Proyecto Fin de Carrera
Ingeniería Informática

Robot de intervención en túneles

Sergio Romero Pradas

Director: José Luis Villarroel Salcedo

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Septiembre 2010

Robot de intervención en túneles

Resumen

En este proyecto se ha desarrollado una aplicación que permite la telemanipulación de uno o más robots. Dicha aplicación está pensada para que funcione en un entorno hostil como lo es un túnel.

Al tratarse de un trabajo de telemanipulación, se compone de dos aplicaciones. Una se ejecuta en un ordenador portátil, (**host**), y es la manejada por el usuario. Ésta se encarga de mandar las órdenes a los robots, así como también de recibir los datos relevantes de ellos, como la odometría (posición y orientación en la que se encuentran) y el láser (distancia a la que se encuentran los obstáculos alrededor de él). La otra se ejecuta en el **robot**. Es un sistema empotrado, encargada de controlarlo, realizando las operaciones necesarias para satisfacer las órdenes mandadas por la aplicación del portátil.

Entre las funcionalidades que se ejecutan en la aplicación del robot se encuentran: la navegación hasta un objetivo evitando obstáculos, para la cuál se utiliza el algoritmo *Nearness Diagram Navigation (ND)*, la localización en un mapa, utilizando el algoritmo *Simultaneous Localization And Mapping (SLAM)*, la navegación autónoma por un pasillo y la toma de fotografías, las cuales son enviadas a la aplicación del host para que puedan ser visualizadas por el usuario.

Hay que destacar que las funcionalidades del ND, del SLAM y de la navegación autónoma por el pasillo ya estaban implementadas. Por lo que el objetivo del proyecto se ha centrado en darles a estas funcionalidades, una vez comprendido su funcionamiento, una estructura analizable desde el punto de vista de Tiempo Real y adaptarlas para que se pudiesen ejecutar sobre el sistema operativo de Tiempo Real *MaRTE OS*. Para lo cual previamente fué necesario comprender el funcionamiento del mismo, así como la forma de compilar y de ejecutar.

En algunas ocasiones fué obligatorio sumergirse en el SO ya que fué necesario modificar algunos drivers que se incluían con la distribución actual de MaRTE, como por ejemplo la linea serie, la cuál comunica el robot con sus dispositivos.

En total, el PFC consta de tres aplicaciones, las cuales se ejecutarán en tres ordenadores distintos. A las dos anteriormente mencionadas se les une una tercera debido a las restricciones establecidas por MaRTE, esta tercera aplicación, que se ejecutará en otro **portátil**, es la encargada de realizar las fotografías y de enviarlas posteriormente a través de ethernet a la aplicación del robot.

Índice general

Índice de figuras **5**

Índice de tablas **6**

1. Introducción **7**
1.1. Contexto 7
1.2. Puntos claves del proyecto 8
1.2.1. MaRTE OS 8
1.2.2. Robot 9
1.3. Estructura de la memoria 10

2. Análisis **11**
2.1. Análisis y descripción de requisitos 11
2.1.1. MaRTE OS 14
2.1.2. RT-WMP 14
2.2. Diseño estructural 15

3. Diseño solución **17**
3.1. Aplicación del robot 17
3.1.1. Tareas de actualización de datos 17
3.1.2. Tarea comunicaciones 18
3.1.3. Tareas de navegación y localización 18
3.1.4. Tarea de las fotografías 20
3.1.5. Establecimiento de prioridades 20
3.2. Aplicación del host 22
3.3. Aplicación del portátil 22
3.4. Análisis Tiempo Real 23
3.5. Diseño detallado y codificación 25
3.5.1. Modificación de los algoritmos 25
3.5.2. Linea serie 26
3.5.3. Pruebas finales 27

4. Fases del proyecto e hitos temporales **28**
4.1. Hitos temporales 28
4.2. Diagrama de Gantt 29

5. Conclusiones y trabajo futuro **30**
5.1. Aportaciones PFC 30
5.2. Trabajo futuro 30
5.3. Conclusiones 31

A. Robot Pioneer 3-AT **32**
A.1. Sensores 33
A.2. Actuadores 34

B. Análisis problema **36**
B.1. DFD nivel 0 36
B.2. DFD nivel 1 37
B.3. DFD nivel 2 39

C. Arquitectura de la aplicación del robot **42**
C.1. Análisis de tiempos 45
C.1.1. Cálculo de tiempos de bloqueo 45
C.1.2. Análisis del cumplimiento de plazos 46

D. Pruebas **50**
D.1. Pruebas de comunicaciones 50
D.2. Pruebas de integración 51
D.2.1. Problemas con el cambio de contexto 51
D.3. Pruebas de cumplimiento de plazos 51
D.4. Pruebas reales 52

E. Compilación y Ejecución **53**
E.1. Aplicación del portátil 53
E.2. Aplicación del host 53
E.3. Aplicación del robot 55

F. Manual de usuario **57**

G. Nearness Diagram Navigation **60**
G.1. Análisis PND 60
G.2. Análisis RND 61
G.3. Estrategia de navegación 62
G.3.1. Navegación en situaciones de baja seguridad 62

G.3.2. Navegación en situaciones de alta seguridad 62

H. Simultaneous Localization And Mapping **64**
H.1. Extracción de características 66
H.2. Asociación de datos 66
H.3. Estimación y actualización del estado 66

Bibliografía **68**

Índice de figuras

1.1. Arquitectura de MaRTE OS para aplicaciones en Ada y C 9
1.2. Imágen de un robot del laboratorio 9

2.1. Diseño estructural del proyecto 15

3.1. Estructura de las tareas y servidores 24

4.1. Diagrama de Gantt del proyecto 29

A.1. Imágen de un robot del laboratorio 32
A.2. Rangos posibles del láser 33
A.3. Imágenes del GPS y del sónar del robot 35

B.1. DFD nivel 0 37
B.2. DFD nivel 1 38
B.3. DFD nivel 2 40

C.1. Estructura de las tareas y servidores 44

F.1. Pantalla inicial de la aplicación 57
F.2. Imágen de la aplicación con el láser y la foto 59

H.1. Visión general del proceso del SLAM 65

Índice de tablas

2.1. Versión inicial tabla de requisitos 11
2.2. Requisitos del Sistema Operativo Tiempo Real 14
2.3. Tabla de requisitos de la aplicación del portátil 14

3.1. Prioridad de las interrupciones y tareas de la aplicación 21
3.2. Información correspondiente a todas las tareas. Tiempos en ms . . . 23

A.1. Configuraciones posibles del láser SICK LMS 200 33

C.1. Bloqueos de todas las interrupciones y tareas del sistema (ms) . . . 47

Capítulo 1
Introducción

1.1. Contexto

El objetivo de este proyecto es el desarrollo de una aplicación que permita la telemanipulación de uno o varios robots (robot móvil Pioneer 3-AT) en el interior de un túnel, garantizando en todo momento que la aplicación que controla al robot cumpla restricciones de Tiempo Real. El proyecto se ha realizado dentro del grupo de Robótica, Percepción y Tiempo Real del Departamente de Informática e Ingeniería de Sistemas (DIIS).

Actualmente todas las aplicaciones de control de navegación y de localización de los robots realizadas en el grupo se basan en la utilización de una estructura cliente/servidor que se ejecuta sobre Linux llamada **Player/Stage** que permite el acceso a los dispositivos que posee el robot: odometría, laser, gps, etc. Sin embargo, aunque el funcionamiento de esta plataforma es correcto, dado que el sistema operativo sobre el que se ejecuta es un SO de propósito general, no se asegura ningún tipo de cumplimiento de restricciones de Tiempo Real, es decir, no se asegura que las tareas de las que consta la aplicación cumplan los plazos de ejecución previstos, o que las tareas se ejecuten conforme a una prioridad establecida. Por lo que en aplicaciones en el que el porcentaje de uso de CPU es muy alto, puede provocar que las tareas no se ejecuten cuando deben y, por consiguiente, que el movimiento del robot no se realice cuando tiene que hacerse, provocando, por ejemplo, que éste se choque contra un obstáculo.

Para evitar que ocurran estas acciones, se ha decidido realizar una aplicación que sí que cumpla restricciones de Tiempo Real, de manera que se asegura que hechos como el anterior no ocurran. Ya que en el caso de una aplicación de Tiempo Real, siempre y cuando se garantice el **cumplimiento de plazos** (método por el cuál se confirma que todas las tareas que del sistema siempre se ejecutarán dentro de su plazo de respuesta), la tarea de movimiento de los motores del robot se hu-

biera ejecutado en el momento correcto y hubiera detectado el obstáculo a tiempo evitando el choque.

Con todo lo anteriormente descrito, se llega a la conclusión de que en sistemas de control como el que nos encontramos es tan importante que las acciones se hagan *cómo* se deben hacer, como que las acciones se realicen *cuándo* se deben hacer. De ahí el interes de la utilización de un sistema operativo de Tiempo Real, el cual nos va a proporcionar un soporte determinista de la ejecución del programa y nos va a permitir asegurar el correcto funcionamiento de la aplicación.

1.2. Sistema Operativo de propósito general vs Sistema Operativo de Tiempo Real

Antes de empezar a describir en más profundidad el proyecto, es conveniente explicar las diferencias que se pueden encontrar entre un SO de propósito general o convencional y uno de Tiempo Real.

Un **SO convencional** se trata de un SO cuyo principal objetivo es realizar todas las actividades del sistema lo más rápido posible. Interesa que nuestro navegador web se carge pronto, que podamos trabajar con nuestro procesador de texto de una forma fluida, etc. Así como, también interesa que se puedan tener varias aplicaciones corriendo y que el ordenador no por ello vaya más lento.

Sin embargo, el objetivo de un **SO de Tiempo Real** es bien distinto. Estos SO tienen como finalidad prestar atención a los procesos en el momento en el que estos lo requieran, es decir, si una tarea de mayor prioridad que la que está ejecutándose pasa al estado de *preparada*, el sistema operativo le dará inmediatamente la CPU para que pueda ejecutarse, expulsando la tarea de menor prioridad.

Entonces, ¿Por qué no es correcto ejecutar una aplicación con restricciones de Tiempo Real en un SO convencional? La respuesta es simple. Como se acaba de explicar, en un SO convencional el objetivo es proporcionar los resultados lo más rápido posible, pero no asegura que se tengan en el momento exacto tal y como lo hace un sistema de Tiempo Real.

Además, el uso de la CPU por las propias actividades del sistema puede ser bastante considerable, lo que puede provocar que estas tareas del sistema, las cuales tienen la máxima prioridad y no pueden ser expulsadas por ningún programa de usuario, retrasen la ejecución de una tarea de nuestra aplicación de Tiempo Real.

Lo que puede provocar que la tarea no se llegue a ejecutar dentro de su periodo, aunque el análisis de cumplimiento de plazos lo asegurase.

Debido a las características especiales que tiene un Sitema Operativo de Tiempo Real, su utilización se restringe a ámbitos especializados, como puede ser los **sistemas empotrados**. En los cuales el ordenador que controla un dispositivo se encuentra dentro del mismo, como es el caso de este proyecto, en el cual la aplicación que controla el robot se ejecuta dentro del robot.

Los sistemas empotrados son muy diversos, ya que van desde pequeños dispositivos como puede ser los teléfonos móviles, pasándo por otros más grandes como electrodomésticos y llegando a controlar grandes aparatos como son los de aeronaútica: aviones, helicópteros, etc.

1.3. Puntos claves del proyecto

Este proyecto está enmarcado dentro del Tiempo Real y de la robótica. Llegando al máximo punto de ello con la existencia de un sistema operativo de tiempo real como MaRTE OS, y la utilización de un robot del laboratorio respectivamente.

Estos dos aspectos van a converger en una aplicación que se ejecutará en el **robot** y tendrá **restricciones de Tiempo Real**.

1.3.1. MaRTE OS

MaRTE OS (Minimal Real Time Operating System for Embedded Aplication [1]) es un sistema operativo de Tiempo Real estricto especialmente indicado para la ejecución de aplicaciones empotradas, como es nuestro caso.

Este sistema operativo está basado en un subconjunto del estándar POSIX.13.1 para Tiempo Real, lo que nos proporciona soporte de Tiempo Real para todas las funcionalidades que se puedan necesitar: gestión de threads, planificación por prioridad, control de mutex y semáforos, etc.

Además, como sistema operativo de Tiempo Real que es, tiene un comportamiento determinista, es decir, en situaciones idénticas el resultado va a ser el mismo, y todos sus servicios tienen un tiempo de respuesta límite. Estas dos condiciones nos van a permitir garantizar que si el análisis de cumplimiento de plazos asegura que todas las tareas se ejecutan en tiempo efectivamente será así.

MaRTE OS permite la compilación de aplicaciones escritas en Ada y en C. En la Figura 1.1 se puede observar la arquitectura del sistema operativo MaRTE dependiendo del lenguaje en el que este escrita la aplicación. Como se puede observar en ambos casos, al mismo nivel que el nucleo de MaRTE se encuentran los drivers de dispositivos, punto muy importante para un sistema empotrado de Tiempo Real, ya que es imprescindible que se permita el acceso a los dispositivos disponibles.

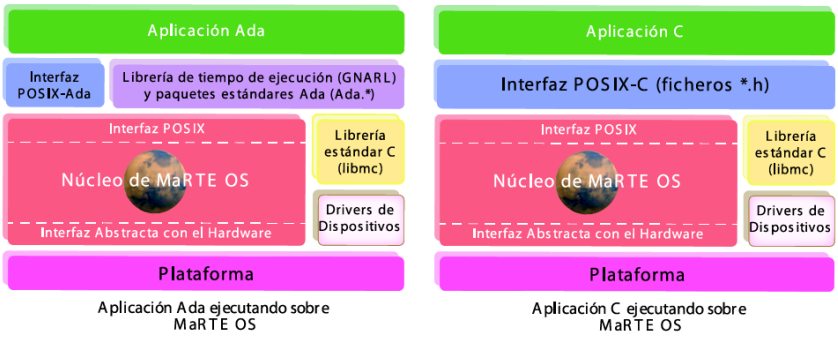


Figura 1.1: Arquitectura de MaRTE OS para aplicaciones en Ada y C

1.3.2. Robot

En el laboratorio del Grupo de Robótica, Percepción y Tiempo Real hay varios robots Pioneer 3-AT (Figura 1.2), modelo utilizado para la realización de este proyecto. Dicho robot está especialmente indicado para el trabajo en escenarios al aire libre ya que se trata de un robot muy robusto. Para una información muy detallada visite [2].

Figura 1.2: Imágen de un robot del laboratorio

El robot Pioneer 3-AT, esta compuesto por una CPU Intel Pentium III 850-MHz, un microcontrolador H8 y por un conjunto de sensores y actuadores que le

permiten obtener información del entorno y desplazarse por él respectivamente. La información utilizada de todos esos sensores y actuadores puede verse a continuación:

- **Posición relativa**: Gracias a la odometría el robot puede saber en cualquier momento en que posición se encuentra, considerando como sistema de referencia la posición y sentido en el que se encontraba en el principio.
- **Obstáculos con láser**: El láser permite conocer la distancia a la que se encuentran los obstáculos que rodean al robot, el cálculo de la distancia se realiza a partir de tiempo que tarda en volver el rebote del laserazo con los obstáculos.
- **Motores**: Los motores son el medio que posee el robot de desplazarse por su entorno. Dado que el robot no puede girar sus ruedas posee dos velocidades, angular y lineal.

Otro elemento importante en el robot es que todas las comunicaciones internas, es decir, las comunicaciones entre la CPU y el microcontrolador, (para leer la odometría y para enviar la velocidad) y entre la CPU y el láser, son a través de la línea serie.

Para un conocimiento más en detalle del robot, de sus sensores y actuadores puede verse el Anexo A.

1.4. Estructura de la memoria

La estructura de la memoria va a ser la siguiente:

- En el capítulo 2 se realiza el análisis de requisitos del proyecto. Por lo que se muestra la tabla de requisitos tanto funcionales como no funcionales del proyecto. También se muestra un primer diseño de la solución.
- En el capítulo 3 se explica cómo se ha llegado a la solución adoptada para la realización de este proyecto. Así como también se muestra un resumen del análisis de Tiempo Real de la aplicación del robot y por último se realizan algunos comentarios sobre la implementación del proyecto.
- En el capítulo 4 se muestran las diferentes fases por las que ha pasado la elaboración del proyecto.
- En el capítulo 5 se encuentran las conclusiones de este trabajo.

Capítulo 2
Análisis

2.1. Análisis y descripción de requisitos

El objetivo de este proyecto es claro, realizar una aplicación de telemanipulación de uno o más robots con restricciones de Tiempo Real. Es decir, se pretende realizar una aplicación con algunas de las funcionalidades desarrolladas en el Grupo dándoles a todas ellas restricciones temporales y, por supuesto, asegurando el cumplimiento de los plazos de ejecución. Para ello, en primer lugar se ha realizado el análisis de requisitos definiendo los siguientes requisitos funcionales (RF) y no funcionales (RNF):

Tabla 2.1: Versión inicial tabla de requisitos

Código	Descripción
RNF1	El proyecto va a constar de dos aplicaciones: una que se ejecutará en el robot y otra en un portátil (host).
RNF2	La aplicación del robot debe funcionar en un robot Pioneer 3-AT [2] del Grupo de Robotica, Percepción y Tiempo Real.
RNF3	La aplicación del robot debe cumplir restricciones de Tiempo Real estricto.
RNF4	La asignación de prioridades de las tareas de la aplicación del host se hará mediante **Deadline Monotonic**.
RNF5	La comunicación entre la aplicación del robot y la del host se realizará de manera inalámbrica, utilizando en ambos casos una tarjeta wi-fi ath5k.
RNF6	La aplicacion del host se ejecutará sobre el sistema operativo Linux.
RNF7	La aplicación del host no deberá cumplir restricciones de Tiempo Real estricto,sino que será un sistema de Tiempo Real acrítico.
RF1	A la aplicación del host debe saber el número de elementos que componen el sistema.

RF2	La aplicación del host debe ser capaz de enviar órdenes a un robot y recibir los datos con los que éste responde.
RF3	La aplicación del host debe mostrar las respuestas de manera gráfica.
RF4	La aplicación del host debe permitir enviar una órden indicando un objetivo relativo al que tiene que navegar el robot.
RF5	La aplicación del host tiene que poder enviar una órden para que el robot navege por un pasillo.
RF6	La aplicación del host debe poder enviar una órden para que el robot deje de navegar automáticamente por el pasillo.
RF7	La aplicación del host tiene que poder enviar periódicamente peticiones de los valores del láser de un robot.
RF8	La aplicación del host tiene que poder enviar periódicamente peticiones de los valores de odometría de los robots.
RF9	La aplicación del host debe permitir modificar el periodo de envío de las peticiones de datos periódicas.
RF10	La aplicación del host debe permitir enviar peticiones de toma de fotografías.
RF11	La fotografías se pedirán bajo demanda, no de forma periódica.
RF12	La aplicación del host debe mostrar la imágen de la fotografía que se ha recibido.
RF13	A cada una de las aplicaciones del robot se le debe proporcionar el número de nodos del sistema y el nodo al que se corresponde.
RF14	La aplicación del robot debe ser capaz de leer periódicamente los valores de odometría del robot.
RF15	La aplicación del robot debe ser capaz de leer periódicamente los valores del láser del robot.
RF16	La aplicación del robot debe poder recibir las órdenes del host.
RF17	La aplicación del robot tiene que saber clasificar las órdenes que le llegan del host.
RF18	La aplicación del robot debe ser capaz de navegar a un objetivo.
RF19	La aplicación del robot debe poder evitar obstáculos durante la navegación.
RF20	La aplicación del robot debe de ser capaz de reconocer si se encuentra en un pasillo.
RF21	La aplicación del robot debe ser capaz, cuando se le ordene, orientarse en la dirección del pasillo y navegar automáticamente por medio de él.

RF22	La aplicación del robot tiene que poder detener la navegación automática por el pasillo cuando se le ordene.
RF23	La aplicación del robot tendrá que detenerse cuando no encuentre un camino al objetivo especificado.
RF24	La aplicación del robot, en caso de que no encuentre un camino al objetivo, debe enviar un mensaje al host indicándoselo.
RF25	La aplicación del robot tiene que poder realizar la localización del robot en un mapa previamente existente.
RF26	La localización del robot se realizará a partir de los datos de odometría y de laser que posea, así como de la localización en un instante anterior.
RF27	La aplicación del robot debe ser capaz de realizar fotografías.
RNF8	La resolución de la calidad de la cámara será baja.
RF28	La aplicación del robot debe poder enviar las fotografías a la aplicación del host.
RF29	En caso de que la fotografía no se pueda enviar en un único mensaje, ésta será particionada.
RF30	La aplicación del host debe ser capaz de recomponer la fotografía correctamente.
RF31	La aplicación del host no puede suponer que las partes de la fotografía van a llegar ordenadas.
RF32	La comunicación entre el host y el robot será síncrona.

Para poder cumplir con el requisito no funcional número 3, el sistema operativo en el que se ejecute la aplicación del robot no puede ser un sistema operativo convencional como puede ser linux, que es el que se esta utilizando en la actualidad en el Grupo, ya que con él no se garantiza el comportamiento de Tiempo Real de las aplicaciones.

Para esta aplicación es necesario que el sistema operativo asegure que las tareas existentes se ejecuten de acuerdo a su prioridad, que no va a haber otros procesos ajenos a la aplicación que la expulsen del planificador y principalmente que el comportamiento sea determinista, lo que permita realizar un estudio de tiempos para asegurar que todas las tareas de la aplicación se ejecuten en tiempo. Por todo ello, y por la existencia de una experiencia previa con él en varios proyectos anteriores, se ha elegido MaRTE OS [1] como sistema operativo sobre el que se ejecutará la aplicación que estará en el robot Pioneer. Por lo que aparece un nuevo requisito no funcional en la tabla de requisitos 2.2.

Código	Descripción
RNF9	La aplicación del robot se ejecutará sobre el sistema operativo de Tiempo Real MaRTE OS.

Tabla 2.2: Requisitos del Sistema Operativo Tiempo Real

Código	Descripción
RNF10	Existirá otra aplicación que se ejecutará en un portátil que se encontrará situado sobre el robot.
RNF11	La aplicación del portátil se ejecutara sobre el sistema operativo Linux.
RF33	Dicha aplicación realizará las fotografías y las enviará a la aplicación del robot.
RNF12	La comunicación entre la aplicación del portátil y la del robot se realizará a traves de un cable ethernet.
RF27b	La aplicación del robot debe ser capaz de recibir las fotografías de la aplicación del portatil y enviarlas al host.

Tabla 2.3: Tabla de requisitos de la aplicación del portátil

2.1.1. MaRTE OS

Como se ha comentado en la introducción (1.2.1) MaRTE OS posee drivers para un considerable número de dispositivos: linea serie, ethernet, impresora, entrada analógica, etc. Pero no posee un driver para una comunicación de alta velocidad como puede ser firewire o usb, tipo de comunicación requerida por las cámaras digitales actuales. Por lo que esta restricción ha provocado la modificación del requisito funcional 27 y la inclusión de algún requisito más en la Tabla 2.3.

2.1.2. RT-WMP

Para la comunicación inalámbrica con el objetivo de que se pueda asegurar un comportamiento de tiempo real en el envio y recepción de los mensajes entre el robot y el host, se ha optado por usar el protocolo RT-WMP, (Real-Time Wireless Multi-hop Protocol [3]), desarrollado por el Grupo y usado actualmente.

Este protocolo de comunicaciones está basado en el paso de testigo, es decir, hay un testigo que se va pasando entre los nodos que indica cuál de ellos tiene el poder de elección en cada momento. Además cada mensaje tiene asociada una prioridad entre 0 y 127, siendo esta última la máxima. Siendo los mensajes de mayor prioridad enviados antes que los de menor.

La gran ventaja de trabajar con un protocolo de comunicaciones de Tiempo Real, así como lo era la de trabajar sobre un sistema operativo de Tiempo Real es que es determinista, por lo que todos los tiempos de envío y de recepción estan acotados. La utilización por parte de la aplicación del protocolo será muy simple ya que de los nodos existentes, uno se corresponde con el host y los demás se corresponden con los robots del sistema.

2.2. Diseño estructural

Una vez que ya se disponen de todos los requisitos, tanto funcionales como no funcionales de la aplicación, lo siguiente es realizar un diseño estructural del proyecto. Es decir, un primer diseño en el que se deje claro las diferentes componentes del trabajo y la principales funcionalidades de las mismas.

En la Figura 2.1 se puede observar que, como se ha determinado en el análisis de requisitos el proyecto consta de tres aplicaciones que se ejecutan en distintos ordenadores.

Figura 2.1: Diseño estructural del proyecto

Las principales funcionalidades de estas aplicaciones son:

- **Portátil**: Se trata de la aplicación que se ejecuta en el ordenador portátil que está conectado al robot. El objetivo de esta aplicación es realizar las fotografías cuando se le ordene, así como de particionarla en trozos para que posteriormente pueda ser reconstruida por el host.

- **Host**: Será la única aplicación a la que tenga acceso el usuario y a partir de la cual se enviarán las órdenes correspondientes a la aplicación que se encuentra en el robot, (recibir información del láser y de la odometría así como navegar hasta un objetivo determinado o de manera autónoma por un pasillo evitando obstáculos y pedir una fotografía). Esta aplicación mostrará gráficamente los datos recibidos, especialmente importante la información del láser y de las fotografías.
- **Robot**: Esta aplicación es la única que tiene restricciones temporales y es la encargada de controlar el robot. Tiene que responder a las órdenes del host. Por lo tanto, debe ser capaz de leer la información que recibe tanto del láser para saber la distancia a los obstáculos como la del microcontrolador para conocer la odometría. También debe poder navegar hasta un objetivo evitando obstáculos con la utilización del algoritmo ND (ver Anexo G), de localizarse en un entorno gracias a un mapa usando el algoritmo SLAM (ver Anexo H) y de comunicarse con la aplicación del portátil para hacerle llegar la petición del host de una fotografía.

Otro apartado importante en este diseño estructural son las comunicaciones. Dado que cada aplicación se ejecuta en un dispositivo diferente se debe determinar bajo que soporte se van a realizar los intercambios de datos entre las aplicaciones.

Como ya se ha comentado en la tabla de requisitos la comunicación entre la aplicación del host y el robot se realizará de manera inalámbrica, utilizando el protocolo de comunicaciones RT-WMP, mientras que la comunicación entre la aplicación de la cámara y el robot se hará mediante ethernet.

Capítulo 3
Diseño solución

3.1. Aplicación del robot

De las tres aplicaciones de las que consta el proyecto, la aplicación más importante y el centro del proyecto, se corresponde con la aplicación del sistema empotrado del robot, por ello se ha intentado encontrar una solución lo más modular posible y siempre teniendo en cuenta que se trata de una aplicación con restricciones de Tiempo Real.

En una aplicación de Tiempo Real, en la cual todas las actividades se van a "pelear" por el acceso a la CPU, lo primero que se debe conocer son las tareas de las que va a constar la aplicación. ¿Cuántas?, ¿Cuáles?, ¿Por qué sólo esas?.

La principal razón para elegir el número de tareas existentes de acuerdo a *Deadline Monotonic* es el periodo de las mismas. Evidentemente dos actividades que, por mucho que tengan que ver y por muy relacionadas que estén, si tienen distintos periodos se tienen que separar, así como, también carece de sentido fusionar dos funcionalidades en la misma tarea si son totalmente independientes, aunque tengan el mismo periodo.

3.1.1. Tareas de actualización de datos

Las tareas más claras son aquellas que actualizan los datos del robot. Del robot, como ya se ha dicho anteriormente se lee el láser, es decir, la distancia a la que se encuentran los obstáculos del robot, y la odometría, la posición absoluta y la orientación en la que el robot cree que se encuentra desde el punto inicial de movimiento.

Por ello, para empezar se tendrán estas dos tareas, las cuales lo único que hacen es actualizar los datos del láser y de la odometría del robot respectivamente, los

cuales serán utilizados por otras tareas de la aplicación.

Se podría pensar en fusionar estas dos actividades en la misma tarea, sin embargo no es posible ya que el periodo de ambas tareas no es el mismo. Éste viene determinado por el tiempo que tarda el robot en enviar los datos actualizados de la odometría y del láser, y mientras que del primero se tiene un mensaje cada 100 ms, del segundo se tiene uno cada 397.15 ms, lo que hace imposible que ambas funcionalidades se encuentren dentro de la misma tarea periódica. El periodo de la la tarea que actualiza los datos del láser viene determinado por el tiempo que tarda el robot en enviar los datos de los 361 laserazos a una velocidad de 19200 baudios a traves de la linea serie, (por restricciones de MaRTE OS la velocidad de la linea serie no puede ser mayor).

Se ha optado por utilizar el láser con un rango de 180 grados para que se pueda ver la mayor amplitud del entorno posible y una resolución de medio grado, lo que da un total de 361 puntos, con el objetivo de tener suficiente información de los obstáculos, sin que todo esto implique un tiempo de espera de la respuesta del láser excesivo. Se pueden ver las distintas configuraciones posibles en el Anexo A.

3.1.2. Tarea comunicaciones

Otra tarea que es independiente a todas las demás es la que se encarga de la comunicación wi-fi, es decir, aquella que se encarga de recibir las órdenes que llegan del host a traves del protocolo RT-WMP, y de enviar las respuestas correspondientes. Dado que se trata de una actividad independiente a cualquier otra funcionalidad de la aplicación del robot se ha decidido que sea una nueva tarea. Se ha optado por un periodo de 200 ms para esta tarea, ya que es lo suficientemente corto para responder a la aplicación del host con la suficiente rapidez para hacerlo dentro del tiempo correspondiente y a que no se colapse la cola de mensajes recibidos.

3.1.3. Tareas de navegación y localización

Por otra parte tenemos todo el bloque de la navegación y localización, el cual se compone de la evitación de obstáculos, de la navegación hasta un objetivo, de la navegación automática por medio de un pasillo y de la localización del robot en un mapa. Estas cuatro funcionalidades están estrechamente realacionadas, ya que haciendo uso de todas ellas se obtiene como salida la velocidad a la que se tiene que mover el robot, por lo que la intención sería que todas ellas pudieran ejecutarse en la misma tarea.

Para saber si esto es posible es indispensable saber el periodo de la tarea. Carecería de sentido, por ejemplo, que esta tarea se ejecutase cada segundo, ya que en ese tiempo la tarea que actualiza el láser con los obstáculos se ha ejecutado más de 2 veces, por lo que el movimiento del robot se realizaría de acuerdo a unos datos obsoletos, lo que podría llevar al choque del robot con un obstáculo.

Por lo tanto, será necesario que la tarea de navegación con la evitación de obstáculos tenga un periodo igual al de la tarea de actualización del láser (397.15 ms), es decir, cada vez que haya datos nuevos del láser, el robot decide su velocidad respecto a ellos. Sin embargo, este periodo es demasiado pequeño para la tarea de localización, la cual necesita un tiempo de cálculo considerable. Por lo que las funcionalidades de navegación y de localización no pueden estar en la misma tarea, ya que no se aseguraría el cumplimiento de plazos, y éste es el punto principal para cualquier aplicación de Tiempo Real.

Se tendrá, entonces, una tarea que realice las funcionalidades de navegación (evitación de obstáculos, navegación hasta un objetivo y navegación autónoma por un pasillo), con un periodo igual a la actualización del láser (397.15 ms), y otra tarea que realizará la funcionalidad de localización y que tendrá un periodo de 1 segundo, tiempo suficiente para que se asegure su cumplimiento de plazos.

Ahora se plantea el problema de como realizar la intererrelacion entre ambas tareas, dado que la tarea de navegación necesita de la de localización para saber donde se encuentra, ya que la odometría del robot, debido a posibles derrapes de las ruedas, no es del todo fiable.

La solución adoptada es bastante simple. Por una parte la localización nos va a proporcionar la posición absoluta en los ejes x e y y el *ángulo* en el que se encontraba el robot en un instante pasado, y por otra almacenará los valores de la odometría que poseía el robot en ese mismo instante. Por lo que a la tarea de navegación le bastará con calcular el movimiento que hay entre su odometría actual y la de dicho instante anterior y aplicar ese movimiento a la localización que se había indicado para ese momento. De forma que como resultado la tarea de navegacion obtendrá su localización para el instante actual.

Si, por el contrario, no hay nuevo valor de localización, (la tarea de localización se ejecuta cada segundo por los 397.15 ms de la de navegación), la posición se calculará aplicando el movimiento producido por iteración anterior a la localización que había en ese instante. De este modo, se tiene la localización lo más actualizada posible.

La posición que se obtiene es un valor aproximado de su localización real, pero este error es mínimo porque la tarea de localización actualizará la posición real del robot periódicamente cada segundo y de esta forma el error no es acumulativo.

3.1.4. Tarea de las fotografías

Por otro lado y como funcionalidad totalmente independiente de los datos del robot, y de su navegación, se encuentra la actividad de pedir las fotografías a la aplicación que se encuentra en el portátil, que está conectado al robot a través del puerto ethernet.

Esta tarea sin embargo, sí que esta relacionada con la tarea de comunicaciones, ya que es ésta la que va a recibir la orden del host de tomar una foto y de enviársela de respuesta. Por lo que se podría pensar en incrustar esta funcionalidad en la tarea de comunicaciones, así como ya lo está el envío de los datos del láser o de la odometría, el cual se hace en la misma tarea de comunicaciones.

El problema en este caso está en el tiempo de ejecución, ya que la aplicación del portátil tarda casi 3 segundos en responder con la foto. Por lo que se necesita una nueva tarea para la funcionalidad de pedir y enviar fotografías debido a la diferencia de periodos.

Pero, a diferencia de todas las tareas anteriores, ésta no se va a comportar como una tarea periódica sino como un servidor esporádico. Dado que las fotografías se piden bajo demanda, se puede considerar la existencia de un evento de tomar fotografías. Luego, el servidor esporádico se quedará bloqueado hasta la existencia de un evento de fotografía, y cuando este suceda pedirá la foto al portátil y la enviará al host, teniendo un periodo de relleno de 3.5 segundos, tiempo suficiente para la ejecución de la tarea. Entendiéndose por *periodo de relleno* el tiempo mínimo entre la respuesta a dos eventos.

Para realizar el análisis de cumplimiento de plazos bastará con quedarse con el peor de los casos, es decir, que el servidor siempre tenga una fotografía que hacer, en tal caso se puede considerar como una tarea periódica de periodo igual a 3.5 segundos.

3.1.5. Establecimiento de prioridades

Una vez que se han determinado las tareas existentes en la aplicación lo siguiente y fundamental para cualquier sistema de Tiempo Real estricto es el estable-

cimiento de las prioridades de cada una de las tareas de la aplicación. Como se indica el el requisito no funcional número 4 (RNF4) de la tabla 2.1, la política de asignación de prioridades es *Deadline Monotonic*, por lo que para que la planificación sea óptima las tareas de menor plazo de respuesta deben ser las que tengan mayor prioridad, en nuestro caso como todas las tareas tienen el mismo plazo de respuesta que periodo nos fijaremos en el periodo.

En el caso de que tengan el mismo periodo, como es en el caso de la tarea de actualizar el láser y la tarea de navegación no importa cual de las dos tenga mayor prioridad ya que en ambos casos se da la solución óptima. Sin embargo se ha optado por que la tarea de actualizar el láser tenga mayor prioridad, ya que de esta forma antes de ejecutarse la tarea de navegación ya se tendrán disponibles los datos del láser actualizados.

Las prioridades de las interrupciones y tareas de la aplicación se pueden ver en la tabla 3.1.

	I.láser	I.P2OS	T.Act.P2OS	T.Com	T.Act.láser	T.Nav	T.Loc	T.Fot
Prio	Hw	Hw	53	52	51	50	49	48

Tabla 3.1: Prioridad de las interrupciones y tareas de la aplicación

Otro factor importante para cualquier aplicación que utilice procesos concurrentes son las variables compartidas. Para garantizar la integridad de los datos se han creado secciones críticas para acceder a ellos.

Las secciones críticas son un punto muy importante en una aplicación de tiempo real ya que pueden provocar lo que se conoce como **inversión de prioridad**, es decir, que una tarea de mayor prioridad se quede bloqueada por otra de menor prioridad debido a que quiere acceder a una sección crítica ocupada por otra tarea de menor prioridad, por lo que tendrá que esperar a que ésta acabe para poder ejecutarse.

Se ha decidido crear más de un servidor, es decir, más de una región crítica para, de esta manera intentar minimizar el bloqueo por inversión de prioridad. Ya que si se tuviese un único servidor con todas las variables compartidas las tareas de mayor prioridad se verían bloqueadas por las de menor prioridad aunque se esté accediendo a varibles que son completamente independientes a las que necesita la otra tarea. Por esto se han establecido diferentes secciones críticas para cada conjunto de variables compartidas.

3.2. Aplicación del host

Respecto a la aplicación del host. La aplicación base ya estaba realizada, por lo que ya se tenía un formato de estructura que ha habido que mantener para las funcionalidades que se necesitaban para este proyecto. En este caso, como se puede observar en el DFD del nivel 2 en el Anexo B, se posee una tarea diferente para controlar cada evento posible, es decir, para cada órden que pueda mandar el usuario: hacer una foto, que el robot avance por medio del pasillo, dar un objetivo relativo al robot, etc. De manera que cuando se realiza una órden, el host envía el comando a la aplicación del robot para que ésta haga las acciones oportunas, y posteriormente se queda esperando la respuesta del robot, se ha optado por esta solución ya que al haber sincronismo entre la aplicación del robot y el host el funcionamiento es mucho más fácil, además de ser éste el comportamiento base pensado para la aplicación.

3.3. Aplicación del portátil

La aplicación del portátil se basa en un programa que lo único que tiene que hacer es esperar a recibir una trama ethernet pidiendole que haga una fotografía, realizar la fotografía, particionarla y enviarla a traves de ethernet, como se puede observar muy claramente en el DFD del nivel 2 en el Anexo B. Los únicos problemas que podían surgir a la hora de plantear la solución de esta aplicación se encontraba en como realizar la fotografía y como enviar los trozos de la foto.

Para la realización de la foto, el requisito funcional número 11 (RF11), de la tabla 2.1 indica que la fotografía se debe realizar bajo demanda. El problema se encuentra en que, por consiguiente, la cámara no hace nada mientras no recibe esa petición de foto, lo que provoca que al realizar la primera, la cámara se encuentre totalmente desenfocada y por lo tanto dicha fotografía no sirva porque no proporciona ningún tipo de información. Por lo que se optó por que cada vez que se recibía la petición de una fotografía se realizasen 10 fotografías consecutivamente enviando como respuesta la última.

Para realizar la partición de las fotografías, con el objetivo de no complicar el tratamiento de los trozos se ha optado por particionarlos de forma que tuviesen un tamaño válido tanto para las tramas ethernet como para las tramas del RT-WMP. Por lo que de esta forma no serán necesarias particiones intermedias.

Por otro lado estaba el problema del envio de las tramas. El problema se encontraba en que, dado que en la aplicación del robot, es decir, en MaRTE OS, se accede

directamente al hardware de ethernet, las tramas enviadas estaban desnudas de protocolos superiores: nivel de enlace, de red, ..., por lo que se necesitaba que las tramas que se enviasen del robot también estuvieran desnudas de estas capas superiores para lo cuál se utilizó un tipo de socket (SOCK_RAW), cuyo comportamiento es análogo al de MaRTE.

3.4. Análisis Tiempo Real

Un apartado fundamental en cualquier aplicación de Tiempo Real es el análisis de cumplimiento de plazos, por el cual se demuestra que todas las tareas que componen el sistema se ejecutan siempre dentro de su plazo de respuesta.

En la Figura 3.1 se muestra detalladamente la información de las tareas y los servidores que se encuentran en la aplicación del robot así como las prioridades correspondientes a las tareas y los techos de prioridad de los servidores.

En este diagrama se puede observar además el tiempo de cómputo en el peor de los casos (C), el periodo (P), y el plazo de respuesta (D) de las tareas y los servicios, con sus respectivos bloqueos, de los servidores que componen la aplicación.

Para que todas las tareas cumplan plazos se debe cumplir que el trabajo requerido al procesador en los plazos de respuesta de las interrupciones y de las tareas sea menor al plazo de respuesta correspondiente. En la tabla 3.2 se muestra de manera resumida toda la información correspondiente a las tareas. Los cálculo de como se ha llegado a estos datos, especialmente los valores del bloqueo y del trabajo se puede observar de una manera detallada en el anexo C.

Tarea	Tpo. Cómputo	Periodo	Bloqueo	Trabajo	Plazo Resp.
Int. láser	0.029	0.4196	0.029	0.058	0.4196
Int. P2OS	0.029	0.833	0.029	0.116	0.833
T. Act. P2OS	0.062	100	0.1986	10.52186	100
T. Com	4.6	200	1.867	27.313	200
T. Act. láser	1.867	397.15	1.1226	53.5336	397.15
T. Navegación	57	397.15	1.1226	110.5336	397.15
T. Localización	157	1000	0.001	460.687	1000
T. Fotografías	9	3500	0	1613.72	3500

Tabla 3.2: Información correspondiente a todas las tareas. Tiempos en ms

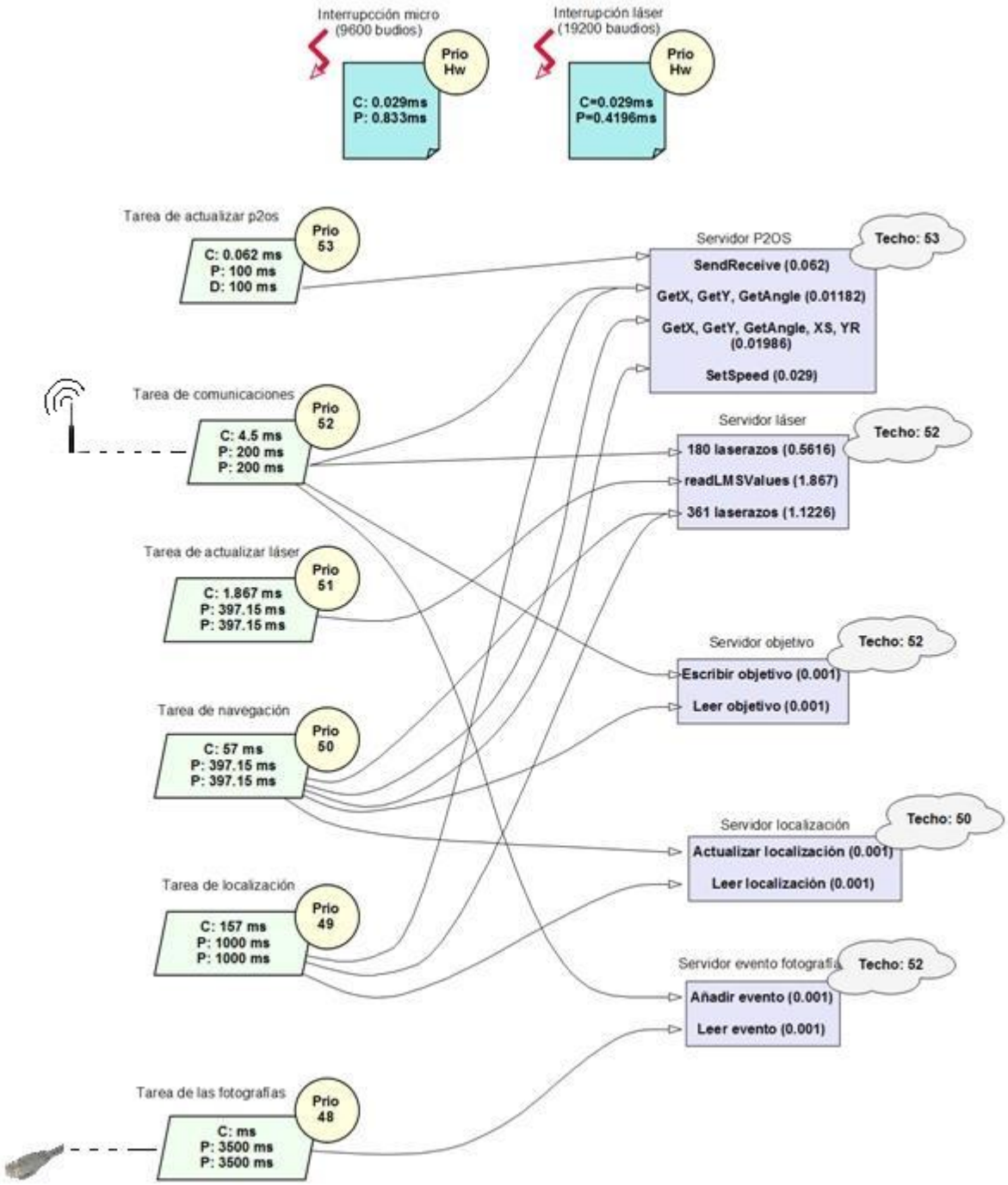


Figura 3.1: Estructura de las tareas y servidores

Se puede observar que para todas las interrupciones y tareas de la aplicación el trabajo en el plazo de respuesta es menor que es plazo de respuesta, condición suficiente para asegurar el cumplimiento de plazos de todas las tareas. Por lo que todas las tareas se ejecutan dentro del tiempo disponible.

3.5. Diseño detallado y codificación

3.5.1. Modificación de los algoritmos

Para la realización de este proyecto ha sido necesaria hacer alguna modificación en los algoritmos principales del proyecto (ND y SLAM), con el objetivo de darles una estructura analizable desde el punto de vista de Tiempo Real.

En una aplicación de Tiempo Real, todos los algoritmos deben estar repartidos en las diferentes tareas que consta la aplicación, y en caso de que haya cualquier tipo de dependencia o intercambio de datos entre ellos debe tratarse con cuidado para que en todo momento se asegure que todas las tareas cumplan sus respectivos plazos de respuesta.

En el caso del algoritmo ND el primer problema encontrado fue que el algoritmo proporcionado no se correspondía en exclusiva con el algoritmo *Nearness Diagram*, sino que además disponía de la funcionalidad de seguimiento (tracking) a una persona. Dado que esta funcionalidad carecía de sentido para nuestra aplicación, la cual está pensada para navegar autónomamente por un túnel, se optó por quitarla. Por lo que el primer problema fué determinar en qué puntos se llevaba a cabo esta funcionalidad para posteriormente eliminarlos y así proporcionarle como objetivo no el objeto en movimiento sino un punto absoluto respecto la odometría del robot.

Posteriomente se pasó a darle la estructura de Tiempo Real. En una tarea periódica el esquema general es bastante simple, ya que consta de una inicialización y de un bucle en el cual se tiene la ejecución de cada iteración. La transformación fué bastante directa ya que bastó con identificar en el algoritmo las inicializaciones llevadas acabo y la parte del código que compone la parte troncal del algoritmo y que queremos que se ejecute periódicamente, con el fin de que, como ya se ha explicado, cada vez que se tengan nuevos datos del entorno del robot se calcule una nueva velocidad para llegar al objetivo deseado evitando los posibles obstáculos del camino. Una vez identificadas estas partes se crea una nueva tarea, la cual se compone de la inicialización de las variables y del bucle que se ejecutará mientras no se finalice la aplicación en el que estará el algoritmo.

Sin embargo, para darle una estructura analizable el bucle de la tarea debe ser periódico, lo que implica que se tiene que medir el tiempo transcurrido desde el comienzo de la ejecución hasta el final y esperar el tiempo restante hasta cumplir el periodo correspondiente. Para ello, en todas las tareas se encuentran las siguientes líneas cuyo objetivo es que la ejecución sea realmente periódica:

```
{
    clock_gettime(CLOCK_REALTIME, &start);
      // ejecución del bucle periódico
    clock_gettime(CLOCK_REALTIME, &stop);
    intervalo = (stop.tv_sec-start.tv_sec)*1000*1000*1000 +
                (stop.tv_nsec-start.tv_nsec);
    espera = periodo-intervalo;
    nanosleep (&ts, NULL);
}
```

En el caso del algoritmo del SLAM el procedimiento para proporcionar una estructura de Tiempo Real fué análogo al algoritmo de navegación, y bastó con crear una nueva tarea periódica con esta funcionalidad.

Con estos algoritmos apareció un problema importante derivado de la mala utilización por parte de MaRTE OS de variables muy grandes como las que se utilizan en estos algoritmos. Un análisis detallado de este problema y de la solución adoptada puede encontrarse en el Anexo D.

3.5.2. Linea serie

A lo largo de este proyecto se han dado varios problemas con el driver de MaRTE OS de la linea serie. La importancia de esta comunicación en el proyecto es grande, ya que, aunque la comunicación entre las aplicaciones sea de manera inalámbrica y por medio de ethernet, la comunicación que existe interna al robot, es decir, la comunicación con el microcontrolador y con el láser, se realiza por este medio.

El problema que se ha dado ha sido que se dejaban de recibir interrupciones, este problema provocaba que no se podían leer los datos de la odometría, no llegaban los datos y tampoco se podía enviar la velocidad al robot, ya que no se recibían las interrupciones que provocan que se envíen los datos. Esto provocaba que la tarea de actualizar la odometría se quedase bloqueada esperando nuevos datos y que la tarea de navegación se quedase bloqueada cuando el búfer de salida se llenaba esperando a que hubiese espacio para mandar la nueva velocidad.

Dado que este problema sólo se producía con la comunicación con el microcontrolador, es decir, con el láser no ocurría, se dedujo que el problema se encontraba en que se enviaban y se recibían datos del mismo puerto serie. Ya que en el caso del láser sólo se leen datos, (la distancia de los obstáculos).

Se descubrió que el problema se producía cuando habiendo datos para enviar llegaba una interrupción de que hay un dato preparado para recibir, de manera que se recibían esos datos pero debido a alguna condición de carrera, ya no se enviaban esos datos que estaban preparados y dejaban de llegar interrupciones. Por lo que la solución pasó por realizar dicha comprobación y, en caso de que hubiese datos preparados para enviar se enviasen en ese momento.

Sin embargo, se pensó que también era necesario acceder en exclusión mutua a la linea serie, con el objetivo de que cualquier otro tipo de condición de carrera no se produjese. En un principio se decididió crear un semáforo que asegurase este acceso, sin embargo esta solución era muy mala ya que al tarea que actualiza los datos de odometría se quedaba bloqueada esperando los datos dentro de una sección crítica, situación muy poco deseable para una aplicación cualquiera y mucho menos en una aplicación con restricciones de Tiempo Real, por lo que esta solución se desechó.

La solución adoptada es que en el manejador de la interrupción se tiene un autómata de estados, de forma que cada vez que recibe una cabecera de datos (0xFA 0xFB) se incrementa una semáforo en el cual está bloqueado la tarea de actualización de la odometría. De esta manera cuando llega una cabecera esta tarea empieza a recibir los datos directamente de la línea serie, evitando que se pudieran dar los problemas anteriormente ocurridos.

3.5.3. Pruebas finales

En el Anexo D se pueden encontrar todos los tipos de pruebas que se han realizado (de comunicaciones, de integración y de cumplimiento de plazos) para comprobar el correcto funcionamiento de las tres aplicaciones que componen el proyecto.

Pero hay que destacar que también se efectuaron pruebas en el entorno pensado para la ejecución del proyecto, más en particular en el túnel de Somport en Canfranc. Proporcinando en todo momento una respuesta satisfactoria y una ejecución como se ha determinado en el apartado 3.4 dentro de su plazo de respuesta.

Capítulo 4
Fases del proyecto e hitos temporales

4.1. Hitos temporales

Hacia Mayo de 2009 decidí realizar este proyecto, tras una presentación del mismo por parte de José Luis Villarroel en clase. Con vistas a empezarlo en el 2º cuatrimestre del año siguiente.

En Marzo de 2010, tras los exámenes, comenzé el proyecto. Este proyecto, encuadrado dentro del Grupo de Robótica, Percepción y Tiempo Real de la Universidad, me pareció muy interesante. Además de que trabajar con algo físico, que se puede percibir, y cuyo objetivo es que el robot se desplaze correctamente dentro de un entorno hostil es una motivación. Entonces empezé por realizar un análisis lo más exhaustivo posible del sistema operativo donde se iba a ejecutar la aplicación del robot, así como comprender el protocolo de comunicaciones inalámbrica que iba a utilizar entre un ordenador y el robot (RT-WMP).

Hacia mediados de ese mes y compaginándolo con la comprensión de MaRTE OS y del RT-WMP realicé algunas aplicaciones de prueba con estos dos elementos, tanto para saber la forma de compilación y ejecución de los mismos, como para entender su funcionamiento.

Posteriormente, en Abril, ya comenzé con la aplicación del proyecto en sí misma, la cual, como ya se ha comentado en varias ocasiones, se compone de tres partes, cada una ejecutándose en un dispositivo distinto. En el diagrama de Gantt (Figura 4.1) se puede observar que la aplicación del host se ha ido modificando conforme se incluían nuevas funcionalidades en la aplicación del robot, así como que la aplicación del portátil con la cámara no se empezó hasta que se habían terminado, verificado e integrado todas y cada una del resto de funcionalidades.

Hacia Mayo de 2010 surgieron los problemas de la linea serie y un poco más tarde, junto a la integración de la funcionalidad de localización los de la integración, que se comenta en el anexo D.

4.2. Diagrama de Gantt

En la Figura 4.1 se puede observar el diagrama de Gantt de este proyecto, en el cual se indica la distribución temporal de las actividades realizadas en el mismo.

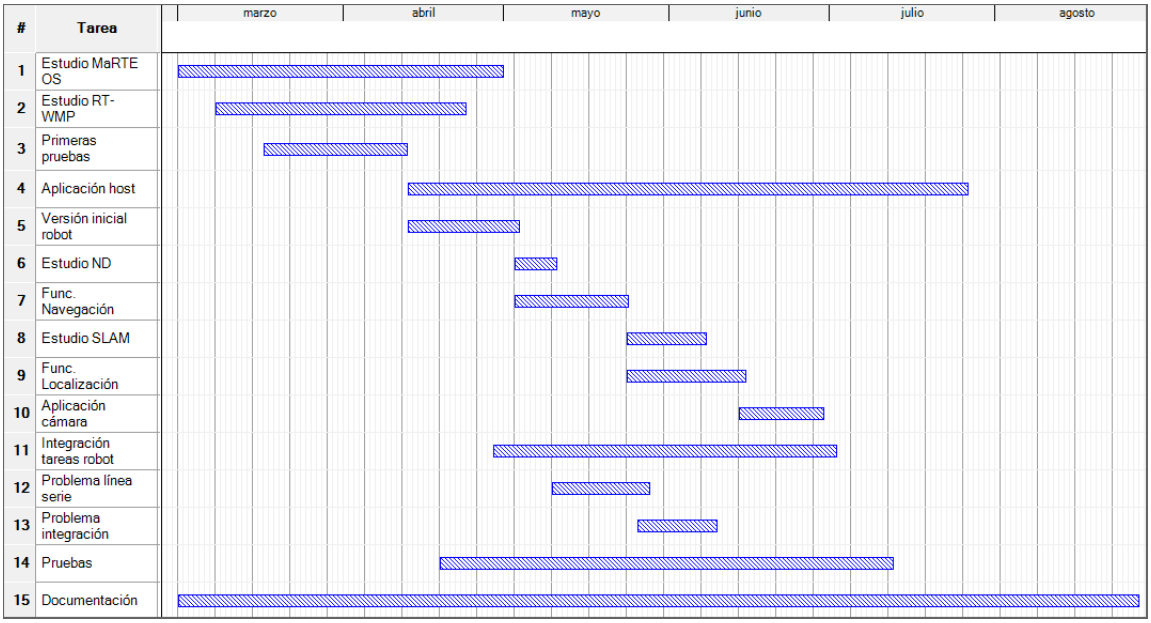


Figura 4.1: Diagrama de Gantt del proyecto

Se puede observar que algunas de las tareas del proyecto fueron realizadas de forma paralela.

Capítulo 5
Conclusiones y trabajo futuro

5.1. Aportaciones PFC

Este PFC ha sido pensado desde un principio con la intención de integrar un conjunto de las funcionalidades de robótica previamente desarrolladas por el Grupo del laboratorio, con el objetivo de que todas ellas se ejecutaran dentro del marco del Tiempo Real. Por tanto, la aportación de este PFC está clara, con este PFC se ha desarrollado una aplicación cuyo comportamiento temporal es mejor que las aplicaciones hasta ahora realizadas en el Grupo, ya que como se había comentado en la Introducción 1.1, bajo la estructura Player/Stage sobre la que se trabaja actualmente no se aseguran las restricciones de Tiempo Real.

Con la finalización de este trabajo se han satisfecho todos los objetivos que inicialmente se habían propuesto. Los objetivos, a parte de los que pueden resultar evidentes como son que las funcionalidades se ejecuten correctamente, iban más allá, especialmente en todo lo relacionado con el Tiempo Real. No hay que olvidar que se trata de un proyecto que se ayuda del Tiempo Real para mejorar el comportamiento de la robótica.

Todas las funcionalidades se comportan correctamente: el robot es capaz de navegar, de localizarse, ..., así como también se asegura el cumplimiento de los plazos de todas las tareas del sistema y todo esto ejecutándose la aplicación del robot sobre un sistema operativo de Tiempo Real.

5.2. Trabajo futuro

Esta aplicación, como cualquier otra, está abierta a mejoras o nuevas funcionalidades. Entre las nuevas funcionalidades, se puede considerar la posibilidad de que la toma de fotografías a través del portátil, deje de ser un hecho pasivo, sim-

plemente proporcionar imágenes al usuario, para transformarse en un valor activo que permita, por ejemplo, la localización del robot, añadiendo objetos reconocibles como las puertas o ventanas a las características de la localización, o incluso, la navegación del robot hasta un objetivo reconocido con la cámara.

Otra posible funcionalidad sería la integración de transmisión de voz a través del portocolo RT-WMP, la cuál se está desarrollando dentro del Grupo del laboratorio. Por lo que la aplicación tendría la capacidad de transmitir tanto imágenes como sonido.

Al fin y al cabo, esta aplicación se puede considerar como una base de la unión entre las funcionalidades de la robótica y las características de Tiempo Real, por lo que cualquier nueva funcionalidad que pueda ser desarrollada dentro del Grupo se podría pensar en integrala a la aplicación.

5.3. Conclusiones

Por último, las conclusiones. Las sensaciones que me deja haber realizado este proyecto son muy positivas, ya que he conocido bastante en profundidad el funcionamiento de un sistema operativo de tiempo real como MaRTE OS, y también me ha permitido entender más en detalle funcionalidades importantes en la robótica como el ND y el SLAM.

El proyecto, aunque pueda parecer en principio no muy complejo, ya que las funcionalidades troncales como la navegación y localización ya estaban implementadas, no es del todo así. El trabajo con un sistema operativo nuevo es complejo y lleva un tiempo entender su funcionamiento, además como se ha comentado a lo largo de la memoria se han dado grandes problemas con la linea serie y la memoria disponible, problemas, todos ellos, intrínsecos al trabajo con un sistema operativo como MaRTE, que han sido, en ambos casos, difíciles de encontrar y en alguna ocasión complicados de solucionar.

La experiencia obtenida con este trabajo ha sido muy buena. Más allá del aprendizaje de nuevos conceptos en temas de robótica, de Tiempo Real o de Sistemas Operativos, me quedo con la sensación de haber realizado un trabajo más complejo de lo que había realizado hasta ahora, y con la capacidad para superar los momentos de obcecación con los problemas.

Anexo A
Robot Pioneer 3-AT

Los robots del grupo, como ya se ha comentado en la introducción son robots Pioneer 3-AT (Figura A.1). Estos robots, poseen un computador Pentium III que, al igual que a cualquier otro ordenador se le pueden conectar diferentes periféricos como una pantalla, teclado o ratón, para facilitar el trabajo con él.

La diferencia con un ordenador normal es que se trata de un sistema empotrado, es decir, se encuentra dentro del dispositivo que controla, como puede ser un teléfono móvil, el computador de un avión, o como en nuestro caso un robot. El hecho de que se encuentre dentro del sistema que controla permite facilmente acceder a todos los dispositivos que posee el mismo robot.

Figura A.1: Imágen de un robot del laboratorio

El robot posee una gran cantidad de sensores que permiten al robot obtener información de su entorno, así como también posee actuadores que le permiten desplazarse por él.

A.1. Sensores

Los sensores que se van a utilizar en este proyecto para el control del robot son los siguientes:

- **Láser**: El láser se corresponderían con los "ojos" del robot. El funcionamiento del láser es fácil de entender, primero se lanza una ráfaga de laserazos hacia el exterior, y en función del tiempo en que tarde en llegar el rebote de los laserazos con los obstáculos se calcula la distancia a la que se encuentran éstos del láser.
  La configuración del laser permite decidir si se desea una ángulo de 100 o de 180 grados (Figura A.2), así como también se puede elegir la resolución que se desea: 1 laserazo cada 1 grado, cada 0.5 grados o cada 0.25, se puede observar las posibles configuraciones en la tabla A.1. Cabe destacar que será importante la configuración del laser, ya que cuanta mayor resolución o cuanto mayor rango, evidentemente mayor será el número de muestras que hay que recibir, por lo que mayor será el tiempo que se tarda en recibir toda la información del laserazo. Luego, en el diseño de la solución (capítulo 3), habrá que buscar un compromiso entre la resolución deseada y el periodo existente entre dos laserazos.

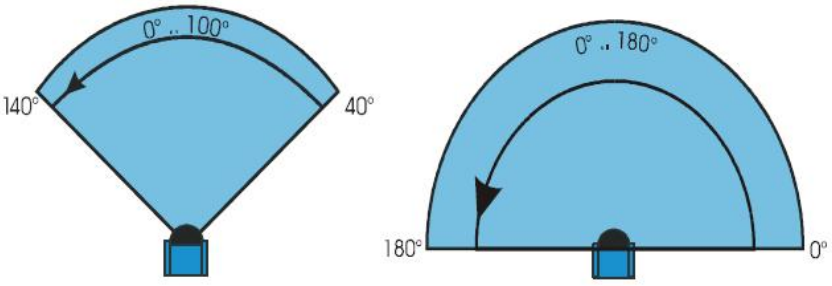


Figura A.2: Rangos posibles del láser

Rango del láser	Resolución del láser	Número de puntos
0º..100º	1º	101
0º..100º	0.5º	201
0º..100º	0.25º	401
0º..180º	1º	181
0º..180º	0.5º	361

Tabla A.1: Configuraciones posibles del láser SICK LMS 200

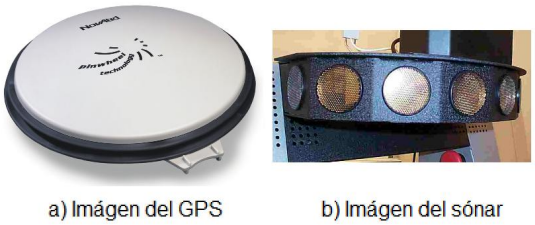
- **Odometría**: La odometría nos indica la posición en la que se encuentra el robot respecto a la posición inicial. La odometría nos proporciona el desplazamiento en el eje X, en el eje Y y el *ángulo* en el que se encuentra respecto la dirección inicial, considerando como 0 grados la orientación en ese momento. Podría parecer que con la odometría es más que suficiente para conocer la posición del robot en cualquier momento, sin embargo esto no es del todo real. Debido a que el suelo puede no ser lo suficientemente rugoso, el robot puede resbalar, lo que implicaría que el robot creería que ha avanzado más de lo que realmente ha hecho, de ahí la importancia de una funcionalidad extra como la localización (ver Anexo H) para el conocimiento exacto de la posición del robot en todo momento.
- **Bumpers**: Los bumpers son unos sensores de choque, como bien indica su nombre. Están situados en la parte frontal y posterior del robot, y su cometido es parar los motores en el caso de que alguno de estos sensores, el robot posee 10, perciba que el robot se ha chocado con un obstáculo.

Sin embargo, el robot también tiene acceso a otros sensores, pero que, debido a la naturaleza del proyecto se ha optado por no utilizarlos ya que no eran convenientes:

- **GPS**: El GPS (Figura A.3) es, realmente, la mejor forma para la localización ya que nos proporciona en todo momento la posición global del robot: longitud y latitud, y se encarga de transformarlas en coordenadas cartesianas. Pero, dado que esta aplicación esta pensada para la intervención en túneles carece de sentido que se utilice, ya que la señal del satélite no llegaría a ser recibida por el robot.
- **Sónar**: El sónar (Figura A.3) es un dispositivo que permite detectar los obstáculos que se encuentran cerca del robot. Éste envía una onda y posteriormente la recibe, calculando, al igual que el láser, la distancia en función del tiempo que tarde en llegar la respuesta. El problema está, en que a diferencia del otro sensor de obstáculos el sónar no envía las ondas en linea recta, lo que puede llevar a la toma de valores erróneos.

A.2. Actuadores

Además de sensores, como se ha comentado al principio del apartado, el robot también posee actuadores que le permiten moverse por su entorno:



a) Imágen del GPS b) Imágen del sónar

Figura A.3: Imágenes del GPS y del sónar del robot

- **Motores**: Los motores son la forma que el robot tiene de moverse. El robot, como se puede observar en la imágen A.1 posee cuatro ruedas, las cuales no se pueden girar, como lo realizan por ejemplo las ruedas delanteras de un automóvil, por lo que será necesario que el robot rote sobre sí mismo para girar en alguna dirección. Por lo tanto, habrá 2 tipos de velocidades en el robot, la velocidad lineal y la velocidad angular. Siendo 2 metros por segundo la velocidad lineal máxima y de 360º por segundo la angular.

Anexo B
Análisis problema

Una vez que se han identificado y analizado los requisitos funcionales de la aplicación, lo siguiente a realizar es estudiar más en profundidad cada uno de ellos y proporcionar un diagrama de flujo de datos (**DFD**) para conocer como los datos son modificados a lo largo de la aplicación. Para obtener al final el resultado deseado (el movimiento del robot, así como también la recepción de una fotografía, los datos del láser, ...).

B.1. DFD nivel 0

En el DFD del nivel 0 (Figura B.1) se puede observar que van a existir tres aplicaciones bien diferenciadas en nuestro problema: la aplicación del robot, la cual tiene restricciones de Tiempo Real estricto y se encarga del control del robot; la aplicación del host, que será la encargada de enviar las órdenes a la aplicación empotrada; y la aplicación del portatil, que, como se ha indicado en el análisis de requisitos, es la encargada de tomar las fotografías.

Se puede observar que existen tres entidades externas, una se corresponde con el usuario, quién es el encargado de indicar las acciones que quiere que el robot realice; otra se corresponde con la cámara la cual proporciona la fotografía actual y la última se corresponde con el sistema operativo del robot. Esta última se debe considerar una entidad externa ya que se le proporcionan datos de salida, como puede ser la velocidad a la que queremos que se muevan las ruedas del robot, y también se leen datos de ella como la odometría y el láser.

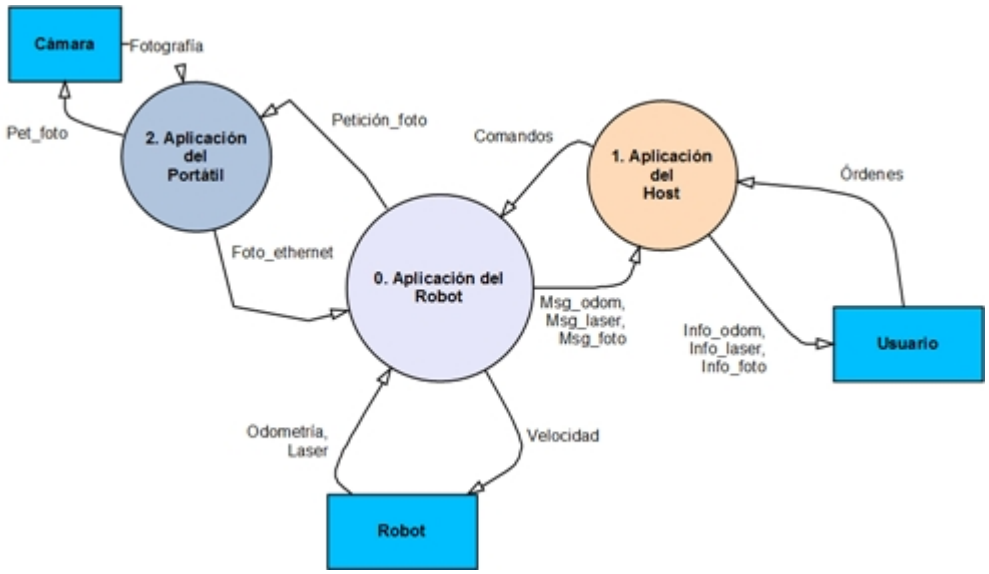


Figura B.1: DFD nivel 0

B.2. DFD nivel 1

En el DFD del nivel 1 (Figura B.2) se observan más en detalle los procesos principales de cada una de las 3 aplicaciones que componen el sistema global.

En la aplicación del portátil, se puede observar que una vez que se recibe la órden de realizar una fotografía es entonces cuando se toma la foto. Como se dice en el requisito funcional 11 (RF11) de la tabla 2.1 las fotografías se realizan bajo demanda, y posteriormente, se envía a traves de la ethernet, particionándola previamente si es necesario.

En la aplicación del host, se observa que en primer lugar se reciben las órdenes del usuario a través de la interfaz gráfica. Estas órdenes pueden ser explícitas del usuario, como por ejemplo, indicar el objetivo al que quiere que llegue el robot, o implícitas, activando un servicio que se ejecuta periódicamente. Una vez que se tiene una órden ésta se prepara para poder enviarla por la wi-fi a través del protocolo RT-WMP. Como se describe en el requisito funcional 32 (RF32) de la tabla 2.1 la comunicación entre el host y el robot es síncrona, por lo que el servicio correspondiente se queda bloqueado hasta recibir la respuesta que se mostrará gráficamente al usuario.

Por último en la aplicación del robot, la más compleja, se puede observar que

para cada una de las funcionalidades de dicha aplicación hay un proceso distinto, salvo para las funcionalidades de navegación y de localización, ya que dada la interrelación que hay entre ellas se ha optado por entenderlo como un único proceso en el DFD del nivel 1. En esta aplicación cabe destacar el acceso a los valores de los sensores del robot, (odometría y láser). Sería impensable, que cada vez que se necesita la odometría del robot se tuviera que leer directamente de la linea serie los valores de posición del robot, ya que eso traería un retraso a cada proceso que quisiese los datos, (dato a tener muy en cuenta en una aplicación con restricciones de Tiempo Real), y además se perdería modularidad, debido a que distintos procesos estarían haciendo lo mismo. Por ello, se ha pensado en crear dos almacenes en los cuales se escribe la odometría y los valores del láser, para, posteriormente ser consultados por los procesos correspondientes.

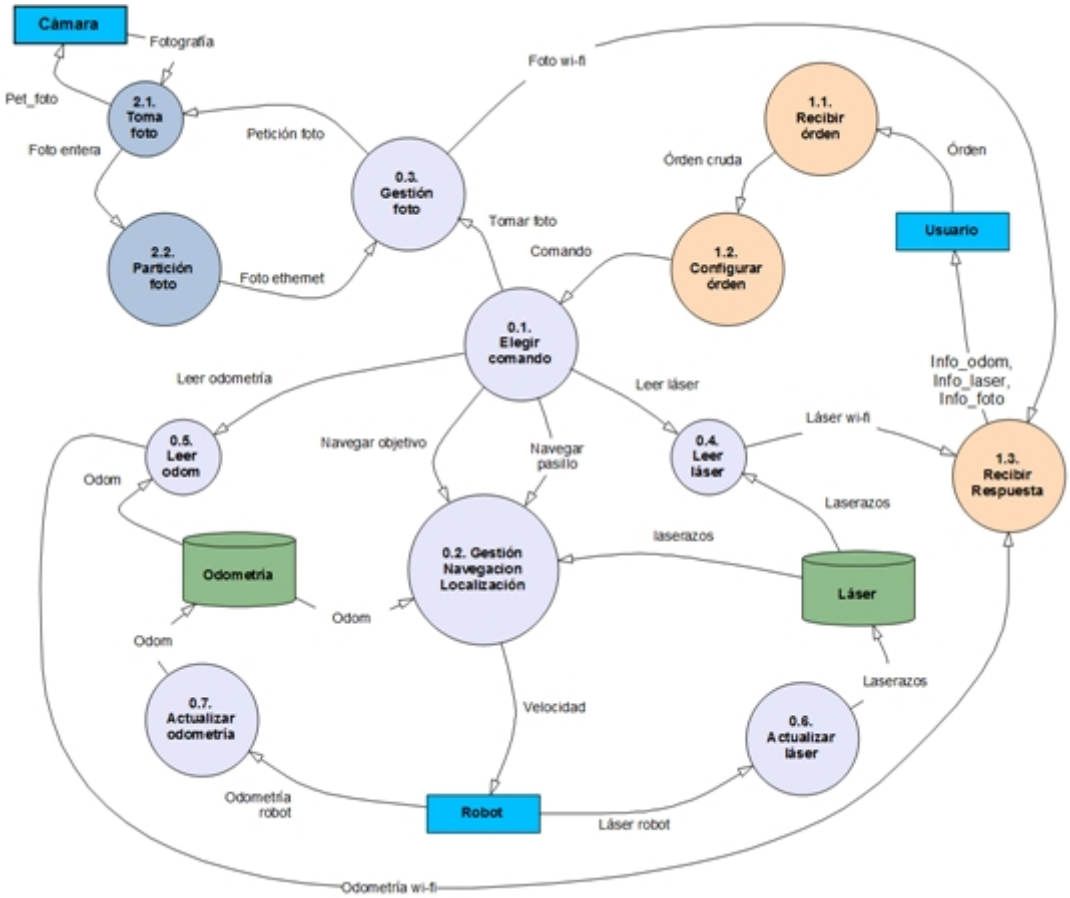


Figura B.2: DFD nivel 1

B.3. DFD nivel 2

Por último en el DFD del nivel 2 (Figura B.3) se realiza un análisis más exhaustivo de cada una de las funcionalidades del proyecto, dividiendo cada uno de esos procesos asociados a cada funcionalidad en subprocesos más simples.

El caso más evidente se corresponde con el proceso 0.2 (*Gestión de navegación y localización*), en el cual se distinguen de una manera muy clara los subprocesos aparecidos en el Diagrama de Flujo de Datos del nivel 1.

En primer lugar, se realiza la localización, con el objetivo de que el robot, gracias a la información que obtiene del entorno, los laserazos y la odometría, pueda obtener la verdadera localización en la que se encuentra. Se puede ver una descripción del funcionamiento del algoritmo de SLAM [5] (Simultaneous Localization And Mapping) para la localización del robot en el Anexo H.

Una vez que ya se tiene la posición del robot, se procede a determinar la posición absoluta del objetivo. Para ello hay que diferenciar si se recibe del usuario la posición relativa del objetivo respecto el robot, o simplemente se trata de una órden de navegación por el pasillo. En caso de que se trate de este segundo, en primer lugar se tendrán que reconocer las paredes del pasillo para saber si ciertamente el robot se encuentra en uno y, posteriormente, en caso afirmativo se procede a calcular un objetivo en medio del pasillo, el cual se irá actualizando periódicamente para que siempre circule por el pasillo, hasta que se le indique lo contrario.

Por último, se realizan las acciones necesarias para obtener el entorno del robot a partír de la información recibida del láser, y que será la entrada para el proceso de cálculo de velocidad, el cual se corresponde con el algoritmo ND, Nearess Diagram [4], el cual se encarga de la navegación del robot entre los obstáculos para llevar al robot al objetivo deseado. Se puede ver una descripción extensa del funcionamiento de este algoritmo en el Anexo G.

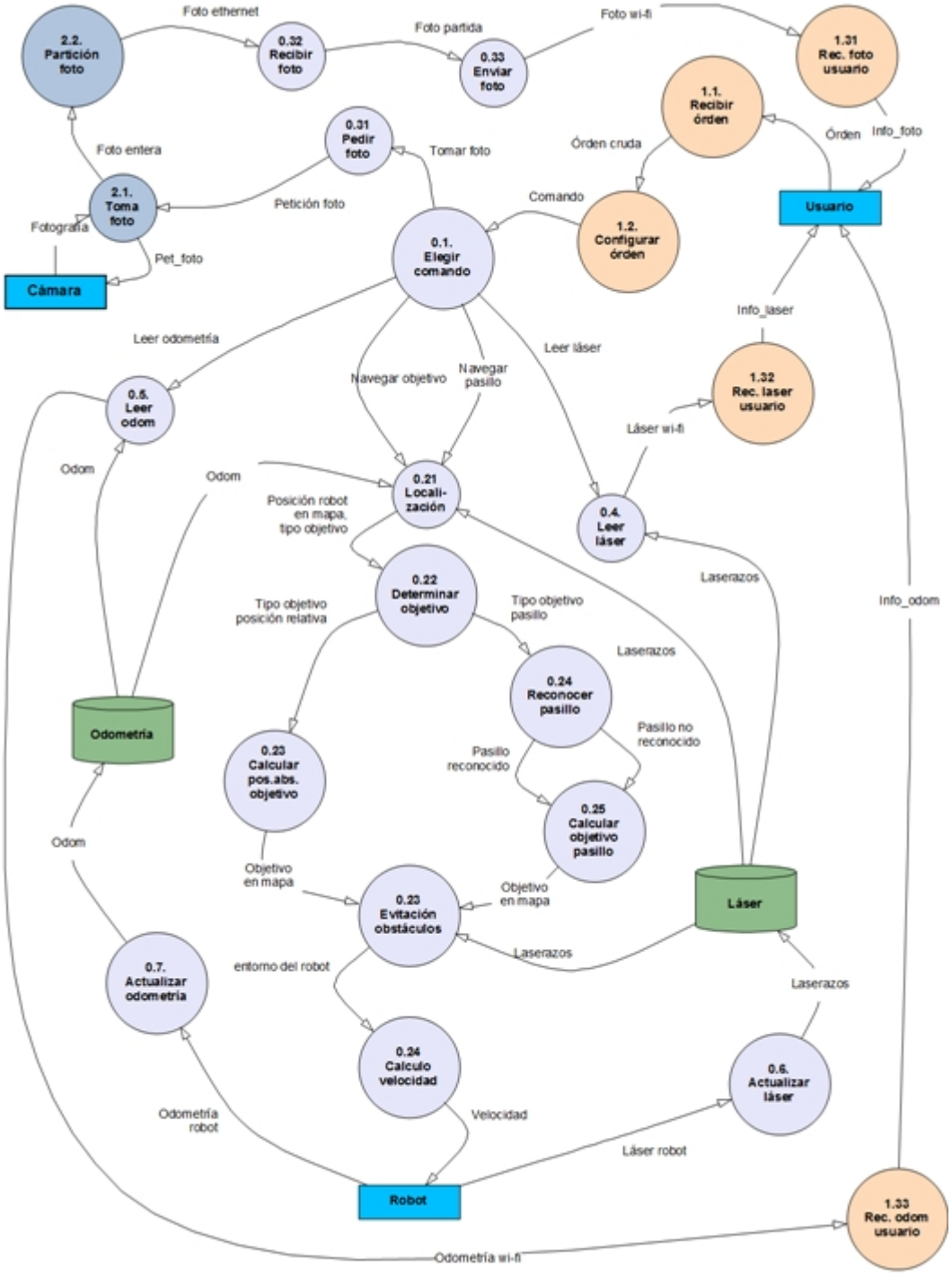


Figura B.3: DFD nivel 2

Para una aplicación de Tiempo Real, el análisis del problema no queda terminado tras realizar el DFD del nivel 2, ya que, como al final únicamente va a haber tareas que se ejecutan concurrentemente en la aplicación se debe relacionar las funcionalidades existentes del robot con sus tareas.

Inicialmente se puede observar que las tareas que actualizan tanto la odometría como el láser del robot se pueden corresponder a una o dos tareas independientes, dependerá de si el periodo de ambas es el mismo o no.

Por otra parte todos los procesos encargados de recibir y enviar datos al host, se pueden unir en una única tarea cuya funcionalidad sea utilizar el protocolo RT-WMP para la comunicación inalámbrica.

Otros procesos que son independientes al resto son los relacionados con la petición de una fotografía, por lo que se podría pensar en crear una nueva tarea que se encargase de la comunicación con el ordenador potátil a traves de ethernet para recibir los trozos de las fotografías.

Por último quedan los procesos relacionados con la navegación y la localización del robot. Al igual que en el caso de las tareas de actualización de datos se podría pensar en unir estas dos funcionalidades en una tarea, pero al igual que en el caso anterior dependerá de si los periodos de ambas funcionalidades son los mismos.

En el capítulo 3 se determinan razonadamente el número de tareas que hay en la aplicación del robot, así como las razones por las que las tareas que hay son esas.

Anexo C
Arquitectura de la aplicación del robot

Para cualquier aplicación que tenga restricciones de Tiempo Real es fundamental identificar claramente las tareas que componen el sistema, como ya se ha hecho en el apartado 3.1, para, a continuación, asegurar que en todo momento se cumplen los plazos de ejecución para todas las tareas del sistema.

En una aplicación con restricciones temporales, es tan importante la prioridad de las tareas que la componen, como la prioridad de los servidores de las variables compartidas. Ya que si los servidores tuvieran una prioridad mayor que todas las tareas, podría provocar que se violasen los periodos de algunas tareas que son independientes de ese servidor.

Para la elección de la prioridad de los servidores se ha optado por una solución simple como puede ser el techo de prioridad inmediato, es decir, la prioridad de un servidor se corresponde con la prioridad de la tarea de mayor prioridad que accede a él. Por lo que si a él accede la tarea de mayor prioridad del sistema cualquier tarea que acceda a ese servidor tendrá la máxima prioridad, pero si por el contrario, sólo acceden al servidor las 2 tareas de menor prioridad, todas las tareas con mayor prioridad no se verán bloqueadas por ellas.

Se ha elegido esta solución porque, a diferencia del protocolo de herencia prioridad en el cual se tiene en cuenta la prioridad de la tarea que entra al servidor, la prioridad de la tarea que se queda bloqueada en el servidor, etc. no hay un gasto de tiempo computacional para obtener la prioridad de la tarea en cada instante, ya que en este caso la tarea obtiene directamente la prioridad del servidor al que accede.

Para que un servidor posea la característica de techo de prioridad inmediato

basta con darle el valor correspondiente al atributo de protocolo de prioridad al mutex que restringe el acceso a las variables compartidas, y a continuación indicarle la prioridad del servidor:

```
pthread_mutexattr_setprotocol(&mutexattr, PTHREAD_PRIO_PROTECT);

pthread_mutexattr_setprioceiling(&mutexattr, 21);
```

En la Figura C.1 se pueden observar tanto las tareas que componen la aplicación del robot, como los servidores que se utilizan para el acceso a las variables compartidas entre distintas tareas, a las cuales se accede en exclusión mutua para asegurar su integridad en todo momento.

En el diagrama se puede observar que para todas las tareas se indica su prioridad (Prio N). Esta prioridad se ha establedido, como se pide en el requisito no funcional 4 (RNF4) de la tabla 2.1, según la estrategia **Deadline Monotonic Scheduling**, en la cual se da mayor prioridad a las tareas que tengan un plazo de respuesta menor, que en nuestra aplicación es igual al periodo.

Para las interrupciones hardware: microcontrolador y láser, la prioridad viene establecida por MaRTE (prioridad hardware), y es superior a la máxima prioridad que puede tomar cualquier tarea, (el rango de prioridades establecido por MaRTE de las tareas va desde 0 hasta 97, mientras que la prioridad de las interrupciones es 98). Lo que implica que la ejecución de las tareas no va a influenciar al de las interrupciones, pero sí al contrario.

En estas figuras se puede observar que se indica:

- **C**: Tiempo de cómputo en el peor de los casos. No se indica un tiempo de cómputo medio de la tarea, sino el peor, ya que se trata del valor necesario para el análisis de cumplimiento de plazos.
- **P**: Periodo. Indica cada cuánto tiempo la tarea se activa para volver a ser ejecutada.
- **D**: Plazo de respuesta. Se corresponde con el tiempo que se posee como máximo para que se lleve a cabo la total ejecución de la tarea. El plazo de respuesta, como es evidente, debe ser menor o igual al periodo de la tarea.

Los rectángulos se corresponden con los servidores de la aplicación. En ellos se puede observar las ejecuciones que se lleva a cabo dentro de ellos, que puede

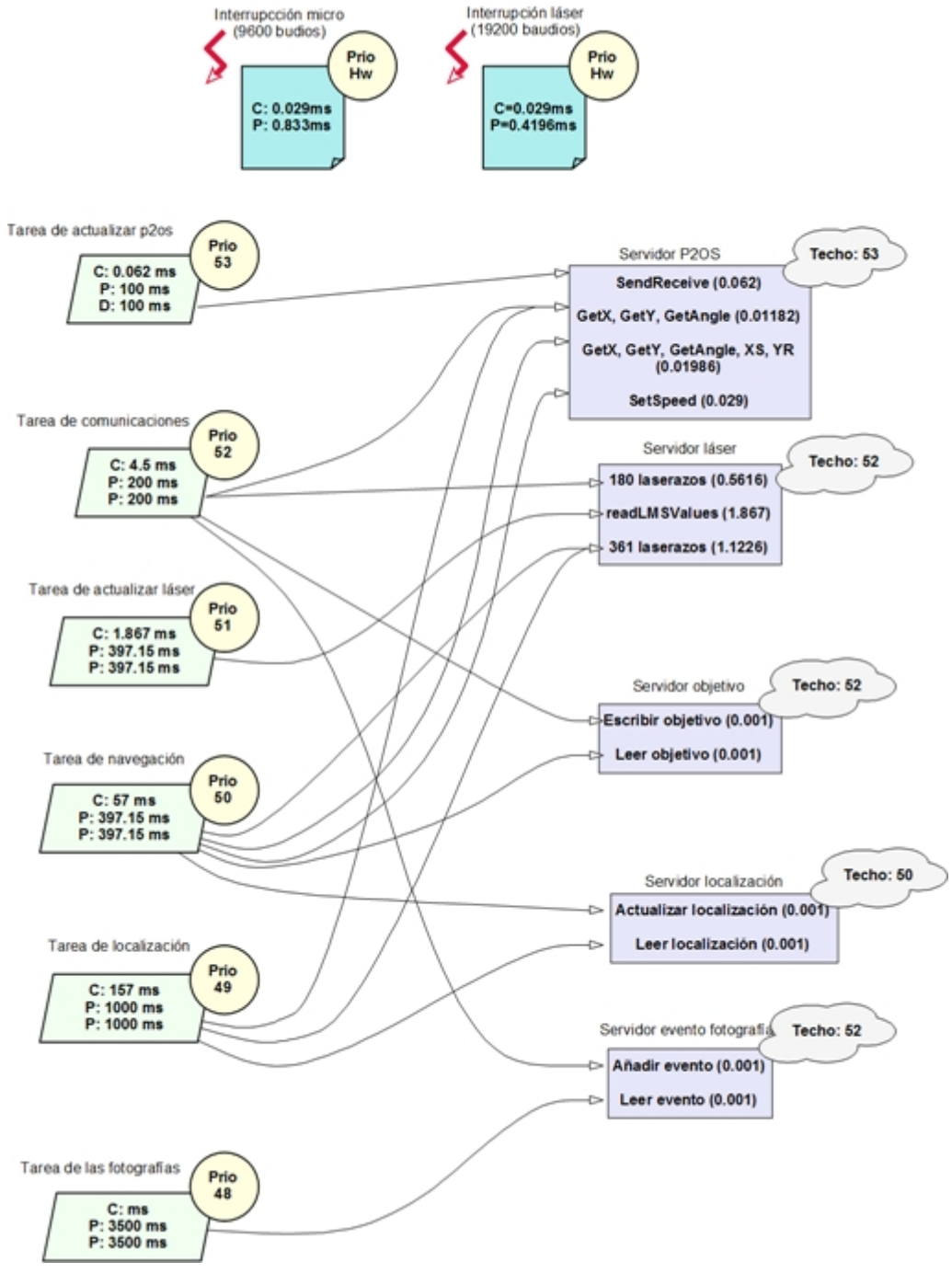


Figura C.1: Estructura de las tareas y servidores

ir desde realizar una operación que puede llevar un tiempo razonable, hasta simplemente leer una variable, para lo cual se tarda un tiempo prácticamente despreciable. Junto a cada operación se indica el tiempo que se tarda en ejecutarlo, es decir, el tiempo de bloqueo que puede provocar en otra tarea de mayor prioridad.

Se puede observar que como la tarea de mayor prioridad, la de actualizar el p2os, accede al servidor del p2os, las tareas que accedan a ese servidor, en ese momento tendrán la mayor prioridad de las tareas, sólo superado por las interrupciones hardware.

C.1. Análisis de tiempos

C.1.1. Cálculo de tiempos de bloqueo

Dado que en el peor de los casos el protocolo de techo de prioridad inmediato se comporta igual que en el caso del protocolo de techo de prioridad, y que para el análisis del cumplimiento de plazos sólo nos interesa ese peor caso, se puede utilizar la misma técnica de análisis del cumplimiento de plazos para el protocolo utilizado que para el protocolo de techo de prioridad.

Por lo tanto lo más dificil es obtener el tiempo de bloqueo de cada una de las tareas. La tarea de menor prioridad del sistema, en nuestro caso la tarea de las fotografías, no puede sufrir este tipo de bloqueo, ya que no hay tareas de menor prioridad que ella. Mientras que las demás tareas hay que tener en cuenta que pueden verse bloqueadas por tareas de menor prioridad aunque no llamen a servidores comunes, siempre y cuando el techo de prioridad de este servidor sea superior o igual a la prioridad de la tarea.

Por lo que en primer lugar habrá que obtener el tiempo de bloqueo para cada una de las tareas del sistema:

- **Rutinas de interrupción**: Dado que la prioridad de las rutinas de interrupción es superior a cualquier prioridad de una tarea. Una rutina sólo puede verse bloqueada por la otra rutina, por lo tanto, el tiempo de bloqueo de una rutina será igual al tiempo de ejecución de la otra. En nuestro caso, como ambas rutinas tienen el mismo tiempo de cómputo, para ambas el bloqueo es de **0.029 ms**.
- **Tarea de actualización del P2OS**: Para obtener el tiempo de bloqueo para cualquier tarea, hay que fijarse en los servidores cuyo techo de prioridad es

igual o superior a la prioridad de la tarea, y quedarse con el servicio más duradero llamado por una tarea de menor prioridad. En el caso de esta tarea el único servidor que le puede bloquear es el servidor al que accede (Servidor P2OS), teniendo el servicio más largo una duración de **0.01986 ms**, llamado por la tarea de navegación.

- **Tarea de comunicaciones**: Esta tarea puede verse bloqueada por varios servidores: el servidor del láser, el del P2OS, el servidor del objetivo y por el de eventos de fotografías. De estos tres, el servicio más largo llamado por una tarea de menor prioridad se corresponde con el servicio *readLMSValues* que tiene una duración de **1.867 ms** y es llamado por la tarea de actualización del láser.
- **Tarea de actualización del láser**: Los servidores que afectan a esta tarea son los mismos que en el caso anterior, pero el bloqueo es distinto. En este caso el servicio más duradero llamado por una tarea de menor prioridad se corresponde con el de leer el número de laserazos y leerlos todos, con un tiempo de ejecución de **1.1226 ms**. Por lo que éste es el tiempo máximo de bloqueo para esta tarea.
- **Tarea de navegación (ND)**: En el caso de la tarea de navegación, el servicio con un mayor tiempo de cómputo llamado por una tarea de prioridad mayor es mismo que para la tarea anterior, el de lectura de los 361 laserazos, en este caso de la tarea de localización. Por lo tanto, el tiempo de bloqueo de esta tarea es de **1.1226 ms**.
- **Tarea de localización (SLAM)**: Para esta tarea, en todos los servidores a los que accede se trata de la tarea de menor prioridad, por lo que no puede ser bloqueada por los servicios de dichos servidores. Pero, por otra parte, la tarea de la cámara (la única de menor prioridad que la tarea de localización), accede a un servidor (instante_evento) con un techo de prioridad superior a la prioridad de esta tarea. Luego, el bloqueo de esta tarea es igual al servicio llamado por la tarea de la cámara, **0.001 ms**.
- **Tarea de la cámara**: Dado que esta tarea se corresponde con la tarea de menor prioridad del sistema, como se ha comentado anteriormente, su tiempo de bloqueo es igual a **0 ms**.

Esta información se puede ver de una forma más clara en la tabla C.1.

C.1.2. Análisis del cumplimiento de plazos

Una vez obtenido los tiempos de bloqueo de las tareas del sistema y conociendo los tiempos de ejecución y periodo de las tareas (Figura C.1) ya se puede realizar

Int. Láser	Int. P2OS	T. P2OS	T. com	T. láser	T. Nav	T. Loc	T. cam
0.029	0.029	0.1986	1.867	1.1226	1.1226	0.001	0

Tabla C.1: Bloqueos de todas las interrupciones y tareas del sistema (ms)

el análisis de cumplimiento de plazos.

Para ello se va a calcular el trabajo requerido al procesador en los plazos de respuesta de cada tarea (ecuación C.1), y en caso de que para todas las interrupciones y tareas este trabajo sea menor al plazo de respuesta correspondiente, se asegura el cumplimiento de los plazos de las tareas del sistema.

$$W(D_i) = \sum_{j<i} \left(\left\lceil \frac{D_i}{P_j} \right\rceil * C_j \right) + C_i + B_i \leq D_i \tag{C.1}$$

Tarea Interrupción láser:

$$W(D_0) = \sum_{j<0} \left\lceil \frac{D_0}{P_j} \right\rceil * C_j + C_0 + B_0 = 0 + 0{,}029 + 0{,}029 = 0{,}058 \leq 0{,}4196$$

Dado que el trabajo es menor al plazo de respuesta, la interrupción cumple el plazo de respuesta.

Tarea Interrupción P2OS:

$$W(D_1) = \sum_{j<1} \left\lceil \frac{D_1}{P_j} \right\rceil * C_j + C_1 + B_1 = \left\lceil \frac{0{,}8333}{0{,}4196} * 0{,}029 \right\rceil + 0{,}029 + 0{,}029 = 0{,}116 \leq 0{,}8333$$

Se puede observar que esta interrupción también cumple su plazo de respuesta.

Tarea de actualización del P2OS:

$$\begin{aligned} W(D_2) &= \sum_{j<2} \left\lceil \frac{D_2}{P_j} \right\rceil * C_j + C_2 + B_2 = \left\lceil \frac{100}{0{,}4196} * 0{,}029 \right\rceil + \left\lceil \frac{100}{0{,}8333} * 0{,}029 \right\rceil \\ &+ 0{,}062 + 0{,}01986 = 10{,}52186 \leq 100 \end{aligned}$$

En este caso la tarea cumple su plazo de respuesta.

Tarea de comunicaciones:

$$\begin{aligned} W(D_3) &= \sum_{j<3} \left\lceil \frac{D_3}{P_j} \right\rceil * C_j + C_3 + B_3 = \left\lceil \frac{200}{0{,}4196} * 0{,}029 \right\rceil + \left\lceil \frac{200}{0{,}8333} * 0{,}029 \right\rceil \\ &+ \left\lceil \frac{200}{100} * 0{,}062 \right\rceil + 4{,}5 + 1{,}867 = 27{,}313 \leq 200 \end{aligned}$$

En este caso la tarea cumple su plazo de respuesta.

Tarea de actualización del láser:

$$\begin{aligned} W(D_4) &= \sum_{j<4} \left\lceil \frac{D_4}{P_j} \right\rceil * C_j + C_4 + B_4 = \left\lceil \frac{397{,}15}{0{,}4196} * 0{,}029 \right\rceil + \left\lceil \frac{397{,}15}{0{,}8333} * 0{,}029 \right\rceil \\ &+ \left\lceil \frac{397{,}15}{100} * 0{,}062 \right\rceil + \left\lceil \frac{397{,}15}{200} * 4{,}5 \right\rceil + 1{,}867 + 1{,}1226 \\ &= 53{,}5336 \leq 397{,}15 \end{aligned}$$

En este caso la tarea cumple su plazo de respuesta.

Tarea de Navegación:

$$\begin{aligned} W(D_5) &= \sum_{j<5} \left\lceil \frac{D_5}{P_j} \right\rceil * C_j + C_5 + B_5 = \left\lceil \frac{397{,}15}{0{,}4196} * 0{,}029 \right\rceil + \left\lceil \frac{397{,}15}{0{,}8333} * 0{,}029 \right\rceil \\ &+ \left\lceil \frac{397{,}15}{100} * 0{,}062 \right\rceil + \left\lceil \frac{397{,}15}{200} * 4{,}5 \right\rceil + \left\lceil \frac{397{,}15}{397{,}15} * 1{,}867 \right\rceil \\ &+ 57 + 1{,}1226 = 110{,}5336 \leq 397{,}15 \end{aligned}$$

En este caso la tarea cumple su plazo de respuesta.

Tarea de Localización:

$$\begin{aligned} W(D_6) &= \sum_{j<6} \left\lceil \frac{D_6}{P_j} \right\rceil * C_j + C_6 + B_6 = \left\lceil \frac{1000}{0{,}4196} * 0{,}029 \right\rceil + \left\lceil \frac{1000}{0{,}8333} * 0{,}029 \right\rceil \\ &+ \left\lceil \frac{1000}{100} * 0{,}062 \right\rceil + \left\lceil \frac{1000}{200} * 4{,}5 \right\rceil + \left\lceil \frac{1000}{397{,}15} * 1{,}867 \right\rceil \\ &+ \left\lceil \frac{1000}{397{,}15} * 57 \right\rceil + 157 + 0{,}001 = 460{,}687 \leq 1000 \end{aligned}$$

En este caso la tarea cumple su plazo de respuesta.

En el capítulo de Diseño de la solución (3), se hizo hincapié en la necesidad de separar la funcionalidad de navegación (ND) de la de localización ($SLAM$), ya que se podía producir una violación del plazo de respuesta en una posible tarea que contuviese ambas funcionalidades. Sin embargo gracias a las dos última ecuaciones se puede observar que para un periodo de respuesta de 397.15 ms se cumplirían perfectamente.

El problema se encuentra en el propio algoritmo del SLAM, cuyo tiempo de ejecución depende directamente del tamaño del mapa que posee, (a diferencia del algoritmo del ND que utiliza un número fijo de laserazos, y por tanto, de información), por lo que en caso de utilizar mapas más grandes que el que se ha utilizado en esta ocasión podría provocar una violación de su plazo de respuesta. Con el objetivo de que esto no llegase a ocurrir se ha optado por una solución conservadora con una tarea para cada funcionalidad.

Tarea de la cámara:

$$\begin{aligned} W(D_7) &= \sum_{j<7} \left\lceil \frac{D_7}{P_j} \right\rceil * C_j + C_7 + B_7 = \left\lceil \frac{3500}{0{,}4196} * 0{,}029 \right\rceil + \left\lceil \frac{3500}{0{,}8333} * 0{,}029 \right\rceil \\ &+ \left\lceil \frac{3500}{100} * 0{,}062 \right\rceil + \left\lceil \frac{3500}{200} * 4{,}5 \right\rceil + \left\lceil \frac{3500}{397{,}15} * 1{,}867 \right\rceil \\ &+ \left\lceil \frac{3500}{397{,}15} * 57 \right\rceil + \left\lceil \frac{3500}{1000} * 157 \right\rceil + C_7 + 0 = 1613{,}72 \leq 3500 \end{aligned}$$

En este caso la tarea cumple su plazo de respuesta.

Dado que para todas las tareas que componen el sistema su trabajo es menor que el plazo de respuesta, se asegura el cumplimiento de plazos de todas las tareas del sistema.

Anexo D
Pruebas

Como último punto de la realización de cualquier aplicación se encuentran las pruebas. En este caso las pruebas se han ido realizando a lo largo del desarrollo de la aplicación probando cada funcionalidad individualmente y posteriormente comprobando que la integración da cada funcionalidad con las ya existentes en el proyecto también fuese correcta.

Dado que el proyecto se compone de tres aplicaciones también se han realizado pruebas individuales de cada componente y pruebas graduales de sistema, es decir, pruebas del correcto funcionamiento del sistema para cada nueva funcionalidad añadida.

Dada la naturaleza de este proyecto, en el cual se trabaja con un sistema empotrado, la realización de las pruebas es un tanto más pesada que para una aplicación que se ejecuta en un ordenador, dado que cada vez que se quería realizar una prueba, ya fuese de una funcionalidad, de integración, de comunicaciones, etc. se debía pasar el ejecutable creado por MaRTE OS, al robot y posteriormente ejecutarlo. Sin embargo, esto es algo inherente de las pruebas en sistemas empotrados, es imposible realizar la prueba de un sistema empotrado sin la utilización del mismo.

D.1. Pruebas de comunicaciones

Además de las pruebas funcionales también se han realizado otro tipo de pruebas como puede ser pruebas de comunicaciones, elemento muy importante en este proyecto, ya que existen 3 componentes y estas se comunican entre ellas de dos maneras diferentes: el robot con el host por wi-fi, y el robot con el portátil por ethernet. Por lo que ha sido necesario entender el funcionamiento del protocolo de comunicaciones inalámbricas y el funcionamiento del driver de ethernet. Otra comunicación muy importante para el proyecto es la comunicación por linea serie, la cuál es la forma de la que se comunica la aplicación que está en el robot con los

dispositivos del mismo (láser y microcontrolador).

D.2. Pruebas de integración

En la aplicación del robot también se realizó mucho esfuerzo en las pruebas de integración es decir, comprobar el correcto funcionamiento de la aplicación integrada con cada una de las funcionalidades.

D.2.1. Problemas con el cambio de contexto

En la integración de nuevas funcionalidades también se encontraron algunos problemas, pero estos problemas no eran problemas de funcionalidad, algo que antes funcionaba correctamente, después no lo hacía.

Tras un tiempo investigando el error, se descubrió que éste se producía al realizar el cambio de contexto de una tarea a otra. De forma que cuando se tenían muchas variables locales a una tarea al tener que cargarlas en memoria todas ellas de nuevo, el sistema operativo (MaRTE OS), no era capaz de hacerlo y producía un error. La solución adoptada fué transformar estas variables locales a variables globales, de forma que en el cambio de contexto ya no las tenía que cargar en memoria porque eran globales. Se intentó incrementar la pila de memoria asociada a cada tarea pero la solución no funcionó, probablemente porque, como se acaba de decir, fuese un problema propio del sistema operativo.

D.3. Pruebas de cumplimiento de plazos

Particularmente para una aplicación con restricciones de tiempo real una de las pruebas más importantes son aquellas que tienen como objetivo la comprobación de que todas las tareas de la aplicación cumplen plazos, es decir, que todas ellas se ejecutan dentro de su periodo. Estas pruebas, aunque gracias al análisis del cumplimiento de plazos C.1 serían innecesarias, es conveniente que se hagan para comprobar que realmente el análisis de cumplimiento de plazos se ha realizado correctamente y que los tiempos de ejecución y de bloqueo se han medido de forma correcta.

Así se dedicó tiempo en ejecutar la aplicación del robot en diferentes entornos comprobando que en ningún momento se producían violaciones del plazo de respuesta para cualquier tarea de la aplicación.

D.4. Pruebas reales

Por último, y por la propia naturaleza del proyecto, también se realizaron pruebas de la aplicación en el entorno pensado para su ejecución, más en particular en el túnel de Somport en Canfranc. Teniendo, en todo momento, un comportamiento esperado y sin producirse ninguna violación de periodo para las tareas de la aplicación con restricciones de temporales.

Anexo E
Compilación y Ejecución

Para la correcta ejecución del proyecto es necesario seguir unos sencillos pasos para poder compilar correctamente cada una de las aplicaciones de las que consta.

E.1. Aplicación del portátil

Para que esta aplicación pueda funcionar correctamente es necesario evidentemente, tener conectada una cámara Hercules del laboratorio al puerto usb del portatil. También es necesario que el kernel del linux sea una version superior o igual a la 2.6.32 para que se pueda instalar el controlador de la cámara, el cual se trata del gspca_sonixj.

Para compilar la aplicación basta con ejecutar:

```
gcc main.c -o mprogram -lrt
```

Y para ejecutar:

```
sudo ./mprogram
```

Hay que tener en cuenta que en esta aplicación se necesita la MAC del portátil y la del robot, por lo que será necesario que antes de ejecutarla se modifiquen las variables correspondientes en el fichero fuente (src_mac, dest_mac) para que el funcionamiento sea el correcto. Así como también será necesario modificar el nombre de la interfaz de ethernet si es necesario, por defecto se conecta a *eth0*.

E.2. Aplicación del host

Para poder compilar esta aplicación, dado que utiliza el protocolo RT-WMP, lo primero a realizar es instalar el protocolo en el host. Una vez descargado del

repositorio, será necesario comipilarlo para linux, ya que se puede compilar para funcionar para linux o para trabajar sobre MaRTE OS, para ello se realizarán los siguientes comandos desde el directorio en el que se haya instalado el protocolo, por ejemplo $HOME/rt-wmp/.

```
autoreconf
autoconf
./configure -with-platform=linux_us -with-llcom=ath5k
make
```

Todos estos pasos para la compilación, así como otras configuraciones se pueden encontrar en el archivo README del protocolo ($HOME/rt-wmp/README).

Una vez que tenemos compilado el protocolo de manera correcta, lo siguiente es configurar la tarjeta wi-fi. Para la comunicación inalámbrica se ha utilizado, como se observa en la compilación, una tarjeta ath5k, para la cual se habian desarrollado los drivers necesarios para la comunicación tanto desde linux como desde MaRTE OS en un PFC anterior [6]. Para poder utilizarla lo primero será compilar el ath5k, por lo que desde la carpeta del ath5k, por ejemplo $HOME/ath5k/ath5k-2.6.32-linux ejecutamos:

```
make
```

lo que nos generará un archivo ath5k_raw.ko. Para instalarlo será necesario ejecutar:

```
sudo rmmod ath5k
sudo insmod ath5k_raw.ko
sudo ifconfig wlan0 up
```

El valor *wlan0* se corresponde con el nombre que se le da a la red, en mi caso le puse el nombre wlan0, puede tener otros nombres como eth1. Por último, antes de compilar el programa es necesario crear un directorio en la carpeta $HOME llamado .rt-wmp, en el cual va a haber dos ficheros (rt-wmp.cfg y rt-wmp.ll), los cuales van a ser leidos para configurar el rt-wmp.

Ahora ya se puede proceder a compilar la aplicación. Desde el directorio src, por ejemplo $HOME/roboVisualizer/src ejecutamos:

```
make
```

Y para ejecutar la aplicación:

```
sudo ./roboVisualizer numero_nodos
```

El parámetro *numero_ nodos* indica cuantos son los nodos que componen el sistema. En el número de nodos se encuentran el número de robots y este programa, por lo que será igual a num_robots+1.

E.3. Aplicación del robot

La aplicación del robot deberá ser compilada en un ordenador, como puede ser el mismo host, y posteriormente copiarla al robot para que el robot arranque de ella.

Lo primero de todo será la instalación de MaRTE OS en nuestro ordenador, para ello se descargará la última versión del sistema operativo, (esta aplicación se ha realizado sobre la versión 1.9). Una vez descargada, la descrompimimos en un directorio, por ejemplo $HOME/ en el cual creará un directorio $HOME/marte en el cual estarán todos los ficheros correspondientes. Dado que el driver del ath5k estaba recién terminado aún no estaba incorporado a la última versión de MaRTE OS, por lo que es necesario crear un link:

```
ln -s $HOME/ath5k/ath5k-2.3.26-marte $HOME/marte/x86_arch/drivers
```

En caso de que el driver ya esté introducido en la versión del sistema operativo no hace falta hacerlo.

Una vez hecho esto, lo siguiente es permitir a MaRTE acceder tanto a los drivers de la linea serie, para comunicarse con el sistema operativo del robot, como de ethernet, para comunicarse con el portátil. Por lo que accedemos al fichero $HOME/marte/arch_dependent_files/marte-kernel-devices_table.ads y se descomentan las lineas correspondientes a la linea serie y al driver de ethernet.

Dado que para este proyecto se han encontrado algunos problemas en los drivers de la linea serie, se deberá cambiar la carpeta serial_port de $HOME-/x86_arch/drivers por la proporcionada en este proyecto, así como también se deberá intercambiar la del p2os por la proporcionada en la documentación. Destacar también que cada robot tiene unas IRQs distintas para comunicarse por la linea serie para la odometría y el laser. En el archivo serial_port_driver_c.c de la carpeta del driver de la linea serie del proyecto es donde se definen estas IRQs y donde deberán ser modificadas en caso de que no se correspondan con las del robot.

Ahora, ya se puede proceder a la instalación de MaRTE OS, para ello seguimos las instrucciones del archivo INSTALL con el que viene la distribución del sistema operativo. Recuerde que tiene que tener instalado el compilador de Ada GNAT para que la compilación se pueda realizar correctamente. Una vez instalado MaRTE habrá que realizar las operaciones oportunas para que la aplicación del robot pueda comunicarse a través de la wi-fi con el protocolo RT-WMP, para ello se copiará la carpeta con_robovisualizer en el directorio $HOME/rt-wmp/examples/MaRTE_OS/ y se modificará el archivo $HOME/rt-wmp/configure.in añadiendo la siguiente linea en el apartado donde están todos los demás example/.

```
example/MaRTE_OS/con_roboVisualizer/Makefile
```

Con esto conseguimos que al compilar el propio RT-WMP ya se genere el Makefile correspondiente para nuestra aplicación del robot. Para compilar el RT-WMP para MaRTE OS basta con ejecutar desde el directorio $HOME/rt-wmp/

```
autoreconf
autoconf
./configure --with-platform=MaRTE_OS --with-llcom=ath5k
make
```

Para compilar la aplicación bastará con ejecutar make desde el directorio $HOME/rt-wmp/example/MaRTE_OS/con_robovisualizer

Una vez realizado todo esto ya tenemos nuestro ejecutable **mprogram** dispuesto para ejecutarse en el robot. Para copiarlo al robot basta con arrancar el robot en linux y copiarle el fichero. Para ello ejecutamos el comando:

```
sudo scp mprogram robotica@192.168.2.50:
```

Siendo *robotica* el nombre de usuario con el que se ha accedido al robot y *192.168.2.50* la IP correspondiente al robot. Ahora habría que copiar el ejecutable desde el HOME a /boot/ para que se pueda arrancar de él. Pero para poder arrancar desde el mprogram habrá que añadir una nueva entrada en el fichero /boot/grub/menu.lst añadiendo:

```
title         MaRTE_OS
root          (hd0, 1)
kernel        /boot/mprogram
```

Con todo esto ya se puede arrancar el robot desde MaRTE OS y la aplicación se ejecutará sobre un sistema operativo de tiempo real, cumpliendo en todo momento las restricciones temporales.

Anexo F
Manual de usuario

En este capítulo se va a realizar un pequeño manual de usuario para la aplicación. Aunque por su forma gráfica es muy fácil de entender es conveniente explicar brevemente su funcionamiento.

En el apéndice E ya se ha explicado detalladamente como se compila y ejecuta cada una de las tres aplicaciones de las que consta el proyecto. Una vez ejecutada la aplicación del portátil (el de la cámara), y la del robot se ejecuta la aplicación del host.

Una vez que se ejecuta la aplicación del host, `sudo ./roboVisualizer 2`, en la Figura F.1 se muestra la siguiente pantalla que compone la aplicacion.

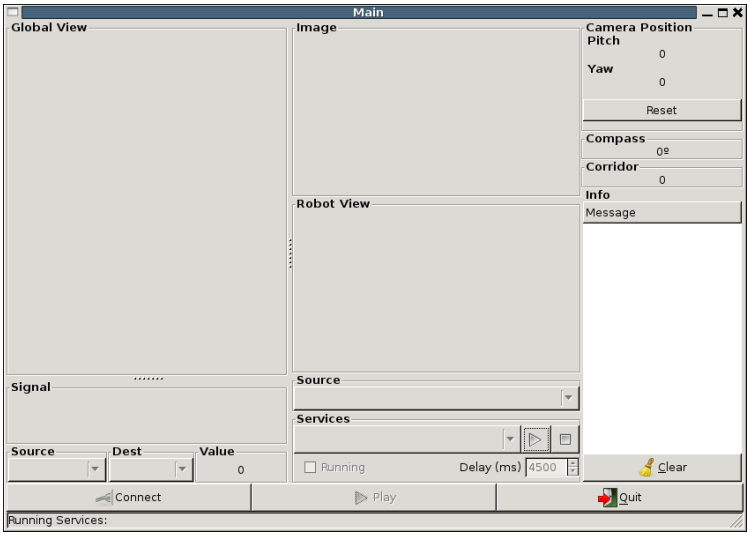


Figura F.1: Pantalla inicial de la aplicación

En esta imágen se puede observar que hay varias secciones, de las cuales no todas se van a utilizar para esta aplicación, ya que, como se ha comentado anteriormente, esta aplicación se trata de una plantilla que ha sido desarrollada por el grupo con el objetivo de que pueda servir para muchas y variadas aplicaciones de tratamiento de robots.

En primer lugar destacar el boton *Connect*, el cual es necesario pulsar para que la aplicación inicialice todo lo necesario, entre ello el protocolo RT-WMP, para lo cual es necesario el número de nodos del sistema, el cual se le da como parámetro al ejecutar. El botón *Quit*, como su propio nombre indica sirve para realizar la desconexión con los robots y terminar la ejecución del programa.

Otros botones que hay que destacar son los menús desplegables de *Sources* y *Services* que están encima del boton de *Quit*. En el primer menú, se puede elegir la fuente de la cual queremos coger los datos, entre ellas podemos encontrar todos los robots que se encuentren en el sistema. Cada robot se llamará de la siguiente forma robot_"numero de nodo". El otro menú, el de Servicios, indica los servicios que se pueden ejecutar, así como, gracias a los botones de *Play* y *Stop* que están adyacentes al menú, se puede poner en marcha un servicio o detenerlo si se considera oportuno.

Entre los servicios que estan disponibles se encuentran los siguientes:

- **Láser**: Este servicio está activo por defecto y su objetivo es mandar órdenes al robot seleccionado de recibir la información correspondiente a sus laserazos. Esta acción se realiza con un periodo de 400 ms por defecto, el indicado en el cuadro de texto Delay cuando está seleccionado este servicio. El periodo de cualquier servicio se puede modificar, pero, dado que se tienen datos nuevos del láser cada 400 ms no tiene sentido pedir el valor de los laserazos más rápido ya que no va a haber valores nuevos. La información de todos los puntos que se devuelven de los laserazos se representan en la ventana Global View, esto se puede ver en la Figura F.2.
- **Pose**: Este servicio proporciona los valores de odometría de todos los robots del sistema. Este servicio está desactivado por defecto con un periodo de 1 segundo. La información de la posición de todos los robots se representa en la ventana llamada Robot View.
- **Photo**: Este servicio se corresponde con la petición de una fotografía. Dado que las fotografías se van a pedir bajo demanda, a diferencia de los dos casos anteriores no se corresponden con tareas periódicas, por lo que al realizar una fotografía el servicio se vuelve a desconectar, por lo que será necesario

volver a pulsar el botón de *Play* si se desea otra foto. La imágen, al igual que la información del laser o de la odometría, también se muestra gráficamente en la aplicación, en este caso en la ventana Image, se puede observar en la Figura F.2.

- **GoAhead, StopGoingAhead**: El servicio de GoAhead se corresponde con la funcionalidad de navegación autónoma por medio de un pasillo. Al igual que en el caso anterior no se trata de una tarea periódica, pero, por el contrario, si que se queda ejecutando, por lo que será necesario activar el servicio de StopGoingAhead para que el robot deje la navegación autónoma por medio del pasillo.
- **MousePointer**: Este servicio es diferente a todos los demás, ya que el hecho de activarlo no hace nada, sino que permite que se quede bloqueado esperando un evento. Este evento no es más que pulsar con el ratón un punto en la ventana de Global View, en la cual se muestra la información del láser del robot seleccionado, indicando un objetivo al cual se desea que se dirija el robot.

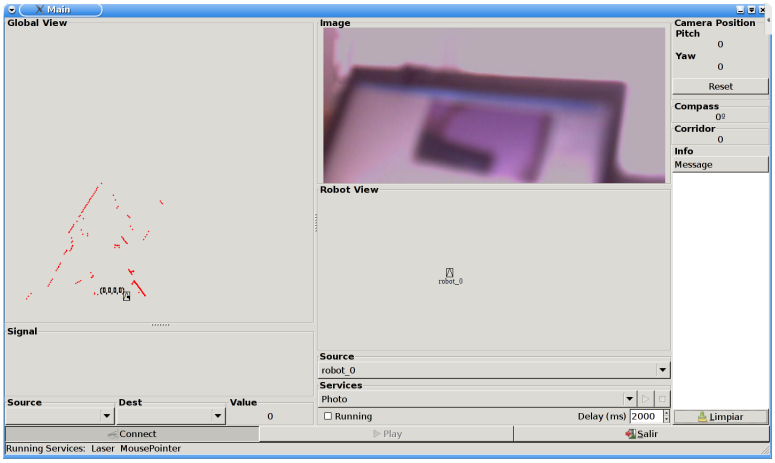


Figura F.2: Imágen de la aplicación con el láser y la foto

Anexo G
Nearness Diagram Navigation

Para la navegación con evitación de obstáculos, como se ha comentado en capítulos anteriores, se utiliza el algoritmo ND (Nearness Diagram Navigation [4]) que permite la navegación hasta un objetivo evitando los obstáculos que se encuentren en su camino.

El método de navegación se puede dividir en tres etapas. En la primera, gracias a la información del entorno se crean dos diagramas de proximidad, el PND (Nearness Diagram from the central Point) y el RND (Nearness Diagram from the Robot). En segundo lugar, el PND es analizado para identificar las regiones existentes y seleccionar una de ellas. Por último, el RND es analizado para evaluar la situación de seguridad en la que se encuentra el robot, es decir, el riesgo potencial de una colisión.

G.1. Análisis PND

El análisis PND, como se acaba de explicar, representa la proximidad de los obstáculos al punto central del robot y es utilizado para extraer la información de las características del entorno (los obstáculos). El análisis PND se realiza en tres etapas. En primer lugar se buscan *espacios libres* en el entorno, después, de ellas se obtienen regiones de espacio libre y finalmente se elige una de ellas según algún criterio.

Un concepto importante a la hora de explicar el algoritmo es el de **discontinuidad**. Entre dos sectores adyacentes hay una discontinuidad si su diferencia de altura en el PND es superior a un valor considerado. Las discontinuidades representan espacios libres entre los obstáculos o espacios libres debidos al final de un obstáculo.

Un conjunto de sectores constituyen un **valle**, el cual tiene un sector izquierdo y otro derecho como extremos, que satisface las siguientes características:

1. Todos los sectores son adyacentes y no hay discontinuidades entre ellos.
2. Si se consideran los sectores adyacentes externos a los extremos del valle, hay dos discontinuidades, una entre el sector extremo izquierdo y el sector adyacente fuera del valle, y otra análogamente con el extremo derecho.

Un caso especial ocurre cuando el robot tiene el objetivo entre él y un obstáculo, ya que el sector que contiene el objetivo no pertenece a un valle. Cuando esta situación es detectada, el algoritmo crea un valle ficticio en el sector del objetivo. Destacar que los valles representan las regiones entre los obstáculos.

El criterio para elegir un valle es seleccionar aquél que tiene una discontinuidad con la distancia mínima sectorial al sector en el que se encuentra la localización del objetivo. Dependiendo de la fuente de información y la geometría del robot, varias estrategias pueden ser propuestas para, si es posible, alcanzar el espacio del área libre por el cual pasar.

G.2. Análisis RND

El análisis RND evalúa la situación de seguridad del robot. Se deben introducir dos conceptos en este momento:

- La distancia de seguridad: que se corresponde con la distancia mínima tolerable a un obstáculo.
- La proximidad de seguridad: que se corresponde con la proximidad máxima tolerable a un obstáculo. Este el valor en el RND, a partir de este valor, el robot se encuentra demasiado cerca de un obstáculo.

Hay dos situaciones de seguridad en las cuales el robot se puede encontrar: LS (Low Safety situation) y HS (High Safety situation).

El robot se encuentra en LS si, en el RND, al menos un sector excede la proximidad de seguridad. Esto significa que la distancia entre el borde del robot y el obstáculo es menor que la distancia de seguridad, siendo esto un riesgo potencial para el robot. En otro caso se encuentra en HS.

G.3. Estrategia de navegación

El ND utiliza cinco leyes de movimiento, adaptada de las 5 situaciones obtenidas en la etapa de análisis e interpretación, en la cual la localización del objetivo es usada directamente sólo en una de ellas. El ND calcula la velocidad transancional y rotacional en cada muestra de tiempo como comandos de movimiento para un robot móvil holonomo, es decir, que no puede girar las ruedas.

G.3.1. Navegación en situaciones de baja seguridad

En situaciones LS el robot está en peligro de colisión, por lo que la solución pasa por poner el robot en una situación segura.

- **Navegación en LS1**: En este caso hay obstáculos cerca de la distancia de seguridad, pero éstos se encuentran sólo a un lado del área libre seleccionada, la cual se encuentra cerca del objetivo. La solución es producir un movimiento que deje el obstáculo fuera de la zona de seguridad, mientras que avanza por el espacio seleccionado. En LS1 el objetivo principal es alejar el robot del obstáculo más cercano, mientras se mueve hacia el espacio del área libre.
- **Navegación en LS2**: En este caso hay obstáculos dentro de la distancia de seguridad del robot a ambos lados de la región libre seleccionada. El objetivo principal es mantener al robot a la misma distancia de los dos obstáculos más cercanos, mientras avanza hacia el espacio seleccionado en busca del objetivo.

G.3.2. Navegación en situaciones de alta seguridad

En situaciones de alta seguridad (HS) el robot no está en peligro de colisión, por lo que las soluciones son elegidas dentro del área libre.

- **Navegación en HSGV (High Safety Goal in Valley)**: En este caso el objetivo se encuentra dentro del área libre. Por lo que la solución es mover el robot hacia el objetivo. En esta situación se usa directamente la localización del objetivo para calcular los comandos de movimiento. Esta navegación conduce el robot directamente al objetivo.
- **Navegación en HSWV (High Safety Wide Valley)**: En este caso el sector del objetivo no se encuentra en el área libre seleccionada, siendo ésta muy ancha. La solución pasa por tener un comportamiento de seguimiento del contorno. Esta navegación da como resultado un movimiento a lo largo del obstáculo hasta llegar al objetivo.

- **Navegación en HSNV (High Safety Narrow Valley)**: En el HSNV el sector del objetivo no se encuentra dentro del área seleccionada, al igual que en el caso anterior, pero en este caso el espacio libre es estrecho. Luego, la solución es moverse hacia el centro del área libre. Esta navegación dirige el robot entre los obstáculos.

Una vez realizadas todas las operaciones, el algoritmo devuelve la velocidad, tanto transaccional como rotacional, a la que se tiene que mover el robot para alcanzar el objetivo. Este algoritmo se tendrá que ejecutar periódicamente cada vez que haya nuevos datos del entorno para que la navegación se realize de una manera correcta.

Anexo H
Simultaneous Localization And Mapping

El algoritmo del SLAM (Simultaneous Localization And Mapping [5]) es el algoritmo encargado de realizar la localización del robot en un mapa ya existente y al cual se tiene acceso. En este mapa está disponible los puntos origen y fin de todas las rectas que lo componen, por lo que se tiene potencialmente acceso a todas las características del entorno.

El SLAM al igual que ocurría con el algoritmo del ND, se debe ejecutar periódicamente, ya que lo que proporciona como resultado se corresponde con la localización del robot en el mapa en un instante determinado, con un entorno específico. Por lo tanto será necesario que se ejecute de esta forma para tener siempre la posición del robot lo más actualizada posible con la información actual del entorno.

El proceso del SLAM consiste de varias etapas: la extracción de las características, la asociación de los datos, la estimación del estado y la actualización del mismo.

El objetivo del proceso es utilizar el entorno para actualizar la posición del robot. Dado que la odometría del robot, como se ha comentado en varias ocasiones, a veces es errónea, no podemos confiar directamente en ella. Por lo tanto se va a utilizar los valores que el láser devuelve del entorno para corregir la posición del robot. Esto se logra extrayendo las características del entorno (en nuestro caso estas caracetrísticas se corresponden con las paredes del entorno) y reobservándolas cuando el robot se vuelve a mover.

El EKF (Extended Kalman Filter) es el centro del proceso del SLAM. Él es el responsable de actualizar dónde se encuentra el robot bansándose en esas características. El EKF realiza un seguimiento de la estimación de la incertidumbre en

la posición del robot y también en la incertidumbre de esas observaciones que se han visto del entorno. En la figura H.1 se puede observar una visión general del proceso del SLAM.

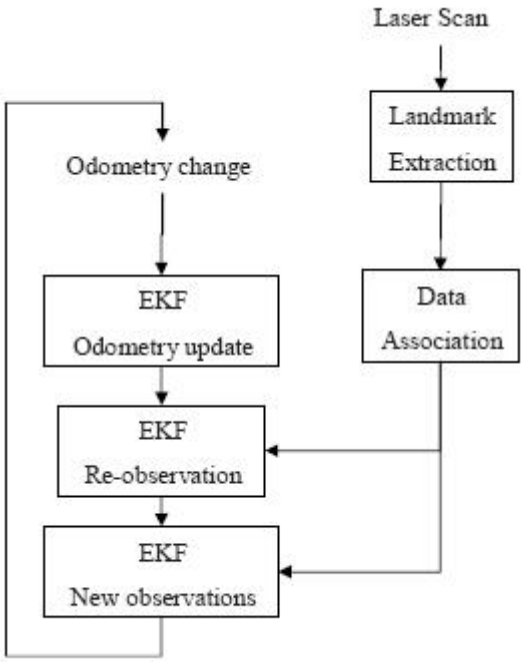


Figura H.1: Visión general del proceso del SLAM

Una vez que la odometría se modifica debido al movimiento del robot, se actualiza la incertidumbre relativa a la nueva posición del robot en EKF gracias a la odometría actual. Entonces, se extraen las nuevas características del entorno proporcionadas por el láser, y a continuación se procede al emparejamiento de estas observaciones con las características obtenidas en la iteración anterior. Las características que se logran emparejar con las características sirven para actualizar la posición del robot, mientras que las que aparecen nuevas, es decir, las que no se habían observado en la iteración anterior, se añaden como nuevas características. En todo momento de este algoritmo, es decir, cada vez que se intenta emparejar una observación con otra característica el EKF posee una estimación de la posición actual del robot, que se irá ajustando conforme se analicen más observaciones.

H.1. Extracción de características

En el apartado de extracción de las características es dónde entra en juego los sensores de los que dispone el robot móvil. Gracias al laser SICK LMS 200 que lleva incorporado se pueden conseguir los puntos que indican la distancia que se encuentra de los obstáculos, transformándo estos puntos en rectas indicarán, en algunos casos, las paredes que observa el robot. Todas estas rectas se formarán de acuerdo a unos parámetros, como por ejemplo, que entre dos valores no haya una distancia mayor a cierta cantidad, o que todos los puntos se aproximen a una recta por mínimos cuadrados con un error máximo menor a otro valor. Estas rectas se corresponderán con las **observaciones** del robot.

H.2. Asociación de datos

En este apartado se procede al emparejamiento (*matching*) de las observaciones del robot, es decir, las rectas observadas con el láser, con las características del mapa, es decir, las rectas que componen el mapa en el que se encuentra el robot.

Para emparejar una observación con una característica se utiliza una técnica llamada aproximación al vecino más cercano (*nearest-neighbor approach*). Para obtener este emparejamiento se utiliza como test de hipótesis la distancia de Mahalanobis, en vez de la distancia euclídea, ya que la primera, a diferencia de la segunda, tiene en cuenta la correlación entre las variables, por lo que proporciona un resultado mucho más aproximado a la realidad.

H.3. Estimación y actualización del estado

El filtro de Kalman Extendido se usa para estimar la posición del robot a partir de los datos de odometría y las observaciones. Tan pronto como se extraen las características y se asocian los datos, en el SLAM se pueden considerar tres etapas:

1. Actualizar el estado actual estimado usando la odometría
2. Actualizar el estado estimado gracias a las características reobservadas
3. Añadir nuevas características al estado actual.

La primera etapa es muy simple. Basta con obtener el desplazamiento del robot basándose en la odometría, para ello se calcula el movimiento (la diferencia), entre la odometría actual y la de la iteración anterior. Y posteriormente se calcula la estimación de la posición aplicando ese movimiento de la odometría a la localización que se había obtenido en la iteración anterior.

En la segunda etapa se consideran las características reobservadas. Usando la estimación de la actual posición del robot es posible estimar la posición en la que se tendría que encontrar dichas características. En esta etapa la incertidumbre de cada característica también se actualiza para reflejar los cambios recientes.

Por último, en la tercera etapa las nuevas características son añadidas al mapa del mundo del robot. Esto se hace usando la información sobre la posición actual y añadiendo información sobre la relación entre las nuevas y las viejas características.

Bibliografía

[1] http://marte.unican.es/

[2] http://www.mobilerobots.com

[3] Danilo Tardioli, José Luis Villarroel "Real Time Communications over 802.11: RT-WMP."

[4] J. Minguez, L. Montano "Nearness Diagram Navigation (ND): A new real time collision avoidance approach."

[5] Hugh Durrant-Whyte, Fellow, IEEE, and Tim Bailey "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms."

[6] Samuel Cabrero, Danilo Tardioli, José Luis Villarroel "Nodo de red ad-hoc con soporte de tráfico de tiempo real y calidad de servicio para comunicaciones inalámbricas en túneles."