Desarrollo de mecanismos de introspección artificial

Adolfo Hernando Marcos

15 de julio de 2008

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Propósito

El propósito de este trabajo es establecer un marco teórico mínimo sobre la introspección artificial y realizar un ejercicio práctico utilizando mecanismos de introspección artificial para examinar los problemas que pueden aparecer al utilizar estos mecanismos y extraer unas conclusiones.

1.2. Sistemas de alto grado de autonomía

Con el fin de establecer un contexto para este trabajo, debemos distinguir diversos elementos a la luz de la estrategia de análisis estándar de la Teoría General de Sistemas:

Entorno: El medio en el cual existe el sistema artificial.

Sistema: El sistema artificial en el que se plantea la introspección. En este trabajo se incluye, como se explicará más adelante, un estudio de aplicación a un sistema robot. Llamaremos *plataforma* al conjunto de recursos que lo componen.

Universo: Unión de sistema y entorno.

Los sistemas artificiales tradicionales están diseñados para operar en entornos estructurados dentro de límites más o menos amplios. Es decir, entornos que tienen unas características definidas y acotadas dentro de determinados rangos admisibles, conocidos en la etapa de diseño. Los diferentes modos de operación y procesos ejecutables por el sistema se diseñan y optimizan para este rango de condiciones.

Los sistemas de alto grado de autonomía pretenden poder adaptarse a operar eficientemente en entornos muy diferentes, adaptando su manera de funcionar o incluso parámetros físicos. Esto implica detectar las características que definen el entorno, y diseñar en tiempo real algoritmos de reconfiguración adecuados. Estrategias de control ya clásicas, como el *Control Adaptativo con modelo de referencia*, MRAC (ver figura 2.2), son un ejemplo de estadios iniciales de autoadaptatividad. En ellos se puede observar el uso de dos tipos de conocimiento de naturaleza distinta. Veamos el caso MRAC ya mencionado:

Conocimiento sobre el entorno: El lazo de control básico del MRAC es un lazo cerrado con un controlador PID. El ajuste de este controlador se hace en función de la propia plataforma del sistema y del entorno en que éste va a operar, que impone una serie de condiciones de contorno al problema.

Conocimiento sobre el propio sistema: En un controlador MRAC se superpone un segundo lazo de control sobre el PID. La misión de este lazo es ajustar dinámicamente los parámetros proporcional, derivativo e integral del controlador a las condiciones de operación del sistema (determinadas en general por el entorno). Para poder realizar esta tarea eficientemente, es necesario que este lazo de control haya sido diseñado de forma que garantice algunos parámetros de prestaciones, y especialmente la estabilidad del sistema en todo momento.

Podemos observar que el segundo lazo de este ejemplo es la solución al problema de *¿cómo debe cambiar el sistema si cambian sus condiciones de operación?* La resolución de este problema requiere conocimiento sobre el propio sistema.

En un sistema como el del ejemplo el conocimiento parte siempre del diseñador. En los sistemas de alto grado de autonomía, en cambio, se persigue que el propio sistema cuente con conocimiento sobre sí mismo y mecanismos para explotarlo adaptándose a su entorno de operación.

1.3. La introspección artificial

En este contexto, debemos entender que los mecanismos de *introspección artificial* forman parte de los mecanismos de explotación de conocimiento del sistema. Tienen por objeto *generar estimaciones del estado del sistema*, en el marco mostrado en la figura 1.1.

Entenderemos que un proceso de introspección parte de datos de entrada de dos naturalezas: o bien de lecturas de los sensores, o bien de una base de

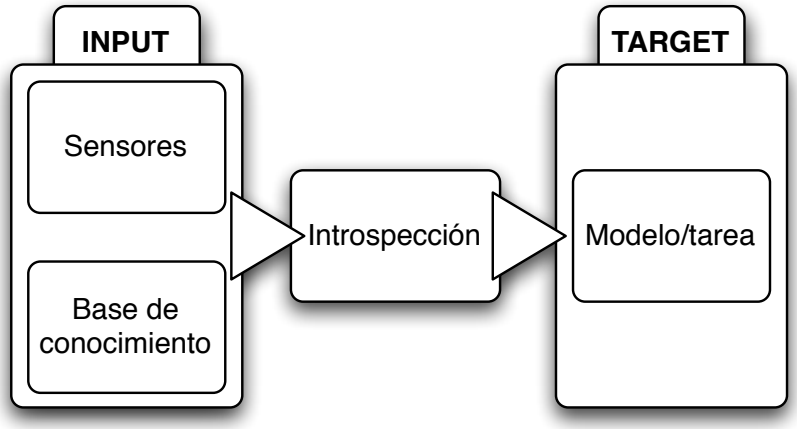

Figura 1.1: Proceso de introspección genérico

conocimiento, y produce una estimación del estado del sistema *adaptada a un modelo o a una tarea*. Es importante detenernos en este segundo punto: un proceso de introspección tiene por objeto describir el estado del sistema:

- Incluyendo el tipo de información que resulte de utilidad para la tarea o el modelo de destino (indicado como *TARGET* en la figura 1.1).
- Procesando esa información adecuadamente para que sea de utilidad en el TARGET. Esto puede implicar acondicionar los valores de las variables presentadas.

Podríamos decir que la introspección artificial es un tipo de percepción especializado en el propio sistema. El ejemplo de mecanismo de introspección más elemental es un sensor para medir una variable de estado del sistema. Ejemplos más evolucionados podrían ser los *observadores* construidos para implementar control en el espacio de estados.

1.4. El alcance de este trabajo

En este trabajo se ha realizado dos tareas. En primer lugar, se han establecido y constituido diversos conceptos de introspección en un marco teórico mínimo. Estos conceptos tienen sus orígenes en la literatura sobre sistemas naturales (bibliografía de sistemas cognitivos) y en la literatura sobre sistemas artificiales. (bibliografía sobre sistemas tolerantes a fallos, ...)

En segundo lugar, se ha realizado un ejercicio práctico basado en una plataforma robótica móvil Pioneer 2AT-8. En términos de la figura 1.1:

INPUT: Dos medidas reales dadas por las librerías ARIA de la plataforma:

- Velocidad angular de giro de las ruedas
- Tensión de las baterías

TARGET: Un proceso básico de toma de decisión: recorrer/no recorrer una trayectoria dada a través de un determinado entorno, en función de la energía disponible en las baterías. Este proceso de toma de decisión estará basado en una estimación del siguiente estado resumido del sistema:

- Energía consumida en caso de recorrer la trayectoria
- Energía disponible en las baterías

Introspección: Se ha implementado dos modelos matemáticos:

- La dinámica del robot
- La capacidad energética de las baterías

1.5. La estructura de este informe

El capítulo segundo se dedica al estado del arte de las técnicas relacionadas con la introspección artificial. Se analiza en una sección los sistemas artificiales con automodelos empleados en robótica. En dos secciones posteriores, se analizan los mecanismos que se pueden utilizar para realizar introspección, clasificados en dos categorías: *hardware* y *software*.

El capítulo tercero contiene el marco teórico mínimo. En la primera sección se describen los sistemas paralelos distribuidos. Después se habla de los conceptos generales sobre introspección y sus aplicaciones.

El capítulo cuarto contiene las aportaciones originales realizadas en esta arquitectura. En una primera sección se describe la arquitectura que se pretende desarrollar, así como su estado de construcción actual. Una sección final describe una realización práctica de una simulación del robot.

Cierra el texto un último capítulo donde se exponen unas conclusiones.

Capítulo 2

Estado del arte.

De las técnicas relacionadas con la introspección artificial.

El conocimiento introspectivo es considerado un componente esencial de la operación de un robot en el mundo del "sentido común", que define una habilidad para auto-observar, modelizar y alterar de una manera inteligente el estado interno [Morris, 2007].

Los mecanismos de introspección que incorporan los sistemas actuales —entendida en el sentido especificado en la sección 1— se derivan de los requisitos impuestos por los sistemas de control o de inteligencia artificial que incorporan. Por tanto, están diseñados para una aplicación concreta, y no como particularización de un caso general.

La introspección robótica es un campo relativamente nuevo e inexplorado de la investigación robótica, comparado con problemas robóticos clásicos como la localización, la planificación de trayectorias y la coordinación multi-robot. Esto es debido a que la metodología de introspección robótica se aproxima al desarrollo del robot como un observador en lugar de como un ingeniero.

La identificación de fallos y la recuperación permiten una operación del robot robusta y ha sido durante mucho tiempo un problema de investigación significativo acerca de sistemas robóticos. Aproximaciones introspectivas, como la de Bongard, Zykov y Lipson, internalizan y construyen modelos dinámicamente de la operación de robots para identificar y recuperar de fallos 2.1.2.

Fuera del campo de la robótica, el trabajo en fiabilidad computacional también contempla la diagnosis y recuperación de fallos. La fiabilidad computacional se identifica con la fidelidad de un sistema computacional para completar una tarea asignada a través de la relación de tres conceptos: fallo, error y avería. Un fallo del sistema se produce cuando el comportamiento del sistema se desvía del requerido por sus especificaciones. Un error es la parte

del estado del sistema que es propensa a conducir al fallo posterior. La causa de un error es una avería.

En esta sección queremos revisar el estado tecnológico actual desde dos puntos de vista que entendemos relacionados con la introspección artificial. Primero, desarrollos que explotan automodelos a nivel de sistema (mayor nivel de abstracción que los sistemas de control avanzado mencionados en las secciones 1 y 2.1.1). Segundo, técnicas básicas existentes para coordinación y comunicación de procesos, dos aspectos fundamentales en la introspección artificial, como se verá en la sección 3.

2.1. Sistemas Basados en Automodelos.

Los animales de orden superior usan una forma de "modelo interno" de sí mismos para planificar acciones complejas y predecir sus consecuencias, pero no es claro si y cómo estos modelos son adquiridos o qué forma tienen. Análogamente, la mayoría de sistemas robóticos prácticos usan internamente modelos matemáticos que son laboriosamente construidos por los ingenieros.

Veamos un automodelo genérico y mencionemos algunos sistemas robóticos que construyen sus propios modelos de sí mismos:

2.1.1. Automodelo genérico

Vamos a realizar el análisis de un automodelo genérico desde el punto de vista de la teoría de control moderna. Un modelo genérico contiene sensores, actuadores, un controlador y la propia planta. Cuando se usan automodelos, el controlador dispone de un modelo de la planta. Al aumentar la complejidad de la planta, también suele aumentar la complejidad del modelo de la planta, y con ella la complejidad del controlador que dispone de dicho modelo.

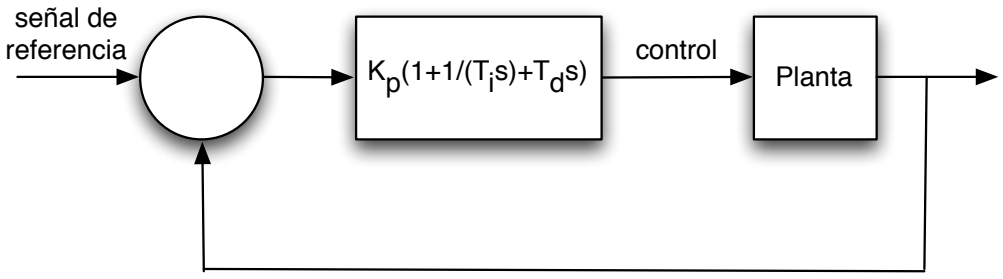

Figura 2.1: Control PID

Uno de los entornos de control más sencillos que podemos imaginar, figura 2.1, es una planta en la que hay un sensor, un actuador y un controlador,

siendo el controlador un PID. El controlador dispone de un modelo de la planta porque las constantes Ki, Kp y Kd están ajustadas para la planta en cuestión. Existen variantes como los controladores PI-D, I-PD y el control PID con dos grados de libertad. Podemos encontrar más información en [Åström and Hägglund, 2006].

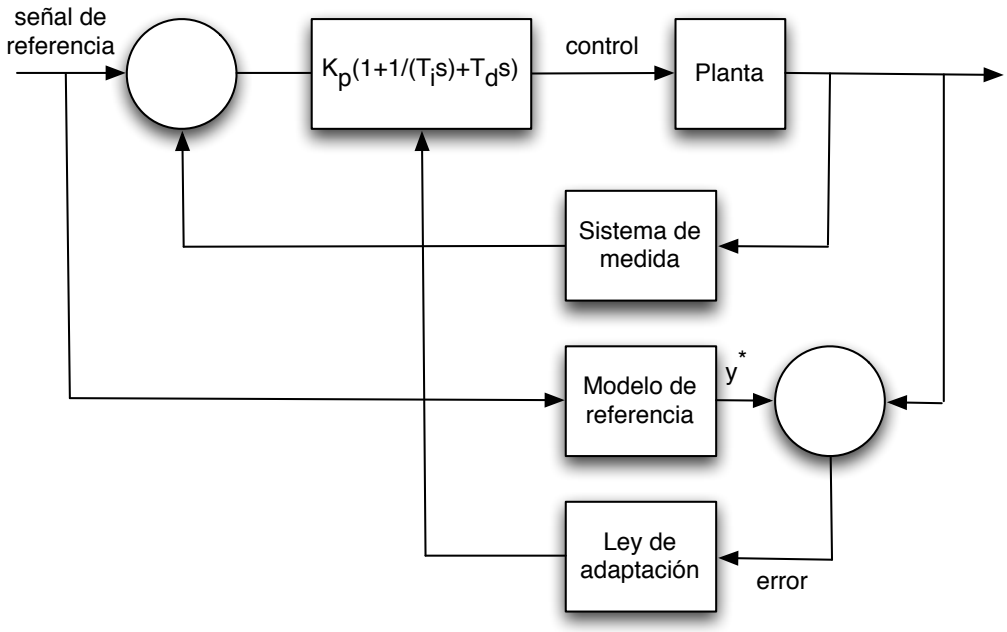

Figura 2.2: *Model Reference Adaptative Control* (MRAC)

Modelos más avanzados de controladores, como el MRAC de la figura 2.2 del que ya hemos hablado en la introducción, ajustan continuamente las constantes proporcional, derivativa e integral (Kp, Ki y Kd) con el fin de lograr que el rango de operación del controlador para dicha planta sea más amplio.

A continuación vamos a ver algunos modelos, que si bien no tienen un carácter matemático tan formal como el automodelo genérico inspirado en sistemas de control, representan un avance en el mundo de la robótica y nos permiten ver casos de uso y aplicaciones de estos automodelos.

2.1.2. Estrella de Mar

Los automodelos pueden carecer enteramente de consciencia y se pueden realizar también en sistemas artificiales. Josh Bongard, Victor Zykov y Hod Lipson (2006) han creado una estrella de mar artificial (figura 2.3) que desarrolla gradualmente un modelo interno explícito de sí misma. Su máquina de

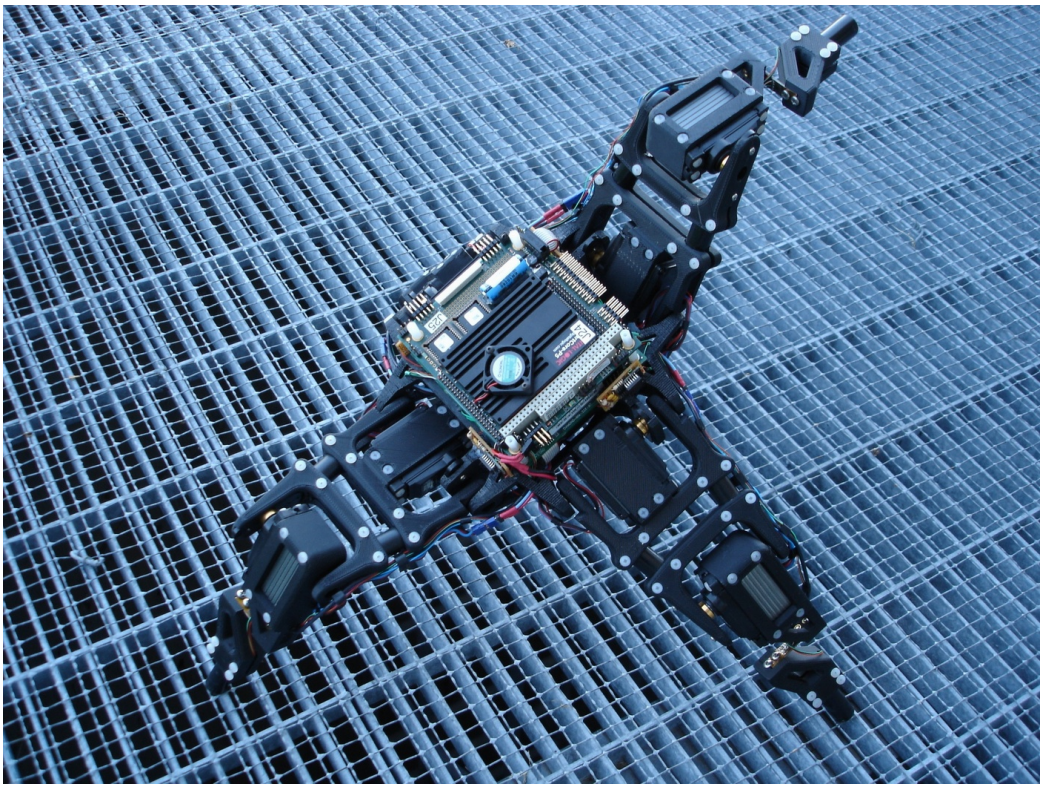

Figura 2.3: Estrella de mar artificial

cuatro patas usa relaciones entre sensaciones y actuaciones para inferir indirectamente su propia arquitectura y utiliza después este automodelo para generar un movimiento de avance hacia delante. Cuando se quita una parte de una pata, adapta su automodelo y genera maneras de andar alternativas - aprende a cojear.

En otras palabras: Es capaz de reestructurar la representación de su propio cuerpo tras la pérdida de un miembro. Puede aprender. Este concepto puede ayudar a desarrollar máquinas más robustas e iluminar la ontología del automodelado en animales, pero es también interesante desde un punto de vista teórico, puesto que demuestra por primera vez que un sistema físico tiene la habilidad, tal y como los autores la pusieron, para recuperar autonomamente su propia topología con poco conocimiento a priori, optimizando constantemente los parámetros de su propio automodelo.

2.2. Técnicas básicas para implementación de mecanismos de introspección

Como se mencionó al comienzo de este capítulo y se verá en detalle en la sección 3, en el caso general de un *sistema paralelo distribuido*, los mecanismos de introspección estarán implementados por una arquitectura de procesos, que estarán sujetos a ciertos requisitos de coordinación con otros procesos (procesos de INPUT, ver figura 1.1) y de intercambio de información. La implementación de estos aspectos, en el caso más general, está basada en algunas técnicas o algunos componentes básicos que se revisará a continuación.

2.2.1. Introspección *hardware*: Sensores

Un sensor es un tipo de transductor que usa un determinado tipo de energía, una clase de señal, y la convierte en una lectura con el propósito de transmitir información.

2.2.2. Mecanismos para introspección de *software*

La mayor flexibilidad a la hora de implementar mecanismos de introspección la dará el software tanto de aplicación como de la plataforma. Existen en la actualidad muchos mecanismos concretos para el acceso a informaciones introspectivas concretas, así como procedimientos generales —instrumentación software— que permiten la construcción de cualquier mecanismo concreto.

Los siguientes puntos reflejan técnicas *software* de utilidad en nuestra implementación de mecanismos para introspección artificial:

Checkpointing Su utilidad principal es resguardar el estado del sistema en programas que realizan cálculos largos para evitar tener que reiniciar el programa desde el principio en caso de error, incluso aunque este error sea debido a la programación del software, permitiendo reiniciar el proceso en el estado existente en el último resguardo de los datos. Se emplean mecanismos similares para migración de procesos de un sistema a otro. No es válido para todo el sistema.

Existen implementaciones de este mecanismo en:

- "Ego + Pego": http://www.laas.fr/~lcourtes/software/ego/
- "Libckpt": http://www.cs.utk.edu/~plank/plank/www/libckpt.html
- CORBA tiene el servicio *"CORBA Persistent State Service"* para persistencia de datos [OMG, 2002].

Programación concurrente *Native POSIX Thread Library* (NPTL). Es una librería basada en el estándar POSIX para manejar concurrencia por medio de hilos. También incluye *mutex*, condiciones, bloqueos de lectura/escritura, *spins* y barreras adecuados para su uso cuando se emplean hilos (*threads*).

Excepciones Las excepciones pueden ser producidas por realización de peticiones de servicio erróneas, (excepciones de la interfaz) o cuando son producidas por un mal comportamiento del componente (excepciones de fallos)

Excepciones CORBA Todas las excepciones CORBA derivan de la clase `CORBA::Exception`. Hay dos categorías: Las excepciones del sistema, que pueden ocurrir en cualquier sistema distribuido y derivan de la clase `CORBA::SystemException`, y las excepciones del usuario, específicas de una aplicación y un dominio determinado, que derivan de la clase `CORBA::UserException`.

Tiempo de cierre de bucles Medición del tiempo que se tarda en cerrar varios bucles del sistema. Esta comprobación puede ser útil para comprobar que el sistema está trabajando dentro de las condiciones de diseño del mismo.

Señales UNIX Mapeo de señales UNIX a excepciones de C++. Las señales UNIX son interrupciones de *software*. Pueden ser generadas por el terminal de control de la aplicación, por excepciones de *hardware*, pueden ser enviadas por otro proceso, por el comando `kill` o por condiciones *software* de las que el proceso deba ser notificado.

Señal 0 La comprobación de la existencia de ciertos procesos en el computador que controla el robot se puede realizar mediante el envío de la señal 0 a dichos procesos desde un proceso que los monitoriza en un sistema UNIX.

Integridad de bases de datos Verificación de la integridad de las bases de datos que se emplean para resguardar datos que sean requeridos por el sistema durante tiempos mayores de un cierto límite temporal.

Símbolos Las funciones y las variables locales y globales de un programa, cuando este se compila, se almacenan como símbolos. Para poder acceder a un símbolo que no es local de una manera trivial es necesario que este símbolo sea exportado. Si el símbolo no es exportado, también es posible acceder al símbolo, pero el procedimiento no es trivial, puesto

que es necesario resolver el símbolo empleando las librerías que usa el *linker* dinámico para hacer esta tarea con los símbolos exportados.

Tracing El trazado es una forma especializada de registro empleada para almacenar información sobre la ejecución de un programa. La especificación POSIX IEEE 1003.1q-2000 describe extensiones para soportar el trazado de programas de usuario mediante una API en lenguaje C. Estas facilidades permiten a un proceso seleccionar un conjunto de tipos de eventos para su traza, activar un flujo de traza de los eventos seleccionados según ocurren en tiempo de ejecución y obtener el registro de estos eventos trazados.

Capítulo 3

Marco teórico mínimo

En este capítulo vamos a desarrollar una serie de términos para formalizar los conceptos relativos a la introspección artificial. Para ello, vamos a asumir que se parte de un sistema ya operativo al que se pretende añadir prestaciones de alto nivel que requieren introspección.

Describiremos el sistema original como una arquitectura de procesos según el paradigma de *sistema paralelo distribuido* [Jalote, 1994] [Burns and Wellings, 2001] [Rumelhart and McClelland, 1987], a la que llamaremos *arquitectura del robot*. Consideraremos que los procesos de introspección que añadiremos constituirán una segunda arquitectura (*arquitectura introspectiva*). En base a esta conceptualización, exploraremos los aspectos fundamentales de la introspección artificial desde un punto de vista teórico.

3.1. Sistemas paralelos distribuidos

Consideremos la arquitectura del robot, representada en azul en la figura 3.1. En un instante de tiempo determinado existirá un cierto número de procesos en ejecución. Cada proceso puede verse como un nodo que se relaciona con los demás a través de intercambios de información. En la figura hemos representado los procesos como círculos azules $1 \ldots n$, y los intercambios de información entre procesos o *enlaces de comunicación* como líneas, numeradas $D, E, \ldots$.

En la práctica, estos enlaces representan: conexiones con sensores o periféricos, mecanismos de memoria compartida, comunicación inalámbrica, etc. Parte de la información que se intercambia por estos enlaces tiene una importancia *estructural* para el sistema: está referida a coordinación de procesos, sincronización, mecanismos de tolerancia a fallos, etc. Como es lógico, un sistema se diseña con capacidad de comunicación suficiente para todos sus

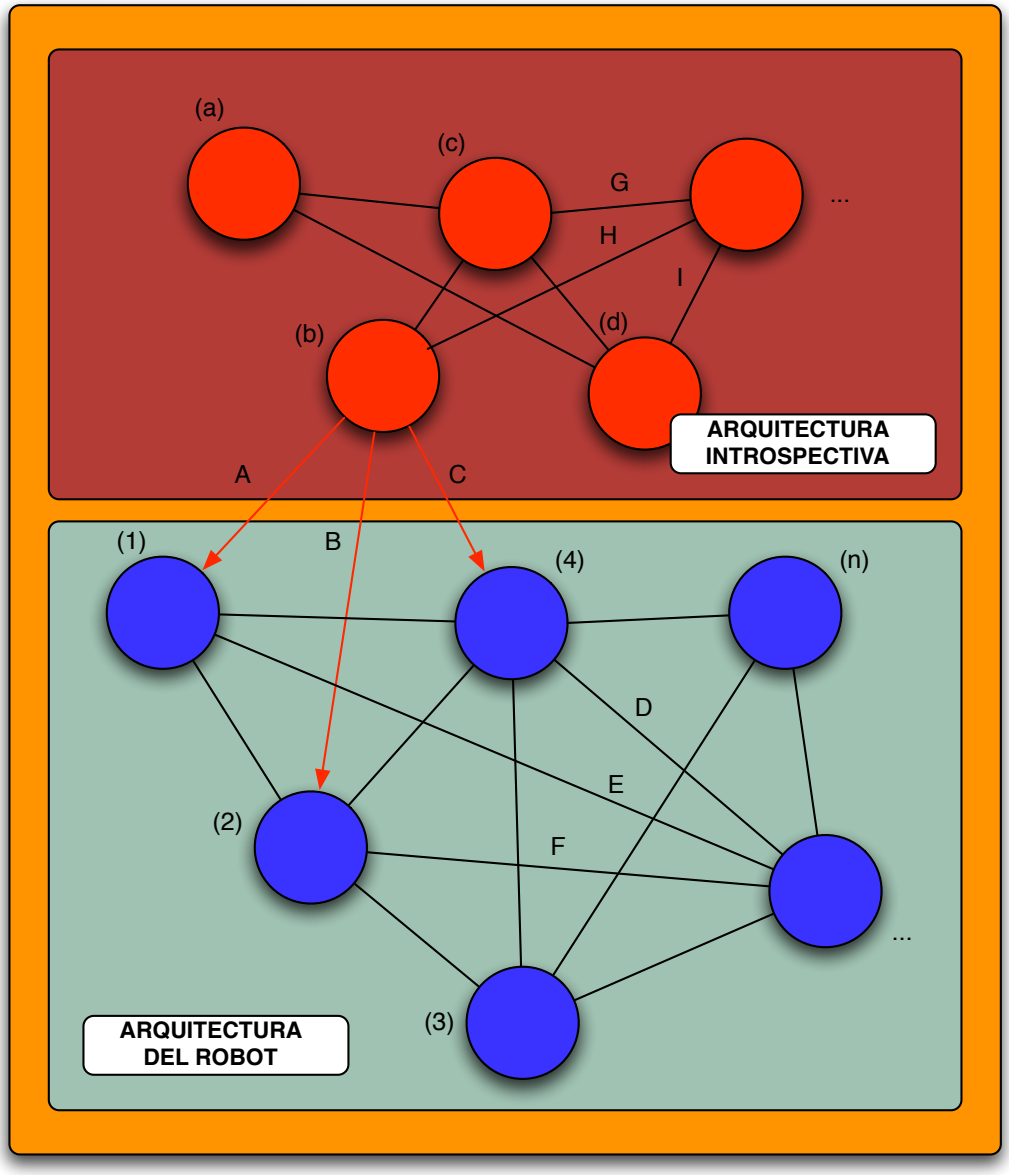

Figura 3.1: Modelo conceptual de un sistema artificial dotado de introspección

procesos. En el caso de que la arquitectura esté prevista para un número y naturaleza de procesos flexible, generalmente incorporará una infraestructura de *broker* (tipo CORBA, por ejemplo) que puede incrementar las necesidades de capacidad de comunicación del sistema, debido al peso de los *overheads*.

Por otro lado, consideremos el conjunto de procesos de introspección artificial como una arquitectura en sí misma (*arquitectura introspectiva*) añadida a la anterior. Hemos representado la arquitectura y sus procesos $(a, b, \ldots)$ en rojo en la figura 3.1. Estos procesos, al igual que los procesos azules, necesitan coordinarse entre sí y compartir información a través de enlaces que se representan en la figura con líneas numeradas $G, H, \ldots$

Como se mencionó en la sección 1, la introspección consiste en una estimación del estado del sistema. En otras palabras, los procesos de la arquitectura introspectiva extraerán información de los procesos de la arquitectura del robot y la procesarán de acuerdo al TARGET. Para ello, la arquitectura introspectiva tendrá que estar vinculada a la del robot a través de enlaces de información indicados $A, B \ldots$ en la figura 3.1.

Podemos observar que la arquitectura del robot no tiene por qué soportar estos enlaces, puesto que estamos asumiendo que era una arquitectura ya operativa (y en cuyo diseño no tuvo por qué preverse su escalabilidad). Esto hace que para un enlace concreto se puedan dar los siguientes casos:

1. **Incompatible:** La variable a medir por el proceso de introspección no puede obtenerse:
   a) La sobrecarga de ancho de banda que producirá imposibilitará la operación correcta de la arquitectura del robot.
   b) La variable no es accesible por software: No existe posibilidad de implementar ningún mecanismo para acceder a esa información (ver sec. 2.2.2) memoria compartida, etc.
   c) La variable no es accesible por hardware: No puede interceptarse la señal.
2. **No previsto:** La variable requerida por el proceso de introspección no existe en la arquitectura del robot.
3. **Compatible:** Existe algún mecanismo de software o hardware que permiten acceder a la variable de INPUT requerida.

El caso más favorable es el 3, en el que la superposición de la arquitectura de introspección sobre la original del robot sería directa. Los casos 1 y 2

pueden resolverse de varias formas. La más general es la construcción de un mecanismo de introspección que estime indirectamente la información de INPUT requerida a traves de otras medidas.

3.2. Introspección

Como se mencionó en el capítulo 1, los mecanismos de introspección artificial tienen como misión conocer el estado del robot. Debemos hacer ciertas consideraciones sobre el estado de un robot:

Cardinalidad: En general, para definir el estado de un robot es necesario un número de variables muy grande. Si bien el estado de las partes mecánicas del robot podrían caracterizarse con suficiente exactitud por un conjunto reducido de variables, las partes electrónicas, y en especial los microcontroladores y microordenadores, presentan un número de estados posible demasiado alto.

Abstracción: A bajo nivel, los estados posibles de un robot (y de los sistemas artificiales en general) se reducen al estado de sus partes mecánicas, de la electrónica y en concreto de sus memorias. Sin embargo, los procesos de un robot no solo operarán a nivel de *bit*, sino que lo harán a diferentes niveles de agregación, dependiendo del caso. Por ejemplo, a nivel de *byte*, *word*, o estructuras propias definidas por el diseñador. En otras palabras, existirán diferentes niveles de abstracción, desde el más bajo representados por un *bit* hasta el más alto, representado por estructuras de datos.

Para definir el estado de un robot de acuerdo a su operación y funcionalidad sería necesario hacerlo en función de todos los niveles de abstracción en que opere.

La complejidad que puede alcanzar definir propiamente el estado de un robot es, de hecho, una barrera. Generalmente se opta por considerar versiones reducidas del estado completo, a las que se llama indistintamente *estado*. En la ingeniería tradicional, se suele considerar el estado de las partes mecánicas. Sistemas tolerantes a fallos incorporan también caracterizaciones del estado de las partes lógicas. Existen dos formas de simplificar el estado de un sistema [Klir, 2001]:

Excluir variables: Se reduce a excluir variables de la caracterización del estado. Por ejemplo, considerando solamente las variables de estado de las partes mecánicas.

Particionar el estado: Consiste en agrupar conjuntos de variables de estado en categorías, asignando una variable de estado por categoría. Las categorías pueden representar partes del robot —o del sistema, en general— que se comportan como una entidad. El ejemplo más claro es el caso de varias piezas sólidas que están unidas rígidamente entre sí, y se consideran como una. De esta forma, se reduce la resolución con que se caracteriza el estado del robot.

Por otra parte, un proceso concreto en ejecución en general no necesitará conocer el estado completo del sistema, sino una reducción. Esta reducción se limitará a algunas variables del estado consideradas a un nivel de resolución determinado.

Volviendo a la consideración inicial, podemos entender que un proceso de introspección tiene como misión conocer el estado del robot *desde un punto de vista* determinado, dado por el modelo o proceso TARGET (ver figura 1.1). En otras palabras, el proceso TARGET requiere un conjunto específico de variables que representará un estado reducido del robot, que denotaremos por S^T. El proceso de introspección obtendrá este estado de un conjunto de variables INPUT, I^T, como se muestra en el caso (A) de la figura 3.2.

Llamaremos *sensorización* al proceso de obtención de lecturas de variables de la arquitectura del robot. En la práctica, este proceso puede equivaler a los siguientes casos:

- Lectura de un sensor instalado específicamente para el proceso de introspección.
- Lectura de una variable ya existente en la arquitectura del robot. Esto puede implicar mecanismos de memoria compartida, *threads* de comunicación específicos, etc. para los que la arquitectura puede no estar preparada, como se vio anteriormente.
- Lectura de una variable estimada de un automodelo. El automodelo puede formar parte de la arquitectura del robot o estar incluido en el proceso de introspección, según la implementación concreta (se ha indicado por AM en la figura 3.2).

Es interesante considerar el caso degenerado en que un proceso de introspección equivalga solamente a una o varias sensorizaciones. Es el caso (B) de la figura 3.2. En ese caso, diremos que se trata de un proceso de *introspección unitaria*.

El caso (C) de la figura 3.2 representa un proceso de introspección en que se sensoriza variables directamente de la arquitectura del robot, así como de

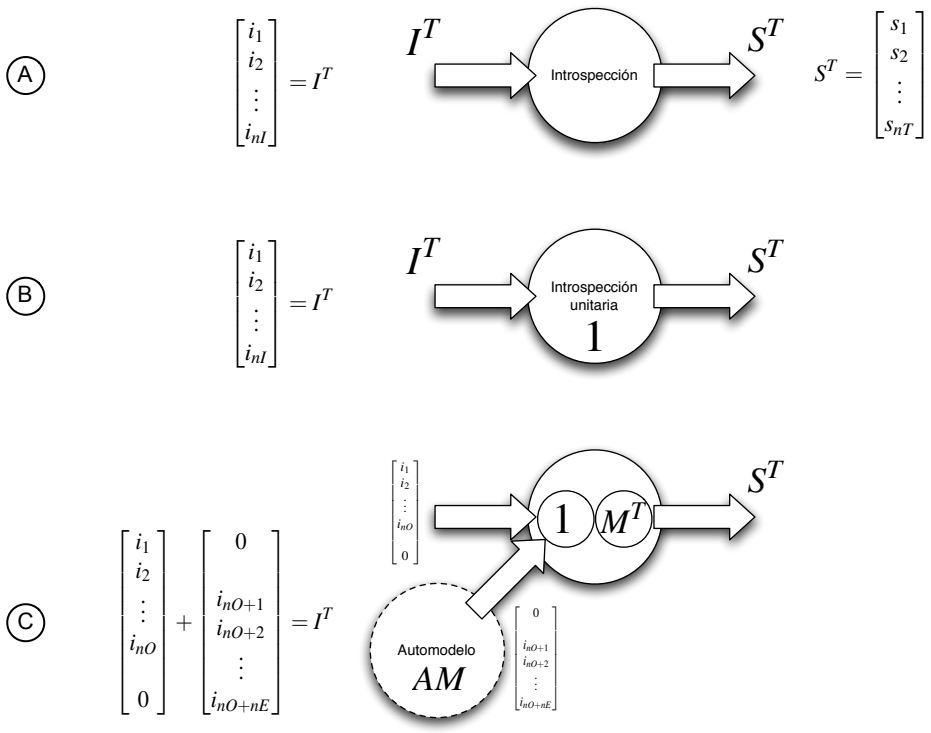

Figura 3.2: Casos de introspección. (A) Caso general. (B) Caso en que el estado de salida se puede obtener directamente de una sensorización. (C) Caso en que solo es posible sensorizar directamente n_{nO} variables, y se sensoriza otras de un automodelo: $i_{nO+1} \ldots i_{nO+nE}$.

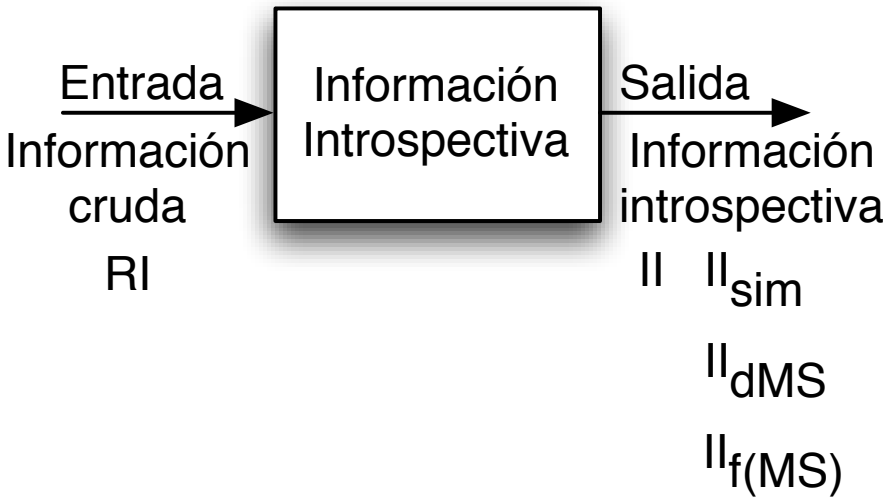

Figura 3.3: Proceso de introspección

un automodelo. La transformación de estas variables en el estado S^T tiene lugar por un *modelo de introspección* indicado por M^T.

Debe mencionarse que el estado reducido S^T puede corresponderse con un estado real del robot —o sistema, en general— o no. En el segundo caso diremos que es un estado *virtual*. Los estados virtuales no están relacionados directamente con el valor de INPUT I^T; equivalen a estados hipotéticos (tales como los que se considerará en el siguiente capítulo de este trabajo).

3.3. Clases de información

Las entradas que recibe un mecanismo de introspección es información cruda (*raw information*), y a partir de esta información se obtiene información introspectiva. Esta información ha recibido un procesamiento previo para obtener la señal de error de la que hablabamos en la sección anterior, que ya es útil para su empleo. En nuestra arquitectura queremos utilizar esta información para tres clases de procesamiento distintas, como podemos apreciar en la figura 3.3:

- Información introspectiva de simulación. (II_{sim})
- Información introspectiva que introduzca nuevas cosas en el modelo de una simulación. ($II_{\triangle MS}$)
- Información introspectiva que mejore cosas ya modelizadas. ($II_{f(MS)}$)

$II = II_{sim} + II_{\triangle MS} + II_{f(MS)}$

La información se puede adquirir directamente o indirectamente. Una información adquirida directamente es aquella que se obtiene a partir de un sensor apropiado para obtener la clase de información necesaria o se lea directamente de un registro. La información adquirida indirectamente es aquella que se computa a partir de otras medidas. Cuando la información se adquiere directamente se puede obtener mediante sensores propios o valiéndose de sensores ya integrados en el sistema a analizar.

3.4. Más allá de la introspección

Una vez obtenida la información introspectiva, esta se puede utilizar en fases posteriores para realizar simulaciones, aprender, tomar decisiones u otros procesos. Podemos apreciar estas fases en la figura 3.4. La introspección es un proceso crítico para poder realizar estas acciones posteriormente.

Una simulación es una imitación de ciertas características o comportamientos de un sistema físico o abstracto determinado. Un ordenador digital es una máquina de estados discretos. Cada transición de estados en este ordenador se llama iteración. En una simulación por ordenador, a partir de una entrada ($< S >_{t0}$) se obtiene una salida ($< S >_{t0+h}$), tras la realización de h iteraciones de la simulación.

El aprendizaje es un proceso en el que a partir de un modelo simulado (MS_{t0}) se obtiene otro modelo simulado (MS_{t1}), tal que el nuevo modelo se obtiene por la realización de mejoras en el modelo inicial y la introducción de cosas nuevas en dicho proceso.

$MS_{t1} = f(MS_{t0}) + \triangle MS_{t0}$

La toma de decisiones es un proceso que elige entre una serie de alternativas basándose en la información introspectiva que se ha obtenido previamente.

3.5. Representación de información sobre magnitudes

Tomamos de la arquitectura RCS [Gazi et al., 1998] [Moore et al., 1999] [Albus et al., 2002] la clasificación de las medidas en niveles distintos en función de su orden de magnitud, con el fin de tratar las medidas con diversas granularidades en función del nivel en el que estemos trabajando. Este es un mecanismo para disminuir la complejidad del análisis de las medidas en

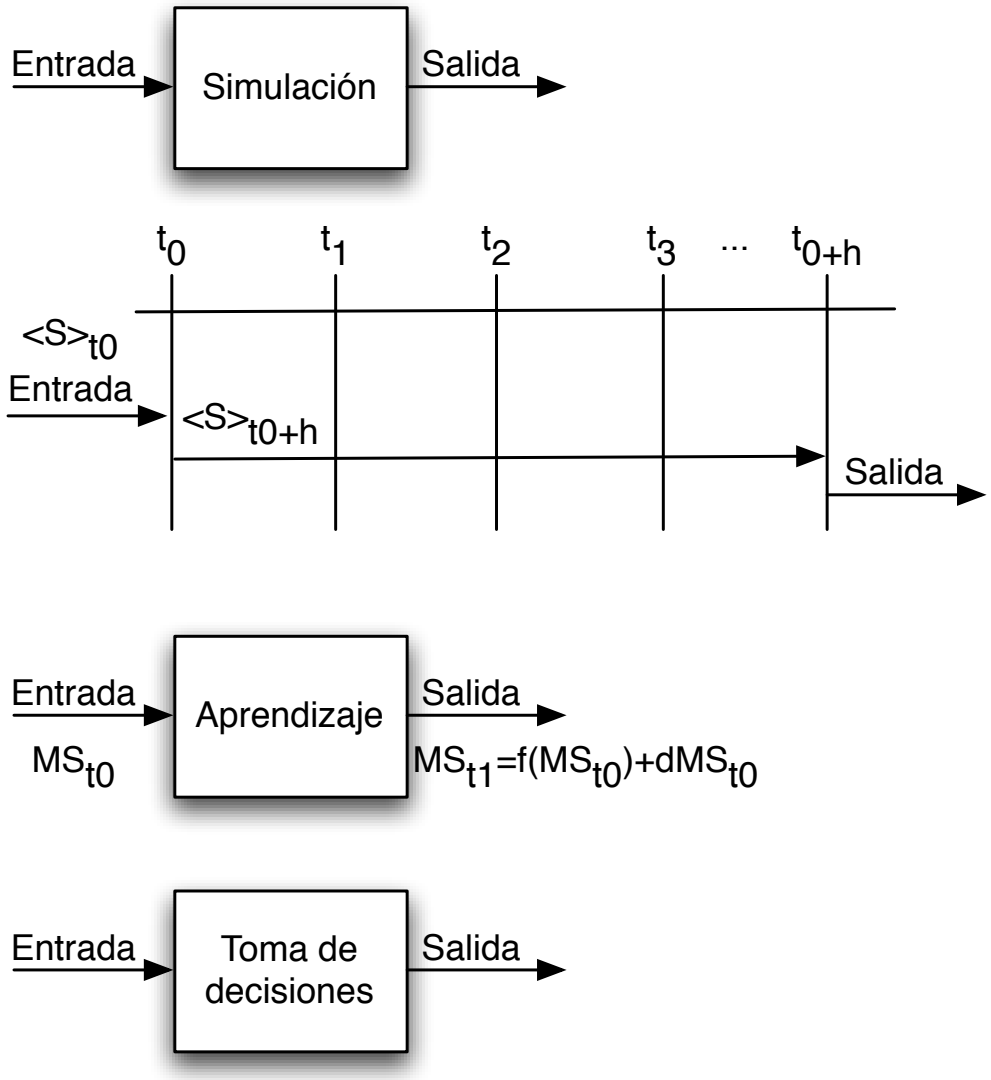

Figura 3.4: Procesos de simulación, aprendizaje y toma de decisiones

función de su nivel de abstracción. La información más abstracta emplea intervalos de espacio y de tiempo más grandes y la información menos abstracta utiliza intervalos de espacio y de tiempo más pequeños.

Veamos brevemente la arquitectura RCS. En esta arquitectura cada nodo tiene cuatro procesos: generador de comportamiento (*behavior generating*), modelado del mundo (*world modelling*), procesado de los sensores (*sensory processing*) y generador de juicios de valor (*value judgment*), junto con una base de conocimiento (*knowledge base*). La metodología RCS comienza con un análisis en profundidad de la misión o la tarea que el sistema debe realizar. Después se codifica el conocimiento de la tarea mediante un árbol de descomposición de tareas en secuencias de subtareas cada vez más y más simples. Este *framework* de descomposición de tareas es mapeado en una jerarquía de unidades organizativas que tienen el conocimiento, las habilidades y las capacidades para llevar a cabo la descomposición de tareas requerida.

En la figura 3.5 podemos apreciar los siguientes componentes: a la derecha el módulo planificador y ejecutor. En el centro hay mapas para representar las características del terreno, las carreteras, los puentes, los vehículos, las posiciones amigas y enemigas y el coste y el riesgo de atravesar las distintas regiones. A la izquierda están las funciones de procesado de datos sensoriales, representación simbólica de entidades y eventos y segmentado de imágenes con regiones etiquetadas. Las transformaciones coordinadas que hay en el medio usan rangos de informaciones para asignar regiones etiquetadas de la jerarquía de entidades de imagenes en el centro-izquierda a las localizaciones en mapas de planificación del centro-derecha. Esto causa que la jerarquía de entidades de clases de la izquierda sea ortogonal a la jerarquía de mapas de planificación usados por los procesos de generación de comportamiento mostrados en la parte derecha.

3.6. Enumeración de mecanismos utilizados para realizar introspección artificial

Los mecanismos que se indican a continuación se pueden utilizar para realizar introspección artificial:

1. Sensores. Permiten obtener medidas del exterior y/o del propio robot.

2. Trazar un proceso. Permite instrumentar el proceso para que facilite información sobre su ejecución.

3. Resolver símbolos. El acceso a variables y funciones de un programa se hace mediante símbolos. Los símbolos exportados son resueltos por

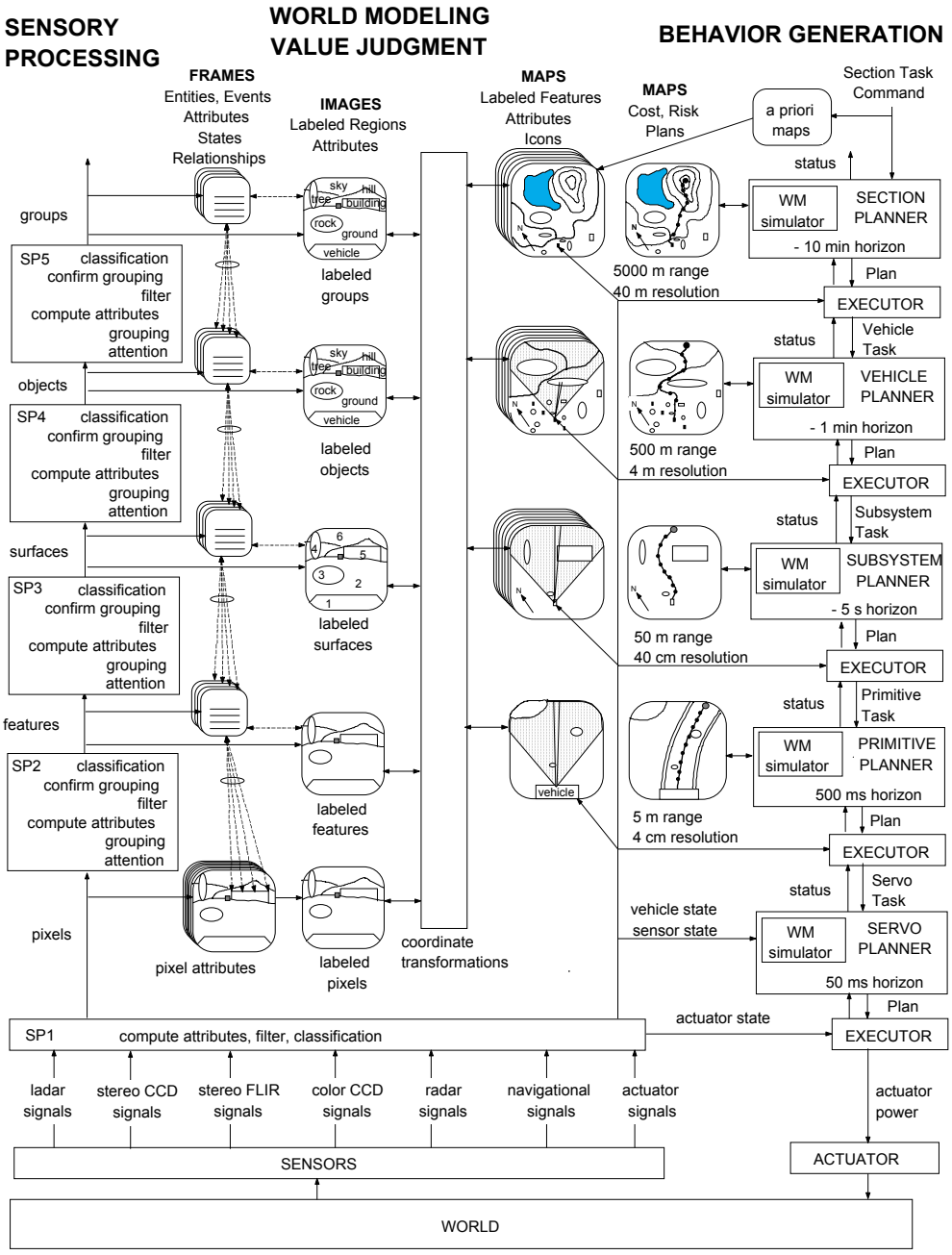

Figura 3.5: Arquitectura RCS en 5 niveles

el *linker* dinámico al comenzar la ejecución de un programa que usa librerías dinámicas. Puede darse el caso de que se necesite resolver un símbolo no exportado. En Linux se puede realizar usando las librerías **libbfd** (*Binary File Descriptor Library*) y **libiberty** (Librería de soporte para el *Toolchain* de GNU: gcc, gdb y binutils)

4. Proceso de intercambio de información. Mecanismos de intercomunicación entre procesos.

Mutex.
Semáforos.
Sockets (UNIX o IP).
Colas de mensajes.
Tuberías (*Pipes*)
Memoria compartida.

Cuadro 3.1: Mecanismos de intercomunicación entre procesos

- Los *mutex* son un mecanismo de exclusión mutua basado en variables compartidas.
- Los semáforos son contadores que se usan para proporcionar acceso a objetos de datos compartidos por múltiples procesos.
- Los *sockets* es uno de los mecanismos de intercomunicación entre procesos más generales que existen, puesto que puede usarse para intercambiar mensajes entre máquinas distintas, además de para intercambiar información entre procesos de una misma máquina, comunicando procesos independientemente de donde se estén ejecutando.
- Las colas de mensajes es un mecanismo de comunicación basado en mensajes. La forma más usada en sistemas UNIX de estas colas de mensajes son las colas de mensajes POSIX. POSIX soporta mesajes asíncronos e indirectos mediante estas colas de mensajes.
- Las tuberías son un mecanismo de comunicación, que en el caso de tuberías sin nombre permite intercambiar datos entre un proceso padre y un proceso hijo tras un `fork()` o mediante las funciones `popen()` y `pclose()`. Las tuberías con nombre (FIFOs) permiten intercambiar datos entre dos procesos cualesquiera, independientemente de que tengan o no un ancestro común.

- La memoria compartida permite que dos o más procesos compartan una determinada región de memoria. Es la forma de comunicación entre procesos más rápida, puesto que los datos no necesitan ser copiados entre el cliente y el servidor, pero tiene el inconveniente de que es necesario sincronizar el acceso a la región de memoria entre los distintos procesos.

Capítulo 4

Introspección artificial para plataforma robótica móvil.

4.1. Plataforma Robótica

El robot es un Pioneer 2AT-8 de ActivMedia, figura 4.1. Es un vehículo con cuerpo de aluminio y cuatro ruedas neumáticas, no directrices, propulsadas por cuatro motores de corriente continua equipados con encoders ópticos de alta resolución para establecer de manera precisa la posición y la velocidad.

4.2. Arquitectura

Una arquitectura robótica define la estructura y la organización del hardware y del software del robot para la creación de capacidades robóticas. Las arquitecturas de robots clásicas se describen en tres arquetipos básicos: deliberativa, reactiva e híbrida. Estos componentes difieren en:

- El manejo de los datos sensoriales.
- Los componentes básicos para construir funcionalidad robótica.
- La disposición de los componentes para facilitar la operación del robot.

Vamos a describir la arquitectura de un robot móvil y el diseño general de los mecanismos de introspección artificial. Estos mecanismos de introspección artificial se integrarán a largo plazo con otros mecanismos para implementar una arquitectura de automodelos que explotar en tiempo real.

Figura 4.1: Plataforma robótica Pioneer 2AT-8

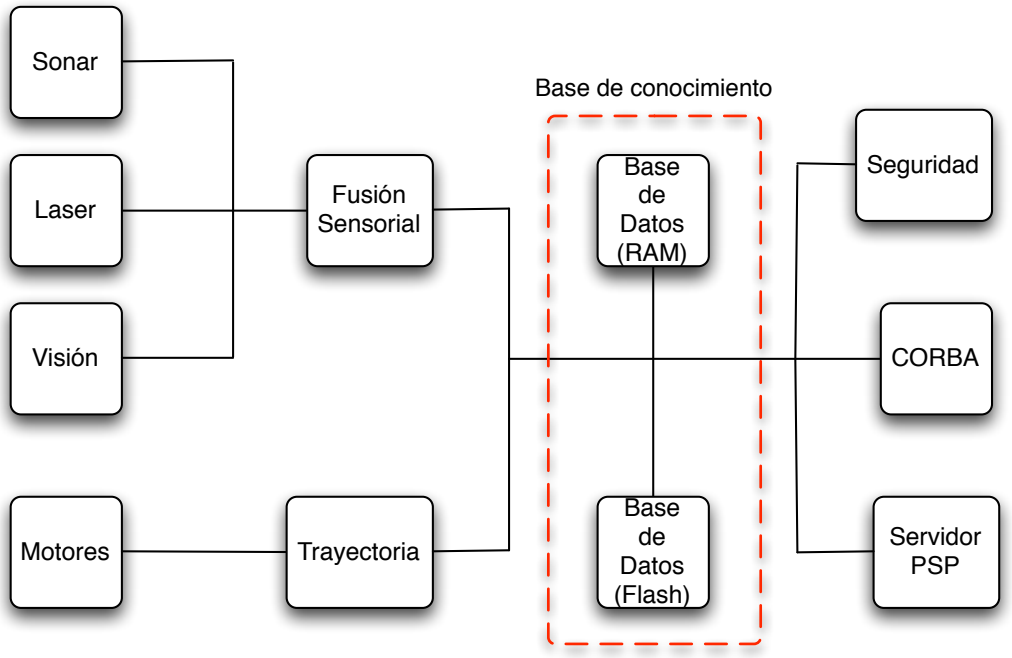

Figura 4.2: Módulos funcionales

4.2.1. Módulos funcionales

El aspecto de la arquitectura que vamos a emplear se puede apreciar en la figura 4.2, que muestra los módulos funcionales de esta arquitectura. La arquitectura que vamos a usar dispone de tres módulos que recogen información de los sensores del robot: sonar, láser y visión. Existe un cuarto módulo que realiza fusión sensorial a partir de los tres módulos ya mencionados.

Otro módulo se encarga del control de los motores del robot y este módulo está acoplado a un segundo módulo que almacena las trayectorias realizadas en memoria y las vuelve a repetir. Este módulo utiliza otro módulo que gestiona una base de datos en memoria RAM, para evitar los lentos tiempos de acceso necesarios para escribir la memoria flash.

Un módulo CORBA enlaza el sistema con otras aplicaciones CORBA y otro que utiliza sockets permite realizar la comunicación con la PSP para utilizarla como control remoto.

Los módulos de CORBA y de comunicación con la PSP, así como los módulos de los sonars, los motores y gestión de trayectorias ya están implementados en una versión inicial. A pesar de ello, el sistema requiere mejoras para afrontar la falta de precisión en los giros debida a la carencia de GPS y/o brújula.

4.2.2. Interfaces

Para la comunicación entre los distintos componentes CORBA empleamos varias IDLs. La interfaz `Pioneer2AT`, incluye las funciones para obtener información de los sensores del robot manejados a través de ARIA y las funciones para controlar el movimiento del robot. La interfaz `Trajectory` controla las funciones relacionadas con el módulo que almacena y repite trayectorias previamente almacenadas. La interfaz `Log` gestiona los registros de actividades y errores relacionados con el sistema en una máquina remota empleando CORBA.

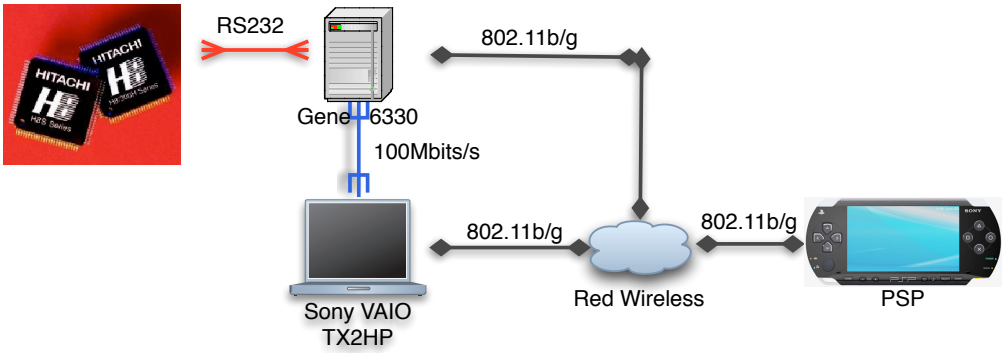

Figura 4.3: Red de comunicaciones del robot

4.2.3. Conectividad

Como muestra la figura 4.3, existen dos ordenadores a bordo del Pioneer 2AT-8: un ordenador empotrado basado en una placa Gene 6330 y un ordenador de 11" de Sony, modelo VAIO VGN-TX2HP. La placa Gene 6330 se conecta al microcontrolador Hitachi H8S del robot a través de un puerto serie RS232. Los dos ordenadores se conectan a otras máquinas remotas empleando redes inalámbricas 802.11b/g. Un ejemplo de máquina remota es la PSP que utilizamos como control remoto. Los dos computadores se conectan entre sí empleando un cable de red cruzado a 100MBits/s *full-duplex*.

Dentro de la gama de protocolos de comunicaciones que puede usarse con el broker CORBA llamado TAO, se ha optado por usar IIOP (sobre TCP/IP).

4.2.4. Base de conocimiento

La base de conocimiento está distribuida entre dos bases de datos, una ubicada en memoria RAM, para agilizar el acceso a la misma, y otra en memoria FLASH, que usaremos cuando el tiempo de acceso no sea tan crítico y queramos que los datos persistan tras un reinicio del sistema.

4.2.5. Interfaz Hombre-Máquina

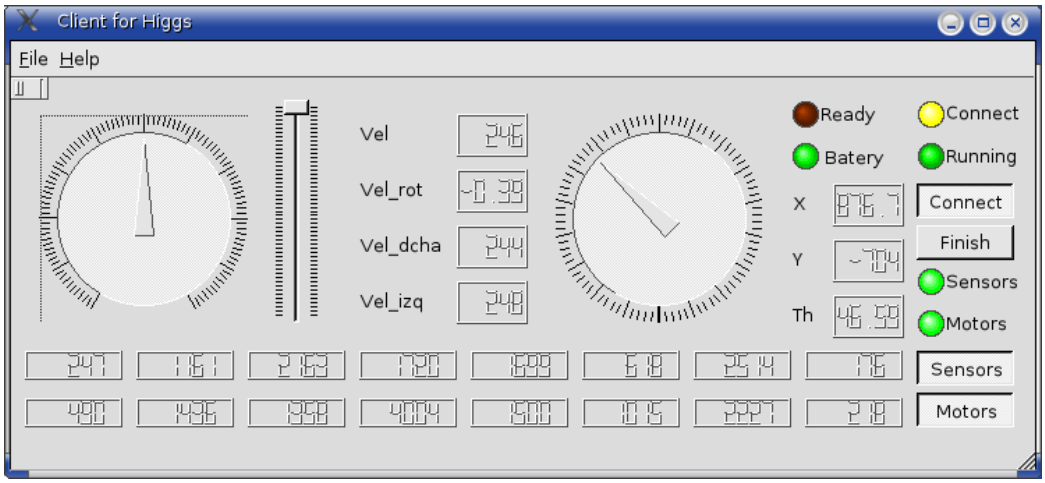

Figura 4.4: Interfaz gráfico de un cliente CORBA

Para facilitar la comunicación con el sistema distribuido se pretende crear diversas interfaces gráficas en ordenadores remotos usando las librerías gráficas Qt de Trolltech, como la figura 4.4 , así como usar el SDK para la PSP conocido como "Homebrew" para realizar un interfaz gráfico para la PSP que se usa como control remoto.

4.3. Ejercicio práctico

Se ha realizado un caso práctico de introspección artificial. En él se simula el comportamiento del robot. Es una simulación dinámica en la que se realizan cálculos de energías. Con esta simulación se pretende poder tomar decisiones acerca de si se sigue o no una trayectoria con el robot.

La pantalla del programa, figura 4.5, representa la topografía del terreno en el que se mueve el robot. Se han realizado varias simplificaciones:

- La primera de ellas es que en la trayectoria del robot no existen obstáculos.

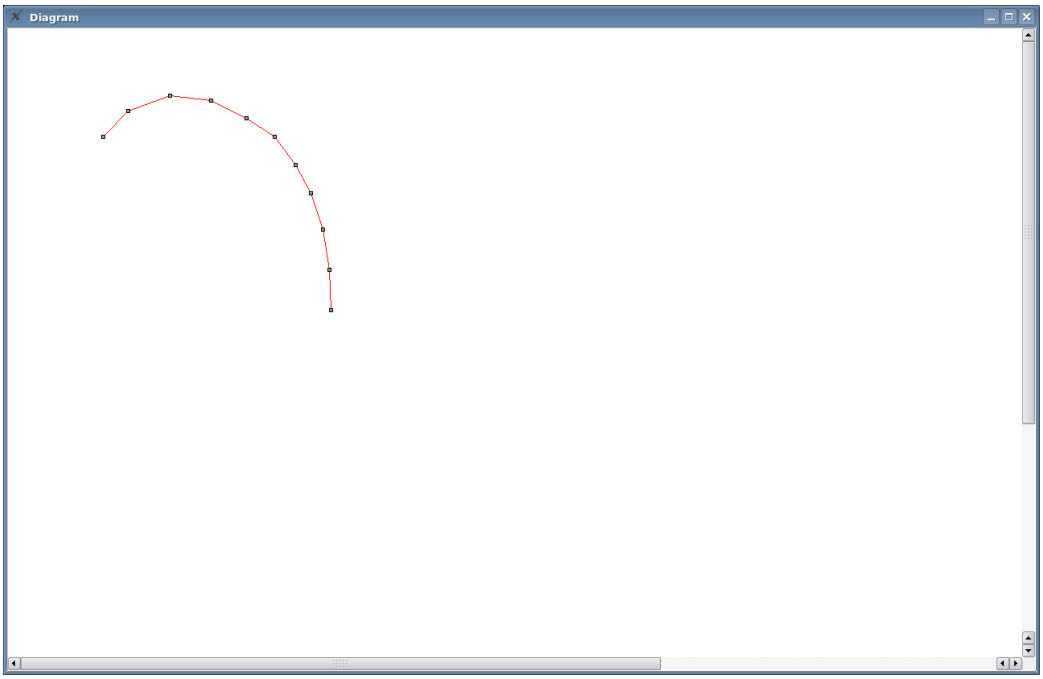

Figura 4.5: Programa práctico de simulación.

- La segunda es que se han modelizado las velocidades mediante unas rampas de velocidad, que se describen en el apartado 4.3.3 de este capítulo.

La simulación resuelve un sistema de ecuaciones diferenciales, que incluye las ecuaciones de los motores: velocidad de rotación, intensidad y voltaje de los mismos, así como las ecuaciones de la velocidad del vehículo: velocidad longitudinal, velocidad lateral, y velocidad de rotación respecto del centro de gravedad del vehículo. Posteriormente se integran las velocidades para poder calcular la posición del vehículo.

4.3.1. Aspectos *software*

El programa ha sido realizado en C y C++, utilizando los algoritmos de [Press et al., 1992] para resolver los métodos Runge-Kutta necesarios para solucionar numéricamente los sistemas de ecuaciones diferenciales (EDOs). La interfaz gráfica se ha realizado utilizando las librerías gráficas Qt de Trolltech. Como los algoritmos del método Runge-Kutta están en C, se ha hecho una clase en C++ llamada *Simulacion* que es un *wrapper* para facilitar el empleo de código en C desde clases de C++. Antes de este desarrollo se

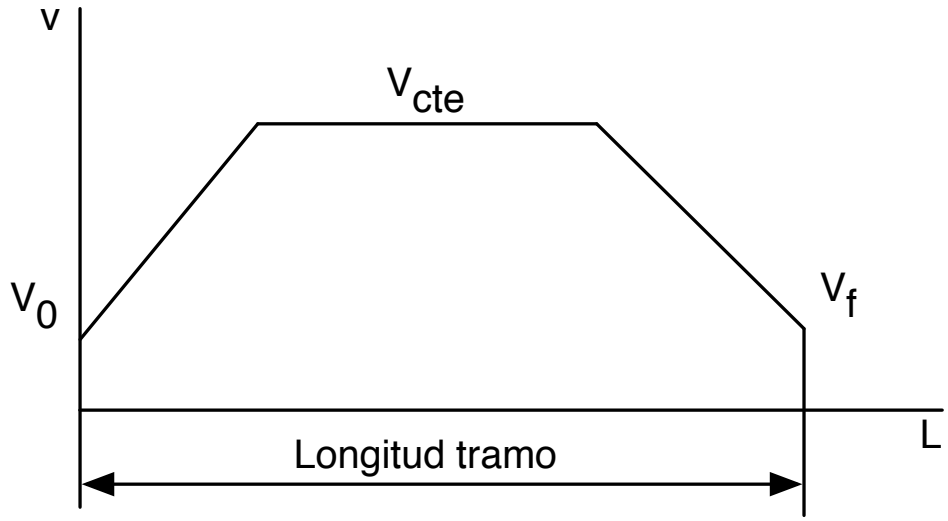

Figura 4.6: Perfil de velocidades.

realizó un desarrollo previo en Matlab, resolviendo las EDOs con el método *ode45* de dicho entorno.

La utilización de un método Runge-Kutta en C, a diferencia del desarrollo anterior, permite utilizar estos algoritmos fuera del entorno de Matlab, permitiendo su uso en un sistema empotrado, como es nuestro robot, sin necesidad de depender de las librerías de Matlab, y con el consecuente aumento de la velocidad de ejecución de dicho algoritmo.

4.3.2. Entorno considerado

El programa permite la introducción del recorrido del vehículo como un segmento o una polilínea. También es posible indicar las cotas de los distintos puntos del segmento o de la polilínea para tomar en consideración la pendiente existente en estos tramos.

4.3.3. Perfil de velocidad

El programa realiza en cada segmento de esta polilínea un recorrido basado en una rampa de aceleración constante, seguido de un tramo de velocidad máxima constante en los casos en los que se llega a alcanzar dicha velocidad, y un tramo de deceleración constante, tal y como muestra la figura 4.6. La velocidad al final de un segmento coincide con la velocidad al comenzar el

segmento siguiente.

En general $v_0 \neq v_f$
$v_{0,j+1} = v_{f,j}$
$v_{max} = v_{cte}$

Hemos establecido la hipótesis de que la velocidad final que se alcanza al terminar cada tramo viene determinada por una fórmula del tipo
$v_f = v_{cte}\left(1 - \frac{|\alpha - 180°|}{180°}\right)$, donde $0° \leq \alpha < 360°$
que limita la velocidad al final de cada tramo en función del ángulo que se forma con el siguiente segmento de la polilínea.

4.3.4. Estimación energía consumida por los motores

El programa realiza una estimación de la energía consumida en el recorrido de la polilínea. Para realizar esta estimación, multiplicamos la intensidad por el voltaje de los motores para poder obtener la potencia instantánea en cada punto del recorrido. Integrando en el tiempo esta potencia obtenemos la energía consumida en el recorrido.

$P = v \cdot i$
$E = \int_0^T v \cdot i \; dt$

4.3.5. Estimación energía remanente de las baterías

El programa también realiza una estimación de la energía que queda en las baterías. Para hacer esta segunda estimación, realizamos una integración a lo largo de la curva de descarga característica de las baterías suministrada por el fabricante de las mismas, figura 4.7. Para escoger la curva característica que vamos a emplear realizamos primero una estimación de la intensidad media consumida a lo largo del recorrido que hemos simulado. Utilizando este dato escogemos la curva característica de descarga correspondiente a esta intensidad media e integramos a lo largo de la curva, desde el tiempo actual hasta la descarga total de las baterías para determinar la energía remanente de las baterías.

Debemos notar, mirando la figura 4.8, que al aproximar la intensidad consumida por los motores por la intensidad media estamos cometiendo un ligero error. En las zonas donde la intensidad consumida es mayor la energía

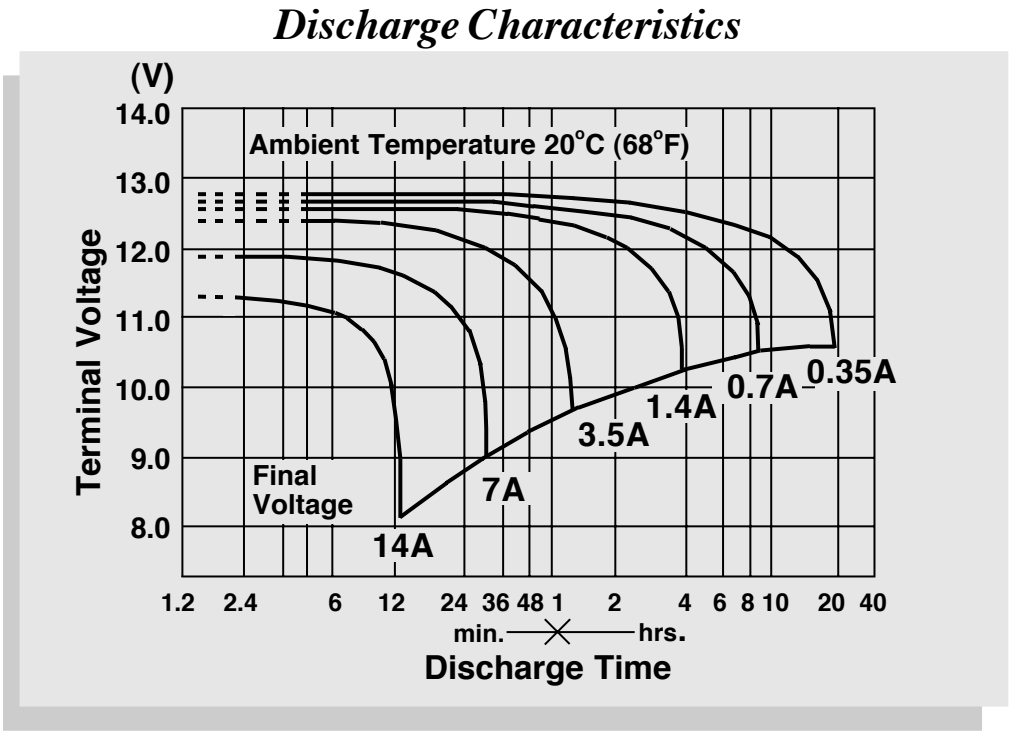

Figura 4.7: Curvas características de descarga de las baterías.

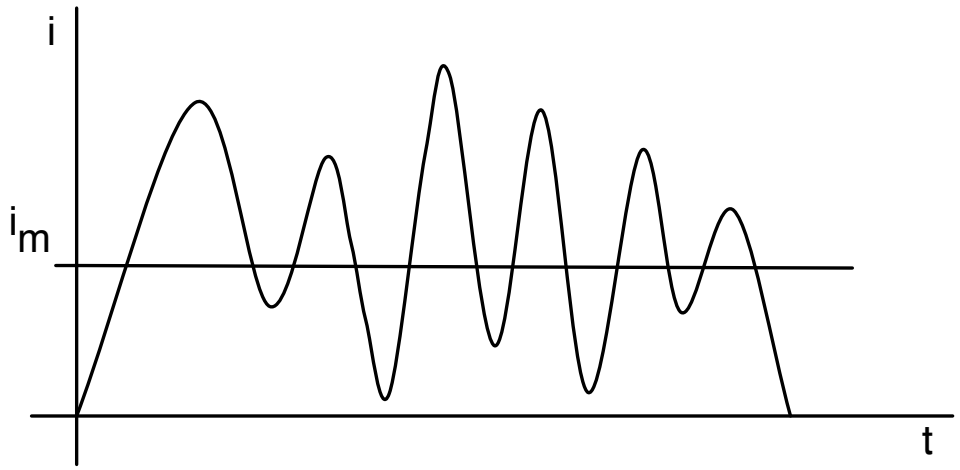

Figura 4.8: Intensidad consumida por los motores. Figura no real, utilizada únicamente con propósitos ilustrativos.

remanente de las baterías fijada por sus curvas de descarga es menor. Ocurre al contrario en las zonas donde la intensidad consumida es menor, siendo la energía remanente de las baterías mayor. El efecto total es un ligero error por exceso en la estimación de la energía remanente de las baterías.

4.3.6. Algoritmo de toma de decisión

La estimación de la energía consumida en un determinado trayecto se utiliza para tomar una decisión respecto a si el trayecto se puede realizar. Si la energía consumida es inferior a la energía total de la batería, entonces el trayecto se puede realizar. En cambio, si la energía consumida es superior a la energía total de la batería, entonces el trayecto no se puede realizar.

$E_{consumida} < E_{total} \Rightarrow$ Trayecto válido
$E_{consumida} > E_{total} \Rightarrow$ Trayecto no válido

Capítulo 5

Conclusión

En este trabajo se ha diseñado un modelo teórico general de introspección. También se ha realizado un ejercicio práctico, con el fin de analizar aspectos críticos de la ingeniería de procesos de introspección. Como resultado de este diseño y la realización del ejercicio práctico, se ha concluido que:

- La adición de una arquitectura introspectiva a un sistema ya existente puede requerir realizar una simulación de variables internas utilizando un automodelo.
- Es posible alterar el funcionamiento normal del sistema, por lo que es necesario tener en cuenta los requisitos de la arquitectura introspectiva, además de los requisitos de la arquitectura del robot.

En el presente trabajo hemos descrito una arquitectura para un robot móvil que implementa un modelo de sí mismo y mecanismos para explotar este modelo por medio de la introspección. Este es el caso de la figura 5.1, en el que se sensoriza parte de las variables directamente, y parte a través de un automodelo.

Para entender esta arquitectura hemos explicado que entendemos por introspección, las clases de información que maneja un mecanismo de introspección, cómo manejamos las medidas en distintos niveles en función de sus magnitudes y los mecanismos que se pueden utilizar para implementar introspección artificial.

Se está realizando una implementación de la arquitectura mostrada en la figura 4.2. En este trabajo se realiza un examen de los mecanismos de introspección utilizados en esta arquitectura. Examinemos primero el estado actual de esta arquitectura: se han realizado módulos para manejar los sónars y los motores, el módulo de trayectorias, así como el módulo CORBA, el módulo servidor para la PSP y una base de datos en memoria RAM. Estos

módulos permiten guiar el robot bajo control humano y repetir trayectorias almacenadas en la base de datos, con el inconveniente de la falta de precisión en la orientación del robot, que de momento carece de brújula y/o GPS para mejorar este aspecto.

En el futuro, se pretende programar los módulos restantes (visión, láser y fusión sensorial), y mejorar los existentes, construyendo una arquitectura de control para que el robot pueda realizar tareas autónomamente.

La figura 3.1 nos muestra un esquema conceptual de la arquitectura de control que usamos, compuesta de la arquitectura del robot y la arquitectura introspectiva. Esta arquitectura utiliza los mecanismos básicos de introspección mencionados en la sección 2.2 y los casos de sensorización mencionados en la sección 3.2.

Hemos comentado que la arquitectura introspectiva es capaz de alterar el comportamiento normal del sistema. Veamos varios grupos de problemas que se pueden producir en esta arquitectura de control: ancho de banda, CPU, memoria, hardware.

- La velocidad de comunicación de varios enlaces de comunicaciones en serie viene limitada por el enlace más lento. En el caso particular de nuestro robot, figura 4.3, el enlace entre el microcontrolador Hitachi H8S y la placa GENE 6330 se realiza a través de un puerto serie, limitando de esta manera el ancho de banda de las comunicaciones del ordenador con el microcontrolador del robot. A través de este enlace se envían paquetes periódicos con información del estado del robot al ordenador y se reciben comandos en el microcontrolador. Aumentando la velocidad de la comunicación de este enlace ha sido posible aumentar la frecuencia de estos paquetes. En el caso de controlar el robot a través de una unidad remota por medio del enlace inalámbrico (como la PSP) o desde el VAIO TX2/HP por medio de un enlace ethernet, como la red inalámbrica (en condiciones favorables) y el enlace ethernet son más rápidos que el enlace serie, este último será el que límite la velocidad de la comunicación.

- La carga de la CPU es también otro factor limitante. Al utilizar el planificador de Linux de tiempo real para mejorar el comportamiento en tiempo real de algunos de los procesos que ejecutamos en la placa GENE 6330 debemos realizar alguna llamada al sistema que bloquee el hilo que estamos ejecutando. De esta manera evitamos que el proceso de tiempo real acapare toda la CPU al tener mayor prioridad que el resto de procesos del sistema, que se ralentizan de una manera extrema

saliéndose de las especificaciones requeridas por los mismos. Estamos estudiando la posibilidad de sustituir la placa GENE 6330 por otra más potente, aunque probablemente tenga unos requisitos de potencia eléctrica mayores.

- La memoria tiene dos factores limitantes: el tamaño y la velocidad. Es usual disponer en los sistemas que se emplean hoy en día de varias clases de memorias. Las memorias más rápidas suelen tener menos capacidad, siendo mayor en cambio la capacidad de las memorias más lentas. Como ejemplo hemos mencionado en la sección 4.2.4 que los requisitos de nuestro módulo de trayectorias nos impide utilizar la memoria flash por su lento tiempo de acceso al escribir en la misma, siendo en cambio adecuada la memoria RAM, más rápida, aunque de menor capacidad. Sistemas más complejos dispondrán de más clases de memorias, con lo que se podrá trabajar con más memorias de distintas capacidades y velocidades. Ya empieza a ser habitual la venta de equipos que disponen de memoria RAM, memorias de estado sólido y discos duros.

- Respecto al hardware, existen otros problemas como puede ser la precisión y la falta de linearidad de los sensores, la calibración de los sensores y los actuadores, el guiado y la odometría de un robot móvil. También podemos destacar las interferencias electromagnéticas de los actuadores, especialmente cuando hay motores, con los sensores más sensibles. Estos problemas desaparecerán utilizando sensores precisos y lineales o usando una etapa de acondicionamiento, calibrando los sensores y los actuadores, utilizando GPS/brújula y separando adecuadamente los sensores de los actuadores que causan interferencias electromagnéticas.

Entre las líneas futuras a seguir, podemos destacar:

- Implementar los mecanismos de introspección sobre el robot real.

- Añadir los módulos funcionales que todavía no existen en la arquitectura del robot y mejorar los existentes.

- Agregar las interfaces IDL necesarias para estos módulos funcionales.

- Estudiar los procesos de toma de decisiones que se realizarán posteriormente a la obtención de información introspectiva.

Examinemos el caso de introspeción realizado en el ejercicio práctico. En la figura 5.1 mostramos un diagrama con este caso de introspección. Como vimos en la sección 1.4 de la introducción, se sensoriza unas medidas reales

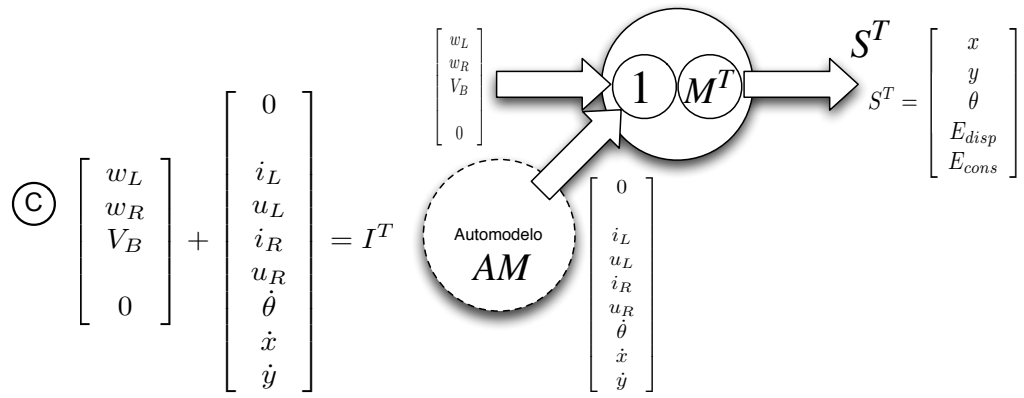

Figura 5.1: Caso introspección ejercicio práctico

dadas por las librerías de la plataforma: las velocidades angulares de giro de las ruedas y el voltaje de las baterías.

El modelo de introspección M^T recoge el perfil de velocidades, sección 4.3.3, la modelización del entorno, sección 4.3.2, y el empleo de la intensidad media consumida en el trayecto para escoger la curva de descarga de las baterías, de la sección 4.3.5.

Como también vimos en la introducción, sección 1.4, el automodelo se ha realizado implementando dos modelos matemáticos: la dinámica del robot, apéndice A, y la capacidad energética de las baterías.

Partiendo de las medidas reales sensorizadas que ya hemos comentado, y utilizando un modelo de introspección y un automodelo basado en los dos modelos matemáticos que acabamos de comentar, logramos hacer una estimación de la posición del robot, así como de la energía disponible en las baterías y la energía consumida durante el trayecto. Estas estimaciones constituyen el estado reducido S^T y serán usadas para tomar la decisión de recorrer o no el trayecto en cuestión, como se contemplaba en la introducción.

Apéndice A

Modelizado

Nuestro objetivo es derivar un modelo dinámico, describiendo el Pioneer 2AT-8, con el fin de controlarlo. A veces un modelo cinemático puede ser una descripción suficiente, pero con el fin de incluir el contacto rueda-superficie, el modelo cinemático ha sido extendido con una parte dinámica.

El modelado del robot móvil con ruedas debe incluir los cambios del entorno, con el fin de poder manejarlos. Antes de la parte de modelado real se deben hacer algunas hipótesis que clarifican y simplifican el proceso. Se introducen a continuación dos hipótesis acerca de la construcción y el entorno circundante.

1. El robot Pioneer 2AT-8 no contiene partes flexibles.
2. Sólo se consideran movimientos a bajas velocidades — por lo que no hay deslizamiento longitudinal.

La primera hipótesis habilita el uso de la mecánica de sólidos rígidos al modelizar la cinemática del vehículo. La segunda hipótesis es razonable para vehículos pequeños, puesto que los movimientos lentos y los bajos pesos implican que este tipo de problema no puede llegar a ocurrir.

Con estas hipótesis, la única influencia del entorno es el contacto entre las ruedas y el suelo. Este contacto tiene que ser analizado y la influencia desconocida o cambiante debe ser descrita. Por ello se necesita un modelo dinámico para el problema propuesto.

El robot móvil con ruedas está equipado con un controlador cuando es suministrado por la fábrica. Como se indica en el manual del vehículo [Robotics, 2002, p. 36-37], es posible controlar la velocidad longitudinal del centro de masas, así como la velocidad rotacional del mismo. Este tipo de control se puede realizar como se muestra en la figura A.1 Como se puede ver en la figura, el vehículo es controlado por un lazo externo que proporciona referencias al controlador del motor. Usando este tipo de control, un lazo

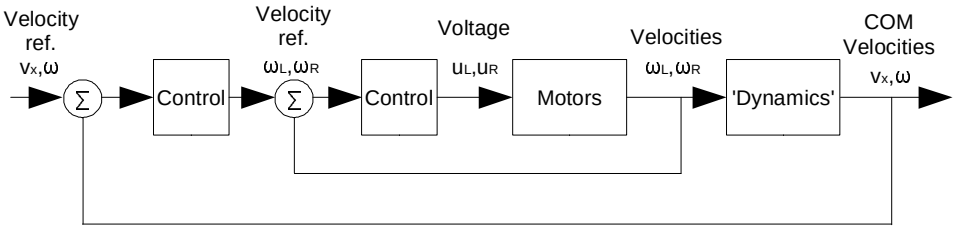

Figura A.1: Controlador instalado en el Pioneer 2AT-8, donde las velocidades del centro de masas son controladas por un lazo de control interno del motor y un lazo de control externo.

de control externo puede controlar el camino del vehículo proporcionando velocidades de referencia al controlador interno. No se indica cómo está implementada la dinámica y el controlador dinámico o si está incluida y cómo la fricción, por lo que este controlador no es usado.

Es posible controlar la velocidad angular de las ruedas en cada lado del vehículo, como explica el manual, en lugar de las velocidades del centro de masas. Mirando la figura, se retira el lazo de control externo y se usa únicamente el lazo de control interno con el motor y el controlador del motor. De esta manera se puede diseñar e implementar la dinámica del controlador, tratando las alteraciones del entorno. Así se puede analizar la dinámica y establecer un modelo.

El lazo de control interno se tiene que analizar también, puesto que este tendrá efecto en los lazos de control externos. De esta manera, un lazo de control nuevo, que reemplace el control dinámico ya disponible, se ocupará de las alteraciones. Finalmente, se tiene que diseñar un lazo de control externo, que permita al vehículo seguir una trayectoria determinada de una posición a otra. La estrategia de control propuesta se muestra en la figura A.2, en la que se usa el controlador del motor interno, y los lazos de control externos, marcados con líneas gruesas, muestran las partes nuevas.

Cuando se mira esta estrategia de control, se debe considerar las entradas para los actuadores y las salidas para los sensores. Mirando al robot móvil con ruedas como una unidad, las entradas disponibles son las velocidades angulares de referencia de los motores. Las salidas son los encoders de los motores, que proporcionan la posición/velocidad angular del rotor del motor, que es proporcional a la velocidad angular de la rueda. También están instalados varios grupos de sonars: un grupo de sonars delantero y otro grupo de sonars trasero. Estos sonars se pueden usar para construir un mapa del entorno. Usando la información de estos sensores puede ser posible estimar la posición y la velocidad del vehículo. En este momento estamos instalando

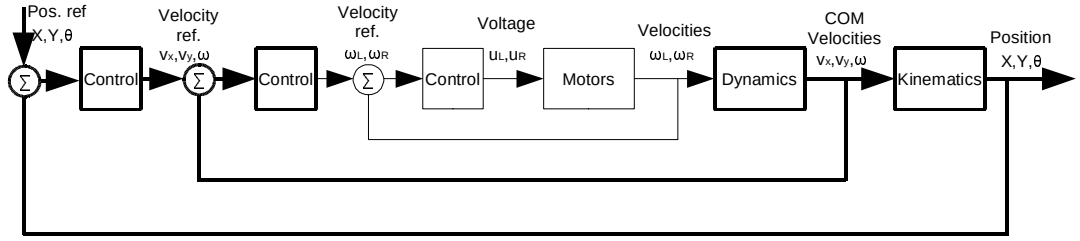

Figura A.2: El lazo de control interno que controla el motor ya está disponible en el vehículo, mientras que los lazos externos de control, de la dinámica y el control de la trayectoria, tiene que ser diseñados.

nuevos sensores, que pueden ser necesarios en una etapa posterior.

En este proyecto se supone que las salidas disponibles son las velocidades longitudinal, lateral y rotacional del centro de masas del vehículo. Una vez que el vehículo esté ajustado con otros sensores, el control puede cambiar completamente.

El modelado del robot móvil con ruedas está dividido en partes para facilitar su visión general. En un primer lugar se estudia la cinemática, después la dinámica y finalmente los motores y los controladores de los motores.

A.1. Cinemática

La cinemática relacionará el vehículo con un marco de referencial inercial.

En primer lugar se introduce un conjunto de ejes de referencia inercial fijos $F(X, Y, Z)$, después se introduce un eje de referencia $f(x, y, z)$, con origen en el centro de masas del Pioneer 2AT-8, quedando el eje x en dirección longitudinal al vehículo, el eje y en dirección transversal al vehículo y el eje z como se muestra en la figura. El marco de referencia se moverá junto con el robot, posicionando al vehículo con respecto al marco de referencia inercial. Los ejes se muestran en la figura A.3, para que el lector pueda verificar los conceptos.

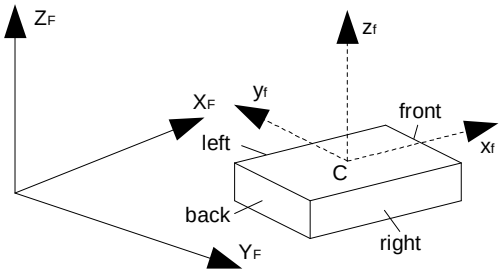

Figura A.3: Marco de referencia inercial $F(X, Y, Z)$ y marco de referencia $f(x, y, z)$ fijo respecto al robot móvil con ruedas.

La cinemática es la descripción de la posición y la orientación del centro de masas, C del vehículo móvil con ruedas en el marco inercial, respecto de las velocidades dadas por el marco del vehículo. En el marco $f(x, y, z)$, el vector $q = [\dot{x}, \dot{y}, \omega]^T$ describe las velocidades lineal y angular del vehículo, en el marco del vehículo. En el marco $F(X, Y, Z)$, $Q = [X, Y, \theta]$ describe la posición del vehículo, donde $[X, Y]$ describe la posición del marco del vehículo, y θ es la orientación del marco del vehículo. En este caso $\dot{Q} = [\dot{X}, \dot{Y}, \dot{\theta}]$ describe las velocidades del marco del vehículo visto desde el marco inercial, como muestra la figura A.4. Se puede verificar fácilmete que la relación entre los marcos se puede describir por medio de la ecuación A.1.

$$\begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ \theta_0 \end{bmatrix} + \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{A.1}$$

La descripción presentada es una descripción general de un marco moviéndose respecto a otro marco de referencia. No se ha incluido ninguna restricción como tal, pero en la realidad el vehículo móvil con ruedas está sujeto a restricciones que deben ser consideradas.

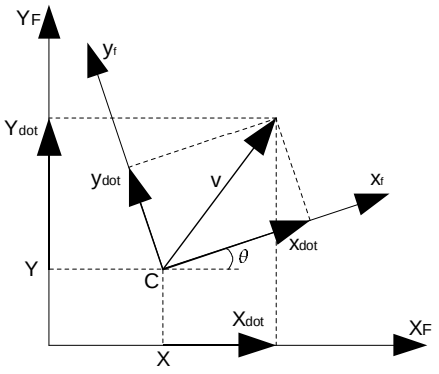

Figura A.4: Vector de velocidades lineales v descrito en el $f(x, y, z)$ así como en el inicial $F(X, Y, Z)$. Como sólo se considera el movimiento en el plano, el eje z ha sido omitido de la figura.

La ecuación A.1 proporciona la descripción de la velocidad del centro de masas del marco de referencia inercial, desde las velocidades del centro de masas en el marco de referencia del vehículo. Como se muestra más tarde, la dinámica es una descripción de cómo afectan las fuerzas de cada rueda en el centro de masas. La descripción dinámica todavía se realiza usando las velocidades del centro de masas, puesto que las velocidades de cada rueda se describen usando las velocidades del centro de masas.

A.2. Dinámica

La dinámica del vehículo se describe a través de las fuerzas que actúan sobre el mismo. La fuerza motriz que actúa desde la entrada, y las fuerzas de fricción que actúan contra la fuerza motriz. Estas fuerzas de fricción son importantes, puesto que se parecen a la conexión entre el vehículo y el entorno. Para el robot Pioneer 2AT-8, las ecuaciones dinámicas se muestran en las ecuaciones A.2, A.3 y A.4 [Luka Caracciolo, 1999].

$$ma_x = \sum_{i=1}^{4} F_{xi} - \sum R_i = F_{1x} + F_{2x} + F_{3x} + F_{4x} - (R_x + R_g + R_r) \tag{A.2}$$

$$ma_y = -F_y \tag{A.3}$$

$$I\ddot{\theta} = d(F_{2x} + F_{4x} - F_{1x} - F_{3x}) - M_r \tag{A.4}$$

Puesto que las ruedas esán acopladas en los lados izquierdo y derecho respectivamente, las fuerzas longitudinales hacia adelante F_{ix} , $i = 1, 2, 3, 4$ están acopladas como $F_{1x} = F_{3x}$ y $F_{2x} = F_{4x}$. Por lo tanto las ecuaciones se representan como:

$$\begin{aligned} ma_x &= 2F_{1x} + 2F_{2x} - (R_x + R_g + R_r) \\ ma_y &= -F_y \\ I\ddot{\theta} &= 2d(F_{2x} - F_{1x}) - M_r \end{aligned}$$

Estas ecuaciones se usan para describir el vehículo mediante un conjunto de ecuaciones diferenciales. Por ello, estas ecuaciones se reescriben como las ecuaciones A.5, A.6 y A.7

$$\ddot{x} = \frac{2F_{1x} + 2F_{2x} - (R_x + R_g + R_r)}{m} \tag{A.5}$$

$$\ddot{y} = \frac{-F_y}{m} \tag{A.6}$$

$$\ddot{\theta} = \frac{2d(F_{2x} - F_{1x}) - M_r}{I} \tag{A.7}$$

Las variables F_x, F_y, R_x, R_g, R_r y M_r deben ser determinadas.

Las fuerzas de estas ecuaciones son las fuerzas que actúan sobre el centro de masas del vehículo. Se calculan teniendo en cuenta las fuerzas que actúan en cada rueda del robot móvil con ruedas, que se representan en la figura A.5.

La fuerza motriz hacia delante F_{ix} está relacionada con el torque del motor, la fuerza de resistencia a la rodadura R_{ix}, que describe la fricción

longitudinal, está relacionada con la velocidad longitudinal de la rueda. La fuerza de resistencia lateral F_{iy} depende de la velocidad lateral de la rueda y, finalmente, el momento resistente M_r alrededor del centro de masas, depende tanto de la velocidad longitudinal como la velocidad lateral y angular. Estas tres últimas están consideradas como fuerzas de fricción.

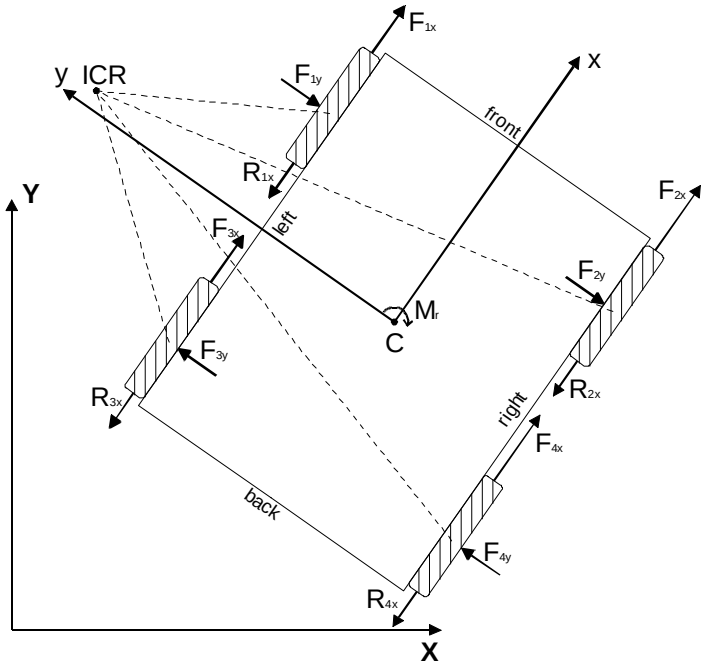

Figura A.5: Fuerzas que actúan sobre el vehículo. Están representadas las fuerzas activas, las fuerzas longitudinal y lateral de fricción y el momento resistente alrededor del centro de masas.

Las fuerzas motrices que actúan sobre las ruedas están relacionadas con el torque de los motores. Se asume que las ruedas son discos no deformables (cuerpos rígidos) y, por ello, el torque está directamente relacionado con las fuerzas motrices:

$$F_{ix} = \frac{\tau_i}{r_i}$$

donde r_i es el radio de la rueda. Se asume que el radio es idéntico para todas las ruedas — por lo tanto $r_i = r$.

Las fuerzas de reacción N_i reaccionan desde la superficie hacia la rueda, y están relacionadas con el peso y la gravedad por la ecuación:

$$\sum_{i=1}^{4} N_i = m \cdot g$$

donde m es la masa del robot móvil con ruedas y g es la aceleración de la gravedad.

Como el vehículo ha sido construido manteniendo la simetría, se supone que el peso está uniformemente distribuido y por ello, teniendo en cuenta la transferencia de carga entre ejes, las fuerzas de reacción se describen mediante las ecuaciones:

$$\begin{aligned} N_1 &= N_2 = \frac{P\cos\Theta\cdot b - \left(\frac{P}{g}\ddot{x} + F_{xa} + P\sin\Theta\right)h - F_{za}b + M_{ya}}{L} \\ &\approx \frac{P\cos\Theta\cdot b - \left(\frac{P}{g}\ddot{x} + P\sin\Theta\right)h}{L} \end{aligned} \tag{A.8}$$

$$\begin{aligned} N_3 &= N_4 = \frac{P\cos\Theta\cdot a + \left(\frac{P}{g}\ddot{x} + F_{xa} + P\sin\Theta\right)h - F_{za}a - M_{ya}}{L} \\ &\approx \frac{P\cos\Theta\cdot a + \left(\frac{P}{g}\ddot{x} + P\sin\Theta\right)h}{L} \end{aligned} \tag{A.9}$$

Como el robot móvil con ruedas sólo se mueve a bajas velocidades, se desprecian las fuerzas centrífugas, y por ello se pueden usar las ecuaciones previas para el vehículo en movimiento.

La fuerza de reacción se emplea para describir tanto la fuerza resistente longitudinal como la fuerza resistente lateral. Ambas están consideradas como fuerzas de fricción y se modelan por medio de la fricción de Coulomb. La fuerza longitudinal se describe como:

$$R_{xi} = \mu_{xc}\cdot N_i\cdot sign(v_{xi}) \tag{A.10}$$

$$R_x = \sum_{i=1}^{4} R_{xi} = \mu_{xc}\cdot\frac{m\cdot g}{2}(sign(v_L) + sign(v_R))\cos(\Theta) \tag{A.11}$$

Hay otras dos fuerzas resistentes, la fuerza resistente a la rodadura y la fuerza resistente gravitacional:

$$R_r = (f_0 + f_v\cdot\dot{x}^n)\cdot mg\cdot sign(\dot{x})\cdot\cos(\Theta) \tag{A.12}$$

$$R_g = mg\sin\Theta \tag{A.13}$$

La fuerza de reacción lateral se representa como:

$$F_{yi} = \mu_{yc}\cdot N_i\cdot sign(v_{yi}) \tag{A.14}$$

$$\begin{aligned} F_y &= \sum_{i=1}^{4} F_{yi} = \mu_{yc}\left(\frac{mg\cos\Theta\cdot b - (m\ddot{x} + mg\sin\Theta)h}{2L}sign(v_F)\right. \\ &\quad \left. + \frac{mg\cos\Theta\cdot a + (m\ddot{x} + mg\sin\Theta)h}{2L}sign(v_B)\right) \end{aligned} \tag{A.15}$$

Finalmente, el momento de resistencia alrededor del centro de masas se representa como:

$$\begin{aligned} M_r &= a(F_{1y}+F_{2y}) - b(F_{3y}+F_{4y}) + d\left((R_{2x}+R_{4x}) - (R_{1x}+R_{3x})\right) \\ &= a\mu_{yc}\frac{mg\cos\Theta\cdot b - (m\ddot{x}+mg\sin\Theta)h}{2L}sign(v_F) \\ &\quad -b\mu_{yc}\frac{mg\cos\Theta\cdot a + (m\ddot{x}+mg\sin\Theta)h}{2L}sign(v_B) \\ &\quad +\mu_{xc}\frac{dmg}{2}(sign(v_R) - sign(v_L)) \end{aligned} \tag{A.16}$$

Respecto a la inercia del vehículo, se han preparado componentes adicionales para su instalación. Estos incluyen un láser Sick, un GPS, una cámara estereoscópica, así como una articulación robótica de dos grados de libertad para girar la cámara. Puesto que estos componentes tienen un peso considerable, se debe considerar como influyen en la inercia del sistema.

Una vez que se han determinado las fuerzas de las ecuaciones del movimiento, se relaciona el torque proporcionado por cada motor con la velocidad del centro de masas. En esta descripción se usan las velocidades longitudinal y lateral de cada rueda, expresándose éstas por medio de estados mensurables, las velocidades del centro de masas del vehículo. Para simplificar los

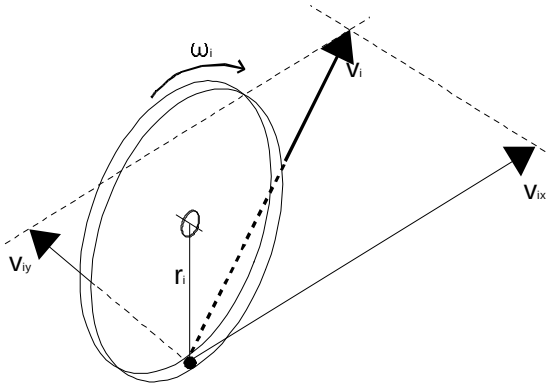

Figura A.6: Las ruedas se modelan como discos uniformes, con un único punto geométrico de contacto con el suelo.

cálculos, se utiliza un modelo reducido de la rueda. Se supone que la rueda es un disco, siempre perpendicular a la superficie, que no es deformable y que tiene un único punto geométrico de contacto con el suelo.

La rueda se muestra en la figura A.6, donde r_i es el radio de la rueda, y ω_i es la velocidad angular de la rueda.

Hay cuatro ruedas, y en este punto cada rueda está caracterizada por una velocidad angular. Se supone que no hay deslizamiento longitudinal entre la rueda y la superficie, por lo que la velocidad longitudinal de las ruedan queda descrita por la ecuación A.17.

$$v_{ix} = r_i \cdot \omega_i \quad , \quad i = 1, 2, 3, 4. \tag{A.17}$$

Cuando el vehículo se mueve, este sigue una trayectoria circular alrededor del CIR. Con el fin de describir como afectan las velocidades de las ruedas al centro de masas, es posible mapear ambas partes respecto al CIR y encontrar la relación.

En la figura A.7, se muestra el robot móvil con ruedas, con las velocidades del centro de masas, de sus ruedas y del CIR.

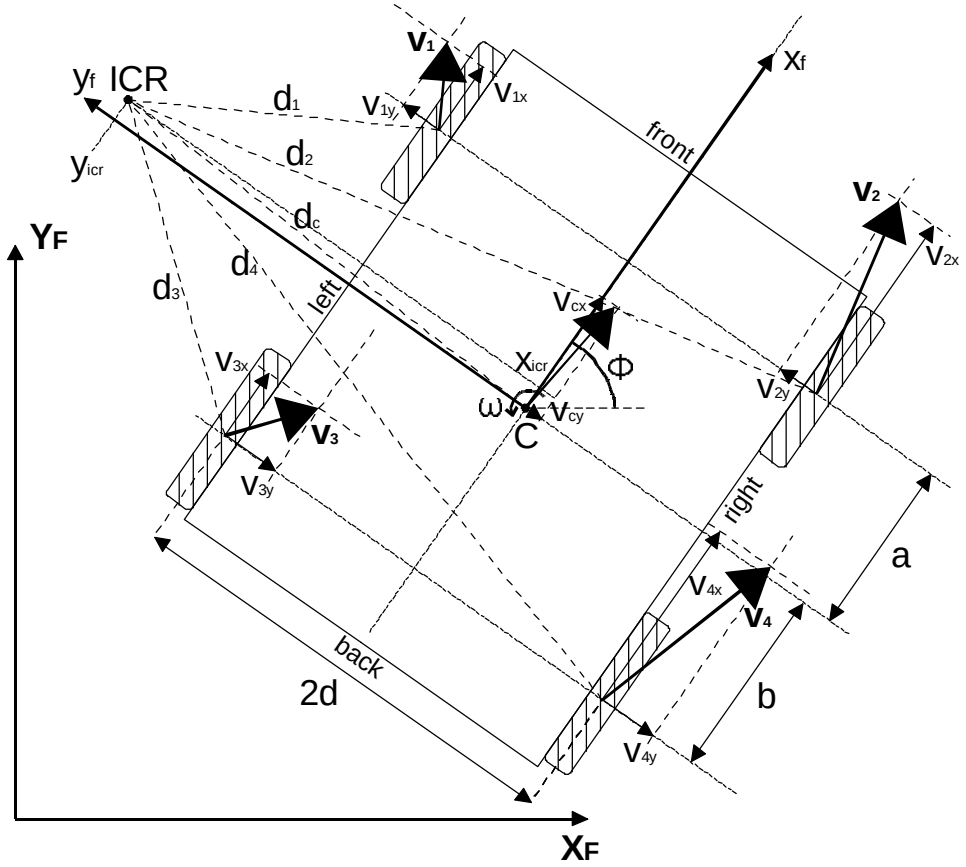

Figura A.7: Vista general del Pioneer 2AT-8 usado en este proyecto, las velocidades son todas perpendiculares al CIR, y se puede establecer una relación.

Los vectores, $d_i = [d_{ix} \quad d_{iy}]^T \quad , \quad i = 1, 2, 3, 4$ de cada rueda al CIR y también $d_c = [d_{cx} \quad d_{cy}]^T$ del centro de masas al CIR están definidos, con el fin de establecer una relación. Puesto que la velocidad de cada rueda sigue una trayectoria circular alrededor del CIR, la siguiente relación es válida, comparada con la ecuación A.17:

$$\frac{||v_i||}{||d_i||} = \frac{||v||}{||d_c||} = |\omega| \tag{A.18}$$

También es válido para las componentes de cada vector, descritas como:

$$\frac{v_{ix}}{-d_{iy}} = \frac{v_x}{-d_{cy}} = \frac{v_{iy}}{d_{ix}} = \frac{v_y}{d_{cx}} = \omega \tag{A.19}$$

Que conduce a las coordenadas del CIR. Las coordenadas del CIR en el marco de referencia local se describen como:

$$\begin{aligned} ICR(x,y) &= (x_{IRC} \ , \ y_{IRC}) = (-d_{cx} \ , \ -d_{cy}) \\ &= \left[\frac{v_y}{\omega}, \frac{v_x}{\omega}\right] \end{aligned} \tag{A.20}$$

La descripción previa también se puede reescribir con el fin de facilitar la vista general de los cálculos siguientes.

$$\frac{v_x}{y_{IRC}} = -\frac{v_y}{x_{IRC}} = \omega$$

A través de la figura A.7 se puede verificar fácilmente que las siguientes ecuaciones son válidas:

$$\begin{aligned} d_{1y} = d_{3y} &= d_{cy} + d \\ d_{2y} = d_{4y} &= d_{cy} - d \\ d_{1x} = d_{2x} &= d_{cx} + a \\ d_{3x} = d_{4x} &= d_{cx} - b \end{aligned} \tag{A.21}$$

Combinando las ecuaciones A.19 y A.21 se encuentran las siguientes relaciones entre las velocidades de las ruedas.

$$\begin{aligned} v_{1x} &= v_{3x} = v_L \\ v_{2x} &= v_{4x} = v_R \\ v_{1y} &= v_{2y} = v_F \\ v_{3y} &= v_{4y} = v_B \end{aligned} \tag{A.22}$$

Donde las velocidades se denotan como velocidad longitudinal izquierda v_L y velocidad longitudinal derecha v_R y velocidad lateral delantera y trasera v_F y v_B. Escribiendo completamente las ecuaciones siguientes se obtiene:

$$\begin{aligned} v_L &= -d_{1y} \cdot \omega = (y_{IRC} - d) \cdot \omega = v_x - d \cdot \omega \\ v_R &= -d_{2y} \cdot \omega = (y_{IRC} + d) \cdot \omega = v_x + d \cdot \omega \\ v_F &= d_{1x} \cdot \omega = (-x_{IRC} + a) \cdot \omega = v_y + a \cdot \omega \\ v_B &= d_{3x} \cdot \omega = (-x_{IRC} - b) \cdot \omega = v_y - b \cdot \omega \end{aligned}$$

Escritas de otra forma, las ecuaciones se transforman en:

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & -d \\ 1 & 0 & d \\ 0 & 1 & a \\ 0 & 1 & -b \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{A.23}$$

De esta manera se usan las velocidades del centro de masas para calcular las velocidades de cada rueda.

A.3. Motor

El modelo del motor es una descripción de como la aplicación de un voltaje al motor resulta en una velocidad angular del rotor del motor. Cuando se compra un robot móvil con ruedas a un fabricante, dicho robot está usualmente equipado con un controlador de velocidad. Esto se puede verificar mirando el manual del robot móvil con ruedas, en el que se menciona que es posible controlar la velocidad de las ruedas de cada lado [Robotics, 2002, p. 36] proporcionando valores a un PID instalado en el controlador. Para construir un lazo de control alrededor del motor, se deriva un modelo, y se diseña un controlador.

Un motor de Corriente Continua (CC) es normalmente modelado utilizando las ecuaciones A.24 y A.25, siendo la primera de las ecuaciones la parte eléctrica, y la segunda la parte mecánica.

$$u_a = \frac{d}{dt} i_a + R_a i_a + K_e \omega_m \tag{A.24}$$

$$J \frac{d\omega_m}{dt} = \tau_m - \tau_l - \tau_f \tag{A.25}$$

Donde u_a es el voltaje del motor, i_a la corrinete del motor, R_a la resistencia del motor, L_a la inductancia del motor, K_e la constante de la fuerza contraelectromotriz y ω_m la velocidad angular del rotor del motor. J es el momento de inercia, τ_m el torque del motor, τ_l el torque de la carga y τ_f el torque de fricción.

La relación entre la corriente y el torque del motor se asume lineal, como se muestra en la ecuación A.3.

$$\tau_m = K_m \cdot i_a$$

donde k_m es la constante de torque del motor. También se asume que los engranajes proporcionan una relación lineal, de manera que la dinámica del engranaje queda excluida. Por ello, el torque de la rueda se calcula como,

$$\tau = n \cdot K_m \cdot i_a$$

donde n es el ratio del engranaje, y la velocidad angular de la rueda es

$$\omega = \frac{1}{n} \cdot \omega_m$$

Como los motores están físicamente acoplados 2 a 2, el modelo del motor se cambia de acuerdo con la figura A.8.

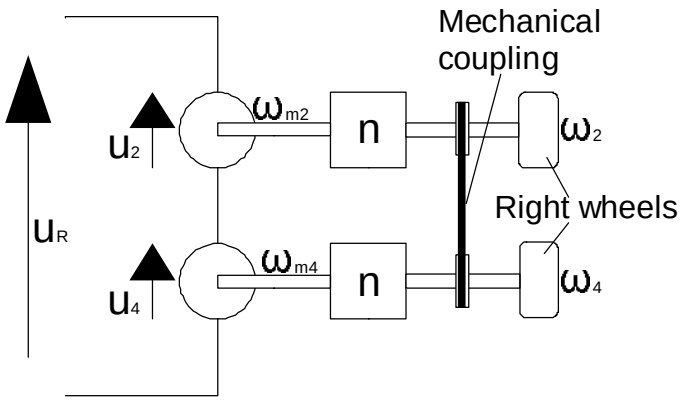

Figura A.8: Los motores de cada lado del vehículo están acoplados, de forma que la velocidad angular de las ruedas de cada lado es la misma. La figura muestra los dos motores/ruedas de cada lado del robot Pioneer 2AT-8.

Las ecuaciones previas se modifican para recoger el acoplamiento:

$$\begin{aligned} u_a &= 2L_a\frac{d}{dt}i_a + 2R_a i_a + 2K_e\omega_m \\ 2J\frac{d\omega_m}{dt} &= \tau_m - \tau_l - 2\tau_f \end{aligned}$$

Los motores del Pioneer 2AT-8 no están accesibles para pruebas libres, y por ello no ha sido posible determinar los parámetros de los motores experimentalmente. Los cuatro motores instalados son todos del tipo Pittman GM9236E204, producidos por PennEngineering Motion Technologies. La hoja de características de los motores incluye los parámetros [Technologies, 2006], y estos parámetros han sido utilizados para la simulación y el diseño del controlador.

De acuerdo con la hoja de características los parámetros son:

$$\begin{aligned} R_a &= 0{,}71\Omega \\ L_a &= 0{,}66mH \\ K_m &= 23\cdot 10^{-3}\frac{N\cdot m}{A} \\ K_e &= 23\cdot 10^{-3}\frac{V}{rad/s} \\ J &= 7{,}06\cdot 10^{-6}kg\cdot m^2 \\ b &= 3{,}54\cdot 10^{-6}N\cdot m \end{aligned}$$

Todos los parámetros presentados anteriormente se encuentran en la hoja de características, exceptuando el torque de la carga, que ha tenido que ser

realimentado de la dinámica. Como es habitual, dos constantes características son iguales, por lo que K_e se puede simplemente reemplazar por K_m en las fórmulas.

El lazo de control es parte del vehículo, y por ello tiene que ser incluido. Se modela como un controlador integral, que tiene la función de transferencia mostrada en la ecuación A.26.

$$\frac{u_a}{\omega_{error}} = \frac{K_I}{s} \tag{A.26}$$

Se ha establecido un modelo del lazo de control interno, y otro del sistema del motor completo. La próxima parte es la compilación de las partes del modelo en un modelo completo.

A.4. Modelo completo

Como se ha descrito anteriormente, el modelo completo es un modelo del vehículo, con las entradas y las salidas disponibles. Las diferentes partes han sido analizadas en las secciones previas y, con el fin de facilitar la visión general del mismo al lector, el modelo está recompilado aquí.

Los estados son:

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \end{bmatrix} = \begin{bmatrix} \omega_L \\ i_L \\ u_L \\ \omega_R \\ i_R \\ u_R \\ \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix} \tag{A.27}$$

Las 9 ecuaciones A.28-A.36 forman un modelo no lineal del vehículo completo - motores, controladores de los motores y dinámica.

$$\dot{\omega}_L = \frac{-B}{J}\omega_L + \frac{K}{2J}i_L - \frac{mg \cdot r}{4n \cdot J}\left(\sin\Theta + f_r sign(v_L)\cos\Theta + \frac{\mu_{xc}}{2}sign(v_L)\cos\Theta\right) \tag{A.28}$$

$$\dot{i}_L = \frac{-K}{L}\omega_L - \frac{R}{L}i_L + \frac{1}{2L}u_L \tag{A.29}$$

$$\dot{u}_L = -K_I\omega_L + K_I\omega_{ref,L} \tag{A.30}$$

$$\dot{\omega}_R = \frac{-B}{J}\omega_R + \frac{K}{2J}i_R - \frac{mg \cdot r}{4n \cdot J}\left(\sin\Theta + f_r sign(v_R)\cos\Theta + \frac{\mu_{xc}}{2}sign(v_R)\cos\Theta\right) \tag{A.31}$$

$$\dot{i}_R = \frac{-K}{L}\omega_R - \frac{R}{L}i_R + \frac{1}{2L}u_R \tag{A.32}$$

$$\dot{u}_R = -K_I\omega_R + K_I\omega_{ref,R} \tag{A.33}$$

$$\ddot{\theta} = \frac{n}{r}\frac{2d}{I}\left(K \cdot i_R - K \cdot i_L\right) - M_r \tag{A.34}$$

$$\ddot{x} = \frac{n}{r \cdot m}\left(K \cdot i_R + K \cdot i_L\right) - \frac{R_x + R_r + R_g}{m} \tag{A.35}$$

$$\ddot{y} = -\frac{Fy}{m} \tag{A.36}$$

El último término de las dos velocidades de los motores es la fricción de la dinámica siendo transferida al rotor del motor. El torque de la dinámica se

calcula usando la corriente de los motores, y después transfiriendo el torque del motor resultante a las ruedas.

De esta manera se ha desarrollado un modelo completo del vehículo. La próxima parte es mirar las incertidumbres presentes en este modelo.

Apéndice B

Modelo Webots

Este capítulo recopila información sobre el simulador Webots de Cyberbotics Ltd. y el modelo/controlador para simular el robot móvil Pioneer 2AT-8

B.1. Modelo y controlador Pioneer 2AT-8

B.1.1. El modelo

La representación gráfica del Pioneer 2AT-8 ha sido realizada a partir de la representación del Pioneer 2-DX incorporada en los ejemplos de simulación de Webots.

B.1.2. El controlador

El controlador utilizado en el primer modelo realizado del Pioneer 2AT-8 actúa controlando la velocidad de giro de las ruedas, mediante las funciones correpondientes del nodo `DifferentialWheels`.

En Linux, todos los controladores de Webots linkan el binario resultante de compilar el controlador con la libreria `libController.so`. Dicha librería se encuentra en el directorio `$(WEBOTS_HOME)/lib`.

En el capítulo 4 de la Guía del Usuario de Webots [web, 2008] se describe como implementar un interfaz a este controlador utilizando el protocolo TCP/IP. De una manera similar, es posible linkar nuestro controlador con las librerías CORBA de TAO, y crear un servant que permite comunicar al controlador con otros objetos CORBA.

El binario resultante realiza llamadas a funciones de la librería de hilos `libpthread.so` lanzando un segundo hilo, de manera que mientras un hilo ejecuta la llamada `orb->run()`, llamada que bloquea y permite al ORB

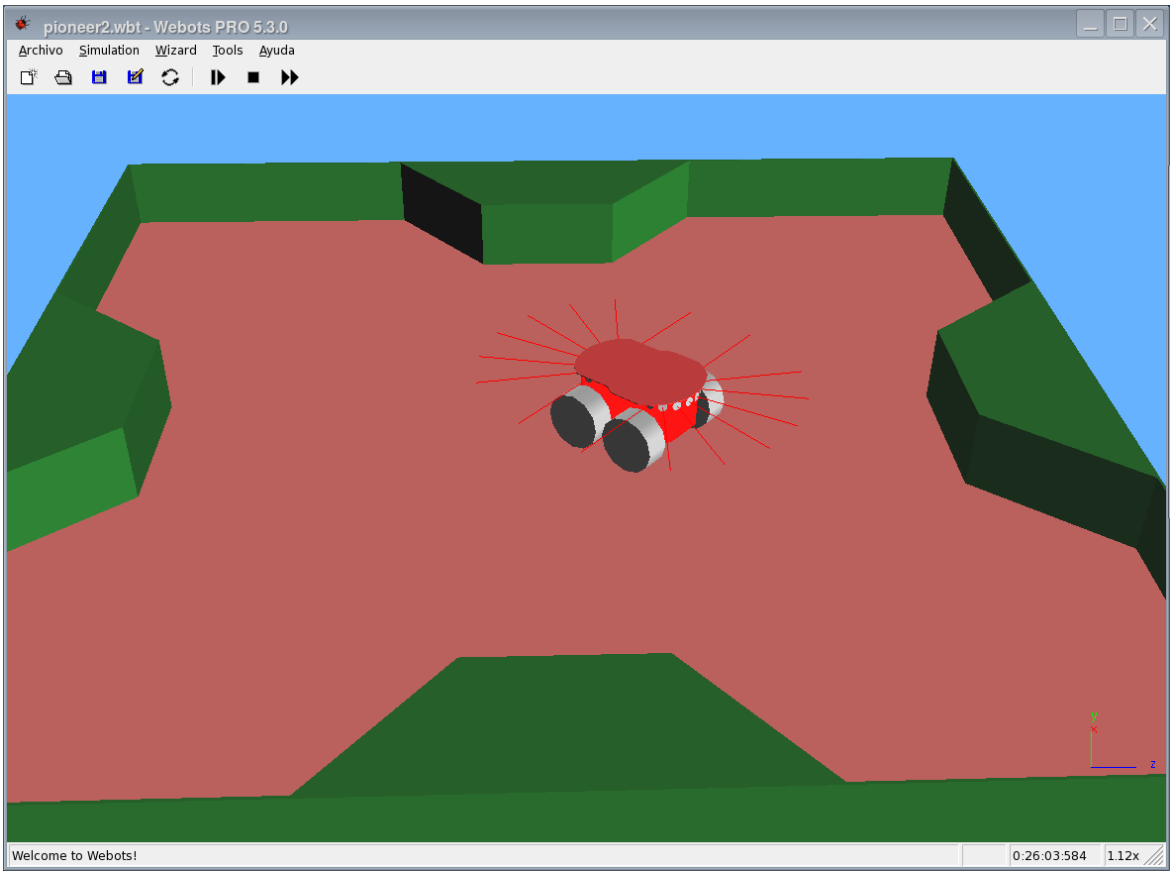

Figura B.1: Webots - Simulación del vehículo Pioneer 2-AT8

Figura B.2: Webots - Logs del controlador del Pioneer 2-AT8

aceptar peticiones, el otro hilo ejecuta la llamada `robot_run(run)`, llamada que también bloquea e inicia el bucle de control del controlador de Webots.

Los lazos de control siguen el esquema mostrado en la figura A.2. El controlador implementa un PID para el control de la coordenada angular de la posición del robot, con constante $K_I = 0$. Es por tanto un control proporcional y derivativo. Este PID ha sido implementado utilizando las enseñanzas del curso "Third ARTIST2 Graduate Course on Embedded Control Systems", con varias sesiones, tanto teóricas como prácticas sobre la programación de PIDs.

B.2. El cliente

El cliente remoto que utilizamos para controlar la simulación es el mismo que utilizamos para controlar el robot real.

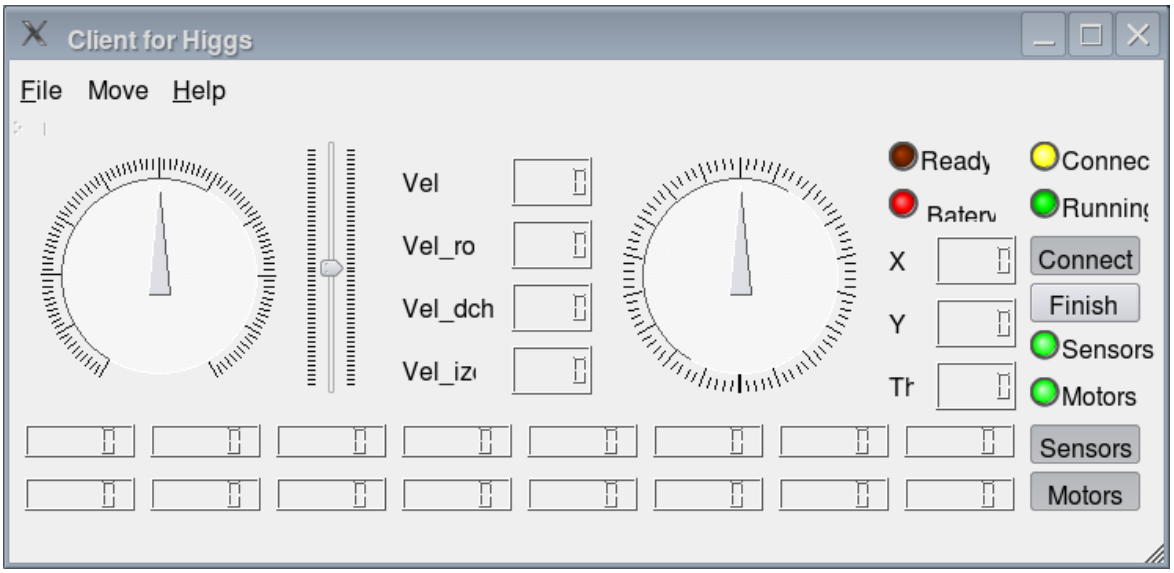

Figura B.3: Webots - Cliente remoto para controlar la simulación del Pioneer 2-AT8

Apéndice C

Interfaces IDL

C.1. Interfaz Pioneer2AT.idl

```
// Pioneer 2AT Interface
// Rafael Chinchilla Camara
// Adolfo Hernando Marcos
// ASLab 2005-2008

module Pioneer {

////////////////////////////////////////////// DATA TYPES

typedef    sequence<long>        SonarRangeSeq;
typedef    sequence<boolean>     SonarNewReadingSeq;

////////////////////////////////////////////// DATA STRUCTURES

struct TimeStamp_Struct{
    long   secs;
    long   usecs;
};

struct Pioneer2AT_State {
    boolean     motorsEnabled;          // True if motors are enabled
    boolean     sonarsEnabled;          // True if sonars are enabled
    boolean     emergencyStop;          // True if ES button pushed
    boolean     connected;              // True if connected (TCP,Serial)
    boolean     running;                // True if executing
    boolean     moving;                 // True if still moving
    boolean     leftMotorStalled;       // True if motors stalled
```

```
    boolean     rightMotorStalled;
    boolean     leftBreakTriggered;     // True if breaks triggered
    boolean     rightBreakTriggered;
    float       battery;                // Battery voltage
    boolean     newReadings;            // True if there are new sonar readings
    TimeStamp_Struct   timeStamp;
};

struct Pioneer2AT_Movement {
    float      speed;                  // Translation speed
    float      leftSpeed;              // Translation speed, left wheel
    float      rightSpeed;             // Translation speed, right wheel
    float      rotSpeed;               // Rotation speed
    float      maxSpeed;               // Maximum translation speed
    float      maxRotSpeed;            // Maximum rotation speed
    float      absMaxSpeed;            // Absolute maximum translation speed
    float      absMaxRotSpeed;         // Absolute maximum rotation speed
    float      accel;                  // Translation acceleration
    float      rotAccel;               // Rotation acceleration
    boolean    headingDone;            // True if rotation movement complete
    boolean    movementDone;           // True if translation movement complete
    TimeStamp_Struct   timeStamp;
};

struct Pioneer2AT_Position {
    float     x;                       // X absolute coordinate
    float     y;                       // Y absolute coordinate
    float     t;                       // Theta absolute coordinate
    float     heading;                 // Heading (degrees)
    float     distanceDiff;            // Distance remaining to complete movement
    float     headingDiff;             // Angle remaining to complete rotation
    TimeStamp_Struct   timeStamp;
};

struct Pioneer2AT_Sensing {
    boolean             hasSonars;      // True if sonars available
    boolean             hasBumpers;     // True if bumpers available
    unsigned short      numSonars;      // Number of sonars available
    unsigned short      numFBumpers;    // Number of front bumpers available
    unsigned short      numRBumpers;    // Number of rear bumpers available
    unsigned short      closestSonar;   // Number of closest sonar
    unsigned long       closestRange;   // Closest sonar reading
    SonarRangeSeq       SonarRanges;    // List of sonar readings
```

```
    SonarNewReadingSeq  SonarNews;      // Bools, true if sonar has new reading
    TimeStamp_Struct    timeStamp;
};


////////////////////////////////////////////// INTERFACE

interface Pioneer2AT{

    // Data exchanging methods ------------------------------------------
    Pioneer2AT_State    getRobotState    ();
    Pioneer2AT_Movement getRobotMovement ();
    Pioneer2AT_Position getRobotPosition ();

    void getRobotSensing  ( out Pioneer2AT_Sensing sensing );

    // Robot fast state checking ----------------------------------------
    boolean isReady     ();         // True if software loaded correctly
    boolean isConnected ();         // True if connection established+Ready
    boolean isRunning   ();         // True if ARrobot running+Connected
    boolean isStalled   ();         // True if some motor stalled
    boolean isEmergency ();         // True if emergency situation
    boolean isMoving    ();         // True if completing a movement
    boolean isEnabled   ();         // True if motors enabled+Running
    boolean isBreaked   ();         // True if any break triggered
    float  getBattery  ();         // Battery voltage
    TimeStamp_Struct getTime();
    void setTimeToNow();

    // CORBA object remote control --------------------------------------
    void            init            ();
    long            connect         ();
    void            start           ();
    void            stop            ();
    void            disconnect      ();
    void            finish          ();
    void            setCycleTime    ( in unsigned long ms );
    unsigned long   getCycleTime    ();
    unsigned long   getRefreshTime  ();
    void            setRefreshTime  ( in unsigned long ms );

    // Movement information ---------------------------------------------
    float  getVelocity        ();
```

```
float  getMaxVelocity      ();
float  getAbsMaxVelocity   ();
float  getLeftVelocity     ();
float  getRightVelocity    ();
float  getAcceleration     ();
float  getDeceleration     ();
float  getRotVelocity      ();
float  getMaxRotVelocity   ();
float  getAbsMaxRotVelocity();
float  getRotAcceleration  ();
boolean isHeadingDone      ( in float delta );
boolean isMovementDone     ();
float  getHeading          ();
float  getHeadingDiff      ();
float  getDistanceDiff     ();

// Movement control ------------------------------------------------
void    enableMotors        (                                      );
void    disableMotors       (                                      );
boolean areMotorsEnabled    (                                      );
void    setVelocity         ( in float vel                         );
boolean setMaxVelocity      ( in float maxVel                      );
boolean setAbsMaxVelocity   ( in float maxVel                      );
void    setLRVelocity       ( in float leftVel, in float rightVel  );
void    setAcceleration     ( in float accel                       );
void    setDeceleration     ( in float decel                       );
void    setRotVelocity      ( in float rotVel                      );
boolean setMaxRotVelocity   ( in float maxRotVel                   );
boolean setAbsMaxRotVelocity( in float maxRotVel                   );
boolean setAbsMaxRotAccel   ( in float maxRotAccel                 );
boolean setAbsMaxRotDecel   ( in float maxRotDecel                 );
void    setRotAcceleration  ( in float rotAccel                    );
void    moveDistance        ( in float distance                    );
void    setHeading          ( in float heading                     );
void    setDeltaHeading     ( in float delta                       );

// Position information -------------------------------------------
float getX             ();
float getY             ();
float getTh            ();
float getCompass       ();
void  resetPosition    ();
```

```
    // Sensing ------------------------------------------------------------
    boolean         hasSonars           ();
    boolean         areSonarsEnabled    ();
    void            enableSonars        ();
    void            disableSonars       ();
    boolean         hasFrontBumpers     ();
    boolean         hasRearBumpers      ();
    unsigned short  getNumSonars        ();
    unsigned short  getNumFrontBumpers  ();
    unsigned short  getNumRearBumpers   ();
    unsigned short  getClosestSonar     ();
    unsigned long   getClosestSonarRange();
    unsigned long   getSonarRange       ( in unsigned short numSonar );
    boolean         hasSonarNewReadings ( in unsigned short numSonar );

};//INTERFACE

};//MODULE
```

C.2. Interfaz Trajectory.idl

```
// Trajectory Interface
// Adolfo Hernando Marcos
// ASLab 2007

module Pioneer {

////////////////////////////////////////////// DATA STRUCTURES

//struct TimeStamp_Struct{
//    long   secs;
//    long   usecs;
//};

////////////////////////////////////////////// INTERFACE

interface Trajectory{

    // Trajectory control -------------------------------------------------
    boolean     writeTrajectory      ( in unsigned long trajectory );
```

```
    boolean     readTrajectory       ( in unsigned long trajectory );
    boolean     closeTrajectory      ();
    boolean     setTrayectoryComment ( in string trajectory_comment );
    boolean     getTrayectoryComment ( out string trajectory_comment );
    boolean     pauseTrajectory      ( in boolean pauseStatus );

};//INTERFACE

};//MODULE
```

C.3. Interfaz Log.idl

```
// Pioneer 2AT Interface
// Adolfo Hernando Marcos
// ASLab 2006

module Log {

////////////////////////////////////////////// DATA TYPES

typedef    string               DataType;
typedef    string               NameType;
enum Status2AT {PIONEER_STATE_OFF, PIONEER_STATE_LOADED, PIONEER_STATE_READY,
                  PIONEER_STATE_CONNECTED, PIONEER_STATE_RUNNING};

////////////////////////////////////////////// DATA STRUCTURES

struct Log2AT_Data {
    NameType sysname;
    DataType logdata;
};

////////////////////////////////////////////// INTERFACE

interface Log2AT{

    // Data exchanging methods ------------------------------------------
    void sendLog ( in Log2AT_Data sdata );
    void stateLog ( in Status2AT estatus );
    void sensorsLog ( in boolean bsensors );
```

```
    void motorsLog ( in boolean bmotors );

};//INTERFACE

};//MODULE
```

Bibliografía

[web, 2008] (2008). *Webots User Guide - release 5.9.2*. Cyberbotics Ltd.

[Åström and Hägglund, 2006] Åström, K. J. and Hägglund, T. (2006). *Advanced PID Control*. Instrumentation, Systems, and Automation Society (ISA).

[Albus et al., 2002] Albus, J., Huang, H.-M., Messina, E., Murphy, K., Juberts, M., Lacaze, A., Balakirsky, S., Shneier, M., Hong, T., Scott, H., Proctor, F., Shackleford, W., Michaloski, J., Wavering, A., Kramer, T., Dagalakis, N., Rippey, W., Stouffer, K., Legowik, S., Evans, J., Bostelman, R., Norcross, R., Jacoff, A., Szabo, S., Falco, J., Bunch, R., Gilsinn, J., Chang, T., and Tsai, T.-M. (2002). 4d/rcs: A reference model architecture for unmanned vehicle systems. version 2.0. Technical report, National Institute of Standards and Technology (NIST).

[Bongard et al., 2006a] Bongard, J., Zykov, V., and Lipson, H. (2006a). Automated synthesis of body schema using multiple sensor modalities. In *Proceedings of the 10th Int. Conference on Artificial Life (ALIFE X)*, pages 220–226.

[Bongard et al., 2006b] Bongard, J., Zykov, V., and Lipson, H. (2006b). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

[Burns and Wellings, 2001] Burns, A. and Wellings, A. (2001). *Real-Time Systems and Programming Languages*. Addison Wesley.

[Gazi et al., 1998] Gazi, V., Moore, M., and Passino, K. M. (1998). Real-time control system software for intelligent system development: Experiments and an educational program. In *1998 IEEE ISIC/CIRA/ISAS Joint Conference*, pages 102–107, Gaithersburg, MD.

[Henning and Vinoski, 1999] Henning, M. and Vinoski, S. (1999). *Advanced CORBA Programming with C++.* Addison-Wesley Professional Computing Series. Addison-Wesley.

[Hernando, 2005] Hernando, A. (2005). Versión rt-corba del servidor pioneer 2at-8.

[Jalote, 1994] Jalote, P. (1994). *Fault Tolerance in Distributed Systems.* PTR Prentice Hall. Prentice Hall.

[Klir, 2001] Klir, G. J. (2001). *Facets of Systems Science.* IFSR International Series on Systems Science and Engineering. Kluwer Academic/Plenum Publishers.

[Luka Caracciolo, 1999] Luka Caracciolo, Alessandro De Luca, S. I. (1999). *Trajectory Tracking Control of a 4-Wheel Differentially Driven Mobile Robot.* Dipartimento di Informatica e Sistemistica, Universit. degli Studi di Roma.

[Monroy, 2007] Monroy, C. G. Q. (2007). *Introspection on Control-grounded Capabilities. An Agent-inspired Approach for Control.* PhD thesis, University of Girona.

[Moore et al., 1999] Moore, M. L., Gazi, V., Passino, K. M., Shackleford, W. P., and Proctor, F. (1999). Complex control system design and implementation using the nist-rcs software library. *IEEE Control Systems*, 19(6):12–27.

[Morris, 2007] Morris, A. C. (2007). *Robotic introspection for Exploration and Mapping of Subterranean Environments.* PhD thesis, Carnegie Mellon University.

[Object Computing, 2005] Object Computing, I. (2005). *TAO Developer's Guide. OCI TAO Version 1.4a. Building a standard in performance.* Object Computing Inc.

[OMG, 2002] OMG (2002). *Persistent State Service, Version 2.0.* Object Management Group.

[Press et al., 1992] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1992). *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press.

[Robotics, 2002] Robotics, A. (2002). *ActivMedia Robotics Pioneer 2/PeopleBot Operations Manual v11.* Technical report, ActivMedia Robotics.

[Rumelhart and McClelland, 1987] Rumelhart, D. E. and McClelland, J. L. (1987). *Parallel Distributed Processing - Vol. 1 Foundations*, volume 1. MIT Press.

[Stevens and Rago, 2005] Stevens, W. R. and Rago, S. A. (2005). *Advanced Programming in the UNIX Environment. Second Edition.* Addison-Wesley Professional Computing Series. Addison-Wesley.

[Technologies, 2006] Technologies, P. M. (2006). *Pittman, LO-COG DC Gearmotors.* Technical report, PennEngineering Motion Technologies.