Master Universitario en Sistemas Telemáticos e Informáticos

Curso académico 2009-2010

Trabajo Fin de Master

Docencia de robótica con *jderobot* 5

Autor: Javier Vázquez Pereda

Tutor: José María Cañas Plaza

Labor omnia improba vincit

Agradecimientos

Me gustaría expresar mi agradecimiento a las personas que me han apoyado durante la realización de este trabajo fin de master. Especialmente referirme a mi mujer Vanessa quien me ha animado durante toda su duración y también ha realizado algún sacrificio para llegar al momento de esta entrega.

Agradecer al *grupo de robótica del GSYC* el soporte técnico prestado durante mi aprendizaje en el uso de la plataforma *jderobot*.

Finalmente dar mi agradecimiento a José María Cañas por su dedicación y ayuda ofrecida durante todo el desarrollo del trabajo.

Resumen

La evolución de la robótica en los últimos años se ha extendido más allá del terrero industrial donde tuvo sus inicios. Numerosas aplicaciones conforman lo que algunos autores denominan *robótica de servicios*. Desde el ámbito personal y doméstico al profesional existen en la actualidad gran cantidad de productos y aplicaciones robóticas en continua evolución. Dicho progreso viene dado por una ferviente actividad de investigación y desarrollo en este dominio. Sirva como ejemplo el programa *DARPA Urban Challenge*, donde universidades, centros tecnológicos y empresas compiten por la creación de un vehículo con capacidad de navegación autónoma casi perfecta.

En un sector profesional con un panorama de crecimiento tan optimista como en de la robótica, la formación técnica, a distintos niveles, es un aspecto clave para la consecución de los objetivos fundamentales de la investigación, el desarrollo y la innovación. En la URJC se apuesta por trabajar en esa línea y fomentar el desarrollo de la robótica.

El TFM toma como base la nueva plataforma de desarrollo de robots de la URJC, *jderobot* 5.0 y aborda la creación de un entorno de enseñanza para el aprendizaje práctico del alumno. Se implementarán diversos componentes que conecten al entorno con el software de simulación *gazebo*, el cual ofrecerá al alumno la flexibilidad de aprender a usar diferentes tipos de sensores y actuadores frecuentes en los robots, así como la libertad de trabajar con varios robots al mismo tiempo y sobre distintos entornos de trabajo. También se desarrollarán herramientas que faciliten la utilización del entorno de enseñanza, basadas en interfaces gráficas para la visualización de señales y control de la ejecución de los programas lanzados. El diseño realizado tendrá en cuenta la necesidad de abstraer al alumno de los detalles técnicos de la infraestructura *jderobot.*

Se tendrá en cuenta el modelo de arquitectura software planteado para la nueva versión, respetándose las diversas reglas de diseño y codificación durante el desarrollo de los componentes, tales como la posibilidad de ejecutar los componentes software una red de cómputo distribuida mediante el uso del *middleware Ice* de *ZeroC*.

En una última fase del TFM, se realiza la codificación de varios algoritmos de navegación sobre la nueva plataforma con el fin de validar los requisitos solicitados para el entorno de enseñanza. Estos problemas son los realizados en la parte práctica de la asignatura Robótica Master, los cuales se centran en técnicas de navegación clásica con robots.

Índice general

Índice de figuras

Índice de listados

Capítulo 1
Introducción

1.1 Robótica

La palabra robot proviene del término checo *robota*, cuyo significado es “servidumbre, trabajo forzado o esclavitud". Principalmente usado para referirse a los siervos Checoslovacos de la época feudal, que eran obligados a trabajar dos o tres días a la semana en las tierras de los nobles sin ningún tipo de retribución. Según la Real Academia Española, el término robot se refiere a:

Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas

No es hasta 1920 cuando la palabra *robot* fue empleada por primera vez, en una obra de teatro llamada *"R.U.R."* o *"Los Robots Universales de Rossum"* escrita por el dramaturgo checo Karel Capek. La trama era sencilla: el hombre construye un robot, y al final el robot mata al hombre. El término continúa usándose en varios escritos posteriores pertenecientes al género literario de la ciencia-ficción, pero no fue hasta años más tarde cuando, Isaac Asimov aportó con varias narraciones relativas a robots. A él se atribuye el acuñamiento del término *robótica*.

La robótica como hoy la conocemos, surge hacia la década de los 50, con la construcción de las primeras computadoras y en donde los robots eran entendidos como manipuladores reprogramables y multifuncionales diseñados para trasladar materias primas, productos, etc, siguiendo una serie de movimientos programados y repetitivos. Posteriormente, con la llegada de la década de los 70 y los primeros microprocesadores el desarrollo de los robots experimenta un vertiginoso avance. En 1954 se introdujo el primer robot *"Unimate'* (Fig 1.1.a). Fue creado por George Devol y utilizó los principios de control numérico para el control del manipulador, que era un robot con cilindros hidráulicos. Posteriormente, en 1978 se introdujo el robot *PUMA (Programmable Universal Machine for Assembly)* (Fig 1.1.b) para tareas de montaje, soldadura, etc, mediante la el uso de brazos hidráulicos.

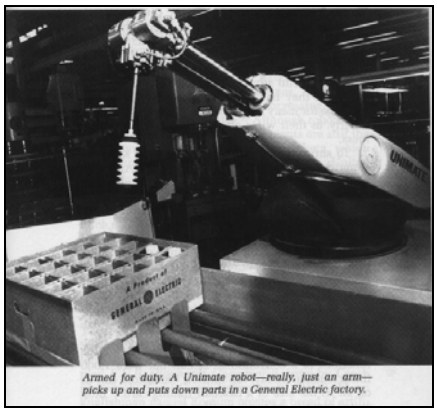


***Figura 1.1: (a)** Unimate. Primer brazo robótico industrial*
***(b)** Unimate PUMA*

Es en la robótica de los sectores industriales, principalmente en la automoción, donde se produce un mayor progreso y se logra en buena medida complementar o sustituir las funciones de los humanos, alcanzando en algunos casos autonomías casi completas. Esta carrera evolutiva queda claramente justificada por los beneficios en productividad, flexibilidad, calidad y seguridad alcanzados. Las mejoras en el aumento de la producción y reducción de costes, sobre todo laborales, energéticos y de almacenamiento, se complementan con los factores de homogeneidad en la ejecución de las tareas, garantizando un nivel de calidad uniforme, así como la reducción de riesgos laborales por el reemplazo del humano por el robot en diversos entornos hostiles (centrales nucleares, océanos, espacio, etc.).

De acuerdo al informe UNECE/IFR (2005), el parque mundial de robots instalados es próximo a los 850.000, con tasas de crecimiento sostenido entre el 5% y el 7%. Aproximadamente el 50% se sitúan en el segmento de la automoción, principalmente automatizando tareas de soldadura y manipulación (Fig. 1.2.a).

***Figura 1.2: (a)** Línea de montaje en automoción*
***(b)** Sistemas de almacenaje KIVA*

No obstante, existen otros sectores empresariales como el alimentario y o el logístico donde también han proliferado la implantación de aplicaciones robóticas. Por ejemplo, las actividades de almacenamiento y clasificación de productos a gran escala pueden ser realizados de una forma organizada y planificada con sistemas robóticos como *KIVA* (Fig 1.2.b), los cuales combinan la capacidad transportadora de los robots, provistos con sensores para permitir su localización y la búsqueda de productos, con la programación de una inteligencia capaz de inventariar y planificar los traslados de pedidos y órdenes.

Frente a la robótica con aplicación industrial, la robótica de servicio es un área emergente, pero con un gran potencial de crecimiento. Cuenta con numerosos campos de aplicación, desde el servicio personal (cuidado de personas mayores *[Marugán Alonso, 2009]*, limpieza, vigilancia doméstica) hasta aplicaciones domésticas (medicina y cirugía, construcción, agricultura, etc.). Así como las sociedades de entre finales del siglo XX y comienzos del XXI fueron invadidas por el auge del computador personal, es previsible que la sociedad del siglo XXI experimente una invasión en el uso de robots dentro de los ámbitos doméstico y

profesional. Bill Gates realizó un pronosticó con esta tendencia ante la *Scientific American* en 2006[1].

Las expectativas del sector servicios han superado en algunos casos las previsiones iniciales, llegando incluso a saturar la capacidad de manufactura. En el caso de la limpieza de suelos (robot Roomba de iRobot), se alcanzó la cifra de 600.000 unidades vendidas a finales de 2.005. En los siguientes cinco años, este número ha crecido hasta los 5 millones de robots.

La aplicación de la robótica en la sanidad es otro ejemplo de desarrollo para el sector servicios. En cirugía, aplicaciones, normalmente comerciales, se clasifican en dos vertientes: cirugía robotizada guiada por imagen y uso de telerobots en cirugía mínimamente invasiva. La primera orientación se aplica en particular a la inserción de implantes ortopédicos de cadera y rodilla. El proceso de selección del implante y vaciado del hueso para el alojamiento del implante, puede ser planificado en una etapa preoperatoria y después ser ejecutado con precisión en el quirófano por un robot bajo la supervisión de un equipo médico (Fig 1.3.a).

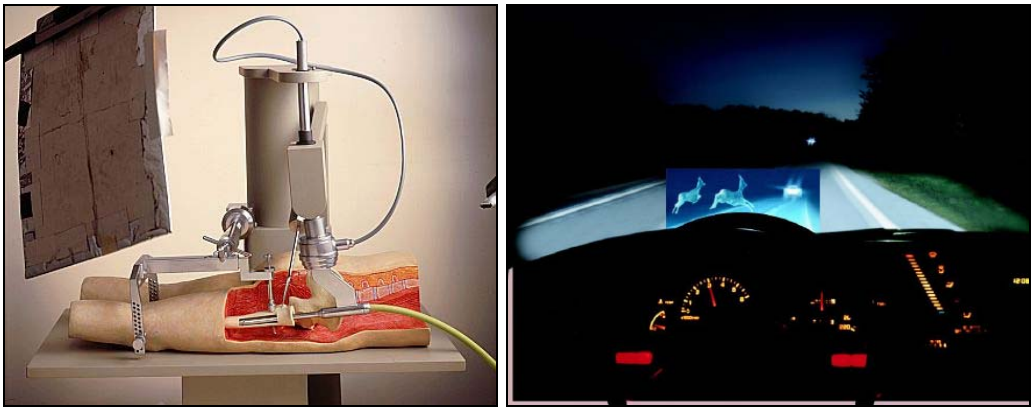


Fig1.3. (a) *Robodoc. Sistema para operaciones quirúrgicas*
(b) *Sistema de asistencia visual para el conductor*

La robótica es una disciplina transversal que necesita de varias áreas de investigación: control, mecánica, electrónica, electricidad e informática. Cada una de estas especialidades se orienta al estudio de alguna de las partes del sistema que representa al robot: sensores, actuadores, computador y programación *[intro, Cañas. 2009]*.

La robótica mecánica estudia la cinemática del robot, el movimiento del robot (articulaciones) respecto a un sistema de referencia. También se contempla cuestiones relativas a la dinámica, cómo se comportarían las fuerzas sobre el robot debido a masas e inercias.

Por su parte, la electrónica abarcaría el estudio en el desarrollo de nuevos motores, servomotores, sistemas de alimentación eléctrica, etc, papel fundamental para el funcionamiento y movimiento del robot. El diseño de nuevos sensores, transductores, cada vez más precisos y rápidos en la obtención y presentación de los datos a los niveles de lógica del robot, son igualmente relevantes para el avance en de los robots.

[1] http://www.sciamdigital.com/index.cfm?fa=Products.ViewIssuePreview&ARTICLEID_CHAR=63449950-2B35-221B-6B873224A4114DEF

Ejemplos son los cada vez más numerosos y sofisticados sistemas incorporados en automóviles. El desarrollo de sistemas inteligentes de procesamiento visual, capaces de reconocer señales de tráfico u obstáculos en la carretera y generar alguna respuesta en forma de aviso al conductor, determinan en parte su eficacia por la precisión, rapidez y seguridad de las imágenes captadas por la cámara. Naturalmente, los algoritmos de procesamiento visual, en el dominio de la robótica informática, también son cruciales para este éxito (Fig 1.3. b).

El campo de la robótica informática estudia la utilización de computadores para la implementación de una lógica inteligente que controle el robot. Las mejoras tecnológicas y en microelectrónica han disparado la capacidad de cómputo disponible y el desarrollo de los más variados sensores, motores y dispositivos. Por ejemplo, sensores precisos como el láser, el GPS, o las cámaras en color, y motores de continua, servos, etc. aparecen frecuentemente como parte del equipamiento de los robots móviles. La velocidad de los computadores se ha multiplicado en los últimos años considerablemente y esto ha hecho factible la materialización de algunas aproximaciones a la robótica móvil que demandan gran capacidad de cálculo. Pero todo este hardware, por potente que sea resulta inútil sin una buena programación que sepa como procesar sus datos y dar las respuestas adecuadas. Por lo que a pesar de estas mejoras tecnológicas, de disponer de los procesadores más rápidos, los sensores y actuadores más avanzados, el cómo combinarlos para generar comportamiento autónomo sigue siendo un problema abierto. Los progresos en el hardware han puesto en evidencia la importancia de una buena organización interna, que se ha convertido en un factor crítico para conseguir el comportamiento autónomo robusto *[jderobot 5 thesis, D. Lobato, 2010]*.

Por tanto, técnicas de ingeniería de software, arquitecturas de referencia (cognitivas), desarrollo de software basado en métodos formales, etc, constituyen cada vez más, herramientas básicas para el desarrollo de aplicaciones reusables, escalables, fácilmente mantenibles y seguras.

1.2 Programación de robots

La autonomía y la inteligencia residen en el programa que controla el robot. Por ejemplo, en los robots móviles el comportamiento principal es su movimiento. Los programas que se ejecutan en el robot determinan la manera en que éste se mueve por el entorno, reaccionando ante obstáculos percibidos por los sensores, acercándose a algún destino, etc. y para ello tienen que enviar continuamente las órdenes adecuadas a los motores *[RIAI, J.M.Cañas, V.Matellán, R. Montúfar, 2006]*.

El modo en que se programan los robots ha ido evolucionando. Históricamente los robots eran desarrollos únicos, no se producían en serie, y los programas de control se construían empleando directamente los drivers para acceder a los dispositivos sensoriales y de actuación. El sistema operativo del robot era mínimo, básicamente una colección de drivers con rutinas para leer datos de los sensores y enviar consignas a los actuadores. En este contexto, el programa de aplicación invocaba directamente las funciones de la librería que ofrecía el fabricante en sus drivers. La aplicación se situaba directamente encima del sistema operativo (Fig 1.4.a).

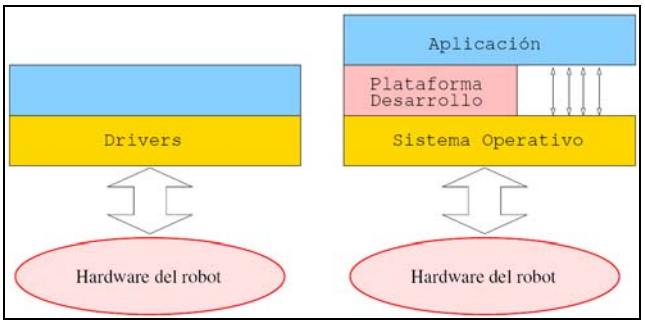


Fig 1.4. *Programación de robots. (a) sobre driver, (b) sobre plataforma de desarrollo*

Con el asentamiento de los fabricantes, el trabajo de muchos grupos de investigación y el paso de los años han ido apareciendo plataformas de desarrollo que simplifican la programación de aplicaciones robóticas. Estas plataformas ofrecen acceso más sencillo a sensores y actuadores, suelen incluir un modelo de programación que establece una determinada organización del software y permite manejar la creciente complejidad del código cuando se incrementa la funcionalidad del robot. El diseñador programa sus aplicaciones robóticas finales sobre esa plataforma de desarrollo (Fig. 1.4,b).

En cuanto a los sistemas operativos sobre los que se soportan las aplicaciones robóticas, hoy en día pueden clasificarse como dedicados o generalistas. Numerosos fabricantes de robots, de ámbito industrial o personal, desarrollan sus aplicaciones sobre un sistema operativo propietario y adaptándolo a las características hardware particulares del robot. Es el caso del sistema operativo ROBIOS para el robot EyeBot (Fig 1.5), compilado para la CPU Motorola 68332 con el que cuenta el modelo de robot y al cual están conectados los sensores y actuadores. Entre los sistemas operativos de propósito general cabe destacar GNU/Linux muy extendido en los entornos de investigación, por su condición de software abierto y adaptable a las necesidades de desarrollo de drivers y plataformas.

Fig 1.5: *Robot EyeBot con sistema operativo ROBIOS*

Como se ha dejado ver en la sección de inicio, durante las últimas décadas la robótica ha tenido una gran presencia en la industria, especialmente en el área de la automoción, lográndose avances significativos. No obstante, siguen existiendo retos tecnológicos como el desarrollo de nuevos sistemas de agarre, donde la relación fuerza-peso sea 1:1, en lugar de la tasas actuales 1:10, el funcionamiento de líneas de producción con balanceo de carga dinámico inteligente, donde la reducción de productividad en ciertos puntos de la cadena, por averías o mal dimensionamiento

industrial, puedan aliviarse con la derivación dinámica del trabajo a otras estaciones paralelas, el desarrollo de sistemas multirobot cooperantes, la mejora en la precisión de sensores y rapidez de actuadores, etc. En varios de ellos, la robótica informática tiene un papel fundamental, dado que se plantan cuestiones que requieren un control inteligente del funcionamiento un robot o la cooperación sincronizada de una planta de robots con un objetivo común.

En las aplicaciones de la robótica de servicios, es de esperar una gran expansión durante los próximos años, la cual planteará numerosos retos tecnológicos. Los problemas de navegación siguen siendo una de las cuestiones aun no resueltas en la programación de robots de exteriores o móviles. Especialmente, factores de fiabilidad y de respuesta rápida del robot antes eventos no esperados. El desarrollo de métodos cognitivos y técnicas de procesamiento para operar en entornos no estructurados y complejos es otro reto que afecta a este tipo de robots. Las arquitecturas cognitivas, que como define *[jderobot 5 thesis, D. Lobato, 2010]*, son aquellas que organizan sus capacidades sensoriales, de procesamiento y de acción para conseguir un repertorio de comportamientos inteligentes interactuando con un cierto entorno, son un enfoque para dar solución a estos problemas. Para comportamientos simples casi cualquier organización resulta válida, pero para comportamientos complejos se hace patente la necesidad de una buena organización cognitiva. El desarrollo de sistemas de percepción del entorno y del usuario, también pueden ser aplicados a la robótica asistencial. Según el Libro Blanco de la Robótica *[Libro Blanco, CEA-GTRob, 2007]*, el envejecimiento de la población hará que los robots asistentes jueguen un papel importante en los próximos años. La población en un futuro seguramente tendrá más salud que la de ahora, pero precisará de mayor asistencia debido al incremento de la esperanza de vida. Un ejemplo de este tipo de aplicaciones es el sistema *ElderCare* [1] desarrollado por el grupo de robótica de la Universidad Rey Juan Carlos, capaz de monitorizar una habitación y alertar si se detecta que una persona se ha caído y permanece inmóvil en el suelo.

Normalmente la creación de productos *robóticos* complejos atraviesa prolongadas fases de investigación y prototipado donde las tecnologías y técnicas aplicadas van madurando con el tiempo. En algunas de estas ocasiones, el reto de lograr robots completamente autónomos es tan alto que se acuden a modelos de cooperación entre diferentes comunidades investigadoras y en otros casos, la velocidad de avance está motivada en el uso de los prototipos en competición. El Programa *Urban Chanllege* (Fig. 1.6.a) es un caso de ellos. Múltiples univesidades, centros tecnológicos y empresas participan en una competición reglada en la que se trata de desarrollar el automóvil autónomo perfecto, capaz de realizar desplazamientos desde un origen y a un destino sirviéndose de técnicas de navegación global con mapas y navegación local para la evitación de obstáculos e imprevistos por el camino. Otro caso de competición, con el propósito de aumentar la destreza de los robots en sus movimientos y mejorar de sus sistemas cognitivos es la *Robocup* (Fig. 1.6.b). Este proyecto agrupa periódicamente a varias universidades para la celebración de campeonatos de futbol con el uso de diferentes clases de robots. La implementación de inteligencias efectivas para estos entornos significa dar solución a grandes retos perceptivos, de actuación y de coordinación con otros robots del equipo.

***Figura 1.6**: **(a)** Coche competición Urban Challenge*
***(b)** Robot jugador de Robocup*

La difusión de la robótica en el mundo se realiza a través de las redes de investigación y sociedades científicas. Un papel fundamental lo juegan las revistas de ámbito científico donde los investigadores de todo el mundo presentan los resultados de sus desarrollos. En España existen numerosos grupos de investigación robótica trabajando en ámbitos multidisciplinares. Cabe destacar al Comité Español de la Automática (CEA) que, a su vez, lidera la Red Nacional de Robótica, subvencionada por el Ministerio de Educación y Ciencia. Dentro de CEA, existe el grupo Español de Robótica (GTRob) como punto de encuentro al que se han adscrito la mayoría de investigadores españoles.

España ocupa un lugar representativo en el ranking de países con mayor número de robots industriales instalados. Con una cifra aproximada de 25.000 unidades alcanzadas en 2.005, ocupa un 7º puesto, por delante incluso de Reino Unido y cerca de Francia. No obstante, no existen empresas españolas que fabriquen o diseñen robots de esta clase y la actividad de investigación al respecto, se focaliza en universidades, el CSIC y los centros tecnológicos.

1.3 Docencia en robótica

La robótica es un sinónimo de progreso y desarrollo tecnológico. Los países y las empresas que cuentan con una fuerte presencia de robots no solamente consiguen una extraordinaria competitividad y productividad, sino también transmiten una imagen de modernidad [Libro Blanco, CEA-GTRob, 2007]. De acuerdo a la anterior afirmación recogida en el Libro Blanco de la Robótica, se puede entender que la formación técnica, a distintos niveles, es un aspecto clave para la consecución de los objetivos fundamentales de la investigación, el desarrollo y la innovación. Además, es una convicción generalizada que el incremento en formación a los diferentes de enseñanza contribuye al aumento de la disponibilidad de personas formada en el ámbito tecnológico. Disponer de tecnologías de automatización avanzada en procesos de manufactura es un aspecto clave y, para ello, son necesarios técnicos bien formados en automatización de robots industriales. Igualmente la formación jugará

un papel muy importante en el desarrollo de la robótica de servicios, dadas las enormes expectativas en un futuro a corto plazo.

Respecto al panorama de formación universitaria, el análisis de la robótica es complejo. En cada escuela o facultad influyen distintos criterios para la configuración de planes de estudio y orientación de las titulaciones. Se ofrecen cursos de robótica en carreras técnicas, superiores, postgrados, máster y doctorado. En algunos casos, la denominación *robótica* se acaba asociando a materias muy diversas que tratan temas flexibles de fabricación, CAM, programación de robots, teoría de control, etc. Por otro lado, el carácter de realización de las asignaturas de robótica puede ser obligatorio, troncal u optativo, dependiendo de la universidad y especialización. Un buen punto de partida para la búsqueda de asignaturas sobre robótica en España es la base de datos EURON[2]. Según esta base de datos, existen actualmente 33 titulaciones universitarias en relación.

Según el informe de la Fundación COTEC, de oportunidades tecnológicas *"Robótica y Automatización"*, se manifiesta la necesidad de desarrollar formación más específica en robótica y automatización. Resulta evidente que la creación de un título orientado a las tecnologías y aplicaciones de robótica se hace necesaria y muy demandada por los sectores productivos.

Según cita el Libro Blanco de Robótica *[Libro Blanco, CEA-GTRob, 2007]*, varias universidades, apoyadas por la ANECA, proponen a través de su Libro Blanco de Titulaciones, un título de grado de Ingeniería de Automática y Electrónica Industrial. Este último y/o el de Ingeniería Robótica podrían cubrir la demanda existente en robótica y automatización. Por otro lado, a diferencia de las restricciones del plan docente para las titulaciones de grado, la ausencia de asignaturas troncales obligatorias en los postgrado, podrían facilitar, en los años siguientes, la especialización de ciertas universidades en esta material. No obstante, GTRob demanda un esfuerzo de homogeneización que facilite el trabajo común y que pueda ser ampliamente compartido.

Consciente del prometedor futuro de la robótica a nivel mundial, la URJC apuesta por la formación de ingenieros en el campo de la robótica informática.

1.4 *Docencia de robótica en la URJC*

Dentro del plan docente de la URJC se imparte *robótica* como asignatura obligatoria en la titulación de *Ingeniería Técnica en Informática de Sistemas* y como asignatura opcional en la titulación de *Master Universitario en Sistemas Telemáticos e Informáticos*. Adicionalmente, la asignatura de *introducción a la robótica* se oferta como materia de libre elección para otras titulaciones.

Los conocimientos impartidos giran alrededor de la generación de comportamiento artificial en robots, tocando multitud de temas como lógica borrosa, visión artificial, estimación, teoría de control, elaboración de mapas, inteligencia artificial, arquitecturas de control, agentes, etc. Se tratan en definitiva de problemáticas de programación de inteligencia en robots.

El programa de la asignatura incluye los siguientes temas de teoría:

- *Introducción a la robótica*

[2] http://euron.upc.es/rcdb

- *Sensores*
- *Actuadores*
- *Construcción de mapas*
- *Localización*
- *Navegación*
- *Sistemas reactivos y control*
- *Visión en robots*

El programa docente de la asignatura también incluye la realización de prácticas de programación de inteligencia en robots sobre la plataforma *jderobot* mantenida por el *grupo de robótica del GSYC*[3]. Estas prácticas exponen a alumno al afianzamiento de los conceptos teóricos y se centran en la programación de comportamientos en un robot autónomo simulado para técnicas de navegaciones VFF, híbridas, basadas en visión, etc.

[3] http://www.robotica-urjc.es

Capítulo 2
Objetivos y metodología

Como hemos visto en la introducción, el desarrollo de la docencia en robótica en la URJC es una actividad que no solo se plantea en el plano de la educación teórica sino en la programación de robots. El mantenimiento del proyecto *jderobot [Cañas Plaza, 2003]* forma parte de los puntos clave del grupo de investigación de esta Universidad. El desarrollo de una plataforma de programación de robots que simplifique el control y permita llevar a cabo aplicaciones complejas mediante una buena organización del software ha sido una línea maestra para el evolución y mantenimiento de la plataforma. Por otro lado, desde las versiones recientes se ha facilitado el acercamiento de este sistema a grupos de usuarios y alumnos durante su etapa de iniciación en la programación de robots. El ámbito de uso de *jderobot* no se restringe solo al grupo docente de la URJC sino también a otras universidades que han mostrado interés por el uso de la plataforma *jderobot [programación jderobot, Cañas Plaza, 2009]*.

En 2.009 comienza el desarrollo de una nueva versión de *jderobot* a la versión 5.0, fruto del trabajo fin de master de David Lobato Bravo[2] *[jderobot 5 thesis, D. Lobato, 2010]*. En esta versión se da énfasis a *la flexibilidad,* esto es, dar la capacidad al programador para determinar la organización de los componentes de su aplicación. Este requisito de partida pretende estar alineado con las últimas tendencias en software para robots.

2.1 Objetivos

El presente trabajo fin de master tiene por objetivo principal, desarrollar un entorno de docencia en robótica mediante la programación de una aplicación en la plataforma *jderobot 5.0.* Dicha aplicación deberá satisfacer con la especificación funcional dada, así como con los requisitos de arquitectura y organización finados para *jderobot* 5. Estos componentes habilitarán el empleo de *jderobot 5.0* como entorno de enseñanza y aprendizaje de alumnos durante la programación de algoritmos de navegación clásica de robots o procesamiento visual. Para ello, se toma como referencia los elementos software de docencia existentes en la versión anterior *jderobot 4.3*.

Conviene precisar la implicación del objetivo principal como descomposición en subobjetivos en los que se centrará el desarrollo de TFM:

- *Soporte software de simulación gazebo desde componentes jderobot 5.0:* El software de simulación *gazebo,* el cual se explicará en detalle en el próximo capítulo de infraestructura, supone una gran ventaja para la programación de robots, dado que facilita la interfaz con uno o varios robots *virtuales* que son emulados dentro de este software. Proporcionan además una interfaz de operaciones para el manejo de los sensores y actuadores del robot igual a la de los modelos reales. El desarrollo de un componente software que realice la conexión de los programas al entorno de simulación *gazebo* resulta

obligatorio para el propósito de docencia. El alumno dispondrán de una interfaz con este entorno mediante la cual podrá validar la inteligencia implementada en sus programas sin el riesgo de ocasionar choches y averías en un robot físico. También solventa la limitación de tener que disponer de un robot físico para la realización del trabajo.

- *Desarrollo de interfaz de operación del robot:* Facilitar al alumno con un componente de presentación del entorno *jderobot* para el desarrollo de aplicaciones. Se realizará la abstracción de toda la complejidad de programación en *jderobot 5.0* en lo que se refiere al cumpliendo de las restricciones arquitectónicas, la comunicación entre componentes a través del *middleware*, etc. El alumno se podráncentrar en la programación de algoritmos que doten de inteligencia al robot, por ejemplo, durante la navegación del mismo, contando con una interfaz simplificada de acceso a los dispositivos del robot. Cuestiones como la presentación gráfica de los parámetros de control del robot, mediante el uso de librerías tipo GTK, XForms, OpenGL, etc, la comunicación con otros componentes mediante el elemento *middleware Ice*, son aspectos que para el alumno deben resultar transparentes. Por lo tanto, el TFM deberá diseñar una solución válida que oculte este trabajo y alivie la dificultad del programador para ejecutar su código.
- *Desarrollo de prácticas de navegación sobre jderobot 5.0:* Como parte del trabajo de validación del entorno de prácticas con la realización de los dos subobjetivos anteriores, se programarán diferentes lógicas de navegación del robot. Cada uno de ellos empleará varios de los sensores disponibles en el robot (laser, camara, etc.) y todos actuarán sobre los motores para lograr la navegación del robot hacia el objetivo propuesto. Este trabajo servirá para la validación correcta del entorno de enseñanza.

2.2 Requisitos

A continuación se enumeran los requisitos identificados inicialmente para el desarrollo del entorno docente sobre el sistema *jderobot* 5.0. Se hará una distinción entre requisitos funcionales y no funcionales y se adoptarán algunos de los requisitos marcados para el desarrollo de la nueva plataforma dentro de los de este TFM.

2.2.1 Requisitos funcionales

Estos requisitos agrupan las características requeridas en los nuevos componentes relativos a las capacidades de los mismos.

- ***Requisito (1) "Entorno docente simplificado y funcional (esqueleto)***": Como se comentó en la sección de objetivos, se requiere que el desarrollo de componentes docentes presente una interfaz de programación simplificada, en la que los detalles de integración entre los componentes y el uso de la infraestructura subyacente sea transparente para el programado. Ofrecer un ámbito de programación sencillo y centralizado donde se pueda acceder por completo a los dispositivos de la percepción y control del robot. No penalizar la funcionalidad o capacidad del programador de definir algoritmos con propósitos variados (navegación, procesamiento visual, etc). En definitiva, el

proyecto deberá entregar un entorno de programación de robots *esqueleto* en el que solo se deba añadir la lógica de inteligencia del robot.

- ***Requisito (2) "Desarrollo orientado a Componente***": Se introduce en la tesis de David Lobato *[jderobot 5 thesis, D. Lobato, 2010]* al *componente* como la unidad mínima del sistema, el cual dispondrá, por parte del sistema, de diferentes mecanismos la comunicación con otros componentes. Para el desarrollo del entorno docente, en un primer paso se identificarán cuantas unidades mínimas se tendrán que definir y de qué modo deberán conectarse.
- ***Requisito(3) "Comunicación de componentes mediante middleware Ice"***: Se debe emplear los mecanismos de comunicación basados en el *middleware Ice* para conectar los elementos software desarrollados.

2.2.2 Requisitos no funcionales

Estos requisitos señalan las restricciones de desarrollo que han sido tenidas en cuenta durante la realización del software.

Requisitos del sistema

- ***Requisito (4) "Rendimiento"***: Los componentes implementados no deben penalizar significativamente el rendimiento del computador, concediendo a las lógicas de control implementadas dentro del *esqueleto*, la mayor parte de los recursos del computador.
- ***Requisito(5) "Robustez"***: El sistema implementado debe ser robusto ante cualquier problema de comunicación con otros componentes del sistema y en el manejo de excepciones de ejecución. Al mismo tiempo que se ofrece una interfaz simplificada para el encapsulación del código desarrollado por el alumno, se debe impedir posibles injerencias del código del usuario mecanismos de funcionamiento interno de los componentes, ya sean por error o intencionados.

Requisitos del proceso de desarrollo

- ***Requisito(6) "Plataforma"***: Se recomienda trabajar en entornos GNU/Linux que son los que se han utilizado principalmente en los últimos años y es donde mayor experiencia se tiene en el grupo. La versión actual de *jderobot* 5 ha sido validada solo en esta plataforma.
- ***Requisito(7) "Documentación"***: Se realizará una implementación de código con comentarios explicativos de cada parte del programa, así como referencia en el contenido de esta memoria, a los algoritmos y aspectos de diseño técnico de cada solución agregada a la plataforma docente.
- ***Requisito(8) "Integración al proyecto jderobot"***: Todos los productos desarrollados con este trabajo, código, documentación y herramientas de integración del proyecto deben ser parte del alcance del proyecto. Se entregará una solución correctamente enlazada en los procesos de configuración y compilación del proyecto *jderobot 5.0*.

2.3 Metodología y planificación del trabajo

Durante la realización del trabajo fin de master se ha empleado una metodología de desarrollo software. Como fue presentado en el texto introductorio de la memoria, la actual complejidad en el desarrollo de aplicaciones para robots desencadena el desarrollo de plataformas y arquitecturas de referencia que simplifiquen la labor. El uso de metodologías de la ingeniería de software reducen la complejidad del problema y aumentan las posibilidades de alcanzar programas reusables, escalables, fácilmente mantenibles y seguros.

Varias propiedades del proyecto son interesantes de resaltar en este punto:

- ***Coordinación con el tutor:*** Durante los diez meses que duró el trabajo, se mantuvieron revisiones periódicas con el Tutor para la medición de los avances logrados y la definición de nuevos pasos hacia el objetivo final.
- ***Aprendizaje de Alumno en plataformas existentes:*** Para la consecución de los objetivos marcados, el alumno tuvo que superar una fase previa de aprendizaje y familiarización con el entorno *jderobot*, así como el estudio de técnicas de navegación con robots. Esta fase de proyecto fue igualmente seguida con el Tutor y en ella no fue empleada una metodología tan rigurosa como la usada en el desarrollo de los componentes de la solución final.
- ***Plazo de entrega:*** Conocido desde el comienzo por la duración del curso académico y normativa de evaluación del TFM, el contenido y objetivos del trabajo fueron dimensionados y distribuyendo en el tiempo de un modo abordable.

Desde el punto de vista de las distintas fases por las que ha ido evolucionando el trabajo realizado, podemos resaltar las siguientes:

- ***Familiarización con la plataforma software jderobot 4.3***: Estudio de la estructura y diseño del proyecto *jderobot* en la versión estable y oficial. Para ello se realizó un trabajo de integración de la plataforma en el entorno de desarrollo empleado por el alumno, así como estudio de documentación existente en la web oficial[4]. Puesto que *jderobot* 4.3 obliga a realizar la programación en el lenguaje C, se dedicó también cierto tiempo a la preparación y familiarización con la plataforma de desarrollo de este lenguaje, preparando el entorno con el resto de librerías de apoyo para la compilación y enlazado.
- ***Implementación de algoritmos de navegación sobre la plataforma 4.3***: Durante la segunda etapa se estudiaron varios algoritmos de navegación clásicos y se realizó su implementación dentro del entorno docente de la plataforma en versión 4.3. Fue utilizado el software de simulación *gazebo* y *player/stage* para la programación de la navegación en el robot Pioneer. Los algoritmos clásicos usados se describirán en el capítulo 5 de prácticas realizadas.
- ***Familiarización con la plataforma jderobot 5.0***: Se dedicó un periodo a la preparación de la nueva plataforma *embrionaria* como entorno de desarrollo de la nueva aplicación docente y se estudió la nueva organización distribuida

[4] http://jderobot.es

y basada en comunicación mediante el *middleware Ice*. Para completar la etapa, también se realizaron algunos desarrollos de prueba para el uso de la infraestructura principal.

- ***Creación del entorno de prácticas en 5.0***: Tomando como referencia los esquemas de la 4.3 destinados a impartir la materia de robótica en la URJC, así como habiendo comprendido de funcionamiento de *jderobot 5.0*, se precisaron los requisitos funcionales del nuevo entorno de enseñanza de robótica. A partir de la especificación anterior, se realizó el análisis, diseño e implementación de los nuevos componentes. Durante esta fase se empleó una metodología evolutiva basada en prototipos en la que se programaron los componentes *driver gazeboserver* e *introrob*. Se emplearon dos ciclos evolutivos para cada programa Los primeros ciclos respectivos de cada componente alcanzaron el soporte del robot con el sensor cámara (*camera*) y tuvieron como principal propósito, realizar un *proof of concept* de la solución. Los ciclos segundos de cada componente completaron la solución con el soporte completo de todos los dispositivos del robot.
- ***Repetición de los desarrollos de la etapa 2 sobre la nueva plataforma***: Portación de los algoritmos de navegación realizados durante la segunda fase del proyecto, sobre el nuevo entorno de prácticas 5.0. Validación de los componentes desarrollados en la fase cuarta y desarrollo de correcciones, mejoras de los componentes 5.0.

La siguiente figura (Fig 2.1) muestra la distribución temporal de las fases nombradas a lo largo de la duración del proyecto:

***Figura 2.1**: Distribución temporal de fases del trabajo*

Como se ha nombrado en la descripción de la etapa cuarta, en el desarrollo del entorno de prácticas, se empleó la metodología de desarrollo en espiral basado en prototipos como modelo de referencia.

Este modelo de desarrollo se caracteriza por la realización de las actividades en un número determinado de ciclos. Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración amplia los objetivos y alcance de la versión anterior. De este modo se muestra flexible ante la adaptación de los requisitos, consecuencia del cambio de necesidades y descubrimiento de una mejor alternativa a la solución.

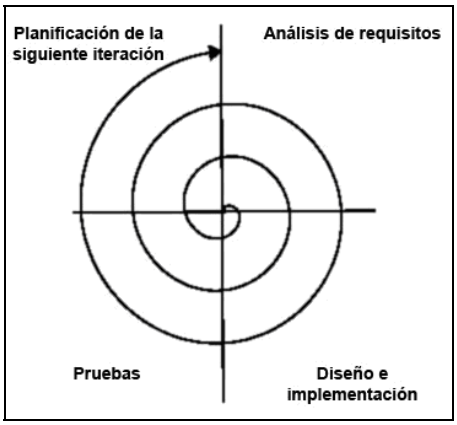


***Figura 2.2**: Diagrama modelo en espiral*

Cada bucle o iteración lo componen cuatro etapas: análisis de objetivos (requisitos), diseño e implementación, pruebas y planificación del próximo ciclo de desarrollo. Con cada iteración se van cumpliendo nuevos objetivos o subobjetivos y, además, se va comprobando que el camino que se está siguiendo es el correcto en la etapa de planificación.

Capítulo 3

Infraestructura

En este capítulo de la memoria se describirán aquellos componentes software en los que se soporta la solución requerida o han servido de referencia para el desarrollo de la misma, como es el caso de la plataforma *jderobot 4.3*, versión previa a la usada en el trabajo. Se presenta inicialmente el software *gazebo* como entorno de simulación de robots con el que el entorno docente desarrollado interactuará para el manejo de los robots simulados. Posteriormente se introducirá cuáles son las características y estructura general de *jderobot* 4.3. También el entorno de prácticas de esta versión como solución de referencia. Finalmente se presentará la plataforma *jderobot* 5.0 y la librería *GTK* de desarrollo de interfaces gráficas de aplicaciones.

3.1 Plataforma de simulación Gazebo

Gazebo es un simulador multi-robot que tiene la capacidad de simular, en un *mundo* tridimensional, una población de robots y objetos. Los primeros simuladores eran poco realistas y almacenaban mundos planos donde había obstáculos estáticos bidimensionales. Con los años se ha ido ganando en realismo, incorporando ruido en los sensores y en las actuaciones. *Gazebo* proporciona retroalimentación de sensores con valores realistas y reproduce las propiedades de cinemática y dinámica sobre los robots y resto de piezas modeladas en su *mundo virtual*. En este tipo de simuladores, más potentes, se ha incluido también la capacidad de representar un conjunto de robots operando en el mismo escenario de modo simultáneo (Fig 3.1.a). Está desarrollado bajo la licencia *GNU Public License* de modo que también ofrece la posibilidad de ser adaptado a las necesidades de un entorno particular.

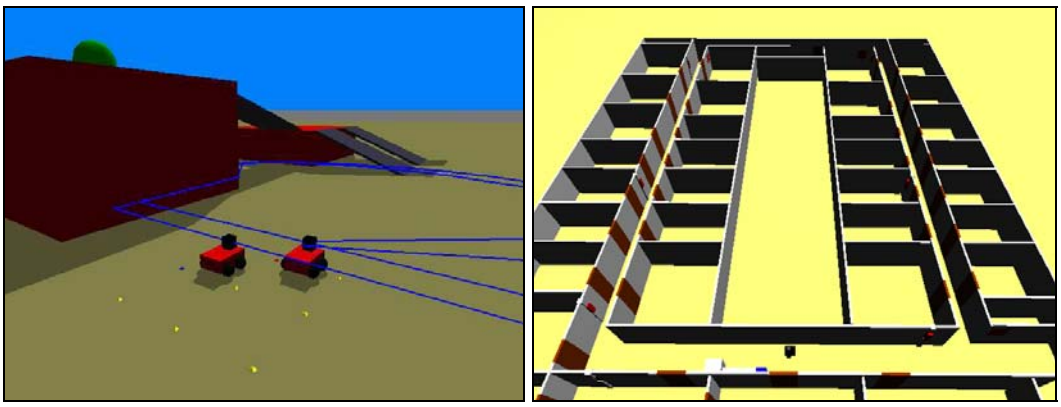


Figura 3.1: *visualización de modelos 3D simulados en gazebo.* ***(a)*** *modelo con 2 robots Pioneer2DX,* ***(b)*** *robot Pioneer2DX en reproducción virtual área de partamental GSYC URJC*

El software *Gazebo* se utiliza normalmente en combinación con un servidor para este sistema de reproducción; El software *Player* (también soportado mediante *driver* en *jderobot 4.3*) ofrece un mecanismo a través del cual los controladores de robot (clientes) pueden interactuar con el hardware del robot. Dicha comunicación con *Player* es realizada a través de los servicios de red que la aplicación expone (Fig 3.2).

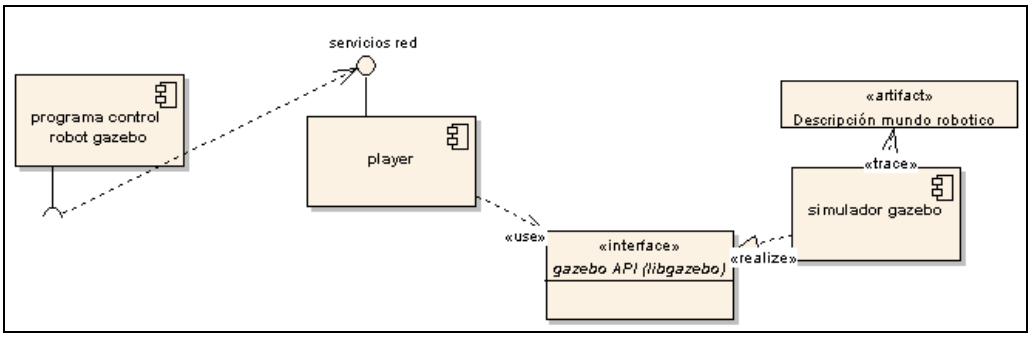


***Figura 3.2**: Diagrama integración Player como interfaz para acceso a gazebo de otras aplicaciones de control de robots*

Como se describe en la figura anterior (Fig 3.2), la implementación interna del acceso de *Player* a las órdenes de *Gazebo* se realiza mediante el *API* C (*libgazebo*). Esta biblioteca se incluye para permitir a los desarrolladores de terceras aplicaciones la integración de este software. En *jderobot 4.3*, el driver *gazebo* utiliza este API para trabajar directamente con el simulador *gazebo* sin pasar por la interfaz *Player*.

En la versión actual del simulador están soportados varios modelos de robots, existentes en la realidad como productos comerciales. Estos son el *Pioneer2DX*, *Pioneer2AT* y *SegwayRMP* (Fig 3.3). Para cada uno de ellos, se simulan los sensores estándares de estos robots, tales como *encoders, cámara, ptencoders, sonars y laser*. También los actuadores *motors y ptmotors*.

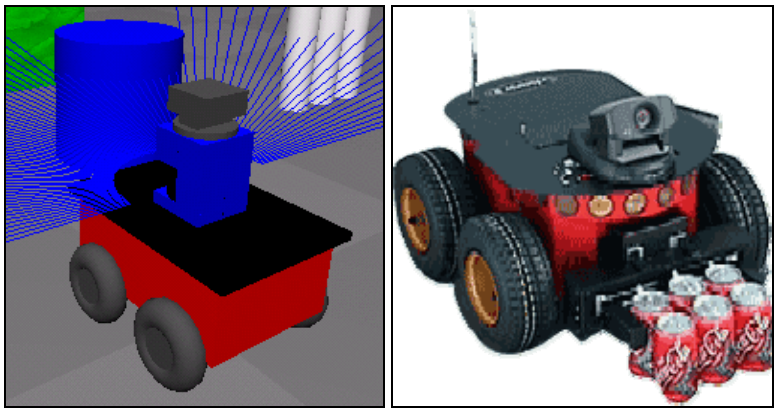


***Figura 3.3** Robot Pioneer2AT*
***(a)** simulación gazebo con sernsor laser*
***(b)** ejemplo real con cámara*

Los simuladores libres tratan de incluir soporte para los robots de diversos fabricantes. La configuración concreta del conjunto de robots a simular, la disposición y parámetros de sus sensores se suele especificar en un fichero de

configuración (List. 3.1). Dos de los simuladores más relevantes de hoy en día son el *SRIsim*[5] de *Kurt Konolige*, que se emplea en los robots de *ActivMedia* y los simuladores *Stage* y *Gazebo*[6] de software libre orientados a multirrobot (Stage soporta 2D y colonias numerosas de robots, y Gazebo soporta 3D).

```
<?xml version="1.0"?>
<gz:World xmlns:gz="http:…. >
  <model:ObserverCam>
    <id>userCam0</id>
    <xyz>10 0 20</xyz>
    <rpy>0 30 40</rpy>
    <imageSize>640 480</imageSize>
    <displayRays>false</displayRays>
    <farClip>1000</farClip>
    <shadeSmooth>true</shadeSmooth>
    <polygonFill>true</polygonFill>
  </model:ObserverCam>
  <model:Pioneer2DX>
    <id>robot2</id>
    <xyz>2. 2. 0.200</xyz>
     <rpy>0.0 0.0 0.0</rpy>
    <model:SickLMS200>
      <id>laser2</id>
      <xyz>0.0 0.0 0.00</xyz>
     <model:SonyVID30>
        <id>camera2</id>
        <xyz>0 0 0.2</xyz>
      </model:SonyVID30>
      <rayCount>180</rayCount>
      <rangeCount>180</rangeCount>
    </model:SickLMS200>
  </model:Pioneer2DX>
</gz:world>
```

***Listado 3.1**: Ejemplo configuración mundo gazebo*

En definitiva, se trata de una herramienta muy útil en la programación de robots que ofrece un entorno virtual en el que emulan las mediciones de los sensores y los efectos de las órdenes sobre los actuadores. Sirven para evaluación, depuración y ajuste del programa de control antes de ser llevado al robot real.

3.2 jderobot 4.3

jderobot es una plataforma que permite desarrollar aplicaciones robóticas y domóticas con sensores y actuadores, que tomen decisiones inteligentes de manera autónoma. Esta plataforma desarrollada en la URJC y que se cimenta sobre la tesis doctoral realizada por José María Cañas Plaza *[Cañas Plaza, 2003]*, establece una base cognitiva para el desarrollo de la arquitectura software del sistema.

[5] http://www.ai.sri.com/~konolige/saphira

[6] http://playerstage.sourceforge.net

En *jderobot [Cañas Plaza et al., 2007]* las aplicaciones se conciben y organizan como un conjunto de componentes concurrentes. Los componentes de la aplicación de un usuario se conocen como esquemas y los que facilitan el manejo de sensores y actuadores como drivers. Cada uno de ellos tiene un hilo de ejecución propio y realizan una tarea concreta de forma iterativa. Todos ellos pueden compartir entre si determinadas variables durante su ejecución.

jderobot 4.3 es la última versión estable de la plataforma desarrollada por el grupo de investigación de robótica de la URJC. Dicho software ha servido de marco para el desarrollo de muchos trabajos realizados en el grupo. Su diseño procede de los conceptos extraídos de la ingeniería del software orientada a componentes, donde las unidades mínimas, los componentes, se comunican a través de una serie de interfaces bien definidas. En *jderobot 4.3*, dichas unidades mínimas son los *esquemas* y *drivers*, los primeros con propósitos de actuación y los segundos con finalidad perceptiva. El sistema está programado íntegramente en el lenguaje C.

Según se resume en *[jderobot 5 thesis, D. Lobato, 2010]*, los principales mecanismos que aporta *jderobot 4.3* son:

- ***Abstracción del hardware***: Mediante el uso de interfaces bien de nidos, se abstrae el uso de los diferentes dispositivos que puede contener un robot, incluso soportando aquellos simulados desde Player/Stage.
- ***Comunicación entre esquemas***: El sistema implementa mecanismos para que los esquemas puedan exportar/importar variables con las que comunicarse.
- ***Carga dinámica de esquemas***: El sistema implementa un mecanismo de carga dinámica de esquemas a modo de plugins.
- ***Comunicación remota entre esquemas***: El sistema implementa mecanismos muy básicos para comunicación remota entre esquemas ejecutando en nodos de cómputo diferentes.
- ***Ejecución cooperativa***: El sistema implementa mecanismos básicos de comunicación entre procesos *jderobot*, de manera que es posible crear aplicaciones con procesos *jderobot* cooperantes, incluso ejecutándose en máquinas diferentes.

Esta posibilidad de de establecer comunicación entre los diferentes esquemas y drivers posibilita el desarrollo de aplicaciones complejas basadas en módulos funcionales más simples y de funcionalidad acotada. Los componentes pueden organizarse en diversos niveles, pudiéndose establecer una jerarquía entre módulos padre e hijos, siendo el padre el que puede activar, desactivar y modular a los hijos según sus necesidades (Fig. 3.4).

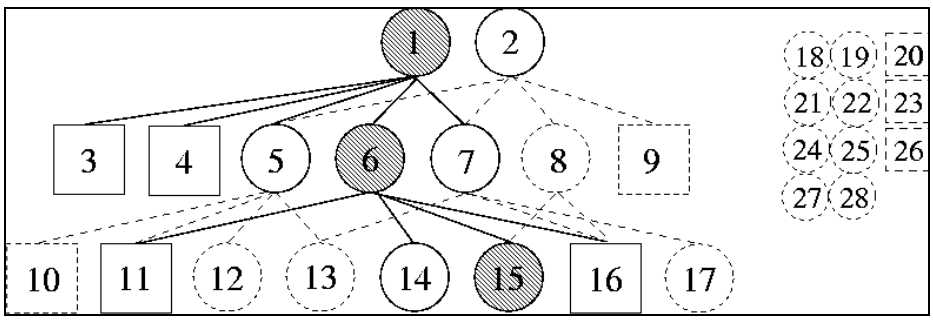


***Figura 3.4**: Ejemplo organización jderobot.*
Los círculos representan esquemas de actuación.
Los cuadrados a los drivers de percepción.

Las lógicas de los esquemas y drivers se ejecutan de manera concurrente como *hebras (pthreads)* controladas por un hilo principal de arranque. Cada una de estos programas, puestos en ejecución como un *hilo*, ejecutan instrucciones de forma iterativa. El *controlador* de *jderobot* decide en cada momento sobre el lanzamiento de una nueva iteración de los hilos y tiene además la capacidad de operar el estado de cada uno (parar, arrancar, suspender, etc). El resultado observado es una ejecución continua de todos los programas capaz de llevar acabo los objetivos de percepción y actuación del robot. El *hilo de arranque* del proceso leerá la configuración de un fichero de entrada, donde estarán indicados todos los componentes que se deberán cargar, así como el estado de inicialización y la activación de su interfaz gráfica, en caso de implementarla. Durante la ejecución de la plataforma, se podrá interactuar con ella a través de un entorno línea de comandos.

Las aplicaciones son desarrolladas de forma modular pero siguiendo ciertas pautas en cuanto a su organización en funciones, debiendo existir unas obligatorias con un propósito predefinido, de manera que la lógica de supervisión pueda funcionar sobre el componente:

- *schema init, schema terminate*: Permite a otros esquemas el arranque o finalización de la *hebra* en el sistema *jdec*.
- *schema run, schema stop:* Sobre un esquema en ejecución, la invocación de estas funciones produce la activación o desactivación del aplicativo. A diferencia de las anteriores, el hilo ya existe en el sistema con la configuración cargada o seguiría existiendo después de la llamada *stop*. Simplemente, la iteración de ejecución sobre todos programas, no concedería turno a la lógica *pausada*.
- *schema iteration*: Llamada mediante la cual se ejecuta la lógica del *schema* durante un ciclo.
- *schema show, schema hide*: Para aquellos programas que implementen una interfaz gráfica, estos procedimientos permitirán activarla o ocultarla. El propósito perseguido es el de poder mantener aplicaciones interactivas, momentáneamente ocultas y en segundo plano o simplemente que no informen al usuario sobre la información que se presenta en la *GUI*.

Otros aspectos a tener en cuenta durante el diseño e implementación en *jderobot 4.3*, son los de la gestión explícita de la exclusión mutua en el acceso a recursos comunes para varios esquemas y drivers. La existencia de varias *hebras* en ejecución plantea esta situación de competencia por los recursos, de modo que un diseño seguro de las mismas, requiere del conocimiento del programador para usar los mecanismos de control dispuestos por la plataforma y basados en recursos *IPC* del sistema operativo (List. 3.2). Si bien el desarrollo de la aplicación es modular, el hecho de tener que realizar una compilación conjunta del proyecto y enlazar espacios de variables comunes para comunicar componentes, obliga a fijar normas de programación concurrente que garanticen un correcto acceso a los recursos compartidos.

```
pthread_mutex_lock(&(all[introrob_id].mymutex));
```

```
…
pthread_mutex_unlock(&(all[introrob_id].mymutex));
```

***Listado 3.2**: llamadas al sistema en código de esquema para el control de acceso con exclusión mutua.*

Por último, junto a los programas de tipo *driver* y *esquema* que se han comentado, existe otra clase de módulos en *jderobot:* los *servicios*. En general, se usan como componentes auxiliares de la plataforma que facilitan la operación del entorno (*MasterGUI* o *Shell*) o permiten la colaboración entre diferentes entornos *jderobot* distribuidos (*networkserver, networkclient*). Además vemos otros elementos que se denominan servicios *Srv GTK y Srv Xforms,* que gestionan el acceso concurrente al entorno gráfico, sobre el cual el esquema que lo requiera puede pintar su interfaz de usuario.

Como se ha comentado anteriormente, *jderobot* se ha empleado como plataforma para el desarrollo de diversas aplicaciones en el ámbito de investigación de la URJC. A continuación se nombran algunos ejemplos:

- ***Eldercare**:* Aplicación capaz de identificar una persona ha sufrido una caída a partir de la información obtenida desde varias cámaras. Se apoya en algoritmos de detección 3D usando cámaras calibradas *[Marugán Alonso, 2009]*.
- ***Detección y seguimiento de caras**:* Programa para la detección y seguimiento de caras en secuencias de vídeo. Utiliza detectores basados en apariencia *[Martín Ramos, 2008]*.

3.3 Entorno de desarrollo en jderobot 4.3

De los diversos drivers y esquemas que se entregan en la versión estable de *jderobot*, podemos distinguir algunos que están destinados a la labor docente o sirven de soporte a los alumnos durante la ejecución y validación de sus prácticas:

- ***Driver player***: este driver conecta con la aplicación *player/stage*, que nos da acceso a un robot simulado con *stage* o real.
- ***Driver gazebo***: este driver implementa la interfaz con el software simulador *gazebo*, que nos da acceso a un robot simulado en un mundo virtual 3D.
- ***Driver graphics xforms***: materializa el soporte para que los esquemas puedan mostrar su interfaz gráfica, desarrollada con la biblioteca *XForms*.
- ***Esquema mastergui*** (Fig. 3.5): permite activar y desactivar a voluntad la ejecución de los distintos componentes cargados.
- ***Esquema teleoperator*** (Fig. 3.5): permite mostrar visualmente las medidas de los sensores del robot (láser, odometría, etc) y teleoperar al robot simulado usando un joystick gráfico.
- ***Esquema introrob***: este es el esquema que se utiliza para programar al robot, empotrando en el código de las prácticas. Por ejemplo, permite una programación sencilla del robot con algoritmos de navegación, simplemente editando uno de sus archivos fuente donde se encapsula esta lógica.

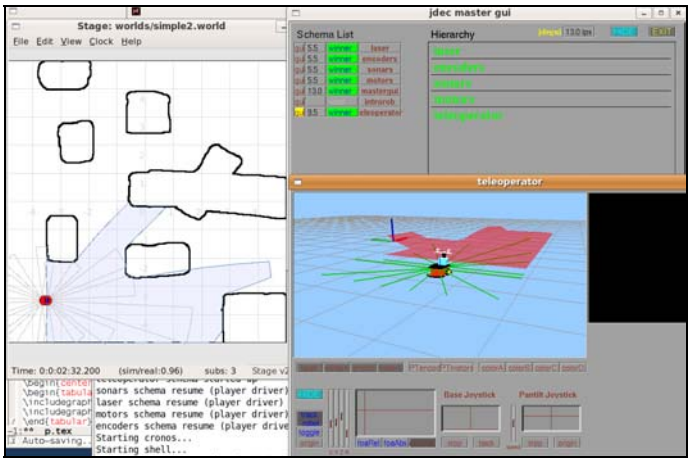


***Figura 3.5**: Esquemas MasterGUI y Teleoperator*

El anterior conjunto de esquemas y drivers lo podríamos representar en un diagrama de componentes como en que muestra la figura 3.6.

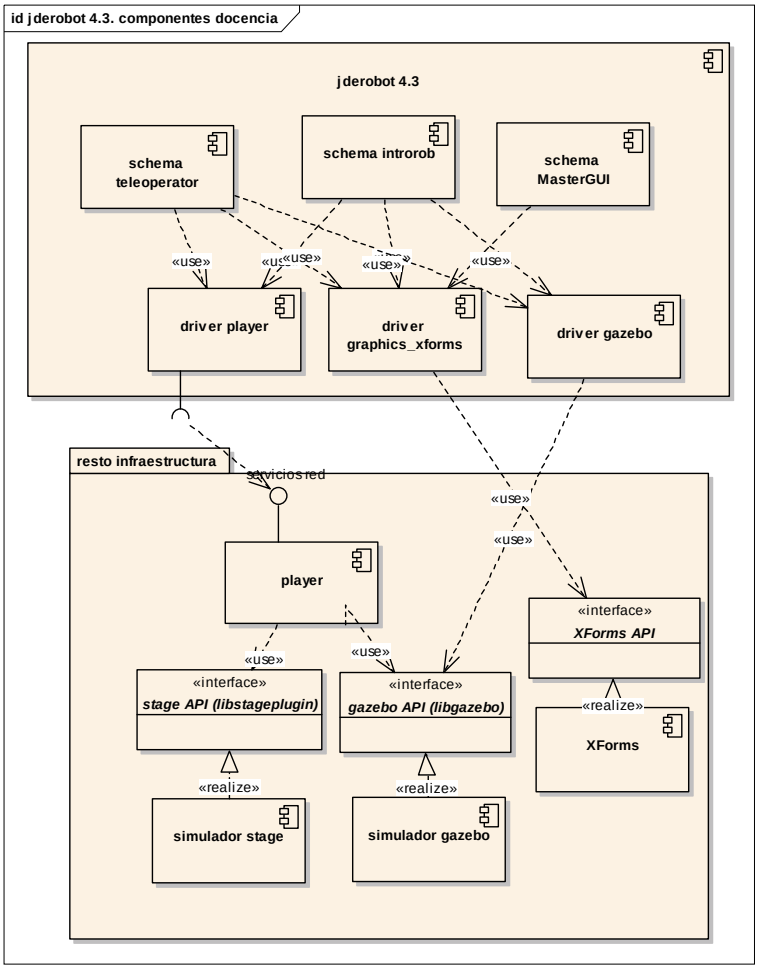


***Figura 3.6**: Organización de drivers y esquemas del contexto docencia de jderobot 4.3. Relación con otros componentes de infraestructura.*

La existencia de drivers de conexión a *player/stage* y a *gazebo* permite al alumno la elaboración de programas sobre una plataforma simulada que siempre estará disponible. El esquema *teleoperador* es una aplicación operadora manual para un robot, real o simulado, que sirve de ayuda para su colocación en posición y orientación deseada, antes de lanzar la ejecución del código. Por su parte, *MasterGUI* es un esquema servicio que sustituye a la interacción con línea de comandos de *jderobot* y desde el cual se pueden operar los estados de cada hebra o programa.

Destacaremos por el propósito docente, el esquema *introrob*, el cual muestra una interfaz gráfica (Fig. 3.7.a) desde la que se puede acceder a la información sensorial del robot, teleoperar y ejecutar el código programado por el alumno.

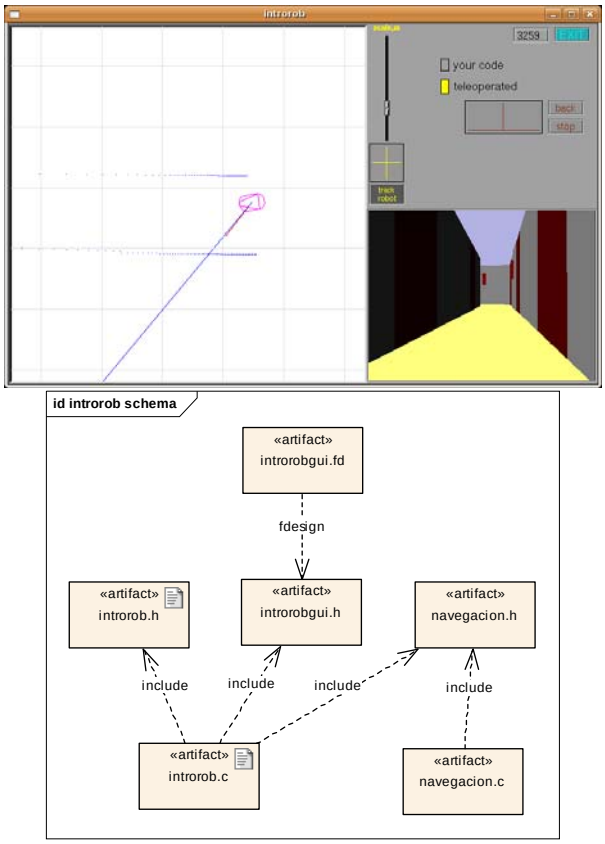


***Figura 3.7**:.**(a)** Interfaz gráfica de esquema introrob.*
***(b)** Ficheros de código C para introrob*

Como se muestra en la figura 3.7.b, el código fuente del programa se distribuye en dos ficheros principales, *intorob.c* y *navegación.c*, con sus respectivos archivos cabecera. También se incluye un diseño de interfaz gráfica *introrobgui.fd* generado con la herramienta *FDesign* y a partir de la cual se convierte a lenguaje C. Dentro del fichero *navega.c*, es donde el programador del robot especifica las líneas de código que realizan la inteligencia del robot. Los detalles de integración en la plataforma, como hilo de ejecución y cumpliendo las reglas de funcionamiento en este entorno, quedan abstraídas en *introrob.c*. Desde *navegación.c*, como describe *[P.robótica. Cañas, 2008]*, los datos del sensor láser (Fig.8.b) se ofrecen a vuestro programa a través de la variable *láser* que es un array con 180 medidas,

correspondientes a las distancias al obstáculo más cercano en cada uno de esos 180 grados.

```
float v;
float w;
float robot[5];
float laser[NUM_LASER];
```

***Listado 3.3**: Definición de variables introrob*

Los datos del sensor de odometría (Fig. 3.8.a) se ofrecen al programa a través de la variable *robot*, que contiene la posición y orientación del robot respecto del sistema de referencia del mundo. En *robot[0]* se tiene la X, en *robot[1]* la Y, y en *robot[2]* la orientación. Para gobernar sus movimientos los programas enviarán órdenes de velocidad de avance (variable *v*) y velocidad de giro (variable *w*) del robot entero. El entorno de desarrollo traduce esas órdenes a las consignas oportunas para el motor de la rueda izquierda y para el motor de la rueda derecha. Al ser la naturaleza de esta plataforma iterativa, resulta idónea para programar algoritmos de navegación local o construir incrementalmente mapas.

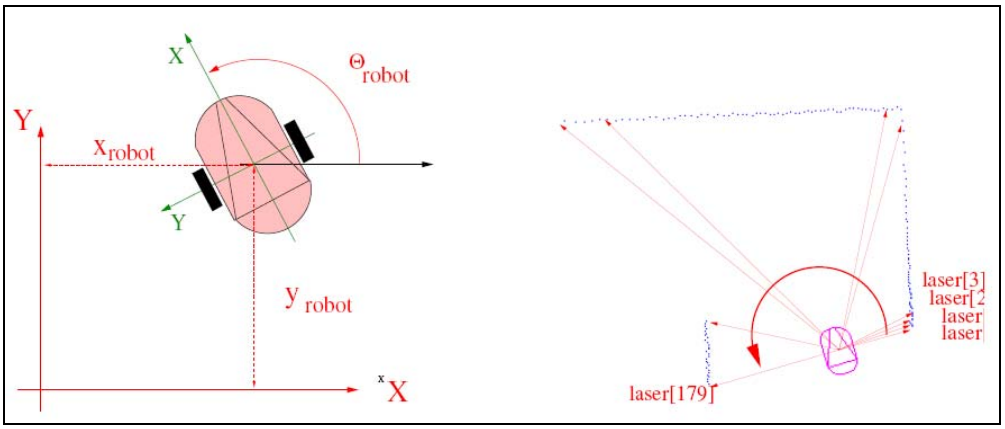


***Figura 3.8**: Variables sensoriales introrob*

3.4 jderobot 5.0

Como se ha referido en una parte previa del documento, *jderobot 5.0* surge del TFM realizado por David Lobato Bravo *[jderobot 5 thesis, D. Lobato, 2010]*. El objetivo global fue la creación de una nueva implementación para la plataforma que aplicase las técnicas encontradas en las últimas tendencias en la materia y apostase por un diseño que facilite la reutilización (orientación a componentes). Además, en el nuevo sistema no se fija ningún tipo de organización de sus partes, permiten ser programadas en múltiples lenguajes y son capaces de ejecutarse en diversas plataformas. La tecnología subyacente se apoya en el *middleware* de comunicaciones Ice de *ZeroC*, que se explicará más adelante en la sección, y que permitirá a los programadores abordar el desarrollo multi-lenguaje y multi-plataforma con un esfuerzo relativamente bajo.

jderobot 5.0 está motivado por varías limitaciones encontrada en la serie 4 y a las que se plantea dar solución.

- Pese a permitir el desarrollo de aplicaciones muy complejas, frecuentemente carecen de flexibilidad. Se apoyan en un modelo cognitivo que no siempre es el más adecuado y restringe al programador sobre la forma de organizar sus componentes.
- La proliferación de lenguajes de alto nivel, cada uno con sus características favorables para determinadas aplicaciones, hace requisito indispensable que un sistema que pretende ser flexible pueda ser programado en múltiples lenguajes.
- El concepto de distribución de componentes por una red de nodos resulta a día de hoy una característica más que deseable en un sistema robótico. Por un lado permite la distribución de la carga de cómputo y por otro permite abordar otro tipo de aplicaciones.

El uso del *middleware Ice* de ZeroC es, sin duda alguna, la característica más importante de esta versión 5. Su uso permite cumplir algunos de los requisitos planteados sin ningún esfuerzo extra. Por otra parte condiciona el diseño de la arquitectura, al tratarse *Ice* de un *middleware* orientado a objetos distribuidos. A continuación se incluyen algunas propiedades del *middleware*, obtenidas en la documentación del producto[7]. Con ellas, se tratará de aclarar algunas de las nuevas capacidades de los componentes *jderobot* de esta versión:

- Proporciona una plataforma de comunicación orientada a objetos para entornos heterogéneos, implementados con distintos lenguajes y tecnologías.
- Reducción de la complejidad de uso e integración, mediante la definición de interfaces entre cliente y servidor con un lenguaje *intermedio* parte del *middleware* denominado *Slice*. La solución de *ZeroC* proporciona traductores de esta especificación a varios lenguajes de programación (*C++*, *Java*, *Python*, *etc.*), implementando de manera automática las interfaces de comunicación de los componentes.
- Mecanismos de seguridad integrados en la plataforma, soporte de encriptación de datos SSL.
- Control de transferencia de información en modo síncrono y asíncrono.
- Distribución de datos a un grupo de clientes (multicast).

La arquitectura software de jderobot se organizará como un conjunto de componentes comunicándose con el esquema cliente-servidor, donde la estructura lógica se podría representar como se muestra en la figura 3.9: Además, otro cambio significativo es el hecho de que cada aplicación se ejecutará como un proceso y no como hilo, independizando completamente el contexto de ejecución y liberando de cuestiones anteriores como la gestión de exclusión mutua o sincronización con recursos *IPC*.

[7] http://www.zeroc.com/doc/

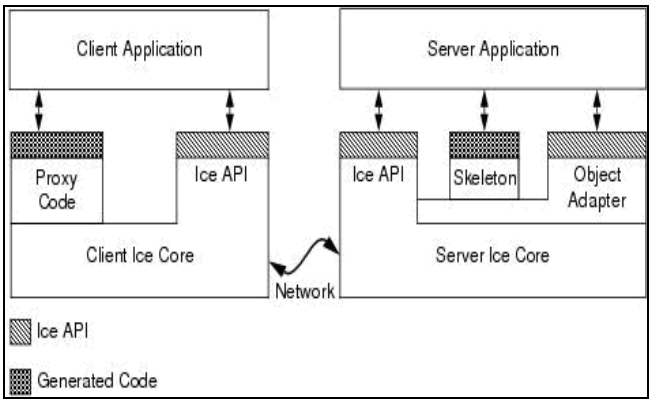


***Figura 3.9**: Diagrama implementación middleware Ice*

Ambos componentes *jderobot* cuentan con librerías creadas a partir del generador de definiciones *slice*.

- Los componentes *cores* soportan el entorno de ejecución para la gestionar la comunicación a través de la red. La *API Ice* proporciona a las aplicaciones cliente y servidor con funcionalidad para establecer y cerrar la comunicación a través del *middleware*.
- El *código proxy* contiene las definiciones de tipos y objetos que han sido especificadas en la definición de la interfaz con el lenguaje *slide*. Tiene dos propósitos fundamentales:
  - *Serialización* de la información a flujos de datos codificados (*marshalling*) que puedan ser distribuidos a través de la red sin complejidad para el código de las aplicaciones y de una forma robusta y segura.
  - Invocación de llamadas RPC del cliente al servidor.
- El *código skeleton* que, al igual que ocurre con otras capas, es generado de forma automática a partir de la especificación *slice* y facilita al lado servidor con las funciones equivalentes a las del *código Proxy* en cliente.
- Finalmente, el *adaptador de objetos* permite a la aplicación servidora el mapeo de objetos de la interfaz y la gestión de uno o varios puntos de transporte de la información.

El diagrama de la figura 3.10 muestra un ejemplo de componentes desarrollados en la actual versión *jderobot*: *cameraserver y cameraview*. La primera aplicación es un servidor de vídeo con la capacidad de obtener las señales de video de varias cámaras de forma concurrente y presentarlas a través de la interfaz. Por su parte, *cameraview* funciona como cliente de reproducción de una señal de video recuperando las imágenes enviadas por *cameraserver* a través de la interfaz *Ice*.

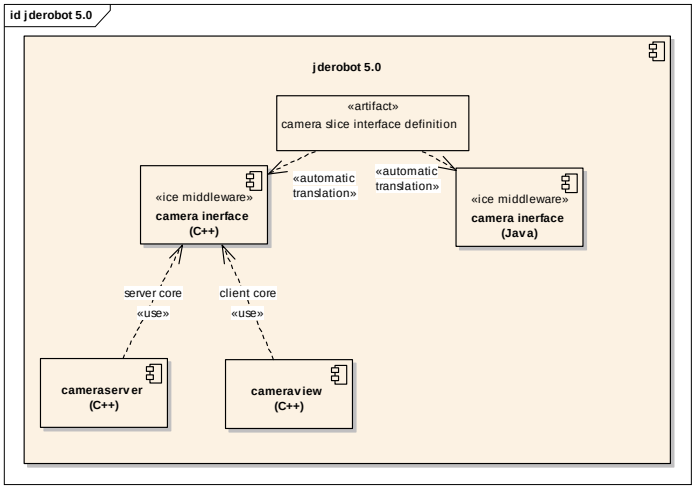


***Figura 3.10**:Diagrama ejemplo componentes jderobot 5.0 con Ice*

Ambos componentes están implementados en lenguaje *C++* aunque, como muestra el diagrama anterior, a partir de la especificación *slice* de la interfaz *camera* que se expone en el Listado 3.4, el código de implementación de las capas del interfaz *middleware* se ha podido generar en otros lenguajes además de *C++*. Aunque no es el caso real, existiría la posibilidad de programar el componente *cameraview* en lenguaje *Java* y lograr una comunicación igual a la conseguida con la codificación en *C++*.

```
module jderobot{
  /**
   * Static description of a camera
   */
  class CameraDescription
  {
    string name;
    string shortDescription;
    string streamingUri;
  };
  /**
   * Camera interface
   */
  interface Camera extends ImageProvider
  {
    idempotent CameraDescription getCameraDescription();
    string startCameraStreaming();
    void stopCameraStreaming();
  };
```

***Listado 3.4**: Especificación de interfaz camera en lenguaje Ice*

Obsérvese en el listado anterior la posibilidad de especificar clases de objetos como en *cameraDescription* o en la posibilidad de extender la definición de la interfaz C*amera* a partir de la realizada para *ImageProvider*. Con esta definición de interfaz, la parte servidora deberá implementar las clases *cameraDescription* y *Camera*, con sus métodos *getCameraDescription()*, *startCameraStreaming()* y

stopCameraStreaming(). La traducción del lenguaje *slice* a la interfaz *Ice* en lenguaje *C++* ha creado una clase con métodos abstractos que deben ser implementados. c*ameraserver* lo realiza definiendo la clase *CameraI* que hereda y extiende clase *Camera* y especifica los algoritmos de cada método, como muestra el Listado 3.5.

```
namespace cameraserver{
  class CameraI: virtual public jderobot::Camera {
  public:
…
    virtual jderobot::CameraDescriptionPtr getCameraDescription(const
Ice::Current& c){
…
    virtual std::string startCameraStreaming(const Ice::Current& c)
…
    virtual void stopCameraStreaming(const Ice::Current& c)
```

***Listado 3.5**: Selección línea de código cameraserver.cpp implementación de métodos de interfaz camera*

Por otro lado, ambos programas se comunican a través de una red usando el protocolo TCP. Los parámetros de conexión se han especificado en ficheros de configuración (List. 3.8) asociados a cada componente, tal y como es de esperar en las buenas prácticas de programación de componentes en *jderobot*. El servidor indica que sus servicios serán accesibles por TCP (Lis. 3.6) y el cliente intenta localizar la instancia con identidad *SimplePrinter* servida por servidor en la misma dirección (Lis. 3.7).

```
Ice::ObjectPrx obj = context().communicator()-
>propertyToProxy("CameraSrv.TopicManager");
```

***Listado 3.6**: Publicación de servicio cameraserver mediante clase auxilias de librería jderobot, Context.*

```
jderobot::CameraPrx cprx =
jderobot::CameraPrx::checkedCast(base);
```

***Listado 3.7**: Localización del servicio en cameraview.*

```
CameraSrv.Endpoints=default -h 0.0.0.0 -p 9999
# client/ erver mode
CameraSrv.DefaultMode=1
#cameras configuration
CameraSrv.Ncameras=1
#camera 0
CameraSrv.Camera.0.Name=cameraA
CameraSrv.Camera.0.ShortDescription=Camera pluged to /dev/video0
CameraSrv.Camera.0.StreamingUri=rtsp://192.168.1.15:8080/test.sdp
CameraSrv.Camera.0.Uri=v4l:///dev/video2
CameraSrv.Camera.0.FramerateN=25
CameraSrv.Camera.0.FramerateD=1
CameraSrv.Camera.0.ImageWidth=320
CameraSrv.Camera.0.ImageHeight=240
CameraSrv.Camera.0.Format=RGB888
#camera 1
```

```
CameraSrv.Camera.1.Name=cameraB
CameraSrv.Camera.1.ShortDescription=Camera simulated from a video
CameraSrv.Camera.1.Uri=http://www.facethewind.com/videos/may29_01.mpg
CameraSrv.Camera.1.FramerateN=15
CameraSrv.Camera.1.FramerateD=1
CameraSrv.Camera.1.ImageWidth=320
CameraSrv.Camera.1.ImageHeight=240
CameraSrv.Camera.1.Format=RGB888
```

***Listado 3.8(a)**: Fichero de configuración cameraserver*

```
Cameraview.Camera.Proxy=cameraA:tcp -h 127.0.0.1 -p 9999
```

***Listado 3.8(b)**: Fichero de configuración cameraview*

Observamos en los listados 3.8 como el parámetro *CameraSrv.Endpoints* de *cameraserver* y el parámetro *Cameraview.Camera.Proxy* de *cameraview* deben estar correspondidos en dirección y en puerto TCP. En este ejemplo, la configuración del servidor no hace explícita la dirección IP del nodo servidor y con el valor *0.0.0.0* se refiere implícitamente a la IP que tenga dicho computador. En el caso de *cameraview* el valor *127.0.0.1* asume que *cameraserver* se está ejecutando en el mismo nodo.

Para acabar con la sección, describiremos la organización de información, distribuida en diferentes directorios, del proyecto *jderobot 5.0* a fecha de redacción de esta memoria, sirviendo como referencia para localizar los diferentes componentes, interfaces y configuraciones:

- ***interfaces***: Contiene una colección de interfaces Ice que se implementan en diferentes componentes. Por ejemplo tenemos interfaces que describen el API de cámaras, lasers, gps, o incluso algoritmos.
- ***libs***: Contiene librerías que implementan los detalles de la arquitectura software Orca, como elementos de logging, depuración o configuración.
- ***components***: Contiene una colección de componentes que implementan e utilizan los interfaces comentados anteriormente. Entre otros tenemos servidores de imágenes, servidores de datos sensoriales (reales o simulados desde Stage) o incluso componentes que implementan navegación local.
- ***utils***: Contiene librerías con utilidades, como un grabador de logs o herramientas para describir un componente mediante un meta-lenguaje que posteriormente permite generar código de manera automática.
- ***examples***: Contiene una serie de ejemplos montados con varios componentes.
- ***hydroXXX***: Contienen las librerías hydro, que son un conjunto de drivers y algoritmos usados posteriormente por los componentes. Entre otros, hay drivers para acceso a hardware (lasers, gps, camaras,...).

3.5 Librería GTK+

GTK+ es la librería de creación de entornos gráficos que principalmente se ha utilizado durante el desarrollo de los primeros componentes de *jderobot* 5.0 programados en lenguaje *C++*. Por ello, y por el hecho de que también se empleará

para la creación de las interfaces de la solución aportada por este TFM, se incluye en esta sección de la memoria donde se describirán sus principales aspectos.

La librería *GTK+* está desarrollada bajo la licencia *GNU Library General Public License (GNU LGPL)*, la cual proporciona la libertad de desarrollar aplicaciones abiertas como es el caso de *jderobot*. Cuenta con una arquitectura basada en *C* orientada a objetos y permite su utilización desde múltiples lenguajes tales como *C++, Objective-C, Guile/Scheme, Perl, Python, TOM, Ada95, Free Pascal* y *Eiffel*. También es una librería multiplataforma soportada en sistemas operativos tipo UNIX y Windows.

La implementación de *GTK+* depende a su vez de otra serie de librerías que se enumerarán a continuación:

- ***GLib**:* Es una librería gráfica de propósito general no específica para la construcción de interfaces gráficas de usuario. Proporciona fundamentalmente definición de tipos de datos útiles, macros, conversores de formatos, tratamiento de Springs, manejo de ficheros, etc.
- ***GObject**:* Es otra librería para la definición de colecciones de tipos, como tipos de objetos o sistemas de señales.
- ***GIO***: Librería para la simplificación en el uso de sistemas de fichero *VFS*, incluyendo abstracciones de ficheros, unidades, volúmenes, streams IO y buses de comunicación por red.
- ***Cairo***: Es una interfaz gráfica 2D que soporta múltiples tipos de dispositivos de salida.
- ***Pango***: Incluye funcionalidades para el tratamiento de formatos de texto internacional.
- ***ATK:*** es un juego de herramientas que proporcionan interfaces genéricas para acceder e trabajar con interfaces gráficas de usuario. Por ejemplo, incluye funcionalidades para la lectura de texto en interfaces gráficas de usuario. Es un componente de accesibilidad en aplicaciones.
- ***GdkPixbuf***: Esta pequeña librería permite insertar objetos de imagen en las interfaces gráficas.
- ***GDK***: es una abstracción de *GTK+* que proporciona facilidades para la representación de ventanas en el sistema X11, Windows y el *framebuffer* de Linux.
- ***GTK+***: La propia librería incluye *widgets*, esto son, componentes de interfaz gráfica como *#GtkButton* o *#GtkTextView*.

Por otro lado, para facilitar el trabajo de diseño de la interfaz gráfica con la librería de clases de *GTK+*, se puede emplear la herramienta *Glade*. Esta utilidad permite definir de modo gráfico la selección de *widgets* en la interfaz, la colocación de estos elementos sobre la ventana, la asignación de valores de formato o comportamiento en los atributos de los objetos y la asociación de eventos con funciones del código de aplicación que levantará la interfaz (Fig. 3.11).

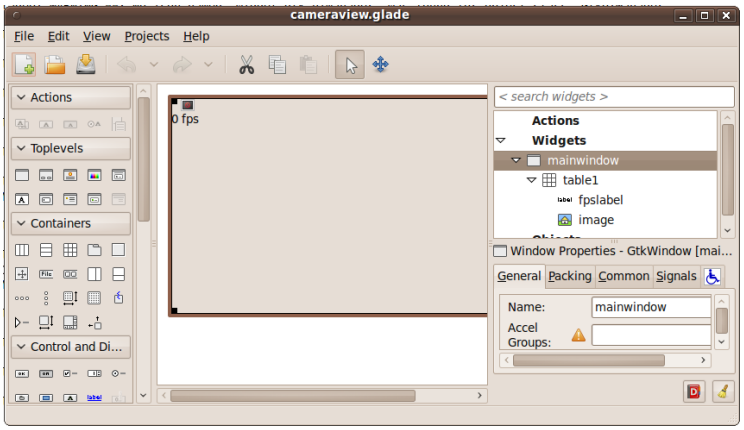


***Figura 3.11**: GUI de cameraview diseñada desde la herramienta Glade*

La definición de la interfaz gráfica descrita con *glade* se guarda en lenguaje XML. Desde el código de la aplicación, los objetos de esta interfaz se pueden instanciar y comenzar a usar desde el código mediante la clase *Gnome::Glade::Xml* de la librería *libglademm* para *C++* (List. 3.9).

```
Glib::RefPtr<Gnome::Glade::Xml> refXml;
Gtk::Image* gtkimage;
Gtk::Window* mainwindow;
Gtk::Label* fpslabel;
Gtk::Main gtkmain;
…
refXml = Gnome::Glade::Xml::create(gladepath);
refXml->get_widget("image", gtkimage);
refXml->get_widget("mainwindow",mainwindow);
refXml->get_widget("fpslabel",fpslabel);
```

***Listado 3.9**: Ejemplo instanciación de objetos GTK++ con librería libglademm.*

No obstante durante el desarrollo del entorno de prácticas que se abordará en el próximo capítulo del documento, se descubrieron algunas limitaciones en el uso de *glade* con ciertas clases de la herencia *GTK*. En concreto, la clase *Gnome::Canvas::Canvas* empleada para la representación 2D del mapa por donde circula el robot, no puede ser definida y posteriormente creado su objeto del modo en que se acaba de explicar para el resto.

Una observación interesante es la que se expone en *[jderobot 5 thesis, D. Lobato, 2010]* respecto al modo de ejecución del proceso gráfico: *GTK+* puede tener su propio flujo de ejecución, o bien integrarse en otro flujo y procesar sus eventos cuando nos interese por medio de una llamada. Esta última es la manera más simple, dado que los eventos se procesan en algún punto de la iteración de nuestro componente, manteniéndose el acceso al modelo completamente serializado, sólo una hebra accede a él. Esta variante tiene sus limitaciones, principalmente en el caso de que procesar los eventos de la vista resulte costoso computacionalmente, ya que el tiempo de procesado de dichos eventos se sumará al tiempo del procesamiento que se haga en cada iteración. La alternativa de crear un flujo de ejecución propio para la

vista que con *GTK+* resulta tan simple como crear una hebra y llamar a *gtk main()* en su flujo de ejecución.

El uso de esta librería en el desarrollo de componentes para *jderobot* supone una simplificación respecto al manejo gráfico de la versión 4.3 por el hecho de que en la versión antigua había que integrar el funcionamiento de las ventanas con los esquemas *Graphics_xform* o *Graphics_gtk*. En la nueva arquitectura, cada aplicación será un proceso independiente y con la capacidad de programar su propia *GUI*.

Capítulo 4

Desarrollo entorno de prácticas

4.1 Diseño Global

Tomando como partida la descripción de los subobjetivos presentados en el capítulo 2, así como los requisitos R1 a R5, se elaboró un diseño global de la aplicación docente como entorno de prácticas de programación de robots en la plataforma 5.0. La Figura 4.1 muestra un diagrama de alto nivel con la solución planteada, que se analizará y justificará a continuación.

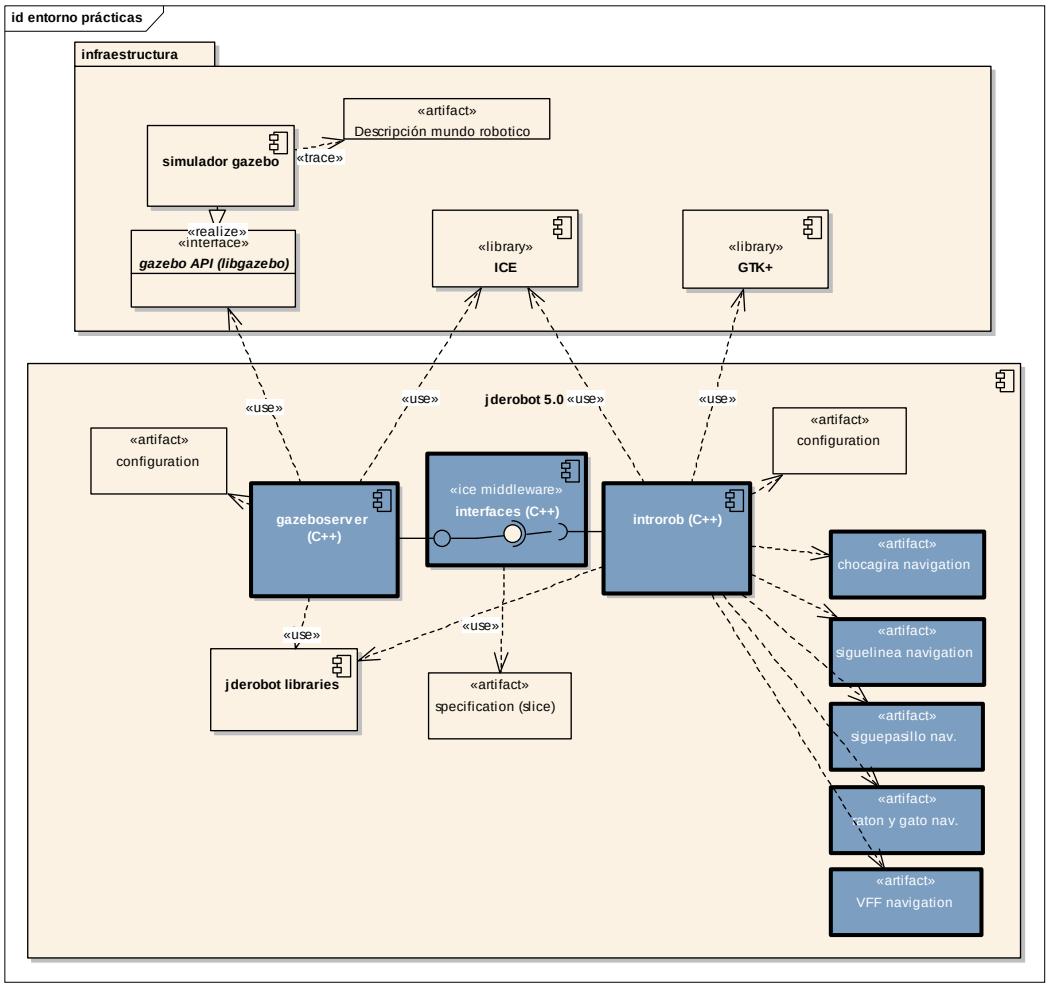


***Figura 4.1**: Diagrama componentes entorno de prácticas jderobot 5.0*

En el diagrama anterior se representan por separado los componentes y librerías de la plataforma *jderobot* de la infraestructura utilizada para el desarrollo del entorno de

prácticas. En el grupo de infraestructura se seguirá empleando el software de simulación *gazebo* como quedó recogido en la especificación de objetivos, el cual será accedido, al igual que en el driver *gazebo* de *jderobot* 4.3, mediante su *API* de programación *libgazebo* a la que nos referiremos más adelante. Para la compilación y uso de las interfaces entre componentes a través del *middleware Ice*, también es necesario la utilización de la *API Ice* por lo que se deja representado en el diagrama como un componente adicional de la infraestructura. Finalmente, se empleará *GTK+* como librería de desarrollo de interfaces gráficas para nuestro entorno. Esta decisión se basa en el uso *GTK+* para el resto de aplicaciones con interfaz gráfica implementadas hasta la fecha en la versión 5.0.

Para el desarrollo de la lógica del entorno, se ha tomado como referencia el diseño empleado en la versión 4.3, considerándose adecuado independizar el acceso e interacción con el software *gazebo* del componente para programar y verificar el funcionamiento de las aplicaciones que realizarán los alumnos. Esta segregación de funcionalidad también se justifica con la necesidad que tendrán otras aplicaciones futuras para acceder a las interfaces de sensores y actuadores de robots simulados en *gazebo*, a través de un componente específico de la plataforma. A este componente se le ha denominado *gazeboserver*. Su esquema de funcionamiento será el de presentar la información de todos los sensores y recibir valores para los actuadores de varios robots mediante el *middleware Ice*. Para ello habrán definido interfaces generales con el lenguaje *slice* que incluso puedan ser compartidas con otros componentes servidores, como es el caso de *gazeboserver* y la interfaz *camera* usada. *gazeboserver* implementará accesos al simulador con la *API libgazebo* programada en lenguaje *C* y única disponible. En el caso del sensor cámara, se ha reusado el código del componente *cameraserver* basado en hebras de ejecución para el procesamiento de cada *streaming* de video concurrente. Esto da la posibilidad a *gazeboserver* de enviar varias señales de video para cada una de las cámaras simuladas en *gazebo.*

En cuanto al desarrollo del componente para la programación de la lógica de control de robot, por ejemplo, con algoritmos de navegación clásica, también se ha tomado como referencia el diseño del esquema *introrob* de la versión 4.3 y se ha tratado de mantener en la solución 5.0. De hecho, se ha considerado conveniente un modelo similar con el objetivo de que programadores de robots acostumbrados al entorno *introrob* de la versión estable no perciban demasiados cambios en la manera de trabajar sobre la 5.0. Quizás solo se observe el cambio del lenguaje *C* al *C++* y la orientación a objetos en el código. Este componente también implementa una interfaz gráfica basada en *GTK+* y apoya su diseño en la herramienta *glade*. La interfaz gráfica (Fig. 4.2) permite consultar la información recibida del *gazeboserver* sobre el estado del robot, así como realizar acciones de control sobre el mismo. Dichas acciones serán teleoperadas desde los mandos de la interfaz o controladas autónomamente por la inteligencia programada en *introrob.*

Introrob ha sido programado como esqueleto del entorno de prácticas y ayuda al programador en tres sentidos: control de la ejecución de su código visualizando los valores de cada sensor, independencia del acceso a los elementos de infraestructura o a otros componentes *jderobot* a través del *middleware* de comunicaciones y acceso sencillo a los actuadores y sensores mediante atributos de la clase que encapsula el código programado.

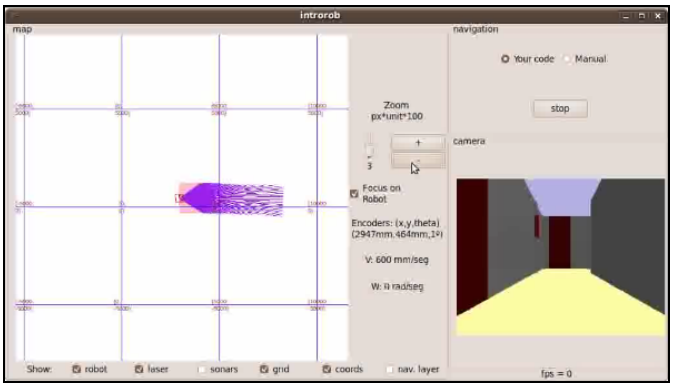


***Figura 4.2**: Interfaz introrob 5.0*

Gazeboserver e *introrob* están programados completamente en *C++* aunque ya se ha aclarado que los servicios expuestos por el componente servidor podrán ser utilizados por futuras aplicaciones implementadas en lenguajes compatibles con *Ice.* Estos dos componentes tendrán en sus ficheros de configuración respectivos, los parámetros de funcionamiento y establecimiento de la comunicación. En el caso de *gazeboserver*, se describe el nombre y localización de las interfaces y cada uno de los sensores y actuadores publicados. En la configuración de *introrob* también se describirá la cadena de conexión a estas.

Como parte de la solución entregada en el TFM, también existirán implementaciones de los diferentes algoritmos de navegación, basados en algoritmos reactivos, que habitualmente se realizan en la asignatura de Robótica Master. En el diagrama de la figura 4.1 se muestran como elementos *"artifact"* del componente introrob dado que en la realidad, como se verá en la sección de desarrollo de *introrob* de este capítulo, el código de cada uno de los algoritmos queda guardado en una de las clases del componente. Variar el comportamiento del robot implicará sustituir la definición de la clase que programará el alumno.

4.2 Componente Driver Gazeboserver

4.2.1 Introducción

Se ha introducido en la sección anterior, que el propósito de este componente es análogo al dispuesto por el *driver gazebo* en la versión 4.3 de *introrob*. La principal diferencia consiste en su implementación de acuerdo a las reglas arquitectónicas fijadas en la 5.0. No obstante, y aunque el desarrollo se ha hecho en lenguaje *C++*, el método de acceso al software de simulación también se basa en el manejo del *API C libgazebo.* Esta librería utiliza mecanismos *IPC*[8] para la interacción con *gazebo* aunque estos detalles quedan ocultos en la implementación de la *API*, la cual proporciona, fundamentalmente:

- ***Definición de tipos de datos***: para cada uno de los sensores y actuadores de los robots soportados, se establece una definición de tipos de datos (List. 4.1) que serán utilizados en la invocación o respuesta de las funciones del *API*.

[8] http://playerstage.sourceforge.net/doc/Gazebo-manual-0.5-html/libgazebo_usage.html

```
/// Common data header
typedef struct gz_data
{
  /// Interface version number
  int version;
  /// Allocation size
  size_t size;
  /// Type of model that owns this interface
  char model_type[GAZEBO_MAX_MODEL_TYPE];
  /// ID of the model that owns this interface
  int model_id;
  /// ID of the parent model
  int parent_model_id;
} gz_data_t;
```

***Listado 4.1**: Ejemplo de tipo de datos del API libgazebo*

- ***Funciones para sensores y actuadores***: Colección de funciones y procedimientos para trabajar con cada clase de sensor y actuador del robot. Acciones como iniciar, parar, bloquear el uso, transmitir valor o realizar lectura de medición, son hechos con estas llamadas. Entre los argumentos de las mismas se debe identificar al robot del entorno, ya que como se ha explicado en el capítulo de infraestructura, *gabezo* permite la simulación de varios robots en un mismo *mundo*.

Se ha tenido en cuenta que, al igual que el *driver* de la versión anterior recoge de un fichero de configuración (List. 4.2) la identificación del robot y los dispositivos disponibles en el diseño de *gazeboserver* también soporte esta descripción dentro del fichero de configuración general *gazeboserver.cfg*.

```
#the device names in the jdec configuration file
#must be the same as in the gazebo configuration file

driver gazebo
provides laser laser2
provides motors robot2
provides encoders robot2
# coordinate adjustment for regular worlds,
# worlds with ground plane in Z=0
# initial position X(mm) Y(mm) Theta(deg)
initial_position -6500 -6500 45
provides sonars robot2
provides colorA camera1
provides pantilt camera1
# varcolorA, varcolorB, varcolorC, varcolorD are fully
supported
```

***Listado 4.2**:Ejemplo de configuración para driver gazebo 4.3*

Para finalizar la introducción, se enumera la relación de sensores y actuadores generales de un robot que deben ser cubiertos por la funcionalidad del *driver*: *motors, ptmotors, sonars, ptencoders, laser* y *encoders*.

4.2.2 Desarrollo del componente

En el desarrollo de *gazeboserver* como componente independiente y comunicado a través de interfaces *Ice*, una de las primeras cuestiones planteadas fue la selección del número de interfaces, así como el formato y su especificación *slice*.

Inicialmente se plantearon las dos alternativas siguientes:

- ***Una sola interfaz Ice***: soportando todas las operaciones con los dispositivos del robot simulado (Fig. 4.3).

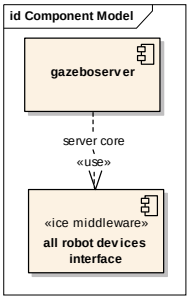


***Figura 4.3**: Modelo con una interfaz Ice para todos los dispositivos*

- ***Una interfaz por cada tipo de dispositivo**:* como se muestra en la figura 4.4, se definiría una interfaz distinta para cada tipo de sensor y actuador. Este modelo conlleva la implementación de una clase extendida para cada interfaz en el código de *gazeboserver*. Cada una de las clases especificará la lógica de los métodos abstractos creados por el *middleware*.

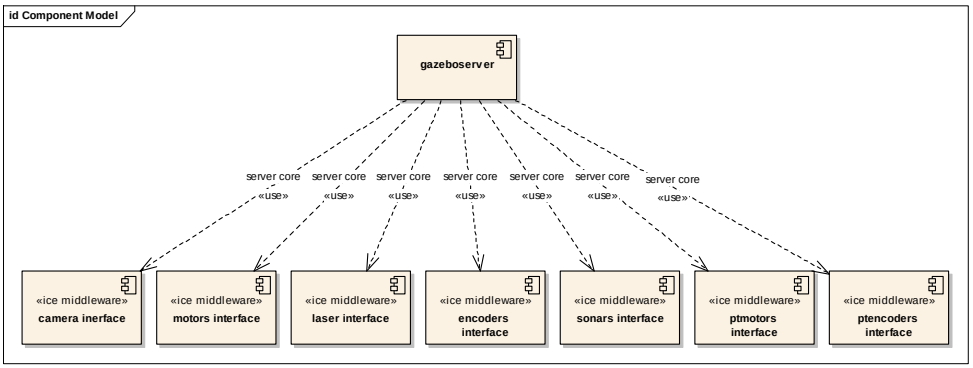


***Figura 4.4**: Modelo con una interfaz Ice por tipo de dispositivo robot*

Con el objetivo de hacer más modular la definición de cada tipo de interfaz y que estas puedan ser compartidas por otros componentes servidores de *jderobot* 5.0, se optó por el segundo modelo. Esta aproximación también da mayor flexibilidad, no solo de reusar definiciones de interfaces por componentes servidores sino por la posibilidad de que una aplicación cliente de la interfaz compartida podrá actuar como cliente de ambos servicios sin necesidad de adaptación.

La anterior consideración es probada desde la fase inicial de desarrollo de *gazeboserver* con el sensor de cámara, dado que ya existe una interfaz *camera* para la transmisión de *streaming de video* en la plataforma y que es empleada por los componentes *cameraserver y cameraview*. Es posible desarrollar *gazeboserver* de modo que sin necesidad de adaptación del software de *cameraview*, únicamente

modificando la especificación del servicio en el fichero de configuración del cliente, se pueda obtener la señal de video del robot *gazebo*. Esta organización se representa en el diagrama de la figura 4.5.

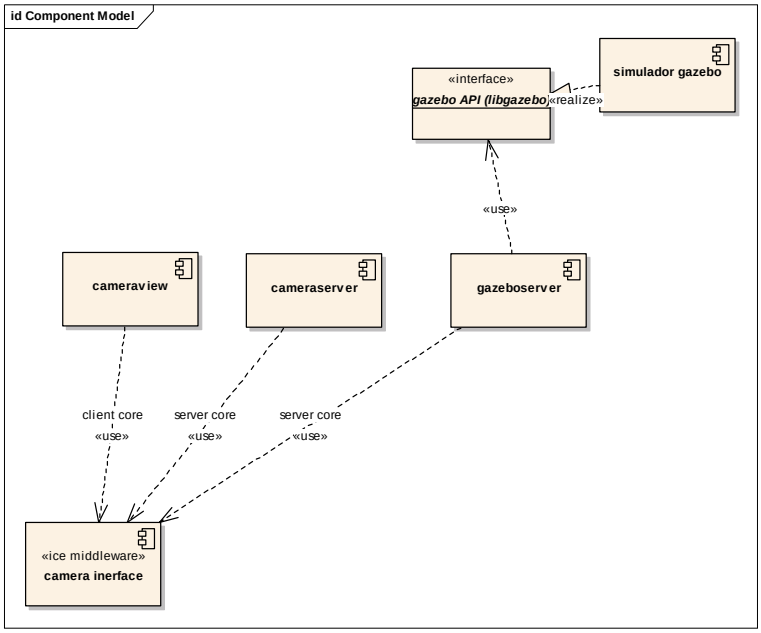


***Figura 4.5**: Integración de cameraview con cameraserver y gazeboserver mediante interfaz común camera.*

La definición de la interfaz *camera* con lenguaje *slice* que se incluyo en el Listado 3.4 del capítulo previo, ha sido la empleada por la implementación de *gazeboserver* para el soporte de la cámara del robot. Consta de dos secciones diferenciadas: una primera donde se declara la clase de objetos para la descripción de la señal de video *CameraDescription* y otra donde se describen los métodos de la interfaz que podrán ser invocados desde el cliente. *startCameraStreaming* y *stopCameraStreaming* serán empleados para activar y desactivar la transferencia de *streaming* de video mientras que *getCameraDescription* facilitará la información respecto a esta cámara. Esta especificación fue apta para el desarrollo de *gazeboserver* con la funcionalidad del sensor cámara, por lo que no fue necesario modificarla y afectar a los componentes *cameraserver y cameraview*.

En el resto de casos, sí fue necesario agregar a *jderobot* la especificación de nuevas interfaces para el resto de dispositivos como muestra el diagrama de la figura 4.6.

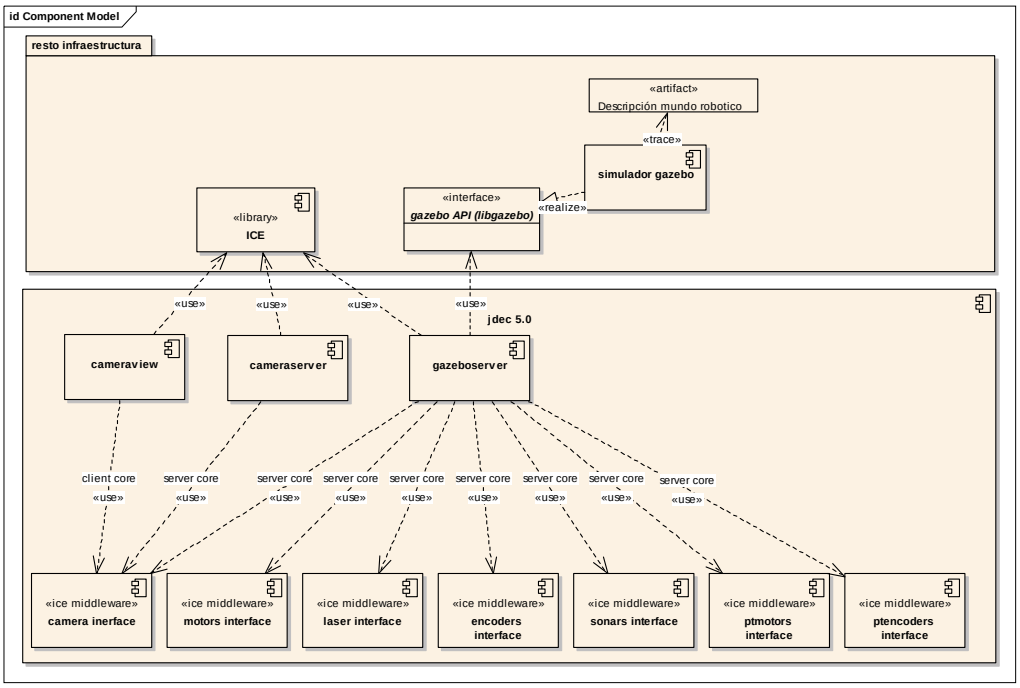


***Figura 4.6**:Diagrama con todas las interfaces definidas para gazeboserver.*

El listado 4.3 muestra un ejemplo de la interfaz para el láser. Define la interfaz *Laser* cuyo único método *getLaserData* es implementado en el código del componente. También define clase *LaserData* para manejar la información de lecturas de láser recibida por el método *getLaserData*. El tipo *slice IntSeq* asociado al atributo *distanceData* será transformado a un arreglo de enteros en su conversión a código *C++*.

```
#ifndef LASER_ICE
#define LASER_ICE
#include <jderobot/common.ice>
module jderobot{
        /* laser information */
  class LaserData
  {
    IntSeq distanceData;
    int numLaser;
 };
  /**
   * Interface to the Gazebo laser sensor.
   */
  interface Laser
  {
     Idempotent LaserData getLaserData();
  };
}; //module
```

***Listado 4.3**:Definición interfaz laser para gazeboserver*

En cuanto al flujo de tareas realizadas por el componente durante su arranque, se podría resumir con el siguiente diagrama de actividad (Fig. 4.7).

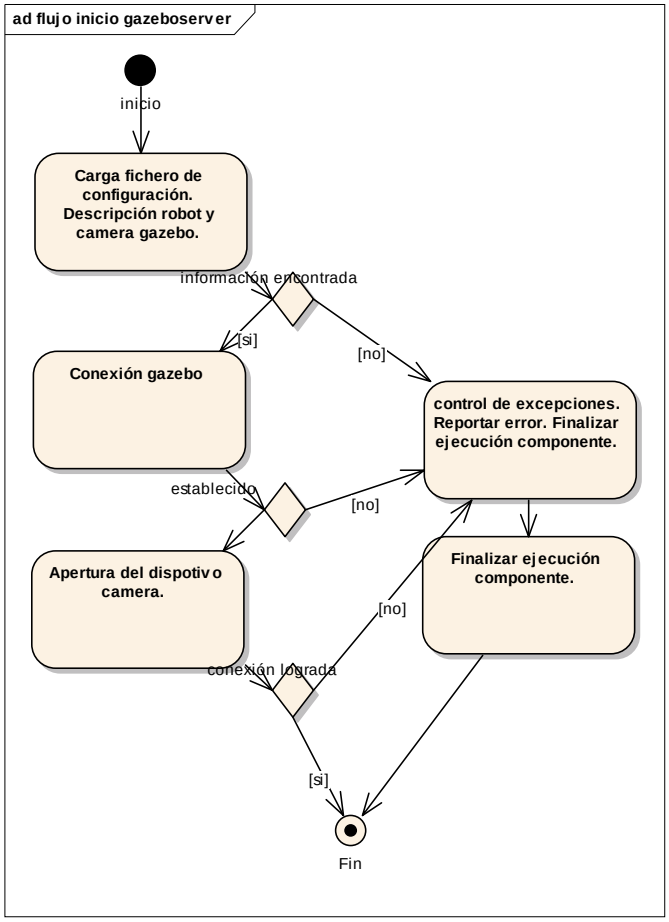


Figura 4.7: *Flujo de inicio de componente gazeboserver.*

Otra de las reglas tenidas en cuenta durante la implementación del componente ha sido el uso de algunas clases de librería desarrollada por David Lobato, para la nueva plataforma. En concreto, se emplearon las clases *jderobotice*::*Application* y *jderobotice*::*Component.* Estas se deben utilizar para poner en ejecución la lógica del componente con la creación de un contexto y control de excepciones de error. El modo en el que son empleadas se muestra en el código del listado 4.4. Este código corresponde a la función *main* de *gazeboserver*. *Application* pone en ejecución al objeto de *Component*.

```
int main(int argc, char** argv){
  gazeboserver::Component component;
  jderobotice::Application app(component);
  return app.jderobotMain(argc,argv);
}
```

Listado 4.4*: Instanciación del objeto component en el arranque de gaseboserver*

Component a su vez, en su redefinición del método constructor, instancia cada uno de los objetos de las interfaces que soporta *gazeboserver*, como se puede ver en el

listado 4.5. Este esquema de arranque se aplicará al desarrollo de *introrob* aunque no se volverá a mencionar en su apartado.

```
class Component: public jderobotice::Component{
  public:
    Component()
      :jderobotice::Component("GazeboServer"),
cameras(0), motors1(0), laser1(0), encoders1(0),
ptmotors1(0), ptencoders1(0), sonars1(0) {}
```

***Listado 4.5**: Cabecera de la definición del constructor Component en el código de gaseboserver*

Otro aspecto del que ya se ha hablado en la memoria es la necesidad de definir clases que extiendan las creadas por *Ice* para cada interfaz, especificando la lógica de los métodos que declara. El siguiente diagrama da una visión global de todas las manejadas en *gazeboserver* y cuales han sido los atributos y métodos añadidos, además de los métodos públicos correspondientes a las operaciones de la interfaz.

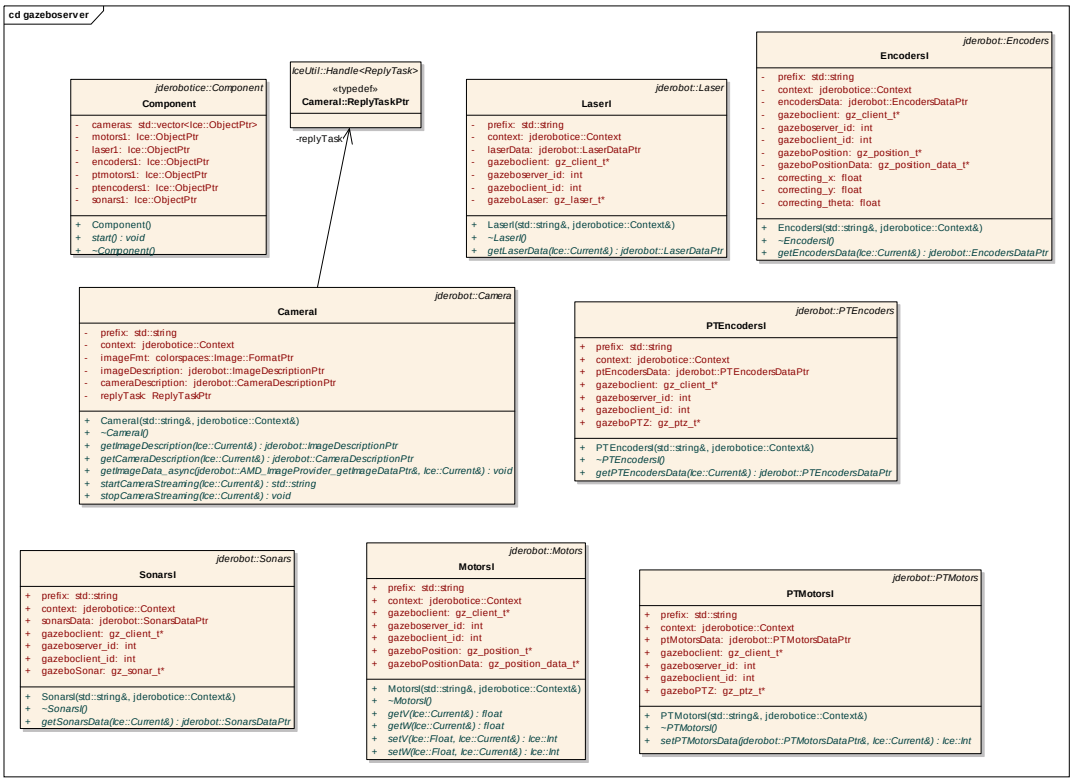


***Figura 4.8**: Diagrama de clases para interfaces implementadas en gazeboserver*

En el lista 4.6 se ha copiado el texto integro de la implementación de la clase *PTEncodersI* con el que se pretende dar una muestra al lector de cómo el acceso a los servicios del simulador *gazebo* son accedidos mediante las funciones del *API*, las cuales comienzan por la nomenclatura "*gz_*". Obsérvese que el método de lectura del sensor realiza una conversión del sistema radial al sexagesimal por igualdad con el devuelto por el *driver* de la versión anterior. La intención es facilitar la portación de código de aplicaciones realizadas en 4.3 a la actual. Transformaciones del mismo tipo

han sido realizadas en métodos de otras clases interfaces como por ejemplo la conversión a rango de 180º del arreglo devuelto por la lectura de láser.

```
//PTENCODERS
  class PTEncodersI: virtual public jderobot::PTEncoders {
       public:
              PTEncodersI(std::string& propertyPrefix, const
jderobotice::Context& context) : prefix(propertyPrefix),context(context)
     {
       Ice::PropertiesPtr prop = context.properties();
       gazeboserver_id=prop->getProperty(prefix+"ServerId");
       gazeboclient_id=prop->getProperty(prefix+"ClientId");
       robotname=prop->getProperty(prefix+"RobotName"):

       // Create a client object
       gazeboclient = gz_client_alloc();

       // Connect to the server
       if (gz_client_connect_wait(gazeboclient,gazeboserver_id,gazeboclient_id)
!= 0) {
          printf("ERROR: Connecting to Gazebo server.\n");
       }
       gazeboPTZ = gz_ptz_alloc();
       if (gz_ptz_open (gazeboPTZ, gazeboclient, robotname) != 0)
       {
          fprintf (stderr, "Error opening \"%s\" ptz\n",robotname);
          //return (-1);
       }
      }
      virtual ~PTEncodersI(){};
      virtual jderobot::PTEncodersDataPtr getPTEncodersData(const Ice::Current&){
          //waiting for next gazebo camera update
          if ( gz_client_wait(gazeboclient) != 0) {
              printf("Error waiting for Gazebo server\n");
          }
          if(!gazeboPTZ){
              printf("Gazebo PTENCODERS model not opened\n");
          }
          gz_ptz_lock(gazeboPTZ,1);
          ptEncodersData->panAngle=-1 * gazeboPTZ->data->pan * RADTODEG;
          ptEncodersData->tiltAngle= -1 * gazeboPTZ->data->tilt * RADTODEG;
          gz_ptz_unlock(gazeboPTZ);

          return ptEncodersData;
       };
       std::string prefix, robotname;
       jderobotice::Context context;
       jderobot::PTEncodersDataPtr ptEncodersData;
       gz_client_t *gazeboclient;
       int gazeboserver_id;
       int gazeboclient_id;
       gz_ptz_t * gazeboPTZ;
};
```

***Listado 4.6**:Fragmento gazeboserver.cpp con implementación de la clase PTEncodersI*

4.2.3 Pruebas de verificación

Durante una primera fase del desarrollo del componente se validó la correcta implementación del sensor cámara. Se aprovechó el componente existente *cameraview* con este propósito. Se obtuvieron resultados de funcionamiento satisfactorios tras una etapa de puesta a punto del código. Se verificó un buen rendimiento del software, con tasas de hasta 40 *frames* por segundo, pese a que el computador utilizado ha sido durante todo el TFM, una instancia de computador virtual.

También se validaron aspectos de servicio concurrente a varios clientes mediante el lanzamiento de varias ejecuciones de *cameraview*. Se obtuvo un resultado correcto, como se muestra en la siguiente figura 4.9.

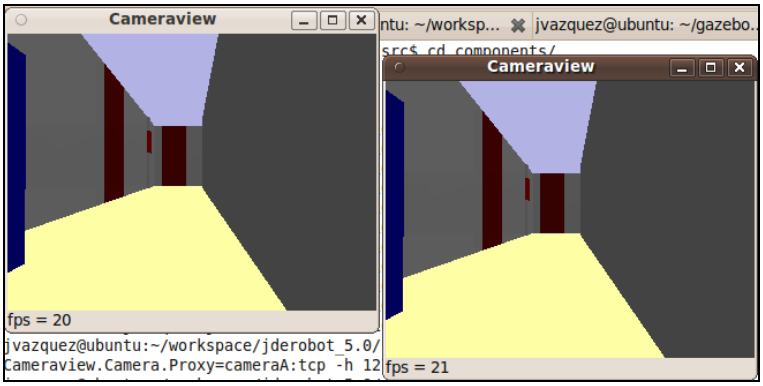


Figura 4.9: Visualización cámara gazeboserver
con el uso de dos instancias camaraview en ejecución

Para la verificación de los sensores láser, encoders, así como el actuador motors, se tuvo que esperar a completar el desarrollo del componente *introrob* que complementa este entorno de enseñanza (Fig. 4.10).

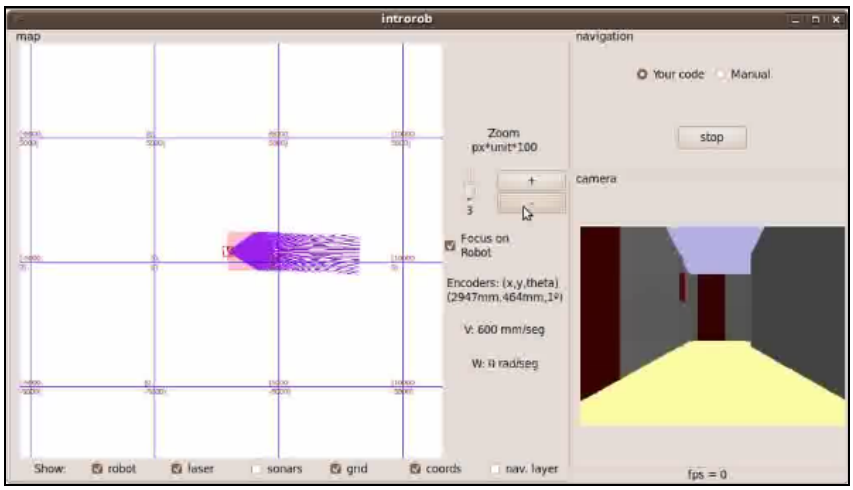


Figura 4.10: *Imagen componente introrob en ejecución.*
Comunicación con gazeboserver

No es objetivo de esta sección detallar la relación de elementos de la interfaz gráfica ni la lógica que hay detrás del componente *introrob*, ya que se abordará en la siguiente sección. No obstante, en la figura anterior se muestra una captura en la que se observa distinta información sensorial que ha sido recibida a través de la interfaz con *gazeboserver*: datos sobre posicionamiento (*encoders*), valores de velocidad lineal y angular (*motors*), la representación de los láser (*lasers*) y la imagen de la cámara (*camera*).

4.3 Componente Introrob

4.3.1 Introducción

Se ha anticipado en la presentación de la solución realizada en el TFM el desarrollo del componente *introrob* como un esqueleto donde un programador de robots pueda especificar la inteligencia de la máquina sin tener que preocuparse por otros detalles de la plataforma como son:

- Controlar de la ejecución de su código visualizando los valores de cada sensor.
- Acceder a los elementos de infraestructura o a otros componentes *jderobot* a través del *middleware* de comunicaciones
- Comprender el formato y complejidad de cada sensor mediante el uso de funciones particulares.

Introrob tiene la responsabilidad de realizar este trabajo por el programador y encapsular su especificación de la forma las clara posible.

Para la consecución del primer punto, se decidió implementar de una interfaz gráfica que arrancase al inicio del componente y mostrase un cuadro de mandos con vista de la información de sensores y control del entorno. A primera vista y para satisfacer las funcionalidades existentes en la versión 4.3 respecto al control gráfico del entorno, son necesarios los siguientes elementos:

- Cámara del robot
- Posición y orientación del robot (encoders), mostradas como representación gráfica 2D planta (vista cenital).
- Lecturas de proximidad de obstáculos con sensor láser, representadas igualmente sobre la vista planta 2D.
- Operador de movimiento del robot en velocidad y sentidos de giro.
- Selector de escala de zoom en canvas 2D y desplazador sobre esta área mediante barras
- Selector modo de funcionamiento teleoperador o autónomo.

Esta interfaz deberá permitir dos modos de funcionamiento: *manual y automático.* En el modo teleoperado (*manual*), el usuario de la aplicación puede dirigir al robot con el joystick de selección de velocidad lineal y radial. En el modo automático, la aplicación ejecutará iterativamente el código programado por el usuario.

Para satisfacer el segundo punto de abstracción de los detalles técnicos en el acceso a la infraestructura y control de la comunicación con otros componentes, el diseño de la aplicación separará en diferentes clases y métodos las actividades de control del componente con la propia de programación del robot.

Finalmente, para facilitar un acceso sencillo a los sensores y actuadores, se definirán atributos en la clase destinada al código de control, que representen los valores de estos dispositivos. La asignación de nuevos valores a los atributos relacionados con actuadores (*motors y ptmotors*) serán entendidos como órdenes al robot.

4.3.2 Desarrollo del componente

Para el desarrollo del componente, se realizó un diseño basado en clases, donde a grandes rasgos, la funcionalidad queda repartida del siguiente modo:

- *introrob*: clase principal con la que arranca el programa, procesa los datos de configuración, establece la conexión con los servicios de *gazeboserver*, arranca la interfaz gráfica (objeto de la clase *introrobgui*) y controla las iteraciones de procesamiento, aplicando la lógica del modo manual o del modo automático (objeto de la clase *navega*).
- *introrobgui*: encapsula la funcionalidad de control de la interfaz gráfica del programa. Para el manejo de la representación gráfica mostrada en el elemento canvas, se invocan a los métodos de la clase auxiliar *introrobcanvas*.
- *introrobcanvas*: Dada la especial complejidad de representación gráfica sobre el *grid*, donde las diferentes lecturas de los dispositivos encoders, lasers y sonars deben mostrarse junto con el robot y el grid de referencia, se aisla esta funcionalidad dentro de esta clase. i*ntrorobcanvas* ofrece a *introrobgui* un conjunto de métodos suficientes para operar la activación y desactivación de capas que se pueden mostrar.
- *navega*: encapsula la lógica de navegación que será empleada cuando el programa esté en modo autónomo. Es la clase donde el programador de la inteligencia del robot escribirá sus líneas de código.

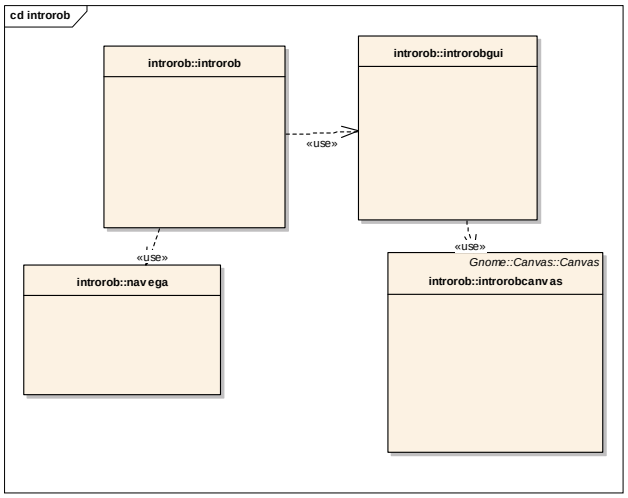


Figura 4.11: *Diagrama de clases para el componente introrob.*

introrobcanvas hereda de la clase *Gnome::Canvas*. Como se comentó en la sección de infraestructura dedicada a la librería *GTK+* y al uso de la herramienta *glade* para

el diseño de la interfaz, esta clase no es soportada por *glade* a la hora de lograr integrarla desde el código *C++* de la aplicación. El tratamiento de este objeto se hace de una forma distinta. En el listado 4.7 es un ejemplo del modo en el que se instancian los objetos *widget* de *GTK+* con la libraría *libglademm*. En el listado siguiente 4.8 se muestra la sección del código del constructor *introrobgui* donde se crea el elemento canvas.

```
refXml->get_widget("showRobot",showRobot);
refXml->get_widget("showLaser",showLaser);
refXml->get_widget("showSonars",showSonars);
refXml->get_widget("showGrid",showGrid);
```

***Listado 4.7**: Ejemplo creación objetos GTK+ con glade*

```
Gnome::Canvas::init();
canvas = new introrob::introrobcanvas();
```

***Listado 4.8**: creación de objeto introrobcanvas*

Para la representación gráfica 2D se hizo una distinción entre tipos de elementos mostrados y se incorporó la funcionalidad de poder activarlos u ocultarlos mediante una lista de *checkboxes* (Fig. 4.12). De izquierda a derecha, estos elementos son: robot, lecturas láser 180º, lecturas sonar, *grid* de coordenadas, coordenadas en los cruces del *grid* y rastro de navegación.

***Figura 4.12**: Selección de elementos mostrados sobre el canvas*

A alto nivel, el siguiente diagrama de flujo describe el funcionamiento de la aplicación (Fig. 4.13). Dicho control de las distintas tareas de inicialización del sistema, establecimiento de la comunicación con *gazeboserver* a través del *middleware* y control interacción con el usuario a través de la *GUI* manejada en *introrobgui* es la función principal del código *introrob.cpp*:

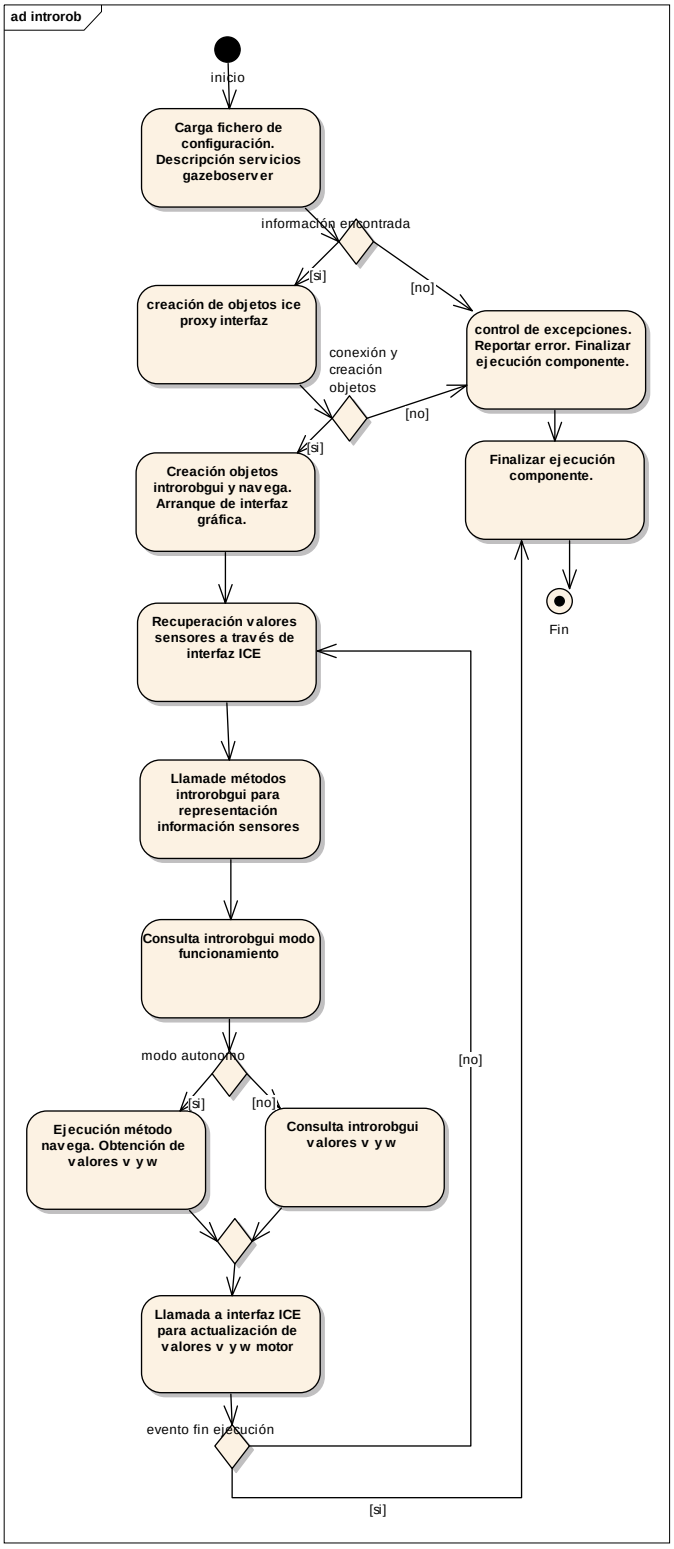


***Figura 4.13**: Esquema de funcionamiento introrob*

Durante la etapa inicial de establecimiento de conexiones con el servidor, se crean los objetos *Ice Proxy* y se busca la denominación de estos canales en el fichero de configuración de la aplicación (List. 4.9). Posteriormente ya estarán disponibles para realizar la comunicación.

```
//contact to camera interface
    Ice::ObjectPrx baseCamera = ic->propertyToProxy("Introrob.Camera.Proxy");
    if (0==baseCamera)
      throw "Could not create proxy with Camera";

    /*cast to CameraPrx*/
    jderobot::CameraPrx cprx = jderobot::CameraPrx::checkedCast(baseCamera);
    if (0==cprx)
      throw "Invalid proxy Introrob.Camera.Proxy";

//contact to motors interface
    Ice::ObjectPrx baseMotors = ic->propertyToProxy("Introrob.Motors.Proxy");
    if (0==baseMotors)
      throw "Could not create proxy with motors";

    /*cast to CameraPrx*/
    jderobot::MotorsPrx mprx = jderobot::MotorsPrx::checkedCast(baseMotors);
    if (0==mprx)
      throw "Invalid proxy Introrob.Motors.Proxy";

//contact to laser interface
    Ice::ObjectPrx baseLaser = ic->propertyToProxy("Introrob.Laser.Proxy");
    if (0==baseLaser)
      throw "Could not create proxy with laser";

    /*cast to CameraPrx*/
    jderobot::LaserPrx lprx = jderobot::LaserPrx::checkedCast(baseLaser);
    if (0==lprx)
      throw "Invalid proxy Introrob.Laser.Proxy";
```

***Listado 4.9**: inicialización de objetos de comunicación Proxy con la interfaz Ice*

En el fragmento de código del listado 4.10, *introrob* implementa un bucle en el que se repite la secuencia de actualizar el valor de los sensores con llamadas a los servicios de *gazeboserver*, ejecución de la lógica de control del robot descrita en la clase *navega*, en caso de que *introrob* esté funcionando en modo autónomo o recuperación del valor de los actuadores de la interfaz gráfica si el modo es manual y nueva invocación a los servicios de *gazeboserver* para enviar las órdenes a los actuadores.

```
while(introrobgui_obj.isVisible() && ! introrobgui_obj.isClosed()){
      jderobot::ImageDataPtr data = cprx->getImageData();
.
.
.
       if (introrobgui_obj.isYourcode() && ! introrobgui_obj.isStopped()){
.
.
.
       navega_obj.navegacion(data->description->width,
                             data->description->height,
```

```
                                &(data->pixelData[0]),
                                laserData,
                                encodersData);
                mprx->setV(navega_obj.getV());
                mprx->setW(navega_obj.getW());
```

***Listado 4.10**: Llamada a método navegación de clase navega para ejecución de la lógica programada en introrob*

La lógica de control programada para el robot estará ubicada en la implementación del método *navegación* de la clase *navega* (fichero *navega.cpp*). Conforme a lo que se muestra en el listado 4.11, dicho método recibe por parámetro la información de los sensores cámara (*width, height* e *imageData*), láser (*laserData*) y encoders *(encodersData*). La actualización de la velocidad lineal y angular del robot deberán ser especificados como valor de los atributos de la clase *v* y *w* respectivamente.

```
void navegacion( const int width, const int height,  unsigned char *
imageData, jderobot::LaserDataPtr laserData, jderobot::EncodersDataPtr
encodersData );

float v, w;
```

***Listado 4.11**: declaración cabecera de método navegación en clase navega (navega.cpp). Atributos v y w para control de motores*

Por último, se recoge un fragmento de la implementación del método referido antes (Lis. 4.12) en la implementación de la práctica *siguelinea*. La clase *navega* ha sido extendida con dos métodos privados adicionales *sensor_camera_red_line_aligned* y *reaccion_camera_red_line_aligned* para separar las lógicas de sensado de las de acción del robot.

```
void navega::navegacion( const int width, const int height,  unsigned char *
imageData, jderobot::LaserDataPtr laserData, jderobot::E
ncodersDataPtr encodersData )
  {
        printf("[alto,ancho]: [%d,%d]\n",height,width);
        sensor_camera_red_line_aligned(width,height,imageData);
        reaccion_camera_red_line_aligned();
  }
```

***Listado 4.12**: implementación navegación para práctica siguelinea*

4.3.3 Pruebas de verificación

La validación del componente en el modo manual fue relativamente sencilla, sirviendo esta fase para depurar la corrección de algunos errores en la integración de los botones de la *GUI* con las funciones correspondientes del código. En la figura 4.14.a, se muestra la captura de la interfaz mientras se teleoperaba el robot sobre el

entorno simulado utilizado como entorno simulado, el correspondiente a la práctica *gato y ratón* que se describirá en el siguiente capítulo de la documentación. Nótese que el aspecto de la interfaz no es el definitivo sino una situación intermedia en la cual solo se habían integrado el actuador de motores y la cámara. No aún los láser, sonar encoders, etc.

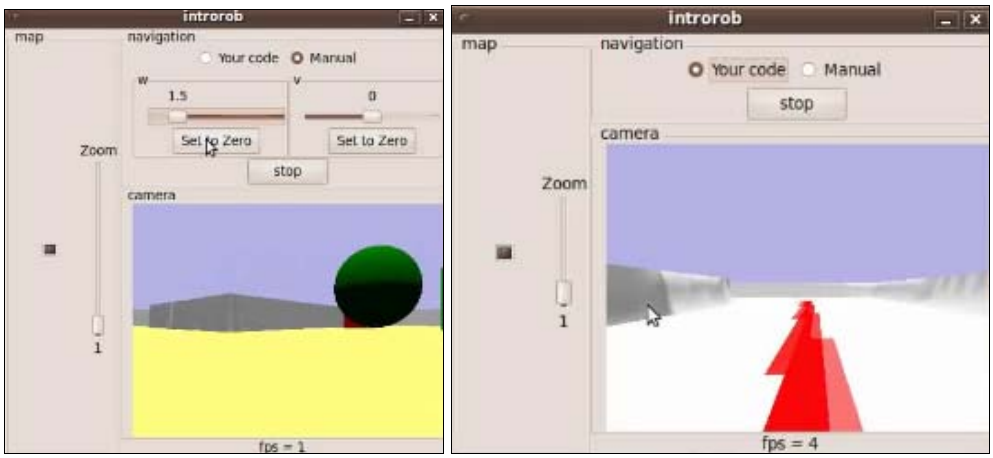


Figura 4.14: ***(a)***: modo manual sobre model *gato y raton.*
(b): modo autónomo con algoritmo *siguelinea.*

Para la validación del modo automático, se empleo en primer lugar el entorno *siguelinea* (fig. 4.14.b), y posteriormente se migró el código de navegación de esta práctica (realizado entre noviembre y diciembre sobre *introrob* 4.3) al método *navegación* de la clase *navega*. El tiempo y dificultad de integración de este código fue muy reducido, dado que el acceso a sensores y actuadores sigue guardado en variables (atributos) y las magnitudes de los valores también se han mantenido con respecto a la versión 4.3.

Para la verificación del código, en una segunda fase con el soporte completo a los dispositivos del robot, se portaron las lógicas de navegación de las prácticas *siguepasillo* (Fig. 4.15) y *gato y ratón* realizadas anteriormente sobre *introrob* 4.3. Al igual que en la adaptación del código de *siguelinea*, el funcionamiento fue rápidamente ajustado tras renombrar los nombres de variables de sensores y actuadores.

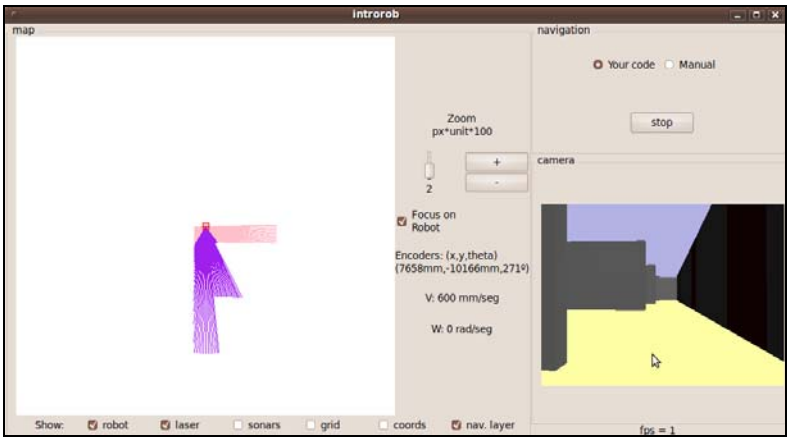


***Figura 4.15**: introrob ejecutando en modo autónomo en práctica siguepasillos*

Durante esta etapa se destinó un número de horas significativo a la puesta a punto de la representación gráfica contenida en el elemento canvas y manejada por la clase *introrobcanvas*.

El rendimiento obtenido durante las pruebas no se puede tomar como referencia para la estimación en otros entornos ya que se volvió a emplear una máquina virtual para estas pruebas. Sí se vio que el descenso en la frecuencia de ciclos invocación del método *navegación* y refresco de la información en la GUI se debió fundamentalmente a la lógica programada en el robot. En los modos de ejecución manual se alcanzaron tasas >10fps de refresco en el dispositivo cámara, el cual se incrementará notablemente si la aplicación fuera ejecutada sobre una estación de cómputo real.

4.3.4 Integración componentes en proyecto jderobot 5

También resultó una parte a resaltar del trabajo, la necesidad de integrar los componentes desarrollados en la estructura del proyecto mantenida con *SVN*. Dos tipos de actividades relacionadas se pueden distinguir:

- Incorporación al repositorio de los ficheros fuentes de los componentes *introrob* y *gazeboserver*.
- Realizar sincronizaciones periódicas del repositorio *SVN* sobre el *workscape* local.

El primer objetivo no solo comprende agregar y mantener actualizado el repositorio central del proyecto 5.0 con versiones estables de los ficheros fuentes de ambos componentes. También agregar las reglas de validación de dependencias de *introrob* y *gazeboserver* y reglas de compilación *makefile.am* para los mismos. De este modo, la compilación de la plataforma, incluidos los dos componentes realizados en el TFM, están listos para su compilación mediante las herramientas *autotools*.

Por otro lado, al no tratarse del desarrollo de una aplicación aislada sobre una plataforma en estable, sino consistir en el desarrollo de nuevos componentes en un proyecto en etapa gestacional, los cambios en ficheros fuentes de otros componentes y librerías de referencia usadas por *gazeboserver* e *introrob* obligaron a realizar actualizaciones periódicas del repositorio y adaptar la programación del código con los cambios producidos en fuentes de librería o interfaces usados.

Capítulo 5

Prácticas sobre *jderobot* 5.0

Uno de los subobjetivos fijados para el TFM fue la programación de diversas prácticas realizadas, durante el curso académico 2009/10 de la asignatura Robótica Master en la plataforma *jderobot* 4.3, en el nuevo entorno de prácticas *jderobot* 5.0. El capítulo se ha organizado con varias secciones para cada una de las prácticas programadas en la nueva plataforma, las cuales han sido *siguelinea*, *siguepasillos* y *gato y raton*. Cada sección describirá el enunciado de la práctica para aclarar los objetivos de la misma, hará referencia a los conocimientos teóricos de navegación que se han implementado en el código y finalmente se mostrarán algunas capturas de la ejecución de estas prácticas sobre *introrob* 5.0

5.1 Siguelínea

Se trata de programar un robot Pioneer, dotado de motores y una cámara, para que siga una línea roja pintada en el suelo. La rapidez en el movimiento es una característica deseable para que se valore positivamente el algoritmo implementado en el robot. Para la realización de la práctica se entrega un mundo virtual modelizado en el simulador *gazebo* y donde existirá el robot Pioneer que controlará nuestro programa.

Este escenario no cuenta con la entrega de un mapa global accesible por la inteligencia de nuestro robot y donde se pudiera describir el recorrido de la línea roja, mediante secuencia de coordenadas ni tampoco la identificación de obstáculos. Esta restricción limita la posibilidad de implantar una técnica de navegación híbrida donde se usen al mismo tiempo técnicas reactivas de evitación de obstáculos o ajuste del recorrido del robot a la línea roja con la navegación con el uso de coordenadas.

Por lo tanto, para la realización del código se empleará una programación reactiva compuesta por dos etapas que se repiten iterativamente a lo largo del tiempo: sensar y actuar (Fig. 5.1).

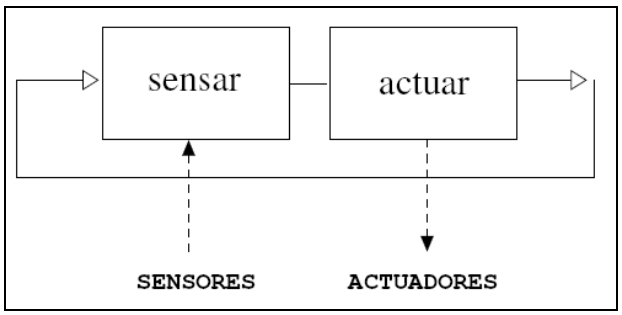


***Figura 5.1**: Esquema de funcionamiento algoritmo reactivo*

La primera es responsable de la identificación de situaciones, mediante la información proporcionada por los sensores del robot. La segunda fase consiste en la aplicación de una acción para la situación encontrada. Normalmente, el comportamiento del robot se puede describir mediante una tabla de correspondencia entre posibles situaciones (estados) y acciones a aplicar.

Uno de los puntos clave de estos modelos en la definición de situaciones mutuamente excluyentes, de modo que no se puedan alcanzar una situación con dos reglas de actuación opuestas, si no se dispone de más información para seleccionar la regla a aplicar (por ejemplo, mediante una técnica híbrida que además cuente con información de un mapa).

Existen diferentes técnicas reactivas *[s. reactivos, Cañas, 2009]* válidas para tratar el caso de esta práctica. El más sencillo sería el *basado en casos*, que como se acaba de explicar, propone la identificación de situaciones, estados o casos del robot para las cuales se aplican acciones concretas. Este tipo de algoritmos tienen como pros un esquema iterativo de funcionamiento, rapidez en la toma de decisiones y capacidad para reaccionar ante imprevistos. No obstante, la sencillez de implementación de las iteraciones desplaza la complejidad para la identificación de los casos. Otra técnica reactiva en la *basada en autómatas finitos*, donde se define una máquina de estados asociada al comportamiento del robot y para cada estado se asocian unos patrones de percepción y una acción a realizar. Las transiciones entre estados se producen al detectarse un cambio en el reconocimiento de estos patrones.

En la realización de esta práctica se ha empleado un sistema reactivo basado en casos. En términos generales, se ha diseñado un algoritmo de sensado capaz de identificar las situaciones siguientes:

a. Sobre la línea roja en recta y alineado.
b. Sobre línea roja en recta y parcialmente desviado a la izquierda.
c. Sobre línea roja en recta y parcialmente desviado a la derecha.
d. Sobre línea roja próximo a la entrada de una curva
e. Sobre línea roja dentro de una curva a izquierda
f. Sobre línea roja dentro de una curva a derecha
g. Fuera de la línea roja

Durante la etapa de actuación, para cada una de estas situaciones se ha establecido unos valores sobre el actuador motor del robot.

Situación	*Actuación en motores*
a	*V=600, W=0*
b	V=600, W=-0.1
c	V=600, W=0.1
d	V=200, W=0
e	V=100, W=-0.5
f	V=100, W=0.5
g	V=0, W=0

El valor de la velocidad lineal (V) y la radial (W) se fue variando durante las pruebas de modo que se intentará alcanzar el máximo rendimiento (rapidez de desplazamiento) sin que los valores asignados comprometiesen la seguridad del robot para salirse del rastro de la línea.

Respecto al algoritmo implementado en la fase de sensado se basó en la monitorización de la densidad de color rojo en varias celdas predefinidas sobre la imagen proporcionada por la cámara. La figura 5.2 muestra la colocación aproximada de estas celdas. Sobre cada una de ellas, en cada iteración se evaluaba la proporción de color rojo sobre el total. La identificación de cualquiera de las 7 situaciones enumeradas anteriormente se realizó comprobando relaciones entre valores umbrales de varias celdas. Por ejemplo, según nuestra la figura, un valor superior al 90% de rojo en las celdas centrales y un color inferior al 20% en las cuatro lateral, identifican la situación "*a*".

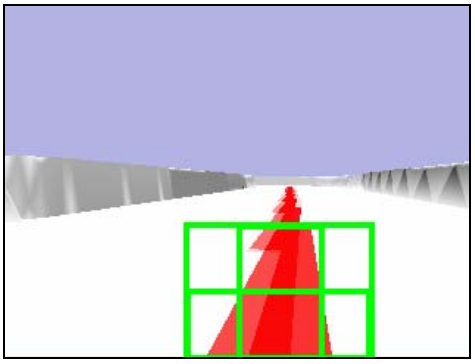


***Figura 5.2**: Identificación de situaciones basado en porcentajes de rojo sobre celdas de referencia*

Concluimos que utilizando este esquema, la eficiencia de la navegación del robot se verá basada en un proceso de ajuste sobre el reconocimiento de situaciones y la asignación de velocidades altas pero seguras. Para el primer problema, se deben precisar los valores umbrales más adecuados para reconocer cada situación de un modo seguro y sin producir ruido a lo largo de las iteraciones.

Las siguientes figuras muestran algunas capturas durante la navegación del robot. .

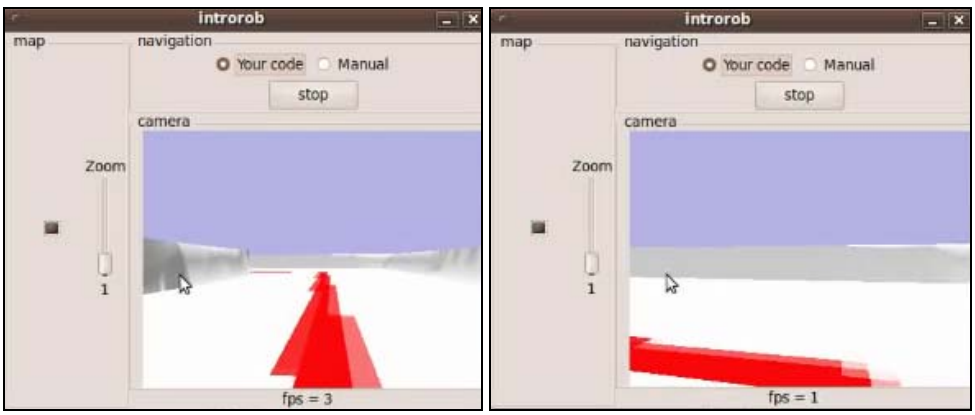


***Figura 5.3**: **(a)** robot navegando en situación a*
***(b)** robot navegando en situación e*

5.2 Siguepasillos

Se trata de una variante de la anterior práctica en la que el robot sigue un pasillo en un entorno de oficinas. En este caso no hay ninguna línea específica pintada en el suelo, pero sí ocurre que el color del suelo es diferente del color de las paredes. También se emplea *gazeboserver* como herramienta de simulación y se entrega un entorno simulado con el área departamental del GSYC de la URJC.

Al igual que en la práctica anterior, también en este caso se implementa la solución con un control reactivo basado en casos, dado que el número de situaciones posibles no era demasiado alto y la técnica para su identificación sobre la imagen proporcionada con la cámara tampoco.

Situación	*Actuación en motores*
a. Robot moviéndose por pasillo, lejos del final del corredor e igualmente distanciado de las paredes laterales.	*V=600, W=0*
b. Robot moviéndose por pasillo, lejos del final del corredor y sensiblemente o notablemente más próximo a la pared izquierda (a un obstáculo en la izquierda como caso particular).	V=600, W=-0.1
c. Robot moviéndose por pasillo, lejos del final del corredor y sensiblemente o notablemente más próximo a la pared derecha (a un obstáculo en la derecha como caso particular).	V=600, W=0.1
d. Robot moviéndose por pasillo próximo al final del corredor y con giro 90º a la izquierda para el nuevo pasillo.	V=200, W=0.3
e. Robot moviéndose por pasillo próximo al final del corredor y con giro 90º a la derecha para el nuevo pasillo.	V=200, W=-0.3
f. Robot moviéndose por pasillo próximo al final del corredor y con nuevos pasillos a ambos lados (giros 90º en ambos sentidos)	V=200, W=-0.3

Para la identificación de las seis situaciones distintas se fijaron 4 áreas sobre la imagen de la cámara que se representan en la figura 5.4. El cuadrado pequeño es en realidad un pixel con el que se monitoriza la llegada de la pared frontal. La identificación de este acercamiento, mediante el rectángulo central inferior avisaba al sistema con demasiado retraso. Los dos rectángulos laterales se utilizan para identificar el sentido de giro al final del pasillo o la desviación durante el desplazamiento.

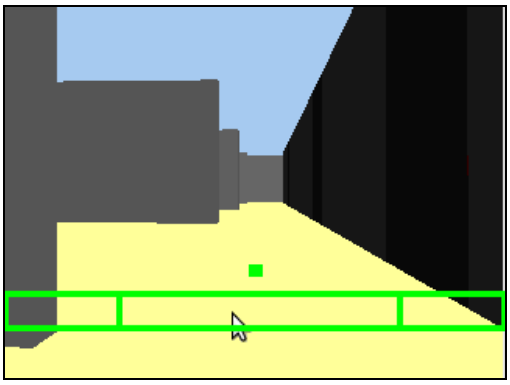


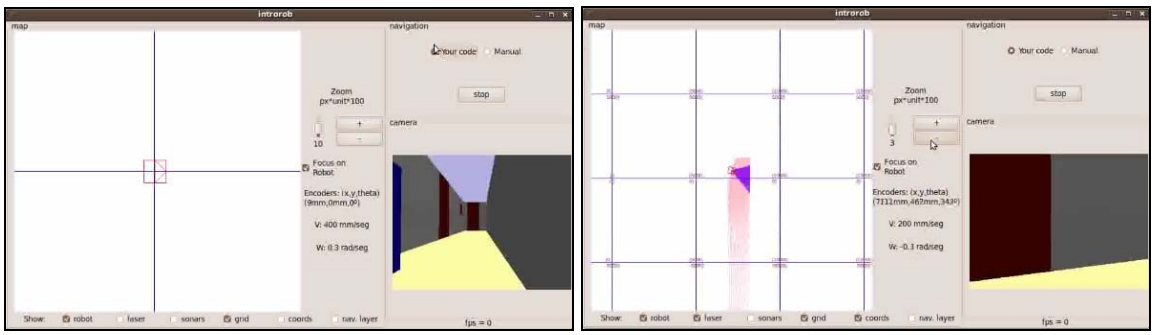
***Figura 5.4**: información procesada para identificación de situaciones*

Para la codificación de esta práctica se partió del esqueleto de métodos auxiliares escritos en la clase *navega* durante la realización del *siguelinea*. El listado 5.1 muestra los dos principales llamados desde *navegación*, los cuales denotan la implementación del algoritmo de seguimiento de línea roja. En realidad, aunque no se renombró en esta práctica, la lógica interna sí fue adaptada con el nuevo planteamiento descrito arriba.

```
void navega::navegacion( const int width, const int height,  unsigned char *
imageData, jderobot::LaserDataPtr laserData, jderobot::E
ncodersDataPtr encodersData )
  {
        printf("[alto,ancho]: [%d,%d]\n",height,width);
        sensor_camera_red_line_aligned(width,height,imageData);
        reaccion_camera_red_line_aligned();
  }
```

***Listado 5.1**: fragmento código navega.cpp para siguepasillos*

A continuación se muestran un conjunto de figuras donde se puede apreciar el funcionamiento del robot (Fig. 5.5). Tras varios ciclos de ajuste sobre los valores umbrales de color en las áreas de referencia y ajuste en los parámetros de velocidad, se consiguió una solución segura sin colisiones y con una navegación relativamente rápida en las curvas.



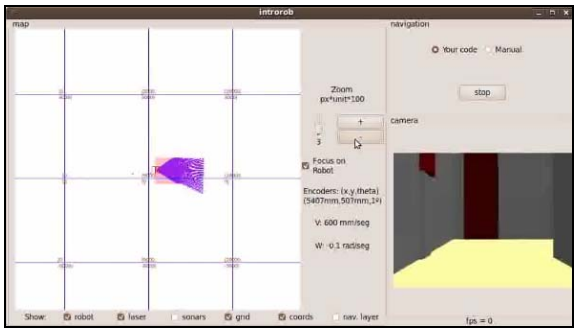


***Figura 5.5**: **(a)** robot en navegación situación a*
***(b)** robot en navegación situación e*
***(c)** robot en navegación situación b*

5.3 Gato y ratón

En esta práctica hay que programar un Pioneer dotado de motores, un láser y una cámara persiga a otro robot como él que se encuentra en el mismo mundo simulado. El robot objetivo (ratón) se puede mover conectando un *jderobot* y teleoperándolo con su *introrob* correspondiente. El robot perseguidor (gato) es el que hay que programar. Típicamente consta de dos partes una parte perceptiva y una parte motora. En la parte perceptiva hay que programar los algoritmos que identifican al otro robot en las imágenes y en el láser. Con ello se tiene una idea en términos relativos de dónde está el objetivo respecto del robot programado. En la parte motora hay que hacer mover al robot de modo que persiga, sin chocar, al otro. Por ejemplo gire hacia la izquierda y si el otro se desplaza en esa dirección, que avance si el objetivo se aleja, etc.

La realización de esta práctica se ha abordado con la misma base de programación reactiva aunque se añaden varias complejidades extra: se debe utilizar como segundo sensor el láser, el objeto a seguir es móvil y la identificación del objetivo se debe basar en un reconocimiento morfológico que según la posición puede variar.

No obstante, la complejidad repercute en la etapa de sensado, ya que la clasificación de situaciones distintas y de acciones no es extensa:

Situación	*Actuación en motores*
a. No se identifica al objeto	*V=0, W=0*
b. Se identifica al objeto y está más cerca de una distancia umbral	V=0, W=0
c. Se identifica al objeto, está más lejos de una distancia umbral y a la derecha respecto a la orientación de la cámara	V=600, W=-0.1
d. Se identifica al objeto, está más lejos de una distancia umbral y a la izquierda respecto a la orientación de la cámara	V=600, W=0.1
e. Se identifica al objeto, está más lejos de una distancia umbral y alineado con la cámara	V=600, W=0

Se han de realizar un par de consideraciones respecto a la anterior clasificación:

- La lógica de sensado considerará al robot objetivo alineado cuando el ángulo entre la cámara y dicho objetivo no supere un valor de margen de desviación. Este criterio evitará el pilotaje constante del robot perseguidor, dado y alcanzar la alineación completa es tarea difícil de lograr y posteriormente mantener.
- Las velocidades expresadas para los casos son constantes. Se puede mejorar el comportamiento del robot haciendo el giro más rápido conforme más cercano esté el objetivo. De este modo se evitará el riesgo de que el robot perseguido pase cerca y ante la cámara del perseguidor sin que a éste le de tiempo de girar.

La identificación del objetivo se ha basado en el reconocimiento morfológico del robot Pioneer simulado en *gazebo*. Se programó un algoritmo de procesamiento de imagen donde se reconociera una zona rectangular azul sobre otra zona rectangular roja, tal que ambas zonas fueran adyacentes y la anchura de la base (rectángulo rojo) fuera mayor que la del sensor láser superior (rectángulo azul). Para ello, el algoritmo trabaja en varias fases:

- Aplicación de filtro de color manteniendo los colores azul y rojo representativos del robot
- Identificación de zona morfológica rectángulo azul.
- Identificación de zona morfológica rectángulo rojo.
- Búsqueda de pareja de rectángulos azul y rojo tal que cumplan las reglas de proximidad y tamaño relativo.
- Cálculo del centro del conjunto formado por ambos rectángulos.
- Cálculo de ángulo de posición del centro identificado
- Obtención de distancia al objetivo mediante lectura del valor láser en el ángulo obtenido en el paso anterior.

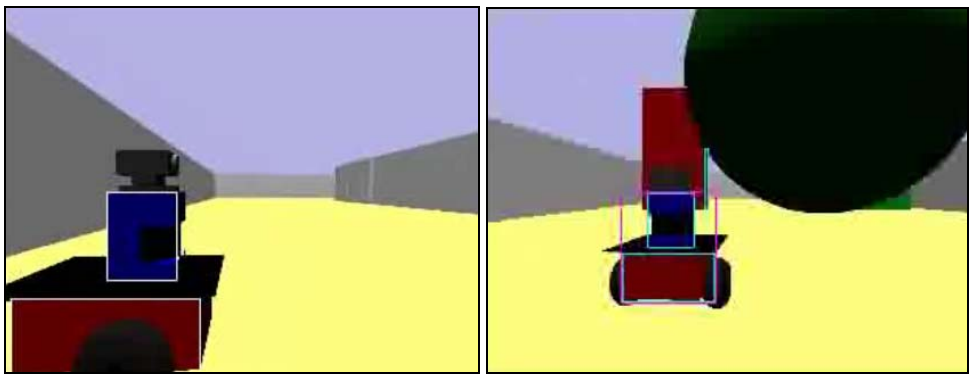


***Figura 5.6**: **(a)** identificación de rectángulos azul y rojo*
***(b)** composición morfológica para identificación del objetivo*

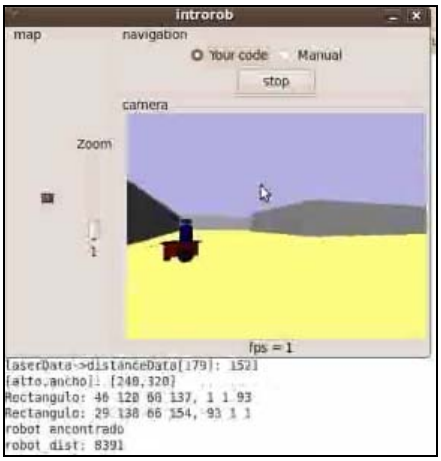


***Figura 5.7**: Ejecución de la práctica en introrob 5.0*

Capítulo 6

Conclusiones y trabajos futuros

En este último capítulo se muestran las conclusiones extraídas del desarrollo de este proyecto, comentando los puntos más relevantes que han permitido el logro del trabajo, y las principales dificultades encontradas. También hablamos de las líneas de trabajos futuros que pueden derivarse de este proyecto para su mejora y ampliación.

6.1 Conclusiones

La conclusión principal extraída de la realización del proyecto es la creación de un nuevo entorno de aprendizaje para la programación de aplicaciones robóticas satisfaciendo los requisitos descritos en el capítulo 2.

Se ha construido un sistema con la posibilidad de simular el uso de robots a través del software *gazebo* pudiendo acceder a todos los sensores y actuadores representados en el modelo. La aplicación realizada expone una interfaz gráfica al alumno desde la cual obtiene información refrescada del estado de cada sensor, incluyendo una de las cámaras del robot. La representación bidimensional en planta del entorno incluyendo la colocación del robot, las coordenadas de referencia y las lecturas láser y sonar ayudarán al usuario a entender lo que ocurre y verificar de un modo sencillo y visual si el comportamiento programado en el código del robot es incorrecto y funciona mal, en definitiva, emplear la representación como ayuda a la depuración de fallos. Asimismo, se implementan controles para el movimiento del robot que en un modo de operación manual, permiten al alumno ajustar la posición y orientación con valores adecuados para el lanzamiento de la lógica del programa. La interfaz gráfica se complementa con la funcionalidad de control del modo de ejecución, entre manual y automático, de manera que el usuario pueda ejecutar sus líneas de código solo cuando lo deseé.

La realización del trabajo aportó gran conocimiento al alumno respecto a la base de infraestructura que se emplea en materia de investigación y docencia durante la programación de aplicaciones robóticas en la URJC. Se adquirió amplio manejo en el desarrollo de aplicaciones sobre la plataforma estable *jderobot* 4.3 con el uso de software de simulación, librerías gráficas, etc. También se ha comprendido el cambio de aproximación entre la arquitectura cognitiva marcada para el desarrollo en 4.3 y la arquitectura software de mayor flexibilidad planteada en la 5.0. Este modelo de organización de componentes y establecimiento de comunicación mediante *Ice* ha sido empleado para el desarrollo de los dos componentes oficiales entregados como aportación del proyecto a *jderobot 5*. De esta manera también se logra cumplir con los requisitos funcionales R2 y R3 descritos en el segundo capítulo de la memoria.

Durante la etapa final del TFM se destina cierto tiempo a configurar los componentes *gazeboserver* e *introrob* mediante las *autotools* logrando incorporar al repositorio del proyecto la aplicación y sus reglas de validación y compilación requeridas (R8). Este punto significó también un aprendizaje para el autor en el uso

herramientas de estructuración de los desarrollos y trabajo concurrente de programación con otros compañeros involucrados en la creación de *jderobot 5.*

Se ha logrado independizar la lógica de acceso a la plataforma de simulación y la ejecución dentro de *jderobot* 5 del espacio destinado a la especificación de la inteligencia del robot. Tal y como marca el requisito funcional R1, el alumno cuenta con un esqueleto soporte y robusto para el desarrollo de sus programas.

Respecto a las cuestiones de rendimiento del entorno (R4), no se pudieron obtener valores de referencia con los que respaldar el la eficiencia del entorno, debido a que la infraestructura hardware utilizada por el autor durante toda la ejecución del proyecto fue una máquina virtual de características notablemente más bajas que las estaciones de trabajo reales. No obstante, se realizaron comparativas de rendimiento entre los componentes *gazeboserver* e *introrob* con respecto a otros existentes en la plataforma, como *cameraserver* y se observó un rendimiento similar respecto al número de frames transmitidas para la cámara del robot. El descenso notable de este rendimiento en *introrob* siempre estuvo asociado a la especificación de una lógica pesada en las iteraciones del sistema reactivo.

Por último, la realización del trabajo permite reforzar la asimilación de conocimientos sobre los algoritmos de navegación, especialmente, los de tipo reactivo durante la realización de diversas prácticas propuestas en esta plataforma.

6.2 Trabajos futuros

El proyecto descrito en este trabajo fin de master pretende ser el entorno de desarrollo de prácticas de futuros alumnos que utilicen esta plataforma como elemento de su proceso de aprendizaje, permitiéndoles centrarse en la implementación de inteligencia en los robots y abstrayéndose de la complejidad que conlleva es uso de una infraestructura formada por numerosos componentes y librerías. Algunas de las tareas concretas para mejorar el entorno de enseñanza que se han identificado durante la realización del proyecto son:

- ***Añadir driver player***: La solución entregada en el proyecto incluye el driver de conexión al software de simulación *gazebo*. Sin embargo, aun no se ha desarrollado el driver que pudiera conectar a la plataforma con el software de simulación *player/stage* que también se usa en la URJC, dentro del dominio de la enseñanza y de la investigación.
- ***Visualización 3D del entorno****:* El componente *introrob* realizado muestra una visión 2D del entorno del robot donde se representan al propio robot y a los datos sensoriales láser y sonar leídos. Esta representación podría ser mejorada mediante una visión 3D utilizando la librería *OpenGL*.
- ***Soporte para dos y más cámaras****:* Añadir a la configuración de *introrob* y a su interfaz gráfica la capacidad de utilizar más de una cámara al mismo tiempo. Esto dará la posibilidad de realizar prácticas con robots que tengan dos cámaras o incluso emplear *introrob* como plataforma para el desarrollo de prácticas de procesamiento visual donde es requerido un número superior a uno de cámaras para lograr el propósito (ejemplo: cálculo de distancia a un objeto por triangulación).
- ***Validación del sensor sonar***: Pese a programarse la interfaz *sonars* en el componente *gazeboserver*, falta realizar la integración y validación de estos

sensores desde *introrob*. No se pudo abordar este trabajo por limitación temporal. No obstante, se estima reducido el tiempo requerido puesto que el soporte de *gazeboserver* para el sensor está realizado y los algoritmos de representación gráfica de la señal en *introrob* se crearon.

- ***Adecuar nomenclatura de parámetros configuración***: Se considera oportuno revisar la definición y nomenclatura de parámetros en los ficheros de configuración de *gazeboserver* e *introrob* para que estén alineados con el del resto de componentes de la plataforma. Por ejemplo, nomenclatura de parámetros de conexión con servicios *Ice*.
- ***Control movimiento actuador ptmotors integrado en introrob:*** Agregar a la interfaz del sistema *introrob* los selectores de operación manual del actuador *ptmotors* del robot Pioneer.

Bibliografía

[Cañas Plaza et al., 2007] José M. Cañas Plaza, Antonio Pineda, Jesús Ruíz-Ayúcar, José A. Santos, y Javier Martín. *Programación de robots con la plataforma jdec*. URJC, 2007.

[Cañas Plaza, 2003] José María Cañas Plaza. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo*. PhD thesis, Universidad Politécnica de Madrid, 2003.

[Martín Ramos, 2008] Javier Martín Ramos. *Proyecto Fin de Carrera. Detección y seguimiento de caras en la plataforma jdec.* URJC, 2008.

[jderobot 5 thesis, D. Lobato, 2010] David Lobato Bravo. Trabajo Fin de Master. *jderobot 5 Entorno de desarrollo basado en componentes para aplicaciones robóticas*. URJC, 2009.

[programación jderobot, Cañas Plaza, 2009] José María Cañas Plaza. *Programación de robots con la plataforma Jderobot*. URJC, 2009.

[Marugán Alonso, 2009] Marugán Alonso, S. (2009). *People 3d tracking using volumetric primitives. Technical report, Universidad Rey Juan Carlos.*

[P.robótica. Cañas, 2008] José María Cañas (2008). *Prácticas de Robótica-Máster*. URJC.

[s. reactivos, Cañas, 2009], José María Cañas (2009). Robótica Master. Tema 8 – sistemas reactivos. URJC.

[Serradilla, 1997] Serradilla, F. (1997). *Arquitectura cognitiva basada en el gradiente sensorial y su aplicación a la robótica móvil*. PhD thesis, Universidad Politécnica. Madrid.

[intro, Cañas. 2009] José María Cañas (2009). Robótica Master. Tema 1 – introducción a la robótica. URJC.