Universidad de Extremadura

Escuela Politécnica de Cáceres

Ingeniería Técnica de Telecomunicación

Especialidad en Sonido e Imagen

Proyecto fin de carrera

Virtual Blackboard: colour and human gestures motion tracking.

Leandro Pavón Serrano
76043575D

Director: Pedro M. Núñez Trujillo

Cáceres, Septiembre de 2011

Abstract

This work belongs to the field of Computed Vision. It describes how two software components are developed by using different tracking systems and adapting it to interactive applications. One of that is a virtual blackboard using Hough transform for the circle detection and Kalman Filter for the prediction task. The capabilities of using a distance filter implemented with Kinect also is explained in a resumed way. The second application uses ARToolKit to tracking and OSG to set up a virtual world becoming an Augmented Reality game as a proposal for rehabilitation exercises.

Resumen

El presente Proyecto Fin de Carrera consiste en el desarrollo de dos aplicaciones informáticas que hacen uso de los métodos habituales de *tracking* de objetos en imágenes de vídeo y pertenece al campo de la visión por computador.

El primero de los programas que se han llevado a cabo ha sido un pizarra virtual, donde el usuario puede pintar simplemente moviendo un círculo rojo delante de una cámara web. Para realizar el seguimiento del objeto de color se ha utilizado la transformada de Hough en la detección y la técnica del Filtro de Kalman para la predicción; además se han realizado algunas pruebas de procesamiento de imagen adicionales como el cambio de espacios de color, binarización, etc. También se hace un sencillo resumen sobre el funcionamiento del dispositivo Kinect, y se ha elaborado un filtro por distancia que hace uso de este dispositivo con la intención de que sirva de ayuda en el seguimiento del puntero de la pizarra virtual.

La segunda aplicación que se ha escrito usa ARToolKit para el seguimiento de marcas y OSG para dar forma a un mundo tridimensional que se superpone sobre la imagen capturada por una cámara, creando de este modo un juego de realidad aumentada. Dicho juego tiene el propósito de facilitar la ejecución de ejercicios de rehabilitación por pacientes, con el fin de evitar los abandonos de dichos ejercicios, especialmente en niños.

La estructura del código de la aplicación sigue el modelo del *software* desarrollado en el Laboratorio de Robótica de la Universidad de Extremadura (UNEX) (RoboLab) de la Escuela Politécnica de Cáceres (EPCC), de forma que es compatible con el resto de componentes de RoboComp[1]. De este modo el código es reutilizable, principalmente para funcionar en robots orientados a la educación o la sanidad.

Descriptores

Realidad aumentada, Kinect, tratamiento de imagen, interacción hombre-máquina, *tracking* de color, imagen 3D

[1] *Software* para robots orientado a componentes desarrollado en el RoboLab.

Índice general

Índice de figuras

1. INTRODUCCIÓN

1.1 Motivaciones

El principal interés en realizar este proyecto, es la oportunidad de participar en la evolución de nuevas tecnologías desde un punto de vista técnico, el cual pone a prueba mis capacidades como Ingeniero Técnico de Telecomunicación. Además el campo de la visión por computador y específicamente la Realidad Aumentada (RA) está en continua expansión, de forma que este Proyecto Fin de Carrera (PFC) supone una iniciación en un área con futuro, tanto en su vertiente investigadora como laboral. Por otro lado, las asignaturas relacionadas con el tratamiento de la imagen o la predicción de señales de audio que he cursado durante la carrera, me han resultado más atractivas, y han sido influyentes para elegir desarrollar esta herramienta. Considero que la consecución de un buen proyecto es la meta de los anteriores años de estudio, y en ese proceso, se necesita desarrollar una actitud y responsabilidad de trabajo adecuadas, buenas costumbres de programación, y capacidad de trabajar en equipo; ya que esta herramienta es un componente que puede ser utilizado en el futuro para proyectos de otros estudiantes. Otro aspecto a destacar de la elección de este proyecto frente a otros, es el carácter práctico y útil del mismo, de forma que el tiempo y esfuerzo invertido en su desarrollo se aproveche por los demás; es por esto que todo el código se publica bajo una licencia de libre distribución.

1.2 Objetivos

Más allá de formar parte del plan de estudios de la Ingeniería Técnica de Telecomunicación (ITT), este proyecto tiene como meta participar en la generación de tecnologías de la imagen, que posteriormente puedan ser utilizadas en beneficio de la sociedad.

De forma global se persigue la creación de un componente que se integre en el *framework Software* para robots orientado a componentes desarrollado en el RoboLab (RoboComp), de forma que futuros componentes se puedan conectar a éste del mismo modo que dicho componente se conecta al componente que gestiona el dispositivo *Kinect*® (Kinect) o a una cámara USB. La finalidad de dicho componente es la detección y el *tracking* de objetos mediante sus características de color, forma, y posición en el espacio. Tras esto, el componente debe ser capaz de reconocer gestos o determinados movimientos de forma que se cree la interacción hombre-máquina. Para comprobar su correcto funcionamiento y su potencial, se pretende crear una aplicación sencilla de dibujo como lo es una pizarra virtual, además se ha desarrollado otra aplicación más compleja usada para ejercicios de

rehabilitación de codo u hombro.

Los tres objetivos principales del proyecto son:

- Investigar los diferentes métodos de *tracking* analizando las posibilidades para su uso en una aplicación sencilla.
- Implementar un algoritmo que use uno de los sistemas estudiados en el punto anterior.
- Desarrollar la interacción entre el usuario y el programa.

A continuación se enumera una serie de objetivos secundarios, los cuales se necesitan cumplir para alcanzar los primeros o son paralelos a éstos:

- Programación de un componente integrado en RoboComp.
- Uso de herramientas de programación en entornos GNU/Linux; como KDevelop, cmake, subversion, etc.
- Desarrollo de clases dentro de la aplicación, de forma que el código se pueda reutilizar con más facilidad.
- Instalación de varias librerías de forma manual y su uso mediante la *Application Programming Interface* (API) de cada una de ellas: ARToolKit, OpenCV, Freenect, OSG, OSGArt, OSGAudio, etc.
- Uso, calibración y comprensión del funcionamiento general de Kinect.
- Desarrollo de interfaces gráficas de usuario con las librerías QT.
- Aprender LaTeXpara escribir la memoria del proyecto.

1.3 Organización de la memoria

Esta memoria está dividida en cinco capítulos principales y tres anexos, además de la bibliografía.

El primer capítulo sitúa al lector sobre qué trata este proyecto, comenzando desde el campo general de la visión por computador, hasta las aplicaciones concretas como la pizarra virtual o la aplicación para ejercicios de rehabilitación. Además se nombran aplicaciones similares y se hace especial mención a las diferencias con este trabajo. También se plantean los objetivos principales y secundarios así como la motivación que ha llevado al autor a dedicarse a este área.

En el segundo capítulo se plantea el diseño y el enfoque de la aplicación, y la programación estimada en un primer momento para cumplir con los objetivos expuestos en el capítulo anterior. Aquí también se explica con más detalle la magnitud del problema que plantea este proyecto y el esquema temporal de todo el trabajo

realizado. Además se hace un repaso de la instalación de las librerías utilizadas y del entorno de trabajo.

El capítulo 3 muestra la solución aportada al problema descrito en el capítulo 2, cómo se ha desarrollado y cómo funciona cada aplicación. Se explican con detalle las clases creadas y algunas de las funciones más importantes que se han utilizado, bien propias o de la API de alguna librería. En el caso correspondiente para cada librería se trata su integración en el programa desarrollado.

En el cuarto capítulo, se explican los problemas que han ido surgiendo, así como los límites de la aplicación desarrollada y las posibilidades de mejora en un trabajo futuro.

El último y quinto capítulo es una valoración personal donde se comenta todo lo aprendido, y se evalúan los objetivos que nos marcamos al inicio.

La bibliografía se ha dividido en dos partes: Libros y Artículos, y Otras fuentes. El motivo es que los libros y artículos son fáciles de encontrar y se pueden tomar como referencias más contrastables, mientras que en otras fuentes consultadas como documentación en internet está sujeta a cambios, por lo que dicha documentación puede haber cambiado de dirección o actualizado el propio contenido. Además, en Otras fuentes se encuentran las direcciones desde las cuales se puede acceder a todo el código al que se hace mención en esta memoria.

Finalmente se añaden tres anexos: un glosario de términos y acrónimos, la licencia bajo la cual se publica todo el código desarrollado y una lista de agradecimiento. El glosario de términos y acrónimos es necesario debido a que en este documento se usan con frecuencia varias siglas, y pretende ayudar al lector a seguir el texto sin problemas. Gracias a que la mayor parte del *software* usado se distribuye bajo licencias libres, el código de este proyecto se publica sin problemas bajo una licencia del tipo GNU Public License (GPL). El motivo de añadir un anexo de agradecimientos se debe a la ayuda recibida de muchas personas que merecen ser mencionadas.

La memoria se ha realizado en LaTeX.

1.4 Estado del arte

Con el fin de aclarar qué aporta este proyecto al campo de la visión por computador, se repasan los últimos avances tecnológicos relacionados y se exponen diferentes proyectos similares a éste.

1.4.1 Visión por computador

La visión por computador es un campo de investigación relativamente nuevo y en constante expansión [16]. La dificultad de procesar los grandes lotes de información de las imágenes retrasó su despegue hasta hace tan solo unos cuarenta años. Esta tecnología cubre una amplia gama de problemas cuyas soluciones suelen relacionarse con otras disciplinas como el procesamiento de imágenes, la inteligencia artificial, visión robótica, etc. Es por este motivo que no hay un estándar definido de los límites entre visión por computador y dichas disciplinas. Además no existe una formulación estándar de cómo la visión por computador debería resolver los distintos problemas que plantea. Muchas de las soluciones a problemas concretos y bien definidos se

encuentran en fase de investigación, pero cada vez existen más métodos que tienen posibles usos comerciales, y esto a contribuido a popularizar la visión por computador y sus aplicaciones.

De los múltiples campos en los que existen aplicaciones, se comentan algunos en la siguiente lista:

- Medicina: Actualmente las aplicaciones médicas basadas en visión por computador se han convertido en herramientas imprescindibles para la diagnosis de enfermedades, ya que permiten destacar las zonas importantes en las imágenes proporcionadas por sensores y escáneres de distintos tipos (ecografía, encefalograma, etc.).
- Educación: Existen diversas aplicaciones en este área con el fin de convertir la enseñanza en más atractiva y dinámica. Un ejemplo sería el uso de libros de realidad aumentada donde podemos interactuar con modelos 3-D.
- Entretenimiento: Si dentro del entretenimiento contáramos sólo los juegos ya tendríamos multitud de aplicaciones desarrolladas en base a visión por computador, pero además existen aplicaciones que van más allá de los videojuegos, como podría ser el uso de reconocimiento de edificios emblemáticos para obtener información desde un punto de vista turístico.
- Robótica: La visión por computador está estrechamente relacionada con los robots autónomos y un ejemplo de uso sería la reconstrucción de escena o el rastreo de caminos.
- Accesibilidad: De forma más pionera, se están desarrollado aplicaciones que suponen una ayuda para personas con discapacidad. Por ejemplo una persona ciega puede aprovechar la visión por computador de forma que una aplicación puede traducir obstáculos en señales acústicas.

1.4.2 Tracking y Pizarra Virtual

La mayoría de los sistemas de *tracking* suelen incluir un ejemplo sencillo donde se pinta en la pantalla siguiendo la trayectoria del objeto a seguir, sin embargo la importancia en estos casos es el desarrollo del *tracking* y el reconocimiento de gestos o trayectoria. El enfoque de este proyecto es similar, ya que le da más importancia al estudio de los métodos usados para seguimiento de objetos que a la propia aplicación final como pizarra virtual.

1.4.3 Realidad Aumentada

La RA se encarga del estudio de técnicas que permiten la integración en tiempo real entre el mundo real (normalmente a partir de imágenes capturadas u otra información como geolocalización) y el mundo virtual (que normalmente añade información digital, cálculos o modelos 3-D). La RA se menciona por primera vez en 1990, pero no es hasta 1999 cuando el concepto está más desarrollado y se puede

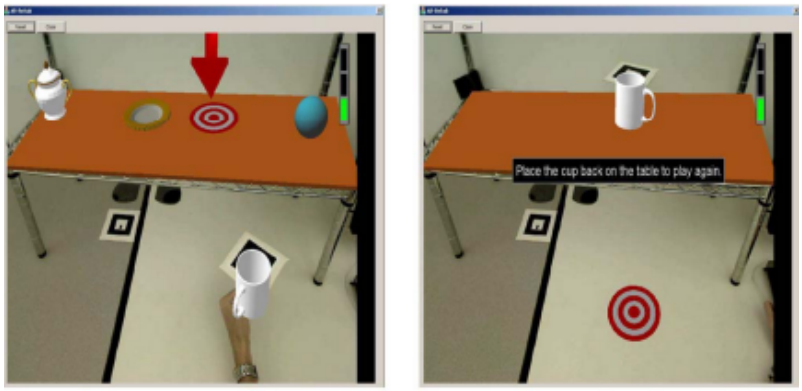

Figura 1.1.: Una aplicación de AR-REHAB, donde el paciente debe llevar el objeto a la posición que le indica la flecha.

llevar a la práctica fácilmente gracias a la librería ARToolKit [3]. Desde entonces cada vez surgen nuevas aplicaciones que hacen uso de esta librería u otras similares, donde la importancia es claramente la interacción.

■ Aplicación para ejercicios de rehabilitación

Además de intentar desarrollar la pizarra virtual, en este proyecto se ha creado una aplicación más avanzada y práctica enfocada al uso de la RA para ejercicios de rehabilitación. A continuación se exponen otros trabajos que también hacen el uso de la RA aplicada a rehabilitación de personas.

- AR-REHAB es un *framework* para desarrollar aplicaciones que usen RA para ejercicios de rehabilitación [2], como se muestra en la Figura 1.1. La idea es que los pacientes practiquen los ejercicios con un objeto al cual se le aplica un *tracking* que determina si el ejercicio se está haciendo de forma correcta o no. Incluye dos aplicaciones orientadas a pacientes con síndrome *poststroke* que hagan ejercicios de rehabilitación de brazo o mano, además de un sistema de perfiles para los pacientes. Para el *tracking* utiliza la librería ARToolKit, para crear el mundo 3-D y controlar las colisiones entre objetos usa CHAI 3D y ODE respectivamente.

- Virtual Harp es un programa especializado en pacientes con discapacidades físicas en manos y dedos [15], y usa la RA para permitirles tocar un arpa virtual, de forma que los ejercicios de rehabilitación consisten en tocar canciones conocidas por el paciente. Usa un Head Mounted Device (HDM) para mostrar al paciente la imagen final, mientras que usa un dispositivo del tacto (Phantom Omni) que permite el *tracking* en el espacio 3-D, pero usa ARToolKit para establecer una referencia. En la Figura 1.2 se ve en la pantalla del ordenador la imagen que el paciente ve a través del HDM.

Figura 1.2.: Virtual Harp usado por pacientes. Se aprecia el uso de HDM, Phantom Omni y la marca de ARToolKit.

2. DISEÑO

Este capítulo pretende explicar todo el trabajo realizado en este PFC desde un punto de vista general, explicando las herramientas que se han usado, las decisiones tomadas, y las distintas etapas a lo largo del tiempo de ejecución.

2.1 Planificación del proyecto

Se empieza a trabajar en este PFC en Febrero de 2011 estableciendo los objetivos iniciales mencionados en la Sección 1.2 de esta memoria, así como las etapas necesarias para cumplir dichos objetivos, las cuales se pueden apreciar en la Figura 2.1 de la página 8 .

Sin embargo esa planificación no ha podido cumplirse completamente, principalmente los tiempos, pero también las etapas desde la A2.10 a la A4.5 han sido suprimidas o sustituidas por otras. Son varios los motivos que han llevado a que se hagan estos cambios, por un lado está el hecho de que los conocimientos previos en cuanto a programación orientada a objetos era nula, pero además el uso de herramientas nunca antes utilizadas, y el desconocimiento de problemas para conseguir un *tracking*, han contribuido a que la estimación preliminar del proyecto fuera muy complicada de cumplir en tiempo. Por su parte los cambios en las etapas, se deben a una propuesta en mitad del desarrollo del PFC en la que se plantea crear un programa para ejercicios de rehabilitación en lugar de la pizarra virtual, debido a que este nuevo programa tendría una aplicación inmediata gracias a la participación de RoboLab en el proyecto ACROSS. Además se podría añadir una etapa más, que consiste en una estancia durante los meses de Julio, Agosto y Septiembre en el Instituto de Sistemas e Robótica (ISR) de la Universidad de Coimbra durante la cual se ha trabajado en un programa que plantea el *tracking* como un problema probabilístico y hace el uso de la librería Professional Bayes Toolkit (ProBT).

De esta forma el PFC planificado para realizarse en cinco meses, se ha convertido en un proyecto de una duración de ocho meses, cuyo resultado es la aproximación al desarrollo de una pizarra virtual usando un dispositivo Kinect, el desarrollo de un programa de RA orientado a ejercicios de rehabilitación, y la mejora de un programa de *tracking* basado en enfoque probabilístico. Las etapas que realmente se han llevado a cabo se pueden observar en el diagrama de Gantt mostrado en la página 10, algunas de ellas como la documentación se han compactado para dar cabida a otras tareas que podría ser más interesante describir como es el caso de la etapa nombrada como A2.1.10., el *tracking* por color. Cabe mencionar que una mayor longitud no implica exactamente una mayor dedicación o profundización en un determinado área, ya que lo que representa el diagrama es el inicio y fin de cada una de las tareas realizadas, en ocasiones algún trabajo se ha prolongado en el tiempo debido a la simultaneidad con otras etapas, otras veces se debe a que

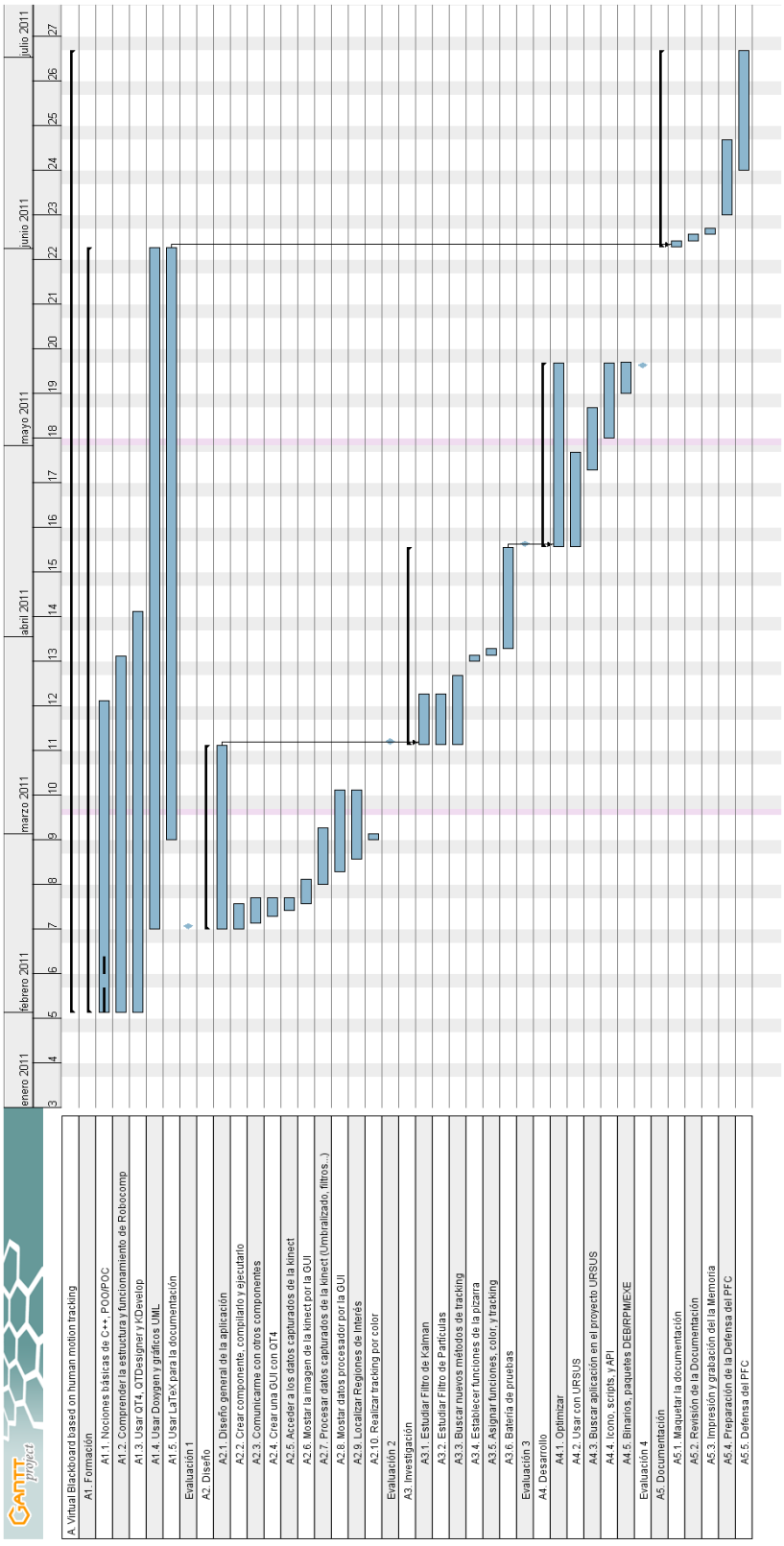

Figura 2.1.: Diagrama de Gantt con la planificación inicial del PFC.

debido a problemas no se avanza hasta que los mismos se solucionan y otras porque efectivamente tienen más complejidad y necesitan tiempo.

El nombre que se le ha dado a la pizarra virtual es Camimic, que surge de unir las palabras *cam* (el apócope en inglés de cámara) y *mimic* (mímica en inglés), en referencia a que el usuario interactúa con el programa haciendo gestos delante de una cámara. La aplicación para ejercicios de rehabilitación se ha nombrado como ARMole, que hace referencia a la RA por sus siglas en inglés y a *mole* (topo en inglés) por el juego *whack-a-mole* que llevamos a la realidad aumentada y enmascara los ejercicios de rehabilitación. Por tanto, esta memoria se referirá con frecuencia a estas aplicaciones por sus nombres Camimic y ARMole a partir de las siguientes líneas.

2.2 Enfoque de cada aplicación

2.2.1 Camimic

El sentido inicial de este programa no es otro que ser ejemplo de aplicación final de técnicas de la imagen como el *tracking* de objetos; y que la misma pueda ser usada por cualquier usuario, sin conocimientos especiales en programación. Comparando los gráficos 2.1 y 2.2, podemos observar como se subestimó la dificultad de crear la parte del *tracking* de objetos, en comparación con otras tareas como crear un componente en RoboComp o la interacción con la *Graphic User Interface* (GUI). Esto se debe al desconocimiento previo del autor de esta memoria sobre los problemas habituales en estos sistemas, los cuales se abordan entre los capítulos 3.3 y 4 con más detalle. En cambio, este apartado se centra en el punto de partida, y la idea general de lo que se pretendía hacer.

Las ideas propuestas para Camimic serían las siguientes:

- Debe ser intuitivo, sin necesidad de conocimientos en el área por parte del usuario, y con ejecución en tiempo real (con frecuencia mayor a 12 imágenes por segundo para percibir movimiento por el ojo humano).

- Uso de dedales de colores que faciliten el *tracking* de los dedos a modo de puntero, para permitir la interacción, de forma que se pueda prescindir del uso del teclado o el ratón.

- Los requisitos de *hardware* para ejecutar el programa serían de bajo coste, descartando dispositivos como láser o cámaras estéreo. Una simple cámara USB cumple perfectamente estas condiciones, pero el uso del dispositivo Kinect, siendo un producto con un precio a la baja, abre las posibilidades al *tracking* en tres dimensiones y a usar un filtro por distancia que facilita la detección del objeto coloreado.

En la Figura 2.3 podemos observar el diagrama de flujo de Camimic. La aplicación entra en un bucle en el que se realiza todo el proceso de captura, *tracking*, interacción y dibujo hasta que el usuario pulse una determinada tecla para cerrar el programa. La forma en la que se pretende llevar a cabo las distintas actividades del diagrama se explican a continuación:

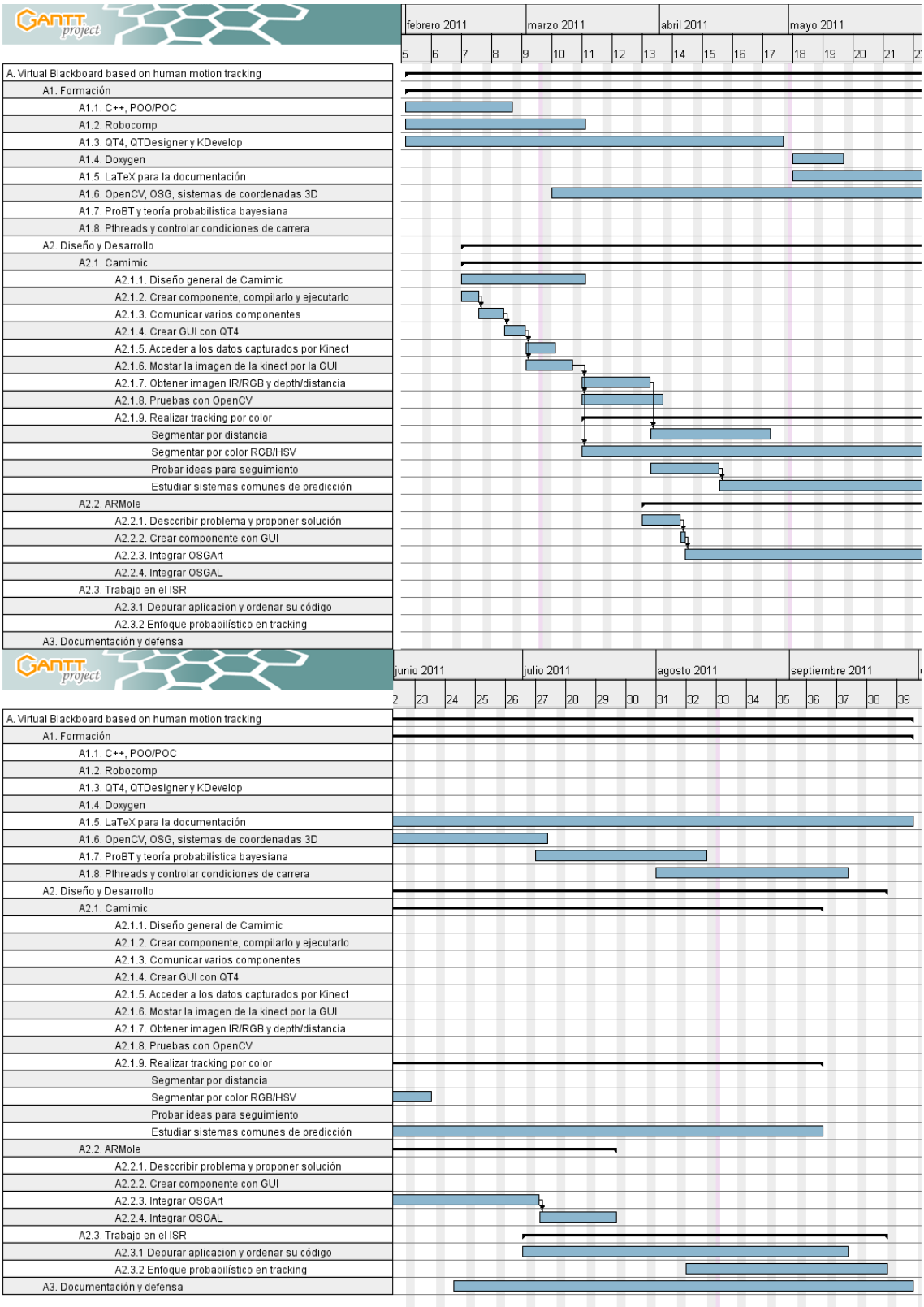

Figura 2.2.: Diagrama de Gantt que muestra el desarrollo real del PFC.

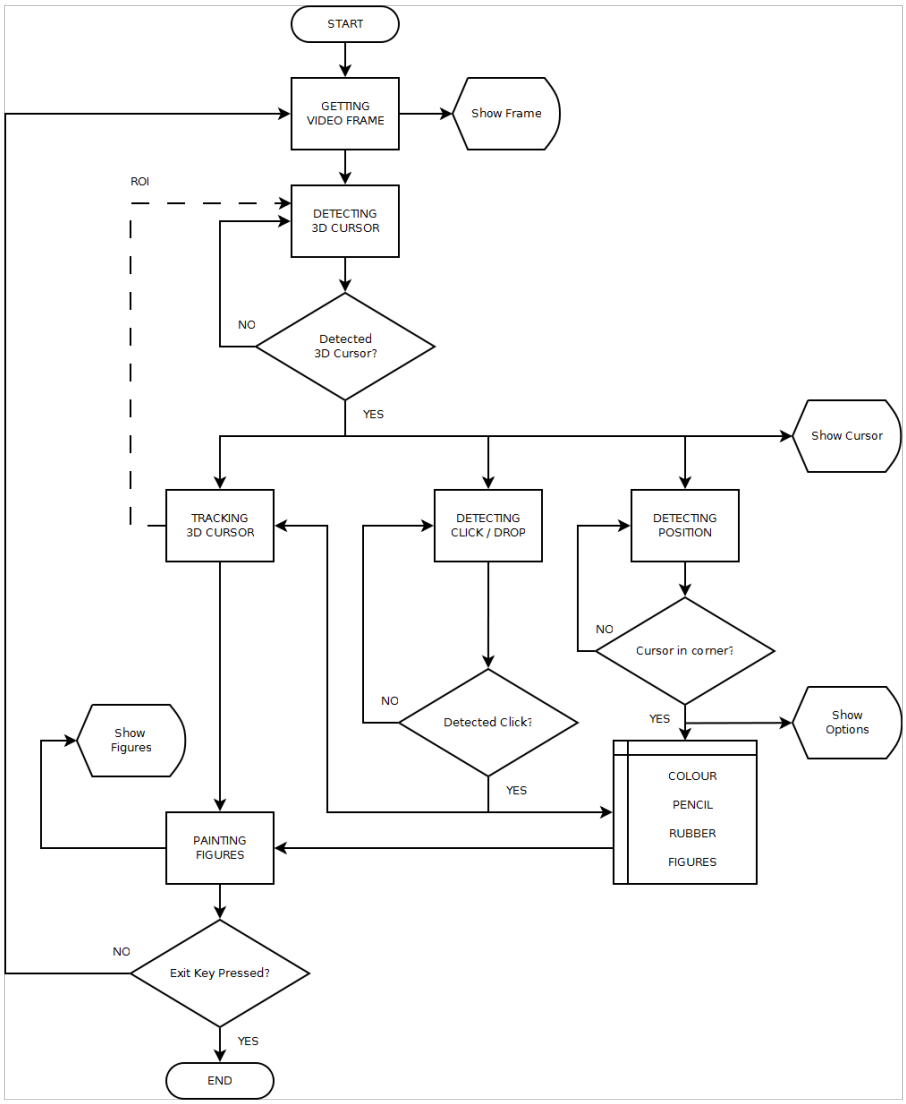

Figura 2.3.: Diagrama de flujo pensado para Camimic.

- Captura de vídeo: Con la idea de aprovechar las ventajas que aporta RoboComp, la aplicación se conectará a KinectComp, de forma que este componente se encarga de tomar las imágenes del sensor Kinect.
- Detección del cursor 3-D: La intención es aplicar un filtro para aislar el color a seguir, y otro filtro por distancia que tendría un umbral fijo para mientras no se detectara el cursor. El cursor es detectado cuando el objeto pasa de manera positiva ambos filtros.
- Detección de posición: Las coordenadas del centro de masas del objeto que pase los filtros del detector sería la posición del cursor 3-D.
- Detección de pulsar (*click*) y soltar (*drop*): Se plantea esta detección más compleja con el uso de dos dedales de colores inversos, de esta forma se puede crear otra imagen con los colores inversos de cada *frame* y hacer pasar dicha imagen por los mismos filtros que a la original. Posteriormente se detecta la situación *click* o *drop* del cursor en base a un umbral en la distancia entre la posición de cada objeto detectado.
- *Tracking*: En principio no hay una opción favorita, y la idea es probar con alguna de las opciones que se describen a partir de la página 27 en esta memoria.
- Selección de opciones: La propuesta para realizar la selección de opciones es usar las esquinas de la imagen como botones que van cambiando su función en base a niveles de navegación.
- Dibujo de figuras: La solución para llevar a cabo esta actividad como se espera, es haciendo uso de la detección de *click* o *drop* del cursor, coloreando los píxeles de la imagen con las mismas coordenadas que la posición del cursor en estado *click*, o no dibujar mientras éste se encuentre en *drop*.

Para tareas como crear de los filtros de color, pintar píxeles concretos, o invertir los colores de la imagen se puede hacer uso de las librerías OpenCV, de las que se da una visión general en la página 21, y además se explica como instalar este *software*. Para la GUI se usarán las librerías Biblioteca de *software* QT (QT) y el programa QTDesigner (QTDesigner), aunque antes de aprender el uso de estas herramientas se realiza un primer boceto que se puede apreciar en la Figura 2.4.

2.2.2 ARMole

La idea general que se pretende llevar a cabo con este programa es crear un juego de realidad aumentada para pacientes que tengan la necesidad de realizar ejercicios de rehabilitación.

■ Definición del problema

El problema al que se intenta dar solución es el de reducir el número de abandonos de los ejercicios de rehabilitación por parte de los pacientes, ya que normalmente los siguientes motivos conducen a ello:

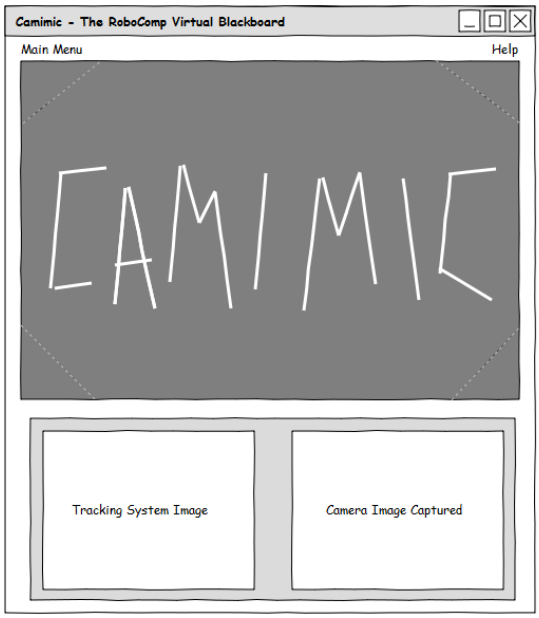

Figura 2.4.: Borrador de la interfaz de Camimic.

- Los progresos en los resultados de los ejercicios en ocasiones suelen ser lentos y la necesidad de seguir con los ejercicios se prolonga en el tiempo en esos casos.
- Cada sesión también puede llegar a necesitar mucho tiempo y además ser repetitiva, lo que consigue que al paciente le resulte aburrido.
- A veces ciertos ejercicios causan molestias e incluso dolor a los pacientes.

El único de los puntos anteriores que podría mejorar un programa informático sería el hacer los ejercicios más divertidos, por ejemplo convirtiendo la obligación de realizar dichos ejercicios en un videojuego, especialmente en niños. Otros trabajos que tratan un problema similar se mencionan en las páginas 5 y 5. Otro de los problemas, abordado de manera general en RoboLab, es que cuando el paciente lleva un cierto periodo realizando los ejercicios, el terapeuta podría dedicar su tiempo a otros pacientes que precisen mayor atención, sin embargo los primeros necesitan un control para asegurarse de que se hacen los ejercicios correctos aunque éstos sean repetitivos. La solución ideal que se trabaja en la EPCC es que gracias a la robótica, los pacientes puedan hacer dichos ejercicios bajo el control de un robot en casa del paciente con el cual el médico pueda comunicarse de forma remota, y corregir los fallos en los ejercicios de rehabilitación. A largo plazo ARMole sería uno de los programas ejecutados por dicho robot, por lo tanto es necesario que nuestro programa no sea rígido y se pueda adaptar para corregir los ejercicios que fueran necesarios; además de estar preparado para usarse por alguien sin conocimientos especiales en informática o programación, de forma que debería tener una interfaz gráfica y ser lo más intuitivo posible. Estas ideas se plantean inicialmente para dos ejercicios de rehabilitación, hombro y codo, de los que podemos ver la respectiva fotografía en las figuras 2.5 y 2.6, tomadas de un vídeo en el que se practican

Figura 2.5.: Ejercicio de rehabilitación de hombro.

ambos ejercicios. Se ha coloreado una aproximación de la trayectoria descrita para cada ejercicio de forma que el lector pueda intuir los movimientos. Sintetizando lo anteriormente expuesto, debemos encontrar un videojuego de manera que los jugadores practiquen los ejercicios de rehabilitación de codo u hombro de las forma más natural y divertida posible, y desarrollar dicho juego con una GUI que permita una posible configuración por parte del médico, y todo ello integrado en RoboComp para que en un futuro se pueda ejecutar en el robot URSUS (URSUS).

■ Solución propuesta

La idea propuesta es desarrollar un juego al estilo del popular *Whack-a-mole* con características de realidad aumentada y una GUI que permita una configuración dinámica de la aplicación. En la Figura 2.7 podemos ver una versión comercial del juego *Whack-a-mole* en la realidad. El juego consiste en golpear con el martillo en la cabeza de los topos (*moles* en inglés) cuando ésta se ilumine y de esta forma obtener puntos. Las cabezas de los topos se van iluminando aleatoriamente y diferentes combinaciones van complicándose según se avanzan los niveles. Se elige este juego como referencia para llevarlo a la RA debido a que la manera natural de jugar es moviendo el brazo, sin embargo habría que tener ciertas consideraciones a la hora de aplicarlo directamente para su uso en pacientes que practiquen ejercicios de rehabilitación. Dichas consideraciones serían las siguientes:

- La forma natural del juego de golpear a los topos incita a hacer movimientos

Figura 2.6.: Ejercicio de rehabilitación de codo.

Figura 2.7.: Juego *Whack-a-mole* que consiste en golpear a los topos.

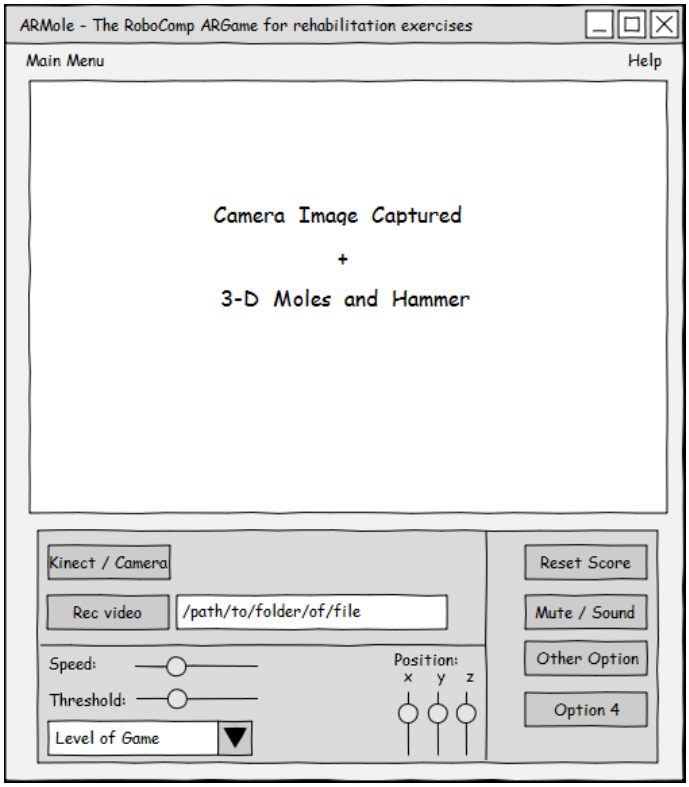

Figura 2.8.: Borrador de la interfaz de ARMole.

bruscos y rápidos, lo cual puede ser muy peligroso dependiendo del problema físico de cada paciente, sería necesario *aprender a jugar*.

- Normalmente en los juegos se consiguen progresos rápidamente y se pasa a niveles más complicados, es necesario escoger el valor crítico por el cual estos progresos no sean elevados para evitar que el ejercicio se haga mal, pero tampoco demasiado bajos para no desmotivar al jugador.

- En el juego real, se ejercita hombro, codo y muñeca en el mismo movimiento, pero en el juego para rehabilitación se debe evitar esto para que podamos controlar que se hace el ejercicio adecuadamente, por lo tanto se tiene que modificar levemente la forma de jugar, ya que aunque sea un juego el paciente no debe olvidar que tiene que cumplir los ejercicios para mejorar. Algunas ideas sobre cómo hacer esto se expone en la correspondiente sección del capítulo donde se explica el desarrollo.

En la Figura 2.8 tenemos un diseño de la interfaz gráfica ideada para ARMole, donde el médico podría configurar la escena con la posición de los topos, la velocidad, seleccionar el nivel del juego, etc.

2.3 Entorno de desarrollo

En esta sección se dan detalles del software y hardware utilizado en la realización del proyecto. Esto tiene importancia principalmente si se quiere realizar algún trabajo o proyecto similar al que se trata en esta memoria, pero también nos ayudará a entender el texto del Capítulo 3 cuando se hable de cómo se han usado estas herramientas. Este PFC hace uso de una cantidad considerable de librerías de *software* de libre distribución, en cada caso la versión de la misma debería ser igual o superior a las que se indican para que el programa creado se ejecute correctamente. En las próximas subsecciones se explica la instalación de las mismas, ya que en algunos casos, se necesita alguna configuración inicial. También es importante tener en cuenta el sistema operativo usado, que en este caso ha sido la distribución basada en GNU/Linux, Ubuntu.

2.3.1 Ubuntu

El sistema operativo utilizado para el desarrollo del proyecto ha sido principalmente Ubuntu (Ubuntu) en su versión 10.10 Maverick Meerkat de 64 bits, que es la que en el momento de comenzar este PFC se encontraba en los equipos de RoboLab, siendo ésta la versión utilizada para el desarrollo de RoboComp. Sin embargo, debido a que el proyecto se ha llevado a cabo desde distintos equipos, en ocasiones se ha utilizado Ubuntu 10.04 en sus versiones de 32 y 64 bits. En principio no debería ser muy complicado una portabilidad a otros sistemas como Windows o MacOS X debido a que las dependencias del proyecto tienen sus respectivas versiones para estos sistemas operativos.

2.3.2 SVN, Doxygen, CMake y BASH

Estas herramientas se consideran básicas en el desarrollo de aplicaciones en entornos GNU/Linux, sin embargo merecen una mención para que en el caso de que el lector de esta memoria no se encuentre familiarizado con dicho entorno, no se pierda cuando se habla de ellas.

- Subversion, también conocido por svn, es un sistema de control de versiones popular que además sirve como método para compartir código fuente y hacer copias de seguridad automáticas, siempre y cuando se disponga de un servidor configurado adecuadamente. En el caso de este proyecto se ha usado el servidor gestionado por RoboLab, que además está preparado para ejecutar *scripts* semanalmente que generan automáticamente toda la documentación de cada componente a través de Doxygen (Doxygen).

- Doxygen es un sistema de generación automática de documentación que genera archivos HTML a través del código doxygen existente en los comentarios del programa a documentar.

- CMake es un sistema de generación automática de archivos Makefile, los cuales almacenan la información necesaria que se tiene que pasar al compilador. Nor-

malmente esa información incluye las rutas de las librerías, cabeceras, ejecutables, etc. que necesitamos para compilar nuestro programa. La tarea de escribir desde cero los archivos Makefile suele ser una tarea tediosa y confusa, ya que suele ser común la necesidad de modificar estos archivos si alguna de las librerías que usamos cambia de versión. CMake nos facilita esta tarea con *scripts* de búsqueda automática de librerías, sin embargo, cuando un programa crece, aún con la ayuda de CMake esta tarea puede resultar tediosa.

- BASH es la línea de comandos por defecto en Ubuntu. Es en este terminal donde usamos aplicaciones como CMake o SVN a través de comandos.

2.3.3 QT4

QT es un conjunto de librerías multiplataforma para crear aplicaciones multipropósito y que cubre una amplia gama de funciones como interfaces gráficas, gestión de redes, soporte de Extensible Markup Language (XML), implementación de gráficos 3-D a través de Open Graphics Library (OpenGL), etc. También nos permite la edición de imagen, aunque desde luego no con tanta potencia como lo hace *Open Source Computer Vision Library* (OpenCV). La ventaja de QT es que te proporciona una capa de alto nivel para programar en C++, abarcando diversos ámbitos donde todas las clases y la forma de usarlas siguen una organización bien estructurada y documentada.

Este PFC aprovecha la facilidad de crear interfaces de usuario de una forma gráfica con QTDesigner, y del método de conectar la GUI de nuestro programa mediante *signals* y *slots*.

2.3.4 KDevelop

KDevelop es un Integrated Development Environment (IDE) popular entre los desarrolladores de *software* que trabajan en sistemas GNU/Linux. En este caso se ha utilizado la versión 4.0.2, que es la última versión estable publicada al comienzo del desarrollo de este proyecto. KDevelop ofrece diversas herramientas para el desarrollo de *software*, como un editor de texto con autocompletado, acceso a la referencia de bibliotecas como QT, entorno de compilación integrado con CMake y GCC, herramientas para hacer *debug*, etc.

KDevelop 4.0.2 no se encontraba disponible en el gestor de *software* de Ubuntu, por lo que ha sido necesario descargarlo e instalarlo por el método tradicional. El motivo de usar esta versión más reciente se debe al soporte para *plugins* (extensiones) y comodidades como los *workspaces*, una función muy útil cuando se trabaja con varios componentes de RoboComp simultáneamente. Se ha compilado con éxito un *plugin* de prueba para KDevelop con la intención de modificarlo con las necesidades específicas de los desarrollos para RoboComp, los pasos a seguir se explican a continuación.

Instalación de plugins en KDevelop

Existen 2 repositorios principales donde buscar *plugins*, Extragear y Playground, siendo el primero más estable y probado, mientras que el segundo más novedoso e inestable. Dichos repositorios se encuentran en las siguientes direcciones:

- https://projects.kde.org/projects/extragear/
- https://projects.kde.org/projects/playground/

El primer paso es descargar el *plugin* deseado (usaremos textitfooplugin a modo de ejemplo) con el comando:

```
git clone git://anongit.kde.org/fooplugin
```

Antes de continuar tenemos que tener satisfechas las dependencias específicas del plugin, normalmente descritas en un archivo del tipo README dentro del directorio del *plugin*. Una vez satisfechas, debemos seguir los pasos configurar, compilar e instalar, que se logran con los siguientes comandos respectivamente:

```
cmake .
make
sudo make install
```

2.3.5 RoboComp svn-version

RoboComp es un *framework* libre desarrollado en el RoboLab de la UNEX orientado a componentes con la finalidad de ejecutarlos en robots. Herramientas similares utilizadas en robótica serían Orca2, Carmen o ROS. Algunas características que podemos destacar de esta RoboComp son las siguientes:

- La abstracción de *hardware*, que es posible gracias a componentes que se comunican directamente con los controladores de dispositivos como pueden ser el láser, la cámara, bases robóticas, etc.
- Soporta el desarrollo de aplicaciones con GUI.
- Permite la generación automática de código para nuevos componentes.
- Es posible que distintos componentes se comuniquen entre sí a través de *proxies*, incluso si estos se ejecutan en distintos equipos o están escritos con distintos lenguajes de programación. Además varios componentes se pueden conectar de forma simultánea a un mismo componente por lo que es posible que un dispositivo como una cámara sea utilizada a la vez por varias aplicaciones.
- Incluye herramientas para la gestión de los componentes en ejecución que nos permite pausar o arrancarlos de forma gráfica.
- Un sistema para reproducir el comportamiento de componentes ejecutados con anterioridad, nos permite hacer pruebas incluso cuando ya no disponemos del dispositivo usado.

En la Figura 2.9 podemos observar la jerarquía de directorios de RoboComp, así como un gráfico representativo de un componente en el que podemos ver los puntos explicados anteriormente.

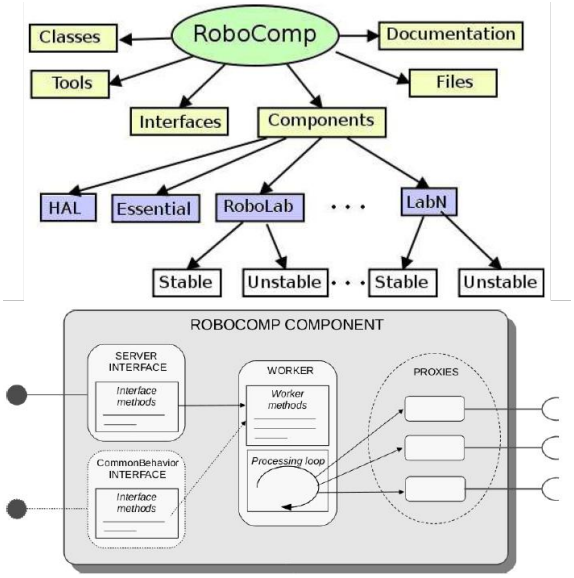

Figura 2.9.: Estructura de directorios y gráfico representativo de un componente de RoboComp.

■ Instalación de RoboComp

Dependiendo del componente que se vaya a usar de los que forman parte de RoboComp o de la función de un nuevo componente que queramos crear, necesitaremos software adicional, como es nuestro caso con las librerías que se explican en las siguientes subsecciones. En en desarrollo del proyecto también ha sido necesario usar la potencia que brinda *Intel® Integrated Performance Primitives Library* (IPP), pero que son usados por componentes ya creados en RoboComp y en realidad no se ha programado nada con estas librerías de Intel, por lo que no se le dedica una subsección. Por tanto, partiendo que de el sistema operativo tiene instaladas todas las dependencias particulares que vamos a usar (como IPP, GAZEBO, PROSILICA, etc.), simplemente tendríamos que descargar un *script* de la siguiente dirección:

- http://robocomp.svn.sourceforge.net/viewvc/robocomp/Tools/robocompInstaller.py

El *script* se puede ejecutar ejecutando el siguiente comando bash desde la carpeta donde lo hayamos guardado:

```
python ./robocompInstaller.py
```

Al ejecutarlo, arranca una interfaz gráfica que instala las dependencias comunes, descarga el software de RoboComp en la carpeta indicada por el usuario, y establece las variables de entorno del sistema, tras lo que tendremos que reiniciar el sistema. A pesar de que pueda parecer que ya está todo instalado, no tenemos que olvidar que es necesario compilar las clases, herramientas, o componentes que vayamos a utilizar.

Para compilar las clases ejecutamos los siguientes comandos desde el terminal:

```
cd $ROBOCOMP/Classes/
```

```
cmake .
make
```

Para compilar las herramientas, serían los siguientes:

```
cd $ROBOCOMP/Tools/
cmake .
make
```

Y el procedimiento para compilar los componentes sería similar, sustituyendo fooComp por el nombre del componente que necesitemos:

```
cd $ROBOCOMP/Components/Robolab/Experimental/fooComp
cmake .
make
```

Si todos los pasos se han seguido sin problema, RoboComp estaría listo para usarse.

2.3.6 OpenCV

OpenCV es una biblioteca de software desarrollada en 1999 por la compañía *Intel*® Corporation para el procesado de imágenes y el desarrollo de la visión por computador. Fue publicada bajo licencia *Berkeley Software Distribution* (BSD) y es multiplataforma, de forma que se puede utilizan en aplicaciones para GNU/Linux, MacOS X o Windows.
En la actualidad OpenCV es desarrollado siguiendo el modelo del software libre, pero está coordinado por el Willow Garage, un grupo de desarrolladores y expertos en robótica. La versión utilizada en este PFC es la 2.2 publicada en Diciembre de 2010, la cual es completamente compatible con la serie 2.x y parcialmente con la 1.x, pudiendo utilizar distintas estructuras para el tratamiento de imágenes.
OpenCV es ampliamente utilizado en campos como la Realidad Aumentada y la Visión por Computador con aplicaciones en la medicina, la seguridad, la robótica y la industria. Su API permite desarrollar programas en varios lenguajes de programación como C, C++, Python o *NVidia*® CUDA (rutinas para computación en la *Graphic Processing Unit* (GPU).
Dispone de más de 500 funciones que se pueden clasificar en los siguientes grupos:

- Procesamiento General de Imágenes
- Segmentación
- Transformación
- Máquina de aprendizaje: Detección y Reconocimiento
- Descriptores geométricos
- Calibración de Cámaras Estéreo o Cámaras 3D
- Árboles de imagen y estructuras de datos
- *Tracking*

■ Instalación de OpenCV 2.2

Para instalar correctamente OpenCV-2.2.0 en Ubuntu se deben seguir las siguientes instrucciones:

- Descargar el paquete OpenCV-2.2.0.tar.bz2 de la siguiente dirección:
- Descomprimir el archivo descargado en la carpeta deseada.
- Abrir una línea de comando BASH en dicha carpeta.
- Ejecutar los siguientes comandos:

```
mkdir release
cd release
sudo apt-get install build-essential libavformat-dev libswscale-dev
&& libpng12-dev libgtk2.0-dev python-numpy libavcodec52
&& libgstreamer0.10-dev libgtk2.0-dev libgstreamer0.10-dev
&& libgstreamermm-0.10-dev libgstreamer-plugins-base0.10-dev
&& libunicap2-dev libunicapgtk2-dev  pkg-config libjpeg8-dev
&& libavcodec-dev libavc1394-dev libdc1394-22-dev
&& zlib1g-dev libjasper-dev python gtk2-engines libv4l-dev
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
sudo ldconfig -v
```

- Se debe tener en cuenta que este método instala OpenCV con las opciones típicas, para activar las opciones como el uso de Threading Building Blocks (TBB), IPP o interfaces avanzadas con QT, primero se debe instalar el *software* correspondiente y después modificar el archivo CMake (CMake) antes de compilar.

2.3.7 Kinect, OpenKinect y OpenNI

Kinect es un sensor desarrollado por la empresa *PrimeSense*® (Prime Sense) con el objetivo inicial de convertirse en un complemento para la videoconsola XBOX® 360 de Microsoft®, de forma que dicho complemento actúe de interfaz *sin manos* y pueda competir contra el WiiMote de la consola Wii de Nintendo. Kinect se concibe desde un principio como un dispositivo de comunicación hombre-máquina, similar al ratón, teclado, o mandos de control remoto, pero buscando una interacción más natural. De forma resumida se podría decir que Kinect es un sensor *todo en uno* que incluye una cámara de vídeo, una cámara de profundidad, micrófonos y altavoces, un acelerómetro y un pequeño motor que le permite hacer movimientos de tipo inclinación (*tilt*). Una fotografía de este aparato se muestra en la Figura 2.10

En Noviembre de 2010 se publica un controlador (*driver*) libre para GNU/Linux con el nombre FreeNect que permite obtener los datos de imagen y profundidad del

Figura 2.10.: Fotografía del dispositivo Kinect.

dispositivo Kinect. En ese momento distintos grupos de investigación comienzan a trabajar con esta nueva cámara 3D, y se forma un grupo para coordinar el desarrollo del driver llamado OpenKinect. Debido al éxito de este dispositivo entre los desarrolladores de *software*, Prime Sense publica un *framework* llamado *Open Natural Interaction* (OpenNI) con modelos y funciones comunes en la interacción con este tipo de dispositivos. Sin embargo este *framework* únicamente soporta plataformas Windows, además de usar una biblioteca de software conocida como NITE (NITE) la cual no tiene licencia libre. En Marzo de 2011 ASUS® comienza a comercializar un nuevo sensor con el nombre de Xtion®, pero el fabricante sigue siendo Prime Sense con un sistema similar al que usa la Kinect. ASUS se suma al grupo que mantiene OpenNI y Microsoft publica un Software Ddevelopment Kit (SDK) para Windows.

Actualmente se está trabajando para integrar el uso de Kinect en OpenCV, y además de usos para todo tipo de videojuegos, se está desarrollando ampliamente para el uso de visión artificial en robots y en Realidad Aumentada.

■ Instalación del driver OpenKinect

La forma de instalar este driver para cada sistema operativo, está disponible en la siguiente dirección:

- http://openkinect.org/wiki/

Sin embargo tenemos que tener cuidado, porque en la fecha en la que se escribe esta memoria la instalación mediante repositorio (la forma más sencilla) para Ubuntu, puede darnos problemas de estabilidad porque instala una versión anticuada. Por tanto, los pasos a seguir serían los que siguen al título *Ubuntu Manual Install* en la dirección web anteriormente indicada.

2.3.8 ARToolKit

ARToolKit es una librería que permite el *tracking* de marcas usando técnicas de reconocimiento de patrones y transformaciones afines, orientada a la producción de aplicaciones de realidad aumentada. Esta librería se publicó en 1999 y actualmente

Figura 2.11.: Marca Hiro, típicamente usada por ARToolKit.

conviven versiones con licencia libre y comercial. El buen funcionamiento y su licencia comercial ha popularizado su uso en los últimos años y es posible ver juegos para videoconsolas y cada vez más aplicaciones para *smartphones* que hacen uso de esta tecnología. Un ejemplo de marca usada por ARToolKit se puede ver en la Figura 2.11 El código, ejemplos de aplicaciones, tutoriales, indicaciones de instalación y más información está disponible en la siguiente dirección:

- http://www.hitl.washington.edu/artoolkit/

■ Instalación de ARToolKit

A pesar que la instalación de ARToolKit está bien documentada en el enlace anterior, hay que destacar la elección de algunas opciones en el proceso, ya que ARToolKit permite el uso de múltiples dispositivos para capturar la imagen, incluyendo algunos poco comunes como la cámara Eye Toy para la videoconsola PlayStation de Sony. Las opciones serían las siguientes:

- *Select a video capture driver:*

  Elegimos la opción *5: GStreamer Media Framework.*

- *Do you want to create debug symbols? (y or n)*

  Elegimos *y* para poder depurar nuestro código en caso de errores.

- *Build gsub libraries with texture rectangle support? (y or n)*

  Elegimos *y* porque aprovecha las ventajas de las tarjetas gráficas modernas.

2.3.9 OpenSceneGraph

OpenSceneGraph (OSG) es una capa de alto nivel para construir mundos virtuales en tres dimensiones sobre OpenGL. OSG nos permite crear árboles de nodos para crear objetos complejos como puede ser el esqueleto de una persona donde la posición

de un nodo como podría ser la muñeca depende de la posición de un nodo padre como sería el codo, de forma que reproducir los movimientos de un modelo de brazo será más sencillo en OSG que implementarlo sobre OpenGL directamente.

■ Instalación de OpenSceneGraph

La instalación de OSG es tan sencilla como ejecutar el *script* escrito en python que podemos encontrar en la carpeta *ThirdParty* dentro de RoboComp. En caso de problemas por la versión del sistema operativo sólo tendríamos que modificar dicho *script*.

2.3.10 OSGArt

OpenSceneGraph-ARToolKit integration (OSGArt) es un conjunto de *software* que sirve de enlace entre OSG y ARToolKit, específicamente nos permite controlar los nodos de OSG con las traslaciones y rotaciones que ARToolKit extrae de las marcas durante el *tracking*. Además es posible usar la imagen capturada por la cámara como fondo de la escena 3D, de forma que el resultado es dibujar objetos en tras dimensiones respecto a la marca detectada por ARToolKit.

■ Instalación de OSGArt

La instalación de OSGArt puede ser un poco más complicada de los esperado, los pasos que se ponen aquí han sido extraídos de un tutorial elaborado por Luis Calderita para RoboLab, que ha recopilado información de varias fuentes en internet:

- Descargar osgART 2.0.RC3 del siguiente enlace: http://osgart.org/files/
- Descomprimir el paquete y abrir terminal en la carpeta descomprimida.
- Introducir los siguientes comandos en dicha terminal:

```
export CFLAGS="$CFLAGS -fPIC -fpermissive"
export CXXFLAGS="$CXXFLAGS -fPIC -fpermissive"
export ARTOOLKIT_2_ROOT="$HOME/ARToolKit"
cmake .
make
sudo make install
```

- El paso anterior crea e instala las librerías dinámicas tracker y video, para comprobarlo ejecutamos:

```
cd /usr/local/lib/osgPlugins-2.8.3
ls osgart_tracker_artoolkit2.so
ls osgart_video_artoolkit2.so
```

Para comprobar que OSGArt está correctamente instalado podemos probar los siguientes pasos, tras los cuales deberíamos ver un cubo 3D encima de la marca HIRO como se muestra en la Figura 2.12:

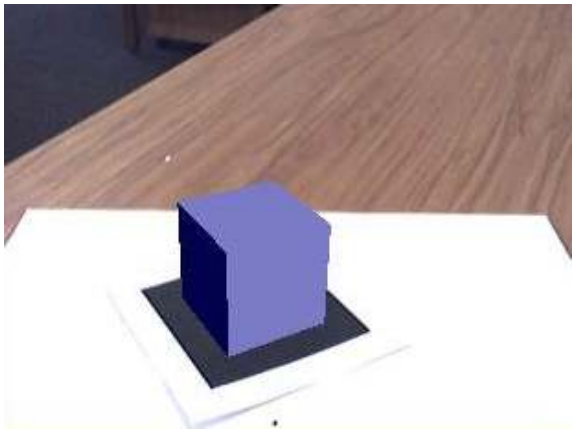

Figura 2.12.: Imagen que obtenemos si OSGArt está correctamente instalado.

```
cp -r $HOME/ARToolKit/bin/Data/ $HOME/osgART_2.0_RC3/bin
mv $HOME/osgART_2.0_RC3/bin/Data $HOME/osgART_2.0_RC3/bin/data
cd $HOME/osgART_2.0_RC3/bin && ./osgartsimple
```

Es posible que a pesar de todo es posible que obtengamos un error con el que la imagen aparece completamente negra a pesar de que arranque la cámara. La solución a este problema aparece en el foro artoolworks.com por el usuario jduran, y sería la siguiente:

```
gedit $HOME/osgART/src/osgART/Video/ARToolKit/ARToolKitVideo.cpp
```

En ese archivo debemos cambiar la condición del *if* (0 por 1) y volver a compilar e instalar las librerías, el fragmento a editar sería el que sigue:

```
--- ARToolKitVideo.cpp.orig   2010-02-09 13:15:27.000000000 +0100
+++ ARToolKitVideo.cpp   2010-02-09 15:32:48.000000000 +0100
@@ -307,7 +307,7 @@
      osg::Timer t;
-      if (0 == ar2VideoCapNext(video))
+      if (ar2VideoCapNext(video))
      {
         if (newImage = (unsigned char*)ar2VideoGetImage(video))
         {
```

Para conseguir un *tracker* fino, que no tiemble tanto, recurrimos a un filtro temporal. Esto se logra de la siguiente editando el siguiente fichero:

```
 /home/... RutaCarpetaOsgART .. /src/osgART/Tracker/ARToolKit/SingleMarker.cpp
```

Modificando la línea 76 por la esta otra:

```
arGetTransMatCont(markerInfo, patt_trans, patt_center, patt_width, patt_trans);
```

Recompilamos OSGArt y reinstalamos la librería.

2.3.11 OSGAL

OSG-OpenAL integration (OSGAL) es una capa de *software* que añade audio tridimensional a los mundos virtuales creados con OSG, mediante el uso de la librería de audio Open Audio Library (OpenAL) y las herramientas OpenAL Utility Toolkit (ALUT), aunque también puede hacer uso de otros subsistemas de sonido como FMOD (FMOD). El sonido tridimensional se consigue añadiendo nodos de sonido a la escena, de forma que si el nodo se aleja o acerca a la pantalla, el sonido final que oímos por los altavoces variará su intensidad. Además también se puede conseguir efecto Doppler, sonido ambiente, etc.

■ Instalación de OSGAL

La instalación es sencilla:

- Instalar la versión de desarrollo de los paquetes OpenAL y ALUT. Estos paquetes se encuentran en el repositorio oficial de Ubuntu.
- Descargar OSGAL de la siguiente dirección: http://www.vrlab.umu.se/research/osgAL/
- Descomprimir el archivo, y editar el archivo CMakeList.txt para asegurarse que está marcada la opción OSGAL/ALUT y desmarcada la correspondiente para usar FMOD, ya que no se pueden usar ambas, y en nuestro caso usaremos OpenAL/ALUT.
- Iniciar una consola en la carpeta descomprimida y ejecutar los siguientes comandos:

```
cmake .
make
sudo make install
```

2.4 Métodos comunes usados para tracking de objetos

Cuando este documento menciona el *tracking*, se refiere de forma general al seguimiento automático de objetos, ya sea en imágenes bidimensionales o tridimensionales. El seguimiento a su vez se suele dividir en dos tareas importantes, la detección y la predicción. Ambas tareas están estrechamente relacionadas ya que la detección inicia y corrige a la predicción en la mayoría de los casos, mientras que la predicción permite afinar y acelerar la detección. Debido a que algunos métodos de predicción son altamente utilizados para aplicaciones de *tracking*, puede resultar confuso cuando son mencionados si se hace referencia a la predicción o a su típico uso en seguimiento de objetos en imagen. En esta sección hablamos de métodos completos de *tracking* pero nombrámdolos por el tipo de predicción que utilizan.

2.4.1 CAMShift

Continuously Adaptive Mean-Shift, también conocido como CAMShift, es un método para el seguimiento de objetos en secuencia de imágenes, que desarrolla a su vez el algoritmo Mean-Shift el cual usa información del color en forma de densidad de probabilidad para cada píxel en la imagen. Una implementación típica de CAMShift para *tracking* consiste en los siguientes pasos:

- Seleccionar una imagen inicial, ya sea capturándola del vídeo o de un fichero, y calcular su histograma.
- Seleccionar una ventana en la imagen, sobre la cual desplazaremos otra del mismo tamaño de la imagen seleccionada anteriormente.
- Se comparan los histogramas de la imagen preseleccionada y el de la ventana pequeña para cada desplazamiento dentro de la ventana grande, cuando se encuentra el desplazamiento en el que los histogramas son más parecidos, se aplica dicho desplazamiento a la ventana de mayor tamaño. El resultado es que esta ventana consigue seguir al objeto y es bastante robusto frente a cambios en la imagen ya que incluso el tamaño de esta ventana puede variar en función de la diferencia entre histogramas.

2.4.2 Filtro de Partículas

El filtro de partículas es uno de los métodos más utilizados en la actualidad para el seguimiento de objetos en visión por computador, el cual consiste en aplicar un método *Secuential Monte Carlo* (SMC), un sofisticado modelo de técnicas de estimación basado en simulación.
Existen variantes del filtro y otras aplicaciones como la estadística o cualquier otro proceso secuencial, sin embargo tienen en común sus cuatro etapas (inicialización, actualización, estimación y predicción) y la aplicación del Teorema de Bayes de la probabilidad condicional. En el PFC "Tracking automático de objetos en secuenciales de imágenes usando Filtro de Partículas" [10] de Eva Chaparro Laso tenemos un ejemplo del filtro de partículas implementado con OpenCV.
El punto fuerte del filtro de partículas es la gestión de una ROI (*Region of Interest*) donde tenemos una mayor probabilidad de localizar el objeto que estemos rastreando.

2.4.3 Filtro de Kalman

Otro de los métodos más utilizados para el *tracking* de objetos en imágenes en movimiento es el filtro de Kalman, que surge de la adaptación del algoritmo de Rudolf E. Kalman. Este método actúa de forma similar a un predictor lineal dónde a partir de una muestra, estimamos la siguiente y posteriormente calculamos el error cometido en dicha predicción, de manera que para el cálculo de la siguiente muestra y teniendo en cuenta dicho error, podamos mejorar la predicción minimizando el error. Este método es particularmente efectivo cuando se dan unas condiciones ideales como la ausencia de cambios bruscos en la muestra o cierta pasividad del entorno.

Figura 2.13.: Tracking de cara con OpenTLD.

Un ejemplo de la aplicación del filtro de Kalman para el seguimiento de objetos en imágenes de vídeo podemos encontrarlo en el PFC "Tracking automático de objetos en secuencias de imágenes usando Filtro de Kalman" [7] de María Isabel Menor Flores.

2.4.4 OpenTLD: Algoritmo Predator

Aunque este proyecto no hace uso del algoritmo Predator, es conveniente citarlo como tecnología de *tracking* de objetos debido al adelanto que representa en este campo debido a su eficacia en comparación con otras tecnologías expuestas anteriormente.
El algoritmo Predator fue publicado bajo licencia libre en Marzo de 2011, y posteriormente se creó el grupo OpenTLD para coordinar los desarrollos con este nuevo algoritmo. La ventaja de este método para el *tracking* de objetos es que es capaz de reescribir el patrón a localizar en el vídeo, las siglas TLD hacen referencia a las palabras seguimiento, aprendizaje y detección (en inglés *tracking*, *learning* y *detecting*). El algoritmo intenta de localizar un objeto patrón en la secuencia de imágenes, si consigue localizarlo, crea un nuevo objeto patrón y en la siguientes imágenes busca alguno de los dos objetos, de forma que va creando una base de datos con las mutaciones del objeto original, algunas de las mutaciones las descarta según algunas restricciones, y cuando tiene una base de datos completa, es cuando alcanza la máxima efectividad.
El motivo por el que no se usa el algoritmo Predator en Camimic es por las limitaciones de plataforma del código publicado, pues usa Matlab y de momento únicamente soporta el sistema operativo de tipo Windows. Sin embargo desde OpenTLD trabajan en una portabilidad a GNU/Linux sustituyendo Matlab por la implementación libre Octave, por este motivo es posible que en el futuro, se pudiera adaptar Camimic para usar dicho método para el *tracking*. La Figura 2.13 muestra una captura del proceso de aprendizaje de OpenTLD.

3. DESARROLLO

3.1 Primeros pasos desarrollando un componente

La mayoría de los pasos que se describen en esta sección se han desarrollado de forma similar tanto para la aplicación Camimic como para ARMole, y serían comunes para futuros componentes de RoboComp con interfaz gráfica que hagan uso de imágenes de vídeo. Es por este motivo que se pretende salir de la dinámica bastante común en las memorias de PFC en las que se nombran una por una todas las funciones implementadas o usadas, además de que por otro lado se ha puesto especial atención en introducir comentarios explicativos en el código para la mayoría de las funciones, y estos comentarios están disponibles en formato web desde el servidor de RoboComp alojado en sourceforge.com.

3.1.1 Generación, compilación y ejecución de un componente

Tras seguir las instrucciones explicadas en la sección Entorno de desarrollo del capítulo anterior, deberíamos tener RoboComp listo para poder ejecutar herramientas y los componentes que necesitemos.

La generación de nuestro componente será tan sencillo como ejecutar el *script* en Python componentGenerator.py que podemos encontrar en la carpeta Tools. Dicho *script* iniciará una GUI en la que se debe elegir un nombre para el componente, otro para su interfaz y establecer los *proxies* que se van a utilizar. Los *proxies* son el sistema con el que nuestro componente se podrá conectar a otros componentes como el que gestiona la cámara de vídeo o el dispositivo Kinect.

Tras elegir los nombres y los *proxies*, RoboComp generará todos los archivos básicos automáticamente en una carpeta con el nombre del nuevo componente. Esta carpeta se encontrará en la siguientes ruta:

```
$ROBOCOMP/Components/Robolab/Experimental
```

Para la compilación de nuestro componente, abriremos un terminal Bourne-again Shell (Bash) para insertar los comandos que siguen a continuación:

```
cmake .
make
```

Con el paso anterior se habría creado el binario en la carpeta bin dentro de la ruta desde donde compilamos en el paso anterior, por lo tanto para ejecutar nuestro componente, tan sólo tendríamos que ejecutar el correspondiente Bash *script* que se encuentra junto al binario, tras comprobar que el mismo tiene permiso de ejecución bastaría con un simple doble clic.

Todos los componentes que se generen por este método usan una instancia de la clase Worker de RoboComp y que usa un *timer* que hace constantemente llamadas a

la función compute() con una frecuencia que podemos controlar de forma dinámica desde el mismo componente.

3.1.2 Desarrollo de la interfaz gráfica de usuario

La forma más sencilla y cómoda de crear una GUI para nuestro componente es diseñándola de forma gráfica con QTDesigner, de forma que podemos aproximarla a nuestro bocetos rápidamente. Una vez creada, debemos conectar las señales de cada *widget* con las funciones internas que desarrollemos en nuestro componente. Las posibles señales dependen del tipo de *widget*, por ejemplo un botón emite señal cuando es pulsado mientras que una barra la emite cuando el usuario la desplaza.

Por tanto, la mencionada conexión necesita principalmente tres datos: el nombre del *widget*, la señal que se quiera controlar y el nombre de la función a la que se llama cuando la señal es activada. Con esos datos ya podemos escribir una línea como la que sigue, y que normalmente se encuentra en el constructor worker():

```
connect( &widget, SIGNAL(event()), this, SLOT(function()) );
```

3.1.3 Comunicación con otros componentes

La comunicación entre componentes de RoboComp es gestionada a través de Internet Communications Engine (Ice), esto permite conexiones entre componentes que estén escritos en distintos lenguajes, distintas máquinas, etc. La forma de comunicarnos es mediante la interfaz que implemente cada componente que podremos usar como una función más en nuestro código, siempre y cuando tengamos correctamente escrito el archivo de configuración para Ice, y por supuesto deberíamos haber establecido los *proxies* en nuestro componente.

Como se explica una líneas más arriba, podemos establecer los *proxies* con el generador de componente, sin embargo RoboComp también dispone de una herramienta para añadir *proxies* de forma automática. Tanto para Camimic como para ARMole esto último no ha sido necesario porque se añadieron para cameraComp y kinectComp desde un principio; también se crearon sus respectivas interfaces pero no se hicieron cambios a los archivos autogenerados.

El archivo de configuración de cada componente se encuentra en la carpeta etc con el nombre de config, las líneas que se suelen modificar se muestran a continuación, aunque puede que otros componentes tengan muchas más opciones como es el caso de cameraComp donde podemos elegir la resolución de la cámara entre otras configuraciones.

```
#RemoteProxy = remote:tcp -h remote_host_ip -p port_number
CameraProxy = camera:tcp -h localhost -p 10001
KinectProxy = kinect:tcp -h localhost -p 10096
```

De las líneas que se ven encima de este texto, la primera de ellas es un modelo del formato seguido para configurar los *proxies*, mientras que las dos restantes son las que usa tanto Camimic como ARMole para conectarse a kinectComp y cameraComp cuando se ejecutan en el mismo computador.

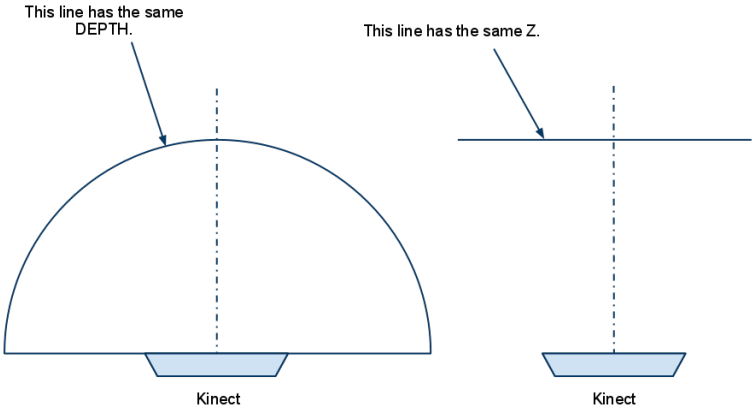

Figura 3.1.: Esquema que muestra las dos clases de distancias.

3.2 Gestión y Calibración del sensor Kinect

Antes de comenzar a usar este novedoso y popular dispositivo, se ha intentado comprender su funcionamiento con la suficiente profundidad para conocer el porqué de los típicos errores en las medidas del dispositivo y sus posibilidades, si bien no se han llegado a hacer experimentos formales que hubieran permitido una caracterización del mismo.

3.2.1 Consideraciones sobre Kinect

Antes de poder comenzar a usar la información de profundidad proporcionada por Kinect, es necesario tener en cuenta ciertos aspectos de su funcionamiento. Merece especial atención el sistema de coordenadas que queramos usar con esta información, ya que cuando hablamos de medidas de distancia o profundidad tenemos que diferenciar entre profundidad en coordenadas cartesianas o esféricas, dependiendo de nuestro caso podemos necesitar la distancia de un punto de la escena hasta Kinect o la distancia entre planos verticales. Esta idea se puede apreciar de forma sencilla en la Figura 3.1.

Además kinectComp tiene dos funciones distintas con las que nos envía las imágenes, éstas pueden considerar el centro la cámara IR o la RGB en dependiendo de la llamada a una u otra función, como podemos observar en la Figura 3.2.

También sería importante tener en cuenta la aparición de sombras en la imagen de profundidad, estas sombras pueden ser de dos tipos: sombra lateral y sombra de objetos.

La sombra lateral es la que menos molesta porque suele caer en el límite de la imagen de color y además es mínima ya que se reduce rápidamente con la distancia, esta sombra se debe al desplazamiento que hay entre la cámara de IR y la cámara RGB, además de que ambas tienen distintos ángulos de su campo de visión. En la Figura 3.3 vemos un dibujo de esta explicación.

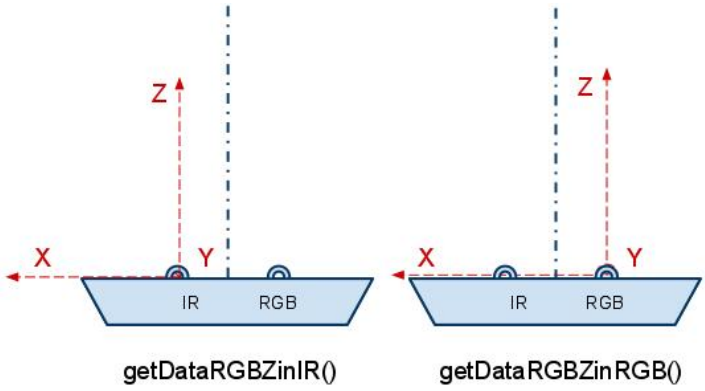

Figura 3.2.: Los métodos de obtener datos desde KinectComp.

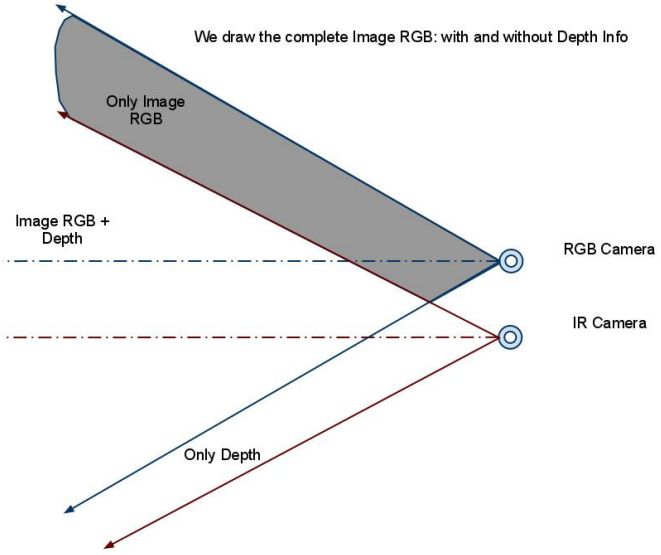

Figura 3.3.: Sombra lateral en Kinect.

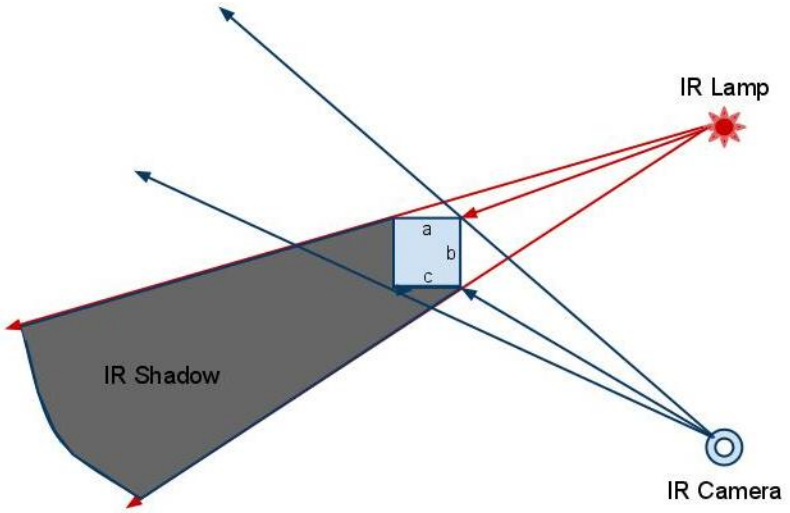

Figura 3.4.: Sombra de objetos en Kinect.

La sombra de objetos puede ser más importante y a menudo resulta molesta para algunas tareas, por ejemplo el filtro de distancia descarta la zona de sombra infrarroja. Como se muestra en la Figura 3.4 se debe por la distancia entre la cámara y la lámpara de infrarrojos. Si bien este efecto también se reduce según nos alejamos, sigue siendo perceptible en la distancia típica a la que solemos interactuar incluso con una buena calibración.

3.2.2 Calibración del sensor Kinect

Durante este proyecto se han realizado calibraciones para los distintos dispositivos Kinect del RoboLab usando el programa RGBDemo de Nicolas Burrus, que no es más que una adaptación del sistema de auto-calibración disponible como herramienta en el paquete OpenCV. En la Imagen 3.5 podemos ver parte del proceso de calibración de Kinect, y en la 3.6 se aprecia como la información de profundidad encaja mejor con la imagen de color.

3.2.3 Modificación de KinectComp para obtención de imagen infrarrojos

A pesar de los errores de medida que se mencionan al inicio de de esta sección, con frecuencia seguirán apareciendo áreas en la imagen sin información de profundidad. Ver la imagen de infrarrojos ayuda de cara a entender el motivo de esos agujeros de información, como se puede ver en la Figura 3.6 estas lagunas coinciden cuando un material absorbe la luz Infrarrojo (IR) o cuando se produce saturación, ya sea brillos la imagen o a que se capta otra fuente de luz infrarroja como es el caso de la luz solar.

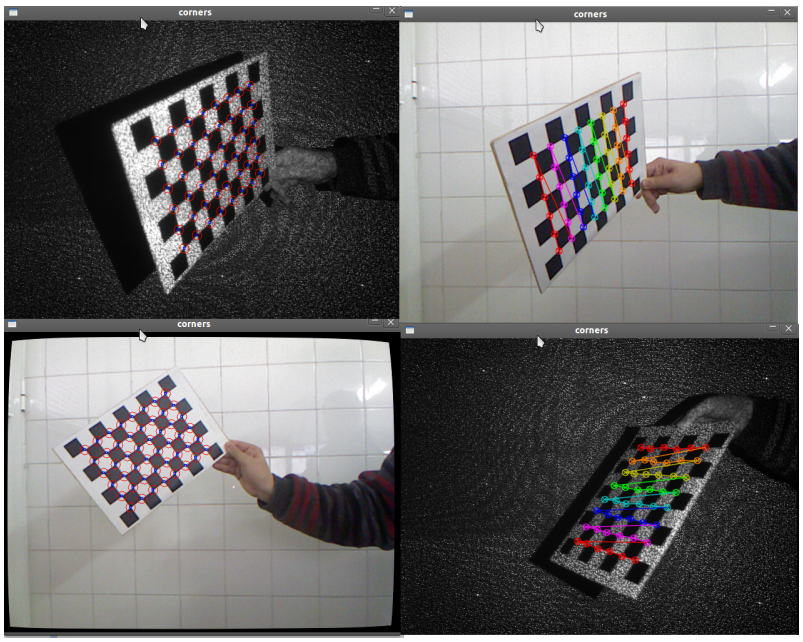

Figura 3.5.: Capturas de imagen mediante la calibración de kinect.

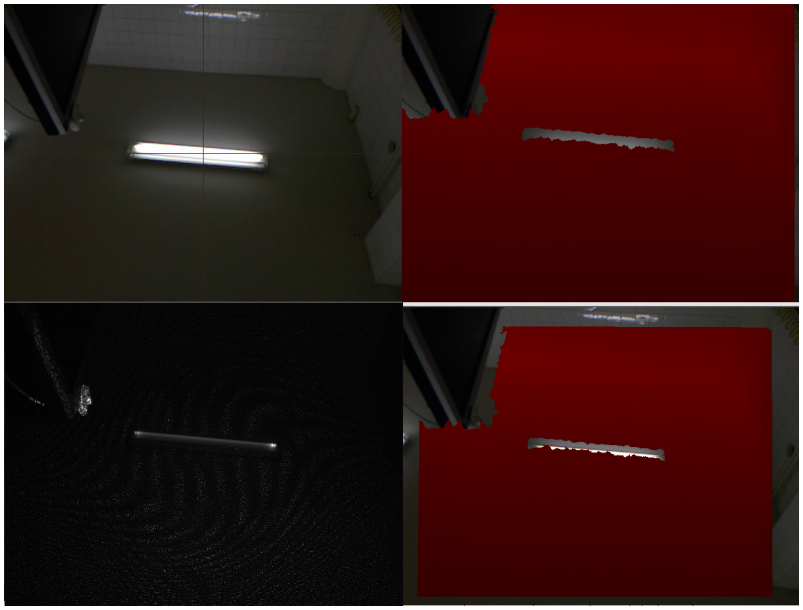

Figura 3.6.: Efecto de la calibración de kinect.

3.2.4 Acotación del espacio en la imagen

El motivo por el que se hace uso de un dispositivo Kinect es precisamente poder filtrar la imagen en base a la distancia a la que se encuentran esos objetos de la cámara. Este proyecto ha desarrollado un filtro de imagen en base a la distancia, de forma que para una distancia y con un margen determinados, se descarten los píxeles de la imagen que no cumplan los parámetros seleccionados. Este filtro está integrado en Camimic y es posible probar su funcionamiento desde la pestaña de configuración, donde cada píxel es coloreado en una escala de color que va de negro a rojo en base a la información de distancia; con las barras de desplazamiento podemos seleccionar la distancia y el margen que deseemos y que pintará de color azul todos los píxeles que pasen el filtro. En los huecos de información se ha dejado la imagen de color original.

Para que el filtro funcione adecuadamente es importante una sincronización espacial entre la imagen de color y la imagen con información de profundidad, en el caso contrario se obtienen desplazamientos y rotaciones no deseadas en el filtro. Para dicha sincronización es importante tener los parámetros de calibración del conjunto de las cámaras de luz IR y visible. En la Figura **??** se muestran las imágenes obtenidas por ambas cámaras y las imágenes del filtro por distancias para los casos con y sin calibración.

3.3 Desarrollo de Camimic

En la página 2.2.1 se hizo una descripción de lo que pretendía ser esta pizarra virtual y del diagrama de flujo propuesto en la fase de diseño. Si bien se ha intentado desarrollar la aplicación conforme a lo planeado, un mayor requisito de tiempo y esfuerzo para desarrollar el *tracking* ha impedido pulir la interfaz de la aplicación así como la integración completa del método finalmente propuesto dentro de la GUI de Camimic.

En la Figura 3.7 se observa el filtro por distancia implementado en Camimic y dos ventanas de OpenCV pre-procesando la imagen para la segmentación de objetos a color. En la imagen del filtro podemos apreciar las barras de desplazamiento que controlan la distancia y el grosor de la línea azul, dicha línea azul indica el área de la imagen que pasaría por el filtro, el área roja tiene distinta saturación en función de la distancia a la que se encuentra cada uno de los píxeles de la imagen.

En realidad parte de las tareas que han quedado sin completar, son menos interesante desde el punto de vista de ingeniería, ya que se tratan de desarrollar tareas simples pero que llevan tiempo. Este sería el caso de activar funciones como cambio de color de la pintura cuando el puntero se acerca a las esquinas de la imagen.

3.3.1 Integración de OpenCV

La integración de OpenCV en Camimic en general ha sido más costosa de lo que se estimó en un principio, principalmente a la hora reutilizar código de otros trabajos externos debido a que por costumbre o por compatibilidad, la mayoría de los trabajos que se han revisado hacen uso de versiones antiguas de OpenCV o incluso usan la

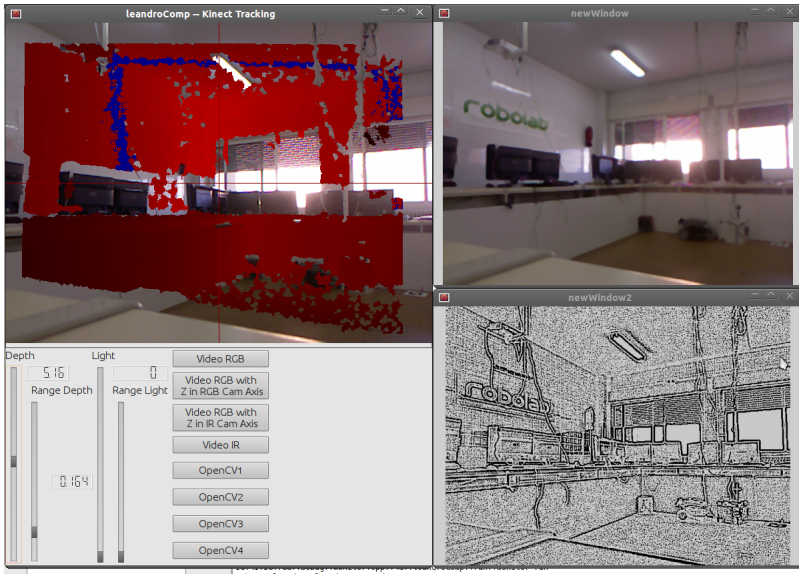

Figura 3.7.: Camimic filtrando por distancia y usando OpenCV.

API de C pese a que sus aplicaciones estén escritas en C++ y teniendo en cuenta que OpenCV tiene su propia API para este lenguaje orientado a objetos. Los tipos de imagen, algo realmente importante cuando trabajamos con este tipo de librerías, también ha cambiado respecto a versiones anteriores, por tanto todo el procesado en el que Camimic hace uso de OpenCV se ha trabado con imágenes Mat en lugar de las populares pero obsoletas IplImage.

3.3.2 Prueba de detección de color

Se han realizado una serie de pruebas para la detección de objetos en base a su color, todas ellas usando distintas funciones de OpenCV a modo de filtro y la interfaz gráfica para poder variar los parámetros de forma cómoda y rápida. Las primeras consideraciones tomadas fueron usar unas tolerancias determinadas para cada canal (rojo, verde y azul) de la imagen, con muy pobres resultados. El motivo de los malos resultados se deben a que tanto en zonas saturadas en la imagen (demasiada luz) o en las zonas con valores bajos de luz (sombras y objetos negros), el ruido de los sensores CMOS (CMOS) permitía que hubiera píxeles aleatoriamente cumpliendo las condiciones del filtro.

El uso del filtro por distancia como paso previo a la detección de color supone una notable mejora en nuestro filtro de color, ya que al eliminar el fondo se descarta un área considerable de la imagen. Además los objetos del fondo quedan representados en la imagen capturada por la cámara con un menor tamaño que los objetos que se encuentran al frente, es por esto que el fondo suele presentar un aspecto más heterogéneo y que dificulta el correcto funcionamiento del filtro.

A pesar de usar el filtro por distancia previamente, la primera versión de nuestro filtro de color no es práctica para su uso en Camimic, principalmente porque las condiciones de iluminación de la escena siguen afectando demasiado, por lo que se

intenta eliminar esta dependencia con la siguiente versión del filtro, filtro de color en el espacio HSV (HSV).

El motivo por que usamos el espacio HSV para eliminar la dependencia de la iluminación de la escena, es que en dicho espacio de color los tres canales son la tonalidad, la saturación y el valor (directamente proporcional a la iluminación), de forma que con podemos ajustar un tono determinado y definir un margen en la saturación, de forma que prescindimos del valor y de esta forma los objetos que queremos detectar pasan el filtro de una forma más independiente de la escena. El problema que no se consideró en un primer momento es que aunque los valores de H, S y V son independientes entre sí, no lo es el ruido que les afecta. Por ejemplo, con valores de V demasiado bajos, el error de saturación es mayor.

El umbralizado y la binarización se han aplicado para segmentar el objeto y poder calcular su centro de masas. De forma externa a Camimic se ha creado un pequeño programa con OpenCV que hace recuento del área en una imagen binarizada. Esta aplicación, que se incluye en la carpeta de programas de este PFC con el nombre PFC tiene la finalidad de caracterizar el comportamiento de la llama en distintos experimentos que se llevan a cabo en laboratorios de ingeniería forestal. Este programa se llama FlameMeter y en el próximo capítulo se explicará con más detalle como lo aplicado para esta tarea puede servir para mejoras y trabajos futuros en el desarrollo de Camimic.

3.3.3 Uso de CAMShift en Camimic

Tras probar el programa de ejemplo de *tracking* por el método Continuously Adaptive Mean-Shift (CAMShift) que se incluye con OpenCV, se intentó integrar en Camimic. Este tipo de integraciones (de ejemplos con un único archivo cpp a una clase dentro de nuestro componente de RoboComp) resultaron ser difíciles las primeras veces, pero la experiencia facilitó la integración posterior para el caso de otros ejemplos en ARMole. Sin embargo se descartó su uso debido a una serie de problemas que se explican con más detalle en el siguiente capítulo.

En este punto el desarrollo de Camimic sufrió una parada y no se volvería a reanudar hasta un tiempo más tarde, tras una mayor profundización en conocimientos del problema que enfrenta Camimic en la parte del *tracking* de objetos por color.

3.3.4 Pizarra virtual externa

Tras comprender con más detalle el funcionamiento del Filtro de Kalman como sistema de predicción de variables más que como sistema integrado de *tracking*, finalmente se desarrolló la aplicación de pizarra virtual si bien esta se creó de forma externa a Camimic para ganar tiempo y tener el sistema de *tracking* en funcionamiento. Ya que se priorizó en esta tarea frente a la integración dentro de Camimic, que por otra parte no encierra demasiada dificultad. En la Imagen 3.8 podemos ver cómo funciona esta pizarra virtual:

- Primeramente separamos el canal rojo de la imagen a color.

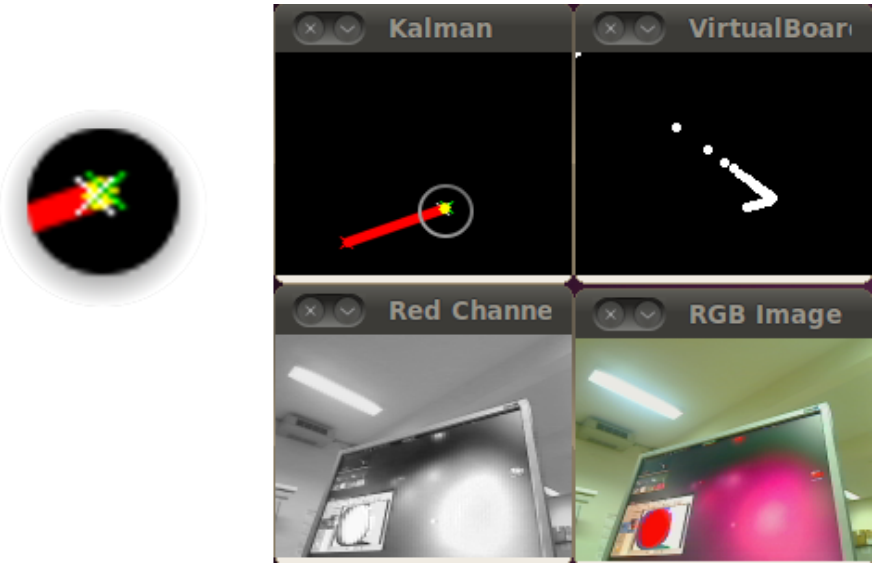

Figura 3.8.: Pizarra virtual usando el Filtro de Kalman.

- Aplicamos la transformada de Hough para la detección de círculos, en la que conseguimos su centro y su radio.
- Lanzamos el predictor del Filtro de Kalman que es actualizado con cada valor del centro detectado y con error de predicción del instante anterior.
- Finalmente en la pizarra negra se pinta de blanco los centros estimados por el predictor.

En la ventana nombrada como Kalman podemos ver un segmento rojo y otro amarillo. Estos segmentos son la distancia entre la posición actual del centro del círculo y la predicción anterior para el caso rojo, y la distancia entre la predicción anterior y la posterior para la línea amarilla. En el caso ideal de funcionamiento de predictor, el segmento rojo debería medir menos que el amarillo de forma que la posición actual del círculo se encuentre en la predicción anterior y la posterior.

El algoritmo desarrollado para la parte de la predicción consiste en una adaptación del ejemplo de OpenCV que aplica el Filtro de Kalman para la predicción de una variable unidimensional que toma valores aleatorios y supone velocidad constante, aunque calculada durante la fase de iniciación del filtro. Si bien el filtro no funciona de forma óptima en la mayor parte de su uso como pizarra virtual, en principio el resultado final para poder dibujar es aceptable. En el capítulo siguiente se expone con detalle la capacidad de mejora del método desarrollado, sobretodo porque es posible usar el trabajo previo y elaborar un *tracking* más robusto, aunque también más complejo.

3.4 Desarrollo de ARMole

Gracias a la experiencia adquirida creando Camimic, el programa ARMole ha tenido una evolución más constante, un resultado más ajustado al diseño inicial, y

un mejor acabado final. En esta ocasión la creación del componente, la obtención de imágenes a través de la cámara o el dispositivo Kinect, y el desarrollo de la interfaz gráfica con las conexiones a funciones del componente, fueron tareas que no requirieron tanto tiempo como en el caso de Camimic.

Además esta aplicación surge a partir de una propuesta por parte de Pablo Bustos (coordinador de RoboLab) en el momento en que se está considerando las posibilidades de usar ARToolKit para Camimic, tras descartar CAMShift. El porqué de este programa se ha explicado anteriormente en el capítulo Diseño.

3.4.1 La clase Osgart

No ha sido necesario la integración total de OSGArt gracias a que ésta ya se había realizado anteriormente en RoboLab por Luis Vicente Calderita para el componente sevillaComp, sin embargo sí que se ha tenido que mejorar y adaptar esta clase para ARMole.

La mejora resultó un trabajo considerable en la que se añadieron una gran cantidad de métodos para generar primitivas más allá de simples esferas, que eran las únicas figuras soportadas por esta clase. También se añadieron métodos para modificar de manera dinámica las propiedades de estas primitivas como pueden ser el color, la posición, rotaciones, etc. Estas mejoras se hicieron de forma conjunta con los estudiantes Pablo Manzano y Alejandro Hidalgo que también necesitaron usar esta clases para su PFC. Finalmente fue necesario una adaptación más específica para el uso en ARMole como es el uso de *tracking* simultáneo de dos marcas, una para establecer la posición de los topos, y otra para el martillo. Otra modificación necesaria ha sido el cálculo de distancias entre marcas, ya que la interacción entre el martillo y los topos no hubiera sido posible de otro modo, pues OSG no soporta colisiones entre primitivas o modelos. En la Figura 3.9 el lector puede observar el uso de la clase Osgart con dichas modificaciones, donde los conos de mayor tamaño tienen su posición referenciada a una marca mientras que las esferas y los conos más pequeños a otra.

3.4.2 Integración de Modelos 3-D en OSG

Gracias a la facilidad que tiene OSG de trabajar con modelos tridimensionales en formato (COLLADA), hemos convertido el modelo para los topos a dicho formato. Dicho modelo se ha seleccionado de la biblioteca pública de modelos 3D de Google SkechtUp. Con intención de suplir la falta de animación del modelo, se ha modificado el original para establecer el estado de los topos cuando deben estar ocultos o cuando son golpeados con el martillo, de esta forma aunque no sea animación real, ésta es simulada con un acabado aceptable. Un diagrama con la escena completa se puede observar en la Figura 3.10, y se describe con más detalle a continuación:

- Cada caja representa un nodo en la escena, y las flechas indican la relación padre-hijo en la jerarquía. Las flechas punteadas significan que esas relaciones son dinámicas, o dicho de otro modo esas relaciones se crean y destruyen a lo largo de la ejecución bajo ciertas condiciones.

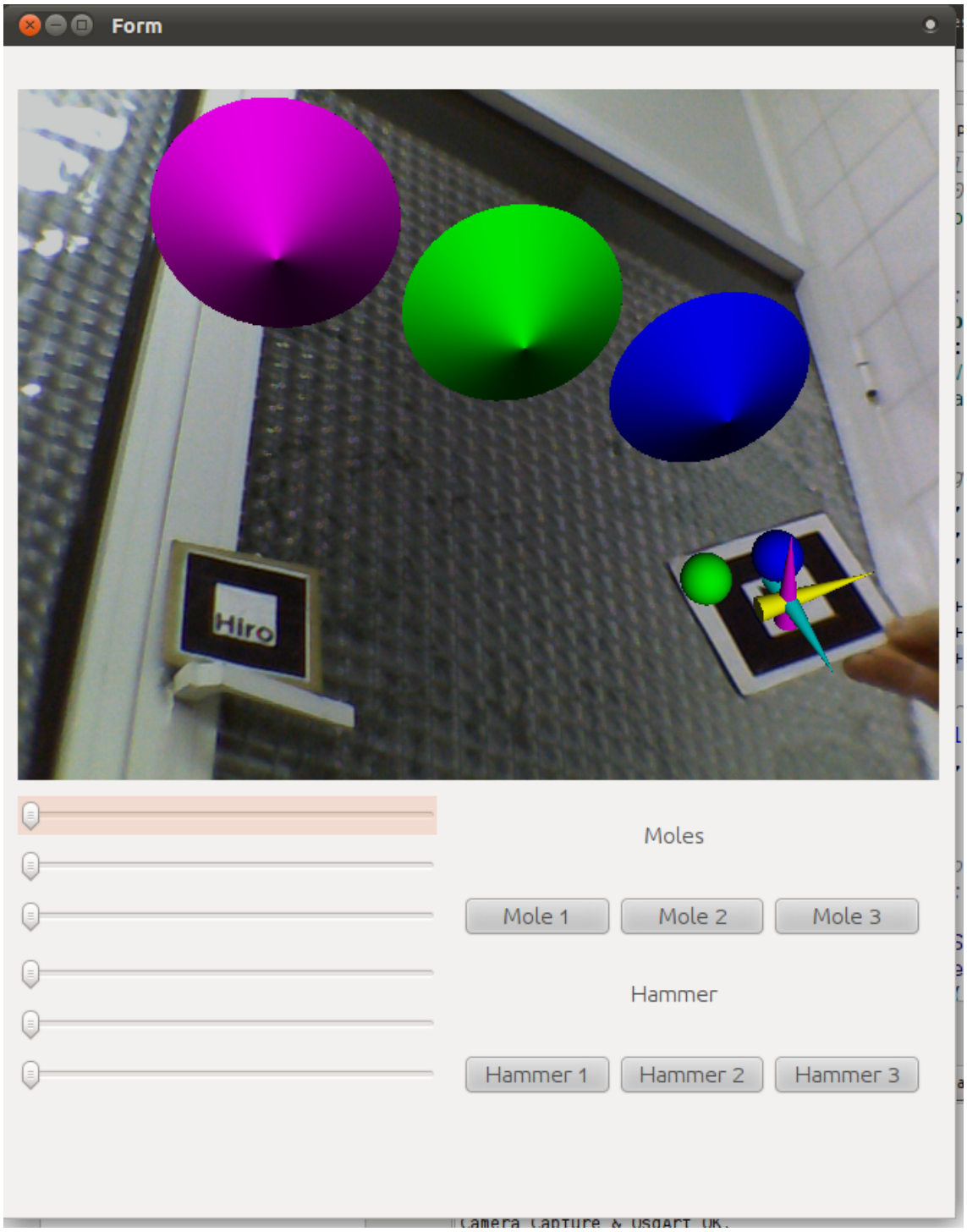

Figura 3.9.: Puebra de la clase Osgart modificada.

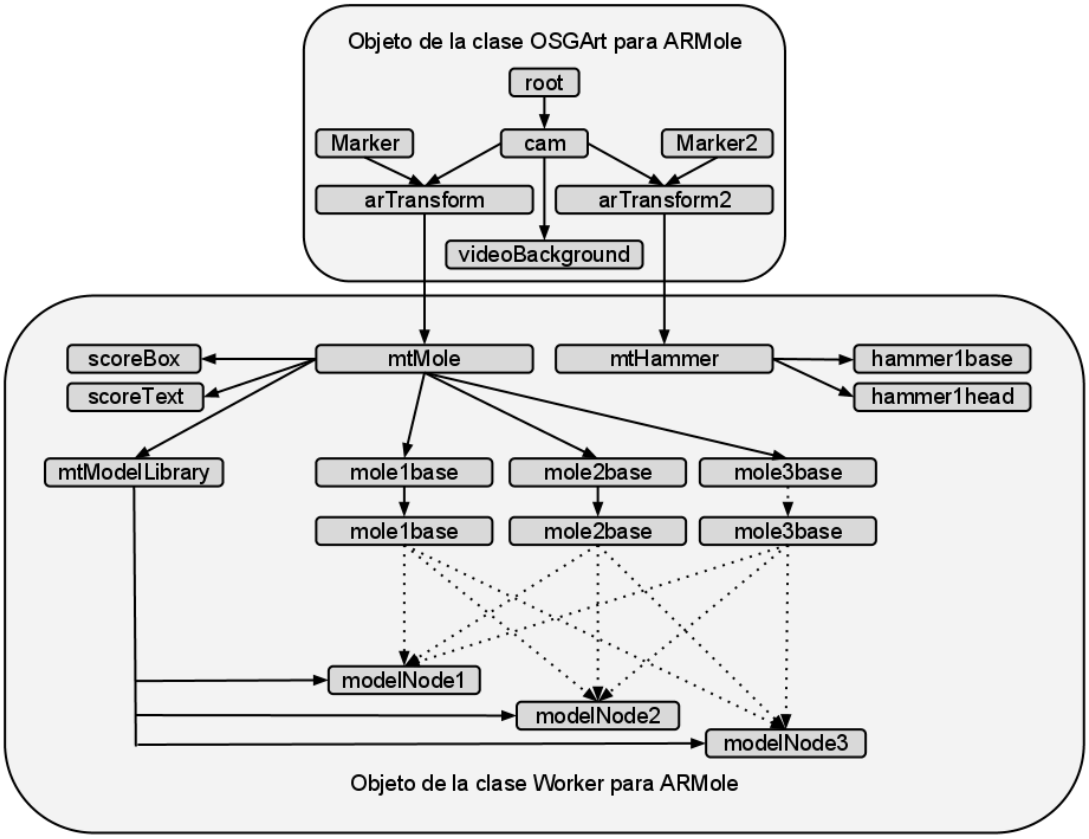

Figura 3.10.: Jerarquía de la escena 3D de ARMole

- La parte estructural interna de la clase Osgart es la misma usada por sevillaComp pero se ha duplicado el código oportuno para soportar la *tracking* de dos marcas al mismo tiempo. El nodo root es común en todas las escenas de OSG y lo mismo ocurre con el nodo cam, que representa la cámara desde la que vemos la escena tridimensional, no es realmente una cámara de vídeo física. Sin embargo a este nodo se le pasa la imagen de vídeo capturada por la cámara real de forma que se usa dicha imagen como fondo de la escena.
- Los nodos arTransform y arTransform2 son matrices de transformación con sus parámetros usando el nodo cam como referencia, pero variando sus valores de forma dinámica según la detección de las marcas de ARToolKit.
- Los nodos mtMole y mtHammer son matrices de transformación para los topos y el martillo respectivamente. Los valores de estas matrices varían según los valores de los nodos padre, Sin embargo internamente ARMole puede añadir valores estáticos internamente.
- scoreBox y scoreText son primitivas para crear el marcador de puntuación del juego.
- mtModelLibrary es la matriz de transformación que controla la posición y rotación de la librería de modelos 3D, sus valores están establecidos para que todos los nodos hijos caigan fuera del campo de visión de la cámara.
- Los nodos modelNode1, modelNode2, y modelNode3 tienen los modelos cargados para poder simular la animación. Aunque en un principio pudiera parecer absurdo tener en la escena nodos que caen fuera del campo de visión, la solución planteada de un resultado mucho más eficiente que tener que cagar de nuevo cada modelo según se vaya necesitando. El motivo es que si un nodo no tiene enlace de algún tipo con el nodo root este nodo será destruido, por lo que para cambio de estado (oculto, descubierto y golpeado) se destruiría en nodo con el modelo y se tendría que recargar constantemente. La carga suele llevar un par de segundos, por lo que no se podría interactuar con la aplicación.
- hammer1head y hammer1base son los cilindros que conforman el martillo, el mazo y el mango respectivamente.
- Los nodos mole1base, mole2base y mole3base están pensados para controlar la matriz de transformación relativa a cada topo, estos nodos actualmente tienen sus respectivos hijos que apuntan a los modelos de la biblioteca mtModelLibrary, pero se podrían añadir otros elementos para individualizar cada uno de esos tres topos.

3.4.3 Interacción entre marcas ARToolKit y OSG

Para poder implementar la interacción entre marcas el primer paso es obtener la distancia entre las mismas, para ello se ha creado una función que calcula la diferencia entre dos vectores tridimensionales, y esa función la usamos de forma iterativa recorriendo la escena de la Figura 3.10 desde la cámara hacia abajo. En la Figura

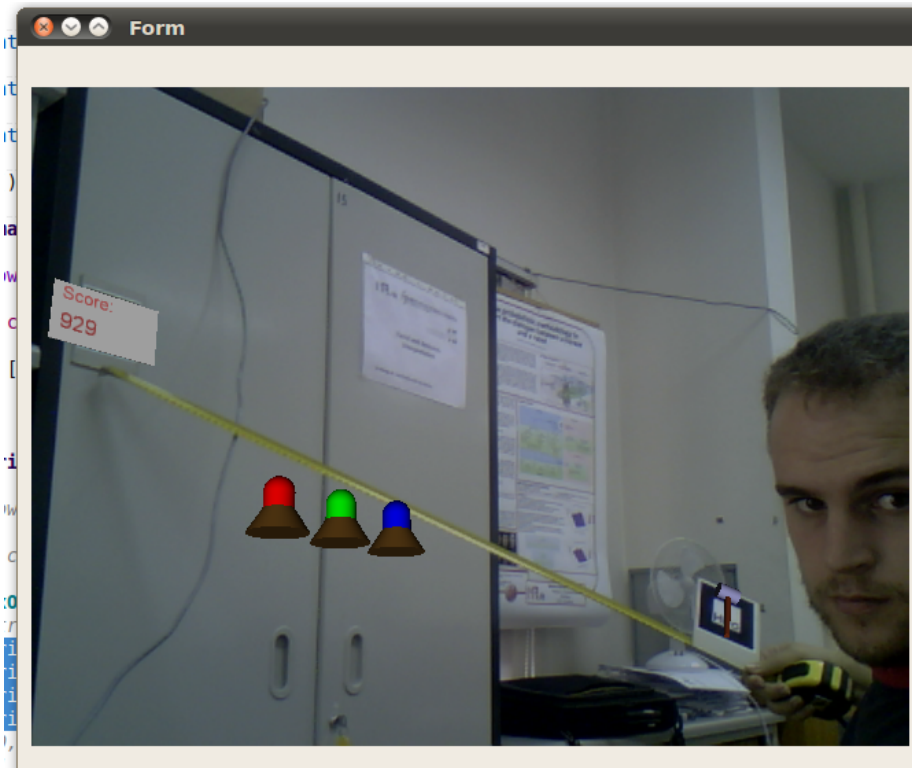

Figura 3.11.: Comprobando la distancia real entre marcas.

3.11 se está comprobando que la distancia calculada con el método expuesto se corresponde con la realidad.

Una vez que tenemos la distancia entre el martillo y cada topo, se establece un umbral en torno a éstos de forma que cuando el martillo traspasa dicho umbral el topo correspondiente apunta al modelo de la librería relacionado con el estado golpeado del topo.

Además se ha creado una variable que toma valores aleatorios, y el estado oculto y descubierto de los topos depende de los valores de dicha variable. Para que cada topo tenga un comportamiento individual, a cada uno de ellos se le ha asignado una reacción a rangos de valores distintos. La cantidad de valores asignados permite controlar la velocidad con la que los topos cambian de estado, pero además de pueden asignar los mismos valores a dos o más topos, lo que generaría el caso en que dos o tres topos tuvieran un comportamiento exacto en determinados momentos. Sin duda se puede jugar con estos valores para crear niveles de dificultad y diversión.

3.4.4 Integración de audio con OSGAL

Finalmente se ha creado una clase para integrar Osgal en ARMole, de forma que la aplicación puede usar sonido de forma bastante sencilla. Aunque hasta el momento ARMole sólo reproduce sonido cuando el martillo golpea a los topos, las posibilidades para añadir música de fondo por niveles, o más sonidos cuando incrementan los

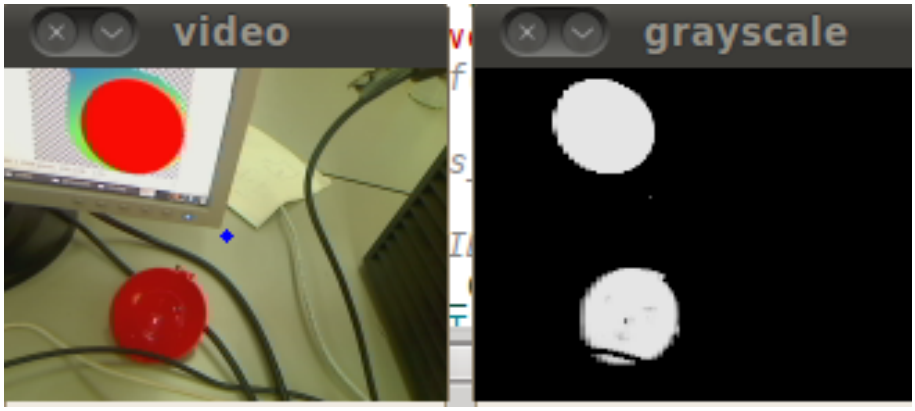

Figura 3.12.: Mapa de probabilidad para objetos rojos.

puntos en el marcador están abiertas. Pero quizás lo más interesante sea, que a partir de esta clase futuros componentes de RoboComp pueden usar OSGAL para distintas aplicaciones, ahorrando el tiempo y el esfuerzo que este PFC a dedicado a esta integración. Esta tarea se ha realizado modificando el archivo CMakeLists.txt de ARMole y adaptando alguno de los ejemplos de la librería a una clase, creando nuevos métodos privados y públicos.

3.5 Enfoque probabilístico para filtro de color

La tarea principal durante la estancia de tres meses en el ISR de Coimbra era la depuración de un programa de relativa complejidad:

- Dos hilos de ejecución además del principal. Estos hilos compartían la imagen capturada por la cámara, por lo que era necesario controlar las condiciones de carrera.
- Tres métodos distintos de modelar el sensor de la cámara (ProBT, Analítico CPU y Analítico GPU).
- Benchmark para controlar el tiempo de ejecución de cada parte del programa.
- Detección de círculos con la transformada de Hough.
- Mapa de probabilidad de ocupación de píxeles, para el caso de objetos rojos se puede observar en la Figura 3.12
- Distribuciones gausianas que podemos ver en la Figura 3.13.
- Control de motores una cámara Logitech Sphere AF para que el centro de la cámara apunte al objeto a seguir.

Se ha aprovechado para estudiar las posibilidades de las técnicas y el enfoque de este programa para aplicarlo a Camimic, e incluso a ARMole. Por lo tanto se ha experimentado con un programa más simple que aplica el teorema de probabilidad

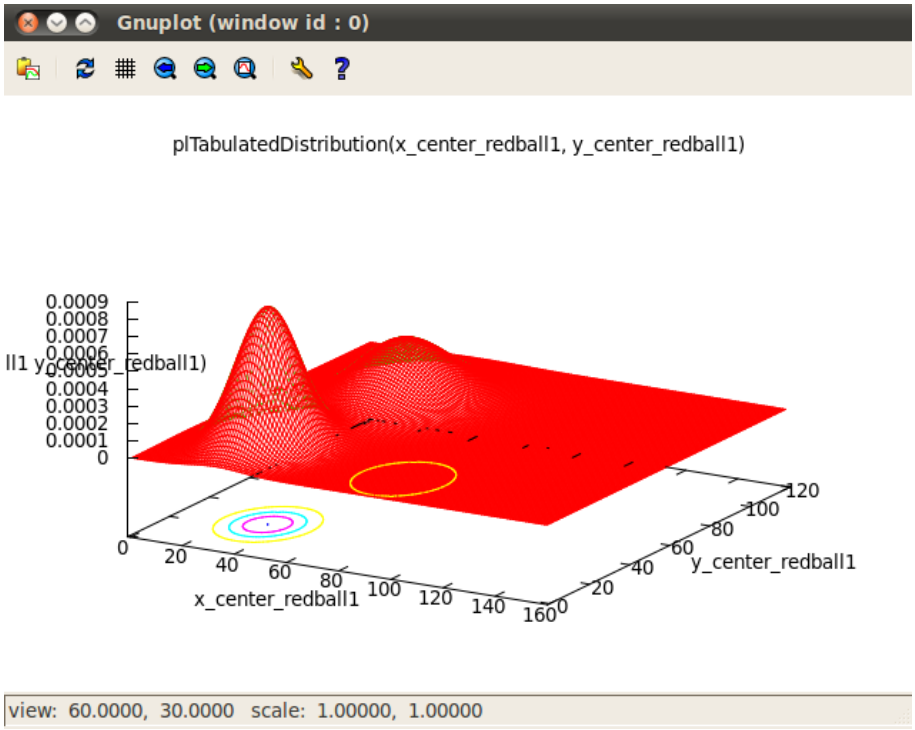

Figura 3.13.: Distribución de probabilidad dibujada con GNUPlot.

condicional o de Bayes para obtener una especie de filtro de color pero con un enfoque probabilístico frente al determinístico que empleábamos al principio. Además tras este trabajo se hizo más fácil comprender que el Filtro de Kalman o el Filtro de Partículas en realidad son casos específicos de redes Bayesianas. El filtro da buenos resultados, y aunque los tiempos de ejecución para cada captura de imagen eran excesivos en los casos de usar ProBT o paralelización en la CPU, corriendo el proceso en la GPU la velocidad era aceptable. En la Imagen 3.14 se puede ver el resultado del filtro con distinto nivel de diezmado para conseguir velocidades de aspecto en tiempo real. El modelo del sensor no está completamente optimizado, si bien utiliza distribuciones alfa-beta con valores obtenidos de experimentos realizados por otros trabajos anteriores, con el uso del espacio de color HSV o HSL se podrían obtener mejores resultados, ya que el ruido de saturación está en cierto modo condicionada a la iluminación.

Los programas creados se han incluido entre los archivos que acompañan a esta memoria.

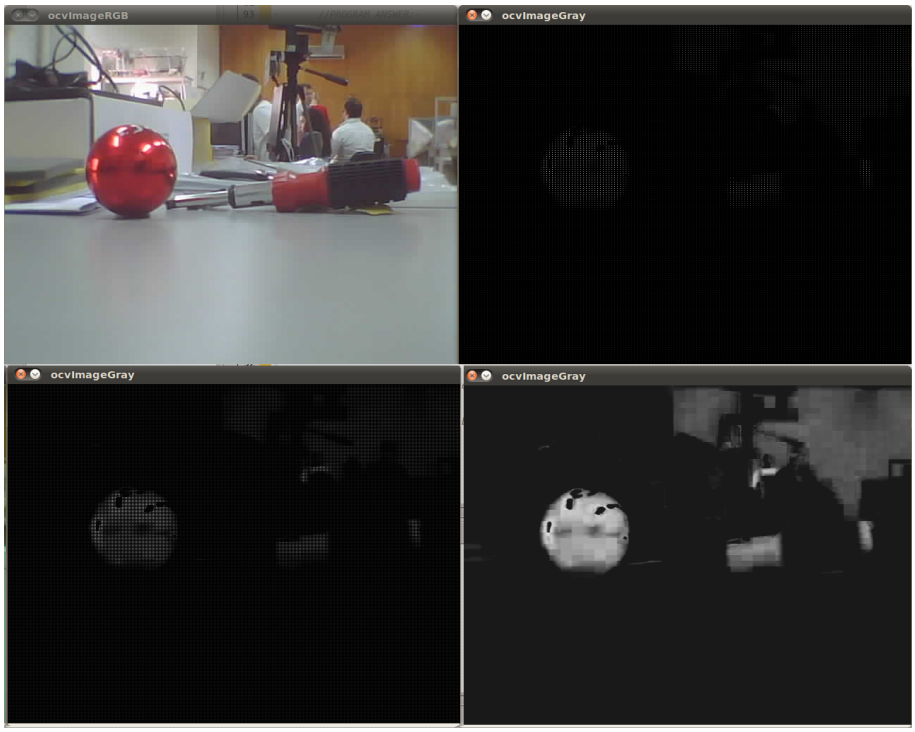

Figura 3.14.: Mapa de probabilidad con diezmado para aumentar velocidad.

4. CONCLUSIONES

En este capítulo comentaremos los problemas que han surgido durante el desarrollo del proyecto, cuales de ellos se han superado y cómo se podrían solucionar aquellos no resueltos. Además se presenta el estado final de las aplicaciones, la utilidad de cada una, y se proponen futuras líneas de trabajo que podrían basarse en lo expuesto en esta memoria.

4.1 Problemas

De forma general, es decir tanto para ARMole como para Camimic, los mayores problemas han surgido a la hora de tener que adaptar el código a distintas versiones de sistema operativo. También ha sido necesario superar una brecha de conocimientos respecto a programación general (no se había trabajado anteriormente con punteros), programación orientada a objetos y a componentes, y en cuanto al uso de de tantas librerías.

En el caso de Camimic, muchos problemas pequeños han surgido por los cambios en los nombres de las funciones según las diferentes versiones de OpenCV. En la Figura 4.1 vemos como el espacio BGR que usa OpenCV resulta confuso y puede llevarnos a pensar que hay algo mal en la cámara. En otras ocasiones la ayuda que se puede encontrar *online* siempre hace uso de la versión obsoleta o de las funciones para C en lugar de para C++. De hecho por el contacto que se ha mantenido a lo largo de este proyecto con otros estudiantes e investigadores, parece ser una práctica común usar funciones C de versiones obsoletas a pesar de tener instalada la última versión de OpenCV y desarrollar aplicaciones en C++. Si bien es cierto que el verdadero problema durante el desarrollo de Camimic, y el más interesante desde el punto de vista de la ingeniería, es el tracking de objetos, no es menos cierto que los errores con OpenCV han ralentizado la capacidad de experimentación con más

Figura 4.1.: La misma imagen en orden común (RGB) y en el orden de OpenCV (BGR).

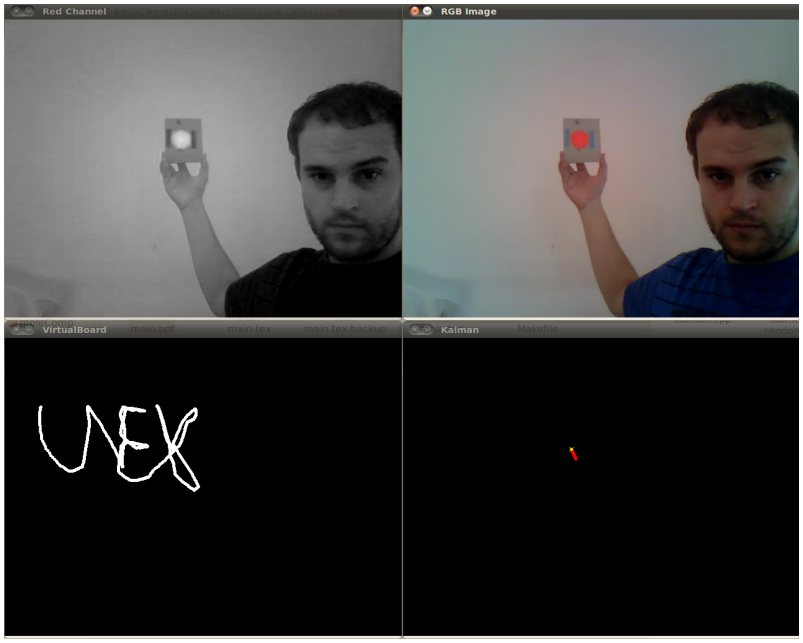

Figura 4.2.: Escribiendo UNEX en la pizarra virtual.

profundidad de los métodos que se nombran en el Capítulo 2. Un problema que ha quedado sin resolver es la obtención de los *handler* de la interfaz de OpenCV desde nuestro componente, o bien pasarle a OpenCV los eventos capturados desde la GUI en QT.

Como ya se comentó con anterioridad ARMole ha supuesto un desarrollo más homogéneo y sin estancamientos prolongados como se puede apreciar en el diagrama de Gantt de la página 10. Aunque en el caso de controlar la distancia entre objetos de OSG, la consecución de dicha tarea llevó cerca de un mes, después de explorar varias opciones finalmente descartadas como osgBullet u osgEdit. Otro problema resuelto en ARMole que merece consideración es la animación simple llevada a cabo con una práctica poco común pero que resuelve el problema de forma muy eficiente. Sin embargo como solución al problema que intentamos resolver (evitar los abandonos de tratamientos de rehabilitación) no está del todo acabado, pues este programa es sólo una muestra de las posibilidades que aporta el campo de la Realidad Aumentada.

4.2 Soluciones Aportadas

4.2.1 Estado final de Camimic

Si bien el estado final de Camimic no es el diseñado en un primer momento, si cumple con la idea básica inicial, que era conseguir una pizarra virtual con la que el usuario pudiera pintar con movimientos y gestos en el aire, capturados por una cámara e implementando un *tracking* que haga uso de la información de color. En la Figura 4.2 podemos ver el uso de la pizarra donde se ha intentado escribir UNEX, en referencia a la Universidad de Extremadura. Esta aplicación no tiene un acabado

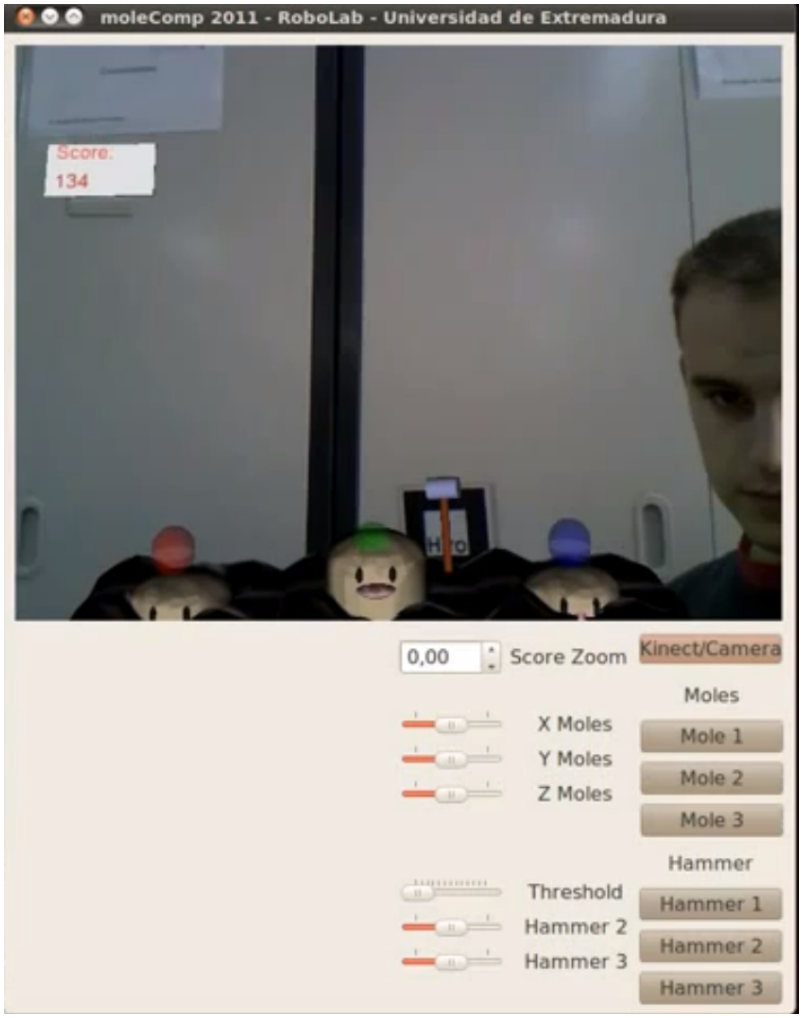

Figura 4.3.: Jugando con ARMole.

final de cara al usuario y no se ha integrado en Camimic por falta de tiempo, sin embargo dicha tarea no tiene mayor complejidad, y más después de la experiencia con integraciones anteriores.

4.2.2 Estado final de ARMole

Este programa se ha finalizado con mejor aspecto audiovisual: una interfaz sencilla, modelos simples pero graciosos, y sonido. Pero desde un punto de vista técnico lo más importante es la interacción que se ha conseguido entre las marcas y los objetos, así como la posibilidad de configurar parte de la escena desde la GUI, y sin tener que reiniciar la aplicación. Una muestra del resultado es la Figura 4.3, si bien es en vídeos o ejecutando la aplicación directamente cuando se puede apreciar el uso potencial de ARMole.

4.3 Propuesta de trabajos futuros

Otros PFC futuros podrían continuar desde el trabajo aquí realizado, pero no sólo proyectos, los programas Camimic y ARMole tienen aplicaciones dirigidas a usuarios finales y empresas o centros de investigación pueden continuar su desarrollo y adaptarlos para sus usos particulares. Este trabajo puede ser especialmente aprovechado desde RoboLab por tratarse de componentes de RoboComp.

En el caso de Camimic el siguiente paso claramente es la integración de la pizarra virtual que usa el Filtro de Kalman, en el propio componente que usa filtro por distancia, cambio de espacio de color, interfaz gráfica de usuario, pequeño control sobre Kinect, etc. Sin embargo resulta interesante nombrar algunas ideas para mejorar el *tracking* que no han podido ser aplicadas por falta de tiempo:

- Optimizar los valores de Q (ruido del proceso) y R (ruido de la medida) mediante experimentos para el uso específico del Filtro de Kalman como pizarra virtual. Se podría hacer un estudio estadístico y detectar si estos valores deberían cambiar según el usuario de dicha pizarra, ya que dado el caso se podría hacer una calibración de los valores Q y R para cada usuario. La idea mejorar el funcionamiento del filtro de forma que el segmento amarillo de la Figura 4.2 tenga mayor longitud que el rojo.

- Añadir los valores de velocidad y aceleración al Filtro de Kalman de forma que se puedan medir y predecir esas variables, aproximando mejor el modelo a la realidad.

- Usar filtro de distancia con Kinect para eliminar el fondo de la escena. Añadir la distancia como una tercera dimensión del Filtro de Kalman de forma que se consiga el *tracking* tridimensional.

- Desarrollar la interfaz de cara al usuario y completando las funciones de la pizarra, como cambio de propiedades del pincel o añadir la posibilidad de dibujar figuras geométricas.

- Seguir el diagrama de flujo de la Figura 2.3 que sigue siendo válido para permitir interacción similar a un ratón y el uso de botones en las esquinas de la imagen.

En principio podría parecer que ARMole no tiene tanta capacidad de mejora técnica, sin embargo es en este punto donde confluye el trabajo aplicado en Camimic y en ARMole, ya que lo ideal sería substituir las marcas de ARToolKit por un sistema de *tracking* menos invasivo, como sería el que usa Camimic. Por lo demás sería mejorar una serie de detalles que sólo se pueden conseguir en base a probar la aplicación en su uso con pacientes y estudiar los problemas que puedan surgir de forma que ARMole se pueda adaptar a los ejercicios de forma más específica. Además otros juegos de Realidad Aumentada con esta u otras aplicaciones pueden derivar a base de modificar el código de ARMole.

5. VALORACIÓN PERSONAL

Este es un pequeño capítulo en el que hago una valoración personal del proyecto y me tomo la libertad de escribir en primera persona.

5.1 Evaluación de objetivos

La evaluación general es positiva, ya que no sólo se han cumplido los objetivos establecidos al inicio del proyecto, sino que se ha pasado por algunas etapas que realmente podrían suponer objetivos principales de un PFC. Quizás el mayor problema que ha enfrentado este proyecto es tomar la decisión de cuando parar, ya que estos ocho meses han supuesto para mí un crecimiento a nivel personal y como estudiante de ingeniería.

A pesar de no ser objetivo de PFC, se ha mantenido un blog en internet con publicaciones sobre el avance, problemas resueltos y síntesis de algunas herramientas utilizadas. La intención de este blog ha sido la divulgación de los temas que trata este proyecto, intención cumplida al haber llegado a más de diez mil visitas. Sin embargo en la práctica ha resultado ser mucho más, pues me ha puesto en contacto con otros estudiantes e investigadores, a la vez que ha servido como borrador de algunas secciones de esta memoria.

5.2 Evaluación de complejidad

Considero este PFC de relativa dificultad partiendo de la base en cuestión de programación con la que los ITT en Imagen y Sonido terminamos las asignaturas. Y sin lugar a dudas el tiempo dedicado supera al que corresponde con los siete créditos y medio del Proyecto Fin de Carrera.

5.3 Evaluación del tutor

No podría finalizar este capítulo de valoración personal sin nombrar la evaluación positiva del tutor de este PFC, Pedro M. Núñez. Si lo cierto es que no hemos podido cumplir con las evaluaciones de etapas del proyecto en las fechas inicialmente establecidas en el diagrama de la Figura 2.1, siempre ha estado disponible bien en persona o vía telemática para responder a mis dudas, evaluar el progreso, y darme consejos para solucionar algunos problemas. Además ha sabido canalizar las inquietudes con las que le solicité la tutoría, en un PFC completo y con una clara aplicación a la sociedad, pero además ofreciéndome la oportunidad de desarrollarlo en las instalaciones de RoboLab y del ISR.

BIBLIOGRAFÍA (LIBROS Y ARTÍCULOS)

[1] Juan Manuel Ahuactzin and Emmanuel Mazer. *ProBT Programmers Guide*, Junio 2007.

[2] Atif Alamri and Jongeun Cha. Augmented reality game for post-stroke patients rehabilitation. Technical report, Multimedia Communications Research Laboratory, 2008.

[3] Ronald T. Azuma. A Survey of Augmented Reality. In *In Presence: Teleoperators and Virtual Environments 6, 4, 355-385.*

[4] Pilar Bachiller, Pablo Bustos, and Luis J. Manso. Robex and robocomp: Building intelligent robots. Documento borrador de RoboLab para ayudar a construir componentes con la versión 0.8 de RoboComp, 2008.

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, Septiembre 2008.

[6] Javier García de Jalón, José I. Rodríguez, José M. Sarriegui, and A. Brazález. *Aprenda C++ como si estuviera en primero.* Escuela Superior de Ingenieros Industriales de la Universidad de Navarra, San Sebastián, Abril 1998.

[7] María Isabel Menor Flores. Tracking automático de objetos en secuenciales de imágenes usando Filtro de Kalman. Master's thesis, Universidad de Extremadura, Septiembre 2009.

[8] Antonio J. Céspedes Izquierdo. AR-Learning: Realidad Aumentada y mecanismos de interacción al servicio de la educación. Master's thesis, Universidad de Extremadura, Junio 2011.

[9] Jorge Dias João Filipe Ferreira. Probabilistic approaches for robotic perception. Documento monográfico en estado borrador, desarrollándose en el ISR, 2011.

[10] Eva Chaparro Laso. Tracking automático de objetos en secuenciales de imágenes usando Filtro de Partículas. Master's thesis, Universidad de Extremadura, Septiembre 2009.

[11] Paul Martz. *OpenSceneGraph Quick Start Guide*, 2007.

[12] Mark Spruiell Michi Henning. *Distributed Programming with Ice*, 2010.

[13] nVidia. *CUDA Programming Guide Version 1.1*, 2007. Manual para programación paralela en la GPU de tarjetas gráficas nVidia.

[14] Pedro M. Núñez. Robocomp for dummies. Documento de RoboLab para construir componentes con GUI en RoboComp, 2010.

[15] Tanasha Taylor and Shana Smith. A virtual harp for therapy in an augmented reality environment. Technical report, Iowa State University and Taiwan National University, 2008.

[16] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

BIBLIOGRAFÍA (OTRAS FUENTES)

[17] Opencv wiki [online]. Diciembre 2010. URL: `http://opencv.willowgarage.com/wiki/`.

[18] Opencv 2.2 c++ reference [online]. Marzo 2011. URL: `http://opencv.jp/opencv-2.2_org/cpp/index.html`.

[19] Openkinect wiki [online]. Marzo 2011. URL: `http://openkinect.org/wiki/Main_Page`.

[20] Qt reference documentation [online]. Marzo 2011. URL: `http://doc.qt.nokia.com/4.7/modules.html`.

A. GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

ACROSS *Auto Configurable RObots for Social Services* es un proyecto que pretende incorporar servicios robóticos en escenarios sociales.

ALUT OpenAL Utility Toolkit

API *Application Programming Interface* es la documentación para poder usar funciones de librerías externas.

ARToolKit Augmented Reality Tool Kit es una librería para el desarrollo de programas que hagan uso de la Realidad Aumentada.

Bash Bourne-again Shell es un tipo de intérprete de comandos popular en los sistemas operativos UNIX.)

BSD *Berkeley Software Distribution* (Distribución de *Software* libre de la Universidad de Berkeley.)

CAMShift Continuously Adaptive Mean-Shift es un método de tracking de objetos basandose en comparación de histrogramas de imágenes a color.

CHAI 3D Librerías para programas que hagan uso de dispositivos del tacto, interactividad y simulación en tiempo real.

CMake CMake es una herramienta de configuración o precompilación muy utilizada en sistemas operatiovs GNU/Linux.

CMOS CMOS es un tipo barato de sensores de vídeo que se utiliza en la mayoría de las denominadas webcam.

COLLADA es un formato para exportar e importar modelos tridimensionales.

Doxygen Doxygen es un sistema de documentación automática para código fuente de programas.

EPCC Escuela Politécnica de Cáceres

FMOD FMOD es un subsistema de audio comercial utilizado en muchos videojuegos.

FreeNect FreeNect es el nombre del controlador libre de Kinect.

GNU/Linux GNU/Linux es un sistema operativo libre.

GPL GNU Public License es un tipo de licencia que permite la libre distribución de software.

GPU *Graphic Processing Unit* (Unidad de Procesado Gráfico)

GUI *Graphic User Interface* (Interfaz Gráfica de Usuario)

HDM Head Mounted Device es un dispositivo que se monta sobre la cabeza del usuario y le permite ver las imágenes de una pantalla como si fueran unas gafas.

HSV HSV es un espacio de color en el que se divide en los canales tono y saturación, de forma independiente de la luminosidad.

Ice Internet Communications Engine (Motor de comunicaciones sobre internet)

IDE Integrated Development Environment (Entorno de desarrollo integrado)

IPP *Intel*® *Integrated Performance Primitives Library* (Biblioteca para usar las Primitivas de Rendimiento Integrado de Intel)

IR Infrarrojo

ISR Instituto de Sistemas e Robótica (Universidad de Coimbra)

ITT Ingeniería Técnica de Telecomunicación

Kinect *Kinect*® (Dispositivo de captura de video y datos de profundidad, fabricado por Prime *Sense*® y comercializado por *Microsoft*®.)

NITE NITE son unas librerías que utiliza OpenNI internamente.

ODE Open Dynamics Engine es una librería que permite simular comportamientos de física en un entorno virtual.

OpenAL Open Audio Library

OpenCV *Open Source Computer Vision Library* (Biblioteca de *software* para visión artificial, desarrollada por Intel. Licencia libre BSD.)

OpenGL Open Graphics Library (Librería gráfica abierta, es el súbsistema gráfico similar el Direct3D en Windows.)

OpenKinect OpenKinect (Es el grupo que gestiona el driver libre Freenect.)

OpenNI *Open Natural Interaction* (Es el software publocado por varias empresas relacionadas con Kinect para crear un estándar de interación natural.)

OSG OpenSceneGraph es una capa de abstración sobre OpenGL y que permite generar movimientos complejos con una pogramación sencilla.

OSGAL OSG-OpenAL integration (Es una librería que permite la integración de audio en escenas creadas con OSG.)

OSGArt OpenSceneGraph-ARToolKit integration (Es una librería que sirve para integrar estas dos tecnologías.)

PFC Proyecto Fin de Carrera

Prime Sense *PrimeSense*® es una empresa originaria de Israel que dearrolla el chip de la Kinect y de Xtion.

ProBT Professional Bayes Toolkit (Software para programación de redes Bayesianas.)

QT Biblioteca de *software* QT (Biblioteca de *software* QT, desarrollada por *Nokia*®. Versión 4.0. Licencia libre Qt GNU GPL v. 3.0.)

QTDesigner QTDesigner es un programa que nos permite diseñar interfaces de usuario de forma gráfica.

RA Realidad Aumentada

RoboComp *Software* para robots orientado a componentes desarrollado en el RoboLab

RoboLab Laboratorio de Robótica de la UNEX

ROI Region of Interest(Región de Interés)

ROS Robot Operating System es un framework para robótica similar a RoboComp pero desarrollado por el mismo equipo que mantiene OpenCV.

SDK Software Ddevelopment Kit (Kit de desarrollo de software)

SMC *Secuential Monte Carlo*(Monte Carlo Secuencial)

TBB Threading Building Blocks (es un software de abstracción para programar tareas que hagan uso de paralelicación en la CPU.)

Ubuntu Ubuntues una distribución del sistema operativo GNU/Linux, este texto hace referencia a la versión 10.10 *Maverick Meerkat* de 64 bits.

UNEX Universidad de Extremadura

URSUS URSUS es un robot desarrollado en RoboLab.

XML Extensible Markup Language (Lenguaje de etiquetas extensible)

B. LICENCIA

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors'reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain

patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program"means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as ɀou".
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.
You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.
In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sub-

license or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipientséxercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and .ªny later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM .ªS IS"WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the çopyright"line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.
If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c'for details.

The hypothetical commands 'show wánd 'show c'should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show wánd 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision'(which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

C. AGRADECIMIENTOS

- A Pedro Núñez, por su amabilidad y su ayuda prestada durante todo el desarrollo de este Proyecto Fin de Carrera.
- A Luis Vicente Calderita, por su ayuda constante cuando me era imposible solucionar los errores del compilador, y compartir nuestros problemas con las referencias de las marcas de ARToolKit y OSG.
- A Pablo Bustos y al resto del equipo de Robolab, por hacerme sentir como uno más en el laboratorio. Mención especial para Pablo Manzano y Alejando Hidalgo por trabajar conjuntamente en partes de nuestros respectivos proyectos.
- Al desarrollador de KDevelop Aleix Pol, por su ayuda altruista ante mis problemas con el *plugin*.
- A Jorge M. Días por su invitación para realizar la estancia en el ISR, y a todo el equipo que él dirije por enseñarme distintos puntos de vista sobre la Visión por Computador.
- A mis compañeros de carrera por todos estos años estudiando juntos, especialmente me gustaría agradecer a Jesús Arias, Luz Rubio, Fátima Rubio, Marisol Márquez y Mar Martín.