Proyecto fin de carrera

Javier Serrano López

Desarrollo de un Software De Navegación para el Robot Móvil RobIAS

Sevilla, 10 de junio de 2003

Tutor: **Jorge Chávez Orzáez**
Departamento de Ingeniería Electrónica,
Grupo de Tecnología Electrónica

Resumen

El Robot Móvil RobIAS tiene por cometido guiar a los visitantes del *Insitut für Automatisierungs- und Softwaretechnik* (IAS) (Instituto de Ingeniería de Software y Automatización Industrial) en la Universidad de Stuttgart (Alemania), y suministrar información acerca de los campos y materias de investigación así como de las plantas de demostración existentes. En este marco se iniciaron dos proyectos: “Software de Navegación” (Navigation software) y “Control de la Visita Guiada” (Tour guide).

Dentro de este proyecto global, y correspondiente a mi proyecto fin de carrera, se implementó el software de control para la navegación autónoma del robot, el denominado “Software de Navegación”. Esto incluye tareas de planificación de caminos, así como evitación de obstáculos en tiempo real y aproximación de la posición real del robot. Además, se detectan situaciones comunes no deseadas como puertas cerradas y caminos bloqueados, y se proporciona una interfaz ActiveX con el “Control de la Visita Guiada” para determinar los destinos a ser alcanzados.

El software para el control de la visita y presentación de la información relativa a las materias de investigación en el Instituto fue desarrollado por otro estudiante del mismo, Kristian Dencovski (WSA 1884). Este software permite la ejecución y administración de la visita guiada a través de la red de área local del Instituto, mediante la determinación de los destinos a alcanzar y la reproducción de textos en los puntos requeridos.

Abstract

The mobile Robot RobIAS guides visitors through the Institute of Industrial Automation and Software Engineering (IAS) and communicates information concerning fields and subjects of research as well as demonstration plants. In this context two projects have been launched: the “Navigation software” and the “Tour guide”.

Within the scope of this research project the control software for the navigation of the robot has been implemented. It includes the path planning activities as well as real-time obstacle avoidance and position approximation. Furthermore, it detects common undesired situations like doors closed and ways blocked, and provides an ActiveX interface to the Tour guide in order to determine the targets that must be reached.

The software for guiding the tour and communication information about the subjects of research has been developed in the other project by Kristian Dencovski (WSA 1884). This software allows the execution and administration of the Guided tour via the intranet of the Institute by giving the robot targets and playing texts at requested points.

Índice de contenidos

1 Introducción

1.1 Motivación

En el día „Tag der offenen Tür“ (Día de las puertas abiertas) en el *Insitut für Automatisierungs- und Softwaretechnik* (IAS) (Instituto de Ingeniería de Software y Automatización Industrial) en la Universidad de Stuttgart (Alemania), todas las personas que lo deseen pueden visitar las diferentes áreas de investigación que están activas en ese momento y otros trabajos de interés que fueron realizados en el pasado en dicho Instituto. La finalidad de este evento es ofrecer a todos los visitantes una imagen atractiva del Instituto, y en particular, ayudar a los futuros estudiantes en la Universidad de Stuttgart a decidir su rama de especialización.

En este contexto, se desea que dichos visitantes obtengan una impresión muy favorable del Instituto, para que se acuerden durante largo tiempo de lo visto e incluso lo comenten a sus amigos y familiares. Para lograr este objetivo, se puso en marcha el desarrollo de una visita guiada y automatizada a través del Instituto con el robot móvil RobIAS.

1.2 Finalidad general

Este proyecto fin de carrera se llevó a cabo en el contexto de un proyecto global de una visita guiada a través del Instituto IAS. Esta visita está guiada por el robot móvil RobIAS, que debía conducir a los visitantes de una manera autónoma a través del Instituto, mostrándoles las diferentes áreas de investigación y hablando un poco de ellas. Para ello, el robot móvil RobIAS pide a los visitantes que lo sigan, y cuando están en el punto deseado, explica en algunas frases qué es lo que se está investigando exactamente en ese punto. Esto podría incluir no sólo una descripción hablado, sino también una actividad demostrativa.

El objetivo de este proyecto fin de carrera, dentro del proyecto global indicado, es el desarrollo de un software de control y navegación para el robot móvil RobIAS. Este software de navegación debe posibilitar además un movimiento autónomo para el robot, incluyendo tareas básicas de planificación de caminos, evitación reactiva (en tiempo real) de obstáculos y aproximación de la posición real del robot. Las presentaciones multimedia y las actividades demostrativas no son parte de este proyecto fin de carrera, y fueron realizadas por otro estudiante del Instituto [12].

1.3 Requerimientos del software

Nuestro robot tiene que realizar varias acciones: planificación de caminos, evitación de obstáculos, localización,... Tenemos que considerar cuidadosamente la sincronización de todas estas acciones, de forma que no haya interferencias entre ellas.

El primer aspecto a considerar es la corrección de una posición inicial incorrecta por medio de la localización del robot. Esto se puede hacer mediante la comparación entre las lecturas de los dispositivos sensores (láser, ultrasonidos) y la información del mapa.

Una vez que el robot esté listo, tenemos que moverlo de su posición actual a la posición deseada. Estamos considerando en este proyecto un entorno conocido: el Instituto IAS. Por tanto, no necesitamos una estrategia de planificación de caminos flexible y general, sino una manera sencilla y efectiva de determinar los puntos intermedios que deben ser cruzados, de acuerdo con el punto inicial y final y el mapa conocido del Instituto.

Cuando se haya procedido a la planificación de la trayectoria, tendremos que mover el robot de acuerdo a esa trayectoria, y es bastante probable que encontremos obstáculos no esperados en el camino. La seguridad en nuestro proyecto es importante, y eso significa que ni los visitantes ni las personas que ocasionalmente estén andando por la trayectoria del robot deben ser lastimados. Por tanto, debemos adoptar medidas que eviten las colisiones, usando los sensores integrados en el robot y la información del mapa del Instituto. Si no existiese un camino libre de obstáculos que se puedan sortear hacia el punto destino, debemos buscar una nueva trayectoria global, y si incluso en este caso tampoco existiera una alternativa viable, entonces deberíamos parar el robot para evitar males mayores.

Cuando el robot alcanza un punto intermedio tenemos que comprobar si la posición del robot es correcta. El sistema de odometría del que dispone el robot funciona bien en general, pero hay desviaciones impredecibles e inevitables que originan que el robot se pierda con bastante facilidad. Estas desviaciones crecerán progresivamente si permitimos que se acumulen sin hacer nada al respecto, y esto deriva en una percepción errónea del entorno por parte del robot (por ejemplo, el robot podría creer que el camino está bloqueado porque „ve“ una pared donde espera encontrar un pasillo). Debido a todo esto, se hace evidente la necesidad de la determinación de la posición real del robot y su corrección. Como en el caso de la planificación de caminos, no se va a implementar un método general de localización, sino uno simplificado que satisfaga nuestras exigencias para esta situación particular.

Como resultado de todo lo presentado anteriormente, tendremos un robot capaz de moverse de forma completamente autónoma a través de los pasillos del Instituto IAS sin perderse (si las

condiciones son las normales). El último paso será la comunicación entre nuestro software de navegación y el software de control de la visita guiada, que es completamente independiente del nuestro. Hay que transferir el control hasta que la presentación correspondiente al punto alcanzado termine, y entonces hay que transferir el control de nuevo al robot, a la vez que se suministra el número del nuevo destino que debe ser alcanzado. Todo esto lo hacemos por medio de una interfaz entre el software de navegación y el control de la visita guiada, que implica una transferencia sencilla de mensajes para conmutar el control de una forma sincronizada entre las dos partes y la determinación por parte del control de la visita guiada del nuevo destino a alcanzar. Esta comunicación también incluye mensajes como „Puerta cerrada, por favor ábrala“ o „Camino bloqueado, no puedo avanzar“, de forma que los visitantes puedan ayudar al robot a hacer su trabajo adecuadamente.

1.4 El Instituto IAS de la Universidad de Stuttgart

Este proyecto fin de carrera fue realizado en el *Insitut für Automatisierungs- und Softwaretechnik* (IAS) (Instituto de Ingeniería de Software y Automatización Industrial) de la Universidad de Stuttgart (Alemania), como parte de un programa de intercambio libre, y de mutuo acuerdo entre mi tutor Jorge Chávez Orzáez del Departamento de Ingeniería Electrónica de la Universidad de Sevilla y mis tutores en Alemania los señores D. Pascal Jost y el Sr. Professor P. Göhner, director del Instituto IAS. La parte técnica del proyecto fue realizada en su totalidad en el Instituto IAS, donde también hube de presentar una memoria y defender el proyecto. Después hube de escribir esta memoria para mi Universidad de Sevilla y defender también aquí el proyecto.

Para poder realizar mi proyecto fin de carrera en el Instituto IAS, se me exigió que me ajustara a todas las líneas de trabajo de dicho Instituto. Ello comprendía lo siguiente:

- El Instituto IAS de la Universidad de Stuttgart está certificado oficialmente con las normas de calidad ISO 9001 por las autoridades correspondientes, y dichas normas son de aplicación a todas las tareas que en él se realizan. Esto incluye también a los estudiantes, que deben realizar sus trabajos fin de carrera ajustándose a dichas normas de calidad. A grandes rasgos, esto implica la redacción de un determinado número de documentos, una planificación del proyecto, cuyos plazos se controlan con las revisiones pertinentes, y un control continuo de la calidad del producto elaborado, en este caso, el proyecto fin de carrera. En el Anexo se detallará mejor cómo se aplican estas normas de calidad en el Instituto IAS y los documentos generados como resultado de las mismas.

- En aplicación a las normas de calidad anteriormente referidas, y al propio sistema educativo alemán, la duración prevista de un proyecto fin de carrera es de 6 meses. De acuerdo a este

criterio, se fijan unos criterios a realizar, de forma que la duración del mismo no exceda de los 6 meses previstos. Por tanto, la carga de dicho proyecto está ajustada a esos 6 meses y al hecho de que el seguimiento de las normas de calidad implica un trabajo extra (especialmente en lo referente a la elaboración de los documentos), tal como se detallará en el Anexo.

- Como en cualquier proyecto fin de carrera realizado en el Instituto IAS, hube de escribir una memoria del mismo (una copia de la cual está a disposición del Tribunal si así lo desea) y grabar un CD con todos los contenidos elaborados (también a disposición del Tribunal). Por dicha memoria, y en general por la realización del trabajo, se me otorgó una puntuación en dicho Instituto de 1,0 según el sistema de puntuaciones alemán, que se corresponde con la nota más alta que un alumno puede recibir por su trabajo. Una fotocopia del certificado que así lo demuestra se halla adjunta en el Anexo.

- También tuve que defender mi proyecto fin de carrera ante el Director del Instituto IAS, el señor Professor P. Göhner, mi tutor, todo el personal del Instituto y otros alumnos del IAS y demás personas que quisieron asistir al acto. Por la calidad de la presentación que llevé a cabo se me otorgó una distinción en forma de certificado (en alemán), que se suele dar solamente un par de veces al año (entre los 50 trabajos que se realizan normalmente en ese mismo período de tiempo), y cuya fotocopia también adjunto en el Anexo.

Por último indicar que también se me entregó un certificado que prueba que realicé el proyecto fin de carrera ajustándome a las normas de calidad ISO 9001 tal como se aplican en el Instituto IAS. Una fotocopia de dicho certificado se encuentra adjunta en el Anexo.

2 Fundamentos

2.1 El robot móvil RobIAS

El robot móvil RobIAS está basado en el Pioneer 2-DXE con Peoplebot Extension construido por ActivMedia Robotics [2]. El robot está provisto de diez sensores de contacto (touch-sensitive bumpers), dieciséis sensores de ultrasonidos (ultrasonic sensors) o sonares (sonars), situados en dos filas a distintos niveles, para detectar obstáculos que se mueven rápido, y un sensor láser de alta resolución (high resolution laser range finder), capaz de proporcionar información detallada del entorno. La interacción con este robot es posible a través de un par de altavoces (speakers) y un micrófono (microphone). Para el control, un microcontrolador y un ordenador personal (embedded PC) están integrados dentro del robot (ver Figura 1).

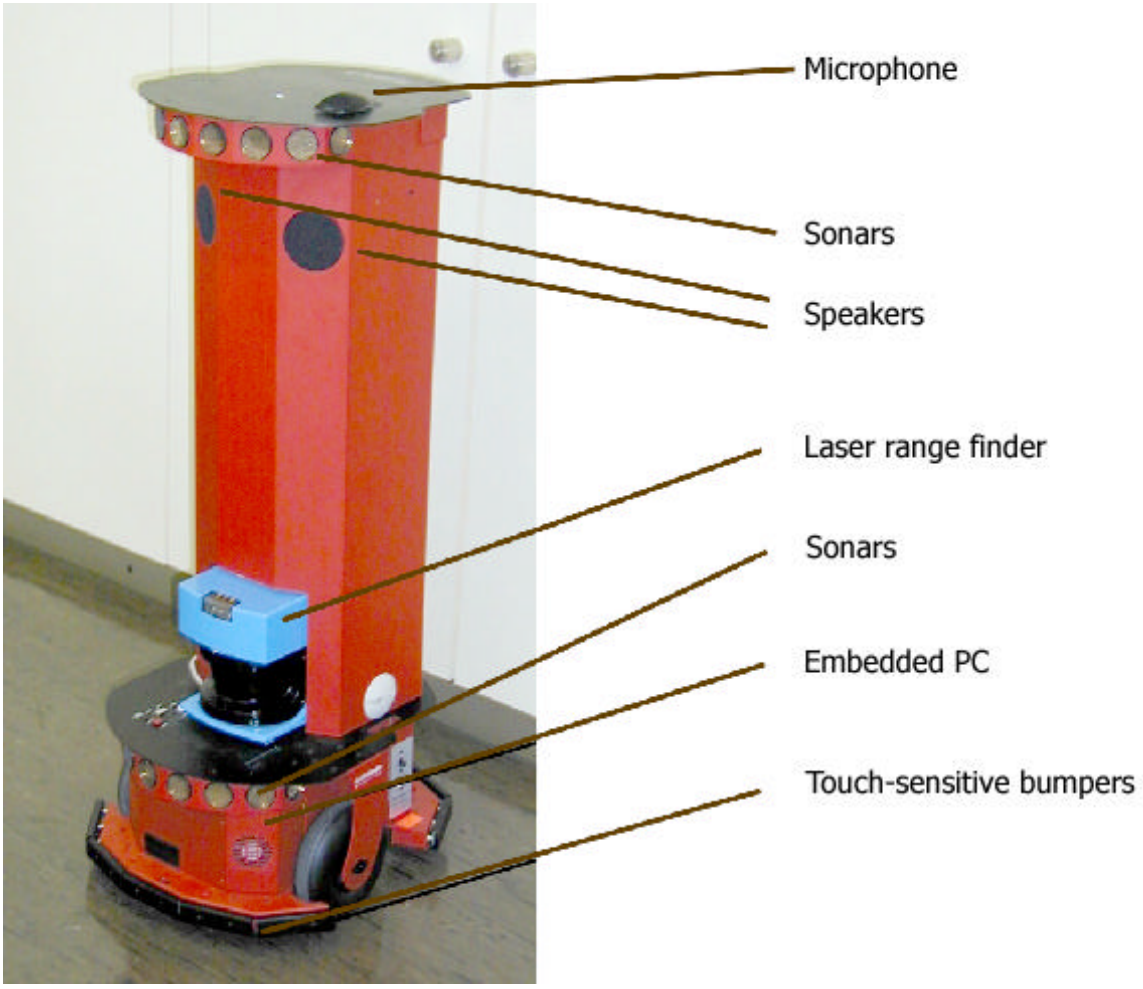


Figura 1: RobIAS

El robot posee además un ordenador personal integrado (embedded PC) con Microsoft Windows 98 SE como sistema operativo, y una tarjeta Wireless LAN (WLAN) para la conexión con la IAS-Intranet (ver Figura 2)

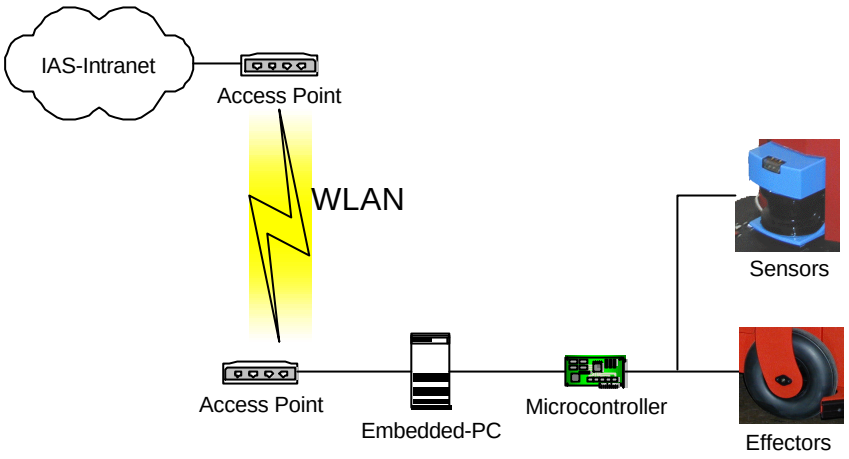


Figura 2: Arquitectura global

Para programar el software de navegación se usan ARIA y Saphira, que son suministrados por el fabricante del robot ActivMedia Robotics [2], [3]. ARIA accede al microcontrolador del robot para permitir la navegación y ofrece un juego muy completo de funciones de alto nivel para el programador, que tienen una interfaz transparente con las operaciones de bajo nivel del microcontrolador. Saphira es simplemente una capa superior que se añade a ARIA, incluyendo la GUI y otras funciones extra, como por ejemplo, la navegación por el método del gradiente. En el próximo subcapítulo 2.2 se dan detalles adicionales acerca de Saphira y ARIA.

2.2 El software suministrado con el robot

2.2.1 Visión de conjunto

Una visión general de lo que son ARIA y Saphira, y de cómo están interrelacionados, se muestra en la Figura 3, donde podemos observar la jerarquía de estos programas y sus relaciones con los efectores del robot y el microcontrolador. Como podemos ver, Saphira y ARIA simplifican la programación del robot, porque gracias a ellos no es necesario acceder directamente a los comandos de bajo nivel del robot, sino que tenemos ya funciones de alto nivel predefinidas para realizar las tareas básicas.

Las últimas versiones de Saphira y ARIA se pueden descargar desde la página web de ActivMedia Robotics[2]. Dichas versiones se actualizan muy a menudo y además se ofrece un

servicio de soporte on-line muy rápido y eficiente (como tuve posibilidad de comprobar en multitud de ocasiones). Los programas Saphira y ARIA incluyen gran cantidad de características muy interesantes. Para más detalles al respecto consúltense los siguientes subcapítulos.

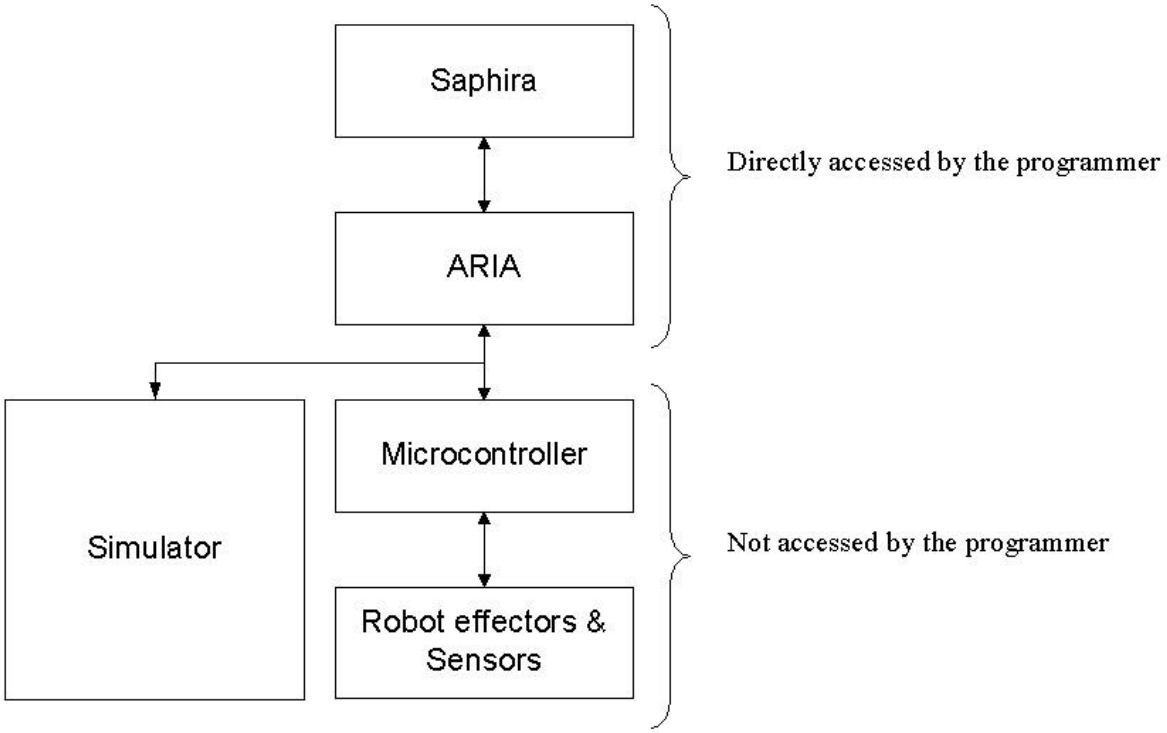


Figura 3: Arquitectura del robot

2.2.2 ARIA

En palabras de ActivMedia Robotics [3], „ARIA es una interfaz para aplicaciones de control de robots orientadas a objeto, para la línea de robots móviles inteligentes de ActivMedia Robotics“. Dicho de otra forma, ARIA es un software para el cliente que permite un acceso fácil pero de gran rendimiento a la gestión del servidor del robot así como a muchos sensores accesorios del robot y efectores.

ARIA tiene una gran cantidad de características. Se puede ejecutar en un hilo múltiple o simple, o implementar acciones y comportamientos (ambos predefinidos y definidos por el usuario), y todo está implementado con clases C++, o sea, que no tenemos que aprender un nuevo lenguaje de programación específico, sino que nos vale uno estándar como C++, y esto lo hace todo mucho más fácil.

Sin embargo, no debemos olvidar que ARIA es solamente una ayuda, un grupo de clases generales que se pueden usar de muchas maneras, pero los programas específicos han de ser escritos por los usuarios. Cada programador implementa su propio código usando estas clases

suministradas, que son una ayuda, pero no un resultado final. Es decir: ARIA no es una solución al problema planteado, sino sólo un medio para resolverlo.

2.2.3 Saphira

Saphira está basado en ARIA, y por tanto, un buen entendimiento de ARIA es necesario para poder trabajar con Saphira, en tanto que Saphira usa muchas de las clases de ARIA. Mientras que ARIA está pensado para suministrar un acceso flexible de bajo nivel a las funcionalidades del robot, Saphira simplifica la tarea del desarrollador mediante el suministro de una interfaz conveniente a todas esas funcionalidades (ver Figura 4)

Saphira es un entorno de desarrollo de aplicaciones robóticas, y la librería de Saphira es un grupo de rutinas para la construcción de clientes. La librería de Saphira integra un número de útiles funciones de ARIA para el envío de comandos al servidor, recopilación de información de los sensores del robot y su agrupación para mostrarlos en una interfaz de usuario gráfica. Adicionalmente, Saphira soporta funciones de alto nivel para el control del robot y la interpretación de los sensores, incluyendo un sistema de navegación por el método del gradiente.

El cliente Saphira se conecta al servidor de robot con los componentes básicos de percepción sensorial y navegación: motores y ruedas, codificadores de posición y sensores. El servidor maneja los detalles de bajo nivel de los sensores del robot y de la gestión de movimiento, envía información y responde a Saphira a través de la interfaz ARIA del robot.

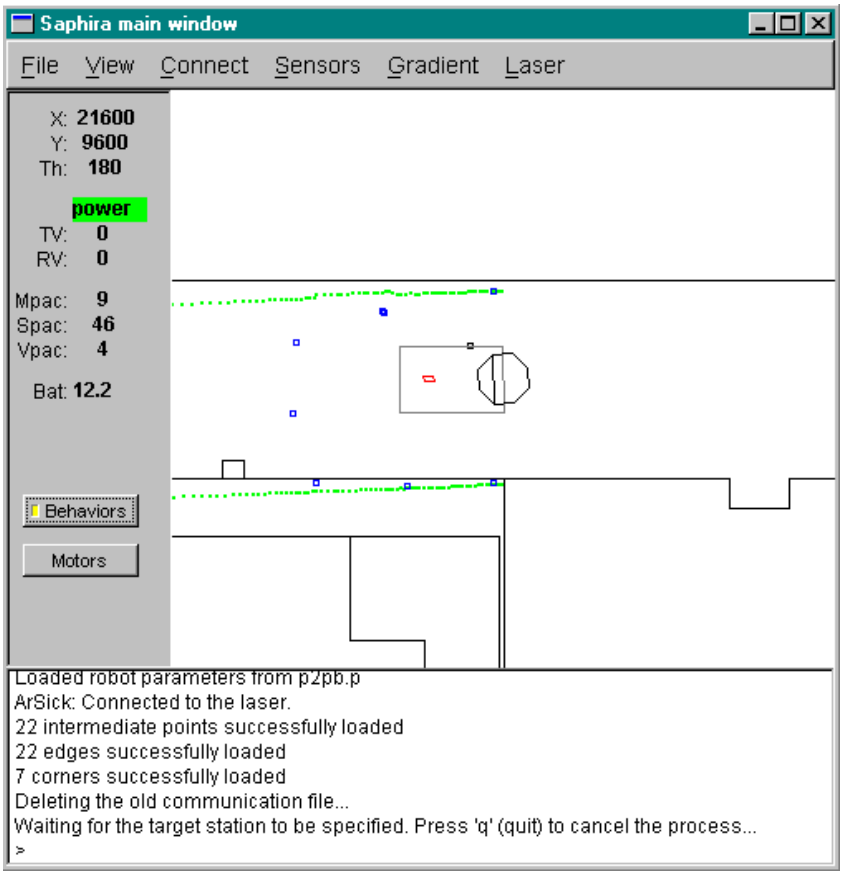


Figura 4: Aspecto de Saphira

2.2.4 El simulador Pioneer

2.2.4.1 El programa

Saphira se distribuye con un software simulador de un robot físico y de su entorno (ver Figura 5). Esta característica permite a los desarrolladores depurar las aplicaciones convenientemente en un ordenador sin necesidad de usar un robot físico (con la pérdida de tiempo que esto puede conllevar). De esta forma, el simulador es una alternativa muy útil al robot físico para el desarrollo de programas robóticos.

El simulador tiene modelos muy realistas para los sonares, sensor láser y codificadores de las ruedas, de forma que, en general, si un programa cliente funciona bien con el simulador, también funcionará adecuadamente en el robot físico. Incluso la interfaz de comunicación es la misma que para el robot físico, así que no necesitamos reprogramar o hacer cambios especiales al cliente para hacerlo correr sobre el simulador o el robot real. La desventaja del simulador es que el modelo del entorno es sólo una abstracción del mundo real, con simplemente segmentos lineales en dos dimensiones en lugar de los objetos geométricamente complejos que el robot encontraría en el mundo real.

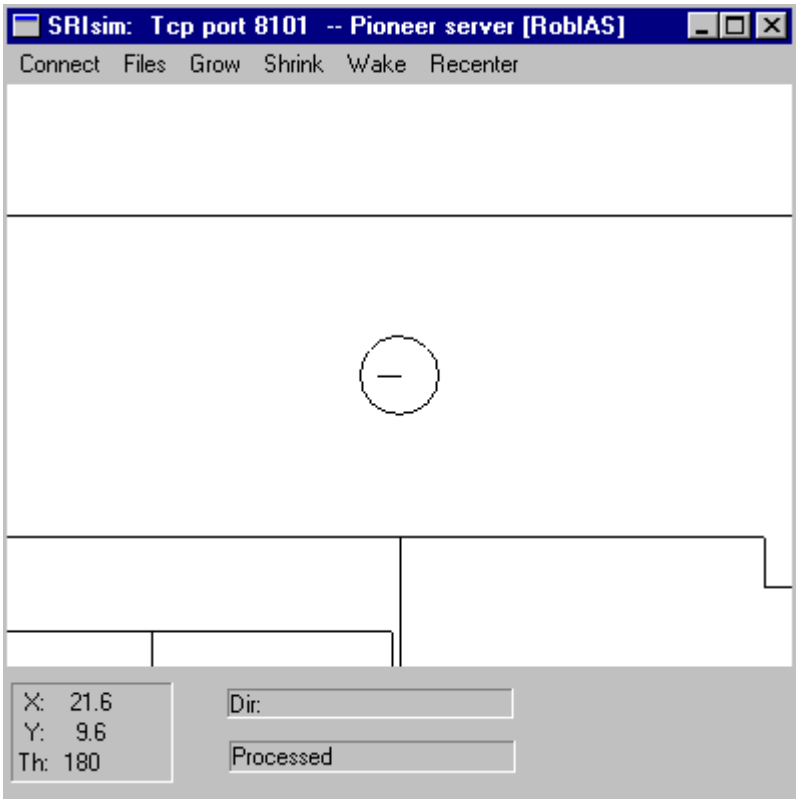


Figura 5: El simulador Pioneer

2.2.4.2 El fichero de descripción del entorno

El simulador Pioneer nos permite además construir modelos en dos dimensiones de entornos reales o imaginarios, llamados *worlds* (*mundos*, conservando el término original inglés empleado en el programa). Estos modelos del entorno son abstracciones del mundo real, con segmentos lineales que representan las superficies verticales de los pasillos, salas y objetos contenidos en ellos. Grabaremos estos modelos bidimensionales en un fichero de texto plano (ASCII), al que normalmente le asignaremos la extensión `.wld,` y que describe el tamaño y en contenido de un mundo (world) simulado. En las próximas secciones daremos más detalles al respecto.

Debido a que modelos bidimensionales del entorno son sólo una abstracción del mundo real, la información contenida en ellos no es suficiente para la navegación autónoma usando el robot real en un entorno real, y por esa y otras razones, también debemos hacer uso de los sensores integrados en el robot.

2.2.4.3 Fichero de parámetros

Usaremos un fichero de parámetros para poder tener en cuenta una gran cantidad de características simuladas del robot, tal como el radio del robot, la posición de los sonares y el láser, y las tolerancias de error en el movimiento. En nuestro caso, el fichero de parámetros de RobIAS será `p2pb.p` contenido en el directorio `params/` donde se halla instalado Saphira. Para más detalles acerca de este fichero de parámetros, consulte por favor la documentación que viene con el programa Saphira [3].

2.3 Problemas asociados a la navegación

2.3.1 Introducción

Existen algunos problemas inherentes asociados a la navegación de vehículos autónomos (como por ejemplo nuestro robot). Dos de estos problemas son la presencia de obstáculos imprevistos y los errores asociados a la determinación de la posición real del vehículo, problemas que también hemos de considerar para nuestro caso particular con el robot móvil RobIAS.

2.3.2 Obstáculos

Una vez que la trayectoria hasta el punto que ha de ser alcanzado por el robot ha sido calculada, el robot deberá moverse siguiendo esta trayectoria. Bajo condiciones ideales, esta trayectoria estará completamente libre de obstáculos y alcanzaríamos el punto destino sin ningún tipo de contratiempo. Sin embargo, en un entorno real, tendremos seguramente obstáculos inesperados e indeseados en este camino a seguir, que impedirán al robot seguir la trayectoria planeada originalmente. Por tanto, no es posible usar un camino de carácter completamente determinista hacia el objetivo, exceptuando los casos particulares en los que no existan obstáculos. En nuestro caso habrá obstáculos, así que no podemos usar esta estrategia determinista.

La solución será entonces usar una combinación de estrategias deterministas y reactivas en tiempo real. Esto significa que en principio planeamos una trayectoria global al punto destino, pero que luego comprobamos en tiempo real si este camino a seguir está libre de obstáculos, para corregirlo dependiendo de la presencia de éstos y eventualmente, recalcular una nueva trayectoria global al destino si esto fuese necesario.

2.3.3 Determinación de la posición

Para una correcta navegación, un vehículo autónomo necesita saber en qué posición está situado. La determinación de esta posición no es un asunto trivial, y puede realizarse de múltiples formas, tales como el empleo de GPS, balizas de señalización, odometría, etc. Para el caso de los robots, un método bastante usual es el empleo de la odometría para la determinación de la posición y orientación del robot. La idea básica es leer periódicamente el valor de los codificadores situados en las ruedas, y con esta información y el modelo cinemático del robot, actualizar la posición real del robot.

Sin embargo, este método lleva asociados algunos problemas. Los más importantes son:

- El valor de los codificadores en las ruedas es leído cada 100 ms (en nuestro caso), y estos valores son además discretos. La doble discretización (en el tiempo y la digitalización de los valores) implica errores numérico; estos errores no son demasiado grandes, pero en tanto que la nueva posición del robot se determina a partir de la posición anterior y el valor de los codificadores de las ruedas, estos errores se irán propagando y se harán grandes de una forma bastante rápida.

- Existen ciertos factores de indeterminación, como la presión de inflado de las ruedas o los deslizamientos de las ruedas, que afectan de forma negativa al cálculo de la posición del robot, haciendo que ésta difiera del valor real.

Debido a todo esto, no vamos a ser capaces de realizar trayectos largos con el robot usando solamente la odometría para determinar su posición y orientación. Por tanto, tenemos que desarrollar métodos de localización para corregir esos errores y permitir al robot navegar de forma satisfactoria.

2.3.4 Percepción del entorno

Ya hemos hablado acerca de los obstáculos y de la determinación de la posición y orientación del robot. Esto implica un proceso de reconocimiento por parte del robot; por tanto, la siguiente pregunta es: ¿Cómo puede el robot móvil RobIAS percibir y reconocer el entorno que lo rodea?

RobIAS no tiene una cámara, pero sí sensores de ultrasonidos (sonares) y un sensor láser. Los sonares son muy rápidos y pueden detectar obstáculos móviles fácilmente, pero a cambio tienen el problema de que son muy imprecisos y funcionan en un rango bastante pequeño. Nos serán útiles como apoyo al sensor láser, el cual es más lento, pero proporciona una información muy detallada acerca del entorno del robot.

En la siguiente Figura 6 se muestra cómo de precisos son los sonares y el sensor láser:

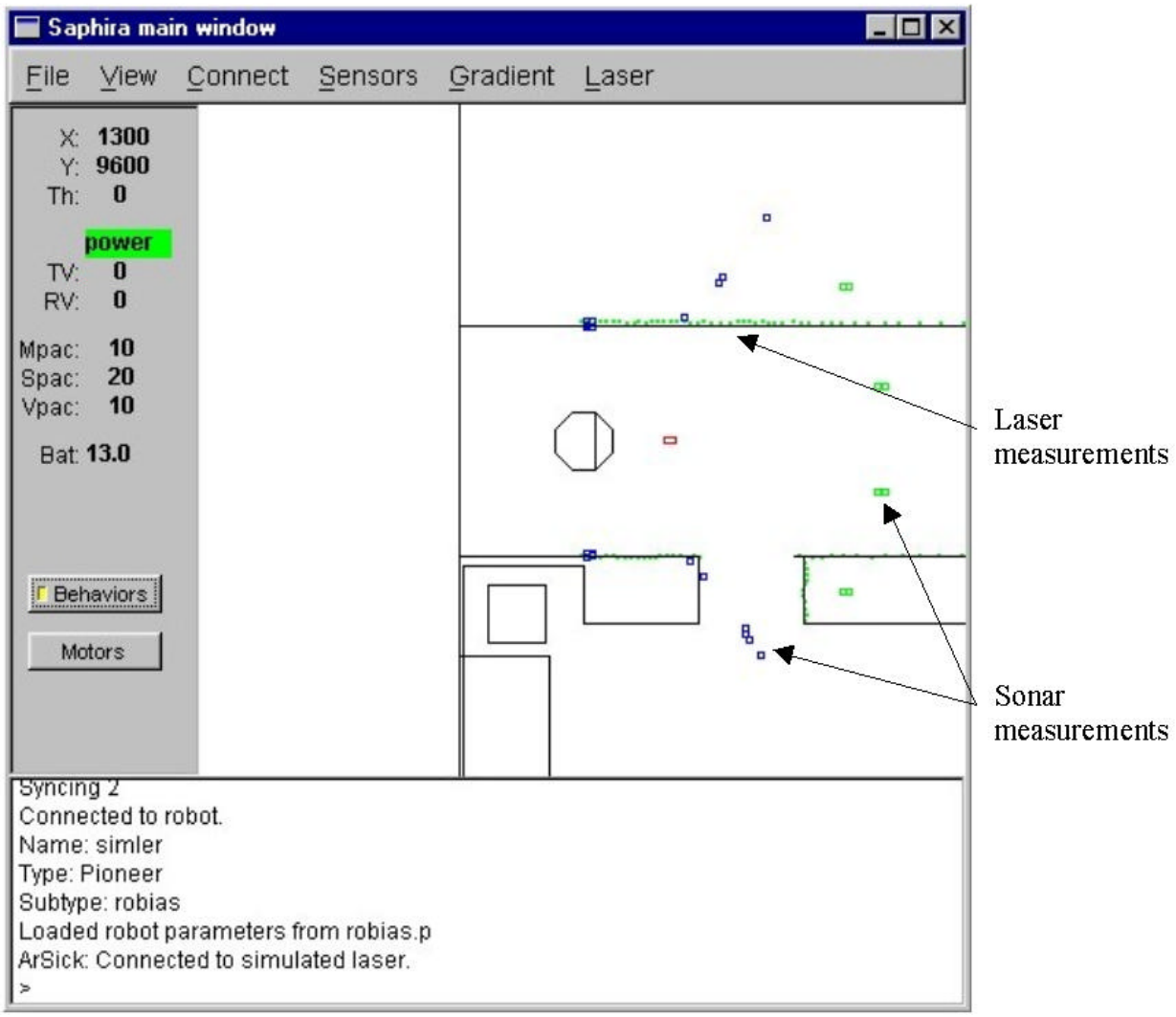


Figura 6: Precisión de los sonares y sensor láser

Las medidas de los sonares están representadas con pequeños cuadraditos de color azul y verde, y las medidas del láser con puntos verdes. Podemos observar que los sonares proporcionan una información muy pobre del entorno, y que sólo podemos detectar algunos obstáculos; en cuanto al láser, que „barre“ con sus rayos en un rango de 180 grados con incrementos de medio grado, proporciona una información muy detallada que podemos usar para detectar los obstáculos y para la localización.

En la siguiente Figura 7 mostramos cómo el láser escanea en todas las direcciones en un rango de 180 grados para obtener una matriz de puntos que después usaremos para nuestros algoritmos:

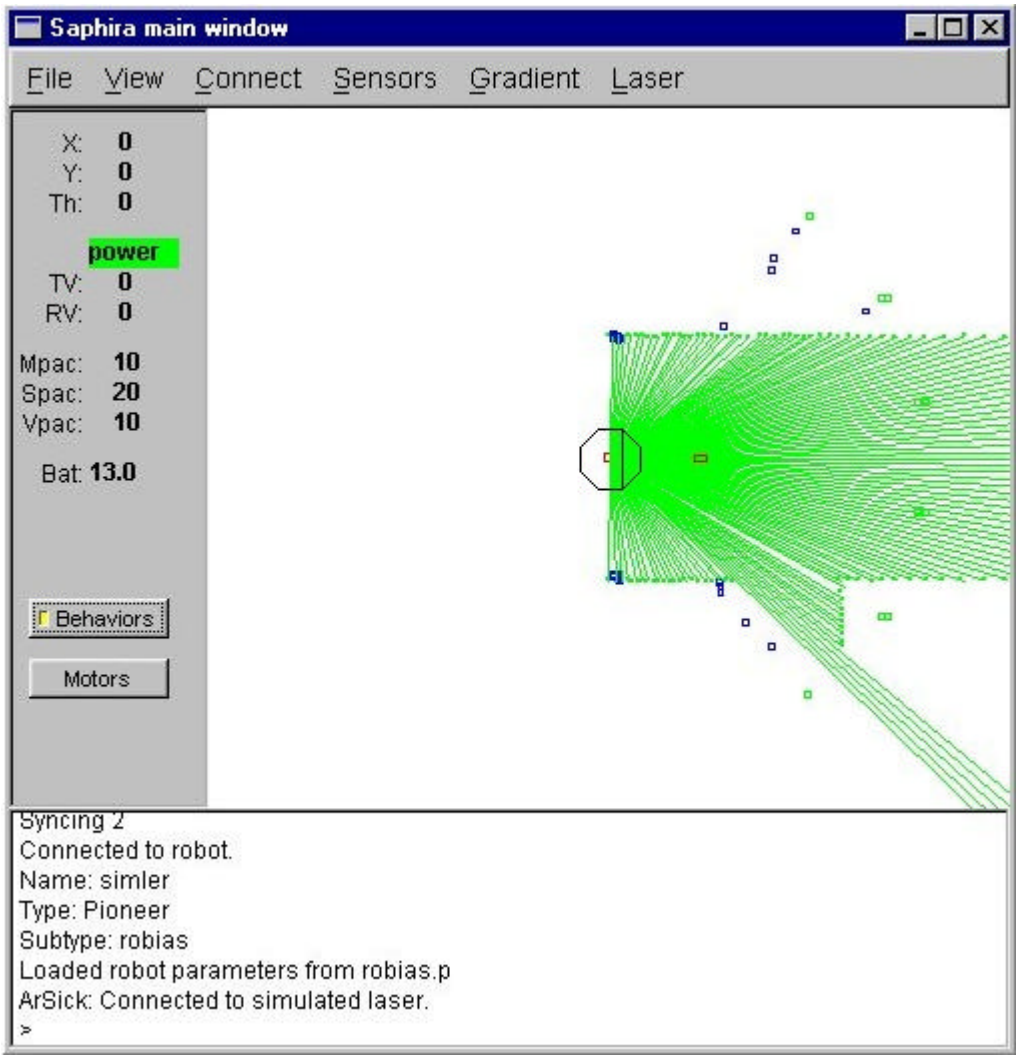


Figura 7: Medidas del láser

2.4 ActiveX: Comunicación con el Control de la visita guiada

Como ya dijimos con anterioridad, el proyecto completo de la visita guiada tiene dos partes: El software de navegación (Navigation software), que se corresponde con este proyecto fin de carrera, y el control de la visita guiada (Tour guide), realizado por otro estudiante. Ambos proyectos fueron realizados de forma independiente, pero fueron diseñados para funcionar de forma conjunta. Debido a ello hubo que desarrollar una interfaz entre ambas partes, para permitir la comunicación entre sí.

En el Instituto IAS se escribió en el pasado un objeto COM llamado WaveformAudioServer bajo Visual C++ para ofrecer una interfaz al reproductor multimedia de Microsoft para el robot RobIAS. Esto, unido a la arquitectura empleada para el proyecto de control de la visita guiada, hizo que debiéramos desarrollar una interfaz para dicho control de la visita usando la tecnología ActiveX-COM [1].

Por medio de esta interfaz ActiveX, el control de la visita guiada (Tour guide) puede acceder al software de navegación (Navigation software) para pasar el número del siguiente destino a ser alcanzado por el robot durante la visita guiada, y reanudar la navegación en el caso de que algún error lo hubiera interrumpido.

Una visión general de conjunto del sistema completo (Navigation software y Tour guide respetando la notación original del proyecto) se muestra en la Figura 8 [12]:

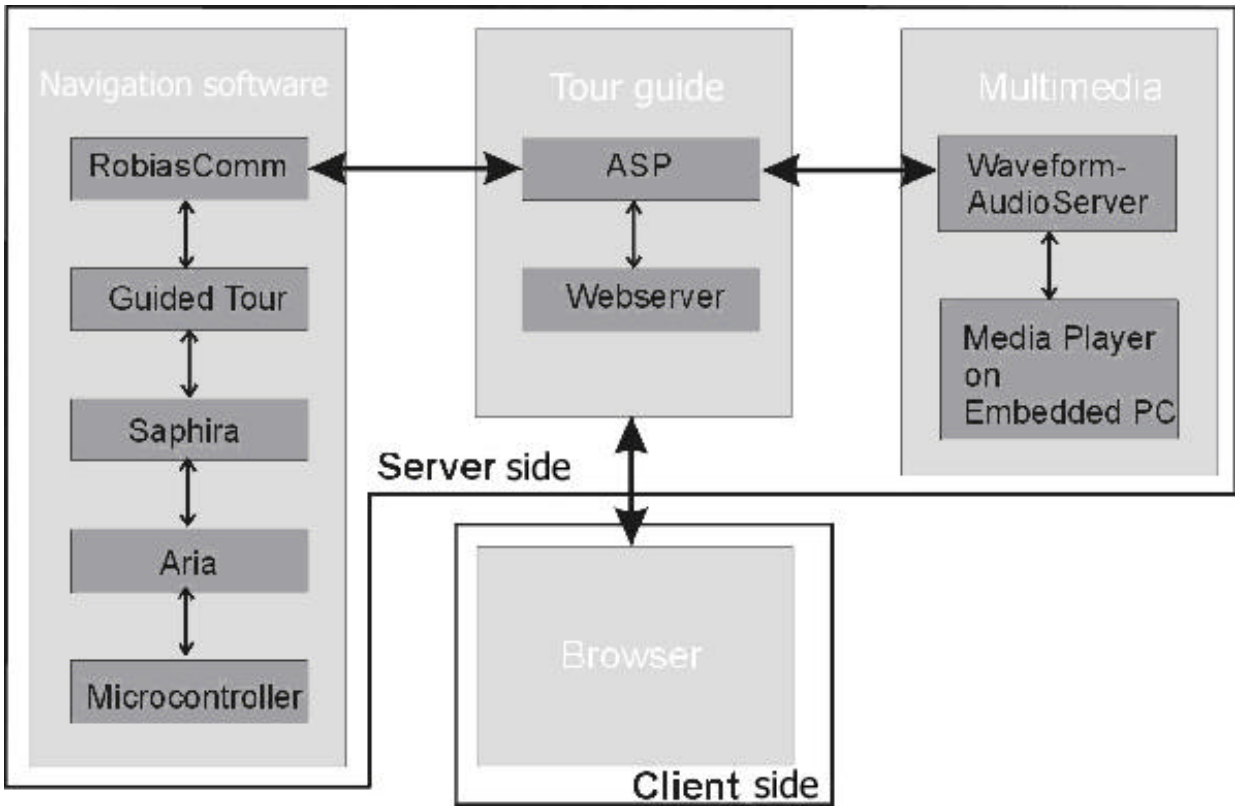


Figura 8: Visión general del sistema completo

3 Desarrollo del software de navegación

3.1 Situación de partida

Antes de comenzar el desarrollo del software de navegación tuve que dedicar algún tiempo a investigar, para conocer mi situación de partida, y éste fue el resultado:

- El robot ya estaba completamente equipado y era completamente funcional. Esto quiere decir, que no tuve que cambiar en lo más mínimo el hardware del robot.
- Ya había habido un proyecto en el Instituto IAS para controlar el robot móvil RobIAS a través de un joystick con una vieja versión de Saphira (en el momento que dicho programa fue desarrollado ARIA no estaba aún disponible). A pesar de que este proyecto no me ayudó en lo más mínimo para la navegación autónoma del robot, gracias a él pude tener una idea de las capacidades del robot y su problemáticas con la determinación de la posición. Este proyecto contenía además una interfaz ActiveX muy sencilla que me vino muy bien como punto de partida para el desarrollo de mi propia interfaz.

3.2 Decisiones de diseño

Como ya sabemos, el robot móvil RobIAS tiene por cometido navegar a través del Instituto IAS para guiar a los visitantes. Esta navegación implica ciertas actividades: Hay que calcular una secuencia de puntos intermedios a cruzar antes de que alcancemos el destino final. Una vez que esta secuencia se ha calculado, el robot debe seguirla, y guiado por sus sensores, encontrar un camino para alcanzar esos puntos y al mismo tiempo evitar los obstáculos que pudieran aparecer en el camino. Decidí usar el módulo de navegación por el método del gradiente (Gradient navigation) de Kurt Konolige [3], [4] como base para el cálculo del camino libre de obstáculos entre dos puntos intermedios, porque es muy potente y sencillo de implementar en los programas basados en Saphira.

Si un determinado punto intermedio no puede ser alcanzado, hay que encontrar uno alternativo, y si no existe ninguna alternativa válida, el robot es bloqueado (y hay que informar de esta situación). Todo esto no sería posible sin un método de localización fiable y sencillo de implementar que debe ser ejecutado de forma regular para determinar y corregir la posición y orientación real del robot.

Además, la comunicación con el control de la visita guiada (Tour guide) es otro aspecto muy importante a considerar, porque va a permitir transmitir el destino, informar de los trabajos completo, de situaciones de bloqueo y reanudar la navegación después de estas situaciones de bloqueo. Por eso, decidí usar la tecnología ActiveX-COM para programar una interfaz con el control de la visita guiada (Tour guide). Esta decisión no la tomé yo en exclusiva, sino que fue el fruto del trabajo conjunto con el otro estudiante participante en el proyecto global y el tutor de ambos. Hubimos de encontrar una solución intermedia que satisficiera a todas las partes y permitiera la compatibilidad entre los dos proyectos.

Debido a todo lo expuesto anteriormente, el programa se hizo constar de los siguientes componentes de software (cuyos nombres originales en inglés respeto):

- Componente **Navigation control** (Componente de control de la navegación), encargado de gestionar los demás componentes.
- Componente **Sequence of intermediate points planning** (Componente de planificación de la secuencia de puntos intermedios).
- Componente **Robot localization and position correction** (Componente de localización y corrección de la posición del robot).
- Componente **Obstacle-free path calculating** (Componente de cálculo de camino libre de obstáculos), que hace uso del módulo de navegación por el método del gradiente (Gradient navigation module)
- Componente **Alternative intermediate point finding** (Componente de búsqueda de puntos intermedios alternativos).
- Componente **Tour guide-Communication** (Componente de comunicación con el control de la visita guiada), que hace uso de la tecnología ActiveX.

Para la implementación de estos componentes decidí usar la versión más nueva disponible en aquel momento de Saphira y ARIA, porque estas versiones incluían una gran cantidad de características nuevas con respecto a la versión más antigua. Ésta fue la única elección que tuve que hacer al respecto, porque usar Saphira y ARIA con los robots Pioneer fabricados por ActivMedia Robotics no es un capricho, sino más bien una necesidad y la decisión más inmediata e inteligente.

3.3 Arquitectura del sistema

3.3.1 Visión general del sistema

En la Figura 9 se representa una visión general de la arquitectura del sistema. Los componentes que se mencionaron en la sección anterior se encuentran incluidos en esta figura (con sus nombres en inglés):

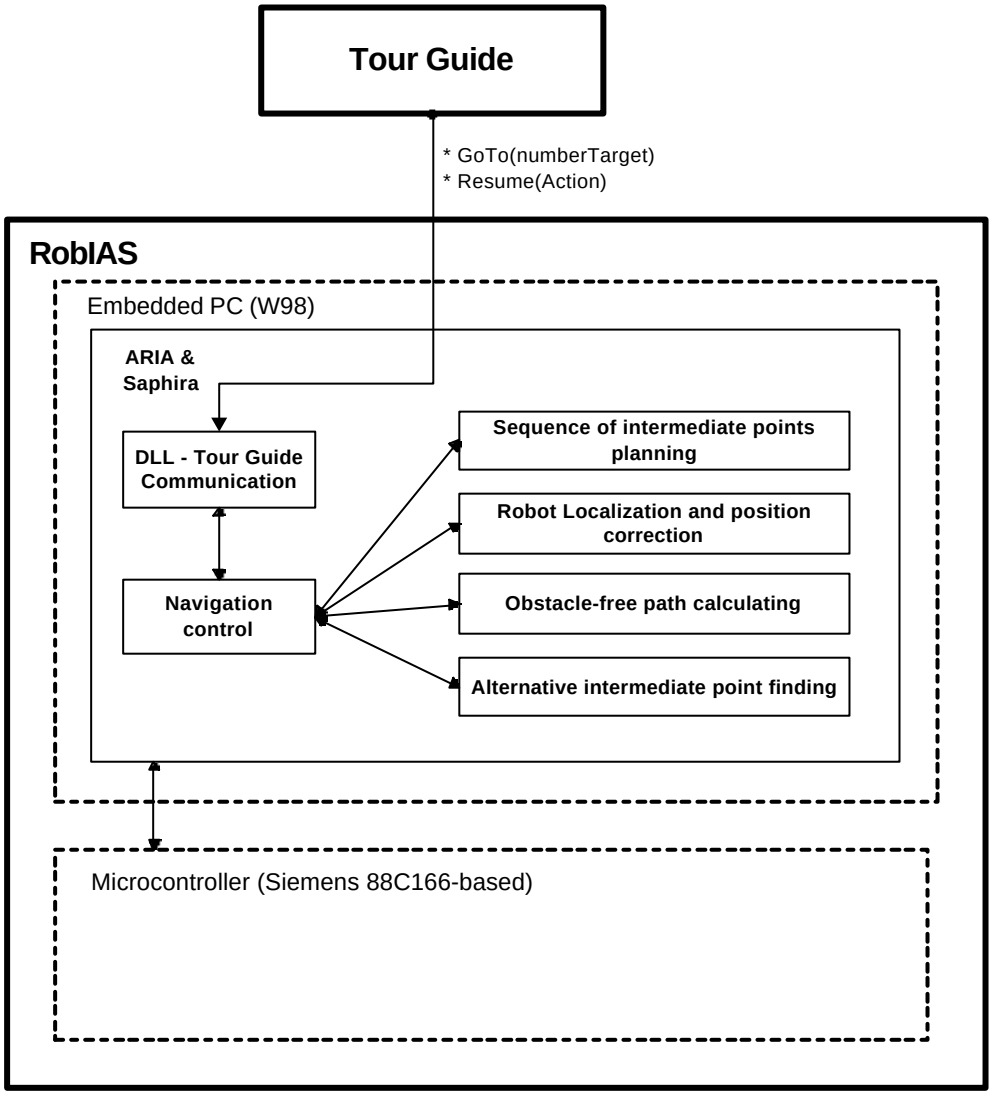


Figura 9: Visión general del sistema

Todos estos componentes están escritos en Visual C++ 6.0 usando las clases proporcionadas por ARIA y Saphira (con excepción hecha del componente „Tour guide-communication“, que no usa estas clases). La función del componente „Navigation Control“ es la gestión de los demás componentes, para lograr una perfecta sincronización entre los mismos (para este cometido

contamos con la ayuda de las *acciones* de ARIA, que pueden ser ejecutadas de forma paralela al programa principal).

El componente para la comunicación con el control de la visita guiada (Tour guide) se encuentra encapsulado en un fichero DLL, porque debe ser accesible al control de la visita guiada (Tour guide) usando la interfaz ActiveX definida.

3.3.2 Diagrama de estados

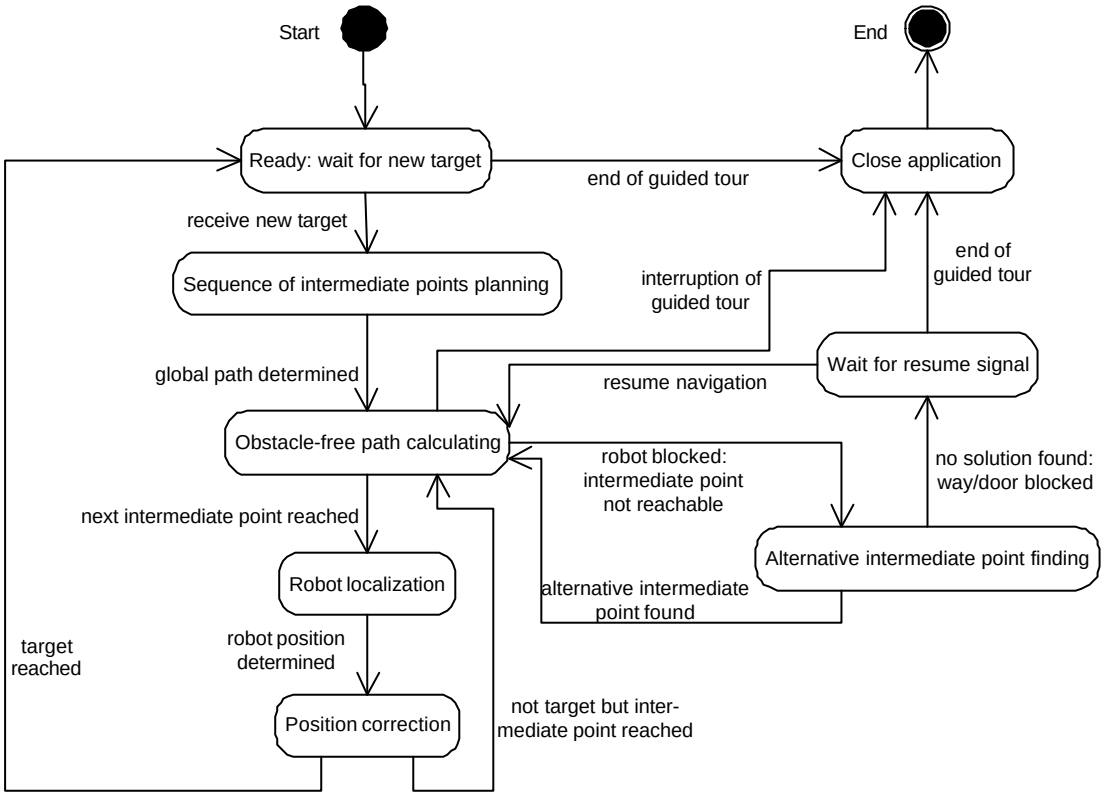


Figura 10: Diagrama de estados

Las correspondencias entre los estados del diagrama mostrado en la Figura 10 (y que se encuentran en inglés) y los componentes de software ya mencionados se encuentran en la siguiente tabla: (nótese que el componente „Navigation control“ no aparece en la tabla, porque su cometido es el de controlar el diagrama de estados completo; los demás componentes son invocados desde este componente central)

Estados del diagrama	Componentes de software
Ready: wait for new target	Tour guide-Communication
Wait for resume signal	
Sequence of intermediate points planning	Sequence of intermediate points planning
Obstacle-free path calculating	Obstacle-free path calculating
Robot localization	Robot localization and position correction
Position correction	
Alternative intermediate point finding	Alternative intermediate point finding

3.3.3 Diagrama de secuencia

En la siguiente Figura 11 se muestra cómo el sistema completo funciona (en inglés):

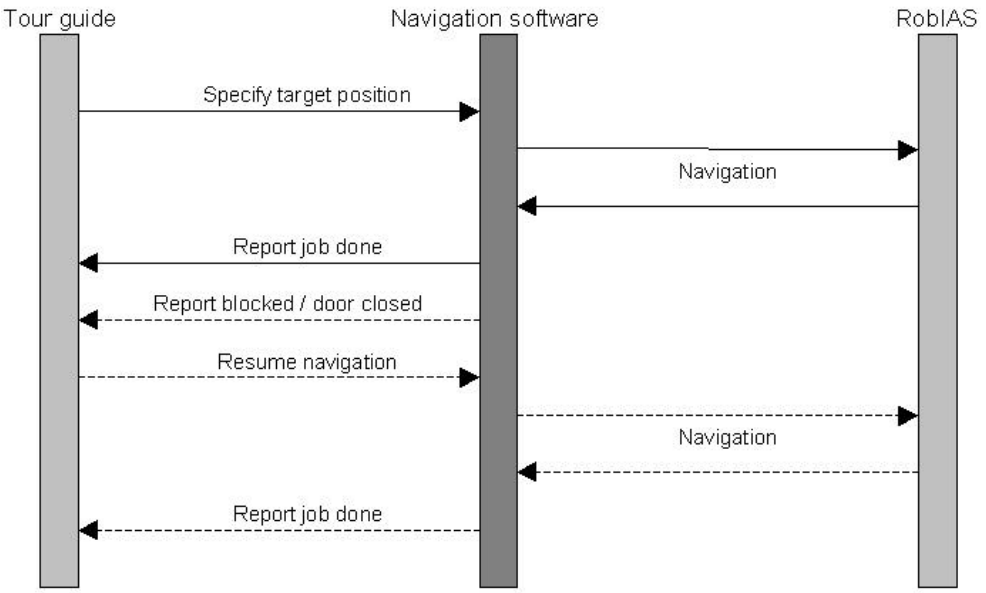


Figura 11: Diagrama de secuencia del sistema

El control de la visita guiada (Tour guide) envía la posición que debe ser alcanzada por el robot (*Specify target position*) al software de navegación (Navigation software), que controla la navegación del robot móvil RobIAS (*Navigation*) para lograr el objetivo propuesto. Si todo funciona sin problemas, el punto destino será alcanzado y se informará de esta circunstancia (*Report job done*) al control de la visita guiada (Tour guide). Pero también podría suceder que el robot se queda bloqueado debido a las puertas que están cerradas (y que deberían estar abiertas) o caminos bloqueados de tal forma que hacen imposible para el robot seguir navegando. En tal caso se envía un mensaje (*Report blocked / door closed*) al control de la visita guiada (Tour guide) para informar sobre esta circunstancia. El control de la visita guiada (Tour guide) debe

llevar a cabo las acciones pertinentes que permitan que el robot vuelva a navegar de nuevo (abrir la puerta o limpiar el camino de obstáculos, mediante, por ejemplo, el envío de mensajes hablados a los visitantes). Cuando el control de la visita guiada (Tour guide) estima que se puede reanudar la navegación, se envía el mensaje pertinente (*Resume navigation*) al software de navegación (Navigation software) y a partir de aquí se procede como en el caso anterior.

3.4 Componentes de software

Ahora vamos a desarrollar cada uno de los componentes de software que presentamos en su momento en la sección 3.2, y cuyos nombres originales en inglés respetaremos por razones de compatibilidad de nombres con el software.

3.4.1 Navigation control

La función principal de este componente (cuyo nombre podríamos traducir por „Componente de control de la navegación„) es la implementación del diagrama de estados presentado en la sección 3.3.2. Este componente invocará por tanto a los otros componentes y será la parte central de todo el software de navegación (Navigation software).

Las funciones de este componente son:

- Establecimiento de una conexión con el robot y con el sensor láser. Consideramos tanto el caso del robot real como del robot simulado.
- Inicialización de los parámetros de navegación.
- Carga de la información grabada sobre el entorno real del robot.
- Inicialización de las acciones que van a ser ejecutadas de forma paralela durante la navegación.
- Gestión de estas acciones.
- Implementación del diagrama de estados (que invoca a los otros componentes).
- Manejo de las situaciones de error.
- Terminación del programa (regular o solicitada por el usuario).

3.4.2 Sequence of intermediate points planning

Podríamos traducir el nombre de este componente por el de „Componente de planificación de la secuencia de puntos intermedios“. A continuación desarrollamos sus características.

3.4.2.1 Los puntos intermedios

Una vez que el punto destino ha sido determinado por el control de la visita guiada (Tour guide), tenemos que encontrar una trayectoria efectiva, corta y segura para mover el robot desde la posición inicial a la posición final.

El componente encargado de realizar dicha tarea no será un método general de planificación de trayectorias, sino uno más simple basado en el hecho de que el entorno real del Instituto IAS es conocido. Y no sólo eso: esta trayectoria no debería ser complicada en absoluto, debido a la estructura típica de oficina del Instituto, con pasillos rectos y anchos, esquinas de 90 grados y habitaciones rectangulares o cuadradas. Por tanto, no vamos a intentar encontrar una trayectoria completamente predeterminada, sino que usaremos un método más simple: la determinación de una secuencia de puntos intermedios entre las posiciones inicial y final. Estos puntos sí estarán predeterminados y el algoritmo se encargará de escoger una secuencia de los mismos. Para entender esto un poco mejor, echemos un vistazo a la Figura 12:

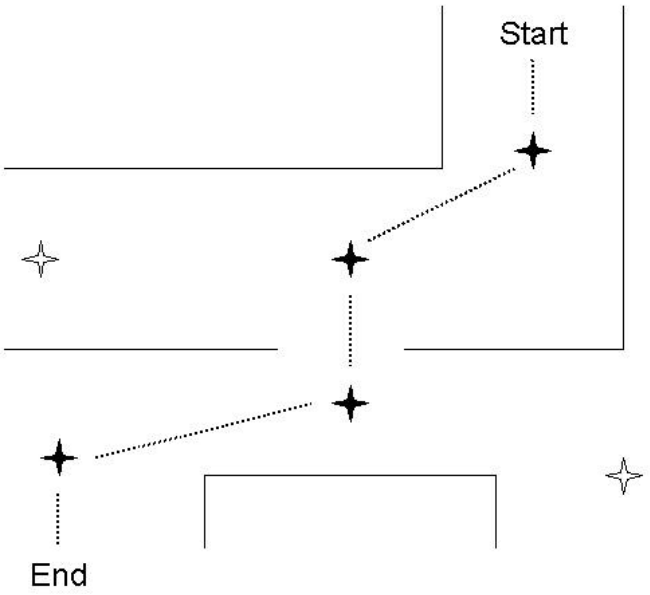


Figura 12: Puntos intermedios

Cada estrella representa un posible punto intermedio predeterminado. La idea es escoger entre todos estos puntos intermedios disponibles, una secuencia adecuada para ir del punto inicial (Start) al final (End), que definirá una **trayectoria** hacia el punto destino. Los puntos seleccionados están representados en la figura anterior con estrellas pintadas en color sólido, y la

trayectoria determinada por dichos puntos, con una línea discontinua. Nótese que no todas las secuencias de puntos son válidas, porque no tienen un significado físico coherente. En el ejemplo de la Figura 13 podemos ver esto con mayor claridad:

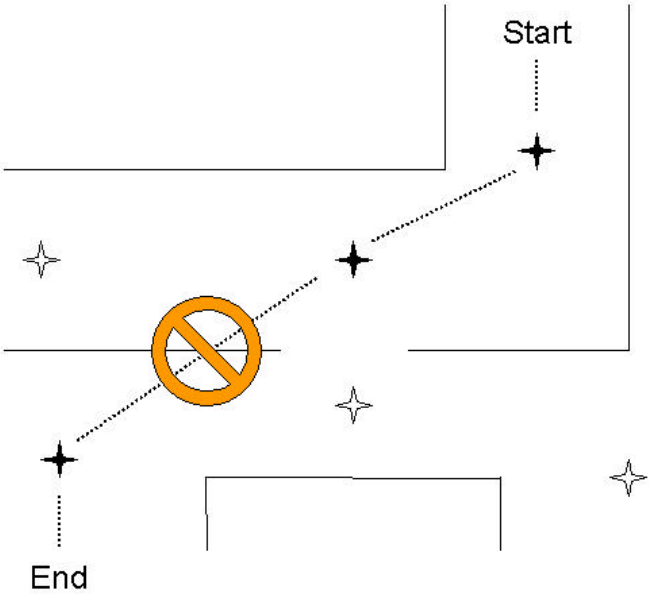


Figura 13: Secuencia inválida de puntos intermedios

En tanto que la trayectoria derivada de esta secuencia de puntos intenta abrirse paso a través de una pared, está solución no será nada buena. El robot podría intentar ir en la dirección incorrecta y nunca alcanzaría el destino de forma adecuada o podría necesitar mucho tiempo para es te cometido. Debemos recordar que estamos implementando la navegación para un robot que debe guiar a las personas, ¡y las personas no deberían seguir a un robot que no sabe hacia dónde va!

3.4.2.2 La representación con grafos

La solución al problema planteado anteriormente es almacenar no sólo los puntos intermedios disponibles, sino también las posibles transiciones entre los mismos. Por tanto, la solución lógica es representar esta información como un grafo no orientado, tal como se muestra en la Figura 14:

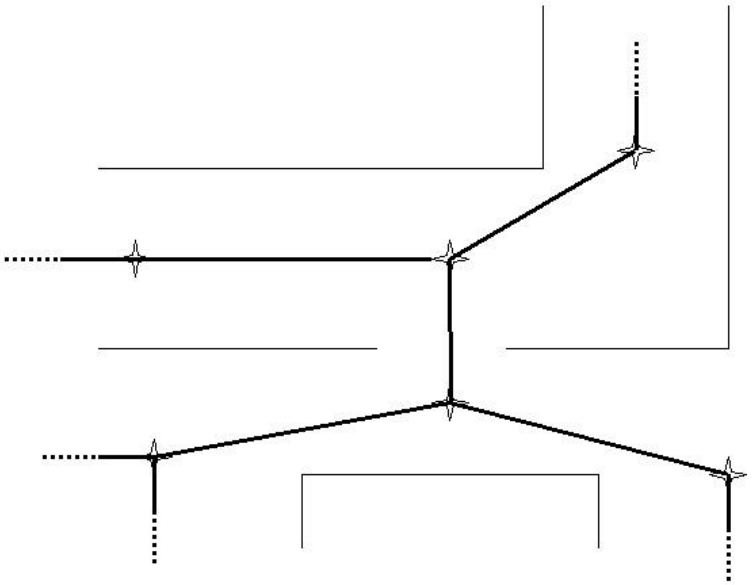


Figura 14: Representación con un grafo no orientado

3.4.2.3 El algoritmo de Dijkstra

Para hallar la secuencia de puntos intermedios vamos a usar un algoritmo muy conocido: el algoritmo de Dijkstra [5], que calcula el camino más corto entre un vértice y los demás en un grafo. Este algoritmo funciona de la siguiente forma:

Considérese un grafo **G=(V,E)**, donde **V** es el conjunto de vértices del grafo y **E** es el conjunto de aristas que conectan dichos vértices. El algoritmo de Dijkstra mantiene dos conjuntos de vértices:

- **S**: el conjunto de vértices cuyos caminos más cortos desde el vértice origen ya han sido determinados.
- **V-S**: los vértices restantes

Las otras estructuras de datos que se necesitan son:

- **d**: matriz con las mejores estimaciones de camino más corto a cada vértices.
- **pi**: matriz con los predecesores de cada vértice

El modo básico de operación es el siguiente:

1. Inicializa **d** (a infinito, excepto para el nodo origen, en cuyo caso d=0) y **pi** (a vacío, sin predecesores),
2. Fija **S** a vacío,
3. Mientras haya vértices en el conjunto **V-S**,
   3.1 Ordena los vértices en **V-S** de acuerdo con la mejor estimación actual de sus distancias al vértice origen,
   3.2 Añade **u**, el vértice más cercano en **V-S**, a **S**,
   3.3 **Relaja** todos los vértices que aún estén contenidos en **V-S** y estén conectados a **u**

Relajación

El proceso de **relajación** actualiza los costes de todos los vértices, **v**, conectados al vértice, **u**, si podemos mejorar la mejor estimación del camino más corto a **v** mediante la inclusión de la arista **(u,v)** en el camino a **v**.

El procedimiento de relajación comprueba si la mejor estimación actual de la distancia más corta a **v** (**d[v]**) puede mejorarse si se va a través de **u** (haciendo **u** el predecesor de **v**). Un posible código fuente para implementar esta relajación (para que podamos entender mejor de lo que estamos hablando) podría ser el siguiente:

```
relax( Node u, Node v, double w[][] )
    if d[v] > d[u] + w[u,v] then
       d[v] := d[u] + w[u,v]
       pi[v] := u
```

3.4.2.4 Aristas con diferentes costes

El algoritmo de Dijkstra tiene otra ventaja añadida: las aristas del grafo pueden tener diferentes costes, y podemos usar esta característica para forzar al robot a "preferir" una trayectoria con menor coste. En la Figura 15 mostramos esto mejor:

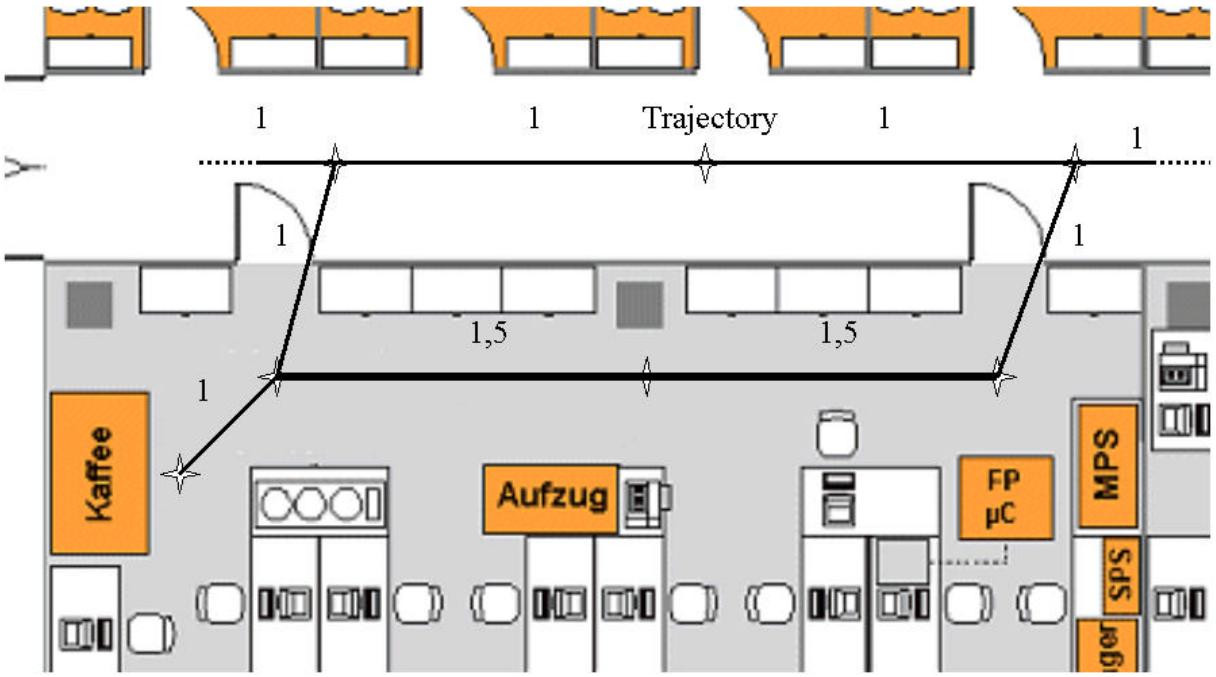


Figura 15: Planificando trayectorias con diferentes costes

Representamos las aristas de alto coste con una línea de mayor grosor. Es obvio que a través de la trayectoria superior la navegación es más fácil, porque no encontramos sillas y otros mobiliarios que nos estorben, sino que vamos a través de un pasillo que presumiblemente está libre de obstáculos de ese tipo. Por medio de los diferentes costes de las aristas podemos forzar al robot para que vaya por el camino más sencillo.

El coste de cada arista será entonces la distancia física entre los dos puntos a los que representan los vértices conectados, multiplicada por un cierto factor de "dificultad de navegación" (que en la Figura 15 se encuentran representados junto a las aristas). Como se ha dicho, estas aristas tendrán un coste mayor si implican navegación dentro de una habitación, con sillas, mesas y obstáculos similares.

Después de la ejecución de este algoritmo, tendremos una secuencia óptima de puntos intermedios que deben ser seguidos por el robot. Esta información será usada para calcular el camino en tiempo real que realmente se va a seguir (ver sección 3.4.4).

3.4.3 Robot localization and position correction

Podríamos traducir el nombre de este componente por el de „Componente de localización y corrección de la posición del robot“. A continuación desarrollamos sus características.

3.4.3.1 Motivación

La localización del robot es un paso imprescindible en nuestro procedimiento de navegación. Como ya se dijo anteriormente (ver subcapítulo 2.3.3), cuando el robot se mueve, el microcontrolador integrado en el robot calcula la posición y orientación reales del robot de acuerdo con las medidas tomadas por los codificadores montados en las ruedas. La integración de dichas medidas usando el modelo cinemático del robot determina la posición y orientación reales del mismo. Pero ya sabemos que esto proceso no está libre de error, y que las desviaciones, que al principio son pequeñas, se van haciendo cada vez más grandes con cada metro recorrido por el robot. Si no se realizan correcciones a la orientación y posición del robot, la navegación se hace imposible porque los errores son tan grandes, que la posición del robot determinada ya no tiene nada que ver con la posición real del mismo (tal y como se muestra en la Figura 16):

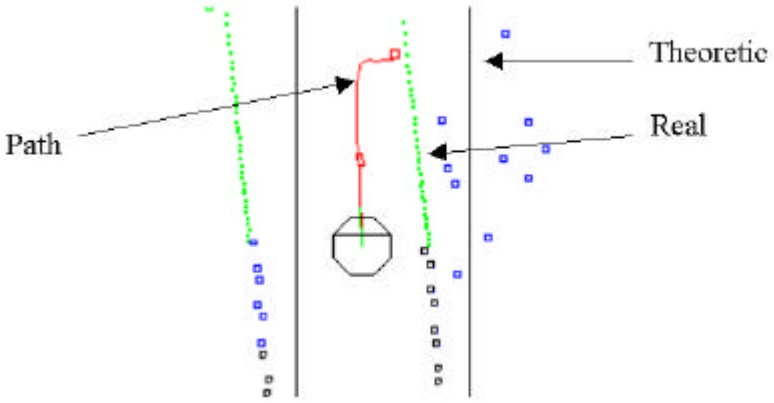


Figura 16: Desviaciones entre la posición y orientación teóricas y reales del robot

Las líneas negras continuas representan el modelo bidimensional del entorno almacenado por el robot, y constituyan la referencia para el robot; es decir, la posición y orientación teóricas del robot son relativas a estas líneas. Las líneas verdes discontinuas representan las medidas del sensor láser, y la posición y orientación reales del robot son relativas a estas líneas. Así, vemos que la posición y orientación teóricas y reales del robot no coinciden, y este error es grande (puede ser incluso sin muchos problemas mucho mayor del representado en la figura, que es sólo una captura de una simulación). Debido a esto, es obvio por qué debemos localizar al robot y corregir su posición, porque si no lo hacemos, el robot se pierde fácilmente y la navegación no tiene éxito.

3.4.3.2 Métodos de localización del robot

Para el algoritmo encargado de localizar el robot usaremos ciertas propiedades conocidas del mapa para determinar la posición y orientación reales del robot de una forma exacta (o muy aproximada). Para ello realizamos una comparación entre la información esperada y aquella obtenida del dispositivo láser integrado en el robot, que nos suministra una información muy detallada del entorno real del robot.

a) Localización con esquinas:

Figura 17: Posibles configuraciones de esquinas: a.1) cóncava, a.2) convexa

Como podemos ver en estas dos capturas tomadas de Saphira en la Figura 17, el dispositivo láser (cuyos puntos de medida están representados con líneas discontinuas verdes) nos da una información muy exacta que puede ser usada para la localización del robot. Tenemos suficientes puntos para realizar una regresión lineal y obtener los parámetros de la esquina (explicaremos esto de una forma más detallada en la sección 3.4.3.3). Mostraremos a continuación que es posible determinar la posición y orientación exactas del robot usando estas medidas del sensor láser. Asumiremos ahora que ya hemos parametrizado de alguna forma la esquina con la información del láser, y que tendremos la información mostrada en la Figura 18(se corresponde con el caso a.1) de esquina cóncava mostrado en la Figura 17; el caso a.2) es completamente análogo):

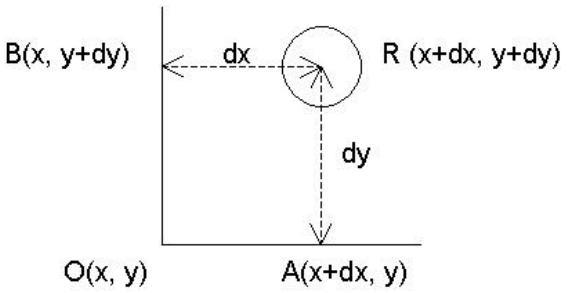


Figura 18: Determinación de la posición y orientación del robot en una esquina cóncava

Una vez que la esquina ya ha sido parametrizada usando la información real proporcionada por el sensor láser, las coordenadas de la esquina O(x,y) respecto al sistema de referencia del robot son conocidas. Del mismo modo, también podemos determinar la orientación del robot y las distancias (dx, dy) respecto a las paredes de la esquina. De esta forma, la orientación real actual del robot se puede determinar fácilmente comparando la orientación medida del robot con respecto a la esquina y la orientación conocida (y almacenada) de la esquina real. La posición del robot R(x+dx,y+dy) también se puede determinar fácilmente, porque las distancias dx y dy son ya conocidas, y la posición de la esquina O(x,y) también. Todo el proceso se reduce a la comparación de la información (real) proporcionada por el sensor láser y la información (teórica) almacenada, lo cual nos permite corregir las desviaciones que se hubieran producido hasta el momento en la orientación y posición del robot. Asumiremos además que las esquinas pueden tener cualquier orientación posible a la hora de implementar el algoritmo, pero la realidad es que en el entorno real del Instituto IAS las esquinas tienen solamente cuatro posibles orientaciones.

Como podemos ver, una sencilla comparación puede determinar la posición y orientación reales del robot. Para el caso de la esquina convexa, los razonamientos son totalmente análogos (de hecho, el software de navegación calcula de la misma forma la orientación y posición de las esquinas para ambos casos, cóncava y convexa).

b) Localización con paredes

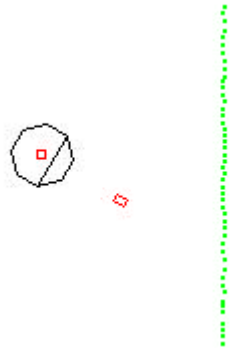


Figura 19: Localización con paredes

Para este caso mostrado en la Figura 19, tenemos que proceder como con las esquinas, comparando la orientación real medida y la distancia a la pared con los valores teóricos de los mismos. Asumiremos que todas las paredes del Instituto IAS son perpendiculares las unas a las otras, de forma que la dirección de una pared siempre se corresponda a uno de los ejes del sistema de referencia cartesiano que estamos usando, el vertical o el horizontal. La gran diferencia con las esquinas es que ahora seremos capaces de determinar solamente una de las coordenadas de la posición del robot y no ambas, porque no existe diferencia alguna entre la posición medida y otra trasladada más allá en la dirección de la pared. Por tanto, con las paredes

verticales sólo podremos determinar la coordenada x, y con las paredes horizontales, sólo la coordenada y.

c) Localización con pasillos

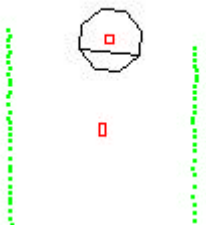


Figura 20: Localización con pasillos

Este caso mostrado en la Figura 20 es bastante similar al de localización con paredes, porque sólo podemos determinar la orientación del robot y una de las coordenadas de la posición, pero no ambas. La cuestión radica en que no podemos obtener más información si tenemos dos paredes, pero esta información será más precisa, porque conocemos además la distancia entre las dos paredes que forman el pasillo.

La elección del método de localización

A la hora de escoger el método de localización a usar en un determinado punto, el procedimiento general será elegir una esquina como método de localización siempre que sea posible (porque con ellas podemos localizar completamente el robot), y si no, escoger una pared o un pasillo. Por una parte, si trabajamos sólo con una pared, podríamos cometer el error de no distinguirla de un obstáculo plano (tal como una caja de cartón de grandes dimensiones), y entonces tendríamos un error bastante grande, pues estaríamos un obstáculo como referencia para la localización. Pero por otra parte, el empleo de paredes es más flexible que el de pasillos, y se puede utilizar no sólo con paredes en sí, sino también con armarios y otro mobiliario de grandes dimensiones (que pueden ser modelados parcialmente como paredes según nuestros intereses). Y no sólo eso: si usamos una pared para localizar al robot, y en el lado opuesto a la pared existe un obstáculo, éste no afectará en lo más mínimo a la localización del robot, mientras que si usamos un pasillo basta con bloquear con un obstáculo una de las paredes para que la localización no funcione. Por decirlo de algún modo, la localización con pasillos es el doble de sensible a obstáculos que la localización con paredes. Por tanto, si tenemos que escoger entre pasillos y paredes, usaremos pasillos siempre que sea posible (porque proporcionan una localización más fiable), pero combinándolos con paredes para asegurarnos de que el robot siempre tiene posibilidades de localizarse.

3.4.3.3 Reconocimiento del entorno

3.4.3.3.1 Visión de conjunto

Ya sabemos qué elementos del entorno real del robot necesitamos: esquinas, pasillos y paredes. Ahora el problema a resolver es cómo reconocemos dichos elementos a partir de la información de los sensores.

El primer paso será siempre dividir la información obtenida a partir del láser en líneas. Con estas líneas podremos encontrar las esquinas, pasillos y paredes que andamos buscando.

3.4.3.3.2 Encontrando líneas con el sensor láser

El sensor láser escanea en un rango de 180 grados con una precisión de medio grado (es decir, el láser „lanza“ rayos con incrementos de medio grado). Con esto obtendríamos una matriz de 360 puntos, que serán normalmente menos porque ARIA filtra de forma automática los puntos que están muy cercanos, debido a que no añaden información adicional, con lo que el tamaño de la matriz se reduce a menos de 360 puntos. Esta matriz de puntos contiene la información „bruta“ que nosotros hemos de convertir en información útil para el robot. Los pasos a realizar para llevar a cabo esta tarea son los siguientes:

En una primera fase los puntos se agrupan en *bloques* (*blocks*). El criterio para decidir si un punto pertenece a un bloque u otro es la distancia al punto anterior. Si dicha distancia es menor que un cierto límite predefinido, podemos considerar que ambos puntos pertenecen al mismo bloque. Si por el contrario la distancia es mayor, entonces el punto pertenece a un nuevo bloque.

A continuación eliminamos los bloques que tengan menos de un cierto número mínimo de puntos, porque estos bloques no contienen información que podamos usar para encontrar elementos de localización.

Los bloques que pasan el proceso de filtrado anterior definirán cada uno una nube de puntos. Mediante el algoritmo de extracción de líneas de Gutmann [11] extraemos las líneas de estas nubes de puntos. Éste es un típico algoritmo de filosofía „divide y vencerás“: cada bloque es aproximado mediante una regresión lineal y se calcula la desviación típica estándar s de la linearización. Usamos la forma normal de Hesse para representar la ecuación de la línea así obtenida:

$$x\cos\alpha + y\sin\alpha - d = 0$$

Para determinar los valores de α, σ y d tenemos que usar las fórmulas siguientes [11]:

$$\alpha = \frac{1}{2}\arctan\frac{-2S_{xy}}{S_{y^2} - S_{x^2}}$$

$$d = \bar{x}\cos\alpha + \bar{y}\sin\alpha$$

$$\sigma^2 = \frac{1}{2}\left(S_{x^2} + S_{y^2} - \sqrt{4S_{xy}^2 + (S_{y^2} - S_{x^2})^2}\right)$$

donde se usa las siguiente notación:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$S_{x^2} = \sum_{i=1}^{n}(x_i - \bar{x})^2 \qquad S_{y^2} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Si la desviación estándar σ del bloque es menos que cierto valor máximo predefinido σ_{max} entonces la aproximación lineal se da por buena y el bloque es ya una línea, y no tenemos que realizar ninguna operación adicional sobre este bloque.

Pero si la desviación estándar σ del bloque es mayor que dicho valor máximo predefinido σ_{max} entonces la aproximación lineal es mala y tenemos que dividir la nube de puntos en dos nuevas nubes (dando lugar a dos nuevos bloques), por el punto en el que la distancia a la regresión lineal es máxima (y siempre que dicho punto no coincida con los límites del bloque). Las dos nuevas nubes de puntos (o bloques) serán de nuevo filtradas con el mínimo número de puntos necesario para ser consideradas bloques (de forma que si tienen menos puntos que ese límite se eliminan) y entonces se vuelven a aproximar con una regresión lineal como se hizo anteriormente. Este proceso se repite de forma recursiva hasta que no haya más bloques o todos los bloques restantes puedan ser ya considerados como líneas.

Después de todas estas operaciones tendremos un conjunto de líneas completamente parametrizadas, que usaremos para nuestros procesos de localización. Ya no nos hará falta a información „bruta“ de los puntos (hasta el próximo ciclo de 100 ms, claro está, donde volveremos a tener nueva información en forma de matriz de puntos), sino los parámetros calculados a partir de ella (α, σ y d, el número de puntos y la posición del punto medio).

3.4.3.3.3 Encontrando esquinas

Para nuestros propósitos de localización, una esquina no será más que un conjunto de dos líneas conectadas por un punto común y que forman un ángulo de aproximadamente 90 grados. Para ver si las líneas están conectadas, el algoritmo que divide los bloques en nuevos subbloques mantiene las referencias de conexión entre los nuevos bloques creados. De esta forma, podemos saber cuándo dos líneas están conectadas o no, además del punto de conexión (el punto en el que el bloque original fue dividido).

3.4.3.3.4 Encontrando pasillos

El proceso de encontrar un pasillo es un poco más elaborado que el de encontrar esquinas. En este caso, tenemos que agrupar todas las líneas existentes en grupos atendiendo a su orientación. Dentro de cada grupo así formado, comparamos cada línea con las demás, para calcular la distancia que las separa, y las distintas combinaciones se evalúan atendiendo al ajuste con el pasillo teórico y al número de puntos total que ambas líneas suman. La combinación mejor evaluada se toma como pasillo detectado para ese grupo si satisface los requerimientos mínimos para ser considerado un pasillo. Este proceso se repite para cada grupo, aunque normalmente no encontraremos más de un pasillo por vez (ocasionalmente dos).

3.4.3.3.5 Encontrando paredes

Encontrar paredes es el más sencillo de los casos que hay. Para ello solamente tenemos que realizar un segundo proceso de filtrado con el número de puntos contenidos en la línea. Aquí el filtrado es más estricto que para la división en líneas, es decir, el mínimo número de puntos para considerar una línea como una pared es mayor que el mínimo número de puntos original de una línea.

3.4.3.4 Corrección de la posición

Una vez que hemos reconocido los elementos que vamos a usar para nuestra localización, tenemos que efectuar la localización del robot en sí, es decir, comparar las referencias reales detectadas y las teóricas almacenadas para determinar la posición y orientación reales del robot en ese momento.

El siguiente paso serán entonces la corrección de la posición. Para hacer esto no vamos a mover el robot a la posición esperada por el software de navegación (Navigation software) después de que se hayan descubierto los errores de posición, sino que actualizaremos los valores de la posición y orientación del robot usados por el software de navegación a los valores reales, para que no existan discordancias entre ambos. Dicho de otra forma, el software de navegación (Navigation software) ve cómo el robot “salta” a su nueva posición, pero externamente sólo vemos que el robot reacciona a la corrección y se dirige al siguiente punto intermedio por el

camino correcto. Esto lo hacemos de esta forma porque no tendría sentido realizar un movimiento adicional para esta corrección. La excepción a la regla la constituirá el punto final, porque en este caso no querremos sólo que el robot conozca que está en una posición final incorrecta, sino que además se mueva a la posición final deseada. No sería muy presentable ver a un robot hablando a los visitantes en la dirección o posición incorrecta por no haber ejecutado la última corrección de posición.

3.4.4 Obstacle-free path calculating

Podríamos traducir el nombre de este componente por el de „Componente de cálculo de camino libre de obstáculos“ . A continuación desarrollamos sus características.

3.4.4.1 Visión de conjunto

Después de haber calculado la secuencia de puntos intermedios (ver subcapítulo 3.4.2), tenemos una lista con los puntos intermedios que deben ser alcanzados de forma consecutiva para llegar al destino. Con esta información y la información que suministran los sensores (láser y sonares), un camino reactivo debe ser calculado y seguido. Esta acción se ejecutará de forma concurrente cada 100 ms para reaccionar a los obstáculos que puedan aparecer en el camino.

3.4.4.2 El módulo “Gradient navigation”

La herramienta básica para calcular un camino óptimo será el módulo de navegación por el método del gradiente (Gradient navigation module) [3], [4] incluido en la distribución de Saphira. Éste calcula el camino de mínimo coste al destino deseado atendiendo a las medidas del láser y los sonares, velocidad actual y, opcionalmente, el mapa conocido del Instituto. En la Figura 21 se muestran dos ejemplos:

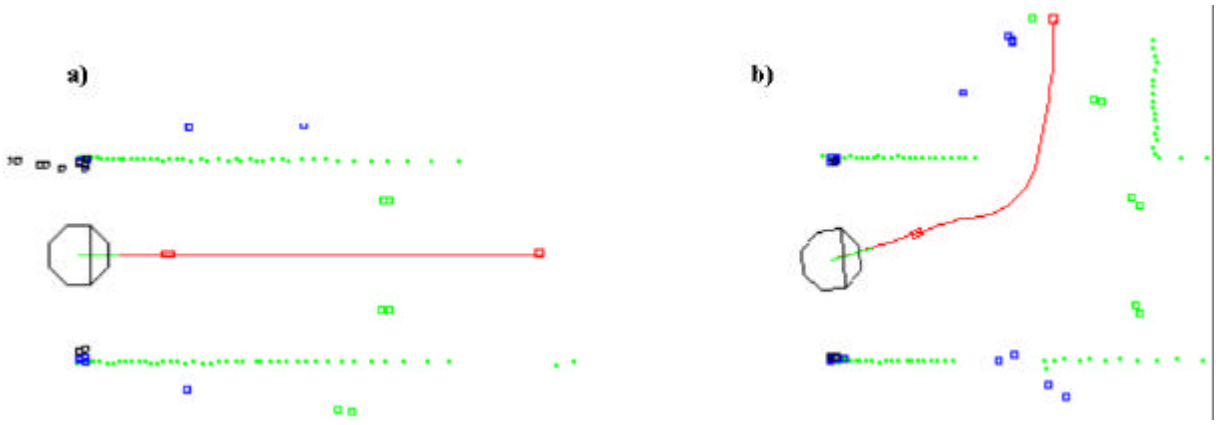


Figura 21: Camino de la navegación por el método del gradiente en a) pasillos, b) esquinas de 90 grados sin obstáculos

En esta figura las líneas verdes discontinuas representan las medidas del láser, los pequeños cuadraditos, las medidas de los sonares, y la línea continua roja, el camino trazado por el modulo de navegación por el método del gradiente. Como podemos ver, dicho método calcula un camino muy eficiente que considera la presencia de obstáculos (ver b) en la Figura 21 cuando se evita la esquina). El mapa ya conocido también se le puede proporcionar al algoritmo, pero tenemos que ser muy cuidadosos con esto: los errores acumulados durante la navegación producen una discordancia entre las referencias almacenadas y las medidas, y esto puede conducir a un comportamiento inadecuado. Esto se muestra en la siguiente Figura 22:

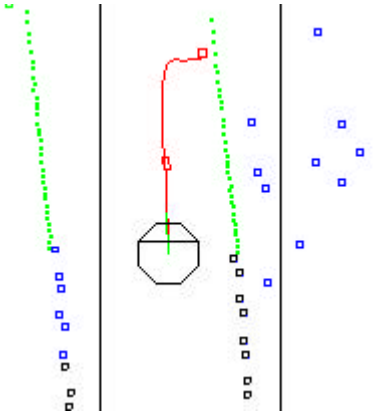


Figura 22: Problemas de usar el mapa conocido en la navegación por el método del gradiente

Las líneas verdes discontinuas inclinadas se corresponden con los puntos medidos por el láser, y las líneas continuas negras se corresponden con el mapa almacenado. Las coordenadas actuales del robot son relativas al mapa almacenado, pero la posición real es diferente. En esta figura el robot quiere alcanzar un punto intermedio situado en un pasillo, pero se estrellará contra la pared derecha (línea verde discontinua). La conclusión inmediata sería que no podemos usar el mapa almacenado para la navegación por el método del gradiente, pero la realidad es que tenemos que usarlo, porque hay obstáculos que no se pueden detectar con el láser o los sonares, como por ejemplo algunas mesas, y ese detalle no puede ser ignorado para nuestra navegación. Esto causará que el robot se pierda más fácilmente, pero evitamos colisiones con esos obstáculos “invisibles”, y preferimos que el robot se pierda a que se estrelle contra una mesa.

Veamos ahora qué pasa cuando un obstáculo que no está almacenado en nuestro mapa es detectado por los sensores del robot. La ilustra precisamente este caso:

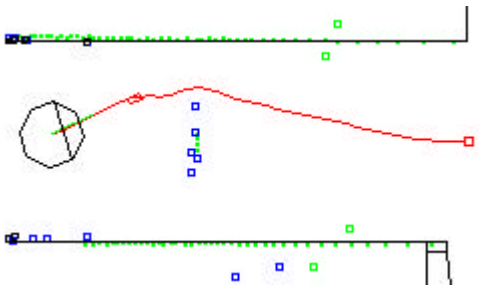


Figura 23: Evitación de obstáculo realizada por el módulo de navegación por el método del gradiente

El obstáculo no aparece en el mapa conocido del Instituto IAS, y por tanto, es inesperado. Pero los sensores del robot (los sonares, cuyas medidas aparecen representadas con pequeños cuadraditos de color azul, y el láser, cuyas medidas son los puntos de color verde) lo detectan y el módulo de navegación por el método del gradiente encuentra en tiempo real un camino alternativo que evita el obstáculo. Por tanto, para obstáculos sencillos, fijos y de pequeño tamaño (e incluso obstáculos móviles en determinados casos), el módulo de navegación por el método del gradiente nos ofrece una navegación muy eficiente casi sin programación extra.

3.4.4.3 Alcance de un punto intermedio

El paso siguiente es decidir cuándo hemos alcanzado el punto intermedio. Además vamos a considerar qué ocurre cuando el punto no es alcanzable o cuando implica navegación sin sentido. Por tanto, vamos a distinguir los siguientes casos principales:

a) El punto intermedio es fácilmente alcanzable. La Figura 24 muestra este caso:

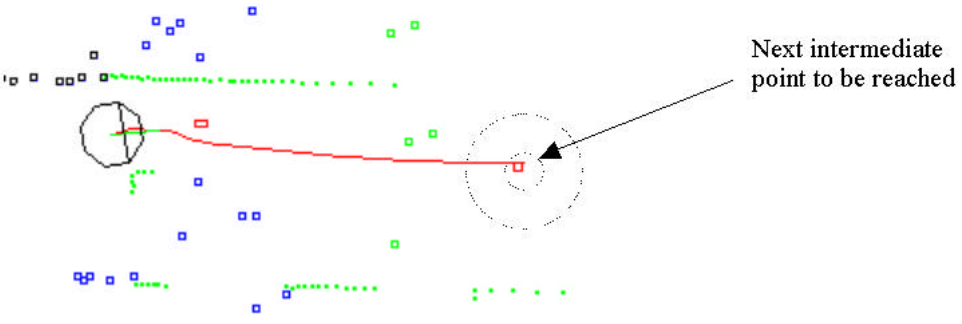


Figura 24: Punto intermedio fácilmente alcanzable

El obstáculo que se está esquivando no representa ningún tipo de problema, porque el punto intermedio a alcanzar está lejos del mismo y no se encuentra obstruido. Cuando el obstáculo es esquivado, la navegación continúa normalmente.

Si este punto intermedio no es el último, vamos a estar interesados en pasar de largo sin alcanzarlo de forma completamente exacta. Por tanto, definimos un área circular alrededor del punto, de forma que cuando el robot alcance los límites de esta área, se dirija al siguiente punto intermedio "olvidando" éste. Si por el contrario el punto intermedio es el último de la secuencia, es decir, es el punto destino, entonces sí viajaremos hacia el punto de forma exacta.

b) El punto intermedio no es fácilmente alcanzable (pero se puede alcanzar). Esta situación se corresponde con la mostrada en la Figura 25:

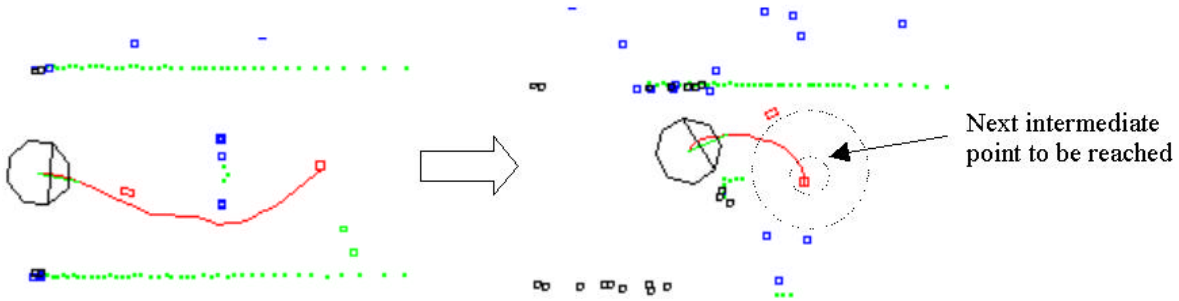


Figura 25: Punto intermedio no fácilmente alcanzable (cercano a un obstáculo)

Como resultado de la cercanía entre el obstáculo y el punto intermedio, observaremos un comportamiento inestable en una primera fase del cálculo del camino. Además, se lleva a cabo una aproximación inútil al punto intermedio. La solución al problema será, como en el caso anterior, definir un área circular alrededor del punto intermedio, de forma que cuando el robot la alcance, se dirija hacia el próximo punto intermedio de la secuencia.

c) El punto intermedio es inalcanzable (coincide con un obstáculo). Representamos este caso en la Figura 26:

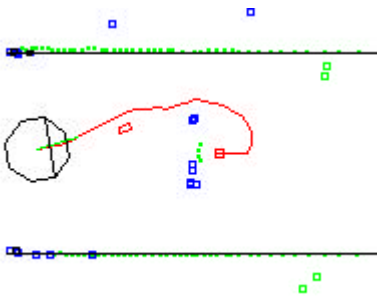


Figura 26: Punto intermedio inalcanzable

Este caso es mucho más problemático, y no puede resolverse de forma tan sencilla definiendo simplemente un área circular alrededor del punto intermedio (la ejecución correspondiente a la simulación a la que pertenece la captura mostrada tuvo como resultado una oscilación indefinida alrededor del punto intermedio, al usar solamente la navegación por el método del gradiente antes de implementar la solución al problema). Tenemos que detectar esas situaciones (que son muy parecidas a las situaciones de bloqueo, como veremos más tarde), y después invocaremos el componente "Alternative intermediate point finding" para determinar un nuevo punto intermedio a ser alcanzado por el robot.

3.4.4.4 Detección de "Camino bloqueado" y "Puerta cerrada"

El componente "Obstacle-free path calculating" tiene otros cometido importante: comprobar si el camino calculado está bloqueado, y si las puertas que deben cruzarse están cerradas (o bloqueadas, o no abiertas adecuadamente, no se hacen distinciones al respecto). En principio hay dos tareas diferentes:

a) **Detección de "Puerta cerrada" ("Door closed")**. Esta comprobación no se debe realizar de forma continua, porque no siempre tenemos que estar cruzando puertas. Tenemos que comprobarlo sólo cuando la puerta entre en acción, debido a lo cual tendremos que marcar de alguna forma las puertas que se necesiten como puntos especiales en nuestra secuencia de puntos intermedios. Estas puertas deben ser alcanzadas con la suficiente precisión (hay que efectuar una localización del robot inmediatamente antes de intentar cruzarlas) y comprobar si la apertura esperada está presente. Si no se detecta esta apertura, o no tiene los parámetros correctos, consideraremos que la puerta está cerrada (o bloqueada, pero en este caso tendrá el mismo significado). La solución será invocar el componente "Alternative intermediate point finding" (ver subcapítulo 3.4.5), que determinará, de ser posible, un nuevo punto intermedio alternativo a alcanzar. Si no fuera posible, esta circunstancia será comunicada al control de la visita guiada (Tour guide).

b) **Detección de "Camino bloqueado" ("Way blocked")**. En contraste con la detección de "Puerta cerrada", esta comprobación debe ser realizada de forma continua, porque el camino podría estar bloqueado en cualquier punto de la trayectoria planificada. El caso es bastante similar al caso "El mundo intermedio es inalcanzable (coincide con un obstáculo)" presentado con anterioridad (ver subcapítulo 3.4.4.3). La solución será, como en el caso a) que acabamos de describir, invocar el componente "Alternative intermediate point finding" (ver subcapítulo 3.4.5), que dilucidará si el robot está realmente bloqueado o no, y determinará, si es posible, un nuevo punto intermedio alternativo a ser alcanzado por el robot.

En la Figura 1Figura 27 tenemos un ejemplo de un obstáculo no evitable que bloquea el camino:

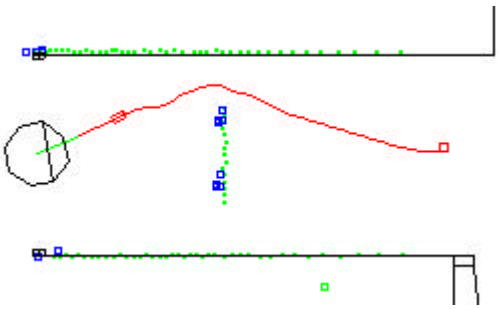


Figura 27: Obstáculo no evitable que bloquea el camino

En teoría se ha encontrado un camino, pero el hueco dejado es demasiado pequeño para el robot. Por tanto, esto derivará en un comportamiento no estable del robot y nunca se encontrará un camino bueno. No sólo eso, el obstáculo es imposible de esquivar, y no se puede encontrar un punto intermedio alternativo: la situación de bloqueo aquí es completamente segura. Tenemos que invocar el componente „Alternative intermediate points finding“ (ver subcapítulo 3.4.5) para comprobar si se puede usar una secuencia alternativa de puntos intermedios.

Otro caso distinto es la presencia de obstáculos móviles. En tanto que sus movimientos no son predecibles, constituirán un problema más grande que los obstáculos sin movimiento, y sus efectos sobre la navegación no podrán ser estimados a priori.

3.4.5 Alternative intermediate points finding

Podríamos traducir el nombre de este componente por el de „Componente de búsqueda de puntos intermedios alternativos“ . A continuación desarrollamos sus características.

3.4.5.1 Introducción

Como ya vimos con anterioridad en la sección 3.4.4, si ocurre que durante la navegación el robot se queda bloqueado, una puerta está cerrada (o bloqueada), o un punto intermedio no se puede alcanzar, entonces hay que encontrar una secuencia alternativa de puntos intermedios. El componente „Alternative intermediate point finding“, responsable de esta tarea, analiza la información disponible, y entonces pueden ocurrir dos cosas:

a) La situación de bloqueo no se puede resolver. Esta circunstancia ha de ser comunicada al control de la visita guiada (Tour guide). Más detalles al respecto se suministran en el siguiente subcapítulo 3.4.6.

b) Se ha encontrado una secuencia alternativa de puntos intermedios. Deshacemos el bloqueo y continuamos con la navegación hacia el siguiente punto intermedio.

Las preguntas importantes aquí son (y que serán respondidas en lo que sigue):

- ¿Cómo sabemos si se puede resolver o no la situación de bloqueo?
- ¿Cómo calculamos un punto intermedio alternativo?

3.4.5.2 La situación de bloqueo

En respuesta a la primera pregunta, podemos considerar las siguientes situaciones posibles:

a) Un obstáculo grande bloquea el camino (o uno móvil que tenga el mismo efecto). En este caso el robot no será capaz de encontrar un camino hacia el siguiente punto intermedio.

b) Un obstáculo ocupa la posición del punto intermedio, pero no bloquea completamente el camino. Pueden ocurrir dos cosas:

b.1) El punto intermedio bloqueado es el punto final. En este caso, existe una situación de bloqueo, a no ser que estemos ya muy cerca de la posición final, en cuyo caso podríamos considerar que la posición final se ha alcanzado.

b.2) El punto intermedio bloqueado no es el punto final. La solución inmediata entonces sería pasar de largo de este punto intermedio, y dirigirnos hacia el siguiente. No obstante, eso no será posible cuando el punto intermedio bloqueado se corresponda con una puerta.

c) Una puerta está cerrada. En este caso, si existe un camino alternativo para alcanzar el punto, se tomará éste, tal como se muestra en la Figura 28. Si no, la situación de bloqueo se comunicará al control de la visita guiada (Tour guide).

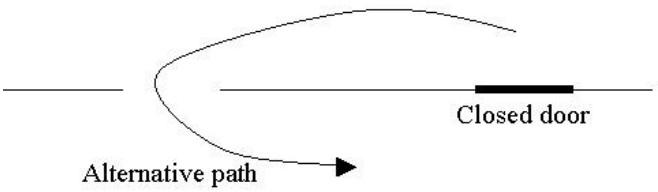


Figura 28: Camino alternativo para una puerta cerrada

3.4.5.3 El punto intermedio alternativo

A la segunda pregunta que formulamos (cómo calculamos un punto intermedio alternativo), la determinación de un nuevo punto intermedio alternativo dependerá de la situación:

Si tenemos un obstáculo ocupando el punto intermedio y no bloquea completamente el camino, y este punto intermedio no es una puerta o el punto final, entonces se tomará el siguiente punto de la secuencia como nuevo punto intermedio a ser alcanzado (ver Figura 29):

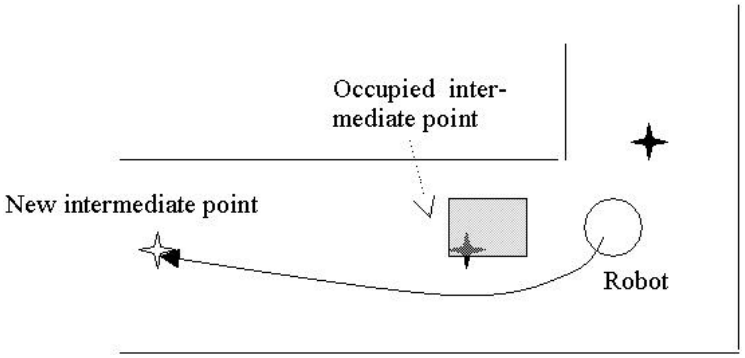


Figura 29: Toma del siguiente punto intermedio de la secuencia

- Si tenemos alguno de los siguientes casos:

  a) Tenemos un obstáculo ocupando el punto intermedio y que no bloquea completamente el camino, pero este punto intermedio necesita cruzar una puerta para ser alcanzado,

  b) La puerta está cerrada,

  c) El camino está completamente bloqueado por un obstáculo móvil o de gran tamaña,

  d) El punto destino está bloqueado por un obstáculo,

  en cualquiera de estos caso, pasar de largo del punto intermedio actual y escoger el siguiente de la secuencia no es un método válido, porque conducirá al robot a situaciones inútiles o potencialmente peligrosas. Por tanto, la solución será recalcular la secuencia de puntos intermedios a alcanzar, pero ahora el último punto intermedio que se alcanzó de forma satisfactoria será considerado como punto de comienzo, y las siguientes modificaciones habrán de llevarse a cabo sobre el grafo que representa a los puntos intermedios y sus posibles conexiones:

  - para los casos a) y b) las aristas que conectan el punto intermedio bloqueado (vértice) a los otros puntos son eliminadas del grafo. De esta forma, se considera que dicho punto intermedio bloqueado ya no es alcanzable en tanto que no tiene aristas que lo conecten a los demás.

- para el caso c) sólo la arista que conecta el último punto intermedio alcanzado de forma satisfactoria al siguiente punto intermedio planeado será eliminada del grafo. Esto es así porque no sabemos si el siguiente punto intermedio en sí (y no sólo el camino que conduce a él) está obstruido o no; asumiremos que no lo está y que por tanto puede ser alcanzado a través de un camino alternativo (si es que existe dicho camino).
- para el caso d) no existe solución posible, puesto que el punto destino está bloqueado y no se puede alcanzar. Lo más que podemos hacer es dar el punto destino por alcanzado si la última posición alcanzada por el robot estuviera muy cercana a dicho punto destino.

Después de todas estas operaciones, intentamos calcular una nueva secuencia de puntos intermedios hacia el punto destino usando el grafo que hemos modificado y el nuevo punto de comienzo. Si encontramos una secuencia válida de puntos intermedios hacia el punto destino, comenzamos con la navegación de nuevo pero usando esta nueva secuencia. Si no existe una secuencia válida de vértices conectados al punto final, consideraremos entonces que el camino está bloqueado (o la puerta cerrada) y lo comunicaremos al control de la visita guiada (Tour guide), porque no existe un camino alternativo disponible.

3.4.6 Tour guide-communication

3.4.6.1 Introducción

Este componente, cuyo nombre podríamos traducir por “Componente de comunicación con el control de la visita guiada”, es responsable como bien dice su nombre, de la comunicación con el control de la visita guiada (Tour guide): tenemos que recibir de dicho control de la visita guiada el número del punto a ser alcanzado, y comunicarle además el éxito de las operaciones o los problemas que pudieran aparecer durante la navegación.

3.4.6.2 Métodos ofrecidos al control de la visita guiada (Tour guide)

Los siguientes métodos están disponibles para el control de la visita guiada (Tour guide):

- *GoTo(int numberOfTarget)*

  Con este método, el control de la visita guiada estable el destino a ser alcanzado por el robot móvil RobIAS. Esta función será ejecutada durante tanto tiempo como la navegación tenga lugar (u ocurra un error), porque el control de la visita guiada (que recordemos no es parte de este proyecto fin de carrera, y que fue implementado por otro estudiante) está programado de tal forma que no puede leer ninguno de los valores de retorno. Las únicas posibilidad que se reconocen son devolver OK o ERROR. Cuando el control de la visita guiada (Tour guide) recibe OK, entonces continúa con las presentaciones habladas para el expositor alcanzado, y

cuando recibe ERROR, informa acerca de esta situación a los visitantes e intenta reanudar la navegación. Por esta razón el método *GoTo()* ha de ser ejecutado tanto tiempo como la navegación tenga lugar, porque necesitamos garantizar una correcta sincronización entre ambos procesos.

- *Resume(bool Action)*

  Este modo permite al control de la visita guiada reanudas la ejecución del proceso de navegación después de una situación previa de bloqueo causada por un camino bloqueado o una puerta cerrada. El parámetro de este método define la acción a realizar:

  - *Action = true* : continúa con la navegación
  - *Action = false* : fin de la visita guiada (Guided tour): el destino no es alcanzado

  También en este caso la ejecución de la función concluye cuando la navegación está completa (o cuando ocurre un nuevo error) debido a las mismas razones de sincronización que para la función *GoTo()*.

3.4.6.3 Situaciones notificadas

Tres son las posibles situaciones que se pueden comunicar al control de la visita guiada (Tour guide):

- La navegación tuvo éxito y el robot está en la posición final
- Camino bloqueado
- Puerta cerrada

Las dos últimas situaciones que siempre hemos considerado como dos casos separados, serán a efectos de comunicación agrupadas en un solo caso, porque el control de la visita guiada (Tour guide) sólo reconoce OK o ERROR como valores de retorno.

El método usado para comunicar estas situaciones al control de la visita guiada ya se ha presentado con anterioridad: las funciones *GoTo()* y *Resume()*, que son invocadas por el control de la visita guiada (Tour guide) tienen sólo dos posibles valores de retorno: OK y ERROR. Si todo sale bien, se devuelve el valor OK cuando la navegación termina y el robot ha alcanzado la posición final. Pero si hay algún error durante la navegación, la función correspondiente será interrumpida y se devolverá el valor ERROR. Esto funciona de esta forma para ambas funciones, y realmente ambos casos presentan un comportamiento bastante parecido.

La solución, fácilmente implementable, adoptada para la sincronización entre el software de navegación (Navigation software) y el control de la visita guiada (Tour guide) es: las funciones *GoTo()* y *Resume()* permanecen en un bucle que se ejecuta de forma permanente si el software de navegación con cambia una cierta variable de control. Esta variable de control es leída desde un fichero (porque la comunicación directa con los métodos del software de navegación no se han implementado; ver más acerca de esto en el siguiente punto 3.4.6.4), con una frecuencia razonable para evitar un colapso de los recursos, y el software de navegación escribe la variable de control en este fichero cuando la navegación termina o cuando hay una situación de bloqueo, con un valor apropiado. De esta sencilla forma podemos implementar una comunicación muy sencilla entre el control de la visita guiada (Tour guide) y el software de navegación (Navigation software).

3.4.6.4 Implementación de la comunicación

Un asunto muy importante es cómo la información suministrada por el control de la visita guiada (Tour guide) es comunicada al software de navegación (Navigation software). Como ya se dijo con anterioridad, se usa un método bastante sencillo consistente en escribir y leer un pequeño fichero para hacer dicha comunicación posible. Los métodos *GoTo()* y *Resume()* están contenidos en un DLL con una interfaz ActiveX. El control de la visita guiada accede a esta interfaz ActiveX, es decir, invoca los métodos *GoTo()* y *Resume()*. Estos métodos escriben la información recibida en un pequeño fichero. Mientras que se espera al nuevo destino, este fichero es leído periódicamente por el software de navegación, para saber si se han dado nuevas instrucciones. Cuando la navegación termina o hay un error, el software de navegación escribe de nuevo esta información en un pequeño fichero, y entonces, los métodos *GoTo()* y *Resume()*, que estaban leyendo periódicamente el fichero mientras esperaban al final de la navegación, pueden interrumpir su ejecución y devolver el control al control de la visita guiada (Tour guide). El proceso completo se encuentra representado en la Figura 30:

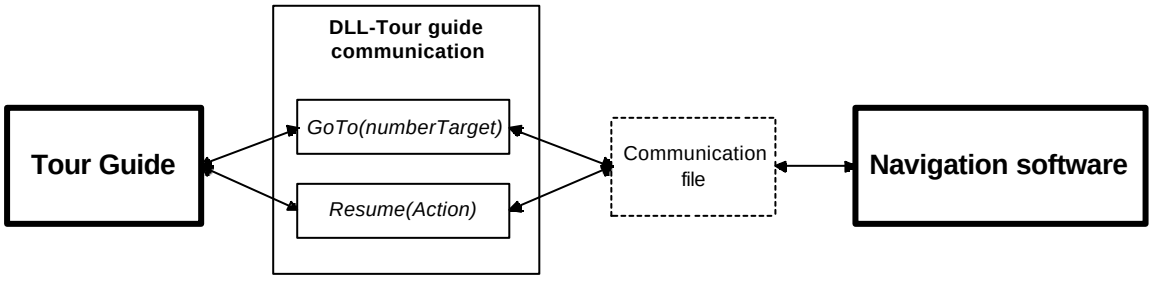


Figura 30: Comunicación entre el software de navegación (Navigation software) y el control de la visita guiada (Tour guide)

Los problemas de este método son:

- Posible error cuando ambos procesos de lectura y escritura se realizan al mismo tiempo.
- Problemas de velocidad, debido a los accesos a disco
- Problemas potenciales de caché.

Todos estos problemas son conocidos, pero consideramos que:

- El tamaño del fichero a escribir/leer es realmente pequeño, apenas 2 o 3 bytes, y eso no debería implicar en absoluto una carga para el ordenador.
- El proceso de escritura se lleva a cabo con muy poca frecuencia, no más de una vez por minuto, y por tanto la probabilidad de colisión con el proceso de lectura puede ignorarse, porque el fichero se lee para comprobar si hay nuevos datos escritos cada aproximadamente 2 segundos.

En la práctica hemos observado que este método funciona a la perfección, y no hemos experimentado, después de cientos de ensayos y simulaciones, ningún problema relacionado con este asunto. Así, podemos considerar que este método, aunque no es muy elegante, es completamente útil y funcional. Además, el estudio de una comunicación refinada entre el control de la visita guiada y el software de navegación no era uno de los objetivos de este proyecto fin de carrera (debido a la limitación de tiempo impuesta por el Instituto IAS para su realización, que obliga a definir una serie de objetivos realistas a cumplir), con lo que podemos considerar que la solución alcanzada es buena: simple y efectiva.

3.5 Especificaciones del test

3.5.1 Introducción

Durante toda la fase de implementación y también al final de la de integración hubo que realizar múltiples tests, para analizar si se mantienen los requerimientos del sistema. La mayoría de estos tests fueron realizados con la ayuda del simulador Pioneer (ver subcapítulo 2.2.4), ya que de esta forma podíamos ahorrar mucho tiempo, pero también hubo que realizar tests con el robot real. En los próximos subcapítulos damos más detalles acerca de los tests a los que nos estamos refiriendo.

3.5.2 Requerimientos del test

Para realizar el test sobre el software de navegación (Navigation software) se requieren los siguientes elementos:

- El robot móvil RobIAS, con las baterías suficientemente cargadas y los cables a la red eléctrica, monitor, ratón y teclado desconectados (eso significa que RobIAS debe ser completamente autónomo). Saphira y el software de navegación (Navigation software) han de ser instalados previamente sobre el embedded PC.
- Un PC conectado a la red de área local del Instituto IAS, para empezar la ejecución del programa de forma remota a través de la tarjeta Wireless LAN integrada en RobIAS.

Si además queremos realizar un test sobre el sistema completo de visita guiada (Guided tour), necesitamos adicionalmente:

- La aplicación de control de la visita guiada (Tour guide) desarrollada por el otro estudiante que participó en el proyecto conjunto [12]. Esta aplicación puede empezar a ejecutarse desde cualquier ordenador conectado a la red de área local del Instituto IAS, aunque correrá sobre el PC de RobIAS.
- El robot RobIAS debe tener instalado en su ordenador un Personal Web Server (PWS) con el motor ASP, los ficheros ASP de la aplicación del control de la visita guiada (Tour guide) correctamente instalados y los ficheros de audio necesarios para las presentaciones habladas de los puntos a recorrer por el robot en su visita. Es necesario instalar otros componentes adicionales en el ordenador de RobIAS, para permitir la ejecución del control de la visita guiada (Tour guide), pero no serán presentados aquí. Por favor, para más información diríjase a la documentación del proyecto del otro estudiante implicado en este proyecto [12].

Adicionalmente se requiere para estos tests:

- El test debe ser realizado en el Instituto IAS, dentro de un entorno real, con puntos reales de partida y de llegada.
- Sería deseable la presencia de espontáneos que intercepten la trayectoria del robot, para comprobar cómo de bueno es el robot esquivando tales obstáculos.
- Deben evitarse las situaciones potencialmente peligrosas (colisiones con elementos frágiles o con personas que no se dan cuenta de la presencia del robot). El test completo debe ser supervisado por una persona en condiciones de parar el robot o avisar a las personas en caso de que sea necesario.

El sistema será objeto de un test consistente en colocar el robot en la posición inicial y dar entonces las órdenes de moverse a la posición de destino. Deberán realizarse dos tipos de tests:

- **Tests sin el control de la visita guiada (Tour guide)**, para comprobar el rendimiento del software de navegación (Navigation software) por separado.
- **Tests con el control de la visita guiada (Tour guide)**, para comprobar el rendimiento de ambos, el software de navegación y de su comunicación con el control de la visita guiada.

3.5.3 Criterios del test

Se considerará que el test ha tenido éxito cuando se satisfagan las siguientes condiciones:

1. El robot alcanza el destino deseado.
2. El robot no choca contra ningún obstáculo existente en el camino
3. El error en la posición final es pequeño (o sea, que permita al robot ir a través de la puerta más pequeña)
4. El tiempo empleado para alcanzar el destino es razonable.
5. Los obstáculos que no se pueden evitar son detectados y comunicados como tal.
6. Las puertas cerradas son detectadas y comunicadas como tal.
7. Se encuentran trayectorias alternativas si se necesitan y están disponibles.
8. El camino usado es lógico (sin bucles sin sentido y similares) y no demasiado largo.
9. La localización del robot y la corrección de la posición son efectivas.

10. La sincronización y comunicación con el control de la visita guiada (Tour guide) son correctas.

3.5.4 Casos de test

Tenemos que realizar los siguientes casos de test para comprobar si el software de navegación (Navigation software) cumple los requerimientos del sistema:

- Tenemos que realizar un test sobre la correcta comunicación entre el control de la visita guiada (Tour guide) y el software de navegación (Navigation software). Eso implica que los métodos *GoTo()* y *Resume()* deben poder ser invocados y la información correspondiente transmitida en ambas direcciones.

- Tenemos que realizar un test sobre la navegación bajo condiciones ideales: sin obstáculos en el camino y con todas las puertas que deben ser cruzadas abiertas. El robot debe navegar sin colisiones, a través de un camino eficiente, corregir regularmente su posición sin problemas y alcanzar el destino deseado con la suficiente precisión.

- Después tenemos que realizar un test sobre la navegación pero en presencia de obstáculos que no se pueden mover y pueden ser sobrepasados. El robot debe esquivar estos obstáculos y tener una navegación normal (como se definió en el caso anterior).

- También hay que realizar un test sobre la navegación con obstáculos móviles que pueden ser sobrepasados, y se esperan los mismos resultados.

- Tenemos que realizar un test sobre el comportamiento del robot con las puertas cerradas. Debe detectar esas puertas cerradas, encontrar una trayectoria alternativa si es que la hay, y si no, informar al control de la visita guiada (Tour guide) y parar la navegación.

- Hay que realizar un test sobre el comportamiento del robot con caminos bloqueados, esperándose el mismo comportamiento que en el caso anterior.

- Finalmente tenemos que realizar un test de todo junto, es decir, una navegación real a través del Instituto IAS con una situación realista. Se espera una navegación sin colisiones y a través de un camino eficiente; se deben poder esquivar los obstáculos que no bloquean el camino, y el robot debe ser localizado de forma regular y su posición corregida. El robot debe alcanzar el destino con la suficiente precisión. Si se detectan situaciones de bloqueo, se debe calcular una trayectoria alternativa. Si dicha trayectoria no existe, el error debe ser comunicado al control de la visita guiada (Tour guide).

El resultado de estos casos de test están detallados en la sección 4.2.

4 Recopilación de resultados

4.1 Uso del software

4.1.1 Instalación

4.1.1.1 Prerrequisitos

Para la operación del sistema usaremos el sistema operativo Microsoft Windows 98, Second Edition, Version: 4.10.2222A instalado en el ordenador integrado de RobIAS. Si quiere usar el sistema completo de visita guiada (Guided tour), por favor consulte la documentación del otro estudiante implicado en el proyecto global [12] (disponible sólo en alemán), donde encontrará las instrucciones de instalación para el control de la visita guiada (Tour guide).

4.1.1.2 Acceso al ordenador integrado de RobIAS

Antes de poder instalar ningún programa o usar los programas que ya están instalados en RobIAS, debe acceder a su ordenador integrado. Para ello tiene tres posibilidades:

a) Consola remota:

1. Conecte el módem Wireless LAN a la red eléctrica de 220V (use el transformador) y a la red de área local del Instituto IAS. Después de algunos segundos debería poder ver las luces verdes de los indicadores PWR, ETHR y L en el módem. Por favor sea cuidadoso con la ubicación del módem; si es posible, colóquelo en la misma planta donde el robot va a trabajar, y no demasiado lejos de él.

2. Desconecte el ratón serie (si estuviera conectado) y encienda el robot RobIAS. Para ello dispone del interruptor con el letrero „Main Power Charge“ en la parte inferior izquierda del robot. Debería poder oír cómo arranca el ordenador (incluyendo el típico pitido al inicio), y eso significa que está siendo cargado. Por favor no descuide que el robot se quede sin baterías para poder funcionar sin problemas. Más detalles al respecto se encuentran disponibles en la sección 4.1.3.2 (Baterías)

3. Ahora deberá esperar dos o tres minutos (incluso más si la última sesión no se cerró correctamente). Ahora, desde cualquier ordenador conectado a la red de área local del Instituto IAS, debe ejecutar el programa *vncviewer.exe.* Cuando vea el mensaje „VNC

Server“ teclee **`ias198`** y entonces se le pedirá un password; teclee: **`ias`**. En la siguiente Figura 31 se muestra el aspecto de este programa:

Figura 31: El programa *vncviewer.exe*

4. Recibirá a continuación el mensaje „Enter Network Password“, teclee **`ias`** de nuevo, y normalmente la ventana del VNC Viewer se cerrará sin más resultados aparentes. Esto se debe a que de momento sólo ha realizado el proceso de login en la sesión. Para conectarse realmente al ordenador, repita la operación indicada en el paso 3: ejecute el programa *vncviewer.exe*, teclee **`ias198`** y después teclee **`ias`**. Ahora debería poder ver un nuevo escritorio de color naranja, correspondiente al Windows 98 que se está ejecutando en el ordenador integrado de RobIAS.

5. Abra un explorador de Windows (puede usted hacer esto haciendo click derecho sobre el botón de Inicio y escogiendo entonces „Explorer“) y busque en la parte izquierda de la ventana „Network Neighborhood“. De este modo puede usted acceder a los datos compartidos de los demás ordenadores conectados a la red de área local del Instituto IAS.

b) Acceso a los ficheros:

1. Conecte el módem Wireless LAN a la red eléctrica de 220V (use el transformador) y a la red de área local del Instituto IAS. Después de algunos segundos debería poder ver las luces verdes de los indicadores PWR, ETHR y L en el módem. Por favor sea cuidadoso con la ubicación del módem; si es posible, colóquelo en la misma planta donde el robot va a trabajar, y no demasiado lejos de él.

2. Desconecte el ratón serie (si estuviera conectado) y encienda el robot RobIAS. Para ello dispone del interruptor con el letrero „Main Power Charge“ en la parte inferior izquierda del robot. Debería poder oír cómo arranca el ordenador (incluyendo el típico pitido al inicio), y eso significa que está siendo cargado. Por favor no descuide que el robot se quede sin baterías para poder funcionar sin problemas. Más detalles al respecto se encuentran disponibles en la sección 4.1.3.2 (Baterías)

3. Abra un explorador de Windows en el ordenador que está usando para acceder de forma remota al robot. En la barra de tareas encontrará usted las siguientes opciones: „Extras => Suchen => Computer“ y teclee entonces el nombre del ordenador: **`ias198`**. En este punto debería ser capaz de ver el ordenador IAS198, que es como se denomina en el ámbito de red local al ordenador de RobIAS. Ya puede examinar la estructura de directorios para llegar al directorio deseado del robot. Ya puede también copiar los ficheros de otras ventanas a ésta.

c) Consola local:

1. Conecte un monitor, un teclado y un ratón serie al ordenador integrado de RobIAS. Los conectores están en la parte inferior derecha del robot.

2. Encienda el robot RobIAS. Para ello dispone del interruptor con el letrero „Main Power Charge“ en la parte inferior izquierda del robot. Debería poder oír cómo arranca el ordenador (incluyendo el típico pitido al inicio), y eso significa que está siendo cargado. Por favor no descuide que el robot se quede sin baterías para poder funcionar sin problemas. Más detalles al respecto se encuentran disponibles en la sección 4.1.3.2 (Baterías).

3. Introdúzcase en la sesión **`ias`** con el password **`ias`** cuando se le pregunte con el mensaje „Enter Network Password“.

4. Abra un explorador de Windows (puede usted hacer esto haciendo click derecho sobre el botón de Inicio y escogiendo entonces „Explorer“) y busque en la parte izquierda de la ventana „Network Neighborhood“. De este modo puede usted acceder a los datos compartidos de los demás ordenadores conectados a la red de área local del Instituto IAS.

Nota: Normalmente se recomienda la opción **c)** para el proceso de instalación, porque puede acceder directamente al ordenador de una forma rápida y cómoda. Las opciones **b)** y **c)** son ambas apropiadas para los procesos posteriores de copiar, modificar o reemplazar ficheros. Para ejecutar la visita guiada (Guided tour) sólo se puede usar la opción **a)**, porque la opción **b)** no permite la ejecución remota de programas y además porque el dispositivo láser y el ratón serie usan el mismo puerto IRQ, con lo que el láser no funcionaría. Esto significa además, que cuando arranque el ordenador usando la opción **c)** y quiera después ejecutar la visita guiada (Guided tour), debe reiniciar antes el ordenador con el ratón serie desconectado.

4.1.1.3 Instalación de ARIA y Saphira

Cuando ya se haya conectado al robot, debe instalar Saphira y ARIA. Debe instalar las versiones **Saphira 8.1.11b** y **ARIA 1.1.11b**, y los ficheros de instalación correspondientes que están contenidos en la carpeta llamada *Version 8.1-11b*. Debe seguir los siguientes pasos:

1. Conéctese al ordenador integrado de RobIAS y abra un explorador de Windows (vea el subcapítulo 4.1.1.2; la opción **c)** es la recomendada en este caso).

2. Cree la siguiente estructura de directorios en el ordenador integrado de RobIAS: *C:\Exchange\Projekte\Robias.* El resto de la documentación estará referida a esta estructura; puede escoger si así lo desea una estructura de directorios diferente para la instalación de ARIA y Saphira, y también funcionará, pero sea en ese caso muy cuidadoso: no mezcle las carpetas. Por todo ello (y porque el programa estará ya normalmente instalado usando esta estructura de directorios), recomendamos encarecidamente dejar esta estructura así.

3. Copie a esta carpeta que acaba de crear la carpeta llamada *Version 8.1-11b,* donde se hallan contenidos los ficheros de instalación de ARIA y Saphira. Instale estos ficheros en el siguiente orden, y usando los directorios propuestos cuando se le pregunte durante la instalación (y no usando los que vienen por defecto):

Nombre del fichero	Carpeta de instalación
Saphira-8.1-11b.exe	C:\Exchange\Projekte\Robias\Saphira
Saphira-laser-8.1-11b.exe	C:\Exchange\Projekte\Robias\Saphira
ARIA-1.1-11b.exe	C:\Exchange\Projekte\Robias\ARIA

4. Si Saphira y ARIA han sido correctamente instalados, debería poder un nuevo grupo de programas llamado *Saphira* (en *Programs*), tal como se muestra en la siguiente Figura 32:

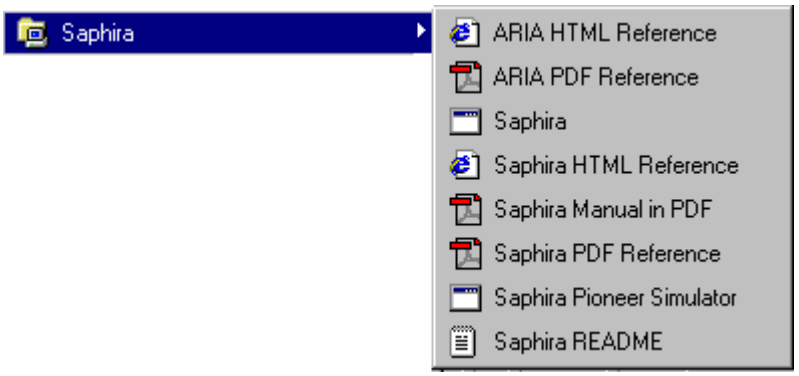


Figura 32: El grupo de programas Saphira

5. Para comprobar si el módulo del láser se ha instalado correctamente, ejecute *Saphira* y comprueba que la palabra *Laser* esté presente en la barra superior del programa, tal como se muestra en la siguiente Figura 33:

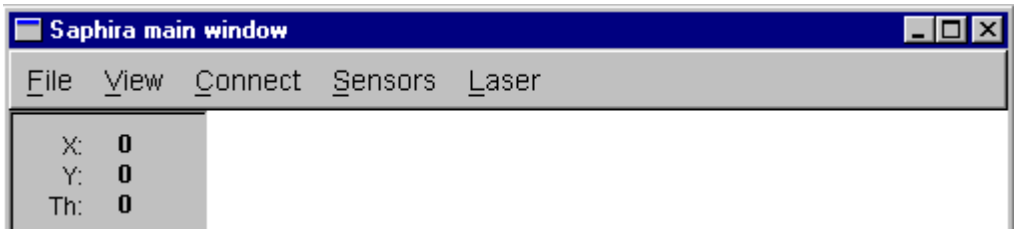


Figura 33: El láser en el programa Saphira

6. Si los puntos 4 y 5 están bien, entonces Saphira y ARIA han sido instalados de forma correcta.

4.1.1.4 Instalación de los componentes de software del software de navegación

Los ficheros necesarios para ejecutar el software de navegación (Navigation software) están contenidos en el fichero comprimido *GuidedTourZip.exe.* Tiene que ejecutarlo sobre el ordenador integrado de RobIAS, y verá lo que se muestra en la Figura 34.

Por defecto, los ficheros se instalan en el directorio *C:\Exchange\Projekte\Robias*. Si usted ha elegido otro directorio, deberá teclearlo ahí. Nótese que la opción „*Dateien ohne Nachfrage Überschreiben*“ no viene seleccionada por defecto. Si la selecciona, se sobreescribirán los ficheros sin pedir confirmación. En este caso, los mapas y otra información importante se puede perder al confirmar la operación. No olvide en tal situación realizar copias de seguridad, si esto fuera necesario. Si no selecciona esta opción, se le preguntará si desea reemplazar el fichero *p2pb.p*, incluso en una instalación nueva. Confirme esta operación sin miedo. Si se le pide reemplazar otros ficheros, eso significa que la instalación ya fue realizada con anterioridad.

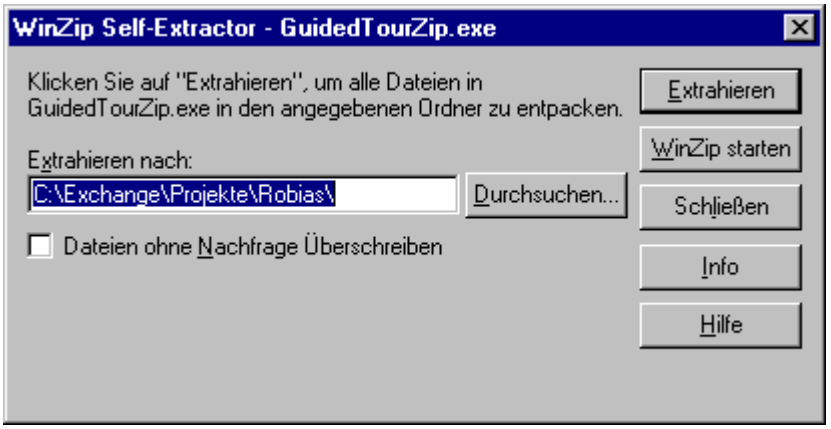


Figura 34: Extrayendo los ficheros necesarios para los componentes del software

Cuando haya decidido el directorio de instalación y la opción de sobreescritura, pulse el botón „*Extrahieren*“ y la extracción de los ficheros comenzará. Todos los ficheros necesarios e copiarán en el directorio seleccionado, tanto los ejecutables como el código fuente, en las carpetas correctas, de forma que usted no tenga que preocuparse al respecto y cometer posibles errores. Por favor no olvide los comentarios anteriores acerca de la sobreescritura de ficheros.

La instalación por defecto contiene ya un mapa de la segunda planta del Instituto IAS, y la información necesaria para la navegación (puntos intermedios, referencias de localización, etc.). Esta información se construyó a partir del trabajo original del otro estudiante participante en el proyecto [12] y ya es completamente funcional. Las modificaciones posteriores a los expositores a visitar o el entorno deben ser realizadas por el propio usuario, atendiendo a las instrucciones contenidas en el subcapítulo 4.1.5 y el manual de usuario (sólo disponible en alemán) de dicho estudiante.

Para instalar el componente de comunicación, debe usted ejecutar el fichero comprimido *CommunicationZip.exe* en el ordenador integrado de RobIAS. En este caso verá usted la ventana mostrada en la Figura 35:

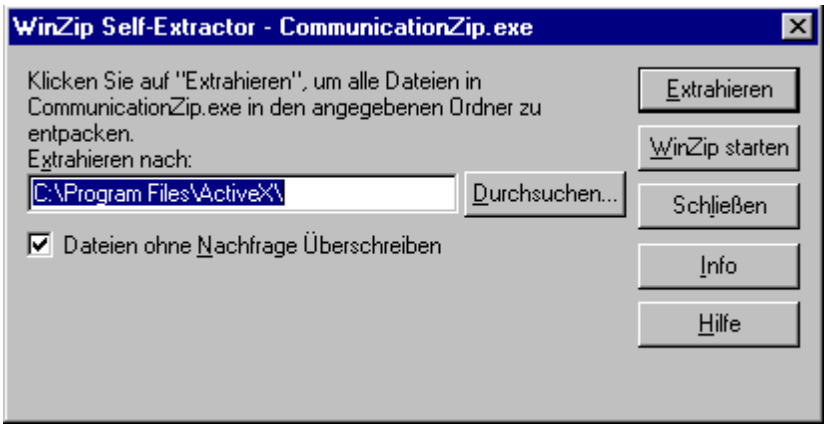


Figura 35: Extrayendo los ficheros de comunicación necesarios

En este caso no debe usted cambiar ninguna de las opciones (tampoco el directorio), y pulsar directamente el botón *Extrahieren* para copiar todos los ficheros necesarios. No hace falta tampoco hacer ninguna copia de seguridad previamente.

4.1.2 Funcionamiento

4.1.2.1 Preparación del robot

Debe usted seguir los siguientes pasos:

1. Si acaba de instalar el software de la visita guiada (Guided tour), vaya al siguiente paso. Si desea empezar desde el principio (el robot está apagado), diríjase al subcapítulo 4.1.1.2 (Acceso al ordenador integrado de RobIAS) y siga las instrucciones correspondientes a la opción **a)** .

2. Asegúrese de que el láser está desconectado, y entonces conéctelo: el láser es el dispositivo en la parte frontal del robot. Encontrará el interruptor en la parte trasera (ON/OFF). Cuando desaparezca la luz naranja del láser y vea la verde, vaya al siguiente paso.

3. Mueva el robot a la posición inicial de la visita guiada. Intente posicionar el robot tan exactamente como le sea posible, ayudará al robot a orientarse más fácilmente en las primeras fases de la navegación.

4.1.2.2 Inicializando el software de navegación (Navigation software)

En la carpeta *C:\Exchange\Projekte\Robias\Saphira\bin* del ordenador integrado en el robot encontrará usted el fichero *GuidedTour.exe.* Ejecútelo (una vez preparado el robot) y espere unos instantes. Mueva entonces la ventana del programa para refrescar su contenido y debería poder algo parecido a lo mostrado en la Figura 36:

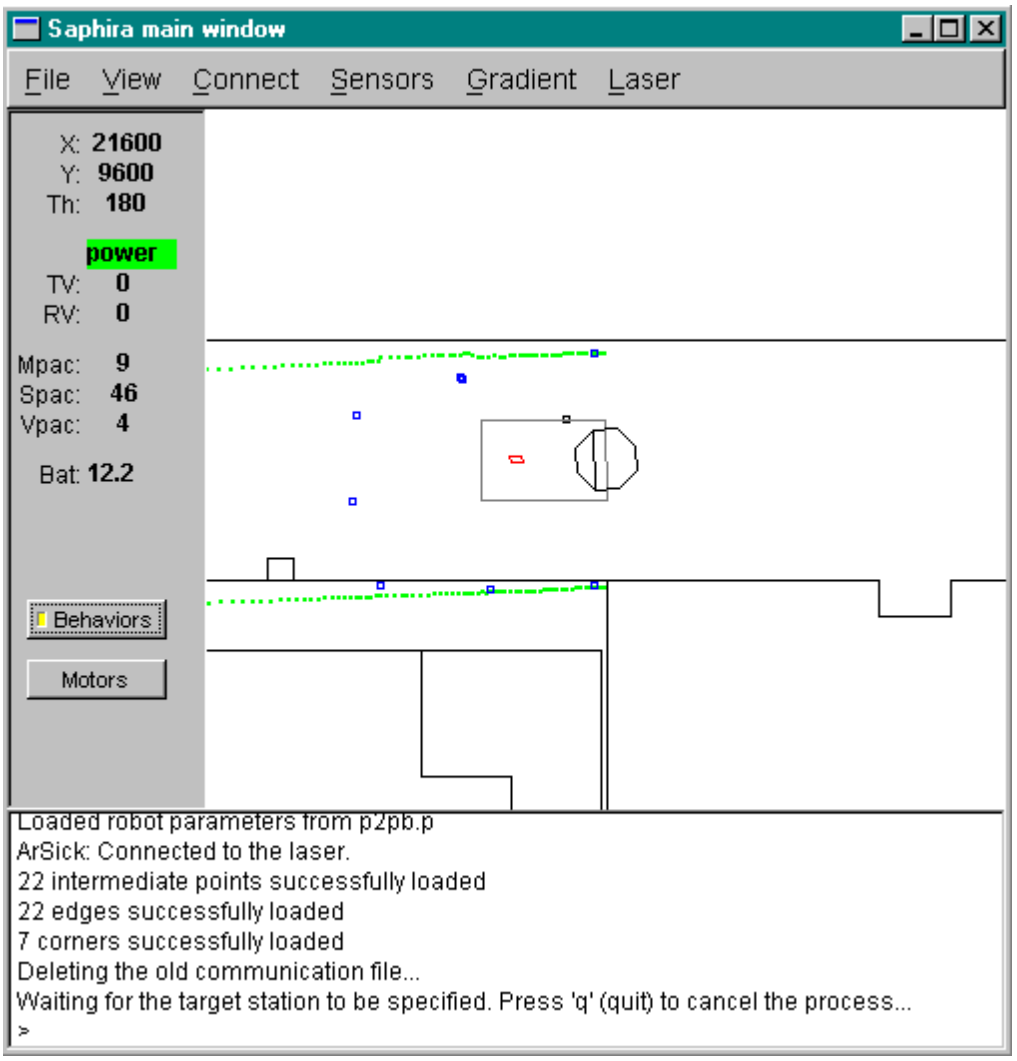


Figura 36: El programa Saphira con la visita guiada recién empezada

Debe usted comprobar lo siguiente para saber si todo ha salido bien:

1. Debería poder ver el mensaje „*Waiting for the target station to be specified. Press ´q´ (quit) to cancel the process...*“. Esto significa que el programa está listo para aceptar el destino a ser alcanzado. No debería usted ver mensajes del tipo „*ArSick: error*“.

2. La información del láser, de color verde, debería ser visible y corresponderse (de forma aproximada) con el mundo cargado (en negro). Si esta información del láser no es visible, puede ser que no se haya inicializado correctamente el láser, y si es visible pero no se ajusta correctamente con el mundo cargado, entonces quizá ha posicionado usted el robot en una posición inicial incorrecta.

Si ve usted que todo es correcto, el robot está preparado y puede usted dirigirse a la siguiente sección. Si no fuera así, presione la tecla „q“ (antes debe usted hacer click con el ratón sobre la ventana de navegación donde se halla dibujado el robot) y entonces, cuando vea usted el mensaje „*You can now close the application*“ (refresque la ventana moviéndola), cierre el programa mediante el botón X en la esquina superior derecha de la ventana y reinícielo de nuevo, a partir del paso número 2 del subcapítulo 4.1.2.1 (Preparación del robot). Si el problema persiste, consulte por favor el Apéndice E (Corrección de errores más frecuentes).

4.1.2.3 Ejecutando la visita guiada (Guided tour)

Una vez que ha inicializado el software de navegación (Navigation software), tiene usted dos posibilidades:

a) Ejecutar la visita guiada (Guided tour) completa. Para ello ejecute el control de la visita guiada (Tour guide) del otro estudiante ya mencionado [12] atendiendo a las instrucciones contenidas en su manual de usuario (sólo disponibles en alemán). El software de navegación ya se encuentra esperando a las órdenes de este control de la visita guiada.

b) Iniciar la ejecución del simulador para mover el robot. Para simular las órdenes dadas por el control de la visita guiada (Tour guide), existen un par de simuladores contenidos en la carpeta *C:\Program Files\ActiveX* del ordenador integrado del robot: *Simulator-GoTo.exe* y *Simulator-Resume.exe.* Si usted desea iniciar la navegación hacia un determinado punto de destino, ejecute el programa *Simulator-GoTo.exe,* escriba el número del destino y presione la tecla „Enter“. El robot se moverá (físicamente, no una simulación) al punto especificado; si existiera un error durante la navegación, puede reanudarla usando el programa *Simulator-Resume.exe,* entrando el número 1 si desea reanudar la navegación, o el número 0 si desea abortarla. Estos programas permanecen abiertos tanto tiempo como dure la navegación del robot, y después se cierran automáticamente cuando ésta concluye.

4.1.2.4 Interrupción del software de navegación (Navigation software) en cualquier momento

En el caso de que necesite detener la ejecución del software de navegación (Navigation software) en cualquier momento (excepto cuando se están reproduciendo los ficheros de sonido), sólo tiene que hacer click con el ratón sobre la ventana de navegación de Saphira donde aparece el robot y presionar la tecla „q“, o el valor 0 en los programas *Simulator-GoTo.exe* o *Simulator-Resume.exe* cuando se le pida. Después de esto, cuando vea el mensaje „*You can now close the application*“ (refresque la ventana moviéndola), cierra Saphira pulsando la X. ¡Sea cuidadoso! Esto no termina la visita guiada completa (Guided tour), que debe completarse. Si experimenta algún problema al respecto, vea la sección 7 (La visita guiada no reacciona) del Apéndice E (Corrección de errores más frecuentes).

4.1.2.5 Terminación normal de la visita guiada (Guided tour)

Después de que la visita guiada (Guided tour) se haya completado, debe cerrar la aplicación y apagar el robot. Siga por favor los siguientes pasos:

1. Si la navegación tuvo éxito, pare el programa y ciérrelo tal como se indica en el subcapítulo 4.1.2.4 (Interrupción del software de navegación (Navigation software) en cualquier momento).
2. Si se pierde la conexión al robot o al láser, pare el programa y ciérrelo tal como se indica en el subcapítulo 4.1.2.4.
3. Apague el láser.
4. Apague el ordenador integrado de RobIAS desde el menú de inicio de Windows. Esto es posible sólo si ha iniciado el robot usando la opción **a)** o **c)** del subcapítulo 4.1.1.2 (Acceso al ordenador integrado de RobIAS). Si ha usado la opción **b),** siempre puede ejecutar el VNC Viewer a posteriori en cualquier momento, como se indica en los pasos 3 y 4 de la opción **a)** en el subcapítulo 4.1.1.2.
5. Espere aproximadamente un minuto para que Windows se cierre adecuadamente y conmute el interruptor con el letrero „Main Power Charge“ en la parte inferior izquierda del robot para apagarlo.
6. El software y el robot están ya completamente apagados.

4.1.3 Elementos de control

4.1.3.1 Software de navegación (Navigation software)

4.1.3.1.1 Introducción

Nuestro software de navegación (Navigation software) ofrece en verdad pocos elementos de control, porque su función principal es el de ofrecer una navegación *autónoma* a través del Instituto IAS. La interacción con el usuario se hace normalmente a través del control de la visita guiada (Tour guide) desarrollada por el otro estudiante implicado en la realización del proyecto global [12].

4.1.3.1.2 Programas que pueden ser iniciados

Sólo existen tres programas que usted puede ejecutar:

a) GuidedTour.exe

Este programa está en la carpeta *C:\Exchange\Projekte\Robias\Saphira\bin* del robot, y contiene el software de navegación (Navigation software) integrado bajo la GUI de Saphira. Podría por tanto usar los elementos de control que proporciona Saphira, pero no los necesitará, porque todo se hace automáticamente por el software de navegación. Las posibles interacciones con Saphira que en todo caso usted podría usar son:

- Presionando la tecla „q“ después de hacer click con el ratón sobre la ventana principal de navegación (aquella en la que se encuentra el robot), la navegación es detenida y el programa queda listo para ser cerrado.

- Presionando el botón X de la ventana, cierra usted la aplicación. Eso puede hacerse sólo cuando usted vea el mensaje „*You can now close the application*“ en la ventana principal de Saphira (puede ser que necesite refrescar la ventana moviéndola para poder ver este mensaje), después de terminar la visita guiada (Guided tour) o pulsando la tecla „q“ como se indicó anteriormente. Si no observa esta regla, el programa se quedará casi con total seguridad residente en memoria y causará que las próximas ejecuciones del programa no funcionen. Si tiene usted que cerrar la aplicación antes de que haya visto este mensaje, consulte por favor la sección 5 (La ventana de Saphira no puede cerrarse) en el Apéndice E (Corrección de errores más frecuentes).

- Puede ser que usted desee usar las opciones adicionales que suministra Saphira, como View (vista) o enable/disable the motors (encender/apagar los motores). Estas opciones no se explican aquí, porque normalmente no se necesitan. Sin embargo, si está interesado en estas opciones, consulte la documentación de Saphira [3] para detalles adicionales.

b) Simulator-GoTo.exe / Simulator-Resume.exe

Estos programas se encuentran en la carpeta *C:\Program Files\ActiveX* del ordenador integrado del robot, y simulan las funciones que normalmente son asumidas por el control de la visita guiada (Tour guide): especificar el destino a visitar y reanudar la navegación en el caso de que anteriormente hubiese habido un error:

- **Simulator-GoTo.exe**: introduzca el número del destino a alcanzar y presione „Enter“. La ventana permanecerá abierta durante la navegación, y se cerrará automáticamente cuando se alcance la estación deseada o cuando haya un error. Introduzca el número 0 si desea concluir la visita guiada (Guided tour).

- **Simulator-Resume.exe**: cuando haya un error durante la navegación, puede especificar de nuevo el número del punto destino usando el programa *Simulator-GoTo.exe* indicado anteriormente, o bien puede ejecutar el programa *Simulator-Resume.exe.* En el último de los casos, si usted introduce el número 1 y después „Enter“, se reanuda la navegación a partir del punto en el que robot se había parado. Si introduce el número 0, la navegación se detiene completamente y se concluye la visita guiada (Guided tour).

4.1.3.1.3 Representación gráfica

El aspecto típico del software de navegación (Navigation software) bajo Saphira es el que se muestra en la siguiente Figura 37:

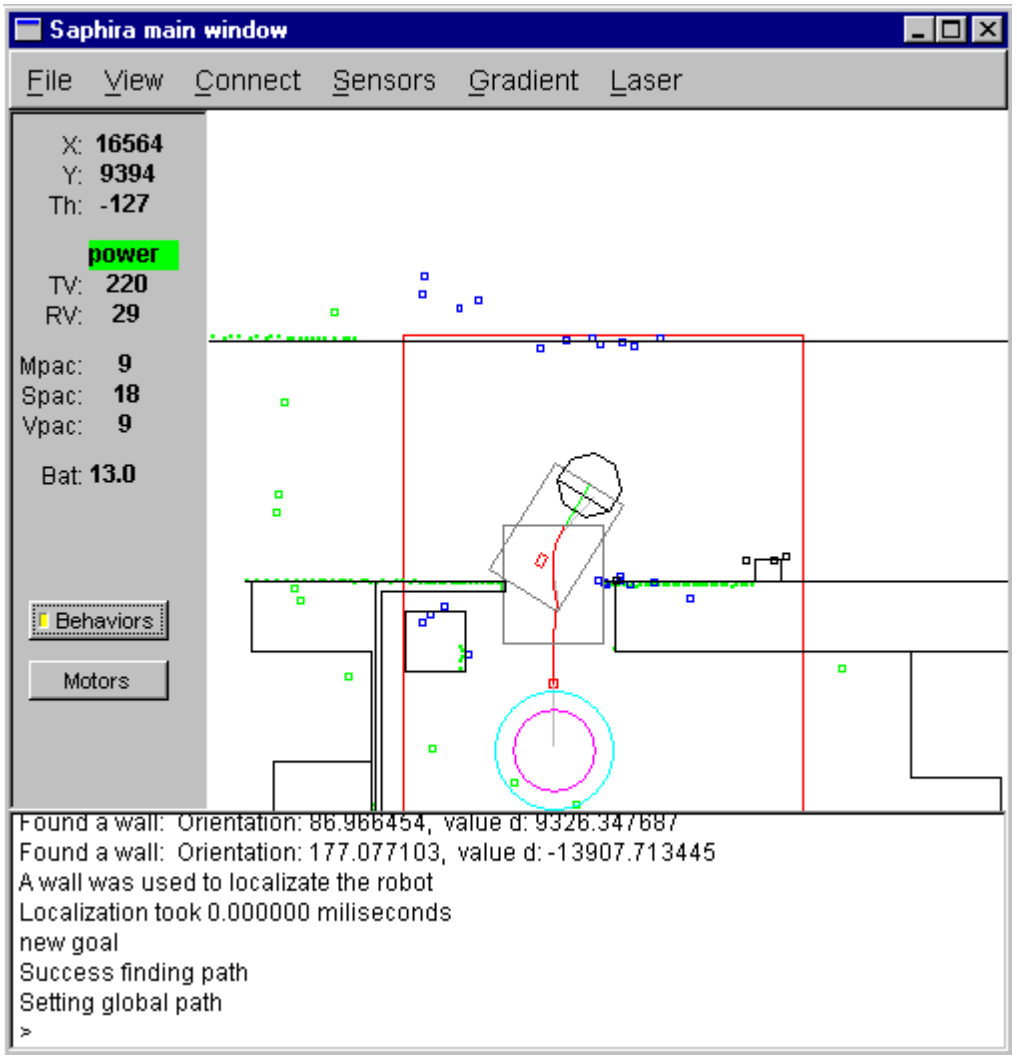


Figura 37: Aspecto del software de navegación (Navigation software) bajo Saphira

En la figura se pueden destacar los siguientes elementos importantes:

1. El **robot**, el círculo negro en el medio. La línea en él contenida nos muestra la orientación del mismo.
2. El **mundo conocido**, con líneas negras.
3. Las **medidas de los sonares**, con grandes puntos de color verde, azul y negro.
4. Las **medidas del láser**, con numerosos puntos pequeños de color verde.
5. El **camino planificado**, con una línea roja.
6. El círculo de color cyan está centrado en el siguiente punto intermedio que debe ser alcanzado. Cuando el robot alcanza este los límites de este círculo, considera que el punto intermedio se ha alcanzado.
7. El círculo de color magenta centrado en el siguiente punto intermedio a ser alcanzado indica el lugar que debe estar libre de obstáculos para que el robot no considere que este punto está bloqueado.
8. La caja en la puerta indica el espacio libre detectado por el robot para cruzar esta puerta.
9. La caja centrada en el robot muestra la zona donde se comprueban las medidas de los sonares (medidas fuera de esta caja son ignoradas, porque la imprecisión de los sonares a largas distancias las hacen inútiles).
10. La información de depuración del programa se encuentra en la subventana inferior.

4.1.3.2 Baterías

Este es un aspecto muy importante del robot. Sin las baterías cargadas de forma adecuada, el robot no viajará correctamente o incluso no funcionará más. Por tanto, ha de observar los siguientes puntos referentes a las baterías:

1. Para ahorrar energía, por favor apague el láser cuando no esté ejecutando el software de navegación (Navigation software). El láser necesita mucha energía, y descargará las baterías de forma muy rápida.
2. Mientras esté usando el ordenador integrado en RobIAS para instalar el programa, copiar ficheros, etc., y no para la navegación en sí, puede usted dejar el transformador de red conectado al robot, de forma que las baterías no se agotarán.
3. El robot necesita un juego de 3 baterías para funcionar. Existen dos juegos de este tipo en el Instituto IAS, y puede usted cargar uno de ellos mientras esté utilizando el otro. Para hacer

esto, simplemente conecte el transformador de red al cubo de carga (Charge Cube) (este cubo tiene tres entradas, conecte el transformados a cualquiera de estas entradas). Este proceso durará normalmente del orden de algunas horas. Puede comprobar si las baterías están completamente cargadas con la ayuda de un voltímetro: 13,1 V es la tensión final normal.

4. Compruebe regularmente la tensión de las baterías del robot en el display LCD que éste lleva incorporado; un valor menor de 12 indica que pronto hay que cargar las baterías o conectar el robot a la red. De otro modo, puede usted experimentar ciertos problemas (típicamente, perder la conexión con el láser y hacer que el software de navegación no funcione más; vea la sección 7: se ha perdido la conexión con el láser, en el Apéndice E: Corrección de errores más frecuentes).

4.1.4 Funciones

La finalidad del software de navegación (Navigation software) es la proveer al robot de un sistema de navegación autónoma desde el punto actual a los puntos determinados por el control de la visita guiada (Tour guide) o por sus simuladores. En tanto que esta navegación se hace de una forma completamente autónoma, el programa no puede usarse durante este proceso, sino solamente iniciado y, ocasionalmente, parado y cerrado. La interacción con el usuario es una tarea del control de la visita guiada (Tour guide). Para más detalles al respecto, consulte la documentación correspondiente [12].

4.1.5 Ejecución de operaciones especiales

4.1.5.1 Visión de conjunto

En el caso de que el entorno en el que el robot viaja cambie, o los puntos de visita se modifiquen, debe usted editar los siguientes dos ficheros: *Ias.wld* y *interpoints.txt*, situados en la carpeta *C:\Exchange\Projekte\RobIAS\Saphira\bin* del ordenador integrado del robot. Estos ficheros contienen la información que el robot debe conocer a priori para poder viajar de forma satisfactoria a través del Instituto IAS.

4.1.5.2 El fichero *Ias.wld*

4.1.5.2.1 Finalidad del fichero

Este fichero contiene el *mundo* (*world*, en palabras de los creadores de Saphira), que puede ser usado por el robot. O sea, el robot usará tanto la información de los sensores como del mundo así cargado.

4.1.5.2.2 Sintaxis del fichero

Los mundos (worlds) se definen como un conjunto de segmentos lineales usando unas coordenadas relativas o absolutas. Los comentarios comienzan con un punto y coma, y las demás líneas no vacías se interpretan como directivas. Las dos primeras líneas del fichero describen la longitud y la altura del mundo, en milímetros. El simulador no dibujará ninguna línea fuera de estos límites definidos. Normalmente es una buena idea incluir un rectángulo con los límites del mundo, tal como se hace en el ejemplo más abajo, para evitar que el robot se mueva fuera del mundo.

Cualquier entrada en el fichero del mundo (world) que comienza con un número se interpreta como la creación un segmento lineal. Los dos primeros números son las *x*,*y* del comienzo del segmento, y los dos segundos son las coordenadas del final del segmento. El sistema de coordenadas para el mundo (world) tiene su origen en la esquina inferior izquierda, con `+Y` apuntando hacia arriba y `+X` hacia la derecha (ver Figura 38)

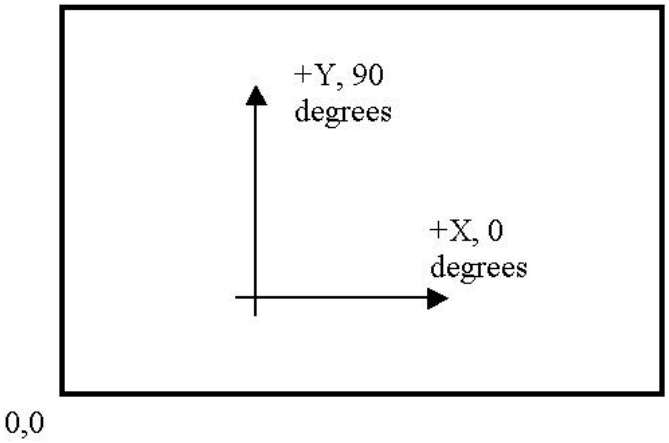


Figura 38: Sistema de coordenadas para la definición del mundo (world)

La posición de los segmentos también puede hacerse relativa a un sistema de coordenadas insertado. La directiva `push x y theta` en el fichero del mundo (world) origina que los sucesivos segmentos usen un sistema de coordenadas con origen en *x*,*y* y cuyo eje *x* apunta en esa dirección. La directiva `push` puede anidarse, en cuyo caso el nuevo sistema de coordenadas está definido con respecto al previo. Una directiva `pop` retorna al sistema de coordenadas previo.

La directiva `position x y theta` posiciona el robot en las coordenadas así indicadas.

El Listado 1 es un fragmento del fichero de descripción del mundo (world) del Instituto IAS `Ias.wld`:

```
;;; IAS world

width 37200
height 30000

   0 0 0 30000               ; World fronteers
   0 0 37200 0
   37200 30000 0 30000
   37200 30000 37200 0

;; Lower side of the main corridor:
;; Prozesslabor 2 walls
   0 8400 1300 8400
   2300 8400 12100 8400
   13100 8400 14400 8400
   14400 8400 14400 0

;; PC-Labor 2 walls
   14400 8400 15700 8400
   16700 8400 21600 8400
   21600 8400 21600 0

;; Cupboards & Windows in PC-Labor 2/Prozesslabor 2
   16800 8400 16800 7700
   16800 7700 21550 7700
   21550 7700 21550 200
   21550 200 50 200
   50 200 50 8300
   50 8300 1300 8300

;; Supporting pillars in Prozess Labor 2
   300 8100 300 7500
   300 7500 900 7500
   900 7500 900 8100
   900 8100 300 8100

....

;; Starting position

position 21600 9600 180
```

Listado 1: Fragmento del fichero de descripción de mundo (world) Ias.wld

El resultado de este fichero de mundo (world file) (usando toda la información contenida en el fichero, y no sólo la mostrada en el listado anterior), se muestra en la Figura 39:

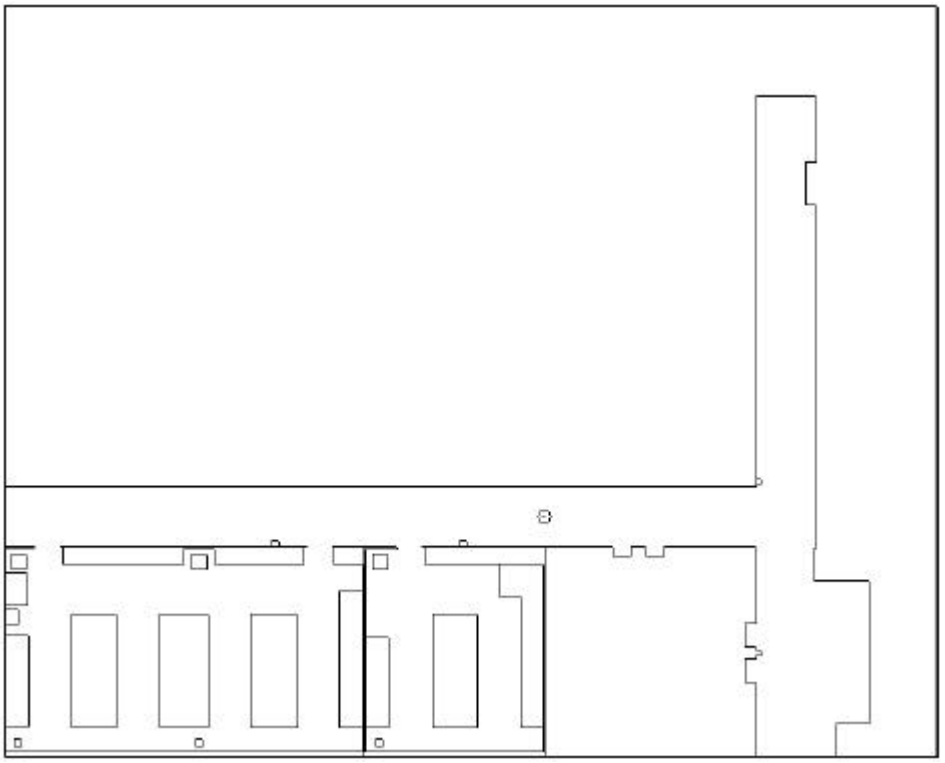


Figura 39: Mundo modelado del Instituto IAS

4.1.5.2.3 Elementos a añadir en el fichero del mundo (world file)

Si el entorno del robot cambio, o desea escribir un nuevo fichero, ¿qué información debe añadir al fichero del mundo (world file)? Las siguientes cosas:

- Hay que añadir todas las paredes.
- Armarios grandes, que son como paredes.
- Las mesas, porque el robot no puede verlas y sólo a través del fichero del mundo (world file) puede verlas.
- Las puertas (abiertas si se pueden cruzar, cerradas si el robot nunca tiene que ir a la habitación en cuestión).
- Pilares.
- Extintores.

- Como normal general, todos los objetos que el robot podría no ver y que siempre se encuentran en la misma posición. No es preciso un proceso exacto de modelado: normalmente es suficiente con definir una caja alrededor del objeto que queremos evitar.

¿Qué elementos no deben ser añadidos al mundo (world)?

- Sillas: sus posiciones varían mucho y no pueden predecirse. En el caso de que tenga un espacio muy estrecho en el que el robot deba moverse, y en consecuencia, problemas con las sillas que pudieran hallarse en él, puede definir una caja alrededor del lugar donde la silla está situada normalmente. De otro modo, la opción recomendada es no incluir las sillas en el mundo y simplemente quitarlas de en medio todo lo que se pueda antes de comenzar con la visita guiada.

- Detalles innecesarios. No necesita un mundo (world) muy detallado, sino definir de forma clara qué zonas están prohibidas para el robot, de manera que éste no intente alcanzar estas zonas.

4.1.5.3 El fichero interpoints.txt

4.1.5.3.1 Finalidad del fichero

Este fichero contiene información adicional que se necesita para una correcta navegación: el punto de comienzo, los puntos a ser alcanzados, los puntos intermedios que el robot puede usar, las conexiones entre dichos puntos y la información de localización. Nótese que el fichero *interpoints.txt* que viene con los ficheros de instalación ya contiene las explicaciones de cómo debe ser escrito.

4.1.5.3.2 Sintaxis general

La sintaxis general del fichero *interpoints.txt* es la siguiente:

```
; comments
COMMAND param1 param2 ... [paramX] [paramX+1] ...  ;comments
```

El significado de esta sintaxis es el siguiente:

- La información que en una línea va detrás de un punto y coma, ya sea al principio o en medio de la misma, es ignorada por el software de navegación (Navigation software), así que utilice esto para escribir sus comentarios.

- Los posibles comandos son el número del punto, la palabra `EDGE` o la palabra `CORNER`.

- Los parámetros sin corchetes son obligatorios; los parámetros con corchetes son opcionales.

- El software de navegación (Navigation software) no ha sido optimizado para reconocer errores de sintaxis en este fichero, porque el objetivo del programa no era construir un compilador. Sin embargo, se reconocen algunas errores frecuentes y se informa de en qué línea aparecen (Para más detalles diríjase a la sección 3: Hay un error con los puntos intermedios, del Apéndice E: Corrección de errores más frecuentes).

4.1.5.3.3 Definición de los puntos de comienzo, fin e intermedios

La definición general de un punto para el robot es la siguiente:

`Position_number X-coord Y-coord [final_angle] [localization_info]`

- **`Position_number`**: número de la posición. Tiene que coincidir con el que está almacenado en el control de la visita guiada (Tour guide). Los números usan la convención siguiente:
  - ➔ El número 0 y los negativos están reservados para el programa y no pueden ser usados.
  - ➔ El número 1 es el correspondiente a la posición inicial.
  - ➔ Los números del 2 al 31 se usan para puntos de destino en la visita.
  - ➔ Los números a partir del 32 se usan para los puntos intermedios predefinidos por el usuario.
- **`X-coord`**: Coordenada X de la posición referida (todas las coordenadas tienen que ser positivas).
- **`Y-coord`**: Coordenada Y de la posición referida (todas las coordenadas tienen que ser positivas).
- **`final_angle`**: Orientación final del robot en una posición final deseada del mismo (en grados de 0 a 360). Obligatorio para los puntos de inicio y destino (si no se escribe ningún valor, se toma 0 por defecto), e innecesario para los puntos intermedios, pero en este caso debe ser escrito si también se escribe el siguiente parámetro.
- **`Localization_info`**: (escriba `final_angle` si escribe este parámetro).Este valor puede ser:
  - ➔ **`CORNER`**: cuando podemos usar una esquina en este punto para orientarnos. No se necesitan parámetros adicionales, las esquinas se comparan de forma automática.
  - ➔ **`CORRIDOR`**: cuando podemos usar un pasillo en este punto para orientarnos. Tenemos que especificar cómo de ancho es el pasillo, y si es `HORIZONTAL` o `VERTICAL` (las dos únicas posibilidades). *MUY IMPORTANTE*: el punto intermedio que usa un pasillo para

localizar el robot debe estar centrado en la coordenada relevante del pasillo. Es decir, debe existir la misma distancia entre el punto intermedio y ambos lados del pasillo.

➔ **`WALL`**: cuando podemos usar una pared en este punto para orientarnos. Tenemos que especificar la coordenada x o y de la pared, y si es `HORIZONTAL` o `VERTICAL` (las dos únicas posibilidades).

En el Listado 2 mostramos un ejemplo extraído del fichero *interpoints.txt* real:

```
;; Start and final points
1  21600  9600 180 CORRIDOR  2400 HORIZONTAL
2  18700  6700 180 WALL      7700 HORIZONTAL
3  16300  2500  90 WALL     14450 VERTICAL
4   1900  5200  90 WALL        50 VERTICAL
5   2000  3100  90 CORNER
6   1700  1200  90 CORNER
7   5100  6700 180 WALL      7700 HORIZONTAL
8   5400  3400  90 CORNER
9   9000  2700  90 CORNER
10 12600  4300  90 CORNER

;; Intermediate points
32 18900  9600   0 CORRIDOR  2400 HORIZONTAL
33 16200  9600   0 WALL     10800 HORIZONTAL
34 12600  9600   0 WALL     10800 HORIZONTAL
35  9000  9600   0 CORRIDOR  2400 HORIZONTAL
36  5400  9600   0 CORRIDOR  2400 HORIZONTAL
37  1800  9600   0 WALL     10800 HORIZONTAL
38  1800  6700   0 CORNER
39  6700  6700   0 CORNER
40  9000  6700   0 CORNER
41 10800  6700   0 WALL      7700 HORIZONTAL
42 12600  6700   0 CORNER
43 16200  6700   0 CORNER
44 12600  5700   0 WALL     14350 VERTICAL
45 16250  4600   0 CORNER
46  9000  4700   0 CORNER

...
```

Listado 2: Ejemplo de puntos de inicio, fin e intermedios

4.1.5.3.4 Definición de las aristas

Las aristas representan las posibles conexiones entre los puntos anteriormente definidos. La sintaxis de una arista es la siguiente:

```
EDGE Vertex-from Vertex-to [Relative_cost] [DOOR] [X-coord]
     [Y-coord] [Orientation]
```

- **`Vertex-from`**: Es el vértice de origen (o destino) de la arista. Nótese que las aristas son bidireccionales, y por tanto, la elección de cuál es el vértice "`Vertex-from`" y cuál el "`Vertex-to`" es completamente arbitraria.
- **`Vertex-to`**: Es el vértice de destino (u origen) de la arista (vea los comentarios del punto anterior).
- **`Relative_cost`**: Es el coste relativo de la arista y se usa para hacer al robot preferir ciertas trayectorias dentro del grafo. Por defecto, si no se especifica nada, el coste relativo de una arista es 1.0. Use números mayores para indicar al robot que ese camino es más difícil para la navegación. Se recomiendan valores del orden de 1.1 o 1.2 (1.5 como máximo) para estos costes. Con costes demasiado altos el robot viajaría distancias muy largas a través de trayectorias sin sentido con tal de evitar estas aristas de alto coste.
- **`DOOR`**: Escriba esta palabra si la arista cruza una puerta. Si se especifica este parámetro debe especificarse además el coste relativo (escriba 1.0 o el valor que desee en este caso). Si no se especifica el parámetro, se asume que la arista no cruza ninguna puerta.
- **`X-coord`**: Es la coordenada X del centro de la puerta. Este parámetro es obligatorio si se ha usado el parámetro `DOOR`.
- **`Y-coord`**: Es la coordenada Y del centro de la puerta. Este parámetro es obligatorio si se ha usado el parámetro `DOOR`.
- **`Orientation`**: `HORIZONTAL` o `VERTICAL`, es la orientación de la puerta. Este parámetro es obligatorio si se ha usado el parámetro `DOOR`.

En el Listado 3 mostramos un ejemplo extraído del fichero *interpoints.txt* real:

```
EDGE  1 32
EDGE 32 33
EDGE 33 43 1.0 DOOR 16200  8350 HORIZONTAL
EDGE 43  2 1.2
EDGE 43 45 1.2
EDGE 45  3 1.2
EDGE 33 34
EDGE 34 42 1.0 DOOR 12600  8350 HORIZONTAL
EDGE 42 44 1.2
EDGE 44 10 1.2
EDGE 42 41 1.2
EDGE 41 40 1.2
EDGE 40 46 1.2
EDGE 46  9 1.2
EDGE 40 39 1.2
EDGE 39  7 1.2
EDGE  7  8 1.2
EDGE  7 38 1.2
EDGE 38  4 1.2
EDGE  4  5 1.2
EDGE  5  6 1.2
EDGE 38 37 1.0 DOOR  1800  8350 HORIZONTAL
EDGE 37 36
EDGE 36 35
EDGE 35 34

...
```

Listado 3: Ejemplo de aristas

4.1.5.3.5 Definición de las esquinas

Las esquinas se usan para localizar al robot. Tienen la facultad de poder localizar completamente al robot, y por tanto debe usted ser muy cuidadoso a la hora de escogerlas. Su sintaxis es la siguiente:

`CORNER X-Coord Y-Coord Orientation Type`

- **`X-Coord`**: Es la coordenada X del punto de intersección de las paredes que determinan la esquina.
- **`Y-Coord`**: Es la coordenada Y del punto de intersección de las paredes que determinan la esquina.
- **`Orientation`**: Para la orientación se usa el siguiente criterio mostrado en la Figura 40 (tanto para las esquinas convexas como para las cóncavas):

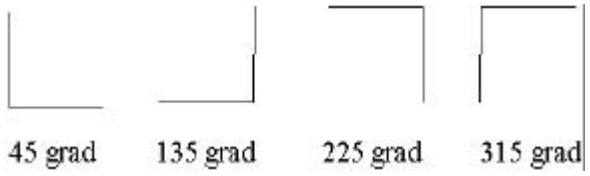


Figura 40: Criterio de orientación para las esquinas

- **Type**: Tenemos que indicar si la esquina es cóncava o convexa con los parámetros `CONCAVE` y `CONVEX` respectivamente.

En el Listado 4 mostramos un ejemplo extraído del fichero *interpoints.txt* real:

```
;; Door in PC-Labor2
CORNER 16800  7700  45 CONVEX
CORNER 15300  7500 135 CONVEX

;; Doors in Prozess-Labor2
CORNER 13150  7700  45 CONVEX
CORNER 12000  7700 135 CONVEX
CORNER  2400  7000  45 CONVEX

;; Supporting pillar in Prozess-Labor2
CORNER  8100  7500 135 CONVEX
CORNER  7500  7500  45 CONVEX

; Tables in PC-Labor2
CORNER 17140  1660 315 CONVEX
CORNER 15390  1660 225 CONVEX
CORNER  18760  5550 315 CONVEX

;Tables in Prozesslabor2
CORNER 13410  3800 315 CONVEX
CORNER  9890  1660 315 CONVEX
CORNER  9890  4340  45 CONVEX
CORNER 11160  4340  45 CONVEX
CORNER  8110  1660 225 CONVEX
CORNER  8110  4340 135 CONVEX

...
```

Listado 4: Ejemplo de esquinas

4.1.5.3.6 Recomendaciones generales para elaborar el fichero *interpoints.txt*

La información contenida en el fichero *interpoints.txt* tiene una gran influencia sobre la navegación del robot. Si escoge adecuadamente esta información, entonces el robot viajará de forma adecuada y se tendrá localizado de forma sencilla. Si por el contrario esta información no se elabora bien, entonces el robot tendrá muchos problemas para la navegación y la localización. Por tanto, debería seguir los siguiente consejos generales:

1. El número del punto inicial y los puntos finales se definen primero en el control de la visita guiada (Tour guide), y su número en el fichero *interpoints.txt*, y su posición y orientación deben coincidir con aquellos definidos en el control de la visita guiada.

2. La elección de los puntos intermedios para ayudar al robot a navegar deben seguir las siguiente reglas:

   ➔ Deberían estar separados (no es una buena idea que estén más lejos de 4 o 5 metros), pero tampoco deben estar muy juntos. No recomiendo en ningún caso que estén más cerca de un metro, y ese límite debería usarse sólo para situaciones particulares, donde la localización sea difícil o el robot tenga tendencia a perderse.

   ➔ Cuando el robot cruce una puerta, debe situar puntos intermedios a ambos lados de la puerta, con las coordenadas correspondientes centradas respecto a la puerta.

   ➔ Cuando se sitúa el punto intermedio en un pasillo, debe centrarlo de forma que tenga la misma distancia a ambas paredes del pasillo.

3. Cada punto intermedio debería tener asociado un método de localización, si esto es posible. El método preferido es el de las esquinas, porque permiten localizar completamente al robot. Después de las esquinas vienen los pasillos y las paredes. Los pasillos son normalmente más exactos, pero las paredes son más versátiles, porque sólo se necesita una pared y no dos como en el caso de los pasillos para poder localizar al robot. Una buena combinación de paredes y pasillos sería una solución magnífica.

   ➔ Las mejores esquinas para la localización son las de los pasillos, porque son grandes y fiables. Los armarios también ofrecen muy buenas esquinas, y los pilares (preferiblemente los grandes), también. Otra posibilidad para permitir la localización en situaciones difíciles donde haya pocas referencias es el empleo de los cajones de las mesas, pero esto dependerá mucho de las sillas existentes que este método funcione o no. No olvide las esquinas cóncavas dejadas por las paredes en las habitaciones cerradas, pueden ser muy útiles.

- ➔ Use pasillos cuando tenga un camino muy largo sin puertas a cruzar. Los puntos intermedios correspondientes deben estar centrados en ese pasillo.
- ➔ Las paredes son muy recomendables cuando hay armarios, y para los puntos intermedios justo antes de una puerta. Es decir, de los dos puntos intermedios asociados a una puerta, use una pared para localizarse en éste del lado del pasillo: será útil en ambas direcciones de la navegación. También se pueden usar paredes donde quiera que haya una superficie grande y recta (armarios y objetos similares).

4. Intente que las aristas que conectan los puntos intermedios tengan "visibilidad directa". El robot también puede navegar con aristas sin visibilidad directa, pero esta navegación está más afectada por los obstáculos presentes en el camino.
5. Elabore caminos alternativos para el robot, por si el camino inicialmente planificado quedara bloqueado.
6. Sea especialmente cuidadoso escogiendo los puntos intermedios y los métodos de localización en la habitaciones con mucho mobiliario. Sin ayuda extra, el robot se perderá probablemente, o se estrellará contra sillas y otros objetos "invisibles" a sus sensores (las esquinas de los cajones de las mesas son una muy buena ayuda).

4.2 Resultados de los tests

En este subcapítulo describiremos brevemente cómo fueron los resultados de los tests de las Especificaciones del test del subcapítulo 3.5.

Durante la fase de desarrollo, se consideraron los casos de test del subcapítulo 3.5.4. Pudimos observar que todos los tests planificados tuvieron éxito, y el control de la visita guiada (Tour guide) del otro estudiante [12] cumplía con las funciones y capacidades esperadas. Por tanto, se pudo completar el software e integrarlo en el sistema completo.

Los tests mostraron que la eficiencia de la navegación es fuertemente dependiente de cómo de bien se definen los puntos intermedios (vea subcapítulo 4.1.5.3). El robot calcula el camino sin errores y se localiza siempre que sea posible, también sin errores. Si no se pudo localizar, eso se debe a que los obstáculos tapaban las referencias de localización o bien a la ausencia de dichas referencias.

Me gustaría resaltar, que el robot no puede navegar sin referencias de localización. Si se esconden o bloquean sistemáticamente dichas referencias, el robot se perderá y comunicará errores inexistentes. Solución: una buena preparación de la visita guiada (Guided tour) y suficientes referencias de localización.

5 Conclusiones

5.1 Resumen

Para este proyecto fin de carrera se ha desarrollado un Software de Navegación para el robot móvil RobIAS para una visita guiada a través del Instituto IAS. El primer paso fue el aprendizaje sobre ciertos aspectos de Robótica, y especialmente las clases de ARIA y el programa Saphira desarrollado por ActivMedia Robotics, el fabricante del robot.

Después de haber adquirido suficientes conocimientos sobre esta herramienta de desarrollo, empecé a trabajar con el Modelo de Procesos del Instituto IAS, que era completamente nuevo para mí (ver Anexo para más información al respecto). También intenté en colaboración con Kristian Dencovski (el otro estudiante que colaboraba en el proyecto global) definir una interfaz entre mi software de navegación (Navigation software) y su control de la visita guiada (Tour guide). Al principio esto no fue del todo posible, porque ambos proyectos estaban en sus fases iniciales y por tanto aún no demasiado definidos, pero avanzamos mucho en esa dirección.

Como resultado de estas reuniones y con la opinión de nuestro tutor de proyecto en Alemania (que era el mismo para ambos), decidimos usar la tecnología ActiveX-COM como solución para nuestros problemas de comunicación, y por tanto empecé a aprender sobre esta tecnología.

Después de un mes de pausa en Sevilla debido a mis exámenes de diciembre, continué con mi aprendizaje sobre ActiveX y ocupé mi tiempo con los importantes documentos System Architecture (arquitectura del sistema) y Software Components (componentes de software). Este proceso no fue muy costoso porque durante el aprendizaje con Saphira y ARIA ya había realizado algunos programas de prueba que más tarde me facilitaron mucho la labor de definir la arquitectura final. La implementación e integración de la arquitectura y los componentes se realizó de forma simultánea, porque el simulador proporcionado por ActivMedia Robotics es de una gran calidad: los programas que funcionaban en el simulador, solían hacerlo también en el robot, de manera que pude integrarlos y probarlos en el robot a la vez, sin tener que usar el robot a diario de forma intensiva, lo cual me hizo ahorrar mucho tiempo.

Al terminar el programa, pude terminar de escribir todos los documentos requeridos, de forma que al final pude entregar mi proyecto fin de carrera (que allí denominábamos Master Thesis) en Alemania. Después hube de traducir la documentación al español, realizar algunos cambios en la misma, añadir el Anexo y defender también el proyecto fin de carrera en la Escuela Superior de Ingenieros de la Universidad de Sevilla.

El resultado de mi proyecto es un software que controla la navegación del robot móvil RobIAS a través del Instituto IAS, desde un punto de inicio predeterminado hacia cualquier punto final deseado en el Instituto. Todo esto se realizó usando la reducida información provista por los sensores integrados en el robot (sonares, láser y detectores de choque), y es posible gracias a los rápidos y eficientes algoritmos que se desarrollaron y a la también reducida información almacenada sobre el entorno, que puede ser fácilmente modificada en caso de que el entorno cambie, o incluso usada para entornos completamente distintos de los del Instituto IAS (siempre que estén basado en la estructura típica de oficina). Es decir, el software no fue optimizado para el caso particular del Instituto IAS, sino que puede ser usado en cualquier sitio que tenga una construcción con estructura de tipo oficina.

Se desarrollaron métodos para evitar colisiones, localizar el robot, calcular caminos de coste mínimo y trayectorias alternativas, para garantizar la fiabilidad de la navegación (en tanto que fuese posible, de acuerdo con las limitaciones del hardware de percepción del entorno).

La determinación de los puntos finales a ser alcanzados y las descripciones habladas en los mismos son parte del trabajo del otro estudiante implicado en el proyecto global [12]. Ambas partes, el software de navegación (Navigation software), y el control de la visita guiada (Tour guide) de este estudiante, están perfectamente sincronizadas y diseñadas para funcionar juntas. Además, el software de navegación realizado para este proyecto fin de carrera ofrece libertad máxima al control de la visita guiada para escoger los puntos a visitar, pues es altamente flexible.

5.2 Experiencias

En primer lugar, me gustaría resaltar mis impresiones personales realmente buenas que he tenido en estos 8 meses que he estado ocupado con mi proyecto fin de carrera.

Mi tutor del proyecto en Alemania, Pascal Jost, siempre estaba dispuesto a ayudarme (incluso cuando apenas tenía tiempo debido a su apretada agenda), y no sólo era simpático y de ayuda, sino que además tenía unos altos conocimientos técnicos para apoyar eficientemente mi trabajo.

Para Kristian Dencovski (el otro estudiante implicado en el proyecto global) sólo tengo buenas palabras: nuestro trabajo conjunto fue realmente productivo y satisfactorio, pero sobre todo, él fue una de las personas que más me ayudo a mi entendimiento de la cultura alemana con la que tuve que convivir la mayor parte de los meses dedicados al proyecto.

Los compañeros que trabajan el Instituto IAS también fueron realmente agradables y de ayuda.

También me gustaría agradecer al Profesor Göhner (director del Instituto IAS) la oportunidad de realizar mi proyecto fin de carrera en el Instituto IAS, poniendo todos los medios necesarios a

mi disposición, incluso cuando no estaba registrado como estudiante normal de la Universidad de Stuttgart, sino como estudiante de programa de intercambio.

Por último agradecer a Jorge Chávez Orzáez, mi tutor en la Universidad de Sevilla, por no encontrar inconveniente en que realizara mi proyecto en Alemania y por su ayuda en la fase final de la realización del proyecto.

En cuanto al Modelo de Procesos del IAS, puedo decir que me ayudó mucho con mi trabajo. Al principio pensaba que solamente era una carga, pero después me di cuenta de cuán equivocado estaba, porque organiza el trabajo de una forma realmente buena y fuerza a realizar todo de una forma más sistemática. Eso se tradujo al final en ahorro de tiempo, eficiencia en el trabajo y un producto final de gran calidad.

He estado interesado durante años en el campo de la robótica. Este proyecto me dio la oportunidad de trabajar con un robot real y profesional, y no solamente con simulaciones como había hecho hasta entonces, y la experiencia resultante fue realmente buena. Mis habilidades con la programación también se han mejorado, y en general me siento ahora mucho más capaz de trabajar en el mundo real de la industria después de los conocimientos adquiridos en el Instituto IAS.

5.3 Problemas

Por supuesto, también aparecieron muchos problemas durante el curso de mi proyecto fin de carrera, que a veces constituyeron un desafío para mis capacidades.

Cuando empecé estaba realmente perdido con el Modelo de Procesos del IAS, porque no era normal para mí planearlo todo de una forma tan sistemática. Pero después vi que este esfuerzo merecía la pena, y aunque en muchas ocasiones me pareció que este Modelo era simplemente una pérdida de tiempo, la realidad es que al final los beneficios compensan el trabajo adicional que hay que realizar al principio.

La interfaz ActiveX también me dio muchos problemas, porque no pude encontrar una solución simple al problema de la comunicación sin usar ficheros en un periodo relativamente corto de tiempo. Es éste un campo complejo de la programación que hubiese requerido mucho más tiempo del que yo tenía para resolver el problema.

También experimenté problemas con la complejidad de ARIA y Saphira. Al principio estimé que el programa era sencillo de manejar, pero cuanto más lo usé, más cosas descubrí de cuya existencia no tenía ni las más mínima idea. La documentación disponible es muy numerosa y de difícil lectura, y pasé por alto ciertos conceptos importantes en el contacto inicial. Después, durante la implementación de mi programa, estos conceptos hicieron acto de aparición en forma

de problemas, y tuve que reestructurar mis ideas en varias ocasiones. Ahora puedo decir que una lectura un poco más detallada de la literatura disponible me hubiera ayudado mucho.

También se presentaron algunos problemas de hardware relativos a los puertos de comunicación en el robot, que persistieron hasta que mi tutor encontró el origen de los mismos y los subsanó.

Apéndice A Índice de Figuras

Apéndice B Índice de Listados

Apéndice C Terminología y Abreviaturas

ActiveX	Tecnología de Microsoft que permite la comunicación entre diferentes componentes de software programados en diferentes lenguajes de programación.
ARIA	Interfaz para aplicaciones de control de robots orientadas a objeto, para la línea de robots móviles inteligentes de ActivMedia Robotics.
Camino	Trozo a ser cubierto por el robot entre dos puntos intermedios. Es el curso real seguido por el robot, incluyendo la evitación de obstáculos.
COM	**C**omponent **O**bject **M**odel (parte de la tecnología ActiveX)
DLL	**D**ynamic **L**ink **L**ibrary (ficheros de Microsoft que encapsulan funciones programadas por el usuario o bien funciones que vienen ya predefinidas)
Embedded-PC	Ordenador integrado en el sistema (en RobIAS en nuestro caso).
Gradient navigation	Traducido por „Navegación por el método del gradiente“, es un planificador de caminos en tiempo real basado en el método del gradiente desarrollado por Kurt Konolige. Determina el camino óptimo para el robot, en tiempo real, que puede tomarse teniendo en cuenta los obstáculos locales, detectados por los sensores láser y sonares, y la información global del mapa, tal como la localización de las paredes y otros obstáculos estructurales.
GUI	**G**raphic **U**ser **I**nterface (Interfaz gráfica de usuario)
Guided tour	Traducido por “Visita guiada”, es la suma de los dos procesos *Tour guide* (control de la visita guiada) y *Navigation software* (software de navegación), que fueron implementados independientemente para al final trabajar juntos dando como resultado la *Guided tour*.

IAS	**I**nstitut für **A**utomatisierungs- und **S**oftwaretechnik (Instituto de Ingeniería de Software y Automatización Industrial)
Láser	Dispositivo usado por el robot RobIAS que proporciona una información detallada del entorno: una matriz con lecturas en un rango de 180 grados.
Localización	Proceso para encontrar la posición y orientación reales del robot, necesitado porque la estimación de dichos valores por medio de la odometría tiene errores asociados.
MT	**M**aster **T**hesis
Navigation software	Traducido por “Software de navegación”, es el resultado de mi proyecto fin de carrera: un software de navegación para mover el robot de su posición inicial al punto final deseado, evitando los obstáculos del camino a través un camino eficiente.
Odometría	Método para estimar la posición y orientación del robot, mediante la lectura de los valores de los codificadores de las ruedas, y el uso de esta información junto con el modelo cinemático del robot.
Punto intermedio	Punto de referencia usado para la navegación; el robot navega usando una secuencia de dichos puntos intermedios, de forma que el problema de navegación “se reduce” a la navegación entre dos puntos intermedios (que no están muy lejos del uno del otro).
RobIAS	Robot móvil de ActivMedia Robotics, basado en el Pioneer 2-DXE con Peoplebot Extension.

Saphira	Saphira es un entorno de desarrollo de aplicaciones robóticas, y la librería de Saphira es un grupo de rutinas para la construcción de clientes. La librería de Saphira integra un número de útiles funciones de ARIA para el envío de comandos al servidor, recopilación de información de los sensores del robot y su agrupación para mostrarlos en una interfaz de usuario gráfica. Adicionalmente, Saphira soporta funciones de alto nivel para el control del robot y la interpretación de los sensores, incluyendo un sistema de navegación por el método del gradiente (Gradient navigation).
Sónar	Dispositivo integrado y usado por el robot RobIAS que proporciona una pobre información del entorno, pero detecta obstáculos que se mueven rápido que no pueden ser detectados por el sensor láser.
Tour guide	Traducido por "Control de la visita guiada", es la parte del proyecto completo de *Guided tour* (visita guiada) que gestiona dicha visita guiada, determinando los puntos a ser visitados y reproduce las descripciones habladas asociadas a dichos puntos.
Trayectoria	Curso global planeado por el robot como resultado de conectar todos los puntos intermedios que deben alcanzarse. El camino debería seguir a la trayectoria tanto como fuese posible.
WLAN	**W**ireless **L**ocal **A**rea **N**etwork. Tecnología de red por radio, sin cables.
WSA	**W**ahl**s**tudien**a**rbeit (trabajo de investigación voluntario según el sistema universitario alemán)

Apéndice D Bibliografía

[1] **Sirotin, Victor:** *ActiveX/DCOM-Programmierung mit Visual C++ 6*, Bonn; Reading, Mass. [u.a.] : Addison-Wesley-Longman, 1999.

[2] **ActivMedia Robotics:**
http://robots.activmedia.com/

[3] **ActivMedia Robotics:**
ARIA and Saphira Manuals and References, ARIA and Saphira installation

[4] **Konolige, Kurt**: *A gradient method for realtime robot control*, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[5] **Morris, John:** *Data Structures and Algorithms: Dijkstra´s Algorithm*
http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html

[6] **Göhner, Peter :** *Skript zur Vorlesung Softwaretechnik I*, IAS, Stuttgart, 2002

[7] **Microsoft:** *MSDN Library*
http://msdn.microsoft.com/library/

[8] **IAS Regulations Manuel Version 3.0**

[9] **Leo (Link Everything Online) Dictionary**, Informatik der Technischen Universität München
http://dict.leo.org/?lang=en

[10] **Página web del Instituto IAS:**
www.ias.uni-stuttgart.de

[11] **Daniela Hauck**, *Generierung einer lokalen Karte durch Fusion von zeitlich aufeinanderfolgenden Messdaten eines Laserscanners*, Diplomarbeit Nr.: 1800, IAS, Stuttgart, 2001

[12] **Kristian Dencovski ,** Wahlstudienarbeit Nr.: 1884, IAS, Stuttgart, 2003

Apéndice E Corrección de errores más frecuentes

En este Apéndice se presentan algunos problemas usuales que aparecen durante la instalación o el funcionamiento del software de navegación (Navigation software). Este Apéndice no pretende ser una guía completa con todas las situaciones de error posibles que podríamos tener, sino simplemente una ayuda extra para algunos errores comunes que aparecían durando el desarrollo del software de navegación.

E.1 No puedo acceder al ordenador integrado del robot usando el VNC Viewer

Si cuando está tecleando el servidor ias198 con el VNC Viewer recibe este mensaje mostrado en la siguiente Error figure 1:

Error figure 1: Error usando el VNC Viewer

- **Solución 1**: Asegúrese de que el robot está encendido.
- **Solución 2**: Asegúrese de haber tecleado correctamente el VNC Server.
- **Solución 3**: Asegúrese de que módem Wireless LAN esté conectado y no muy lejos del robot.

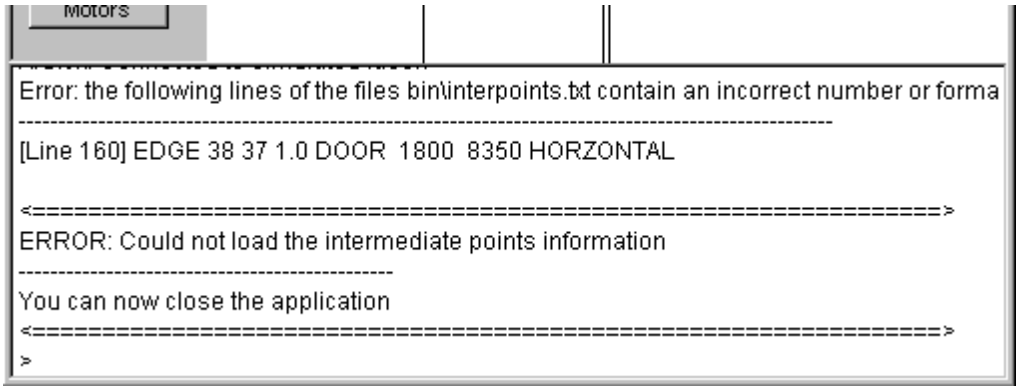
E.2 El láser no funciona

Si no puede ver los puntos verdes del láser en la ventana del software de navegación:

- **Solución 1**: Asegúrese de que el láser está encendido (el interruptor está en la parte trasera del mismo). Debería ver el LED frontal de color verde, no parpadeante.
- **Solución 2**: Si ha iniciado con el ratón enchufado, reinicie el embedded PC (ordenador integrado de RobIAS).

E.3 Hay un error con los puntos intermedios

Si al iniciar el programa de navegación recibe un mensaje como éste mostrado en la Error figure 2:



Error figure 2: Error cargando los puntos intermedios

- **Solución:** Ha introducido mal la sintaxis de los parámetros en el fichero *interpoints.txt*. En la ventana o en el fichero *errors.dat* que se halla contenido en la carpeta *C:\Exchange\Projekte\RobIAS\Saphira\bin* del robot, encontrará qué líneas del fichero le están dando problemas. Lea cuidadosamente estas líneas, y usando la información del subcapítulo 4.1.5.3, corrija los fallos.

E.4 No se encuentra la trayectoria al destino o el programa se cuelga

Si su programa no encuentra la trayectoria al principio, cuando no debiera haber problemas (no confundir esto con el caso en que no se encuentran caminos alternativos después de que el robot se haya quedado bloqueado), o se cuelga sin razón aparente:

- **Solución:** Casi seguramente ha cometido usted un error al escribir el fichero *interpoints.txt*. Esto puede ser bien un error lógico (por ejemplo, no haber conectado dos puntos, lo cual explicaría por qué no se encuentra ninguna trayectoria), o bien ha tecleado usted algo mal (recuerde que el programa no reconoce todos los errores posibles que pudieran aparecer. En tanto que no hay un informe de errores en este caso, debe leer muy cuidadosamente todo el fichero para encontrar los errores.

E.5 La ventana de Saphira no puede cerrarse

Si ha intentado cerrar la ventana de Saphira sin seguir el método recomendado (vea el subcapítulo 4.1.2.4: Interrupción del software de navegación (Navigation software) en cualquier momento), probablemente tendrá este problema: la ventana de Saphira no se cierra y el programa se queda residente en memoria.

- **Solución:** Al estar usando el VNC Viewer para ver la ventana de Saphira, pulsar Ctrl+Alt+Del no mostrará el administrador de tareas del ordenador de RobIAS, sino el de acceso remoto al mismo. Debe conectar un teclado (y opcionalmente un monitor) al robot, y con él pulsar Ctrl+Alt+Del para cerrar Saphira. En este punto puede ir al paso 3 del subcapítulo 4.1.2.5 (Terminación normal de la visita guiada (Guided tour).

E.6 Se ha perdido la conexión con el láser

Si ve este mensaje en la ventana de Saphira durante la navegación, significa normalmente que las baterías del robot tienen ya una tensión muy baja.

- **Solución:** Detenga el software de navegación (Navigation software) (vea 4.1.2.4: Interrupción del software de navegación (Navigation software) en cualquier momento), apague el láser, cargue las baterías y reinicie la visita guiada (Guided tour).

E.7 El control de la visita guiada (Tour guide) no reacciona

Si está intentando iniciar el control de la visita guiada (Tour guide) después de un error, y dicho control de la visita guiada no termina o no reacciona:

- **Solución:** Debe cerrar la ventana del control de la visita guiada, cerrar el software de navegación y reiniciar el ordenador del robot y entonces reiniciar todo el proceso.

E.8 El robot se pierde muy fácilmente

Si durante la navegación el robot tiene muchos problemas encontrando el sitio correcto, o ve caminos bloqueados donde no hay obstáculos:

- **Solución:** Quizá haya definido un fichero *interpoints.txt* con errores o con muy poca información de localización. Piense que cuanto más complicado es el entorno, más ayuda necesita el robot. Esto se traduce en más referencias de localización.

Anexo Las normas de calidad en el Instituto IAS

Introducción

En este Anexo se va a dar una información general de cómo se aplican se aplican las normas de calidad ISO 9001 en el Instituto IAS de la Universidad de Stuttgart (Alemania).

El *Insitut für Automatisierungs- und Softwaretechnik* (IAS) (Instituto de Ingeniería de Software y Automatización Industrial) ha sido certificado de acuerdo con la norma DIN EN ISO 9001. De todas las normas disponibles para la certificación (9001, 9002, 9003), ésta es la más exhaustiva y aplicable a todos los tipos de empresas.

El IAS se considera a sí mismo un servicio de empresa para sus clientes; éstos son los estudiantes, personal laboral, socios en la industria, y por último, pero no por eso en peor consideración, la sociedad, que está comprometida como parte de la institución educacional llamada Universidad.

El IAS ofrece los siguientes servicios:

- Educación de estudiantes en aprendizaje e investigación en las clases, cursos, internados y mediante el asesoramiento de los estudiantes durante sus proyectos fin de carrera;
- Un entorno profesional de trabajo;
- Asesoramiento durante la tesis doctoral;
- Cooperación con empresas en trabajos de investigación;
- Intercambio científico con otras universidades en Alemania y de otros países.

A cambio, los estudiantes y el personal realizan servicios en forma de proyectos fin de carrera (Master Thesis, MT), tesis doctorales y otras actividades científicas para el IAS.

En el marco un sistema de gestión de calidad orientado a procesos, todos los procesos del trabajo y servicios son registrados; se representan en forma de unas normas de procedimiento en el Manual de Gestión de Calidad (QMM, Quality Management Manual) y se supervisan mediante la aplicación de instrumentos adecuados, como por ejemplo, encuestando a personas implicadas en el proceso de trabajo. Las medidas de comprobación se usan para registrar las desviaciones de los objetivos fijados, para reconocer posibilidades de mejora en la representación y ejecución de los procesos de trabajo, y para implementar las correcciones de forma adecuada.

En el marco de los proyectos fin de carrera en el IAS, los estudiantes comienzan su trabajo de acuerdo a un proceso predefinido, para asegurar la calidad de su trabajo y evaluarlo junto con sus tutores. El proceso del proyecto fin de carrera se define en el Modelo de Procesos del IAS, que ofrece 5 clases de modelo de proceso, y se ilustra profundamente en el Manual de Regulaciones del IAS. Después sigue una descripción del Entorno de Desarrollo del IAS (DE, Development Environment) donde se explican todas las herramientas necesarias para la ejecución de los proyectos fin de carrera. Adicionalmente al Manual de Regulaciones del IAS, los procedimientos para los proyectos fin de carrera se describen en forma de normas de procedimiento en el QMM. Las clases del modelo de proceso se resumen además en el QMM y están integrados en los procedimientos UPMT (Undergraduate Projects and Master Theses).

En la dirección http://www.ias.uni-stuttgart.de/iso9000/index.htm puede encontrar más información sobre el Manual de Calidad (Quality Manual) del Instituto (sólo disponible en alemán).

Condiciones del IAS

El Instituto IAS exige ciertas condiciones a los estudiantes que realizan en él su proyecto fin de carrera, y que deben cumplirse, de acuerdo con las normas de calidad referidas anteriormente. Entre las más importantes de estas condiciones están:

- **Duración del proyecto**: La duración efectiva de un proyecto fin de carrera es de 6 meses. Al principio del proyecto, el estudiante y su tutor establecen una fecha límite preliminar, donde se tienen en cuenta los factores que pudieran afectar a la duración del proyecto. La prolongación a esta fecha límite es una excepción a la regla y debe ser bien razonada, además de solicitada mediante un documento en el que se contenga un razonamiento de la misma.
- **Observar las reglas de trabajo en las salas de ordenadores y laboratorios**.
- **Elaboración de los documentos pertinentes** (que se detallarán después), de acuerdo con el tipo de proceso escogido.
- **Respeto de los plazos intermedios de control**
- **Presentación del proyecto**, a realizar 6 semanas antes de la conclusión prevista del proyecto.
- **Memoria del proyecto**, cuyas características también han de ajustarse a unas ciertas exigencias. Un ejemplar de dicha memoria original está disponible para el Tribunal si así lo desea.

- **Elaboración de un CD** denominado AISA-CD, en el que se hallan contenidos los resultados del proyecto fin de carrera, además de otras presentaciones de interés acerca del Instituto IAS. Una copia de dicho CD está disponible para el Tribunal si así lo desea.

- **Participación en los coloquios del IAS**, donde los estudiantes presentan sus proyectos fin de carrera, y ocasionalmente el personal científico del IAS expone el resultado de sus trabajos de investigación. Con ello se pretende que los estudiantes tengan conocimiento de qué temas se están investigando en el Instituto.

Modelo de proceso del IAS

La información completa puede encontrarse en la dirección siguiente de internet: http://www.ias.uni-stuttgart.de/pm/index.htm. A continuación sólo detallaré los aspectos que más afectan a la realización del proyecto fin de carrera, y en especial, a aquellos que tienen que ver con los documentos que hube de escribir como parte de las exigencias de las normas de calidad impuestas en el Instituto.

El Modelo de Proceso del IAS (IAS-PM, IAS Process Model) define 5 clases diferentes de modelo de proceso:

- Modelo para el desarrollo de software
- Modelo para el desarrollo de hardware
- Modelo para el desarrollo de sistema
- Modelo para proyectos concepcionales
- Modelo abierto

El modelo de proceso (en cualquiera de sus 5 clases) consiste en 4 submodelos, tal como se muestra en la figura siguiente:

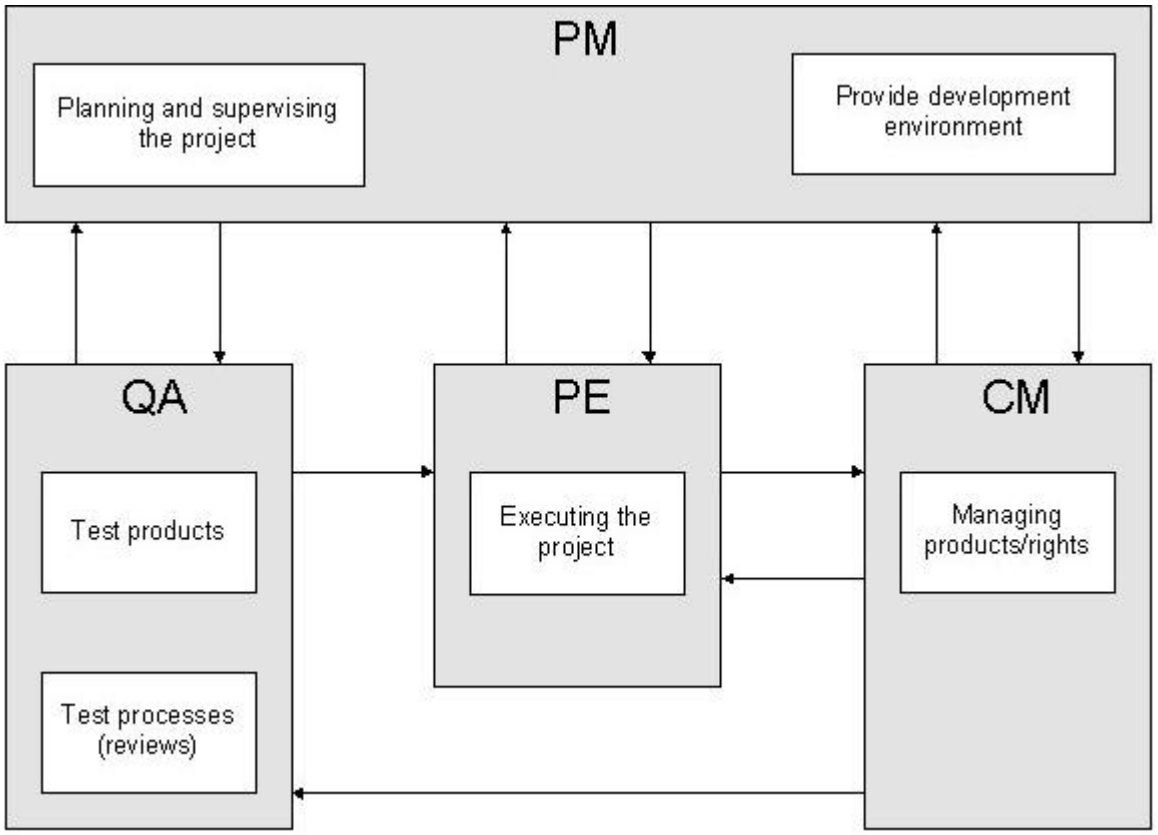


Figura: Los cuatro submodelos PE, QA, CM y PM

Además del submodelo Ejecución del Proyecto (Project Execution, PE), están los submodelos Garantía de Calidad (Quality Assurance, QA), Gestión de la Configuración (Configuration Management, CM) y Gestión del Proyecto (Project Management, PM). Las tareas de garantía de la calidad comprenden la definición de requerimientos como métodos de test y criterios de test y todo el testeo de los productos. Los resultados se comunican a la gestión del proyecto. El objetivo de la gestión de la configuración es crear un producto, por ejemplo, un documento, que es identificable de forma única en sus características funcionales y exteriores. Esta identificación proporciona un control sistemático de los cambios y asegura integridad. La gestión del proyecto planifica, comprueba y controla las actividades internas de los proyectos. Es además la interfaz a unidades externas del proyecto o de otros proyectos. El modelo del proceso controla la interacción entre los submodelos individuales. Esto permite la asignación de varias tareas a diferentes personas o equipos y garantiza además un procedimiento uniforme.

Mientras que los submodelos PM, CM y QA son los mismos para todas las clases del modelo de proceso, el submodelo PE es completamente diferente para cada clase. El modelo PE tendrá el nombre SWD (Software Development) en nuestro caso para diferenciarlos de los otros nombres que se asignan para las otras clases.

La estructura organizacional del IAS-PM es la que se muestra en la siguiente figura. No entraremos en más detalles, puesto que para nuestro caso no resultan de especial interés. Para más información diríjase a la Part I en http://www.ias.uni-stuttgart.de/pm/index.htm.

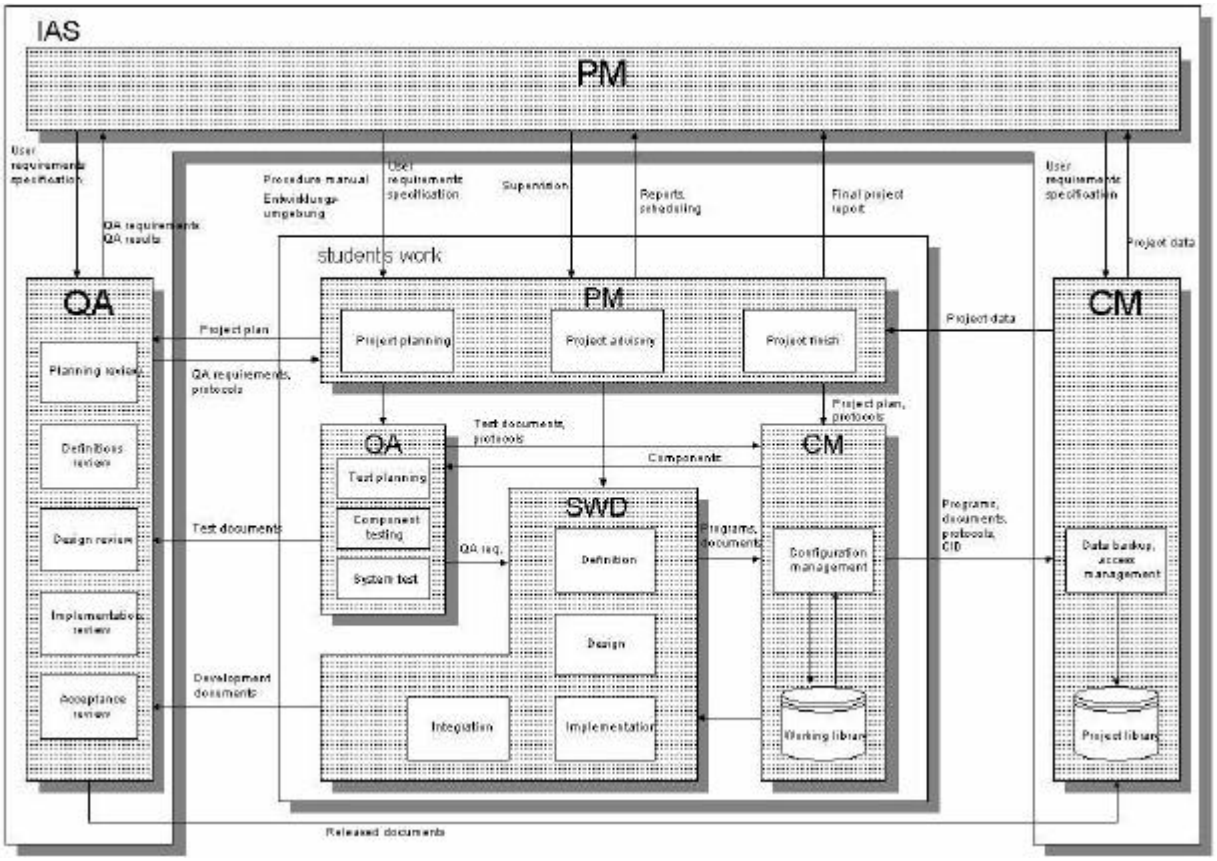


Figura: Estructura organizacional del IAS-PM

Modelo para proyectos de desarrollo de software

El modelo para proyectos de desarrollo de software (Model for software development projects) define el proceso de proyectos fin de carrera que se centran en la creación de un sistema de software. Durante el desarrollo del software se siguen cuatro fases:

- Fase de definición (SWD 1)
- Fase de diseño (SWD 2)
- Fase de implementación (SWD 3)
- Fase de integración y aceptación (SWD 4)

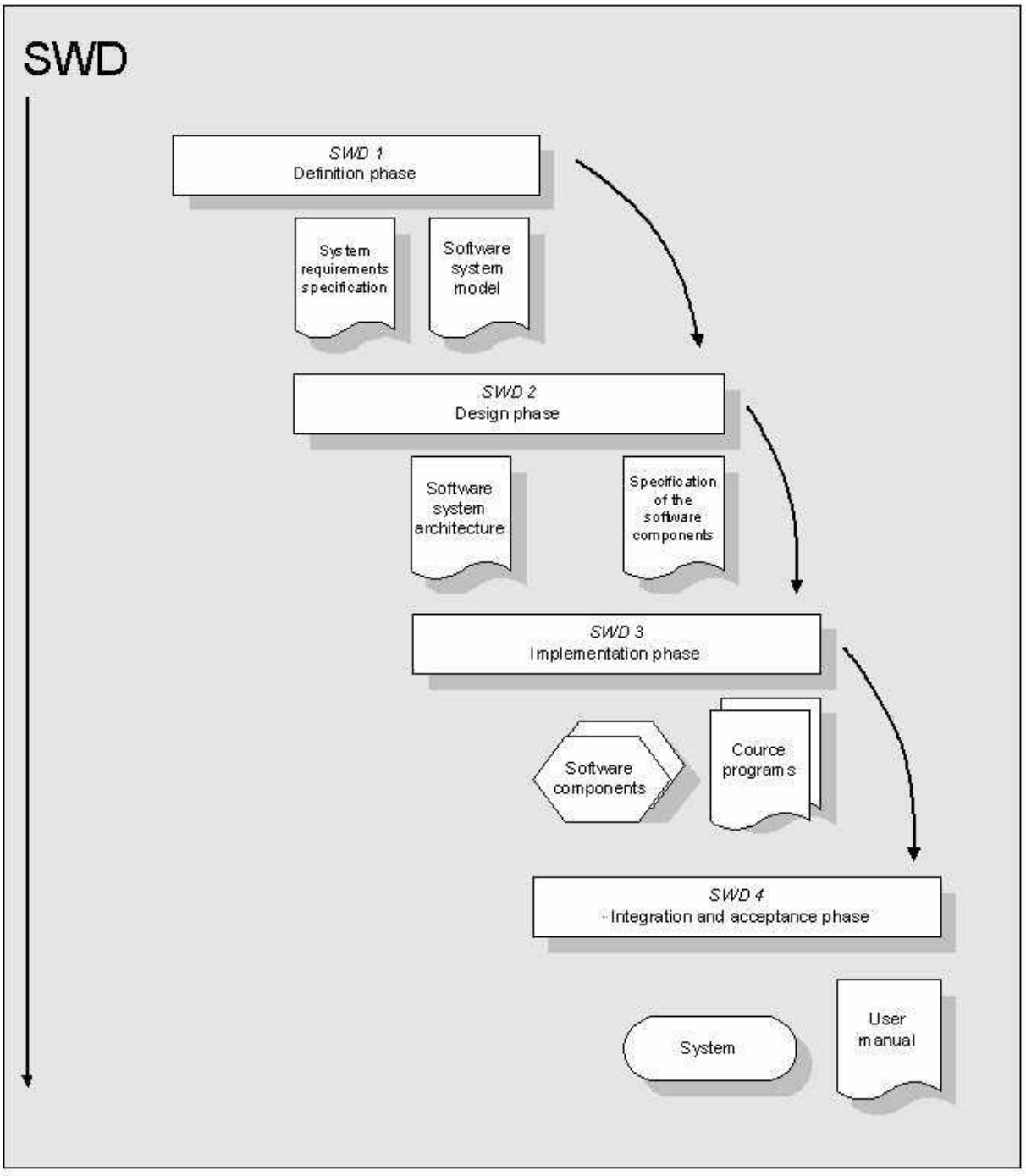
En la tabla siguiente se muestra una tabla con las actividades y productos que hube de realizar representados en las diferentes fases:

	Actividades a realizar	Nombre del documento/producto
SWD 1 Fase de definición		
SWD 1.1	Determinar y definir los requerimientos	System requirements specification
SWD 1.2	Analizar los requerimientos	Software system model
QA 1.2		Definitions review
SWD 2 Fase de diseño		
SWD 2.1	Diseñar la arquitectura del sistema de software	Software system architecture
SWD 2.2	Diseñar y especificar los componentes de sofware	Specification of software components
QA 2.1	Crear las especificaciones del test	Test specification
QA 1.3		Design review
SWD 3 Fase de implementación		
SWD 3.1	Implementar los componentes de software	(Source programs with integrated documentation)
QA 2.2	Testear los componentes de software	Test protocol (components)
QA 1.4		Implementation review
SWD 4 Fase de integración y aceptación		
SWD 4.1	Integrar e instalar el sistema	Installed software system
SWD 4.2	Crear manual de usuario	User manual
QA 2.3	Testear el sistema instalado completo	Test protocol (system)
QA 1.5		Acceptance review

Tabla: Visión general de las actividades y productos en el modelo para proyectos de desarrollo de software

En la siguiente figura se muestra una visión de conjunto del proceso de desarrollo de software:

Figura: Fases y productos en el modelo para desarrollo de software



A continuación vamos a hablar un poco en qué consiste cada una de las fases representadas en la figura anterior:

- **Fase de definición (SWD 1):** En la fase de definición se definen los requerimientos para el sistema de software a desarrollar. Estos requerimientos determinan las características cualitativas y cuantitativas de un sistema desde la perspectiva de un contratista. La

definición del sistema da lugar a los documentos *System requirements specification* y *Software system model.*

- **Fase de diseño (SWD 2):** En la fase de diseño se desarrolla una solución técnica del sistema en la forma de una arquitectura del software del sistema desde los requerimientos del software. Mediante la arquitectura del software de sistema se divide el software completo en componentes de software, que tienen asignadas funciones de sistema individuales. También hay que definir las interfaces entre los distintos componentes. En esta fase se originan los documentos *System architecture* y *Software components.*

- **Fase de implementación (SWD 3):** Durante esta fase se realizan las actividades de programación. El diseño es el punto de comienzo para la fase de implementación. Se asume por tanto que durante la fase de diseño se diseñó la arquitectura del software de sistema y se especificaron los respectivos componentes de software. En esta fase se conciben las estructuras de datos y algoritmos, se estructuran los programas, se documentan y se realizan con un lenguaje de programación, y se crean programas de test de acuerdo con las especificaciones de test. Como resultado de esta fase tendremos los códigos fuente incluyendo la documentación integrada, programas objeto, ficheros compilados y programas de test.

- **Fase de integración y aceptación (SWD 4):** En esta fase, los componentes de software individuales que se han implementado en la fase de implementación deben ser integrados. Después de integrar el sistema completo se envía al test de sistema. Como resultado de esta fase tendremos el *User Manual.*

Todas las fases anteriormente referidas deben ser controladas, y al final de cada una (cuyas fechas fijamos con los denominados *Milestones*), tendremos una revisión, *Review*, con la que controlaremos que todos los objetivos fijados en esa fase se han cumplido.

Documentos generados

Siguiendo las normas de calidad y las diferentes fases que ya hemos mencionado, los siguientes documentos hubieron de ser escritos (en un orden cronológico aproximado, teniendo en cuenta que algunos se podían escribir en paralelo):

Nombre del documento	Finalidad del documento
User Requirements Specification (escrito por el tutor)	Este documento es un resumen de todos los requerimientos académicos básicos que el sistema propuesto debe cumplir. Los requerimientos básicos representan una concentración de características fundamentales del sistema y su descripción en un nivel abstracto. Este documento es el primer producto que describe los requerimientos de un nuevo sistema de una manera general. Aquí el tutor elige la clase de modelo del proceso a seguir (en mi caso, desarrollo de software)
System Requirements Specification	Este documento es un resumen de todos los requerimientos académicos que el sistema necesita cumplir desde el punto de vista del contratista. Contiene la vista académica del sistema en referencia a su función, datos, rendimiento y cualidad. El documento define qué debe hacer el sistema y no cómo.
Project Plan	Este documento consiste en la planificación y definición de la organización y secuencia del proyecto, tiempo y cantidad de recursos para los cuatro submodelos. Es un instrumento usado por la gestión del proyecto para planificar, guiar y supervisar. El *project plan* consiste en la descripción de los paquetes individuales y una representación gráfica de la organización del proyecto con dichos paquetes.
Software System Model – Use Cases	La creación de una especificación de requerimientos del sistema verbal no es suficiente, y por tanto se debe crear un modelo del sistema semiformal o formal, el llamado *software system model*. Este documento describe el concepto académico de forma completa, no ambigua y consistente. En el caso de que haya que desarrollar un sistema de software con una compleja interacción entre el usuario y el ordenador, el *system model* describe cómo el sistema se presenta a sí mismo al usuario y cómo se usa.
Definition Review Protocol	Con este documento se comprueban los documentos *Project Plan*, *System Requirements Specification* y *Software System Model*. Con ello se pretende identificar errores, buscar inconsistencias frente a las especificaciones. Esto se hace con ayuda del tutor, que supervisa la validez de la revisión, y habiendo fijado para ello anteriormente una fecha.

Software System Architecture	Este documento describe la estructura del sistema de software con los componentes de software y sus relaciones/interfaces entre cada uno. Se representan con un diagrama. En adición a la lista de componentes de software con una breve descripción, el documento contiene una definición completa de las interfaces entre los componentes.
Software Components Specification	Este documento define la escala funcional y de rendimiento par cada componente. Así mismo, sirve como fundamento para la implementación de los componentes y es muy importante para el mantenimiento y actualizaciones del sistema en una fecha posterior.
Test Specification	Este documento incluye una descripción de los requerimientos de test y objetivos, métodos de test y criterios de test derivados de los requerimientos así como los casos de test. Se documentarán todos los casos de test, y con ayuda de este documento, se podrá tomar la decisión de si un test tuvo éxito o no.
Design Review Protocol	En este documento se comprueban y revisan los otros documentos *Software System Architecture*, *Software System Components* y *Test Specification*. Adicionalmente se pueden hacer mejores derivadas de las revisiones de la definición.
Implementation Review Protocol	Durante esta revisión se comprueban los programas fuente generados en la revisión de código, y se confirma la adhesión a las regulaciones de programación. Adicionalmente se pueden hacer mejores derivadas de las revisiones del diseño.
User Manual	Este documento contiene toda la información necesaria para usar el sistema. Esto incluye tanto instalación y funcionamiento como corrección de errores.
Test Protocol	Este documento contiene el proceso de tests individuales registrados por el testeador, prestando especial atención a la comparación entre los resultados esperados y los logrados.
Acceptance Review Protocol	Cuando el sistema está listo para ser aceptado, se comprueban su funcionalidad y operabilidad por parte del contratista (el estudiante) y el tutor. Los resultados se registran en este documento.
Project Final Report	Este documento contiene un informe final sobre la realización del proyecto, con un breve resumen del mismo, las experiencias acumuladas durante su desarrollo y los problemas que hubieran

	surgido durante el mismo.
Terminology	Este documento contiene una lista de los términos que más se usan y que son claves para entender el proyecto.
Abbreviations	Este documento contiene una lista de abreviaturas que se usan y que son claves para entender el proyecto.
Literature	Este documento es la bibliografía del proyecto, donde se listan las fuentes de información que hicieron falta para su realización.

Cada documento tiene un determinado estado en cada momento, que puede ser “en progreso” (in progress), cuando el estudiante está elaborando el documento, “enviado” (submitted), cuando se envía al tutor para que lo revise, y “aceptado” (accepted), cuando el tutor da por válido el documento. Además el documento está numerado con una versión: al empezar el documento está “en progreso” y recibe la versión 0.1. En esta fase se puede cambiar el documento a voluntad. La primera versión aceptada recibe el número de versión 1.0, y después de esto, modificaciones posteriores deben ser hechas bajo el estado “enviado” y siempre documentando los cambios. Cada versión posterior se irá numerando con 1.1, 1.2 y así sucesivamente.

A continuación mostramos todos los documentos que aparecen en la lista anterior (sólo en su versión última, que es la definitiva). Recuerde que dichos documentos están todos escritos en inglés, pues éste es el idioma en el que originalmente escribí mi proyecto fin de carrera. De ahí la notación “MT Serrano López” que encontrará en cada página, de “Master Thesis” y mis apellidos, para identificar todos los documentos. Aún así, la primera página de cada documento ofrece información adicional con:

- Tipo de trabajo
- Título del trabajo
- Autor
- Número de versión
- Fecha
- Fichero
- Estado
- Etc.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

User Requirements Specification

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
0	MT	Model for Conceptions	Serrano López	Jo/Kn	<dd.mm.jj>	<dd.mm.jj>

Document:	Version:	Author:	Date:	Status:
User Requirements Specification	1.0	Jost	15.04.03	accepted

File Name:	Pages:	Print Date:	Template:
sw-user-requirements-specification-v10.doc	6	28/05/03 17:28	co-user-requirements-specification.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	27.08.02	in progress	Creation
0.1	Serrano López	Jo	27.08.02	in Bearb.	Creation
1.0	Serrano López	Jo	27.08.02	in Bearb.	
1.0	Serrano López	Jo	27.08.02	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

0 Table of Contents

1 Introduction into the Project

The mobile robot RobIAS should guide visitors through the institute. Therefore, two projects were started. In one project a guided tour including multimedia presentations of the subjects of research will be developed. A second project will focus on the navigation through the institute. This includes path planning activities as well as obstacle avoidance and position approximation.
RobIAS is a PeopleBot build by ActivMedia. It is based on the Pioneer 2DX. The hardware structure of the robot is shown in Figure 1 including a front-view and a back-view. The robot is provided with ten touch-sensitive bumpers (Drucksensoren) in the front and the back to detect crashes. To avoid crashes two kinds of non-tactile sensors are mounted at the front side of the robot. Sixteen ultrasonic sensors (Ultraschallsensoren) are placed in two rows, one about 20 centimeters and the other about 120 centimeters above the ground. They are very fast, but have a low resolution. A laser range finder (Laserscanner) is included, too. This sensor has a very high resolution, but is very slow. So it can not be used for the detection of fast moving a obstacles. E.g. a person crossing the way of the robot. RobIAS is moved by two wheels. Further, a pair of speakers (Lautsprecher) and a microphone (Mikrofon) are included for the communication with persons. For the control a micro controller and a personal computer are mounted inside the robot.

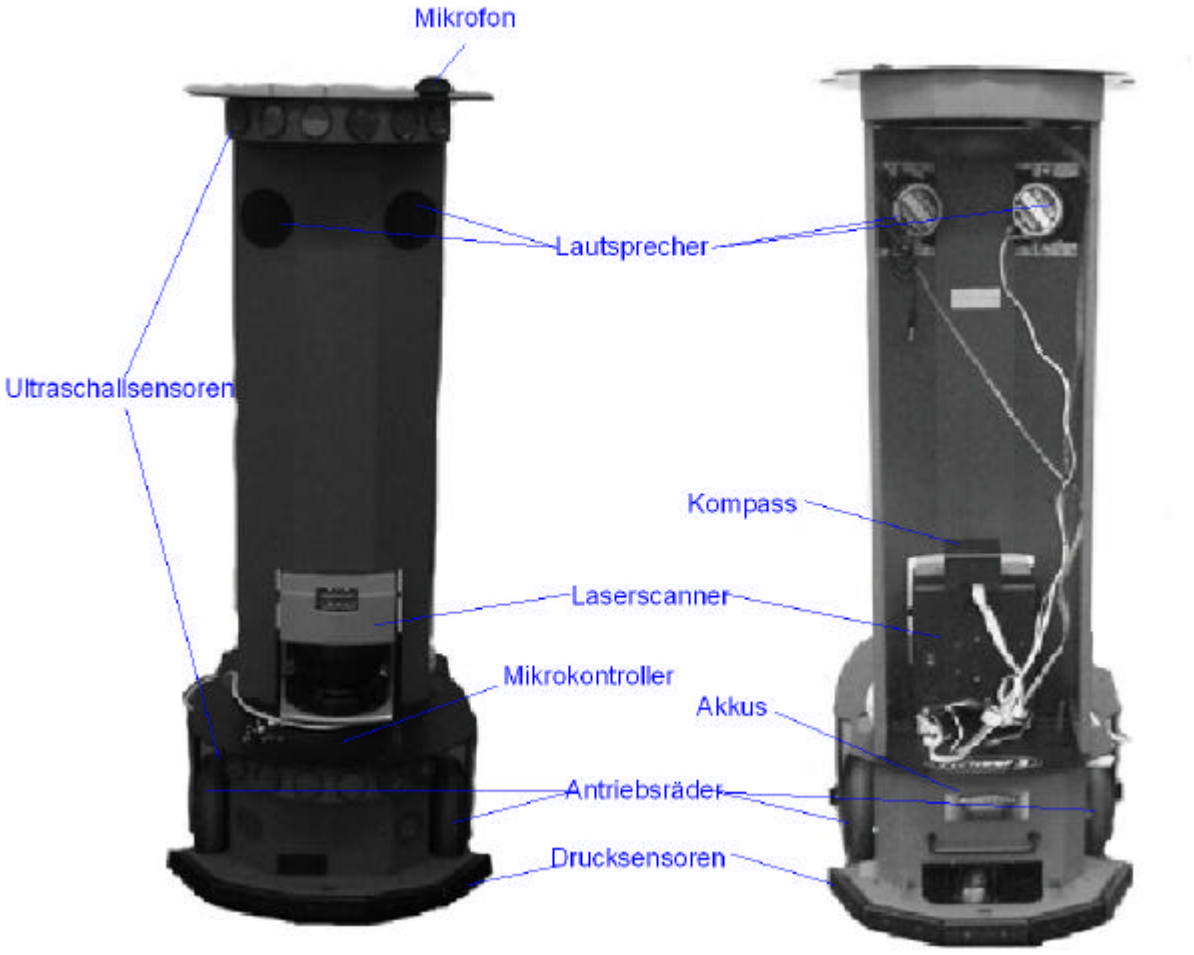


Figure 1: Hardware of RobIAS, the mobile robot

2 Objectives

During this project a control software has to be developed to move the robot RobIAS through institute (or any other known building) autonomously. The robot will need the following skills:

- Planning a path from the current position to a given destination inside the institute (map)
- Move along this path
- Detect deviation from the position reported by the wheel sensors and the actual position of the robot (e.g. shifting by slippery underground)
- Avoid collisions with any obstacle by stopping or adjusting the planned path
- Interaction with the software of the guided tour (other project) which will set the destinations and need information about current position

3 Operational Area

The result of this project will be used to guide visitors through the IAS by the autonomous moving robot RobIAS.

4 Requirements to the Conception

/LA10/ A map has to be build based on sensor measurement

/LA20/ The robot has to plan a path from its current position to a destination based on the map.

/LA30/ The robot has to be moved to the given destination in a save way. Therefore, the robot has to detect obstacles and avoid collisions.

/LA40/ The position of the robot has to be checked and updated if needed. Therefore, a comparison of the environment and the map as well as a compass can be used.

/LA50/ The control software has to provide an interface to interact to an upper layer control which e.g. will provide the destinations to go to (guided tour).

/LA60/ For the concept several existing possibilities and algorithms has to be considered. E.g. Libraries provided with the robot and several existing diploma thesis.

5 Quality Requirements

Product Quality	very high	high	Normal	not relevant
Theory	x			
Functionality		x		
Usability		x		
Applicability			x	
Portability				x

6 Execution

The thesis has to be executed according to the "IAS Process Model" (Model for Software Development). In the following table the documents are listed, which have to be created.

Project Phase			**MT/DA/SA**	**WSA**
CD 1 Definition Phase				
CD 1.1	Analysis of Requirements	System Requirements Specification	X	X
QA 1.2		→ Definition Review	X	X
CD 2 Conception Phase				
CD 2.1	Analysis of the Basis	Basis	X	X
CD 2.2	Creation of the Conception	Conception	X	X
QA 2.1	Creation of the Evaluation Specification	Evaluation Specification	X	-
QA 1.3		→ Design Review	X	X
CD 3 Prototyping-Phase				
CD 3.1	Implementation of the Prototype	Description of the Prototype	X	-
QA 2.2	Evaluation	Evaluation Protocol	X	-
QA1.4		→ Prototyping-Review	X	-
QA1.5		→ Acceptance Review	X	X

Table 1: Documents, which have to be created

Legend:

X Document is mandatory
O Document is optional

The state of the thesis and the results have to be discussed with the tutor every 2 weeks.

The IAS guidelines have to be respected.

7 Additions

None.

8 Literature

[1] Pressman, R.S.: Software Engineering: A Practitioner's Approach. New York: MCGraw Hill Book Company, 1987

[2] Sommerville; I.: Software Engineering, Reading, MA.: Addison Wesley, 1989

[3] Jalote, P.: An Integrated Approach to Software Engineering, New York, Springer-Verlag, 1991

[4] Blanchard, B.: System Engineering and Analysis, New York, Prentice Hall, 1998

[5] ANSI/IEEE Std. 729-1983: IEEE Standard Glossary of Software Engineering Terminology. New York: IEEE Inc. 1983

[6] ActiveMedia home page: http://www.activmedia.com

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

System Requirements Specification

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2002

Document:	Version:	Author:	Date:	Status:
System Requirements Specification	1.0	Jost	15.04.03	accepted

File Name:	Pages:	Print Date:	Template:
sw-system-requirements-specification-v10.doc	9	28/05/03 17:32	sw-system-requirements-specification.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	03.12.02	in progress	Creation
1.0	Serrano López	Jo	20.01.03	in progress	
1.0	Serrano López	Jo	20.01.03	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

0 Table of Contents

1 General Purpose

This Master Thesis takes place in context of a global project consisting of a guided tour through the IAS institute. This tour is guided by the mobile robot RobIAS, which should conduct visitors in an automatic way through the institute, showing the laboratories and the projects which are being investigated in them. For that purpose, the mobile robot RobIAS will ask the visitors to follow it, and when they are in a room, it will explain what is being researched in that moment, accompanied with a multimedial presentation which includes a spoken description and, possibly, a demo activity.

The purpose of this Master Thesis is, within the scope of the referred global project, the development of a navigation and control software for the mobile robot RobIAS. This navigation software should provide an autonomous movement for the robot, including basic path planning activities, reactive obstacle avoidance and position approximation. The multimedial presentation and demo activities are not part of this master thesis, they are realized by another student.

1.1 Mandatory Criteria

Our Robot has to perform several actions: path planning, obstacle avoiding, repositioning,...We have therefore to consider carefully the synchronization of all this actions, so that there is no interferences between them.

The first thing we have to consider to is the dectection of wrong start conditions, and their correction if possible. Under wrong start condition we understand for example an incorrect initial position of the robot, low battery and sonar/laser not found. The localization of the robot can be made according to the comparation between the range devices readings and the map information. This topic will be discussed later, but we can already say, that due to certain limitations in the objectives of this Master Thesis, only little deviances respect to the desired initial position will be able to be corrected. If no correction to these wrong start conditions is possible, then we have to warn the users.

Once the robot is ready, we must move it from the actual position to the desired position. We are considering in this project a very simple path planning strategy, with a known environment: the IAS institute. Therefore we will not need a flexible and general path planning strategy, but a simple and effective way to determine which intermediate points have to be crossed, according to the start and end point and the given map from the institute.

When the path is already planned, we will very probably find obstacles in our originally planned trajectory. The safety in our project is important, and that means that neither the visitors nor the occasional people who walk in this trajectory of the robot should be injured. Therefore we have to think about the measures to adopt to avoid these collisions. In order to perform an adequate reaction against these dangerous situations, we will try first to find an alternative obstacle-free path. This operation will be made in a semireactive way, attending to the current range devices readings, because here only the map information doesn´t suffice. If no alternative is found, then we will have to stop the robot to avoid further consequences.

Related to this, we have to develop also the communication with the visitors during the navigation. This communication would include error messages like „Door blocked, please

open it“, „Way blocked, I can´t go on“ or similar, so that the visitors could help the robot to do its work properly. We don´t exclude other type of messages to be delivered like at regular intervals remainings, advices, etc...

When the robot reaches its target (and maybe in the intermediate points too), we have to check if the position is right. The odometry system provided by the robot works in general good, but there are unpredictable and unavoidable deviations which cause our robot to get lost. These deviations will grow progressively if we allow them to accumulate. Therefore, a position approximation and correction is definitively needed. That means, we have to localize our robot when it is already in the target position. We do this by means of a very simple localization method, consisting of reading the laser scanner measures and comparing them with the map. When properly done, we should be able to correct the position and the angle of the robot in a very easy way, and with less computer cost (and then the visitors don´t have to wait for too long). Like in the path planning, we don´t intend a complex and general localization method, but a simplified one for our particular situation, which suffices for our purposes.

As result of the previous aspects, we should have a robot able to move along the corridors for hours (supposed we had enough battery) without getting lost. The last step would be to communicate our software with the presentations software, which is completely independent from ours. The control should be transferred, and when the presentation ends, the control and the next target point should be transferred back to the robot.

1.2 Optional Criteria

The communication interface between the tour control and our project should be developed if possible. This communication implies a very simple message transfer protocol, to switch the control in a synchronized way between the navigation module and the presentation module, which determines the next target to reach. Although we will not make emphasis in this part of the project. It should be developed if possible, but it is not our main priority.

1.3 Limitations to the Criteria

1. The software to be developed has to be used locally in the robot´s embedded-PC. Any internet communications with other computers are intended.
2. The robot will work in a know environment, with available maps of it. It is not intended a complete reactive strategy, valid for every building structure.
3. The planning path strategy is not intented to be a general strategy for every kind of map, but simple and office-type environments. This strategy is mainly based in the interpolation of known intermediate points between the start and the end point.
4. The position approximation will be used in a particular way: we will use it under certain circumstances, with known variables, in known places of the map (for example, corners). This makes possible a simplified localization algorithm. It isn´t intended the creation of a whole localization system, which could work with any map and position.
5. The multimedial presentation of the current research topics in the institut is not in the scope of this Master Thesis. Here we provide exclusively an easy, safe and reliable navigation system which has to work under the concrete circumstances of this guided tour.

2 Operational Area

2.1 Areas of Application

The result of this project will be used to guide visitors through the IAS by the autonomously moving robot RobIAS. This project will be used in context of the research and educational purposes of the IAS. Further applications of this project are not contemplated at the moment.

2.2 User Group

Users of this project are mainly the foreign visitors of IAS. Students and employees are also users of this project.

2.3 Operating Conditions

The robot has to move in the know IAS environment, as part of a prepared tour guide. Occasional obstacles (mostly people who walk along the corridors, or even the visitors which follow the robot in an unproper way) are expected but not desired.

3 Environment

3.1 Software

1. The embedded PC in the robot uses Windows 98 as operating system.
2. The Saphira Client 8.1.5 provides us an easy and robust interface to the robot, as well as a simplified activities language, Colbert. We could even say, that provides a primitive (but efficient) operating system for the robot.
3. The Laser Scanner Module 8.1.5 for Saphira, which has to be installed independently.
4. Our navigation control software which runs on this embedded PC.

3.2 Hardware

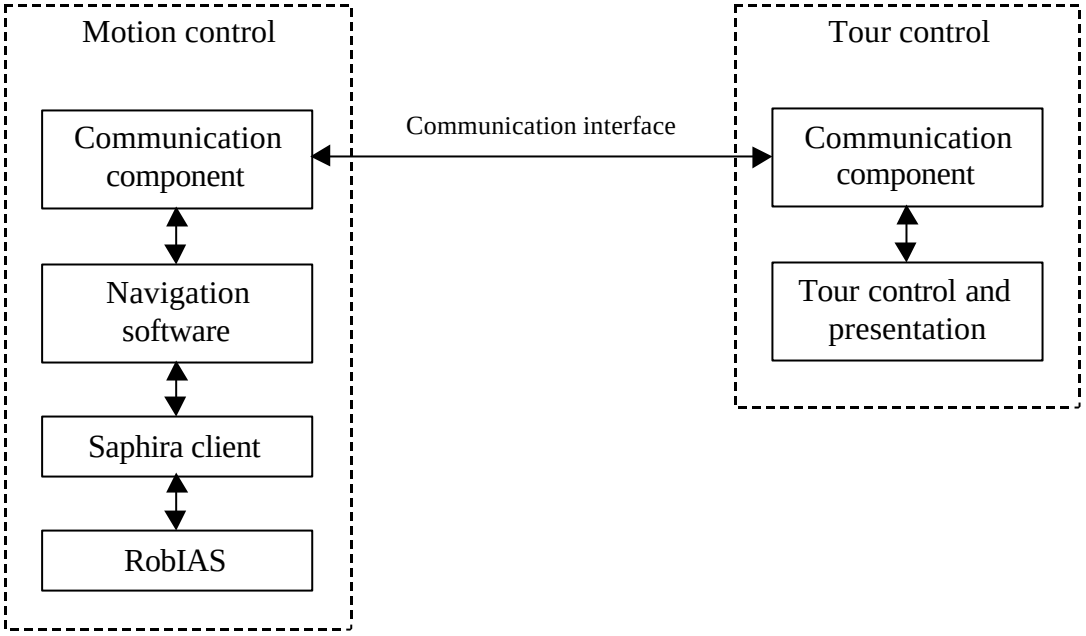
The robot RobIAS. This is a PeopleBot robot built by ActivMedia, and based on the Pioneer 2DX. The robot is provided with ten touch-sensitive bumpers, sixteen ultrasonic sensors (sonars) placed in two rows to avoid fast moving obstacles, and a high resolution laser range finder to give detailed information of the environment. Interaction with this robot is possible through a pair of speakers and a microphone. For the control a micro controller and a personal computer are mounted inside the robot

3.3 System Interfaces

We have to define an interface with the project which cares about the control of the guided tour, including multimedial presentations of the subjects of research in the institute, to allow the communication between both projects.

In addition we should define an interface between our software (programmed in Visual C++) and the robot. Fortunately, the Saphira client takes care about it, and does this work for us.

Here we show the whole diagram with the different interfaces in our system:



4 Functional Requirements

/F10/ A synchronization of the actions performed by the robot has to be considered: checking of initial conditions, the navigation and the localization, so that they do not interfere with each other.

/F11/ A synchronization with the upper layer (tour guide) is also needed, because navigation and control can not work simultaneously: the control part gives orders to the navigation part, which executes them, and then reports the end of its activities to the control part. (See /F62/)

/F20/ Wrong start conditions should be detected and corrected if possible. (see /F50/ for robot localization; only little deviances can be corrected). If no correction can be performed, then we have to warn the users, using the speakers integrated in the robot.

/F30/ The robot has to plan a path from its current position to a given destination (by user or by tour control) . [/LA20/]

/F31/ The institut map is previously obtained with measurements and then saved in a file. Therefore, the map is known, we don´t have to determine it in real time. [/LA20/]

/F32/ The path planning problem is reduced to find the proper intermediate points between the start and the end. These intermediate points will be also determined for each individual map, and saved. The path planning will search the best sequence of intermediate points that have to be reached in order to go to the target. [/LA20/]

/F40/ The robot has to be moved to the given destination in a safe way. This means that no people and the robot should suffer damage. [/LA30/]

/F41/ Therefore, the robot has to detect obstacles (for us an obstacle is everything which doesn´t appear in the saved map and interferes with the robot trajectory) and avoid collisions, by means of its sonars and laser scanner, but always trying to reach the given destination. [/LA30/]

/F42/ In case of collision, recover strategies have to be used. [/LA30/]

/F43/ “Closed doors detection” and “blocked way detection” algorithms have to be developed. We have to distinguish between mobile and static obstacles, that

correspond to people or furniture (or similar) respectively, and also closed doors, which can be found only in certain places of the map. [/LA30/]

/F50/ The position of the robot has to be checked and updated if needed. [/LA40/]

/F51/ Therefore, a comparison between the environment detected by range devices and the known map has to be performed. To do this we compare the laser readings and the map information in certain places of the map like corners and walls. [/LA40/]

/F60/ The control software has to provide an interface to interact to an upper layer control which e.g. will provide the destinations to go to (guided tour). [/LA50/]

/F61/ The communication has to be possible in both directions: from the upper layer to the navigation software, to determine the next action to be performed; and from the navigation software to the upper layer to inform about the state of the operation performed. [/LA50/]

/F62/ The synchronization between these two layers has to be considered. While one layer executes its actions, the other should wait. When the process ends, there must be an exchange of information, and according to this information, new actions have to be started by the other part.

5 Non-Functional Requirements

/N10/ Result of previous master thesis about the robot RobIAS as well as already developed modules for the Activmedia robots can be used. [/LA60/]

/N20/ The robot has to inform the users about certain situations like doors closed, way blocked, etc... with its speakers.

/N21/ The guide control layer has to be also informed about these and other situations, when they become unrecoverable or when they stay long time without solution.

/N30/ The alredy in IAS developed module to play sounds can be used.

6 Requirements for the User-Interface

None

7 Quality Requirements

Product Quality	very high	high	normal	not relevant
Functionality			X	
Correctness			X	
Safety	X			
Security				X
Reliability		X		
Maturity			X	
Fault Tolerance		X		
Recoverableness		X		
Usability			X	
Comprehensibility			X	
Ability to learn				X
Operability		X		
Efficiency			X	
Time Performance	X			
Consumption behavior				X
Alteration capability		X		
Analysis possibility			X	
Modifiableness		X		
Portability				x

8 Global Test Scenarios and Test Cases

The robot navigation software can be tested in two different ways:

- With the simulator provided by ActivMedia. This simulator is 100% compatible with the Saphira client, and allows us to model a very realistic behaviour of the robot, including sonars high speed and impreciseness, laser low speed and high precision and odometry errors. With this simulator it is not only possible to save a lot of time during the development (because we do not have to move phisically the robot), but also to avoid robot damage: we test on the robot only programs which run right on the simulator.
- We need also to check the real behaviour of the robot. Therefore we will need some initial tests to check this behaviour, so that the software development considers these aspects. And we will also need regular tests with the robot too, to check how the current software version works, when we have already tested it on the simulator. These tests will take place on the institute corridors, under special security measures to avoid personal injuries. We have to test the following:

    - ⇨ Start conditions check and correction.
    - ⇨ Navigation without obstacles, to test how good is the path planning.
    - ⇨ Inclusion of obstacles in the trajectory, to test the robot´s reaction and new found path.
    - ⇨ Blocking of the trajectory, to test if the robot reacts good against this situation.
    - ⇨ Doors and paths closing to test also the interactivity of the robot with the users.
    - ⇨ Localization of the robot: how much deviation there is and its correction.

9 Development Environment

9.1 Software

- Operating system: Microsoft Windows NT 4.0
- Microsoft Visual C++ 6.0 (this version is required by Saphira client)
- Saphira Client 8.1.5
- The Laser Scanner Module 8.1.5 for Saphira

9.2 Hardware

- Robot RobIAS
- One normal PC capable of running Microsoft Visual C++ 6.0

9.3 Orgware

- MS-Office 97
- MS-Project 4.1
- Visio 2000/5.0

9.4 Development Interfaces

None

10 Execution

The thesis has to be executed according to the "IAS Process Model" (Model for Software Development).

The state of the thesis and the results have to be discussed with the tutors in a period of 2 weeks.

The IAS guidelines have to be respected.

11 Additions

This Master Thesis was realized as part of a colaboration with the University of Sevilla. Its results will be presented also in Sevilla, where I will obtain my Title of Engineer of Telecommunications.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Project Plan

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Project Plan	1.1	Serrano López	15.04.03	accepted

Filename:	Pages:	Print Date:	Template:
project-plan-v11.doc	6	28/05/03 17:35	project-plan.dot

Document Version Management

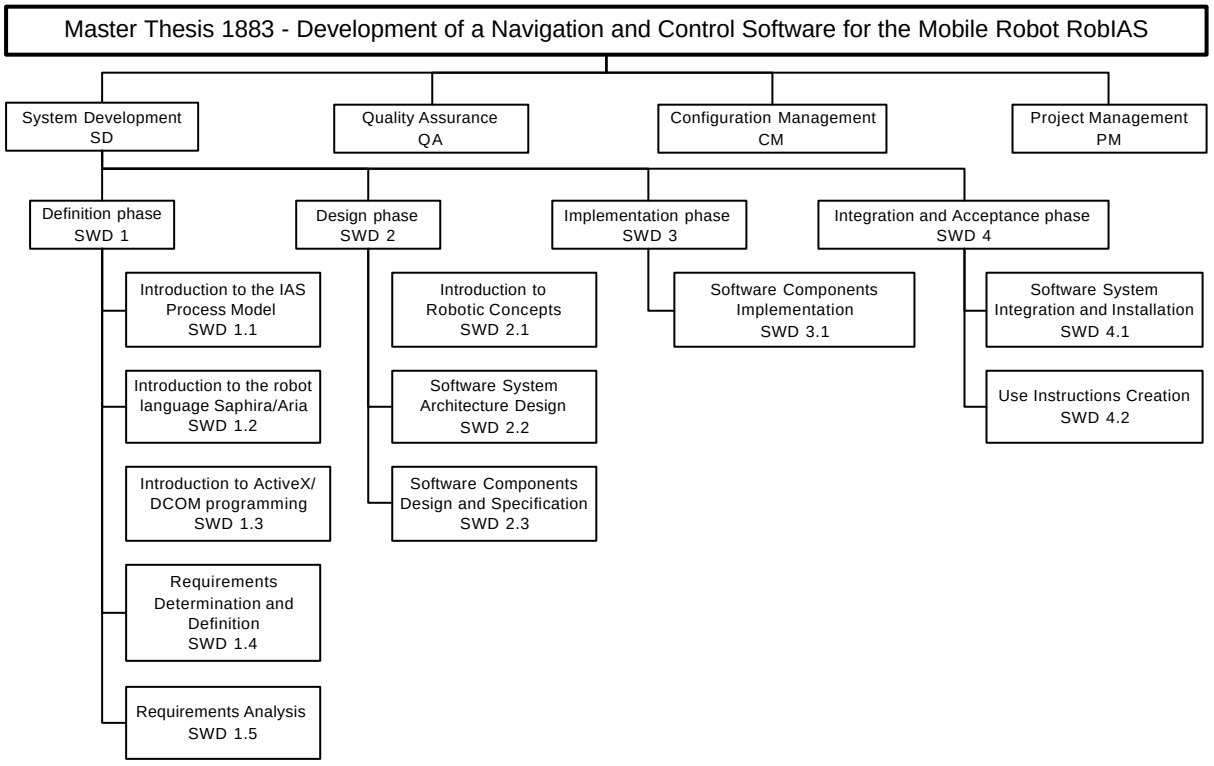
Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	03.12.02	in progress	Creation
1.0	Serrano López	Jo	20.01.03	submitted	
1.0	Serrano López	Jo	20.01.03	accepted	
1.1	Serrano López	Jo	09.04.03	in progress	Actual plan inserted
1.1	Serrano López	Jo	14.04.03	submitted	
1.1	Serrano López	Jo	15.04.03	accepted	

0 Table of Contents

1 Project Organisation

Name	Field of Duty	Address, Telephone, Email
Prof. Göhner (Gö)	Main Tutor	IAS, 7301, goehner@ias.uni-stuttgart.de
Jost	Tutor Quality Assurance (QA)	IAS, 7304, jost@ias.uni-stuttgart.de
Konnertz	Co-Tutor	IAS, 7319, konnertz@ias.uni-stuttgart.de
Serrano López, Javier	Project Management System Development Configuration Management	Stuttgart, 01627319521, herrserrano@yahoo.es
Dencovski, Kristian	Guided Tour Control	kristian@dencovski.de
System Administrator (SysAd)	Data Protection Computer Administrating	IAS, sysad@ias.uni-stuttgart.de

2 Project Structure Plan



3 Work Packages

3.1 Definition Phase (SWD 1)

The requirements for the system to be developed are defined here

3.1.1 Introduction to the IAS Process Model (SWD 1.1)

At first an introduction to the „IAS Process Model“ documentation is needed (in this case, the „Model for Software Development“), and a little introduction to the tools we well use to create our documents. We do this in this work package.

3.1.2 Introduction to the robot language Saphira/Aria (SWD 1.2)

The programming language Saphira/Aria, provided by the robot manufacturer Activmedia Robotics, is studied in this work package.

3.1.3 Requirements Determination and Definition (SWD 1.3)

The systems requirements document is created in this work package; this is done using the “User Requirements Specification” document. The requirements determined in the “User Requirements Specification” are refined, detailed and written in the “System Requirements Specification” document. This document contains functional, non-functional, user interface and quality requirements, global test scenarios and test cases as well as information about the development environment and the operational area.

3.1.4 Introduction to ActiveX/DCOM programming (SWD 1.4)

In order to make the interface between this project and the guided tour project, we will need an adequate knowledge of ActiveX/DCOM programming. We do this in this work package.

3.1.5 Requirements Analysis (SWD 1.5)

The software system model document is created in this work package, with the help of a formal method. This document should provide a better presentation of the requirements of the system.

3.2 Design Phase (SWD 2)

The system to be developed is designed here on the basis of the “System Requirements Specification” and the “Software System Model”. The whole system is divided here into its components. In the Software System Architecture this structure is hold. The component functionality and interfaces between these interfaces are here defined.

3.2.1 Introduction to Robotics Concepts (SWD 2.1)

In order to develop our own algorithms, we well need an introduction to the path planning, obstacle avoidance and position localization concepts. This is done here in this work package.

3.2.2 Software System Architecture Design (SWD 2.2)

The broad structure of the system to be developed in terms of software components is described here, and the software system in software components separated. The system functions are assgined in the software components and documented. Aditionally, the interfaces between the individual components are identified and defined.

3.2.3 Software Components Design and Specification (SWD 2.3)

Software components are described in detail and the scope of the functions and services is determined in this work package. Ein component can be for example a class or a method.

3.3 Implementation Phase (SWD 3)

All programming activities are done during this phase.
SD 3.1 Software Components Implementation - programs are written according to the system design, programs are documented with comments, test programs are created according to a test specification [8].

3.3.1 Software Components Implementation (SWD 3.1)

Programs are written according to the system design, programs are documented with comments, test programs are created according to a test specification.

3.4 Integration and Acceptance Phase (SWD 4)

3.4.1 Software System Integration and Installation (SWD 4.1)

The developed software components are integrated, installed and tested in this work package. The integrated components build the software system. The result of this work package is the executable system.

3.4.2 Use Instructions Creation (SWD 4.2)

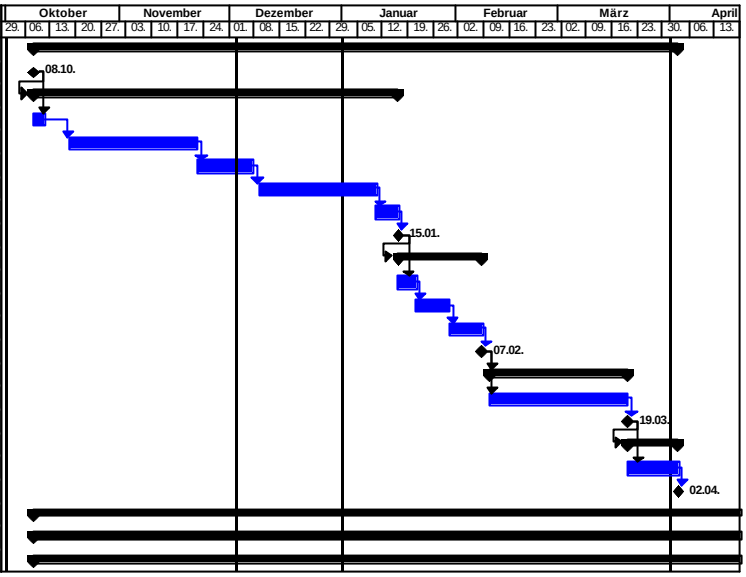
The use instructions (a user manual) are written after the system is operable. This contains a clear description of the installation steps and of the system functions from a user's point of view. Here is the information that the user needs the use the system.

4 Milestones

Date	Milestone	Documents, Products
08.10.2002 27.11.2002	Start	• Receipt of the User Requirem. Specification • Start conversation with Prof. Göhner
15.01.2003	Definition Phase	• Definition Review
07.02.2003	Design Phase	• Design Review
19.03.2003	Implementation Phase	• Implementation Review
02.04.2003	Acceptance	• Acceptance Review
25.04.2003	End of the Master Thesis	• All the documents of the project

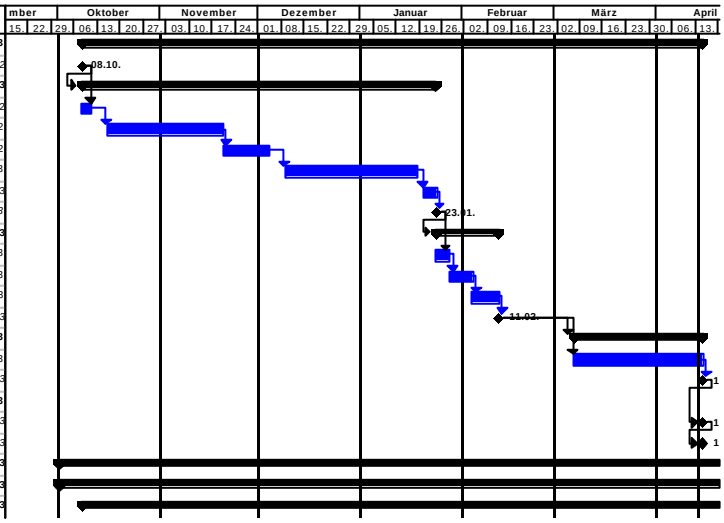
5 Bar Chart (planned)

Nr.	Vorgangsname	AP	Dauer	Anfang
1	**System Development**	**SD**	**117t**	**Die 08.10.02**
2	*Start*		*0t*	*Die 08.10.02*
3	**Definition Phase**	**SWD 1**	**62t**	**Die 08.10.02**
4	Introduction to the IAS Process Mode	SWD 1.1	3t	Die 08.10.02
5	Introduction to the robot language Sa	SWD 1.2	25t	Fre 18.10.02
6	Requirements Determination and Def	SWD 1.3	10t	Fre 22.11.02
7	Introduction to ActiveX/DCOM prograi	SWD 1.4	15t	Mon 09.12.02
8	Requirements Analysis	SWD 1.5	4t	Fre 10.01.03
9	*Definition Review*		*0t*	*Mit 15.01.03*
10	**Design Phase**	**SWD 2**	**17t**	**Don 16.01.03**
11	Introduction to Robotics Concepts	SWD 2.1	3t	Don 16.01.03
12	Software System Architecture Desig	SWD 2.2	7t	Die 21.01.03
13	Software Components Design and S	SWD 2.2	7t	Don 30.01.03
14	*Design Review*		*0t*	*Fre 07.02.03*
15	**Implementation Phase**	**SWD 3**	**28t**	**Mon 10.02.03**
16	Software Components Implementatio	SWD 3.1	28t	Mon 10.02.03
17	*Implementation Review*		*0t*	*Mit 19.03.03*
18	**Integration Phase**	**SWD 4**	**10t**	**Don 20.03.03**
19	Software System Integration and Insi	SWD 4.1	10t	Don 20.03.03
20	*Acceptance Review*		*0t*	*Mit 02.04.03*
21	**Quality Assurance**	**QA**	**134t**	**Die 08.10.02**
24	**Configuration Management**	**CM**	**134t**	**Die 08.10.02**
27	**Project Management**	**PM**	**134t**	**Die 08.10.02**



6 Bar Chart (actual)

Nr.	Vorgangsname	AP	Dauer	Anfang	Ende
1	**System Development**	**SD**	**123t**	**Die 08.10.02**	**Mon 14.04.03**
2	*Start*		*0t*	*Die 08.10.02*	*Die 08.10.02*
3	**Definition Phase**	**SWD 1**	**66t**	**Die 08.10.02**	**Don 23.01.03**
4	Introduction to the IAS Process Mode	SWD 1.1	3t	Die 08.10.02	Don 10.10.02
5	Introduction to the robot language Sa	SWD 1.2	25t	Mit 16.10.02	Die 19.11.02
6	Requirements Determination and Defi	SWD 1.3	10t	Mit 20.11.02	Die 03.12.02
7	Introduction to ActiveX/DCOM prograi	SWD 1.4	21t	Mon 09.12.02	Fre 17.01.03
8	Requirements Analysis	SWD 1.5	4t	Mon 20.01.03	Don 23.01.03
9	*Definition Review*		*0t*	*Don 23.01.03*	*Don 23.01.03*
10	**Design Phase**	**SWD 2**	**13t**	**Fre 24.01.03**	**Die 11.02.03**
11	Introduction to Robotics Concepts	SWD 2.1	2t	Fre 24.01.03	Mon 27.01.03
12	Software System Architecture Desig	SWD 2.2	5t	Die 28.01.03	Mon 03.02.03
13	Software Components Design and Sy	SWD 2.2	6t	Die 04.02.03	Die 11.02.03
14	*Design Review*		*0t*	*Die 11.02.03*	*Die 11.02.03*
15	**Implementation Phase**	**SWD 3**	**27t**	**Fre 07.03.03**	**Mon 14.04.03**
16	Software Components Implementatio	SWD 3.1	27t	Fre 07.03.03	Mon 14.04.03
17	*Implementation Review*		*0t*	*Mon 14.04.03*	*Mon 14.04.03*
18	**Integration Phase**	**SWD 4**	**0t**	**Mon 14.04.03**	**Mon 14.04.03**
19	*Software System Integration and Insi*	*SWD 4.1*	*0t*	*Mon 14.04.03*	*Mon 14.04.03*
20	*Acceptance Review*		*0t*	*Mon 14.04.03*	*Mon 14.04.03*
21	**Quality Assurance**	**QA**	**134t**	**Die 01.10.02**	**Die 22.04.03**
24	**Configuration Management**	**CM**	**134t**	**Die 01.10.02**	**Die 22.04.03**
27	**Project Management**	**PM**	**129t**	**Die 08.10.02**	**Die 22.04.03**



University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

System Model - Use Cases

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
System Model - Use Cases	1.0	Serrano López	15.04.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-system-model-use-cases-v10.doc	6	28/05/03 17:37	sw-system-model-use-cases.dot

Document Version Management

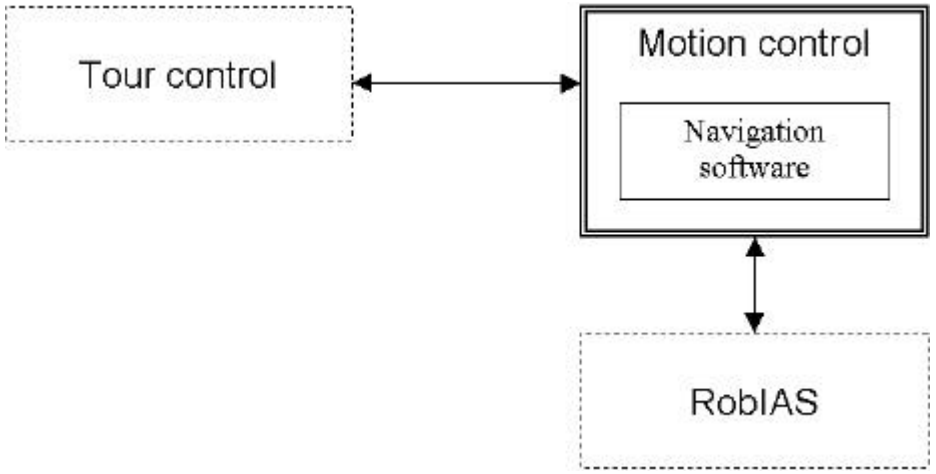
Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	20.01.03	in progress	Creation
1.0	Serrano López	Jo	21.01.03	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

0 Table of Contents

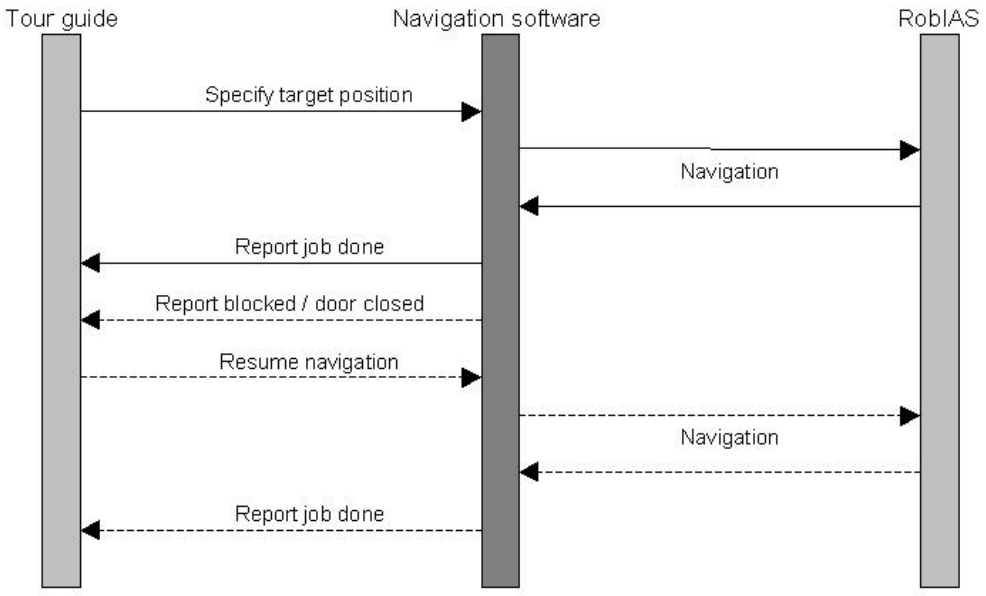
1 Introduction

In this project a guided tour through the IAS Institute must be performed. In this context, two external actors are present and interact with the previously named „Motion Control“, the software to be developed and purpose of my Master Thesis. These two actors are the „Tour Control“ and the robot „RobIAS“. The „Tour Control“ includes the HTML-oriented application which manages the guided tour (and that is made by another student).

In the following simplified diagram we represented these actors and their interaction with our software:



And in this other diagram below, how the whole system works



The „Tour Control“ sends the position to be reached (*Specify target position*) to the „Motion Control“, which navigates the robot „RobIAS“ (*Navigation*) in order to achieve this goal. If everything goes without problems, the target point will be reached and this will be reported back (*Report job done*) to the „Tour Control“. But it could happen too, that the robot gets blocked due to door which are closed (und should be open) or paths closed in such a way which make impossible the further navigation of the robot. In this case, a message (*Report*

blocked / door closed) is sent to the „Tour Control“ to inform about this circumstance. The „Tour Control“ has to perform the correspondent actions which allow at the end the robot to navigate again (open the door, clear the way from obstacles). When the „Tour Control“ estimates that everything can go on, a message (*Resume navigation*) is sent to the „Motion Control“ and then we do like in the case before.

2 Use Case Sheet for the Use Case “Specify target position”

Use Case:	Specify Target Position
Aim:	The transmission from the Tour Control to the Motion Control (Software) of the new place in the institute which has to be reached.
External Actors:	Tour Control
Conditions:	• The robot must be ready. That means, that the robot and its software must be started before we give the first position order. • The robot must be also in a correct situation, not blocked.
Description:	After the corresponding presentations and demos are made in the Tour Control part, an order must be given to Motion Control, specifying which is the next position to reach inside the institute.
Alternative Sequences:	None

3 Use Case Sheet for the Use Case “Navigation”

Use Case:	Navigation
Aim:	The navigation of the robot RobIAS to reach the specified position in the institute, including also the path planning activities, reactive obstacle avoidance and position approximation.
External Actors:	RobIAS
Conditions:	• The robot must be ready. That means, that the robot and its software must be started before the order to navigate is received. • The target to reach must be previously specified, we have to wait for this order.
Description:	When the target is known, the robot must go to this target in an autonomous way. The first thing will be planning the path. Then we have to move the robot through this planned path, and avoiding all the possible obstacles which could appear in this path. In addition, regularly a position approximation routine must be executed to mantain the robot well localized.
Alternative Sequences:	a) The robot can be switched off. b) Invalid target position specified.

4 Use Case Sheet for the Use Case "Report blocked / door closed"

Use Case:	Report blocked / door closed
Aim:	The transmission from the Motion Control to the Tour Control of the undesired reached state of the robot, which can´t go to the specified position due to an unvoidable obstacle in the way or a closed door.
External Actors:	Tour Control
Conditions:	The robot must be navigating in order to reach an eventual blocked state. There must be also doors closed or unavoidable obstacles in the way.
Description:	When the robot navigates and has to go through a door, and this door is closed, then we will have this use case. Other possible source of problems is the presence of many obstacles or a configuration of them which frustrates the execution of every avoidance obstacle strategy which we have. We have to transmit not only the information that the robot is blocked, but also why got the robot blocked (so far as possible)
Alternative Sequences:	a) The robot can be switched off b) Normal navigation without blocking.

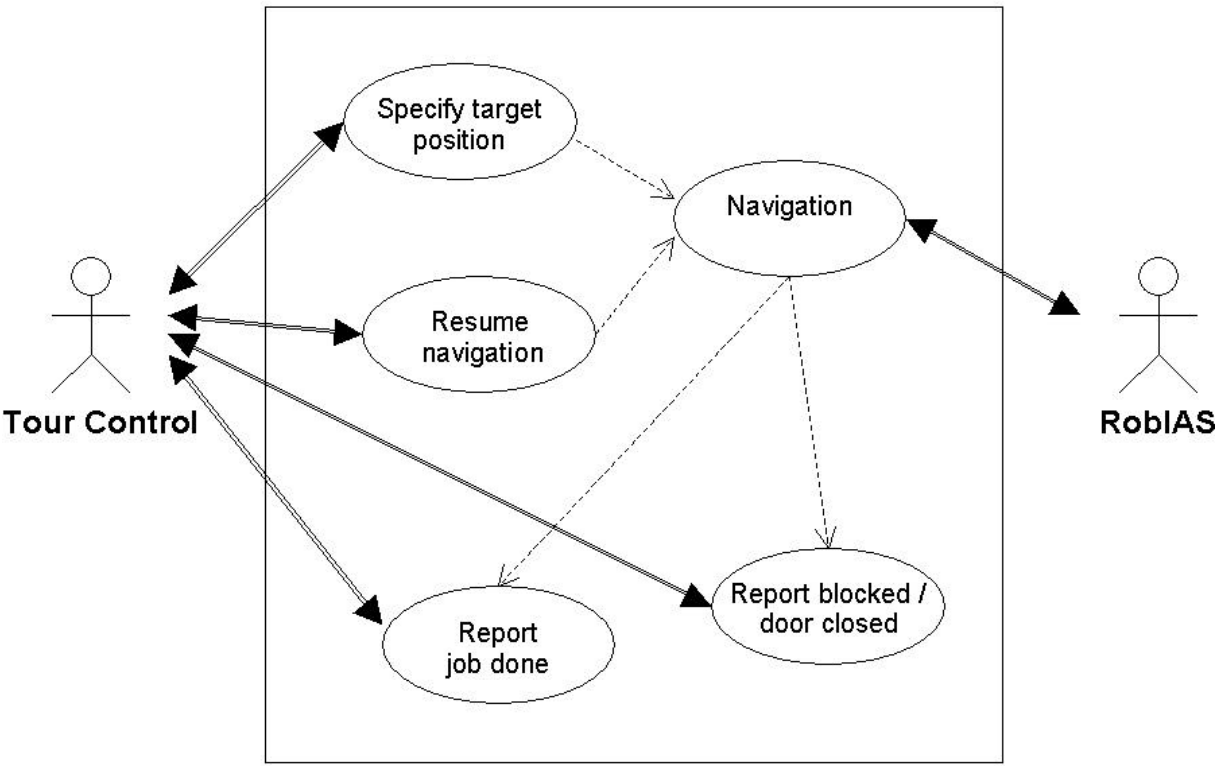
5 Use Case Sheet for the Use Case "Report job done"

Use Case:	Report job done
Aim:	The transmission from the Motion Control to the Tour Control of the information, that the robot has successfully reached the desired position and the multimedial presentation can go on.
External Actors:	Tour Control
Conditions:	• The robot had to be navigating before in order to reach this state now. • No blockings are allowed.
Description:	When the robot achieves to avoid all the present obstacles, to be always correctly located and to reach the target, then this success must be notified to the Tour Control.
Alternative Sequences:	Navigation blocked: target not reached.

6 Use Case Sheet for the Use Case "Resume navigation"

Use Case:	Resume navigation
Aim:	The transmission from the Tour Control to the Motion Control of the information, that the circumstances which origined the blocking situation have been removed and the robot can continue with its normal procedure to the desired target.
External Actors:	Tour Control
Conditions:	• The robot had been blocked during the navigation, and this was reported. • The circumstances which origined this blocking have been already removed (that means, the door must be opened or the obstacles away). • The software of the robot is ready to receive the order.
Description:	The Tour Control has to perform the adequate actions which allow that the blocking situation disappears. When this ist done, the Tour Control must inform the Motion Control that the way to the target is now free and that it can go on to the prespecified target.
Alternative Sequences:	No removal of the blocking situation. The robot stays blocked and the target is not reached.

7 Use Case Diagram



University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Definition Review Protocol

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Definition Review Protocol	1.0	Serrano López	23.01.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-definition-review-v10.doc	7	28/05/03 17:40	sw-definition-review.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	21.01.03	in progress	Creation
1.0	Serrano López	Jo	23.01.03	submitted	
1.0	Serrano López	Jo	23.01.03	accepted	

Place, Date:	Stuttgart, 21.01.2003	
Participants:	Javier Serrano López, Jost	

CHECKLIST Project Plan

Checkpoints	**Actions/ Comments**
1) Are all (also external) persons listed?	yes
2) Is the partitioning of the work packages practical?	yes
3) Are project structure plan and bar chart consistent?	yes
4) Is the time schedule realistic for a max. operating time of 5?	yes
5) Are interruptions (e.g. vacation) considered?	yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted	
				yes	no
11.12.02	1.0 / 20.01.03	Jo		X	

Comments:

Extra time needed for the exams realized in Spain in December.

CHECKLIST System Requirements Specification

Checkpoints	**Actions/ Comments**
1) Is the project practically defined?	yes
2) Are the requirements complete?	yes
3) Are the requirements understandable and consistent?	yes
4) Are all requirements from the user specification referenced and consistent?	yes
5) Are decisions for the implementation already anticipated by detailed requirements?	no
6) Are the quality requirements defined in the sense of the customers/orderers?	yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted	
				yes	no
11.12.02	1.0 / 20.1.03	Jo		X	

Comments:

CHECKLIST System Model

Checkpoints	**Actions/ Comments**
1) Is the model complete and consistent?	yes
2) Are the substantial use cases considered?	yes
3) Is the analysis of the requirements specified in the system requirements specification sufficiently executed?	yes
4) Are alternatives sufficiently examined?	yes
5) Are different sub-functionalities of the software described sufficiently and understandably?	yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
21.01.03	1.0 / 21.01.03	Jo		X	

Comments:

CHECKLIST GUI Concept

Checkpoints	**Actions/ Comments**
1) Are all use cases described in the system model considered?	
2) Are the suggested control concepts conclusive and user friendly?	
3) Are the individual control elements (edit-, list and combo boxes etc.) practically selected?	
4) Does the organisation of the user interface correspond to the requirements of the customers/orderers?	

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no

Comments:

CHECKLIST Project Progress	
Checkpoints	**Actions/ Comments**
1) Is the planned timetable of the project (Bar Chart) kept?	no, 1 week behind
2) Is the inter-office slip up to date?	yes

Acceptance		
Date of Inspection	Inspector	
21.01.03	Jo	

Comments:

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

System Architecture

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
System Architecture	1.0	Serrano López	05.02.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-system-architecture-v10.doc	13	28/05/03 17:41	sw-system-

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	30.01.03	in progress	Creation
1.0	Serrano López	Jo	04.02.03	submitted	
1.0	Serrano López	Jo	05.02.03	accepted	

0 Table of Contents

1 Surrounding and Boundary Conditions

The purpose of this Master Thesis is the development of a navigation software for the mobile robot RobIAS, to be used for a guided tour through the IAS (Institut für Automatisierungs- und Softwaretechnik).

1.1 Hardware

Robias is a PeopleBot based on the Pioneer 2DX, built by ActivMedia. To interact with the environment, the robot is provided with:

- ⇨ 10 touch-sensitiv bumpers, which detect collisions
- ⇨ 16 ultrasonic sensors, in two semirings of 8 sensors each, which provide fast but unprecise information about the obstacles
- ⇨ a laser range finder (slow, but very precise and with large range).
- ⇨ a compass to measure the orientation of the robot.
- ⇨ acoustic communication is possible through a pair of spekers mounted on the top, and a microphone.

In the *Figure 1* a Pioneer 2-DXE is represented, very similar to the mobile robot RobIAS (RobIAS doesn´t have a camera, and has a laser range finder, not represented in the figure).

Figure 1: Pioneer 2-DXE (similar to RobIAS)

The roboter is also equipped with:
- ⇨ an embedded PC (intel 82371AB 266 MHz Processor, 128 MBytes RAM) with Windows 98 as operating system
- ⇨ a Wireless LAN for the connection to the internet (and the IAS LAN)
- ⇨ a Siemens 88C166-based microcontroller which is used by Saphira to calculate the actual position of the robot and other functions.

1.2 Software

Our software will be executed on the embedded PC of RobIAS. This PC has Windows 98 as operating system, and has a relatively slow performance. Therefore CPU usage has always to be kept in mind.
The navigation software runs on the Saphira 8.1.11 client. The Saphira client will work as a little operating system, offering a transparent communication between our program and the microcontroller, and simplifying enormously the communication with the mobile robot.

2 Fundamental Design Decisions

As we already know, the mobile robot RobIAS has to navigate through the IAS Institute. This *navigation* (presented as Use Case Navigation in the System Model) implies some activities: a sequence of intermediate points to be crossed before we reach the final target has to be calculated. Once this sequence is determined, the robot must follow it, and guided by its sensors, find a path to reach these points and at the same time avoid the obstacles which could appear on the way. If an intermediate point can not be reached, an alternative one has to be found, and if no alternative choice is available, the robot gets blocked (and this situation is reported). This all will not be possible without a reliable and easily implementable localization method which has to be regularly executed in order to determine and correct the actual position of the robot. All these described functions will correspondend to software components.

Furthermore, the communication with the tour guide is another important issue which has to be considered, because will allow to *specify the target*, *report the job done*, *report blocking situations* and *resume the navigation* after these situations (these Use Cases were also specified in the System Model). From here we have another software component.

All the software components which realize these functionalities, are contained in DLL programs. These DLLs are called from the Saphira environment, by means of one other several colbert programs (where *Colbert* is the an easy programming language integrated in Saphira). These Colbert programs will correspond to the last software component: „Navigation control“.
Therefore, we will distinguish the following software components:

- **Navigation control** component
- **Sequence of intermediate points planning** component
- **Robot localization and position correction** component
- **Obstacle-free path calculating** component
- **Alternative intermediate point finding** component
- **Tour guide-Communication** component

3 Diagram of the Software System Architecture

3.1 System Architecture overview

In the following diagram, an overview of the system architecture is represented. The components before listed are also included in this diagram:

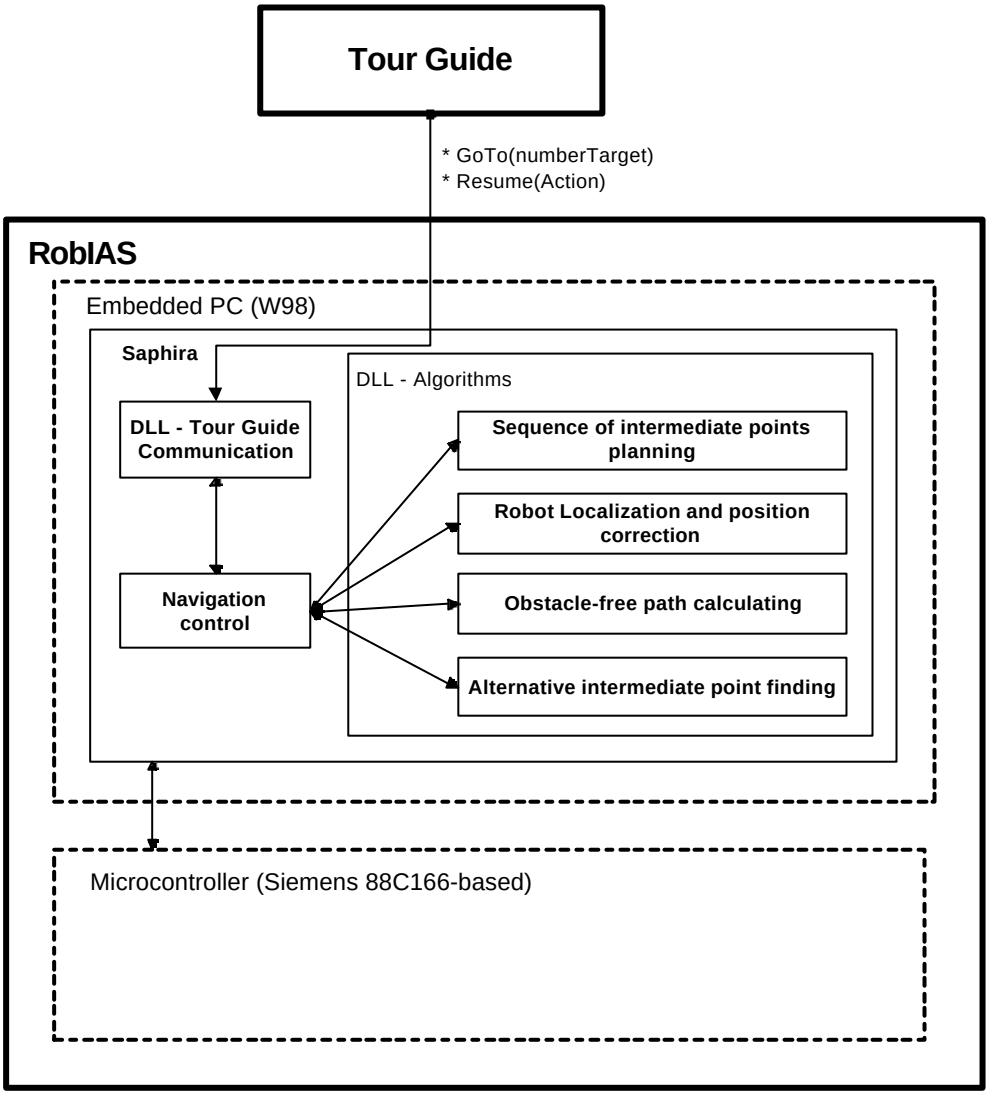


Figure 2: System Architecture overview

The „Navigation control“ correspond to colbert programs specific to Saphira, programmed in a very simple language named „Colbert“. This language allows the execution of concurrent tasks and other simple features in a very easy way. Our components, encapsuled in DLL-programs, will define „Actions“ which can be used by the navigation control in order to perform the desired behaviours of the robot.

3.2 State diagram

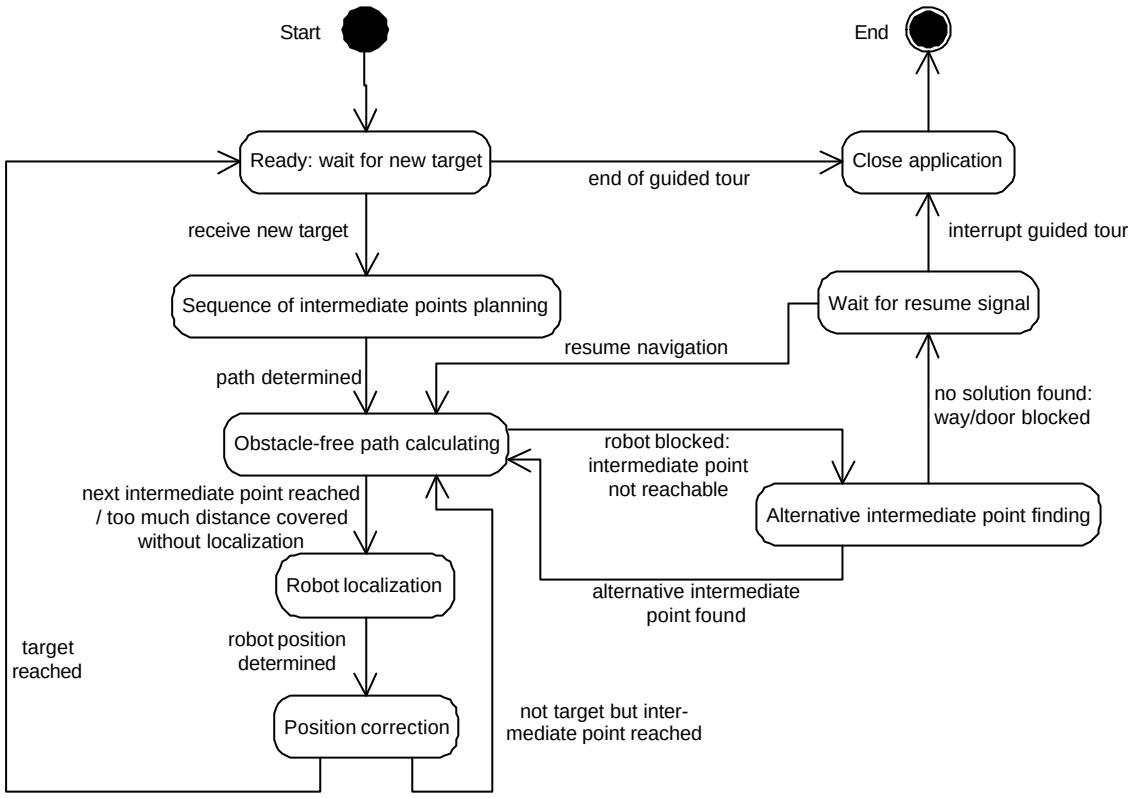


Figure 3: State Diagram

The correspondences between the states of this diagram and the software components are in the following table:

States of the diagram	Software components
Ready: wait for new target	Tour guide-Communication
Wait for resume signal	
Sequence of intermediate points planning	Sequence of intermediate points planning
Obstacle-free path calculating	Obstacle-free path calculating
Robot localization	Robot localization and position correction
Position correction	
Alternative intermediate point finding	Alternative intermediate point finding

The „Navigation control“ component does not appear in this table, because it controls the whole state diagram. The different components are invoked from this component.

4 Software Components

4.1 Navigation control

The „Navigation control“ component is the component which invokes the other ones. It is a program written in the programming language *Colbert*. In words of Activmedia (the robot manufacturer), „*Colbert is a programming language for robots. Its primary purpose is an executive in a more complex 3-tier robot control architecture called Saphira/Aria (see Figure 4). In particular, Colbert can issue motion commands, can sequence robot actions, and can execute complex hierarchical control strategies using Saphira´s repertoire of sensing and control routines*“

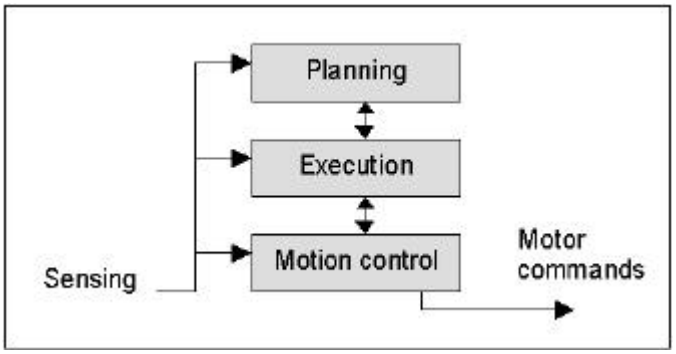


Figure 4: State Diagram

Colbert is an executable language, and the users can issue Colbert commands. These commands can load files, define robot activities, and start and control robot programs. And that is exactly what we do here: we write the software components (sequence of intermediate points planning, robot localization and position correction, etc.) in C++ and then write companion Colbert activities that interface to these programs. Therefore, these Colbert activities, which implement the *Navigation control*, define a software component in our system architecture.

4.2 Sequence of intermediate points planning

Once the target to be reached is determined by the tour guide, we have to find an effective, short and secure way to move the robot from the start position to the end position.
The routine to do this, as said in the System Requirements Specification, will not be a generic path planning method, but a simple one based on the fact, that the map of the institute is completely known. Furthermore, this path should not be complicated at all, due to the office-structure of the institute, with straight and wide corridors, 90-grades corners, and square rooms. Therefore, the function of the „sequence of intermediate points planning“ component will not be finding a complicated completely predetermined path, but a sequence of intermediate points to be consecutively reached attending to criteries like distance, easiness and security (for example, if we have to access a room with several doors, it will be preferred the solution which doesn´t imply navigation amongst chairs and similar objects which position is in some way unpredictable). The distance between these intermediate points will be determined by the position estimation errors (see next component, „Robot localization and position correction“).

4.3 Robot localization and position correction

During the navigation, the position determination of the robot is affected by little deviances in the integration of the wheels measurements. This can be caused by numerical errors, skid of wheels, incorrect pressure of wheels, etc. These deviances are little at the beginning, but their effects accumulate and after some meters, the total deviance suffered by the robot can be critical for the navigation. Therefore, a localization routine has to be executed regularly to determine and correct the robot current position.
This routine, as said in the System Requirements Specification, will not be a generic localization method, but a simple one based on known and easily recognizable properties of the environment, like corners and walls. With a comparison between the information provided by the laser in these places and the known map of the institute, and using the integrated compass of RobIAS, we should be able to calculate the real actual position of the robot with enough precision to suffice our purpose of mantaining the robot always under certain limits of position error.
This software component will be invoked by the „Navigation control“ component, but the decision about when has to be this localization performed, is taken by the „Obstacle-free path calculating“ component. This component will end its activity, telling the „Navigation control“ component that a localization is needed.
This robot localization should be performed when we have reached an intermediate point, so that the navigation to the next one can be done in a proper way. The distance of these intermediate points, other in other words, the frequency of use of the localization method, will be idetermined experimentally attending to the real positioning errors experimented by the mobile robot RobIAS. Other factors should be also considered: the positioning errors will be „proportional“ to the distance covered by the robot; therefore, we may need to localizate the robot again if a persistent presence of obstacles makes this covered distance much bigger as expected.
Finally, after the robot localization routine is executed, the position of the robot has to be corrected (or eventually the coordinates of the robot modified), moving it to the expected position to be reached.

4.4 Obstacle-free path calculating

After the „Sequence of intermediate points planning“ component execution, we have a list of intermediate reference points to be reached in a consecutive way. If no obstacles are present on the way, to calculate the path will be so easy as connecting these points. But if some unexpected obstacle (furniture, people, etc.) is detected by the sonar and/or the laser, an alternative reactive (not previously planned) path has to be calculated and followed, according always to the next intermediate point that has to be reached, and the current measurements of the sensors. This component will be then cyclic.
For this purpose, the Gradient Navigation Module (which is included in the Saphira distribution that we use) will be very helpful for us, because calculates a minimal distance/time path to the desired target point attending to the measurements of the laser/sonar, current speed and position of the robot and the known map of the institute. This will complemented with our own orders, to adapt this generic algorithm to our specific needs.
This component will be also responsible of the detection of possible error conditions like „way blocked“ (which will appear when no free way to the next intermediate point is available), „door closed“ (when a door is closed where is supposed to be open) and „collision“ (when the robots collides with an obstacle). This component will not find a solution for these errors (that is the task of the „alternative intermediate point finding“), only finds possible error situations.

4.5 Alternative intermediate point finding

If during the navigation („obstacle-free path calculating“ component) the robots gets blocked, or an intermediate point can not be reached, then the „Navigation control“ component executes the „Alternative intermediate point finding“ component in order to find a solution for this situation. Here the available information will be analized, and then two things can happen:

- The blocking situation can not be resolved. This circumstance must be then reported („Tour guide-Communication“ component).
- An alternative intermediate point was found. We undo the blocking situation and go on with the navigation („Obstacle-free path calculating“ component) towards this new intermediate point.

4.6 Tour guide-Communication

The purpose of this component is to provide the guide tour a set of methods which will be used to interact with our program. By means of this component:

➔ the next target to be reached can be communicated to our navigation software (that means, the tour guide sets the new target to be reached),
➔ the navigation can be resumed by the tour guide, after it was stopped due to a “way blocked” or “door closed” situation.
➔ the navigation success or a blocking situation can be reported to the tour guide.

This component defines the interface for the tour guide, and takes care, in a simple way, that Saphira can know this information provided by the tour guide. With other words, Saphira can not directly access this component, but the component should provide mechanisms which allow Saphira to access this information sent by the tour guide. In an earlier version, this will be made by writing/reading a file. If possible, a more reliable and fast procedure will be developed (this is, as said in the System Requirements Definitions document, an optional criteria, and will be fully implemented only if possible).

5 Interface Definitions

5.1 Interface <Tour Guide> - <Tour guide-Communication>

5.1.1 <Tour Guide> => <Tour guide-Communication>

For the Tour Guide, the following methods are made available:

- *GoTo(int numberOfTarget)*
  With this method, the Tour Guide sets the target to be reached by the mobile robot RobIAS. The number associated to each target must be known by both, the tour guide and the tour guide-communication component of our project.

- *Resume(bool Action)*
  This method allows the Tour Guide to resume the execution of the navigation process after a previous blocking situation caused by a way blocked or a door closed. The parameter defines the action to be realized:
  - *Action = true* : continue with the navigation
  - *Action = false* : end of the guided tour (the target is not reached)

I would like to remark here the importance of the coordination of boths projects, ours and the Tour Guide. The meaning of all the parameters must be perfectly defined, to interpret the number of the stations and the actions to be performed in the same way. Therefore, in order to separate the navigation software from the election of targets to be reached (position, number and name), the list of possible targets and their coordinates will be stored in a text file, easily editable, and the navigation software will take the information from this file. If a new target point is added, removed or changed, the navigation software must not be changed, only this text file with the essential information.
It is also clear that the information contained in this file must exactly coincide with the information used by the tour guide to set the targets to be reached.

5.1.2 <Tour guide-Communication> => <Tour Guide>

The Tour Guide is programmed in such a way that can´t define an interface which could be invoked by the Tour guide-Communication component. Therefore, in order to allow the synchronization of Tour Guide and navigation software, the execution of the methods *GoTo* and *Resume* ends when the target is reached or the robot blocked, allowing then the processing of the rest of the Tour Guide sentences.

5.2 Interface <Navigation control> – <Tour guide-Communication>

To access the information delivered by the Tour Guide about the target to be reached and the navigation resuming, a very simple method will be used at beginning (only if possible, a better one will be developed; this is an optional criteria in our System Requirements Definition).The basic idea is to write the information received to a file, which will be known by the navigation control. The navigation control (probably by means of some functions written in C++ and contained in the DLL) should be then access this information from the file.
The problems of this method are obvious:

- Possible error when both processes read and write are performed at the same time.
- Speed problems, due to the accesses to the disk.
- Potential cache problems, to be studied.

These potential problems are known and their effects will be shortly studied, but to solve them is not our priority, and therefore, it will be made only if possible. Nevertheless, this option, although it is not optimal, will be considered as a valid solution because:

- The size of the file to be written/read is very small, just some bytes, and that should not imply a lot of time for the computer.
- The writing process happens very rarely, not more often than once per minute, and therefore the probability of „collision“ can be ignored if we take care that the reading process does not take place very often. So, we have to read the file to check if some data was written for example each 5 seconds.

5.3 Interface <Navigation control> – <Sequence of intermediate points planning>

Once the target is defined, the navigation control program has to inkove the software component „Sequence of intermediate points planning“ in order to obtain the sequence of points which have to be used as intermediate points to reach the final target. A possible invoking declaration would have more or less this form:

- **int SequenceOfIntermediatePointsPlanning(ArPose currentRobotPosition,*
  int numberOfTarget)

(where *ArPose* is an Aria class which stores position and angle information of the robot).

That could be the first option, supposing the number of all the predefined positions known everywhere in our software. Therefore, the return value would be a pointer to a matrix containing the numbers of the intermediate points. Other possibility could be:

- **ArPose SequenceOfIntermediatePointsPlanning (ArPose currentRobotPosition,*
  ArPose targetPosition)

But we will use the first one, because of this planning algorithm that we will use, consisting only of the determination of the best (already known) intermediate points.

5.4 Interface <Navigation control> – <Robot localization and position correction>

The function of the „Robot localization and position correction“ software component is to determine in a precise way the actual position of the robot, known the map and the actual measured position, and the robot position correction. Therefore, the declaration to this component will have more or less this aspect:

- *bool LocalizeAndRepositionRobot(ArPose currentRobotPosition)*

The return value should inform us about the success of the operation:
- *return value = false* : the localization of the robot was not possible. We ignore where is the robot exactly positioned.
- *return value = true* : the localization and position correction of the robot was performed successfully. The navigation can go on.

5.5 Interface <Navigation control> – <Obstacle-free path calculating>

The software component „Obstacle-free path calculating“ calculates in a reactive way the path to be followed by the robot attending to the next intermediate point to be reached and the current sensor information. Therefore, this component must know to which intermediate point are we trying to go. A possible definition of the call to this component would be:

- *bool ObstacleFreePathCalculating(ArPose nextIntermediatePoint)*

Note that here we are not interested in the number of the intermediate point, but in its coordinates. The return value says if there was a blocking situation or not:
- *return value = false* : there is a possible blocking situation which must be solved (by the „Alternative intermediate point finding“ component).
- *return value = true* : intermediate point successfully reached.

5.6 Interface <Navigation control> – <Alternative intermediate point finding>

The aim of the „Alternative intermediate point finding“ component is to solve a blocking situation generated in the „Obstacle-free path calculating“ component. We don´t define an interface between these components, because they are both independent and used by the „Navigation control“ component.

In order to perform its functionality, the „Alternative intermediate point finding“ software component will need to know the current position of the robot, the intermediate point where the blocking happens, and also some global information: the whole sequence of intermediate points. Therefore a possible definition of the call to this component would be:

- *int AlternativeIntermediatePointFinding(ArPose robotCurrentPosition,*
  int nextIntermediatePoint,
  **int SequenceOfIntermediatePoints)*

Note that, again, we give the number of the intermediate points instead of their coordinates, because, as we said before, the meaning of these numbers (that means, its translation into coordinates) will be known in the whole navigation software.
The return value will be the alternative intermediate point that we will use instead of the last blocked one. If no alternative could be found and the blocking situation can not be repared here, an specific value (i.e, zero) will be returned. So defined, no distinction between a way blocked and door closed if possible. To solve this, we could use not one specific value, but 2 or more, if needed. The thing here is to consider if the Tour Guide will be able to process this information. If not, more than one specific value will have no sense.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Components Specification

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Components Specification	1.0	Serrano López	10.02.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-components-specification-v10.doc	16	28/05/03 17:43	sw-components-specification.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	10.02.03	in progress	Creation
1.0	Serrano López	Jo	10.02.03	submitted	
1.0	Serrano López	Jo	10.02.03	accepted	

0 Table of Contents

1 Component Navigation control

The main function of this component is the implementation of the state diagram presented in the System Architecture document. This navigation control component will be mainly written in the Saphira specific programming language: Colbert (also present in the System Architecture). Nevertheless, some parts will have to be developed with C++, to implement certain functions of this component.
Since this component is the central one, the other five components will be invoked by this one. To do this, the colbert language offers us some possibilities:

- We can use ***Actions****:* This is the name for synchronous tasks which are executed every 100 ms. These tasks are programmed as C++ classes and have a constructor, an *invoke* function which defines an interface to colbert and a *fire* function which is cyclical executed. The function *deactivate()* allows us to stop this task and the *draw* method makes possible using the Saphira GUI for graphical representations. These actions will be contained in DLL files, and interfaced when the file is loaded (command *loadlib* in Colbert). An example of *Action* in our software will be the „Obstacle-free path calculating“ component, because it must be cyclical executed.
- We can also use ***asynchronous tasks***. These tasks will run concurrently with the others (synchronous or asynchronous), and the task management is made by Saphira, we don´t have to take much care about it. These tasks are also contained in DLL files and programmed in C++.
- Other possibility is to implement **usual C++ functions**, which can be normally invoked within a colbert program. These functions are compiled into a DLL file, which must contain a function named *sfLoadInit* where they are declared (this function *sfLoadInit* is also needed for synchronous and asynchronous tasks, because they are declared there). An example of this type of functions is the „Sequence of intermediate points calculating“ component. It has to be invoked asynchronously but not concurrently (therefore we do not implement it with a task).
- Colbert offers ***activities***: they are very similar to the synchronous tasks, but they are not programmed in C++, but in Colbert language. There are easy commands for this activities: *start, resume, stop,* etc., which are the same as for the actions. Actually, an *Action* is an user-defined *activity*.
- Finally, colbert offers a simple but powerful **robot-oriented programming language**, very similar to C++, which will be used to implement the state machine.

The „Navigation control“ component may use some of the presented possibilities (but not necessary all of them), in order to achieve its goal of implementing the state machine and coordinating the other software components.

2 Component Sequence of intermediate points planning

Once the target to be reached is determined by the tour guide, we have to find an effective, short and secure way to move the robot from the start position to the end position. As said in some previous documents (i.e., System Architecture), we will use a very simple method consisting in the determination of a sequence of intermediate points between the initial and the final position. These points will be pre-determined and the algorithm will only choose a sequence amongst them. To understand this concept, we will explain it with figure 1:

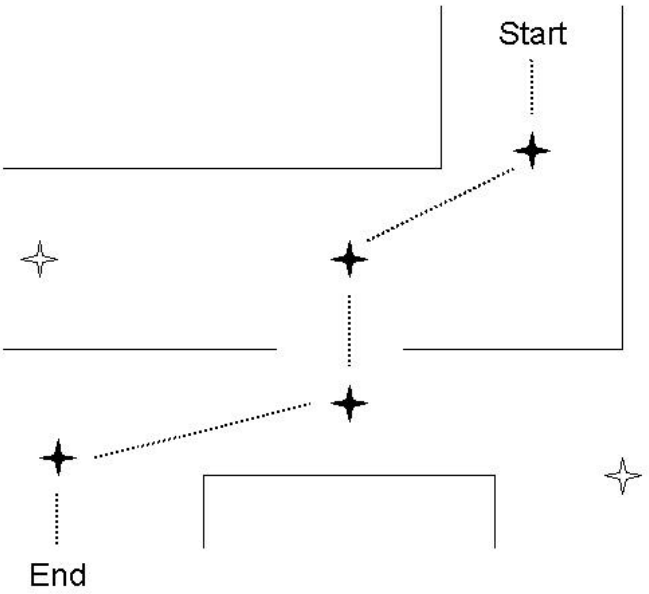


Figure 1: Intermediate points

Each star here represents a pre-determined possible intermediate point. The idea is to choose amongst all these available intermediate points, an appropiate sequence in order to go from "Start" to "End". The selected points are represented in the figure with solid stars, and the path determined by these points is represented with a dashed line. Note that not all the sequences are valid, because they have no physical meaning. In the example in figure 2:

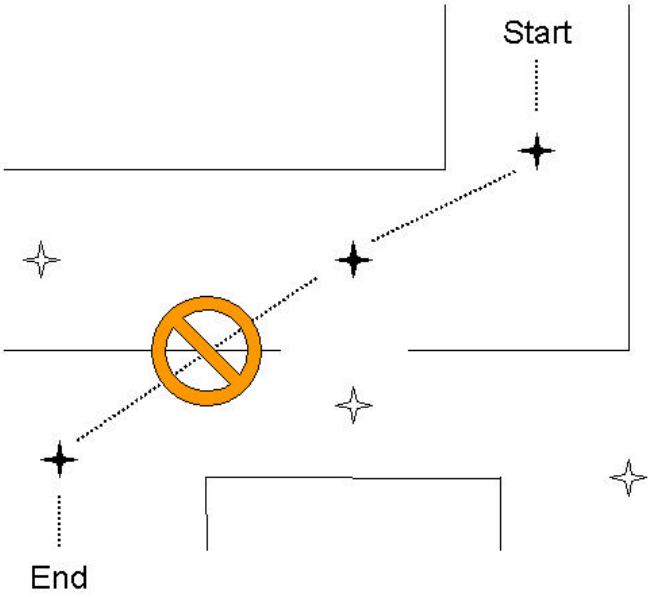


Figure 2: Not valid sequence

Since the path derived from this sequence of points trys to go through a wall, this solution is not good at all. The robot could go in the wrong direction and would never reach its desired target in a proper way or may need too much time for this purpose. We must remember that we are implementing the navigation for a robot which must guide people, and the people should not follow a robot which does not know where it goes!

The solution to this problem is to store not only the available intermediate points, but also the possible transitions between them. Therefore, we need represent this information as a not-oriented graph, as shown in fig. 3:

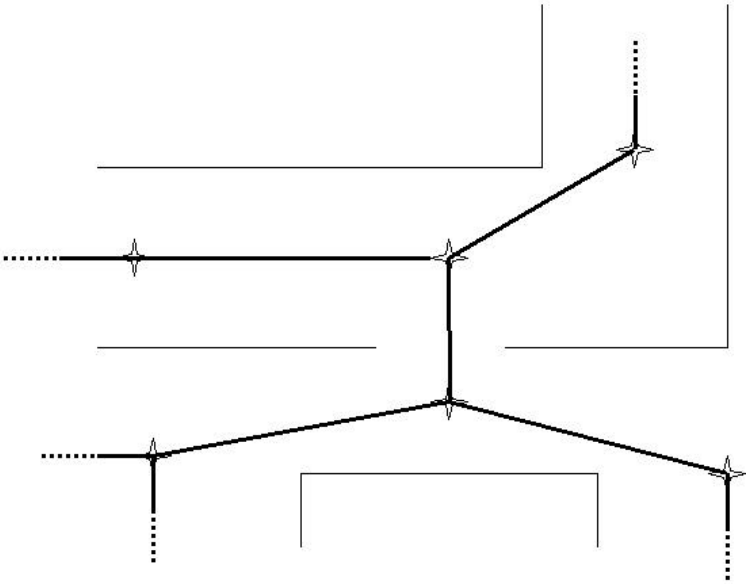


Figure 3: Not oriented graph representation

In order to find the sequence we will use a very known algorithm: the Dijkstra´s algorithm [2], that calculates the shortest path between two vertices in a graph. This algorithm has another advantage: the edges can have different costs, and we can use this to force the robot to prefere a path with less cost. In the figure 4 we can see this better:

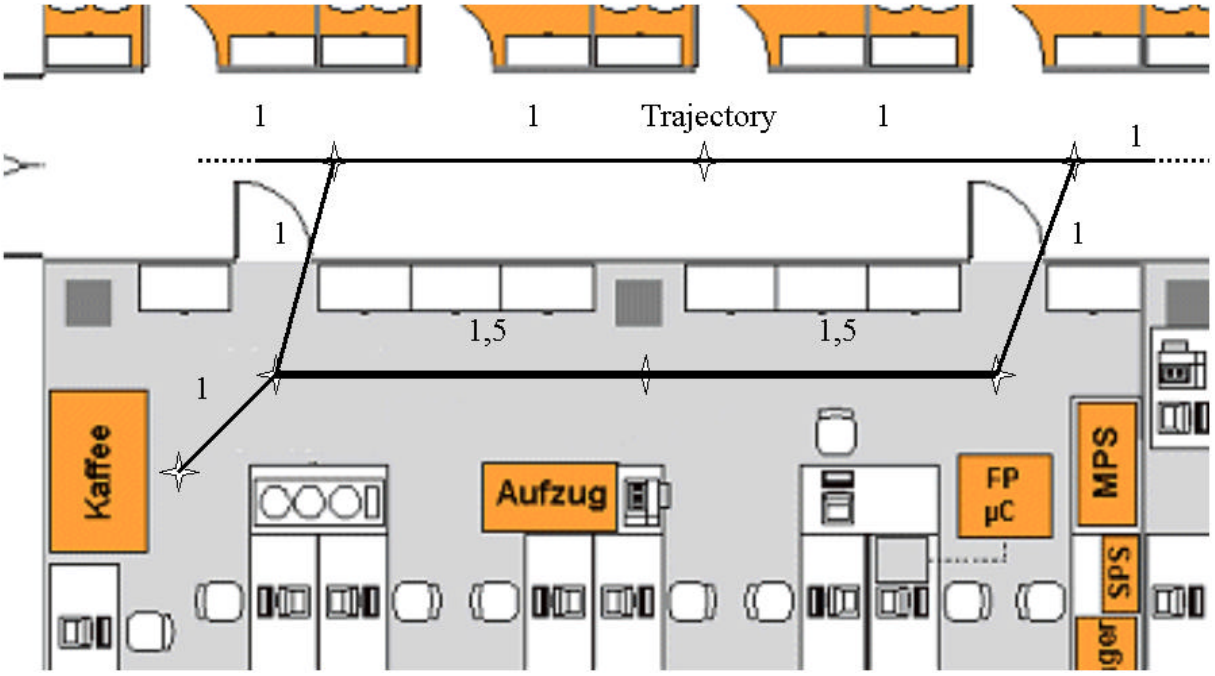


Figure 4: Planning paths with different costs

We represent the high-cost edges with an extra bold. It is obvious, that we would better go through the upper path, because we do not find chairs and similar there. By means of the different edge costs we can force the robot to go through the easiest way.
The cost of each edge will mean the physical distance between the points represented by the connected vertices. As said, this edge will have a higher cost if it implies navigation inside a room, with chairs and such obstacles.

After the execution of the algorithm, we will have a very effective sequence of intermediate points which will be followed by the robot. This information will be used to calculate the real-time path that will be really used (see "Obstacle-free path calculating" component).

3 Component Robot localization and position correction

The robot localization is a very necessary step in our navigation procedure. When the robot moves, the Saphira program calculates the actual position and orientation of the robot according to the measurements taken by the encoders mounted on the wheels. The integration of these measurements using the cinematic model of the robot determines the position of the robot. But as said in the System Architecture document, this process is not error-free, and the deviances, which are little at the beginning, grow with each meter covered by the robot. If no corrections are realized, the navigation becomes impossible because the errors are so big, that the actual position of the robot does not have any more a meaning (see fig. 10 for example).
Due to these reasons, it is obvious why we must perform a robot localization and then a position correction. The frequency of this operation will be determined by the experimental errors suffered (not yet measured) and the path followed by the robot (an intricate path as result of many obstacles is not expected, but increases considerably the position errors). This component is not responsible of this frequency: that will be made in the following one "Obstacle-free path calculating". Here we perform only the localization and the position correction.

3.1 Robot localization

The algorithm used to localize the robot will be very simple, and does not intent to be a general localization method. We will use certain known properties of the map in order to determine the exact (or very approximate) actual position of the robot. We do that by means of a comparation between the expected information and the real information from sensors (specially from laser, that has a very high precision).

The first comparison element will be a **corner**, shown in fig. 5:

Figure 5: Possible corner configurations: a) concave, b) convex

As we can see from these two screenshots from Saphira in the fig. 5, the laser (represented with the dotted lines) provides a very exact information which can be used for the robot localization. We have enough points to perform a linear regression and obtain the parameters of the corner. We will show now that it is possible to determine the exact orientation and position of the robot using these measurements from the laser. Assuming that we have already calculated the linear regression from the sensor data, we have (for the case a) concave corner from the last figure; the case b) is completely analog):

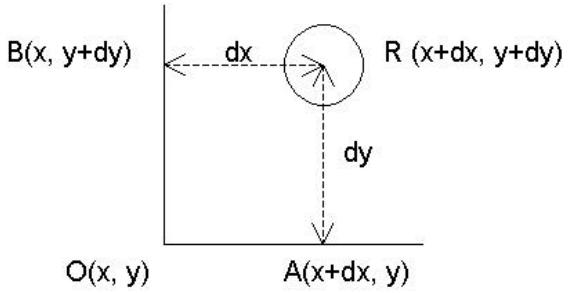


Figure 6: Determination of the position and orientation in a concave corner

The position coordinates of the corner O(x,y) are known. The robot orientation and distance respect to the walls (dx, dy) are also known due to the laser measures. Therefore, the orientation can be easily determined by comparing the current measured robot orientation respect to the walls and the orientation of the walls. The real walls orientation is known, then the real orientation of the robot can be easily determined (the compass mounted on the robot will provide us extra information to help us in this task). The position of the robot R(x+dx,y+dy) can be also easily determined, because the distances dx and dy are calculated, and the corner position O(x,y) is known.

As we can see, with an easy comparation we can determine the position of the robot. For the convex corner the reasonings are analog, and for a rotated diagram, a similar proceeding is needed (we have to use the functions *cos()* und *sin(),* it is a little more elaborated, but with the same philosophy).

The second comparison element will be a single or double **wall**, shown in fig.7:

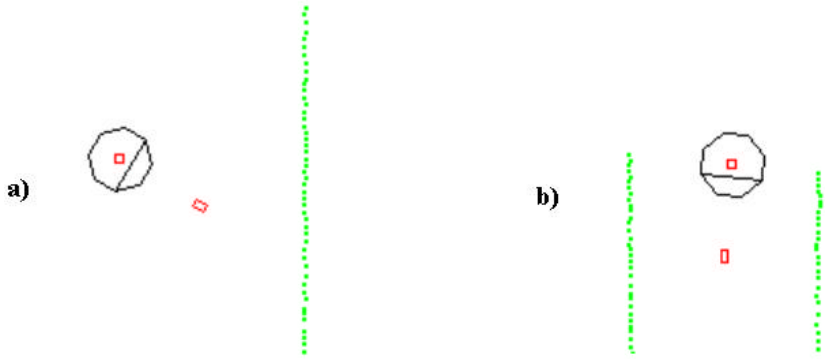


Figure 7: Possible wall configurations: a) simple, b) double

For the case a), proceeding like in the case before (corners), we will be able to determine only one coordinate of the robot and the orientation. Assuming that all the walls from the IAS Institute are perpendicular to the others, the wall direction corresponds always to an axis. Therefore, with “vertical” walls we will able to determine the y-coordinate, and with

"horizontal" walls, the x-coordinate. We can not get more information if we have two walls, but this information will be more precise, because we know the distance between the walls too. It is possible to work with one wall, but we could not distinguish a wall from a flat obstacle, and we would commit a very big error if we use an obstacle as reference! Therefore, we will try not to use the single-wall localization method. Actually, we will use the double-wall localization method only when necessary, because a corner will always provide a more reliable localization.

The third comparison element will be a **door** (see fig. 8):

Figure 8: Robot localization by means of a door

The proceeding here is very similar to the proceeding for a simple wall. The difference is now, that the wall has a "hole" (the door) which can help us to determine the missing coordinate (in the figure, the x-coordinate). The problems of this method are these from the wall-method, and additionally:

- If the door is closed, the method fails completely
- If something stays close to the door, the measurements will be erroneous. Therefore we should check how big is the detected hole (it should be as big as the door).

Other references are possible in order to perform the robot localization, but will not be as good as the corners (and double walls). The corners will be preferred, and the other possibilities used only when necessary, with a double wall as better second option.

3.2 Robot position correction

Once the localization operation has been completely performed, the robot position must be corrected. This could be made in two ways:

- The robot can be moved to the position expected by the navigation software
- The stored robot position can be updated to the real physical position value.

Which option do we select? We will prefer the second option, because it would not make sense to perform an extra movement for such a correction. We will simply update the value of the robot position to the new real one, and the path calculating algorithm will do the rest to reach the next intermediate point.

4 Component Obstacle-free path calculating

After the „Sequence of intermediate points planning“ component execution, we have a list of intermediate reference points to be reached in a consecutive way. With this informatin and that from the range devices (laser, sonar), a reactive path has to be calculated and followed. This action will be cyclical executed in order to react to the obstacles which appear on the way.

4.1 The Gradient Navigation Module

The basic tool to calculate an optimal path will be the Gradient Navigation Module included in the Saphira distribution. It calculates a minimal distance/time path to the desired target point attending to the measurements of the laser/sonar, current speed and robot position and, optionally, the known map of the institute. We show here (fig. 9) a pair of examples:

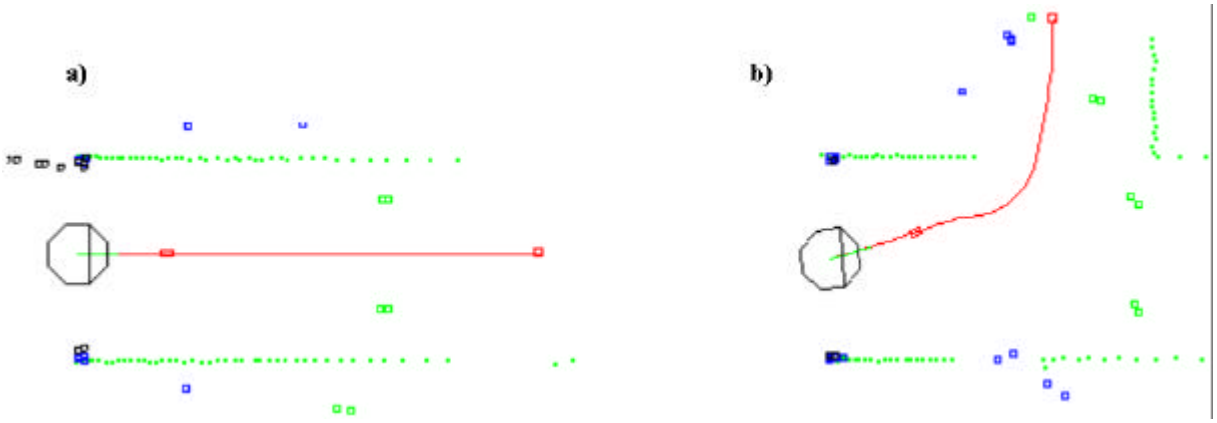


Figure 9: Gradient Navigation path in a) corridors, b) 90-corners without obstacles

In the figure 9 the dotted lines represent the laser measures, the little squares, the sonar measures, and the continuos line, the path traced by the Gradient Navigation Module. As we can see, the Gradient Navigation method calculates a very efficient path which considers the presence of obstacles (see b) for avoiding of the corner). The already known map can be also given to this algorithm, but we have to be then very careful: the accumulated navigation errors produce a discordance between the measured and the stored references, and this can lead the robot to an inadequate behaviour. We show this in the following figure 10:

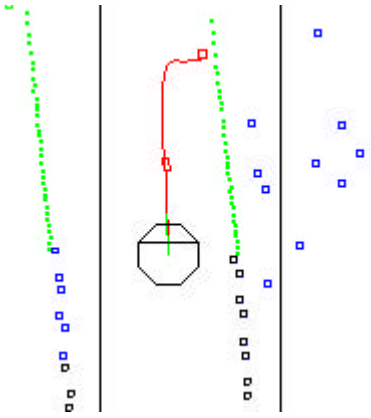


Figure 10: Problems of using the known map in the Gradient Navigation

The inclined lines correspond to the points measured by the laser, and the continuous lines correspond to the stored map. The current robot coordinates are relative to the stored map, but the real position is different. In this figure the robot wants to reach an intermediate point in a corridor, but he will crash into the right wall (dotted line). The conclusion is that we can not use a map for the Gradient Navigation without a proper robot localization.
Let us see now what happens when an obstacle not stored in the map is detected by the robot´s sensors. The figure 11 illustrates this case:

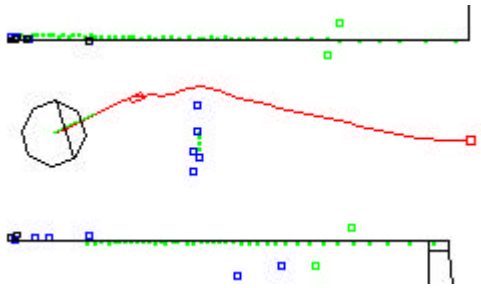


Figure 11: Obstacle avoiding performed by the Gradient Navigation Module

The obstacle was not expected, because it is not contained in the known map of the institute. But the robot´s sensors (the sonars, represented with little squares, and the laser, with dots) detect it and the Gradient Navigation Module finds an alternative path which avoids this obstacle. Therefore, for single, little and fixed obstacles, the Gradient Navigation offers us a very efficient navigation almost without programming.

4.2 Reaching an intermediate point reached

The next step is deciding when have we reached the intermediate point. We have also to consider what happens when this point is not reachable or implies useless navigation. Therefore we will distinguish these main cases:

- **The intermediate point is easily reachable**. The following figure 12 describes this case:

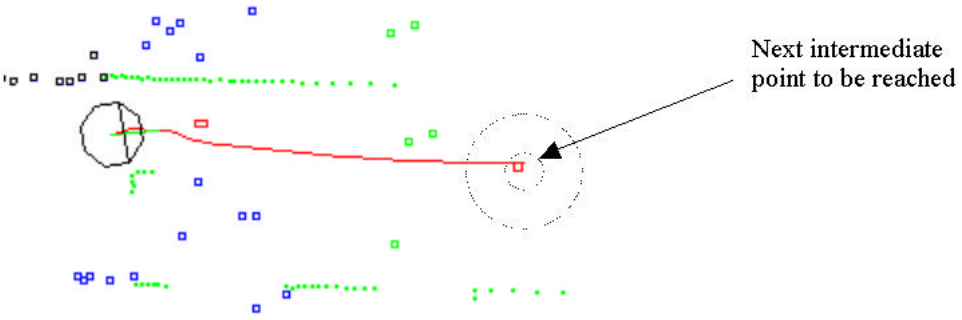


Figure 12: Intermediate point easily reachable

The obstacle which is being avoided does not represent a problem, because the intermediate point to be reached is far away from it. When the obstacle is avoided the navigation continues normally.
If this intermediate point is not the last one, we could be interested in passing it by without reaching it exactly. We will define a circular area around the point, so that when the robot reaches this area, goes to the next intermediate point and “forgets” this one.

- **The intermediate point is not easily reachable** (but can be reached). This situation would correspond to shown figure 13:

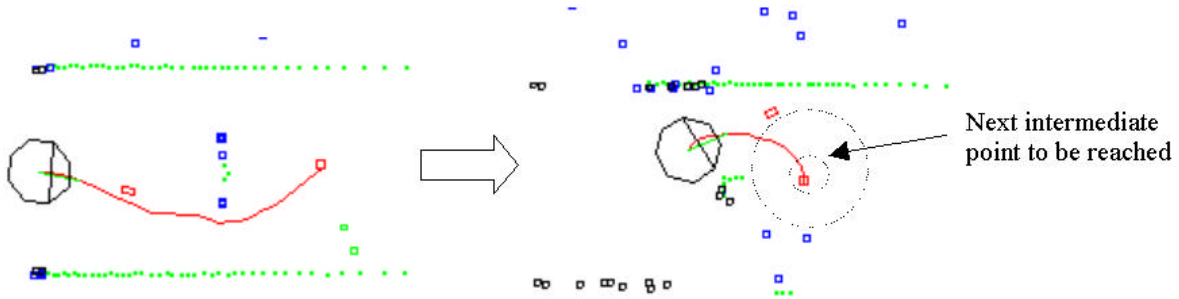


Figure 13: Intermediate point not easily reachable (close to an obstacle)

As a result of the closeness between the obstacle and the intermediate point, we observe an inestable behaviour in an initial phase of path calculating. Furthermore, an useless approximation to the intermediate point is performed. The solution is, like in the case before, to define an area around the point, and when the robot is in this area, go to the next intermediate point.

- **The intermediate point is not reachable (coincides with an obstacle).** We represent this case with the figure 14:

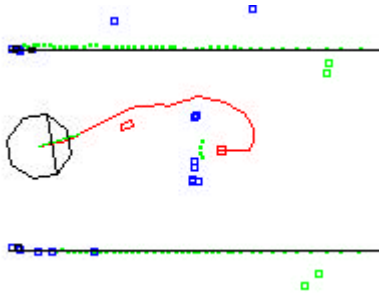


Figure 14: Intermediate point not reachable

This case is much more problematic, and can not so easily solved defining an area around the intermediate point (the execution corresponding to this screenshot ended with a crash into the obstacle, using only the Gradient Navigation). We have to detect these situations (which are very similar to the blocking situations, as we will see later), and then, we invoke the "Alternative intermediate point finding" component in order to determine the new intermediate point to be reached.

4.3 Detection of "way blocked" and "door closed"

The "Obstacle-free path calculating" has another important task: has to check if the path calculated is blocked, and if the doors which have to be crossed are closed (or blocked, or not properly opened). In principle there are two different tasks:

- **"Door closed" detection**. This comprobation has not to be continuosly performed, because we do not have to cross doors continuously. We have to check it only when the door is needed, and we will not need all the possible doors which can be found in the institute. Therefore we will have to mark somehow the needed doors as special points in our sequence of intermediate points. These doors have to be reached with a good precision (a robot localization has to be inmediately before performed), and then we check if the

expected opening is present. If this opening is not detected, or has not the correct parameters, we will consider the door to be closed (or blocked, but in this case it will have the same meaning). This circumstance will be reported to the Tour Guide (see "Tour guide-Communication" component for further details).

- **"Way blocked" detection**. In contrast to the "door closed" detection, this comprobation has to be continuously performed because the way can be blocked in any point of the planned trajectory. This case is very similar to the case "The intermediate point is not reachable (coincides with an obstacle)" presented before. The solution will be in both cases to invoke the "Alternative intermediate point finding" component, which will determine if the robot is really blocked or not, and determine, if possible, a new alternative intermediate point to be reached. Here (fig. 15) we have an example of not avoidable obstacle, which blocks the way:

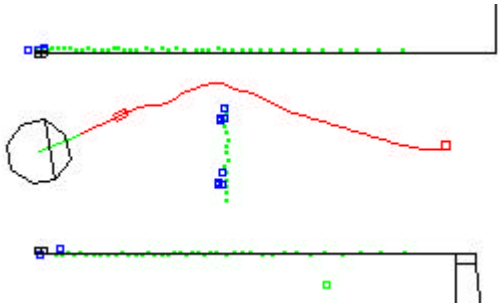


Figure 15: Not avoidable obstacle which blocks the way

In theory a path is found, but the gap is too small for the robot. Therefore, this derives in an inestable behaviour and a successful path is never found. Furthermore, this obstacle will not be avoidable at all, and no alternative intermediate point will be found: the blocking situation here is sure.
Other case is the presence of mobile obstacles. Since their movements are not predictable, they will be much more problematic than the fixed obstacles.

5 Component Alternative intermediate point finding

As we have seen before („obstacle-free path calculating“ component), if during the navigation the robots gets blocked, or an intermediate point can not be reached, then the „Navigation control“ component executes the „Alternative intermediate point finding“ component in order to find a solution for this situation. Here the available information will be analized, and then two things can happen:

- The blocking situation can not be resolved. This circumstance must be then reported („Tour guide-Communication“ component).
- An alternative intermediate point was found. We undo the blocking situation and go on with the navigation („Obstacle-free path calculating“ component) towards this new intermediate point.

The important questions here are (and will be answered in next subchapters):

- How do we know that the blocking situation can be resolved or not?
- How do we calculate an alternative intermediate point?

5.1 The blocking situation

To the first question, many possible situations can appear:

- **A big obstacle blocks the way** (or a mobile one which has the same effect). In this case, the sensors will be provide us information about the size and position of this obstacle. We know how much place we need for the navigation (the size of the robot is known), and also how wide are the corridors. Therefore, we will be able (if the size of the obstacle was determined) to know if the further navigation is possible.
- **An obstacle occupies the position of the intermediate point but does not completely block the way**. A very similar case was described in 4.2 (see Figure 14). Two things can happen:
  - The intermediate point blocked is the final point. In this case, there is a blocking situation, unless we are already very close to the final position. In this case, we could consider that the final position was reached (a limit has to be defined)
  - The intermediate point blocked is *not* the final point. Then the inmediate solution would be to pass this intermediate point by, and go to the next one. Nevertheless, that will not be possible when the obstructed point corresponds to a door, a corner or other critical point (see 5.2 later for further details). Therefore, these critical vertices from the graph will be have to be marked as special points.
- **A door is closed**. This case was described in 4.3. If there is an alternative way to reach the point, this will be taken. If not, the blocking situation will be reported to the Tour Guide.

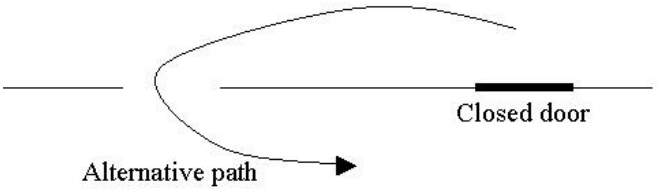


Figure 16: Alternative path for a closed door

5.2 The alternative intermediate point

To the second question that we made, the determination of the new alternative intermediate point will depend on the situation:

- If we have an obstacle occupying the intermediate point and that does not completely block the way, and this intermediate point is not critical (that means, is not in a corner, door or similar, and is not the target point), the next point in the sequence will be taken as new intermediate point to be reached (see fig. 17):

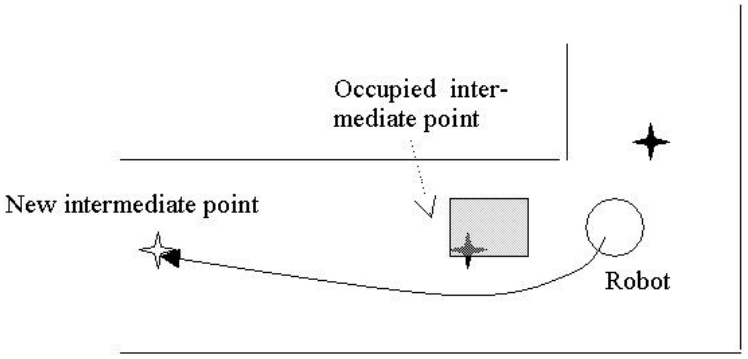


Figure 17: Taking the following next intermediate point of the sequence

- If we have some of the following cases:
  a) We have an obstacle occupying the intermediate point and that does not completely block the way, but this intermediate point is critical,
  b) The door is closed,
  c) The way is completely blocked by a large-size or mobile obstacle,
  d) The target point is occupied by an obstacle.

  in any of these cases, passing the current intermediate point by and choosing the next one from the sequence is not a valid method, because will lead the robot to a potentially dangerous or useless situation. Therefore, the solution will be to recalculate the sequence of intermediate points to be reached, but now the last intermediate point successfully crossed is considered to be the start point, and the following modifications will be made on the graph representing the intermediate points:

  - for the cases a) and b) the edges connecting the blocked intermediate point (vertex) to the other points are removed from the graph. That means, that the blocked intermediate point is considered not reachable any more.
  - for the case c) only the edge connecting the last intermediate point successfully crossed to the next intermediate point planned will be removed. This is so because we do not know if the next intermediate point is obstructed or not; we will assume that not and that it can reached via an alternative path (if there is).
  - for the case d) there is no solution posible, since the target is blocked and can not be reached at all.

  If after these operations the vertex in the graph representing the end point has not any edge connected to it, we will consider that the way is blocked (or the door closed) and report it to the Tour Guide, because no alternative is available.
  In other case, we calculate a new sequence of intermediate points with the modified graph and the new start point. If a valid sequence is found, we proceed with it. If not, we will consider that the way is blocked (or the door closed) and report it to the Tour Guide.

6 Component Tour guide-communication

This component is responsible of the communication with the Tour Guide: we have to accept its orders, and at the same time, report the success of the operations or the problems which could appear during the navigation.

6.1 Methods offered to the tour guide

The following methods are made available for the Tour Guide:

- *GoTo(int numberOfTarget)*
  With this method, the Tour Guide sets the target to be reached by the mobile robot RobIAS. This function will be executed so long as the navigation takes place (or an error occurs), because the Tour Guide is programmed in such a way that can not read the return values. The only possibility is to return OK or ERROR. When the Tour Guide receives OK, goes on with the presentations, and when receives ERROR, informs about this situation and then try to resume the navigation. For this reason the method *GoTo()* must be executed so long as the navigation takes place, because we need to guarantee a correct synchronization between both processes.

- *Resume(bool Action)*
  This method allows the Tour Guide to resume the execution of the navigation process after a previous blocking situation caused by a way blocked or a door closed. The parameter defines the action to be realized:
  - *Action = true* : continue with the navigation
  - *Action = false* : end of the guided tour (the target is not reached)

  Also in this case the execution of the function will be ended when the navigation is complete (or an error occurs) due to the same reasons as for the function *GoTo()*.

Another important issue is how is this information provided by the Tour Guide communicated to the navigation software. As we said before in previous documents, the communication between the Tour Guide and the navigation software is not a priority of this project. A very simple method consisting in writing and reading a little file will be used to it possible. The methods *GoTo()* and *Resume()* write the information received to a little file. This file is periodically read by the navigation software, in order to know if new instructions were given. As we discussed in the System Architecture document (see 5.2), there are some possible errors associated to this technic, but we will ignore them if experimentally everything goes on.

6.2 Reported situations

Three possible situations will have to be reported to the Tour Guide:
- The navigation was successful and the robot is in the final position
- Way blocked
- Door blocked

The last two situations, that were always considerated as separated cases, will be for the communication purposes groupped as a single one, because the Tour Guide can only recognize OK or ERROR as return values.

The method used to report these situations to the Tour Guide has been previously already presented: the functions *GoTo()* and *Resume()* invoked by the Tour Guide have only two possible return values: OK and ERROR. If everything goes ok, the value OK will be returned when the navigation is finished and the robot has reached the final position. But if there is an error, the function will be interrupted and the value ERROR returned. This works so for both functions, and actually both cases have a very similar behaviour.

The easy implementable solution proposed is: the functions *GoTo()* and *Resume()* stay in a loop which is executed if the navigation software does not change a control variable. This control variable will be also read from a file (because the direct communication to the methods from the software has not been implemented), with a reasonable frequency to avoid ressources collapse, and the navigation software will write the control variable to this file when the navigation is finished or there is a blocking situation with an appropiate value. In this easy way we can implement a very easy communication between the Tour Guide and the navigation software.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Test Specification

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Test Specification	1.0	Serrano López	06.02.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-test-specification-v10.doc	7	28/05/03 17:45	sw-test-specification.dot

Document Version Management

Version	Author	QS	Date	Status	Changes
0.1	Serrano López	Jo	03.02.03	in progress	Creation
1.0	Serrano López	Jo	06.02.03	submitted	
1.0	Serrano López	Jo	06.02.03	accepted	

0 Table of Contents

1 Test Requirements

For the navigation software test, the following elements are required:

- The robot RobIAS, with the batteries properly charged, and the cables to the electric network, monitor, mouse and keyboard unplugged (that means, RobIAS must be completely autonomous). The required version of Saphira (see System Architecture) and the navigation software must be previously installed on the embedded PC.
- A PC connected to the IAS LAN, to remotely start the program by means of the Wireless LAN card integrated on RobIAS.

If we want also test the whole "guided tour system", then we need additionally:

- The "Tour-Guide" application developed by Kristian Dencovski. This tour guide will be executed on the same PC used to remotely start the navigation program, but could be executed on another computer.
- The robot RobIAS must also have on the embedded PC a Personal Web Server (PWS) with ASP engine installed, the ASP tour guide application files and the corresponding audio wave files required for the multi-media based presentations. Other components must be installed on this embedded PC in order to allow the execution of the tour guide, but they will be not be presented here. Please for further details, see the project of Kristian Dencovski (Wahlstudienarbeit 1884).

Additional test requirements are:

- The test must be realized in the IAS institute, within a real environment, with real start and end points.
- Presence of spontaneous walkers which intercept the trajectory of the robot are desired, in order to test how good is the robot avoiding such obstacles.
- Potential dangerous situations (collisions with fragile elements, or with people who do not notice the presence of the robot) must be avoided. The whole test must be supervised by a person able to stop the robot or warn the people if necessary.

2 Test Methods

The system will be tested placing the robot in a initial position, and then giving the order to move to a target position. Two types of test have to be realized:

- Tests without tour guide, to check only the performance of the navigation.
- Tests with tour guide, to check both the performance of the navigation and the interaction with the tour guide.

During these tests the following information must be considered:

- Duration of the navigation.
- Quality (distance and easiness) of the path planned.
- Presence or not of collisions.
- Position calculation errors committed by the robot during the navigation (we are not responsible of these errors; they are implicit from the navigation, but we have to observe them).
- Localization errors (here we are responsible of these errors).

3 Test Criteria

The test will be considered successful if the following conditions are fulfilled:

1. The robot reaches its desired target.
2. The robot does not collide with any existing obstacle on the way.
3. The final position error is small (that means, allows the robot to go through the narrowest door).
4. The time used to reach the target point is short.
5. The not avoidable obstacles are detected and reported.
6. The closed doors are detected and reported.
7. The used path is logical (no meaningless loops and similar) and not too long.
8. The robot localization and position correction brings the robot to the real position.
9. The synchronization and communication with the tour guide is correct.

4 Test Cases

4.1 Communication with the tour guide: *GoTo()*

Test case	Communication with the tour guide: *GoTo()*
To be proven	The correct communication between the navigation software and the tour guide. Only the *GoTo()* method is to be proven here; the method *Resume()* has its own test case (Communication with the tour guide: *Resume()*), and the reporting of errors situations is checked in other test cases (Closed doors detection, Way blocked detection).
Initial condition	The navigation program is correctly started and the robot situated in the initial point.
Necessary inputs	The tour guide must execute the *GoTo()* method.
Expected results	The robot calculates a sequence of intermediate points and then begins to move in the desired target direction. In this case, the method invoking was successful.

4.2 Navigation without obstacles and door opened

Test case	Navigation without obstacles and door opened
To be proven	The correct navigation of the robot under ideal circumstances: no obstacles on the way and all the needed doors opened.
Initial condition	The navigation program is correctly started, the robot situated in the initial point and the way completely clear.
Necessary inputs	The navigation must be started by the user (simple test) or by the tour guide (complete test).
Expected results	The robot navigates withouth collisions, through an efficient path, corrects regularly the position without problems and reaches the desired target with enough precision (see point 3. from test criteria)

4.3 Fixed obstacles avoiding

Test case	Fixed obstacles avoiding
To be proven	The correct navigation of the robot when there are obstacles on the way which don´t move and can be passed by.
Initial condition	The navigation program is correctly started and the robot situated in the initial point.
Necessary inputs	The navigation must be started by the user (simple test) or by the tour guide (complete test).
Expected results	The robot navigates withouth collisions, through an efficient path, avoids the fixed obstacle(s), corrects regularly the position without problems and reaches the desired target with enough precision (see point 3. from test criteria)

4.4 Mobile obstacles avoiding

Test case	Mobile obstacles avoiding
To be proven	The correct navigation of the robot when there are obstacles on the way, which move but can be passed by (they allow the navigation of the robot).
Initial condition	The navigation program is correctly started and the robot situated in the initial point.
Necessary inputs	The navigation must be started by the user (simple test) or by the tour guide (complete test).
Expected results	The robot navigates withouth collisions, through efficient path, avoids the mobile obstacle(s), corrects regularly the position without problems and reaches the desired target with enough precision. (see point 3. from test criteria).

4.5 Closed doors detection

Test case	Closed doors detection
To be proven	The detection of the closed doors, when a trajectory through this door was planned.
Initial condition	The navigation program is correctly started, the robot is situated close to the door (the navigation will be not proven here) and the door to be crossed, closed.
Necessary inputs	In this case the navigation should be started by the tour guide, because we can test at once both the closed door detection, and the reporting of this information to the tour guide. A simplified test to check only the detection (and not the reporting) can be done without tour guide.
Expected results	The robot stops when he detects the closed door, and reports this to the tour guide.

4.6 Way blocked detection

Test case	Way blocked detection
To be proven	The detection of blocked ways which can not passed by and do not allow the navigation of the robot to its target.
Initial condition	The navigation program is correctly started and the robot situated in the initial point, and the way that it needs to reach its target, blocked with an obstacle which can not be avoided.
Necessary inputs	In this case the navigation should be started by the tour guide, because we can test at once both the way blocked detection, and the reporting of this information to the tour guide. A simplified test to check only the detection (and not the reporting) can be done without tour guide.
Expected results	The robot tries to find an alternative way but stops when he detects that the obstacle can not be avoided, and reports that the way is blocked to the tour guide.

4.7 Communication with the tour guide: *Resume()*

Test case	Communication with the tour guide: *Resume()*
To be proven	The correct communication between the navigation software and the tour guide. Only the *Resume()* method is to be proven here; the method *GoTo()* has its own test case (Communication with the tour guide: *GoTo()*, seen before), and the reporting of errors situations is checked in other test cases (Closed doors detection, Way blocked detection, seen before).
Initial condition	The navigation has suffered a blocking situation before (way blocked or door closed) and waits for the resume signal. The blocking situation must be removed.
Necessary inputs	The tour guide must execute the *Resume()* method.
Expected results	The robot moves again in order to reach the target. We test here the communication with the tour guide, that means, that the resume invokation by the tour guide makes the robot to move again. The navigation should be all right if the obstacle was removed and the previous test cases were successful.

4.8 Real navigation for the tour guide

Test case	Real navigation for the tour guide
To be proven	The correct working of the navigation software for a realistic situation, which could appear during a guided tour through the Institute.
Initial condition	The navigation program is correctly started, the robot situated in the initial point and the real circumstances used (possible closed doors, people not warned, furniture on the way, etc.).
Necessary inputs	The tour guide must be started by the Tour Guide.
Expected results	The robot navigates withouth collisions, through an efficient path, avoids all the avoidable obstacles, corrects regularly the position without problems, reacts to blocking situations, report them, reacts to resume signal and finally reaches the desired target with enough precision (see point 3. from test criteria)

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Design Review Protocol

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Design Review Protocol	1.0	Serrano López	11.02.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-design-review-v10.doc	6	28/05/03 17:46	sw-design-review.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	11.02.03	in progress	Creation
1.0	Serrano López	Jo	11.02.03	submitted	
1.0	Serrano López	Jo	11.02.03	accepted	

Place, Date:	IAS, 11.February 2003	
Participants:	Pascal Jost, Javier Serrano	

CHECKLIST System Architecture

Checkpoints	**Actions/ Comments**
1) Are the boundary conditions considered (target platform)?	yes, software was designed for RobIAS.
2) Is the selected structure of the system components practical?	yes
3) Are alternatives examined?	yes
4) Is (are) the architecture diagram(s) clear and comprehensible?	yes
5) Are the defined interfaces practically selected?	yes
6) Are all the requirements realised by the architecture?	yes, completely.

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
11.02.03	1.0/04.02.03	Jo		X	

Comments:

CHECKLIST Components Specification

Checkpoints	**Actions/ Comments**
1) Are all components described?	yes
2) Is the description of the components understandable and unique?	yes
3) Is the implementation of all components possible due to the specifications?	yes, if the referenced algorithms are considered

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
11.02.03	1.0/10.02.03	Jo		X	

Comments:

CHECKLIST Test Specification

Checkpoints	**Actions/ Comments**
1) Are the general test requirements defined?	yes
2) Are the selected test methods applicable to the evaluation items?	yes
3) Are the test criteria selected in such a way that a practical result assessment can take place?	yes
5) Are the test cases completely specified? (What is checked, starting situation, inputs, expected results).	yes
6) Do the test cases cover the requirements specified in the system requirements specification?	yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
11.02.03	1.0/06.02.03	Jo		X	

Comments:

CHECKLIST Project Progress

Checkpoints	**Actions/ Comments**
1) Have all items of the last review, which had to be revised, been completed?	no open items
2) Is the planned timetable of the project (Bar Chart) kept?	yes
3) Is the inter-office slip up to date?	yes

Acceptance

Date of Inspection	Inspector	Items to be revised	Accepted	
			yes	no
11.02.03	Jo		X	

Comments:

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Implementation Review Protocol

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Implementation Review Protocol	1.0	Serrano López	14.04.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-implementation-review-v10.doc	5	28/05/03 17:46	sw-implementation-review.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1 1.0	Serrano López	Jo	14.04.03	in progress accepted	

Place, Date:	Stuttgart, 14.04.2003	
Participants:	Javier Serrano López, Jost	

CHECKLIST Source Code

Checkpoints	**Actions/ Comments**
1) Are the programs well structured and readable?	Yes
2) Are the identifiers meaningful?	yes
3) Are the valid programming guidelines kept?	yes
4) Are the programs well commentated?	yes
5) Is the interface separate from the implementation in all modules?	yes
6) Are the modules implemented in accordance with their design specification?	yes
7) Is the scope of functions, classes, modules etc. acceptable?	yes
8) Are all demanded requirements implemented?	yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
14.04.03	1.03/08.04.03	Jo	-	X	

Comments:

CHECKLIST Test Protocol

Checkpoints	**Actions/ Comments**
1) Are all test cases executed and documented?	Yes, they are consistent
2) Is the total result of the tests sufficient for the acceptance?	yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted	
				yes	no
14.04.03	1.0/14.04.03	Jo	-	X	

Comments:

CHECKLIST Project Progress

Checkpoints	**Actions/ Comments**
1) Have all items of the last review, which had to be revised, been completed?	No open items
2) Is the planned timetable of the project (Bar Chart) kept?	The last two reviews are delayed, but the deadline will be kept.
3) Is the inter-office slip up to date?	Yes.

Acceptance

Date of Inspection	Inspector	Items to be revised	Accepted	
			yes	no
14.04.03	Jo	-	x	

Comments:

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

User Manual

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
User Manual	1.0	Serrano López	12.04.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-user-manual-v10.doc	21	28/05/03 17:48	sw-user-manual.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	09.04.03	in progress	Creation
1.0	Serrano López	Jo	09.04.03	submitted	
1.0	Serrano López	Jo	12.04.03	accepted	

0 Table of Contents

1 Installation

For the operation of the system we will use the operating system Microsoft Windows 98, Second Edition, Version: 4.10.2222A installed on the embedded PC from RobIAS. If you want to use the whole „Guided tour“ system, please consult the documentation of the WSA 1884 from Kristian Dencovski (in German), where you will find the installation instructions for the „Tour guide“.

1.1 Accessing the embedded PC of RobIAS

Before you can install any programs or use the programs already installed on RobIAS, you have to access its embedded computer. You have three possibilities to do this:

a) Remote console:

1. Connect the Wireless LAN modem to the 220V power net (use the transformator) and to the IAS LAN Network. After a few seconds you should see green lights on the indicators PWR, ETHR and L on the modem. Please be careful with the location of the modem; if possible place it in the same floor where the robot is going to work, and not very far away.
2. Unplug the serial mouse (if connected) and turn the robot RobIAS on. To do this toggle the interruptor with the message „Main Power Charge“ on the left lower side of the robot to on. You should be able to hear the computer starting (including a beep), and that means that it is being booted. Please observe that the robot has always enough batteries to run properly. For more about this topic, see please the point 3.2 (Batteries)
3. You will have to wait 2 or 3 minutes (even 5 if in the last session the embedded computer was not properly shut down). Then, from any computer connected to the IAS LAN, you have to start the program *vncviewer.exe.* To the message „VNC Server“ you have to type **`ias198`** and then you will be asked for the password; you have to type: **`ias`**. In the following figure you can see the appearance of this program:

Figure 1: The program *vncviewer.exe*

4. Then you will receive the message „Enter Network Password“, type **`ias`** again, and usually the VNC Viewer window will be closed without apparent results: this is so because you have only logged in the session. Now you have to connect really to the computer; repeat the operation in point 3: start the program *vncviewer.exe,* type **`ias198`** and then type **`ias`**. Now you should see a new desktop (in an orange color or similar), this one from the Windows 98 running on the embedded PC from RobIAS.

5. Open a Windows Explorer (you can open a Windows Explorer by right-clicking on the Start button and then choosing Explorer) and search on the left part of the window the „Network Neighborhood“. In this way you can access the shared data of the other computers connected to the IAS LAN.

b) File access:

1. Connet the Wireless LAN modem to the 220V power net (transformator) and to the IAS LAN Network. After a few seconds you should see green lights on the indicators PWR, ETHR and L on the modem. Please be careful with the location of the modem; if possible place it in the same floor where the robot is going to work, and not very far away.
2. Unplug the serial mouse (if connected) and turn the robot RobIAS on. To do this toggle the interruptor with the message „Main Power Charge“ on the left lower side of the robot to on. You should be able to hear the computer starting (including a beep), and that means that it is being booted. Please observe that the robot has always enough batteries to run properly. For more about this topic, see please the point 3.2 (Batteries)
3. Open a Windows Explorer on the computer you are using to remotely access the robot. In the tool bar you will find the following options: „Extras => Suchen => Computer“ and then type the name of the computer: **`ias198`**. At this point you should be able to see the IAS198 computer, then you can browse the directory structure so that you can reach the desired folder on the robot. You can copy files from other windows to this one, or use the programs on the robot.

c) Local console:

1. Plug in a monitor, keyboard and serial mouse to the embedded PC of RobIAS. The connectors are on the right lower side of the robot.
2. Turn the robot RobIAS on. To do this toggle the interruptor with the message „Main Power Charge“ on the left lower side of the robot to on. You should be able to hear the computer starting (including a beep), and that means that it is being booted. Please observe that the robot has always enough batteries to run properly. For more about this topic, see please the point 3.2 (Batteries).
3. Login the session **`ias`** with the password **`ias`** when asked for „Enter Network Password“.
4. Open a Windows Explorer (you can open a Windows Explorer by right-clicking on the Start button and then choosing Explorer) and then search on the left part of the window the „Network Neighborhood“. In this way you can access the shared data of the other computers connected to the IAS LAN.

Note: Usually the option **c)** will be recommended for the installation process, because you can directly manage the computer in a fast and comfortable way. The options **b)** and **c)** are also both appropriate in the later steps to copy, change or replace files. To execute the guided tour only the option **a)** is possible, because the option **b)** can not execute programs and because the laser and the serial mouse use the same IRQ and then the laser will not work. That means also, that when you start the computer using the option **c)** and then you want to execute the guided tour, then you have to restart the computer with the mouse unplugged. Installation of Aria and Saphira

1.2 Installation of Aria and Saphira

When you are already connected to the robot, you have to install Saphira and Aria. You have to install the versions **Saphira 8.1.11b** and **Aria 1.1.11b**, and the corresponding install files are contained in the folder named *Version 8.1-11b*. You have to follow these steps:

1. Connect to the embedded PC and open a Windows Explorer (see section 1.1; option **c)** recommended in this case).
2. Create the following directory structure: *C:\Exchange\Projekte\Robias* on the embedded PC of RobIAS. The rest of the documentation will be referred to this structure; you can also choose a different structure and the program will work too, but then please be very careful: do not to mix the folders. Therefore (and because the program will be usually already installed using this directory structure), we strongly recommend to leave this structure.
3. To this folder that you have just created we copy there the folder named *Version 8.1-11b*. This folder contains the install files of Saphira und Aria. Install these files in the following order, and using the following folders when asked during the installation (not the defaults):

File name	Installation folder
Saphira-8.1-11b.exe	*C:\Exchange\Projekte\Robias\Saphira*
Saphira-laser-8.1-11b.exe	*C:\Exchange\Projekte\Robias\Saphira*
ARIA-1.1-11b.exe	*C:\Exchange\Projekte\Robias\Aria*

4. If Saphira and Aria were successfully installed, you should see the new program group called *Saphira* (under *Programs*), as shown in the following figure:

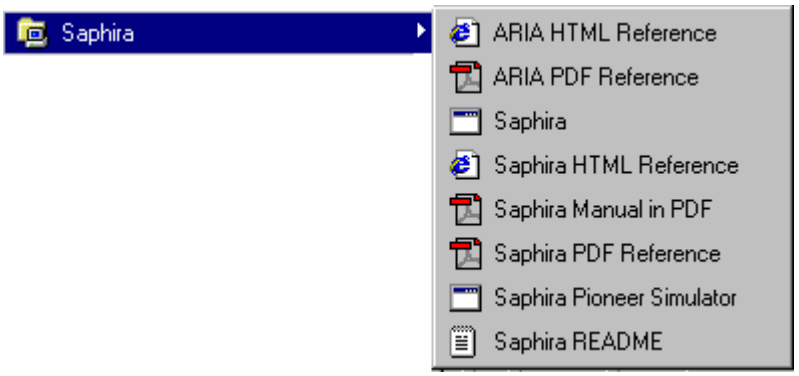


Figure 2: The program group *Saphira*

5. To check if the laser module was successfully installed, start *Saphira* and check if the word *Laser* is present in the upper program bar, as shown in the following figure:

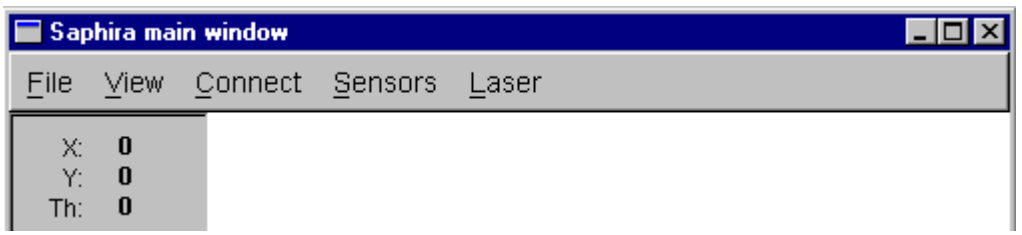


Figure 3: The laser in the program *Saphira*

6. If the points 4. and 5. are all right, Saphira and Aria were successfully installed.

1.3 Installation of the Guided Tour software components

The needed files to execute the guided tour are contained in the archive file *GuidedTourZip.exe.* You have to execute it, and you will see this figure:

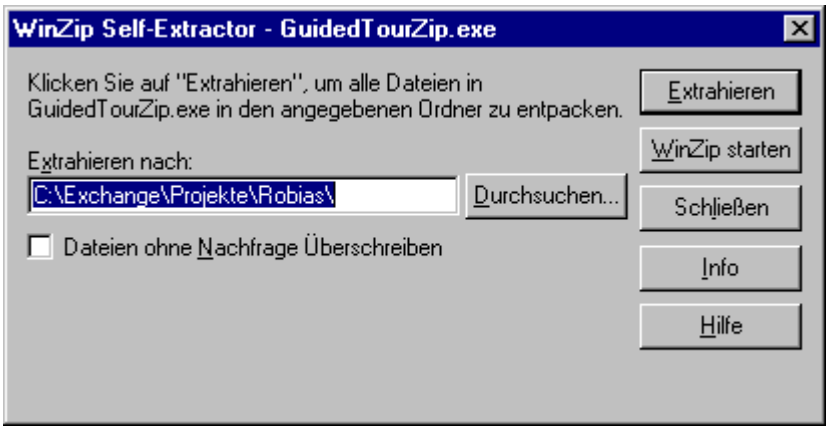


Figure 4: Extracting the needed files

By default, the files are installed in the directory *C:\Exchange\Projekte\Robias*. If you have chosen other directory, you have to type it there. Please note that the option „*Dateien ohne Nachfrage Überschreiben*“ is not selected. If you select it, the files will be overwritten without asking. In this case, observe that the maps and other information will be lost if you confirm this operation! Don´t forget in this case to make security copies if needed. If you do not select this option, even when making a new installation, you will be asked to replace the file *p2pb.p*. Please confirm this operation. If you are asked to replace other files, that means that the guided tour software has been already installed.
When you have decided your folder and if you select this option, press the key „*Extrahieren*“ and the extraction of files will start. All the needed files will be copied in the selected directory, both the executables and the source code, in the correct folders, so that you do not have to worry about it and make mistakes. Please observe the comments before about the files to be overwritten.
The standard installation already contains a map of the IAS second floor, and the needed information (intermediate points, localization references, etc) for the navigation. This information was built attending to the original work from Kristian Dencovski for his WSA 1884 and it is already completely functional. Further modifications of the stations to be visited or the environment must be done by the user, as indicated in the section 5 and the user manual (in german) from Kristian Dencovski.

To install the communication component, you have to execute the file *CommunicationZip.exe.* In this case you should see this window shown in the figure 5:

Figure 5: Extracting the needed communication files

In this case you must not change the options, only press the key *Extrahieren* and the files will be correctly copied. In this case there is no data to be saved before you do this operation.

2 Operation

If you have already successfully installed the needed programs, you can use the robot for our guided tour in a very easy way.

2.1 Preparation of the robot

You have to follow these steps:

1. If you have just installed the guided tour software, go to next step. If you want to start it from the beginning, see the section 1.1 (Accessing the embedded PC of RobIAS) and follow the instructions of the option **a)** .
2. Be sure that the laser is disconnected, and then connect it: the laser is the blue device in the frontal part of the robot. You will find the interruptor on its rear side (ON/OFF). When the red and orange lights disappear and you can see the green one, you can go to the next step.
3. Move the robot to the start position of the guided tour. Try to positionate the robot as exactly as possible, it will help the robot to be more easily oriented.

2.2 Starting the navigation software

In the folder *C:\Exchange\Projekte\Robias\Saphira\bin* of the robot you will find the file *GuidedTour.exe.* Just simply execute it and wait a little. Then move the program window (so that it is refreshed) and you should see something similar to the following figure (you will see it only if you have started the VNC Viewer):

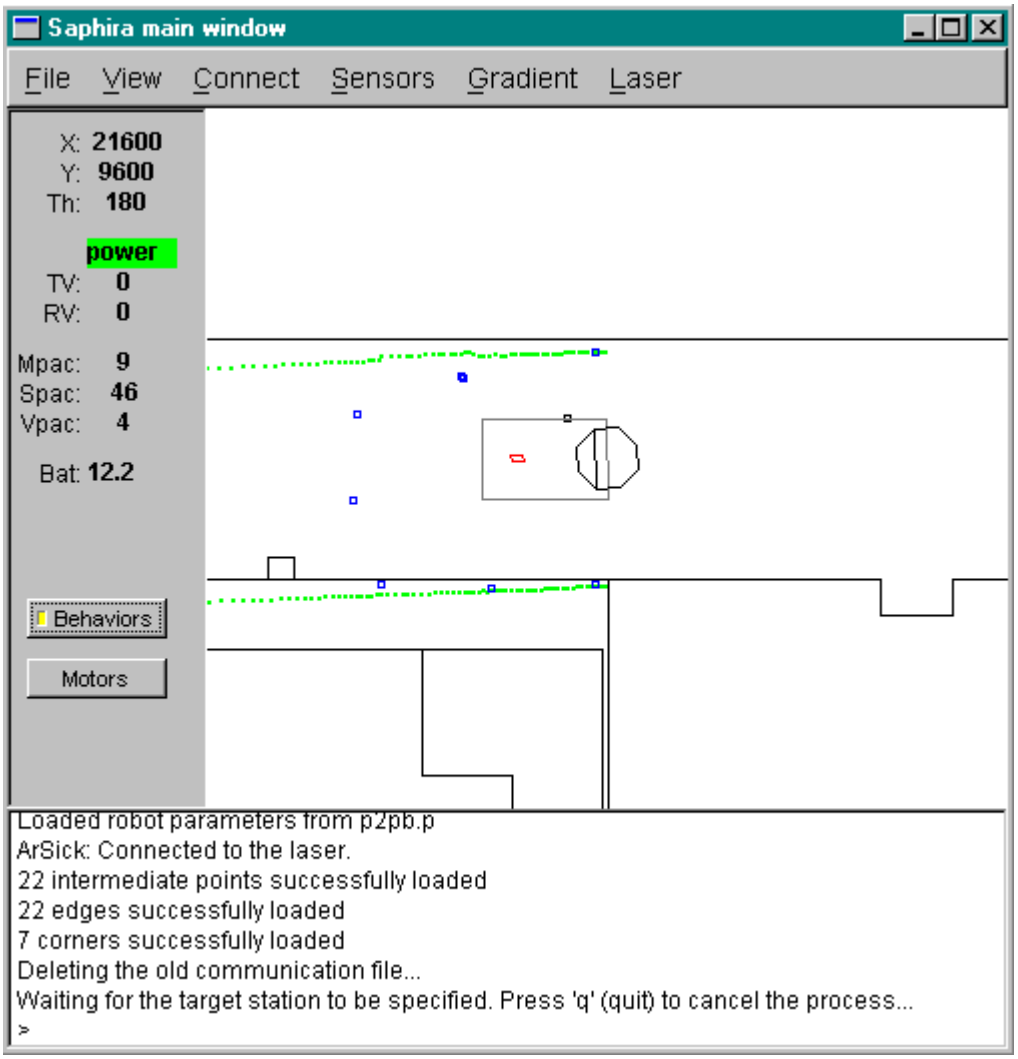


Figure 6: The program *Saphira* with the guided tour just started

You have to check the following things to know if everything was all right:

1. You should see the message *„Waiting for the target station to be specified.Press ´q´ (quit) to cancel the process...*“. That means, that the program is ready to accept the target to be reached. You shouldn´t see something like *„ArSick: error*“ or similar messages.
2. The laser information, on green color, should be visible and correspond (approximately) to the loaded world (in black color). If it is not visible, the laser could not be started, and if it is visible but does not correspond at all, perhaps you have positioned the robot in a wrong start point.

If you see that all is correct, the robot is prepared and you can go to the next section. If not, press the key „q“ (before you have to click with the mouse on the navigation window, where the robot is) and then, when you see the message „**You can now close the application**“ (refresh the window by moving it), close the program by pressing the X on the right-top of the window and then start again, from the step 2. in the section 2.1 (Preparation of the robot). If your problem persists, please read the section 6 (Error Removal).

2.3 Running the „Guided Tour“

Once we have started the navigation software, you have two possibilities:

a) Start the complete Guided Tour. To do this we have to start the „Tour guide“ from Kristian Dencovski (WSA 1884) attending to the instructions contained in his User Manual (Benutzungsanleitung, in German). The navigation software waits already only for the orders.

b) Start a simulator to move the robot. To simulate the orders given by the „Tour guide“, in the folder *C:\Program Files\ActiveX* of the robot you can find the files *Simulator-GoTo.exe* and *Simulator-Resume.exe.* If you want to start the navigation to a determined station, start the *Simulator-GoTo.exe,* write the number of the station and press return. If there is an error during the navigation, you can resume it using the program *Simulator-Resume.exe,* typing the number 1 if you want to resume the navigation, or the number 0 if you want to abort it. These programs are open as long as the robot navigates, and then they are closed.

2.4 Interruption of the navigation software at any moment

In case you need to stop the navigation software at any moment (excepting when playing wave files), you only have to click with the mouse on the navigation window from Saphira (where you see the robot) and press the key „q“, or type the value 0 in the programs *Simulator-GoTo.exe* or *Simulator-Resume.exe.* After this, when you see the message „*You can now close the application*“ (refresh the window by moving it), close the program by pressing the X on the right-top of the window. Be careful! This will not end the whole „Guided Tour“! The „Tour guide“ must be completely executed. If you are experiencing problems about this, see the section 6.7 (I can´t start/stop the „Tour guide“).

2.5 Regular termination of the „Guided tour“

After you have completed the „Guided tour“ you have to close the application and turn the robot off. Please follow these steps:

1. If the navigation was successful, stop the program and close it as indicated in the section 2.4 (Interruption of the navigation software at any moment).
2. If the connection to the robot or to the laser are lost, stop the program and close it as indicated in the section 2.4.
3. Turn the laser off.
4. Shut the embedded computer down from the Windows Start menu. This is only possible if you have started the robot using the option **a)** or **c)** from the section 1.1 (Accessing the embedded PC of RobIAS). If you have used the option **b),** you can anyway start the VNC Viewer later, as indicated on the points 3. and 4 of the option **a)** in the section 1.1.
5. Wait approximately one minute, so that windows is successfully closed and then toggle the interruptor with the message „Main Power Charge“ on the left lower side of the robot to off.
6. The software and the robot are now completely off.

3 Control Elements

3.1 Navigation Software

Our navigation software offers actually few control elements, because its main function is the *autonomous* navigation through the Institute. The interaction is usually made with the „Tour guide“ developed by Kristian Dencovski (WSA 1884).

3.1.1 Programs which can be started

There are only three programs that you can start:

a) GuidedTour.exe

This program is in the folder *C:\Exchange\Projekte\Robias\Saphira\bin* of the robot, and contains the navigation software, integrated under the GUI of Saphira. You could use therefore the controls from Saphira, but you will not need them, because everything is made automatically by the software. The possible interactions with this Saphira GUI which you could use are:

- By pressing the key „q“ after clicking on the main window (where the robot is), the navigation is stopped and the program ready to be closed.
- By pressing the X button of the window, you close the application. This can be done only if you see the message „*You can now close the application*“ on the main window (you may need refresh the window by moving it, to see this message), after ending the guided tour or pressing the key „q“ as indicated before. If you don´t observe this rule, the program will almost surely stay resident on the memory and will cause the next executions of the program not to work. If you have to close the application before you have seen this message, please consult the section 6.5 (The program can not be closed).
- You may want to use other extra options from Saphira, like View or enable/disable the motors. These options are not explained here, because they will not be needed. However, if you are interested in these options, please consult the Saphira documentation for further details.

b) Simulator-GoTo.exe / Simulator-Resume.exe

These programs are in the folder *C:\Program Files\ActiveX* of the robot, and simulate the functions which are usually assumed by the „Tour guide“: specify the target to be visited and to resume the navigation in case there was an error:

- **Simulator-GoTo.exe**: type the number of the station to be reached and then press return. The window will stay open during the navigation, and will be automatically closed when the station is reached or there is an error. Type the number 0 if you want to conclude the Guided Tour.
- **Simulator-Resume.exe**: when there is an error during the navigation, you can either specify again the station number using the program *Simulator-GoTo.exe* or execute the program *Simulator-Resume.exe.* In the last case, if you type the number 1 and then return, the navigation is resumed from the point where the robot had to be stopped. If you type the number 0, the navigation is aborted and the Guided Tour concluded.

3.1.2 Graphical representation

The typical aspect of the navigation software (under Saphira) is shown in the following figure:

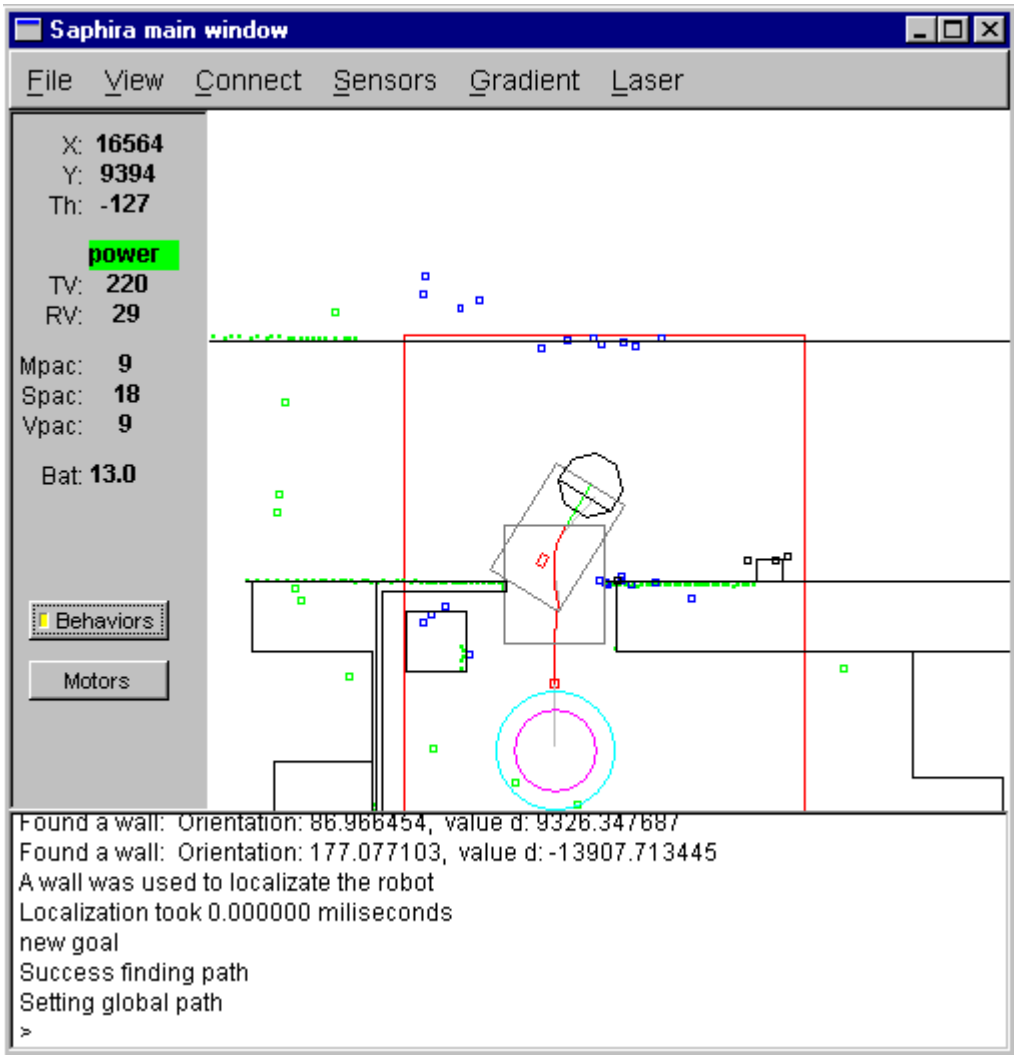


Figure 7: Aspect of the navigation software under Saphira

The following important elements are present in this figure:

1. The **robot**, the black circle in the middle. The line contained in it shows the orientation.
2. The **known world**, with black lines.
3. The **sonar measures**, with green, blue and black big dots.
4. The **laser measures**, with numerous little green dots.
5. The **planned trajectory**, with a red line.
6. The cyan circle is centered on the next intermediate point to be reached. When the robot reaches this circle, it considers that the intermediate point was reached.
7. The magenta circle centered on the next intermediate point to be reached indicates the place which has to be obstacle free so that the robot doesn´t consider this point as blocked.
8. The box on the door indicates the free place to cross this door.
9. The box centered on the robot shows the zone where the sonars are checked.
10. The program „debug information“ on the lower window.

3.2 Batteries

This is a very important aspect of the robot. Without the batteries properly charged, the robot will not travel correctly or could even no longer work. Therefore you have to observe the following points related to the batteries:

1. To save energy, please turn the laser off when you are not running the navigation software. The laser needs a lot of power, and will discharge your batteries very fast.
2. While you are using the computer to install the program, copy files, etc, and not for the navigation, you can leave the power transformer plugged in to the robot, so that the batteries don´t run out.
3. The robot needs a set of 3 batteries to run. There are 2 sets available, and you can charge one of them while using the other. To do this, simply connect the power transformer to the Charge Cube (this cube has 3 inputs, connect the transformer to any of these inputs). The process will usually take some hours. You can check if the batteries are completely charged with a voltmeter: 13.1 V is the normal final voltage.
4. Please check regularly the batteries voltage of the robot; a value smaller than 12 indicates that the batteries must be soon charged or the robot connected to the power net. Otherwise, you can may get problems (typically, you lose the connection with the laser and the navigation software doesn´t work anymore: see section 6.6: The laser connection is lost).

4 Functions

The purpose of the navigation software is an autonomous navigation from the current point to the points determined by the “Tour guide” (or by its simulators). Since this navigation is made in a completely autonomous way, the program can not actually be “used” during this process, but only started and occasionally, stopped and closed. The interaction with the user is a task of the “Tour guide”. For further details about this “Tour guide” please read the User Manual of Kristian Dencovski (WSA 1884, in German).

5 Execution of special Operations

In case the environment where the robot travels changes or the stations to be visited are modified, then you have to edit the following two files: *Ias.wld* and *interpoints.txt*, located in the folder *C:\Exchange\Projekte\RobIAS\Saphira\bin* of the robot. These files contain the information that the robot must know to successfully travel through the Institute.

5.1 The file *Ias.wld*

This file contains the world (in words of the Saphira creators) which can be used by the robot. That means, the robot will use both the information from its range devices and the information from the loaded world.

5.1.1 Syntax of the file

The file has the following general syntax:

```
; Title and comments

width [width]
height [height]

0 0 0 [height]            ; World fronteers
0 0 [width]0
[width] [height]0 [height]
[width] [height] [width]0

; comments
[x1] [y1] [x2] [y2]  ;comments

position [xpos] [ypos] [angle]
```

The words in brackets are numbers, and the words without brackets are commands. We explain this syntax:

- `[width]` and `[height]` are the width and height of the world (place where the robot can move)
- The line `[x1] [y1] [x2] [y2]` is used to define a line from the position (x1,y1) to (x2,y2). So, a square needs 4 such lines. Please note that all the values have to be positive (this is not a limitation of Saphira, we but have chosen in this way for the navigation software).
- The line `position [xpos] [ypos] [angle]` is used only once to indicate the initial position and orientation of the robot in this world. This is also the last line of the file.

Example: (some extracted lines from original *Ias*.wld)

```
;;; IAS world

width 37200
height 30000

   0 0 0 30000            ; World fronteers
   0 0 37200 0
   37200 30000 0 30000
   37200 30000 37200 0
```

```
;; Prozesslabor 2 walls
   0 8400 1300 8400
   2300 8400 12100 8400
   13100 8400 14400 8400
   14400 8400 14400 0

;; Supporting pillars in PC-Labor 2
   14700 8100 14700 7500
   14700 7500 15300 7500
   15300 7500 15300 8100
   15300 8100 14700 8100

;; tables in Prozesslabor 2
   0 5900 600 5900
   600 5900 600 5300
   600 5300 0 5300

;; Starting position
position 21600 9600 180
```

5.1.2 Elements to be added to the world file

If the environment of the robot is changed, or you want to make a new file, which information must be added to the world file? The following things:

- All the walls have to be added.
- Cupboards, they are like walls.
- The tables, because the robot can not see them and only with the world file it can see them.
- The doors (open if they can be crossed, closed if the robot does not have to go in the corresponding room).
- Supporting pillars.
- Fire extinguishers.
- Generally, all the objects which the robot could not see and have always the same position. An exact modelling is not required: usually suffices defining a box around the obstacle which has to be avoided.

Which elements DO NOT have to be added to the world?

- Chairs: their positions vary a lot and can not predicted. In the case you are having a narrow space for the robot and in consequence, problems with the chairs, you can define a box around the place where the chair is usually located. Otherwise, the recommended option is not to include these chairs in the world and simply make them away before the “Guided tour” is started.
- Innecesary details. You do not need a very detailed world, but clearly defines which zones are forbidden for the robot, so that it does not try to travel to these zones.

5.2 The file *interpoints.txt*

This file contains extra information needed for the correct navigation: the start point, the stations to be reached, the intermediate points which can be used by the robot, the connections between these points and the localization information. Please note that the file *interpoints.txt* already contains the explanations about how must be written.

5.2.1 General syntax

The general syntax of the file *interpoints.txt* is:

```
; comments
COMMAND param1 param2 ... [paramX] [paramX+1] ...  ;coments
```

The meaning of this syntax and other important issues:

- The information in a line after the symbol ";", at the beginning or in the middle, is ignored by the navigation software, so you can write your comments there.
- The possible commands are the number of the point, the word `EDGE` or the word `CORNER`.
- The parameters without brackets are mandatory; the parameters with brackets are optional.
- The navigation software was not optimized to recognize syntax errors, because the aim of the program was not to build a compiler. However, some common mistakes and the line where they appear reported (Please see Section 6.3 for further details).

5.2.2 Definition of start, end and intermediate points

The general definition of a point for the robot is:

`Position_number X-coord Y-coord [final_angle] [localization_info]`

- **`Position_number`**: number of the position. It has to coincide with this in the Tour Guide. The numbers use the following convention:

  ➔ The number 0 and the negative ones are reserved for the program and can not be used
  ➔ The number 1 is used for the initial position
  ➔ The numbers 2 to 31 are used for final stations
  ➔ The numbers greater than 31 (32 and so on) are used for the user-defined intermediate points

- **`X-coord`**: X-coordinate of the referred position (all the coordinates have to be positive).
- **`Y-coord`**: Y-coordinate of the referred position (all the coordinates have to be positive).
- **`final_angle`**: final orientation of the robot in a desired final position (in degrees 0-360). Mandatory for start and final points (if no written, the value 0 is taken by default), and useless for intermediate points, but in this last case must be written if the next parameter is written too.
- **`Localization_info`**: (write final_angle if you write this param).This value can be:
  ➔ **`CORNER`**: when in this point we can use a corner to orientate us. No additional parameters are needed, the corners are compared automatically).
  ➔ **`CORRIDOR`**: when in this point we can use a corridor to orientate us. We have to specify how wide is the corridor, and if it is `HORIZONTAL` or `VERTICAL` (the

two only possibilities). *VERY IMPORTANT*: the intermediate point using a corridor to localizate the robot must be centered in the important coordinate. That means, there must be the same distance between the intermediate points and both sides of the corridor.

➔ **`WALL`**: when in this point we can use a wall to orientate us. We have to specify the x or y coordinate of the wall and if it is `HORIZONTAL` or `VERTICAL` (the two only possibilities).

Example: (extracted from the actual *interpoints.txt*)

```
;; Start and final points
1  21600  9600 180 CORRIDOR  2400 HORIZONTAL
2  18700  6700 180 WALL      7700 HORIZONTAL
3  16300  2500  90 WALL     14400 VERTICAL
6   1700  1200  90
;; Intermediate points
32 18900  9600   0 CORRIDOR  2400 HORIZONTAL
33 16200  9600   0 WALL     10800 HORIZONTAL
38  1800  6700   0 CORNER
44 12600  5700   0 WALL     14400 VERTICAL
```

5.2.3 Definition of edges

The edges represent the possible connections between the points defined before. The syntax of an edge is:

```
EDGE Vertex-from Vertex-to [Relative_cost] [DOOR] [X-coord]
     [Y-coord] [Orientation]
```

- **`Vertex-from`**: Start (or end) vertex of the edge. Note that the edges are bidirectional, and therefore, the choice of which vertex is "`Vertex-from`" and wich vertex is "`Vertex-to`" is completely arbitrary
- **`Vertex-to`**: End (or start) vertex of the edge (see comment before).
- **`Relative_cost`**: Is used to make the robot prefer a path. By default, if no specified, the relative cost of an edge is 1.0. Use greater numbers to indicate the robot, that this way is more difficult for the navigation. I recommend 1.1 or 1.2 (maximum 1.5!) for these costs. With very high costs the robot will travel very long ways.
- **`DOOR`**: Write this word if the edge crosses a door. If this parameter is specified, the relative cost must be also specified (write 1.0 or the value you want in this case). If this parameter is not specified, we assume that the edge doesn´t cross a door.
- **`X-coord`**: X-coordinate of the center of the door. Needed if we use `DOOR` parameter.
- **`Y-coord`**: Y-coordinate of the center of the door. Needed if we use `DOOR` parameter.
- **`Orientation`**: `HORIZONTAL` or `VERTICAL`, is the door orientation. Needed if we use the DOOR parameter.

Example: (extracted from the actual *interpoints.txt*)

```
EDGE  1 32
EDGE 32 33
EDGE 33 43 1.0 DOOR 16200  8350 HORIZONTAL
EDGE 43  2 1.2
```

5.2.4 Definition of corners

The corners are used to localizate the robot. They can completely localizate the robot, and therefore you must be careful by choosing them. Their syntax is:

`CORNER X-Coord Y-Coord Orientation Type`

- **`X-Coord`**: X-Coordinate from the the intersection point of the walls which determine the corner.
- **`Y-Coord`**: Y-Coordinate from the the intersection point of the walls which determine the corner.
- **`Orientation`**: For the orientation the following criteria shown in the figure is used (for both the convex and concave corners):

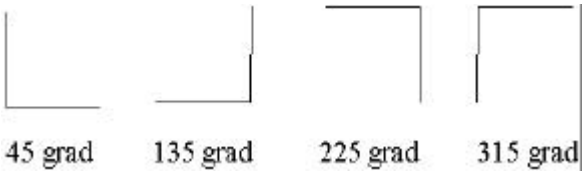


Figure 8: Orientation criteria for the corners

- **`Type`**: We have to indicate if the corner is `CONCAVE` or `CONVEX`.

5.2.5 General recommendations to elaborate the file *interpoints.txt*

The information contained in the file *interpoints.txt* has a big influence on the navigation of the robot. If this information is properly chosen, then the robot will travel properly and will be easily localizated. But if this information is not properly made, then the robot will have a lot of problems for the navigation and localization. Therefore, you should follow these general tips:

1. The number of the start and final points are defined firstly in the "Tour guide" and their number in the *interpoints.txt*, and their position and orientation must coincide with these defined there.
2. The choice of the intermediate points to help the robot to navigate must follow these rules:
   - ➔ They should not be very separated (more than 4 or 5 meters is not a good idea at all), but they do not have to be very together. I don´t recommend in any case that they are closer than 1 meter, and this limit should be used only for particular situations, where the localization is difficult or the robot has a tendence to get lost
   - ➔ When the robot has to cross a door, you have to place an intermediate point on both sides of the door, with the corresponding coordinate centered relating to the door.
   - ➔ When the intermediate point is placed in a corridor, you must place it centered, so that you have the same distance to both walls of the corridor.
3. Each intermediate point should have associated a localization method, if possible. The preferable method are the corners, because they allow to completely localizate the robot. Then, you will prefer to use corridors and walls. Corridors are usually more exact, but the walls are more versatile, because you need only one wall and not two like in the corridors. The good combination of both corridors and walls would be a great solution.

- ➔ The best corners for the localization are the corners in the corridors, because they are very big and reliable. The cupboards offer also very good corners, and the supporting pillars (preferably the big ones) too. Other possibility to allow the localization in difficult situations is using the drawers of the tables, but it will depend on the chairs present if this method will work fine or not. Don´t forget the concave corners let by walls in the closed rooms, they can also be very useful.
- ➔ Use corridors when you have a long way without doors to be crossed. The corresponding intermediate point must be then centered on this corridor.
- ➔ Walls are strongly recommended when there are cupboards, and for the intermediate points just before a door. That means, from the two intermediate points associated to a door, use a wall to localizate for this one on the corridor side: it will be useful in both directions of navigation. They can be also used anywhere where there is a straight and big surface (cupboards and similar big objects).

4. Try that the edges connect intermediate points have “direct visibility”. The robot will navigate with edges without direct visibility too, but this navigation can be more easily be affected by the obstacles on the way.
5. Make alternative ways for the robot, in case the planned way is blocked.
6. Be specially careful choosing the intermediate points and localization methods in the rooms with a lot of furniture. Without extra help, the robot will probably get lost or collide a chair or other “invisible” obstacles (the corners from drawers of the tables are a very good help).

6 Error Removal

Here we show a list nown typical errors and their possible solutions:

6.1 I can´t access the embedded PC of the robot with VNC Viewer

If when typing the server ias198 with the VNC Viewer you receive the message shown in the following figure:

Figure 9: Error using the VNC Viewer

- **Solution 1**: Be sure that the robot is turned on.
- **Solution 2**: Be sure that the VNC Server is correctly typed.
- **Solution 3**: Be sure that the Wireless LAN modem is connected and not very far away from the robot.

6.2 The laser does not work

If you can not see the green dots of the laser in the navigation software window:

- **Solution 1**: Be sure that the laser is turned on (interruptor on the rear side). You should see a non-blinking green LED on the front of the laser.
- **Solution 2**: If you have started with the mouse plugged in, restart the embedded PC.

6.3 There is an error with the intermediate points

If when starting the navigation program you receive a message like this one shown in the figure:

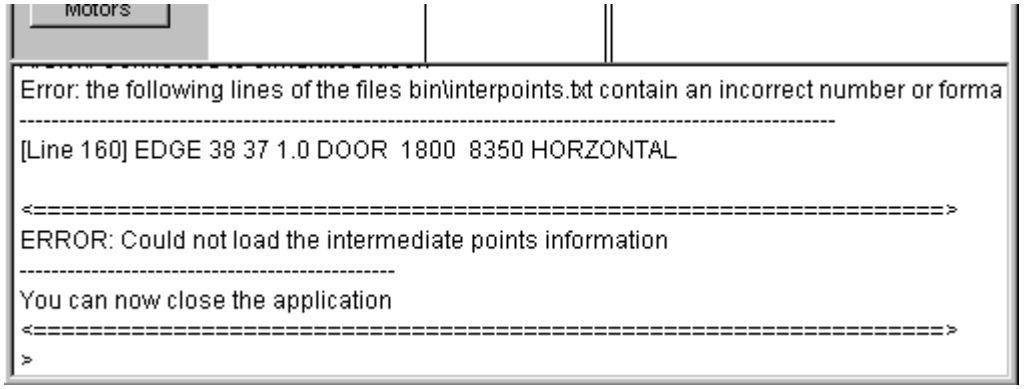


Figure 10: Error loading the intermediate points

- **Solution:** You have wrong typed the parameters syntax in the file *interpoints.txt*. In the window or in the file *errors.dat* contained in the folder *C:\Exchange\Projekte\RobIAS\Saphira\bin* of the robot, you will find which lines are giving problems. Please read carefully these lines, and using the information from the section 5, correct the mistakes.

6.4 The path to the target is not found at the beginning or the program crashes

If your program is not finding a path to to the target at the beginning, when it should have no problem (not to mix with path not found due to a way blocked), or it crashes without apparent reason:

- **Solution:** You have almost surely made a mistake in the file *interpoints.txt*. This can be a logical mistake (for example, you haven´t connected two points) or you may have typed something wrong (remember that the program does not recognize all the possible errors which could appear). Since there is not an error report, you must read very carefully the whole file to find the mistakes.

6.5 The Saphira window can not be closed

If you have tried to close the Saphira window not following the recommended method (please see section 2.4: Interruption of the navigation software at any moment), you will probably have this problem: the Saphira window can not be closed and the program stays resident in the memory.

- **Solution:** Since you will be using the VNC Viewer to see the Saphira window, if you press the combination Ctrl+Alt+Del you will not see the Task Manager of the embedded PC, but this one from the computer you are using to remotely access the robot. You have to plug in a keyboard (and optionally a monitor) to the robot, and with this keyboard, you can press the combination Ctrl+Alt+Del to close the program that is not responding. At this point, you can go on with the step 3 in the section 2.5 (Ending the "Guided tour").

6.6 The laser connection is lost

If you see this message on the Saphira window during the navigation, that means that the battery has a low voltage.

- **Solution:** Stop the navigation software (see the section 2.4: Interruption of the navigation software at any moment), turn the laser off, charge or replace the batteries and then start again the "Guided tour".

6.7 The "Tour guide" does not react

If you are trying to start again the "Tour guide" after an error, and the "Tour guide" does not end or react at all.

- **Solution:** You have to close the window where you have the "Tour guide", close the navigation software (if open) and then restart the PC of the robot and then restart the whole process again.

6.8 The robot gets lost very easily

If during the navigation the robot has a lot of problems finding the right place, or sees blocked ways where there are no obstacles.

- **Solution:** You may have defined a *interpoints.txt* file with errors or with very little localization information. Please think, that the more complicate is the environment, the more help the robot needs. That means more localization references.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Test Protocol

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Test Protocol	1.0	Serrano López	14.04.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-test-protocol-v10.doc	7	28/05/03 17:48	sw-test-protocol.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	09.04.03	in progress	Creation
1.0	Serrano López	Jo	14.04.03	submitted	
1.0	Serrano López	Jo	14.04.03	accepted	

0 Table of Contents

1 Communication with the tour guide: *GoTo()*

1.1 Reference to the Test Specification

In the following the test case “Test Case: Communication with the tour guide: GoTo()” (chapter 4.1) will be performed.

1.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”.

1.3 Results

The test was successful: the sequence of intermediate points is calculated and the robot moves in the desired direction.

2 Navigation without obstacles and door opened

2.1 Reference to the Test Specification

In the following the test case “Test Case: Navigation without obstacles and door opened” (chapter 4.2) will be performed.

2.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”. Wait until the target is reached and observe what happens.

2.3 Results

The test was successful: the robot navigates without collisions through the planned path, corrects regularly the position and reaches the desired target with enough precision.

2.4 Discussion

The efficiency of the navigation is strongly dependent of how good were defined the intermediate points. The robot calculates the path without errors and localizates itself if possible without errors too. If the localization could not be performed, that is due to the obstacles which hide the localization references and the absence of enough localization references.
I would like to remark, that the robot can not navigate without references. If the references are systematically hidden or blocked, the robot will get lost and report inexisting errors. Solution: good preparation of the “Guided tour” and enough localization references.

3 Fixed obstacles avoiding

3.1 Reference to the Test Specification

In the following the test case “Test Case: Fixed obstacles avoiding” (chapter 4.3) will be performed.

3.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”. Wait until the obstacle is reached and observe what happens.

3.3 Results

The test was successful: the robot avoids the fixed obstacle which can be passed by and reaches the desired target with enough precision.

3.4 Discussion

The limititations to the efficiency of this test are the same than for the chapter 2.

4 Mobile obstacles avoiding

4.1 Reference to the Test Specification

In the following the test case “Test Case: Mobile obstacles avoiding” (chapter 4.4) will be performed.

4.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”. Wait until the obstacle is reached and observe what happens.

4.3 Results

The test was successful: the robot avoids the mobile obstacle which can be passed by and reaches the desired target with enough precision.

4.4 Discussion

The limititations to the efficiency of this test are the same than for the chapter 2.

5 Closed doors detection

5.1 Reference to the Test Specification

In the following the test case “Test Case: Closed doors detection” (chapter 4.5) will be performed.

5.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”. Wait until the closed door is reached and observe what happens.

5.3 Results

The test was successful: the robot detects the closed door. If other path is available, it uses it, and if not, reports the error to the “Tour guide”.

6 Way blocked detection

6.1 Reference to the Test Specification

In the following the test case “Test Case: Way blocked detection” (chapter 4.6) will be performed.

6.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”. Wait until the point the way is blocked is reached and observe what happens.

6.3 Results

The test was successful: the robot detects the blocked way. If other path is available, it uses it, and if not, reports the error to the “Tour guide”.

7 Communication with the tour guide: *Resume()*

7.1 Reference to the Test Specification

In the following the test case “Test Case: Communication with the tour guide: Resume()” (chapter 4.7) will be performed.

7.2 Procedure of Testing

1. Start the robot, prepare it and start the navigation software.
2. Start the program *Simulator-GoTo.exe* and enter the number of a station. Force and error (block the way or close a door) and then execute the program *Simulator-Resume.exe* and enter the number 1. The navigation will be resumed if the error is no longer present. If you start the “Tour guide”, all will be automatically made: simply wait.

7.3 Results

The test was successful: the navigation is resumed after the error.

8 Real navigation for the tour guide

8.1 Reference to the Test Specification

In the following the test case “Test Case: Real navigation for the tour guide” (chapter 4.8) will be performed.

8.2 Procedure of Testing

3. Start the robot, prepare it and start the navigation software.
4. Start the program *Simulator-GoTo.exe* and enter the number of a station, or start the “Tour guide”. Wait until the target is reached and observe what happens.

8.3 Results

The test was successful: the robot navigates without collisions through the planned path, corrects regularly the position and reaches the desired target with enough precision, avoiding the obstacles which do not block the way and detecting if the doors are closed or the way blocked.

8.4 Discussion

The efficiency of the navigation is strongly dependent of how good were defined the intermediate points (as indicated in the chapter 2). Aditionally, the robot can react to normal blocking situations, but if you try to find exceptional situations for the robot, it is very possible that it does not find the way any more, because the software was developed to run under certain limits of “problems”.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Acceptance Review Protocol

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Acceptance Review Protocol	1.0	Serrano López	14.04.03	accepted

Filename:	Pages:	Print Date:	Template:
sw-acceptance-review-v10.doc	6	28/05/03 17:49	sw-acceptance-review.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1				in progress	Creation
1.0	Serrano López	Jo	14.04.03	accepted	

Place, Date:	Stuttgart, 14.04.2003	
Participants:	Javier Serrano López, Jost	

CHECKLIST System

Checkpoints	**Actions/ Comments**
1) Does the system behave in accordance with the functional request and expectation?	Yes
2) Does the system fulfil the placed non-functional requirements?	Yes, all
3) Does the system fulfil the quality requirements?	Yes
4) Is the application created in accordance with the GUI concept and common guidelines?	No GUI was developed.
5) Does the installation work as described in the user manual ?	Yes
6) Do errors occur during the operation?	No (See user manual for known problems)

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted Yes	Accepted no
14.04.03	14.04.03	Jo	-	X	

Comments:

CHECKLIST Test Protocol

Checkpoints	**Actions/ Comments**
1) Are all test cases specified in the test specification executed and documented?	Yes
2) Are meaningful testing procedures applied?	Yes
3) Are the prerequisites of the tests, the environment and steps of the test specified?	Yes
4) Are all significant data of the test documented and saved?	As far as possible (Real environment in IAS can not be saved)
5) Is the total result of the test sufficient for the acceptance?	Yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
14.04.03	1.0 / 14.04.03	Jo	-	X	

Comments:

CHECKLIST User Manual

Checkpoints	**Actions/ Comments**
1) Is the user manual clear and well structured?	Yes
2) Is the installation described understandably?	Yes
3) Are all control elements described?	No control elements, but command-line options and configuration files
4) Are all functions described?	Yes
5) Are all important operations sufficiently described?	Yes
6) Are well-known errors, notes to the elimination of errors and function limitations specified?	Yes

Acceptance of the Document

Date of Inspection	Document Version/Date	Inspector	Items to be revised	Accepted yes	Accepted no
14.04.03	1.0/09.04.03	Jo	-	x	

Comments:

CHECKLIST Project Progress

Checkpoints	**Actions/ Comments**
1) Have all items of the last review, which had to be revised, been completed?	No open items
2) Is the planned timetable of the project (Bar Chart) kept?	The last two reviews are delayed, but the deadline will be kept.
3) Is the inter-office slip up to date?	Yes

Acceptance

Date of Inspection	Inspector	Items to be revised	Accepted	
			yes	no
14.04.03	Jo	-	x	

Comments:

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Project Final Report

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Project Final Report	1.0	Serrano López	15.04.03	accepted

Filename:	Pages:	Print Date:	Template:
project-final-report-v10.doc	4	28/05/03 17:50	project-final-report.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	08.04.03	in progress	Creation
1.0	Serrano López	Jo	15.04.03	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

0 Table of Contents

1 Summary

For this project, a navigation and control software of the robot RobIAS for the guided tour through the IAS Institute was developed. The initial step was to learn about some robotic topics and specially the Aria classes and the Saphira program developed by Activmedia Robotics, the robot manufacturer.
After I had acquired enough knowledge about the features of this development tool, I began to work with the IAS Process Model, which was completely new for me. I also tried in collaboration with Kristian Dencovski to clearly define the interface between my navigation software and his tour guide. This was not possible in a complete way in the first moments, because both projects were in their initial stage, but we advanced a lot in this direction.
As a result of these meetings and the opinion of our tutor, we decided to use the ActiveX-ATL COM technology as solution for our communication problems.
After this pause, I resumed my learning in ActiveX and I occupied my time with the System Architecture and Software Components documents. This was not a very costly process because while learning Saphira and Aria I had realized some test programs that helped me a lot to define the final architecture. The implementation and integration of this architecture and components were realized simultaneously, because the simulator provided by Activmedia Robotics has a really excellent quality: the things that worked on the simulator worked usually on the robot, so I could integrate and test them on the robot at the same time I was developing them, without using the robot every day and that made me save a lot of time.
When the program was finished, I could write all the required documents, so that I could at the end submit my Master Thesis.

The result of my project is a software which controls the navigation of the mobile robot RobIAS through the IAS Institute, from a predefined start point to every desired and also predefined final point in the Institute. This all is realized using the reduced information provided by the sensors integrated in the robot (sonar, laser and bumpers), and it is possible due to the efficient and fast algorithms developed and the stored information about the environment, which can be easily adapted in case this environment changes, or even used for other office-based completely new environment. That means, the software was not optimized for the particular case of the IAS Institute, but rather can be used everywhere according to this office-based construction.
Methods to avoid collisions, localize the robot, calculate minimal-cost paths and alternative paths were developed in order to guarantee the navigation reliability (as far as this is possible, according to the perception-hardware limitations).
The determination of the final points to be reached and the spoken explanations corresponding to the points reached are part of the work from Kristian Dencovski for his Wahlstudienarbeit number 1884. Both parts, my Navigation and Control Software, and his Tour Guide, are perfectly synchronized and were designed to work together.

2 Experiences

First, I would like to indicate my really good personal impressions in these six and half months that I have been occupied with my Master Thesis.
My project tutor, Pascal Jost, has helped me always a lot (even when he had no time due to his tight agenda), and was not only friendly and helpful, but he had also good knowledges to efficiently support my work (no difference if that meant brazing or a program debugging).
For Kristian Dencovski I have only good words: our work together was really productive and satisfactory, but above all, he was one of the people who helped more for my understanding of the german culture and I have won a very good friend.
The colleagues who work in the IAS Institute were very friendly and helpful, too.
At last I would like to thank Mr. Prof. Göhner, because he gave me the opportunity to do my Master Thesis in the IAS Institute, even when I was not registered as a usual student at the University Stuttgart, but exchange program student.

Relative to the IAS Process Model I can say that it helped me a lot with my work. At the beginning I thought it was only a charge, but later I realized how wrong I was, because it organizes the work really good and forces to do everything in a more systematic way. That meant at the end time saving, work efficiency and a high quality final product.

I have been interested some years in the robotic field. This project gave me the opportunity to work with a real and proffesional robot and not only simulations like before, and the resulting experience was really good. My programming skills were improved too, and in general I feel more capable to work in the industry after all these knowledges acquired here in IAS.

3 Problems

Of course, many problems appeared during my Master Thesis, and sometimes they constituted a challenge for my capabilities.
When I started I was really lost with the IAS Process Modell, because it was not usual for me to plan everything in such a systematic way. But later I saw that this effort was worthwhile, and although in many occasions it seems that this Model is only a time waste, the reality is that at the end the profits compensate the extra work which has to be realized at the beginning.
The ActiveX gave me a lot of problems, because I could not find a simple solution to the communication problem without using files. It is this a very complex field which would request more time than I had to solve this problem.
I experienced also problems with the complexity of Aria and Saphira. At the beginning I estimated that the program was relatively easy to be managed, but the more I used it, the more things I discovered and that I had no idea about its existence. The available literature is really huge and I missed some important concepts in the initial contact. Later, during the implementation of my program, these concepts arised as problems and I had to reestructure my ideas in some occasions. Now I can say, that a more detailed lecture of this available literature could have helped me a lot.
Some hardware problems relative to communication ports in the robot were present too, and they persisted until my tutor found their source and repaired it.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Terminology

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Terminology	1.0	Serrano López	15.04.03	accepted

Filename:	Pages:	Print Date:	Template:
terminology-v10.doc	4	28/05/03 17:50	terminology.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	10.02.03	in progress	Creation
1.0	Serrano López	Jo	15.04.03	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

Terminology

ActiveX Microsoft technology that allows the communication between different software components programmed in different programming languages.

ARIA Object-oriented, robot control applications-programming interface for ActivMedia Robotics' line of intelligent mobile robots.

Embedded-PC Computer integrated in the system (in RobIAS in our case)

Gradient navigation Realtime path planner based on the gradient method from Kurt Konolige. It determines optimal paths for the robot, in real time, which can take into account both local obstacles, sensed by sonars and/or laser range-finder devices; and global map information such as the location of walls and other structural obstacles.

Guided tour Addition of both processes *Tour guide* and *Navigation software*, which were independently developed to work together resulting in an *Guided tour*.

Intermediate point Reference point used for the navigation; the robot navigates using a sequence of such intermediate points, so that the navigation problem is “reduced” to the navigation between two intermediate points (which are not very far away)

Laser range-finder Device used by the robot RobIAS which gives a detailed information of the environment: a matrix with readings in a range of 180 degrees.

Localization Process to find the actual robot´s position and orientation, needed because the estimation of these values with odometry has errors.

Navigation software This is the result of my Master Thesis: a Control and Navigation software to move the robot from the start point to the desired final points avoiding obstacles through an efficient path.

Odometry Method to estimate the robot´s position and orientation, by reading the values of the wheel encoders and using this information and the kinematic model of the robot.

Path	Way to be covered by the robot between two intermediate points. It is the real way followed by the robot, including the obstacle avoidance.
RobIAS	Mobile robot from ActivMedia, based on the Pioneer 2-DXE with Peoplebot Extension
Saphira	Saphira is a robotics application development environment and the Saphira library is a set of routines for building clients, which integrates a number of useful functions for sending commands to the server, gathering information from the robot's sensors, and packaging them for display in a graphical window-based user interface. In addition, Saphira supports higher-level functions for robot control and sensor interpretation, including a Gradient navigation system.
Sonar	Device used by the robot RobIAS which gives poor information about the environment but detects fast mobile obstacles which can not be detected by the Laser range-finder
Tour guide	Part of the whole *Guided tour* project which manages the Guided tour, determining the stations to be visited and gives spoken explanations to these stations.
Trajectory	Global way planned by the robot as result of connecting all the intermediate points that have to be reached. The path should follow the trajectory as much as possible.

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Abbreviations

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Abbreviations	1.0	Serrano López	15.04.03	accepted

Filename:	Pages:	Print Date:	Template:
abbrevations-v10.doc	3	28/05/03 17:51	abbrevations.dot

Document Version Management

Version	Author	QS	Date	Status	Changes
0.1	Serrano López	Jo	10.02.03	in progress	Creation
1.0	Serrano López	Jo	15.04.03	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

Abbreviations

COM	**C**omponent **O**bject **M**odel
DLL	**D**ynamic **L**ink **L**ibrary
GUI	**G**raphic **U**ser **I**nterface
IAS	**I**nstitut für **A**utomatisierungs- und **S**oftwaretechnik (**I**nstitute of Industrial **A**utomation and **S**oftware Engineering)
MT	**M**aster **T**hesis
WLAN	**W**ireless **L**ocal **A**rea **N**etwork. Cordless radio network technology
WSA	**W**ahl**s**tudien**a**rbeit (voluntary student research project)

University of Stuttgart
Institute of Industrial Automation and Software Engineering
Prof. Dr.-Ing. Dr. h. c. P. Göhner

Receivers
Prof. Göhner
Jost
Konnertz
Serrano López

Master Thesis

Development of a Navigation and Control Software for the Mobile Robot RobIAS

Javier Serrano López

Literature

Thesis No:	Type:	Class of Process Model:	Student:	Tutor:	Begin:	End:
1883	MT	Model for Software Development	Serrano López	Jo/Kn	08.10.2002	25.04.2003

Document:	Version:	Author:	Date:	Status:
Literature	1.0	Serrano López	15.04.03	accepted

Filename:	Pages:	Print Date:	Template:
literature-v10.doc	3	28/05/03 17:52	literature.dot

Document Version Management

Version	Author	QA	Date	Status	Changes
0.1	Serrano López	Jo	10.02.03	in progress	Creation
1.0	Serrano López	Jo	15.04.03	submitted	
1.0	Serrano López	Jo	15.04.03	accepted	

Literature

[1] **Sirotin, Victor:** *ActiveX/DCOM-Programmierung mit Visual C++ 6,* Bonn; Reading, Mass. [u.a.] : Addison-Wesley-Longman, 1999.

[2] **ActivMedia Robotics:** http://robots.activmedia.com/

[3] **ActivMedia Robotics:** *ARIA and Saphira Manuals and References,* ARIA and Saphira installation

[4] **Konolige, Kurt**: *A gradient method for realtime robot control*, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[5] **Morris, John:** *Data Structures and Algorithms: Dijkstra´s Algorithm,* http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html

[6] **Göhner, Peter :** *Skript zur Vorlesung Softwaretechnik I*, IAS, Stuttgart, 2002

[7] Microsoft: ***MSDN Library,*** http://msdn.microsoft.com/library/

[8] IAS Regulations Manuel Version 3.0

[9] **Leo (Link Everything Online) Dictionary**, Informatik der Technischen Universität München, http://dict.leo.org/?lang=en

[10] **IAS-Homepage:** www.ias.uni-stuttgart.de

[11] **Daniela Hauck**, *Generierung einer lokalen Karte durch Fusion von zeitlich aufeinanderfolgenden Messdaten eines Laserscanners*, Diplomarbeit Nr.: 1800, IAS, Stuttgart, 2001

[12] **Kristian Dencovski,** *Development of a Navigation and Control Software for the Mobile Robot RobIAS,* Wahlstudienarbeit Nr.: 1884, IAS, Stuttgart, 2003