lineal del punto central del robot C, como:

$$\Delta U = (\Delta U_L + \Delta U_R)/2.$$

Adicionalmente, puede calcularse la variación de la orientación mediante la ecuación:

$$\Delta\theta = (\Delta U_R - \Delta U_L)/b$$

Donde “b” es la base del robot (distancia entre ambas ruedas).

4.2.6.2.3.2.- Implementación

Atendiendo a las especificaciones del Starter Kit, y midiendo su factor de relación de transmisión “n” como n=2, se han programado los cálculos matemáticos necesarios para llevar a cabo la odometría en el caso de un desplazamiento en línea recta, que es el que nos atañe.

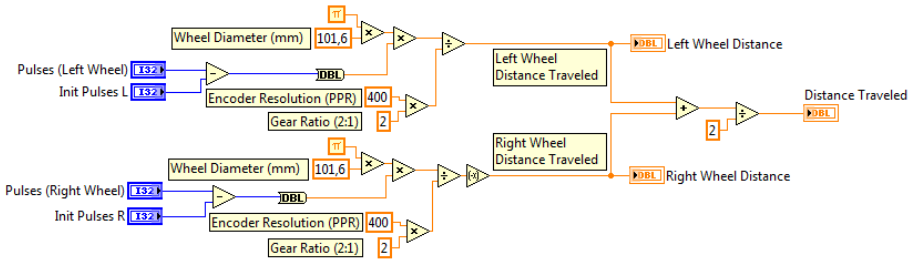

Figura 4.18. SubVI "Odometry"

4.2.6.3.- Caso Giro

4.2.6.3.1.- Descripción

La orden Giro se divide a su vez en dos órdenes posibles a partir de un Case que lee la variable Dato. Si ésta indica el valor "90" se ejecutará un giro de 90º a la derecha, mientras que si indica "-90" se ejecutará un giro de 90º a la izquierda.

Aunque en un principio puede estimarse la variación de la orientación del robot mediante la odometría, se decidió por incorporar una brújula digital al hardware, de forma que dispondremos de una medición directa y no de una estimación.

En el caso del giro, a diferencia del anterior, no se ha optado por mantener una velocidad constante de los motores durante todo el movimiento. Cuando se alcance cierto incremento del ángulo de orientación se reducirá drásticamente la velocidad.

Durante las primeras pruebas del algoritmo, los giros se producían con errores superiores a lo esperado. Tras realizar comprobaciones exhaustivas y descartar posibles fuentes de error como la propia brújula, se concluyó que el motivo era la propia inercia del movimiento. Si la velocidad de giro superaba cierto umbral, el tiempo transcurrido entre dar la orden de parada al alcanzar los 90º requeridos y que los motores se detuviesen completamente era el suficiente como para que el robot girase del orden de 3-6º extra.

Para evitar esta situación hay dos opciones: reducir la velocidad de giro al mínimo o, como se ha optado finalmente, mantener una velocidad normal hasta llegar a cierto umbral de cercanía con la orientación deseada, a partir del cual se reduce la velocidad al mínimo para que el giro sea lo más exacto posible.

4.2.6.3.2.- Implementación

A continuación se expone el código correspondiente al giro de 90º a la derecha. Dado que ambos casos parten de la misma base, para el caso de giro a la izquierda solo se indicaran los cambios respectivos.

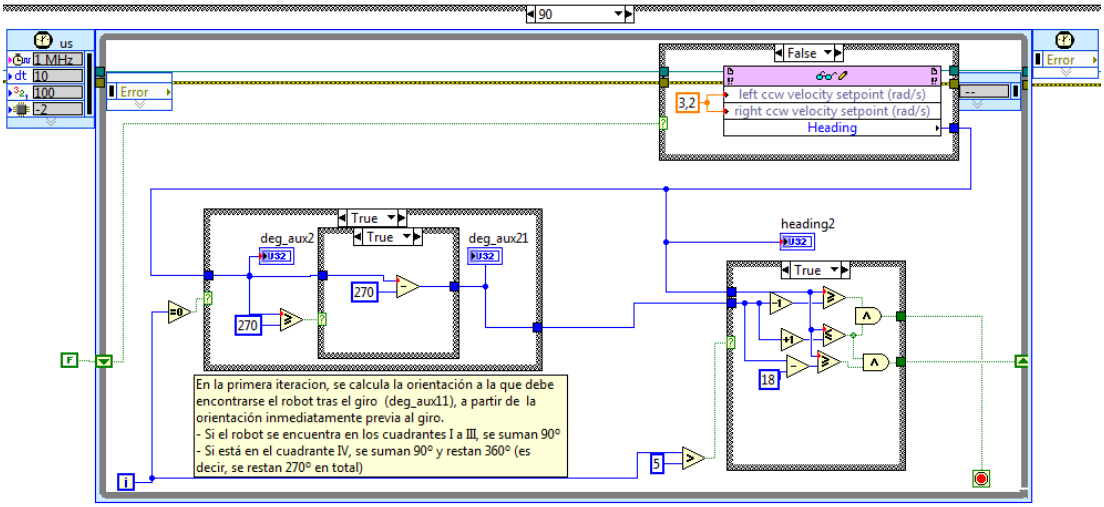

Figura 4.19. Caso "Giro"

El caso Giro se basa en un bucle Timed Loop, el cual se inicia fijando la velocidad de las ruedas en contraposición a 3.2 rad/s mediante la función Read/Write FPGA. En el caso contrario se fijan ambas a -3.2rad/s. Dicha función también se empleará para leer la salida de la brújula digital (variable Heading).

En la primera iteración del bucle se lee el Heading inicial y se calcula a partir de él el Heading destino. Si nos encontramos en los cuadrantes I al III simplemente se sumaran 90º al Heading inicial. De encontrarnos en el cuadrante IV, se restarán 270º.

En el caso de giro a la izquierda, si el robot se encuentra en los cuadrantes II a IV, se restan 90º, mientras que si está en el cuadrante I, se restan 90º y suman 360º (es decir, se suman 270º en total).

En las siguientes iteraciones se deja al bucle funcionar normalmente, para evitar finalizar abruptamente el giro apenas comenzado debido a un posible error introducido por la brújula en la que diese un valor 1 grado por debajo del heading inicial en las iteraciones iniciales. Esto es una posibilidad, aunque remota, ya que la resolución a la que nos permite trabajar la brújula montada sobre el robot es del orden de unidades.

A partir de la séptima iteración (recordar que el bucle parte de la iteración cero) se comienza a evaluar el incremento de Heading respecto al inicial, estableciendo como condición de finalización del bucle que se alcance el Heading destino calculado, con un error contemplado de +1/-1 grados.

A su vez, también se establece un umbral por el cual cuando se alcance una distancia de 18º con el Heading destino, se activará una señal booleana, la cual modificará la velocidad de giro mediante una estructura Case, reduciéndola para evitar el problema expuesto en la sección anterior.

El bucle finalizará con una orden de parada de los motores, al igual que la orden Recta, empleando la función Read/Write FPGA.

4.3.- Navegación autónoma con evitación de obstáculos

4.3.1.- Introducción

Este algoritmo constituye una modificación del anterior. Se parte del mismo objetivo, esto es, que el robot navegue un mapa conocido de antemano para ir del punto inicial a la meta, pero dotándole de la capacidad de evitar obstáculos que no se encontrasen en el mapa inicial.

4.3.2.- Descripción

Se parte del código del algoritmo anterior, pero encapsulándolo en un bucle While y haciendo que todas las variables importantes (como son las que almacenan el mapa, ruta, etc) se desplacen de una iteración a la siguiente empleando Registros de Desplazamiento.

Para llevar a cabo esta aproximación, se introduce la necesidad de emplear el sensor de ultrasonidos. Éste detectará cualquier obstáculo a una distancia de una cuadricula del mapa, y detendrá la ejecución del algoritmo de navegación si dicho obstáculo no corresponde con una casilla no atravesable del mapa inicial.

Dada la situación en que se haya detenido el algoritmo por un obstáculo no previsto, se iniciará una nueva iteración del bucle While, es decir, a efectos prácticos estamos iniciando una nueva instancia del algoritmo de navegación, pero en la que emplearemos como mapa inicial conocido el mapa anterior con la casilla del nuevo obstáculo actualizada, y como punto inicial la coordenada en la que se encontrase el robot en el momento de la detención de la anterior ejecución. La coordenada meta se mantendrá, naturalmente.

El bucle While principal tiene como condición de finalización la llegada a meta, por lo tanto, mientras que no se alcance ésta el robot continuará desplazándose por la ruta calculada, y calculando nuevas rutas en el caso de encontrar obstáculos no previstos.

4.3.3.- Implementación

Partiendo de que la estructura y funcionamiento básico de este algoritmo es el mismo que el anteriormente explicado, nos centraremos únicamente en explicar los añadidos necesarios para que se procese la información de una iteración del bucle a la siguiente.

Como ya se ha comentado, el código será encerrado en un bucle While, fuera del cual inicializaremos las variables que queremos que se propaguen a posteriores iteraciones si encontrásemos un obstáculo, mediante Registros de Desplazamiento. En concreto, será necesario inicializar el mapa, las coordenadas de inicio y meta, y la orientación actual del robot.

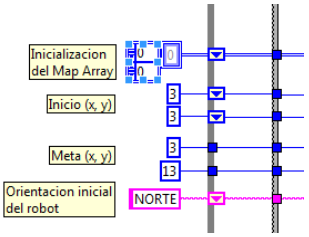

Figura 4.20. Inicialización de variables

La primera parte del código, correspondiente a la Planificación de Ruta, permanecerá inalterada en la primera iteración del bucle. En posteriores iteraciones, mediante su encapsulamiento en una estructura Case, cambiaremos el susodicho código por otro similar, pero que descarta la inicialización del mapa, tomando el mapa actualizado de la iteración inmediatamente anterior, así como la posición inicial como la posición en la que se encontraba el robot entonces. Esta nueva porción de código puede observarse en la Figura 4.21.

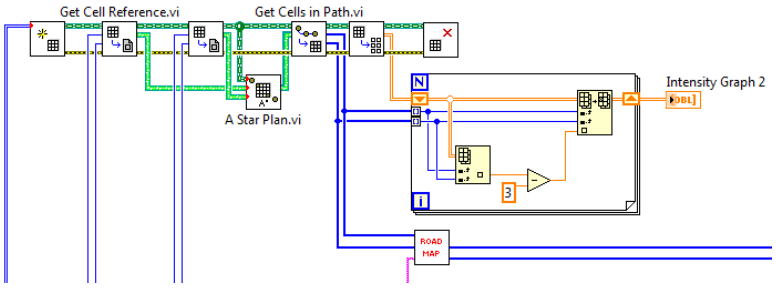

Figura 4.21. Generación de Hoja de Ruta alternativa

En la siguiente parte del código, la correspondiente al algoritmo de desplazamiento, realizaremos nuevos añadidos en cada tipo de movimiento.

Dentro de cada caso Giro, se añade un breve segmento de código en forma de estructura Case que determina la nueva orientación del robot tras el giro, en función de la orientación anterior y el tipo de giro que se ha realizado.

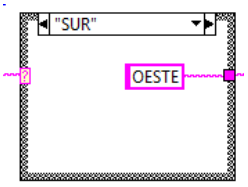

Figura 4.22. Actualización de la orientación del robot

En el caso Recta, se ha añadido una porción de código que va actualizando en cada momento la coordenada del mapa en la que nos encontramos, a la par que comprueba mediante el sensor de ultrasonidos si en la casilla siguiente de nuestra ruta se encuentra un obstáculo que no estuviese en el mapa. De producirse este evento, se generará una señal booleana que finalizará el bucle For que gestiona el desplazamiento, parando el robot.

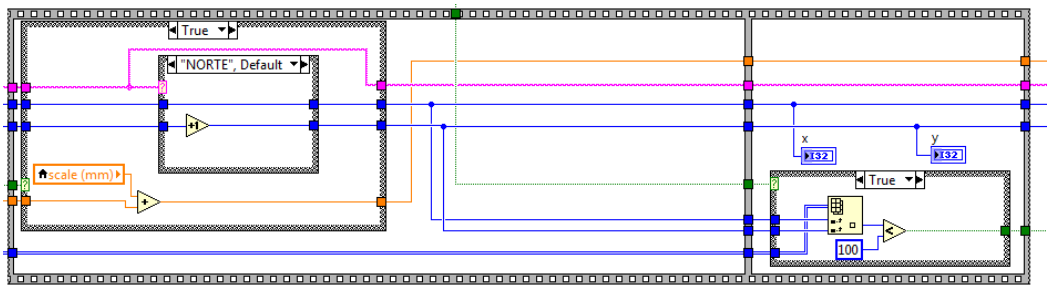

Figura 4.23. Actualización de la coordenada actual y detección de nuevo obstáculo en la ruta

De producirse esta parada debida a un nuevo obstáculo, la señal booleana también activará un Case en el siguiente segmento de código, el cual se encarga de actualizar el Map Array, fijando la casilla del obstáculo como “no atravesable”. Finalmente, se reanudará una nueva iteración del bucle propagando las variables actualizadas necesarias para calcular la nueva ruta.

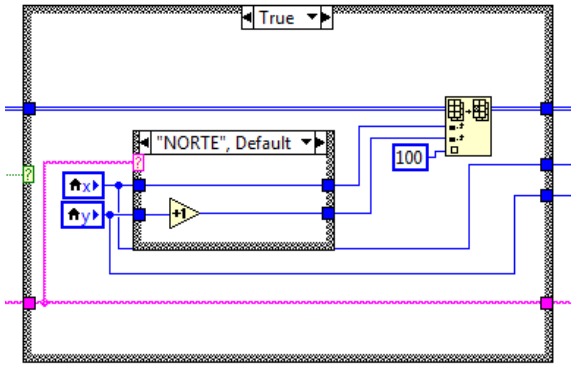

Figura 4.24. Actualización del mapa con el nuevo obstáculo

4.4.- Teleoperación

4.4.1.- Introducción

LabVIEW ofrece la posibilidad de comunicación entre VIs a través de protocolos como el TCP. En la práctica, esto se traduce en ser capaces de interactuar con un programa que se esté ejecutando en el momento de forma autónoma en el Starter Kit desde un VI secundario ejecutado en una máquina remota (un PC, en nuestro caso). Esto nos permite tanto solicitarle información, como transmitirle órdenes en tiempo real.

Con la configuración inicial de hardware, la única posibilidad para llevar a cabo esta comunicación pasa por tener un ordenador conectado físicamente al robot mediante un cable Ethernet, estableciendo un enlace físico entre sus tarjetas de red. Como puede deducirse, este montaje inicial resulta aparatoso y poco práctico.

Siguiendo la filosofía de configurar un robot autónomo, se optó por incluir un Router WiFi a bordo del Starter Kit, que se alimentase de la propia batería de éste. Conectándolo al puerto Ethernet del robot y configurando dicho Router como Punto de Acceso, bastaría con conectarnos a dicho PA desde una máquina remota para poder establecer una comunicación sin cables. El proceso seguido para lograrlo, junto al resto de asuntos relacionados con el hardware, se detalla en el capítulo 3.

4.4.2.- Descripción

Como muestra de las posibilidades que brinda comunicarse en tiempo real con el programa en ejecución del robot, y como ejemplo de la forma en la que se debe realizar, a continuación se expone un ejemplo de teleoperación.

El programa consta de dos partes: el VI que se ejecutará en el robot (Remote Receiver), y el VI que ejecutaremos en la máquina remota (Key Control). El primero deberá estar incluido como VI principal en un proyecto Robotics, mientras que el segundo se podrá ejecutar como un VI independiente en cualquier máquina remota que tenga instalada la misma distribución de LabVIEW Robotics.

El funcionamiento de Remote Receiver constituye una variación del ejemplo Starter Kit Roaming provisto por Robotics, donde se sustituyen los bloques correspondientes a la elección de dirección de desplazamiento por un control directo de los motores izquierdo y derecho a partir de los comandos introducidos por teclado en el terminal remoto mediante el VI Key Control, valíendose del protocol TCP para establecer la comunicación entre ambos programas.

En esencia, en este ejemplo el Starter Kit se convierte en un robot teledirigido mediante el teclado de un PC.

4.4.3.- Implementación: Remote Receiver

El Vi Remote Receiver consiste en un bucle While que se ejecutará de forma indefinida recogiendo e interpretando los datos provenientes de la conexión TCP en cada iteración para traducirlos en velocidad de cada motor, y por ende, en desplazamiento.

Se comienza con la inicialización estándar de la referencia a la FPGA y activando los motores, como puede observarse en la siguiente Figura.

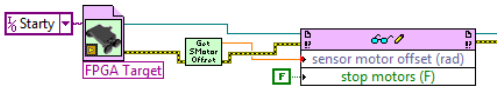

Figura 4.25. Inicialización del Remote Receiver

El cuerpo del programa lo constituye un bucle Timed Loop al que se introduce la referencia a la FPGA y a la conexión TCP, para lo cual se emplea la función “TCP Listener” (a la cual debeermos indicarle como entrada el puerto que se ha de escuchar).

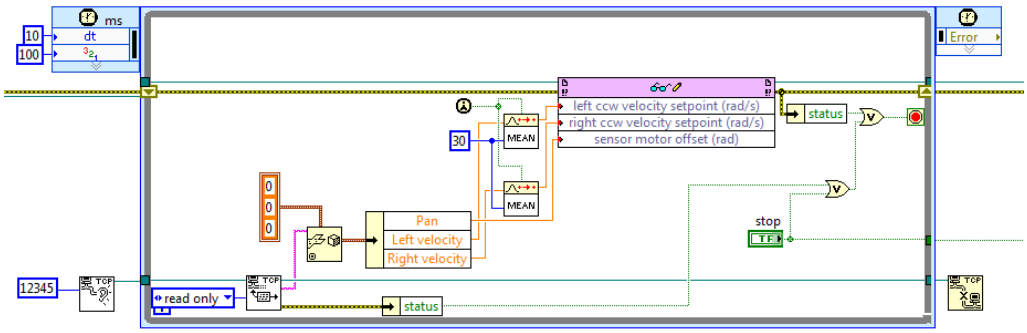

Figura 4.26. Remote Receiver – Bucle principal

Pasaremos la referencia de la conexión a la función TCP Send Receive, la cual es polimórfica, por lo que habrá que configurarla en estado de lectura. De esta forma se leerá la información procedente de la conexión, ofreciéndose en forma de String.

A continuación emplearemos la función Unflatten from String para convertir los datos del String en un cluster, del cual leeremos los datos correspondientes mediante un Unbundle By Name.

Finalmente, para suavizar la transición entre los comandos de velocidad, se han incluido un par de funciones Mean Point By Point, que tomarán las últimas 30 muestras y calcularán su media, siendo ésta la velocidad final que se aplique, en radianes por segundo, a cada motor.

Nótese que la estructura y composición del String que se recibe a través de la conexión TCP es conocida, puesto que se debe crear en el VI Key Control, el cual veremos a continuación.

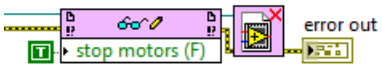

Figura 4.27. Remote Receiver – Cierre del programa

El código termina dando la orden de parada a los motores, y cerrando la referencia a la FPGA.

4.4.4.- Implementación: Key Control

El Vi Key Control consta de un bucle Timed Loop. De forma previa es necesario inicializar la comunicación con el teclado del equipo mediante la función Initialize Keyboard, la cual devuelve una referencia al mismo.

De la misma manera se inicializará la estructura del cluster de datos donde se encapsularan las órdenes que serán enviadas al robot, y se abrirá un canal de comunicación a través de la función TCP Open Connection, para lo cual debe indicarse la IP del Starter Kit y el puerto TCP por el que se realizará la comunicación.

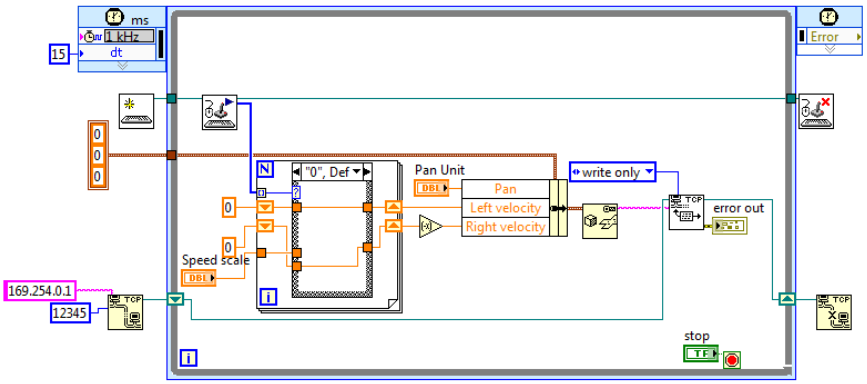

Figura 4.28. Key Control – Diagrama de bloques completo

El bucle iniciará cada iteración leyendo la información introducida por teclado mediante la función Acquire Input Data, la salida de la cual determinará el estado de una estructura Case. Cada estado corresponderá a una pulsación de una tecla, en este caso limitadas a:

- W/UP ARROW
- S/DOWN ARROW
- A/LEFT ARROW
- D/RIGHT ARROW

Cada uno de los estados registrará un valor de velocidad fijado por la variable Speed Scale a cada motor, variando el sentido de dicha velocidad en función del estado. Esto es, por ejemplo en el caso W/UP se fijará un valor positivo para la velocidad del motor izquierdo y uno negativo en el derecho, traduciéndose esto en que el robot se desplazará hacia delante en línea recta. El resto de estados son una variación de éste.

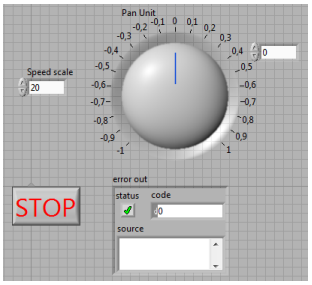

Figura 4.29. Key Control – Panel Frontal

El siguiente paso consiste en encapsular los datos de velocidades, más el de orientación del sensor Ping (el cual se introduce a través del panel principal con un controlador) empleando un Bundle By Name, teniendo como resultado un Cluster compuesto por los tres datos.

Finalmente, mediante un Flatten String convertiremos el Cluster en una cadena String para su envío a través de la función TCP Send Receive, esta vez configurada en modo escritura.

El programa finaliza cerrando la conexión TCP y la referencia al dispositivo de entrada (teclado en este caso) cuando se de la orden de parar el bucle.

4.4.5.- Puesta en marcha

Remote Receiver debe incluirse en un proyecto Robotics del cual se realizará una Build y se cargará directamente en el Starter Kit como VI principal, mientras que Key Control se ejecutará en una maquina remota.

Una vez que el robot esté listo, se debe comenzar activando el interruptor Master, con lo que proveeremos de corriente al Router que actúa como Punto de Acceso. A continuación conectaremos el PC a la red Wifi del PA (la cual se recomienda proteger con contraseña para evitar posibles interferencias externas), siguiendo el proceso explicado en el capítulo 3.

Finalmente, activaremos el interruptor Motors del Starter Kit, y ejecutaremos desde el PC Key Control. Mientras el VI Key Control esté ejecutándose podremos enviar órdenes al robot empleando el teclado, como se ha designado en el código.

CAPÍTULO 5

PRUEBAS DE CAMPO

5.1.- Introducción

En el presente capítulo se describirán los procedimientos realizados para verificar el funcionamiento de los algoritmos expuestos en el capítulo 4 al ejecutarse sobre el modelo Starter Kit en entornos reales.

5.2.- Algoritmo de navegación autónoma

Las pruebas se realizaron en la oficina de la nave donde se desarrolló el proyecto. El primer paso a realizar consistió en tomar las medidas del recinto para realizar un mapa, para a continuación trasladarlo al código del programa en forma de Map Array booleano.

Para llevar a cabo este último paso, será necesario abrir el Front Panel del VI principal del algoritmo dentro del proyecto Robotics. En dicho panel podemos encontrar una matriz en forma de cuadrículas. Éste es nuestro mapa inicial, el cual podremos modificar simplemente haciendo click sobre la casilla que queramos cambiar de estado. Las casillas en blanco serán interpretadas por el programa como atravesables, mientras que las casillas en gris serán consideradas no atravesables (paredes u obstáculos).

Se ha configurado la variable Escala a 0.5, es decir, cada cuadrícula de nuestro mapa representa un área de 50x50cm.

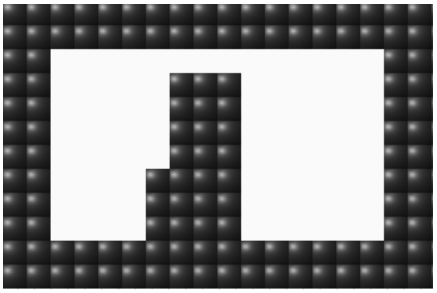

Figura 5.1. Mapa de la oficina

A la hora de fijar las coordenadas de inicio y fin en el programa, debe tenerse en cuenta que el Map Array identifica las coordenadas X e Y empezando por (0,0) desde la primera cuadrícula (abajo a la izquierda).

Para la prueba, se ha escogido un punto de partida en la zona izquierda del mapa, requiriendo al robot que trace una ruta al otro lado, esquivando la separación central de la habitación. En concreto, el robot partirá de la coordenada (3,3) en orientación NORTE, siendo la meta (13,3).

El primer paso que realiza nuestro programa es la Planificación de Ruta, la cual podemos realizar previamente desde un VI independiente en un PC (ya que es puro cálculo) para conocerla antes de que el robot la calcule y trace por sí mismo, pudiendo así comprobar que su desplazamiento se corresponde con el esperado.

Nótese que el A* encontrará siempre el mejor camino, arrojando el mismo resultado siempre que no varíe el mapa ni las condiciones inicial y final.

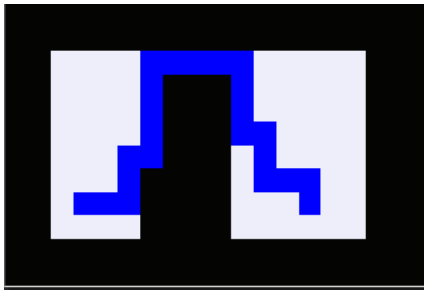

Figura 5.2. Mapa con la ruta calculada (en azul)

A continuación, el módulo Hoja de Ruta debe traducir dicha información en un conjunto de órdenes, como se ha visto en el capítulo anterior. Se recuerda que el tipo 0 corresponde a un avance en línea recta, el tipo 1 al giro, y el tipo 2 indica la llegada a meta.

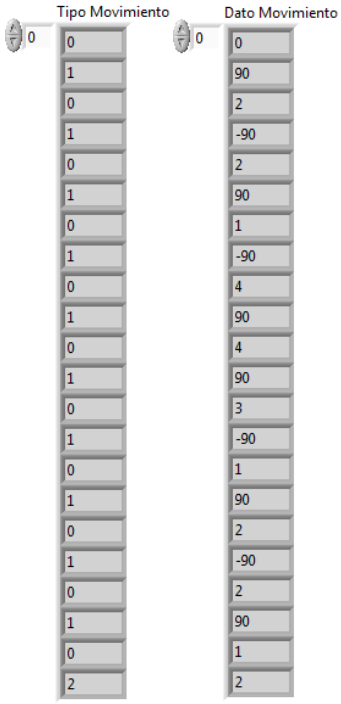

Figura 5.3. Hoja de Ruta calculada – Contenido de los arrays Tipo y Dato Movimiento

En cuanto al algoritmo de desplazamiento, opera con normalidad dentro de los límites esperados de funcionamiento. El robot partió de la coordenada fijada y trazó la ruta hasta llegar a la meta. Debido a la estimación de las distancias realiza por la odometría, y posibles errores de desviación mínimos introducidos por la brújula digital, el robot normalmente se para al considerar que ha llegado a la meta con un error de unos 10 cm en la horizontal y 6cm en la vertical de media.

Éste representa un error asumible teniendo en cuenta la distancia recorrida total, la cantidad de giros, y el hecho de que, aun con dicho error, el robot logra llegar a la coordenada meta definida por el cuadrado de 50x50.

5.3.- Algoritmo de navegación autónoma con evitación de obstáculos

Las pruebas de este algoritmo se han llevado a cabo de la misma manera que el anterior, pero en éste se ha dispuesto de un obstáculo no esperado en la ruta que conocíamos de antemano tomaría el robot.

En concreto, se ha dispuesto de un obstáculo en la coordenada (5,5), el cual al ser detectado origina la finalización del algoritmo de desplazamiento actual, saltando a la siguiente iteración del algoritmo principal, donde se recalcula una nueva ruta a partir de los datos actuales de posición y el mapa inicial actualizado con el obstáculo detectado en la anterior iteración.

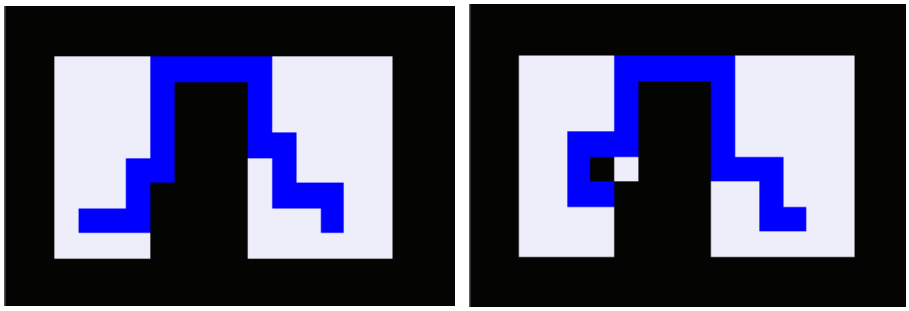

Figura 5.4. Ruta inicial (izquierda) y nueva ruta tras encontrar el obstáculo no esperado (derecha)

Aunque la nueva ruta es ligeramente distinta debido a que las condiciones iniciales han cambiado, sigue resultando un camino satisfactorio hasta la posición meta original, por lo que en ese aspecto cumple perfectamente con su cometido.

A la hora de traducir esta nueva ruta en desplazamiento, también se produce de forma satisfactoria, alcanzando la meta designada. Sin embargo, hay que tener en cuenta que los errores introducidos son proporcionales tanto a la distancia y número de giros totales realizados, como a la cantidad de cálculos de nuevas rutas necesarios. Por su propia naturaleza, la odometría tiene ciertos límites de fiabilidad en cuanto a distancia estimada, puesto que los errores son acumulativos.

5.4.- Teleoperación

Para la prueba de teleoperación se empleó el mismo escenario. Siguiendo los pasos descritos en el capítulo 4, se puso en marcha el Starter Kit y se inició la comunicación entre nuestro terminal remoto y el robot a través del Router.

Como era de esperar, el robot respondía a las indicaciones introducidas por teclado. Se comprobó que ejecutara los desplazamientos esperados para cada entrada, el control del servo del sensor de ultrasonidos, y el alcance de la señal.

En concreto, la teleoperación fue posible en la totalidad del entorno sobre el que se realizó la prueba, aunque es necesario hacer hincapié en que los límites de la comunicación estarán en el alcance de la propia señal Wifi.

CAPÍTULO 6

CONCLUSIONES Y FUTUROS PROYECTOS

6.1.- Conclusiones

En este capítulo final se enumeran las conclusiones y objetivos alcanzados en el desarrollo del proyecto final de carrera, relativas al entorno de programación empleado, los algoritmos desarrollados y el hardware sobre el que se ha trabajado.

- Se ha llevado a cabo un proceso de documentación y recopilación de información previo a la elaboración del PFC. Éste ha incluido el estudio de fundamentos de robótica (Introduction to AI Robotics, Murphy R.R), un repaso del entorno LabVIEW (llevado a cabo mediante los tutoriales y cursos ofrecidos en la página web de National Instruments), y la lectura de diversos artículos y tutoriales publicados en el Code Exchange de National Instruments con relación tanto a proyectos basados en el Starter Kit como a otras aplicaciones más generales en robótica.

- Se ha profundizado en los conceptos teóricos necesarios para la comprensión del hardware y software implicado, como son: la cinemática de un robot diferencial, el funcionamiento de un sensor de ultrasonidos, los algoritmos de planificación de ruta, la odometría, etc.

- Introducción la simulación en un entorno 3D del comportamiento de un robot real a través de la aplicación RobotSim, así como su integración con el entorno de desarrollo LabVIEW.

- Se ha estudiado el robot diferencial Robotics Starter Kit, familiarizándose con su estructura interna y funcionamiento. A su vez, se han propuesto y llevado a cabo ampliaciones en el hardware que le dotan de nuevas funcionalidades prácticas de cara al desarrollo de aplicaciones propias.

- Se han propuesto, implementado y testeado algoritmos de navegación desarrollados para el Starter Kit (aunque exportables a plataformas similares) que sacan partido de los sensores y encoders integrados en el mismo.

6.2.- Futuros proyectos

Como conclusión, a continuación se propondrán posibles futuros trabajos y aplicaciones derivados del presente proyecto.

6.2.1.- Curso práctico de iniciación a la robótica

El Robotics Starter Kit parte de la premisa de ofrecer una plataforma completamente funcional para su uso en el entorno académico. Como se ha expuesto en repetidas ocasiones a lo largo de este PFC, el entorno de desarrollo LabVIEW en combinación con el robot Starter Kit ofrece un banco de pruebas y experimentación con una curva de aprendizaje que facilita la rápida introducción en la materia, a la vez que ofrece tanto un hardware base ya montando, como la posibilidad de expandirlo sin excesivas complicaciones.

Todas estas características hacen del Starter Kit un excelente punto de partida para la enseñanza en el campo de la robótica, permitiendo establecer cursos de prácticas en los que en el transcurso de un cuatrimestre puede iniciarse al alumno en el manejo de LabVIEW Robotics y la programación de algoritmos básicos de desplazamiento.

Tal es el caso del curso que llevó a cabo en el 2011 la UNCC (Universidad de Carolina del Norte, EEUU), denominado Introduction to Robotics. En este curso, compuesto por 8 sesiones de prácticas, se introdujo al alumno al entorno Robotics (aunque exigía un mínimo de experiencia con LabVIEW), tratando temas como la instalación de nuevo hardware en el Starter Kit (acelerómetro), la odometría, y la programación correspondiente de la FPGA. El curso finalizó con la elaboración de un algoritmo de navegación estándar.

Un curso basado en el trabajo elaborado en este PFC podría disponer de una estructura como la propuesta a continuación:

1- Introducción a LabVIEW: Donde se repasarían las nociones básicas del entorno. Existen multitud de tutoriales, tanto oficiales de National Instruments como realizados por terceros, en los que se trata este tema.
2- LabVIEW Robotics: Introducción al módulo Robotics, nuevas funciones y metodología de trabajo basada en proyectos Robotics.
3- Robotics Starter Kit: Presentación del robot diferencial, descripción de su cinemática, listado del hardware básico que lo compone y ejecución del ejemplo Starter Kit Roaming provisto con Robotics para visualizar su funcionamiento.
4- Ampliación del Starter Kit: Donde se trataría como añadir hardware adicional al modelo y su correspondiente programación en la FPGA, partiendo del ejemplo de la brújula digital.
5- Desarrollo de un algoritmo de navegación autónoma: Para finalizar, se propondría al alumno una guía para que realizase un algoritmo similar al expuesto en el capítulo 4, que combinase todo lo aprendido hasta el momento.

6.2.2.- Mejora del hardware para implementar mapeo

En un principio el sensor de ultrasonidos PING incluido en el Starter Kit permite detectar obstáculos en el rango de 2 metros, aunque la fiabilidad de dichos sensores se pone en entredicho a la hora de poder realizar la función de mapear un área.

En estos casos se recomienda el uso de un sensor LIDAR (Light Detection And Ranging), el cual es capaz de detectar la distancia entre un emisor láser y un objeto o superficie en el camino del haz láser pulsado emitido. Su funcionamiento es similar al sensor de ultrasonidos detallado en el capítulo 3, puesto que mide el tiempo de retraso entre la emisión del pulso y la detección de la señal reflejada en el obstáculo.

El LIDAR constituye una de las soluciones más eficientes al problema del mapeo. Sin embargo, el elevado rango de precio de estas piezas de hardware llevó a descartar su inclusión en el presente PFC. Con el sensor de ultrasonidos es posible realizar tareas simples que constituyen en cierta medida un mapeo, como vimos en el segundo algoritmo del capítulo 3, pero para la generación de un mapa con exactitud, sería preciso añadir un LIDAR o hardware de similares características.

LabVIEW Robotics ofrece funciones específicas para el control de diversos modelos comerciales de LIDAR, por lo que su implementación sería casi inmediata, dependiendo únicamente de la propia instalación del hardware, y el desarrollo de algoritmos que le saquen partido al sensor. Esto se deja propuesto como una posible mejora para el Starter Kit.

BIBLIOGRAFÍA

Capítulo 1

[1] Murphy R.R, “Introduction to AI Robotics”, Massachusetts Institute of Technology, 2000

[2] http://www.ni.com/ (Web de la empresa National Instruments)

[3] “LabVIEW User Manual”, National Instruments, Abril 2003

[4] http://es.scribd.com/doc/65854017/Programacion-en-LabView-basica (Valery Moreno Vega y Adel Fernández Prieto, “Programación en Labview”, Instituto Superior Politécnico José Antonio Echeverría, Abril 2005)

Capítulo 2

[5] http://www.ni.com/white-paper/10483/en (“Getting Started with NI LabVIEW Robotics”)

[6] http://www.cogmation.com/betasite/robotSim%20documentation/robotSim%20User%20Manual%20Index.html (“RobotSim User Manual”)

[7] http://classes.engineering.wustl.edu/ese497/images/c/ca/LabVIEW_Robotics_Tutorial.pdf

(“LabVIEW for Robotics Starter Kit Tutorial”, Washington University in St. Louis)

[8] http://www.ni.com/white-paper/11564/en (“Overview of the LabVIEW Robotics Module”)

[9] G. Dudek, M. Jenkin, “Computational Principles of Mobile Robotics”, Cambridge University Press, 2000

Capítulo 3

[10] http://sine.ni.com/ds/app/doc/p/id/ds-217/lang/en (Ficha técnica del LabVIEW Robotics Starter Kit)

[11] Manual técnico del sensor de ultrasonido Parallax PING)))

[12] http://www.cs.brown.edu/people/tld/courses/cs148/02/sonar.html (“Ultrasonic Acoustic Sensing”, Brown Computer Science Department)

[13] http://www.pitsco.com/store/detail.aspx?ID=6016&bhcp=1 (NI Encoder Kit)

[14] http://www.robotroom.com/OpticalEncoder.html (“Digital Quadrature Optical Encoder”)

[15] http://mechatronics.mech.northwestern.edu/design_ref/sensors/encoders.html (“Optical Encoders”, Northwestern University Mechatronics)

[16] Manual técnico de la brújula digital CMPS09

[17] http://www05.abb.com/global/scot/scot260.nsf/veritydisplay/8bb7f20332ba74d5852575d20063da5a/$file/1sxu141184x0201.pdf (“Limit Switches - 101”)

[18] Ficha técnica del Interruptor V166-1C5

[19] https://decibel.ni.com/content/docs/DOC-12952 (“Robot Recipe: Teleoperation Mode for NI Robotics Starter Kit”)

Capítulo 4

[20] https://decibel.ni.com/content/docs/DOC-8983 ("An Introduction to A* Path Planning using LabVIEW")

[21] http://www.cs.utah.edu/~hal/courses/2009S_AI/Walkthrough/AstarSearch.pdf ("A* Search", University of Utah)

[22] Russell S., Norvig P., "Artificial Intelligence – a modern approach", PH, 2003

[23] http://rossum.sourceforge.net/papers/DiffSteer/ ("A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators", G.W. Lucas)

[24] J. Borenstein, H. R. Everett, L. Feng, "Where am I? - Systems and Methods for Mobile Robot Positioning", Marzo 1996)

Capítulo 6

[25] http://webpages.uncc.edu/~jmconrad/ECGR4161-2011-05/lab.html ("Introduction to Robotics", University of North Carolina)

[26] http://www.ni.com/white-paper/8157/en ("Robotics Fundamentals Series: LIDAR")

[27] http://www.ni.com/code/ (Página web del National Instruments Code Exchange)