UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

IMPLEMENTACIÓN DE ALGORITMO SLAM BASADO EN SENSOR LÁSER HOKUYO 04LX - UG01

AUTOR: Rafael Forcada Ferrer

TUTOR: Pedro García Gil

COTUTOR:

Curso Académico: 2013-14

A toda mi familia y amigos en general y a mis padres y hermanas en particular por haberme apoyado y animado en los momentos más duros.

A Pedro, tutor de este proyecto por su tiempo y dedicación, y a Luis por su inestimable ayuda.

Gracias.

ÍNDICE GENERAL

I. RESUMEN

Este proyecto tiene como objetivo dotar a un UAV (Unmanned Aerial Vehicle) quadrotor de un sistema de mapeo y localización simultánea SLAM (Silmultaneous Location and Mapping) basado en un LIDAR (Laser Imaging Detection and Ranging) del tipo Hokuyo-URG - 04LX. Capaz de mapear el entorno y dibujar la trayectoria del UAV. Se ha desarrolado un sistema capaz utilizar el entorno de trabajo ROS (Robot Operating System) con este fin de manera externa, es decir, sin necesidad de la instalación del *framework* en el hardware interno del robot.

Además se utilizarán los datos del entorno aportados por el LIDAR para realizar la implementación de un algoritmo de control de posición x-y mediante regulador PD que permita la detección y evasión de obstáculos por parte del UAV.

II. MEMORIA

1. INTRODUCCIÓN.

1.1. Objetivos.

Los objetivos de este proyecto pueden ser clasificados según sus distintas etapas, de la forma siguiente:

1.1.1. Implementación Láser en PC

En esta fase se ha logrado la conexión del sensor láser Hokuyo URG-04-LX-UG01 a un computador con SO Linux, y escribir un código en C++ que actúe a la manera de un driver y permita almacenar las medidas del láser. El objetivo de esta fase es conocer de manera experimental el funcionamiento del láser empleado en el proyecto, se ha conseguido tanto evaluar la calidad de los datos como conocer y trabajar con el formato de estos suministrados por el dispositivo, para la posterior redacción de un programa en C++ que se encargue de tratar los datos de manera óptima para ser utilizados por el software encargado del SLAM, así como por el control de posición x-y.

1.1.2. Programación

Redacción de un programa en lenguaje C++ que permita aprovechar de manera adecuada la información suministrada por el LIDAR en el algoritmo de control y en el SLAM.

1.1.3. Algoritmo de Control de Posición x-y.

Diseño de un algoritmo de control de posición x-y utilizando las lecturas del LIDAR. Se ha conseguido que el quadrotor detecte obstáculos y cambie de dirección, manteniendo como referencia una distancia programada de antemano.

1.1.4. SLAM

Implementación de un sistema SLAM a partir del sensor láser y librerías SLAM de ROS. Utilizando ROS y hector_slam, construir un mapa del entorno basándose en las medidas obtenidas con el láser. También se pretende que la trayectoria seguida por el sensor quede reflejada en dicho mapa.

1.1.5. Implementación en Microcomputadora de Placa Reducida

En esta fase, el código escrito para obtener los datos del láser y tratarlos será compilado para arquitectura ARM con el fin de implementarlo en una Raspberry Pi modelo B a la que irá conectada el Hokuyo-URG-04-LX-UG01, de forma que la Raspberry pi y su código sean totalmente independientes del paquete de software encargado del SLAM y no sea necesaria la instalación de ningún tipo de software en la tarjeta (aparte del ejecutable asociado al código encargado de recoger las medidas del láser, almacenarlas y enviarlas). Los datos se transmitirán por comunicación socket UDP, en una red inalámbrica Ad-Hoc en la que la Raspberry pi hará de servidor y el PC encargado de procesar los datos para el SLAM, de cliente. La comunicación socket UDP también deberá de ser programada en C++. El objetivo de esta fase es poder instalar la Raspberry pi y el LIDAR en un quadrotor y hacer SLAM, así como comprobar el funcionamiento del control reactivo en vuelo real.

1.2. Motivación.

El trabajo desarrollado a lo largo de este proyecto se encuentra englobado tanto dentro del campo de la robótica móvil y el de la programación informática industrial, como en el del control automático.

Cada uno de estos campos son mundos en constante y creciente desarrollo debido a la gran extensión de aplicaciones y posibilidades que presentan. Se pueden encontrar aplicaciones robóticas en numerosos ámbitos de la vida, desde el control de proceso y fabricación en una planta de montaje de la industria automovilística, hasta en la limpieza del hogar (robot roomba), pasando por las recientes y prometedoras aplicaciones de la robótica a campos tan cruciales como la medicina.

Por otra parte, y cada vez más, los constantes avances tecnológicos aumentan el abanico de posibilidades en el uso de vehículos aéreos no tripulados. Tales como operaciones de rescate y exploración, control de tráfico, retransmisión televisiva de grandes eventos, inspecciones en ingeniería civil, entre otras. Si bien es cierto que la actual legislación no permite el uso comercial y/o civil de vehículos aéreos no tripulados en España, expresamente prohibido por la Agencia Estatal de Seguridad Aérea (AESA), dependiente del Ministerio de Fomento; tal y como dicta la normativa publicada el 7 de Abril del presente año “No está permitido su uso para aplicaciones civiles. Es decir, no está permitido, y nunca lo ha estado, el uso de aeronaves pilotadas por control remoto con fines comerciales o profesionales, para realizar actividades consideradas trabajos aéreos” (AESA, 2014 [16]). Sin embargo, AESA ha anunciado que trabaja en la regulación específica de estas aeronaves y admite que está trabajando en colaboración con la industria para incluir disposiciones particulares para ellas, que sustituyan o complementen a las generales y hagan posible su vuelo con determinadas condiciones y limitaciones.

En cualquier caso, la proyección que se prevé en este sector, así como la posibilidad de uso de este tipo de vehículos automatizados en el ámbito militar junto con una posible revisión de la normativa al respecto en un futuro, hacen más que plausible y atractivo el desarrollo de trabajos y aplicaciones relacionadas con vehículos aéreos no tripulados en general y quadrotors en particular.

Combinando GPS y sensores inerciales, se han conseguido desarrollar UAV's que presentan una impresionante gama de actividades al aire libre. Pero sin embargo los entornos indoor y muchas partes de zonas exteriores continúan sin tener acceso a sistemas de posicionamiento externo tales como el GPS; es aquí donde entra la importancia del SLAM (Bechrach y otros, 2011. [1]).

Es por esto que en este proyecto se propone implementar un sistema de SLAM y evasión de obstáculos basado en sensor láser con el ánimo de mejorar las prestaciones de un UAV quadrotor.

Existen múltiples precedentes de estudios y proyectos basados en algoritmos de SLAM, sin ir más lejos el trabajo realizado por el Team Hector de la Technische Universität Darmstadt (Kohlbrecher y otros, 2013 [2]), utilizado en este proyecto. O el RANGE (Robust Autonomous Navigation in GPS-denied Environments) (Bechrach y otros, 2011. [1]), realizado por investigadores del Massachusetts Institute of Technology

Para la consecución gráfica del mapeo y la localización del UAV dentro de dicho mapa conviene utilizar una herramienta de software, primordialmente portable y flexible con el ánimo de que pueda ser utilizada en otras plataformas y con otros sensores láser. Para este proyecto se ha decidido utilizar ROS, en conjunción con el paquete hector_slam desarrollado por el Team Hector de la Technische Universität Darmstadt, en Alemania. Se barajó la posibilidad de utilizar MatLab2012 combinado con CAS-Toolbox (OpenSLAM), no obstante, los resultados, el consumo de recursos informáticos, el precio del software y el hecho de que la opción MatLab estuviese diseñada originalmente para robots terrestres fueron motivos suficientes para la elección de ROS.

2. PLATAFORMA PARA SLAM Y CONTROL DE POSICIÓN.

En este capítulo se describirán y explicarán los elementos tanto software como hardware, que se han utilizado para la realización del proyecto. En cuanto a hardware se van a describir la microcomputadora de placa reducida Raspberry pi modelo B y el sensor láser Hokuyo URG-04-LX-UG01. Y en cuanto al software se describirá el entorno de trabajo ROS (Robot Operating System), el paquete hector_slam, que implementa los algoritmos que llevan a cabo el SLAM, y el IDE Code::Blocks, donde se ha escrito y compilado la mayoría de código C++ usado en el proyecto así como el algoritmo de control de posición.

2.1. Software.

2.1.1. Robot Operating System (ROS).

ROS es un *framework* de desarrollo de robótica, y ofrece un conjunto de herramientas y librerías para el desarrollo de aplicaciones en este campo. A pesar de su nombre, no se trata tanto de un sistema operativo (funciona sobre Linux, Windows o iOS) como de un entorno de trabajo con una gran cantidad de repositorios que ofrecen paquetes de software de todo tipo para aplicaciones robóticas, además de ser multi-lenguaje (admite tanto C++ como python o java entre otros). ROS es OpenSource, es decir, es software completamente gratuito, al igual que todos sus paquetes (ros.org, 2014 [3]).

Hay varias distros o versiones de ROS, en este proyecto se utilizará ROS-groovy, por estar más testada que la actual ROS-hydro para la cual no se han actualizado todavía muchos de los repositorios. Además ROS-groovy ha sido utilizado satisfactoriamente sobre Raspbian (sistema operativo derivado de Linux-Debian instalado en la Raspberry pi modelo B a utilizar en este proyecto), aunque uno de los objetivos es conseguir que no sea necesario instalar ROS en la Raspberry, pues se pretende que únicamente sea usado en el PC para el mapeo y la localización gráficos.

Una de sus características principales es la abstracción de hardware, es decir sus repositorios no están dirigidos u orientados a tipos concretos de robots, sino que están encapsulados en interfaces estables y las diferencias quedan mantenidas en archivos de configuración. Todo esto proporciona portabilidad a las aplicaciones que se desarrollan con dicha plataforma (ros.org, 2014 [3]).

Otra de las características significativas de ROS es su mecanismo de comunicaciones, distribuido entre nodos del sistema. Un nodo es cualquier pieza de software del sistema (desde un algoritmo de SLAM, hasta un driver para el manejo de un motor). El objetivo de este método de comunicación es por una parte lograr la abstracción y reutilización de software y por otra parte adquirir independencia en cuanto a la localización del nodo. Los nodos se comunican entre ellos mediante mecanismo de paso de mensajes, del tipo

Publish/Suscribe. Mediante canales llamados topics, los nodos envían y reciben mensajes basados en estructuras de datos a otros nodos suscritos al mismo topic.

Para entender mejor el sistema de comunicación, y poder subsanar posibles errores durante el desarrollo de aplicaciones, ROS cuenta con múltiples herramientas como rqt_graph, que representa de manera gráfica el conjunto de nodos y topics en funcionamiento, así como sus relaciones.

Figura 2.1. rqt_graph (ROS Tutorials) (ros.org, 2014 [3]).

Por ejemplo, en la Figura 1, el nodo */teleop_turtle* (azul) es el publisher, se comunica con el nodo /*turtlesim* (verde), el suscriber; mediante el topic */turtle1/command_velocity* (rojo).

Otra de las ventajas de ROS es su flexibilidad en cuanto al grado de su uso; se podría implementar por completo el control del UAV, así como la sincronización de todos sus sensores utilizando este *framework* y sus paquetes de software. O por el contrario, como es el caso de este proyecto, utilizarlo únicamente en una aplicación externa desde PC sin tener que instalarlo en el hardware del robot. En este caso ROS será utilizado como interfaz de comunicación entre el comunicador socket que recibirá las lecturas del LIDAR enviadas desde la Raspberry pi y los distintos nodos del paquete hector_slam utilizados para la realización del SLAM.

2.1.2. Hector_Slam.

Hector_Slam es realmente un metapackage de ROS que contiene distintos nodos relacionados con el mapeo, el control de trayectoria, la navegación y localización. Tales como hector_compressed_map_transport, hector_geotiff, hector_geotiff_plugins, hector_imu_attitude_to_tf, hector_map_server, hector_map_tools, hector_mapping, hector_marker_drawing, hector_nav_msgs, hector_slam_launch, hector_trajectory_server. En este trabajo se usarán en concreto:

- Hector_mapping, el nodo encargado del SLAM propiamente dicho, utiliza las medidas del laser para realizar un mapa del entorno. Puede utilizar datos de odometría (de una IMU o de un encoder) para complementar al LIDAR, o puede realizar la odometría basándose únicamente en los datos aportados por el sensor láser, también puede ser usado en plataformas que tengan movimiento de roll o pitch por lo que es adecuado para quadrotors.

- Hector_geotiff, que guarda los mapas y la trayectoria del robot en formato geotiff (un estándar de metadatos que permite que información georreferenciada sea encajada en un archivo de imagen de formato TIFF que puede ser automáticamente posicionada en un sistema de referencia espacial).

- Hector_trajectory_server, que almacena la trayectoria del robot basada en transformaciones tf (una herramienta de ROS que permite el uso de multiples sistemas de referencia a lo largo del tiempo. Las relaciones entre los sistemas de referencia se mantienen en una estructura de árbol almacenada en buffer en el tiempo, y permite la transformación de puntos, vectores, etc… entre dos sistemas de referencia).

Hector_mapping está suscrito al topic /*scan* y recibe mensajes del tipo *sensor_msgs/LaserScan*. Por tanto se ha compilado un programa en C++ que recibe los datos enviados desde la Raspberry pi vía socket UDP, y hace de Publisher, enviando las medidas del LIDAR en forma de *sensor_msgs/LaserScan* al topic *scan* para que puedan ser recibidas por el nodo de SLAM. Hector_mapping se comunica mediante topics internos con hector_geotiff y hector_trajectory_server. Tal y como puede verse en la Figura 2.

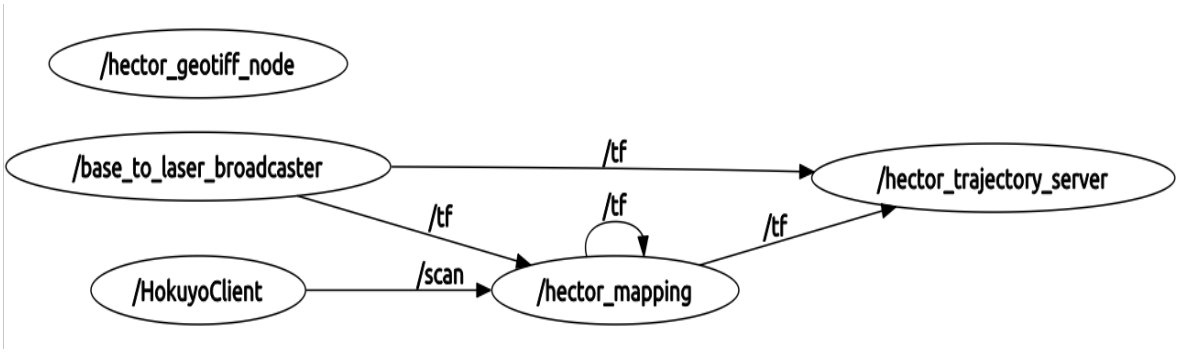

Figura 2.2 (Nota: hector_geotiff no se está comunicando ya que solo lo hará al salvar el mapa generado, cosa que no ocurría al realizar el grafo de nodos mediante rqt_graph)

Para plotear los mapas creados por estos nodos se utilizará rviz, una herramienta de visualización 3D de ROS.

2.1.3. Code::Blocks

Code::Blocks es un entorno de desarrollo integrado (IDE del inglés integrated development environment) multiplataforma para el desarrollo de programas en lenguaje C y C++. Es gratuito y está bajo licencia pública general GNU. Se basa en la plataforma de interfaces gráficas WxWidgets, por lo que puede utilizarse libremente en los principales sistemas operativos. Este IDE permite realizar *cross compiling* (o compilación cruzada) después de configurarlo debidamente y de instalar el compilador de la arquitectura para la que queramos compilar, algo esencial en este proyecto ya que se debe compilar tanto para arquitectura Intel en el PC como para arquitectura ARM en la Raspberry pi.

También permite la creación de proyectos con varias cabeceras y ficheros de código fuente. Utiliza el compilador gcc/g++ (mientras funcionemos en Linux y para arquitectura Intel) y el compilador arm-linux-gnueabi-gcc-4.7 para hacer compilación cruzada para ARM, aunque se podría haber utilizado otro siempre que se instalase y se configurase el IDE correctamente para su uso. Code::Blocks también dispone de debugger que en Linux por defecto es gdb.

2.2. Hardware

2.2.1. Hokuyo URG-04LX-UG01

Se trata de un pequeño sensor laser (50x50x70mm), y 160g de masa. Funciona con una conexión miniUSB/USB que sirve tanto para la alimentación del dispositivo como para la transferencia de datos recogidos por el sensor y el envío de comandos al microcontrolador interno.

Figura 2.3 Hokuyo URG-04LX-UG01

El laser tiene un rango de ±120º aprox. Rango que está dividido en 682 arcos o “steps”, como se puede ver en la figura 2.4, en cada uno de los cuales el dispositivo proporciona

una medida de la distancia entre el entorno y él; con un alcance máximo de 5600 mm. Realiza barridos completos a una frecuencia de 10 Hz

		URG-04LX
Step 0	First Measurement Point	0
Step A	Initial measurement Step of Detection Range	44
Step B	Sensor Front Step	384
Step C	End point of Detection Range	725
Step D	Last Measurement Point	768
E	Detection Range	239.77
F	Slit Division	1024

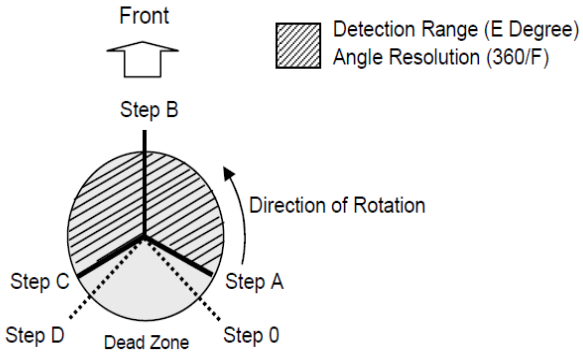

Figura 2.4 Rango de detección, ángulos y steps (Hokuyo Automatic CO., 2006 [4]).

El sensor es compatible con los protocolos de comunicación SCIP1.1 y SCIP2.0, por recomendación del fabricante se empleará el protocolo SCIP2.0. Concretamente bajo el comando MD (comando para la adquisición de datos de manera continua).

El sensor y su host intercambian datos usando un estándar de comandos predefinidos, con un formato específico, tal como indica las figuras 2.5

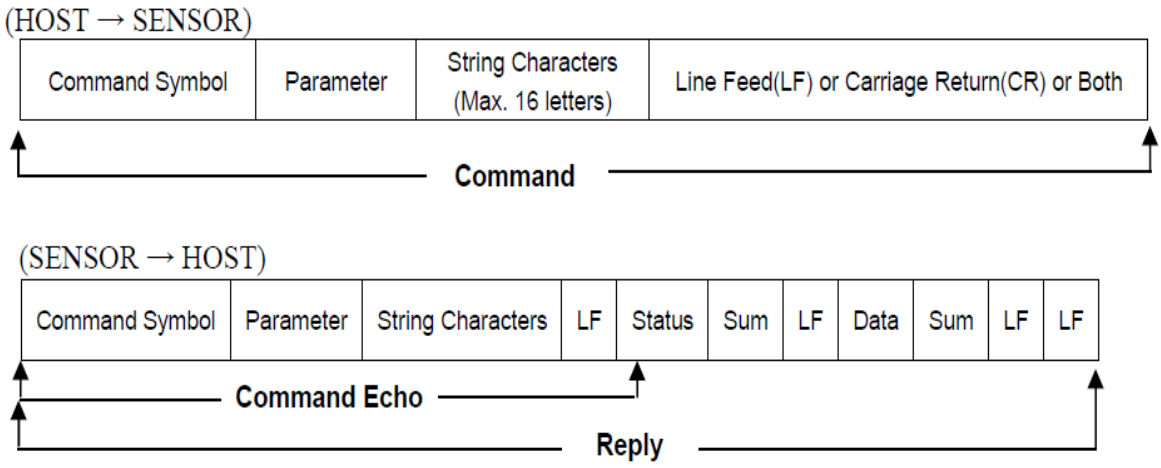

Figura 2.5 Formato de comunicación Host/Sensor (arriba) y Sensor/Host (abajo) (Hokuyo Automatic CO., 2006 [4]).

2.2.2. Raspberry pi Modelo B.

La Raspberry pi es una computadora de placa reducida de bajo consumo desarrollada en el Reino Unido. Cuenta con un "system-on-a-chip Broadcom" BCM2835 con una CPU del tipo ARM1176JZF-S a 700MHz, pero que puede funcionar, sin perder la garantía, hasta un máximo de 1GHz (Raspberrypi.org, 2014 [5]).

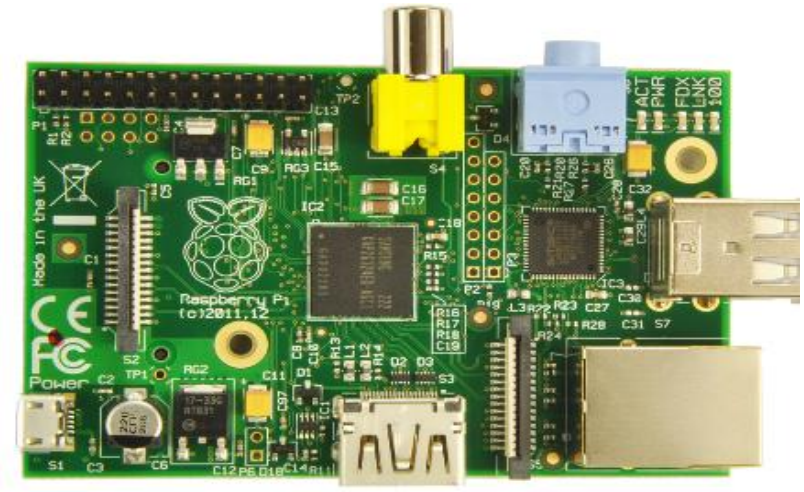

Figura 2.6 Raspberry Pi Model B

Tiene 512 MiB de memoria SDRAM. Aunque no tiene disco duro ni unidad de estado sólido, cuenta con ranura de lectura para tarjeta SD. Se utilizará un micro SD adapter con una tarjeta micro SD de 8GB, en la cual se instalará el sistema operativo Raspbian (derivado de Linux-Debian) y los ejecutables de los programas compilados para el proyecto.

Cuenta con un juego de instrucciones de tipo RISC de 32 bits, y al ser arquitectura ARM se trata de un sistema de los denominados middle-endian, que puede trabajar tanto con datos en formato big-endian como con datos en formato little-endian (El término "endiannes" designa el formato en el que se almacenan los datos de más de un byte, de izquierda a derecha o de derecha izquierda) (Aubrey y Kabir, 2008 [6]).

Dispone de una diversa serie de puertos Entrada/Salida, cuya distribución se puede ver en la Figura 2.7. Dos puertos USB 2.0 (vía hub integrado), un puerto HDMI, un conector RCA, una entrada de video MIPI CSI, un conector de 3.5mm para Jack Audio, Puerto 10/100 para cable Ethernet (RJ-45), y pines GPIO. No cuenta con fuente de alimentación propia, pero dispone de un puerto Micro USB para tal efecto. Se alimentará por dicho puerto con una fuente que proporcione 5 V y al menos 0,7 A, la que se ha utilizado en este trabajo proporciona 5 V y 2 A.

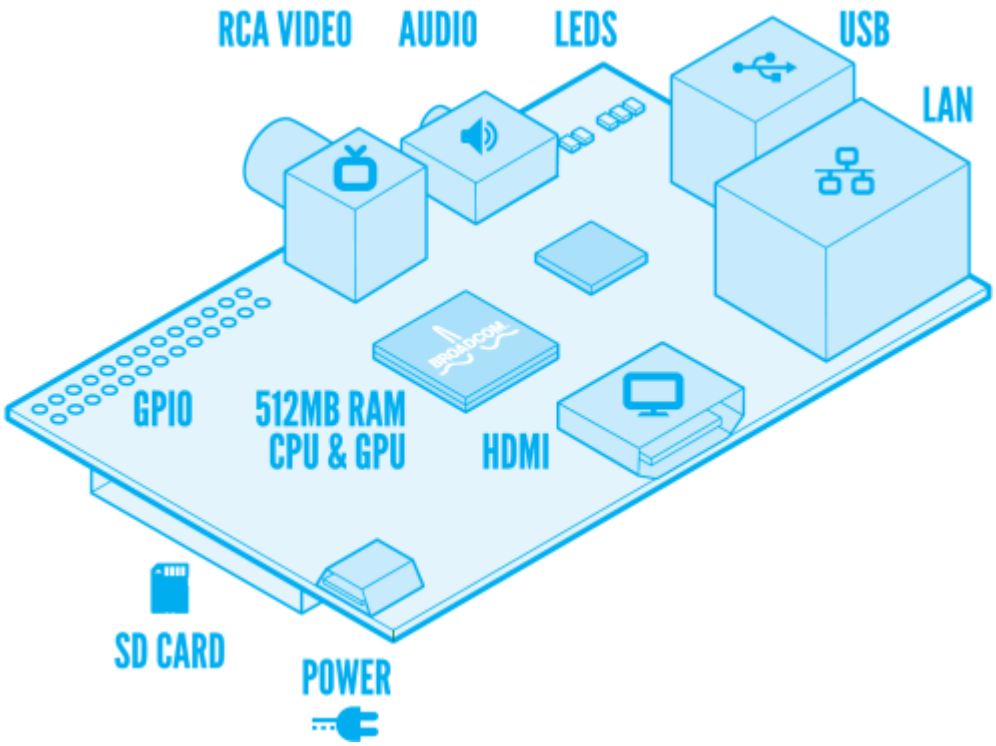

Figura 2.7 Distribución puertos E/S Raspberry Pi (Raspberrypi.org, 2014 [5]).

En este proyecto se utilizarán únicamente los dos puertos USB 2.0, para conectar el LIDAR y el dongle WiFi 802.11b/g/n Nano USB con antena externa, usado para la conexión y el envío de datos de manera inalámbrica. Así como el micro USB para alimentar tanto la tarjeta como el láser. También ha sido usado el puerto para cable Ethernet RJ-45 para configurar la Raspberry vía conexión ssh así como para realizar pruebas y simulaciones antes de haber establecido la comunicación inalámbrica.

3. ALGORITMOS SLAM, NAVEGACIÓN ABSOLUTA Y RELATIVA.

Aquí se explicarán los algoritmos en los que se basa el método SLAM así como el funcionamiento del SLAM sin odometría (o usando odometría láser) usado por hector_slam. Los cuales se manejarán a través de las funciones recogidas en las librerías de este paquete de software que implementan el funcionamiento de dichos algoritmos.

También se definirán los conceptos de posicionamiento local y global.

3.1. Sistemas de Posicionamiento Absoluto y Relativo.

En este apartado se van a explicar las diferencias entre los sistemas de referencia (o sistemas coordenados) absolutos y relativos.

Se puede definir un sistema de referencia con un sistema de coordenadas y un punto de referencia. Dependiendo de ese punto, tendremos un tipo de sistema de referencia u otro.

En aplicaciones de robótica móvil se considera que es un sistema de referencia local cuando se toma como punto de referencia un punto del robot. Estos sistemas de referencia se utilizan para la navegación mediante control reactivo. Si se toma como referencia un punto fijo externo al robot, se habla de un sistema de referencia global.

Para transformar un punto de un sistema de referencia a otro hay que seguir un procedimiento matemático que consiste en tomar coordenadas de ese punto y multiplicarlas por una matriz de rotación y sumarles una matriz de traslación.

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \cos\varphi & -sen\varphi \\ sen\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix} + \begin{pmatrix} Xo \\ Yo \end{pmatrix} \qquad (3.1)$$

Transformación de un punto de un sistema de referencia local a las correspondientes a un sistema de referencia global (Sanz, 2008 [7]).

Siendo:

(X, Y) coordenadas en el SR global del punto a transformar.

(X_o, Y_o) coordenadas en el SR global del centro del SR local.

(x', y') coordenadas en el SR local del punto que estamos transformando.

φ ángulo de diferencia entre los dos sistemas de referencia.

Dependiendo del tipo de sistema de referencia que se empleé para el posicionamiento del robot, podemos hablar de de navegación global o navegación local.

3.1.1. Navegación con Posicionamiento Relativo.

La utilización de un sistema de posicionamiento relativo da lugar a lo que se conoce como navegación local.

Como la navegación está basada en un sistema de posicionamiento relativo, las posiciones que toma el robot están referidas a su posición inicial. Normalmente se hallan las posiciones del robot relativas al inicio con las ecuaciones correspondientes a la odometría del robot, que tienen en cuenta las dimensiones y las características cinemáticas de este. Este sistema de odometría se basa en sumar incrementos de posición a la posición inicial del robot, lo que se conoce como *Dead-Reckoning*. El gran problema de este método es la acumulación de errores que lleva asociada en cuanto a la estima de posicionamiento del robot, creciente con el tiempo y con la distancia recorrida (Sanz, 2008 [7]).

Al trabajar sin un mapa del entorno se considera que este tipo de navegación es reactiva, es decir que el robot se desplaza de un punto a otro, utiliza sus sensores externos para detectar obstáculos, y sigue algún tipo de protocolo de actuación para evitarlos.

3.1.2. Navegación con Posicionamiento Absoluto.

La utilización de un sistema de posicionamiento absoluto da lugar a lo que se conoce como navegación global.

Este tipo de de navegación se basa en un sistema de posicionamiento absoluto, las posiciones que toma el robot a lo largo de sus movimientos están referidas a un punto externo a él.

Cuando no se dispone de un mapa del entorno, se está haciendo una navegación tipo SLAM. Es decir, se construye un mapa del entorno desconocido en el que se encuentra el robot, a la vez que mantiene información de su trayectoria dentro de dicho entorno. Este tipo de navegación es más robusta que la mencionada anteriormente, ya que los errores de posicionamiento asociados al *Dead-Reckoning* se van corrigiendo a través de un método Predictor-Corrector como puede ser un filtro de Kalman extendido, con los datos proporcionados por el sensor o sensores utilizados en la elaboración del mapa del entorno (West y Syrnos, 2006 [8]).

También es posible hacer navegación global con mapa, por ejemplo con un mapa construido previamente mediante SLAM. El método más popular, son los mapas métricos de celdillas. Las celdillas representan espacio libre o espacio ocupado de forma geométrica (Kohlbrecher y otros, 2013 [2]),

Trabajando con mapa, es necesario, además del mapa, un sistema de posicionamiento absoluto que es la posición absoluta del robot con el que se trabaja dentro de dicho mapa. Y un navegador local para que se encargue de la evasión de obstáculos.

3.2. El Proceso de SLAM

En este apartado se explicará a grandes rasgos en qué consiste un proceso de mapeo y localización simultanea empleando un filtro extendido de Kalman, sin pretender en momento alguno entrar en una explicación detallada y pormenorizada; para lo cual se referencia al lector al capítulo de bibliografía donde encontrará una serie de publicaciones y artículos especializados en el tema. A continuación se explicará de la misma forma el modo de funcionamiento del SLAM sin odometría (o con odometría láser) implementado en hector_slam y utilizado en este proyecto.

El proceso de SLAM consiste en una serie de pasos. Su objetivo principal es usar los datos del entorno proporcionados por un sensor (generalmente un LIDAR) para actualizar la posición del robot. Ya que, como hemos dicho, los datos aportados por sensores de odometría tales como IMUs o encoders van acumulando un error a la hora de estimar la posición a medida que avanza el tiempo y la distancia recorrida; no se puede confiar en exceso en este tipo de estima de la posición. Sin embargo se pueden usar los escaneos de un sensor láser para corregir dicho error de posición. Esto se consigue mediante la extracción de características del entorno y la re-observación cuando el robot se mueve a través de él. El responsable de la actualización de la estima de posición basada en dichas características del entorno es el Filtro de Kalman Extendido (EKF por sus siglas en inglés), estas características del entorno son generalmente llamadas *landmarks.* El EKF realiza un seguimiento de la incertidumbre en la estimación de posición del robot y en dichas landmarks (West y Syrnos, 2006 [8]).

Cuando los datos de odometría cambian debido al movimiento del robot, la incertidumbre perteneciente a la nueva posición es actualizada en el EKF, son entonces extraídas las landmarks del entorno desde la nueva posición. Entonces se intenta asociar estas landmarks a las observaciones de landmarks previamente hechas. Las que sean re-observadas son actualizadas y usadas para actualizar la nueva posición del robot en el EKF. Las que no hayan sido re-observadas son añadidas como nuevas, para que puedan re-observarse en pasos posteriores. Las figuras siguientes explican el proceso con más detalle en forma de diagrama.

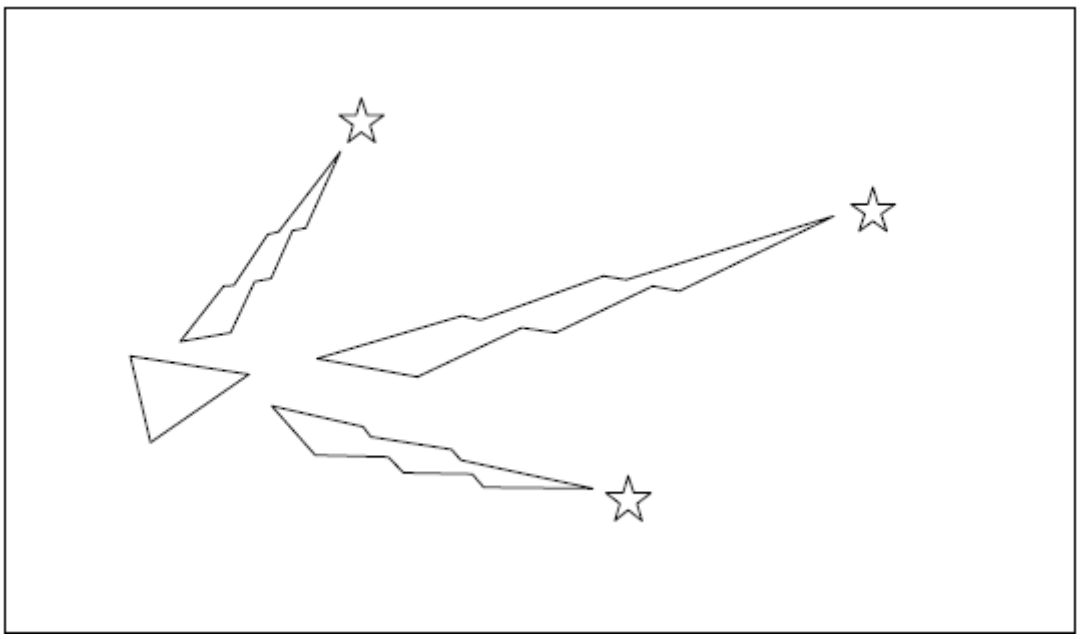

Figura 3.1 El robot está representado por el triángulo, y las estrellas representan landmarks. Las landmarks son detectadas inicialmente por el sensor láser.(las medidas del sensor vienen representadas por rayos) (Riisgaard y Rufus, 2005 [9]).

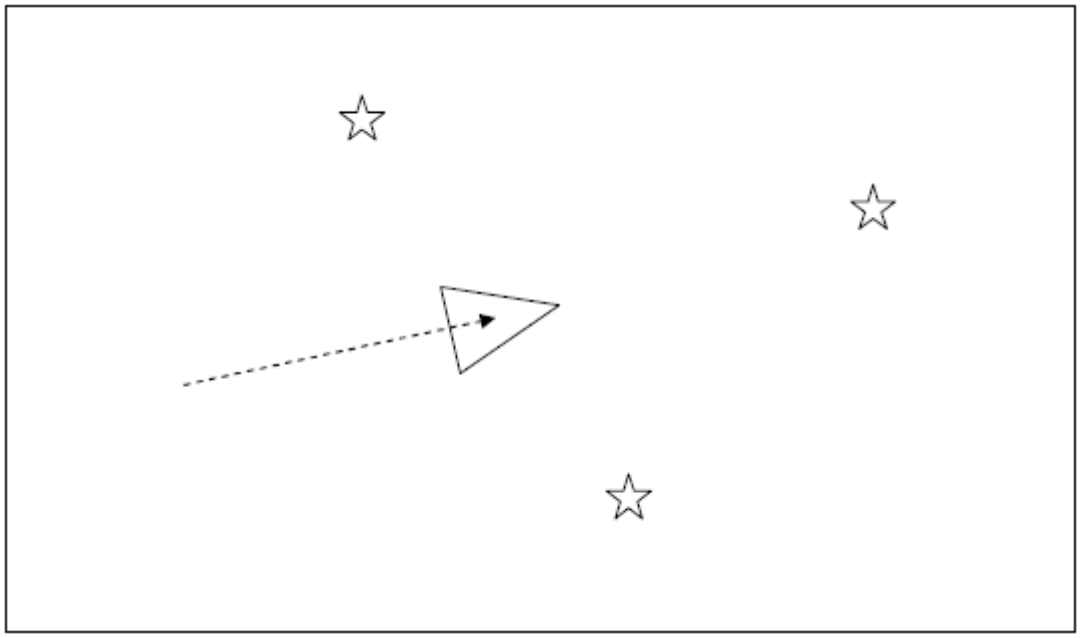

Figura 3.2 El robot se mueve, y según los datos aportados por los sensores de odometría su nueva posición estimada es la mostrada (Riisgaard y Rufus, 2005 [9]).

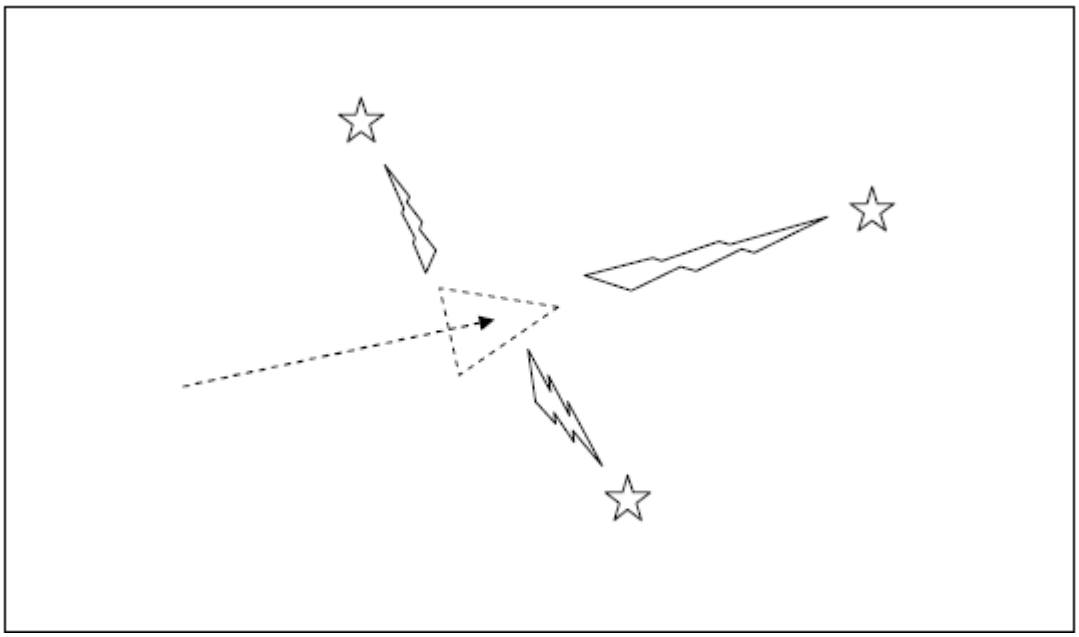

Figura 3.3 El robot mide de nuevo la localización de las landmarks usando el sensor láser. Pero no se encuentran donde deberían estar (dada la posición estimada según los sensores de odometría). La posición estimada es errónea (Riisgaard y Rufus, 2005 [9]).

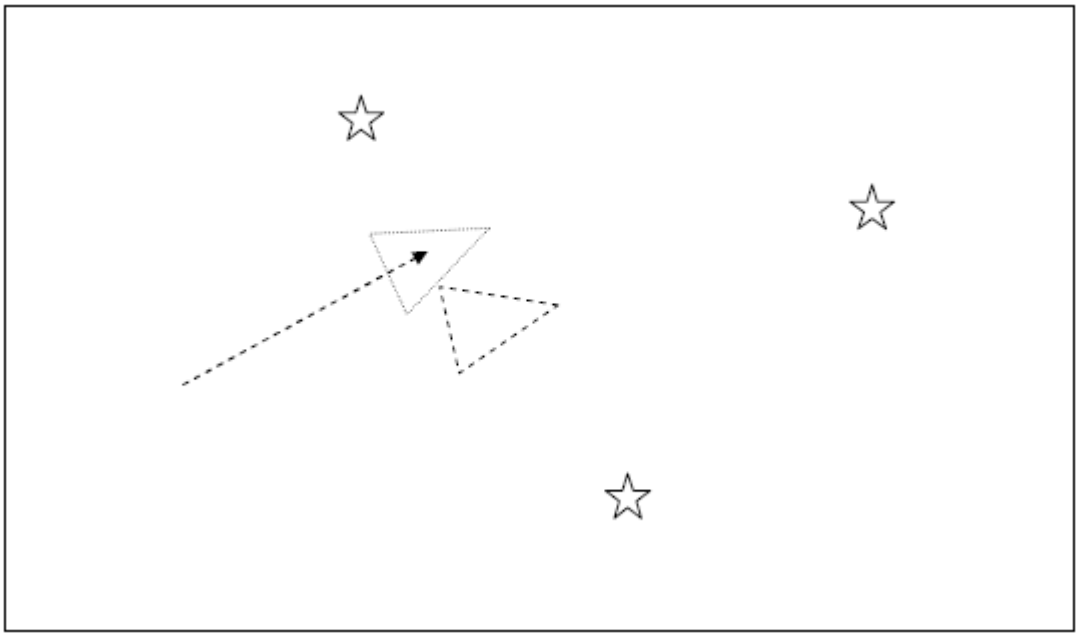

Figura 3.4 Considerando más fiables los datos aportados por el sensor láser, la nueva información acerca de la posición real de las landmarks es usada para corregir la posición estimada subsanando así una parte importante del error de posición acumulado. El triángulo a trazos representa la posición estimida inicial, y el punteado la posición estimada después de la corrección (Riisgaard y Rufus, 2005 [9]).

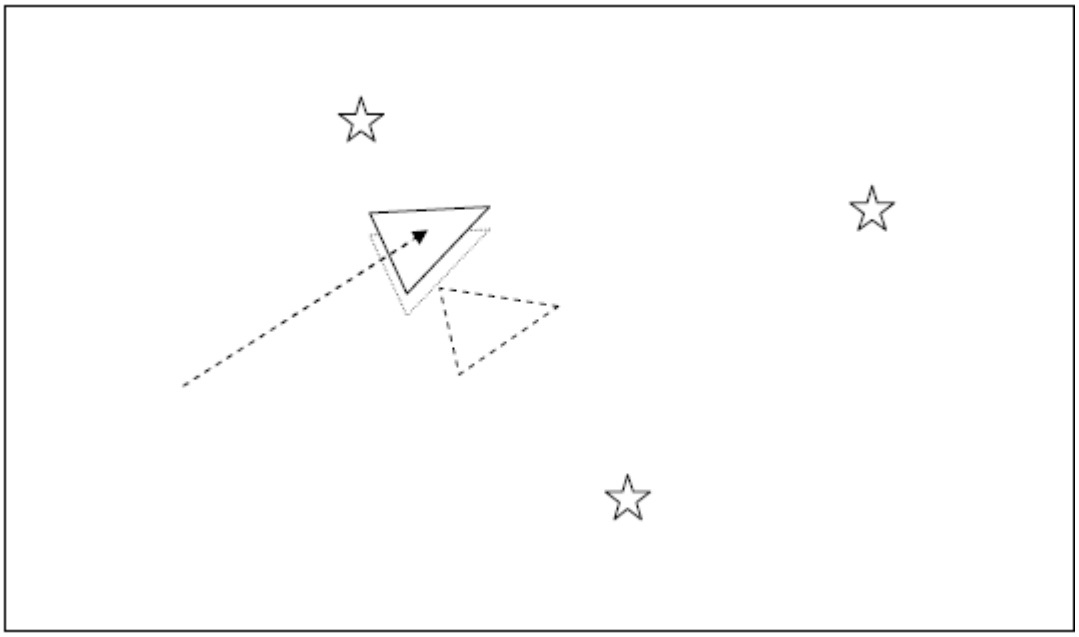

Figura 3.5 Realmente la posición estimada corregida (triangulo punteado) no se corresponde al 100% con la posición real (triángulo de trazado continuo). Los sensores no son perfectos y también introducen pequeños errores. No obstante la nueva estimación es mejor que la inicial utilizando únicamente odometría (Riisgaard y Rufus, 2005 [9]).

Además de esta ventajosa corrección de la posición estimada, las landmarks que hemos ido utilizando a lo largo del proceso sirven para realizar un mapa del entorno, que podrá ser utilizado tanto para plotearlo en una aplicación de mapeo por pantalla o para construir un mapa de celdillas de ocupación con un sistema de referencia global para ser empleado en navegaciones posteriores.

3.2.1. Las Landmarks.

Se hará una breve explicación acerca de en qué consisten las citadas landmarks y sus métodos más comunes de extracción.

Se trata de características del entorno que sean fácilmente re-observables desde distintas posiciones y ángulos. Las landmarks que utilice un robot en un proceso de SLAM dependerán, obviamente, del tipo de entorno en el que se encuentre. Deben ser lo suficientemente singulares para que puedan ser identificadas en distintos instantes de tiempo y distintas mediciones. Las landmarks que un robot debe reconocer deben ser las suficientes para que el robot no esté un periodo de tiempo extendido sin suficientes referencias y "se pierda". Deben ser fijas para poder ser tomadas como referencia.

Las dos formas más comunes de extracción de landmarks en entornos indoor son las denominadas spike landmarks (landmarks punta) y RANSAC (Random Sampling Consensus).

Las spike landmarks encuentran valores en el barrido del láser que difieran entre sí en más de cierta cantidad; esto suele reflejar, por ejemplo cuando una parte del barrido detecta

un muro o una pared con una abertura. O cuando detecta una esquina entre dos paredes (Riisgaard y Rufus, 2005 [9]).

El Random Sampling Consensus es un método que se basa en extraer líneas rectas del scaneo del láser. Estas líneas pueden ser usadas como landmarks. Usa una aproximación por mínimos cuadrados en muestras de la medida del laser encontrando la mejor recta que aproxima dichas medidas, una vez hecho esto se comprueba cuantos del escaneos del laser casan con esta mejor recta, si este número supera cierta cantidad se puede asumir con seguridad que hemos visto una recta (que suele coincidir con un segmento de pared) (Riisgaard y Rufus, 2005 [9]).

Por su puesto, para que algo sea tomado como landmark debe de ser observado primero un cierto número de veces para reducir el número de errores y falsas landmarks.

3.3. SLAM sin Odometría.

Sin embargo, los datos de odometría no son indispensables para llevar a cabo labores de mapeo y localización simultanea. Uno de los objetivos del proyecto es lograr llevar a cabo el SLAM con independencia de la IMU, encargada de la odometría del quadrotor. Es por esto, entre otras razones, que se ha elegido el paquete de software hector_slam, que ofrece esta posibilidad.

El SLAM sin odometría se basa únicamente en las medidas proporcionadas por el sensor láser, es por esto que la frecuencia de escaneo de dicho sensor es relevante en grado sumo. En este caso, las landmarks se extraen de la misma forma que en un algoritmo de SLAM clásico, tal y como se ha explicado anteriormente. La principal diferencia radica en la forma de actualizar la posición estimada del robot, lo cual se hará directamente con la reobservación de landmarks; de ahí la importancia de la frecuencia de las medidas y de disponer de un algoritmo de extracción de landmarks robusto como el implementado en hector_slam.

Las siguientes figuras muestran en forma de diagrama el proceso de actualización de la posición estimada usado en el SLAM sin odometría.

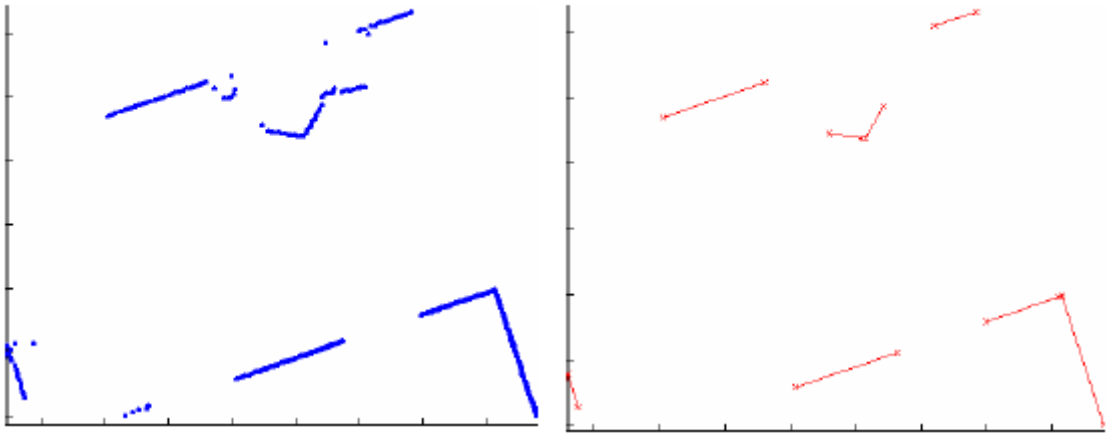

Figura 3.6 Suponiendo que el sensor láser proporciona las medidas mostradas (azul), las apropiadas son transformadas en landmarks, en forma de segmentos (rojo) por el algoritmo de extracción (Neira, 2007 [10]).

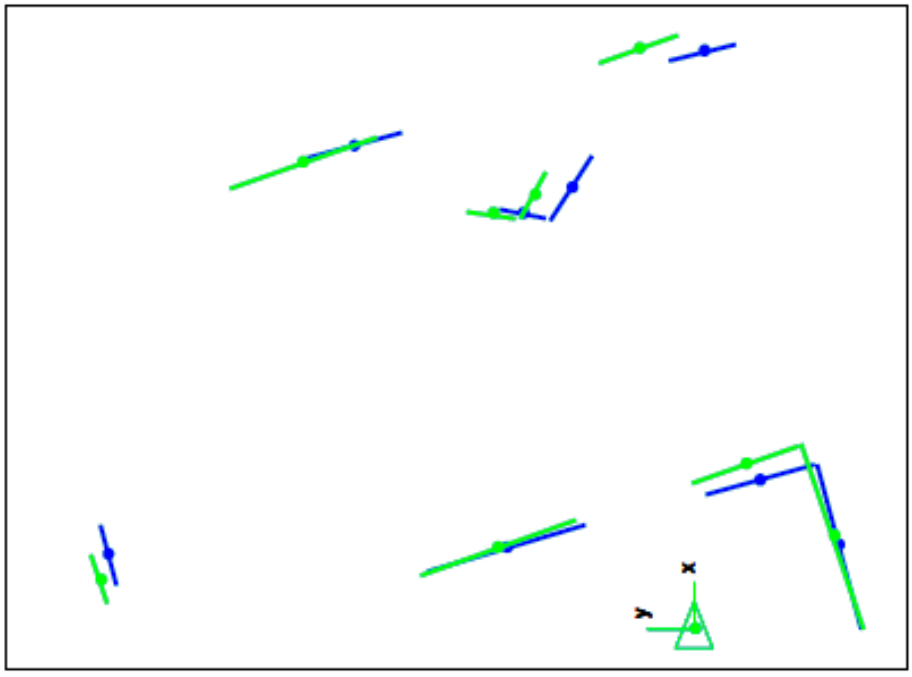

Figura 3.7 El robot efectúa cierto movimiento, y el sensor láser proporciona unas nuevas medidas del entorno de las que se extraen landmarks (verde), las cuales se solapan con las extraídas anteriormente (azul) (Neira, 2007 [10]).

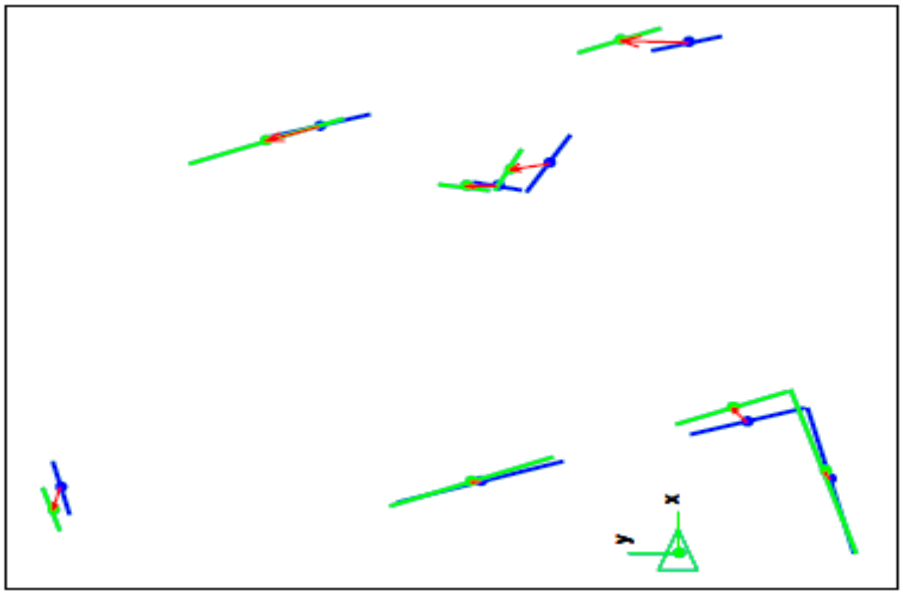

Figura 3.8 Las landmarks son caracterizadas mediante puntos significativos, como pueden ser puntos medios, extremos o ángulos. De esta forma las landmarks antiguas y las nuevas son relacionadas mediante vectores de desplazamiento y marcadas como re-observadas (las que lo sean) (Neira, 2007 [10]).

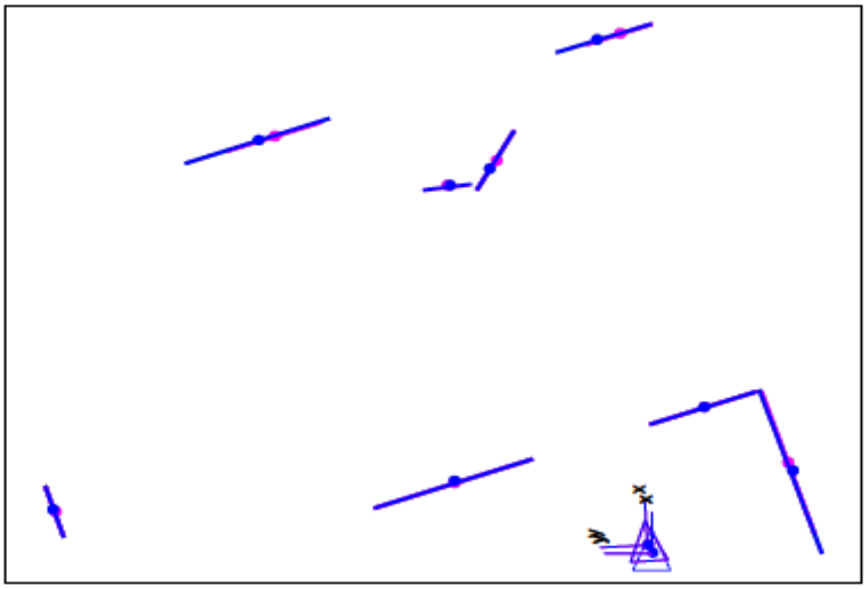

Figura 3.9 Los vectores que representan el cambio de posición de las landmarks son empleados para calcular la nueva posición estimada del robot. El error derivado de este método y sus cálculos, se reduce de manera significativa con el aumento de la frecuencia de scaneo del láser, y aumenta con la velocidad de desplazamiento del robot (Neira, 2007 [10]).

4. IMPLEMENTACIÓN SISTEMA SLAM.

En este apartado se procederá a explicar la implementación que se ha llevado a cabo en este proyecto. Es decir, puesta en funcionamiento del sensor láser, redacción de código en C++ para almacenar las medidas, configuración de la Raspberry pi, implementación del algoritmo de SLAM, y configuración de la conexión inalámbrica y comunicación socket UDP.

Antes de empezar se ha llevado a cabo la descarga e instalación de ROS – groovy y hector_slam en el PC base, mediante la terminal de comandos, tal y cómo se describe en (ros.org, 2014 [3]).

La idea es tomar los datos del láser con la raspberry pi, que no tiene instalado ningún tipo de software (aparte de su SO y el ejecutable de ProgHok, escrito y compilado por el proyectista). Y enviar dichos datos vía WiFi a PC base, en el cual está instalado ROS y el paquete de SLAM, donde serán tratados de forma conveniente por el programa HokuyoClient, para que puedan ser enviados y usados por el nodo de SLAM. En la siguiente figura se puede ver el funcionamiento del sistema implementado.

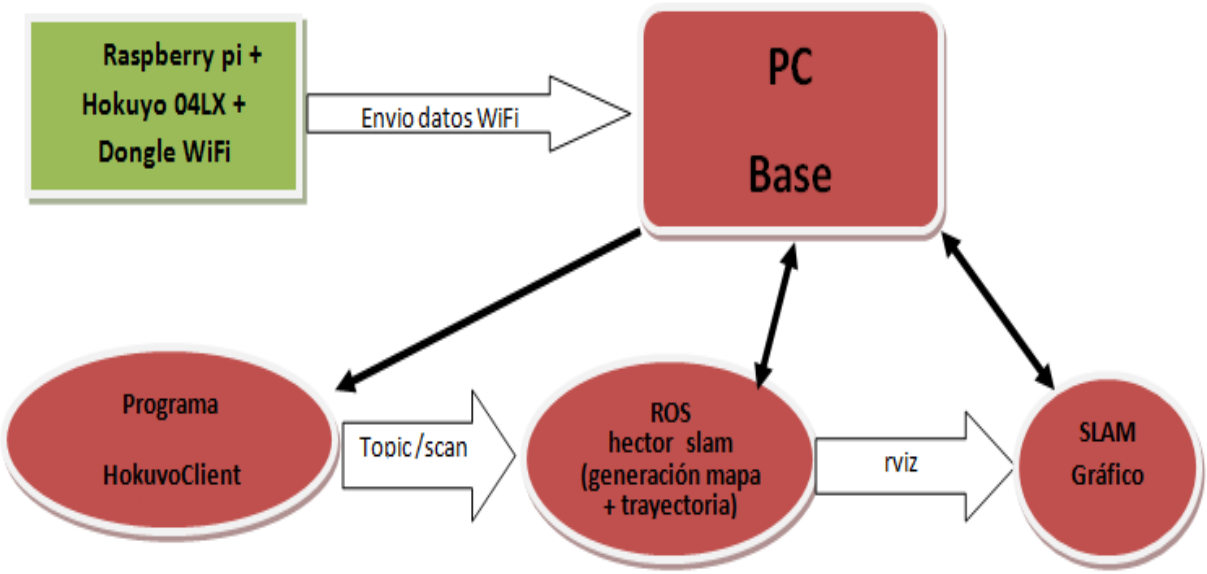

Figura 4.1

4.1. Drivers Para Hokuyo URG-04LX-UG01.

El fabricante del sensor láser proporciona dos programas para visualizar las medidas del LIDAR tanto gráficamente mediante una interfaz, como numéricamente en una hoja de cálculo Excel. No obstante estos programas únicamente funcionan sobre Windows, por lo que se hace necesario la redacción de un código en C++ que haga la función de driver para el sensor sobre Linux, que es el sistema operativo en el que se ha decidió trabajar en este proyecto.

El fabricante, *Hokuyo Automatic CO. LTD.*, proporciona una serie de librerías y funciones en C++, las cuales se han tomado como ejemplo. Aunque para programar los drivers se han usado funciones contenidas en el proyecto MK1_Roboard, cuyo autor es la empresa *Robotnik Automation S.L.L.* con sede en Valencia, después de introducir los cambios necesarios para que el funcionamiento se adecuase al de este trabajo.

Los drivers están escritos según lo que se conoce como programación orientada a objetos, que difiere en muchos aspectos de la programación estructurada en tareas que se enseña en la escuela.

El funcionamiento de los drivers, llamados *urg_laser*, es el siguiente.

Primero debe construirse un objeto de la clase urg_laser, para poder disponer de las variables y las funciones definidas para dicha clase, utilizando el constructor *urg_laser();* dicho constructor inicializa una serie de variables además de asignar el nombre del puerto USB que va a ser utilizado, y configura la variable booleana *UseSerial* a 0 (esta variable está pensada para un modelo del Hokuyo 04LX que puede usar puerto RS-232).

Posteriormente se debe invocar la función *Setup()*, que se encargará de abrir el puerto USB en el que está conectado el láser mediante la función *Open(const char *PortName, int use_serial, int baud)*, esta función abrirá el puerto en modo "r+" (Linux trata a los puertos USB como ficheros en los que se puede escribir y de los que se puede leer) y si ha habido algún problema mostrará un mensaje de error al abrir el puerto; después activará el protocolo de comunicación SCIP2.0 (recordar que el láser podría funcionar también con SCIP1.1[4]) y comprobará su correcto funcionamiento mediante la función *QuerySCIPVersion()*; y configurará los parámetros asociados al láser, correspondientes a nuestro modelo, tales como los ángulos de medida máximo y mínimo, el alcance máximo y mínimo, los tamaños de los vectores para almacenamiento de datos, etc... Y los almacenará en una estructura de datos, mediante la función *GetSensorConfig(laser_config_t *cfg)*.

Después se debe invocar la función *ReadforSLAM()*, escrita especialmente para este trabajo o a *ReadLaser()* (que almacena las medidas correspondientes al eje X, eje Y positivo y eje Y negativo). La función *ReadforSLAM()* invoca a *GetReadings(urg_laser_readings_t *readings, int min_i, int max_i)* que envía al LIDAR el comando MD (obtención continua de medidas) , y extrae los datos de distancia en mm de la respuesta enviada por el sensor; estos datos son almacenados en un vector de tipo float

de la estructura *readings*, de índole privada, teniendo en cuenta la configuración de variables almacenada anteriormente en *cfg* por la función *GetSensorConfig()*. La función *ReadforSLAM()*, toma este vector y lo pasa al vector float *SLAM_Data[]* de índole pública, después de haber pasado los datos de milímetros a metros. De esta forma ya se dispone de un vector que contiene los datos del entorno captados con el laser en un barrido, en metros.

4.2. Programa ProgHok.

Este es el programa que será ejecutado en la Raspberry pi para captar los datos del láser y enviarlos mediante comunicación socket UDP de manera inalámbrica a PC.

Los drivers anteriormente descritos se utilizarán en un programa compilado especialmente para este trabajo, llamado ProgHok, que introduce una clase nueva del mismo nombre cuyo constructor se encarga de la creación del objeto de la clase urg_laser y del set up de dicho objeto.

Dispone además de una serie de funciones que permiten utilizar el laser para el algoritmo de SLAM, o, tomando un haz de medidas de aproximadamente 85º en cada eje (eje X, eje Y positivo y eje Y negativo) escoger la medida menor de cada haz de medidas, que será utilizada en el control de posición x-y. También se han implementado funciones para escribir en fichero los datos aportados por el láser con vistas a usarlos si finalmente se escogía la opción MatLab/CAS-Toolbox.

Este programa se ha compilado para arquitectura ARM y se ha transferido a la Raspberry pi. El cross compiling se ha llevado a cabo usando Code::Blocks, y en forma de proyecto/console application; cuenta con tres ficheros de código fuente: *main.cc, ProgHok.cc, urg_laser.cc*; y dos ficheros de cabecera, *ProgHok.h* y *urg_laser.h*. (Todos los ficheros se pueden consultar en el Anexo de este trabajo).

La función main, establece el servidor socket que se usará en la comunicación inalámbrica, bajo el identificador "fd_socket" y asociado al puerto 15555. Después recibe un dato sin importancia (un entero) del cliente socket, con la intención de guardar su dirección (ya que se trata de una IP dinámica desconocida, mientras que la del servidor será estática y conocida de antemano) [11].

Una vez conocida la dirección del cliente, crea el objeto de la clase ProgHok y después efectuará una pregunta por pantalla acerca de la opción de uso del láser, para SLAM o para medidas en eje. Según la opción escogida, el programa entrará en un bucle en el cual ejecuta las funciones de lectura en cada iteración. Usando *ReadforSLAM()* a través de *measuresforSLAM()* si ha sido escogida la opción de SLAM, o usando *ReadLaser()* a través de *measures()* si se ha escogido la opción de medir por ejes; en este caso también se ejecutarán las funciones que darán como resultado la medida mayor y la menor en los tres ejes para cada escaneo. En el caso del SLAM se envía el vector de medidas a través del

socket UDP en cada iteración del bucle. El concreto funcionamiento del programa puede verse en el diagrama de la figura 4.2 en la página siguiente.

4.3. Cliente Socket y Publisher Láser.

Se trata de otro programa escrito expresamente para este trabajo. Este programa, llamado HokuyoClient, tiene como objetivo hacer de cliente en la comunicación socket, establecer conexión con *ProgHok* usando el descriptor "fd_socket", recibir el vector de medidas del láser y publicarlo en el topic */scan*, al que está suscrito el nodo hector_mapping del paquete de software hector_slam. Necesitamos enviar un mensaje del tipo *sensor_msg/LaserScan*, al topic */scan*, ya que es el mensaje que necesita hector_mapping para funcionar. Un mensaje del tipo *sensor_msg/LaserScan*, no es más que una estructura de datos definida de antemano en ROS y que tiene la siguiente forma:

- *Header header* una subestructura formada por
    - *unsigned int seq*, variable secuencial incrementable.
    - *time stamp* variable temporal que quedará definida mediante la función de ROS ros::time::now()
    - *string frame_id*, señala el tipo de referencia, se le asignará "laser".
- *float angle_min*, ángulo en el que comienza a medir el láser. Medido partiendo del eje X y en sentido anti horario.(radianes)
- *float angle_max*, ángulo en el que termina de medir el sensor láser.(radianes)
- *float angle_increment*, distancia angular entre medidas.(radianes)
- *float time_increment*, tiempo entre medidas consecutivas. (segundos)
- *float scan_time*, tiempo que tarda en completarse un barrido completo. (segundos)
- *float range_min*, alcance mínimo de medida. (metros)
- *float range_max*, alcance máximo de medida. (metros)
- *float ranges[]*, vector de medidas captadas por el láser, los valores por encima de *range_max* y por debajo de *range_min* deben ser descartados.

El programa comienza estableciendo el socket, como cliente, y bajo el descriptor “fd_socket”, asociado al puerto 15555 y a la dirección IP “10.0.0.1” que es la dirección IP estática asignada al servidor Ad-Hoc generado en el arranque de la Raspberry pi. Al tener una dirección conocida, no hay ningún problema en mandar un dato arbitrario (un simple entero) al servidor socket en ProgHok; de esta manera se da a conocer la dirección

Figura 4.2 Diagrama de funcionamiento de ProgHok.

completa del cliente al servidor, incluida su IP, que hasta entonces es desconocida, pues se trata de una IP dinámica asignada por el servidor dhcp asociado a la conexión Ad-Hoc generada por la Raspberry pi.

Una vez establecida la conexión, se prepara para ejercer de Publisher en el topic */scan*. Para ello se inicializa el programa en ROS y le asignamos un nombre y un identificador.

Es necesario enviar un mensaje del tipo *sensor_msg/LaserScan*, por tanto se declara que el programa va a mandar un mensaje de ese tipo, al topic mencionado. Se selecciona la frecuencia con la cual se quiere enviar el mensaje al topic mediante la función *ros::Rate loop_rate()*, se han seleccionado 10 Hz pues es la frecuencia de funcionamiento del sensor láser. Después se le asigna un nombre al mensaje que se va a enviar, *msg* en este caso y se entra en el bucle que enviará un mensaje en cada iteración. Dentro del bucle, se asignan los valores estáticos del mensaje, es decir, todos menos la variable secuencial *unsigned int seq* y *float ranges []*, para rellenar el vector de medidas se hará usando los datos recibidos por el socket UDP. Existe el problema de que la variable *float ranges[]* no está definida con un tamaño de vector dentro de la estructura de datos de *sensor_msg/LaserScan*; por lo tanto para no cometer un error de segmentación al ejecutar el programa se deberá rellenar este vector como uno de la clase *std::vector*, (Alonso y otros, 1998 [12]) en lugar de cómo un vector clásico con tamaño predefinido, es decir usando *msg.ranges.push_back((float)buff[h])*, en un bucle de numero de iteraciones igual al tamaño del vector buff, que es el vector que contiene las medidas del láser enviadas por el socket.

Una vez el mensaje está completo, se publica en el topic */scan*, se incrementa en uno la variable secuencial y se pasa a la siguiente iteración del bucle principal.

El funcionamiento se puede ver en el diagrama de la figura 4.4.

Puesto que este programa funciona utilizando algunas de las librerías de ROS y tiene que ser capaz de comunicarse con sus topics, ha tenido que ser compilado en un catkin workspace, (espacio de trabajo construido para trabajar con ROS y linkado con sus librerías), y compilado mediante la herramienta catkin_make, desde la terminal de comandos de Linux. Para llevar a cabo esta labor es necesaria la redacción de una CMake file, cuya redacción está explicada en el Manual del Usuario de este trabajo, y se puede ver en el Anexo.

En la figura 4.3 puede verse como HokuyoClient, mediante el topic */scan*, se comunica con los nodos de hector_slam, encargados del SLAM.

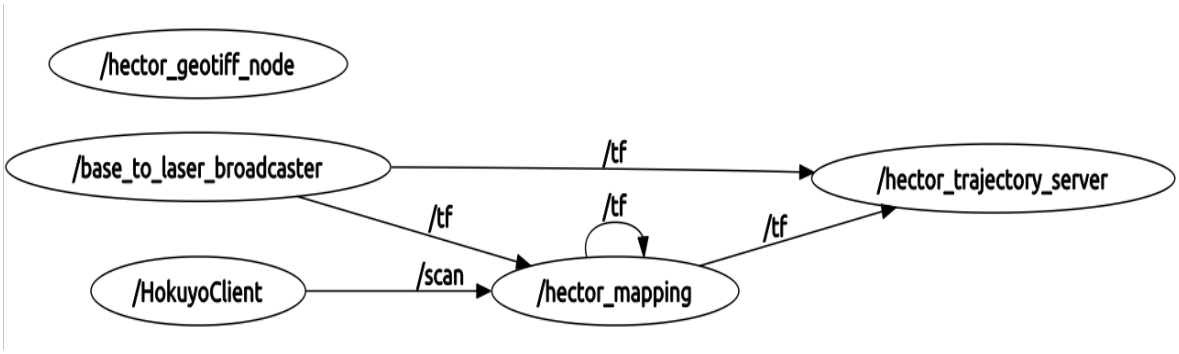

Figura 4.3 Comunicación entre HokuyoClient y hector_slam.

4.4. Configuración de la Raspberry pi para la Creación de un Servidor Ad-Hoc.

Para poder llevar a cabo la transferencia de datos de manera inalámbrica entre la Raspberry pi y el PC encargado de ejecutar el software de SLAM ha sido necesario crear una red Ad-Hoc entre los dos dispositivos, ejerciendo la Raspberry pi de servidor.

Para ello, primero se debe instalar el dispositivo WiFi, en este caso el dongle 802.11b/g/n Nano USB con antena externa. Después de ello se ha instalado un servidor DHCP en la Raspberry pi, en este caso el isc-dhcp-server, necesario para asignar

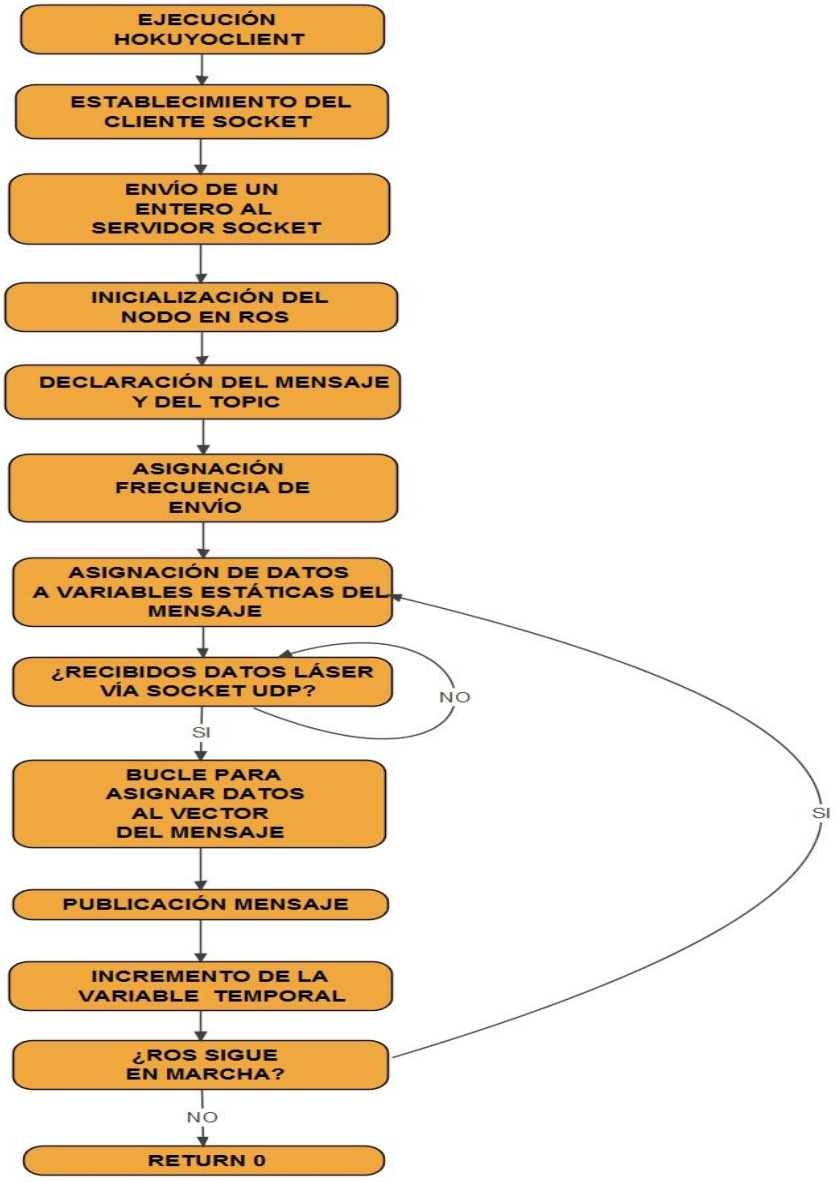

Figura 4.4 Diagrama de funcionamiento de HokuyoClient.

direcciones IP a los clientes que se conecten a le red ad-hoc, y se ha modificado su fichero de la carpeta “default” para que escuche las peticiones de IP en la interfaz “wlan0”.

También se ha editado el fichero de configuración de los interfaces de red (figura 4.4), para permitir el arranque automático de la interfaz “wlan0” y la asignación estática de la IP.

Otro tanto se ha hecho con el fichero de configuración del dhcp, llamado “dhcpd.conf” (figura 4.5). Este fichero establece la configuración del servidor DHCP y se deben incluir en él los tiempos por defecto y máximo para intentar establecer una conexión, la subnet-mask, el nombre de la red, la IP estática del servidor, y el rango de direcciones IP que tomaran los clientes que se conecten a la red.

Por último, y para que la red ad-hoc sea creada automáticamente cuando arranque la Raspberry pi, se ha editado uno de los ficheros de arranque, rc.local, (figura 4.5) añadiéndole un script que crea y configura la red inalámbrica cada vez que el dispositivo se enciende. La edición se ha hecho mediante conexión ssh vía cable Ethernet, usando la terminal de comandos y el editor de textos nano.

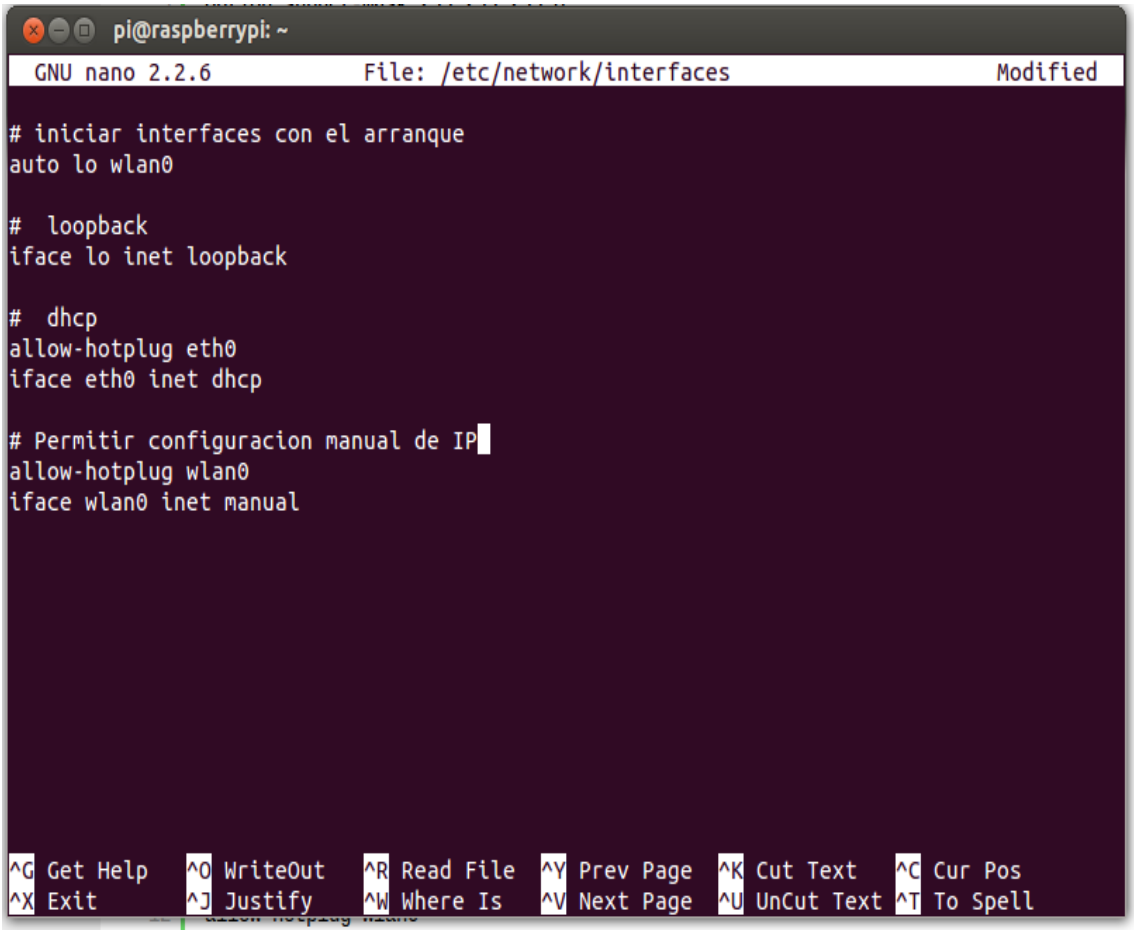

Figura 4.4 Fichero de configuración de los interfaces de red.

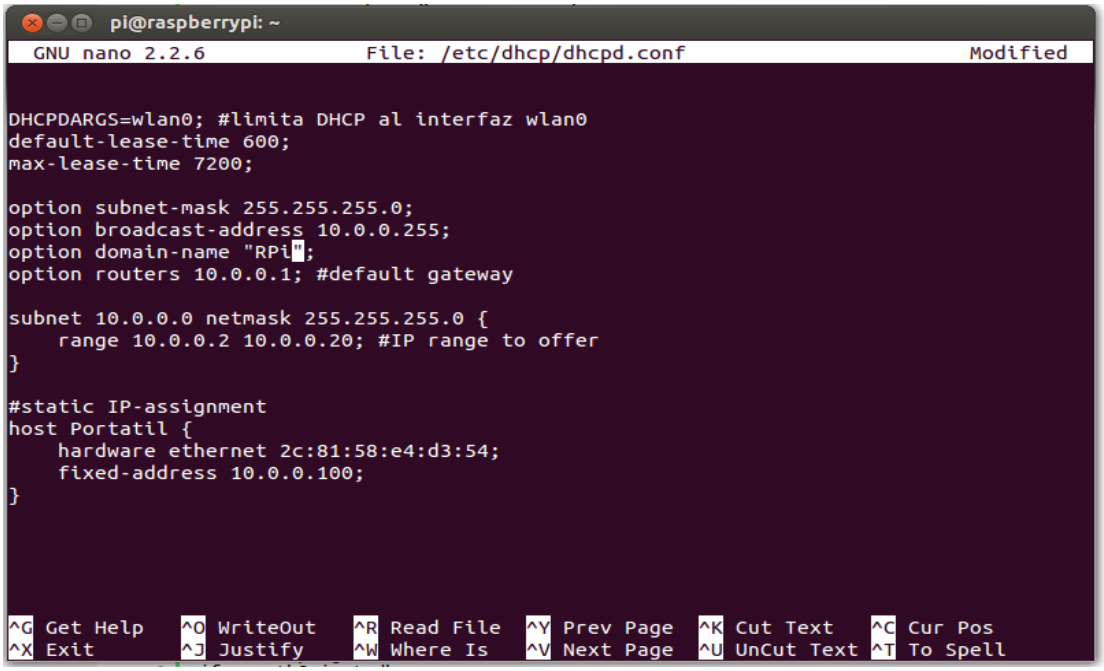

Figura 4.5 Fichero de configuración del servidor dhcp.

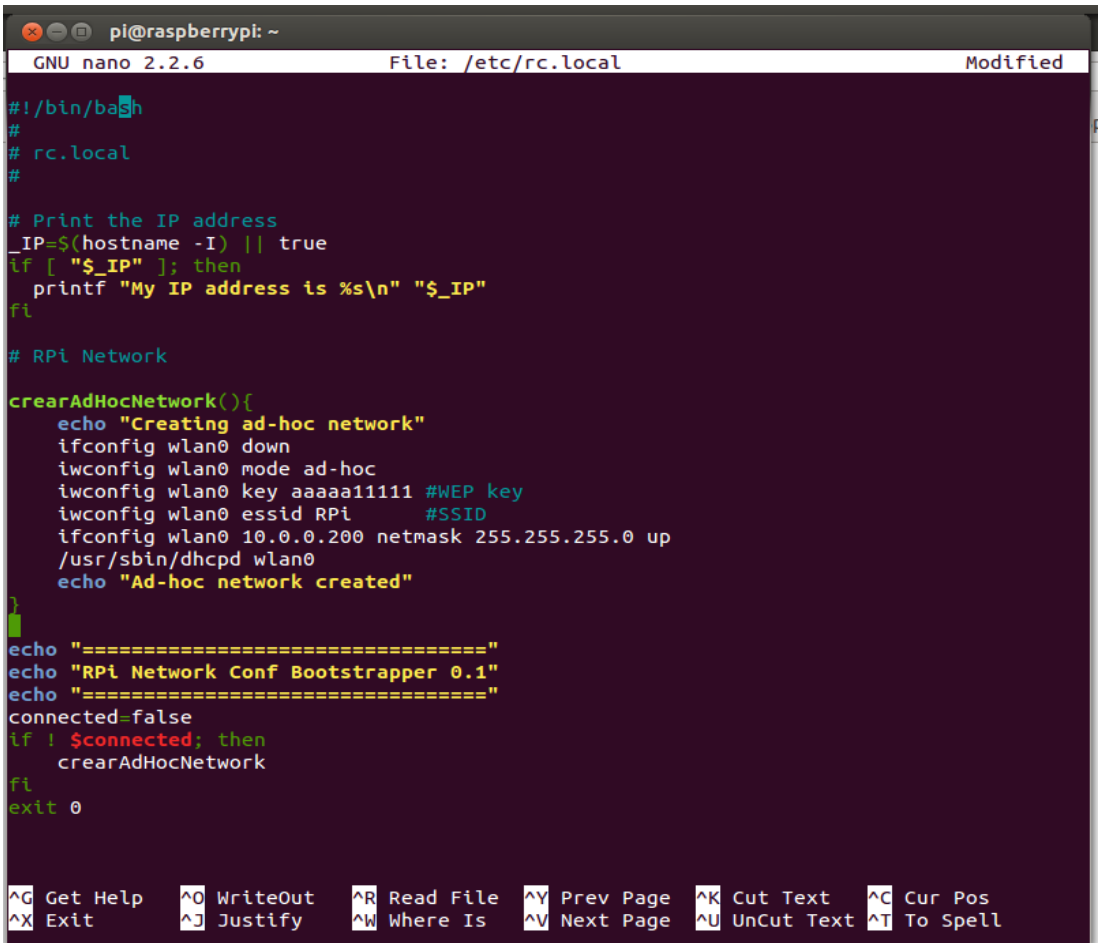

Figura 4.6 Script agregado en el fichero rc.local para la creación de red ad-hoc durante el arranque de la Raspberry pi.

5. ALGORITMO DE CONTROL.

5.1. El Regulador PD.

Para llevar a cabo el control de posición x-y para detección y evasión de obstáculos, se ha utilizado un control de tipo PD (acción proporcional derivativa), ya que el controlador derivativo se opone a desviaciones de la señal de entrada, con una respuesta que es proporcional a la rapidez con que se producen estas.

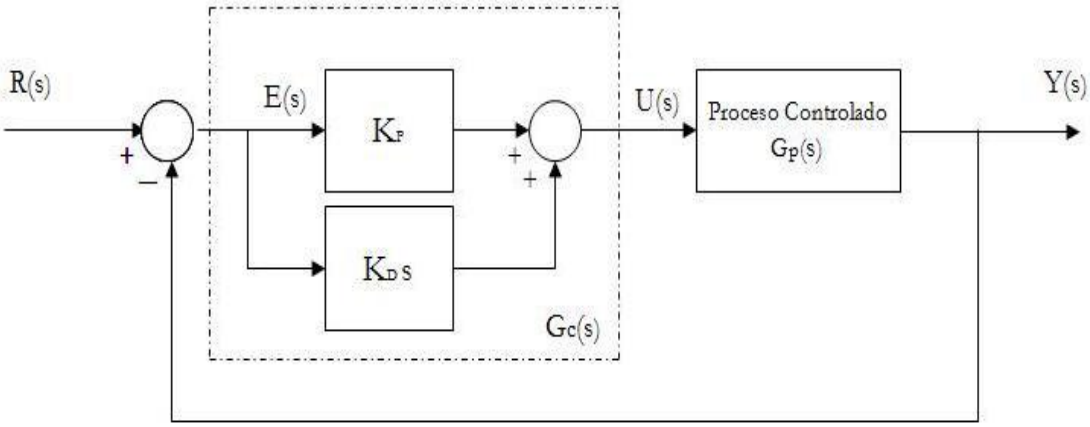

Figura 5.1 El controlador PD.

La salida de este regulador derivativo es:

$$y(t) = t_d \cdot \frac{\delta e(t)}{\delta t} \qquad (5.1)$$

Que en el dominio de Laplace es:

$$Y(s) = T_d \cdot s \cdot E(s) \qquad (5.2)$$

Por lo que su función de transferencia será:

$$G(s) = \frac{Y(s)}{E(s)} = T_d \cdot s \qquad (5.3)$$

Si la variable de entrada es constante, no da lugar a respuesta del regulador diferencial, cuando las modificaciones de la entrada son instantáneas, la velocidad de variación será muy elevada, por lo que la respuesta del regulador diferencial será muy brusca, lo que haría desaconsejable su empleo. Es por esto que no actúa exclusivamente, si no que siempre lleva asociada la actuación de un regulador proporcional (por eso se habla de regulador PD), la salida del bloque de control responde a la siguiente ecuación.

$$y(t) = K_P \cdot t_d \cdot \frac{\delta e(t)}{\delta t} + K_P \cdot e(t) \qquad (5.4)$$

K_p y t_d son parámetros ajustables del sistema. A t_d se le llama tiempo derivativo y es una medida de la rapidez con que un controlador PD compensa un cambio en la variable regulada, comparado con un controlador proporcional puro. Esta acción básica genera una acción de control que es proporcional a la derivada del error. Se trata de conseguir una reacción ante las tendencias del error (medido en forma de su derivada), es decir, se dota al regulador de un cierto sentido de anticipación (Blasco y otros, 2000 [13]).

La salida del bloque de control en el dominio de Laplace, será:

$$Y(s) = K_P \cdot T_d \cdot s \cdot E(s) + K_P \cdot E(s) \qquad (5.5)$$

Por lo que su función de transferencia:

$$G(s) = \frac{U(s)}{E(s)} = K_P \cdot (T_d \cdot s + 1) \qquad (5.6)$$

La respuesta que ofrece el controlador PD se anticipa a la propia señal de error. Este tipo de controlador se utiliza en sistemas que deben actuar muy rápidamente, ofreciendo una respuesta tal que provoca que la salida continuamente esté cambiando de valor. La ventaja de este tipo de controlador es que aumenta la velocidad de respuesta del sistema de control. Al actuar conjuntamente con un controlador proporcional las características de un controlador derivativo provocan una apreciable mejora de la velocidad de respuesta del sistema, aunque pierde precisión en la salida (durante el tiempo de funcionamiento del control derivativo).

5.2. Implementación.

Debido a las características y disposición habituales en un quadrotor (hélices, varillas de protección, etc…) y en concreto del quadrotor del ai2 (Instituto de Automática e Informática Industrial), utilizado para validar el algoritmo diseñado en este proyecto, ha sido necesario modificar los drivers del láser, para generar una lectura que no se viese contaminada por las medidas que proporcionaría el dispositivo al detectar elementos del quadrotor, tales como pueden ser las varillas de protección

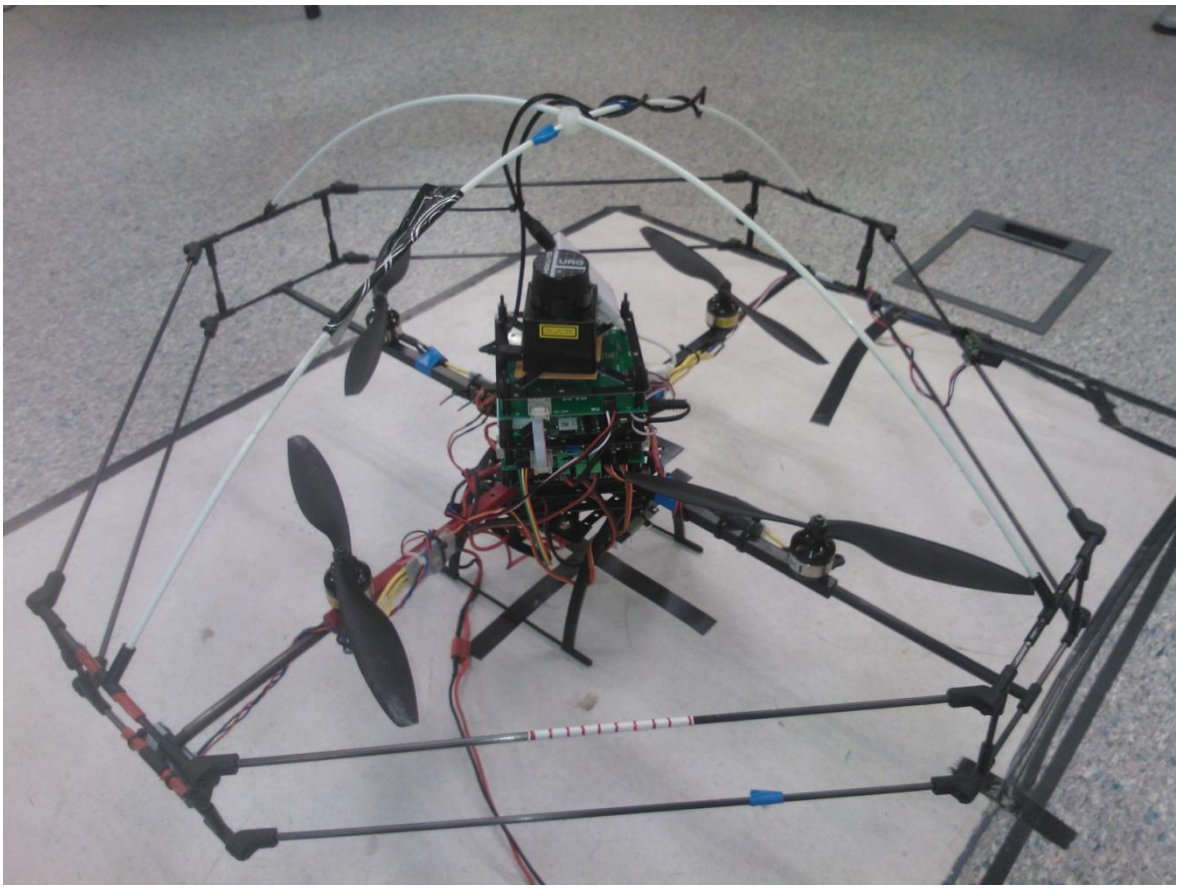

Figura 5.2 Quadrotor del ai2.

Para este efecto, se utiliza la función *ReadLaser ()*. Esta función toma todas las medidas del sensor, y las divide según tres direcciones cartesianas, a saber eje X, eje Y positivo, y eje Y negativo (debido a los aspectos constructivos del LIDAR, es imposible tomar las medidas que corresponderían al eje X negativo). Tomando 200 steps, o puntos de medida (aproximadamente un haz de 85º), para cada dirección, comparándolos entre sí y obteniendo de esta forma la medida más pequeña en cada eje, es decir la distancia del obstáculo más próximo según cada dirección.

Estas son las tres medidas que se le pasaran al algoritmo de control (*medida_X, medida_Ypos, medida_Yneg*), sobre las cuales estará basado el control de posición x-y con medidas laser.

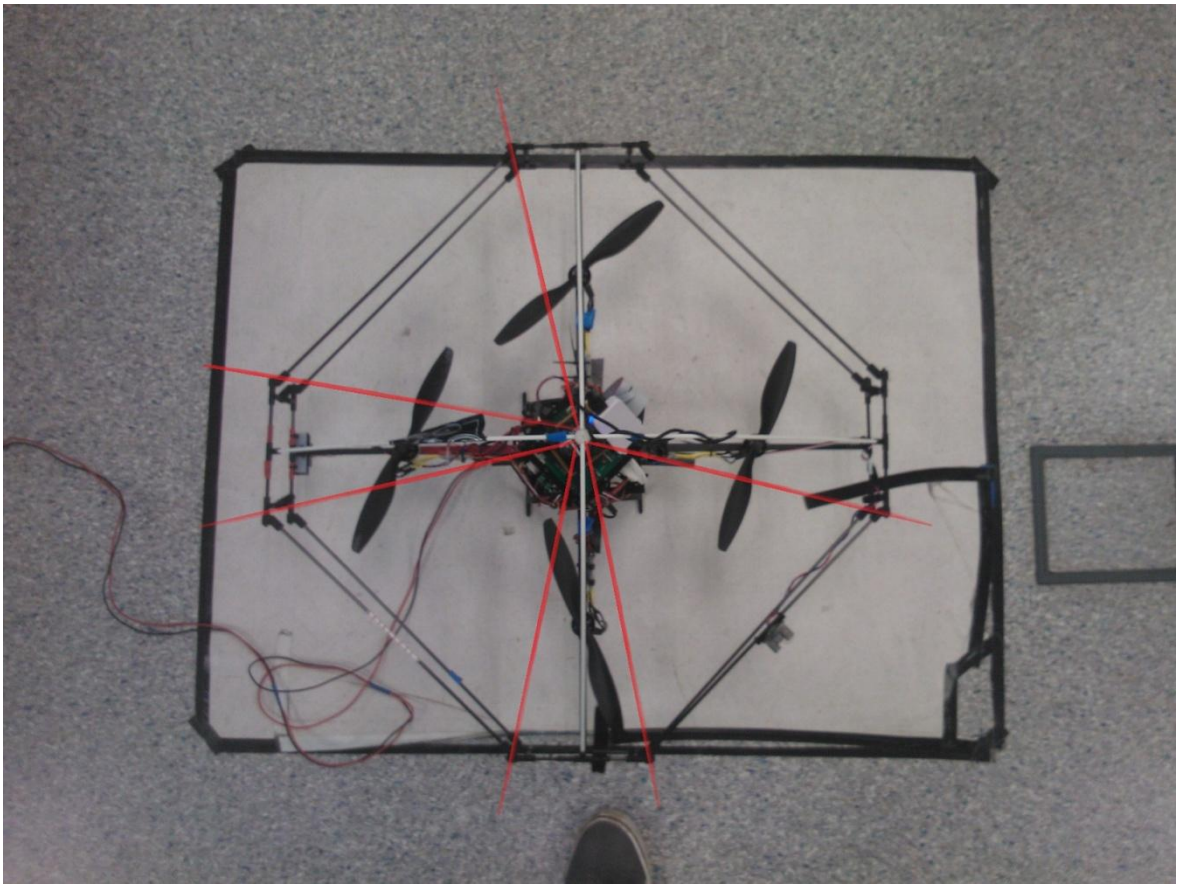

Figura 5.3 Representación gráfica de los haces de medida de aproximadamente 85º del LIDAR empleados para el algoritmo de control, con el objetivo de evitar la interferencia de las varillas de protección.

Una vez obtenidas estas medidas, se filtraran, mediante un filtro paso bajo digital de primer orden (o filtro promediador), con el fin de atenuar el posible ruido de las medidas, y cambios bruscos de la referencia a alta frecuencia. El filtro responde a la siguiente expresión:

$$Medida_{filtr} = Medida_{actual} \cdot (n) + Medida_{filtr\ ant} \cdot (1 - n) \qquad (5.7)$$

Con n, un número entre [0,1].

Una vez las medidas han sido filtradas, se calcula la derivada, mediante la diferencia entre la medida filtrada y la medida filtrada anterior, multiplicada por 10 (frecuencia de toma de medidas, en lugar de dividir entre el tiempo entre medidas). Una vez hecho esto, la derivada se filtra mediante un filtro paso bajo digital, igual al anterior.

Después se establece una estructura de if's, para determinar si se activa el control o no, en función de las medidas recibidas.

Este proceso se lleva a cabo para las tres medidas mencionadas, con una frecuencia de 10Hz (condicionada por el funcionamiento del láser). En caso de activar el control de posición sobre alguno de los ejes, se calcula una variación del valor de la referencia para el ángulo de orientación del quadrotor necesario, es decir, el ángulo roll en el caso del eje Y, y el ángulo de pitch en el caso del eje X. Según

$$w_i = K_p \cdot \left(Valor_{ref} - Medida_{filtr}\right) - K_d \cdot \left(Derivada_{filtr}\right) \qquad (5.8)$$

Siendo K_p y K_d las ganancias proporcional y derivativa respectivamente y $Valor_{ref}$, la distancia programada de antemano para la detección y evasión de obstáculos, es decir la referencia que el control debe mantener. Para ambos términos de la ecuación se establecerán unos límites de saturación, para evitar variaciones demasiado elevadas de los ángulos de posición del quadrotor; es decir si alguno de los dos términos sobrepasa el nivel inferior o superior de estos límites, su valor quedará igualado al valor del límite. Para el término proporcional se establecen unos límites de saturación de ±20, y para el término derivativo, un valor de ±10.

De esta forma se obtiene el valor *wi*, (según el eje, *wx* para el eje X, *wyy* para el eje Y, y *wyy1* para el eje Y negativo), este valor será sumado al valor de referencia del ángulo de pitch en el caso del eje X, restado al valor de referencia del ángulo roll en el caso del eje Y positivo, y sumado al valor de referencia del ángulo roll en el caso del eje Y negativo.

De esta forma, se introducen cambios en los ángulos de pitch y roll, que dominan el movimiento x-y del quadrotor, haciéndole efectuar un cambio en su posición cuyo objetivo será a la minimización del parámetro w_i, es decir el movimiento que efectúe el quadrotor tenderá a aproximar $Medida_{filtrada}$ y $Valor_{ref}$, y tenderá a anular la derivada, y de esta forma se mantendrá la distancia referencia entre quadrotor y obstáculo programada de antemano.

6. Resultados Obtenidos.

En este apartado se muestran los resultados obtenidos a lo largo de distintas pruebas a las que se ha sometido el sistema implementado de mapeo y localización simultanea, siendo mostrados por la interfaz gráfica rviz así mismo se comentan ciertos aspectos de interés y se hacen consideraciones al respecto.

Sobre el algoritmo de control con acción Proporcional-Derivada, se han realizado pruebas en vuelo real para ajustar las ganancias de manera empírica y para comprobar su correcto funcionamiento.

6.1. Algoritmo SLAM.

- Mapeo de una habitación.

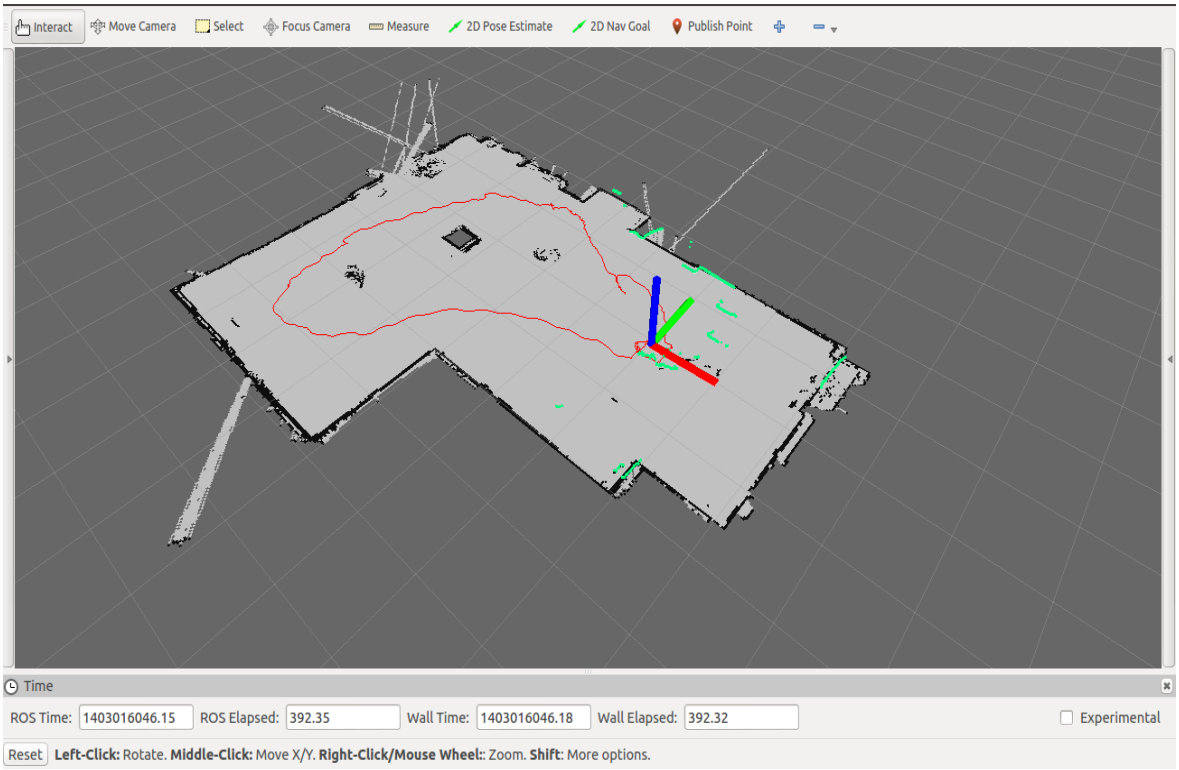

Figura 6.1 Mapeo de una habitación.

Como puede observarse los resultados de esta primera prueba son satisfactorios. La velocidad de la transmisión de datos a través de la conexión ad-hoc es lo suficientemente rápida para enviar las medidas del láser en tiempo real al PC encargado de ejecutar el algoritmo de SLAM.

Puede observarse la trayectoria seguida marcada con una línea roja, el recorrido ha consistido en dar una vuelta a la habitación y volver al punto de partida. Se hace notar también una de las limitaciones del sensor, que proporciona medidas falseadas cuando el haz láser se refleja en superficies tales como cristales o espejos. El mapa de celdas es un mapa de ocupación, en el que cada celda tiene un lado de 1 metro, y a su vez está dividida en celdas de menor tamaño, y estas en otras menores; de forma que el mapa pueda ser salvado en formato geotiff.

- Recorrido de varias habitaciones.

En este prueba se ha procedido a realizar el proceso de SLAM a través de varias habitaciones y pasillos, para finalmente pasar de un entorno indoor a un entorno outdoor. Los resultados pueden verse en la figura 6.2.

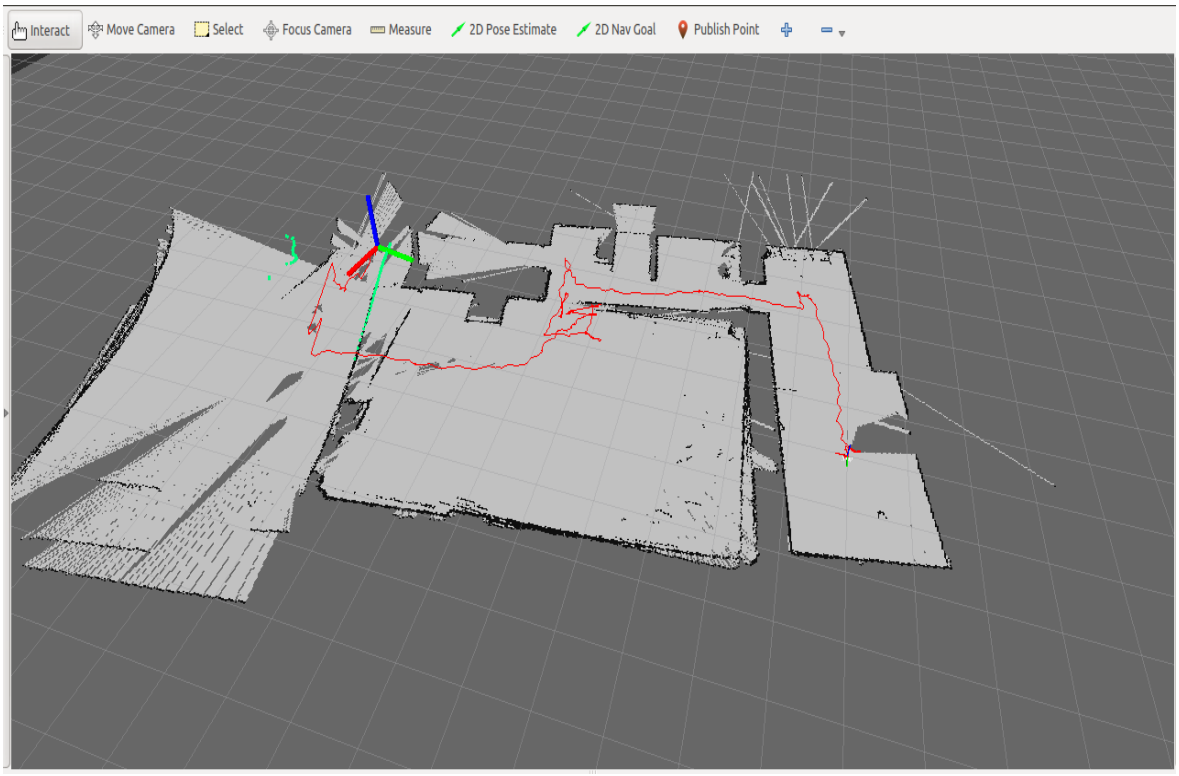

Figura 6.2 SLAM a través de varias habitaciones, hasta entorno outdoor.

Como puede observarse, el resultado obtenido es satisfactorio, más allá de pequeños errores provocados por el cálculo de la posición estimada. En la parte izquierda del mapa se puede observar el paso a entorno outdoor. Los resultados de esa zona son constatablemente peores, ya que el sensor láser empleado está diseñado para trabajar en entorno indoor; no obstante los obstáculos que ha llegado a detectar han sido marcados y mapeados de manera correcta.

En esta caso también se ha visto puesta a prueba la conexión inalámbrica, de forma que la distancia entre Raspberry pi y el PC encargado de ejecutar el software fuese creciendo

progresivamente, mientras que en el resto de pruebas se han mantenido relativamente próximos. En este aspecto el funcionamiento ha sido satisfactorio manteniéndose la transmisión de datos aportados por el LIDAR a velocidad suficiente para mantener en funcionamiento el algoritmo de SLAM.

- Recorrido de varias habitaciones y retorno al punto de partida.

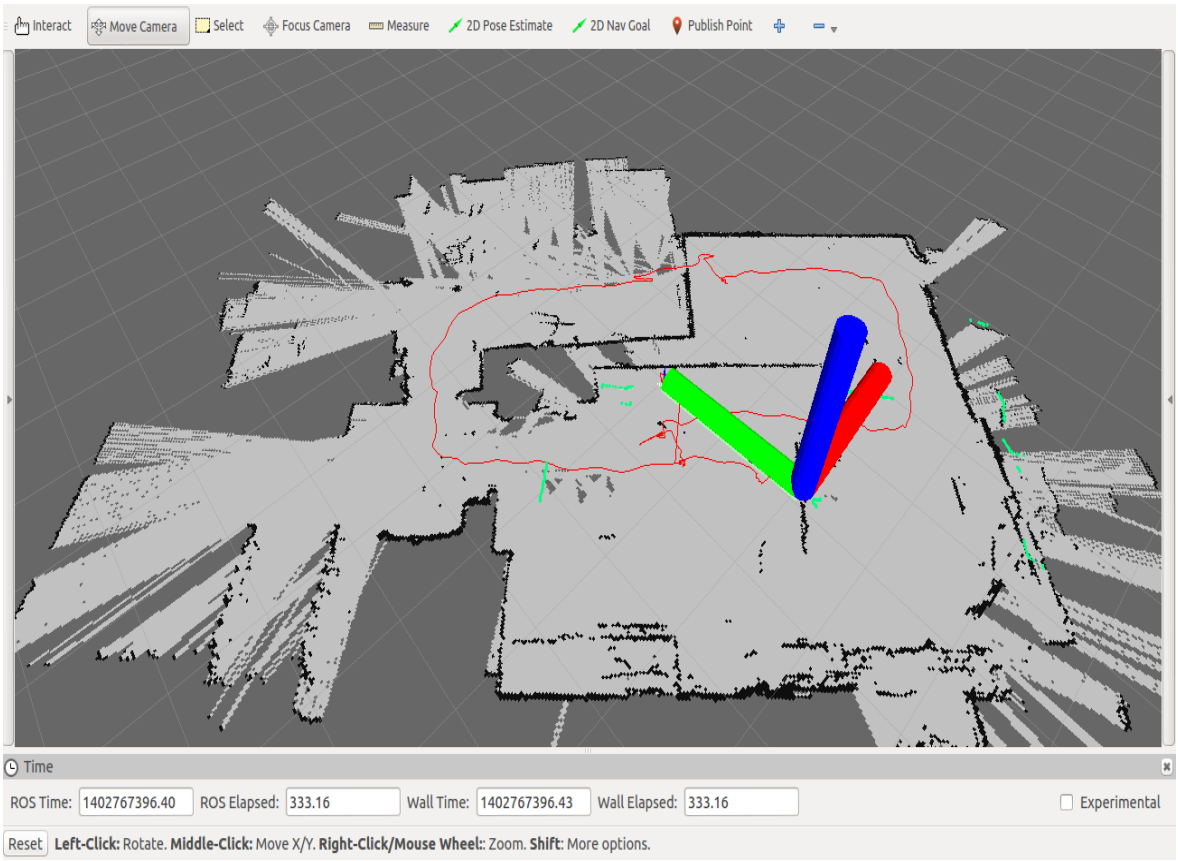

Figura 6.3 Recorrido de varias habitaciones y retorno al punto de partida.

En este caso se vuelven a producir los problemas que provocan el hecho de que las paredes del entorno sean de un material que refleje el haz laser de manera distinta a como espera el sensor. Por otra parte, como se puede ver en la parte superior de la figura 6.3, hay una pequeña perdida de referencia que se refleja en un cambio brusco de la trayectoria, que no se produjo en la realidad, esto provoca que la habitación de la parte superior izquierda aparezca ligeramente rotada, cuando sus paredes debieran ser completamente paralelas entre sí. No obstante este pequeño error, el resultado sigue siendo satisfactorio. El motivo de que algunas habitaciones no están mapeadas completamente es que el recorrido fue demasiado rápido, no dejando tiempo a re-observar las landmarks necesarias para que fuesen marcadas como mapeadas el 100% de las zonas recorridas. Hay que recordar que se

está limitado por la frecuencia de escaneo del láser, es decir 10 Hz. En la prueba que muestra la figura 6.4 se ponen de manifiesto estos dos problemas, pues un aumento de la velocidad de desplazamiento, combinado con la existencia de paredes de cristal, provocan una perdida completa de la referencia durante el retorno a punto de partida, lo que genera que zona ya mapeada se vuelva a mapear en distinta posición y que por lo tanto la trayectoria sea errónea puesto que hay puntos en el mapa que están mapeados dos veces en distintas posiciones.

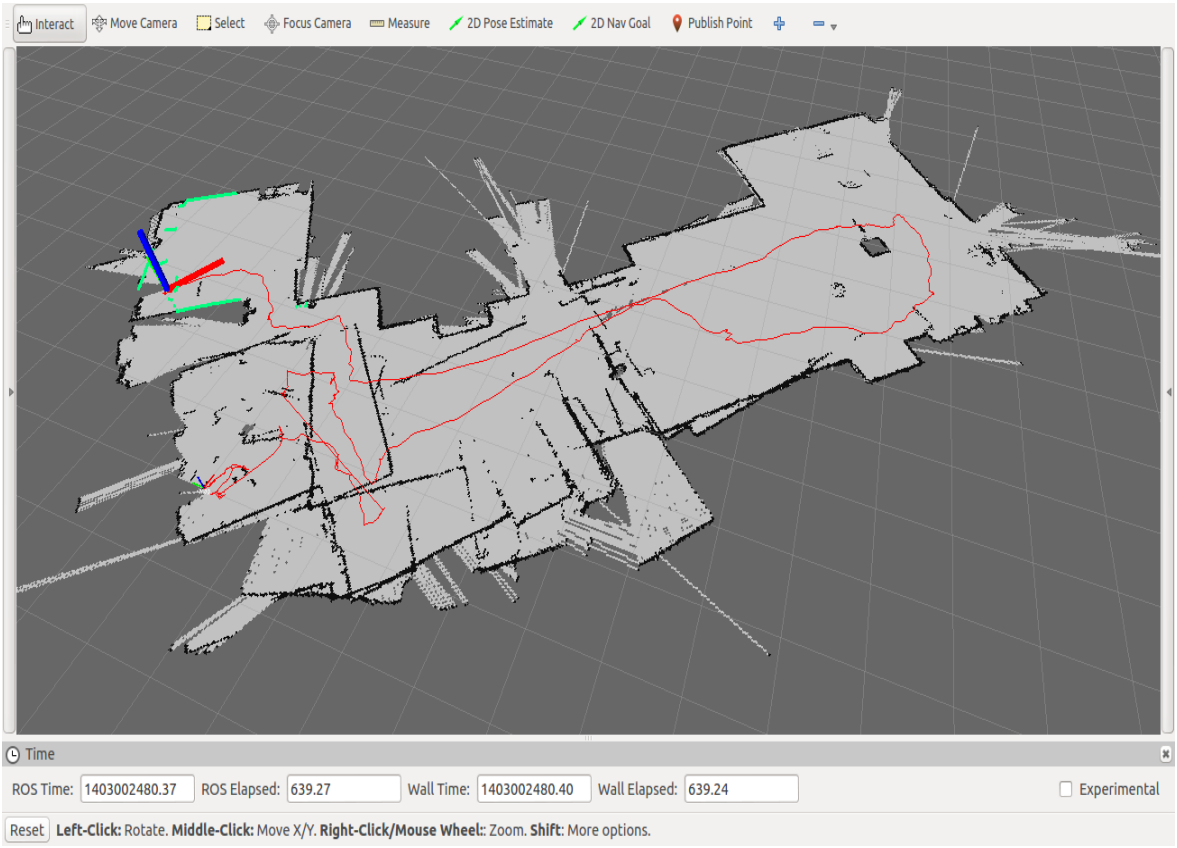

Figura 6.4

Como se puede observar en la zona izquierda del mapa de la figura 6.4 la perdida de referencia provocada por una velocidad de desplazamiento alta y superficies reflectantes hace que en el tramo final del mapa la trayectoria ploteada sea errónea, y haya zonas mapeadas dos veces. Este tipo de errores se mitigarían con una mayor frecuencia de muestreo del sensor láser, lo que significaría mayor velocidad en la extracción de landmarks y más fiabilidad en la estima de posición basada en las medidas del sensor. Aunque con el que se ha trabajado en este proyecto funciona a un máximo de 10Hz, el mismo fabricante tiene en catalogo sensores con una frecuencia de funcionamiento cuatro veces mayor, como puede ser el Hokuyo-UTM30. No obstante, también hay que señalar que los LIDAR's capaces de realizar el muestreo a una mayor frecuencia, también tienen un precio más elevado.

6.2. Algoritmo de Control de Posición x-y.

Para validar y probar el algoritmo de control de posición, se ha implementado sobre el quadrotor del ai2. Originalmente se usaba un SOTR (Sistema Operativo de Tiempo Real) llamado Xenomai, pero resultó del todo imposible, sin motivo aparente, hacer funcionar el dispositivo láser sobre dicho sistema operativo (seguramente algún tipo de problema con la configuración de los niveles de tensión del puerto serie, o con el modo de funcionamiento de los puertos USB) por lo que se decidió que era mejor pasar a un sistema donde ya estuviera correctamente configurado adecuadamente el puerto USB, para ello se reestructuro el programa de funcionamiento del quadrotor usando la librería pthread.h y el sistema operativo original de base Linux, perdiendo así las ventajas de trabajar en tiempo real, con el detrimento que esto tiene para las labores de control.

Las pruebas fueron realizadas en vuelo real, con dos personas situadas en las direcciones *X* e *Y*, haciendo la función de obstáculos móviles, de forma que el quadrotor tuviese que modificar su posición para mantener constante la referencia de distancia a los obstáculos programada de antemano.

En la primera prueba que se ha realizado, se ha incluido el programa para tomar las medidas del sensor dentro de otro hilo para ajustarse a la frecuencia de funcionamiento del láser; y tiene como objetivo establecer una aproximación empírica para las ganancias del algoritmo de control K_p y K_d, tras diversos ensayos de prueba y error, se consigue establecer $K_p = 3$ y $K_d = 2$. Y para una referencia a mantener de 0,7 metros, y activando la estrategia de control para distancias menores de 1,5 metros y desactivándolo para distancias mayores de 2 metros; se obtienen los siguientes resultados, mostrados en la Figura 6.5.

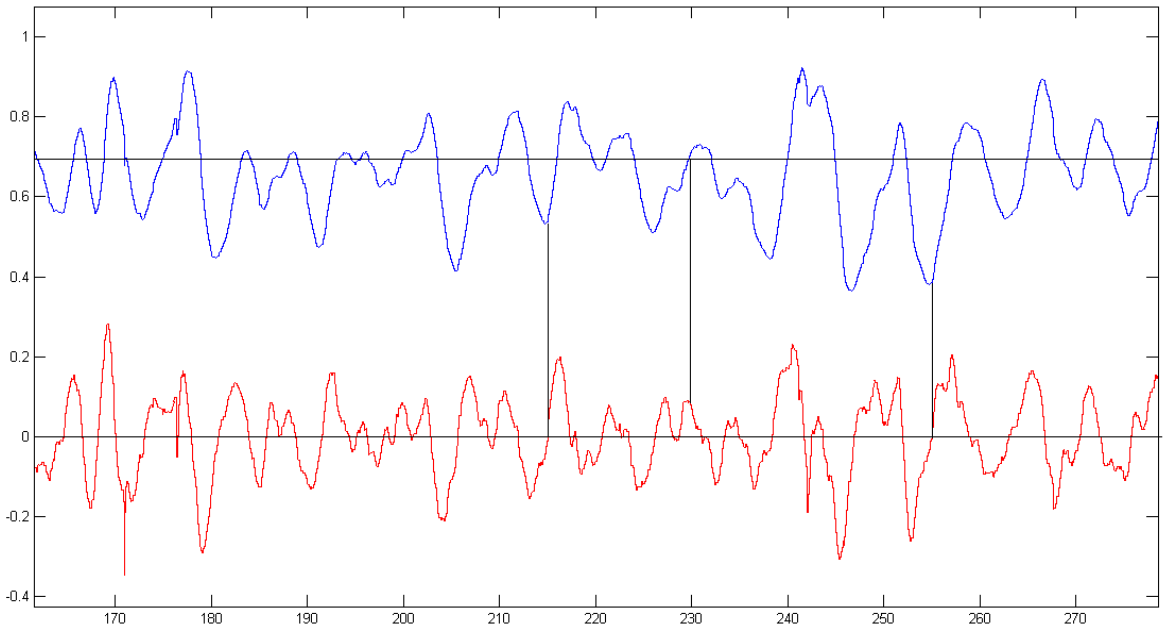

Figura 6.5 Representación de la posición del quadrotor según las medidas del LIDAR (azul) y de la derivada filtrada (rojo) a lo largo del tiempo. En metros y segundos.

Como puede observarse, la distancia de referencia programada (0,7 m) es mantenida gracias al algoritmo de control, también puede observarse que los valores en de tiempo en los que la señal de la derivada pasa por cero, coinciden con los puntos de inflexión de la señal de posición (máximos y mínimos), y viceversa, los pasos por cero de la señal de posición coinciden con los máximos y mínimos de la señal de la derivada filtrada. Hay que tener en cuenta que se trata de las combinaciones de las medidas de los tres ejes evaluados, es decir eje X, eje Y positivo y eje Y negativo, tomando en cada escaneo (cada 100 ms) la menor de las medidas de distancia.

Cabe destacar que la labor de control no es tan precisa como se pudiera esperar debido a la imposibilidad de utilizar un SOTR, ya que el control de orientación sufre una suerte de retardo debido a los cambios de contexto entre tareas dentro del programa del quadrotor.

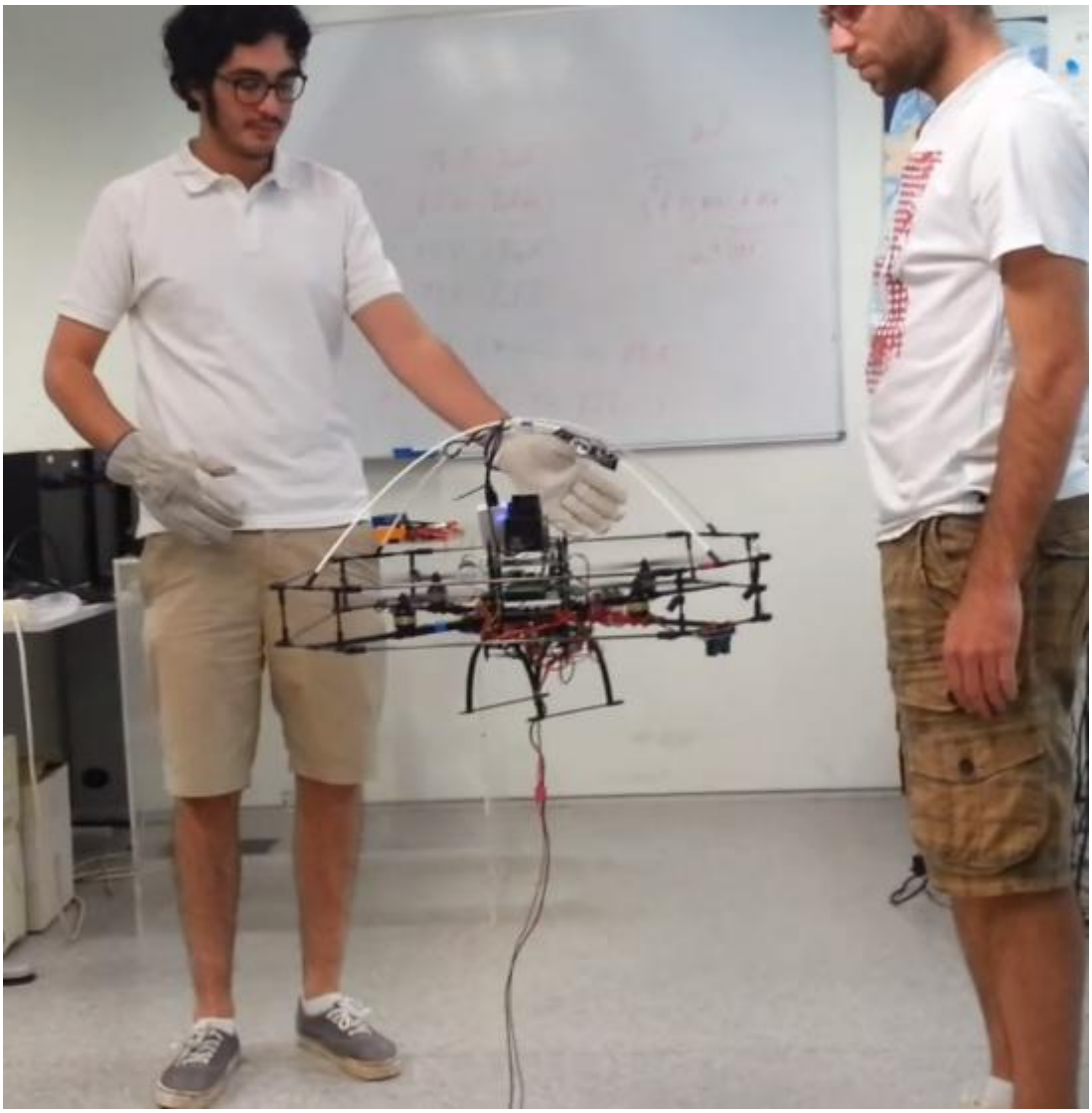

Figura 6.6 Ensayo en vuelo real sobre el algoritmo de posición x-y.

Para tratar de solucionar o mitigar este problema, se implementó la toma de medidas del láser dentro de su propio hilo, con la meta de minimizar los desajustes provocados por los cambios de contexto entre tareas mencionados antes. En este caso las ganancias K_p y K_d, se ajustaron de nuevo a 2 y a 3 respectivamente, y se reajustó la activación de las tareas de control para que se produjese cuando se detectase un obstáculo a una distancia igual o inferior a un metro, y se desactivase cuando esta distancia fuese superior o igual a 1,5 metros. Este reajuste se hizo con el fin de impedir que el entorno en el que se realizaron las pruebas interfiriese en las mismas, y los únicos obstáculos detectados por el sensor láser fuesen las personas situadas en las direcciones *X* e *Y*.

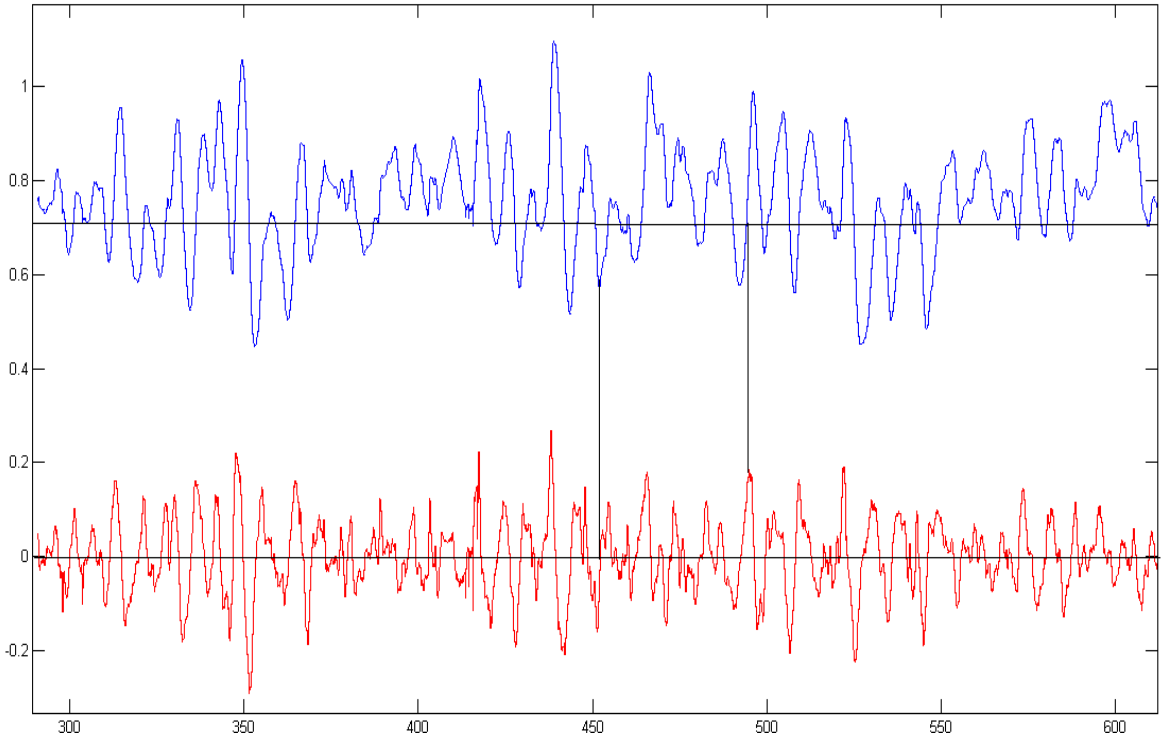

Figura 6.7 Representación de la posición del quadrotor según las medidas del LIDAR (azul) y de la derivada filtrada (rojo) a lo largo del tiempo. En metros y segundos.

A pesar de que durante la prueba los resultados obtenidos eran mejores que en la anterior prueba documentada, la información al respecto obtenida al plotear las variables almacenadas indicaba que la mejoría no era significantemente apreciable, como puede verse en la Figura 6.7.

De nuevo puede apreciarse que la estrategia de control trata de ajustarse constantemente a la referencia de 0,7 metros, así como los puntos de inflexión de la señal de posición del quadrotor (medidas del LIDAR), significan un paso por cero de la señal de la derivada filtrada, y viceversa.

7. CONCLUSIONES Y TRABAJOS FUTUROS.

7.1. Conclusiones.

La finalidad de este trabajo ha sido desarrollar un sistema basado en el sensor láser Hokuyo-URG-04LX-UG01, capaz de realizar labores de SLAM, así como utilizar las medidas aportadas por el LIDAR para realizar un control PD para detectar y evitar obstáculos. Señalar que en un principio se pretendía realizar ambas funciones de manera complementaria, sin embargo la dimensión de este objetivo ha sobrepasado el tiempo disponible para la realización de este proyecto. No obstante, lo conseguido en este trabajo queda preparado para emprender esta labor desde un punto notablemente avanzado.

El uso de hector_slam combinado con ROS ha resultado útil sobre todo a la hora de solventar el problema de la comunicación entre los programas encargados de captar y recibir las medidas del láser y el software encargado del proceso de SLAM en sí, ya que gracias al sistema de topics y mensajes de ROS, únicamente había que disponer de los datos del láser en el formato adecuado para el tipo de mensaje predefinido y publicarlo en el topic al que estaba suscrito el nodo de mapeo.

La principal dificultad ha residido en la programación de los drivers del láser, ya que las funciones disponibles tuvieron que ser sometidas a numerosos cambios para adaptarse a las necesidades de este trabajo, además de la dificultad añadida de tener que adaptarse para trabajar con el protocolo de comunicación SCIP2.0. Por este motivo ha sido necesario hacer pruebas para ir adaptando ciertos parámetros y funciones de forma empírica, para lograr el correcto funcionamiento de los drivers.

Ha sido provechoso trabajar con la Raspberry pi, ya que esta microcomputadora cuenta con una gran cantidad de usuarios y por lo tanto es sencillo encontrar información acerca de sus configuraciones, sus posibilidades y sus limitaciones.

La mayor dificultad de trabajar con el entorno de trabajo de ROS ha sido la novedad de compilar usando CMake files y la terminal de comandos, así como la redacción de los ficheros *.launch necesarios para lanzar tanto los algoritmos de SLAM como la interfaz gráfica rviz (en este último caso ficheros *.rviz). Una vez conocido y dominado el funcionamiento y la forma de trabajar de este framework, es relativamente sencillo desarrollar aplicaciones para robótica móvil, siempre que se tengan los conocimientos requeridos y un dominio suficiente del lenguaje de programación a emplear.

En las pruebas realizadas se puede comprobar que el sistema de mapeo y localización simultánea funciona de manera correcta; salvo cuando se producen perdidas de la referencia, provocadas bien por un exceso en la velocidad de desplazamiento del robot o bien por un entorno material que impide que el sensor láser desempeñe correctamente su cometido. Como se ha reseñado estas pérdidas de referencia podrían mitigarse o reducirse empleando un LIDAR que funcionase con una mayor frecuencia de muestreo; no obstante

siempre que el entorno sea el adecuado y se mantenga la velocidad de desplazamiento dentro de un máximo, esto no tendría por qué representar un problema grave.

También deben ser mencionados los notables conocimientos adquiridos tanto durante la realización del trabajo como durante la fase previa de documentación y búsqueda de información preliminar; sobre todo en programación en C++, manejo de software a alto nivel, el trabajo con sensores láser, protocolos de comunicación entre dispositivos, robótica móvil, sistemas de posicionamiento, etc…

Comentar por último, que quedan pendientes trabajos futuros, tanto para mejorar y complementar el sistema diseñado en este proyecto, como para utilizarlo y beneficiarse de sus posibilidades, tal y como se indica en el siguiente apartado.

7.2. Trabajos Futuros.

1. **Uso de los mapas generados en formato geotiff.**

Una vez mapeado un entorno, consistiría en salvar dicho mapa en formato geotiff mediante hector_geotiff_node. El formato geotiff permite que información georreferenciada sea encajada en un archivo de imagen, de forma que dicha imagen pueda ser automáticamente posicionada en un sistema de referencia espacial.

El trabajo consistiría en pasar dichos mapas georreferenciados a la unidad de control central del UAV, de forma que el quadrotor siga una ruta marcada dentro de dicho mapa.

2. **Ampliación del número de pruebas sobre el sistema SLAM.**

Aunque los problemas encontrados son debidos a la combinación de dos factores como son el entorno material y la velocidad de desplazamiento del robot, se propone un número más exhaustivo de pruebas documentadas debidamente y observadas en profundidad; con el objetivo de determinar una relación más exacta entre el error de pérdida de la referencia, el tipo de entorno, la velocidad de desplazamiento del UAV y la frecuencia de muestreo del sensor láser.

Los tipos de entorno podrían ser clasificados en función del porcentaje de tramos de pared acristalada o tramos de ventanas que posean. De esta forma se podría poner una cota superior a la velocidad de desplazamiento del robot, según el tipo de entorno a navegar, de manera que no se incurriese en errores de pérdida de la referencia sin necesidad de tener que invertir en un LIDAR de precio más elevado.

3. Creación de aplicaciones utilizando otros paquetes de software de ROS.

Además del paquete de software utilizado en este proyecto, el framework ROS cuenta con una gran cantidad de paquetes, repositorios y librerías pensados para el desarrollo de múltiples aplicaciones de robótica móvil. Y de esta forma aprovechar tanto la versatilidad en su grado de uso, como su método de comunicación entre nodos.

Por ejemplo, se podría conseguir un sistema de control de trayectoria y estima de la posición utilizando imágenes proporcionadas por una cámara. O tratar de realizar un mapeo en tres dimensiones combinando varios sensores láser.

4. Implementación del sensor Hokuyo –URG – 04LX – UG01 en un SO de tiempo real.

Una de las limitaciones que han marcado a este proyecto ha sido la imposibilidad de implementar el LIDAR en el entorno de tiempo real Xenomai, una circunstancia que habría significado una notable mejoría en las labores de control. Así pues uno de los posibles trabajos futuros sería encontrar la raíz del problema en la conexión entre el SO Xenomai y el controlador interno del láser, y reconfigurar el primero, de forma que se hiciese posible el uso de este sensor en conjunción con el SOTR.

5. Uso combinado del algoritmo de control y sistema SLAM.

Seguramente el futuro trabajo más interesante. Consistiría en hacer uso a la vez del sistema de mapeo y localización simultáneos con el algoritmo de control de posición, de forma que al detectar y evitar un obstáculo, la unidad de control inteligente fuese capaz de tomar una decisión acerca de la nueva dirección a seguir en función del mapeo del entorno y la posición del quadrotor en él realizados hasta el momento. Utilizando una transmisión de datos rápida entre quadrotor y PC base; o incluso, realizando el proceso de SLAM en una de las microcomputadoras del quadrotor, esto sería realizable incluso en entornos desconocidos, es decir mientras se está generando el mapa con las medidas de los sensores, tomar decisiones acerca de la dirección a seguir, ya sea hacia la zona más libre de obstáculos, o hacia zonas abiertas sin mapear para continuar con una hipotética exploración.

III. MANUAL DE USUARIO

En esta parte se pasará a explicar las acciones necesarias a realizar para conseguir la correcta puesta en marcha de la aplicación SLAM que se ha desarrollado para el robot.

1. Software Necesario.

Para utilizar las aplicaciones programadas en este proyecto es necesario disponer de un ordenador con una distribución de Linux instalada. En este manual se especificarán las instrucciones para una distribución de Ubuntu 12.10. Además se deberá tener descargado e instalado ROS-groovy junto a las librerías necesarias para su funcionamiento, además del paquete hector_slam. Para la compilación de programas y aplicaciones bastará con gcc, g++ y las librerías habituales de C++. En este proyecto se ha usado Code::Blocks, pero podría servir cualquier IDE capaz de realizar compilación cruzada con su correspondiente compilador para arquitectura ARM.

También será necesario disponer de una Raspberry pi con el sistema operativo Raspbian. Para realizar la conexión inalámbrica hará falta tener instalado isc-dhcp-server en la microcomputadora, así como un dongle WiFi.

Pese a no ser necesario, se recomienda consultar al menos los tutoriales básicos de ROS, disponibles en su página web.

2. Instrucciones a Seguir.

Una vez estén escritos los archivos de configuración y el script en rc.local en la raspberry pi para generar la red Ad-hoc en el arranque, así como los códigos en C++ tanto para las Raspberry, como para el PC, se deben compilar estos para ROS mediante la herramienta Catkin_Make. Para ello primero debemos crear un catkin workspace.

Un catkin workspace no es más que una carpeta con una serie de sub-carpetas tales como src/ build/ devel/ e install/, (en este trabajo llamada catkin_ws) y que se linka a las librerías de ROS mediante el comando:

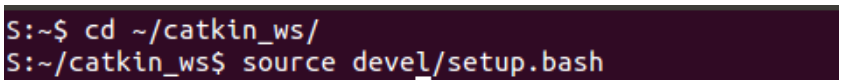

Figura 2.1

Una vez hecho esto, se debe crear un package que contenga el código escrito en C++. Para crear el package simplemente se deberá crear una carpeta dentro de catkin_worspace/src/, con el nombre que se elija, además esta carpeta debe contener un fichero CMake.txt y un fichero package.xml. Para ello se usará la terminal de comandos.

Para crearlo, se abre la terminal de comandos, y sobre la carpeta del catkin workspace se usará el comando catkin_create_pkg <nombre> dependencias. Como dependencias se añadirán: std_msgs sensor_msgs y roscpp, que son los tipos de mensajes y las bibliotecas de ROS para C++ que usa el código.

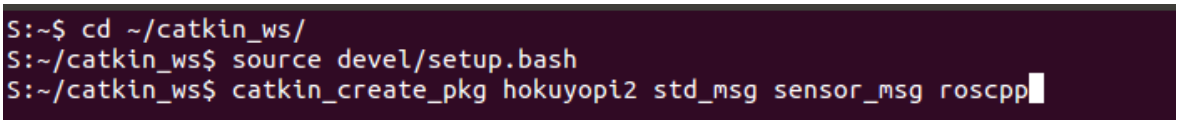

Figura 2.2

Esto genera el package y los ficheros CMake.txt y <nombre_package>.xml, los cuales deben ser rellenados. Para rellenar estos ficheros se puede consultar el Anexo de este trabajo donde aparecen redactados. Como nombre se ha elegido “hokuyopi2”.

Una vez hecho esto, ya se puede agregar el fichero de código, y compilarlo desde la terminal de comandos tecleando lo siguiente.

Figura 2.3

Luego se deberá redactar un fichero *.launch y un fichero *.rviz. En este trabajo llamados slam2.launch y mapping_slam.rviz. Ambos se pueden ver en el Anexo. Estos ficheros se guardaran en un nuevo package llamado raspberryslam, en una carpeta llamada launch. La finalidad del fichero slam2.launch es lanzar los nodos encargados del SLAM, y el fichero mapping_slam.rviz, que contiene los argumentos que hay que proporcionar a la interfaz gráfica rviz para que pueda representar el mapa (colores y tamaños de los ejes, de la línea de trayectoria, tipos de datos, tamaño de las celdillas, etc...).

Una vez hecho esto, ya se puede poner en funcionamiento la aplicación. Para ello será necesario arrancar la Raspberry pi con el sensor láser y el dongle WiFi conectados. Se esperará a que aparezca como disponible la conexión inalámbrica “RPi”, y se conectará el PC a ella. Una vez conectados a la red Ad-Hoc, se abrirá un nuevo terminal y se tecleará

Figura 2.4

Para establecer una conexión ssh entre los dos dispositivos, pedirá un password. En Raspberry pi, por defecto, el nombre de usuario es “pi” y el password “raspberry”. Ante cualquier pregunta se responderá afirmativamente, y pasaremos a encontrarnos en la terminal de comandos de la Raspberry pi. El ejecutable del programa ProgHok se encuentra en una carpeta en la dirección Desktop/ProgHok/bin/Debug/. Se ejecuta desde la terminal de comandos.

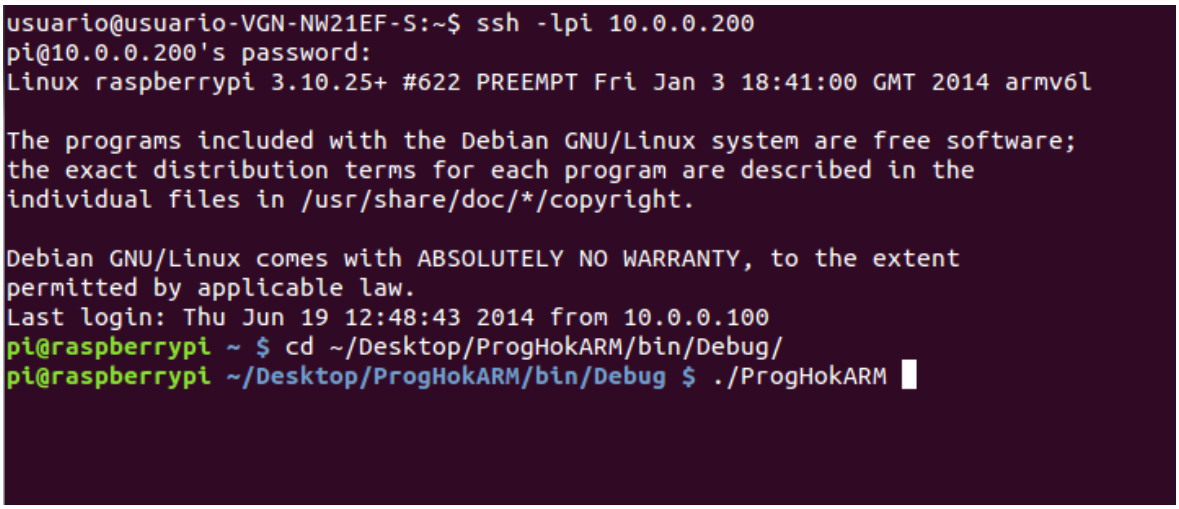

Figura 2.5

Antes de ejecutar es necesario cambiar los permisos de la carpeta a lectura, escritura y ejecución para todos los usuarios. En caso contrario el programa no podrá ser ejecutado.

En un nuevo terminal del PC inicializamos el master de ROS con el comando roscore, este paso es necesario para que se pueda utilizar el sistema de topics y mensajes de ROS.

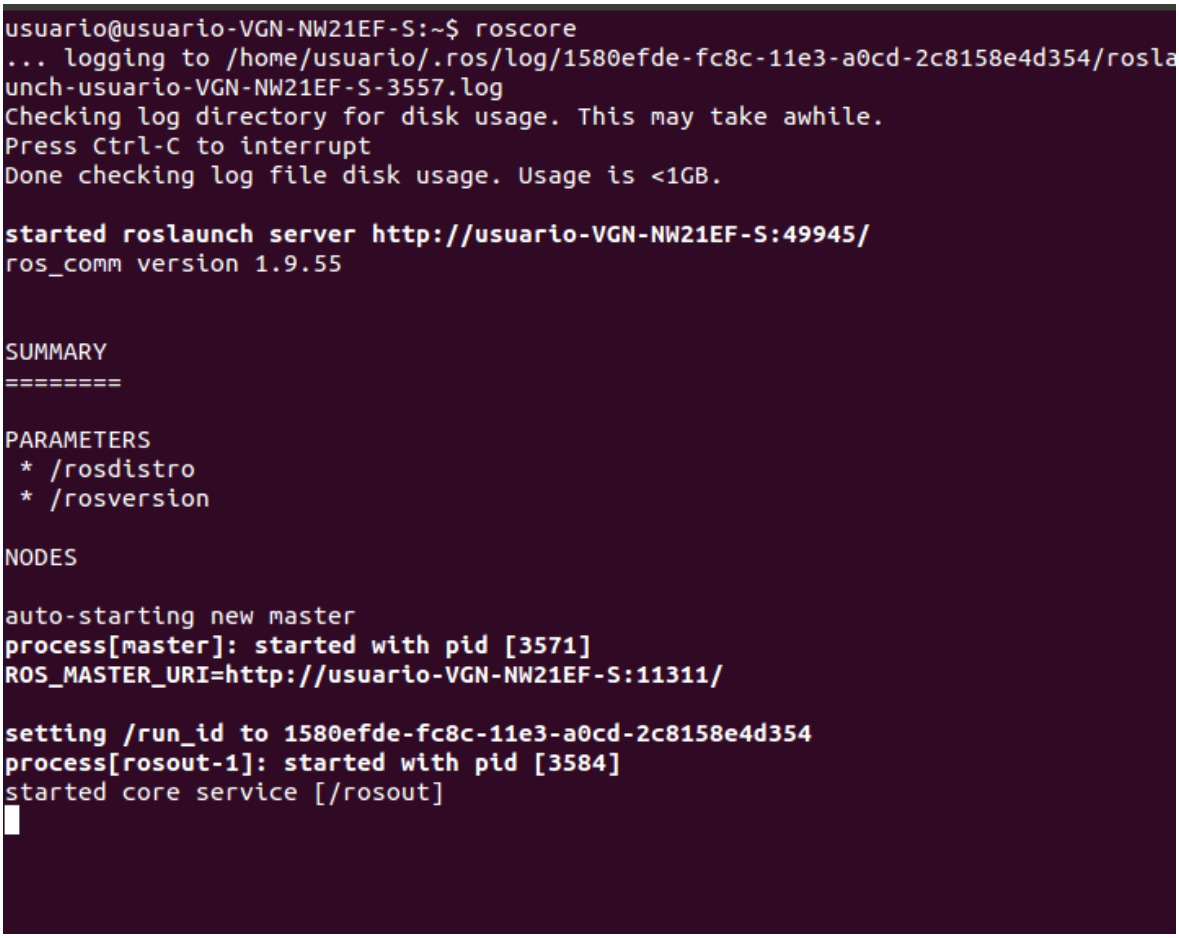

Figura 2.6

Una vez hecho esto, en un nuevo terminal y sobre la carpeta de el catkin_workspace, ejecutamos el programa hokuyoclient, con el comando rosrun. Al ejecutarlo, en el terminal conectado a la Raspberry pi, aparecerá una pregunta acerca de la opción de medidas del láser, escogemos la opción SLAM.

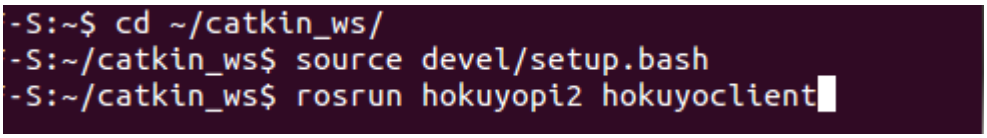

Figura 2.7

Por último, utilizaremos el fichero slam2.launch para lanzar los algoritmos encargados del SLAM, y la interfaz gráfica rviz, con el comando roslaunch.

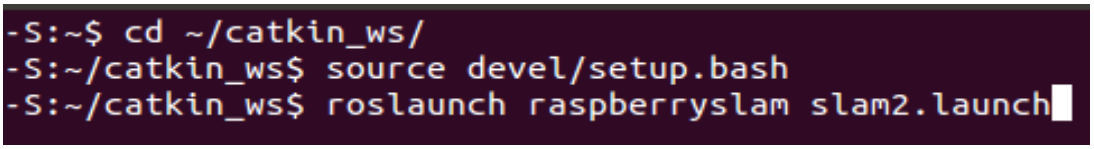

Figura 2.8

Es posible encontrarse con un error, consistente en un fallo al abrir el puerto USB de la Raspberry pi al que está conectado el láser. Esto es debido a que el dispositivo a asignado al puerto un nombre distinto a /dev/ttyACM0, que es el nombre que le debe asignar por defecto, y le ha asignado /dev/ttyACM1 (o 2, 3, 4....). Esto se produce si se conecta y desconecta el láser cuando la Raspberry pi está encendida, la primera vez le asigna /dev/ttyACM0, que es el nombre que espera ProgHok para abrir el puerto, la segunda vez, al estar ocupado el nombre por defecto, cambia el número. Para solucionarlo basta con reiniciar la raspberry con el comando *sudo reboot*. En ese caso se deberá volver a establecer la conexión ssh

IV. PLIEGO DE CONDICIONES

Este apartado tiene como objetivo enunciar las condiciones generales facultativas y técnicas que regularán la correcta realización de este Trabajo Final de Grado.

1. Condiciones Generales Facultativas.

- **Promotor del proyecto**

El presente documente tiene como intención la realización de un Trabajo Final de Grado, cuyo promotor indirecto es la Escuela Técnica Superior de Ingenieros Industriales de la UPV, y cuyo promotor directo es D Pedro García Gil, que actúa como representante del Departamento de Ingeniería de Sistemas y Automática de la UPV, así como del ai2 (Instituto de Automática e Informática Industrial)

- **Obligaciones y derechos del proyectista**

El proyectista está suscrito a una serie de obligaciones y derechos que se enumerarán a continuación.

A saber, obligaciones:

1. Cumplir con la legislación vigente. Respetar licencias de software y derechos de autor.
2. Llevar a cabo el proyecto según las indicaciones efectuadas por el promotor del proyecto.
3. Cumplir con la normativa vigente en la Escuela Técnica Superior de Ingenieros Industriales de la UPV.
4. Consultar con el promotor del proyecto cualquier modificación de las especificaciones técnicas, así como proponer soluciones alternativas a los problemas que puedan surgir.
5. Informar periódicamente al promotor acerca del estado del proyecto.

Y derechos:

1. Posibilidad de realizar pruebas y ensayos con la plataforma y el hardware necesario para ello.
2. Disponer de un equipo adecuado para la realización del proyecto.
3. Recibir soporte técnico para cualquier problema que pudiera surgir.

4. En caso de ausencia del promotor, el proyectista tendrá plena potestad en la toma de decisiones relativa al proyecto, que deberá ser asumida por el promotor.

- **Condiciones generales de la ejecución del proyecto.**

El inicio del proyecto será indicado por el promotor del mismo. En caso de que surjan discrepancias sobre esta fecha, se tomará como fecha de inicio la fecha en que el proyecto fue adjudicado al proyectista.

El ritmo de los trabajos será fijado por ambas partes, siempre siguiendo las disponibilidades de tiempo justificadas por el proyectista. El plazo de entrega y las condiciones generales serán establecidos de mutuo acuerdo entre proyectista y promotor. Dentro de los límites impuestos por la ETSII y su normativa vigente.

- **Ensayos.**

Para comprobar el correcto funcionamiento del software desarrollado, deberá facilitarse los dispositivos e instalaciones necesarias por parte del promotor, para la realización de pruebas y ensayos.

- **Condiciones económicas.**

Por las características de un Trabajo Final de Grado, no se prevé pago alguno, de ninguna cantidad económica en concepto de honorarios.

- **Condiciones legales.**

Aunque ninguna ley tiene repercusión directa sobre el sistema desarrollado en este trabajo, salvo aquellas relacionadas con licencias de software y derechos de autor ya mencionados. Se debe señalar que el vuelo o utilización de cualquier dron o quadrotor al que se pretenda implementar este sistema debe seguir y respetar la Ley 48/1960 sobre Navegación Aérea, así como tener la autorización pertinente de la Agencia Estatal de Seguridad Aérea, dependiente del Ministerio de Fomento.

2. Condiciones Generales de Índole Técnico.

A continuación se detallan las características técnicas de los elementos físicos y las herramientas software utilizadas durante el desarrollo de este proyecto. Para la reproducción de este proyecto deben utilizarse equipos y/o recursos iguales o similares.

2.1. Equipos Físicos.

- **Sensor láser Hokuyo URG-04LX-UG01.**

Dimensiones	50x50x70 mm (largo-ancho-altura)
Masa	160 g
Tensión de funcionamiento	5 V ±5% (USB Bus Power)
Intensidad de funcionamiento	500 mA
Intensidad de arranque	750-850 mA
Frecuencia de funcionamiento	10 Hz
Rango de detección	10-5000 mm
Resolución	1 mm
Resolución angular	0.36º
Angulo de barrido	240º
Tiempo de barrido	100 ms/scan
Fuente láser	Diodo láser semiconductor (λ=785 nm)
Interfaz	USB 2.0 FS mode (12Mbps)
Precisión	Distancia 10mm –1000mm: ±30mm Distancia 10mm – 5000mm: ±3% de la medida

Tabla 2.1 Especificaciones técnicas sensor Hokuyo URG-04LX-UG01.

- **Raspberry pi Modelo B.**

SoC	Broadcom BCM2835 (CPU + GPU +DSP + SDRAM + Puerto USB)
CPU	ARM 1176JZF –S a 700 MHz
Juego de instrucciones	RISC de 32 bits
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEF-2 y VC-1, 1080p30 H.264/MPEG-4 AVC
Memoria (SDRAM)	512 MiB (Compartidos con la GPU)
Almacenamiento integrado	SD / MMC / ranura para SDIO
Periféricos de bajo nivel	8x GPIO, SPI, UART
Consumo energético	700 mA (3.5 W)
Alimentación	5 V vía Micro USB o GPIO header
Dimensiones	85.6 mm x 54 mm (largo – ancho)
Masa	5 g
Sistemas operativos soportados	Raspbian (Debian), Pidora (Fedora), Arch Linux ARM, RISC OS

Tabla 2.2 Especificaciones técnicas Raspberry pi Modelo B.

- **Ordenador Sony – Vaio VGN – NW21EF.**

Procesador	Intel Pentium Dual – Core T4300
Frecuencia de reloj del procesador	2.1 GHz
Chipset	Intel GM45
Memoria	4 GB DDR2
Frecuencia de reloj de la memoria	800 MHz
GPU	ATI Mobility Radeon HD 4570
WiFi Standard	802.11n
Alimentación	19.5 V DC, 4.7 A.

Tabla 2.3 Especificaciones técnica Sony – Vaio VGN – NW21EF.

- **Dongle WiFi 802.11N**

Wireless standards	IEEE 802.11n/g/b
Interfaz	USB 2.0/1.1 Alta velocidad
Velocidad de envío	802.11n : 150 Mbps 802.11g: 54/48/36/24/18/12/9/6 Mbps 802.11b: 11/5.5/2/1 Mbps
Chipset	Ralink RT5370
Banda de frecuencia	2.4 GHz ISM Band
Seguridad	64/128 bit encryption WPA, WPA2, WPA – PSK, WPA2 – PSK. TKIP/AES
Masa	5 g
Radio channel	1 – 14 canales (Universal Domain Selection)

Tabla 2.4 Especificaciones técnica dongle WiFi

2.2. Software.

- Sistema operativo Ubuntu 12.10.
- Sistema operativo Windows 7.
- Procesadores de textos (OpenOffice4.2, Microsoft Word 2007, Gedit, Nano).
- Framework ROS – Groovy (con instalación completa)
- Metapackage hector_slam
  (hector_compressed_map_transport, hector_geotiff, hector_geotiff_plugins, hector_imu_attitude_to_tf, hector_map_server, hector_map_tools, hector_mapping, hector_marker_drawing, hector_nav_msgs, hector_slam_launch, hector_trajectory_server).
- Sistema operativo Raspbian (Debian 7.0).
- IDE Code::Blocks (GNU/Linux) Versión 10.05.
- Compiladores gcc, g++, gcc ARM gnueabi, g++ ARM gnueabi.
- Debugger gdb.

V. PRESUPUESTO

El presupuesto es el documento que refleja el coste del proyecto, la inversión necesaria para llevarlo a cabo (Gómez – Senent y otros, 2000 [14]).

En este apartado del trabajo se hará una estimación del coste total que supone la ejecución del mismo. Se ha dividido en cuatro capítulos. A saber, estudio preliminar y documentación, diseño de software y programación informática, implementación, y pruebas y ensayos.

- CAP01 Estudio preliminar, documentación y redacción.

Este capítulo está referido a al tiempo dedicado a la búsqueda de información útil con objeto de ser utilizada durante la realización del proyecto, la comprensión y documentación acerca de conceptos nuevos, el repaso de conceptos ya conocidos, así como la redacción del documento que se presenta. Se contemplará la mano de obra necesaria para llevar a cabo dichas tareas, así como los recursos empleados. En este capítulo se ha definido una unidad de obra correspondiente a la tarea de documentación y búsqueda de información durante una hora (UO01), y otra correspondiente a una hora de redacción (UO02).

- CAP02 Diseño de software y programación informática.

En este capítulo se contemplan las labores de diseño de software y programación informática, como bien indica su nombre, necesarias para llevar a cabo el presente trabajo. Se contemplan tanto la mano de obra, como los materiales software y hardware necesarios para tal efecto. Para este capítulo definimos una única unidad de obra, correspondiente a una hora de desarrollo de software y programación informática (UO03).

- CAP03 Implementación.

En este capítulo del presupuesto se representa el coste económico de llevar a cabo la implementación física del software y programas informáticos previamente diseñados. En este capítulo se define como unidad de obra, el coste de llevar a cabo una hora de implementación del sistema diseñado (UO04)

- CAP04 Pruebas y ensayos.

Reflejará el coste económico que va a asociado con la realización de pruebas y ensayos sobre el sistema implementado, con el fin de obtener resultados acerca de su funcionamiento y en caso necesario realizar los cambios oportunos para mejorarlo. Para este capítulo, se utilizará como unidad de obra, el coste de una hora de pruebas y ensayos para el sistema diseñado (UO05); teniendo en cuenta que la mitad del tiempo se han realizado en vuelo real, y la otra mitad no. Para el coste de utilizar un quadrotor para las

pruebas en vuelo real durante una hora, se ha supuesto un 0,05% del presupuesto calculado para el quadrotor utilizado que asciende a 46.370 € (Rodenas, 2013 [15]).

1. Capítulos y Unidades de Obra.

- **CAP01 Estudio preliminar, documentación y redacción.**

UO01

Descripción	Clase	Rendimiento	Precio	Importe
Graduado en Ingeniería de Tecnologías Industriales	Mano de obra	1	40 €/h	40 €
Laptop Sony – Vaio	Maquinaria	1	2,25 €/h	2,25 €
Costes Directos Complementarios	Medio auxiliar	2%		0,85 €
			Precio Unitario	**43,1 €**

UO02

Descripción	Clase	Rendimiento	Precio	Importe
Mecanografiado	Mano de obra	1	10 €/h	10 €
Laptop Sony – Vaio	Maquinaria	1	2,25 €/h	2,25 €
Microsoft Office 2007	Recurso	0,033	120 €	4 €
Costes Directos Complementarios	Medio auxiliar	2%		0,33 €
			Precio Unitario	**16,58 €**

- **CAP02 Diseño de software y programación informática.**

UO03

Descripción	Clase	Rendimiento	Precio	Importe
Graduado en Ingeniería de Tecnologías Industriales	Mano de obra	1	40 €/h	40 €
Laptop Sony – Vaio	Maquinaria	1	2,25 €/h	2,25
ROS – groovy	Recurso	-	0 €	0 €
Code::Blocks	Recurso	-	0 €	0 €
Editor Gedit	Recurso	-	0 €	0 €
Editor Nano	Recurso	-	0 €	0 €
SO Linux/Ubuntu 12.01	Recurso	-	0 €	0 €
Costes directos complementarios	Medio auxiliar	2%		0,85 €
			Precio Unitario	**43,1 €**

- **CAP03 Implementación.**

UO04

Descripción	Clase	Rendimiento	Precio	Importe
Graduado en Ingeniería de tecnologías Industriales	Mano de obra	1	40 €/h	40 €
Hokuyo –URG – 04LX – UG01	Material	0,01667	1.107 €	18,45 €
Raspberry Pi Modelo B	Material	0,01667	31,69 €	0,53 €
Cable USB – Micro USB	Material	0,01667	1,83 €	0,03 €
Tarjeta Micro SD 8 GB Class 4	Material	0,01667	7,35 €	0,12 €
Dongle WiFi 802.11n	Material	0,01667	12,50 €	0,21 €
Compilador gcc/g++ ARM gnueabi	Recurso	-	0 €	0 €
Costes directos complementarios	Medio auxiliar	2%		1,19 €
			Precio Unitario	**60,53 €**

- **CAP04 Pruebas y ensayos.**

UO05

Descripción	Clase	Rendimiento	Precio	Coste
Graduado en Ingeniería de Tecnologías Industriales	Mano de obra	1	40 €/h	40 €
Técnico de laboratorio	Mano de obra	0,5	20 €/h	10 €
Quadrotor	Maquinaria	0,5	23,19 €/h	11,60 €
Laptop Sony – Vaio	Maquinaria	1	2,25 €/h	2,25 €
Costes directos complementarios	Medio auxiliar	2%		1,15 €
			Precio Unitario	**65 €**

2. Presupuesto de Ejecución Material.

Estado de las mediciones.

Para las mediciones de las distintas unidades de obra, se ha utilizado el tiempo utilizado para llevar a cabo la labor que describen, en horas; como viene siendo habitual en proyectos de software e informática. Así bien, las mediciones de las distintas unidades de obra quedan de la siguiente forma

- Medición para UO01, 40 h
- Medición para UO02, 30 h
- Medición para UO03, 200 h
- Medición para UO04, 60 h
- Medición para UO05, 20 h

Capitulo	Importe
CAP01	2.221,4 €
CAP02	8.620 €
CAP03	3.631,8 €
CAP04	1.300 €
Total Ejecución Material	**15.773,2 €**

3. Gastos Generales y Beneficio Industrial.

Este punto abarca los gastos generados por las instalaciones utilizadas durante la realización de este trabajo, más el beneficio industrial. A efectos de cálculo se estima el apunte de los gastos generales como el 17%, y el del beneficio industrial como un 6% del presupuesto de ejecución material del proyecto.

El valor de los gastos generales asciende a 2.681,44 €.

El valor del beneficio industrial asciende a 946,39 €.

Gastos Generales + Beneficio Industrial……**3.627,83 €**

4. Honorarios de Dirección.

Se han calculado los honorarios por dirección, como el 7% del presupuesto de ejecución material.

Honorarios por dirección……...**1.104,12 €**

5. Presupuesto de Inversión.

Este concepto incluye el presupuesto de ejecución material, los gastos generales, el beneficio industrial y los honorarios por dirección.

Ejecución material	15.773,2 €
Gastos generales	2.681,44 €
Beneficio industrial	946,39 €
Honorarios dirección	1.104,12 €
TOTAL	**20.505,15 €**

6. Presupuesto Base de Licitación.

Presupuesto de inversión	20.505,15 €
IVA (21%)	4.306,08 €
TOTAL	**24.811,23 €**

VI. BIBLIOGRAFÍA

[1] Bechrach, He, Prentice, Roy.(MIT). *Robust Autonomous Navegation in GPS – denied Environments*. (2011) Journal of Field Robotics (JFR 2011).

[2] Kohlbrecher, Meyer, Graber, Petersen, von Stryk, Klingauf. *Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots.* (2013).

[3] Sitio web de ROS "http:// www.ros.org/about-ros"

[4] Hokuyo Automatic CO. LTD. *Communication Protocol Specefication For SCIP2.0 Standard* (2006).

[5] Sitio web de Raspberry pi "http://www.raspberrypi.org/help/faqs/"

[6] Aubrey, Kabir. *Data Movement Between Big – Endian and Little – Endian Devices*. (2008) Freesclase Semiconductor, application note (rev 2.2, 3/2008).

[7] I. Sanz Alonso. *Generación de trayectorias con Player para la navegación de un robot móvil en espacios inteligentes* (2008). Proyecto Final de Carrera. Universidad de Alcalá.

[8] West, Syrnos. *Robust Stochastic Mapping towards the SLAM Problem* (2006). IEEE International Conference on Robotics and Automation.

[9] Riisgaard, Rufus Blas. *A Tutorial Approach to Simultaneous Localization and Mapping* (2005). MIT Cognitive Robotics Spring 2005.

[10] Neira. *The State of the Art in the EKF Solution to SLAM* (2007). Robotics Universidad de Zaragoza.

[11] Sitio web http://www.chuidiang.com/clinux/sockets/udp/udp.php.

[12] Alonso Jordá, García Granada, Onaindía de la Rivaherrera. *Diseño e Implementación de Programas en Lenguaje C* (1998). Serv. Publicaciones UPV – 98.367

[13] Blasco Ferragud, Martínez Iranzo, Senent Español, Sanchis Sáez. *Sistemas Automáticos* (2000). Serv. Publicaciones UPV – 2000.4186

[14] Gómez – Senent Martínez, Sánchez Romero, González Cruz. *Cuadernos de Ingeniería de Proyectos II. Del Diseño de Detalle a la Realización* (2000). Serv. Publicaciones UPV - 2000.989

[15] L. Rodenas Lorda. *Plataforma de desarrollo para el control de estabilidad en tiempo real de un vehículo aéreo del tipo quadrotor* (2013). Proyecto Final de Carrera. UPV - ETSID.

[16] AESA. *El Uso de los Drones en España*. (2014) Comunicado Oficial.

VII. ANEXO

En este documento se recogen los códigos en C++ redactados para la realización de este trabajo, así como los ficheros CMake, *.xml necesarios para realizar las compilaciones para ROS y los ficheros *.launch y *.rviz necesarios para poner en marcha el sistema de SLAM.

1. Programa HokuyoClient.

Este es el programa encargado de recibir vía socket UDP las medidas aportadas por el láser, inicializar el sistema de comunicación intermodal de ROS y establecerse como Publisher del topic /scan, y por último enviar por dicho topic las medidas realizadas por el LIDAR en formato sensor_msg/LaserScan.

```
#include <stdio.h>

#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include <sstream>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main(int argc, char **argv)
{
//!!!!!!!!!!!!!!!SOCKET!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
int fd_socket, flow=3;
float buff[MAX_READINGS+1];
struct sockaddr_in bs;
struct sockaddr_in server;
unsigned int longserv = sizeof (server);
server.sin_family = AF_INET;
server.sin_port = htons(15555);
char ip[] = "10.0.0.200";
server.sin_addr.s_addr = inet_addr(ip);
bs.sin_family = AF_INET;
bs.sin_port = htons(0);
bs.sin_addr.s_addr = htonl (INADDR_ANY);
fd_socket = socket (AF_INET, SOCK_DGRAM, 0);
bind (fd_socket, (struct sockaddr *)&bs,sizeof (bs));
sendto(fd_socket, (char *)&flow, sizeof(flow),0,(struct sockaddr *)&server,
longserv);
```

```
    //!////////////////////////////FIN SOCKET////////////////////////////////////
    ros::init(argc, argv, "HokuyoClient");
    ros::NodeHandle n;
    ros::Publisher chatter_pub = n.advertise<sensor_msgs::LaserScan>("scan",
1000);
    ros::Rate loop_rate(10);
    printf ("Drivers Hokuyo 04-LX para SLAM en Raspberry pi funcionando... \n");
    int idx = 0;
    while (ros::ok())
    {
    sensor_msgs::LaserScan msg;
    int i=44,j=0;
    float angle_min=-2.08621382713, angle_max=2.08621382713,
angle_increment=0.006135923152, time_increment=9.76562514552e-05,
scan_time=0.10000000149, range_min=0.0, range_max=5.600, ranges[769];
    //!//Rellenamos mensaje ROS////////
    msg.header.frame_id = "laser";
    msg.header.seq = idx;
    msg.header.stamp=ros::Time::now();
    msg.angle_min = angle_min;
    msg.angle_max = angle_max;
    msg.angle_increment = angle_increment;
    msg.time_increment = time_increment;
    msg.scan_time = scan_time;
    msg.range_min = range_min;
    msg.range_max = range_max;
    //!///Recibimos medidas láser via socket///////
    recvfrom (fd_socket, (char *)&buff, sizeof(buff), 0, (struct
sockaddr*)&server, &longserv);
    msg.ranges.clear();
    for (int h=0; h<MAX_READINGS;h++){
    ranges[h] =buff[h];
    msg.ranges.push_back((float)ranges[h]);
    }
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    idx++;
    }
    return 0;
    }
```

2. Programa ProgHok.

Este es el programa compilado para arquitectura ARM, que tiene como objetivo actuar como driver del LIDAR, configurarlo, enviarle el comando MD para funcionamiento continuo, y almacenar sus medidas en una variable vector.

Si es usado para SLAM, también enviará dichas medidas por comunicación socket UDP, y si es usado para poner en funcionamiento el algoritmo de control de posición x-y, tomará las medidas en haces según los ejes de dirección, como se ha explicado anteriormente.

Consta de los siguientes ficheros, main.cc, ProgHok.cc, ProgHok.h, urg_laser.cc y urg_laser.h

2.1. main.cc

```
#include <iostream>

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "../Headers/urg_laser.h"
#include "../Headers/ProgHok.h"
using namespace std;
int main(void)
{
//!////////////////////////////////////SERVER SOCKET/////////////////
int fd_socket, comp;
struct sockaddr_in bs;
struct sockaddr_in cliente;
unsigned int longclient = sizeof(cliente);
float buffer;
bs.sin_family = AF_INET;
bs.sin_port = htons(15555);
bs.sin_addr.s_addr = htonl (INADDR_ANY);
fd_socket = socket (AF_INET, SOCK_DGRAM, 0);
bind (fd_socket, (struct sockaddr *)&bs,sizeof (bs));
recvfrom (fd_socket, (char *)&buffer, sizeof(buffer), 0, (struct sockaddr
*)&cliente, &longclient);
comp=buffer;
comp++;
//!////////////////////////////FIN SOCKET/////////////////////////////////
ProgHok *ProgHok_1;
int choice;
//Construye Objeto Hokuyo
ProgHok_1=new ProgHok();
printf("Press 1 for axis measurements, or press 2 for SLAM measurements \n");
scanf ("%d",&choice);
int i=0;
if (choice==1){
while (i<30){
```

```
    //Toma medidas ejes
    ProgHok_1->measures();
    i++;
    };
    };
    if (choice==2){
    int j=0;
    printf ("SLAM running....\n");
    while (1){
    ProgHok_1->measuresforSLAM();
    sendto(fd_socket, (char *)&(ProgHok_1->datlaser), sizeof((ProgHok_1-
>datlaser)), 0, (struct sockaddr *)&cliente, longclient);}
    j++;
    };
    return 0;
    }
```

2.2. ProgHok.cc

```
    #include "../Headers/ProgHok.h"

    #include <stdio.h>
    #include <math.h>
    #include <assert.h>
    #include <stdio.h>
    #include <string.h>
    #include <unistd.h>
    #include <errno.h>
    #include <stdlib.h>
    #include <fcntl.h>
    ProgHok::ProgHok()
    {
    int error=0;
    Hokuyo = new urg_laser();
    error=Hokuyo->Setup();
    if (error==-1){
    printf("Error al realizar el Setup \n");
    }
    else{
    printf("Setup realizado con exito \n");
    }
    };
    ////////////////////////////////////////////////////////////////////////
    ProgHok::~ProgHok(){
    if (Hokuyo->Shutdown()==0){
    printf("Puerto cerrado, objeto Hokuyo destruido \n");
    Hokuyo->~urg_laser();
    delete Hokuyo;}
    fclose(archivo);
    };
    //Función para datos-medidas
    ////////////////////////////////////////////////////////////////////////
    void ProgHok::measures(void){
    //Inicializamos variables de medida
```

```
    Hokuyo->ReadLaser();
    //Almacenamos medidas del laser en variables de medida
    medida_Ypos=(Hokuyo->Small_Data[0]+Hokuyo->Small_Data[0])/2;
    medida_Yneg=(Hokuyo->Small_Data[4]+Hokuyo->Small_Data[4])/2;
    medida_X=(Hokuyo->Small_Data[2]+Hokuyo->Small_Data[2])/2;
    Min= MinorDist(medida_Ypos, medida_Yneg, medida_X);
    Max= MaxDist (medida_Ypos, medida_Yneg, medida_X);
    fprintf(archivo, "Archivo escrito \n");
    printf("MedidaY %lf MedidaY- %lf MedidaX %lf \n",medida_Ypos,medida_Yneg,
medida_X);
    }
    ////////////////////////////////////////////////////////////////////////
    void ProgHok::measuresforSLAM(void){
    int i=0;
    Hokuyo->ReadforSLAM();
    for (i=0;i<769;i++){
    datlaser[i]=(Hokuyo->SLAM_Data[i]);
    }
    }
    ////////////////////////////////////////////////////////////////////////
    double ProgHok::MinorDist (double medida_Ypos, double medida_Yneg, double
medida_X){
    if (medida_X<medida_Ypos)
    medidamenor=medida_X;
    else
    medidamenor=medida_Ypos;
    if (medida_Yneg<medidamenor)
    medidamenor=medida_Yneg;
    else
    medidamenor=medidamenor;
    return (medidamenor);
    }
    ////////////////////////////////////////////////////////////////////////
    double ProgHok::MaxDist (double medida_Ypos, double medida_Yneg, double
medida_X){
    if (medida_X>medida_Ypos)
    medidamayor=medida_X;
    else
    medidamayor=medida_Ypos;
    if (medida_Yneg>medidamayor)
    medidamayor=medida_Yneg;
    else
    medidamayor=medidamayor;
    return (medidamayor);
    }
```

2.3. ProgHok.h

```
#include <assert.h>

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <stdio.h>
#include "../Headers/urg_laser.h"
#ifndef PROGHOK_H_INCLUDED
#define PROGHOK_H_INCLUDED
class ProgHok{
public:
//Objeto Hokuyo
urg_laser *Hokuyo;
FILE *archivo;
double medida_Ypos,medida_Yneg,medida_X, medidamenor, medidamayor, Min, Max;
urg_laser_readings_t *Readings;
float datlaser[769];
//Operaciones
//constructor
ProgHok();
//Destructor
~ProgHok();
//Medidas
void measures(void);
double MinorDist(double medida_Ypos, double medida_Yneg, double medida_X);
double MaxDist (double medida_Ypos, double medida_Yneg, double medida_X);
void PrintFile (double medida_Ypos, double medida_Yneg, double medida_X);
void measuresforSLAM(void);
};
#endif // PROGHOK_H_INCLUDED
```

2.4. urg_laser.cc

```
#include <stdio.h>

#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <termios.h>
#define MAX_READINGS 769
#include <math.h>
#define RTOD(r) ((r) * 180 / M_PI)
#define DTOR(d) ((d) * M_PI / 180)
#ifndef FRECUENCIA_HOKUYO
#define FRECUENCIA_HOKUYO 20
#endif
#include "../include/urg_laser.h"
int urg_laser::ReadUntil_nthOccurence (int file, int n, char c)
{
int retval = 0;
unsigned char Buffer[2];
Buffer[0] = 0;
Buffer[1] = 0;
for (int i = 0; i < n; i++)
{
do
{
retval = ReadUntil (file, &Buffer[0], 1, -1);
} while (Buffer[0] != c && retval > 0);
}
return retval;
}
//////////////////////////////////////////////////////////////////
int urg_laser::QuerySCIPVersion ()
{
unsigned char Buffer [18];
memset (Buffer, 0, 18);
int file = fileno (laser_port);
tcflush (fileno (laser_port), TCIFLUSH);
fprintf (laser_port, "V\n");
memset (Buffer, 0, 18);
sleep(1);
ReadUntil (file, Buffer, 18, -1);
for(int i=0;i<18;i++) printf("%c \n",Buffer[i]);
if (strncmp ((const char *) Buffer, "V\n0\n", 4) != 0)
{
tcflush (fileno (laser_port), TCIFLUSH);
fprintf (laser_port, "VV\n");
int file = fileno (laser_port);
memset (Buffer, 0, 18);
ReadUntil (file, Buffer, 7, -1);
tcflush (fileno (laser_port), TCIFLUSH);
if (strncmp ((const char *) Buffer, "VV\n00P\n", 7) != 0)
{
printf ("> E: QuerySCIPVersion: Error reading after VV command. Answer: %s\n",
Buffer);
return (-1);
```

```
}
SCIP_Version = 2;
return (0);
}
else
{
ReadUntil_nthOccurence (file, 2, (char)0xa);
memset (Buffer, 0, 18);
ReadUntil (file, Buffer, 5, -1);
if (strncmp ((const char *) Buffer, "FIRM:", 5) != 0)
{
tcflush (fileno (laser_port), TCIFLUSH);
this->SCIP_Version = 3;
this->num_ranges = 1128;
return(0);
}
ReadUntil (file, Buffer, 1, -1);
Buffer[1] = 0;
int firmware = atol ((const char*)Buffer);
ReadUntil_nthOccurence (file, 4, (char)0xa);
if (firmware < 3)
{
SCIP_Version = 1;
return 0; }
else
{
tcflush (fileno (laser_port), TCIFLUSH);
fprintf (laser_port, "SCIP2.0\n");
memset (Buffer, 0, 18);
ReadUntil (file, Buffer, 2, -1);
if (strncmp ((const char *) Buffer, "SC", 2) != 0)
{
SCIP_Version = 1;
return 0;
}
else
{
memset (&Buffer[2], 0, 16);
ReadUntil (file, &Buffer[2], 8, -1);
if (strncmp ((const char *) Buffer, "SCIP2.0\n0\n", 11) != 0)
{
SCIP_Version = 1;
return (0);
}
SCIP_Version = 2;
fprintf (laser_port, "BM\n");
ReadUntil_nthOccurence (file, 3, (char)0xa);
tcflush (fileno (laser_port), TCIFLUSH);
return 0;
}
}
}
}
////////////////////////////////////////////////////////////////////////////////
/
int urg_laser::GetSensorConfig (laser_config_t *cfg)
{
cfg->intensity = 0;
if (SCIP_Version == 1)
```

```
    {
    unsigned char Buffer[10];
    memset (Buffer, 0, 10);
    tcflush (fileno (laser_port), TCIFLUSH);
    fprintf (laser_port, "V\n");
    int file = fileno (laser_port);
    ReadUntil (file, Buffer, 4, -1);
    if (strncmp ((const char *) Buffer, "V\n0\n", 4) != 0)
    {
    printf ("> E: GetSensorConfig: Error reading command result: %s\n", Buffer);
    tcflush (fileno (laser_port), TCIFLUSH);
    return (-1);
    }
    ReadUntil_nthOccurence (file, 2, (char)0xa);
    ReadUntil (file, Buffer, 5, -1);
    if (strncmp ((const char *) Buffer, "FIRM:", 5) == 0)
    {
    ReadUntil (file, Buffer, 1, -1);
    Buffer[1] = 0;
    int firmware = atol ((const char*)Buffer);
    if (firmware < 3)
    {
    ReadUntil_nthOccurence (file, 4, (char)0xa);
    tcflush (fileno (laser_port), TCIFLUSH);
    return (-1);
    }
    }
    ReadUntil_nthOccurence(file, 1, (char)'(');
    ReadUntil_nthOccurence(file, 1, (char)'-');
    int i = 0;
    do
    {
    ReadUntil (file, &Buffer[i], 1, -1);
    } while (Buffer[i++] != '[');
    Buffer[i-1] = 0;
    int max_range = atol((const char*)Buffer);
    ReadUntil_nthOccurence (file, 2, (char)',');
    i = 0;
    do
    {
    ReadUntil(file, &Buffer[i], 1, -1);
    } while (Buffer[i++] != '-');
    Buffer[i-1] = 0;
    int min_i = atol ((const char*)Buffer);
    i = 0;
    do
    {
    ReadUntil (file, &Buffer[i], 1, -1);
    } while (Buffer[i++] != '[');
    Buffer[i-1] = 0;
    int max_i = atol ((const char*)Buffer);
    ReadUntil (file, Buffer, 4, -1);
    if (strncmp ((const char *) Buffer, "step", 4) != 0)
    {
    printf ("> E: GetSensorConfig: Error reading angle_min_idx and angle_max_idx.
Using an older firmware?\n");
    tcflush (fileno (laser_port), TCIFLUSH);
    return (-1);
    }
```

```
    cfg->max_range = max_range / 1000.0;
    cfg->min_angle = (min_i-384)*cfg->resolution;
    cfg->max_angle = (max_i-384)*cfg->resolution;
    printf ("> I: URG-04 specifications: [min_angle, max_angle, resolution,
max_range] = [%f, %f, %f, %f]\n",
    RTOD (cfg->min_angle), RTOD (cfg->max_angle), RTOD (cfg->resolution), cfg-
>max_range);
    tcflush (fileno(laser_port), TCIFLUSH);
    }
    else if(SCIP_Version == 2)
    {
    unsigned char Buffer[10];
    memset (Buffer, 0, 10);
    tcflush (fileno (laser_port), TCIFLUSH);
    fprintf (laser_port, "PP\n");
    int file = fileno (laser_port);
    ReadUntil (file, Buffer,7, -1);
    if (strncmp ((const char *) Buffer, "PP\n00P\n", 7) != 0)
    {
    printf ("> E: GetSensorConfig: Error reading command result: %s\n", Buffer);
    tcflush (fileno (laser_port), TCIFLUSH);
    return (-1);
    }
    int i = 0;
    ReadUntil_nthOccurence (file, 2, (char)0xa);
    ReadUntil_nthOccurence (file, 1, ':');
    do
    {
    ReadUntil (file, &Buffer[i], 1, -1);
    i++;
    } while (Buffer[i-1] != ';');
    Buffer[i-1] = 0;
    cfg->max_range = atol ((const char*)Buffer);
    cfg->max_range /= 1000;
    ReadUntil_nthOccurence (file, 1, ':');
    i = 0;
    do
    {
    ReadUntil (file, &Buffer[i], 1, -1);
    i++;
    } while (Buffer[i-1] != ';');
    Buffer[i-1] = 0;
    cfg->resolution = DTOR (360.0 / atol ((const char*)Buffer));
    ReadUntil_nthOccurence (file, 1, ':');
    i = 0;
    do
    {
    ReadUntil (file, &Buffer[i], 1, -1);
    i++;
    } while (Buffer[i-1] != ';');
    Buffer[i-1] = 0;
    cfg->min_angle = atol ((const char*)Buffer);
    cfg->min_angle -= 384.0;
    cfg->min_angle *= cfg->resolution;
    ReadUntil_nthOccurence (file, 1, ':');
    i=0;
    do
    {
    ReadUntil (file, &Buffer[i], 1, -1);
```

```
    i++;
    } while (Buffer[i-1] != ';');
    Buffer[i-1] = 0;
    cfg->max_angle = atol ((const char*)Buffer);
    cfg->max_angle -= 384.0;
    cfg->max_angle *= cfg->resolution;
    ReadUntil_nthOccurence (file, 4, (char)0xa);
    printf ("> I: URG-04 specifications: [min_angle, max_angle, resolution,
max_range] = [%f, %f, %f, %f]\n",
    RTOD (cfg->min_angle), RTOD (cfg->max_angle), RTOD (cfg->resolution), cfg-
>max_range);
    }
    else
    {
    cfg->min_angle = DTOR(-141.0);
    cfg->max_angle = DTOR(141.0);
    cfg->resolution = DTOR(282.0/1128.0);
    cfg->max_range = 30.0;
    printf ("> I: TOP-URG specifications: [min_angle, max_angle, resolution,
max_range] = [%f, %f, %f, %f]\n",
    RTOD (cfg->min_angle), RTOD (cfg->max_angle), RTOD (cfg->resolution), cfg-
>max_range);
    }
    return (0);
    }
    ////////////////////////////////////////////////////////////////////////////
/
    int urg_laser::ReadUntil (int fd, unsigned char *buf, int len, int timeout)
    {
    int ret;
    int current=0;
    struct pollfd ufd[1];
    int retval;
    ufd[0].fd = fd;
    ufd[0].events = POLLIN;
    do
    {
    if(timeout >= 0)
    {
    if ((retval = poll (ufd, 1, timeout)) < 0)
    {
    perror ("poll():");
    printf ("poll()error");
    return (-1);
    }
    else if (retval == 0)
    {
    puts ("Timed out on read");
    printf ("Timed out on read");
    return (-1);
    }
    }
    ret = read (fd, &buf[current], len-current);
    if (ret < 0)
    return ret;
    current += ret;
    if (current > 2 && current < len && buf[current-2] == '\n' && buf[current-1]
== '\n')
    {
```

```
    puts ("> E: ReadUntil: Got an end of command while waiting for more data, this
is bad.\n");
    printf ("E: ReadUntil: Got an end of command while waiting for more data, this
is bad.\n");
    return (-1);
    }
    } while (current < len);
    return len;
    }
    ////////////////////////////////////////////////////////////////////////////////
/
    urg_laser::urg_laser ()
    {
    SCIP_Version = 1;
    num_ranges = 769;
    laser_port = NULL;
    memset (&Data, 0, sizeof (Data));
    memset (&Conf, 0, sizeof (Conf));
    Readings = new urg_laser_readings_t;
    assert (Readings);
    Conf.min_angle = DTOR (-120);//120
    Conf.max_angle = DTOR (120);
    user_min_angle = Conf.min_angle;
    user_max_angle = Conf.max_angle;
    Conf.resolution = DTOR (360.0/1024.0);
    Conf.max_range = 4.0;
    Conf.range_res = 0.001;
    Conf.intensity = 0;
    BaudRate = B115200;
    Port = (char*)"/dev/ttyACM0";
    UseSerial = 0; //usb
    }
    ////////////////////////////////////////////////////////////////////////////////
/
    urg_laser::~urg_laser ()
    {
    if (PortOpen ())
    fclose (laser_port);
    }
    ////////////////////////////////////////////////////////////////////////////////
/
    int
    urg_laser::ChangeBaud (int curr_baud, int new_baud, int timeout)
    {
    struct termios newtio;
    int fd;
    fd = fileno (laser_port);
    if (tcgetattr (fd, &newtio) < 0)
    {
    perror ("urg_laser::ChangeBaud:tcgetattr():");
    close (fd);
    return (-1);
    }
    cfmakeraw (&newtio);
    cfsetispeed (&newtio, curr_baud);
    cfsetospeed (&newtio, curr_baud);
    if (tcsetattr (fd, TCSAFLUSH, &newtio) < 0 )
    {
    perror ("urg_laser::ChangeBaud:tcsetattr():");
```

```
close (fd);
return (-1);
}
unsigned char buf[17];
memset (buf,0,sizeof (buf));
if (SCIP_Version == 1)
{
buf[0] = 'S';
switch (new_baud)
{
case B19200:
buf[1] = '0';
buf[2] = '1';
buf[3] = '9';
buf[4] = '2';
buf[5] = '0';
buf[6] = '0';
break;
case B57600:
buf[1] = '0';
buf[2] = '5';
buf[3] = '7';
buf[4] = '6';
buf[5] = '0';
buf[6] = '0';
break;
case B115200:
buf[1] = '1';
buf[2] = '1';
buf[3] = '5';
buf[4] = '2';
buf[5] = '0';
buf[6] = '0';
break;
default:
printf ("unknown baud rate %d\n", new_baud);
return (-1);
}
buf[7] = '0';
buf[8] = '0';
buf[9] = '0';
buf[10] = '0';
buf[11] = '0';
buf[12] = '0';
buf[13] = '0';
buf[14] = '\n';
}
else
{
buf[0] = 'S';
buf[1] = 'S';
switch (new_baud)
{
case B19200:
buf[2] = '0';
buf[3] = '1';
buf[4] = '9';
buf[5] = '2';
buf[6] = '0';
```

```
buf[7] = '0';
break;
case B57600:
buf[2] = '0';
buf[3] = '5';
buf[4] = '7';
buf[5] = '6';
buf[6] = '0';
buf[7] = '0';
break;
case B115200:
buf[2] = '1';
buf[3] = '1';
buf[4] = '5';
buf[5] = '2';
buf[6] = '0';
buf[7] = '0';
break;
default:
printf ("unknown baud rate %d\n", new_baud);
return (-1);
}
buf[8] = '\n';
}
fprintf (laser_port, "%s", buf);
memset (buf, 0, sizeof (buf));
int len;
if (((len = ReadUntil (fd, buf, sizeof (buf), timeout)) < 0) ||
(buf[15] != '0'))
{
puts ("failed to change baud rate");
return (-1);
}
else
{
if (tcgetattr (fd, &newtio) < 0)
{
perror ("urg_laser::ChangeBaud:tcgetattr():");
close (fd);
return (-1);
}
cfmakeraw (&newtio);
cfsetispeed (&newtio, new_baud);
cfsetospeed (&newtio, new_baud);
if (tcsetattr (fd, TCSAFLUSH, &newtio) < 0 )
{
perror ("urg_laser::ChangeBaud:tcsetattr():");
close (fd);
return (-1);
}
else
{
usleep (200000);
return (0);
}
}
}
////////////////////////////////////////////////////////////////////////////
/
```

```
int urg_laser::Open (const char * PortName, int use_serial, int baud)
{
if (PortOpen ())
this->Close ();
laser_port = fopen (PortName, "r+");
if (laser_port == NULL)
{
printf ("> E: Open: Failed to open Port: %s error = %d:%s\n",
PortName, errno, strerror (errno));
return (-1);
}
int fd = fileno (laser_port);
if (use_serial)
{
puts ("Trying to connect at 19200");
if (this->ChangeBaud (B19200, baud, 100) != 0)
{
puts ("Trying to connect at 57600");
if (this->ChangeBaud (B57600, baud, 100) != 0)
{
puts ("Trying to connect at 115200");
if (this->ChangeBaud (B115200, baud, 100) != 0)
{
puts ("failed to connect at any baud");
close (fd);
return (-1);
}
}
}
puts ("Successfully changed baud rate");
}
else
{
struct termios newtio;
memset (&newtio, 0, sizeof (newtio));
newtio.c_cflag = CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
newtio.c_oflag = 0;
newtio.c_lflag = ICANON;
tcflush (fd, TCIFLUSH);
tcsetattr (fd, TCSANOW, &newtio);
usleep (200000);
QuerySCIPVersion ();
tcflush (fd, TCIOFLUSH);
}
return (0);
}
//////////////////////////////////////////////////////////////////////////////
/
int
urg_laser::Close ()
{
int retval;
assert (this->laser_port);
tcflush (fileno (this->laser_port), TCIOFLUSH);
retval = fclose (this->laser_port);
this->laser_port = NULL;
return (retval);
}
```

```
    ////////////////////////////////////////////////////////////////////////////
/
    bool urg_laser::PortOpen ()
    {
    return laser_port != NULL;
    }
    ////////////////////////////////////////////////////////////////////////////
/
    int urg_laser::GetReadings (urg_laser_readings_t * readings, int min_i, int
max_i)
    {
    unsigned char Buffer[16]="\0";
    assert (readings);
    if (!PortOpen ())
    return (-3);
    if (SCIP_Version == 1)
    {
    tcflush (fileno (laser_port), TCIFLUSH);
    fprintf (laser_port, "G00076801\n");
    int file = fileno (laser_port);
    ReadUntil (file, Buffer, 10, -1);
    if (strncmp ((const char *) Buffer, "G00076801", 9) != 0)
    {
    printf ("> E: GetReadings: Error reading command result: %s\n", Buffer);
    tcflush (fileno (laser_port), TCIFLUSH);
    return (-1);
    }
    ReadUntil (file, Buffer, 2, -1);
    if (Buffer[0] != '0')
    return Buffer[0] - '0';
    for (int i=0; ; ++i)
    {
    ReadUntil (file, Buffer, 2, -1);
    if (Buffer[0] == '\n' && Buffer[1] == '\n')
    break;
    else if (Buffer[0] == '\n')
    {
    Buffer[0] = Buffer[1];
    if (ReadUntil (file, &Buffer[1], 1, -1) < 0)
    return -1;
    }
    if (i < MAX_READINGS)
    readings->Readings[i] = ((Buffer[0]-0x30) << 6) | (Buffer[1]-0x30);
    else
    printf ("Got too many readings! %d\n",i);
    }
    }
    else if(SCIP_Version == 2)
    {
    tcflush (fileno (laser_port), TCIFLUSH);
    fprintf (laser_port, "MD0000076801001\n");
    int file = fileno (laser_port);
    ReadUntil (file, Buffer, 13, -1);
    if (strncmp ((const char *) Buffer, "MD0000076801001\n", 12) != 0)
    {
    printf ("> E:1 GetReadings: Error reading command result: %s\n", Buffer);
    tcflush (fileno (laser_port), TCIFLUSH);
    return (-1);
    }
```

```
   ReadUntil (file, Buffer, 3, -1);
   Buffer[2] = 0;
   if (Buffer[0] != '0' || Buffer[1] != '0'){
   return (Buffer[0] - '0')*10 + (Buffer[1] - '0');
   };
   ReadUntil_nthOccurence (file, 2, (char)0xa);
   for (int i = 0; ; ++i)
   {
   ReadUntil (file, Buffer, 3, -1);
   if ((Buffer[1] == '\n') && (Buffer[2] == '\n'))
   break;
   else if (Buffer[2] == '\n')
   {
   if (ReadUntil(file, &Buffer[1], 2, -1) < 0)
   return (-1);
   }
   else if (Buffer[0] == '\n')
   {
   if (i <= MAX_READINGS)
   {
   readings->Readings[i - 1] = ((readings->Readings[i - 1] & 0xFFC0) |
(Buffer[1]-0x30));
   Buffer [0] = Buffer [2];
   if (ReadUntil (file, &Buffer[1], 2, -1) < 0)
   return (-1);
   }
   else
   printf ("> E: Got too many readings! %d\n",i);
   }
   else if (Buffer[1] == '\n')
   {
   Buffer[0] = Buffer[2];
   if (ReadUntil (file, &Buffer[1], 2, -1) < 0)
   return (-1);
   }
   if (i < MAX_READINGS)
   {
   readings->Readings[i] = ((Buffer[0]-0x30) << 12) | ((Buffer[1]-0x30) << 6) |
(Buffer[2]-0x30);
   if ((readings->Readings[i] > 5600) && (i >= min_i) && (i <= max_i))
   printf ("> W: [%d] read error: %i is bigger than 5.6 meters\n", i, readings-
>Readings[i]);
   }
   else
   printf ("> E: Got too many readings! %d\n",i);
   }
   }
   else
   {
   tcflush (fileno (laser_port), TCIFLUSH);
   fprintf (laser_port, "GD0000112700\n");
   int file = fileno (laser_port);
   ReadUntil (file, Buffer, 13, -1);
   if (strncmp ((const char *) Buffer, "GD0000112700", 12) != 0)
   {
   printf ("> E: 2GetReadings: Error reading command result: %s\n", Buffer);
   tcflush (fileno (laser_port), TCIFLUSH);
   return (-1);
   }
```

```
   ReadUntil (file, Buffer, 3, -1);
   Buffer[2] = 0;
   if (Buffer[0] != '0' || Buffer[1] != '0')
   return (Buffer[0] - '0')*10 + (Buffer[1] - '0');
   ReadUntil_nthOccurence (file, 2, (char)0xa);
   for (int i = 0; ; ++i)
   {
   ReadUntil (file, Buffer, 3, -1);
   if ((Buffer[1] == '\n') && (Buffer[2] == '\n'))
   break;
   else if (Buffer[2] == '\n')
   {
   if (ReadUntil(file, &Buffer[1], 2, -1) < 0)
   return (-1);
   }
   else if (Buffer[0] == '\n')
   {
   if (i <= MAX_READINGS)
   {
   readings->Readings[i - 1] = ((readings->Readings[i - 1] & 0xFFC0) |
(Buffer[1]-0x30));
   Buffer [0] = Buffer [2];
   if (ReadUntil (file, &Buffer[1], 2, -1) < 0)
   return (-1);
   }
   else
   printf ("> E: Got too many readings! %d\n",i);
   }
   else if (Buffer[1] == '\n')
   {
   Buffer[0] = Buffer[2];
   if (ReadUntil (file, &Buffer[1], 2, -1) < 0)
   return (-1);
   }
   if (i < MAX_READINGS)
   {
   readings->Readings[i] = ((Buffer[0]-0x30) << 12) | ((Buffer[1]-0x30) << 6) |
(Buffer[2]-0x30);
   if(readings->Readings[i] >= 30000)
   readings->Readings[i] = 30000;
   if ((readings->Readings[i] > 30000) && (i >= min_i) && (i <= max_i))
   printf ("> W: [%d] read error: %i is bigger than 30.0 meters\n", i, readings-
>Readings[i]);
   }
   else
   printf ("> E: Got too many readings! %d\n",i);
   }
   }
   return (0);
   }
   ////////////////////////////////////////////////////////////////////////////
   int urg_laser::GetIDInfo ()
   {
   unsigned char Buffer [18];
   memset (Buffer, 0, 18);
   int i;
   int id;
   if (!PortOpen ())
   return -3;
```

```
tcflush (fileno (laser_port), TCIFLUSH);
if (SCIP_Version == 1)
{
fprintf (laser_port, "V\n");
int file = fileno (laser_port);
ReadUntil (file, Buffer, 2, -1);
if (strncmp ((const char *) Buffer, "V", 1) != 0)
{
printf ("> E: GetIDInfo: Error reading command result: %s\n", Buffer);
tcflush (fileno (laser_port), TCIFLUSH);
return (-1);
}
ReadUntil (file, Buffer, 2, -1);
if (Buffer[0] != '0')
return Buffer[0] - '0';
Buffer[0] = 0;
for (i = 0; i < 4; i++)
{
do
{
ReadUntil (file, &Buffer[0], 1, -1);
} while (Buffer[0] != 0xa);
}
ReadUntil (file, Buffer, 6, -1);
for (i = 0; ; i++)
{
ReadUntil (file, &Buffer[i], 1, -1);
if (Buffer[i] == 0xa)
break;
}
id = atol ((const char*)Buffer);
ReadUntil (file, Buffer, 1, -1);
}
else
{
fprintf (laser_port, "VV\n");
int file = fileno (laser_port);
ReadUntil (file, Buffer, 7, -1);
if (strncmp ((const char *) Buffer, "VV\n00P\n", 7) != 0)
{
printf (">E: GetIDInfo: Error reading command result: %s\n", Buffer);
tcflush (fileno (laser_port), TCIFLUSH);
return (-1);
}
Buffer[0] = 0;
for (i = 0; i < 4; i++)
{
do
{
ReadUntil (file, &Buffer[0], 1, -1);
} while (Buffer[0] != 0xa);
}
ReadUntil (file, Buffer, 6, -1);
for (i = 0; ; i++)
{
ReadUntil (file, &Buffer[i], 1, -1);
if (Buffer[i] == ';')
{
Buffer[i] = 0;
```

```
    break;
    }
    }
    id = atol ((const char*)Buffer);
    ReadUntil (file, Buffer, 3, -1);
    }
    return id;
    }
    ////////////////////////////////////////////////////////////////
    int urg_laser::Setup ()
    {
    if (Open(Port, UseSerial, BaudRate) < 0)
    {
    return (-1);
    }
    if (QuerySCIPVersion ()==-1)
    printf ("Fallo ScipVersion \n");
    else
    printf ("ScipVersion funciona bien, y SCIP= %d \n",GetSCIPVersion());
    GetSensorConfig (&Conf);
    int half_idx = GetNumRanges() / 2;
    min_i = 0;
    max_i = 1000;
    if(GetSCIPVersion() < 3)
    {
    if (min_i < URG04_MIN_STEP)
    min_i = URG04_MIN_STEP;
    if (max_i > URG04_MAX_STEP)
    max_i = URG04_MAX_STEP;
    }
    Conf.min_angle = (min_i - half_idx) * Conf.resolution;
    Conf.max_angle = (max_i - half_idx) * Conf.resolution;
    return (0);
    }
    ////////////////////////////////////////////////////////////////
    int urg_laser::Shutdown ()
    {
    delete Readings;
    Close ();
    return (0);
    }
    ////////////////////////////////////////////////////////////////
    void urg_laser::ReadLaser(){
    static int iSend=0;
    Data.min_angle = Conf.min_angle;
    Data.max_angle = Conf.max_angle;
    Data.max_range = Conf.max_range;
    Data.resolution = Conf.resolution;
    Data.ranges_count = (max_i - min_i) + 1;
    Data.intensity_count = 0;
    GetReadings (Readings, min_i, max_i);
    for (unsigned int i = 0; i < Data.ranges_count; ++i)
    {
    Data.ranges[i] = Readings->Readings[i+min_i] < 20 ? (Data.max_range*1000) :
(Readings->Readings[i+min_i]);
    Data.ranges[i] /= 1000.0;
    }
    Small_Data[4]=Data.ranges[581];
    for (unsigned int i=0; i <200;i++)
```

```
    {
    if ((Data.ranges[581+i]< Small_Data[4]) and (Data.ranges[581+i]>0.2))
Small_Data[4]=Data.ranges[581+i];
    }
    Small_Data[0]=Data.ranges[25];
    for (unsigned int i=0; i <200;i++)
    {
    if ((Data.ranges[25+i]< Small_Data[0]) and (Data.ranges[25+i]>0.2))
Small_Data[0]=Data.ranges[25+i];
    }
    Small_Data[2]=Data.ranges[325];
    for (unsigned int i=0; i <200;i++)
    {
    if ((Data.ranges[325+i]< Small_Data[2]) and (Data.ranges[325+i]>0.2))
Small_Data[2]=Data.ranges[325+i];
    }
    if(iSend>=FRECUENCIA_HOKUYO*25){
    iSend=0;
    }
    iSend++;
    }
    ////////////////////////////////////////////////////////////////
    void urg_laser::ReadforSLAM(){
    static int iSend=0;
    Data.min_angle = Conf.min_angle;
    Data.max_angle = Conf.max_angle;
    Data.max_range = Conf.max_range;
    Data.resolution = Conf.resolution;
    Data.ranges_count = (max_i - min_i) + 1;
    Data.intensity_count = 0;
    GetReadings (Readings, min_i, max_i);
    for (unsigned int i = 0; i < Data.ranges_count; ++i)
    {
    Data.ranges[i] = Readings->Readings[i+min_i] < 20 ? (Data.max_range*1000) :
(Readings->Readings[i+min_i]);
    Data.ranges[i] /= 1000.0;
    SLAM_Data[i]=Data.ranges[i];
    }
    if(iSend>=FRECUENCIA_HOKUYO*25){
    iSend=0;
    }
    iSend++;
    }
```

2.5. urg_laser.h

```
ifndef URG_LASERO_H_INCLUDED

#define URG_LASERO_H_INCLUDED
#include <stdio.h>
#define MAX_READINGS 769
#define SMALL_MAX_READINGS 20
#define LASER_MAX_SAMPLES 1024
#define URG04_MIN_STEP 0
#define URG04_MAX_STEP 768
#define SLAM_READINGS 769
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
typedef unsigned int uint32_t;
typedef unsigned char uint8_t;
/** @brief Data: scan (@ref LASER_DATA)
The basic laser data packet. */
typedef struct laser_data_t
{
float min_angle;
float max_angle;
float resolution;
float max_range;
uint32_t ranges_count;
float ranges[LASER_MAX_SAMPLES];
uint32_t intensity_count;
uint8_t intensity[LASER_MAX_SAMPLES];
uint32_t id;
} laser_data_t;
/** @brief Request/reply: Get/set scan properties.
The scan configuration (resolution, aperture, etc) can be queried by
sending a null @ref LASER_REQ_GET_CONFIG request and modified by
sending a @ref LASER_REQ_SET_CONFIG request. In either case, the
current configuration (after attempting any requested modification) will
be returned in the response. Read the documentation for your driver to
determine what configuration values are permissible. */
typedef struct laser_config_t
{
float min_angle;
float max_angle;
float resolution;
float max_range;
float range_res;
uint8_t intensity;
} laser_config_t;
typedef struct urg_laser_readings_t
{
unsigned short Readings[MAX_READINGS];
} urg_laser_readings_t;
class urg_laser
{
public:
urg_laser_readings_t *Readings;
float Small_Data[SMALL_MAX_READINGS];
float SLAM_Data [SLAM_READINGS];
private:
```

```
laser_data_t Data;
laser_config_t Conf;
bool UseSerial;
int BaudRate, min_i, max_i;
float user_min_angle, user_max_angle;
char * Port;
public:
urg_laser();
~urg_laser();
int Open(const char * PortName, int use_serial, int baud);
int Close();
int ChangeBaud(int curr_baud, int new_baud, int timeout);
int ReadUntil(int fd, unsigned char *buf, int len, int timeout);
int ReadUntil_nthOccurence(int file, int n, char c);
bool PortOpen();
int GetReadings (urg_laser_readings_t * readings, int min_i, int max_i);
int GetIDInfo ();
float GetMaxRange ();
int GetSensorConfig (laser_config_t *cfg);
int GetSCIPVersion() { return(this->SCIP_Version); }
int GetNumRanges() { return(this->num_ranges); }
void ReadLaser();
void ReadforSLAM();
int Setup();
int Shutdown();
private:
int QuerySCIPVersion ();
int SCIP_Version;
int num_ranges;
FILE * laser_port;
};
#endif // URG_LASERO_H_INCLUDED
```

3. Ficheros CMake, *.xml, *.launch, *.rviz

Estos son los ficheros necearios para compilar los programas escritos para usar con la interfaz ROS (CMake.txt y hokuyopi2.xml) y los ficheros necesarios para poner en funcionamiento el software encargado del SLAM (slam2.launch y slam_mapping.rviz)

3.1. CMake.txt

Fichero CMake del paquete hokuyopi2, creado expresamente para este proyecto.

```
cmake_minimum_required(VERSION 2.8.3)
project(hokuyopi2)
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  sensor_msgs
  std_msgs
  genmsg
)
## Dependencias para mensajes
 generate_messages(
   DEPENDENCIES
   sensor_msgs   std_msgs
 )
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES hokuyopi2
#  CATKIN_DEPENDS roscpp rospy sensor_msgs std_msgs
#  DEPENDS system_lib
)
include_directories(
 include ${catkin_INCLUDE_DIRS}
)
## Declaración ejecutables
 add_executable(hokuyopi2_node src/hokuyopi2_node.cpp)
 add_executable(hokuyoclient src/hokuyoclient.cpp)



## Dependencias ejecutables


 add_dependencies(hokuyopi2_node hokuyopi2_generate_messages_cpp)
 add_dependencies(hokuyopi2_node ${catkin_EXPORTED_TARGETS})
 target_link_libraries(hokuyopi2_node
   ${catkin_LIBRARIES}
 )
```

```
add_dependencies(hokuyoclient hokuyopi2_generate_messages_cpp)
add_dependencies(hokuyoclient ${catkin_EXPORTED_TARGETS})

target_link_libraries(hokuyoclient
  ${catkin_LIBRARIES}
)
```

***NOTA**: Puede observarse que se declaran ejecutables y dependencias para hokuyopi2_node, que no aparece en este proyecto; se trata de otro programa de prueba escrito para comprender mejor el funcionamiento del sistema de mensajes de ROS y como utilizarlo en combinación con el sensor láser.*

3.2. Hokuyopi2.xml

```
<?xml version="1.0"?>
<package>
  <name>hokuyopi2</name>
  <version>0.0.0</version>
  <description>The hokuyopi2 package. Drivers para recoger medidas del URG Hokuyo 04LX
y convertirlas en LaserScan.msg, para poder usarlo en ROS (hector_mapping)</description>

  <maintainer email="raforfer@etsii.upv.com">usuario</maintainer>

  <license>BSD</license>

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>std_msgs</build_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>std_msgs</run_depend>
</package>
```

3.3. Slam2.launch

Notar que este fichero se apoya en hectormapping.launch y en geotiff_mapper.launch, ambos contenidos en el paquete de software hector_slam.

```
<launch>

<param name="/use_sim_time" value="false"/>ros

<node pkg="rviz" type="rviz" name="rviz"
args="-d $(find raspberryslam)/launch/mapping_slam.rviz"/>

<include file="$(find raspberryslam)/launch/hectormapping.launch"/>

<include file="$(find raspberryslam)/launch/geotiff_mapper.launch">
<arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
</include>

</launch>
```

3.4. mapping_slam.rviz

```
Panels:
  - Class: rviz/Displays
    Help Height: 78
    Name: Displays
    Property Tree Widget:
      Expanded:
        - /Global Options1
        - /Status1
        - /LaserScan1/Autocompute Value Bounds1
      Splitter Ratio: 0.5
    Tree Height: 565
  - Class: rviz/Selection
    Name: Selection
  - Class: rviz/Tool Properties
    Expanded:
      - /2D Pose Estimate1
      - /2D Nav Goal1
      - /Publish Point1
    Name: Tool Properties
    Splitter Ratio: 0.588679
  - Class: rviz/Views
    Expanded:
      - /Current View1
    Name: Views
    Splitter Ratio: 0.5
  - Class: rviz/Time
    Experimental: false
    Name: Time
    SyncMode: 0
    SyncSource: LaserScan
```

```
Visualization Manager:
  Class: ""
  Displays:
    - Alpha: 0.4
      Cell Size: 1
      Class: rviz/Grid
      Color: 160; 160; 164
      Enabled: true
      Line Style:
        Line Width: 0.03
        Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
        X: 0
        Y: 0
        Z: 0
      Plane: XY
      Plane Cell Count: 80
      Reference Frame: <Fixed Frame>
      Value: true
    - Alpha: 1
      Autocompute Intensity Bounds: true
      Autocompute Value Bounds:
        Max Value: 1
        Min Value: -1
        Value: false
      Axis: Z
      Channel Name: intensity
      Class: rviz/LaserScan
      Color: 255; 255; 255
      Color Transformer: AxisColor
      Decay Time: 0
      Enabled: true
      Invert Rainbow: false
      Max Color: 255; 255; 255
      Max Intensity: 4096
      Min Color: 0; 0; 0
      Min Intensity: 0
      Name: LaserScan
      Position Transformer: XYZ
      Queue Size: 10
      Selectable: true
      Size (Pixels): 3
      Size (m): 0.01
      Style: Points
      Topic: /scan
      Use Fixed Frame: true
      Use rainbow: true
      Value: true
    - Alpha: 0.7
      Class: rviz/Map
      Color Scheme: map
      Draw Behind: false
      Enabled: true
      Name: Map
      Topic: /map
      Value: true
    - Alpha: 0.5
      Buffer Length: 1
      Class: rviz/Path
      Color: 255; 0; 0
      Enabled: true
      Name: Path
      Topic: /trajectory
      Value: true
    - Alpha: 1
```

```
    Axes Length: 1
    Axes Radius: 0.1
    Class: rviz/Pose
    Color: 255; 25; 0
    Enabled: true
    Head Length: 0.3
    Head Radius: 0.1
    Name: Pose
    Shaft Length: 1
    Shaft Radius: 0.05
    Shape: Axes
    Topic: /slam_out_pose
    Value: true
  - Class: rviz/TF
    Enabled: true
    Frame Timeout: 15
    Frames:
      All Enabled: true
      base_footprint:
        Value: true
      base_link:
        Value: true
      base_stabilized:
        Value: true
      laser:
        Value: true
      map:
        Value: true
      nav:
        Value: true
      scanmatcher_frame:
        Value: true
    Marker Scale: 1
    Name: TF
    Show Arrows: true
    Show Axes: true
    Show Names: true
    Tree:
      map:
        nav:
          base_footprint:
            {}
          base_link:
            laser:
              {}
          base_stabilized:
            {}
    Update Interval: 0
    Value: true
Enabled: true
Global Options:
  Background Color: 48; 48; 48
  Fixed Frame: map
  Frame Rate: 30
Name: root
Tools:
  - Class: rviz/Interact
    Hide Inactive Objects: true
  - Class: rviz/MoveCamera
  - Class: rviz/Select
  - Class: rviz/FocusCamera
  - Class: rviz/Measure
  - Class: rviz/SetInitialPose
    Topic: /initialpose
  - Class: rviz/SetGoal
    Topic: /move_base_simple/goal
  - Class: rviz/PublishPoint
```

```
        Single click: true
        Topic: /clicked_point
    Value: true
    Views:
      Current:
        Class: rviz/Orbit
        Distance: 10
        Focal Point:
          X: 0.174727
          Y: -1.5029
          Z: 0.988644
        Name: Current View
        Near Clip Distance: 0.01
        Pitch: 0.785398
        Target Frame: base_link
        Value: Orbit (rviz)
        Yaw: 0.785398
      Saved: ~
  Window Geometry:
    Displays:
      collapsed: false
    Height: 846
    Hide Left Dock: false
    Hide Right Dock: false
    QMainWindow State:
000000ff00000000fd0000000400000000000001330000002c4fc0200000008fb0000001200530065006c0065006
300740069006f006e00000001e10000009b0000006400fffffffb0000001e0054006f006f006c00200050007200
6f007000650072007400690065007302000001ed000001df00000185000000a3fb0000001200560069006500770
07300200054006f006f02000001df000002110000018500000122fb000000200054006f006f006c002000500072
006f007000650072007400690065007300320300000288000011d000002210000017afb0000001000440069007
30070006c006100790073010000028000002c4000000dd00ffffffb0000002000730065006c00650063007400
69006f006e002000620075006600660065007202000001380000000aa0000023a00000294fb00000014005700690
064006500530074006500720065006f02000000e6000000d2000003ee0000030bfb0000000c004b0069006e0065
006300740200000186000001060000030c000002610000000100000010f000002c4fc0200000003fb0000001e005
4006f006f006c002000500072006f007000650072007400690065007301000000410000007800000000000000000
fb0000000a005600690065007700730100000028000002c4000000b000ffffffb0000001200530065006c00650
06300740069006f006e010000025a000000b2000000000000000000000000020000049000000a9fc0100000001fb
0000000a005600690065007700730300000004e00000080000002e10000019700000003000004b00000003efc010
0000002fb0000000800540069006d00650100000000000004b0000002f600ffffffb0000000800540069006d00
650100000000000004500000000000000000000000262000002c4000000040000000400000008000000008fc00000
001000000020000000100000000a0054006f006f006c00730100000000ffffffff0000000000000000
    Selection:
      collapsed: false
    Time:
      collapsed: false
    Tool Properties:
      collapsed: false
    Views:
      collapsed: false
    Width: 1200
    X: 135
    Y: 77
```

4. Programación en C++ del algoritmo de control de posición.

```
    x3drefpitch=Board->refpitch+wx;

    x3drefroll=Board->refroll-wyy+wyy1;
    if (kk>50){
    Medida_X=Hokuyo->medida_X;
    Medida_X_f= Medida_X*0.6+Medida_X_f_ant*0.4;//ANTES ERA 0.05
    Medida_X_rate = (Medida_X_f-Medida_X_f_ant)*10;
    Medida_X_rate_f=0.6*Medida_X_rate+Medida_X_rate_f_ant*0.4;
    Medida_X_f_ant=Medida_X_f;
    Medida_X_rate_f_ant=Medida_X_rate_f;
    Medida_X_ant = Medida_X;
    Board->a_29=Medida_X;
    Board->a_30=Medida_X_f;
    Board->a_31=Medida_X_rate_f;
    Medida_Y=Hokuyo->medida_Ypos;
    Medida_Y_f= Medida_Y*0.6+Medida_Y_f_ant*0.4;
    Medida_Y_rate = (Medida_Y_f-Medida_Y_f_ant)*10;
    Medida_Y_rate_f=0.6*Medida_Y_rate+Medida_Y_rate_f_ant*0.4;
    Medida_Y_f_ant=Medida_Y_f;
    Medida_Y_rate_f_ant=Medida_Y_rate_f;
    Medida_Y_ant = Medida_Y;
    Medida_Y1=Hokuyo->medida_Yneg;
    Medida_Y1_f= Medida_Y1*0.6+Medida_Y1_f_ant*0.4;
    Medida_Y1_rate = (Medida_Y1_f-Medida_Y1_f_ant)*10;
    Medida_Y1_rate_f=0.6*Medida_Y1_rate+Medida_Y1_rate_f_ant*0.4;
    Medida_Y1_f_ant=Medida_Y1_f;
    Medida_Y1_rate_f_ant=Medida_Y1_rate_f;
    Medida_Y1_ant = Medida_Y1;
    kk=0;
    }
    kk++;
    if(Medida_X<1) Control_X_activo=1;
    if(Medida_X>1.5) Control_X_activo=0;
    if ( Control_X_activo) {
    wx= saturacion(Board->kpx*(Board->xref-Medida_X_f),20,-20)-saturacion(Board-
>kdx*(Medida_X_rate_f),10,-10);
    }
    else wx=0.0;
    if(Medida_Y<1) Control_Y_activo=1;
    if(Medida_Y>1.5) Control_Y_activo=0;
    if ( Control_Y_activo) {
    wyy= saturacion(Board->kpy*(Board->yref-Medida_Y_f),20,-20)-saturacion(Board-
>kdy*(Medida_Y_rate_f),10,-10);
    }else wyy=0.0;
    //Control en Y1 :RIGTH
    if(Medida_Y1<1) Control_Y1_activo=1;
    if(Medida_Y1>1.5) Control_Y1_activo=0;
    if ( Control_Y1_activo) {
    wyy1= saturacion(Board->kpy*(Board->yref-Medida_Y1_f),20,-20)-saturacion(Board-
>kdy*(Medida_Y1_rate_f),10,-10);
    }else wyy1=0.0;
```