RealSim: Simulador Multirobot Genérico

Antidio Viguria Iván Maza

Índice general

1. Introducción **3**
1.1. Introducción . . . 3
1.2. Objetivos . . . 3

2. Descripción de la arquitectura **4**
2.1. Arquitectura general del sistema . . . 4
2.2. Centro de Control (CC) . . . 5
2.2.1. Lenguaje para la especificación de misiones y tareas . . . 5
2.2.2. Operaciones sobre tareas y estado de las mismas . . . 7
2.2.3. Interfaz gráfica . . . 7
2.2.4. Interfaz en modo texto . . . 9
2.2.5. Modos de funcionamiento . . . 9
2.3. Servidor de tiempos (TS) . . . 10
2.4. Arquitectura general para un robot heterogéneo . . . 10
2.4.1. RAL (Robot Abstraction Layer) . . . 12
2.4.2. MML (Module Manager Layer) . . . 15
2.4.3. RIL (Robot Implementation Layer) . . . 16
2.5. Simulación de la arquitectura . . . 21

3. Conclusiones y desarrollos futuros **26**
3.1. Conclusiones . . . 26
3.2. Desarrollos futuros . . . 27

Índice de figuras

2.1. Entidades que forman la arquitectura del sistema 5
2.2. En esta figura se observa la interfaz gráfica del CC con un submenú para el envío de una tarea. 8
2.3. En esta figura se observa la ventana LogWindow con información relativa al estado de las tareas durante una misión. 9
2.4. Arquitectura de un robot dentro del sistema completo 11
2.5. Módulos que forman la Robot Abstraction Layer (RAL) 12
2.6. Módulos que forman el Module Manager Layer (MML) 15
2.7. Módulos que forman el Robot Implementation Layer (RIL) 17
2.8. Entradas y salidas de un módulo genérico del RIL 17
2.9. Vista frontal y lateral del vehículo autónomo Romeo-4R 18
2.10. Estructura de los módulos del RIL en el robot Romeo 19
2.11. Vista lateral del vehículo autónomo Hero 20
2.12. Estructura de los módulos del RIL en el robot Hero 21
2.13. Estructura al día de hoy de los módulos del RIL en el robot Hero 22
2.14. Estructura de los módulos del RIL en los robots simulados 22
2.15. Diferencias en el RIL entre simulación y realidad 23
2.16. Módulos utilizados en simulación en el RIL de Romeo-4R 24
2.17. Módulos utilizados en simulación en el RIL de HERO 25

3.1. Visión personal del futuro de la arquitectura 28

Capítulo 1

Introducción

1.1. Introducción

Cuando se trabaja con varios robots heterogéneos empieza a ser importante la definición de una arquitectura común entre todos los robots, esto no sólo permitirá una mayor organización y una mayor rapidez a la hora de implementar un nuevo algoritmo en un robot, sino que además permitirá crear un nivel de abstracción común a todos los robots y por lo tanto desde el nivel más alto todos los robots se tratarán de la misma manera. Esto facilita enormemente la cooperación y coordinación entre los robots e incluso la introducción de nuevos robots en el sistema.

1.2. Objetivos

El objetivo principal de este documento es explicar de forma resumida la arquitectura del sistema y que a su vez este documento sirva de introducción a cualquier persona que quiera implementar una nueva funcionalidad en un robot que siga esta arquitectura.

Capítulo 2

Descripción de la arquitectura

2.1. Arquitectura general del sistema

El sistema completo está formado por las siguientes entidades (ver Figura 2.1): el centro de control (CC), un servidor de tiempos (TS), un módulo de representación en 3D (3DRepresentation) y un conjunto de robots. La comunicación entre estas entidades se realiza a partir de la interfaz GRI (General Robot Interface), que se detalla más adelante en este documento.

Las funcionalidades básicas de estas entidades son las siguientes:

Control Center (CC): permite tener una visión global del conjunto de robots. Los robots tienen una interfaz estándar mediante la que le comunican al CC su estado y el estado de ejecución de sus tareas. Asimismo, desde el CC es posible el envío de diversos tipos de tareas a los robots, así como el control de la ejecución de misiones completas.

TimeServer (TS): se utiliza para tener una base de tiempo común en todas las entidades del sistema. De esta forma por ejemplo, los históricos de las acciones de distintos robots tienen marcas de tiempo comunes. Asimismo, el TS permite la ejecución de acciones de manera sincronizada por parte de los robots.

Módulo de representación 3D (3DRepresentation): permite visualizar la ejecución de una misión del equipo de robots en tiempo real en un escenario tridimensional. Se puede modificar tanto el modelo 3D del escenario, como los modelos 3D de cada uno de los robots. Asimismo, este módulo también se encarga de grabar el estado de todos los robots durante la misión y permite visualizar el desarrollo de la misma fuera de línea de cara a analizar el correcto comportamiento de algoritmos tales como los que se encargan de la asignación distribuida de tareas.

Equipo de robots: la arquitectura de cada robot es jerárquica y está organizada por capas. Dentro de cada capa se pueden encontrar uno o varios módulos software. Por otra parte cada robot intercambia información con el resto de robots, el centro de control, el módulo de representación 3D y el servidor de tiempos.

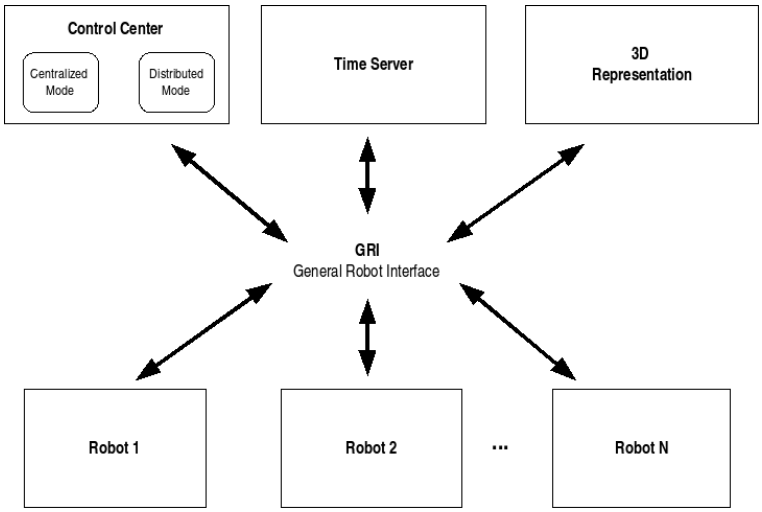

Figura 2.1: Entidades que forman la arquitectura del sistema

2.2. Centro de Control (CC)

El centro de control permite lanzar una misión para el equipo de robots. El usuario del centro de control puede diseñar la misión como un conjunto de tareas especificadas en un determinado formato. Asimismo, el centro de control permite supervisar la marcha de una misión en curso mediante las ventanas que componen su interfaz gráfica. También es posible el envío de tareas en línea a los robots durante la ejecución de una misión.

En las siguientes secciones se pasan a describir las principales características del centro de control, comenzando por el lenguaje de especificación de tareas en el que se basa el funcionamiento del mismo.

2.2.1. Lenguaje para la especificación de misiones y tareas

Se ha desarrollado un lenguaje para la especificación de las tareas y de las misiones. La descripción de ese lenguaje escapa al ámbito de este documento y solamente se van a comentar algunas sentencias del mismo. Por ejemplo, la sentencia para indicar a un robot que empiece una tarea es la siguiente:

```
x number_precond <preconditions> command <param> robot_id
```

donde `command` es un identificador asociado la tarea. En la siguiente tabla se muestra la lista de las tareas posibles que se pueden asignar a un robot y el código asociado a cada una de ellas en el lenguaje desarrollado:

Cada una de estas tareas tiene un conjunto de parámetros asociado. Por ejemplo, para la tarea GOTO_XYZ los parámetros son las coordenadas X, Y y Z del punto a visitar y la velocidad para el trayecto. Asimismo, es posible establecer precondiciones para la ejecución de una tarea.

Para finalizar, se va a explicar una sentencia del lenguaje a modo de ejemplo de lo anteriormente expuesto:

Id. Tarea	***Tarea***	***Acción***
1	GOTO_XYZ	El robot se debe mover desde la posición actual al punto de destino a la velocidad especificada y alcanzarlo con velocidad 0.
2	GOTOLIST_-XYZ	El robot debe visitar desde la posición actual una lista de puntos para los que se especifican también las velocidades de paso. El robot no se detiene hasta llegar al último punto de la lista.
3	WAIT	El robot se queda parado hasta que la tarea es abortada.
4	HOME	Es equivalente a una tarea GOTO_XYZ al punto donde el robot comenzó la misión.
5	TAKE_OFF	El robot despega si está capacitado para ello.
6	LANDING	El robot aterriza si es un robot aéreo y se encuentra volando.
7	PAN_TILT	Mueve el pan&tilt que va a bordo del robot hasta alcanzar una determinada orientación.
8	GET_IMAGES	El robot empieza a tomar imágenes.
9	DETECT	Se pasa a modo detección en el procesamiento de las imágenes que se van captando.
10	TRACKING	El robot sigue un determinado objeto móvil.

Cuadro 2.1: Lista de tareas posibles.

Operación	*Comentario*
START	Ejecutar la tarea.
ABORT	Abortar la tarea si es posible. En caso de que no sea posible aparecerá un error en el estado de la tarea.
SUPPRESS	Deja de transmitir el estado de la tarea.

Cuadro 2.2: Lista de operaciones que se pueden hacer sobre las tareas.

```
x 1 765321.175 4145050.933 40.0 3.0 3
```

Esta sentencia especifica que el robot con identificador 3 comience a ejecutar una tarea GOTO_XYZ sin precondiciones. Esa tarea tiene cuatro parámetros: las coordenadas GPS del punto dadas como latitud, longitud y altitud sobre el nivel del mar (765321.175, 4145050.933, 40.0), y la velocidad para el trayecto (3.0 m/s).

Las sentencias del lenguaje pueden agruparse en ficheros de texto plano para la especificación de misiones completas. Esos ficheros se pueden cargar desde el centro de control para su ejecución haciendo uso de las opciones de la interfaz gráfica.

2.2.2. Operaciones sobre tareas y estado de las mismas

En la tabla 2.1, aparecen las tareas que se pueden enviar a los robots. Las operaciones que se pueden hacer sobre esas tareas son iniciarlas, abortarlas o suprimirlas. La descripción de estas operaciones aparece en la tabla 2.2.

Por otra parte, cada tarea puede encontrarse en distintos estados a lo largo de la ejecución de una misión. Los estados que se han considerado suficientes para la gestión de las tareas aparecen listados y descritos en la tabla 2.3.

2.2.3. Interfaz gráfica

La interfaz gráfica del centro de control ha sido desarrollada empleando `Gtk2.0` y la librería `visface` del Grupo de Robótica, Visión y Control. Está compuesta fundamentalmente por tres ventanas como se puede apreciar en las figuras 2.2 y 2.3:

MainWindow: Ventana con el menu principal desde el que se pueden cargar misiones desde archivos o enviar tareas específicas a los distintos robots. Una vez que se selecciona una opción, se abre una ventana en la que se pueden especificar los parámetros relativos a cada opción.

MapWindow: Ventana donde se puede visualizar el mapa del entorno, junto con las posiciones, orientaciones e identificadores de los robots. Asimismo, en esta ventana también se muestran los puntos a visitar por los robots indicando mediante distintos colores si el punto se ha visitado o no.

Estado	***Descripción***
EMPTY	No hay asociado ningún estado a esta tarea
SCHEDULED	La tarea está programada y está esperando a que se terminen de ejecutar otras tareas.
RUNNING	La tarea se está ejecutando.
ABORTING	La tarea está siendo abortada. Si finalmente es abortada se pasará al estado ABORTED. De lo contrario, volverá al estado RUNNING y tendrá asociada un error.
ABORTED	La tarea ha sido abortada y ha dejado de ejecutarse. Esto puede pasar porque haya sido abortada o porque el robot no haya podido terminar la tarea con éxito.
ENDED	La tarea ha finalizado correctamente.

Cuadro 2.3: Lista de estados posibles para las tareas.

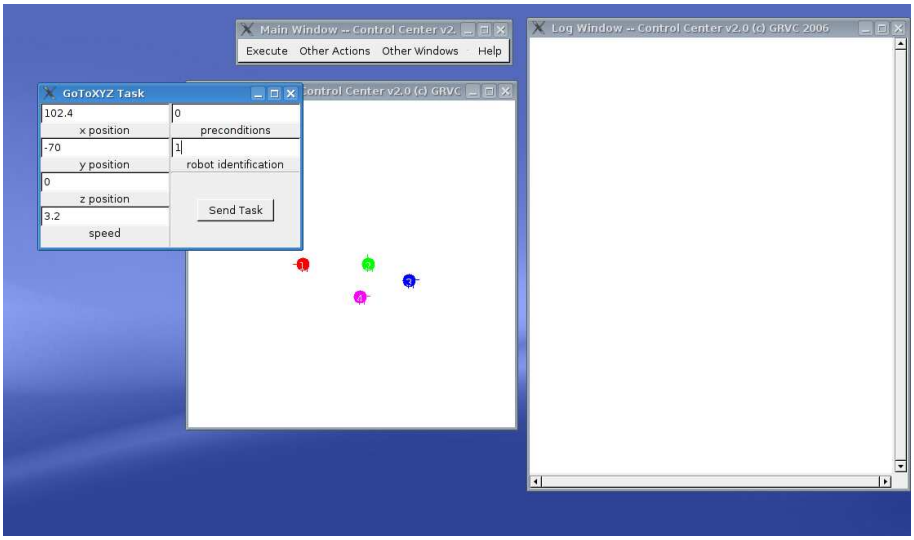

Figura 2.2: En esta figura se observa la interfaz gráfica del CC con un submenú para el envío de una tarea.

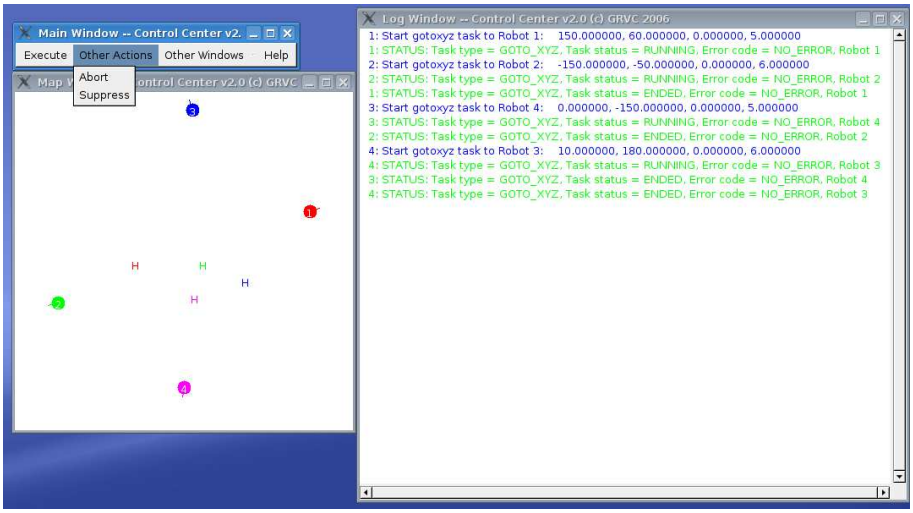

Figura 2.3: En esta figura se observa la ventana LogWindow con información relativa al estado de las tareas durante una misión.

LogWindow: Ventana en la que se va mostrando el estado de ejecución de las distintas tareas que componen la misión. Además, el centro de control guarda en ficheros el estado de cada robot, las tareas que se le han enviado y el estado de éstas en cada momento.

2.2.4. Interfaz en modo texto

De cara a poder ejecutar el centro de control en sistemas que no posean interfaz gráfica, se decidió desarrollar una interfaz auxiliar en modo texto para el envío de misiones al conjunto de robots. En dicha interfaz se dispone de una ayuda que le indica al usuario el formato de los comandos que puede ejecutar. Desde la interfaz en modo texto se pueden enviar misiones almacenadas en fichero o enviar tareas específicas a un determinado robot del equipo.

2.2.5. Modos de funcionamiento

Se pueden especificar dos modos de funcionamiento para el centro de control:

Centralizado: En este modo, la asignación de tareas a los robots la hace el operador humano. Es decir, se especifica qué robot hace qué tarea de las que componen la misión de manera manual.

Distribuido: En este caso, los robots deciden de manera autónoma y distribuida qué tarea hace cada uno de ellos. Para ello, emplean un algoritmo de negociación distribuida.

Señalar que el funcionamiento centralizado o distribuido puede combinarse sin tener que reiniciar el software: se pueden mandar tareas directamente a un robot para que las ejecute o lanzar el algoritmo de negociación para que se las repartan automáticamente entre ellos.

2.3. Servidor de tiempos (TS)

Esta entidad se encarga de transmitir una base de tiempo común a todas las entidades del sistema (robots, CC y módulo de representación 3D). De esta forma:

- Todos los datos que se registran durante la ejecución de cada misión (estado de cada robot, estado de las tareas, etc.) tienen la misma referencia de tiempos para su posterior análisis y tratamiento.
- Se logra la sincronización en la ejecución de tareas de distintos robots entre las que hay dependencias.
- Durante el proceso de negociación distribuida, es posible implementar tiempos de espera máximos comunes para todos los robots.

Por último, señalar que el servidor de tiempos sólo transmite datos y no recibe información de ninguna otra entidad.

2.4. Arquitectura general para un robot heterogéneo

Se ha optado por una arquitectura jerárquica organizada en capas, en la que a medida que vamos de las capas inferiores a las superiores, disminuye el grado de dependencia con las características más específicas del robot. Entre todas las capas existen interfaces comunes para todos los robots. De esta forma, aunque la implementación de las capas inferiores sea distinta en cada robot, las interfaces serán completamente independientes. Finalmente, comentar que cada capa está formada por uno o varios módulos software.

Las distintas capas son:

1. **RAL (Robot Abstraction Layer)**: esta capa permite que todos los robots se vean igual desde el centro de control u otras entidades del mismo nivel. Actualmente esta capa se ocupa de la gestión de las tareas enviadas por el centro de control, del protocolo para la asignación distribuida de tareas y de la sincronización entre las tareas. También es capaz de integrar información sensorial del entorno de varios robots para aplicaciones de evitación de colisiones con otros robots, generación de mapas de forma distribuida, etc. Asimismo, comentar que esta capa es independiente del robot por lo que la misma implementación debe servir para todos los robots.

2. **MML (Module Manager Layer)**: la implementación de este módulo depende del tipo de robot. La función de esta capa es la de configurar y conectar los distintos módulos que componen la última capa (RIL) con el objetivo de que el robot ejecute la tarea que se le ha asignado. Además, se encarga de traducir la telemetría del robot a un estado genérico común a todos los robots y transmitirla al centro de control u otras entidades superiores.

3. **RIL (Robot Implementation Layer)**: esta capa está formada por varios módulos que incluso pueden llegar a ser procesos independientes en distintas máquinas. Cada uno de estos módulos se ocupa de una función específica, como por ejemplo la generación de un camino, el movimiento del pan&tilt, etc.

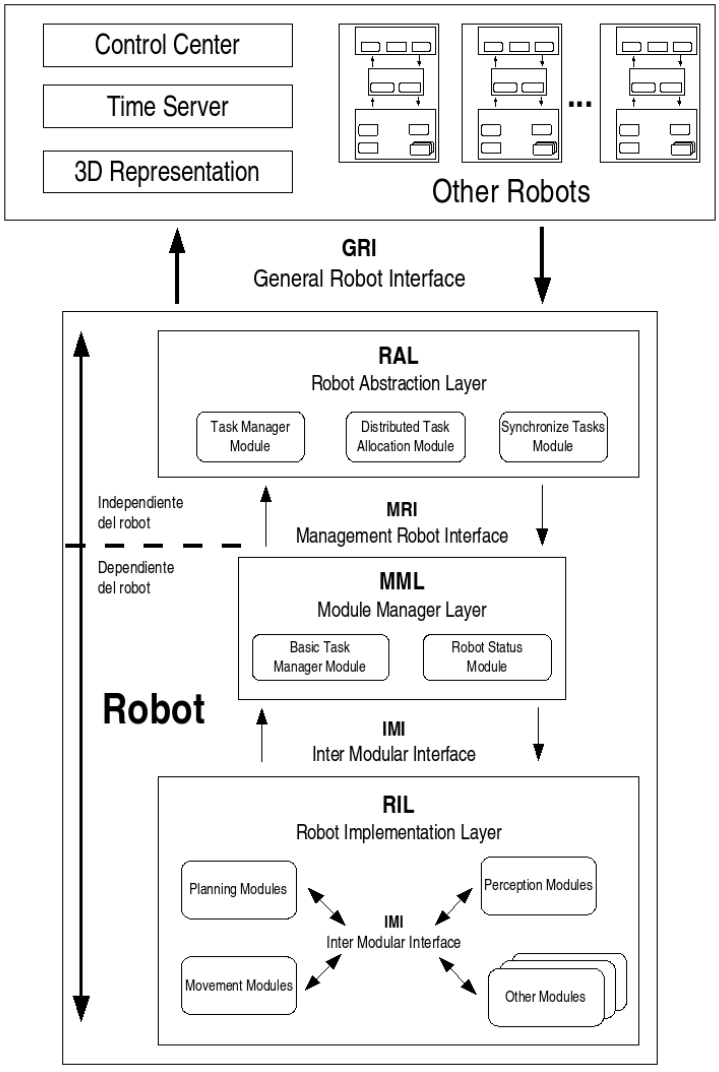

Figura 2.4: Arquitectura de un robot dentro del sistema completo

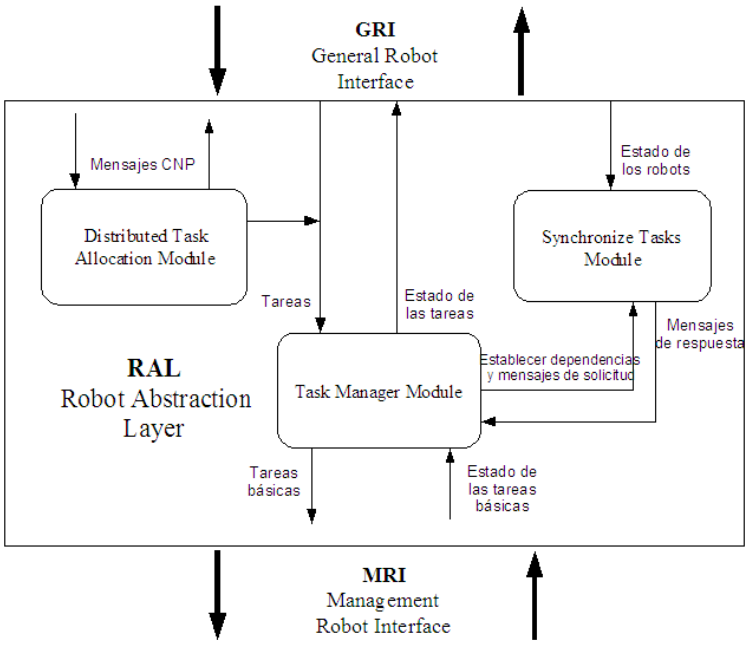

Figura 2.5: Módulos que forman la Robot Abstraction Layer (RAL)

Entre cada una de estas capas existe una interfaz que es independiente del robot:

1. **GRI (Generic Robot Interface)**: interfaz entre el centro de control, el servidor de tiempos y las distintas capas RAL de los robots. Esta interfaz está formada por las tareas de alto nivel y su estado, los mensajes CNP (Contract Net Protocol) y el reloj del servidor de tiempos. Esta interfaz es completamente genérica para todos los robots.

2. **MRI (Management Robot Interface)**: interfaz entre el RAL y el MML que está formada por las tareas básicas y sus respectivos estados.

3. **IMI (Inter Modular Interface)**: interfaz entre el MML y el RIL. Esta interfaz es la que se utiliza para configurar los distintos módulos y sus respectivas conexiones.

En las siguientes secciones se va a explicar con mayor detalle tanto cada una de las capas como de las interfaces.

2.4.1. RAL (Robot Abstraction Layer)

Esta capa es independiente del robot y por lo tanto la misma implementación software sirve para todos ellos. Por otra parte, los módulos presentes en esta capa no tienen control de estado (parar un módulo, despertarlo, etc.) porque deben estar siempre funcionando, dado que son una parte fundamental de la arquitectura.

Esta capa está formada por tres módulos (ver figura 2.5):

- **CNP Manager**

  En este módulo se implementa el algoritmo de negociación para la asignación de tareas distribuidas. El funcionamiento de este protocolo se puede asimilar a un mercado donde cada agente (en este caso un robot) puja con un coste por cada tarea que se anuncia. Los pasos del algoritmo básico son:

  - Inicialmente uno de los robots publica una tarea en el sistema. Al comenzar la misión, ese cometido lo lleva acabo el CC, aunque durante la ejecución de la misión cualquier robot puede publicar nuevas tareas.
  - Cada uno de los robots calcula el coste que tiene la tarea publicada para él y lo publica. El coste se calcula como el tiempo en ejecutar la tarea, aunque se pueden seleccionar métricas más complejas.
  - El agente que ha publicado la tarea espera a que todos los robots le respondan durante un tiempo determinado. Esto permite que si un robot pierde las comunicaciones, el protocolo siga adelante. Dicho agente evalúa el mejor de los costes recibidos y lo compara con su propio coste para esa tarea. Finalmente, se le asigna la tarea al robot con mejor coste.

  Solamente se publica una tarea de cada agente en cada momento y para llevar el control de quién está publicando se utiliza un token. Por lo tanto para poder publicar una tarea hace falta estar en posesión del token, y en caso de no estar en posesión del mismo, es necesario solicitarlo.

  Las entradas y salidas del módulo son:

  - *Entradas*: este módulo sólo tiene como entradas los mensajes del protocolo CNP (GRI). Estos mensajes se dividen en 2 conjuntos: mensajes asociados a la negociación de tareas (GRI) y mensajes asociados con la gestión del token (GRI).
  - *Salidas*: la salida del módulo son los mensajes CNP intercambiados con los demás robots y las tareas ganadas al finalizar el algoritmo de asignación que le son enviadas al módulo TaskManager.

- **TaskManager Module** Este módulo se encarga de gestionar las tareas que debe ejecutar el robot. En primer lugar comentar que las tareas le pueden llegar del CC (modo de funcionamiento centralizado) o del CNPManager (modo de funcionamiento distribuido). Señalar que para este módulo esto es transparente y realmente no tiene conciencia del modo de funcionamiento que se está utilizando, de tal manera que incluso se podría utilizar un modo de funcionamiento mixto.

  Este módulo se encarga de gestionar las tareas en el sentido de controlar el orden de ejecución, de abortarlas en caso que sea necesario, etc. Además está preparado para gestionar tareas que se pueden ejecutar simultáneamente, como tareas de percepción y de movimiento. Hay tres tipos de tareas que se pueden gestionar en paralelo: percepción, movimiento y tareas relativas al pan&tilt. Además, el sistema está pensado para poder aumentar el número de tareas en paralelo que es capaz de gestionar con facilidad. Estas tareas son

traducidas a un conjunto de tareas básicas (actualmente la mayoría de las tareas básicas son una copia de la tarea de alto nivel). Por ejemplo, la traducción de una tarea HOME sería una tarea básica GOTO_XYZ. El TaskManager sólo comunica al MML una tarea por cada tipo de tarea que se puede gestionar en paralelo a la vez, por lo que hasta que no se termine una tarea básica no se le manda otra.

Por otra parte, este módulo se comunica internamente por medio de una sección crítica con el módulo SyncronizeTask que se encarga de la sincronización de tareas. Básicamente el funcionamiento es el siguiente:

- Cuando una tarea nueva llega al TaskManager, en caso de que tenga precondiciones, se le comunica al módulo SynchronizeTask el identificador de la tarea y sus precondiciones (que son identificadores de otras tareas) por medio de un mensaje tipo SET_DEPENDENCIES.
- Al recibir el módulo SynchronizeTask un mensaje del tipo SET_DEPENDENCIES, guarda esa información.
- En todo momento el módulo SynchronizeTask va leyendo del sistema de comunicaciones el estado de las tareas del propio robot y del resto. En el momento en que una tarea aparece como ENDED, elimina todas las precondiciones asociadas a dicha tarea.
- Antes de que el TaskManager comunique a la capa inferior la ejecución de una tarea, le pregunta al SynchronizeTask si ya se han cumplido todas las precondiciones y por lo tanto si se puede ejecutar la tarea. Esta acción se realiza mediante un mensaje tipo QUERY donde el único parámetro es el identificador de la tarea en cuestión.
- El SynchronizeTask responde con un mensaje tipo RESPONSE donde únicamente se comunica si la tarea con un identificador dado se puede ejecutar o no.
- En caso positivo, la tarea es transmitida a la capa inferior. De lo contrario, no se envía y se volverá a preguntar más adelante. Por lo tanto, la comprobación se realiza mediante polling.

Las entradas y salidas del módulo son:

- *Entradas*: tareas que el robot debe ejecutar (GRI), estado de las tareas básicas (MRI) y la respuesta del SynchronizeTask, por medio del mensaje RESPONSE (para saber si puede ejecutar una tarea o no), que sería una comunicación interna.
- *Salidas*: estado de las tareas del robot (GRI), mensajes tipo SET_DEPENDENCIES y QUERY al módulo SynchronizeTask (comunicación interna) y por último las tareas básicas que le son enviadas al MML (MRI).

- **SynchronizeTasks Module**

  Como se ha dicho anteriormente este módulo se encarga de llevar un registro de las tareas que se van a ejecutar en el robot y sus respectivas precondiciones. El módulo va comprobando constantemente qué precondiciones se han cumplido para responder si una tarea se

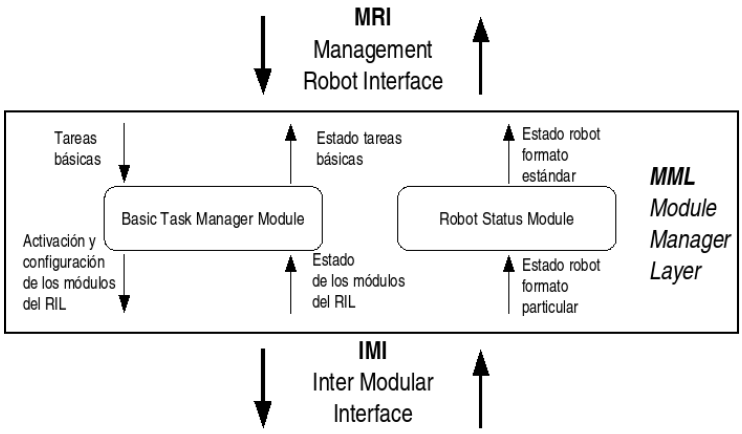

Figura 2.6: Módulos que forman el Module Manager Layer (MML)

puede ejecutar o no cuando alguien lo requiera. Debido a que normalmente los módulos SynchronizeTasks y TaskManager van a estar en el mismo proceso, la comunicación va directamente por medio de la sección crítica compartida por ambos hilos.

Teniendo en cuenta que este módulo sólo se comunica con el TaskManager, las entradas y salidas del módulo son:

- *Entradas*: estado de las tareas de los distintos robots, mensajes del tipos SET_DEPENDENCIES y QUERY (comunicación interna).
- *Salidas*: mensaje del tipo RESPONSE (comunicación interna).

2.4.2. MML (Module Manager Layer)

Este módulo es el encargado de configurar y activar los distintos módulos del RIL (Robot Implementation Layer) para que se ejecute una determinada tarea básica. Es importante resaltar que a diferencia del RAL esta capa ya es dependiente del robot en el que se encuentre implementada.

Esta capa está formada por dos módulos independientes entre sí (ver figura 2.6):

- **BasicTask Manager** Este módulo se encarga de gestionar las tareas básicas que le son enviadas desde el módulo TaskManager del RAL. Como se dijo anteriormente el RAL manda las tareas básicas de tal forma que este módulo sólo gestiona un tipo de tarea a la vez, lo cual simplifica su funcionamiento. Cuando se habla de gestionar una tarea básica se quiere decir que el módulo activa los módulos del RIL necesarios para la ejecución de la tarea, configura las conexiones entre ellos convenientemente y les transmite los datos que sean necesarios para cada tipo de tarea. Por otra parte, este módulo es el encargado de calcular el estado de la tarea básica a partir de los estados de los distintos módulos del RIL que intervienen en la ejecución de la tarea. Este módulo además se encarga de comprobar si la tarea básica

es coherente con el tipo de robot y el estado de este, en caso de que no sea aborta la tarea y transmite un error al nivel superior.

Las entradas y salidas del módulo son:

- *Entradas*: tareas básicas transmitidas por el RAL, estado de los distintos módulos del RIL.
- *Salidas*: estado de las tareas básicas y datos necesarios por parte de los módulos del RIL para la ejecución de las tareas básicas.

- **RobotStatus Module** Este módulo tiene como función traducir el estado del robot en un estado estándar que debe ser común para todos los robots. Este estado es transmitido por el sistema de comunicaciones al resto de las entidades del sistema.

  Las entradas y salidas del módulo son:

  - *Entradas*: estado en el formato particular del robot.
  - *Salidas*: estado en el formato estándar, común para todos los robots.

2.4.3. RIL (Robot Implementation Layer)

En esta capa se encuentran los módulos que de forma conjunta ejecutarán las distintas tareas que se le transmitan al robot. Cada uno de estos módulos implementará una o varias funcionalidades que nos permitirán realizar un número de tareas. Estos módulos se pueden dividir en las siguientes categorías:

- Módulos de planificación: se encargan de planificar los movimientos en función de las tareas y de la información proveniente de los módulos de percepción.
- Módulos de movimiento: controlan al robot para que siga un determinado camino.
- Módulos de percepción: obtienen y procesan la información de los distintos sensores. Esta información puede ser utilizada para calcular el estado del robot, implementar comportamientos reactivos, detectar objetos, etc.
- Otros módulos: como por ejemplo un módulo que se encargue de controlar el pan&tilt de una cámara.

Como se puede observar en la figura 2.7 la comunicación entre los módulos y entre éstos con el MML se realiza por medio del Inter Modular Interface (IMI). Además uno o varios de estos módulos serán los encargados de comunicarse con el hardware del robot.

Las entradas y salidas de cada uno de estos módulos son (ver figura 2.8):

- *Entradas*: activación y configuración del módulo y los datos necesarios para poder ejecutar la funcionalidad implementada en el módulo. Los datos de entrada pueden provenir del Module Manager Layer o de otro módulo en el caso que sean necesarios más de un módulo para la ejecución de una tarea.

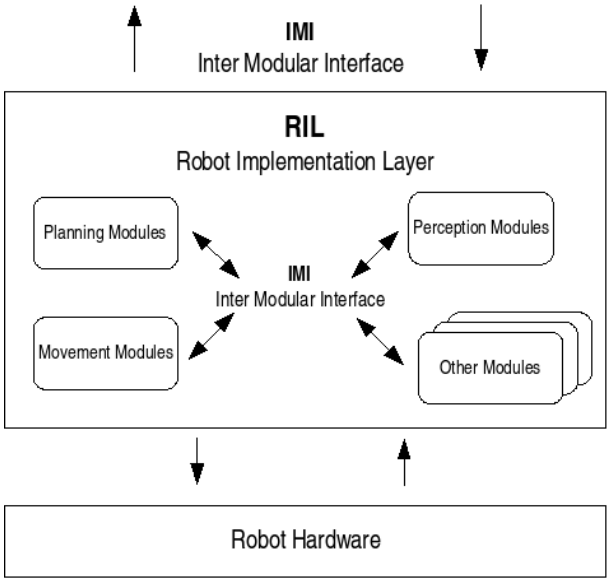

Figura 2.7: Módulos que forman el Robot Implementation Layer (RIL)

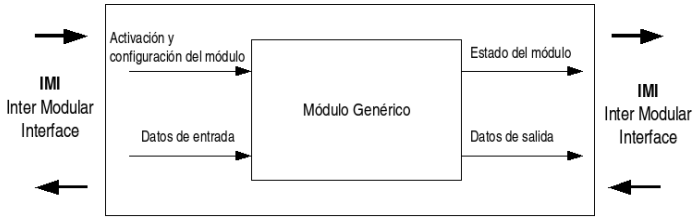

Figura 2.8: Entradas y salidas de un módulo genérico del RIL

Figura 2.9: Vista frontal y lateral del vehículo autónomo Romeo-4R

- *Salidas*: estado del módulo y datos de salida resultado de la ejecución de una funcionalidad. Normalmente los datos de salida se transmitirán hacia otro módulo como datos de entrada.

Con el objetivo de clarificar el flujo de datos entre los módulos, se va a utilizar un ejemplo ilustrativo. Imaginemos que un robot recibe la tarea de visitar una serie de puntos, estos puntos son transmitidos al Module Manager como datos adjuntos a la tarea. A continuación, el Module Manager activa los módulos de planificación y seguimiento de camino y configura sus conexiones para que la salida de datos del planificador sea la entrada de datos del seguidor de camino. De esta forma cuando el planificador le llegan los puntos a visitar y calcula el camino que debe seguir el robot, estos son automáticamente transmitidos al seguidor de caminos que controlará al robot para que realice la tarea.

Como es lógico, no todos los robots tendrán los mismo módulos ni estos tendrán implementadas las mismas funcionalidades. Por ello, a continuación, se va a explicar la estructura de módulos del RIL para cada uno de los robots que siguen esta arquitectura.

Romeo-4R: Vehículo terrestre

ROMEO-4R es un vehículo autónomo móvil desarrollado por el Grupo de Robótica Visión y Control del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla, siendo inicialmente un coche concebido para ser utilizado en la EXPO'92 para transportar a personas y que tiene una estructura muy parecida a los coches de los campos de golf y de fútbol (ver figura 2.9). Sin embargo tras ser adquirido por el departamento se le ha añadido la mecánica y electrónica necesaria para que pueda ser utilizado como plataforma de investigación en navegación autónoma para exteriores [1] y [2].

Los módulos que se han desarrollado para este robot son (ver figura 2.10 donde se han omitido los flujos de datos correspondientes al control de los módulos y al estado de éstos):

- **Trayectory Generator**: a partir de uno o varios puntos planifica un camino que debe seguir el robot para visitar dichos puntos. Siguiendo la estructura genérica de un módulo (ver apartado 2.4.3), los datos de entrada para este módulo serían los puntos a seguir y los de salida el camino ya calculado.

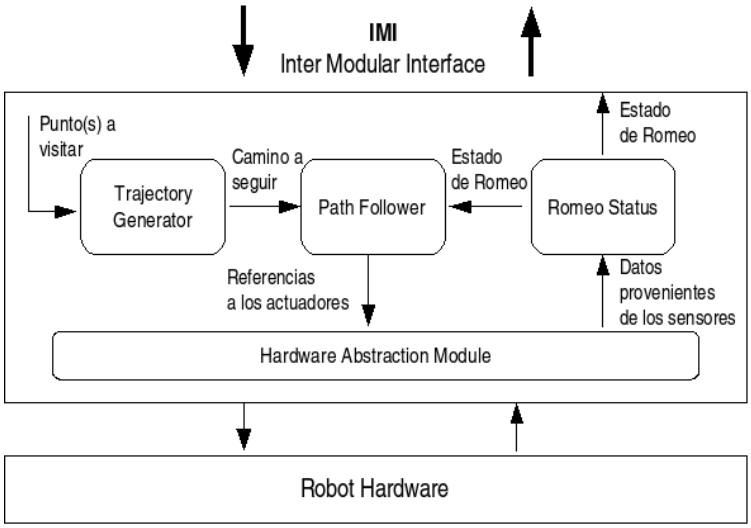

Figura 2.10: Estructura de los módulos del RIL en el robot Romeo

- **Path Follower**: este módulo se encarga de controlar al robot para que siga un camino determinado. En este módulo los datos de entrada es el camino calculado por el módulo generador de trayectorias y los de salida las referencias a los controladores de bajo nivel que controlan los actuadores del robot.
- **Romeo Status**: se encarga de realizar un log de los datos recibidos y de calcular el estado del robot, normalmente su posición y orientación en un determinado sistema de coordenadas. Sus datos de entrada son la información proveniente de los distintos sensores con los que cuente el robot y los de salida el estado del robot en el formato específico de este.
- **Hardware Abstraction Module**: es el módulo que se encarga de interactuar con el hardware instalado en el robot. Los datos de entrada son las referencias de los actuadores y los de salida los datos provenientes de los sensores.

HERO: Vehículo aéreo

HERO es un vehículo aéreo no tripulado o UAV (*unmanned aerial vehicle*) desarrollado por el Grupo de Robótica Visión y Control del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla. Este vehículo está basado en un helicóptero de aeromodelismo el cual ha sido modificado para que pueda ser controlador autónomamente (ver figura 2.11).

Los módulos que se han desarrollado para este robot son (ver figura 2.12 donde se han omitido los flujos de datos correspondientes al control de los módulos y al estado de éstos):

- **Hero Status**: al igual que en el Romeo, este módulo se encarga de calcular el estado del robot, normalmente su posición y orientación en un determinado sistema de coordenadas.

Figura 2.11: Vista lateral del vehículo autónomo Hero

Siguiendo la estructura genérica de un módulo (ver apartado 2.4.3), los datos de entrada son la información proveniente de los distintos sensores con los que cuente el robot y los de salida el estado del robot en el formato específico de este.

- **Trayectory Generator**: a partir de uno o varios puntos planifica un camino que debe seguir el robot para visitar dichos puntos. Para este módulo los datos de entrada serían los puntos a seguir y los de salida el camino ya calculado.
- **Bridge Module**: debido a las fuertes restricciones en tiempo real necesarias para controlar un helicóptero autónomo, es necesario que el control del robot se lleve a cabo por un microcontrolador de altas prestaciones, como por ejemplo un DSP (*Digital Signal Processor*). Por lo tanto este módulo sirve de interfaz entre los módulos de la arquitectura y el DSP. Los datos de entrada son el camino a seguir y las órdenes de aterrizaje y despegue, mientras que los de salida son los datos provenientes de los sensores.

Sin embargo, actualmente no se ha terminado de desarrollar todo el software y la estructura de módulos al día de hoy queda reflejada en el figura 2.13, como se puede observar fundamentalmente queda por desarrollar el módulo Trajectory Generation.

Robots Simulados: Omnidireccional terrestre y aéreo

A diferencia de los dos robots comentados anteriormente donde los módulos se han implementados en robots reales, éstos sólo existen en simulación (ver sección 2.5) y tienen una estructura de módulos dentro del RIL mucho más sencilla. Por otra parte, la única diferencia entre el robot omnidireccional terrestre y el aéreo es el modelo de robot utilizado y por lo tanto también el controlador implementado. El robot omnidireccional terrestre sólo permite movimientos en 2 dimensiones, mientras que el aéreo permite moverse en los 3 ejes del espacio.

En estos dos robots, el RIL está formado únicamente por un módulo que se encarga del movimiento del robot. Este módulo llamado Movement Module (ver figura 2.14 donde se han omitido los flujos de datos correspondientes al control de los módulos y al estado de éstos) es el encargado de generar el camino y controlar al robot para que lo siga. Los datos de entrada de este módulo son los puntos a visitar (en el caso del robot aéreo además los comandos de aterrizar y despegar) y los datos de salida el estado del robot.

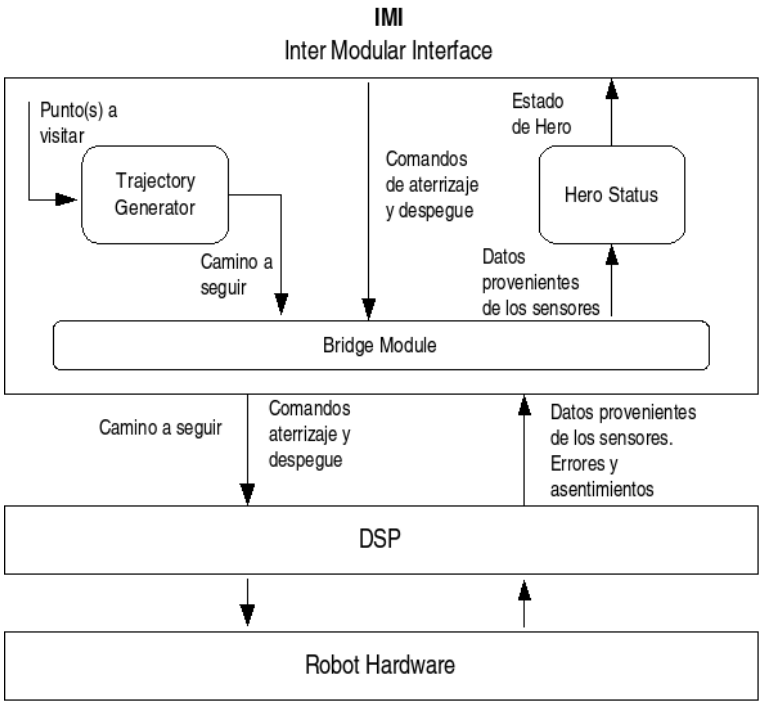

Figura 2.12: Estructura de los módulos del RIL en el robot Hero

2.5. Simulación de la arquitectura

Como se comentó en la sección introductoria, uno de los objetivos de esta arquitectura era poder simular múltiples robots heterogéneos en condiciones lo más parecidas a la realidad. En este apartado se va a comentar como se ha conseguido este objetivo.

En primer lugar destacar que las capas superiores (RAL y MML) son completamente software por lo que se pueden utilizar tanto en la realidad como en simulación. La idea es entonces poder utilizar los módulos que se implementen en el RIL en ambos casos sin efectuar ningún cambio. Para ello se han desarrollado una serie de módulos que sustituyen a los módulos que interactúan con el hardware en cada robot y que tienen exactamente la misma interfaz. Además estos módulos implementan un modelo del robot para que los algoritmos de control funcionen correctamente.

Por otra parte se ha desarrollado un simulador del entorno que permite implementar la simulación de los distintos sensores que tengan los robots. Esto permite que no sólo se puedan simular estrategias planificadas o deliberativas sino que se puedan simular comportamientos reactivos, como la evitación de obstáculos, antes de probarlos en la realidad. En la figura 2.15 se muestra para un robot genérico, la diferencias en la estructura del RIL entre simulación y lo que realmente está implementado en el robot real.

A continuación se va a comentar como se ha implementado los módulos que permiten la simulación de los dos robots en los que se ha implementado esta arquitectura.

- **Simulación de Romeo-4R**: para la simulación del robot terrestre Romeo-4R lo único

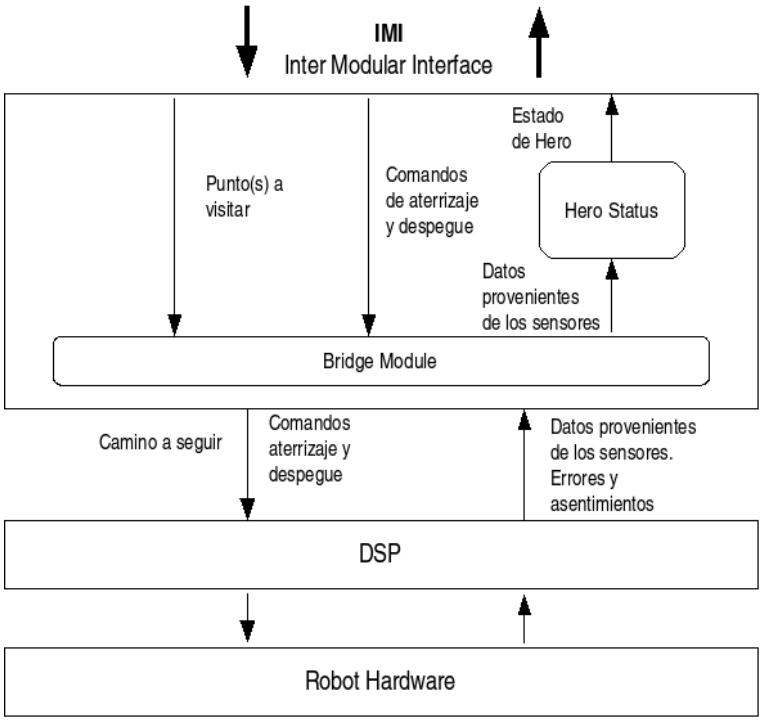

Figura 2.13: Estructura al día de hoy de los módulos del RIL en el robot Hero

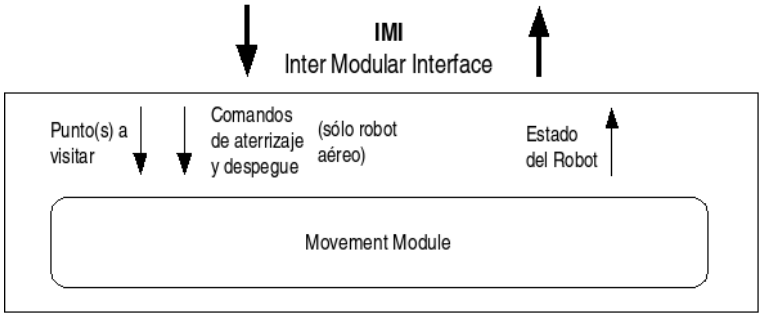

Figura 2.14: Estructura de los módulos del RIL en los robots simulados

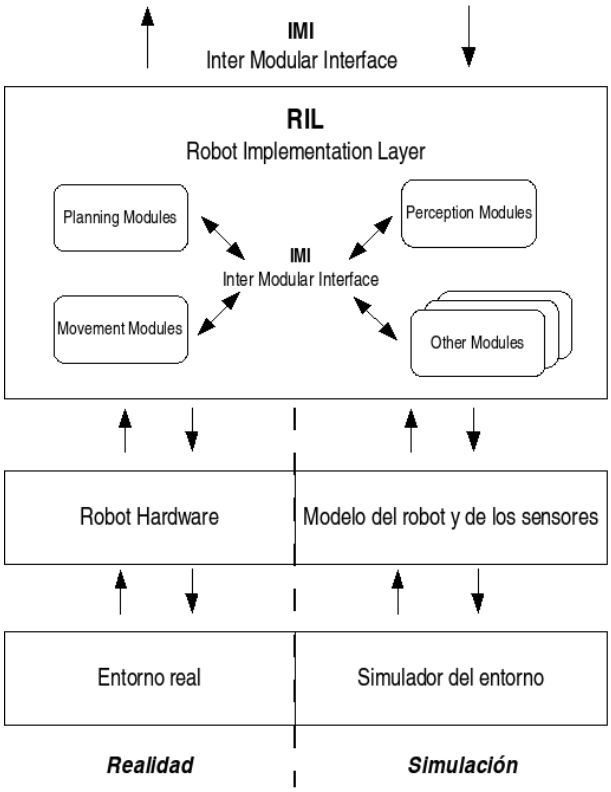

Figura 2.15: Diferencias en el RIL entre simulación y realidad

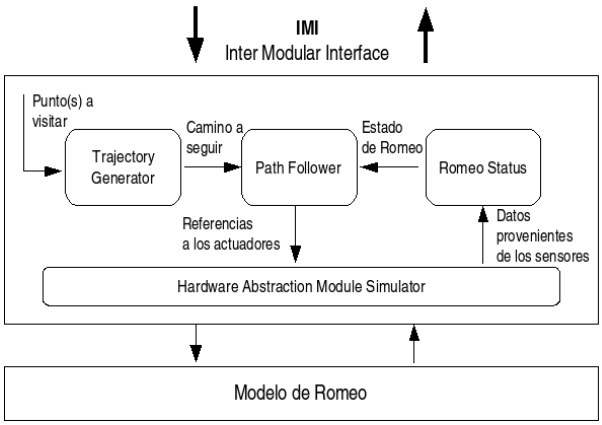

Figura 2.16: Módulos utilizados en simulación en el RIL de Romeo-4R

necesario ha sido desarrollar un módulo que sustituya al Hardware Abstraction Module (ver figura 2.16) y que implemente un modelo del Romeo-4R. Este módulo, llamado Hardware Abstraction Module Simulator, tiene exactamente la misma interfaz que el módulo al que sustituye por lo que los demás módulos no son conscientes del cambio.

- **Simulación de Hero**: en este caso se ha decidido simular al DSP completo. La idea es que tenga la misma interfaz y el módulo Bridge no note la diferencia entre el DSP real y el simulado. Dentro del simulador del DSP se ha implementado un modelo del helicóptero y un algoritmo para su control.

Por último comentar que la simulación de los sensores no aparece en las figuras anteriores porque todavía no ha sido implementada. Esto junto con el desarrollo de comportamientos reactivos en simulación y probados en la realidad son una de las líneas a seguir en los desarrollos futuros (ver sección 3.2).

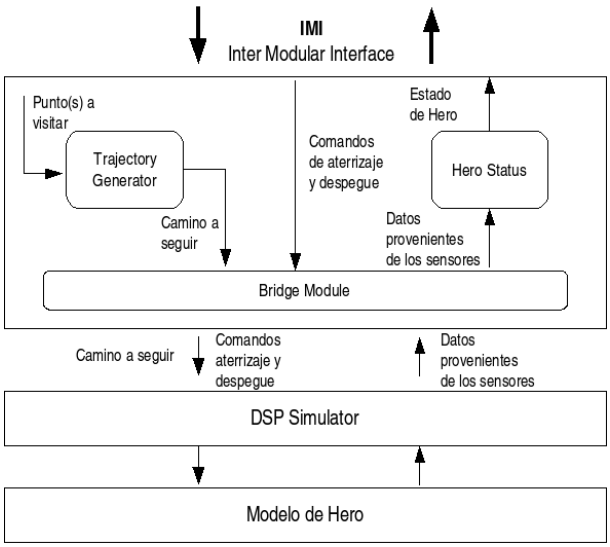

Figura 2.17: Módulos utilizados en simulación en el RIL de HERO

Capítulo 3

Conclusiones y desarrollos futuros

3.1. Conclusiones

En primer lugar comentar que en este documento se detalla una arquitectura multirobot. Estos robots pueden ser heterogéneos y tan diferentes como un robot móvil terrestre con dirección Ackerman y un UAV basado en un helicóptero de aeromodelismo. Además se ha desarrollado la arquitectura pensando que se pueda utilizar tanto en la realidad como en simulación y tanto con un único robot para probar estrategias de control, comportamientos reactivos, etc, como con múltiples robots para testear algoritmos de cooperación y coordinación.

Por otra parte entre las múltiples ventajas de una arquitectura de este tipo, resaltan las siguientes:

- Disminución del tiempo de desarrollo y puesta en marcha de cualquier nuevo robot que se quiera construir en el grupo. Básicamente habría que desarrollar el Hardware Abstraction Module, modificar levemente algunos de los módulos del RIL y adaptar el MML a las particularidades del robot. A partir de ese momento ya no sólo se tendría un robot completamente funcional sino que también podría cooperar con el resto de robots que tengamos y sigan esta arquitectura.

- Reutilización de algoritmos. Una vez que tengamos implementados una serie de algoritmos estándar para una serie de problemas, estos se pueden reutilizar en todos los demás robots.

- Forma de trabajo centrada en el algoritmo y no en el robot. La persona que vaya a desarrollar un algoritmo para un robot sólo tiene que aprender como funciona el módulo con el que va a trabajar y no necesita aprender como funciona todo el resto del robot como sucedía hasta ahora.

- Desarrollo en simulación, prueba en la realidad. Esta arquitectura nos permitirá contar con un simulador multirobot donde la única diferencia con la realidad será que el Hardware Abstraction Module estará simulado al igual que la dinámica del robot. Esto facilitará y acelerará el desarrollo del algoritmo y que a la hora de probarlos en la realidad den mucho menos problemas.

- Desarrollo de algoritmos multirobot. Esta arquitectura está permitiendo y permitirá investigar en el campo de múltiples robots.

3.2. Desarrollos futuros

Las posibles líneas a seguir en el futuro, podrían ser:

- Implementación de la arquitectura en los demás robots del grupo de investigación. Estos robots serían: coche de radio control, carretilla y avión. Con la incorporación de estos robots a la arquitectura se ampliarían considerablemente las posibilidades y se tendría un verdadero equipo de robots heterogéneos.

- Simulación del entorno: actualmente está desarrollado una primera versión de la simulador del entorno, sin embargo falta utilizar esta información para darle valores coherentes a los diferentes sensores de los robots. Esto supondría un cambio importante en la arquitectura ya que se podrían empezar a simular no sólo algoritmos deliberativos sino también comportamientos reactivos o incluso algoritmos deliberativos–reactivos.

- Algoritmos deliberativos–reactivos. Sería interesante investigar la combinación de estos dos tipos de algoritmos, como por ejemplo que se creen tareas nuevas cuando los sistemas de percepción detecten algo interesante y que esta tarea se reparta entre los distintos robots utilizando técnicas de asignación distribuida de tareas.

- Revisión de las estructuras de datos. A partir de la generación de este documento, es evidente la necesidad de revisar las estructuras de datos de las interfaces y conservar sólo aquellos datos que realmente son necesarios e incluir aquellos que falten. Finalmente habría que ponerse de acuerdo al respecto de ciertos temas cómo el sistema de coordenadas global y la definición de los ángulos del vehículo. También habría que definir exactamente que significan los estados de finalización en las tareas de percepción.

- Aumentar las relaciones entre las tareas, actualmente la única relación existente son las precondiciones. Para ciertas tareas de percepción podría ser interesante otro tipo de relaciones, como por ejemplo que cuando acabe una tarea acabe la otra, que se podría aplicar a una misión de vigilancia donde hay un tarea que se ejecuta a la vez que la tarea de detectar un determinado objeto, ambas tareas podrían terminar cuando se llegue al final del camino o cuando se detectara algo.

Aunque no sea exactamente un futuro desarrollo, es importante mantener actualizada la documentación de los módulos del RIL para cada uno de los robots ya que en este documento sólo se da una visión general y no se entra en excesivo detalle. Sin embargo con el fin de facilitar la continuidad del trabajo es completamente indispensable una buena documentación totalmente actualizada.

Finalmente, se va a explicar brevemente la visión personal que se tiene del futuro de la arquitectura. En ella en primer lugar no sólo habría una cooperación deliberativa de las capas altas del robot (ver figura 3.1), sino que además habría una cooperación entre los módulos de

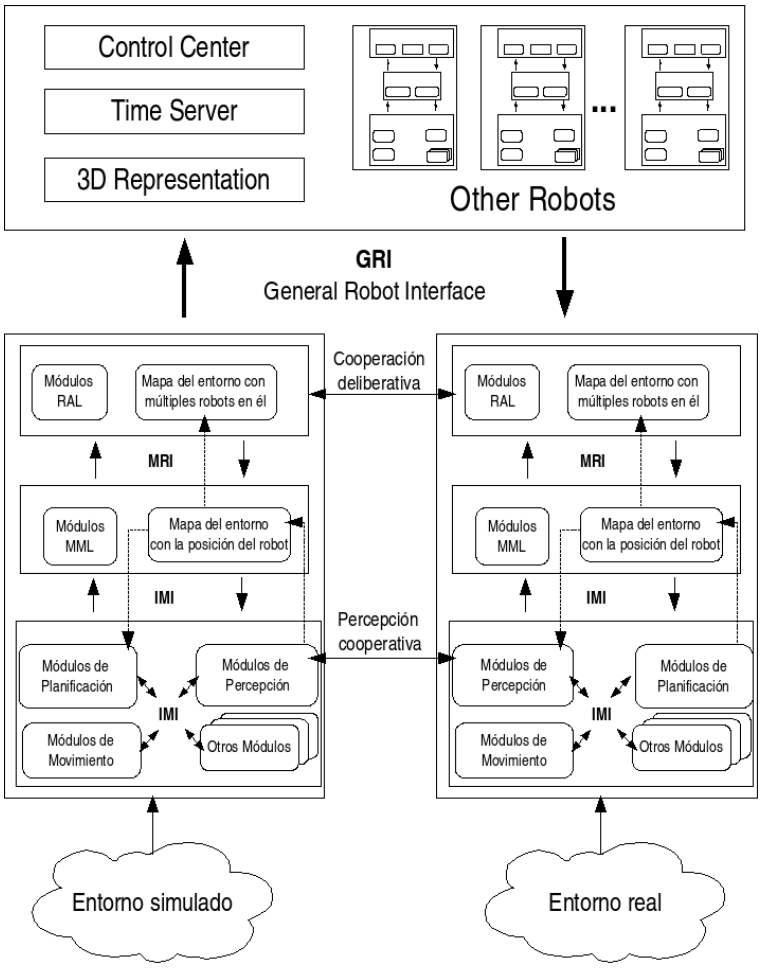

Figura 3.1: Visión personal del futuro de la arquitectura

percepción de los distintos robots. Además creo que sería muy interesante que la información de percepción se fusionara entre los robots reales y los simulados de tal forma que por ejemplo se pudiera probar la evitación de obstáculos con obstáculos ficticios o la evitación de colisiones entre dos robots aéreos con uno siendo real y otro virtual, de tal forma que no se ponga nunca en peligro el real. Finalmente la información ya procesada y de alto nivel de percepción sería utilizada para realimentar los algoritmos deliberativos de alto nivel y permita, a partir de la información del sistema de percepción, que los robots replanifiquen o mejoran la estimación de los costes que serán utilizados en los algoritmos de asignación distribuida de tareas.

Bibliografía

[1] A. Ollero. *Robótica, Manipuladores y Robots Móviles*. Marcombo, Barcelona, España, 2001.

[2] A. Ollero, B.C. Arrue, J. Ferruz, G. Heredia, F. Cuesta, F. Lopez-Pichaco, and C.Ñogales. Control and perception components for autonomous vehicle guidance. application to the romeo vehicles. *Control Engineering Practice*, (10):1291–1299, 1999.