UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Informática

Proyecto Fin de Carrera

Técnicas visuales de reconocimiento de objetos en robots móviles.

Autor: Ramón Cintas Peña.

Fdo:

Director: Pablo Bustos García del Castro.

Fdo:

Tribunal Calificador

Presidente:Moreno del Pozo, José
Fdo:

Secretaria: Bachiller Burgos, Pilar
Fdo:

Vocal: Nuñez Trujillo, Pedro
Fdo:

CALIFICACIÓN:
FECHA:

Resumen

El empleo de la robótica en los entornos industriales es una práctica generalizada.

La industrialización y automatización de los almacenes es un recurso cada vez más utilizado. Para conseguir estos fines, se ha desarrollado un creciente apoyo tecnológico a las actividades de almacenaje a partir del empleo de equipos cada vez más sofisticados y automatizados.

Este proyecto intenta abordar la problemática del transporte de mercancías dentro de los almacenes desde un punto de vista más sencillo. En concreto, se pretende automatizar una carretilla elevadora de forma que pueda llevar a cabo la tarea de forma autónoma. Para ello se simulará el entorno mediante la plataforma robótica RobEx.

El desarrollo del proyecto se centra en el diseño de técnicas que permitan el reconocimiento visual de un palé, la generación de una trayectoria de aproximación que permita llegar hasta él con la orientación adecuada para, a continuación, llevarlo a una posición de destino.

La identificación se lleva a cabo utilizando la información extraída de los colores de la imagen así como de una serie de transformaciones geométricas que permiten estimar la posición de un objeto partiendo únicamente de esta información. Mientras que la trayectoria se aborda desde un punto de vista matemático.

La memoria del proyecto, indica los problemas encarados y cómo han sido resueltos, muestra los resultados de los experimentos, e introduce RobEx y RoboComp, las plataformas hardware y software usadas.

El código del proyecto fin de carrera se ha incorporado a RoboComp (repositorio de componentes software para robótica). Además, también se puede encontrar en el disco que acompaña a la memoria del proyecto, junto con vídeos y fotografías tomadas durante la fase de experimentación.

Parte I - Índices

Índice de contenido

Índice de figuras

Parte II

1 Capitulo 1: Introducción

La industrialización y automatización de los almacenes es un recurso cada vez más utilizado. Las funciones que se desarrollan en estos entornos pueden englobarse en dos: el almacenaje propiamente dicho y el manejo de cargas.

Para conseguir estos fines se ha desarrollado un creciente apoyo tecnológico a las actividades de almacenaje a partir del empleo de equipos cada vez más sofisticados y automatizados. El control, supervisión y gestión de este tipo de equipos puede suponer una tarea muy compleja [1] . Existen numerosos tipos de robots encargados de colocar y recuperar palés con los que se optimiza el uso del espacio disponible dentro de los almacenes, ganando, además, seguridad y velocidad en el tratamiento de los mismos.

1.1 Estudio del problema

Se ha trabajado mucho sobre la optimización del espacio de los almacenes así como la gestión automática de los mismos. Existen numerosas alternativas en cuanto a maquinaria encargada de izar y colocar los palés en las estanterías, abarcando desde simples elevadores a complejos sistemas automáticos de ubicación, colocación y

recogida que integran diferentes mecanismos robóticos interconectados. Estos sistemas son cada vez más robustos, rápidos e inteligentes. Sin embargo, el transporte de la mercancía hasta estos elevadores no ha avanzado en sintonía con ellos.

Figura 1: Sistemas automatizados de almacenamiento de palés

En la imagen anterior pueden verse un par de ejemplos:

- A la izquierda aparece *"Stratus"* un elevador capaz de colocar y recoger automáticamente palés almacenados en estanterías verticales.
- A la derecha aparece *"Comissioner"*un complejo sistema que incorpora la gestión autónoma y automática de toda una columna de estanterías. Realmente es un almacén automatizado en sí mismo.

Un tema muy poco tratado supone el desplazamiento de dichos palés dentro del centro de elaboración (en el caso de fábricas) o entre el punto de entrada y salida (en almacenes) hasta estas máquinas de "colocación".

La gran mayoría de las soluciones pasan por incorporar cintas automatizadas sobre las que se desplazan estos palés.

Otra alternativa consiste en insertar una serie de raíles a lo largo de todo el almacén y que sea el propio robot "organizador" el que se desplace y coja los palés.

Ambas soluciones presentan una serie de inconvenientes bastante grandes:

- Se trata de soluciones muy costosas ya que hay que incorporar numerosas máquinas de empuje.
- Son muy poco flexibles, los palés han de ser encajados perfectamente en la cinta para obtener un comportamiento correcto, y esto sin contar con modificaciones acerca del tamaño o disposición de los mismos.
- Este tipo de transporte trabaja muy bien en línea recta. La colocación de estos sistemas dentro de entornos con rutas desniveladas o sinuosas aumenta considerablemente el coste y a su vez reduce el rendimiento.
- Conllevan un alto coste en mantenimiento debido a la gran cantidad de piezas móviles que incorporan.

Como ventajas cabría destacar:

- Gran velocidad de procesamiento en entornos no demasiado grandes.
- Son sistemas muy fiables

Estas soluciones son buenas cuando el espacio total a cubrir no resulta demasiado grande, no se preveen modificaciones sobre el almacén o la cantidad de palés a tratar es muy elevada, lo que requiere de un alto rendimiento.

En otro tipo de casos resulta mucho más útil disponer de una máquina pequeña que realice dicho trabajo.

Existen en el mercado diferentes tipos de máquinas para el transporte de palés mucho más sencillas como pueden ser las carretillas elevadores (más comúnmente conocida como toros). La desventaja principal es que requieren de un operador que los maneje.

Figura 2: Carretilla elevadora

La propuesta consiste en simular la automatización de una de estas carretillas. Para ello se utilizará la plataforma robótica Robex que será comentada en el capítulo siguiente.

Si bien las pruebas no se van a realizar en un entorno real, las posibles ventajas de implementar un sistema de estas características son:

- La incorporación de un sistema de movimiento autónomo no supone la pérdida de ningún aspecto del movimiento manual del toro. Bastaría con desconectar el software para que el sistema dejase de funcionar. Esto puede parecer una característica poco interesante, sin embargo, presenta grandes ventajas. En caso de ser necesario el almacén puede ser utilizado de forma totalmente manual, cosa que no podría ocurrir si se implantaran raíles o desplazadoras automáticas.

- En caso de modificar el tamaño de los palés a transportar basta con cambiar la horquilla de anclaje del toro, lo que supone un coste mínimo.

- No es necesario llevar a cabo un mantenimiento especial, basta con continuar con los planes de mantenimiento preestablecidos.

- Proporciona una gran independencia del lugar de trabajo. La carretilla podría utilizarse en innumerables superficies sin necesidad de ningún tipo de adaptación.

- Otra de las grandes ventajas de esta solución es el precio de implantación del sistema. Un toro resulta mucho más barato que un almacén automatizado y, además, no necesita una estructura de soporte específica como suele ocurrir con las máquinas encargadas de izar y colocar los palés.

1.2 Adaptación del problema al entorno del laboratorio

Como plataforma de experimentación se dispone de un robot de la clase Robex (comentado en el capítulo siguiente). Para adaptar las capacidades del robot al problema a tratar se le incorporará un elevador de cargas para hacer las veces del elevador de una carretilla. Como es lógico, los palés con los que se va a trabajar están diseñados a la escala del robot.

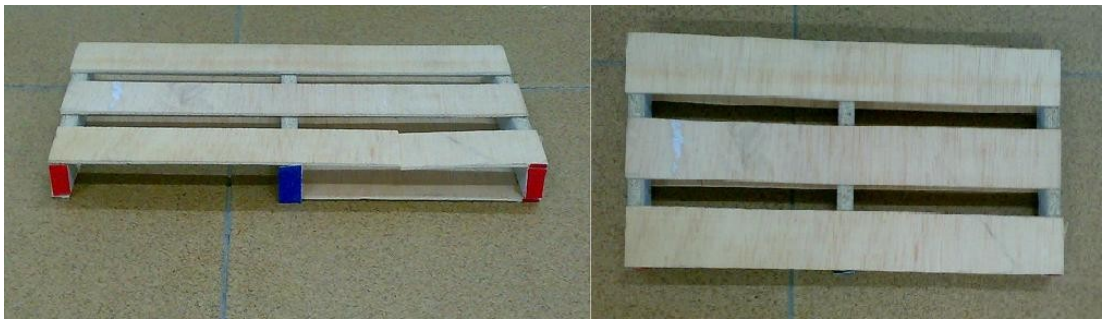

Figura 3: Vista frontal y superior del palé

Estos pales están construidos con madera de balsa lo que los hace muy ligeros, necesidad impuesta ya que el elevador no tiene demasiada fuerza. Las dimensiones son: 1 cm de alto, 30 cm de ancho y 15 cm de fondo.

Como fuente de información tan solo se va a disponer de:

- Una cámara motorizada al menos con dos grados de libertad para permitir el movimiento en el eje X y en el Y.
- La información de la posición del robot mediante el control de sus movimientos, es decir, la información odométrica de éste.

1.3 Objetivos

El objetivo fundamental del proyecto es implementar un sistema robusto capaz de localizar palés y llevarlos a una posición de destino fijada. No obstante, este objetivo global puede subdividirse en tareas más pequeñas para simplificar la acometida del sistema completo. Estos subobjetivos quedan caracterizados como:

- Implementar un sistema de identificación del palé utilizando como único medio de obtención de información una de las cámaras de la torreta.
- Conseguir obtener una buena estimación acerca de la posición real del palé utilizando la información extraída de las imágenes.
- Implementar un software de comunicación que permita controlar el servo del elevador de una manera simple y eficaz.
- Conseguir generar una trayectoria de aproximación que permita encarar el palé con una orientación adecuada.
- Crear un sistema de planificación para ejecutar las tareas necesarias para localizar, coger, transportar y dejar el palé en una posición determinada.

Además, estos objetivos han de ser satisfechos intentando cumplir las siguientes restricciones:

- La implementación ha de ser independiente del entorno. No tendría sentido restringir la funcionalidad del sistema a un entorno controlado como el del laboratorio, sino que ha de abordarse el problema desde una perspectiva más amplia.
- Se debe diseñar teniendo siempre presente el modelo de programación basado en componentes de forma que los nuevos componentes diseñados puedan interactuar con los ya existentes.
- Aunque el sistema se vaya a implementar y probar sobre Robex y la plataforma Robocomp (comentados en posteriores capítulos) se ha de intentar

generalizar los algoritmos de forma que puedan ser usados sobre otras plataformas.

- Debe funcionar en tiempo real. Todos los cálculos y procesos necesarios se ejecutarán a medida que sean necesarios.
- El sistema debe ser lo suficientemente robusto como para adaptarse a cambios en la iluminación.
- En ningún caso han de utilizarse indicadores externos al palé ni presuponer una determinada carga siempre constante.
- El entorno en que se probará el sistema se encontrará libre de obstáculos. Sin embargo, ha de tenerse en cuenta una posible comunicación con algún componente capaz de proporcionar información acerca de los obstáculos presentes.

1.4 Estructura del documento

La estructura de este documento se divide en una serie de capítulos bien diferenciados, para facilitar el entendimiento y seguimiento del trabajo realizado a lo largo del proyecto. A continuación se muestra una pequeña descripción de cada uno de ellos.

Capítulo 1 – Introducción: En esta parte de la documentación se hace una breve descripción del problema a resolver, establecimiento del contexto en el que se encuadra el proyecto y los objetivos que se pretenden conseguir. Capítulo en el que se encuentra el lector.

Capítulo 2 – RobEx y RoboComp: Aquí se proporciona una toma de contacto con el robot sobre el que se realizarán las pruebas y la arquitectura de componentes, estructura a seguir en la implementación de la solución.

Capítulo 3 – Diseño del sistema: En este capítulo se detalla la estructura de componentes que se propone como solución al sistema, se comentan los

componentes preexistentes a utilizar y se establecen las funcionalidades de los nuevos.

Capítulo 4 – Plan de acción: En este capítulo se aborda de una forma general el conjunto de tareas necesarias para conseguir llevar un palé desde una posición a otra. Se establece un diseño global mediante el que abordar el problema.

Capítulo 5 – Algoritmos más importantes: Dentro de este capítulo se comentan los principales algoritmos desarrollados durante la elaboración del proyecto. Son estos algoritmos los que han supuesto la mayor carga de trabajo.

Capítulo 6 – Especificación detallada del plan: Aquí se lleva a cabo un análisis a fondo del plan, especificando la forma de abordar cada una de las tareas.

Capítulo 7 – Experimentos: En esta capítulo se comentan los principales experimentos llevados a cabo junto con sus resultados.

Capítulo 8 – Conclusiones y trabajos futuros: Dentro de este capítulo se hace una valoración global del trabajo realizado y se apuntan algunas tareas a desarrollar en el futuro.

2 Capitulo 2: Robex y Robocomp

El entorno en el que se desarrolla el proyecto queda caracterizado por el robot sobre el que será integrado (familia RobEx) y por el sistema de componentes (RoboComp).

El robot en el que se llevarán a cabo las pruebas es un robot de la serie RobEx. Estos son robots móviles con conectividad wireless y/o Ethernet, disponen de un ordenador de a bordo en el que se ejecutan algunos componentes y desde el que pueden interactuar con otros componentes ejecutados de forma remota. En general las tareas del ordenador de a bordo se centran en el control del hardware del robot: movimiento de la base y captura de las señales de los dispositivos incorporados, láser, cámaras, micrófonos, etc.

El sistema de componentes llamado RoboComp puede definirse como un grafo de componentes o procesos que pueden ser distribuidos en varios procesadores proporcionando un sistema de comunicación que permite el intercambio de información entre sí mediante una síntesis muy simple.

El proyecto consistirá en la creación de los componentes necesarios para llevar a cabo los objetivos planteados anteriormente. Para ello se aprovechan todas las funcionalidades ya cubiertas por los componentes existentes así como los conceptos y metodología de desarrollo. Por la razón fundamental de la reutilización en la programación orientada a componentes, estos están pensados con la esperanza de que en un futuro no muy lejano, otros los utilizarán para incorporar nuevas características al robot.

Tanto RobEx como RoboComp son proyectos desarrollados en el Laboratorio de Robótica de la Universidad de Extremadura. Son libres y se puede acceder a ellos mediante sus correspondientes páginas web [12], [13]. Dado que los resultados del proyecto de fin de carrera se han incluido dentro de RoboComp, el software desarrollado en él se distribuye bajo la misma licencia

A lo largo de este capítulo se describirá el entorno de desarrollo y herramientas usadas, se hará una breve descripción del robot RobEx, se introducirá el paradigma de la programación orientada a componentes y, finalmente, se hará una breve descripción del repositorio de componentes RoboComp.

2.1 Robex

El robot RobEx [12] , [14] es una base robótica libre desarrollada en el Laboratorio de Robótica y Visión Artificial de la UEx, Robolab [16].

Es de tipo diferencial, esto es, el movimiento del robot depende únicamente de la velocidad de rotación de sus dos ruedas motrices. Además de las ruedas motrices, el robot dispone de una tercera rueda, de giro libre, que sirve de apoyo. La simplicidad de diseño y de los cálculos asociados al modelo diferencial son las principales razones que han definido el diseño, no sólo de RobEx, sino de otros muchos robots y gamas de ellos, como pueden ser: Segway, Kephera, o Roomba.

Sus planos se distribuyen bajo la licencia “Creative Commons Attribution-Share Alike 3.0” [15]. Por tanto, su diseño está abierto a cualquiera que lo quiera consultar [12] o contribuir a él. Al ser de tipo diferencial su construcción es relativamente sencilla. Si se tienen conocimientos de electrónica, cualquiera puede construirla.

RobEx está diseñado para llevar uno o varios ordenadores portátiles a bordo para realizar los procesos complejos que sean necesarios, hasta hoy relacionados con la visión, la audición o la inteligencia artificial.

Tanto los motores de la base, de corriente continua, como el resto de la electrónica, se alimentan de una batería de polímero de litio como las que suelen usarse para extender la autonomía de los ordenadores portátiles. La base está controlada por un microcontrolador dedicado al que se accede a través de una interfaz RS232 sobre USB. [14], [4].

Su principal orientación es la investigación y la docencia. Aunque cada vez resultan más versátiles y pronto darán el salto a la realización de trabajos concretos en empresas. Llevan utilizándose más de tres años a diario en las asignaturas de Robótica y Teoría de Sistemas que se imparten en la carrera de Ingeniería en Informática de la UEX. El origen de su diseño y desarrollo hasta su estado actual nació de los diferentes proyectos de investigación realizados en Robolab a partir de su creación en el año 1999. Desde su primera versión, cada mejora y nueva funcionalidad que incorpora se prueba intensivamente tanto en el laboratorio como en las aulas, por lo que se consigue una robustez considerable.

Los objetivos de diseño de los robots RobEx son:

- Ser apropiado para entornos estructurados, pero que a la vez pueda ser modificado para otros tipos de terreno.
- Conseguir un robot de bajo coste y fácil de construir con capacidad de procesamiento intensivo a bordo.
- Que el precio no esté reñido con la calidad: fiabilidad y robustez.
- Poder ser ampliado con diversos accesorios de sensorización y manipulación, así como llevar varios portátiles a bordo.
- Servir de plataforma para formar a futuros investigadores.

- Dar cabida a la experimentación utilizando hardware real en entornos controlados, una de las asignaturas pendientes en un plan de estudios tan teórico.

Como se indicó antes, el robot RobEx es hardware libre. Todo el software desarrollado se distribuye bajo GPL. Con esto se pretende:

- Que cualquier persona tenga acceso al diseño y software del robot, y pueda fabricarlo por sí misma.
- Que la comunidad participe en la mejora y evolución del robot.
- Que cualquier empresa pueda usar o vender RobEx, pero que cualquier modificación hecha al robot o a su software se haga pública y mantenga la misma licencia.

2.1.1 Características técnicas

En la figura se muestra una representación del diseño de la parte mecánica de RobEx.

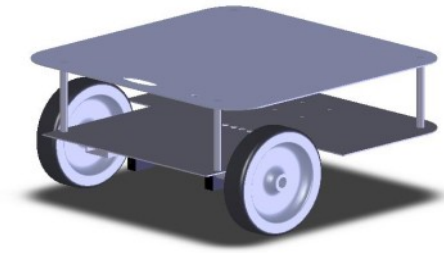

Figura 4: Representación del chasis de RobEx.

La estructura del robot está formada por dos planchas de aluminio que se separan y se afianzan entre sí mediante cuatro tubos de acero. Para colocar los motores en el chasis se usan soportes de acero, unidos a estos motores se encuentran las dos ruedas motrices (las que aparecen en la figura), mientras que la rueda de giro libre va colocada en la parte posterior de la plataforma.

Para disponer de autonomía energética, el robot incorpora una batería recargable a bordo. Para ello se utiliza un batería de polímero de litio de 21,6V y 3500mAh. Estas baterías son de consumo doméstico común y se suelen utilizar para ampliar la autonomía de ordenadores portátiles, por lo que son asequibles y están sobradamente probadas. Estas baterías proveen al robot de una autonomía de algo menos de dos horas.

El sistema de control también se encarga de realizar las operaciones y cálculos necesarios para el control PID de los motores. El bucle de control PID de cada motor funciona a una frecuencia de 1 Khz. Por tanto, cada milisegundo se calcula la tensión de salida más apropiada para alcanzar el objetivo actual. En otros términos, el motor sólo está "sin control" o en lazo abierto por periodos de 1milisegundo.

La base dispone de un circuito integrado, LSI7266R1, que hace de contador de los pulsos procedentes de los dos codificadores ópticos. Éste se comunica con el microcontrolador mediante dos buses, uno de datos de 8 líneas de entrada y salida y otro de control de 5 líneas de entrada. Esto permite obtener los datos necesarios para llevar las cuentas de la odometría.

Figura 5: RobEx primitivos, izquierda con cámara y derecha con láser.

En la figura se pueden ver dos RobEx primitivos a los que se les ha añadido una cámara (ejemplar de la izquierda) y un escáner láser (ejemplar de la derecha).

2.1.2 Características adicionales

A esta forma "básica" de RobEx se han incorporado una serie de componentes adicionales que aumentan sus capacidades, una torreta estéreo y un elevador de cargas. Además de estos, RobEx dispone de una serie de componentes adicionales que no se comentarán puesto que no han sido usados durante el desarrollo del proyecto. Como ejemplo podemos destacar un sistema de captura y digitalización de audio, un sensor láser o un sistema inercial de 5 gdl's, 3 acelerómetros lineales y 2 giróscopos

2.1.2.1 Torreta estéreo

La torreta estéreo (Figura 6: Torreta estéreo) está formada por una estructura en forma de U invertida con dos pies sobre los que se sitúan las cámaras, esta estructura esta diseñada en aluminio lo que le confiere una gran solidez. Cada cámara está acoplada a su base a través de una sujeción ensamblada a un motor que proporciona un giro independiente sobre el eje vertical. En el interior de la U, en uno de sus laterales, se sitúa un tercer motor que hace girar toda la estructura, proporcionando un movimiento rotatorio simultáneo de ambas cámaras sobre el eje horizontal.

Este diseño admite, por lo tanto, 3 grados de libertad sobre la cabeza robótica: uno de elevación (tilt) común a las dos cámaras, y dos para el giro (pan) de cada cámara. No obstante, el rango de giro de cada cámara está limitado a causa del diseño mecánico para evitar el choque de éstas con la estructura, por lo que no todos los movimientos son posibles.

Figura 6: Torreta estéreo

Los motores utilizados son servos Dynamixel RX-10 con microcontrolador incorporado y bus digital de comunicación convertible a USB mediante la que se conectan al ordenador de a bordo. Toda esta estructura es controlada por un componente "cammotionComp" diseñado específicamente para esta tarea. Este componente será comentado en posteriores capítulos.

2.1.2.2 Elevador de cargas

El elevador de cargas es un actuador de un grado de libertad diseñado para simular entornos industriales de transporte de palets.

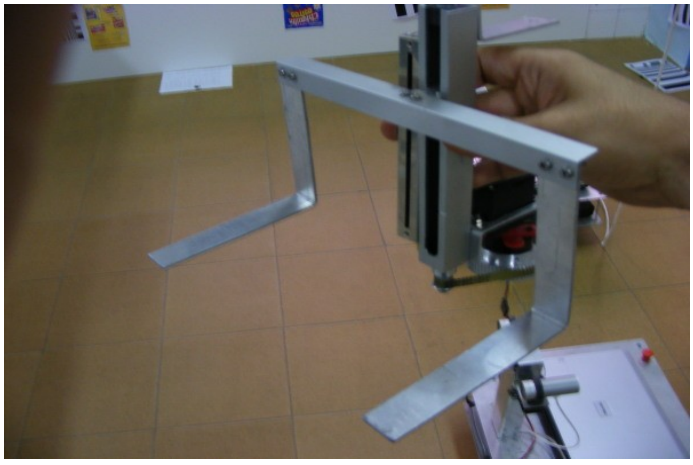

Figura 7: Elevador de cargas

El desplazamiento vertical se consigue con un tornillo sin fin movido por un servo motor situado en la parte inferior.

El recorrido de la pinza se controla en lazo cerrado con un potenciómetro lineal alineado con el tornillo sin fin mediante un microcontrolador como el utilizado en el control de la base de Robex.

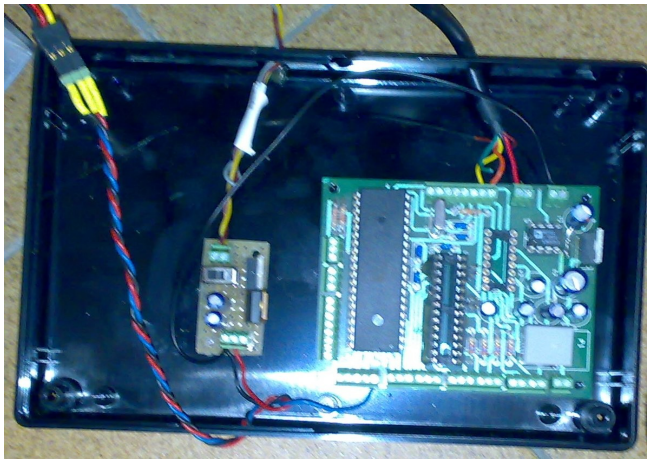

Figura 8: Microcontralador del elevador de cargas

El controlador consta de un microcontrolador ATmega32. Este miconcontrolador se encarga de tres cosas:

- Comunicación con el pc mediante una interfaz serie.(luego se utiliza un cable con conversión serie USB)
- Genera un tren de pulsos con ancho variable. En ancho del pulso determina en que posición va a estar el motor El ancho es equivalente al valor que se le manda para indicar la posición a la que debe dirigirse. Para el valor 00000 el ancho es 0,3 milisegundos y para 32.768 (posición máxima) es de 2,5 ms. el tiempo entre pulso y pulso no es de 20 ms.
- Conversor Analógico-digital (DAC): Se encarga de leer la posición de la pinza mediante el valor de tensión presente en el potenciómetro de ésta. Tiene una resolución máxima de 10 bits (de 0 a 1024).

El RobEx que se va a emplear durante el desarrollo de este proyecto es una versión bastante superior a los mostrados en la imagen (Figura 5: RobEx primitivos,

izquierda con cámara y derecha con láser.), A esa base se han incorporado la torreta estéreo y el elevador de palés. El resultado es el siguiente:

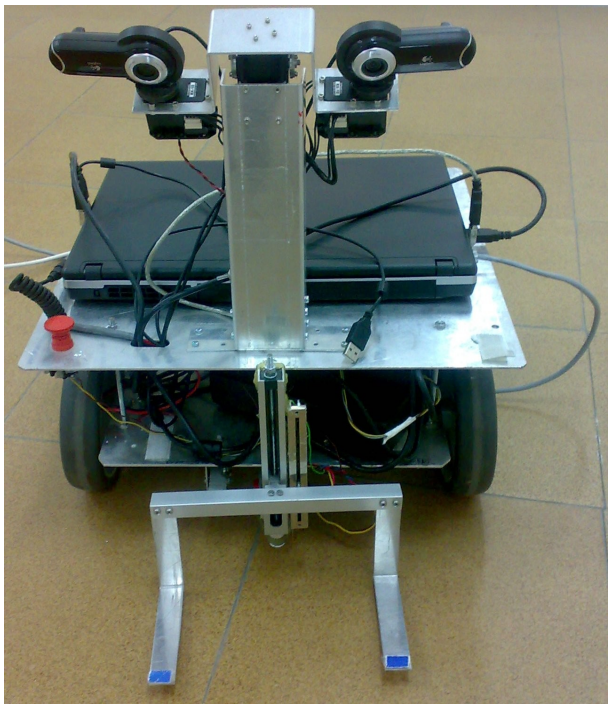

Figura 9: Robex con torreta estéreo y elevador de cargas

2.2 Programación orientada a componentes

Dos de los principales problemas que se presentan cuando se crea software son la escalabilidad y la reusabilidad. Estos problemas son especialmente agudos cuando se trata de software que se va a emplear en robótica, pues la reutilización es algo excepcional en este campo. A pesar de la importancia de la reusabilidad, generalmente se suele perder de vista este aspecto y se acaba creando software monolítico y poco utilizable.

En el ámbito de la robótica es muy común que los investigadores implementen todos los algoritmos con un diseño rígido y orientado a una tarea y/o a un robot específico, en muchos casos debido a unos requerimientos de tiempo y funcionalidad muy específicos. De ser así, cuando finaliza la etapa de implementación el software desarrollado acaba siendo imposible de utilizar. Suele estar tan ligado a una

plataforma o tarea específica que resulta más práctico empezar de cero (debido a las dependencias y efectos colaterales derivados de su rigidez).

La programación orientada a componentes [16] surge como solución a este tipo de problemas. Es un enfoque que no tiene necesariamente que ver con concurrencia o computación distribuida, sino con cómo se organiza el software. La programación orientada a objetos representó un gran avance respecto a la programación estructurada, sin embargo, cuando el número de clases y sus interdependencias crece, resulta demasiado difícil entender el sistema globalmente. Es por tanto beneficioso disponer de un grado mayor de encapsulamiento, que aúne diferentes clases relacionadas bajo una interfaz única, y permita comprender el sistema con menor grado de detalle. Muchos ven la programación orientada a componentes, que se propuso para solucionar este tipo de problemas, como el siguiente paso tras la programación orientada a objetos. [10]

Un componente es un programa que provee una interfaz que otros programas o componentes pueden utilizar. Es una pieza de código elaborado o a elaborar que codifica una determinada funcionalidad. Son los elementos básicos en la construcción de las aplicaciones en esta arquitectura, que se juntan y combinan para llevar a cabo una tarea concreta. A su vez, estos programas hacen uso de programación orientada a objetos (así como en programación orientada a objetos se hace uso de programación estructurada). Esta división en piezas de software de mayor tamaño que las clases, implementadas como subprogramas independientes ayuda a mitigar los problemas de los que se ha hablado antes y a aislar errores. Además, desde su carácter intrínsecamente distribuido ayudan a repartir la carga de cómputo entre núcleos, incluso, dependiendo de la tecnología usada, entre diferentes ordenadores en red.

Desde el punto de vista del diseño se pueden ver como una gran clase que ofrece métodos públicos. La única diferencia desde este punto de vista es que la complejidad introducida por las clases de las que depende el componente (o clase) que no son del dominio del problema, desaparece porque la interfaz del componente las esconde. Un componente puede ser arbitrariamente complejo, pero un paso atrás,

lo único que se ve es la interfaz que ofrece. Esto es lo que lo define como componente.

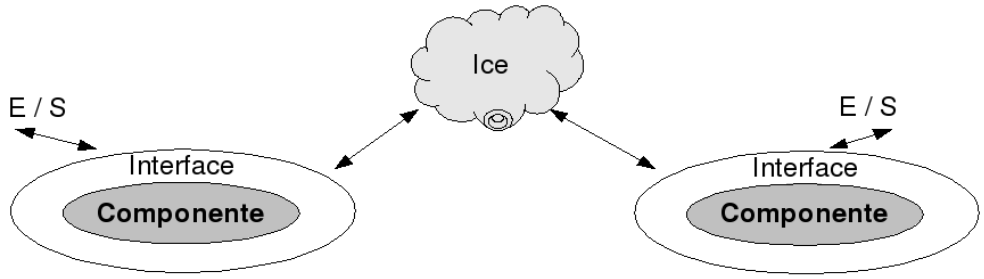

Figura 10: Representación genérica de componentes

Por tanto, si cada componente realiza una serie de tareas o responde a una serie de órdenes, necesitamos quien se encargue de esa comunicación. Es donde entra en juego Ice, pieza clave para la comprensión de la arquitectura y para la funcionalidad de la misma. Ice es un framework de comunicación muy interesante para su uso en robótica, sus características principales serán comentadas posteriormente. A pesar de estar desarrollado por una empresa privada es de fuentes abiertas y está licenciado bajo GPL. Además de su carácter libre hay otras dos razones para elegir Ice.

- La primera razón es su facilidad de uso: al contrario que otras tecnologías como Corba, la filosofía de Ice es soportar sólo aquellas características que los usuarios vayan realmente a necesitar y evitar introducir otras características que raramente se usan y dificultan el aprendizaje.

- La segunda razón es la eficiencia: si bien Ice codifica la comunicación eficientemente tanto en términos de tiempo como de espacio, otras alternativas suelen codificar la comunicación en XML. El uso de XML tiene ventajas en otras aplicaciones donde es conveniente que los humanos puedan entender el tráfico y no sean factores críticos ni la latencia ni la eficiencia, pero dentro de un robot esto no ocurre.

Ice soporta dos tipos de comunicación, por llamada remota (tipo RPC) o por suscripción (mediante un servicio que hace de servidor de mensajes). Sin embargo, este último introduce un paso intermedio cuya latencia hace desaconsejable su uso en robótica donde la ejecución en tiempo real es fundamental.

Para realizar una conexión con un componente lo único que se ha de conocer es su “endpoint”, es decir, la información necesaria para realizar la conexión: la dirección o nombre del ordenador al que se va conectar, el protocolo, el puerto y el nombre de la interfaz. Por ejemplo:

$$< \text{nombre} >:< \text{tcp/udp} > -\text{p puerto} - \text{h host}$$

donde 'nombre' es el nombre de la interfaz del componente al que se quiere hacer la conexión, 'puerto' es el puerto tcp o udp en el que el componente esté escuchando y 'host' el nombre del ordenador donde el componente se está ejecutando.

Desde el punto de vista del programador, una vez la conexión está hecha, el uso de componentes Ice es extremadamente simple. Cuando se realiza una conexión se crea una instancia de un proxy al componente en forma de objeto.

Al ejecutar un método del componente proxy el framework se encarga automáticamente de redirigir la llamada al componente remoto, por lo que la instancia del proxy se utiliza como si se tratase de una instancia de un objeto con la funcionalidad del componente.

Esta idea de usar un recurso remoto dentro de un programa no es nueva. Antes de la llegada de la programación orientada a objetos, ya había un protocolo en Unix llamado RPC para hacer llamadas remotas a procedimientos. Más tarde surgieron tecnologías como RMI, CORBA, DCOM, o Ice. [8]

Para usar programación orientada a componentes se ha de dividir el diseño del software en piezas que ofrezcan una interfaz. A cambio obtendremos mayor reusabilidad, utilizando los mismos componentes en diferentes contextos, de esta forma se reduce considerablemente el tiempo, el coste y el esfuerzo de desarrollo de nuevas aplicaciones, aumentando a la vez la flexibilidad, reutilización y fiabilidad de

las mismas. Será más fácil aislar y encontrar fallos, consiguiendo eliminar la necesidad de contemplar cientos de clases para comprender el software desarrollado.

2.3 RoboComp

RoboComp [13], es un repositorio de componentes basados en Ice con aplicaciones en robótica y visión artificial. RoboComp se comenzó a desarrollar en Robolab en 2005. Actualmente el proyecto ha sido migrado a SourceForge, donde, además de tener la página del proyecto (http://sf.net/projects/robocomp), dispone de un wiki (http://robocomp.wiki.sf.net/), donde hay documentación y un repositorio al que se puede acceder incluso directamente con un navegador web (https://robocomp.svn.sf.net/svnroot/robocomp).

Dispone de componentes para captura y visualización de vídeo, control del robot RobEx, detección y mantenimiento de regiones de interés (ROI), lectura y visualización de láser, lectura de joystick y navegación, entre otros muchos. Además dispone de un generador automático de componentes nuevos, de un programa gráfico de manipulación y control de componentes, managerComp y hasta de un componente encargado de grabar los datos producidos por los sensores para su posterior reproducción, replayComp.

La figura muestra un grafo en el que se pueden ver los componentes de los que dispone RoboComp y las dependencias entre ellos.

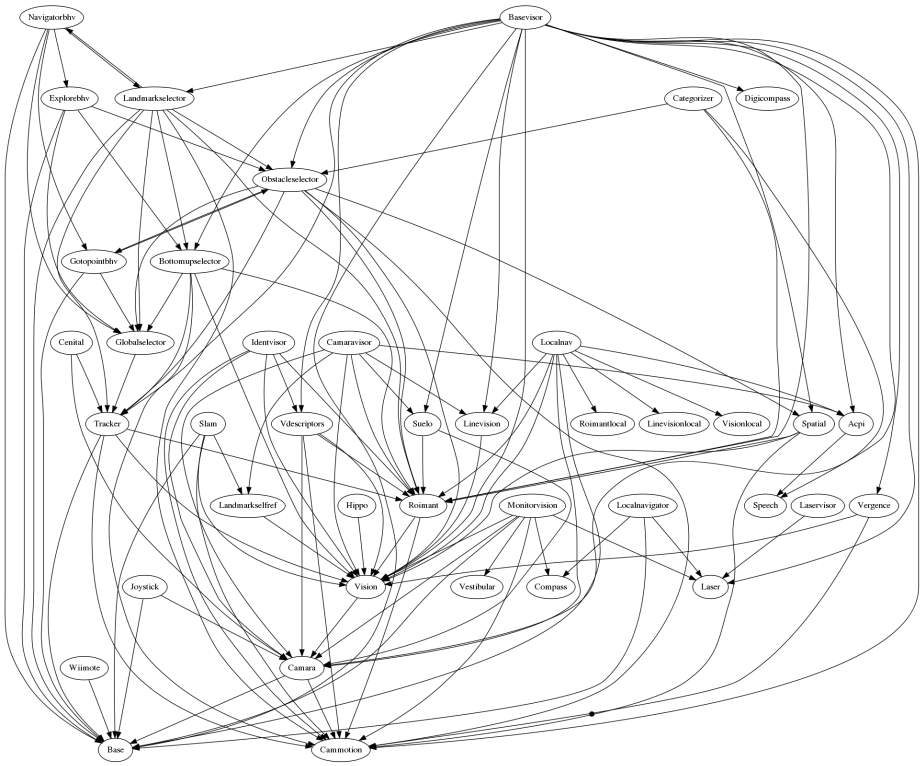

Figura 11: Grafo de componentes de RoboComp.

Todos los componentes, así como clases auxiliares que se desarrollen a lo largo del proyecto se encontrarán inmersas dentro de este repositorio por lo que resulta esencial conocer su estructura.

2.4 Entorno de desarrollo

A la luz de lo leído es fácil suponer que para desarrollar una aplicación basada en esta arquitectura, es necesario utilizar un conjunto de herramientas más o menos complejas y librerías que soporten las tareas exigidas a las capacidades del robot. La naturaleza de los problemas, eminentemente en tiempo real, a su vez exige que el tiempo de proceso sea mínimo y siempre se persigue, por filosofía del laboratorio y en la medida de lo posible, un mínimo consumo de recursos.

En las siguientes páginas se describe el software utilizado.

2.4.1 IPP/FrameWave

IPP y FrameWave son dos bibliotecas para procesar señales de una y dos dimensiones. La elección de estas bibliotecas frente a otras de similar objeto se tomó por su gran eficiencia. Ambas bibliotecas son competencia directa, de hecho, mientras IPP es desarrollada por Intel, FrameWave es desarrollada por AMD. La diferencia en el rendimiento respecto a sus competidores radica en que hacen uso de las instrucciones SIMD que aportan las extensiones de x86, 3DNow!, MMX y SSE y derivadas que ambos fabricantes incluyen en sus procesadores.

Ambas bibliotecas ofrecen APIs muy similares, pero se diferencian en la licencia bajo la que se distribuyen: mientras FrameWave se distribuye bajo la licencia libre Apache 2.0, IPP es software privativo. A pesar de que es gratuito para el uso personal bajo GNU/Linux, hay que pagar la licencia si se desea usar comercialmente, y su código fuente no es público.

La eficiencia del software desarrollado, no sólo en este proyecto sino en el resto del software se debe en gran parte a la de estas bibliotecas. De ellas se han usado multitud de funciones en otros componentes relacionadas con el tratamiento de imágenes. En este trabajo se ha utilizado para el análisis de la señal en el dominio de la frecuencias.

2.4.2 Qt4

Qt4 es un framework de desarrollo cuyo objeto principal es la creación de interfaces gráficas. A pesar de que sea este su principal uso, engloba una gran cantidad de funcionalidades distintas: interfaz multiplataforma con el sistema operativo (sistema de ficheros, procesos, hilos entre otras cosas), comunicación por red mediante sockets, interfaz con OpenGL, conexiones SQL, renderizado de HTML, y módulos de reproducción y streaming multimedia.

Atendiendo al lenguaje de programación, Qt está escrita en C++, pero existen multitud de bindings que hacen posible su uso desde otros lenguajes como Java, C#, Python, Perl y otros muchos.

Inicialmente fue desarrollada por Trolltech, una empresa noruega, bajo una licencia privativa. Después de varios cambios en la política de licencias pasó a distribuirse bajo una doble licencia GPL/QPL (esta última privativa). Finalmente, tras la compra de Trolltech por parte de Nokia, Qt fue licenciada bajo LGPL en 2009, eliminando así cualquier debate sobre la licencia de la biblioteca.

2.4.3 Qt4 designer

Qt 4 designer es una herramienta de la empresa Trolltecha para diseñar y construir interfaces gráficos de usuario desde los componentes Qt. Permite diseñar y construir widgets y dialogs de una forma sencilla y eficaz.

Una característica esencial de este diseñador es que permite aprovechar las señales y slots de Qt lo que facilita la tarea de conexión del interfaz con el código interno de la aplicación.

2.4.4 Ice

Ice es el middleware del que se habló en la sección anterior que permite crear componentes software que pueden funcionar de forma distribuida sobre plataformas heterogéneas, de forma que puedan comunicarse entre si y llevar a cabo un trabajo conjunto.

Es usado por RoboComp y, por tanto, por los componentes desarrollados en el proyecto de fin de carrera. Es el principal producto de la empresa estadounidense ZeroC, y se distribuye bajo una doble licencia GPL+ privativa para habilitar que las empresas desarrollen software privativo con Ice, a cambio del pago de una licencia. Las principales características de Ice frente a otras tecnologías similares es su rapidez, baja latencia y escalabilidad.

Uno de los problemas a resolver a la hora de crear componentes software es la creación de un lenguaje de definición de interfaces. Ice usa Slice, un lenguaje que se creó específicamente para este propósito.

2.4.5 CMake

CMake es una aplicación que permite generar automáticamente ficheros Makefile y ficheros de proyecto de varios IDE como KDevelop o Eclipse entre otros. CMake permite delegar la creación de ficheros Makefile, consiguiendo un resultado multiplataforma y robusto, sin llegar a perder el control del proceso de compilación.

CMake es una iniciativa libre que nació como respuesta a la ausencia de una alternativa suficientemente buena durante el desarrollo de una librería llamada ITK. Si bien dispone de soporte para Qt y algunas otras extensiones, es muy simple hacer nuevas extensiones.

2.4.6 KDevelop

Es un entorno de desarrollo integrado para sistemas GNU/Linux y otros sistemas Unix, publicado bajo licencia GPL, orientado al uso bajo el entorno gráfico KDE, aunque también funciona con otros entornos, como Gnome.

El mismo nombre alude a su perfil: KDevelop - KDE Development Environment (Entorno de Desarrollo para KDE). A diferencia de muchas otras interfaces de desarrollo, KDevelop no cuenta con un compilador propio, por lo que depende de gcc para producir código binario.

2.4.7 managerComp

La aplicación managerComp permite visualizar, tanto gráficamente como en una lista, el estado de los componentes configurados en tiempo real. A pesar de estar integrado en RoboComp, se detalla independientemente por no ser un componente propiamente dicho. Además de la visualización del estado de los componentes, también permite “arrancarlos” y “pararlos”.

2.4.8 OpenSSH

OpenSSH es una implementación libre del protocolo SSH (Secure SHell) que ofrece tanto la parte del servidor como la del cliente. Esta herramienta es importante en la puesta en marcha de proyecto porque, junto a managerComp, nos permite ejecutar rápida y remotamente los componentes que se deseen. Además, gracias a

que permite autenticación basada en llaves, se puede trabajar de forma segura sin necesidad de escribir la contraseña cada vez que se quiere cambiar el estado de algún componente.

OpenSSH es una iniciativa de los desarrolladores de OpenBSD. Se distribuye con distintas licencias (dependiendo de la pieza), pero a pesar de esto, todas ellas son libres, GPL o BSD.

2.4.9 GtkTerm

Es un terminal escrito con GTK+ ligero y simple para comunicarse con puertos serie. Se trata de una versión para linux del conocido Hyperterminal e Windows. Adicionalmente, tiene la capacidad de comunicar y recibir los caracteres en formato hexadecimal.

Se trata de una aplicación sencilla pero esencial en la comunicación con interfaces RS-232.

2.4.10 GNU/Linux

Finalmente, el sistema operativo usado durante el desarrollo y los experimentos, GNU/Linux. Gracias a él disponemos de un entorno de desarrollo gratuito y libre, herramientas de compilación, depuración, y una gran cantidad de bibliotecas complementarias. Se ha utilizado la distribución Kubuntu y Ubuntu pero por razones subjetivas.

3 Capitulo 3:Diseño del sistema

Dentro de este capítulo se va a tratar la estructura de componentes diseñada como solución al problema. Además de especificar los nuevos componentes a añadir a RoboComp, también se comentan las principales características y funcionalidades de los componentes preexistentes que van a ser utilizados en el proyecto.

Gracias al uso de la programación orientada a componentes, se puede conseguir tener una visión global del sistema bastante fiel a la realidad sólo con ver el grafo de componentes. En la figura se puede observar este grafo. En ella, los componentes desarrollados a lo largo del proyecto se encuentran sombreados en color blanco mientras que los componentes ya existentes aparecen en azul. La dirección de los enlaces hace referencia a la dependencia existente para el funcionamiento, no al sentido en el que se establece la comunicación.

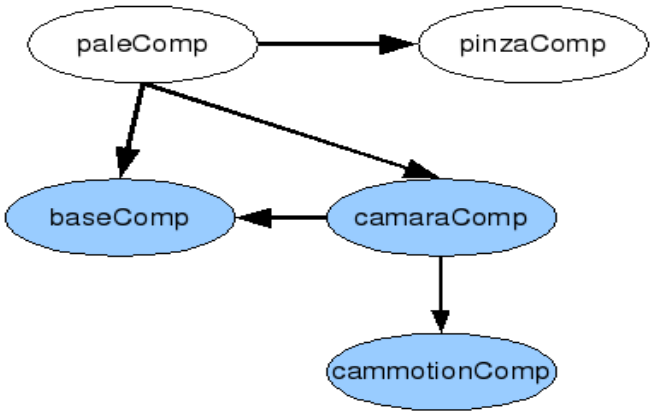

Figura 12: Grafo de componentes del sistema

El nodo *Base* representa a baseComp. Este es el componente encargado de gestionar la velocidad de cada una de las ruedas motrices del robot así como de devolver la información de la odometría. Es un componente indispensable en cualquier situación que implique desplazar el robot.

El nodo *Camera Motion* representa a camMotionComp. El objetivo de este componente es proporcionar un acceso sencillo al movimiento de las cámaras así como devolver la posición actual de los motores que controla.

El nodo *Camera* hace referencia a una instancia del componente cameraComp. Como se puede ver en el grafo, depende de camMotionComp, (lo llama con el fin de obtener la información de la posición de las cámaras). La tarea que realiza camaraComp es capturar vídeo en tiempo real y ofrecer las imágenes junto con la información más reciente que obtenga de la torreta y la odometría (proporcionada por la base). De este componente se obtienen las imágenes que posteriormente serán tratadas.

En el resto de la sección se tratará cada uno de los componentes en más detalle, pero desde un punto de vista conceptual. Para cada uno de ellos se hará una descripción de su fin y del origen de los datos con los que trabaja. Además, gracias al uso de programación orientada a componentes, se usarán los ficheros Slice que definen sus interfaces, ya que son una buena manera de entender el rol que tiene cada uno dentro del sistema.

3.1 Punto de partida

Como se ha comentado previamente, el proyecto parte de una serie de componentes ya desarrollados. En esta sección se van a comentar estos componentes. Todos ellos se encuentran en continua revisión y ampliación, puede dirigirse a "http://robocomp.wiki.sourceforge.net" para obtener la última versión disponible o ampliar la información.

3.1.1 baseComp

El componente baseComp, que fue uno de los primeros componentes en entrar en el repositorio de RoboComp, es el que se encarga de hacer de interfaz con la base del robot físicamente. La conexión al robot RobEx se realiza mediante una conexión RS-232 sobre USB. El comportamiento del componente queda definido por las invocaciones remotas que reciba mediante su interfaz Ice y la información del robot que lea del puerto serie. La interfaz permite leer la posición y velocidad de las ruedas de la base, así como solicitar la modificación de dichos valores de diferentes formas mediante la invocación de sus métodos remotos.

Dado que el componente fue creado previamente y no forma parte del proyecto, no se ha considerado apropiado explicar su funcionamiento interno. Lo importante es conocer su rol dentro del sistema. La interfaz Ice y los datos con los que trabaja el componente baseComp se definen en su correspondiente fichero Slice:

```
module RobolabModBase{
  exception HardwareFailedException{ string what; };
  exception OutOfRangeException{ string what; };

  struct TBaseState{
    float x;    //sideways displacement in mm
    float z;  //from displacement in mm
    float alfa;  //angle rotated since last reset in rads
    float advV;  //current advance speed
    float rotV;  //current rotation speed
    float adv;  //Incremental measures
    float rot;
    bool isMoving;
  };
  interface Base  {
    // All speed parameters are in radians and radians/sg
    //Get Base local state
```

```
    void getBaseState( out TBaseState state );
    // Set base advance and rotation speed
    idempotent bool setSpeedBase( float adv , float rot);
    //Set base to (x,y) at v speed en mm/sg and rads/sg resp.
    bool setPosBase(int x,int y ,float alfa,float adv ,float rot );
      // Stop set base position
    void stopSetPosBase();
    // Stop the base
    bool stopBase();
    //Reset Odometer
    idempotent bool resetOdometer();
    //Set Odometer to given value
    idempotent void setOdometer( TBaseState state );
  };
};
```

Tabla 1: Interfaz Ice del componente baseComp

El componente es llamado por camaraComp para poder adjuntar la información de la odometría a las imágenes y por paleComp. Como veremos, este último se encargará de controlar el movimiento del robot en las tareas que nos conciernen.

3.1.2 cammotionComp

Este componente, al igual que baseComp, también se reutiliza y no ha sido desarrollado dentro del proyecto. Ofrece una interfaz de control para una torreta estéreo, en concreto una de tilt común y pan independiente en cada cámara. Su objetivo es mover los servos de la torreta tal y como se le ordene, así como responder a las consultas de la configuración de la torreta (las posiciones de los servos que controla).

Ha sido programado para ser compatible con distintos tipos de servos digitales: Diolan, Dynamixel y Megarobotics. Además, la gama de servos con la que trabaja se puede extender fácilmente con la implementación de una clase virtual que se creó para ello

```
//Stereo head control
//Al angles in radians , radians/sg , etc
module RobolabModCamMotion{
  exception HardwareFailedException{ string what; };
  exception OutOfRangeException{ string what; };

  struct TMotorRanges{ //in Radians
    float min, max;
    byte number;
  };
```

```
  struct TMotorState{ // In Radians
    float pos, vel, power;
    int p,  v;
    bool isMoving;
  } ;
  struct THeadRanges{ // In Radians
    TMotorRanges left, right, tilt leftTilt;
    int baseline;
  };
  struct THeadState{ // In Radians
    TMotorState left, right, tilt, leftTilt;
    bool isMoving;
  };
  struct TParams{    //Configuration Params
    int TILTMOTOR, LEFTMOTOR, RIGHTMOTOR;
    int LEFTCAMERA, RIGHTCAMERA, BOTHCAMERAS;
    int RIGHTZEROPOS, LEFTZEROPOS, TILTZEROPOS;
    string device;  //Communications port
    string handler;  //Drive for servomotor hardware
    int baseline;
    bool tiltInvert, leftInvert, rightInvert;
  };
interface CamMotion{
    void resetHead(); //Send cameras to ZEROPOS and set zero speed.
    void stopHead();               //Stop head where it is now
    void setPanLeft(float pan);    //Set PanI servo to pan rads
    void setPanRight(float pan);   //Set PanD servo to pan rads
    void setTiltLeft(float tilt);  //Set Tilt servo to tilt rads
    void setTiltRight(float tilt); //Set Tilt servo to tilt rads
    void setTiltBoth(float tilt);  //Set both cameras to tilt rads
    void getMotorRanges(byte motor, out TMotorRanges info);
    void getMotorState(byte motor, out  TMotorState state);
    void getHeadState(out THeadState state);
    void getHeadRanges(out THeadRanges ranges);

    void setRadSaccadic(float pan, float tilt, int cam);
    bool isMovingMotor(byte motor);
    bool isMovingHead();
  };
};
```

Tabla 2: Interfaz Ice del componente cammotionComp

Dentro del sistema, camMotionComp se encarga únicamente de realizar el control de los servos y atender a las peticiones de información de camaraComp. Para facilitar la detección y posterior seguimiento de los objetivos, el componente paleComp hará uso de la torreta por lo que será necesario conocer la funcionalidad de este componente.

3.1.3 camaraComp

El componente camaraComp es otro de los primeros componentes de RoboComp. Su principal función es capturar vídeo y atender a las peticiones de imágenes que otros componentes realicen. Además, ha de adjuntar a las imágenes la configuración de la posición que tenía la torreta y el estado de la odometría cuando la imagen fue tomada. Tiene capacidad tanto para mandar imágenes de una cámara, como de dos a la vez.

Para evitar problemas de sincronización debidos a la latencia de la comunicación, cuando trabaja con dos cámaras, puede devolver ambas imágenes simultáneamente en una única llamada. Esta característica es básica para trabajar con visión estereoscópica. Otra de las características interesantes de camaraComp es que permite trabajar con distintos tipos de cámaras: v4l2 (Video For

Linux v2), IEEE 1394 (FireWire), o mediante una tubería Unix hacia Mplayer (lo que además de extender aun más el abanico de cámaras soportadas, permite la captura de imágenes desde ficheros de vídeo y recientemente del simulador de robtos Gazebo).

```
module RobolabModCamara {
  exception HardwareFailedException { string what; };
  exception MovingImageException { string what; };

  sequence<byte> imgType;
  sequence<int> intVector;
struct TCamParams {
    string name, driver, device, mode;
    int focal, width, height, size, FPS;
    int numCams, leftCamera, rightCamera, bothCameras;
    int inverted, leftInverted, rightInverted;
    int saturation;
    int lineFreq;
    bool talkToBase, talkToCamMotion;
  };
  interface Camara {
    // YUV420 format - 2 planes unpacked
    idempotent void getYUVImage(int cam, out imgType roi,
     out RobolabModCamMotion::THeadState hState,
     out RobolabModBase::TBaseState bState
     ) throws HardwareFailedException;
    // Luminance - 1 Plane
    idempotent void getYImage(int cam, out imgType roi,
     out RobolabModCamMotion::THeadState hState,
     out RobolabModBase::TBaseState bState
```

```
     ) throws MovingImageException;
    // Luminance in LogPolar
    idempotent void getYLogPolarImage(int cam, out imgType roi,
     out RobolabModCamMotion::THeadState hState,
     out RobolabModBase::TBaseState bState
     ) throws MovingImageException;
    // Luminance - 1 Plane. Compressed and resized.
    idempotent void getYImageCR(int cam, int div, out imgType roi,
     out RobolabModCamMotion::THeadState hState,
     out RobolabModBase::TBaseState bState
     ) throws MovingImageException;
    // RGB packed for visualization - 3 planes
    idempotent void getRGBPackedImage(int cam, out imgType roi,
     out RobolabModCamMotion::THeadState hState,
     out RobolabModBase::TBaseState bState
     ) throws MovingImageException;
    // Lum + RGB - 4 planes unpacked
    idempotent void getYRGBImage(int cam, out imgType roi,
     out RobolabModCamMotion::THeadState hState,
     out RobolabModBase::TBaseState bState
     ) throws MovingImageException;
    // Return relevan comp params
    TCamParams getCamParams();
    // Inner feeding of images
    idempotent void setInnerImage(imgType roi);
  };
};
```

Tabla 3: Interfaz Ice del componente camaraComp

Todos los componentes del sistema trabajan con imágenes en color, y no se usan otras capacidades del componente que no sean las de captura de imágenes.

El único método de camaraComp invocado por otros componentes del sistema es getRGBPackedImage(). Además, aunque se dispone de visión estereoscópica no se va a hacer uso de ella. El desarrollo del sistema se va a llevar a cabo utilizando solo una de las cámaras. Como se puede ver, el método devuelve la imagen con la información de la configuración de la torreta y la odometría en una sola llamada. Esto hace que la mayoría de los componentes no necesiten conectarse, ni a baseComp, ni a camMotionComp. En este caso si será necesaria una conexión directa puesto que el componente paleComp realiza llamadas para posicionar tanto la base como la torreta.

3.2 Creación de un componente genérico

Antes de proceder con la especificación de los componentes construidos se mostrará como crear un componente genérico. Cuando se incluye un nuevo componente en RoboComp se usa generalmente una herramienta que lo genera automáticamente. El generador de código también incluye el código necesario para crear una conexión a los componentes que se le especifiquen. A los componentes que se generen automáticamente se le pueden añadir después todas las clases que sean necesarias. Además, muchas veces se desea añadir en el fichero de configuración algún parámetro específico y, seguramente, modificar el fichero de interfaz por defecto (que no ofrece ningún método). De no usar el generador automático de código, la creación de componentes, incluso pequeños, sería bastante tediosa, y susceptible de errores ya que es necesario integrar el código del middleware Ice y el código propio del componente.

Por tanto, la creación de los dos nuevos componentes se llevará a cabo utilizando esta herramienta. Tanto pinzaComp como paleComp siguen la estructura básica definida mediante esta creación. Esta estructura es la que se observa en la figura siguiente.

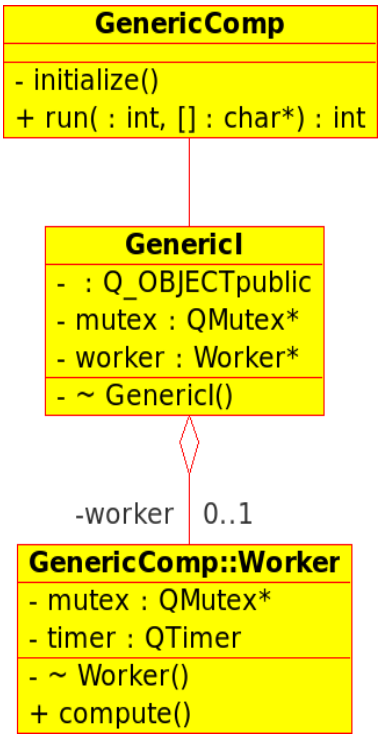

Figura 13: Diagrama de clases de un componente genérico.

Como se puede observar en la figura, además del código de programa principal, el componente plantilla consta de tres clases: GenericComp[16] , GenericI y Worker. Cuando se usa el generador de código éste modifica los nombres de las clases GenericComp y GenericI en función al nombre del componente cambiando Generic por lo que corresponda en función al nombre del componente.

Cada componente consta de, al menos, dos hilos de ejecución, siendo estos el hilo principal y un hilo separado para responder a las llamadas Ice. El último de estos dos hilos se crea cuando se instancia una interfaz Ice (a la clase creada se le tiene que pasar como parámetro la clase que se hace cargo de las llamadas). El hilo principal de ejecución se encarga de la parte activa del componente, de hacer las llamadas Ice necesarias si las hay, y los cálculos asociados al comportamiento del componente. Para facilitar la comprensión del código la clase GenericComp delega en Worker todo el trabajo y GenericI queda como una clase que, de forma transparente y asíncrona, responde a las llamadas remotas.

Las clases Worker y GenericI, que como ya hemos visto se encargan de la parte interna y externa respectivamente, se ejecutan en exclusión mutua gracias a un mutex que comparten. El bloqueo del mutex se realiza cada vez que GenericI recibe una llamada o que Worker va a cambiar alguna parte de su estado que pueda modificar las respuestas de GenericI a sus invocaciones remotas.

Como se puede ver en el diagrama, GenericI tiene acceso a la clase Worker. Dicha condición, que sólo se da en ese sentido, es necesaria porque generalmente toda la información relativa al estado del componente se guarda dentro de Worker.

3.3 Diseño de los nuevos componentes

En esta sección se van a comentar los nuevos componentes que conformarán el sistema.

En el caso del primero de ellos, pinzaComp, debido a su sencillez quedará totalmente explicado en esta sección, mientras que en el caso del otro, paleComp, será detallado a lo largo de posteriores capítulos.

3.3.1 pinzaComp

El componente pinzaComp va a ser un componente de los de más bajo nivel, al igual que los componentes base o head, es decir, no se va a comunicar con ningún otro, se limitará simplemente a servir de puente entre las peticiones del resto de componentes y el hardware del elevador, podríamos decir que se trata de un proveedor de servicios.

En esta parte del documento solo se va a comentar el software diseñado, todos los aspectos técnicos del elevador ya han sido comentados en una sección previa.

Como se ha comentado con anterioridad, el movimiento de la pinza está gestionado por un servo al cual se accede mediante controlador diseñado específicamente para esta tarea, la comunicación con el microcontrolador se establece mediante una interfaz USB. Este microcontrolador se comunica con el portátil a través del puerto USB y ofrece una sencilla interfaz de comandos de

arranque, parada, posicionamiento y estado. Mediante el es posible indicar una orden de desplazamiento al microcontrolador o leer la posición en la que se encuentra.

El funcionamiento interno es algo más complejo, realmente puede dividirse en dos funciones diferentes:

3.3.1.1 Posicionar la pinza

La orden de posicionar la pinza no es enviada directamente al servo sino que es almacenada y se indica una orden de movimiento al servo hasta que la posición del potenciómetro se iguala con la posición enviada, es decir, el servo se "mueve" hasta que el potenciómetro le indica que ha llegado a su destino.

Al llevar a cabo el desplazamiento de esta forma se consigue que la pinza mantenga siempre su posición puesto que el microcontrolador se encarga de ir verificando la posición. Este sistema puede parecer ventajoso, en caso de que el potenciómetro varíe de posición lo más mínimo (a causa de un salto de la base o un toque con algún objeto) el servo intentará desplazar de nuevo el elevador a su posición correcta.

Esta forma de actuar puede parecer óptima puesto que ante cualquier cambio el elevador se desplaza a su posición original, sin embargo, la realidad es que el servo está continuamente ejerciendo presión sobre el elevador, ante la más mínima vibración se pone en marcha el sistema lo que produce un considerable consumo energético.

El microcontrolador tiene un sistema de ordenes sencillas mediante las que comunicarnos con él.

Para indicar la posición a la que desplazarse se utiliza el rango de posiciones del potenciómetro. Este va desde la posición "7000" que se corresponde con la posición más alta, a la "23000", que sería la más baja.

La estructura del comando a enviar sería:

P0 : XXXXX*

Donde XXXXX indica la posición a la que debe desplazarse.

El sistema de control responde para comunicar que la orden ha llegado correctamente, esta respuesta es:

l0*

Por tanto, para enviar la pinza a la posición "zero", el centro de su movimiento, el comando quedaría:

P0 : 15000*

3.3.1.2 Leer la posición actual

A la hora de leer la posición en la que se encuentra la pinza, lo que se lee es el pulso de movimiento del servo. Si la posición a leer fuera la contenida en el microcontrolador tendríamos un problema, y es que esta posición es almacenada al indicarle una orden de movimiento. Esto provoca que al leer una posición mientras la pinza se encuentra en movimiento se obtiene la posición a la que se esta desplazando y no la posición real en la que se encuentra en ese instante.

Como ocurre al enviar la posición, en la lectura es necesario utilizar la escala de pasos del servo, dicha escala va desde la posición "200", que se corresponde con la para de abajo, y la "540" que se corresponde con la posición más alta.

El comando de lectura tiene la forma:

C*

Mientas que la respuesta del microcontrolador es:

C = XXXX*

Donde XXXX indica la posición del servo.

Por tanto, si la pinza se encontrase en la posición enviada anteriormente, el centro de su movimiento, la respuesta obtenida sería:

C = 0370*

El enviar una posición o leerla implica llevar a cabo un proceso de cambio de escala, es obvio que los pasos de movimiento del servo y la lectura de posición del potenciómetro se encuentran en diferentes escalas por lo que es necesario adaptarlas.

La escala ha sido establecida en un rango de 800 posiciones de forma que el cero sería el centro, -400 sería la posición más baja posible y 400 la más alta.

Para facilitar el manejo de la pinza se han incorporado una serie de métodos encargados de posicionar la misma. De esta forma, se dispone de los métodos "setUpPosition", "setDownPosition" y "setZeroPosition" que desplazan la pinza hasta la posición superior, inferior y cero respectivamente. También se ha modificado el rango de actuación de la pinza de manera que la escala de movilidad resulte más sencilla de interpretar puesto que el movimiento real viene establecido en pulsos del servo.

También se ha incorporado un pequeño visor en el que se dispone de una serie de botones y un slider para comprobar el correcto funcionamiento de la pinza.

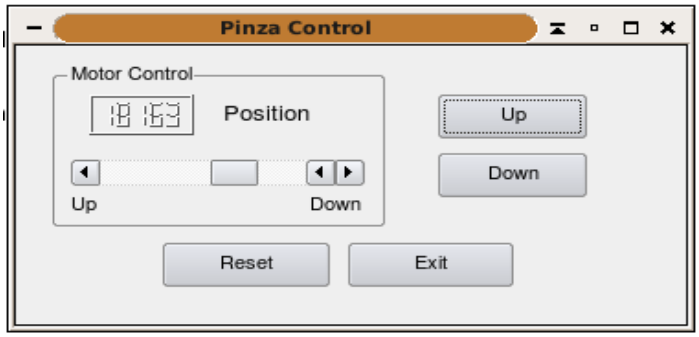

Figura 14: Controlador para la pinza

Así pues, el interfaz desarrollado para este componente quedaría:

```
module RobolabModPinza {
  interface Pinza{
      // Set Pinza Position in mm; upwards positive, downwards
negative from zero
      bool setPinzaPosition( int pos );
      //Get Pinza position
      int getPinzaPosition();
      //Set Zero
      void setZeroPosition();
      //Set down
      bool setDownPosition();
      //set up
      bool setUpPosition();

  };
};
```

Tabla 4: Interfaz Ice del componente pinzaComp

Como puede verse, consta de muy pocas funciones y cada una de ellas se encarga de una tarea bastante simple. Por ello, en posteriores capítulos simplemente se hará mención a las funciones utilizadas, no entrando a comentar nuevamente las características de la misma.

3.3.2 paleComp

Este es el componente que va a llevar toda la carga computacional. Prácticamente todo el trabajo del proyecto se ha centrado en la elaboración de este componente. Sus tareas han crecido de una forma un tanto descontrolada por lo que uno de los primeros trabajos a llevar a cabo en un futuro consiste en reducir la carga de trabajo de este componente subdividiéndolo en otros más sencillos.

Las principales tareas que lleva a cabo pueden resumirse en:

- Comunicación con el resto de componentes para obtener de ellos diferentes funcionalidades: generar movimiento para baseComp, obtener las imágenes de camaraComp, realizar movimientos del elevador utilizando pinzaComp y posicionar la cabeza mediante cammotionComp.

- Generar y controlar el plan de ejecución de todas las tareas necesarias para localizar, aproximarse, coger, buscar destino y dejar el palé. La especificación de este plan será comentada en el capítulo siguiente.

- Implementar un sistema capaz de generar una trayectoria precisa que permita ir hacia el palé con una orientación correcta, que permita introducir la pinza.

- Obtener información de las imágenes con la que se pueda determinar si lo que se esta viendo es o no un palé y estimar, además, su posición respecto al robot.

Debido a la gran cantidad de tareas a llevar a cabo y su complejidad, este componente será explicado en los posteriores capítulos: centrando el capítulo 4 en la gestión y elaboración del plan, el 5 en la explicación e implementación de los principales algoritmos y el 6 en detallar la funcionalidad de cada una de las etapas del plan.

4 Capitulo 4: Plan de acción

El problema principal a tratar es la creación de un plan de acción. Aunque el hecho de “coger” un palé puede parecer simple, la realidad es que han de llevarse a cabo diferentes acciones de forma coordinada.

En general, a la hora de elaborar un plan de acción han de tenerse en cuenta tres cuestiones principales: las tareas que debe realizar el plan, el orden en el que se han de llevar a cabo dichas tareas y la necesidad o no de establecer puntos de retorno entre las diferentes etapas del plan. A medida que avanza un plan, a veces, es necesaria la reevaluación de parámetros o situaciones que se dieron por ciertos en una etapa anterior, más aún cuando se trabaja en un entorno real, el cual suele variar de una forma poco predecible.

4.1 Diseño del plan

En primer lugar hay que determinar una serie de tareas que han de llevarse a cabo para conseguir ir hasta el palé, cogerlo y llevalo a una posición destino.

Una primera aproximación al diseño sería:

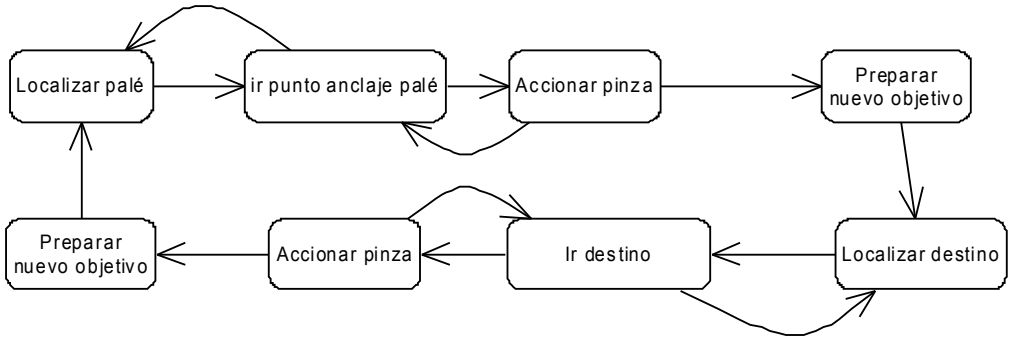

Figura 15: Especificación general del plan

La primera línea del plan se corresponde con la búsqueda del palé, mientras que la segunda hace referencia a la búsqueda del destino de este palé. Realmente, durante el proceso de ir hacia el palé o de llevarlo a su destino se llevan a cabo las mismas tareas, solamente varía la función que se encarga de identificar que es un palé o que es el destino de uno. Por tanto, podemos agrupar estas tareas como si se tratase de la misma, en adelante en lugar de hablar de "palé" y "destino" trataremos a ambos como objetivos con lo que el esquema del plan final quedará más claro. El esquema a resolver se puede presentar como:

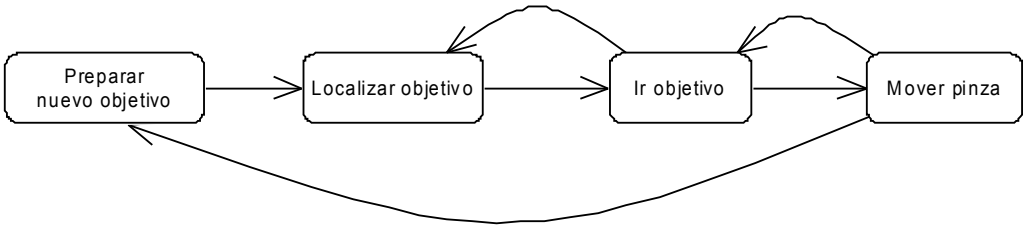

Figura 16: Plan en función de objetivos

A priori, parece que el número de las tareas a llevar a cabo es pequeño. Sin embargo muchas de ellas tendrán una subdivisión interna.

No hay que olvidar que se trata de un sistema basado en componentes, lo que quiere decir que es necesario establecer una comunicación con cada uno de los componentes implicados en el proceso. Por tanto, hay que tener en cuenta posibles

fallos en dicha comunicación. Será necesario establecer estados de “emergencia” a los que se llegará ante el fallo de alguna de estas comunicaciones y en los que será necesario preguntar al usuario cual ha de ser la acción a llevar a cabo. Estos estados no han sido incluidos en el modelo para simplificar su entendimiento.

Lo más importante del plan es la posibilidad de retorno entre las etapas “Ir objetivo” y “Localizar objetivo”y entre la etapa “mover pinza” e “ir objetivo” se trata de un proceso crucial para el correcto desarrollo de la misión.

En el entorno real en el que se mueve el robot no ha de fiarse sencillamente de lo que ha determinado la función de localización como correcto una única vez. Si bien el palé no va a cambiar de posición el hecho de tener que llevar a cabo un proceso de aproximación, en el que posiblemente deje de estar visible, y los constantes cambios en la iluminación de las imágenes obtenidas, debidos a este movimiento puede crear falsos positivos El tema de la iluminación es crucial en los reconocimientos basados en color y será tratado ampliamente en posteriores secciones.

La posibilidad de retorno entre las etapas “mover pinza” e “ir objetivo” refleja la necesidad de regenerar una trayectoria de aproximación si se detecta que es imposible llevar a cabo el movimiento de la pinza correctamente, por ejemplo, por no haberse llevado a cabo bien la aproximación, de forma que una de las partes de la pinza queda fuera del palé.

4.2 Especificación del plan

Analizando un poco más en profundidad el plan se detectan una serie de subtareas que han de llevarse a cabo. Las tareas intentan englobar funcionalidades independientes unas de otras.

El esquema anterior incluyendo esta subdivisión de tareas quedaría:

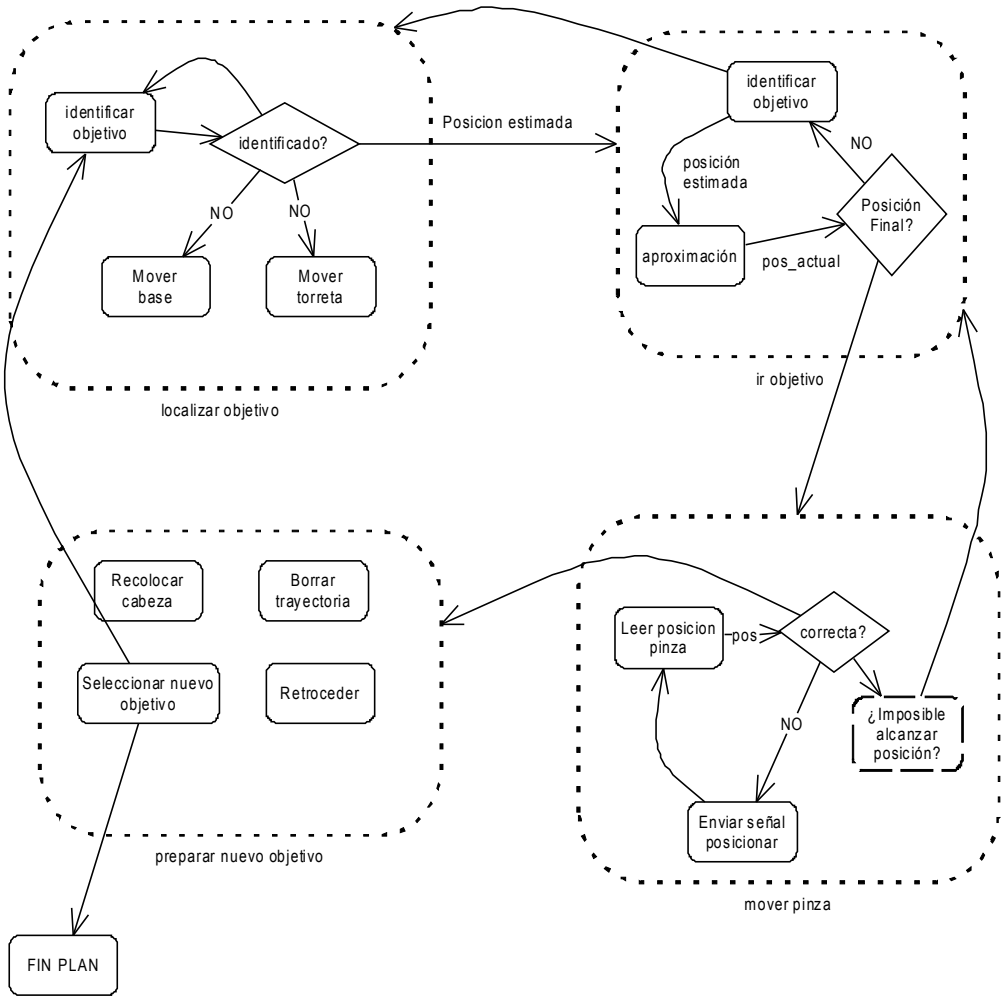

Figura 17: Desarrollo del plan

En este esquema, a su vez, aparecen tareas complejas que necesitan de la consiguiente subdivisión en otras tareas más simples. Entre esta últimas se encuentran las que engloban el movimiento de la pinza y también las encargadas de inicializar el proceso de detección, mientras que la carga de trabajo va a residir en las tareas que llevan a cabo la identificación y el movimiento de la base (aproximación).

Son estas tareas las que suponen un mayor problema y en las que se centra gran cantidad del tiempo empleado en la elaboración del proyecto. Tanto para discernir que se trata de un palé, para estimar su posición o para aproximarse a él se ha hecho uso de una serie de algoritmos más complejos que serán tratados en el capítulo siguiente.

La subdivisión de la identificación del objetivo queda de la siguiente forma:

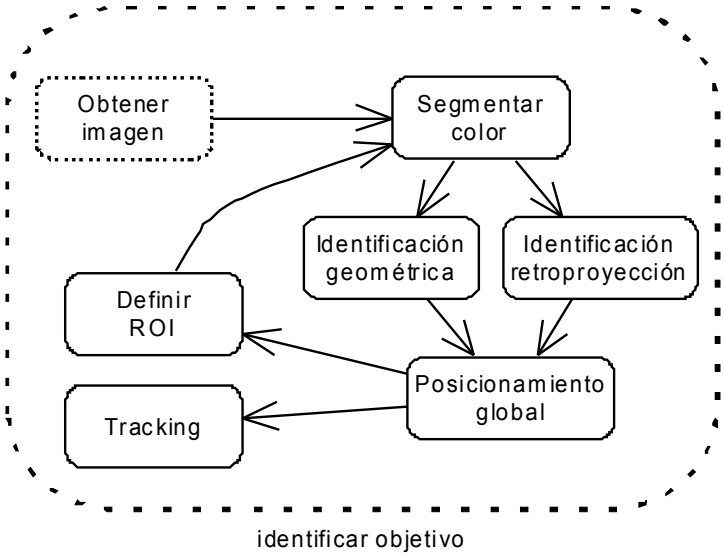

Figura 18: Fase de identificación del objetivo

Mientras que la aproximación al objetivo constaría de las siguientes subetapas.

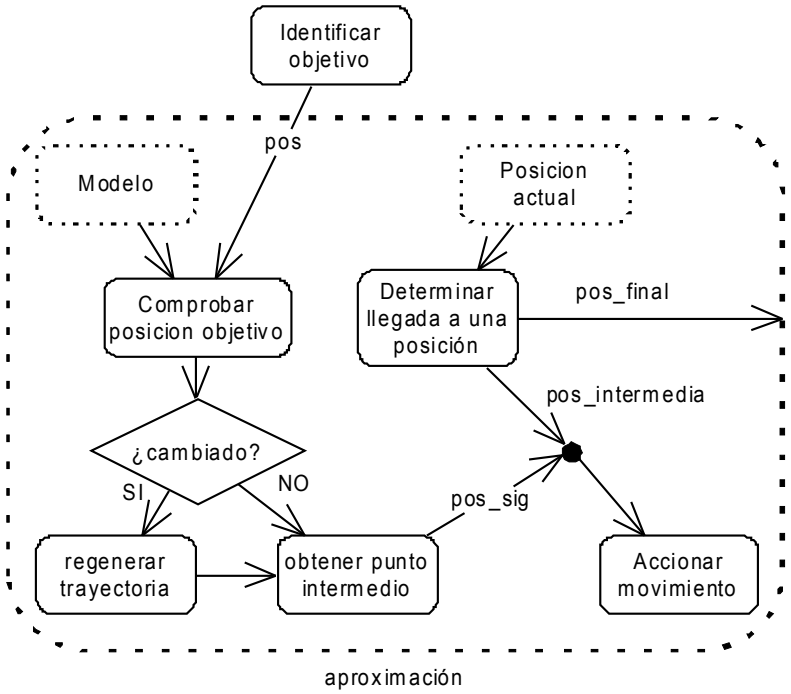

Figura 19: Fase de aproximación

Todas las tareas presentes en el plan serán abordadas en profundidad en un capítulo posterior.

4.3 Control

Una tarea que no aparece en el plan es la consecuencia del mismo. Son necesarias una serie de funciones que aseguren el correcto funcionamiento del plan, encargándose de producir los cambios entre unas etapas y otras y comprobando la correcta ejecución de las mismas.

La comprobación de la ejecución de las etapas es muy diferente en función de cual se vaya a evaluar, mientras en unas se trata de la lectura de una posición de la pinza o la base, en otras hay que comprobar la información obtenida de las imágenes o que no quedan más tareas pendientes por lo que el plan ha finalizado.

Además, existen unas precondiciones que se han de cumplir antes de ejecutar cada una de ellas. No tiene sentido llevar a cabo el movimiento de la pinza si antes no se ha realizado la aproximación. Se ha definido una función encargará de asegurar que estas transiciones entre etapas sean las correctas.

El encontrarnos en un entorno real, entra el juego la problemática de la concurrencia, no podemos abarcar totalmente el control de la base, de la pinza o de la torreta sino que han de establecerse intervalos que permitan la comunicación de estos componentes con otros. Por ello, todas las etapas están controladas mediante *timers* encargados de activar su ejecución. De esta forma se dispone de pequeñas rodajas de *tiempo libre* entre la una ejecución y otra.

Realmente, es la activación y desactivación de estos *timers* la que controla las transiciones entre unas etapas y otras, existiendo siempre por encima un *timer* del plan global. De esto forma quedan establecidos 4 grandes temporizadores:

- Encargado del plan
    - Controla las precondiciones
    - Ejecución de las etapas
    - Paso de una etapa a otra
- Encargado del proceso de aproximación

- Encargado del proceso de localización
- Encargado del movimiento del elevador

Otra de las posibles tareas futuras consiste en la implementación de este modelo de plan mediante una máquina de estados. Esta transformación simplificará la gestión de los mismos aportando además un mayor control a las transiciones.

5 Capítulo 5: Algoritmos principales

En este capítulo se van a tratar los algoritmos principales del plan.

Cada una de las secciones se centra en la solución de uno de los problemas detectados durante la elaboración del plan general de acción (capítulo anterior), en primer lugar se introduce una explicación teórica para a continuación mostrar su aplicación al problema. Así, los principales problemas a tratar son:

- Estimación de la posición del palé mediante la información proporcionada por la cámara.
- Identificación del palé.
- Generación de una trayectoria de aproximación

5.1 Estimación de la posición

En esta sección se van a comentar las transformaciones matemáticas necesarias para llevar a cabo la proyección de un punto de imagen a una posición del mundo y su correspondiente retroproyección ([3],[9]). Estas transformaciones se utilizan durante el proceso de identificación del palé.

Todas estas relaciones están definidas por el tipo de cámara, más concretamente, por la forma que tiene esta de captar la realidad. Para plasmar este proceso se suele emplear el modelo de cámara *pin-hole*.

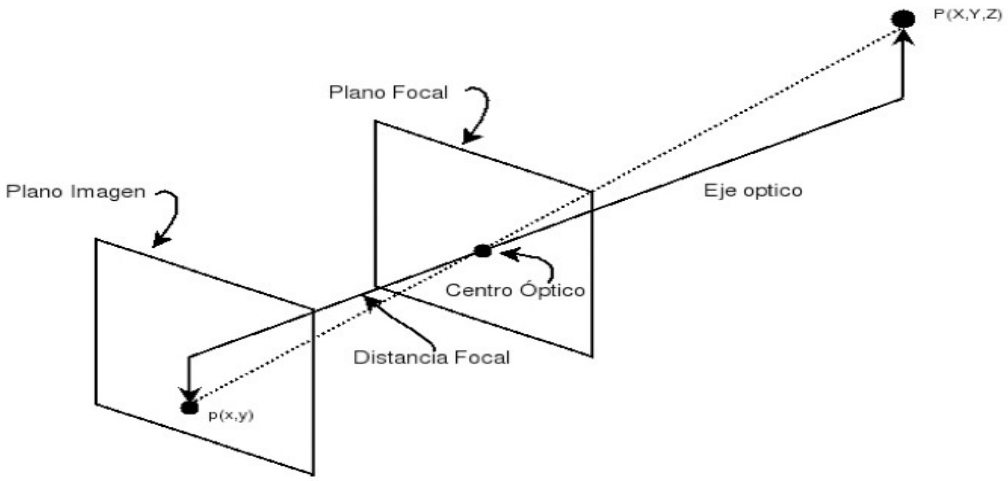

Figura 20: Modelo de cámara pin-hole

Para entender correctamente el funcionamiento es conveniente repasar algunos conceptos geométricos básicos. Estos nos permitirá construir sencillos modelos de cómo los objetos tridimensionales del mundo se proyectan en la cámara, dada la posición y orientación del robot respecto a un sistema de referencia conocido. A lo largo del texto iremos abordando diferentes problemas de distinta complejidad.

5.1.1 Cambios de sistema de referencia en 2D

Se comenzará repasando la geometría básica de los sistemas de referencia y las ecuaciones que permiten cambiar de uno a otro. Con este material se pueden modelar las distintas partes móviles de RobEx y sus relaciones con el fin de obtener sus

ecuaciones cinemáticas. Estas ecuaciones permiten definir varios sistemas de referencia, unos dentro de otros. Además, si queremos representar algo que el robot está viendo en su cámara, tendremos que ir desde el sistema de referencia de la cámara, al de la base del robot y, de ahí, al del mundo.

En primer lugar se van a obtener las ecuaciones permiten cambiar de un sistema de referencia *P,* situado en un plano, a otro *Q* situado en el mismo plano, conociendo el vector de traslación T que va desde *P* a *Q* y el ángulo que ha rotado *Q* respecto a P(l). Hay que especificar, por tanto, tres grados de libertad: las dos coordenadas del vector de traslación T y el *ángulo a* que se puede rotar respecto del eje vertical perpendicular al suelo

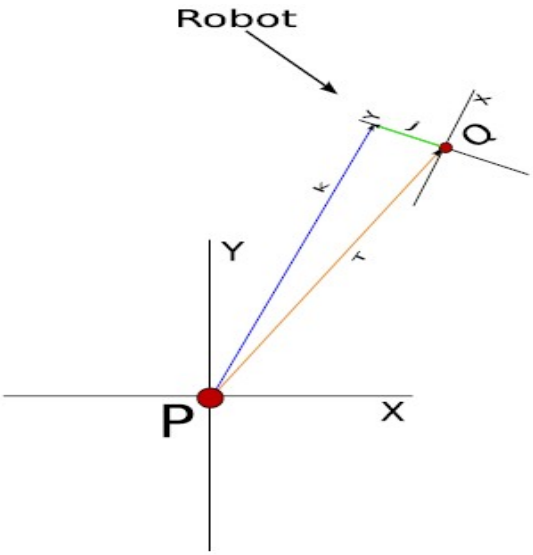

Figura 21: Cambio de sistema de referencia en 2D

Si se piensa en el robot en el laboratorio, podemos definir unos ejes (X, Y) en el centro alienados con las paredes. La posición del centro del robot vendría dada por las coordenadas (x, y) de algún punto definido como su centro y por el ángulo que esté girado respecto a una posición $a = 0$ inicial.

La primera ecuación que se obtiene es la forma directa y permite responder a la siguiente pregunta: ¿cómo se ve un vector *j* (0,1)definido en el sistema de referencia *Q,* desde el sistema de referencia P? La respuesta evidente es el vector A; que se obtiene sumando T y *j*. Recordemos que la suma de dos vectores definidos en el

mismo sistema de referencia se obtiene como el vector que va desde el inicio del primero al final del último. Es la regla del paralelogramo. Pero no podemos sumar T definido en P con *j* definido en *Q,* ya que obtendríamos *T* + (0,1) en *P* que es el vector amarillo. La solución es rotar primero *j* el ángulo *ay* y luego sumarlo a T.

Para rotar un vector en el plano un ángulo *ά* radianes, en el sentido contrario a las agujas del reloj, se puede utilizar una matriz de rotación R:

$$R=\begin{bmatrix}\cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha\end{bmatrix}$$

Se rotas un vector cualquiera *x* multiplicándolo por la derecha por esta matriz.

Así, se puede construir la ecuación matricial que proporciona las coordenadas de un vector definido en *Q,* visto desde *P:*

$$k=Rj+T$$

La segunda cuestión plantea la pregunta contraria, ¿cómo se ve un vector fc(x, y)definido en *P,* desde *Q?* Ya que se tiene la ecuación directa se puede usar el álgebra y despejar directamente *j:*

$$j=R^{-1}(k-t)$$

Se puede pensar en su interpretación geométrica para entenderlo mejor, (k - T) es el vector diferencia entre k y T, lo que según el gráfico se puede obtener como el vector que va desde la punta de T a la punta de k. Se obtiene utilizando la regla del paralelogramo, poniendo -T en lugar de T. El vector resultante es *j* pero definido en P. Para obtener *j* definido en Q todavía hay que rotarlo un ángulo alfa en el sentido contrario al que se aplico al principio, esto es - ά.

Para deshacer una rotación en 2D se multiplica el vector rotado por la matriz inversa de la rotación inicial. Aprovechando la propiedad de las matrices de rotación, por la que su inversa y su transpuesta coinciden para escribir:

$$j=R^{T}(k-t)$$

que es mucho más fácil de programar.

5.1.2 Cambio del sistema de referencia en 3D

En este apartado se va a generalizar la situación anterior a las 3 dimensiones para poder situar las cámaras que no están sobre el suelo y para poder referirnos a puntos que tampoco lo estén. Para esto es necesario utilizar matrices de rotación en 3 dimensiones. Estas matrices se pueden componer desde una rotación en 2 dimensiones si primero rotamos respecto del eje OX, luego del eje y OY, y finalmente, respecto del eje OZ. Esta sucesión de rotaciones se traduce en una multiplicación de matrices:

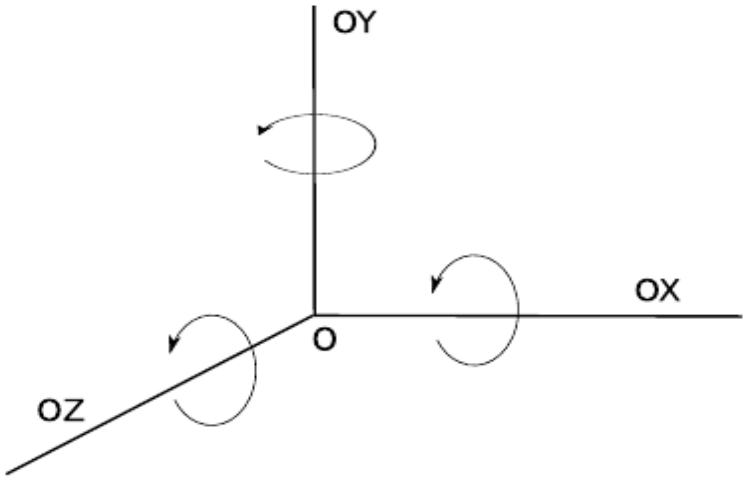

Figura 22: Rotación en 3D

Cada una de las rotaciones se puede describir como:

$$Eje\,OX\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha\\ ¿0 & \sin\alpha & \cos\alpha\end{bmatrix} Eje\,OY\begin{bmatrix}\cos\beta & 0 & \sin\beta\\ 0 & 1 & 0\\ ¿-\sin\beta & 0 & \cos\beta\end{bmatrix} Eje\,OZ\begin{bmatrix}\cos\gamma & -\sin\gamma & 0\\ \sin\gamma & \cos\gamma & 0\\ ¿0 & 0 & 1\end{bmatrix}$$

Las versiones 3D de las ecuaciones 2D de la sección anterior son exactamente iguales a como cabría esperar.

La forma directa:

$$k = Rj + T$$

y la inversa:

$$j = R^T(k - t)$$

Estas ecuaciones sistema referencia (SR en adelante) a otro, encadenando el resultado de una con la siguiente. De esta forma se puede representar la posición y orientación del robot y la posición y orientación de cada cámara. Para cambiar de uno a otro sólo hay que aplicar las ecuaciones correspondientes

5.1.3 Definición de los sistemas de referencia del robot

A partir de ahora se utilizaran sistemas de referencia 3D en todos los casos. El primer SR y más externo es el del mundo. Los ejes *X y Z* están sobre el suelo y el eje Y^+ apunta hacia el techo. En el robot, el primer SR es el de la base y tiene su origen en la proyección sobre el suelo del punto medio del eje que une sus ruedas motrices.

El eje Z^+ apunta hacia adelante y el eje X^+ hacia su derecha. En las cámaras, la configuración es la misma: el eje Z^+ hacia delante, el eje Y^+ hacia arriba y el eje X^+ hacia su izquierda.

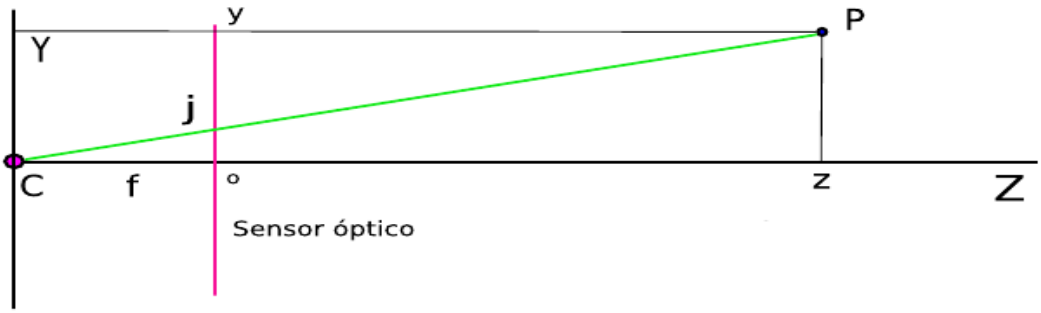

Figura 23: Geometría de la proyección

Con estos datos se puede calcular fácilmente como se ve un punto definido en el SR del mundo desde el SR de la cámara izquierda. Si el punto es $p_c = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, se tiene que $P_{cam} = \underbrace{R_{cam}^T(\overbrace{R_{base}^T(p_c - T_{base})}^{mundo \to base} - T_{cam})}_{base \to torreta}$

Mientras que la relación inversa, un punto visto desde el SR de la cámara al SR del mundo queda:

$$P_c = R_{base}(R_{camPcam} + T_{cam}) + T_{base}$$

Otro sistema de referencia importante es el de la ventana. Normalmente, las ventanas tienen el origen en la esquina superior izquierda con el eje Y^+ hacia abajo y el eje X^+ hacia la derecha. En este caso, la librería Qt permite convertir este SR en otro lógico de tal forma que se pueda dibujar sobre una ventana con las coordenadas que se quiera.

5.1.4 De 3D a 2D a través de una cámara

La secuencia de imágenes que proporciona una cámara está compuesta por cuadros o *frames* de un cierto tamaño y obtenidos cada cierto intervalo de tiempo o periodo. La característica más importante de estos cuadros es que son imágenes 2D obtenidas de un mundo 3D. Se puede modelar este proceso de *proyección* partiendo de una sencilla construcción geométrica en la que se representa al sensor óptico lateralmente por la línea roja, en realidad en una conversión 2D a 1D:

Este modelo de proyección se llama *pin-hole* (Figura 20: Modelo de cámara pin-hole) y asume que la luz llega limpiamente desde el punto del mundo *P* hasta el centro óptico *C* que también es el origen de coordenadas. Ese rayo óptico forma un triángulo con el segmento *Cz* y el segmento *Pz* que es semejante al formado por *C, j, O* por lo que se puede establecer la siguiente relación:

$$\frac{z}{f} = \frac{y}{j} \quad \text{que despejando queda:} \quad j = f\frac{Y}{Z}$$

Esto lleva a la ecuación de proyección básica en la que se ve cómo la profundidad *Z* del punto actúa dividiendo a la coordenada *Y.* La f es la distancia focal de la cámara que aquí debe expresarse en las mismas unidades que j. El mismo razonamiento se seguiría para la coordenada x si hiciéramos el dibujo anterior desde arriba en lugar de desde un lado, obteniéndose:

$$i = f\frac{X}{Z}$$

Las dos ecuaciones permiten obtener las coordenadas *(i,j)* de la imagen dada un punto 3D y una cámara situada en el mismo sistema de referencia. Aún faltan dos detalles para llegar a la ecuación que normalmente se usa. Primero, las coordenadas de imagen que se obtienen están centradas en cero. Eso implica que obtendremos valores negativos. Para obtener coordenadas de imagen con el origen en la esquina superior izquierda hay que trasladar cada coordenada sumándole la mitad del ancho o alto de la imagen, según corresponda. Agrupando todo esto queda la expresión

$$\begin{bmatrix} x \\ y \\ \lambda \end{bmatrix} = \begin{bmatrix} f & 0 & c_i \\ 0 & f & c_j \\ ¿0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

La coordenada λ se llama *proyectiva* y, de momento, sólo se utilizará para dividir por ella a *x* e *y* y así obtener la ecuación proyectiva inicial:

$$x = \frac{x}{\lambda} \quad y = \frac{y}{\lambda}$$

La matriz que se ha definido se llama *matriz de parámetros intrínsecos* porque define características propias de la cámara.

5.1.5 Rendering: proyección desde el mundo hasta la cámara

En los juegos con visualizaciones 3D y en otras disciplinas se utiliza el concepto de *rendering* como el conjunto de procesos que permiten proyectar la escena 3D sobre una cámara virtual. En general, el problema puede ser bastante complicado si se tienen en cuenta cosas como la selección de la región del mundo 3D que va a entrar en el campo visual, iluminación, sombras, texturas, etc. En nuestro caso se va a reducir el problema inicialmente a la proyección de puntos aislados. Resolver este problema equivale a contestar a la pregunta: ¿cómo se ve en la imagen de la cámara este punto del mundo p_C con coordenadas (*xyz)*. Uniendo los resultados de las dos secciones anteriores, se puede resolver fácilmente el problema: primero se calculan las coordenadas del punto en el sistema de referencia (todavía 3D) de la cámara y luego se proyecta ese punto utilizando la matriz de parámetros intrínsecos:

$$\begin{bmatrix} i \\ j \\ \lambda \end{bmatrix} = \begin{bmatrix} f & 0 & c_i \\ 0 & f & c_j \\ ¿0 & 0 & 1 \end{bmatrix} * R_{cam}^{T}(R_{base}^{T}(p_c - T_{base}) - T_{cam})$$

La transformación de SR's de la parte derecha de la ecuación genera un vector [x y z] que se multiplica por la matriz de cámara para realizar la proyección. Las coordenadas de imagen finales se obtienen dividiendo por λ la coordenada *i* e *j*. Con estas coordenadas se puede dibujar sobre la ventana en la que se esté viendo la imagen de la cámara.

5.1.6 Recuperando las coordenadas 3D de puntos del suelo mediante una cámara

Si bien proyectar un punto del mundo sobre la cámara no resulta difícil, un problema más complejo es realizar el camino contrario. Dado un punto/píxel de la cámara, obtener las coordenadas 3D [*x y* z] del punto del mundo que causó la proyección no es, ni mucho menos, tan fácil.

Como se vio, en el proceso de proyección se pasa de tres dimensiones a dos. La coordenada Z pasa dividiendo a las coordenadas X e Y por lo que, sin información adicional, no podemos recuperar esa información. Sin embargo, hay formas sencillas de proporcionar esa información. Una de ellas es hacer algunas suposiciones. Por ejemplo, si se supone que el punto que se esta viendo se encuentra sobre el suelo y se sabe dónde está el suelo, entonces se puede recuperar la información perdida en la proyección.

En la figura se puede ver esta situación en una vista lateral. Un punto sobre el suelo se proyecta en la cámara de focal *f* generando una coordenada *j*. Conocemos *h* y el ángulo $c = \pi/2$. El ángulo α necesario para despejar *L* es la suma de *c*, el ángulo conocido que forma la cámara con la vertical, y *b*, el ángulo que forma el rayo óptico (0,Z) con el eje óptico (0,f) y tomando *b* negativo hacia abajo. Se puede calcular *ba* partir de *f* y *j* sabiendo que *[0,j]* con *[f,j]* forman un ángulo recto:

$$b = atan(\frac{j}{f})$$

Así, se obtiene L aplicando el teorema del seno: $\frac{L}{\sin}(a)=\frac{h}{\sin}(d)=\frac{R}{\sin}(\frac{\pi}{2})$

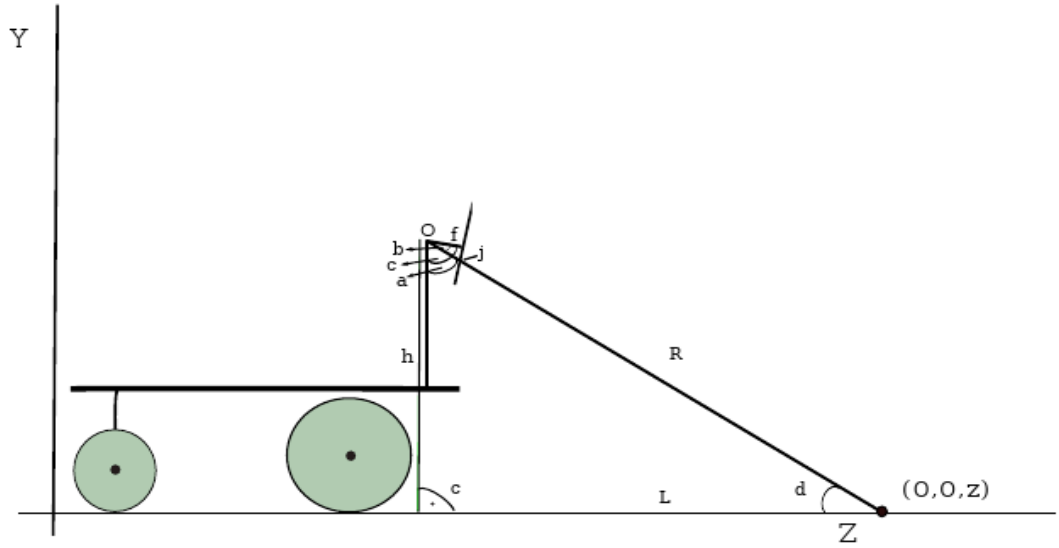

Figura 24: Geometría de un punto del suelo

Despejando L y sabiendo que d = π/2 – ά (los tres ángulos de un triángulo suman 180º en la geometría euclídea):

$$L=\frac{h\sin(a)}{\sin(\frac{\pi}{2}-a)}$$

Esto proporciona la coordenada Z del punto. La coordenada Y es cero por hipótesis (se esta resolviendo un punto sobre el suelo). La coordenada X se puede obtener fácilmente siguiendo un razonamiento similar.

Mediante estas transformaciones se obtiene un método matemático capaz de estimar la posición de un objeto en el mundo mediante su posición en la imagen y la operación inversa. Estas transformaciones se utilizan durante el proceso de estimación de la posición del palé sobre el suelo, en la obtención del modelo del palé en el mundo y también a la hora de llevar a cabo el proceso de retroproyección de los puntos durante la identificación. Estos procesos serán comentados en un apartado posterior.

5.2 Identificación del palé

Como la superficie frontal es muy pequeña, se ha pensado en colocar unas balizas de color en los tres travesaños del palé de forma que sirvan de referencia a la hora de analizar las imágenes que ve el robot, esto unido al conocimiento de las dimensiones del palé deben ser información suficiente para detectarlo.

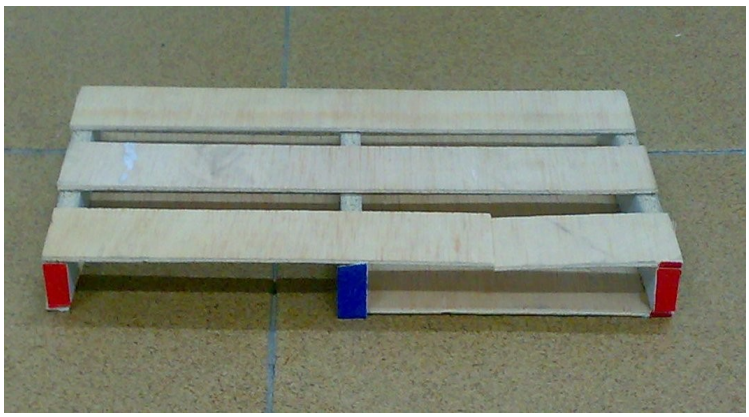

Figura 25: Palé con las balizas pegadas

Como se observa en la imagen anterior, la superficie del palé es muy pequeña lo que no contribuye en nada al proceso de detección, más aún si pensamos en la multitud de posibles cargas diferentes que puede soportar. La identificación ha de basarse única y exclusivamente en la zona frontal del mismo, zona que no va a ser nunca ocupada por una carga.

Hay que tener en cuenta además la influencia de la distancia en la capacidad de captar los colores por parte de la cámara. En las siguiente imágenes se muestra el palé situado a 80 cm, a 150 cm y a 300 cm. Queda de manifiesto la dificultad de la identificación.

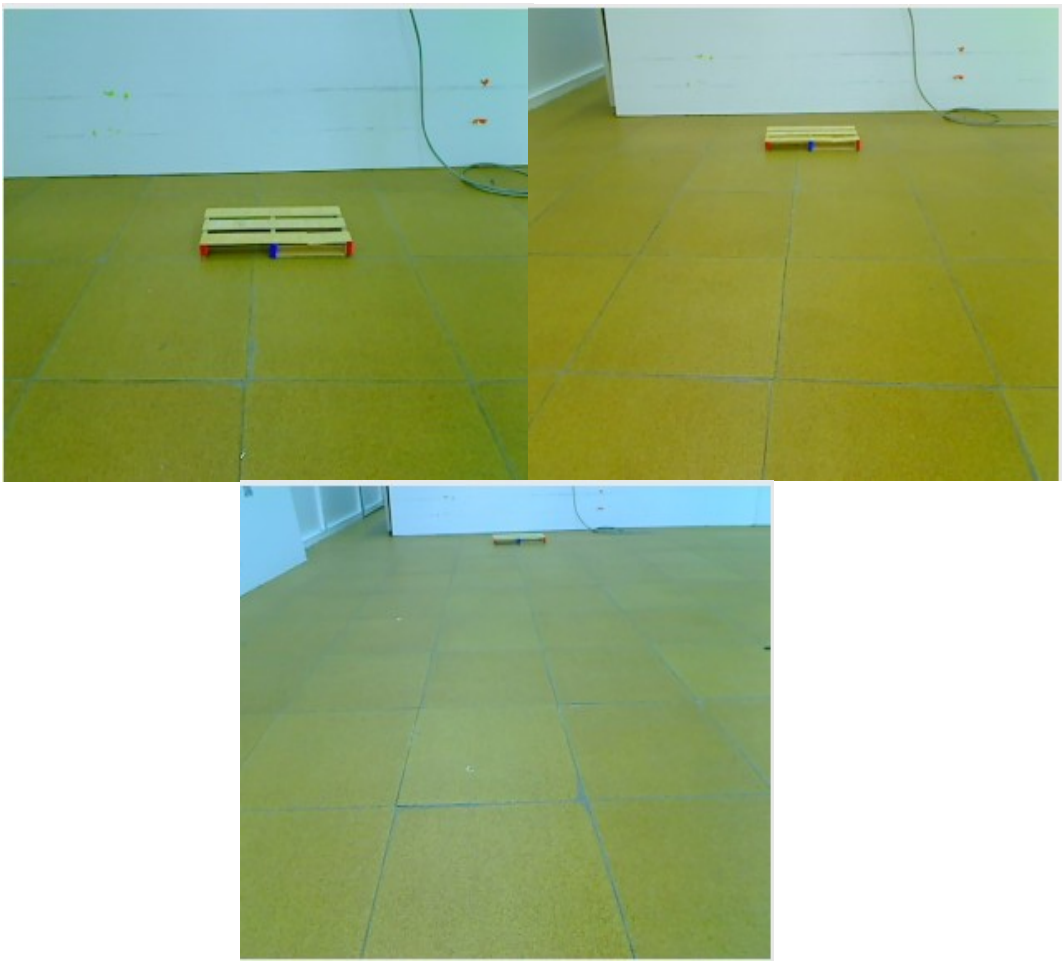

Figura 26: Visión del palé a diferentes distancias: 80 cm arriba izquierda, 150 cm arriba derecha y 300 cm abajo

La detección del palé se basa en la información proporcionada por las balizas de color presentes en la parte frontal del palé. Mediante esta información se generarán dos tipos de identificación diferentes: en primer lugar se transforma la posición de las balizas en la imagen a su posición en el mundo de forma que pueda utilizarse la información relativa a la estructura y dimensiones reales del palé, en segundo término se lleva a cabo la transformación inversa, es decir, suponiendo una posición del palé en el mundo, en que posición han de verse las balizas en la imagen.

Dicho esto, la segmentación del color de la imagen es una fase muy determinante en la correcta identificación del palé. Si durante esta segmentación se produce mucho ruido pueden aparecer falsos positivos, mientras que si se elimina demasiada información de la imagen pueden aparecer situaciones en las que no se "vea" el palé.

5.2.1 Segmentación de la imagen

En esta sección se van a comentar los modelos de color utilizados en el proceso de segmentación así como el diseño de los algoritmos que permiten obtener las zonas de color de las imágenes.

5.2.1.1 Introducción al color

El color en la robótica es una premisa sumamente útil porque a partir de él podemos extraer información del entorno del robot. El uso de colores permite diferenciar rápidamente entre distintos entornos o focos en los que centralizar la visión. Es muy común el empleo del color a la hora de establecer balizas que permitan a los robots conocer su posición o la acción que han de realizar.

Existen diferentes enfoques para la representación del color, estos son los modelos de color. El propósito de un modelo de color es facilitar la especificación de los colores en un sistema estándar. Los criterios establecidos así como la representación son completamente diferentes en cada uno de ellos por lo que la transformación entre los diferentes modelos no suele ser trivial.

En esencia, el modelo de color es un sistema de representación de coordenadas en 3-D. En este sistema, cada color viene representado como un punto (3-D) dentro del modelo.

El empleo de un modelo u otro viene determinado por las necesidades de los sistemas que lo van a emplear, así, el modelo RGB es usado en gráficos por computador, YUV o YcbCr se utiliza en sistemas de vídeo, PhotoYCC en entornos de fotografía o HSL en robótica.

La transferencia de información entre los diferentes sistemas casi siempre conlleva una necesaria transformación matemática.

De los diferentes modelos existentes solo se van a comentar dos, los empleados en el problema:

- Modelo RGB
- Modelo HSL

5.2.1.1.1 Modelo RGB

El modelo de color RGB es dependiente del dispositivo y suele ser usado por los monitores de televisión, ordenadores y la mayoría de las cámaras de gama baja.

Este modelo emplea las componentes primarias del color: rojo (R), verde(G) y azul(B) pero no trabaja de manera directa con magnitudes relacionadas directamente con la iluminación. A partir de la combinación de estos tres se obtiene el resto de colores, por lo tanto, hablamos de un modelo aditivo de color.

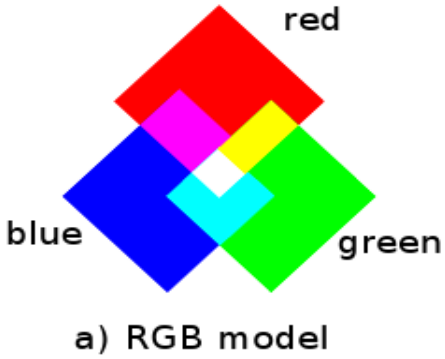

Figura 27: Modelo de color RGB

Mediante la adicción de estos tres colores primarios se genera toda la gama de color. Por ejemplo, para obtener un magenta puro, es necesario tener la máxima concentración de rojo y azul a la vez; la generación del blanco puro se consigue al concentrar los tres colores primarios con toda su intensidad mientras que el negro supone la ausencia de todos ellos. El valor de cada color en la mezcla viene determinado mediante un valor numérico. Aunque el intervalo de valores podría ser cualquiera (valores reales entre 0 y 1, valores enteros entre 0 y 37, etc.), es frecuente que cada color primario se codifique con un byte. Así, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255. Según esta escala, un rojo puro se correspondería con la codificación (255,0,0), el magenta comentado anteriormente con (255,0,255), mientras que (255,255,255) resulta el blanco puro y (0,0,0) el negro.

Representación geométrica de este espacio de color

La representación geométrica facilita la interpretación del modelo. El modelo RGB se representa mediante un cubo euclídeo (figura 13), donde las esquinas perpendiculares y no adyacentes son los tres colores primarios de la imagen, es decir rojo, verde y azul, la esquina más próxima al origen representa el color negro mientras que la más lejana se corresponde con el blanco. La escala de grises viene determinada por la diagonal que une las esquinas negra y blanca respectivamente.

Cada uno de los colores queda definido mediante el vector que une el punto del color y el origen

Según esta representación, una imagen puede verse como la unión de tres planos de color, correspondientes con los tres colores primarios.

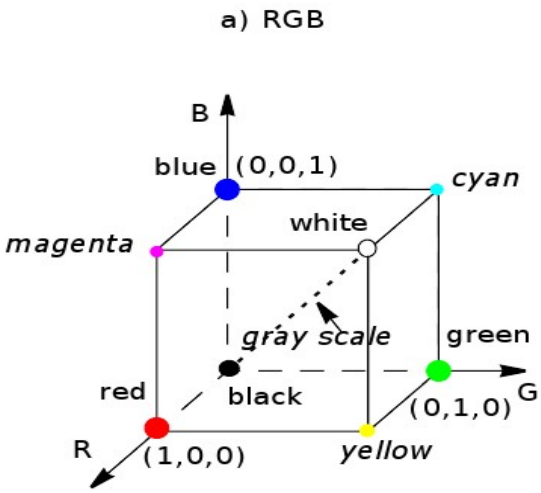

Figura 28: Representación geométrica del modelo de color RGB

El uso de este modelo presenta algunas ventajas, por un lado, se trata de un modelo sencillo de entender y de implementar y por otro, es el utilizado a la hora de representar los colores en la pantalla, está es una característica muy importante a la hora de utilizar el modelo para tareas de dibujo. Sin embargo, no resulta demasiado bueno a la hora de representar objetos del mundo real ya que el color mostrado depende en gran medida del sistema utilizado para recogerlo y las condiciones en ese instante

5.2.1.1.2 Modelo HSL

El modelo HSL es ampliamente usado en el mundo de la robótica porque presenta una gran robustez frente a los cambios de iluminación, característica muy importante a la hora de trabajar en entornos cambiantes.

El modelo de color HSL fue diseñado teniendo en mente el modo en que artistas y diseñadores gráficos denotan cada una de las características del color:

- *El tinte (hue o H):* es el tono del color (color en si mismo), se corresponde con los extremos del hexágono de la figura 14.
- *La saturación (saturation o S):* muestra la viveza de un tono determinado y se representa en la figura posterior como el radio del centro del cono.
- La intensidad (intensity o I): es la iluminación del color, su brillo, y se representa como la altura dentro del cono.

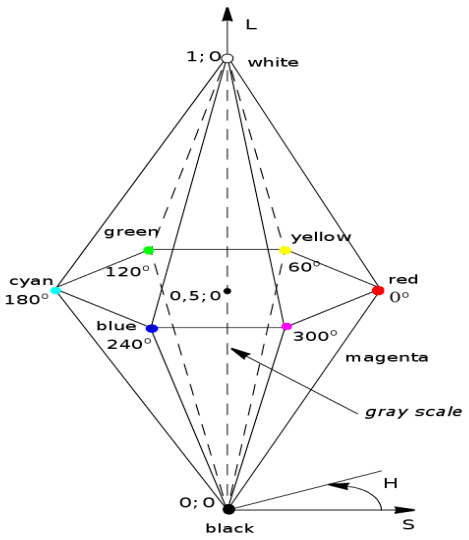

Figura 29: Representación geométrica del modelo color HSL

Como se observa en la figura anterior, la representación geométrica de este modelo es un doble cono. Se basa en coordenadas polares en 3-D, obteniéndose un espacio en forma de dos conos unidos en la base. Según esta representación, la punta superior del cono se corresponde con el color blanco, la punta inferior con el negro,

la unión entre ambas determina la escala de grises. Por último, los tonos puros del color se encuentran en la unión de la base de ambos conos, en esa unión puede definirse un hexágono cuyas puntas representan los colores más característicos de este modelo.

La característica más importante de este modelo de color es la separación del valor de la intensidad del resto de la información del color. de forma que esta se vea menos afectada ante los cambios en la iluminación de la escena.

5.2.1.2 Aplicación al problema

Como puede verse en las imágenes, el cambio de iluminación unido al autoenfoque de la cámara hace que los colores cambien mucho, parece mentira que el suelo sea el mismo viendo la representación del color en una y otra imagen. Esto proporciona una buena aproximación al problema a tratar.

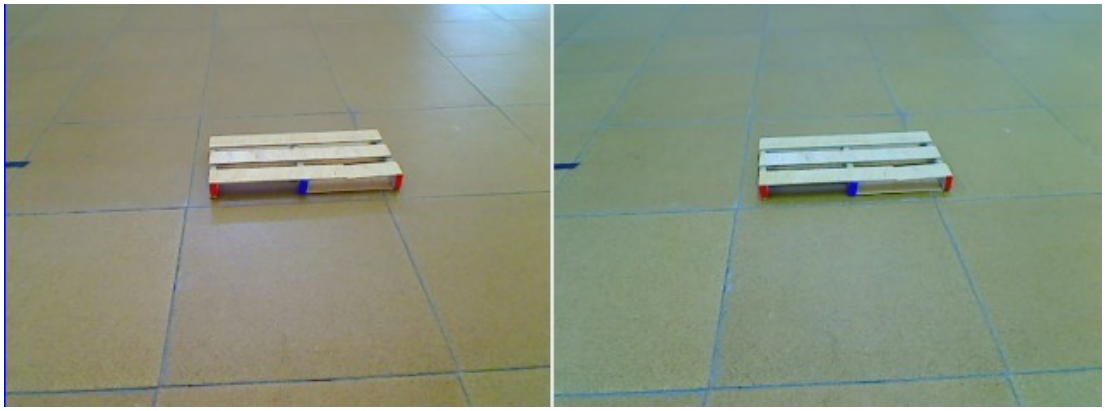

Figura 30: Cambio de iluminación

En primer término, se utilizó el espacio de color RGB. Mediante la definición de un filtro bastante permisivo se conseguía mantener de una forma más o menos constante los colores. Los problemas aparecían al aparecer en escena un mayor número de objetos, la permisividad del filtro definido hacía casi imposible determinar cuales de las zonas representadas se correspondían con las zonas buscadas por lo que se decidió emplear un modelo de color más robusto. El modelo escogido fue HSL, comentado anteriormente . Este modelo presenta una mayor "resistencia" a las variaciones de iluminación por lo que se podría definir un filtro mucho más estricto.

Hay que tener en cuenta que las balizas utilizadas tienen colores muy vivos, el uso del modelo HSL en otro tipo de colores no produce buenos resultados.

5.2.1.2.1 Transformación entre RGB y HSL

Las imágenes proporcionadas por el componente camaraComp vienen en el modelo RGB por lo que aparece el problema de la transformación de un modelo a otro.

En la siguiente tabla se muestra la representación de los colores más característicos en ambos modelos, la escala de saturación y luminosidad se establece en el rango (0 – 100) mientras que el valor de tono viene dado en (0 – 360).

Nombre	Muestra	RGB			HSL		
Rojo		255	0	0	0°	100%	100%
Amarillo		255	255	0	60º	100%	100%
Verde		0	255	0	120º	100%	100%
Cyan		0	255	255	180º	100%	100%
Azul		0	0	255	240º	100%	100%
Magenta		255	0	255	300º	100%	100%
Blanco		255	255	255	-	-	100%
Negro		0	0	0	-	-	0%

Tabla 5: Representación del color en diferentes modelos

A priori, aparecían diferentes soluciones para el proceso de transformación:

- Utilizar las librerías IPP de Intel. Se realizaría una trasformación de la imagen completa o al menos de una zona.
- Transformar el valor de los píxeles de la imagen uno a uno mediante las funciones de Qt
- Implementar una transformación manual.

Uso de las librerías IPP de Intel

En estas librerías existen funciones específicas para transformar una imagen en el modelo de color RGB al modelo HSL. La transformación mediante estas librerías

es muy sencilla, basta con rellenar los parámetros de la función de forma adecuada, para ello es necesario proporcionar la imagen RGB desempaquetada, esto no resultad ningún problema puesto que el componente "camaraComp" nos permite obtener la imagen en dicho formato.

Ventajas: Muy sencillo de utilizar, no hay más que hacer una llamada a la función.

Inconvenientes: Consume muchísima CPU. Además, no permite seleccionar la zona de imagen a tratar, es necesario transformar la imagen completa.

Uso de las funciones de Qt

Qt proporciona funciones para transformar un píxel en codificación RGB a la codificación HSV, esta codificación es similar a HSL por lo que podría utilizarse. Además, al realizar la transformación de píxeles uno a uno permite definir una zona de imagen a tratar con lo que aumenta el rendimiento.

Inconveniente: Al igual que ocurría con IPP, la transformación de los píxeles consume una gran cantidad de CPU.

Como estas dos opciones no parecían suficientemente satisfactorias se optó por implementar una transformación propia que permitiera optimizar el uso de los recursos.

Transformación manual

La transformación de un modelo a otro consiste, básicamente, en el empleo de una serie de fórmulas matemáticas. A pesar de que esto pueda parecer sencillo, se buscan las ecuaciones y se aplican, la realidad es que existen numerosas variantes sobre la transformación de un píxel en color RGB al correspondiente HSL. Después de diferentes pruebas, la transformación que ha aportado los mejores resultados es una combinación entre el método empleado en las librerías IPP[6] y en el proyecto "Sistemas de localización relativa basado en visión artificial" [7].

El resultado queda resumido en las expresiones siguientes, donde R, G y B son la cantidad de componente roja, verde y azul, respectivamente, medidas entre los

valores 0 y 1. I nos indica la intensidad luminosa, entre 0 y 1, H el ángulo del color, entre 0 y 2Л y S la saturación, entre 0 y 1.

$$L = \frac{(R+G+B)}{3}$$

$$S = 1 - \left(\frac{3}{(R+G+B)} \right) * min(R, G, B)$$

$$H = \arccos\left(\frac{\frac{1}{2}((R-G)+(R-B))}{\sqrt{(R-G)^2+((R-B)*(G-B))}} \right)$$

Como puede observarse, el proceso de conversión resulta costoso, implica varias operaciones en coma flotante y una relación trigonométrica por cada uno de los píxeles a transformar. Para aumentar la velocidad de computo se utilizará una tabla precalculada. La tabla funciona como una *look up table* o LUT (a partir de ahora "tabla HSL") y esta indexada mediante los valores primeros del color RGB, de forma que sabiendo el color del punto la conversión resultad directa. Por cada índice de la tabla se filtra el color RGB usando la conversión anterior y se guarda únicamente el resultado del filtrado. El ahorro de tiempo al filtrar la imagen completa es considerable al cambiar el calculo de expresiones complejas por la consulta de un valor en una tabla.

A estos valores HSL se les aplica un filtro de color, consideramos que pasan el filtro si los valores de saturación y tono están dentro de los márgenes máximos y mínimos del filtro, esto define una sección en torno al color, buscando con un cierto margen de tolerancia.

Sin embargo, estas componentes no son del todo independientes a la iluminación, por lo que esta también ha de tenerse en cuenta. Cuando los puntos están próximos al blanco o al negro (valores extremos de L), el rango de variación de la saturación baja y la precisión es cada vez mas pobre en cuanto al color. En estos casos no es posible discernir correctamente el color. Por este motivo se eliminan los puntos que presentan valores de iluminación cercanos a los límites, es decir, los colores muy oscuros o los excesivamente claros.

Por tanto, el filtro define una sección mucho más pequeña del doble cono HSL, a este cono se le seccionan la parte superior e inferior y se elimina gran cantidad del centro.

Para mejorar aún más el rendimiento, se va a emplear una segunda tabla LUT[3]. Al inicio del sistema, se utiliza la especificación del filtro del color para recorrer por completo la tabla LUT inicial aplicando dicho filtro y almacenando en la segunda tabla (tabla color) el valor del color encontrado. Dado que el modelo de color HSL determina seis tonos en el centro del cono, serán estos los colores admitidos, de esta forma, mediante la codificación RGB de un píxel accederemos directamente a una información de color, pudiendo ser: rojo, amarillo, verde, cyan, azul, magenta o ninguno de los anteriores.

Una de las premisas establecidas en el comienzo del proyecto consiste en procurar consumir la menor cantidad de recursos posibles, por ello y para evitar que el tamaño de ambas tablas sea excesivo se eliminan los dos bits menos significativos de cada canal de la codificación RGB. De esta forma se construye un índice de 18 bits en lugar de los 24 bits del píxel RGB. Esta operación reduce enormemente la cantidad de información que es necesario almacenar, reduce el tamaño de la tabla de unos 16 millones de posiciones a tan solo 260.000, sin que esto suponga una perdida demasiado significativa.

En el caso de la tabla HSL, los valores obtenidos de la transformación vienen en formato (0 – 1) para iluminación y saturación y (0 – 2Л) para el tono. Esto supone almacenar 3 valores reales lo que se traduce en:

$$3 * 32\text{bits} * 260.000\ posiciones = 3120000\text{bytes} = 3\ MB$$

Por ello, se lleva a cabo una transformación de los valores a la escala (0 – 255) de forma que sean almacenados como uint_8 o lo que es lo mismo mediante 8 bits, con lo que el tamaño final de la tabla resulta 756KB. Para la tabla color se lleva a cabo una operación parecida, en este caso, la codificación de los colores se realiza también en formato unit_8 (se desperdicia una gran cantidad de espacio ya que solo se necesitarían 3 bits, pudiendo utilizarse el resto para determinar la exactitud del color) con lo que se consigue reducir la cantidad de información a almacenar a

256KB esta es la memoria real que va a ocupar la codificación del color en tiempo de ejecución.

Todas estas transformaciones han sido encapsuladas en una clase denominada "tablacolor" de forma que el acceso a toda la información relativa al color quede correctamente agrupada. Dentro de esta clase también se conservan las operaciones del filtro RGB aunque ya no son utilizadas.

Uno de los pasos posteriores al desarrollo del proyecto consistirá en extraer toda la información que pueda resultar útil para otras tareas por lo que previsiblemente esta clase salga pronto del componente "paleComp" pasando a ser una clase independiente.

5.2.1.2.2 Obtención de las balizas

Mediante la interfaz de la clase "tablacolor" se dispone de un método que determina el color de un píxel de imagen de una forma rápida y eficaz. Es momento de abordar la segmentación de la imagen en busca de agrupación de color por zonas, es decir, elaborar una lista de posibles balizas.

Se pretende desarrollar un algoritmo que recorra la imagen una sola vez y agrupe los píxeles cercanos de un determinado color en una "baliza" de ese determinado color, devolviendo una lista de las balizas del color/es solicitado/s.

Como a lo largo de todo el proyecto, el objetivo es implementar dicho algoritmo de la forma más eficiente y genérica posible. De esta forma podrá ser utilizado para otros fines.

Como se han determinado 6 colores principales mediante el modelo HSL, son estos los colores de los píxeles a buscar. Cuando se utilizaba el modelo RGB el rango de colores se limitaba a los tres primarios.

En primer lugar es necesario determinar que píxeles han de considerarse "cercanos", pertenecientes a la misma baliza, o no. Para ello vamos a suponer que las balizas a buscar tienen una forma homogénea, en realidad la cercanía se va a establecer mediante la distancia euclídea por lo que la forma a buscar va a ser

siempre una circunferencia. Como es lógico, esta distancia se calculará entre el punto actual y el centro de la baliza a comprobar, a medida que se añaden puntos a una baliza resulta necesario aumentar el rango de distancia que determina si el punto pertenece o no.

El funcionamiento del algoritmo sería:

1.- Coger punto imagen
2.- Comprobar color del punto
    Si es un color buscado pasar al paso 3
    Sino volver al paso 1
3.- Obtener a lista de balizas de dicho color
    Si la lista esta vacía saltar al paso 10
4.- Sacar baliza de la lista
    si la lista esta saltar al paso 10
5.- Comprobar distancia al centro de la baliza
    si es mayor que el rango volver al paso 4
6.- Añadir punto a la baliza
7.- Recalcular el centro de la baliza
8.- Si es necesario incrementar el rango de distancia
9.- Volver al paso 1.
10.- Crear una nueva baliza con el punto como centro
11.- Insertar la baliza en la lista del color
12.- Volver al paso 1

Tabla 6: Algoritmo para la obtención de las balizas

Se aplica este algoritmo a todos los puntos de la imagen y como resultado se obtienen 6 listas, una por cada uno de los colores buscados.

Existen situaciones en las que este algoritmo no ofrece buenos resultados, cuando aparezcan balizas con formas “extrañas” o no dispongan de un “centro” alrededor del cual se encuentren el resto de los puntos. No se trata de un error es la implementación, simplemente se parte de una suposición, las balizas son homogéneas y de pequeño tamaño. El resto de caso no se contemplan.

De nuevo entra en juego la intención de optimizar el proceso, para ello se ha tenido en cuenta la posibilidad de realizar todo este proceso a una región de la imagen lo que puede suponer un gran ahorro en tiempo de computación. También se permite definir a priori cuales de los 6 colores son los que se quieren buscar, esto reduce considerablemente el proceso en entornos en los que se repita mucho un color que no sea necesario.

Al igual que ocurre con la clase “tabla color”, la funcionalidad de esta clase “baliza” se ha desarrollado de forma genérica por lo que puede tener otros usos en un futuro cercano, dejando de formar parte de este componente.

Una vez se dispone de la lista de balizas buscadas se puede comenzar la verdadera identificación del objetivo.

5.2.2 Identificación mediante información geométrica

En este proceso se utiliza la información de la estructura del palé. A priori se conocen las dimensiones del palé, esta son: 1 cm de alto, 15 cm de fondo y 30 cm de ancho. También se sabe que la distancia entre las balizas es de 15 cm.

Para codificar los palés se utilizan balizas de color como las mostradas en la figura 25. Se ha determinado que las balizas exteriores sean del mismo color mientras que la baliza central pueda ser del mismo u otro.

Teniendo en cuenta que se han clasificado 6 colores se dispone de una gran cantidad de combinaciones posibles. Como no es necesario el empleo de tal cantidad, se utilizarán los colores que mejor sean reconocidos, estos son: rojo, verde y azul.

Para simplificar el entendimiento del algoritmo se supondrá que el palé a buscar tiene una baliza roja en sus extremos y una azul en el centro. Por tanto, al segmentar la imagen se obtendrán las listas de balizas rojas y azules.

Antes de comenzar el algoritmo se comprueba que la lista de externos tenga al menos dos elementos y que la de centros disponga de al menos uno, en caso contrario es imposible que se este “viendo” un palé.

El funcionamiento del algoritmo sería el siguiente.

1. Tomar una baliza de la lista de externos (roja)
2. Transformar ese punto a su posición en el mundo mediante las ecuaciones de proyección comentadas al principio de este capítulo.
3. Tomar un segundo punto de la lista de externos (obviamente no el mismo).
4. Transformar ese punto al sistema de referencia del mundo.
5. Comprobar la distancia entre ambos, ha de ser unos 30 cm.
   - Si no es correcto volver al paso 3.
   - Si se trata del último de la lista se vuelve al punto 1, tomando

como primer extremo otra de las balizas.
6. Si se han conseguido unos posibles extremos, se calcula la recta que los une. Las balizas del palé forman siempre una línea recta.
7. Mediante la pendiente de la recta calculada se estima la posición que ha de ocupar el punto del centro, teniendo en cuenta que ha de estar a unos 15 cm de uno de los extremos.
8. Se escoge un punto de la lista de puntos centrales.
9. Se transforma dicho punto al sistema de referencia del mundo.
10. Se comprueba que la distancia entre el punto y la estimación generada mediante la recta no se desvía demasiado.
    - En caso de no cumplirse se vuelve al paso 8
    - Si es el último de los puntos se vuelve al paso 3.
11. Se lleva a cabo una comprobación adicional consistente en evaluar la distancia entre este posible punto central y cada uno de los extremos, que ha de ser de unos 15 cm.
12. Si se cumplen todas las restricciones se determina que se trata de un palé.

Tabla 7: Algoritmo para la detección del palé mediante información geométrica

Una vez concluido el proceso de detección y en caso de haber resultado positivo, se almacena la posición estimada del palé y se lleva a cabo la orientación de la torreta y la definición de la región de interés. Estas funciones son comentadas en el capítulo siguiente.

5.2.3 Identificación mediante retroproyección

En este caso se utiliza la información del palé a la inversa.

Este método parte de la idea de no necesitar las tres balizas para determinar que lo que se esta viendo es el palé, siempre que se haya visto anteriormente. Mediante la información del modelo generado, el movimiento de la cámara y de la base es posible estimar la posición que debe ocupar el palé en el instante siguiente. A esto se aplica la retroproyección para determinar en que posición de la pantalla han de estar las balizas.

Para ello se obtiene la posición de cada uno de los extremos del palé mediante la información almacenada en el modelo. Estos puntos son transformados a puntos en la imagen mediante el *rendering* comentado en al sección anterior.

Una vez se dispone de dicha información, se recorre la lista de balizas para determinar si alguna de ellas se corresponde.

Este método presenta la ventaja de no necesitar visualizar las tres balizas para determinar que lo que se esta viendo es el palé, lo que permite llevar a cabo el proceso de aproximación de una forma mucho más robusta.

Aunque alguna de las balizas salga de la imagen, por quedar fuera del campo visual o no ser detectada por no pasar el filtro de color debido a la iluminación, se dispone de información suficiente para determinar que se sigue tratando del palé.

5.3 Generación de la trayectoria

A la hora de llevar a cabo el desplazamiento hasta el palé es necesario tener en cuenta que no solo vale con llegar al punto en el que este el palé sino que el ángulo en que se lleva a cabo esa aproximación es crucial puesto que va a determinar la correcta entrada de las puntas del elevador en los huecos del palé.

Por tanto, dicho proceso no es ni mucho trivial. Es necesaria la generación de una trayectoria que permita encarar de frente el palé a una distancia suficiente.

Para ello una posible solución es aplicar las *curvas de Bézier*, sistema que permite generar una trayectoria curva pariendo de un punto origen, un punto destino y dos puntos de control.

5.3.1 Curvas de Bézier

Se denomina curvas de Bézier a un sistema que se desarrolló hacia los años 1960, para el trazado de dibujos técnicos, en el diseño aeronáutico y de automóviles.

Si en lugar de unir dos puntos con una recta se unen con una curva, surgen los elementos esenciales de una curva Bézier: los puntos inicial y final se denominan puntos de anclaje o nodos. La forma de la curva se define por unos puntos invisibles en el dibujo, denominados puntos de control, manejadores o manecillas[2][5].

En función del número de puntos de control obtenemos curvas de Bézier de diferentes grados, en la siguiente imagen se muestran algunas de ellas:

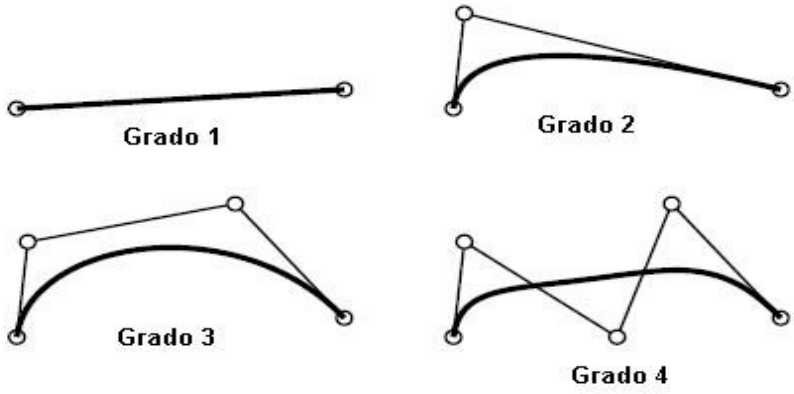

Figura 31: Ejemplo curvas Bézier

Cuatro puntos del plano P0, P1, P2 y P3 definen una curva cúbica de Bézier. La curva comienza en el punto P0 y se dirige hacia P1, finalmente llegad a P3 viniendo de la dirección del punto P2. Usualmente, no pasará ni por P1 ni por P2. Estos puntos sólo están ahí para proporcionar información direccional. La distancia entre P0 y P1 determina que "longitud" tiene la curva cuando se mueve hacia la dirección de P2 antes de dirigirse hacia P3.

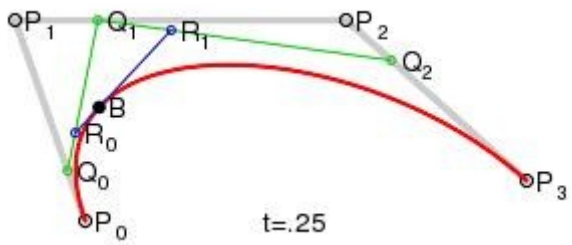

Figura 32: Curva cúbica

Para curvas cúbicas se pueden localizar puntos intermedios Q0, Q1 y Q2 que describen las curvas lineales de Bézier y los puntos R0 y R1 que describen las curvas cuadráticas:

La forma paramétrica de la curva cúbica es:

$$B(t) = P_0(1-t)^3 + 3P_1 t(1-t)^2 + 3P_2 t^2(1-t) + P_3 t^3, \quad t \in [0-1]$$

Las curvas de Bézier han sido ampliamente usadas en los gráficos generados por ordenador para modelado de curvas suaves. Como ejemplo de este tipo de usos tenemos la integración del método de Bézier en el lenguaje PostScript que permite el

desarrollo de sistemas de impresión de alta calidad desde el ordenador. La simplicidad del método y su facilidad de uso ha conseguido que se estandarice en el diseño gráfico, extendiéndose también a programas de diseño de páginas web como Adobe Flash, y retoque fotográfico como Photoshop, donde se usa para crear formas cerradas o selecciones. Aunque prima su utilización el apartado gráfico, existen numerosas aplicaciones, síntesis de ondas sonoras, diseño de piezas para automóviles o la descripción de los pasos para el movimiento de objetos en animaciones.

5.3.2 Aplicación al problema

Mediante el uso de las curvas de Bézier se ha generado una trayectoria de aproximación al palé de forma que se pueda llevar a cabo un anclaje con la orientación correcta.

El problema radica, esencialmente, en el ángulo de aproximación al punto de anclaje puesto que no basta solo con "llegar" con la suficiente precisión a dicho punto sino que además, el ángulo ha de ser el correcto puesto que sino la entrada de la punta de la pinza no sería la correcta.

Para la definición de la trayectoria a realizar se utiliza una curva cúbica de Bézier. El punto de origen de la trayectoria (P0) es el punto en el que se encuentra en robot hasta de comenzar el desplazamiento, este punto es obtenido mediante la lectura de la odometría proporcionada por la base. El punto final de la trayectoria (P3) se corresponde con el punto de anclaje del palé, punto en el que se ha de introducir la pinza, quedando listo para su alzamiento. Por tanto, quedan por definir los dos puntos intermedios (P1 y P2) que definan la curva. Estos dos puntos son lo que permiten generar una trayectoria de aproximación óptima.

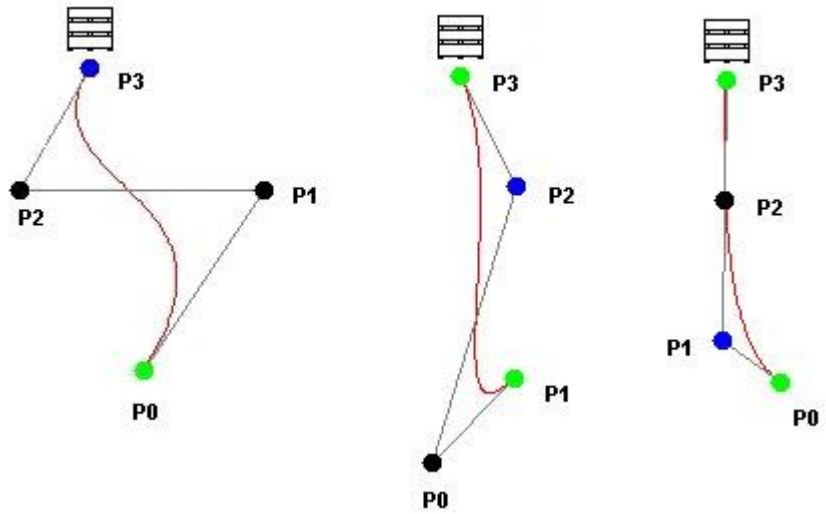

Figura 33: Curvas de aproximación

Tras probar diferentes posibilidades, se ha concluido que la disposición "correcta", la que genera una mejor trayectoria, resulta de colocar P1 y P2 en la perpendicular a la recta formada por el palé. De esta forma se obtiene una trayectoria que enfrenta el robot al palé al menos desde el punto P2 con lo que se consigue llegar al punto de anclaje con una orientación correcta.

Una vez determinada la posición que han de ocupar los puntos de control, lo que queda por establecer es la distancia ha la que han de situarse. Esta distancia no puede establecerse de una forma arbitraria. Si los puntos quedan definidos siempre en la misma posición pueden generar trayectorias inviables. Si establecemos los puntos cerca de la posición del palé, queda muy limitada la movilidad en los casos en los que el punto de arranque (P0) se encuentre muy distante. Por otro lado, si establecemos los puntos lejos del palé, las trayectorias incorrectas se producen al encontrarse el punto de arranque cerca más cerca, la trayectoria obligaría a retroceder, alejando al robot del palé, no siendo necesario. Por tanto, la posición de estos puntos ha de definirse de forma dinámica. Una buena aproximación consiste en establecer los puntos en función de la distancia entre el punto de arranque y el punto final. El punto P1 se encontrará a un 30 % de la distancia mientras que el punto P2 lo hará a un 80%.

Mediante esta disposición se consigue afrontar el palé con al orientación adecuada desde una distancia suficiente.

Al definir la curva de esta forma se llega a un compromiso entre las diferentes trayectorias que se deberían generar, caso de que no fuera necesario realizar un giro demasiado brusco (el palé se encuentra casi frente al robot) y el caso opuesto (la posición del palé requiere de un giro de más de 90º). Este compromiso lleva a afrontar algunas de las trayectorias de una forma demasiado "brusca".

Ejemplos de la disposición de los puntos y trayectorias obtenidas:

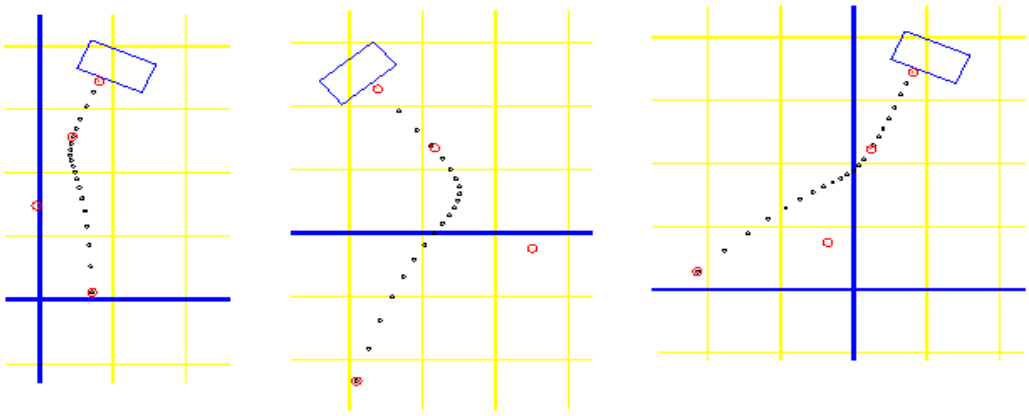

Figura 34: Trayectorias reales

En la figura anterior se observan ejemplos de las curvas generadas. Los puntos representados en rojo se corresponden con los puntos P0, P1, P2 y P3, mientras que el resto de puntos conforman la curva generada.

6 Capítulo 6: Especificación detallada del plan

Dentro de este capítulo se abordan en profundidad todas las tareas que aparecieron durante la elaboración del plan de acción.

Para cada una de ellas se comentará su cometido principal así como la forma de abordarlo. En muchos casos se trata de tareas inter -relacionadas por lo que separar sus funcionalidades no resulta sencillo.

Para simplificar el entendimiento y seguimiento de las descripciones, las diferentes tareas se van a englobar dentro de cuatro grandes grupos, las encargadas de realizar movimientos, las relacionadas con la pinza, las encargadas de "comprender" lo que se esta viendo y por último, un cuarto grupo en el que se engloban las tareas independientes. De esta forma la clasificación quedaría:

- Relativas al movimientos
  - Ir al objetivo
  - Retroceder
- Movimiento pinza

    - Leer posición actual
    - Subir pinza
    - Bajar pinza
- Análisis de la imagen
    - Identificación del objetivo
    - Localización objetivo
- Tareas independientes
    - Borrar trayectoria
    - Seleccionar nuevo objetivo
    - Eliminar historial de posiciones
    - Recolocar la torreta

6.1 Funciones relacionadas con el movimiento

Para simplificar el entendimiento se seguirá determinando como "objetivos" tanto al palé como a la zona de destino del mismo. Realmente, en el proceso de ir hacia el palé o hacia el destino de este, tan solo varía la función encargada de identificar y posicionar en el mundo tanto el palé como la zona de destino.

- Aproximación al objetivo
- Retroceder

6.1.1 Ir al objetivo

Se trata de una de las tareas más completas del proceso, además de intervenir el proceso de identificación, se lleva a cabo una tarea de estimación de la posición y de control del movimiento del robot.

A medida que se produce el movimiento se va reestimando la posición del palé, a medida que nos acercamos la estimación va mejorando) lo que implica una regeneración de la trayectoria de aproximación a la vez que se controla el movimiento de la base para asegurarnos de que sigue dicha trayectoria.

Realmente todas las funciones son llevadas a cabo por la tarea "aproximación" por lo que será esta la tarea comentada.

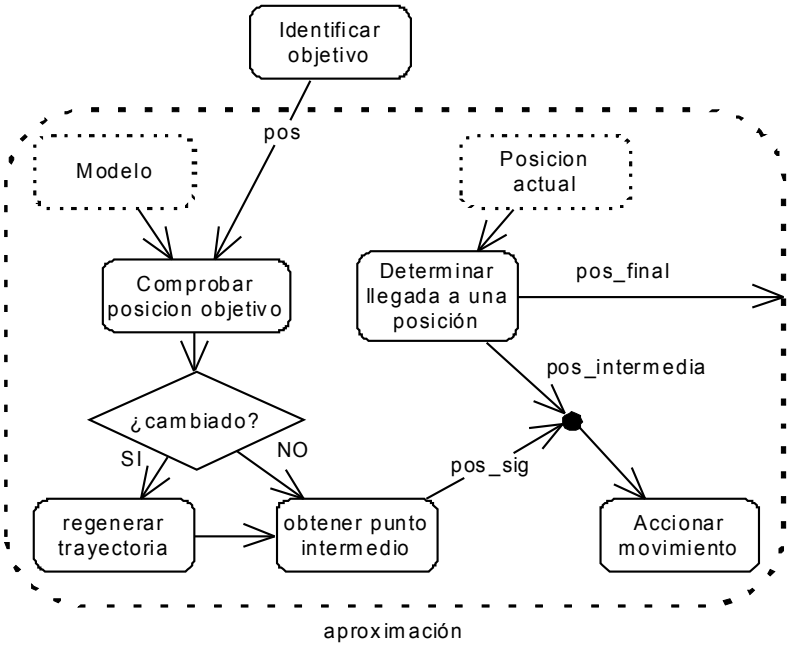

Figura 35: Proceso de aproximación

Debido a la gran cantidad de tareas a llevar a cabo, estas han sido clasificadas por separado y se comentan a continuación:

- Identificar objetivo.
- Comprobar posición objetivo
- Regenerar la trayectoria
- Accionar el movimiento de la base
- Obtener el siguiente punto intermedio
- Determinar la llegada a una posición

La tarea de identificación, a pesar de formar parte del conjunto de tareas a realizar, no se comentará en esta sección puesto que se ha creado una sección específica para ella.

6.1.1.1 Comprobar posición objetivo

En cada ciclo se obtiene la estimación de la posición actual del objetivo en función de la información obtenida mediante la cámara. Esta información es acumulada de forma que se va auto actualizando la posición, sin embargo, para evitar que el movimiento del robot presente oscilaciones, no siempre se regenera la trayectoria a realizar. Se va almacenando un historial de posiciones (modelo), mediante este historial se comprueba la variación registrada entre la posición con la

que se generó la trayectoria y la posición actual, si estas varían considerablemente (más de 2 cm) se regenera la trayectoria.

Gracias a esta forma de actuar, se consigue ir reestimando la posición del palé (se presupone que a medida que nos encontremos más cerca la estimación de la posición será mejor) y a la vez se evitan movimiento demasiado bruscos.

6.1.1.2 Elaborar trayectoria

Para elaborar la trayectoria de aproximación se utilizan las curvas de Bézier (comentadas en el capítulo 5). Utilizando la estimación de la posición del palé y la información procedente de la odometría de la base se genera una curva de aproximación que asegura llegar con el ángulo correcto.

6.1.1.3 Obtener el siguiente punto intermedio

Una vez determinada la trayectoria a recorrer, es necesario pasar dicha información a la base. Como se ha comentado en un apartado anterior la base dispone de un método encargado de llevarla hasta un punto dado, no es posible indicarle una ruta completa por lo que es necesario emplear el método del "burro y la zanahoria". Al comenzar el movimiento se le proporciona a la base el primero de los puntos de la trayectoria (t = 0.1 en la curva de Bézier), se va comprobando la distancia que queda hasta dicho punto y antes de alcanzarla se adelanta la zanahoria. Gracias a este método se consigue trasladar la trayectoria a la base de una forma sencilla y además el movimiento realizado por esta es bastante suave puesto que los cambios de trayectoria se abordan con una mayor antelación.

6.1.1.4 Accionar movimiento de la base

Esta acción consiste en comunicar a la base el siguiente punto al que debe dirigirse, este punto es obtenido mediante la función anterior.

6.1.1.5 Determinar la llegada a una posición

Determinar esta posición resulta bastante sencilla puesto que se dispone de la información de la odometría proporcionada por la base.

Dentro de esta función se llevan a cabo dos comprobaciones: si se ha llegado al punto final de la trayectoria, ya se ha introducido la pinza en el palé o si se ha llegado al siguiente punto intermedio de la trayectoria.

En primer lugar se comprueba la llegada al punto final (anclaje del palé), se obtiene la estimación del punto en el que se encuentra el palé y se calcula la distancia euclídea entre la posición actual y dicho punto, estableciendo un margen de error razonable. En caso de haber completado la trayectoria se sale de esta etapa del plan, pasando a accionarse la pinza.

En caso de no haber completado la trayectoria se comprueba la llegada hasta un punto intermedio, al igual que en el caso anterior, se utiliza la distancia euclídea, en caso de haber alcanzado dicho punto intermedio se envía la posición del siguiente punto a la base.

6.1.2 Retroceder

Esta acción representa un movimiento hacia atrás de longitud más o menos la profundidad del palé. Aunque pueda parecer una acción sin sentido es totalmente necesaria, se emplea una vez se ha izado el palé (se asegura que el robot pueda llevar a cabo cualquier movimiento libremente sin preocuparse de que el palé estuviera encajonado entre otros objetos) o bien se ha dejado en su destino (es necesario retroceder puesto que si gira puede arrastrarlo mediante la pinza).

6.2 Análisis de la imagen

Dentro de esta sección se ha englobado, además de la propia tarea de identificación, la localización del objetivo. Realmente en el proceso de localización también intervienen los movimientos de la base y la torreta pero como su función principal es obtener la primera estimación de la posición del palé se ha decidido incluirla aquí.

6.2.1 Identificación del objetivo

Igual que ocurre con la aproximación, la identificación de un objetivo conlleva la ejecución de varias acciones.

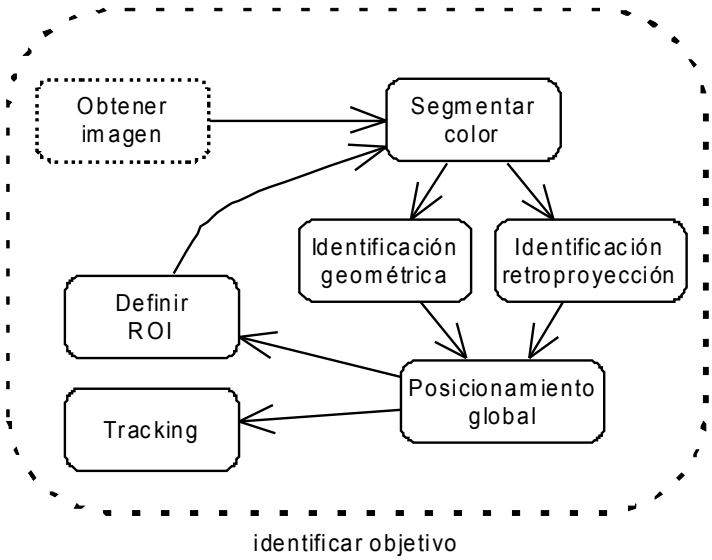

Figura 36: Proceso de identificación

La obtención de la imagen es una llamada al método correspondiente de la clase cámara por lo que no se va a comentar, aparece en el esquema para facilitar el entendimiento global.

La carga computacional de esta fase recae en las tares de "identificación geométrica" e "identificación mediante retroproyección", estas tareas no van a ser comentadas puesto que han sido ampliamente tratadas en el capítulo anterior.

Una vez obtenida la estimación de la posición existen una serie de tareas a realizar para almacenar dicha información y facilitar posteriores localizaciones:

- Almacenar la estimación de la posición del objetivo (posicionamiento global)
- Seguimiento del objetivo mediante la torreta (tracking)
- Acotar la zona de imagen a buscar (definir ROI)

6.2.1.1 Almacenar la estimación de la posición del objetivo

Mediante cada uno de los frames en los que se consigue identificar el palé se obtiene una estimación de la posición de este en el mundo real, esta estimación se consigue a través de las transformaciones matemáticas comentadas en el capitulo 5.

A medida que disminuye la distancia al palé, durante el proceso de aproximación, la veracidad de estas estimaciones va aumentando, vemos mejor a

medida que nos acercamos. Estas variaciones han de ser añadidas a la estimación global de la posición de manera que sea cada vez mas exacta.

Para que el modelo no se vea demasiado influencia por una mala estimación (pueden producirse errores debido a cambios de iluminación, falsos positivos...), se aplica un factor de corrección, aportando un mayor peso a la información existente que a la nueva.

La información de la posición es utilizada durante el proceso de aproximación y también en la identificación mediante retroproyección.

6.2.1.2 Tracking

Debido a los múltiples cálculos necesarios para establecer al proyección y retroproyección de un punto (comentados en el capitulo 5) y también para asegurar el correcto seguimiento del palé durante el proceso de aproximación, se lleva a cabo un tracking.

Este tracking trata de centrar el palé en la imagen mediante el movimiento del *tilt* y el *pan*. Con esto se consigue aumentar el tiempo que se "ve" el palé y reducir el error introducido por las transformaciones matemáticas.

En el repositorio de RoboComp existe un componente encargado de llevar a cabo este tipo de trackings pero por simplicidad se ha preferido no incluirlo ya que la tarea ha llevar a cabo es bastante simple.

6.2.1.3 Acotar la zona de imagen a evaluar

Una vez que se ha conseguido ver el palé y disponiendo de la información procedente de la odometría de la base y el movimiento de los servos de la cámara, no tiene demasiado sentido tener que reevaluar por completo cada uno de los frames que se van obteniendo puesto que se puede predecir la zona de la imagen en la que debe encontrarse el palé.

Esta función se encarga de ir redefiniendo esta zona con lo que se reduce considerablemente la cantidad de píxeles de imagen a trata lo que se traduce en una

disminución del coste computacional de los algoritmos implicados en el proceso de identificación del palé.

6.2.2 Localizar objetivo

El proceso de localización se centra en obtener la primera estimación de la posición del objetivo para poder comenzar el proceso de aproximación.

La parte más importante del proceso de localización es la identificación de la imagen, pero como esta tarea ya ha sido comentada en la sección anterior no se hará mención alguna.

A pesar de esto, existen algunas subtareas necesarias:

- Mover torreta
- Mover base

6.2.2.1 Mover torreta

Para aumentar las posibilidades de que el palé entre en la zona cubierta por la cámara, se lleva a cabo un movimiento secuencial de esta. A medida que avanza el proceso de localización la cámara va describiendo dos movimientos simultáneos: uno vertical y otro horizontal.

Mediante estos dos movimientos se consigue cubrir un espacio mucho más amplio lo que acorta el tiempo necesario para llevar a cabo la detección del palé.

6.2.2.2 Mover base

A la hora de llevar a cabo la localización se dispone de información acerca de la zona del mundo en la que se encuentra el palé.

Esta información viene impuesta de la imposibilidad de reconocer el palé a una distancia superior a 3 metros, debido al pequeño tamaño de este y la resolución de las imágenes captadas por la cámara.

Como se dispone de la información acerca de una posible posición en la que se encuentre el palé, esta es comunicada a la base para que se dirija a dicha zona

Cuando se produce una identificación positiva, se anula el movimiento de cámara y base cediendo su control a la etapa posterior, la aproximación.

Hay que aclarar que una identificación positiva no es simplemente una única detección correcta del palé sino que es necesaria la correcta detección en un conjunto de frames consecutivos lo que elimina la posibilidad de detectar falsos positivos.

6.3 Tareas independientes

Aquí se han englobado tareas muy diferentes entre sí. Cada una de ellas tiene un cometido simple y claro.

Se trata de las funciones con menos carga de trabajo, eso sí indispensables para la correcta ejecución del plan.

- Borrar trayectoria
- Seleccionar nuevo objetivo
- Eliminar la región de interés
- Recolocar la torreta

Borrar trayectoria

Una vez que ha finalizado el proceso de aproximación correctamente, es necesaria la eliminación de la trayectoria generada.

Seleccionar nuevo objetivo

Los objetivos a cumplir, tanto palés como destinos de estos, se almacenan en una lista ordenada de forma que al finalizar uno es necesario eliminarlo de la lista y obtener el siguiente.

Eliminar la región de interés

Al comenzar la búsqueda de un nuevo objetivo no se dispone de información acerca de la zona de la pantalla en la que se espera encontrarlo por lo que resulta necesario evaluar toda la imagen obtenida, la región de interés pasa a ser toda la imagen.

Recolocar la torreta

Al finalizar la tarea de aproximación, ya sea hacia el palé o hacia la posición de destino de este, la disposición de la torreta queda claramente apuntando hacia el suelo lo que reduce considerablemente la amplitud del campo visual. Por ello, antes de comenzar la búsqueda de un nuevo objetivo es necesario llevar a cabo este proceso de recolocación.

6.4 Movimiento pinza

Estas tareas se basan en el componente desarrollado durante la elaboración del proyecto, las funcionalidades definidas para el han sido comentadas en el capítulo 3 de esta documentación.

Las únicas tareas a realizar consisten en leer la posición de la pinza y enviar una nueva orden de movimiento en caso de no encontrarse en la posición correcta, en caso contrario no se lleva a cabo ninguna tarea.

La ejecución de las mismas es simplemente una llamada a los métodos definidos mediante el interfaz del componente.

Lo único destacable es la inclusión de una nueva función encargada de comprobar si es posible el movimiento. Esta función no se encuentra actualmente implementada. La idea es poder detectar aproximaciones incorrectas, por ejemplo, que al accionar la pinza pueda detectarse si ha sido introducida correctamente en el palé o al ir a dejarlo en su destino detectar que esa zona se encuentra ocupada, no siendo posible dejarlo ahí.

7 Capítulo 7: Experimentos

La experimentación va a consistir en situar el palé y el destino del mismo en diferentes posiciones y comprobar como resulta de eficiente la ejecución del plan.

Junto con esta documentación se entregan una serie de vídeos e imágenes que reflejan el proceso de desarrollo del proyecto así como estos experimentos.

Las pruebas se van a englobar en las tres tareas principales llevadas a cabo, esto es: detección del objetivo, aproximación a él y ejecución del plan.

7.1 Detección

Las pruebas de esta fase consisten en ubicar el palé en diferentes posiciones y comprobar que la detección se lleva a cabo correctamente, además se intentará modificar las condiciones de luz para ver si resulta robusto.

En la siguiente imagen se observa la identificación mediante información geométrica a la izquierda y a su derecha una imagen en la que no se ha detectado correctamente una de las balizas por lo que no se puede detectar correctamente el palé, pero se utiliza la otra técnica de identificación.

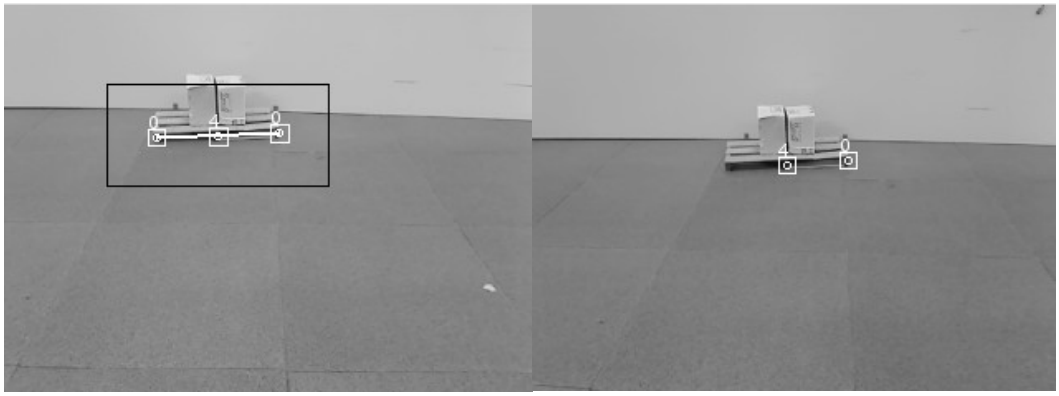

Figura 37: Detección geométrica (izquierda) y detección retroproyección (derecha)

A continuación se muestran diferentes posiciones del palé y como son vistos desde el robot:

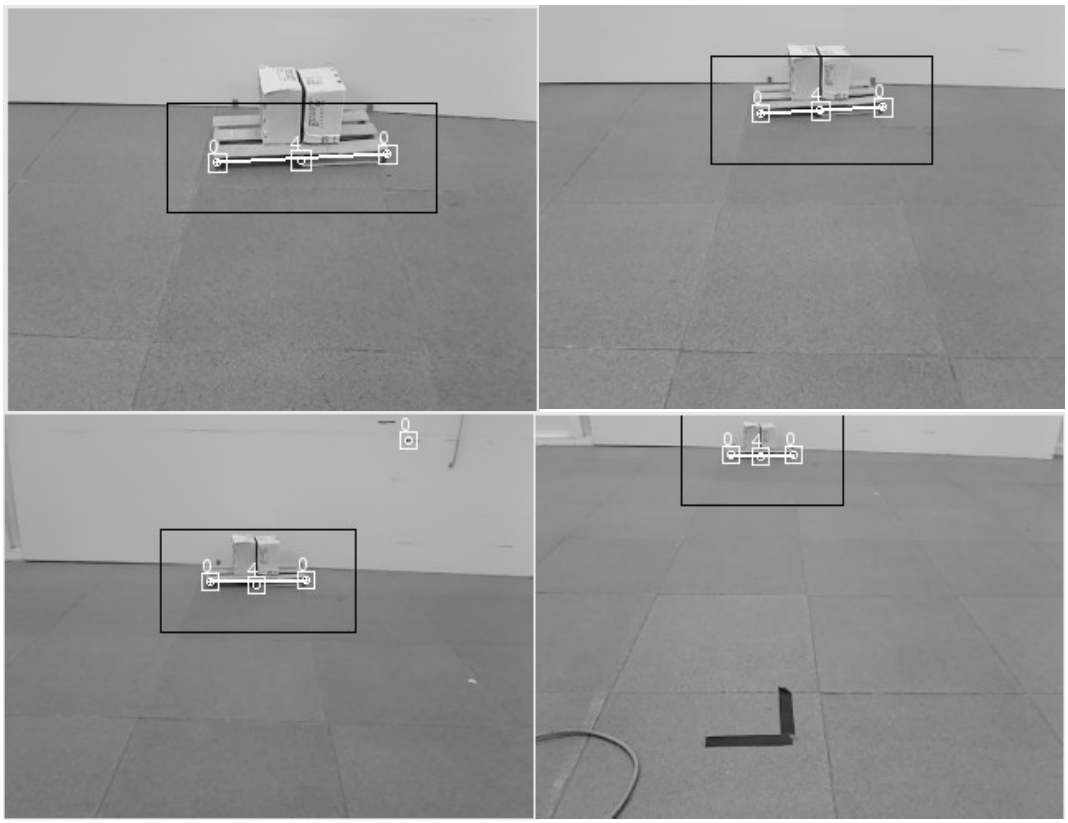

Figura 38: Detección desde diferentes distancias: 80 cm arriba izquierda, 130 cm arriba derecha, 170 cm abajo izquierda y 240 cm abajo derecha

Durante la experimentación se ha observado que se obtienen mejores resultados por la mañana, cuando al luz es más intensa, que por la tarde. Esta variación se debe a la obtención de las balizas de color de la imagen. Cuando la luz baja, resulta más

difícil detectar dichas balizas por lo que la identificación del palé se lleva a cabo más tarde lo que se traduce en una menor capacidad de maniobra a la hora de generar la trayectoria de aproximación. Además, durante este proceso de aproximación se necesita seguir "viendo" el palé de forma que las estimaciones generadas vayan convergiendo en la posición real del mismo, si a lo largo de este proceso son muchas las ocasiones en la que no se consigue detectar el palé, la estimación global obtenida será mala con lo que probablemente la aproximación fracase.

Otra de las pruebas ha consistido en la ejecución del plan completo, situando una carga encima del palé. Como la única zona utilizada para la identificación es la frontal, la aparición de esta carga sobre el palé no ha supuesto ningún impedimento para la correcta ejecución.

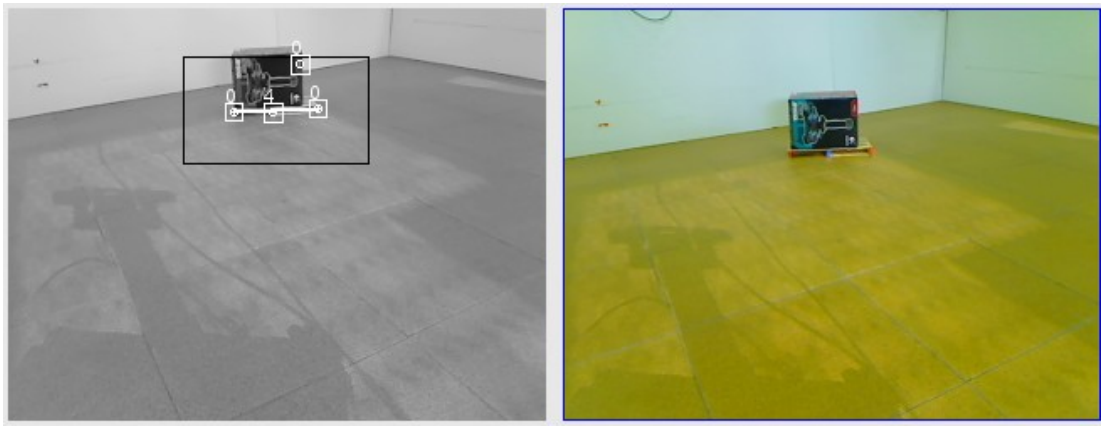

Figura 39: Detección del palé con una carga

En la siguiente imagen se utiliza la misma carga, pero modificando la orientación de forma que aparece una zona roja (el código numérico que acompaña la zona indica el color de esta, utilizando el 0 para el rojo y el 4 para el azul), esta zona impide la correcta detección de las balizas. El gran tamaño que tiene consigue absorber las balizas como si se tratasen de la misma zona. En futuras versiones se deberá repasar la generación de las balizas estableciendo algún tipo de limitación para intentar evitar estas situaciones.

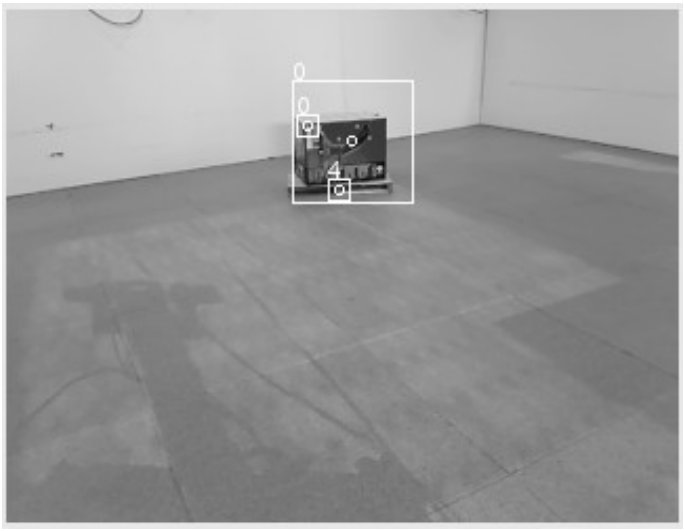

Figura 40: Detección incorrecta de las balizas

Otra de las funciones a evaluar durante la detección es la encargada de realizar el tracking. En la siguiente imagen puede verse como al detectar el palé se modifica la posición de la cámara para centrarlo en la pantalla.

Figura 41: Movimiento de la torreta para centrar el palé

7.2 Aproximación

Realmente estos experimentos resultan difíciles de reflejar en el papel. Su seguimiento es mucho más sencillo observando los vídeos que acompañan la documentación.

Existen situaciones en las que no es posible alcanzar la posición del palé con la orientación adecuada. Esto se debe principalmente al compromiso llevado a cabo durante la generación de la trayectoria, entre las trayectorias rectas (evitar giros

innecesarios) y las curvas (obtener curvas más pronunciadas). En posteriores versiones debería incluirse un método capaz de detectar estas situaciones de forma que en lugar de intentar llevar a cabo la aproximación se realizase un movimiento en sentido opuesto, el cual ampliase el radio de giro del robot permitiendo obtener una trayectoria correcta. Un ejemplo de este tipo de situaciones se muestra en la imagen siguiente:

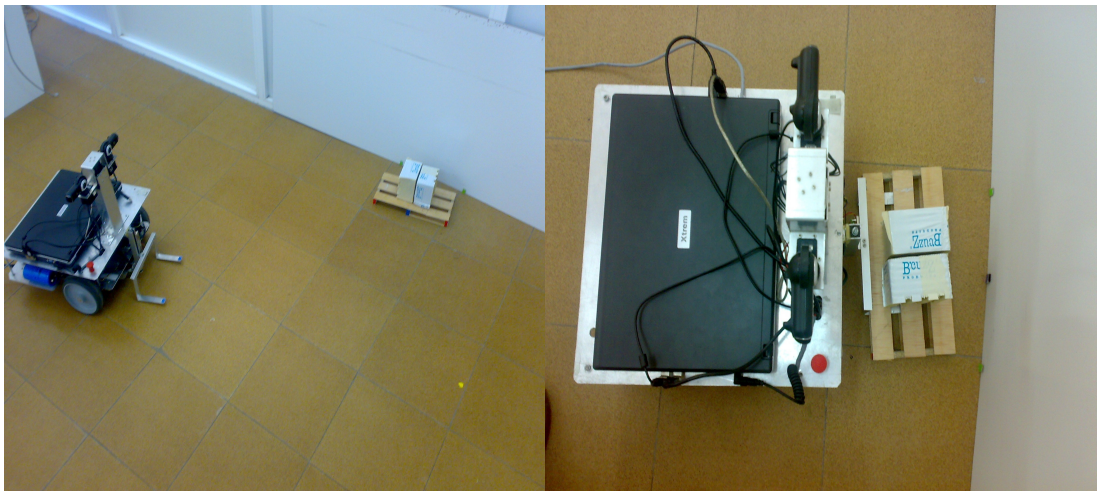

Figura 42: Aproximación al palé. Izquierda punto de comienzo. Derecha posición final

Como cabría esperar, el sistema responde mejor cuanto menos brusco sea el giro a realizar durante el proceso de aproximación. Si es necesario llevar a cabo un giro muy brusco y no se dispone de la distancia adecuada aparecen ocasiones en las que la aproximación no se lleva a cabo correctamente.

En algunas ocasiones no se estima bien la posición del palé, se le ubica más alejado de lo que realmente esta. Esto provoca un cierto desplazamiento del palé a la hora de alcanzar el punto de anclaje puesto que se ha colocado más lejos. El desplazamiento producido no es mucho, apenas unos centímetros, pero puede suponer un gran problema si el punto de origen del palé o el de destino están pegados a una pared puesto que se produciría una fuerza de empuje del robot contra el palé lo que puede romperlo o desestabilizar la carga.

Como ejemplo de una correcta ejecución se adjunta la imagen siguiente, en este caso RobEx consiguió detectar el palé correctamente, ir hasta él, izarlo, llevarlo hasta su posición destino, dejarlo y realizar un movimiento de retroceso, quedando el plan finalizado y el robot en este posición.

Figura 43: Posición tras finalizar el plan

7.3 Ejecución del plan

La mayor parte de los problemas encontrados durante la gestión del plan se derivan de las comunicaciones. Sobre todo de la comunicación con el componente "pinzaComp", dicho componente tarda mucho en responder a una petición sobre la posición actual del elevador, introduciendo además un retraso en la posición proporcionada lo que, a su vez, provoca un retraso en la aplicación de las tareas del plan.

8 Capítulo 8: Conclusiones y trabajos futuros

Después de varios meses de trabajo se consiguió llegar a una solución satisfactoria que permite al robot alcanzar la posición del palé de una forma correcta y conseguir llevarlo a su posición de destino. Las pruebas realizadas demuestran que las técnicas desarrolladas hacen que el robot se pueda desenvolver bien en entornos libres de obstáculos, tarea a solventar en futuras ampliaciones. Además, debido a su diseño basado en componentes, sería fácil ampliarlo o acoplarlo a otros componentes existentes para que funcionen conjuntamente, extendiendo así las capacidades del robot.

Hay que destacar la gran cantidad de información que podemos obtener mediante el análisis de una simple imagen. La estimación de la posición mediante la proyección ha resultado ser un método bastante preciso. Esto unido a la aproximación ha resultado ser lo suficientemente robusto. El margen de error cometido es mínimo.

Las cámaras utilizadas durante el desarrollo del proyecto no han resultado lo buenas que se esperaba. Están muy influenciadas por la iluminación de la escena y el autoenfoque que llevan incorporado ha empeorado la situación. El tratamiento del color de las imágenes obtenidas se ve marcado por los cambios de color asignados por la cámara lo que no facilita en nada la tarea de identificar el palé. En posteriores usos de estas cámaras es necesario tenerlo en cuenta. Si bien la resolución de las imágenes es buena, por lo que pueden resultar útiles en situaciones en las que no se emplee el color.

En resumen, necesitamos una cierta calidad en las imágenes capturadas, sin embargo, los resultados obtenidos demuestran que es posible generar un método de aproximación robusto y fiable disponiendo de una información pobre. Básicamente la característica necesaria radica en la existencia de colores fácilmente distinguibles del fondo y del resto de objetos presentes en la escena.

8.1 Trabajos futuros

Algunas de las líneas de trabajo futuras para ampliar y mejorar el funcionamiento del componente serían:

- La primera de las actuaciones futuras debería centrarse en extraer del componente “paleComp” toda la información relativa a la generación de los colores y la definición de las balizas puesto que estas tareas no deberían pertenecer a dicho componente. Gracias a esto se libera de peso el componente y pasa a ser más reutilizable el código que genera el modelo de color HSL así como también la generación de balizas lo cual puede ser muy útil para el posterior desarrollo de componentes que utilicen imágenes en color.

- Especificar el plan mediante una maquina de estados de forma que la información relativa a cada fase de la ejecución del plan quede almacenada dentro de esos estados pudiendo eliminar gran cantidad de variables de clase que solo son utilizadas durante cada fase individual.

- Otra de las posibles tareas futuras consiste en la implementación de este modelo de plan mediante una máquina de estados. Esta transformación simplificará la gestión de los mismos aportando además un mayor control a las transiciones.

- Añadir algún método más de reconocimiento basado en la imagen de forma que el proceso de identificación sea aún más robusto.

- Establecer estados de emergencia ante posibles fallos de alguno de los proxis de comunicación que permitan recuperar el sistema ante un fallo.

- Conectar el sistema a un componente que proporcione información acerca de los obstáculos de forma que se pueda llevar a cabo la tarea esquivando los posibles obstáculos que surjan en el camino.

- Otro de los aspectos a mejorar, podría ser la segmentación de la imagen, introduciendo o combinando métodos de detección de bordes y líneas además de la actual búsqueda de zonas de color.

- Diseñar un método capaz de identificar fallos durante el movimiento de la pinza. Este método podría indicar, por ejemplo, aproximaciones incorrectas, cuando la pinza no haya entrado correctamente en el palé.

- El diseño del sistema se basa en la suposición de encontrarse el palé sobre el suelo. En posteriores versiones debería evaluarse esta limitación de forma que pueda detectarse un palé colocado sobre alguna superficie de forma que se simule un entorno de estanterías.

Parte III - Bibliografía

Bibliografia

[1] F. J. Maldonado, J. R. Vega, *Diseño y control del sistema de manipulación en un almacén automático.*

[2] Dr Thomas Sederberg, BYU Bézier curves, http://www.tsplines.com/resources/class_notes/Bezier_curves.pdf

[3] Pablo Barrera González, *Apliación de los métodos secuenciales de monte carlo al seguimiento visual 3D de múltiples objetos.* Tesis doctoral,2007

[4] Pilar Bachiller Burgos. *Percepción dinámica del entorno en un robot móvil.* Tesis doctoral, 2008.

[5] Curvas de Bézier http://es.wikipedia.org/wiki/Curva_de_Bézier

[6] Intel® Integrated Performance Primitives (Intel® IPP) http://software.intel.com/en-us/intel-ipp/

[7] Anton Girod Fortuño *Sistema de localización relativa basada en visión artificial*. Proyecto fin de carrera 2008.

[8] Michi Henning and Mark Spruiell. *Distributed Programming with Ice.*, revision 3.3.0. 2008.

[9] Pablo Bustos, RoboComp V1, 2008

[10] Luis Joyanes., *Fundamentos de programación. Algoritmos, estructuras de datos y objetos.* 3ª edición 2003

[11] Luis J. Manso. *Navegación visual en robots móviles*. Proyecto fin de carrera. 2009.

[12] Robolab. Robex arena - http://robexarena.com. 2009.

[13] Robolab. RoboComp project - http://robocomp.wiki.sourceforge.net. 2009.

[14] Agustín Sánchez Domínguez. *Diseño y construcción de un controlador de motores dc basado en microcontroladores*. Proyecto fin de carrera. 2007.

[15] Creative Commons España. http://es.creativecommons.org. 2009.

[16] Página del laboratorio de robotica de la Uex – http://robolab.unex.es 2009