Estructuras Secuenciales para Percepción Activa

Ramón Cintas Peña

Índice general

Índice de figuras

Resumen

El problema de la interacción con objetos simples en un entorno semicontrolados constituye un reto para la robótica móvil. En este proyecto se aborda un problema de este tipo mediante el diseño y construcción de un plan de acción secuencial para resolver una tarea de transporte de un palé, utilizando como única información sensorial la proporcionada por un sistema de visión estéreo y la odometría interna del robot.

El trabajo desarrollado se ha centrado fundamentalmente en el diseño de técnicas que permitan el reconocimiento visual del palé así como de todos los procesos necesarios para localizarlo, aproximarse, obtener su posición y finalmente cogerlo para transportarlo a una posición de destino. Todos estos procesos son ejecutados dentro de un plan secuencial que permite resolver la tarea global. Como resultado, el sistema final puede ser fácilmente modificado y adaptado a nuevas situaciones que puedan surgir, como, por ejemplo, el cambio del objeto a detectar.

La memoria del proyecto indica los problemas encarados y cómo han sido resueltos, además de varios resultados experimentales. También introduce RobEx y RoboComp, las plataformas hardware y software usadas.

Gracias al diseño basado en componentes de la implementación, que ha sido incorporada a RoboComp, es fácil de ampliar o acoplar con otros componentes.

Palabras clave

robótica autónoma, máquina de estados,comportamiento.

Abstract

For mobile robots, the problem of interaction with simple objects in a semi-controlled environment is a rich source of challenging situations. In this work we deal with this kind of problem through the design and construction of a sequential action plan to solve a task of transporting a pallet, using only the sensory information provided by a stereo vision system and the internal odometry of the robot.

The work has focused on the design of techniques for visual recognition of the pallet as well as all the processes required to locate, approach, get the position and finally pick it up for transporting it to a target position. All these processes are executed in a sequential plan that solves the overall task. As a result, the final system can be easily modified and adapted to new situations that may arise, as for example, the change of the object to be detected.

This work presents the problems encountered and how they have been solved, in addition to the results of the initial experiments. Also introduces Robex and RoboComp, the hardware and software platforms used.

It has been implemented using component-based design, and has been included in the RoboComp framework. Thus, it is easy to extend and connect with other components.

Capítulo 1

Introducción

Uno de los principales problemas con los que se enfrenta la robótica móvil en la actualidad consiste en la creación de estructuras que representen comportamientos complejos.

Muchos trabajos se centran en la creación de sistemas muy complejos que resuelven tareas de forma eficiente, sin embargo, el diseño de dichos sistemas es totalmente cerrado. La aplicación resultante solo puede utilizarse para la tarea que fue diseñado: brazos robóticos en cadenas de montaje, maquina empaquetadoras, robots de videovigilancia, etc .

Estos planteamientos chocan frontalmente con la idea de la robótica móvil, en la que los robots son equipados con una serie de sensores y "comportamientos" básicos y el problema fundamental radica en la combinación de muchas de estas pequeñas acciones para acometer tareas mayores. Esto incrementa notablemente la dificultad para llevar a cabo una tarea concreta, no obstante, aporta una gran reusabilidad y escalabilidad.[17]

Además, estos nuevos algoritmos han de coexistir con los comportamientos "típicos" de la robótica móvil: navegación local, reconocimiento de objetos, localización, generación de mapas, calibración, etc.

Todo esto limitado a su vez por las problemáticas asociadas a la ejecución en el mundo real: detección de ruidos recogidos por los diferentes sensores, limitación en el tiempo o número de accesos a los datos proporcionados por dichos sensores, la detección de problemas del hardware, el ingente número de operaciones y algoritmos

que han de convivir en equipos reducidos y sobre todo la limitación en el tiempo (las tareas han de llevarse a cabo en tiempo real).

Para abordar todos estos problemas dentro de un grupo de desarrolladores trabajando al mismo tiempo, con la necesidad de reutilizar algoritmos y comportamientos ya prediseñados y teniendo siempre presente la flexibilidad y escalabilidad de los comportamientos generados, es necesario el uso de middlewares orientados a componentes [6][2]. Estos middlewares proporcinan un medio para dividir, organizar y reutilizar la gran cantidad de complejos y cambiantes algoritmos típicos de los entornos de desarrollo de software para robótica.

La arquitectura que se va a utilizar, llamada RoboComp[10], pude definirse como un framework de componentes orientados a robótica, dicho framework será comentado en un capítulo posterior 2.

Utilizando esta infraestructura es más sencillo abordar el problema en cuestión, dejando de lado la conexión, control y seguimiento del resto de procesos que se llevan a cabo en paralelo.

1.1. Objetivos

Como ejemplo sencillo pero realista de una tarea compleja a realizar por el robot se ha seleccionado la manipulación de un palé.

Dicho comportamiento involucra una serie de subtareas complejas en si mismas: búsqueda, detección, identificación, estimación de la posición, maniobras de aproximación, coger el palé y llevarlo a su destino.

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 otros más pequeños 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.

- 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 el capitulo: 2) se ha de intentar generalizar los algoritmos de forma que puedan ser usados sobre otras plataformas similares.

- 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. Esta interacción será añadida en futuras versiones.

1.2. Estructura del documento

La estructura de este documento se divide en una serie de capítulos bien diferenciados. 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 2: Robex y RoboComp: Dentro de este capítulo se detalla la arquitectura sobre la que se construirá la solución al problema. Así como una breve descripción del robot con el que se realizarán las pruebas.

Capítulo 3: Planteamiento del problema: 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 4: Soporte matemático del sistema propuesto: En esta parte se explica el modelo matemático utilizado para la realización de todas las transformaciones de puntos entre unos sistemas de referencia y otros. Con este soporte matemático se puede resolver la cuestión ¿Donde esta el punto (x, y) de la pantalla en el mundo real? o ¿Como se ve un punto con coordenadas sobre el mundo visto desde la cámara izquierda?.

Capítulo 5: Descripción del sistema: Aquí se trata el sistema diseñado para identificar el palé dentro de una imagen y la especificación detallada de cada una de las fases del plan de acción.

Capítulo 6: Experimentos: Esta capítulo esta dedicado a las pruebas llevadas a cabo para validar la solución presentada.

Capítulo 7: Conclusiones y trabajos futuros: Dentro de este capítulo se hace una valoración global del trabajo realizado y se apuntan las líneas más importantes sobre las que se han de centrar los esfuerzos en el futuro.

Capítulo 2

Robex y RoboComp

El proyecto consistirá en la creación de los componentes necesarios para llevar a cabo los objetivos planteados anteriormente. Para ello se aprovecharán todas las funcionalidades ya cubiertas por los componentes existentes así como los conceptos y la metodología de desarrollo.

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 inalámbrica 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.

Por otra parte, la arquitectura que se va a utilizar, llamada RoboComp pude definirse como un framework de componentes orientados a robótica que aporta unas importantes características técnicas sin perder de vista la sencillez de uso.

Tanto RobEx como RoboComp son proyectos desarrollados en el Laboratorio de Robótica y visión artificial (RoboLab) de la Universidad de Extremadura. Son libres y se puede acceder a ellos mediante sus correspondientes páginas web [12],[11].

A lo largo de este capítulo se describirán de forma resumida las características

principales del robot así como los componentes que hacen uso de dichas características.

2.1. Robex

El robot RobEx [14][15], es una base robótica libre desarrollada en el Laboratorio de Robótica y Visión Artificial de la UEx, RoboLab [13].

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 del 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 "Creativa Commons AttributionShare Alike 3.0". Por tanto, su diseño está abierto a cualquiera que lo quiera consultar [11] o contribuir a él.

Tanto los motores de la base, de corriente continua, como el resto de la electrónica se alimentan de una batería del tipo que suele 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 [15].

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 cuatro 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

A la forma "básica" de RobEx se han incorporado una serie de componentes adicionales que aumentan sus capacidades, una cabeza estéreo con dos cámaras 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

2.1.1.1. Cabeza estéreo

La cabeza estéreo está formada por una estructura en forma de T con dos soportes en los extremos de la T donde se sitúan las cámaras. Es una estructura diseñada en aluminio, lo que le confiere una gran solidez. Cada cámara está acoplada a un soporte anclado a un motor que proporciona un giro independiente sobre el eje vertical. En el medio de la T se encuentra un tercer motor que hace girar toda la estructura, proporcionando un movimiento rotatorio simultáneo de ambas cámaras sobre el eje horizontal. En la parte inferior de la cabeza se encuentra el cuarto motor, que proporciona un giro a toda la estructura en el eje vertical, simulando el comportamiento del cuello humano.

Este diseño admite, por lo tanto, 4 grados de libertad sobre la cabeza robótica: uno de elevación (tilt) común a las dos cámaras, dos para el giro (pan) de cada cámara y uno para el giro de toda la cabeza y a su vez común a las dos cámaras (neck).

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.

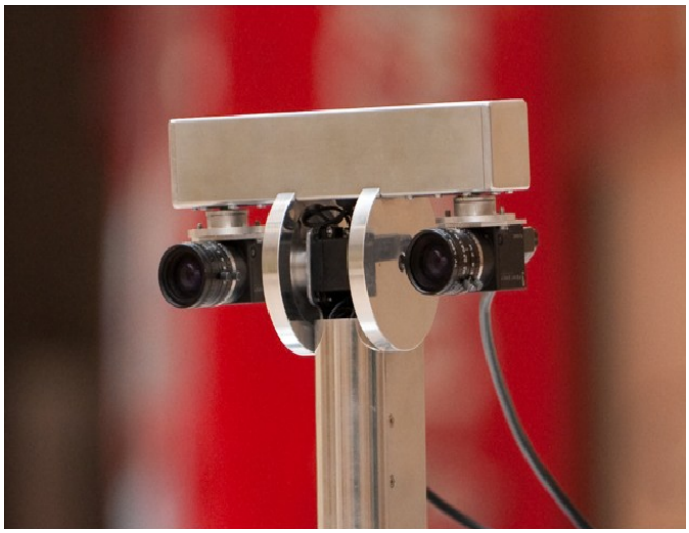

Figura 2.1: Cabeza estéreo

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 los componentes "jointmotor" y "headnt2p" diseñados específicamente para esta tarea.

2.1.1.2. Elevador de cargas

El elevador de cargas es un actuador de un grado de libertad diseñado para simular entornos industriales de transporte de palés. Todo el diseño se ha realizado usando CNC, lo que aporta una gran precisión y un acople perfecto al resto del robot. El desplazamiento vertical se consigue con un tornillo sin fin movido por un servo motor situado en la parte superior. Se trata de un funcionamiento muy simple a la vez que efectivo. El tornillo sin fin permite posicionar la carga en cualquier posición del recorrido sin que sea necesaria la aplicación de fuerza, por parte del servomotor, para mantenerla. Esto resulta una cualidad muy deseable si nos encontramos en entornos en los que la disponibilidad de electricidad es limitada, como es el caso de la robótica móvil.

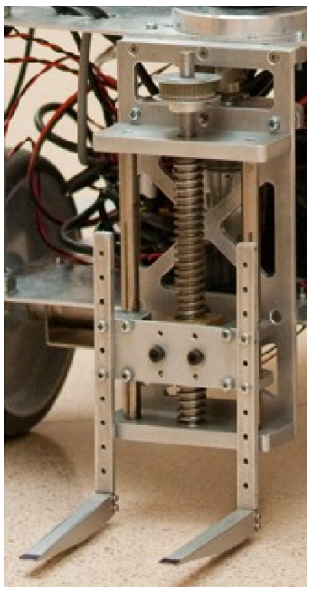

Figura 2.2: Elevador de cargas

Para aumentar la capacidad de carga del elevador, se dispone de una reductora acoplada al motor. El uso de esta reductora repercute en la velocidad de ascenso y descenso de la plataforma por lo que hay que llegar a un compromiso entre ambas.

Como resultado, se ha obtenido una velocidad de 20 mm/seg lo que es más que suficiente para el entorno en el que va a trabajar.

En cuanto al control software del elevador, se utiliza un microcontrolador ATmega32. Mediante este microcontrolador se gestionan las peticiones de movimiento del servo así como la asignación de velocidad. Además, contiene un contador LSI utilizado para leer la posición del enconder del microcontrolador. La comunicación con el dispositivo se basa en el protocolo serie y una interfaz Serie-USB.

2.2. RoboComp

Como se ha comentado en la introducción, la arquitectura que se va a utilizar, llamada RoboComp [9][10], pude definirse como un framework de componentes orientados a robótica. RoboComp utiliza Ice, un middleware liviano y de calidad industrial que aporta gran fiabilidad (ha sido usado en varios proyectos críticos [18]).

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, RoboComp posee unas características muy útiles como son:

- Modelo de componentes.
- Una estructura de directorios flexible.
- Scripts para la generación y modificación de los componentes
- Un gestor de componentes gráficos que permiten monitorizar su comportamiento de forma dinámica, “managerComp”.
- Capacidad de simulación mediante conexión con el simulador Gazebo.

- Capacidad para realizar logging.
- Herramientas de monitorización del funcionamiento de los componentes en tiempo real.
- Una herramienta capaz de los datos producidos por los sensores para su posterior reproducción, “replayComp”.

2.2.1. Componentes

RoboComp proporciona una amplia variedad de componentes. Existen, por ejemplo, componentes dedicados a interfaz hardware (p.e. cameraComp, differentialRobotComp o laserComp), a la implementación de comportamientos (p.e. gotopointComp o wanderComp) o al procesamiento de datos (p.e. visionComp y roimantComp para detección de características visuales, y cubafeaturesComp para la detección de características láser).Todos ellos se encuentran en continua revisión y ampliación. Hay que dirigirse a "http://robocomp.wiki.sourceforge.net"para obtener la última versión disponible. Además, existe una documentación que se genera semanalmente de forma automática.

Dentro de esta sección se va a tratar la estructura de componentes que se ha utilizado para solucionar el problema, de todos los componentes existentes en RoboComp solo se van a comentar los utilizados durante el desarrollo de este proyecto. Especificando sus principales características y funcionalidades.

Gracias al uso de la programación orientada a componentes se puede conseguir una visión global del sistema bastante fiel a la realidad sólo con ver el grafo de componentes. En la figura 2.3 se puede observar este grafo.

2.2.1.1. DifferentialRobot

Este componente proporciona una API para controlar una base robótica diferencial, almacenando además una odometría generada a partir de la posición de los encoders de cada una de sus ruedas. Mediante estas posiciones se va calculando el desplazamiento total del robot así como su velocidad lineal y angular.

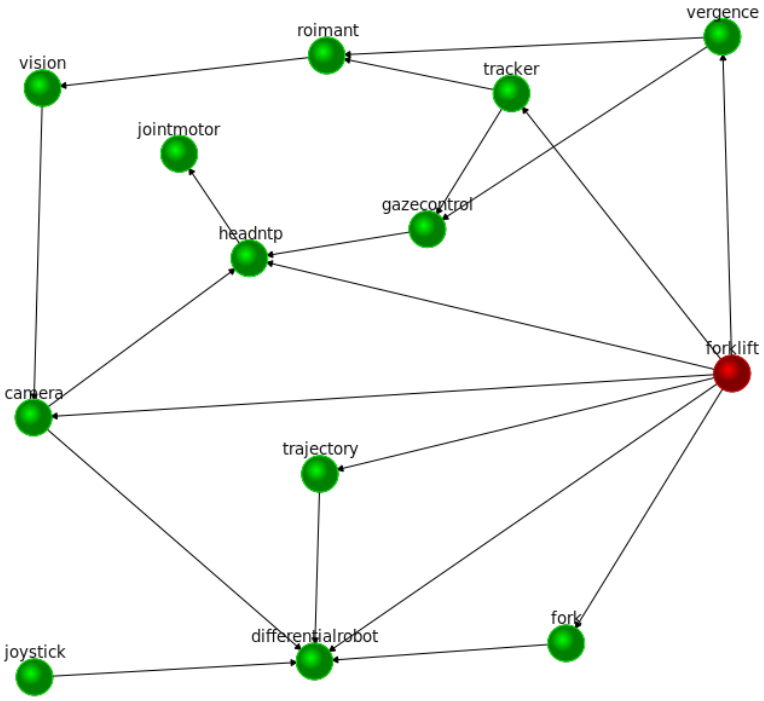

Figura 2.3: Componentes de RoboComp utilizados dentro del proyecto

Para el control de la plataforma Robex se utiliza un driver especificamente diseñado para comunicarse con el microcontrolador Atmel encargado de gestionar la potencia asignada a cada uno de los motores de las ruedas.

Sin embargo, el componente también tiene la capacidad de conectarse al simulador de código abierto Gazebo y a la capa de abstración de hardware Player. Esta es una característica muy importante, posibilita el desarrollo de software independientemente de la utilización del robot real Robex, de una simulación en gazebo o haciendo de soporte para cualquier robot diferencial que disponga de un driver para player.

Para el desarrollo de este proyecto no se utiliza directamente la gestión de la base robótica proporcionada por este componente sino que se accede a ella a través del componente "robotTrajectory" que será comentado más adelante.

2.2.1.2. Camera

Su principal función es capturar video y atender a las peticiones de imágenes que otros componentes realicen. Tiene capacidad para proporcionar imágenes de una cámara o de ambas 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.

Además, la cámara puede conectarse a "differentialRobot" y a "HeadNTP" para adjuntar a las imágenes, la información de posición de la base y de los servomotores de la cabeza respectivamente. Resulta muy importante disponer de las imágenes y de dichas informaciones al mismo tiempo, no solo hay que identificar que se esta viendo sino conocer también "donde" se esta viendo.

Otra de las características interesantes de cameraComp es que permite trabajar con distintos tipos de cámaras: v4l2 (Video For Linux v2), IEEE 1394 (FireWire), 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), Gazebo y los sdk's específicos de las cámaras Prosilica y Point Grey. Cualquier driver nuevo puede añadirse fácil y rapidamente, basta crear una nueva clase específica heredando de la clase abstracta "cámara" definida a tal efecto.

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 cameraComp 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. Si bien, es necesaria la existencia de ambas para un correcto funcionamiento del tracking y la vergencia.

2.2.1.3. JointMotor

El objetivo de este componente es proporcionar un acceso sencillo a un array de motores que comparten el mismo bus de comunicaciones. El componente es capaz de modificar los parámetros de configuración del bus así como los de cada uno de los motores, proporcionando una API de control para cada uno de ellos de forma independiente o el acceso a varios de forma síncrona.

Esta 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 facilmente con la implementación de una clase virtual que se creó para ello.

En función de la disposición de los motores en el robot se requerirá que función con una sincronización determinada. No hay que sincronizar el movimiento de los mismos motores si estos conforman una torreta estéreo que si forman parte de un brazo. Sin embargo, la implementación del componente permite la realización de diferentes configuraciones con solo variar los identificadores de los motores que recibirán dicha orden. Esta es una característica muy deseable para la configuración de motores incluida en la torreta estéreo, la realización de sacádicos se lleva a cabo en una sola orden y de forma asíncrona por todos los motores implicados en el movimiento.

2.2.1.4. HeadNT2P

Este es el componente que servirá de interfaz para el manejo de la cabeza, que ha sido añadida a la plataforma RobEx, utilizada para la realización del proyecto. Se trata de una cabeza con cuatro grados de libertad, dos para movimientos independientes sobre el eje vertical (sendos "pan" de ambas cámaras), uno para movimiento simultáneo sobre el eje horizontal ("tilt" de las dos cámaras a la vez) y otro para movimiento simultáneo sobre el eje vertical ("neck" para un mayor rango de visión de las cámaras).

Mediante la API de este componente se pueden realizar movimientos individuales sobre cada uno de los ejes, utilizar sacádicos o incluso movimientos que impliquen todos los motores a la vez.

Dentro del sistema se hace uso de los movimientos de la torreta a traves de este componente, sin embargo, la carga de trabajo más importante se lleva a cabo a través del "trackerComp". Al tracker se le ordena el seguimiento de un punto en pantalla y será él el encargado de gestionar los movimientos de la torreta de forma que se contrarresten los efectuados por la base. De esta forma conseguimos mantener el objetivo centrado en la cámara sin importar los movimientos que se van asignando a la base robótica.

2.2.1.5. Vision

Este componente no es utilizado directamente dentro del proyecto. Es necesaria su utilización puesto que forma parte de los cálculos necesarios para que funcione correctamente el tracker.

Su trabajo consiste en la obtención de regiones de interés, a partir de las imágenes proporcionadas por la cámara, mediante pirámides de Harris-Laplace. La imagen de entrada es reescalada y a cada una de las imágenes resultantes se le aplica el detector. Esto proporciona una serie de regiones de intéres en las diferentes etapas de la pirámide. Como salida puede devolver las regiones detectadas junto con toda la pirámide de imágenes generada.

Estas regiones son solicitadas por el componente "roimantComp", comentado a continuación.

2.2.1.6. Roimant

Al igual que ocurre con el componente anterior. Este tampoco es usado directamente por el sistema, también forma parte de los cálculos necesarios para el tracker. Roimant recibe la lista de regiones de interés proporcionadas por "vision". Su cometido es estabilizar dicha lista de regiones. Para ello mantiene una copia en memoria de las regiones visibles alrededor del robot, de esta forma es capaz de casar las regiones que van llegandole en tiempo real con las que tiene almacenadas en memoria. Como salida, roimant proporciona esta lista de regiones estabilizadas.

Si nos encontramos en un sistema con configuración estéreo. Roimant es capaz de calcular las coordenadas 3D de las regiones usando medidas de correlación y la

geometría epipolar proporcionada por HeadNT2P.

2.2.1.7. Tracker

Este componente hace el seguimiento, por parte de la cámara dominante (la izquierda), de una región de interés o de un punto del mundo.

En el caso del problema que se trata en este proyecto, es un componente muy útil, ya que nos permite mantener el objeto de interés dentro del campo de visión mientras nos acercamos a él.

Para conseguir el seguimiento de puntos o regiones de pantalla utiliza la información proporcionada por roimant. Mediante la información de las regiones y el movimiento de la base es capaz de calcular cuál de las regiones proporcionadas es la que debe seguir. Calcula las nuevas orientaciones para los servos y ordena al componente "gazecontrol" que mueva los servos de "pan", "tilt" y "neck" para que la cámara dominante centre en su campo de visión la región o punto del punto del que se está haciendo el seguimiento También es capaz de aplicar correlaciones sobre toda la pirámidde de Harris-Laplace para recuperar la información en caso de producirse un fallo.

Este componente se utiliza durante la fase de aproximación para maximizar la cantidad de imágenes en las que disponemos de una visión del objetivo.

2.2.1.8. Vergence

Este componente es el encargado de realizar, con las dos cámaras ancladas a la cabeza, un movimiento similar al que realizan los ojos humanos cuando miran hacia el mismo punto del mundo. Se encarga de controlar la posición de la cámara no dominante (en nuestro caso la derecha) de forma que su rayo principal converja con el de la cámara dominante repecto al punto 3D que esta está mirando. Para ello hace uso del componente roimantComp, que proporciona las regiones de interés de la imagen así como su posición en el mundo, y permite, mediante un "matching" o emparejamiento de las regiones obtenidas por las cámaras izquierda y derecha, obtener un punto común en ambas. Una vez establecido dicho punto común, se encarga de lanzar un movimiento de pan a la cámara no dominante para casarlo con

el punto visto en la dominante.

Como ocurre con "visionComp" o "roimantComp", este componente no es utilizado directamente por nuestra aplicación sino que es necesario para el correcto funcionamiento del tracker. El uso de la vergencia asegura que las regiones de interes en la cámara dominante coincidan con las vistas desde la cámara no dominante, es decir, que ambas cámaras están mirando al mismo sitio con lo que puede llevarse a cabo una correlación entre las regiones proporcionadas para cada una de las cámaras.

El correcto funcionamiento de la vergencia maximiza la cantidad de campo de visión superpuesto en una configuración de cámaras estéreo.

2.2.1.9. GazeControl

Los dos componentes de control de movimiento de cámaras, Tracker y Vergence, funcionan en paralelo y de manera asíncrona. Tras calcular la nueva posición de la cámara correspondiente, envían dicha posición al componentes GazeControl. Este último componente utiliza la información recibida para hacer un movimiento sincronizado de ambas cámaras que permita al robot dirigir su mirada hacía la región del entorno deseada.

GazeControl también se encarga de mejorar el control global de movimientos de cámara de diferentes formas. Por un lado, cuando recibe una nueva posición de la cámara dominante, solicita al componente Vergence el cálculo de la correspondiente posición de vergencia para evitar retardos en el movimientos de la cámara secundaria. A través de esta predicción, el componente GazeControl puede asegurar la fijación del mismo objetivo en las dos cámaras de manera simultánea. Además, también le permite obtener información sobre la nueva posición de la región de seguimiento en el espacio 3D. La información 3D obtenida es enviada al componente Tracker, quien la utiliza para actualizar el tamaño de la ventana de imagen que representa a la región de seguimiento.

Cabe señalar que si el componente Vergence no puede hacer una predicción sobre la posición de vergencia, el único efecto es un retraso en el movimiento de la cámara secundaria que se cancela tan pronto como el objetivo reaparece en el campo de

visión de las dos cámaras.

2.2.1.10. RobotTrajectory

Se encarga de generación y/o seguimiento de trayectorias locales usando la odometría proporcionada por la base.

El componente proporciona una API mediante la cual se pueden realizar diferentes trayectorias:

- Giros de α radianes desde la posición actual.
- Giros hasta situarse a α radianes en relación al mundo.
- Trayectorias en línea recta hasta un punto destino, con y sin ángulo final.
- Trayectorias generadas mediante un polinomio de Beziér.
- Una trayectoria combinando las características de las dos anteriores.
- También es capaz de seguir una trayectoria de puntos proporcionada desde fuera.

En todas trayectorias, el componente se encarga de gestionar la velocidad asignada a cada una de las ruedas [21]. Mediante este algoritmo se sigue la trayectoria establecida de la forma más fiel posible.

Gracias a este componente podemos asignar una trayectoria a la base robótica sin tener que ir supervisando su correcta ejecución.

2.2.1.11. Fork

ForkComp es un componente de la capa de abstracción de hardware. Como se ha comentado con anterioridad, el movimiento de la pinza lo realiza un servo al cual se accede a través de una placa basada en un microcontrolador.

Este componente se encarga de facilitar el acceso a dicho microcontrolador. Proporcionando una API muy sencilla para mover el elevador, asignarle velocidad o leer la posición en la que se encuentra. Además, han sido definidas dos funciones

adicionales encargadas de llevar el elevador a su posición mínima y máxima respectivamente. Estas dos funciones junto con la petición de la posición son las utilizadas dentro de este proyecto.

2.2.1.12. Joystick

Proporciona una API de acceso a un joystick genérico en Linux para controlar el movimiento de la base diferencial.

Este componente no lleva a cabo ninguna tarea dentro del plan de acción, pero ha sido muy utilizado para controlar la base durante la realización del mismo por lo que se ha considerado adecuada su inclusión aquí.

Capítulo 3

Planteamiento del problema

3.1. Adaptación del problema al entorno del laboratorio.

Como plataforma de experimentación se dispone de un robot de la clase Robex (comentado en el capítulo anterior). Como es lógico, los palés con los que se va a trabajar están diseñados a la escala del robot.

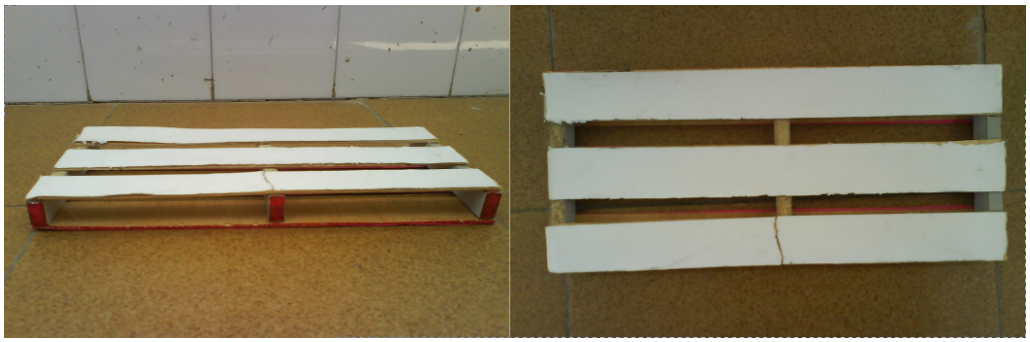

Figura 3.1: Vista frontal y superior del palé

Las dimensiones de estos palés son: 2,7 cm de alto, 30 cm de ancho y 15 cm de fondo, y están construidos mediante madera de balsa lo que los hace muy ligeros, para facilitar la tarea del elevador de cargas. En la simulación en el laboratorio no vamos a requerir levantar demasiado peso, no tiene sentido montar un sistema hidráulico para el elevador si para llevar a cabos simulaciones.

3.2. Creación del mundo sintético

Para la creación del mundo 3D que represente la zona en la que se encuentra el robot se ha utilizado OSG. Se trata de un mundo con un suelo plano de un tamaño mayor que la sala en la que se encuentra la plataforma robex utilizada y al que se le pueden añadir los objetos que se deseen.

Los principales objetos que se verán definidos en este mundo 3D serán el suelo, las paredes que se corresponden con las paredes del laboratorio, el modelo 3D del robot utilizado y el modelo 3D de palé creado.

El objetivo de este mundo sintético es el de realizar comparaciones de la imagen que se obtiene de la cámara real con las imágenes obtenidas de una cámara colocada en el robot añadido a OSG y que corresponden en posición y orientación, de tal forma que obtengan el mismo campo visual en ambos mundos.

3.2.1. Palé sintético

El palé real sobre el que se ha hecho el modelo es el que se muestra en la Figura 3.1.

El modelo creado para el mundo virtual y su resultado en la imagen de la cámara virtual se muestra en la Figura 3.2

Utilizando las transformaciones descritas en el capítulo anterior podemos proyectar el modelo de palé sobre la imagen de la cámara real situándolo previamente en la posición 3D adecuada. En la Figura 3.2 se puede observar el resultado de esta operación (imagen de la izquierda, superpuesto a la imagen del palé real).

3.3. Diseño del plan

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:

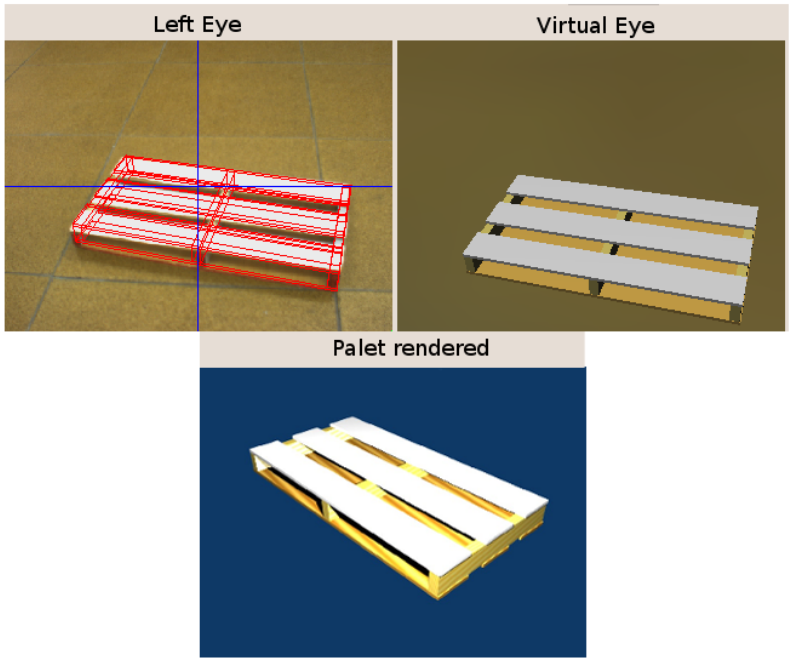

Figura 3.2: Proyección del palé sobre la imágen del palé real (izquierda), palé sintético visto desde la cámara sintética (derecha) y palé renderizado (abajo).

- 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 3.3.

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

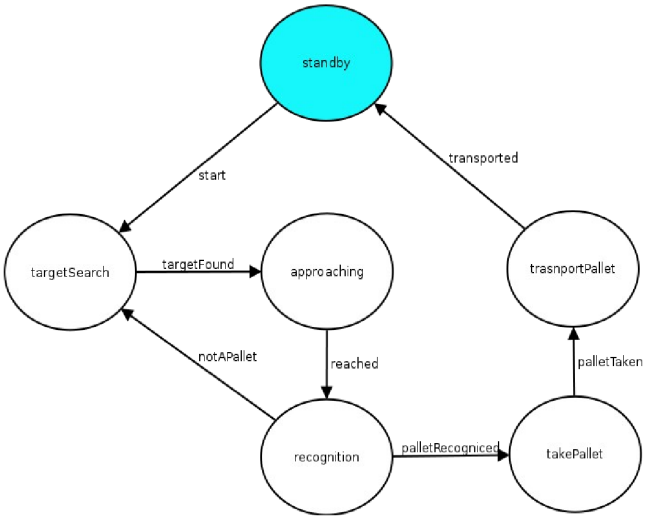

Figura 3.3: Especificación general del plan de acción

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.

Una parte importante del plan es la posibilidad de retorno establecida entre varias de las etapas:

- En el caso de la transición de retorno entre las etapas "approaching" y "targetSearch". En el entorno real en el que se mueve el robot no hay que 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, pueden crear falsos positivos.

- Del mismo modo, la posibilidad de retorno entre las etapas "recognition" y "targetSearch". Esta transición responde a la necesidad de volver a la fase de búsqueda si tras llegar a una posición desde la que se dispone de un buen punto de vista del objetivo se detecta que dicho objetivo no se corresponde con un palé.

A priori, un objeto de las mismas dimensiones que un palé, visto desde lejos

puede pasar como válido. Sin embargo, una vez estemos en disposición de efectuar un análisis más detallado podemos comprobar que en efecto no se trataba de un palé por lo que habrá que buscar un nuevo objeto candidato.

Cada uno de los estados representados (Figura:3.3) corresponde a una tarea principal a realizar para conseguir el objetivo:

Stanbdy: Es un estado de "descanso" durante el que no se realiza ninguna tarea importante para la consecución del objetivo. El robot se encuentra a la espera de una señal que le indique el comienzo del procedimiento.

TargetSearch: En este estado se realizará una búsqueda de posibles palés en el mundo. El robot se encargará de realizar los movimientos necesarios de la cabeza y de la base para mirar a distintos puntos del mundo y comprobar la existencia o no de algún objeto interesante en el suelo.

Approaching: Una vez se ha detectado un posible palé, nos disponemos a acercarnos a él. Es recomendable que se vayan haciendo medidas sistemáticas de la posición del objeto para acabar en una buena aproximación a él. También hay que tener en cuenta la aparición de falsos positivos, en muchos casos debidos a la variabilidad de la iluminación. Por esto hay que establecer una ruta de retono a la etapa anterior. En caso de visualizar un posible candidato y que este deje de serlo a medida que nos acercamos parece lógico volver a la fase de búsqueda en lugar de concluir la aproximación para posteriormente comprobar que efectivamente no se trataba de un palé.

Recognition: Una vez se ha finalizado la aproximación al objeto, llega el momento de comprobar si se trata en realidad de un palé y averiguar cual es su orientación y posición o, por el contrario, nos encontramos con cualquier otro tipo de objeto. Si se determina que el objeto es un palé se procederá a su localización en el mundo. En cualquier otro caso se volverá al estado de búsqueda de objetos en el suelo.

TakePallet: Una se ha reconocido la existencia de un palé, su orientación y su posición exacta, podemos comenzar con la tarea de recogerlo. En este estado

se calcularán los puntos de aproximación necesarios para encarar al palé en la posición correcta para que las pinzas del toro entren en él.

TransportPallet: Una vez se ha cogido el palé, se transportará al punto de destino definido. Para depositarlo allí y comenzar la búsqueda un nuevo candidato.

3.4. Máquina de estados

En el plan de acción que se ha definido, los comportamientos no solo dependen de la información que llega sino también de la fase en la que nos encontramos. Esto es una característica muy fácil de llevar a cabo gracias al uso de máquinas de estado.

En este caso vamos a utilizar el framework de máquinas de estados de Qt [5]. Este proporciona clases para la creación y ejecución de grafos de estados. Los conceptos y notaciones con los que se ha creado este framework estan basados en los "statecharts" propuestos por Harel [19].

Además de esta característica, el uso de máquinas de estado aporta otras muchas facilidades:

- Ejecución de estados de forma paralela
- Encapsulación de algoritmos dentro de métodos asociados a los estados.
- Transiciones sencillas entre los estados. Estas transiciones se activan mediante el uso de señales.
- Establece una jerarquía en las etapas, asegurando la ejecución en el orden correcto.
- Permite establecer puntos de retorno entre etapas de manera muy simple.
- Pueden generarse etapas que engloben a otras.

A esta implementación de Qt le falta una visualización en tiempo real de los estados que se encuentran activos y las transiciones que se llevan a efecto. Esta es una parte bastante importante, sobre todo a la hora de testear el correcto funcionamiento

y la detección de posibles errores. Para solventar esta incapacidad se ha creado una clase que envuelve a dicha máquina " QstateMachineWrapper".

Esta clase se encarga de generar, en tiempo de ejecución, una representación visual del funcionamiento de la máquina de estados. Representando en color amarillo los estados que se encuentran activos en ese momento 3.4.

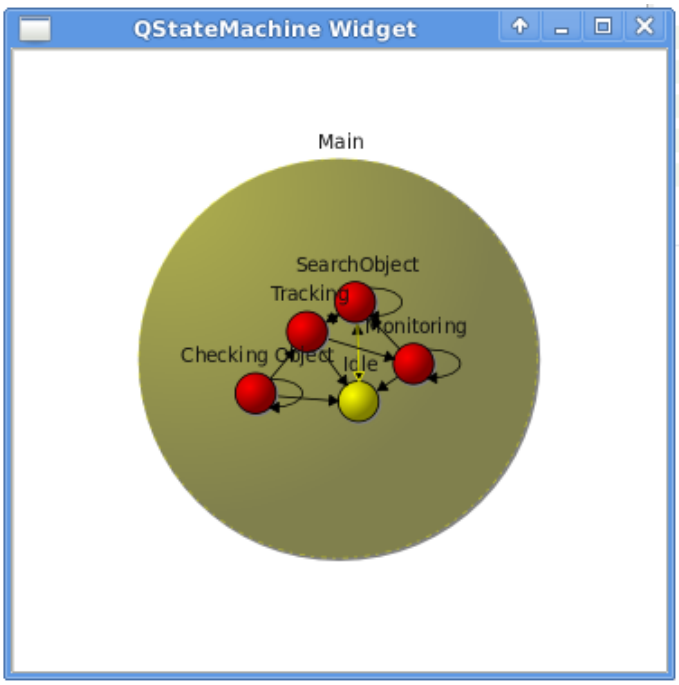

Figura 3.4: Visualización de la máquina de estados en tiempo de ejecución.

Además de esta representación en tiempo real, también se ha generado un parser ("QstateMachineParser"). Mediante este parser se ha creado una herramienta para la representación completa de la máquina de estados sin que esta tenga que estar activa.

Esta herramienta se ha diseñado para facilitar la visualización y entendimiento de una máquina de estados cualquiera, siempre que haya sido diseñada mediante la herramienta de Qt. Se pretende que en futuras versiones también permita la creación o modificación del código de forma visual.

En la figura 3.5 se observan las cuatro zonas principales:

1. En la zona central se representan los estados y las transiciones entre ellos. Los estados que contienen subestados pueden ser expandidos o colapsados haciendo

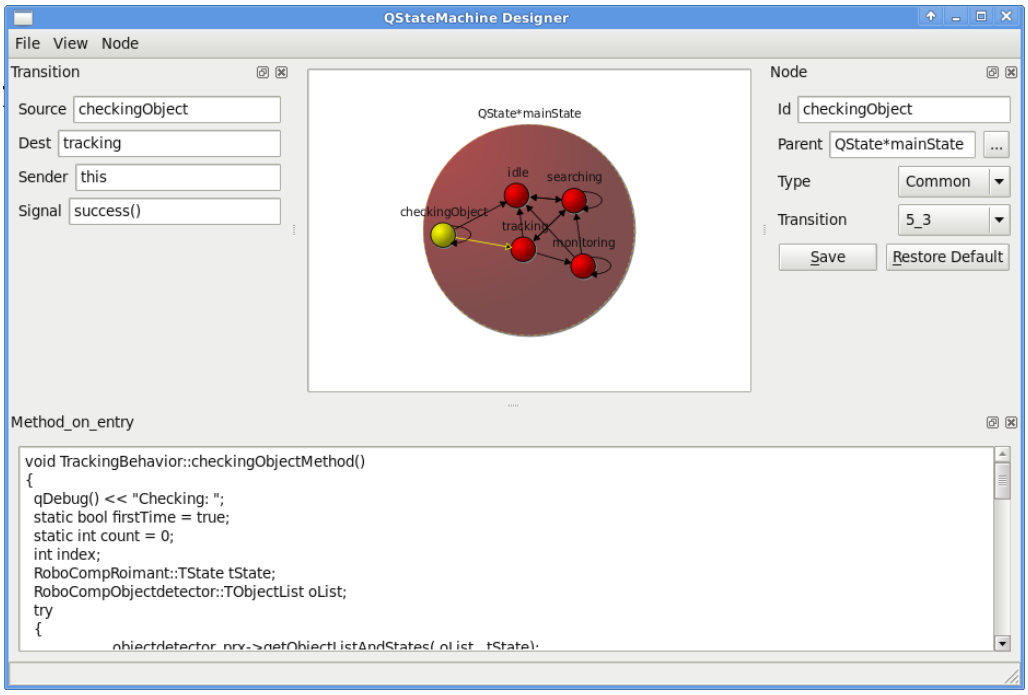

Figura 3.5: Herramienta de visualización de máquinas de estado.

click sobre ellos. Sobre esta zona central también puede llevarse a cabo "zoom" o "pan" para facilitar la visualización. Esto supone una gran ventaja para máquinas en las que aparecen un gran número de estados.

2. En la parte derecha se muestra toda la información relativa al estado que se tiene seleccionado. Los estados se seleccionan mediante el ratón desde la zona central. La información que se muestra para cada uno de los estados es: identificador, identificador del padre (en caso de tenerlo), el tipo y una lista que contiene todas las transiciones que parten o llegan a dicho estado.

3. En la parte izquierda se muestra la información relativa a las transiciones. Para seleccionar la transición que se desea consultar se debe recurrir a la lista que aparece en la visualización de la información de los nodos. Para cada una de las transiciones se muestra: el estado origen, el estado destino, quien emite la señal que activa la transición y de que señal se trata.

4. Por último, en la parte de abajo se muestra el contenido de la función que se activa cuando se entra en el estado. Para cambiar la información que se desea

visualizar basta con seleccionar un nuevo nodo en la ventana central.

Esta herramienta ha sido diseñada como un "MainWindow". Esto permite modificar la visualización al gusto del usuario. Cada una de las subventanas puede ser ocultada, cambiada de posición, puesta como tabulaciones o incluso pueden ser extraídas y aparecer como ventanas flotantes.

Capítulo 4

Soporte matemático del sistema propuesto

4.1. Sistemas de referencia de Robex

Robex está formado por distintas partes rígidas unidas entre sí por articulaciones móviles. Estas partes incluyen a la base del robot, el cuello, los motores que mueven las cámaras de la cabeza robótica y la pinza para manipular palés. Aunque la mayoría son articulaciones giratorias movidas por motores, también hay articulaciones planares como la que relaciona al robot con el suelo al desplazarse sobre él y articulaciones virtuales que se utilizan para representar objetos del entorno, como palés o cámara virtuales. Estas articulaciones virtuales pueden ser modificadas dinámicamente por el robot para representar cambios en su entorno. En la figura 4.1 se pueden ver todos los sistemas de referencia utilizados en Robex.

Toda este conjunto de articulaciones necesita ser modelado matemáticamente para obtener funciones que nos permitan transformar puntos de un sistema de referencia a otro. Así, nos podría interesar conocer las coordenadas de un punto del palé $(x, y, z)_{Pale}$ en el sistema de referencia de la cámara, o las coordenadas del robot $(0, 0, 0)_{Robot}$ en el mundo $(x, z, \alpha)_{Mundo}$. Las articulaciones del robot están organizadas formando un árbol con el $Mundo$ en la raíz como se puede ver en la figura 4.2.

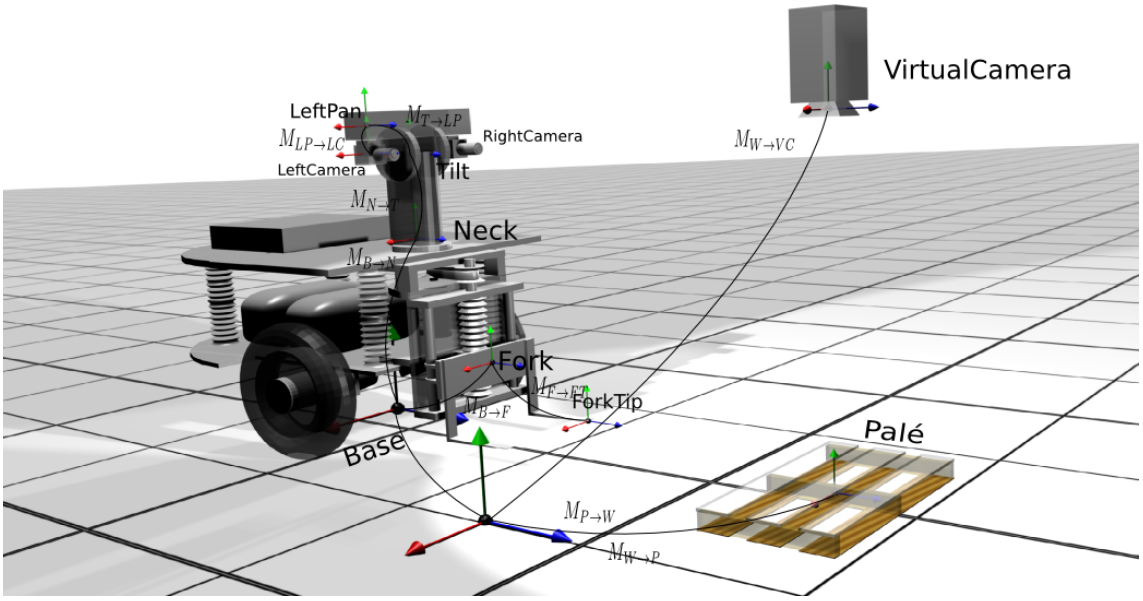

Figura 4.1: Sistemas de referencia de Robex

Para poder definir correctamente las funciones de transformación definimos la dirección $hijo \rightarrow padre$ como la transformación directa entre esos dos nodos.

Sea A un nodo cualquiera del árbol que tenga un padre B, definimos entonces la transformación directa entre ambos como:

$$M_{A \rightarrow B} = \{R, t\} \tag{4.1}$$

donde R codifica la rotación de A respecto de B y t la translación como un vector con origen en A y final en B. Expresado de otra forma, $M_{A \rightarrow B}$ permite responder a la pregunta ¿cómo vería un observador situado en B un punto definido en el sistema de referencia A? La transformación inversa, como veremos más adelante, permite responder a la pregunta ¿como vería un observador situado en A un punto en el sistema de referencia B? En la figura 4.2 se pueden ver algunas de las transformaciones directas con las flechas dirigidas en el sentido hijo $\rightarrow$ padre. También se pueden observar transformaciones inversas con las flechas apuntando en sentido padre $\rightarrow$ hijo

Antes de poder escribir completamente estas funciones necesitamos definir las

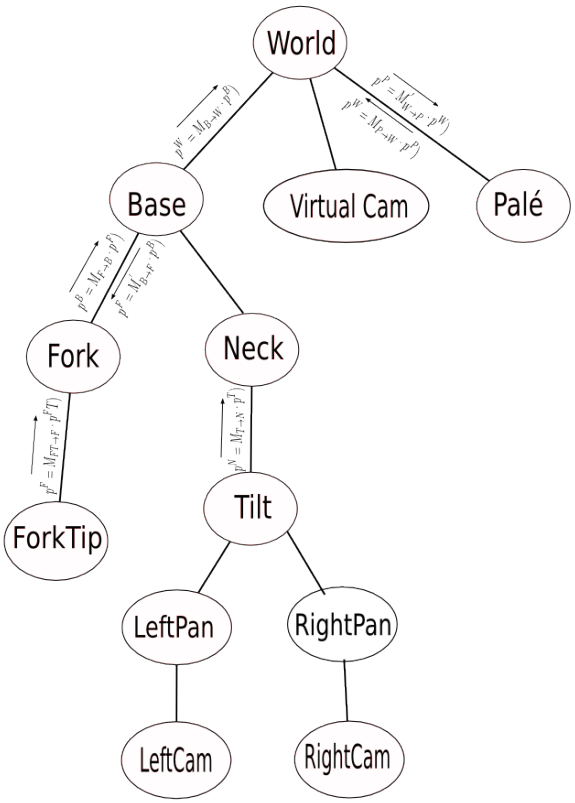

Figura 4.2: Estructura en árbol de los sistemas de referencia de Robex

siguientes convenciones geométricas:

- Los ejes se orientan en el robot con el eje Z positivo dirigido hacia delante, el eje Y positivo dirigido hacia arriba y el eje X positivo dirigido hacia la derecha del robot (visto desde arriba y mirando hacia delante). Ésta es la disposición que se observa en la figura 4.1 y que corresponde a la regla de la mano derecha.

- Si reducimos el mundo a una proyección sobre el suelo ($y = 0$), el robot se puede describir con una *pose* 2D compuesta por dos coordenadas (x, z) de posición y un ángulo de orientación α. La dirección positiva de este ángulo se muestra en la figura 4.3. La matriz de dos dimensiones que codifica un giro de α radianes en el plano es:

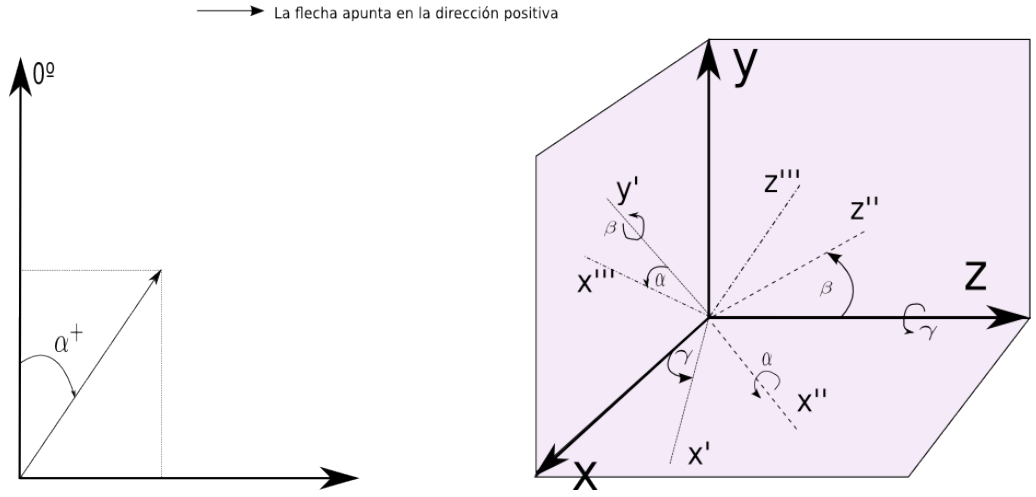

Figura 4.3: Dirección de ángulos

$$R = \begin{pmatrix} cos\alpha & sin\alpha \\ -sin\alpha & cos\alpha \end{pmatrix} \tag{4.2}$$

- Las *poses* en 3D se representan por una posición (x, y, z) y tres ángulos de rotación. Cada rotación respecto a un eje coordenado se representa con una matriz cuadrada de dimensión 3:

$$Rx = \begin{pmatrix} 0 & 0 & 1 \\ 0 & cos\gamma & sin\gamma \\ 0 & -sin\gamma & cos\gamma \end{pmatrix} \quad Ry = \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix} \quad Rz = \begin{pmatrix} cos\alpha & sin\alpha & 0 \\ -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

Seguimos la convención habitual de que el orden de las rotaciones es respecto a los ejes z, y y x respectivamente. Cada rotación se aplica siguiendo los ejes trasformados hasta el momento, no los originales. En la parte derecha de la figura 4.3 se puede ver el proceso. La matriz de rotación compuesta se obtiene multiplicando las tres matrices de rotación: $R = R_x * R_y * R_z$:

$$R = \begin{pmatrix} cos\alpha cos\beta & -sin\alpha cos\beta & -sin\beta \\ -cos\alpha sin\beta sin\gamma + sin\alpha cos\gamma & sin\alpha sin\beta sin\gamma + cos\alpha cos\gamma & -cos\beta sin\gamma \\ cos\alpha sin\beta cos\alpha + sin\alpha sin\gamma & -sin\alpha sin\beta cos\gamma + cos\alpha sin\gamma & cos\beta cos\gamma \end{pmatrix} \tag{4.4}$$

- Finalmente, la traslación entre los sistemas de referencia A y B se define como un vector con origen en A y destino en B

Podemos determinar ahora geométricamente la forma de $M_{A \to B}$ a partir de la siguiente figura:

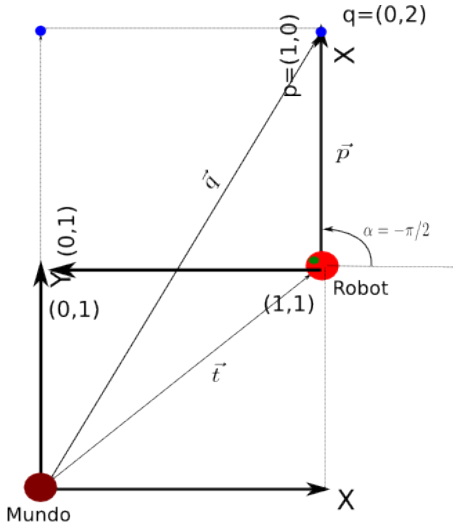

Figura 4.4: Transformación entre sistemas de referencia

Un punto p definido en el sistema de referencia del *Robot* con coordenadas $(1, 0)$ se ve en el sistema de referencia del *Mundo* como $\vec{q}$ con coordenadas $(2, 0)$. Geométricamente $\vec{q}$ es la suma de $\vec{t}$ y $\vec{p}$ definido en el sistema de referencia del mundo, esto es, en lugar de tener coordenadas $(1, 0)$ tendría coordenadas $(0, 1)$. La forma de pasar de uno a otro es rotando $\vec{p}$ el mismo ángulo que está rotado el sistema de referencia del *Robot* respecto al del *Mundo*

$$\vec{p}_{Mundo} = \vec{q} = R\vec{p} + \vec{t} = M_{Robot \to Mundo)} \cdot \vec{p} \tag{4.5}$$

De forma inversa, un punto $\vec{q}$ definido en el sistema de referencia del Mundo con coordenadas $(2, 0)$ se ve en el sistema de referencia del Robot como $\vec{p}$ con coordenadas $(1, 0)$:

$$\vec{p}_{Robot} = \vec{p} = R^T(\vec{q} - \vec{t}) = R^T\vec{q} - R^T\vec{t} = M'_{Mundo \to Robot} \cdot \vec{p}_{Mundo)} \tag{4.6}$$

4.2. Matrices Homogéneas

Las matrices homogéneas proporcionan una representación más compacta y apropiada para la programación. La matriz homogénea que transforma un punto homogéneo $\begin{bmatrix} p & 1 \end{bmatrix}^T$ desde el sistema de referencia A al B es:

$$H_{A \to B} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tag{4.7}$$

y la transformación inversa que lleva de B a A:

$$H_{B \to A} = \begin{pmatrix} R^T & -R^T t \\ 0 & 1 \end{pmatrix} = H^{-1}_{A \to B} \tag{4.8}$$

Para expresar secuencias de transformaciones entre dos sistemas de referencia que no estén directamente conectados basta con multiplicar estas matrices. La matriz resultado de este producto permite transformar puntos directamente entre ambos sistemas.

Por ejemplo, si en Robex queremos obtener la función $M_{RC \to W}$ que transforma puntos en el sistema de referencia de la cámara derecha RC al sistema de referencia del mundo W, basta con multiplicar las matrices correspondientes a cada articulación subiendo por la rama del árbol que lleva desde *RightCamera* a *World*.

$$p^W = [M_{RC \to T} \cdot M_{T \to N} \cdot M_{N \to B} \cdot M_{B \to W}] \cdot p^{RC} \tag{4.9}$$

De igual forma, para construir la función inversa que transforme puntos desde el mundo al sistema de referencia de la cámara derecha:

$$p^{RC} = [M'_{W\to B} \cdot M'_{B\to N} \cdot M'_{N\to T} \cdot M'_{T\to RC}] \cdot p^{W} \tag{4.10}$$

que es igual a:

$$p^{RC} = [M_{RC\to T} \cdot M_{T\to N} \cdot M_{N\to B} \cdot M_{B\to W}]^{-1} \cdot p^{W} \tag{4.11}$$

ya que:

$$[A \cdot B]^{-1} = B^{-1} \cdot A^{-1} \tag{4.12}$$

Para tener un sistema de transformaciones todavía más completo y versátil es conveniente incluir los elementos sensoriales en el árbol cinemático

4.3. Modelo de cámara

Introducimos ahora el modelo matemático de las cámaras utilizadas en Robex. El modelo de uso más extendido es el conocido como *pinhole*. En la figura 4.5 se puede ver la geometría de este modelo para una dimensión.

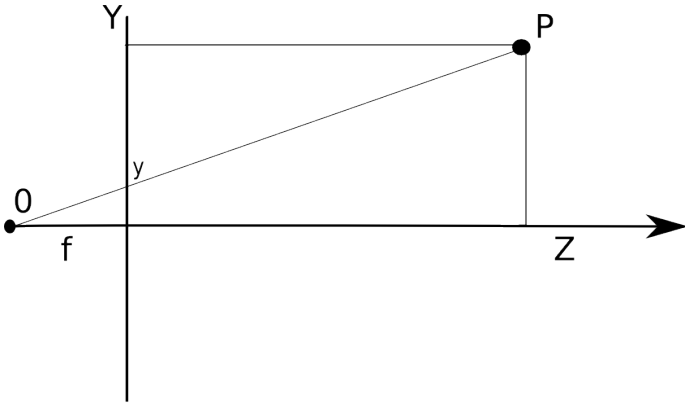

Figura 4.5: Modelo de cámara *pinhole*

El centro óptico está en O y el plano (línea) de imagen está en $(f, 0)$ a una distancia focal f. El punto P se proyecto siguiendo un rayo óptico que le una a O intersectando con el plano de imagen en p. Si escribimos las relaciones entre triángulos semejantes de la figura:

$$\Delta OZP \simeq \Delta Ofy \tag{4.13}$$

llegamos a la siguiente relación:

$$\frac{Y}{y} = \frac{Z}{f} \tag{4.14}$$

de donde obtenemos una expresión no lineal para la proyección y en la imagen del punto P:

$$y = f\frac{Y}{Z} \tag{4.15}$$

Para el caso de un plano de imagen bidimensional, podemos escribir la ecuación de proyección en forma matricial como:

$$\begin{bmatrix} x \\ y \\ \lambda \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4.16}$$

con

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} \frac{x}{\lambda} \\ \frac{y}{\lambda} \end{bmatrix} \tag{4.17}$$

Las coordenadas de imagen (i, j) dependen no-linealmente de las coordenadas del punto P del mundo, como expresaba la ecuación 4.15. El modelo habitual de cámara incluye algunos elementos más como las coordenadas del centro de la imagen, el ratio entre ancho y alto de los píxeles o un modelo de distorción radial y tangencial. Las coordenadas del centro de la imagen o *punto principal* cx, cy se incluyen en la matriz para convertir coordenadas de imagen, normalmente con origen en la esquina superior izquierda y dirección positiva de la y hacia abajo, a coordenadas de cámara, con origen en el centro de la imagen y dirección positiva de la y hacia arriba:

$$x = (i - o_x) \tag{4.18}$$

$$y = -(j - o_y) \tag{4.19}$$

La matriz resultante se denomina matriz de *parámetros intrínsecos* o K y tiene dimensiones 3x3.

$$K = \begin{bmatrix} f_x & 0 & -o_x \\ 0 & -f_y & -o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.20}$$

Ponemos el signo negativo de la conversión de la y en la focal f_y.

Finalmente, para poder proyectar puntos situados en el sistema de referencia del *Mundo* W sobre la cámara necesitamos transformar esos puntos al sistema de referencia de la cámara. Para ello seleccionamos la transformación apropiada y multiplicamos la matriz homogénea correspondiente por la derecha de K. A la matriz resultante se le denomina *matriz de proyección*. Si observamos el árbol de la figura 4.2 podemos ver que para llegar desde el sistema de referencia del mundo al de la cámara es necesario bajar por el árbol encadenando transformaciones inversas. Por lo tanto, la transformación final tendrá la forma $M'_{W \to RC}$ y llamaremos a la matriz homogénea correspondiente, sin la última fila, matriz de parámetros extrínsecos o E y tiene dimensiones 3x4:

$$E = \begin{bmatrix} R^T & -R^T t \end{bmatrix} \tag{4.21}$$

Al resultado de multiplicar la *matriz de parámetros intrínsecos* por la derecha de la *matriz esencial* se denomina *matriz de proyección* o *matriz de cámara* y tiene dimensiones 3x4.

$$P = \begin{bmatrix} f_x & 0 & -o_x \\ 0 & -f_y & -o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R^T & -R^T t \end{bmatrix} \tag{4.22}$$

Para proyectar un punto del mundo P en la cámara, primero se pasa a coordenadas homogéneas y luego se multiplica como vector columna por la derecha:

$$\begin{bmatrix} p_i \\ p_j \\ \lambda \end{bmatrix} = P \cdot \begin{bmatrix} P_X \\ P_Y \\ P_Z \\ 1 \end{bmatrix} \tag{4.23}$$

4.4. Cámara virtual y proyección inversa de puntos de la imagen

En la figura 4.1 aparece una cámara virtual situada por encima y hacia delante del robot. La razón de situar este elemento ahí es poder calcular de forma muy eficiente las coordenadas 3D en el suelo de puntos de la imagen, suponiendo que lo que está viendo la cámara es el suelo. Esta operación es muy útil para estimar la posición en el mundo de objetos que suponemos se encuentran en el suelo, o para comprobar si efectivamente están sobre el suelo.

Para ello utilizamos la *proyección inversa* que es la línea en el espacio 3D que pasa por el centro de imagen y por el píxel (i, j). La intersección de la proyección inversa de un píxel (i, j) con el plano del suelo es la coordenada buscada. La ecuación en coordenadas paramétricas de la proyección inversa de un píxel (i, j) se obtiene multiplicando la coordenada de imagen por la inversa de la matriz de parámetros intrínscos:

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \lambda K^{-1} \tag{4.24}$$

donde λ es el parámetro. Los puntos situados delante del plano de imagen se corresponden con $\lambda \geq 1$.

Para calcular la intersección de esta recta con el plano del suelo tenemos que resolver un sistema de ecuaciones. Si queremos hacerlo para todos los puntos de la imagen será necesario resolver tantos sistemas de ecuaciones como píxeles tenga la imagen.

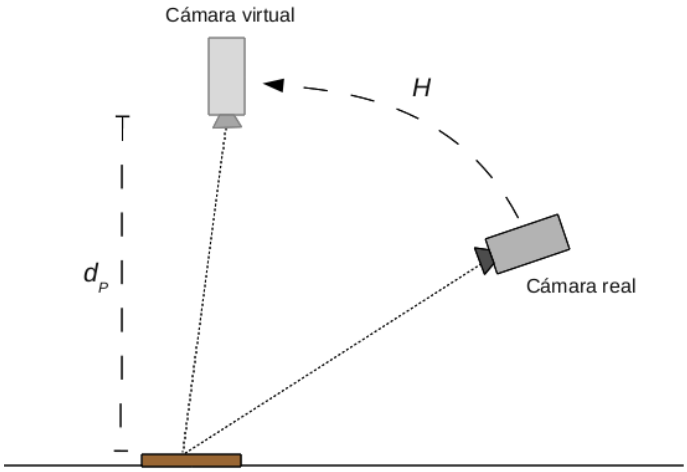

Figura 4.6: Relación entre cámaras y palé en el método de proyección inversa desarrollado

Para evitar esta costosa operación, se ha desarrollado un método alternativo basado en homografías aplicando una matriz de transformación común a todos los puntos de la zona de imagen que se quieren transformar. Para obtener esta transformación se utiliza una cámara virtual con el plano de imagen paralelo al plano del suelo y situada a una cierta distancia d_P de éste (ver figura 4.6). Cada píxel de esta cámara virtual se caracteriza porque su proyección p_v (ecuación 4.15) se corta con el plano del suelo a una distancia perpendicular d_P. Por tanto, dado un píxel p_v y suponiendo que dicho píxel es una proyección en imagen de un punto del suelo P, es posible obtener las coordenadas 3D del punto mediante la expresión:

$$P = d_P \cdot p_v/\lambda \tag{4.25}$$

Dado que el plano del suelo es conocido, p está relacionado con su píxel homólogo de la imagen real (p_r) a través de una homografía H:

$$p_v = H * p_r \tag{4.26}$$

La matriz H define la transformación de puntos entre dos vistas suponiendo que eso puntos son proyecciones de un plano $\pi = (n^T, d)^T$. Dado dicho plano (definido

en el sistema de referencia de la cámara real), las matrices de intrínsecos de las dos cámaras (K_v y K_r) y la rotación R y traslación t entre ambas cámaras, la matriz H presenta la estructura:

$$H = K_v(R - tn^T/d)K_r^{-1} \tag{4.27}$$

Utilizando una expresión modificada de la homografía:

$$H' = K_v^{-1}H = (R - tn^T/d)K_r^{-1} \tag{4.28}$$

se obtiene directamente la proyección del punto de la cámara virtual:

$$r_v = H'p_r \tag{4.29}$$

necesaria para obtener el punto 3D según la expresión 4.25.

Además, expresando el punto 3D en coordenadas homogéneas P'_v, a partir de la ecuación 4.25 se obtiene la expresión:

$$P'_v = \begin{vmatrix} d_P r_v & r_{vZ} \end{vmatrix}^T \tag{4.30}$$

Sustituyendo r_V en la ecuación anterior por su equivalente de la expresión 4.29 se llega a la siguiente relación:

$$P'_v = M_H * p_r \tag{4.31}$$

siendo M_H una matriz de transformación común a todos los píxels de la cámara real:

$$M_H = \begin{vmatrix} d_P H' \\ H'[2, 0:2] \end{vmatrix} \tag{4.32}$$

4.5. InnerModel y el árbol cinemático

Para que todos estos modelos matemáticos puedan ser utilizados eficientemente dentro de RoboComp, se ha creado una clase en C++ llamada InnerModel que construye, actualiza y da acceso al árbol cinemático de Robex.

Esta clase está basada en una descripción XML de la cinemática del robot que se encuentra almacenada en un fichero (ver listado 4.1). En este fichero, se identifican y describen cada uno de los nodos de transformación que forman parte del robot y de su entorno. Entre otros atributos, cada nodo incluye una etiqueta identificativa que permite referenciarlo en cada operación. InnerModel incluye una amplia variedad de métodos que proporcionan el soporte matemático necesario en el desarrollo de un sistema como el propuesto.

Listing 4.1: Ejemplo de descripción XML de *InnerModel*

```
<innerModel>
 <translation id="world">
  <transform id="base">
   <plane id="floor" ny="1" />
   <translation id="central" y="200">
    <rotation id="pan">
     <translation id="upperMotor" y="80">
      <camera id="upperCamera" focal="400" />
     </translation>
     <translation id="lowerMotor" y="-80">
      <camera id="lowerCamera" focal="400" />
     </translation>
    </rotation>
   </translation>
  </transform>
 </translation>
</innerModel>
```

Capítulo 5

Descripción del sistema

5.1. Segmentador de imágenes

A partir del componente "cameraComp" se obtiene una imagen RGB empaquetada. Es necesario un proceso de análisis de dicha imagen para discernir a que corresponde lo que se esta viendo.

Para este primer análisis se utiliza un segmentador por color. Este segmentador por color [20] analiza cada uno de los píxeles de pantalla y los va agrupando en zonas de imagen que tienen un color similar.

Como salida del segmentador se obtiene un vector en el que se asocia cada uno de los píxeles de la imagen de entrada con un identificador. Cada uno de estos identificadores constituye una de las regiones clasificadas por el segmentador. En la figura 5.1 puede verse una imágen de entrada (izquierda) y la clasificación realizada por el segmentador (derecha).

Hay que analizar ese vector de salida para obtener una información más completa. En primer lugar se realiza un barrido sobre todos los píxeles de la imagen para obtener una lista de regiones. La información que se almacena para cada una de las regiones sería:

- Centro de la región (x,y).
- Listado de todos los puntos asociados a dicha región.
- Color promedio en r, en g y en b.

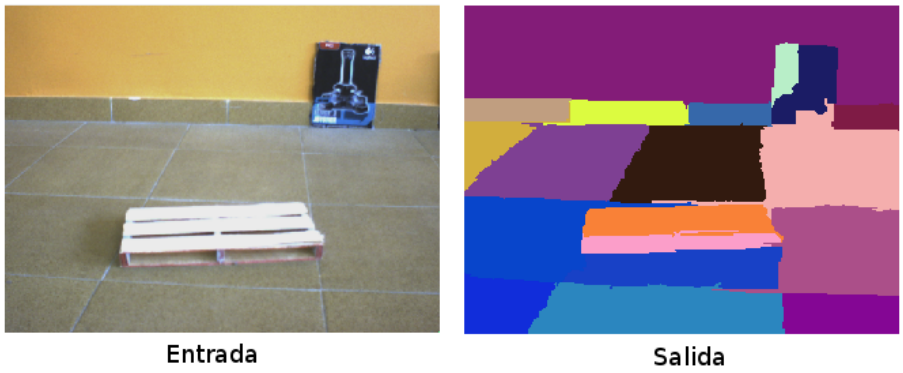

Figura 5.1: Imagen de entrada al segmentador (izquierda) y salida proporcionada por este (derecha).

- Número total de puntas asociados a dicha región.

Mediante esta lista de regiones tenemos una segmentación inicial de la imagen de entrada. La lista de regiones obtenidas queda representada en la imagen central de la fila superior de la figura 5.2.

El clasificador es bastante sensible a los cambios de tono de color, esto provoca la aparición de varias subregiones cuando la realidad es que constituyen una sola. Estos fallos suelen aparecer asociados a sombras.

Para suavizar este efecto se lleva a cabo un procesamiento adicional sobre ellas. Codificamos los posibles tonos de color de la cada una de las regiones a una escala de grises. Para ello aplicamos la siguiente transformación:

$$color = 100 * (R + G + B)/755 \tag{5.1}$$

Con esto reescalamos todos los posibles tonos de color en RGB a tan solo 100 tonos en escala de grises.

Construimos una nueva imagen a partir de las regiones detectadas por el segmentador. A cada uno de los píxeles de la región se le asigna uno de los tonos de gris generados en el paso anterior. De esta forma se obtiene una imágen en blanco y negro en la que algunas de las regiones diferenciadas por el segmentador presentan el mismo tono de gris.

Aplicamos un proceso de FloodFill al que se le pasa el punto central de cada

región. Este proceso nos devulve un área que contiene todos los píxeles asociados a dicha región así como el computo global de píxeles. Estas áreas son almacenadas junto con la información de las regiones. Esta información será utiliza para agrupar algunas de las regiones de forma que se consigan regiones de mayor tamaño.

Una vez completada la información para cada una de las regiones, hay que ir analizándolas para determinar si corresponden a zonas del suelo, pueden ser eliminadas por resultar poco útiles, o son posibles candidatas a ser un palé.

Tras quedar determinados los posibles candidatos a ser un palé pasaremos a una fase de clasificación de los mismos.

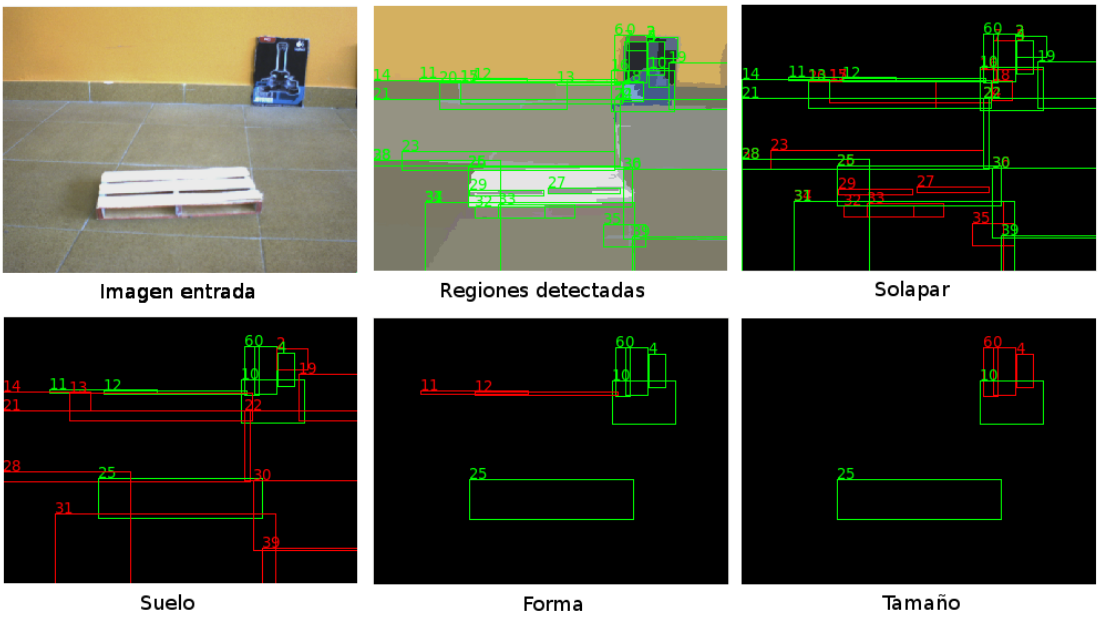

Figura 5.2: Procesamiento de las regiones desde la captura de la imágen incial a las regiones obtenidas como resultado

5.1.1. Obtención de los posibles candidatos

Esta clasificación de las regiones conlleva una serie de pasos. En cada uno de ellos se realiza un análisis atendiendo a un parámetro diferente: posición, forma, color y tamaño. El análisis consta de cuatro fases:

- Agrupar regiones pequeñas

- Eliminar las regiones correspondientes al suelo
- Eliminar regiones en funcion de su forma
- Eliminar regiones en base a su tamaño

Cada una de estas fases será analizada a continuación.

5.1.1.1. Agrupar regiones

Algunas de las regiones detectadas por el segmentador puede formar parte del mismo objeto o pueden haber sido detectadas debido a la aparición de alguna sombra. Por esto, la primera parte del análisis consiste en elaborar regiones más grandes uniendo las regiones cuya envolvente se solape o este contenida dentro de otra.

En el caso de la detección del palé, cuando este no se encuentra muy distante, aparecen claramente diferenciadas las tres bandas que constituyen la parte superior. Para mejorar la detección del mismo es necesario proceder a su agrupación dentro de una única región.

Hay que añadir dos condiciones a este proceso:

- Solo se agrupan regiones si la diferencia entre sus colores promedio no es demasiado alta.
- Solo se consideran superpuestas las regiones que lo hacen en al menos un 75 % en la mayor y un 50 % en la menor. Esto asegura no agrupar regiones muy grandes con otras muy pequeñas.

Gracias a este proceso se obtienen regiones más grandes y estables. En la figura 5.2 (imagen superior derecha) se observan en rojo las regiones que han sido absorbidas por otras y en verde las regiones resultantes.

5.1.1.2. Análisis de color

Con este análisis se pretende separar las regiones que forman parte del suelo del resto de regiones.

Para ello, se reordena la lista de regiones en función del valor de su coordenada Y en orden descendente, es decir, primero se analizarán las regiones que aparecen en la parte de abajo de la pantalla (las que se encuentran más cerca del robot).

Para cada una de ellas se compara su color promedio con el color promedio obtenido del suelo (inialmente mediante el estado "captureFloorTexture" 5.2.2). Si la distancia entre el color promedio de la región y el del suelo esta por debajo de un umbral prefijado, se determinada que dicha región forma parte del suelo. Esta región es eliminada de la lista de candidatos y se actualiza el valor promedio del suelo.

Para la actualización del color promedio del suelo se tiene en cuenta la posición que ocupa dicha región en la imagen. Esto es, se le da un mayor peso cuanto mayor es su coordenada Y. Si la región aparece en la parte baja de la imagen quiere decir que esta más próxima al robot por lo que resulta más probable que se trate de un parche de suelo. Además, el color recogido para una zona más cercana será más fiel que el recogido para una zona que se encuentre más alejada. La formalización de esta operación resulta:

$$factor = (parche.valorY/window.height) * 0{,}5 \tag{5.2}$$

$$floor.r = parche.r * factor + floor.r * (1 - factor) \tag{5.3}$$

La operación se repite para las tres componentes del color: r, g y b.

Mediante este análisis conseguimos eliminar el suelo de la imagen de entrada. La imagen inferior izquierda de la figura 5.2 ejemplifica este proceso. En rojo se dibujan las regiones detectadas como suelo y en verde las restantes.

Además, se va adecuando el valor de suelo almacenado de forma que el robot puede ir adaptándose a pequeños cambios en la tonalidad del mismo. En muchos casos estos cambios se deben a variaciones en la intensidad de luz o a la aparición de sombras, quizás debidas a la presencia del robot o a la posición en la que se encuentran "mirando" las cámaras.

5.1.1.3. Análisis de forma

Esta parte del análisis consiste en obtener aproximadamente la disposición de los píxeles que conforman dicha región. Nos interesa eliminar regiones con formas poco homogéneas, por ejemplo, las líneas de las uniones de las baldosas del suelo. Para ello obtenemos la matríz de covarianza:

$$M = \begin{pmatrix} \frac{\sum(\bar{x}-x_i)^2}{N} & \frac{\sum(y_i-\bar{x})(x_i-\bar{y})}{N} \\ \frac{\sum(y_i-\bar{x})(x_i-\bar{y})}{N} & \frac{\sum(\bar{y}-y_i)^2}{N} \end{pmatrix} \tag{5.4}$$

Mediante esta matriz de covarianza obtenemos los autovalores:

$$(M - \lambda I)x = 0; \tag{5.5}$$

La relación entre ambos valores de λ nos da una idea de la forma que tiene dicha región. En caso de que la diferencia entre ambos sea muy alta eliminamos la región de la lista de regiones candidatas. En la imagen central de la fila inferior de la figura 5.2 se observa este proceso.

5.1.1.4. Análisis de tamaño

De la lista de regiones candidatas eliminamos las que tengan un tamaño demasiado grande o demasiado pequeño para ser un palé. Se tiene el cuenta la posición relativa a la que se encuentran a la hora de evaluar dicho tamaño.

Mediante el innerModel (comentado en 4.5), se puede calcular la posición en el mundo que ocupa la mancha vista desde la cámara izquierda suponiendo que esta se encuentre en el suelo.

De esta forma tenemos una aproximación al tamaño que tiene el objeto en el suelo. En este caso hay que ser bastante permisivo, ya que la estimación de la superfície del objeto depende de lo bien que seamos capaces de detectarlo y de la aproximación realizada. Esta fase del análisis se corresponde con la imagen inferior derecha de la figura 5.2.

5.1.2. Clasificación de los candidatos

Cuando ha finalizado el análisis anterior, disponemos de una lista de regiones que cumplen las condiciones para ser un palé (Figura 5.3).

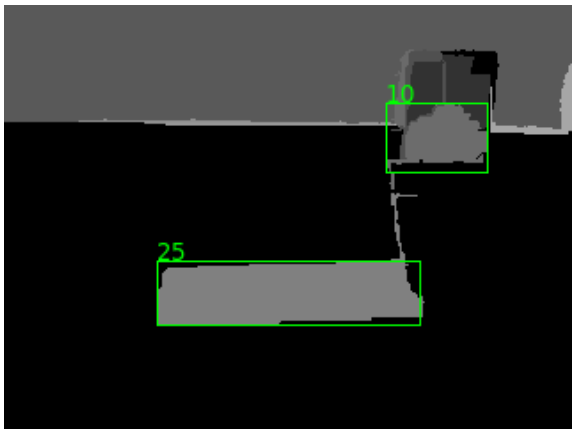

Figura 5.3: Regiones tras efectuar el análisis

De todas ellas hay que escoger la mejor candidata. Esta elección se hace en base a una serie de parámetros:

- Trasladando las esquinas del área que contiene la región podemos aproximar el tamaño del objeto en el mundo real independientemente de la distancia a la que nos encontremos y la consecuente proyección vista desde la cámara. El área que contiene a la región puede no coincidir con el contorno de la región (FloodFill devuelve el área que contiene todos los puntos clasificados, no el perímetro que los contiene). Por esto, calculamos la densidad de puntos contenidos en dicha área, esto nos da una idea de lo real que es la aproximación del área al perímetro real.

- La diferencia entre el color promediado de la región con el color promedio del palé que tenemos almacenado en memoria.

- Autovalores obtenidos durante la búsqueda de la región. El palé es el doble de ancho que de alto por lo que la relación entre los autovalores ha de ser más o menos 2:1

5.2. Implementación de la máquina de estados

En esta sección se explica en detalle el funcionamiento de cada uno de los estados de la máquina de estados.

En primer lugar hay que comentar la existencia de dos estados paralelos. Por una parte aparecería el plan de acción y por otra el estado encargado de asegurar el correcto funcionamiento de todo el sistema. Estos estados han sido denominados: "uncommonWork" y "commonWork" respectivamente, Figura 5.4.

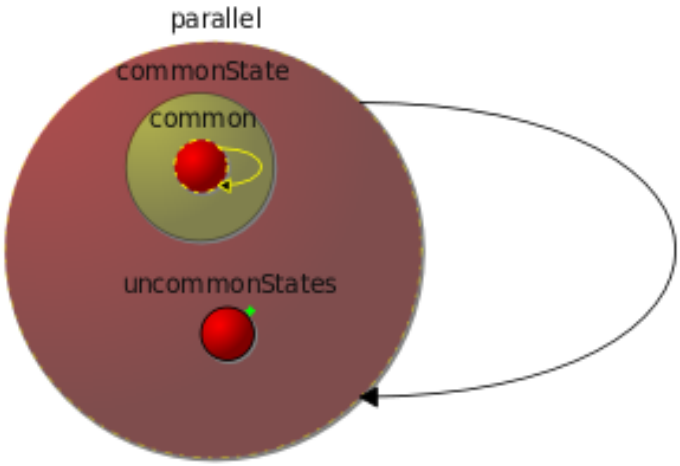

Figura 5.4: Estados paralelos de la máquina de estados.

Dentro del estados "commonWork" se realizan tareas comunes a todas las etapas del plan:

- Atención a la petición de eventos desde pantalla.
- Solicitud de las imágenes a la cámara.
- Refresco de la posición del robot y del elevador de cargas.
- Actualización del mundo sintético y de la cámara virtual.

Todo el plan de acción funciona de forma paralela al "commonWork" anterior, quedando englobado dentro del estado "uncommonWork".

A continuación se detalla el funcionamiento de cada uno de los estados principales de la máquina. Estos estados quedan representados en la figura 5.5

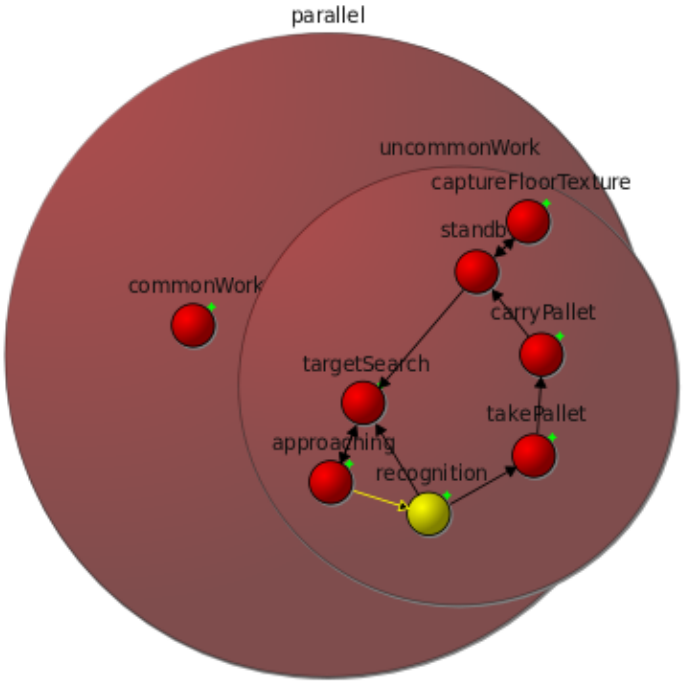

Figura 5.5: Estados que componen el plan de acción de la máquina.

5.2.1. Standby

Este es un estado de espera en el que no se realiza ninguna de las operaciones requeridas para detectar el palé.

Es el estado inicial de la máquina de estados, del que parten las dos posibles vías de ejecución: la toma de la textura del suelo y la ejecución de todo el proceso de detección de palé.

5.2.2. CaptureFloorTexture

En este estado se capturar la textura del suelo para poder realizar todo el proceso de identificación de forma correcta.

Se activa a través de una señal (captureFloorTexture) que se dispara al pulsar un botón de la interfaz. Es necesaria la ejecución de este estado si se aprecia una diferencia clara entre la textura del mundo 3D y el suelo del mundo real. Es un estado que se activa manualmente.

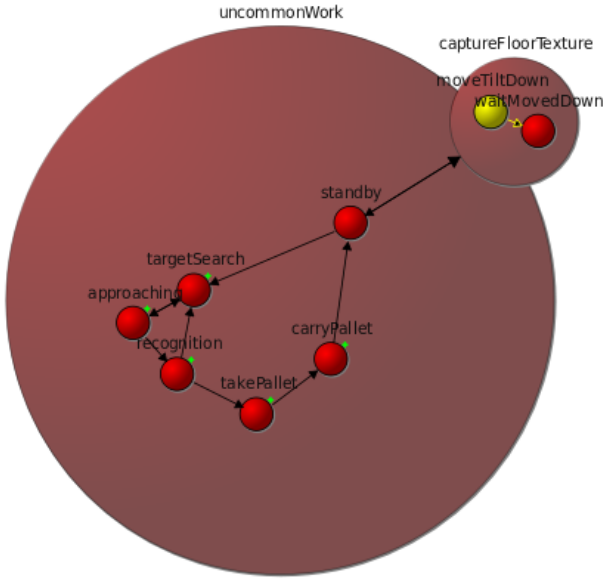

Figura 5.6: Subestados pertenecientes al estado "captureFloorTexture"

Como puede verse en la imagen 5.6, este estado está constituido por dos subestados.

En el primero de ellos se envía la orden de bajar la torreta al componente "headnt2pComp". También se almacena la posición en la que se encuentra la torreta para restaurarla al acabar el proceso de obtención de la textura del suelo.

El otro estado es en encargado de esperar a que el movimiento de la cabeza se haya completado. Una vez que se esta mirando el suelo se captura una pequeña sección central de la imagen para aplicarla como textura al mundo sintético. Mediante este parche de imagen también se calcula el primer promedio de color del suelo que será utilizado en el segmentador.

A continuación, se envía la orden de devolver la torreta a su posición original y activar la transición para volver al estado "standby". En la figura 5.7 puede verse el resultado de esta operación una vez que se ha aplicado la textura al mundo sintético.

Este es un ejemplo de estados asociados a una espera. Tras solicitar la ejecución de una acción a otro componente es necesario esperar a que dicha acción sea llevada a cabo. Este tipo de esperas aparecen en varias etapas de la ejecución del plan pero no van a ser comentadas para simplificar el entendimiento del comportamiento global.

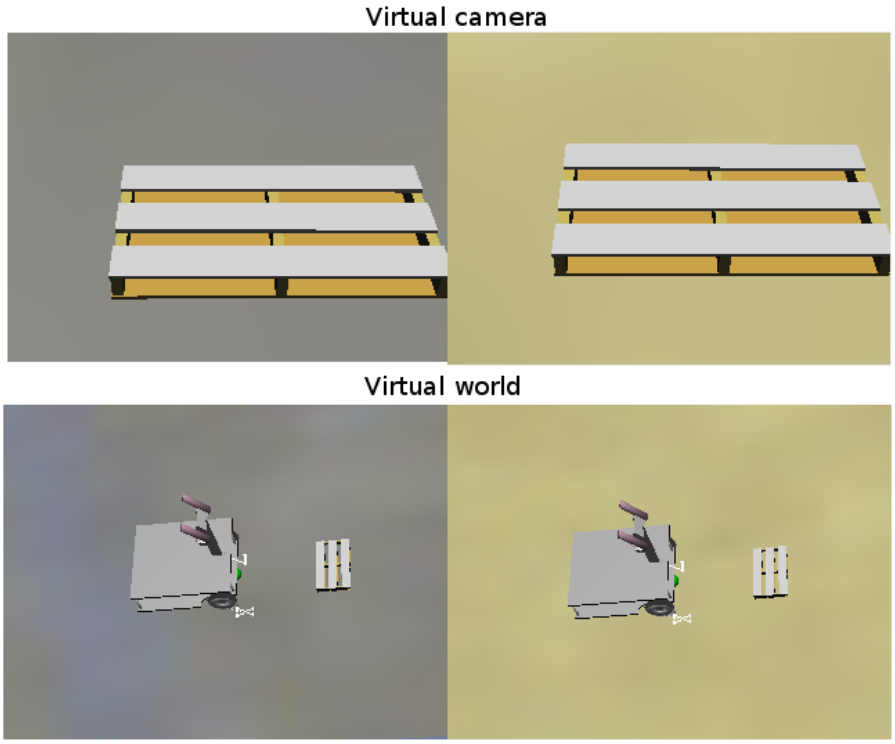

Figura 5.7: Mundo sintético antes(izq) y despues de la captura del suelo(der).

En futuras versiones será necesario el diseño de un sistema capaz de llevar a cabo estas esperas de manera sencilla y eficiente. Gracias a esto se liberarán gran cantidad de estados, con lo que se ganará sencillez y funcionalidad en los comportamientos complejos.

5.2.3. TargetSearch

Este es el estado en el que se inicia todo el proceso de detección del palé. La idea es hacer que el robot mire a diferentes punto del mundo y aplique el algoritmo de identificación inicial.

La cabeza del robot comienza mirando hacia el centro y ligeramente hacia abajo. Mueve las cámaras lateralmente, hasta que alcanzan el máximo del rango de movimiento, momento en el que vuelven al centro y realizan la misma acción en el otro sentido.

Si se ha cubierto el rango completo de movimiento de las cámaras sin haberse

realizado ninguna identificación inicial, realiza un movimiento giratorio de la base para ampliar aún más el campo visual explorado.

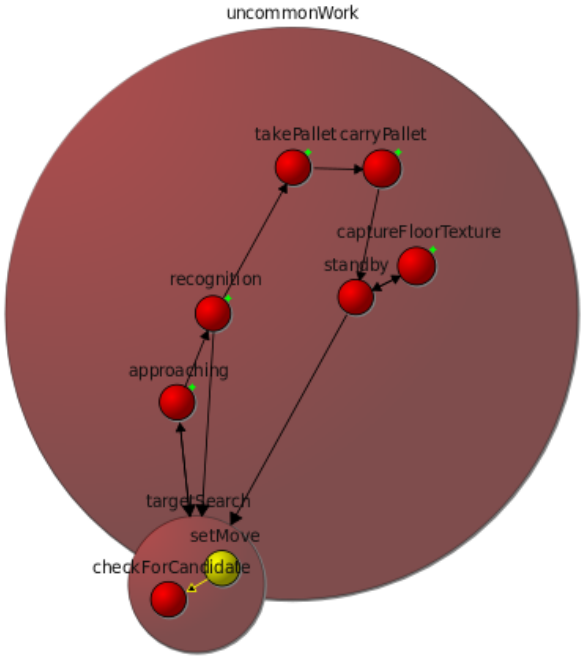

Figura 5.8: Subestados pertenecientes al estado “targetSearch”

5.2.3.1. SetMove

Subestado en el que se calcula cual es el siguiente movimiento a realizar por las cámaras o la base del robot. En cada ejecución se mueve la cabeza del robot y la base de forma que cambie el campo visual abarcando otra zona del mundo.

5.2.3.2. CheckForCandidate

Se utiliza el segmentador 5.1 para obtener las posibles regiones candidatas.

En caso de obtener algún posible palé, se mandará la señal para la ejecución de los estados posteriores. En caso contrario, se vuelve a ejecutar el estado en el que nos encontramos.

5.2.4. Approaching

Al entrar en este estado el robot ya tiene una mancha candidata para ser un palé. Para comprobar si dicha mancha corresponde o no a un palé es necesario aproximarse a ella para tener un "buen punto de vista" y poder discernir si se trata o no de un palé.

Además de los algoritmos desarrollados para el computo de la posición del palé existen diferentes tareas a llevar a cabo para asegurar el correcto funcionamiento 5.9.

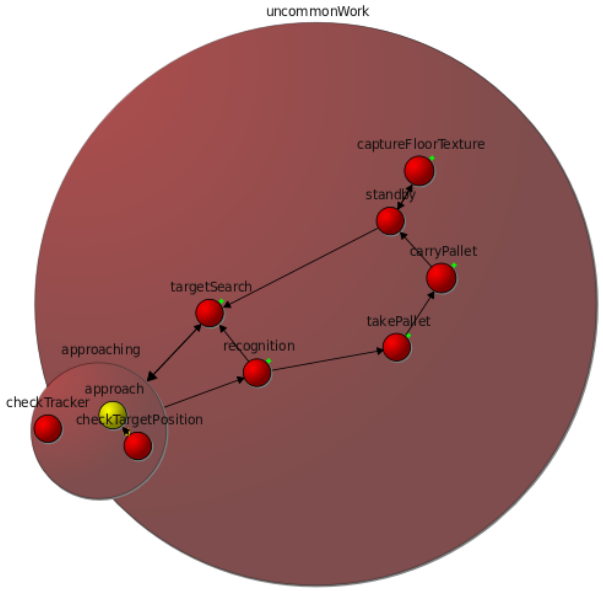

Figura 5.9: Subestados pertenecientes al estado "approaching"

Para facilitar las operaciones sobre las imágenes, se utiliza el componente "trackerComp" 2.2.1.7, que hace un seguimiento del objeto de interés durante la aproximación. De esta forma nos aseguramos de que el objetivo no se salga del campo visual con lo que los algoritmos de estimación operan con las imágenes correctas.

Por otra parte, para alcanzar la posición en el mundo se hace uso del componente "robotTrajectory" 2.2.1.10. A dicho componente se le proporciona el punto y la orientación a la que debe dirigirse y el se encarga de calcular la trayectoria más adecuada.

5.2.4.1. Approach

Durante el proceso de aproximación se van evaluando las imágenes obtenida para ir recalculando la posición de la mancha.

La estimación de la posición de la mancha se realiza matemáticamente, suponiendo que dicha mancha esta en el suelo y conociendo la posición exacta de cada uno de los grados de libertad de la torreta (nos indica donde esta mirando la cámara). Con esto, se puede calcular la posición $3D$ correspondiente al centro de la misma. Como es de suponer, no se trata de un método exacto, estas operaciones se en influenciadas por diferentes factores:

- A cierta distancia la mancha no se ve completa por lo que su centro tampoco resultará ser el centro real.
- El segmentador puede incluir o excluir diferentes partes de la mancha en función de la variabilidad de la luz existente por lo que tampoco asegura el centro real.
- La posición ofrecida por los servomotores que controlan la torreta tiene cierta holgura, una décima de grado no parece un error muy importante, sin embargo, se convierte en un valor a tener en cuenta si lo utilizamos para computar una posición a tres metros de distancia.
- La resolución con la que se trabaja es de $320x240$, a cierta distancia el palé pasa a ser un conjunto muy reducido de píxeles por lo que resulta difícil operar con ellos sin que se cometan errores.
- Todos los cálculos se basan en la posición que tiene el robot dentro del mundo, esta posición es computada mediante la odometría de la base por lo que hay que tener en cuenta los posibles errores derivados de una incorrecta medición de la posición del robot.
- Además de todo lo anterior, el calculo de la posición tiene sus propios errores matemáticos debido sobre todo a las aproximaciones que se llevan a cabo por lo que aún disponiendo de los valores exactos, la posición real de la mancha en el mundo puede variar.

Debido a todas estas limitaciones es necesario llevar a cabo un proceso de aproximación en el que se vaya reestimando la posición del objetivo, cuanto más cerca de él nos encontremos mejores serán las medidas tomadas y por tanto menor será el error cometido en los cálculos.

5.2.4.2. CheckTargetPosition

En este subestado se envía la orden al componente "robotTrajectory". Esto solo se lleva a cabo si la estimación de la posición ha variado sensiblemente respecto a la anterior. De esta forma se evita estar continuamente enviando la misma posición. Si hicieramos esto, el componente "robotTrajectory" tendría que estar recalculando la posición constantemente con lo que el movimiento de la base no se llevaría a cabo de forma suave.

Mediante los valores de posición (x, y, α) del mundo, y haciendo uso de la posición actual del robot, el componente calcula la trayectoria a realizar mediante curvas de Beizér [16]. Los polinomios de tercer grado de Beziér generan curvas sencillas y fáciles de utilizar cuando se conoce la orientación inicial y final. De esta forma se consigue una aproximación al objetivo con una trayectoria suave que además asegura llegar con la orientación correcta.

Además, también se comprueba si se ha llegado a la posición necesaria para realizar la identificación del palé, es decir, si se tiene un buen punto de vista. Este punto de vista viene determinado por la distancia al palé y por el porcentaje de pantalla que ocupa. En caso de encontrarnos en la posición adecuada se activa la transición al siguiente estado.

5.2.4.3. CheckTracker

Cada vez que se recalcula la posición del palé se lleva a cabo una comprobación del seguimiento que realiza "trackerComp" 2.2.1.7. Se verifica que el componente esta "mirando" donde debe mirar, es decir, se comprueba que no se ha perdido. En caso de que el tracker se haya perdido se le asigna de nuevo la región de imagen sobre la que debe hacer tracking.

Este seguimiento durará todo el proceso de aproximación y finalizará cuando se pase al estado de reconocimiento.

5.2.5. Recognition

Cuando llegamos a este estado el objetivo ocupa un 80 % del campo visual. Esta resulta una distancia suficiente para estimar la posición con exactitud. En el caso del palé, esta distancia es de unos $500mm$ aproximadamente.

La tarea principal de este estado, es la de comparar lo que se está viendo en la imagen real (el supuesto palé real) con el modelo de palé sintético 3.2.1. Para mediante estas comparaciones obtener la posición y orientación exacta en la que se encuentra.

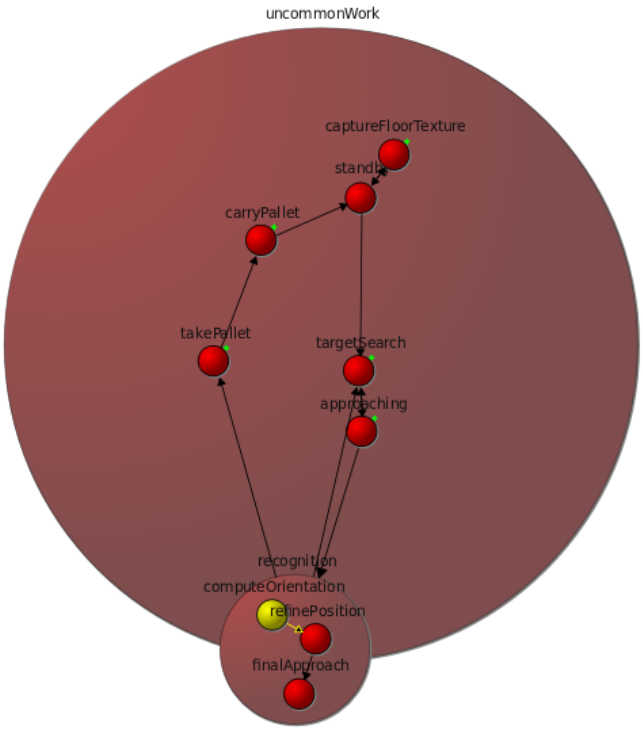

Figura 5.10: Subestados pertenecientes al estado “recognition”

En primer lugar se lleva a cabo un análisis de la orientación del palé (computeOrientation) para utilizarla como punto de partida en la estimación de la posición (refinePosition).

5.2.5.1. ComputeOrientation

Dentro de este subestado se calcula de forma aproximada la orientación del palé en el mundo. Esta primera estimación servirá de entrada al subestado siguiente.

De forma análoga a la estimación de la posición, para la estimación de la orientación se producen una serie de errores, sobre todo debidos a la correcta clasificación de los píxeles como pertenecientes o no a la mancha objetivo.

Para evitar incorporar errores a la hora de estimar la orientación del palé se limita la zona de la imagen sobre la que se va a trabajar. A partir de la estimación de la posición del palé en el mundo, tras un largo de proceso de reestimación esta resulta bastante precisa, se obtiene la zona de la imagen en la que se debe estar viendo el palé mediante el proceso de reproyeccción. Una vez disponemos de la posición que ocupa en la pantalla, recortamos dicha zona de la imagen. Será sobre esta sección de imagen sobre la que se lleven a cabo todos los procesos posteriores.

Para calcular la orientación del palé, se calculan los filtros de Sobel vertical y horizontal.

Mediante los valores obtenidos por ambos filtros se genera un histograma de gradientes. Sean S_H y S_V las componentes horizontal y vertical del gradiente en cada píxel. Calculamos el módulo y el ángulo del gradiente como:

$$mod = \sqrt[2]{S_{V^2} + S_{H^2}} \tag{5.6}$$

$$\alpha = \frac{S_V}{SH} \tag{5.7}$$

Obteniendo una imagen de módulos y otra de ángulos. A continuación, construimos un histograma con los ángulos, utilizando una discretización de cien divisiones. La posición del máximo en este histograma indica la orientación del palé en la imagen.

Para calcular el ángulo en el que se encuentra el palé en el mundo basta obtener dos puntos de imagen que formen una recta con dicha orientación. Por ejemplo, el punto central y un punto desplazado la mitad del ancho y del alto de la imagen con el valor del ángulo obtenido.

A continuación mediante el método de proyecccion trasladamos ambos puntos de la cámara izquierda al mundo. Una vez que tenemos ambos puntos podemos calcular

su arcotangente y con esta obtenemos el ángulo del palé en el mundo.

Gracias a este procedimiento podemos establecer una orientación inicial sobre la que empezar a realizar las comparaciones. Este método reduce el rango de búsqueda considerablemente, pasando de 360º a 11.5º, equivalente a $\pi/16$.

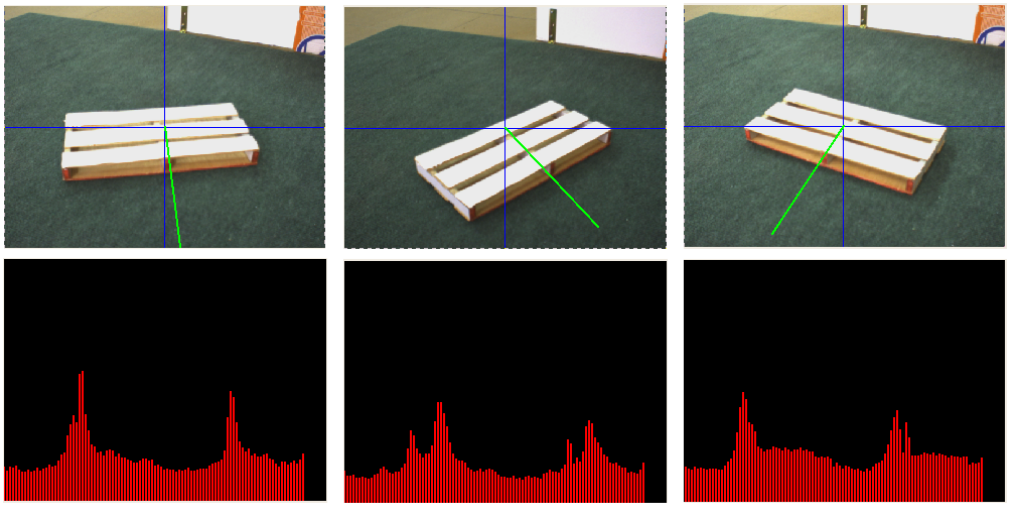

Figura 5.11: Imagenes del palé con sus correspondientes histogramas de gradientes

5.2.5.2. refinePosition

La detección final de orientación y posición del palé se va a realizar comparando el palé real que estamos viendo en la imagen con el que se está viendo en la cámara virtual.

Cada comparación realizada corresponde a una posición y orientación del palé virtual dentro de un rango de posiciones y orientaciones posibles según lo calculado anteriormente. Para cada una de estas comparaciones (figura 5.12) se llevan a cabo los siguientes pasos:

1. Se aplica un filtro Gaussiano a las dos imágenes, la real y la sintética. Con este filtro evitamos una identificación exacta píxel a píxel, debido a los posibles errores de colocación de la cámara virtual.

2. Se obtiene una nueva imagen producto de la resta de las dos anteriores.

3. La imagen resultante es convertida a escala de grises.

4. Calculamos el umbral Otsu y umbralizamos la imagen. Mediante este proceso se obtiene una imagen binaria en el que los $0's$ representan coincidencias entre palé real y virtual y los $1's$ puntos no coincidentes.

5. Se contabilizan el número de puntos coincidentes.

6. Almacenamos la posición y orientación de la comparación que posee un menor número de $1's$

De esta forma, obtenemos una secuencia de comparaciones como la que aparece en la Figura 5.12.

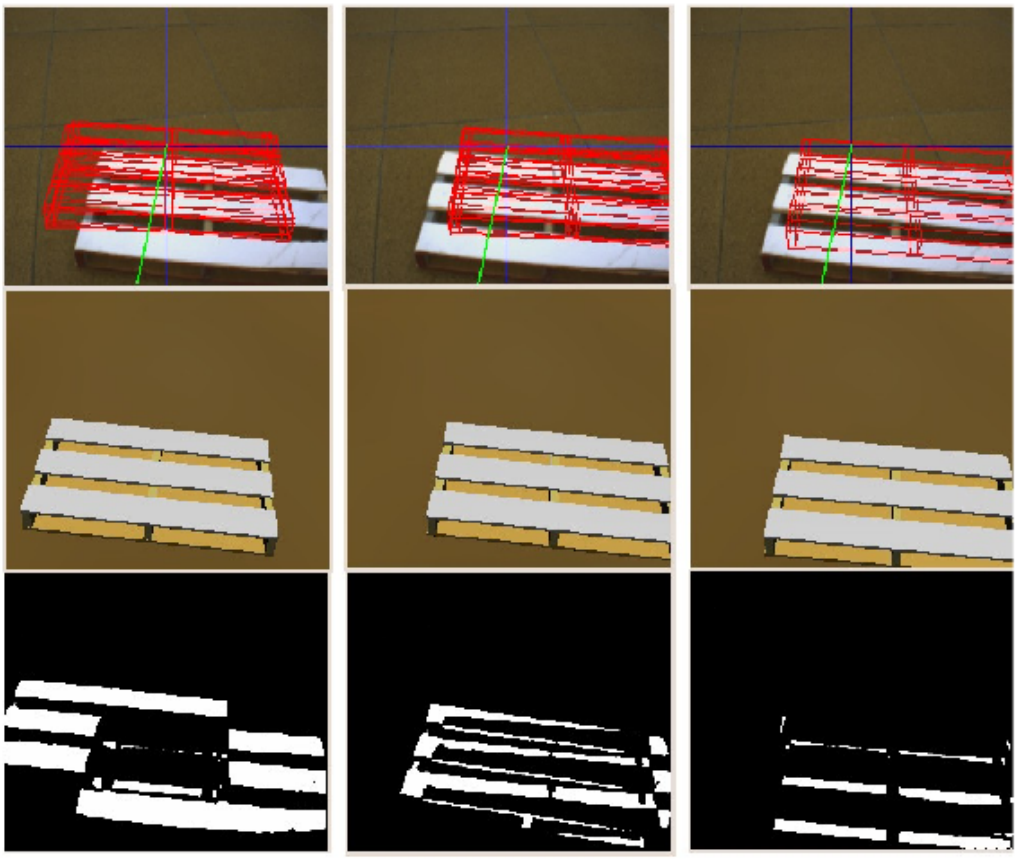

Figura 5.12: Imagenes del palé con sus correspondientes histogramas de gradientes

Este proceso se lleva a cabo dos veces:

- Una vez para un rango de posiciones de $140mm$ con variaciones de $15mm$ y orientaciones en un rango de $\pi/16$ radianes con incrementos de 0,05 radianes.
- Una segunda vez para un rango de posiciones de $30mm$ con variaciones de $4mm$ y orientaciones en un rango de $\pi/32$ radianes con incrementos de 0,02 radianes sobre la posición calculada en el paso anterior.

De esta forma obtenemos una optimización jerárquica de la ejecución del algoritmo, acercándonos cada vez más a la posición y orientación real del palé.

5.2.5.3. FinalApproach

En este subestado disponemos de la posición y orientación exactas del palé. Hay que calcular el punto al que debe dirigirse el robot para quedar alineado con él, para en un movimiento posterior, 5.2.6 avanzar para introducir en él la pinza.

Si d es la distancia del punto al palé (aproximadamente unos $500mm$), α es la orientación del palé en el sistema de referencia del mundo y x_p e y_p son las coordenadas del palé también en el mundo, entonces:

$$x = -d * \sin(\alpha) + x_p \tag{5.8}$$

$$y = -d * \cos(\alpha) + y_p \tag{5.9}$$

Las coordenadas del punto que buscamos son (x, y).

Una vez calculado este punto se envía la orden de alcanzarlo al componente "robotTrajectory" y de bajar el elevador al componente "forkComp". Ambas acciones se realizan simultáneamente.

5.2.6. TakePallet

En este estado se lleva a cabo el proceso de coger el palé.

En primer lugar se lleva a cabo una revisión final de la posición y orientación del mismo. A continuación, se ordena a la base robótica avanzar para introducir completamente la pinza en el palé y proceder a alzarlo. Estas tres acciones quedan representados como los subestados de la figura 5.13.

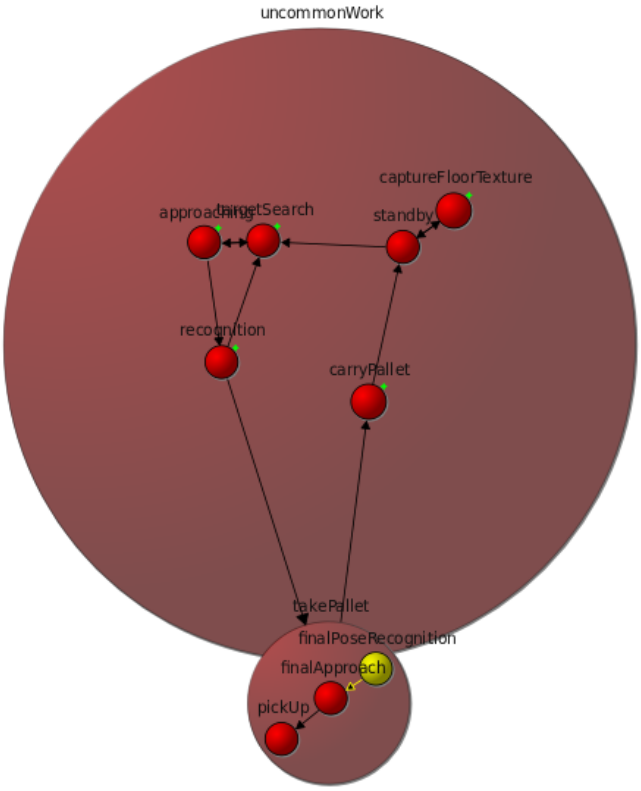

Figura 5.13: Subestados pertenecientes al estado "takePallet"

5.2.6.1. FinalPoseRecognition

Al entrar en este subestado se lleva a cabo un nuevo proceso de estimación de posición y orientación como el comentado en el apartado 5.2.5.2. Con esta nueva estimación nos aseguramos de que la trayectoria hasta el punto frente al palé se ha llevado a cabo correctamente. Los valores obtenidos son utilizados en el siguiente subestado.

5.2.6.2. PickingManeouvre

Con la posición y orientación exactas del palé (obtenidas en el paso anterior), tenemos que calcular la distancia y la rotación en la que debe avanzar el robot para que introduzca completamente el elevador en el palé y este listo para alzarlo.

Siendo d_{p-r} la distancia entre el palé y el robot, W_p el ancho del palé, α_{ent} el ángulo de entrada del robot y x_r e y_r las coordenadas del robot en el mundo,

mediante la ecuación:

$$x = (d_{p-r} - W_p/2) * \sin(\alpha_{ent}) + x_r \tag{5.10}$$

$$y = (d_{p-r} - W_p/2) * \cos(\alpha_{ent}) + y_r \tag{5.11}$$

(x, y) serán las coordenadas del punto que queremos calcular. Una vez se ha calculado se envía al robot hacia él. Cuando el robot llegue a dicho punto, la pinza del elevador estará en el palé y podremos proceder a izarlo.

5.2.6.3. PickUp

Este es uno de los estados más sencillos. En el solamente se ordena al elevador levantar la pinza y se espera hasta que dicha acción ha sido llevada a cabo. A continuación, se activa la transición al estado siguiente.

5.2.7. CarryPallet

Dentro de este estado no se llevan a cabo operaciones complejas. Su funcionamiento se limita a enviar a la base a un punto destino prefijado para depositar allí el palé (subestado "movetoDestination").

Una vez que el robot alcanza dicho punto, procede a bajar el elevador para dejar el palé sobre el suelo (subestado “release”), y luego retrocede para que el siguiente movimiento no arrastre al palé con la pinza.

Una vez concluido esto, se activa la transición que devuelve el plan al estado de reposo (“standby”). Con esto el robot queda listo para comenzar la búsqueda de un nuevo objetivo.

Aunque en el diagrama solo se muestran dos subestados (Figura 5.14), la realidad es que esta consituido por muchos subestados. Todos debidos a las necesarias esperas que hay que establecer entre las diferentes operaciones.

- Enviar al robot al destino $\longrightarrow$ Hay que esperar a que llegue al destino.
- Bajar elevador $\longrightarrow$ Comprobar la posición hasta que el elevador llega hasta abajo.
- Retroceder el robot $\longrightarrow$ Esperar hasta que el robot haya retrocedido.

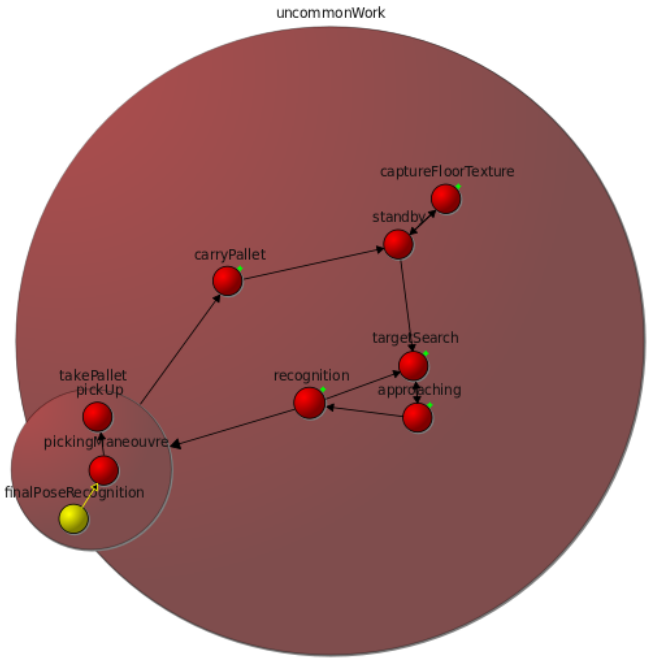

Figura 5.14: Subestados pertenecientes al estado “carryPallet”

Capítulo 6

Experimentos

Los experimentos se realizaron en un espacio dedicado a los robots dentro de RoboLab (ver Figura 6.1).

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

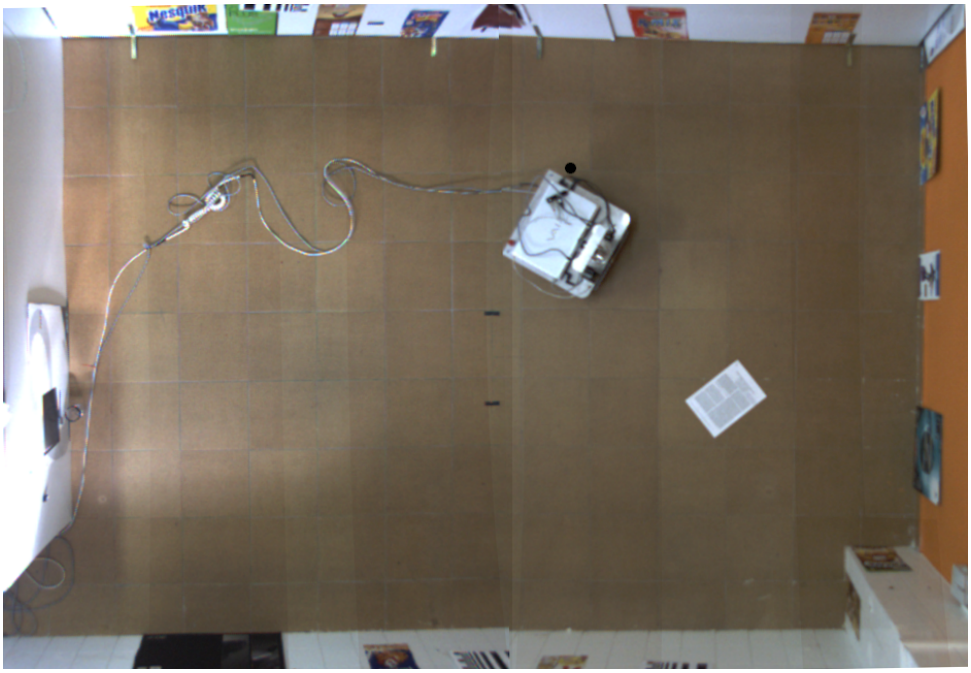

Figura 6.1: Captura cenital del entorno en el que se realizaron los experimentos.

Durante la ejecución de estos experimentos se ha detectado un error en la estimación de la orientación del robot. En esta fase se utiliza la imagen para calcular

el histograma de gradientes. Como dicha fase se produce al final del proceso de aproximación, en ocasiones, la imagen capturada por la cámara estaba movida (se ha capturado antes de encontrarse el robot o la torreta totalmente parados). Esto provocaba un error muy grande en la estimación de la orientación por lo que las subsiguientes etapas no funcionaban correctamente.

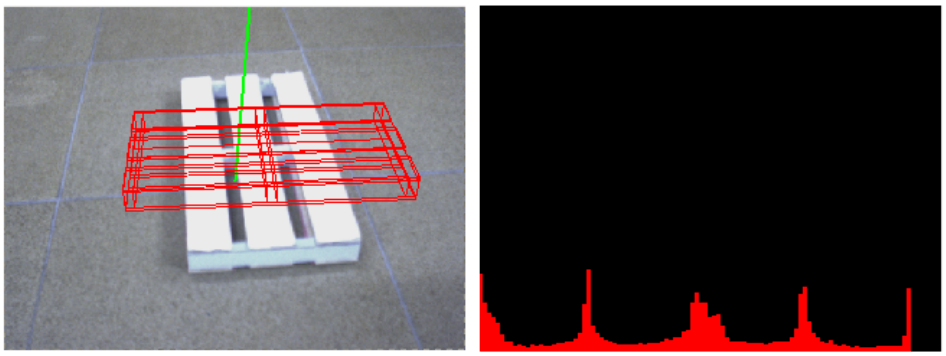

Figura 6.2: Detección de la orientación del palé incorrecta.

Una vez subsando el error anterior, se procede a comentar el desarrollo de los experimentos.

Para validar los algoritmos desarrollados y la máquina de estados que los ejecuta se ha hecho el siguiente diseño experimental: la relación entre el robot y el palé se presenta con dos variables, d y a, que codifican la distancia entre ambos y el ángulo relativo respectivamente. La distancia se ha discretizado en tramos de 100, 150 y 200 cm. El ángulo relativo se ha discretizado en siete tramos de $-\pi/2$, $-\pi/3$, $-\pi/4$, 0, $\pi/4$, $\pi/3$ y $\pi/2$ radianes. Para cada uno de estos valores de distancia y ángulo el experimento se repite tres veces, sumando un total de 63 pruebas. Los resultados de estos experimentos se muestran en la siguente tabla:

	$-\pi/2$	$-\pi/3$	$-\pi/4$	0	$\pi/4$	$\pi/3$	$\pi/2$
100	✓ X	✓	✓	✓	✓	✓	✓ X
150	✓ X	✓	✓	✓	✓	✓	✓ X
200	✓ X	✓	✓ X	X	✓ X	✓	✓ X

Durante la realización de los experimentos se han detectado diferentes situciones en las que no se consigue el objetivo. Estas situaciones puede ser englobadas en dos grupos:

- Los fallos que se han detectado en las orientaciones del palé de $-\pi/2$ y $\pi/2$ radianes se deben principalmente a los errores provocados en la estimación de la orientación del palé. En estos casos, la orientación no se estima correctamente debido a la aparición en el histograma de orientaciones de varios picos 6.2. Uno de ellos es el que corresponde a la correcta orientación del palé, sin embargo, si no es este el escogido se produce un desvio en dicha orientación con lo que el cálculo del punto al que debe dirigirse el robot para quedar frente al palé y poder así cogerlo es erroneo. Una posible solución pasaría por la obtención de una serie de estimaciones diferentes y una votación entre ellas para escoger la más prometedora.

- El otro tipo de error se debe a la distancia a la que se encuentra el palé. En algunas ocasiones, el palé no es detectado correctamente durante la fase de búsqueda de objetivos. Como puede verse en la figura 6.3 la cantidad de píxeles correspondientes al palé es muy reducida. Es posible llevar a cabo la detección si disminuimos el umbral establecido en el segmentador, sin embargo, esto redunda en la aparición de falsos positivos. En la figura 6.3 se muestra el palé con una orientación de $-\pi/2$ y 0 radianes respectivamente. Se observa que zona de imagen que ocupa el palé cuando este se encuentra girado respecto al robot aumentan. Como puede verse en los resultados de los experimentos, la detección mejora al encontrar una mayor superfície del palé. Como solución a este problema se propone la realización de una espiral mediante la base robótica en los casos en los que no se detecte ningún objetivo inicialmente.

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

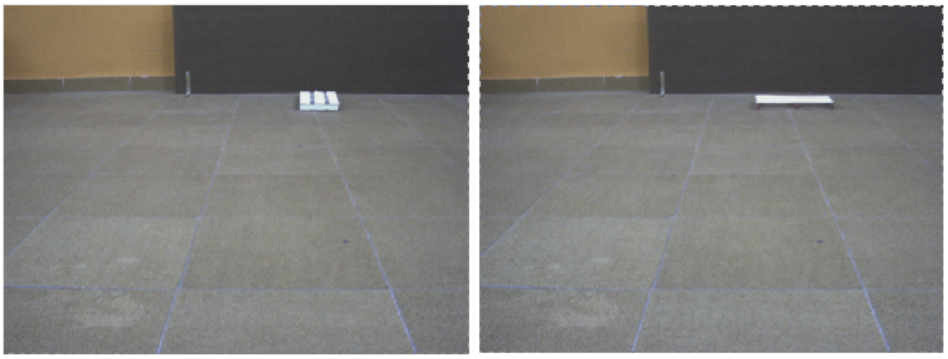

Figura 6.3: Vista del palé a dos metros de distancia.

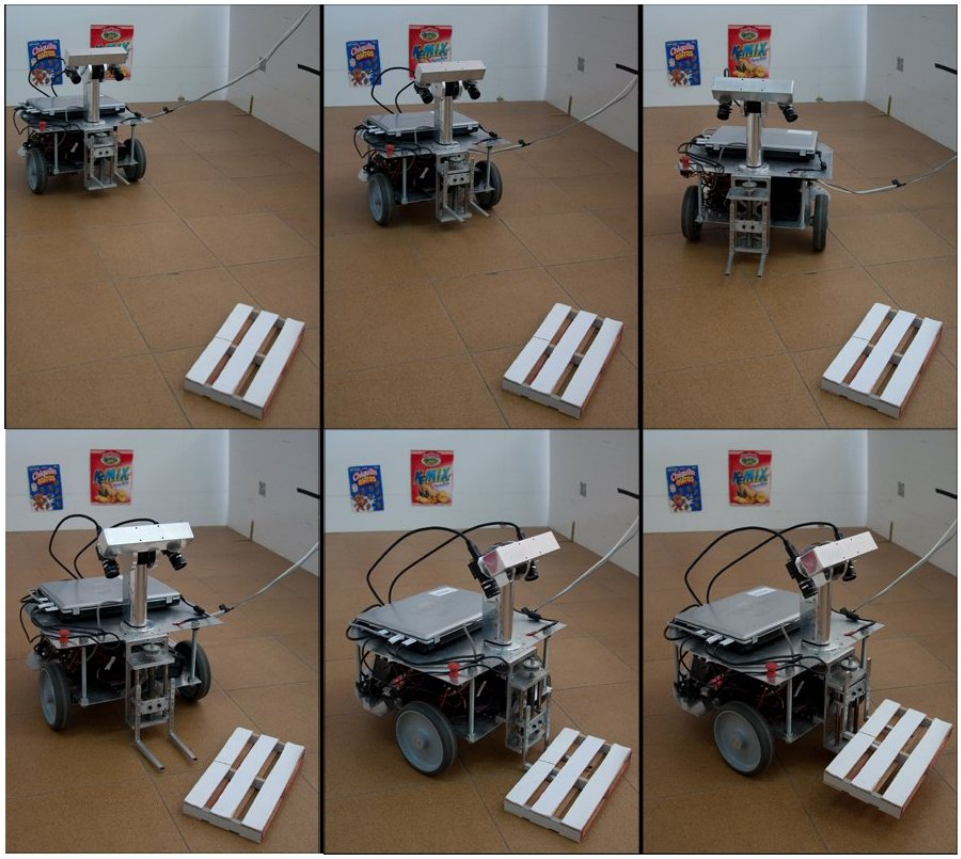

Figura 6.4: Robex cogiendo un palé.

Capítulo 7

Conclusiones

Las pruebas realizadas demuestran que las técnicas desarrolladas hacen que el robot se pueda desenvolver bien en entornos libres de obstáculos, dificultad 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.

El uso de técnicas de diseño de escenarios en tres dimensiones para la detección de objetos en el mundo resulta bastante novedoso y ha proporcionado muy buenos resultados. La calidad del diseño 3D del entorno aumentará cuando mayor sea la información obtenida de él. Gracias al diseño de componentes, esta información puede aumentar con los datos recibidos de los sensores que se pueden añadir a la plataforma robex.

La interacción con el resto de comportamientos del robot, en especial con los que requieren tiempo, ha supuesto una gran fuente de problemas. En futuras versiones será necesario el diseño de un sistema de acceso sencillo y efectivo para poder simplificar el desarrollo de comportamientos complejos.

En general, se necesita un buen funcionamiento de los componentes utilizados y una cierta calidad en las imágenes recibidas. Sin embargo, los resultados obtenidos demuestran que es posible generar un método de aproximación robusto y fiable disponiendo de poca información externa.

7.1. Trabajos futuros

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

- Aumentar el detalle del mundo virtual, añadiendo más información adquirida del escenario por medio de componentes como "cubeComp" o "laserComp". También se puede aumentar el detalle aplicando texturas o colores a las paredes del mundo y a la superficie del palé, que actualmente es blanca.

- 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.

- Aumentar la precisión de la detección incluyendo un algoritmo de identificación positiva del objeto que estamos analizando, es decir, comprobar si aquello que estamos observando y calculando posición y orientación, se trata realmente de un palé.

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

- Expandir el ámbito del problema a palés con carga encima de ellos, ya que actualmente no se contempla esa situación.

- Implementar un sistema de espera para las acciones que conllevan un tiempo. De esta forma se podría reducir notablemente el número de estados involucrados en la máquina.

- 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.

- Implementar algún tipo de memoria en la que almacenar la información sobre objetos candidatos que ya hayan sido descartados. De esta manera se evitaría la repetición del análisis sobre objetos que ya han sido analizados previamente. También eliminaría errores provocados por la aparición de objetos secundarios durante la fase de aproximación.

Bibliografía

[1] Bachiller P. *Percepción dinámica del entorno en un robot móvil*. Tesis doctoral. 2008.

[2] Brooks A, Kaupp T, Makarenko A, Williams S, Oreback A. *Orca: A Component Model and Repository*. Software Engineering for Experimental Robotics, Springer, pp. 231-251, 2007.

[3] Harel D. *Statecharts in the Making: A Personal Account* Communications of the ACM Vol 52 N^{o} 3, March 2009

[4] Manso L, Bustos P, Bachiller P. *Multi-cue Visual Obstacle Detection for Mobile Robots*. Journal of Physical Agents. 2010;4(1):3-10.

[5] Qt State Machine Framework *http://doc.trolltech.com/solutions/4/qtstatemachine/*

[6] Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R, Ng A. *ROS: an open-source Robot Operating System*. ICRA Workshop on Open Source Software, 2009.

[7] Seelinger M, Yoder J. *Automatic Pallet Engagment by a Vision Guided Forklift*. Proceedings of the 2005 IEEE International Conference on Robotics and Automation. 2005;(April):4068-4073.

[8] Wasik Z, Saffiotti A. *A hierarchical behavior-based approach to manipulation tasks*. In Proceedings of the IEEE International conference on robotics and automation.Vol 2. Pag: 2780–2785. 2003

[9] L. J. Manso, P. Bustos, P. Bachiller, P. Núnez, R. Cintas y L. Calderita. *Un framework de desarrollo para robótica* .I Jonadas jóvenes investigadores.2010;(Abril).

[10] L. Manso , Bustos P, Bachiller P, Cintas R, Calderita L, Núñez P. *RoboComp: a Tool-based Robotics Framework.* SIMPAR, International Conference on Simulation, Modeling, and Programming for Autonomous Robots. 2010

[11] *Robolab. Robex arena* http://robexarena.com. 2010.

[12] *Robolab. RoboComp project* http://robocomp.wiki.sourceforge.net. 2010.

[13] *Página del laboratorio de robótica y visión artificial de la Universidad de Extremadura.* http://robolab.unex.es 2010

[14] J. Mateos, A. Sánchez, L. J. Manso, P. Bachiller, P. Bustos. *RobEx: an open-hardware robotics platform.* WAF Valencia 2010.

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

[16] Choi, Ji-wung, Elkaim, Gabriel Hugh. *Bézier Curve for Trajectory Guidance.* Computer Engineering. Pag: 0-5.

[17] Wasik Z, Saffiotti A. *A hierarchical behavior-based approach to manipulation tasks.* In Proceedings of the IEEE International conference on robotics and automation.Vol 2. Pag: 2780–2785. 2003

[18] ZeroC. *ZeroC Customers, http://zeroc.com/customers.html.*

[19] Harel D. *Statecharts in the Making: A Personal Account Communications* of the ACM Vol 52 No 3, March 2009

[20] Pedro F. Felzenszwalb and Daniel P. Huttenlocher *Efficient Graph-Based Image Segmentation* International Journal of Computer Vision, Volume 59, Number 2, September 2004

[21] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, S. Thrun. *Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing.* Autonomous Robots 18(1), 81–102 (2005).