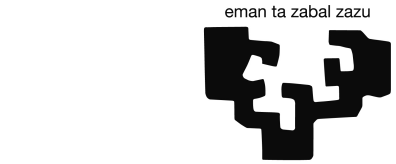


Universidad del País Vasco

Euskal Herriko Unibertsitatea

Escuela Universitaria de Ingeniería Vitoria-Gasteiz

Ingeniaritzako Unibertsitate Eskola Vitoria-Gasteiz

Análisis de un sistema de visión artificial de bajo coste

Memoria del proyecto

presentada para optar al grado de

Ingeniería Informática de Gestión y Sistemas de la Información por

Daniel Laguna Soto

Director

Pablo González Nalda

Septiembre de 2015

Agradecimientos

Gracias a mi madre, por apoyarme en todos los momentos difíciles de mi vida, eres increíble y se que eres capaz de todo con tal de que seamos felices. Para mi siempre seras un modelo a seguir.

Gracias a mi padre, sin el no hubiese llegado a ser lo que soy hoy.

Gracias a mi hermana, aunque te cueste demostrarlo, se que siempre vas a estar ahí a mi lado.

Gracias a Fran, mi amigo de la infancia, gracias por estar ahí siempre que te he pedido ayuda. Te deseo buena suerte en lo que te queda de carrera, estoy seguro de que la terminarás con buenos resultados.

Gracias a Diego, mi compañero de trabajos, sin ti todos estos años no hubiesen sido lo mismo. Me alegro de haberte conocido.

Gracias a Iban, por acompañarme todo este tiempo en la carrera con tan buenos (y *frikis*) momentos.

Gracias a Pablo y Alberto por aquellas conversaciones matutinas camino a Vitoria. Las voy a echar de menos.

Gracias a mis amigos, pues no le habéis dado importancia al hecho de que no os haya dedicado el tiempo que os merecéis durante el desarrollo de este proyecto.

Quiero finalizar agradeciendo a mi familia, sobre todo a mi abuela gracias por todo lo que has hecho y haces por nosotros.

Mi director de trabajo de fin de grado me ha movido este agradecimientos al final porque no podía estar antes de su familia y amigos. Quiero dar las gracias a Pablo, mi tutor de proyecto, por todo ese tiempo que me has dedicado cuando te he pedido y más aún por todo aquel tiempo que no te he pedido. Gracias por estar ahí todo este tiempo, sobre todo en aquellas ocasiones en las que he estado perdido y has sabido guiarme. Este proyecto ha sido una experiencia inolvidable.

Índice general

Índice de figuras

Índice de tablas

Resumen y organización de la memoria

Resumen

En este trabajo se lleva a cabo el diseño y la implementación de un sistema de visión estéreo que permite la estimación de distancias a los objetos que se encuentran en la escena. El objetivo es hacer una primera aproximación a estas funcionalidades y tecnologías para servir de base a futuros trabajos de visión artificial en sistemas autónomos. Para ello se observan la precisión y velocidad de varios algoritmos para dos plataformas informáticas y se busca la mejor solución.

Se han comparado dos algoritmos para conocer su precisión en el cálculo de distancias para unos recursos computacionales determinados, es decir, la mejor relación entre completitud de mapa de disparidad y velocidad de ejecución. Se ha medido el error en situaciones de luz natural y en interiores. Asimismo, se ha analizado el rendimiento y paralelizado para aprovechar la limitada potencia de un ordenador de placa única (*Single*

Board Computer) muy utilizado, como la Raspberry Pi 2. Como referencia se ha tomado un PC.

El sistema ha sido construido mediante materiales de bajo coste: dos cámaras web, un PC y una Raspberry Pi 2. Todo el software ha sido desarrollado utilizando el lenguaje de programación C++ y utilizando la librería de visión por computador OpenCV. Además se ha paralelizado el software de búsqueda de correspondencia utilizando OpenMP.

El software desarrollado consta principalmente de una aplicación que nos permite el correcto calibrado del sistema estéreo y de diversas aplicaciones con distintos algoritmos de búsqueda de correspondencias, que se encargan de generar mapas de disparidad y de estimar la distancia a los distintos objetos clave.

Abstract

This project develops the design and implementation of a stereo vision system that estimates distances to the objects found at the scene. The purpose is to make a first approach to these features and technologies and provide a basis for future works on autonomous machine vision systems. To fulfill these goals, the accuracy and speed of several algorithms for two computing platforms are evaluated.

A comparison of two algorithms are made to know the precision of the distance estimation for a certain computational resources, meaning that, try to found the best ratio between completeness of disparity map and speed of execution. The error has been measured in situations of natural light and indoors. It has also been analyzed the performance of both systems and parallelized to take advantage of the limited power of a single board computer such as the Raspberry Pi 2. For reference has taken a PC.

The system has been built using low-cost components: two webcams, one PC and a one Raspberry Pi 2. All software has been developed using C++ programming language and the computer vision library OpenCV. In addition, the correspondence matching software has been parallelized using OpenMP.

The developed software consists mainly of an application that allow us the proper calibration of the stereo system and various applications with different correspondence matching algorithms, which are responsible of generating disparity maps and estimate the distance to the principal objects of the scene.

Organización

La primera parte de esta memoria la forma el capítulo en el que se presenta el alcance del proyecto, donde se hace una descripción del mismo, se enuncian los objetivos que debe

cumplir, se presenta detalladamente el estudio de viabilidad realizado, riesgos, etc.

La segunda parte hace un recorrido sobre la visión por computador, el modelo matemático de un sistema de visión por computador, teoría de la estereoscopia. Además, se plantea el desarrollo del software, enunciando las bases teóricas, los distintos algoritmos y las herramientas utilizadas para poder desarrollarlo.

La tercera parte describe el trabajo desarrollado. Para ello se expone el método de calibración utilizado, un estudio de los dos algoritmos de correspondencia utilizados y los métodos elegidos para estimar distancias. Asimismo, se hacen distintas pruebas de rendimiento del software desarrollado para ver su viabilidad en una aplicación a tiempo real.

Por último, en la sexta parte se encuentra la desviación sufrida respecto a la planificación inicial del proyecto, se elaboran las conclusiones y se plantea el trabajo futuro.

Como información adicional, se incluyen en los anexos el manual de usuario, el pliego de condiciones, 3 apéndices técnicos (la actualización de Raspbian Wheezy a Jessie, la instalación de ROS en Raspberry Pi 2 y la paralelización del software) y la licencia del documento.

Parte I

Alcance del proyecto

Capítulo

1 Descripción, Motivación y Objetivos del proyecto

1.1. Descripción

En este trabajo se desarrolla un sistema de estereoscopia utilizando recursos materiales de bajo coste, principalmente dos cámaras web y una Raspberry Pi 2. Este sistema permite realizar una comparación entre varios algoritmos de búsqueda de correspondencias, encontrar las ventajas de cada uno de los dos métodos de estimación de distancia y recoger información de las pruebas de rendimiento realizadas con el software desarrollado en versión serie y paralela. Estas pruebas de rendimiento han sido probadas tanto en una plataforma x86 (Intel Core i5 750) y en ARM (ARMv7 Cortex-A7) con el objetivo de evaluar las posibilidades de un sistem similar en el mundo de La Internet de las Cosas (IoT) o en el de los vehículos no tripulados (drones).

En resumen y como resultado global se obtiene *una panorámica*, un análisis de las posibilidades que proporciona un sistema sencillo y de bajo coste para futuros proyectos

y trabajos.

Para el funcionamiento del sistema estereoscópico se ha desarrollado diversos programas de los que se destacan:

- Software de calibración: con este software se obtienen los parámetros intrínsecos y extrínsecos del sistema mediante un patrón de calibración. Los parámetros se guardarán en un fichero que podrá ser utilizado para rectificar las cámaras en otros programas.

- Cálculos de disparidad y estimación de distancia: se han desarrollado dos programas con dos algoritmos distintos que calculan disparidades. Con estos programas y un buen calibrado de cámaras obtenido del software anterior, podremos obtener una estimación de la distancia a los objetos que componen la escena.

1.2. Objetivos

El objetivo principal de este proyecto es analizar las diferentes posibilidades que nos proporciona un sistema de visión artificial de bajo coste. Con dos cámaras web y una SBC (*Single Board Computer*, Ordenador de Placa Única) se puede conseguir información fiable de una escena tridimensional a partir de información bidimensional utilizando para ello técnicas y algoritmos de visión estéreo.

Como el software va a ser ejecutado en un SBC, un sistema embebido ARM (Raspberry Pi 2) cuya potencia es bastante menor comparada a la de un equipo de escritorio, se requiere que el software este lo más optimizado posible. Por ello en el trabajo siempre se analizan los métodos para obtener la mejor relación entre información fiable y rendimiento.

Este trabajo tiene también como objetivo que pueda ser usado como base para otros proyectos sobre visión por computador. De hecho, las herramientas usadas en este proyecto son todas libres (C++, OpenCV, OpenMP, LaTeX, GNU/Linux, Raspberry Pi).

1.3. Motivación

Empecé a hacer este proyecto a propuesta del director del mismo, Pablo González Nalda. La idea inicial era implementar este proyecto en paralelo a otro proyecto que él mismo está desarrollando junto con otros profesores del centro. Este proyecto se comenzó a desarrollar debido a la curiosidad que sentía el desarrollador sobre el mundo de la visión artificial y la reconstrucción de espacios en tres dimensiones mediante un sistema de visión estereoscópica. Además este proyecto surge del auge que se vive actualmente con los drones tripulados y sobre todo los no tripulados.

Este trabajo se encuadra en las dos primeras fases de las que componen la Inteligencia Artificial en la robótica clásica, las correspondientes a la adquisición de información del entorno y la creación de un modelo. Posteriormente se crea un plan de actuación y se lleva a cabo. Estas fases se resumen en la expresión SMPA (Sense-Model-Plan-Act). En un primer análisis, antes de conocer técnicas de visión estéreo, se pensó en usar un sensor capaz de obtener información de la escena. Se pensó en crear un telémetro como candidato a sensor (Sense), pero debido a su poca utilidad, ya que solo obtendría información de un único punto, se desechó la idea por otra, que consistía en una cámara en conjunto con un sensor 3D. Con la información conseguida del sensor, se crearía un modelo de mundo (Model) utilizando para ello la librería OpenCV. Una vez creado un modelo sólo quedaría planificar y actuar. Para ello el director del proyecto sugirió utilizar ROS (Robot Operating System) que nos ayudaría a cumplir estas dos ultimas partes (Plan-Act) del paradigma SMPA y unirlas a las demás gracias a su modularidad para transmitir información entre diferentes módulos del sistema.

En un segundo análisis y conforme el desarrollador iba entrando en materia se aprendieron técnicas y algoritmos de visión por computador y estéreo. Debido a que la universidad no disponía en esos momentos de un sensor 3D, se optó por descartar esta idea, utilizando como sustituto dos cámaras que formarían el sistema estereoscópico usado.

Una de las grandes motivaciones de este trabajo es hacer un desarrollo de software libre, utilizando para ello el lenguaje de programación C++. Debido a que el lenguaje al que estaba acostumbrado el desarrollador era JAVA, el aprendizaje de C++ no fue muy complicado. Al usarse CMake como *build system* los ficheros fuentes pueden ser compilados en múltiples sistemas operativos y el sistema es muy portable.

Capítulo

2 Viabilidad

En este capítulo se analizan los requisitos del trabajo para cumplir los objetivos y se define una planificación de los recursos humanos, de tiempo y económicos. Además se estudian los riesgos que pueden dificultar, retrasar o impedir la realización de los objetivos.

2.1. Requisitos funcionales del trabajo

En los análisis previos al trabajo se definió una serie de requisitos que éste debía cumplir:

- Detección de objetos que se encuentren frente a las cámaras
- Estimación de la distancia a los objetos detectados
- Tiene que poder ejecutarse tanto en arquitectura ARM como en x86

- El trabajo se implementará de forma modular en ROS

2.2. Planificación del tiempo

La planificación del tiempo del proyecto se realiza descomponiendo el trabajo en fases, tareas y entregables.

2.2.1. Estructura de descomposición del trabajo

La EDT estructura de manera jerárquica los distintos entregables y tareas que componen el trabajo. El propósito de la EDT es organizar y definir el alcance total del proyecto.

Por lo tanto, la planificación del proyecto se establece según las figuras 2.1 y 2.2.

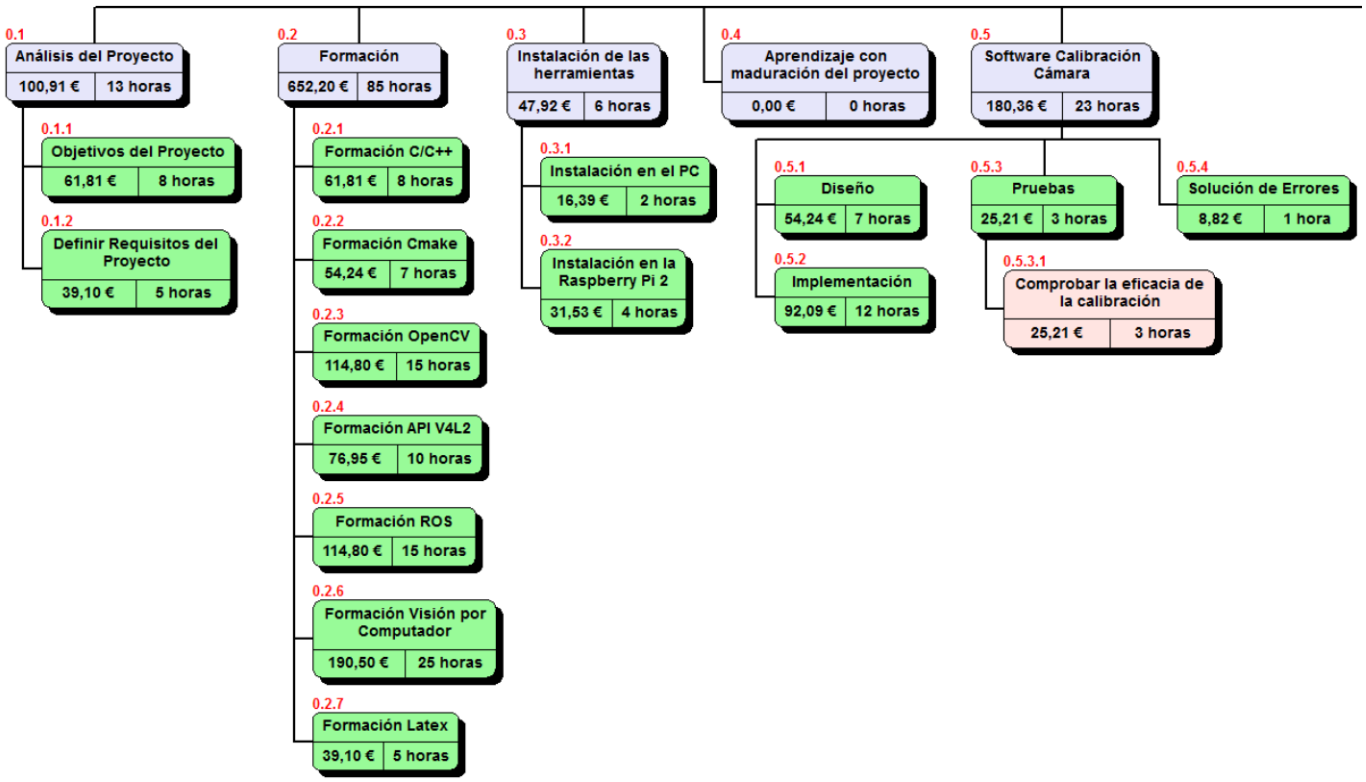


Figura 2.1: Estructura de Descomposición del Trabajo 1

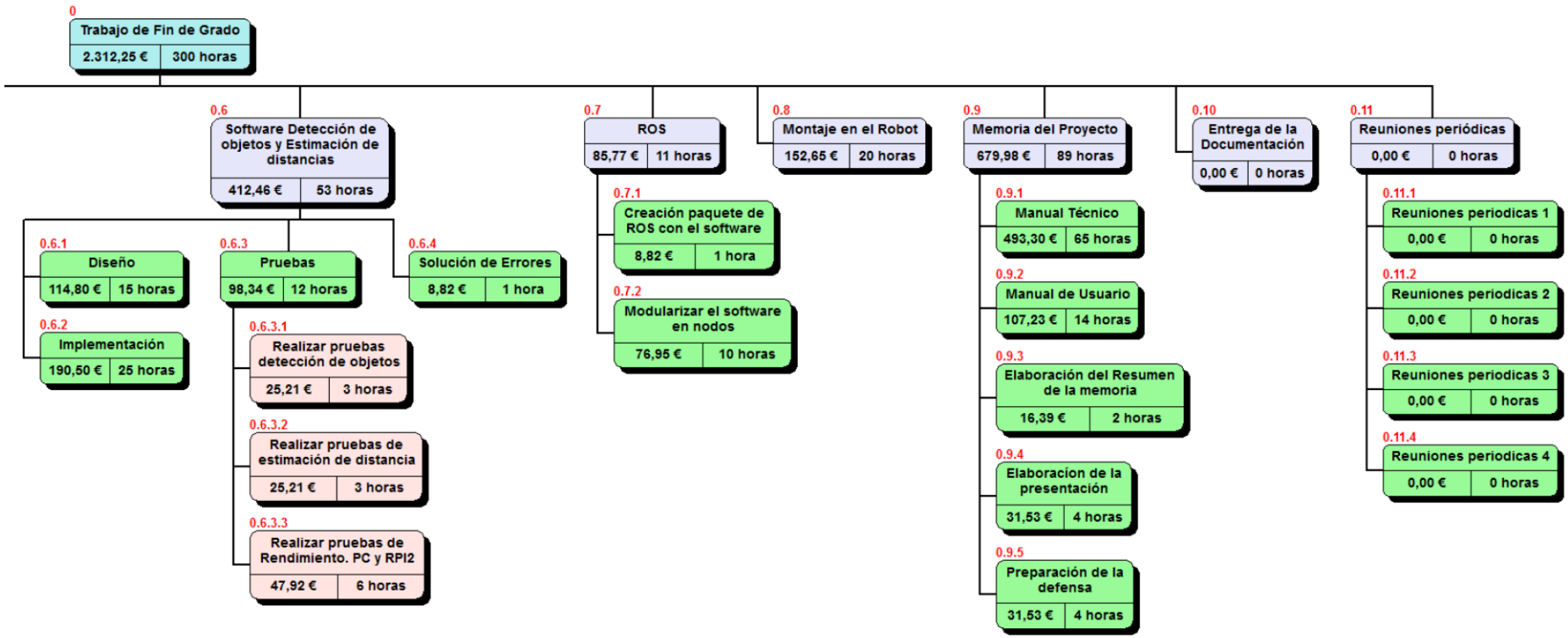


Figura 2.2: Estructura de Descomposición del Trabajo 2

2.2.2. Fases, tareas y entregables

En esta sección se muestran y describen las fases, tareas y entregables planificados en la realización del proyecto.

A continuación se muestra el listado de tareas que dispone el trabajo:

0. **Trabajo de fin de grado**
   - 0.1. **Análisis del Proyecto**
     - 0.1.1. Objetivos del proyecto
     - 0.1.2. Definir requisitos del proyecto
   - 0.2. **Formación**
     - 0.2.1. Formación C/C++
     - 0.2.2. Formación CMake
     - 0.2.3. Formación OpenCV
     - 0.2.4. Formación API V4L2
     - 0.2.5. Formación ROS
     - 0.2.6. Formación visión por computador
     - 0.2.7. Formación Latex
   - 0.3. **Instalación de las herramientas**
     - 0.3.1. Instalación en el PC
     - 0.3.2. Instalación en la Raspberry Pi 2
   - 0.4. Aprendizaje con maduración del proyecto
   - 0.5. **Software calibración cámara**
     - 0.5.1. Diseño
     - 0.5.2. Implementación
     - 0.5.3. **Pruebas**
       - 0.5.3.1. Comprobar eficacia de la calibración
     - 0.5.4. Solución de errores
   - 0.6. **Software detección de objetos y estimación de distancias**
     - 0.6.1. Diseño
     - 0.6.2. Implementación
     - 0.6.3. **Pruebas**
       - 0.6.3.1. Realizar pruebas detección de objetos
       - 0.6.3.2. Realizar pruebas de estimación de distancia

0.6.3.3. Realizar pruebas de rendimiento. PC y RPI2

0.6.4. Solución de errores

0.7. **ROS**

0.7.1. Creación paquete de ROS con el software

0.7.2. Modularizar el software en nodos

0.8. Montaje en el robot

0.9. **Memoria de proyecto**

0.9.1. Manual técnico

0.9.2. Manual de usuario

0.9.3. Elaboración del resumen de la memoria

0.9.4. Elaboración de la presentación

0.9.5. Preparación de la defensa

0.10. Entrega de la documentación

0.11. **Reuniones periódicas**

0.11.1. Reuniones periódicas 1

0.11.2. Reuniones periódicas 2

0.11.3. Reuniones periódicas 3

0.11.4. Reuniones periódicas 4

A partir de aquí, se explicará brevemente en qué consiste cada tarea:

Número: 0.1.1.

Nombre: Objetivos de proyecto.

Descripción: Hacer un análisis previo para determinar los objetivos que se intentarán alcanzar con este proyecto.

Trabajo estimado: 8 horas.

Número: 0.1.2.

Nombre: Definir requisitos del proyecto.

Descripción: Establecer los requisitos funcionales que abordará el proyecto.

Trabajo estimado: 5 horas.

Número: 0.2.1.

Nombre: Formación C/C++.

Descripción: Adquirir los conocimientos necesarios para utilizar el lenguaje C/C++.

Trabajo estimado: 8 horas.

Número: 0.2.2.

Nombre: Formación CMake.

Descripción: Realizar un estudio exhaustivo de la herramienta CMake para poder compilar el software.

Trabajo estimado: 7 horas.

Número: 0.2.3.

Nombre: Formación OpenCV.

Descripción: Realizar un estudio exhaustivo de la librería OpenCV necesaria para el desarrollo del trabajo.

Trabajo estimado: 15 horas.

Número: 0.2.4.

Nombre: Formación API V4L2.

Descripción: Familiarizarse con la API V4L2 necesaria para el manejo de las cámaras en Linux.

Trabajo estimado: 10 horas.

Número: 0.2.5.

Nombre: Formación ROS.

Descripción: Aprender a utilizar el framework que se utilizará para implementar el software en el Robot.

Trabajo estimado: 15 horas.

Número: 0.2.6.

Nombre: Formación visión por computador.

Descripción: Formación básica en técnicas de visión por computador.

Trabajo estimado: 25 horas.

Número: 0.2.7.

Nombre: Formación Latex.

Descripción: Familiarizarse con Latex para escribir la memoria.

Trabajo estimado: 5 horas.

Número: 0.3.1.

Nombre: Instalación en el PC.

Descripción: Instalación de las herramientas necesarias para el desarrollo del proyecto en el PC.

Trabajo estimado: 2 horas.

Número: 0.3.2.

Nombre: Instalación en la Raspberry Pi 2.

Descripción: Instalación de las herramientas necesarias para el desarrollo del proyecto en la Raspberry Pi 2.

Trabajo estimado: 4 horas.

Número: 0.4.

Nombre: Aprendizaje con maduración del proyecto.

Descripción: Como no se sabe exactamente como van a ser abordadas las siguientes tareas, se crea esta. Pues lo mas seguro, es que mas adelante surgirán nuevas tareas.

Trabajo estimado: 0 horas.

Número: 0.5.1.

Nombre: Diseño.

Descripción: Diseño del software necesario para calibrar la cámara.

Trabajo estimado: 7 horas.

Número: 0.5.2.

Nombre: Implementación.

Descripción: Implementación del software encargado de calibrar la cámara.

Trabajo estimado: 12 horas.

Número: 0.5.3.1.

Nombre: Comprobar la eficacia de la calibración.

Descripción: Se realizarán pruebas para determinar la eficacia de la calibración.

Trabajo estimado: 3 horas.

Número: 0.5.4.

Nombre: Solución de errores.

Descripción: Se solucionaran los errores que surjan en el software de calibración.

Trabajo estimado: 1 hora.

Número: 0.6.1.

Nombre: Diseño.

Descripción: Diseño del software encargado de detectar los objetos y estimar las distancias.

Trabajo estimado: 15 horas.

Número: 0.6.2.

Nombre: Implementación.

Descripción: Implementación del software encargado de detectar los objetos y estimar las distancias.

Trabajo estimado: 25 horas.

Número: 0.6.3.1.

Nombre: Realizar pruebas detección de objetos.

Descripción: Se realizarán pruebas para determinar la función de detección de objetos.

Trabajo estimado: 3 horas.

Número: 0.6.3.2.

Nombre: Realizar pruebas de estimación de distancia.

Descripción: Se realizarán pruebas que evaluarán el error de la detección de objetos.

Trabajo estimado: 3 horas.

Número: 0.6.3.3.

Nombre: Realizar pruebas de rendimiento. PC y RPI2.

Descripción: Se evaluará el rendimiento del programa tanto en el PC, como en la Raspberry Pi 2.

Trabajo estimado: 6 horas.

Número: 0.6.4.

Nombre: Solución de errores.

Descripción: Corrección de errores que surjan en el software de detección de objetos y estimación de distancias.

Trabajo estimado: 1 hora.

Número: 0.7.1.

Nombre: Creación paquete de ROS con el software.

Descripción: Se creará un paquete para ROS con el software desarrollado.

Trabajo estimado: 1 hora.

Número: 0.7.2.

Nombre: Modularizar el software en nodos.

Descripción: Se separará en módulos las funcionalidades del software desarrollado, con la idea de intentar utilizarlo de forma distribuida.

Trabajo estimado: 10 horas.

Número: 0.8.

Nombre: Montaje en el robot.

Descripción: Montaje del trabajo en el robot.

Trabajo estimado: 20 horas.

Número: 0.9.1.

Nombre: Manual técnico.

Descripción: Redactar un manual que explique el proceso seguido para el desarrollo del proyecto.

Trabajo estimado: 65 horas.

Número: 0.9.2.

Nombre: Manual de usuario.

Descripción: Redactar un manual detallado que explique al usuario como utilizar el software de este trabajo.

Trabajo estimado: 14 horas.

Número: 0.9.3.

Nombre: Elaboración del resumen de la memoria.

Descripción: Elaborar el resumen necesario de la memoria.

Trabajo estimado: 2 horas.

Número: 0.9.4.

Nombre: Elaboración de la presentación.

Descripción: Elaborar las diapositivas necesarias para la defensa del proyecto.

Trabajo estimado: 4 horas.

Número: 0.9.5.

Nombre: Preparación de la defensa.

Descripción: Realizar ensayos previos a la presentación para conseguir una buena exposición.

Trabajo estimado: 4 horas.

Número: 0.10.

Nombre: Entrega de la documentación.

Descripción: Hito.

Trabajo estimado: 0 horas.

Número: 0.11.1 - 0.11.4.

Nombre: Reuniones periódicas.

Descripción: Reuniones con el director del proyecto cada primer lunes de cada mes.

Trabajo estimado: 3 horas por reunión.

2.2.3. Entregables

Al finalizar el proyecto se entregará el software desarrollado junto con un manual técnico y de uso que explicará de forma sencilla y concisa la compilación y utilización del trabajo.

2.2.4. Agenda del proyecto

Se trabajará 4 horas cada día de Lunes a Viernes y se seguirá el calendario laboral 2.1 oficial del BOE para Álava-Araba. De esta forma se estima que el trabajo se realizará desde el día Lunes 19 de enero de 2015 al Martes 12 de mayo de 2015. Esta planificación

Fecha	**E**vento
1 de Enero	Año nuevo
6 de Enero	Día de Reyes
19 de Marzo	San José
2 de Abril	Jueves Santo
3 de Abril	Viernes Santo
6 de Abril	Lunes de Pascua
28 de Abril	San Prudencio
1 de Mayo	Día del trabajo
25 de Julio	Santiago Apostol
5 de Agosto	Virgen Blanca
15 de Agosto	Asunción de la Virgen
12 de Octubre	Fiesta Nacional de España
8 de Diciembre	Inmaculada Concepción
25 de Diciembre	Navidad

Tabla 2.1: Calendario de días festivos

es totalmente compatible con las clases de universidad a las que tiene que acudir el desarrollador. Por otra parte, se ha dejado margen hasta el Martes 1 de Septiembre en caso de requerir más tiempo del previsto para finalizar el trabajo.

Se ha estimado lo siguiente:

- Trabajo total estimado: 300 horas
- Duración total estimada: 300 horas
- Costo estimado: 2.938,50 €

Puede verse desglosado el costo del proyecto en la sección 2.3.

2.2.5. Cronograma

El diagrama de Gantt 2.3 y 2.4 expone el tiempo de dedicación previsto para las diferentes tareas a lo largo del tiempo total determinado para el proyecto.

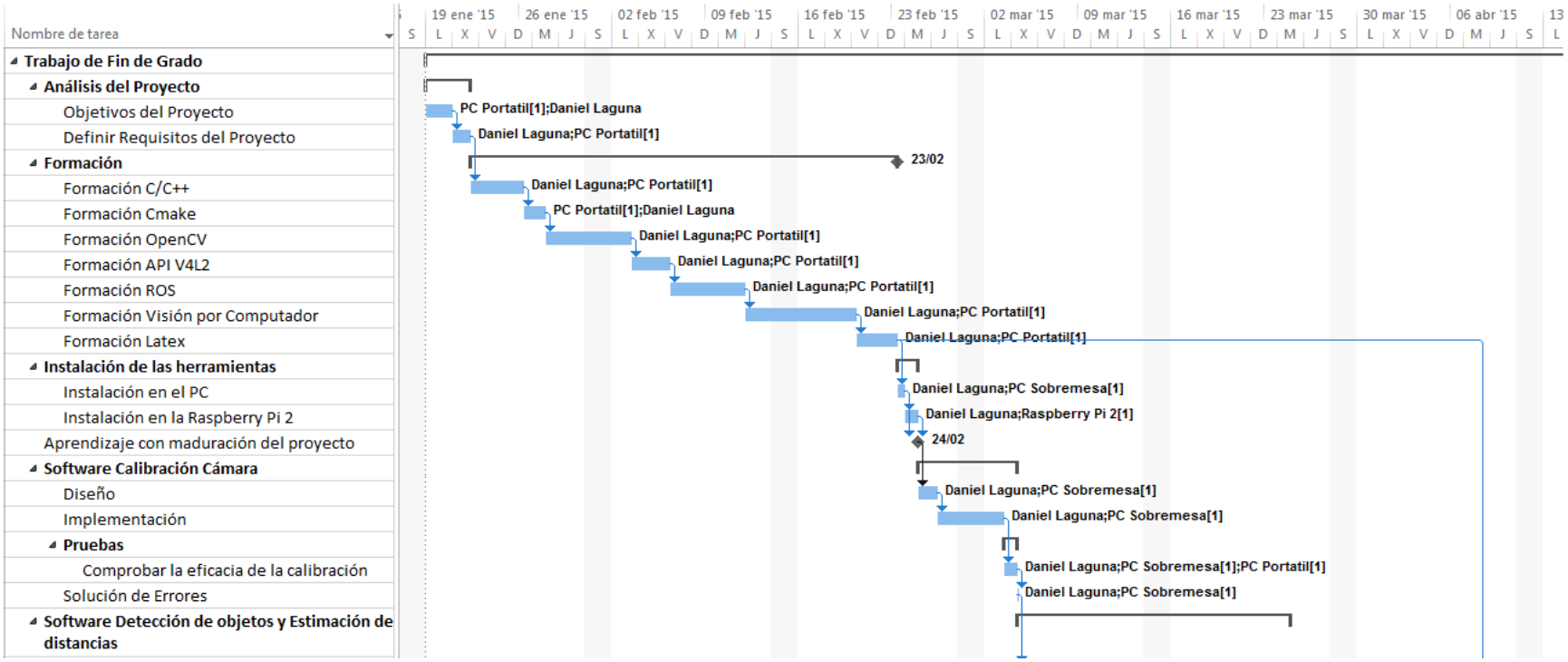


Figura 2.3: Diagrama de Gantt 1

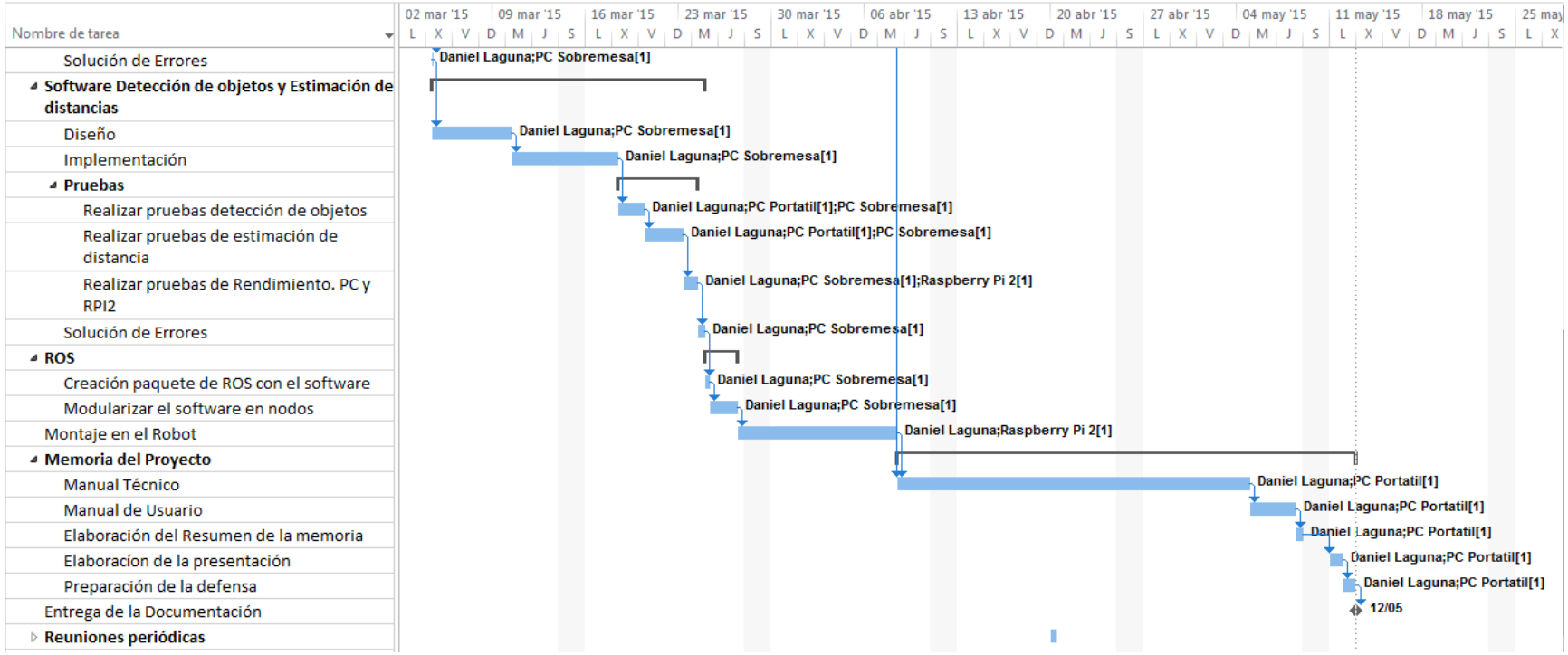


Figura 2.4: Diagrama de Gantt 2

2.3. Gestión de costos

A continuación se muestran los recursos materiales y humanos que se utilizarán en la realización del trabajo en la figura 2.5.

Nombre del recurso	Tipo	Etiqueta de material	Iniciales	Grupo	Capacidad máxima	Tasa estándar	Tasa horas extra	Costo/Uso	Acumu	Calendario base
Daniel Laguna	Trabajo		D		100%	17,57 €/hora	0,00 €/hora	0,00 €	Prorrateo	Calendario EUI
PC Sobremesa	Material		PCS			1,25 €		0,00 €	Prorrateo	
PC Portatil	Material		PCP			1,25 €		0,00 €	Prorrateo	
Raspberry Pi 2	Material		RPI			1,00 €		0,00 €	Prorrateo	
Logitech C310 1	Material		L			1,00 €		0,00 €	Prorrateo	
Logitech C310 2	Material		L			1,00 €		0,00 €	Prorrateo	

Figura 2.5: Recursos humanos y materiales

2.3.1. Presupuesto

Para realizar la estimación del presupuesto se han tenido en cuenta los datos que aparecen de la tabla 2.2 a 2.8. En ellas se indica lo que cobran los recursos humanos, que tienen una tasa estándar por hora correspondiente a la de un analista programador. Los recursos materiales se consideran con un coste por uso de 1,25 € en concepto de luz ADSL/Fibra óptica y otros gastos que se cobran independiente del tiempo de uso excepto la Raspberry Pi 2 y las camaras que tendrán un coste de 1 €.

Los datos de tiempo y trabajo han sido extraídos de las siguientes vistas de Microsoft Project. En la realización de este presupuesto se tiene en cuenta tanto los recursos materiales como las licencias de software necesarias para el desarrollo del trabajo.

Para el cálculo de las amortizaciones se ha considerado un tiempo de amortización de 3 años. El cálculo del coste unitario de amortización es la división entre el coste unitario y el tiempo de amortización. Se considera el tiempo de amortización como $3\ años\ \times\ 200\ días\ laborables\ \times\ 8\ horas = 4800\ horas$.

Concepto	**C**oste
Daniel Laguna	17,57 €/h

Tabla 2.2: Coste de recursos trabajo

Concepto	**C**oste
PC Sobremesa	550,20 €
PC Portátil	512,71 €
Raspberry Pi 2	39,99 €
Logitech C310	40,00 €
Logitech C310	40,00 €
Tabla de madera	1,50 €
Cinta americana	1,00 €

Tabla 2.3: Coste recursos materiales (hardware)

Concepto	**C**oste Unitario	**N**umero de Licencias
Windows 7 Professional N	125,00 €	1
Microsoft Project 2013	1.369,00 €	1
Chart Pro	0,00 €	1
Manjaro Linux	0,00 €	1
Arch Linux	0,00 €	1
Vim	0,00 €	1
Raspbian	0,00 €	1
Libreria OpenCV	0,00 €	1
ROS	0,00 €	1
Gimp	0,00 €	1

Tabla 2.4: Recursos materiales (software)

Concepto	**T**rabajo (h)	**T**rabajo horas extra	**C**oste	**C**oste horas extra	**I**mporte
Daniel Laguna	300	0	17,57 €/h	0	5271,00 €
Total	300	0	17,57 €/h	0	5271,00 €

Tabla 2.5: Costo recursos trabajo

Concepto	**U**nidades	**C**oste	**I**mporte
PC Sobremesa	1	1,25 €/uso	16,25 €
PC Portatil	1	1,25 €/uso	21,25 €
Raspberry Pi 2	1	1,00 €/uso	3,00 €
Logitech C310	2	1,00 €/uso	12,00€
Total			52,50 €

Tabla 2.6: Costo recursos materiales

Concepto	**C**oste unitario	**T**iempo de amortización	**C**oste unitario de amortización	**T**iempo de uso	**I**mporte
PC Sobremesa	550,20 €	4800 horas	0,114625 €	89 h	10,20 €
PC Portatil	512,51 €	4800 horas	0,106773 €	196 h	20,93 €
Raspberry Pi 2	39,99 €	4800 horas	0,008331 €	30 h	0,25 €
Logitech C310	40,00 €	4800 horas	0,008333 €	60 h	0,50 €
Logitech C310	40,00 €	4800 horas	0,008333 €	60 h	0,50 €
Windows 7 Professional N	125,00 €	4800 horas	0,026042 €	25 h	0,65 €
Microsoft Project 2013	1.369,00 €	4800 horas	0,285208 €	16 h	4,56 €
Chart Pro	0,00 €	4800 horas	0,000000 €	10 h	0,00 €
Manjaro Linux	0,00 €	4800 horas	0,000000 €	130 h	0,00 €
Arch Linux	0,00 €	4800 horas	0,000000 €	130 h	0,00 €
Vim	0,00 €	4800 horas	0,000000 €	60 h	0,00 €
Raspbian	0,00 €	4800 horas	0,000000 €	30 h	0,00 €
Librería OpenCV	0,00 €	4800 horas	0,000000 €	50 h	0,00 €
ROS	0,00 €	4800 horas	0,000000 €	15 h	0,00 €
Gimp	0,00 €	4800 horas	0,000000 €	3 h	0,00 €
Total					37,59 €

Tabla 2.7: Amortizaciones de hardware y de software

Concepto	**I**mporte
R.T.	5271,00 €
R.M.	52,50 €
Costo Fijo	0,00 €
Amortización	37,59 €
SUMA	
Gastos generales (10 %)	536,11 €
Beneficio (15 %)	804,16 €
SUBTOTAL	
IVA (21 %)	1407,29 €
TOTAL	8108,65 €

Tabla 2.8: Total presupuesto

Por tanto el presupuesto asciende a ocho mil ciento ocho con sesenta y cinco euros (8108,65 €)..

Firma: Daniel Laguna Soto

Vitoria-Gasteiz, a 22 de Septiembre de 2015

2.4. Gestión de riesgos

En este apartado se hace un análisis que permite la identificación de las amenazas que pueden perjudicar el desarrollo del trabajo. Para ello se han evaluado una serie de riesgos y se ha valorado la gravedad del impacto en el trabajo.

Riesgo	**P**eligrosidad
Dedicación no exclusiva al trabajo	Media
Cambios/Ampliación de requisitos	Media
Estancamiento en la codificación	Media
Problema con recursos materiales	Media
Perdida de Información	Alta
Personal poco experimentado	Alta
Enfermedades	Alta
Elección equivocada de herramientas, algoritmos o métodos	Alta
Planificación muy optimista	Alta

Tabla 2.9: Riesgos

2.4.1. Explicación de los riesgos y plan de contingencia

A continuación se describe con mas detalle los riesgos de la tabla 2.9.

- Dedicación no exclusiva al trabajo
  - *Descripción:* Por alguna circunstancia, podría darse que el desarrollador no pudiese dedicarse exclusivamente al trabajo.
  - *Probabilidad:* 20 %
  - *Peligrosidad:* Media, retrasaría el trabajo.
  - *Medidas preventivas:* No previsible.
  - *Medidas correctoras:* Se elevará el numero de horas al día dedicadas al trabajo para eliminar el retraso. En el peor de los casos, se podría optar por atrasar la entrega del trabajo.
- Cambios/Ampliación de requisitos
  - *Descripción:* Es posible que debido a los resultados obtenidos al realizar alguna tarea, estos no sean los esperado y tenga que cambiarse o añadirse un requisito funcional.
  - *Probabilidad:* 30 %
  - *Peligrosidad:* Media.
  - *Medidas preventivas:* No previsible.

  - *Medidas correctoras:* Se retrasaría el trabajo.

- Estancamiento en la codificación

  - *Descripción:* Podría llegar a darse algún problema de difícil solución que paralizaría momentáneamente el avance del proyecto.
  - *Probabilidad:* 47 %
  - *Peligrosidad:* Media.
  - *Medidas preventivas:* Se estudiará con detalle las herramientas que manejara el desarrollador.
  - *Medidas correctoras:* Replanificar las tareas posteriores y dedicar mas formación a las herramientas.

- Problema con recursos materiales

  - *Descripción:* Los materiales utilizados pueden no superar el rendimiento necesario para realizar la aplicación e incluso sufrir algún percance que los deje inservibles.
  - *Probabilidad:* 15 %
  - *Peligrosidad:* Media.
  - *Medidas preventivas:* Realizar un mantenimiento de los materiales y tener dispositivos que puedan usarse como alternativa en caso de emergencia.
  - *Medidas correctoras:* Buscar una solución ya sea reparando los materiales o adquiriendo unos nuevos.

- Perdida de información

  - *Descripción:* Debido a un fallo en el disco duro en el que está almacenado el trabajo.
  - *Probabilidad:* 13 %
  - *Peligrosidad:* Alta, perder los datos supone empezar de cero.
  - *Medidas preventivas:* Se crearán copias de seguridad del proyecto cada Viernes en otro dispositivo de almacenamiento.
  - *Medidas correctoras:* Se intentará recuperar la información de los dispositivos de almacenamiento.

- Personal poco experimentado

  - *Descripción:* Como el trabajo trata distintas materias en las que el desarrollador no tiene ninguna experiencia previa, es posible que se de este riesgo.

- *Probabilidad:* 70 %
- *Peligrosidad:* Media.
- *Medidas preventivas:* Se estudiará con detalle todas las materias involucradas en el desarrollo del trabajo.
- *Medidas correctoras:* Se estudiará con mas detalle las distintas materias.

- Enfermedades
  - *Descripción:* Podría suceder por causa de enfermedad o accidente.
  - *Probabilidad:* 5 %
  - *Peligrosidad:* Alta.
  - *Medidas preventivas:* No previsible.
  - *Medidas correctoras:* Dependiendo de la gravedad del problema se podría continuar con el trabajo. Si no, este se retrasaría.
- Elección equivocada de herramientas, algoritmos o métodos
  - *Descripción:* Es posible que con alguna de las herramientas escogidas no sea posible cumplir los objetivos planteados.
  - *Probabilidad:* 50 %
  - *Peligrosidad:* Alta.
  - *Medidas preventivas:* Se estudiarán las diferentes tecnologías y se tendrán planteadas alternativas.
  - *Medidas correctoras:* Cambiar a las alternativas en el menor tiempo posible.
- Planificación muy optimista
  - *Descripción:* Una mala planificación podría acarrear retrasos en el desarrollo del trabajo.
  - *Probabilidad:* 20 %
  - *Peligrosidad:* Alta.
  - *Medidas preventivas:* Se realizará una planificación pesimista para evitarlo.
  - *Medidas correctoras:* Modificar la planificación de forma que la duración del trabajo se alargue.

Parte II

Conceptos generales, tecnologías y herramientas

Capítulo

3 Conceptos generales

En este capítulo se recoge información elemental sobre el modelo de un sistema de visión artificial, teoría básica de estereoscopia, distintos algoritmos de búsqueda de correspondencia y varias herramientas. La finalidad de este capitulo es que el lector se familiarice con los conceptos y herramientas utilizadas en el desarrollo del software de este trabajo.

3.1. Visión por computador

La visión por computador es un campo de la inteligencia artificial cuyo objetivo es conseguir datos clave sobre la escena por medio de un ordenador. Uno de sus usos como análisis de una escena puede ser encontrar distintas fallas visuales en un producto.

3.2. Modelo de un sistema de visión por computador

En esta sección se va a mostrar cómo funciona un sistema de visión y el modelo matemático con el cuál se calcula la función de transferencia de la escena hacia el plano imagen.

3.2.1. Modelo *Pinhole*

La cámara Pinhole o cámara oscura consiste en un centro óptico C, que es el punto de la lente tal que cualquier rayo de luz que pasa por él no sufre desviación, y un plano R sobre el que va a ser proyectada la imagen. El plano imagen se va a ubicar a una distancia focal f del centro óptico y perpendicular al eje óptico Z.

El plano imagen R se encuentra en $Z = f$ y no es más que un plano imaginario donde se forma la imagen captada sin inversión alguna, al contrario de lo que ocurre en el sensor de la cámara.

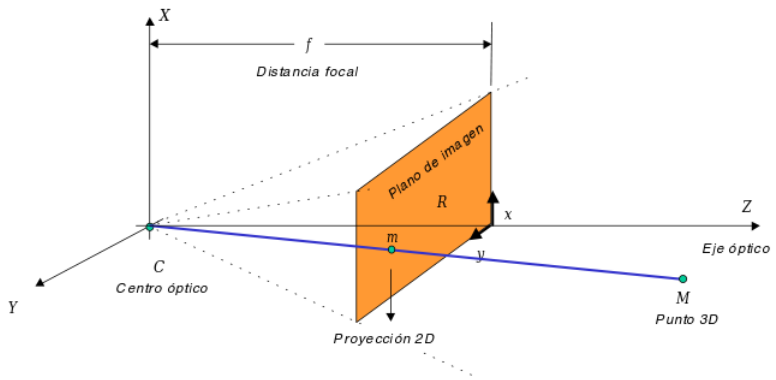


Figura 3.1: Modelo geométrico de una cámara Pinhole [9]

Un punto en la escena M será proyectado en en el plano imagen como m, de tal forma que m es el resultado de la intersección de la recta $\overline{CM}$ y el plano R:

$$m = \overline{CM} \cap R \tag{3.1}$$

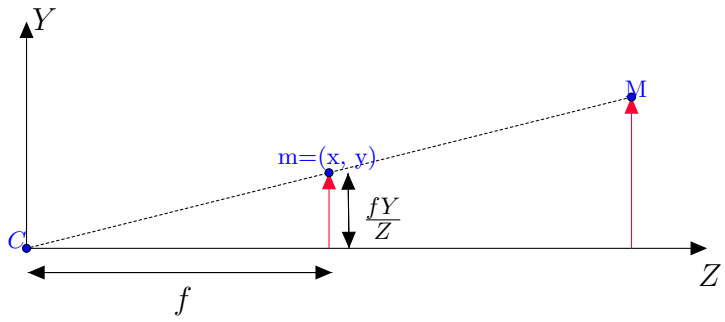


Figura 3.2: Modelo Pinhole visto perpendicular a la dirección del eje Y

Si nombramos a las coordenadas de $M = (X,Y,Z)^T$ y $m = (x,y)^T$ y observando la geometría de la figura 3.2, se cumple una relación de semejanza de triángulos, tanto para Y como para X, tal que:

$$\frac{y}{f} = \frac{Y}{Z} \qquad\qquad \frac{x}{f} = \frac{X}{Z}$$

$$\begin{cases} x = \frac{fX}{Z} \\ y = \frac{fY}{Z} \end{cases} \tag{3.2}$$

o bien en coordenadas homogéneas:

$$Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.3}$$

Según este modelo, un punto M se transforma en un punto m según la relación

$$\lambda m = PM \tag{3.4}$$

donde P es la matriz 3×4 llamada *matriz de proyección*, mientras que λ es un factor de escala.

La matriz de proyección está formada por la matriz 3×3 K que es la matriz de la cámara y un vector de valores nulos.

$$P = \begin{bmatrix} K & 0 \end{bmatrix} \tag{3.5}$$

De momento supondremos que la transformación de perspectiva se hace con una conversión 1/1. Se verá más adelante que con las cámaras CCD esto no ocurre, pues necesitamos de un factor de escala.

Hasta ahora, hemos asumido que:

1. El origen de coordenadas del espacio 3D coincide con el centro óptico.
2. El eje óptico coincide con el eje Z de este espacio.
3. El origen de coordenadas del plano imagen coincide con la intersección de Z con R. Esta intersección se conoce como el punto principal c de la imagen.

¿Pero queé pasa si estas condiciones no se dan? Imaginemos un nuevo sistema de coordenadas (X', Y', Z') como los de la figura 3.3. Las dos primeras condiciones anteriores no se cumplen, ya que su origen no coincide con el centro óptico y el eje Z' no coincide con el eje óptico Z.

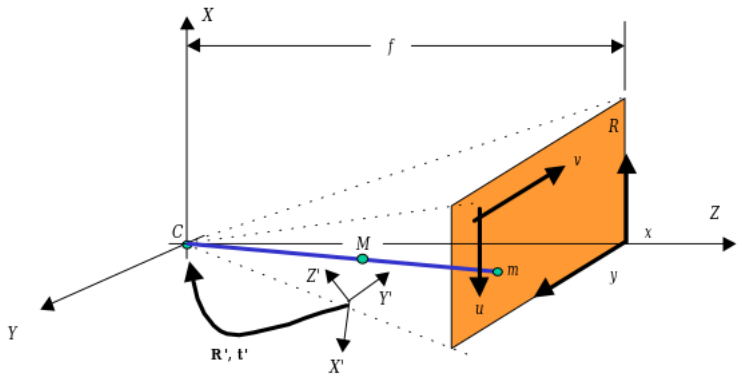


Figura 3.3: Modelo de una proyección con rotación de ejes [9]

Para conseguir que los ejes coincidan se debe hacer una transformación sobre un espacio Euclideo en $\mathbb{R}^3$, ya que las únicas transformaciones implicadas son la rotación y la traslación del objeto.

Suponiendo que la rotación de los ejes X', Y' y Z' sobre los ejes X, Y y Z está definida por la matriz ortonormal 3×3 $\mathbf{R}$ y que la traslación aplicada al eje está definida por el vector 3×1 $\mathbf{t}$ se puede definir:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \tag{3.6}$$

o bien

$$M = SM' \tag{3.7}$$

Obtenemos que la proyección de un punto M' en la imagen es dado por:

$$\lambda m = KSM' \tag{3.8}$$

3.2.2. Cámara CCD

La cámara CCD (*charge-couple-device*) está formada por elementos semiconductores fotosensibles dispuestos en una matriz, de forma que cada receptor se traduce en un píxel de la imagen.

En un sensor CCD la luz se va a transformar en energía eléctrica que será digitalizada por un conversor analógico-digital.

En la sección 3.2.1 se mencionó que hasta ese momento habíamos asumido ciertas cosas, y una de ellas era:

El origen de coordenadas del plano imagen, coincide con la intersección de Z con R. Esta intersección se conoce como el punto principal c de la imagen.

Con una cámara CCD no podemos asumirlo, debido a que las coordenadas proyectadas x, y en la sección anterior son transformadas en un nuevo sistema de coordenadas u, v debido a lo siguiente:

Cambio de escala: Para medir la escena utilizamos unidades métricas, como por ejemplo milímetros, mientras que la imagen será medida en pixels. Por eso, en la transformación $(x, y) \to (u, v)$ se necesita de un factor de escala. Además es posible que el factor de escala no sea el mismo para cada eje, ya que los píxeles no son cuadrados, sino rectangulares. Los factores que vamos a utilizar son α_x y α_y, normalmente expresados en [pixel/mm].

Traslación del origen: Como es prácticamente imposible que el eje óptico intersecciones con el punto central del CCD, denotaremos (u_0, v_0) como el nuevo punto principal dentro del nuevo sistema de coordenadas. Es decir (u_0, v_0) corresponden al punto $x = 0$, $y = 0$, lo que viene a ser el desplazamiento del centro de coordenadas del CCD respecto al punto principal, tal como se muestra en la figura 3.4.

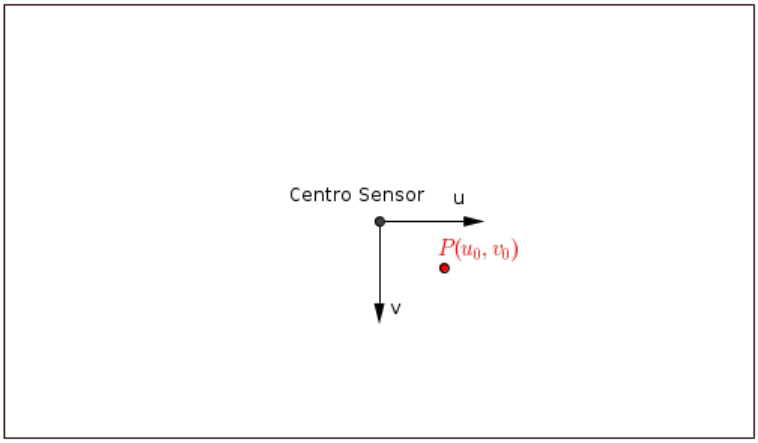


Figura 3.4: Desplazamiento del punto principal respecto al centro del sensor

Factor de torcimiento: Es posible que los ejes u, v no sean ortogonales, debido a que los pixels del CCD son rectangulares. En casos como éste es necesario introducir un factor de torcimiento (*skew factor*) s. En la gran mayoría de las cámaras s es cero.

De tal forma que la transformación de coordenadas $(x, y) \to (u, v)$ puede definirse de la siguiente forma:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.9}$$

podremos definir entonces:

$$w = Km \tag{3.10}$$

con $w = (u, v, 1)^T$, $m = (x, y, 1)^T$ y K la matriz calibración de la cámara.

De esta forma la proyección $(X', Y', Z') \to (u, v)$ se puede escribir de la siguiente forma.

$$\lambda w = KSM' \tag{3.11}$$

Si nos fijamos el modelo consta de 12 parámetros, pero si hacemos la multiplicación

$$\begin{bmatrix} \alpha_x & s & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha'_x & s' & u_0 \\ 0 & \alpha'_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

reducimos el modelo a 11 parámetros, que se pueden descomponer en 5 intrínsecos $(\alpha'_x, \alpha'_y, u_0, v_0, s')$ y 6 extrínsecos $(t_X, t_Y, t_Z, w_X, w_Y, w_Z)$.

3.2.3. Calibración de la cámara

La calibración es el método mediante el que podemos estimar los parámetros intrínsecos y extrínsecos de la cámara. Estos parámetros definen la forma en la que la cámara capta la escena y el modo en que ésta se ve afectada por imperfecciones en el hardware (lentes, sensores, etc). Sabiendo estos parámetros es posible corregir estas alteraciones vistas en la captura de la imagen.

Existen varios métodos de calibración. Se ha elegido el método propuesto por Zhengyou Zhang [17] que se describirá más adelante.

3.3. Estereoscopia

La estereoscopia es una técnica utilizada para la recuperación de las coordenadas tridimensionales de una escena a partir de dos o más imágenes tomadas desde distinta perspectiva.

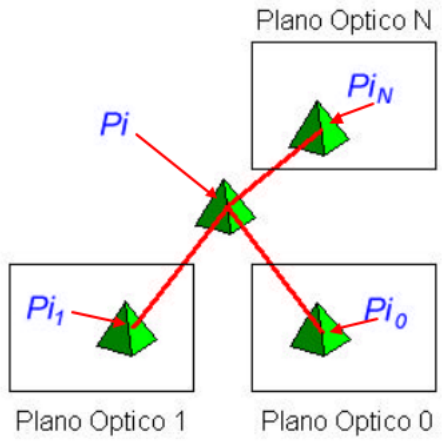


Figura 3.5: Proyección del punto P_i en la vista de cada cámara [14]

Dadas N imágenes, un objeto en la escena y un punto M perteneciente a este objeto, el proceso consiste en resolver dos problemas principales:

1. Encontrar el conjunto de puntos m_n correspondientes a la proyección del punto M en los planos imagen de cada cámara. Este proceso es llamado como *búsqueda de correspondencias* o *matching*.

2. Conocido el conjunto m_n, calcular las coordenadas tridimensionales de M.

Para resolver estos problemas, el proceso estereoscópico consta de varias etapas.

3.3.1. Adquisición de imágenes

Un sistema estéreo consta de dos o varias cámaras. La adquisición de las imágenes varía dependiendo del número de éstas, su resolución o la posición entre ellas, ya que distintos *baselines* (distancia entre los centros ópticos de las cámaras) altera el proceso estereoscópico. En este trabajo se hace uso de un sistema binocular al igual que en el sistema de visión humano.

3.3.2. Modelo de las cámaras

En el capítulo anterior se explicó el modelo matemático de una cámara. Por medio de la calibración podíamos estimar los parámetros intrínsecos y extrínsecos que definían como captaba la escena la cámara. En el modelo estéreo, ademas tendremos un componente que relacionará las coordenadas de ambas cámaras.

La etapa de calibración es crucial ya que nos permite simplificar el proceso de matching y obtener una representación tridimensional. La etapa de calibración es explicada con más detalle en el capítulo 5.2.

3.3.2.1. Geometría epipolar

La geometría epipolar se define en la figura 3.6. Las proyecciones del punto M en sus respectivos planos son m_1 y m_2. Sólo con la proyección m_1 no somos capaces de saber dónde se encuentra M, porque al proyectar perdemos la profundidad. Sin embargo, podemos afirmar que M se encuentra en la recta $\overline{C_1 m_1}$.

Si queremos obtener la ubicación de m_2 debemos proyectar en el otro plano los posibles puntos correspondientes a la proyección de m_1. Se puede observar que, de todos los puntos de la recta proyectados, uno de ellos es m_2. Aun así no podemos averiguar la ubicación exacta de m_2 por medio de m_1, sólo podemos afirmar que m_2 pertenece a la proyección de la recta $\overline{C_1 m_1}$ realizada en este segundo plano óptico. Esta recta se llama linea epipolar. También es posible definir las lineas epipolares como la intersección entre los planos ópticos y el plano formado por M, C_1 y C_2. Como se puede observar también hay dos epípolos e_1 e_2, los epípolos están contenidos en todas las rectas epipolares.

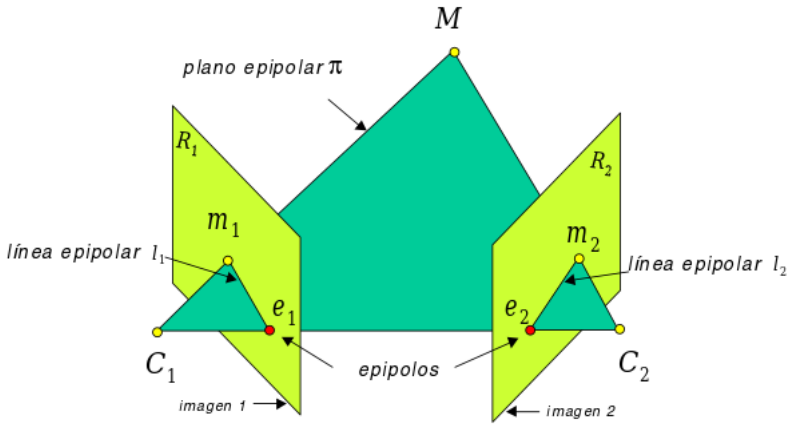


Figura 3.6: Rectas epipolares y epípolos [9]

La restricción epipolar señala que para que dos puntos m_1 y m_2 sean correspondientes, el punto m_2 debe estar en la linea epipolar de m_1 o al contrario. Esto es una condición necesaria pero no suficiente, pues no todos los puntos de la linea epipolar de m_1 son correspondientes a éste. Aunque no lo parezca esto es de gran utilidad, pues simplificará mucho el problema de matching o búsqueda de correspondencias. Esto es debido a que, en vez de buscar en toda la imagen (dos dimensiones), se buscará en una única recta (una dimensión). Es decir, si tenemos una imagen de tamaño $N \times N$ simplifica un problema de tamaño N^2 en uno de tamaño N.

De esta forma, si los ejes ópticos tienen la misma dirección y son paralelos, m_1 y m_2 de coordenadas (u, v) compartirán una coordenada, la v, haciendo que la linea epipolar sea una única recta paralela a la recta que une los centros ópticos, es decir el baseline, y que los epípolos se encuentren en el infinito. Solo diferirán en la coordenada u, esta diferencia es llamada disparidad.

3.3.2.2. Definición de disparidad y triangulación

La disparidad no es más que la diferencia de la coordenada u entre las dos imágenes, suponiendo que conocemos los valores intrínsecos de las cámaras, que estos son iguales en ambas, que sus ejes ópticos son paralelos y despreciamos las distorsiones que puedan causar las lentes. Observando la figura 3.7 podemos calcular las siguientes ecuaciones mediante semejanza de triángulos:

$$\frac{Z}{f} = \frac{x}{u_1} \qquad\qquad \frac{Z}{f} = \frac{x - B}{u_2}$$

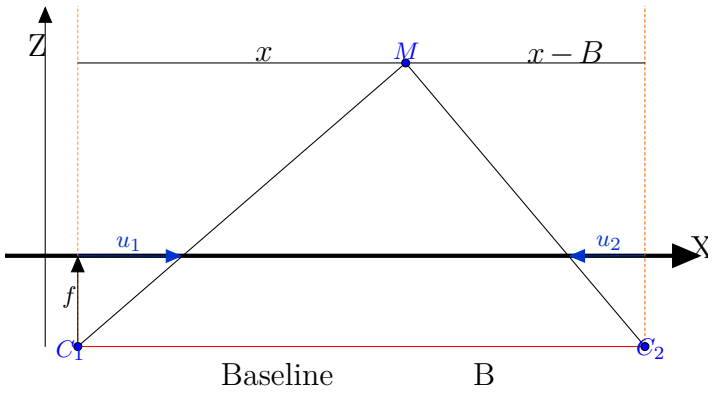


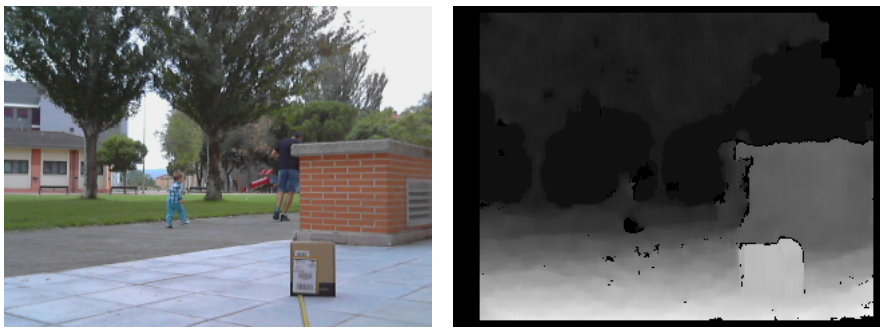
Figura 3.7: Vista de Planta de un sistema binocular

Y a partir de estas semejanzas podremos calcular la coordenada Z mediante

$$Z = \frac{Bf}{u_1 - u_2} = \frac{Bf}{d} \tag{3.13}$$

Como se puede apreciar, la disparidad será inversamente proporcional a la profundidad. Además la disparidad también varía si modificamos el baseline o la distancia focal. De esta forma podemos tener resultados distintos con múltiples baselines o distancias focales.

Suponiendo que el sistema ha sido adecuadamente calibrado y las imágenes han sido rectificadas e proceso de búsqueda de correspondencias nos devolverá como resultado un mapa de disparidad. Un mapa de disparidad es una imagen en escala de grises que representa la disparidad de cada punto en la escena. Como se puede ver en la figura 3.8 los puntos de la escena más lejanos serán representados con colores negros o grisáceos, por ser los de menos disparidad, mientras que los cercanos serán representados con tonos blancos, por ser los de más disparidad.



(a) Escena (b) Mapa de disparidad

Figura 3.8: Mapa de disparidad tomado en el exterior (Algoritmo BM)

3.3.3. Elección de características

Las características usadas tienen que ser extraibles de las imágenes.

- **Píxeles:** Es la unidad elemental de una imagen, y contiene información lumínica de los puntos a comparar.

- **Bordes:** Pueden tener atributos geométricos de forma (curvas, líneas, etc), orientación e intensidad. Pueden ser un problema y es que a veces las cámaras no ven las mismas partes del objeto.

- **Regiones:** Se pueden usar regiones de la imagen como características a comparar. La dificultad de usar regiones se debe a la dificultad de como segmentar la imagen.

3.3.4. Búsqueda de correspondencias o *matching*

Ésta es la tarea principal y a la vez la más complicada de la estereoscopia. El proceso de matching de las proyecciones de un punto en los distintos planos ópticos es complejo debido a los siguientes factores:

- **Variación de la intensidad luminosa:** la intensidad luminosa de un punto observado varía dependiendo de la posición del observador. En la figura 3.9 se aprecia que las capturas tomadas son de distinta intensidad lumínica.

- **Fenómeno de oclusión:** un punto de la escena visible en una imagen no tiene por que ser visible en la otra. En la figura 3.9 se aprecia como en la captura derecha se ve el objeto de detras de la caja, mientras que en la captura izquierda la caja lo tapa.

- **Textura de los objetos:** algunos objetos, pueden presentar texturas que provocan un aumento de la ambigüedad entre puntos. Por ejemplo una textura con patrones periódicos (como una pared de ladrillos) aumenta el número de puntos similares, mientras que un objeto de textura lisa o de escasa textura, puede dificultar la búsqueda del punto correspondiente ya que todos los puntos del objeto son parecidos. Esto provoca múltiples correspondencias falsas.

(a) Captura izquierda (b) Captura derecha

Figura 3.9: Capturas donde se aprecia el problema de la intensidad luminosa y del fenómeno de oclusión

Para minimizar estos efectos podemos aplicar algunas restricciones:

- **Restricción epipolar:** ya ha sido mencionada anteriormente, y como se dijo, nos permite reducir el problema de búsqueda en dos dimensiones en una.
- **Restricción de oclusión:** basándonos en el fenómeno de oclusión y mediante una comparación de ambas imágenes podemos descartar aquellos puntos que no tienen ninguna correspondencia.

3.3.5. Determinación de la profundidad

Una vez que las disparidades están calculadas, como se explicó anteriormente, podemos calcular las coordenadas tridimensionales mediante triangulación. Obtendremos buenos o malos valores dependiendo de lo confiables que sean los resultados de los procesos anteriores.

3.3.6. Interpolación

A veces es necesario aplicar una interpolación debido a que en nuestro mapa de disparidad múltiples puntos no encontraron correspondencia o encontraron una errónea. Por eso mediante la interpolación se consigue llenar vacíos o corregir errores.

3.4. Algoritmos propuestos

En esta sección se habla a modo de resumen, de algunos de los algoritmos que han sido utilizados y tratados en este trabajo.

3.4.1. Correspondencia densa local. BM

Algoritmo local que sólo busca correspondencias para cada píxel utilizando para ello píxeles vecinos contenidos dentro de una ventana. El proceso consiste en comparar las intensidades de varias ventanas, con otra que será la del píxel a buscar correspondencia. Entre éstas se elegirá la que tenga intensidades similares. De este tipo se ha elegido el algoritmo Block Matching (BM) por ser uno de los que implementa OpenCV.

3.4.2. Correspondencia densa global. Graph-Cuts

Los algoritmos de correspondencia global buscan correspondencias entre intensidades de píxeles a nivel de imagen completa. Para ello se crean áreas de píxeles con el mismo nivel de similaridad. De esta forma, la correspondencia de un píxel afecta a sus vecinos. Dado un número de áreas de disparidad, se etiquetan todos los píxeles intentando que la diferencia de las disparidades de píxeles en una misma etiqueta sean mínimas, y máximas entre las distintas, en un numero infinito de iteraciones. Es decir, trata de minimizar una función a nivel de imagen completa. OpenCV cuenta con una implementación de un algoritmo de este tipo llamado Graph-cuts, en este trabajo no ha sido utilizado debido a que el tiempo necesario para calcular un mapa de disparidad es muy alto, e imposible de implementar en una aplicación que requiera un calculo en tiempo real.

3.4.3. Correspondencia densa híbrida. SGBM

Estos algoritmos son un punto intermedio entre los locales y globales. Su objetivo es tratar de acercarse a la precisión de los globales sin sacrificar la eficiencia de los locales.

Al igual que los locales, dividen la imagen en ventanas pero (al buscar correspondencias) no buscan similaridad entre píxeles, sino que tratan de minimizar una función a nivel de ventana. OpenCV implementa el algoritmo SGBM, del que se ha hecho uso en este trabajo.

Capítulo

4 Tecnologías y herramientas

4.1. OpenCV

Es una librería de visión por computador de código libre bajo una licencia BSD. Proporciona un compendio de más de 2500 algoritmos ya optimizados, como el de reconocimiento facial, identificación de objetos, seguimiento de objetos en movimiento o extracción de modelos en 3D a partir de información 2D, etc... Son tantas sus funciones que esta librería es un estándar utilizado en ámbitos como el tratamiento de imágenes o la inteligencia artificial.

La librería esté escrita en C++, C, Python, Java y MATLAB y es compatible con Windows, GNU/Linux, Android y Mac OS. Muchas de sus funciones pueden ser utilizadas para aplicaciones en tiempo real y se está desarrollando una versión compatible con CUDA y OpenCL para acelerar el cómputo en el hardware de las tarjetas gráficas.

4.2. Robot Operating System

ROS es un *framework* para el desarrollo de software para robots que es mantenido por la Open Source Robotics Foundation (OSRF). Podría definirse como un meta sistema operativo, pues proporciona servicios de los que dispone un sistema operativo, incluyendo abstracción del hardware, control de dispositivos a bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y gestión de paquetes.

4.2.1. Características importantes de ROS

ROS se ejecuta como una red de procesos (nodos) *peer-to-peer* de tal forma que cada nodo pueda recibir mensajes de estado de los demás. Cada nodo se agrupa en paquetes que pueden ser compartidos en otros sistemas ROS. Además es independiente del lenguaje, pues puede ser implementado en cualquier lenguaje de programación moderno, estando implementado ya en Python, C++ y Lisp, mientras que tambien existen librerías experimentales en Java y Lua.

ROS es escalable, pues gracias a que se ejecuta de forma distribuida es posible disponer de una única instancia de ROS en varios dispositivos.

Por otra parte es totalmente compatible con OpenCV, lo que nos posibilita implementar los programas desarrollados en ROS.

Principalmente ROS sólo puede ser ejecutado en plataformas basadas en Unix, siendo las principales Ubuntu y Mac OS X, mientras que gracias a la comunidad también esta adaptada a otros sistemas operativos como Debian, Arch Linux, Gentoo y otras plataformas que usen el kernel Linux, consideradas como experimentales.

4.2.2. Conceptos básicos de ROS

ROS esta basado en un arquitectura de grafos que procesa los datos en conjunto. Los principales elementos de esta red de grafos son nodos, master, servidor de parametros, mensajes, servicios, tópicos y bolsas, todos ellos proporcionan datos a la red de distintas maneras.

- **Nodos:** Cada nodo dentro de la red es un proceso que realiza una tarea. ROS está diseñado para ser modular, con lo que de esta forma podemos tener varios nodos ejecutando distintas tareas. Esta modularidad nos proporciona muchas posibilidades y siguiendo en la línea del proyecto podríamos utilizar un nodo para capturar el entorno, otro encargado de crear el mapa de disparidad, otro que estime distancias, uno que tome decisiones respecto a los datos obtenidos, etc...

- **Maestro:** El ROS Master es el nodo maestro que proporciona un registro de nombres de tal forma que nos permita hacer una búsqueda dentro del grafo de computación, de forma parecida a un servidor DNS. Sin el nodo maestro los distintos nodos del grafo no podrían encontrarse entre ellos, intercambiar mensajes o invocar servicios.

- **Servidor de parámetros:** El servidor de parámetros nos permite guardar datos identificados por una clave. El servidor de parámetros forma parte del ROS Master.

- **Mensajes:** Los mensajes son estructuras de datos, compuestos de distintos campos tipados. Dentro de estos campos se utilizan distintos tipos primitivos como enteros, booleanos, float, etc... Algunos mensajes pueden incluir distintas estructuras o arrays anidados. Los nodos se comunican entre ellos intercambiando mensajes.

- **Tópicos:** ROS cuenta con un modelo de comunicación publicador/suscriptor. De esta forma un nodo manda un mensaje y lo publica en un tópico, entonces un nodo interesado en ese tópico se suscribirá y recibirá los mensajes publicados por el otro nodo. Esto nos proporciona una abstracción entre la producción y el consumo de la información pues puede haber varios nodos publicadores y varios suscriptores y ninguno tiene por que saber de la existencia de los otros.

- **Servicios:** Los nodos para comunicarse también cuentan con un modelo Cliente/-Servidor. De esto se encargan los servicios encargados de servir la tarea para la que se les invoca. Estos están definidos por un par de mensajes: uno para la petición y otro para la respuesta. De esta forma un nodo ofrece un servicio y otro nodo lo utiliza enviando una petición y esperando una respuesta.

- **Bolsas:** Las bolsas son un almacén de datos. Nos permiten guardar y reproducir distintos tipos de datos que podemos utilizar posteriormente en distintos nodos.

4.3. Raspberry Pi 2

La Raspberry Pi 2 es la segunda versión de la plataforma embebida de mismo nombre. Es un ordenador completo de bajo consumo en una única placa. Esta trae varias mejoras respecto a su anterior modelo, de las cuales destacaremos:

- Un procesador ARMv7 Cortex-A7 de 4 núcleos a 900MHz
- 1GB de memoria RAM

La Raspberry Pi 2 consume poca energía y al ser de tamaño y precio reducido la hacen perfecta para un montón de tareas sin tener que afrontar un gran desembolso.

Se puede instalar diversos sistemas operativos de los que predominan los basados en Unix. Muchas de estas distribuciones están destinadas a diversos propósitos, en este proyecto se ha utilizado *Raspbian*, una distribución basada en Debian wheezy con soporte para coma flotante.

Las imágenes de Raspbian pueden encontrarse fácilmente en la pagina web oficial de Raspberry, para ser escritas en una tarjeta de memoria. Por tanto, para cambiar de sistema operativo sólo es necesario extraer una memoria microSD e insertar otra.

Como en Raspbian wheezy aparece en los repositorios una versión antigua de OpenCV a la que se ha utilizado, en el apéndice C se ha escrito una guía para actualizar a Raspbian jessie. También en el apéndice D se encuentra una guía sobre como instalar ROS en Raspbian.

Parte III

Desarrollo del proyecto

Capítulo

5 Sistema estereoscópico

En este capítulo y en los siguientes se encuentra el trabajo realizado en el proyecto, comprendiendo desde la construcción del sistema estéreo, el calibrado del mismo, los distintos algoritmos de correspondencia utilizados y los distintos resultados de pruebas, tanto de estimación de distancia como de rendimiento. La finalidad del apartado es que el lector entienda los conceptos clave y sea capaz de replicar el mismo desarrollo realizado en este proyecto de forma sencilla.

5.1. Construcción del sistema estereoscópico

Para la construcción del sistema estereoscópico se buscaron distintas cámaras, entre ellas se destacó la Logitech C270 por varios motivos:

1. La capacidad de grabar imágenes de un tamaño de 1280x720 a 30 FPS son más que

adecuadas para el objetivo del proyecto.

2. La distancia focal es fija y además es posible variarla modificando fácilmente la cámara. Importante, puesto que si cambiase en medio del proceso, los datos conseguidos de la calibración dejarían de ser validos para el sistema.
3. Dispone de un micrófono, que podría ser útil y puede ser convertida en cámara infrarroja.
4. Es compatible con GNU/Linux a través de V4L2 (Video for Linux v.2), siendo este el sistema en el que ha sido desarrollada la aplicación. Aunque como ya se ha comentado, los fuentes son compilados con CMake, por tanto son validos también para otros sistemas.
5. El precio es bastante reducido, pues es posible encontrar cada cámara alrededor de un coste de 20€.

Finalmente no se utilizaron estas cámaras, sino una versión mayor, la C310 debido a que se disponía de ellas en el departamento y así no era necesario hacer este desembolso. La C310 mejora a la C270, en varias características y pueden ser encontradas con un coste de 40€.

El sistema cuenta con una distancia de baseline de un total de 7,5 cm y para fijar las cámaras se ha utilizado la estructura que se ve en las figuras 5.1 y 5.2.

Se han fijado las cámaras a una tabla de madera con cinta americana de tal modo que éstas no consigan moverse. Esta estructura es lo bastante estable para no moverse siempre que no haya algún elemento externo al sistema que las mueva. Siendo suficiente para poder desarrollar el trabajo siguiente. Sin embargo, esta estructura podría no servir si se montase el sistema en un vehículo, por lo tanto habría que reforzarlo. Para ello se podrían sujetar ambas cámaras con tornillos y hacer unas hendiduras en las que encajar las cámaras.

Figura 5.1: Sistema visto de frente

Figura 5.2: Sistema visto desde arriba

El sistema cuenta con una distancia de baseline de un total de 7,5 cm y para fijar las cámaras se ha utilizado la estructura que se ve en las figuras 5.1 y 5.2.

Se han fijado las cámaras a una tabla de madera con cinta americana de tal modo que éstas no consigan moverse. Esta estructura es lo bastante estable para no moverse siempre que no haya algún elemento externo al sistema que las mueva. Siendo suficiente para poder desarrollar el trabajo siguiente. Sin embargo, esta estructura podría no servir si se montase el sistema en un vehículo, por lo tanto habría que reforzarlo. Para ello se podrían sujetar ambas cámaras con tornillos y hacer unas hendiduras en las que encajar las cámaras.

5.2. Calibrado y Rectificado del sistema estereoscópico

En este apartado se va a explicar el calibrado del sistema. El calibrado es necesario para obtener los parámetros intrínsecos y extrínsecos del sistema para poder rectificar las imágenes capturadas por el par estéreo y calcular un buen mapa de disparidad.

Hay varios métodos para el calibrado de cámaras. En este trabajo, las cámaras han sido calibradas utilizando OpenCV y el algoritmo de calibración desarrollado por Zhengyou Zhang.

Se ha elegido este método por varias razones:

- Fácil de implementar utilizando OpenCV.
- Sencillo de ejecutar, ya que únicamente se debe hacer fotos de un objeto de calibración.

- Su validez es tan buena como la de cualquier otro método que devuelva buenos resultados.

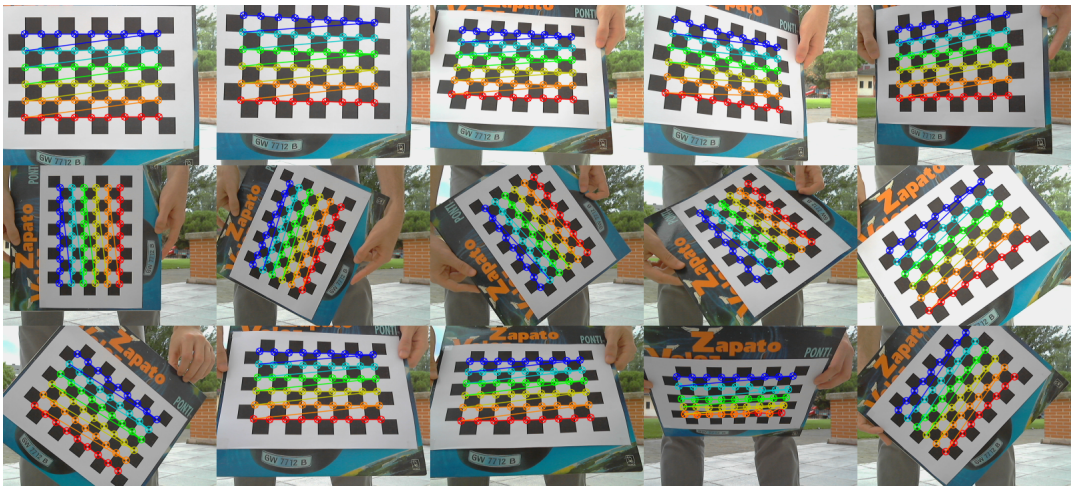


Figura 5.3: Patrones de calibración

En este método utilizamos un objeto de calibración del que conocemos las coordenadas tridimensionales exactas. Así, para cada par de puntos correspondiente identificado, conoceremos la posición tridimensional. De este modo, como paso previo necesitamos de un objeto de calibración. Se ha utilizado para ello un tablero de ajedrez, que es un objeto que sigue un patrón de zonas blancas y negras fácilmente diferenciables, lo que permite que el algoritmo identifique las esquinas interiores de éste de forma precisa y trabaje eficazmente. En la figura 5.3 puede verse una serie de 15 capturas realizadas con un damero en el proceso de calibración.

El tablero utilizado es de 9×6 esquinas interiores, y el tamaño de cada cuadrado es de $2,5$ cm. A continuación se explicará el proceso seguido para el cálculo de los parámetros intrínsecos y extrínsecos mediante el siguiente diagrama de flujo.

Los pasos del algoritmo son estos:

1. Se asigna a cada esquina interior una coordenada tridimensional. Empezando por la primera a la izquierda, se tomará esta como origen de coordenadas de tal forma que crearemos una lista de coordenadas $(0,0,0)$, $(25,0,0)$, $(50,0,0)$ $\ldots$ $(200,125,0)$

2. Se buscan las esquinas interiores del tablero, en ambas cámaras. Para ello utilizaremos el método:

```
bool findChessboardCorners( image, patternSize, corners,
    CALIB_CB_ADAPTIVE_THRESH|CALIB_CB_FILTER_QUADS )
```

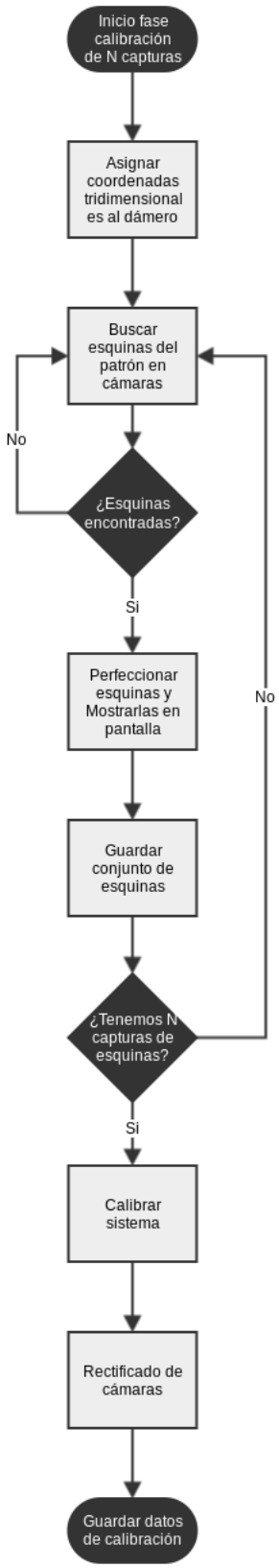


Figura 5.4: Diagrama de flujo del calibrado

Si las esquinas son encontradas devolverá *true*. El argumento *image*, es una matriz de entrada que contendrá la imagen del tablero sobre el que se quieren detectar las esquinas. *patternSize* es el numero de esquinas interiores a buscar, en nuestro caso *patternSize = Size(9,6)*. *corners* las esquinas detectadas. El flag *CALIB_CB_ADAPTIVE_THRESH* indica que se realizará una normalización previa de la imagen y el otro flag *CALIB_CB_FILTER_QUADS* hace que se añada un criterio adicional para descartar falsos positivos.

3. Si se han encontrado esquinas, se pasan las imágenes a escala de grises y se procesan, para hacer mas precisos los resultados con la función:

```
void cornerSubPix(image, corners, winSize, zeroZone, criteria)
```

4. Seguido de esto se mostrarán las esquinas encontradas en las imágenes de los tableros utilizando la función:

```
void drawChessboardCorners(image, patternSize, corners,
    patternWasFound)
```

   Que tiene como argumentos: *image* es la imagen que contiene el tablero. *patternSize* tiene el mismo significado que en el método `findChessBoardCorners()` y *corners* son las esquinas encontradas anteriormente.

5. Se guardarán las esquinas encontradas en un vector (Si se encontraron) y se repetirá la búsqueda, hasta que dispongamos de un numero de esquinas necesario para llevar a cabo la calibración.

6. Una vez se tengan una cantidad de esquinas necesarias se pasará a realizar el calibrado del sistema utilizando la función:

```
double stereoCalibrate(objectPoints, imagePoints1, imagePoints2, CM1,
    distCoeffs1, CM2, distCoeffs2, imageSize, R, T, E, F, TermCriteria(
    TermCriteria::COUNT+TermCriteria::EPS, 30, 1e-6),
    CV_CALIB_SAME_FOCAL_LENGTH | CV_CALIB_ZERO_TANGENT_DIST)
```

   El primer argumento *objectPoints* es el array que contiene las coordenadas tridimensionales de las esquinas del tablero (las que se asignaron al principio). Los argumentos *imagePoints1* y *imagePoints2* son arrays que contienen las coordenadas de las esquinas capturadas desde la cámara izquierda y derecha. *CM1* y *CM2* son las matrices que contienen los parámetros intrínsecos de las cámaras mientras que *distCoeffs1* y *distCoeffs2* son los coeficientes de distorsión de las lentes de las cámaras. En *imageSize* introduciremos la resolución de la imagen. *R, T, E, F* son las matrices de Rotación, Traslación, Esencial y Fundamental. Con *TermCriteria* definimos cuántas veces se ejecutará el método, ya sea porque ha alcanzado un numero máximo de iteraciones o porque ha minimizado el error por debajo de uno dado.

Con las banderas introducidas indicamos que las cámaras tienen la misma distancia focal y que la distorsión tangencial es despreciable.

7. Para rectificar las cámaras llamaremos a la función:

```
void stereoRectify(CM1,  distCoeffs1, CM2, distCoeffs2, imageSize, R,
   T, R1, R2, P1, P2, Q, CALIB_ZERO_DISPARITY, alpha, newImageSize,
   roi1, roi2)
```

Los primeros parámetros *CM1, distCoeffs1, CM2, distCoeffs2, imageSize, R* y *F* son los mismos que los obtenidos en el método `stereoCalibrate()`. *R1, R2, P1* y *P2* son las matrices de rotación y proyección de cada cámara. Con la matriz *Q* podremos proyectar puntos desde una imagen hacia la escena. En esta función solo hay un flag como argumento, *CV_CALIB_ZERO_DISPARITY* que hará que los puntos principales de cada cámara tengan las mismas coordenadas en las vistas rectificadas. *alpha* es un parámetro del tipo double para escalar la imagen, este va de 0 a 1, si es cero eliminará los puntos que no son validos y si es uno los conservará, si es un valor intermedio obtendremos resultados intermedios. Los últimos argumentos *newImageSize, roi1* y *roi2* son: la nueva resolución de la imagen después de hacer el rectificado y las regiones importantes de cada imagen, ya que dentro de ellas estarán los píxels validos.

8. Por ultimo calcularemos los mapas de pixeles (*map1x, map1y, map2x, map2y*) necesarios para transformar cualquier imagen tomada con el sistema, en una imagen rectificada y sin distorsión.

```
1 void initUndistortRectifyMap(CM1, distCoeffs1, R1, P1, size, CV_32FC1,
      map1x, map1y)
2 void initUndistortRectifyMap(CM2, distCoeffs2, R2, P2, size, CV_32FC1,
      map2x, map2y)
```

Para aplicar las transformaciones usaremos el método:

```
1 void remap(image1, imageU1, map1x, map1y, INTER_LINEAR)
2 void remap(image2, imageU2, map2x, map2y, INTER_LINEAR)
```

El flag *INTER_LINEAR* indica que la interpolación utilizada sera bilinear.

Una vez guardados los datos de la calibración podremos utilizarlos en nuestros programas. Es muy importante que las cámaras no se muevan, pues al moverse se descalibrarán y los parámetros calculados dejarán de servir. Aun así, antes de ejecutar cualquier otro programa es recomendable calibrar el sistema para evitarlo.

Figura 5.5: Capturas no rectificadas

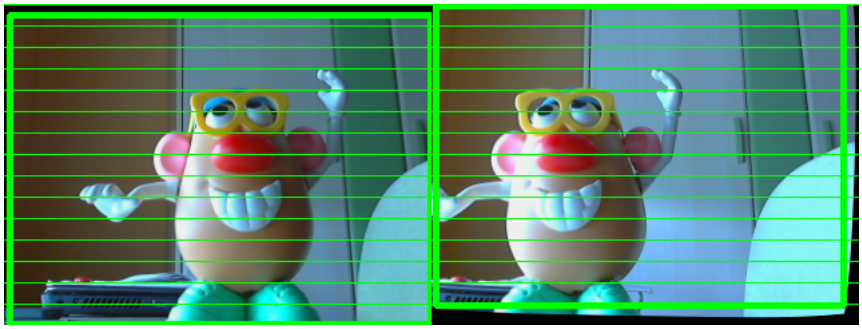


Figura 5.6: Capturas rectificadas

Se obtendrán resultados como los vistos en las figuras 5.5 y 5.6. Se observa claramente cómo en la primera figura las imágenes no están rectificadas pues los puntos de las lineas epipolares no coinciden. Mientras que en la segunda puede encontrarse correspondencia en los puntos siguiendo la línea epipolar. Puede verse claramente en la mano del muñeco.

Capítulo

6 Búsqueda de correspondencias y estimación de distancia

En este capítulo se aborda la búsqueda de correspondencias, el problema más complicado del proceso estereoscópico. Para ello en este capítulo se explican los dos algoritmos principales utilizados en el trabajo y se realiza una comparación entre ellos. También se aborda la estimación de distancia y se aportan resultados de distancias obtenidos con el sistema creado.

6.1. Algoritmos para la Búsqueda de Correspondencias

A la hora de buscar algoritmos de correspondencia se han probado algoritmos de correspondencia discreta como SURF, de correspondencia densa local como BM y de correspondencia densa híbrida como SGBM. Se ha destacado estos dos últimos, ya que son con los que mejores resultados se ha obtenido.

6.1.1. Algoritmo BM

El algoritmo *Block Matching* trabaja con ventanas de píxeles que utiliza para buscar correspondencias entre la imagen izquierda y derecha, y como se menciono anteriormente se basa en las intensidades de los píxeles para encontrar correspondencias. Este método obtiene buenos resultados en zonas con mucha textura, sin embargo en zonas sin textura puede dar como resultado que algunos píxeles se queden sin correspondencia.

El algoritmo BM implementado en OpenCV puede dividirse en 3 pasos:

1. **Pre-procesado:** Los píxeles de la imagen serán normalizados, de esta forma disminuirá la diferencia de iluminación entre ambas imágenes. Para normalizarlas se utilizarán ventanas de tamaño variable que por defecto son de tamaño 5×5. Para obtener la intensidad de un píxel en concreto le restaremos la media de la intensidad de los píxeles de su ventana y estableceremos unos valores máximos y mínimos que por defecto son de 30 y -30. De esta forma

$$I = min[max(I_c - \overline{I}, -I_{cap}), I_{cap}] \tag{6.1}$$

   donde I_c es el centro de la ventana, que coincide con el píxel sobre el que estamos haciendo la normalización, I_{cap} es el limite numérico que imponemos y $\overline{I}$ es la media de las intensidades de los píxeles de la ventana actual.

2. **Búsqueda:** Para ello se hace uso de SAD (Suma de Diferencias Absolutas) sobre diferentes ventanas de píxeles. Esta técnica es muy sencilla y rápida, SAD utiliza ventanas centradas en los píxeles sobre los que se van a calcular los costes. Para ello se elige una ventana en la primera imagen, y para cada píxel de la linea epipolar de la segunda imagen se crean ventanas del mismo tamaño que la primera y se comparan. Para comparar las ventanas, primero se suman los valores de intensidad de los píxeles de estas. Seguido de esto se restará el valor de la ventana de la primera imagen a las de las otras ventanas guardando el resultado de la diferencia en valor absoluto. Por ultimo, una vez procesada toda la linea epipolar, para determinar la correspondencia se tomará la ventana cuyo resultado sea el valor más cercano a 0.

   A la hora de hacer la búsqueda no se busca en toda la linea epipolar, sino que se define una región dentro de esta. Esta región se llama *horópter* y esta definida por dos parámetros limite mínimo de disparidad (punto de comienzo de la búsqueda) y número de disparidades (número de píxeles en los que se buscan correspondencias). La variación de estos parámetros implica que algunos puntos no encuentren correspondencias.

   Como se menciono anteriormente los puntos lejanos tienen valores de disparidad bajos, mientras que los puntos cercanos altos. De esta forma si variamos el nume-

ro de disparidades a un valor bajo, no encontraremos correspondencia para puntos cercanos. Si al contrario, lo aumentamos seremos más capaces de encontrar correspondencia para puntos cercanos pero con el inconveniente de que habrá que hacer más comparaciones y por tanto el gasto computacional aumenta.

3. **Post-procesado:** A continuación de la búsqueda de correspondencias se realiza un procesado para mejorar los resultados obtenidos.

Para hacer uso en OpenCV del algoritmo BM debemos de utilizar la clase *StereoBM* y iniciaremos el calculo de disparidad con el siguiente operador

```
void StereoBM::operator()(imageUG1, imageUG2, disp, CV_32F)
```

Donde *imageUG1* y *imageUG2* son el par de imágenes rectificadas y en escala de grises, *disp* la matriz en la que se guardará la disparidad. Si el ultimo argumento fuese de tipo CV_16S indicaríamos que la matriz de disparidad estará compuesta por elementos de 16 bits con signo, de esta forma no tendríamos los valores reales de disparidad, si no que estarían escalados, teniendo que dividirlos entre 16 a cada uno para obtener los verdaderos. Si indicamos que sea de tipo CV_32F no será necesario hacerlo, pues el mapa de disparidad ya vendrá con los valores correctos. Después de calcular el mapa de disparidad es necesario que sea normalizado a CV_8U para ser mostrado en pantalla, de tal forma que el mapa de disparidad contenga elementos de 8 bits sin signo.

A la hora de ejecutar el algoritmo podemos configurarlo con diferentes parámetros que afectarán a cada una de las etapas:

- **Pre-procesado**
  - prefilter_size: Define el tamaño de la ventana que se utiliza para normalizar las intensidades. Sus valores van desde 5..255
  - prefilter_cap: Limite de intensidad para los píxeles que van a ser normalizados. Sus valores van desde 1..63
- **Búsqueda**
  - SAD Window Size: Tamaño de la ventana que utilizará la técnica SAD. Valores altos implicaran menos ruido, pero también un mapa de disparidad menos preciso, además de afectar al rendimiento. Mientras que con valores pequeños obtendremos un mapa de disparidad más detallado pero con más ruido y correspondencias erróneas. Sus valores deben ser impares e ir desde 5..255 sin ser mayores que la anchura o altura de la imagen.
  - Min Disparity: Mínimo de disparidad que indica el punto de comienzo de la búsqueda en el horópter. Dicho de otra forma, el offset desde el punto actual a

buscar correspondencia al punto inicial del horópter. Si su valor es positivo, será más fácil encontrar objetos cercanos a las cámaras, pero más difícil encontrar los lejanos. Mientras que valores negativos pueden ser útiles si las cámaras no son paralelas.

- Num Disparities: Define el tamaño del horópter, cuanto más alto es su valor menor rendimiento tendrá el algoritmo. Sus valores deben ser positivos y divisibles entre 16.

- **Post-procesado**

  - Uniqueness Ratio: A la hora de buscar una correspondencia siempre elegimos el resultado que tenga mayor parecido al bloque original. Esto puede llevar a falsos positivos, para evitarlos se filtran los resultados quedándonos solo con los que satisfagan

$$uniqueness\_ratio > \frac{(mejor\_match - siguiente\_mejor\_match)}{siguente\_mejor\_match} \tag{6.2}$$

  - Texture Threshold: Se utiliza para comprobar que la ventana utilizada en la búsqueda cuenta con un nivel de textura mayor que el introducido.
  - Speckle Window Size: Filtra regiones de disparidad que sean de menor tamaño que un numero de píxeles. Por ejemplo, para un valor de 100 indicará que las regiones con menos de 100 píxeles no se tendrán en cuenta.
  - Speckle Range: Se utiliza para agrupar regiones de disparidad. Las regiones formarán un grupo solo si se encuentran entre una distancia de píxeles menor o igual que la indicada en este valor.

6.1.2. Algoritmo SGBM

El algoritmo SGBM (*Semi Global Block Match*), a diferencia del BM, no busca correspondencias entre píxeles, si no que trata de minimizar una función de energía a nivel de ventanas.

De esta forma llamemos I_r a la imagen de referencia, y I_t la imagen objetivo. Definiremos p como el punto de coordenadas (x,y) en I_r y q como el punto de coordenadas $(x+d,y)$ en I_t, estos serán dos puntos para los que la correspondencia esta actualmente evaluándose. Definiremos $w_n^r(i,j)$, $w_n^t(i,j)$ como dos ventanas de tamaño n centradas en (i,j) respectivamente en I_r y I_t. A este par de ventanas $w_n^r(i,j)$, $w_n^t(i+d,j)$ lo vamos a definir como $W_n(i,j,d)$.

Llamaremos $S(p,q)$ al conjunto de pares de ventanas definidas en I_r y I_t. Si evaluamos la correspondencia entre los puntos (p,q) debemos tomar un subconjunto de $S(p,q)$ al

que llamaremos $S_V(p,q)$. En este subconjunto se buscara la correspondencia intentando minimizar una función de energía.

El agrupar áreas de píxeles en base a su similaridad de disparidad, permite que este algoritmo identifique el entorno con mayor precisión reduciendo los problemas generalmente producidos por oclusiones, cambios de iluminación, poca textura...

OpenCV implementa una modificación del algoritmo presentado por H. Hirschmuller [6] y nos permite modificar una serie de parámetros para variar el funcionamiento de este. Estos parámetros son los mismos que en el algoritmo BM, pero se añade alguno más como P1 y P2 que son dos parámetros de penalización que nos permiten variar la suavidad en los cambios de disparidad, de tal forma que cuanto más altos sean sus valores más suave sera la disparidad.

- **P1:** Es la penalización del cambio de disparidad entre un píxel vecino. A continuación se muestra un buen valor para P1

$$8 * Numero\_canales\_imagen * SADWindowSize^2 \tag{6.3}$$

- **P2:** Es la penalización del cambio de disparidad entre más de un píxel vecino. Es necesario que P2 >P1. A continuación se muestra un buen valor para P2

$$32 * Numero\_canales\_imagen * SADWindowSize^2 \tag{6.4}$$

- **fullDP:** Es un parámetro *booleano* que por defecto se encuentra en *false*. Esto implica que el algoritmo utilice su variante *single-pass* lo que hará que se consideren solo 5 direcciones en vez de 8. Si le asignamos *true* se activara la otra variante, con el inconveniente de que gastara mucha más memoria. El gasto de memoria en bytes viene dado por la siguiente función

$$O(W * H * numDisparities) \tag{6.5}$$

El algoritmo busca correspondencias mediante bloques de píxeles, pero si especificamos que el parámetro SADWindowSize sea igual a uno, reduciremos los bloques a un único píxel. Mientras que el proceso de búsqueda es distinto, los pasos del pre-procesado y post-procesado son iguales que en el algoritmo BM.

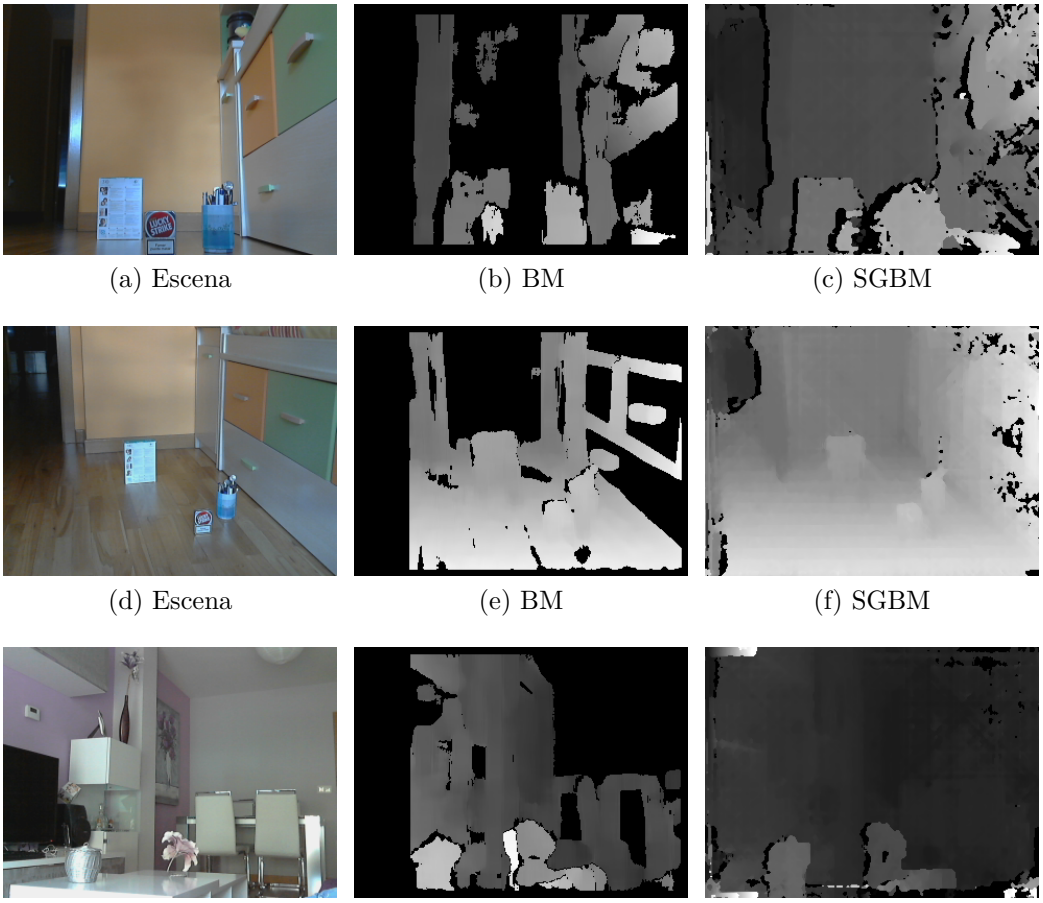
Para hacer uso de este algoritmo en OpenCV utilizaremos la clase *StereoSGBM* y iniciaremos el calculo de disparidad con

```
void StereoSGBM::operator()(imageU1, imageU2, disp)
```

Donde *imageU1* y *imageU2* son el par de imágenes rectificadas, pudiendo ser de uno o tres canales y *disp* el mapa de disparidad de 16 bits con signo, por lo que será necesario dividir cada elemento entre 16 para obtener los valores en coma flotante. También sera necesario normalizar el mapa de disparidad a CV_8U para mostrarlo en pantalla.

6.2. Experimento para la Comparación entre algoritmos

En este apartado se muestra el experimento realizado para la comparación entre el algoritmo BM y SGBM. Para ello se han tomado 3 capturas distintas para ver las diferencias en los mapas de disparidad entre los dos algoritmos.



(a) Escena (b) BM (c) SGBM

(d) Escena (e) BM (f) SGBM

(g) Escena (h) BM (i) SGBM

Figura 6.1: Mapas de disparidad generados por BM y SGBM

Como se puede observar en la figura 6.1 las diferencias entre los dos algoritmos son notables. Se observa que el algoritmo SGBM devuelve un mapa de disparidad más completo y detallado sin obtener apenas ruido. Se aprecia que donde existen variaciones de textura, como la caja de (a) o los bordes del mueble de (d) el algoritmo BM no tiene problemas para encontrar correspondencia, pero donde la textura es más uniforme como la pared de (a) o las sillas de (g) el algoritmo BM tiene dificultades para encontrar correspondencia mientras que el SGBM la encuentra fácilmente.

En alguna de las figuras correspondientes a SGBM puede parecernos que la disparidad del fondo es la misma, ya que no se diferencia en la figura, pero al pasarlas por una función de umbral e ir aumentando el valor, vemos que no es así, pues lo más alejado va desapareciendo del fondo de la imagen.

También se ha medido el rendimiento de los algoritmos, este tema se desarrolla en el capítulo 7.

6.3. Consideraciones sobre la Estimación de Distancia

A la hora de estimar la distancia no nos interesa únicamente un único punto del espacio, sino todos los que pertenecen a un objeto, y por ello se hace uso de los *blobs*.

Un *blob* es un grupo de píxeles conectados entre sí que comparten una característica en común. Por ejemplo, es posible filtrar blobs por su color, su tamaño o forma.

(a) Escena (b) BM (c) Blobs

Figura 6.2: Blobs obtenidos después de aplicar una función de umbral al mapa de disparidad

6.3.1. Detección de Blobs

El objetivo de la detección de blobs, como su nombre indica, es identificar estas regiones y marcarlas. En este proyecto nos interesa detectar objetos, y detectar el blob que forma ese objeto, pues una vez que tenemos el blob tenemos los píxeles que lo componen y podemos triangular para obtener la distancia a la que se encuentra.

Para ello se ha utilizado la librería OpenCVBlobsLib que presenta varias mejoras frente a la antigua librería cvBlobsLib, puesto que se adapta perfectamente a la implementación en C++ de OpenCV. Por ello nos permite utilizar clases como Mat en lugar de lplImage o beneficiarnos de cuenta con soporte multi núcleo.

La detección de blobs se ha planteado de la siguiente forma en este proyecto:

1. Primero aplicaremos un umbral al mapa de disparidad. La idea es eliminar objetos con baja disparidad, es decir objetos lejanos, dando prioridad a los que están más cercanos y por tanto de color más claro en el mapa de disparidad.

2. Como es posible que dos objetos que se encuentran muy cercanos en la imagen sean detectados como una única región, lo que haremos será aplicar un filtro morfológico de erosión. Para ello llamaremos a

```
erode(src, dst, Mat())//Dimension kernel por defecto 3x3
```

3. Buscaremos los blobs y los filtramos según su área, de tal forma que si son de área menor que un valor sean excluidos. Así nos evitamos la detección de ruido o de objetos que son demasiado pequeños para ser de interés.

```
1  CBlobResult blobs(dst);
2  CBlobResult blobs_filtered;
3  blobs.Filter(blobs_filtered, FilterAction::FLT_EXCLUDE, CBlobGetArea
       (), FilterCondition::FLT_LESS, min_area);
```

4. Una vez que tenemos los blobs, los dibujaremos en la pantalla, mostraremos sus bounding boxes y los guardaremos para usarlos posteriormente. Tal y como se ve en la figura 6.2.

6.3.2. Cómo se estima la distancia

En el proceso de calibración se mencionó la matriz Q. Esta matriz la obteníamos con el método `stereoRectify()` y con ella podremos proyectar un punto de la imagen a la escena. La matriz Q es una matriz 4×4 y dado un píxel (x, y) podemos calcular sus coordenadas homogéneas como se muestra en la formula 6.6.

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q * \begin{bmatrix} x \\ y \\ disparidad(x,y) \\ 1 \end{bmatrix} \tag{6.6}$$

Y obtener sus coordenadas tridimensionales

$$(X, Y, Z) = (\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}) \tag{6.7}$$

Esto ya viene implementado en OpenCV de forma que utilizando la siguiente función ya tendremos una matriz de tres canales, cada uno con las respectivas coordenadas (X, Y, Z) de los píxeles.

```
reprojectImageTo3D(roi, depth, Q)
```

La región *roi* es la región que nos interesa del mapa de disparidad (el bounding box del blob), *depth* la matriz en la que se guardarán las coordenadas.

La estimación de distancia a los objetos se ha hecho de las siguientes maneras:

- **Media:** Tomando todos los valores individuales de la coordenada Z de la región del blob, se calcula la media y se utiliza el resultado como distancia al objeto. Se desprecia el valor de disparidad cero, para que la media no se desvirtúe.

- **Moda:** Igual que con la media, se toman todos los valores de distancia, pero en vez de hacer una media elegimos el que más se repita. Igual que en la media el valor cero no se tiene en cuenta.

6.4. Experimentos sobre Estimación de Distancia

En este apartado se plantea el experimento y los resultados obtenidos con el software desarrollado a la hora de estimar la distancia.

Por lo que se han realizado una serie de pruebas con el algoritmo BM para ver si los resultados devueltos por el programa desarrollado son fiables, para ello se han tomado 28 capturas de una caja a distintas distancias, 14 de ellas en un parque y las otras restantes en el interior de una casa. De esta forma comprobaremos si distintas iluminaciones, una con luz natural y la otra con una única luz artificial, influyen en los resultados.

(a) Exterior

(b) Interior

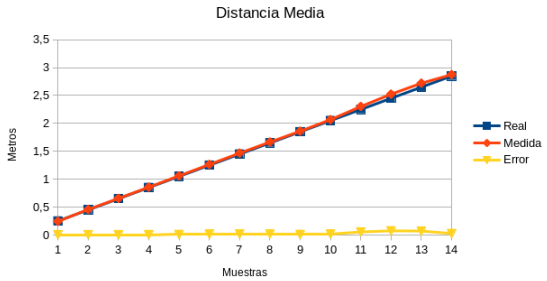
(c) Mapa Disparidad Exterior

(d) Mapa Disparidad Interior

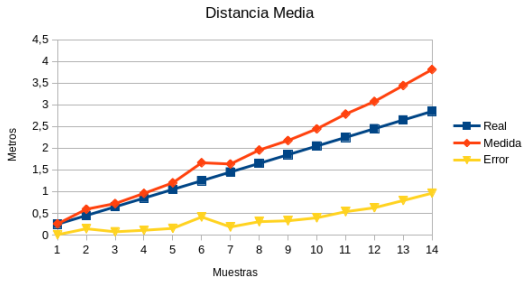
Figura 6.3: Escenarios de la prueba

Una vez realizadas las pruebas con los dos métodos, se ha obtenido los resultados reflejados en las gráficas de la figura 6.4. En ellas se hace una comparación entre la medida real y la estimada por el programa. Se puede observar cómo en el exterior la estimación de distancias se comporta muy bien, pues el error máximo obtenido fue de $7,4\ cm$ en la décimo segunda medida, con una distancia real de $2,45\ m$. Mientras que en el interior con

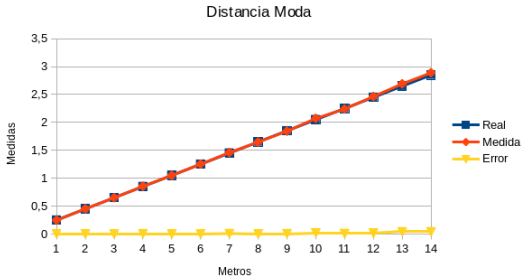
el método de la media en la décimo segunda medida obtenemos un error de $62,8\ cm$ y se alcanza un error máximo de $96,3\ cm$ en la última medida correspondiente a $2,85\ m$.



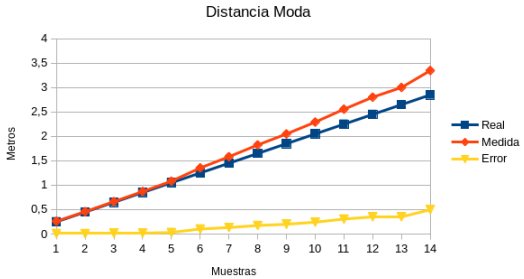
(a) Media Exterior



(b) Media Interior



(c) Moda Exterior



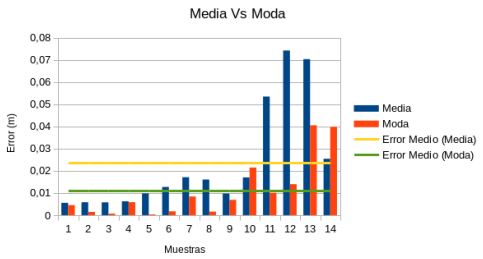
(d) Moda Interior

Figura 6.4: Resultados obtenidos junto con el error en la medida

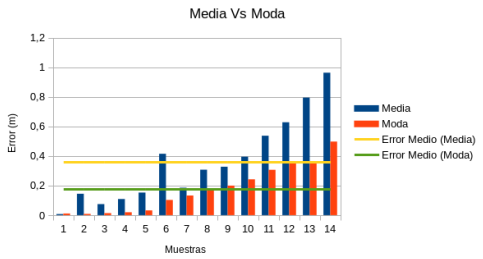
Observando los resultados, se puede afirmar que el algoritmo se comporta mejor en el exterior que en el interior.

Además de la iluminación, en el interior hay un problema más que se puede observar en la figuras 6.3 (b) y (d). En estas se detectan correspondencias confusas provocadas por el tipo de suelo que crea un reflejo de la caja que desvirtúa la medida estimada.

En la figura 6.5 se muestra el error cometido por la estimación en cada muestra. Por la escala, puede parecer que el error es mayor en la gráfica del exterior, pero si nos fijamos con más detenimiento, nos damos cuenta de que las unidades del eje y son distintas.



(a) Error Exterior



(b) Error Interior

Figura 6.5: Error resultante en cada muestra junto con su error medio

De la figura 6.5 podemos concluir que generalmente el método de la moda nos da estimaciones más precisas, pues en el exterior se comporta de forma bastante aceptable y en el interior el error es relativamente bajo hasta los $1,25\ m$ siguiendo en aumento a partir de ahí. También se ha calculado el error medio para los métodos de la media y la moda, siendo respectivamente para el exterior de $2,5\ cm$ y $1,1\ cm$, y para el interior de $36\ cm$ y $17,5\ cm$.

Capítulo

7 Rendimiento de algoritmos y plataformas

En este capítulo se describen las distintas pruebas de rendimiento realizadas con el software desarrollado y los resultados obtenidos tanto en tiempo como en tasa de *frames* (FPS, marcos por segundo).

Para ello se ha medido el rendimiento de los programas y algoritmos tanto en un ordenador de sobremesa y en una Raspberry Pi 2 para ver su comportamiento, ver qué algoritmo elegimos y detectar cuellos de botella. El ordenador que se ha utilizado dispone de un procesador Intel Core i5 750 de 4 núcleos a 2,67GHz, mientras que la Raspberry dispone de un ARMv7 Cortex-A7 de 4 núcleos a 900MHz.

7.1. Algoritmos de Correspondencia

En este apartado se han probado los algoritmos de correspondencia BM y SGBM en las dos plataformas, con los parámetros de la tabla 7.1. Para medir el rendimiento se ha ejecutado cada algoritmo 100 veces en un par de imágenes ya rectificadas y se ha hecho la media con los resultados de tiempo. Los resultados obtenidos pueden verse en las tablas 7.2 y 7.3.

De los resultados de la prueba podemos observar que BM es muchísimo más rápido que SGBM, 4 veces más rapido en el Intel i5 y 10 veces más rápido en el Cortex-A7. Como el objetivo de este trabajo es montarlo en la plataforma embebida, se opta por el algoritmo BM, pues el algoritmo SGBM, a pesar de conseguir un mapa de disparidad mejor, no consigue llegar a 1 FPS, lo que lo hace inútil.

Algoritmos	**B**M	**S**GBM
Pre-Filter Size	5	–
Pre-Filter Cap	32	63
SAD Window Size	13	5
Min Disp	0	0
Num Disp	3	3
Texture Threshold	1000	–
Uniqueness Ratio	15	5
Speckle Window Size	200	50
Speckle Range	32	32

Tabla 7.1: Parámetros utilizados para las pruebas

Algoritmos	**B**M	**S**GBM
Media (s)	0,0104	0.046
FPS	96,15	21,74

Tabla 7.2: Rendimiento BM y SGBM en Intel Core i5 750

Algoritmos	**B**M	**S**GBM
Media (s)	0,1444	1,5452
FPS	6,93	0,65

Tabla 7.3: Rendimiento BM y SGBM en ARMv7 Cortex-A7

7.2. Rendimiento del programa en serie

En este apartado se ha medido el rendimiento en conjunto del programa (sin paralelizar el algoritmo), para ello en esta sección se va a desglosar lo que tarda cada apartado del

programa: Captura de cámaras, búsqueda de correspondencias y detección de blobs junto con la estimación de distancia.

Para ello utilizaremos los datos del apartado anterior del algoritmo BM y se seguirá el mismo proceso, es decir tomaremos 100 capturas con la cámara y calcularemos 100 veces los blobs de un mapa de disparidad, para posteriormente calcular el promedio. Los resultados obtenidos pueden verse en las tablas 7.4 y 7.5.

Adquirir Imagenes		**C**orrespondencias (BM)		**B**lobs y Distancia	
0,069 - 0,033	15 - 30	0,0104	96,15	0,0036	277,77

Tabla 7.4: Resultados de tiempos y tasa de FPS en Intel Core i5 750

Adquirir Imagenes		**C**orrespondencias (BM)		**B**lobs y Distancia	
0,069 - 0,049	15 - 20	0,1444	6,93	0,0174	57,5

Tabla 7.5: Resultados de tiempos y tasa de FPS en ARMv7 Cortex-A7

Al realizar las pruebas con las cámaras se ha advertido que la tasa de FPS varía dependiendo de la iluminación de la escena. Parece ser que cuando la iluminación es algo baja, las cámaras hacen un ajuste para obtener capturas de mejor calidad sacrificando para ello la tasa de FPS. Ejemplos de iluminaciones se pueden ver en la figura 7.1.

(a) 30 FPS (b) 15 FPS

Figura 7.1: Resultados con distintas iluminaciones en Intel Core i5

Teniendo en cuenta que en el peor de los casos las cámaras obtendrán imágenes a una tasa de 15 FPS, éstas formarán un cuello de botella en el Intel, mientras que en la Raspberry, que es donde se enfoca este trabajo, el cuello de botella lo causa el algoritmo de búsqueda de correspondencia. De esta forma el programa completo se ejecutará a una tasa de unos 4 FPS de promedio en la Raspberry Pi 2.

Como nos es imposible hacer que las cámaras capturen imágenes a mayor velocidad y dado que nuestro principal cuello de botella es el algoritmo BM, se ha paralelizado el programa de forma que mitigue estos problemas.

7.3. Rendimiento de la implementación en paralelo

Utilizando el programa que se ha propuesto en el apéndice E, se han medido los tiempos para comprobar si se ha conseguido reducir la tasa de FPS del programa. Como se explica en el apéndice, el programa se ejecuta con cuatro hilos. Uno de ellos es el encargado de capturar la escena y llenar un buffer con pares de imágenes, mientras que los otros tres restantes se encargan de calcular el mapa de disparidad, detectar los blobs y estimar distancias. Para medir el rendimiento de la implementación en paralelo se ha tomado como medida el tiempo que hay entre que termina uno de estos tres últimos hilos hasta que termina otro que se encargue de hacer las mismas operaciones. Como en los demás apartados, también se ha realizado el promedio de 100 muestras de tiempo.

Se han sumado los tiempos medios de la tabla 7.5 y se han añadido a la tabla 7.6 para hacer una comparación con los resultados obtenidos del programa multi-hilo.

Mono-Hilo		**M**ulti-Hilo	
Tiempo(s)	FPS	Tiempo(s)	FPS
0,2308	4,33	0,0873	11,45

Tabla 7.6: Comparación entre el programa mono-hilo y el multi-hilo con tres tareas

Los resultados son bastante satisfactorios, pues el programa multi-hilo es capaz de triplicar la tasa de FPS del mono-hilo. El inconveniente de ejecutar el programa multi-hilo con tres tareas simultáneas es que la carga del procesador estará al 100 % con los consiguientes problemas de sobrecalentamiento y consumo de baterías. Podremos reducir esta carga bajando el número de tareas, lo que conlleva a una perdida de rendimiento. Se ha hecho la misma prueba tanto para una y dos tareas, los resultados se pueden ver en la tabla 7.7.

Una Tarea		**D**os Tareas		**T**res Tareas	
0,1738	5,75	0,107	9,35	0,0873	11,45

Tabla 7.7: Resultados de tiempos y FPS obtenidos del programa multi-hilo, con distintas tareas

Se puede observar como tanto con una, dos o tres tareas la tasa de FPS es mayor que en el programa mono-hilo. Esto es debido a que las cámaras trabajan en paralelo con las demás partes del programa, mientras que en el mono-hilo lo hacen de forma secuencial.

Parte IV

Análisis del trabajo. Conclusiones y trabajo futuro

Capítulo

8 Desviación en la planificación

En este apartado se muestra la desviación resultante en la planificación inicial. Debido a que el trabajo dispone en cada tarea una cantidad de variables desconocidas, era imposible a priori estimar o planificar con precisión la dirección a la que se iba a dirigir el proyecto.

En la sección 2.1 se muestran los requisitos funcionales que en un primer análisis se establecieron en el trabajo:

- Detección de objetos que se encuentren frente a las cámaras
- Estimación de la distancia a los objetos detectados
- Tiene que poder ejecutarse tanto en arquitectura ARM como x86
- El trabajo se implementará de forma modular en ROS

En la medida de lo posible se han cumplido los tres primeros requisitos, si bien, el primero de ellos no con los mejores resultados esperados, pues al aplicar una única función de umbral al mapa de disparidad, algunos objetos se confunden con otros objetos o con el suelo (ruido) y acaban formando un único blob. De esta forma si se utiliza la función media para estimar la distancia, ésta se desvirtúa.

El ultimo no llegó a cumplirse debido a varios cambios en la planificación inicial.

Conforme se avanzaba en el proyecto y se iban investigando nuevas herramientas y algoritmos, estos se implementaban en el software desarrollado, y varias veces se eligieron herramientas erróneas que no satisfacían los requisitos correctamente o en su totalidad. Por ejemplo se investigó el método de búsqueda de correspondencias SURF, pero debido a que no satisfacía los requisitos funcionales correctamente se descartó y se siguió buscando más.

A medida que se terminaba con la codificación del software, tal y como se planificó en un principio, se realizaron las pruebas en las distintas plataformas que disponíamos. Sin embargo, y como puede verse en la sección 7 el rendimiento en la Raspberry Pi 2 no fue el esperado, pues la tasa de FPS hacía que el software fuese inservible en esta plataforma.

Por ello se decidió hacer cambios en la planificación inicial para paralelizar el software del cálculo de disparidades. Para introducir las tareas necesarias para la paralelización, se optó por retrasar la implementación del software en ROS y el montaje en el robot, ya que se valoró la paralelización del software como una tarea más crítica y de más importancia por sus consecuencias.

Al terminar la paralelización del software el tiempo destinado al proyecto estaba agotado por lo que se clasificó como trabajo futuro las tareas correspondientes a ROS.

En cuanto a las distintas tecnologías utilizadas en el trabajo, la mayoría eran desconocidas para el desarrollador. De esta forma se han realizado un gran número de pruebas para intentar comprender las distintas tecnologías y algoritmos que iban sucediendo conforme se seguía profundizando en la investigación. Por ello gran parte del esfuerzo de este proyecto se ha dedicado en la formación del desarrollador en estas tecnologías y algoritmos.

Capítulo

9 Conclusiones y trabajo futuro

El trabajo tiene como principal objetivo obtener información tridimensional a partir de información bidimensional. De forma que se investigaron distintos métodos como SURF, BM, SGBM y Graph-cuts. Entre estos se eligió BM debido a su completitud del mapa de disparidad, con un rendimiento más que aceptable, teniendo en cuenta que el objetivo era buscar la mejor relación entre mapa de disparidad y rendimiento.

Aunque este trabajo está enfocado a la robótica, sus aplicaciones pueden ser muy variadas, como por ejemplo el modelado de objetos reales. Los resultados obtenidos en la sección 7 indican que estos algoritmos pueden ser utilizados en aplicaciones en tiempo real en equipos de sobremesa, incluso en una placa como una Raspberry Pi 2, aunque esta ya presenta más limitaciones.

Debido a la distorsión que se causa en las imágenes al capturar un objeto que se mueve a gran velocidad no hacen a este trabajo el perfecto para hacer seguimiento de distintos

objetos. Por otra parte estos métodos, tienen algunas ventajas sobre los basados en láser, debido a que el láser solo obtiene mediciones del entorno si los objetos se encuentran en la trayectoria del haz. Con un sistema estereoscópico el grado del entorno percibido es mayor, por tanto es más difícil que un objeto no sea percibido.

Este trabajo se aborda mediante el paradigma de la robótica clásica SMPA. Por una parte se ha cumplido con las dos primeras Sense y Model, pero las dificultades propias de un trabajo de investigación como éste no permiten cumplir todos los requisitos funcionales y por tanto hubo que modificar la planificación y se abordaron otros objetivos.

El trabajo estaba planteado de una forma completamente abierta, por lo que puede crecer y mejorar en varios aspectos:

- Mejorar el sistema de detección de objetos, pues el implementado cumple el propósito, pero es muy simple y en algunos casos es posible detectar un objeto donde no lo hay o detectar un objeto más grande de lo que es. Por ejemplo, en vez de una, podría aplicarse varias funciones de umbral a los mapas de disparidad y comparar los resultados obtenidos.

- Si se cambian las cámaras por unas capaces de capturar a una tasa mayor de FPS, se reducirá el cuello de botella causado por las mismas en el equipo de sobremesa.

- Hacer uso de una GPU de Nvidia para calcular los mapas de disparidad. OpenCV es compatible con CUDA para utilizar el algoritmo BM.

- Es posible que, utilizando la version 3 de OpenCV, podamos cambiar el formato de captura de las cámaras de 'YUYV' a 'MJPG'. Cambiando a 'MJPG' quizás se pueda obtener una captura estable de 30 FPS incluso en escenas con poca iluminación. Esto no puede hacerse con la versión utilizada en este proyecto, pues aunque en la documentación aparece la función que nos permite hacer el cambio de formato, en GitHub se puede observar cómo la función no está implementada.

- Modularizar el software en ROS. De esta forma, se podría ejecutar de forma distribuida el software con más de una Raspberry Pi 2. Para ello habría que implementar cada etapa del software por separado como un nodo, es decir un nodo que capture imágenes, otro que calcule mapas de disparidad y un último que calcule blobs y estime la distancia. Por tanto, utilizando el modelo de comunicación publicador/-suscriptor será posible tener varios nodos que se dediquen a la misma tarea, ya que los tópicos actuarían como buffers.

- Hacer uso de la biblioteca de la nube de puntos. De esta forma se añade una funcionalidad más al trabajo que nos permitirá modelar escenarios.

- Compilar OpenCV con soporte para QT y crear una interfaz de usuario mas amigable.

Parte V

Apéndices y bibliografía

Apéndice

A Manual de Usuario

Este capítulo esta dedicado a cualquier persona que quiera hacer uso o modificar el software desarrollado en este proyecto. Por ello este manual muestra una explicación sobre cómo compilar y hacer uso del trabajo.

A.1. Requisitos mínimos

El software de este proyecto ha sido desarrollado utilizando el lenguaje C++. Todos son compatibles con cualquier sistema operativo, ya sea una distribución Linux, Windows o OS X. Además se ha utilizado CMake como *build system*, de esta forma se facilita el compilado de los programas en las distintas plataformas.

Para poder utilizar el software es necesario:

- Un compilador de C/C++

- CMake version 2.8.11 o mayor
- Librería OpenCV 2.4.9.1 o mayor. Es posible que funcione con versiones menores, pero ésta ha sido la menor versión que se ha probado
- Libreria OpenCVBlobsLib. Ésta no es necesaria por el programa de calibración
- OpenMP. Aunque esta ya debería encontrarse incluida en el compilador. En este proyecto se han utilizado dos compiladores GCC 5.2.0 para el ordenador y GCC 4.9.2 para la raspberry. Ambos incluyen la version 4.0 de OpenMP
- Dos cámaras. No es necesario que sean iguales, pero ayuda en el proceso

A.2. Dónde descargarlo

Puede descargarse todo el código fuente desde aquí `https://github.com/labellson/stereo-vision`

O bien puede usarse git para bajar el repositorio entero:

```
$ git clone https://github.com/labellson/stereo-vision.git
```

A.3. Estructura del proyecto

El proyecto está formado por siete programas y una librería que sirve para guardar datos de calibración.

Los cinco programas están agrupados en carpetas:

- *calibrar* es un programa que puede ser utilizado para calibrar una única cámara. No tiene un uso en el trabajo desarrollado, pero se desarrolló al principio del trabajo y se utilizó como base para el programa de calibración estéreo
- *calibrateStereo* se utiliza para calibrar el sistema estéreo.
- *stereo_matchSGBM* es el programa que ejecuta el algoritmo SGBM para calcular los mapas de disparidad y estimar distancias
- *stereo_matchBM* como el programa *stereo_matchSGBM* pero con el algoritmo BM
- *stereo_matchBM_threads* exactamente igual que el anterior pero además es multihilo
- *VideoCapture* es el programa utilizado para hacer las pruebas de tasa de FPS con las cámaras
- *surf_object_detector* es el programa desarrollado con el algoritmo SURF, que finalmente no se utilizó en el proyecto

La librería desarrollada se encuentra en la carpeta *lib* y tiene como nombre *SCalibData*.

A.4. Como compilar el Software

Siempre que se cumplan los requisitos de la sección A.1 se podrá compilar el Software. Compilar los programas es muy sencillo, pues dentro de cada carpeta se ha creado un fichero de configuración *CMakeLists.txt* encargado de generar los *makefiles* correspondientes a cada programa. Además, en la carpeta padre a los programas se ha incluido otro *CMakeLists.txt* que incluye todos los programas para que puedan ser compilados a la vez.

Para compilar se hará uso de una terminal, nos ubicaremos en la carpeta raíz del proyecto, y crearemos una nueva carpeta *build*, entraremos dentro y generaremos los *makefile*, para finalmente compilar.

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Puede que nos interese, en caso de que no queramos depurar los programas compilados, compilar en modo *release*, para ello podremos indicarlo a la hora de generar los *makefiles* con:

```
$ cmake -DCMAKE_BUILD_TYPE=RELEASE ..
```

A.5. Cómo utilizar la librería SCalibData

La librería SCalibData ha sido creada para almacenar datos de calibración estéreo. Con ella podremos guardar en el disco un fichero de calibración, o si ya disponemos de uno podremos utilizarla para cargarlo.

Su uso es muy sencillo, y suponiendo que lo que queremos es guardar datos de calibración la utilizaremos de la siguiente forma:

1. Una vez que tenemos los datos de calibración, crearemos un objeto de la clase SCalibData con el siguiente constructor.

   ```
   SCalibData(Mat CM1, Mat CM2, Mat D1, Mat D2, Mat R, Mat T, Mat E, Mat F, Mat R1, Mat R2, Mat
   P1, Mat P2, Mat Q, Rect roi1, Rect roi2, int frame_width, int frame_height);
   ```

   Donde *CM1* y *CM2* son las matrices de calibración de las cámaras. *D1* y *D2* son los coeficientes de distorsión de las lentes. *R*, *T*, *E* y *F* son respectivamente la matriz de rotación, traslación, esencial y fundamental del conjunto binocular. *R1*

y *R2* son las matrices de rotación de cada cámara. *P1* y *P2* son las matrices de proyección de cada cámara. *Q* la matriz de disparidad a profundidad. *roi1* y *roi2* son las regiones de interés dentro de cada imágen rectificada. Y por último *frame_width* y *frame_height* es el la resolución que se ha utilizado para capturar las imágenes en el proceso de calibración. No es necesario utilizar todos los argumentos, en caso de que no queramos guardar uno lo indicaremos como *NULL*.

2. Cuando tengamos el objeto instanciado con todos sus atributos podremos guardarlo en el disco utilizando la función `write(FileStorage& fs)` de la clase *SCalibData*. Sólo es necesario pasar como argumento un objeto de la clase *FileStorage* que es la encargada de escribir ficheros *YAML* o *XML* en OpenCV.

Si, por el contrario, lo que queremos es cargar un fichero de calibración, utilizaremos únicamente la función `read(FileStorage& fs)` de la clase *SCalibData*, a la que tendremos que pasar también un objeto de la clase *FileStorage*.

Puede verse el código de la implementación de esta pequeña librería en los ficheros *SCalibData.cpp* y las cabeceras *SCalibData.h*.

A.6. Ejecución del software

A continuación se va a explicar como ejecutar cada programa, así como sus argumentos. A la hora de ejecutar los programas, siempre es posible indicar la opción -h que nos mostrará en pantalla información sobre su uso.

A.6.1. Software de calibración del sistema estéreo

El software de calibración se ejecuta de la siguiente manera

```
$ ./stereo_calib [width] [height] [-a]
```

Los argumentos encerrados en corchetes son opcionales, de esta forma podemos añadir la resolución deseada para la calibración. Si no la indicamos, tomara por defecto una resolución de 640x480. Con la opción -a activamos el modo automático. Con este modo el programa capturara las esquinas del tablero que se muestren a las cámaras, sin ninguna intervención del usuario. Aun así este modo no se recomienda si se quiere obtener unos buenos resultados de calibración.

Una vez ejecutado el programa se preguntará al usuario el número de esquinas tanto horizontales como verticales del tablero, el tamaño de los cuadrados, el número de fotos a hacer y el número de la cámara izquierda y derecha. Este ultimo variará dependiendo de como haya detectado las cámaras nuestro sistema operativo (Ej: Para /dev/video0 indicaremos 0).

En cuanto indiquemos todo lo anterior nos aparecerán dos ventanas con lo que están capturando las cámaras. Puesto que para el programa es imposible saber que cámara es la izquierda y cual la derecha, el usuario tendrá que comprobarlo, y si no están bien, cerrar el programa y volver a iniciarlo correctamente.

Una vez esté todo correctamente ajustado, se procede a mostrar el tablero a las cámaras. El programa detectará las esquinas interiores del tablero y nos las mostrará. Cuando las esquinas se muestren en las ventanas, es cuando podremos guardar las capturas, para eso pulsaremos la barra espaciadora. Si se activo el modo automático no sera necesario, pero se insiste en no usarlo, pues los resultados no serán los mejores.

Cuando se haya hecho el número de capturas indicadas por el usuario, se empezará a calcular los parámetros intrínsecos y extrínsecos. Este proceso puede tardar, y dependerá de el número de capturas que hayamos indicado.

Al final del programa se mostrará una ventana más en la que aparecen las dos capturas de las otras ventanas rectificadas, junto con sus regiones de interés y unas lineas horizontales. En este punto se puede pulsar la tecla r para parar la captura y congelar las imágenes. Con las imágenes congeladas podremos comprobar mucho mejor si el proceso de calibración ha sido bueno. También podremos pulsar la barra espaciadora para tomar una captura de las imágenes.

Finalmente para salir pulsaremos la tecla escape. En el directorio donde se ejecutó el programa se habrá guardado un fichero de calibración de nombre *stereo_calib_<height>.yml*.

A.6.2. Software de cálculo de disparidad y estimación de distancia

En esta sección se pretende mostrar cómo utilizar los tres programas de calculo de disparidad. Como su uso es similar, exceptuando algunas opciones distintas entre ellos, se van a explicar en conjunto y mostrando también las pocas diferencias que hay.

Para ejecutar uno de estos programas lo haremos de la siguiente manera

```
$ ./stereo_matchBM <camera0> <camera1> [-l image0 image1] [-c
    calib_file] [-d 1|2] [--test] [-h]
$ ./stereo_matchSGBM <camera0> <camera1> [-l image0 image1] [-
    c calib_file] [-d 1|2] [--test] [-h]
$ ./stereo_matchBM_threads <camera0> <camera1> [-l image0
    image1] [-c calib_file] [-t n][-d 1|2] [--test] [-h]
```

Como se ve su uso es similar excepto la opción -t que es exclusiva de *stereo_match_BM_threads*, con esta opción especificaremos el número de tareas que lanzará el software para calcular los mapas de disparidad y la estimación de distancias. Solo cambiaremos el número de tareas, que no es lo mismo que el número de hilos que ejecuta el programa, que seguirán

siendo cuatro. Si se quisiese cambiar el número de hilos se tendrá que modificar el código fuente del software.

Si queremos ejecutar uno de estos programas sin tener que utilizar una cámara podremos cargar un par de imágenes por medio de la opción -l, el único requisito es que las dos imágenes que vamos a cargar estén rectificadas, además será obligatorio añadir un fichero de calibración, pues es necesario para estimar las distancias. Para cargar un fichero de calibración lo haremos mediante la opción -c. Podemos utilizar la opción -d para cambiar el método para estimar las distancias, si elegimos el 1 estableceremos la media como método, si por el contrario se elige la opción 2 se utilizará la moda, si no se especifica se elegirá el 1. La opción –test nos servirá para iniciar el modo test del programa que ejecutará 100 ciclos de programa guardando los resultados de tiempo y calculando un promedio como resultado.

Una vez iniciado el programa nos aparecerán 6 ventanas correspondientes a la cámara izquierda y derecha, mapa de disparidad, mapa de disparidad umbralizado, blobs y configuración.

En la ventana de configuración podremos ajustar los parámetros de los algoritmos o configurar el umbral o el tamaño mínimo de los blobs. Debido a que OpenCV no permite números negativos en las *trackbars*, se puede cambiar la disparidad a valores negativos si pulsamos el botón m.

En la ventana de blobs, nos aparecerán los objetos que se han encontrado junto con su distancia estimada.

En la ventana de disparidad, si se activa el modo depuración del programa (solo puede activarse modificando el código fuente. En el CD viene activado el modo depuración en los binarios compilados), se podrá seleccionar una región cuadrada del mapa de la que se estimará la distancia utilizando el método de estimación de distancia en uso.

Igual que en el software de calibración, si pulsamos la tecla r, congelaremos las imágenes que las cámaras están capturando.

También es posible cambiar el método de estimación de distancias mientras el software se esta ejecutando, para ello se pulsara la tecla d.

Si al usuario le parece molesto que se este mostrando por pantalla el tiempo que lleva ejecutar cada ciclo, lo puede desactivar pulsando la tecla s.

Podremos capturar todas las imágenes que se muestran en las ventanas si pulsamos la barra espaciadora. Las imágenes se guardarán en el disco en el mismo directorio en el que se ejecuta el programa.

Para finalizar el programa pulsaremos escape. Si se utilizó la opción –test al ejecutar el programa, se habrá activado el modo test. De esta forma el programa no finalizará

inmediatamente, si no que se ejecutará cien veces más guardando los resultados de tiempo y calculando el promedio.

A.6.3. Software de captura de cámaras

Este software es muy simple y muy sencillo de utilizar. Para ejecutarlo lo haremos de la siguiente manera:

```
$ ./VideoCapture <camara0> <camara1> [-s <width> <height>] [-t
  ]
```

Será necesario pasar como primeros argumentos los números de las cámaras que queramos utilizar. Como argumentos opcionales tenemos -s y -t. Con el primero indicaremos la resolución de la grabación y con el segundo iniciaremos el modo test de forma que se ejecuten 100 ciclos y se calcule el promedio de tiempo para cada ciclo.

A.6.4. Software algoritmo SURF

Este software ejecuta el algoritmo SURF, y no ha sido utilizado en el proyecto. El software tiene como finalidad detectar un objeto de la escena, a partir de un objeto dado. Para ello, una vez que el programa este iniciado seleccionaremos arrastrando el ratón en la ventana el objeto de la escena que queramos detectar. Cuando se seleccione el objeto, empezará a ejecutarse el algoritmo SURF y si hay suficientes correspondencias dibujará en la escena un recuadro alrededor del objeto detectado.

Para ejecutar el programa lo haremos de la siguiente manera:

```
$ ./surf_object_detector [-bw] [-c camera] [-m min_matches]
```

Con la opción -bw iniciaremos la captura de las cámaras en blanco y negro. Con -c se puede indicar la cámara a utilizar, si no se indica se utilizará la 0. Por ultimo con la opción -m indicaremos el mínimo de correspondencias mínimas que tiene que haber para que se dibuje un recuadro en el objeto detectado.

Mientras el programa esté ejecutándose, se podrá pulsar r para congelar la captura y la barra espaciadora, para tomar una captura. Si queremos finalizar el programa bastará con pulsar la tecla escape.

Como dato adicional, este programa no se ha podido compilar con la versión de OpenCV que se ha utilizado en la Raspberry Pi 2, debido a que esta no dispone de las librerías del modulo *nonfree* en las que se encuentra implementado el método SURF.

Apéndice B

Pliego de condiciones

B.1. Condiciones de hardware

Las características mínimas requeridas por el ordenador para un correcto funcionamiento son las siguientes:

Para Microsoft Windows o Linux:

- Para plataformas x86, un procesador Intel Pentium Dual-Core o equivalente a 1,3 GHz multi-núcleo o más rapido.
- Para plataformas ARM, un ARMv7 Cortex-A7 de 4 núcleos a 900MHz o más rapido.
- 1GB de memoria RAM
- Monitor con una resolución mínima de 1024×768 píxeles. Pero se recomienda uno de mayor resolución.

Cámaras: No es necesario disponer de dos cámaras para hacer uso del proyecto, pues puede utilizarse con imágenes que previamente hayan sido rectificadas y de las que dispongamos un fichero de calibración. Aun así es muy recomendable disponer de dos cámaras para hacer un uso total de las funcionalidades del software.

Ratón: Es recomendable su utilización para la comodidad del usuario en la ventana de configuración, pero su uso no es necesario para el buen funcionamiento del software.

B.2. Condiciones de software

Además de los distintos sistemas operativos mencionados anteriormente sólo es necesario para el correcto funcionamiento el siguiente software:

- OpenCV 2.4.9.1 o mayor
- OpenCVBlobsLib. No es necesaria para el programa de calibración
- OpenMP, aunque solamente para el programa multi-hilo

B.3. Condiciones generales

La presente aplicación ha sido realizada para ser presentada al concurso público convocado por la Escuela Universitaria de Ingeniería de Vitoria-Gasteiz. El objetivo es hacer una investigación de los distintos algoritmos que nos permiten obtener información tridimensional a partir de información bidimensional y hacer una comparación entre ellos utilizando como criterio la relación entre completitud de mapa de disparidad y rendimiento.

Se busca facilitar el trabajo futuro de distintos desarrolladores que tendrán a disposición este trabajo para poder implementarlo dentro de sus propios desarrollos.

Estimando que los datos aportados en el presente pliego de condiciones son suficientes, se somete el proyecto a la aprobación del comité directivo de la EUI de Vitoria-Gasteiz.

Vitoria-Gasteiz, a 22 de Septiembre de 2015

Apéndice

C Actualización de la Raspberry Pi 2 a Raspbian Jessie

Como la distribución instalada Raspbian Wheezy solo disponía en los repositorios una versión de OpenCV menor que la usada (OpenCV 2.4.11), que no nos permitía compilar los programas desarrollados en C++, se optó por actualizar a la versión Jessie, que actualmente esta marcada como experimental.

Para iniciar la actualización, primero debemos editar las fuentes de los repositorios. Para ello se edita el fichero que se encuentra en **/etc/apt/sources.list** y se cambian todas las lineas en las que aparezca wheezy por jessie.

Finalmente con permisos root iniciaremos la actualización del sistema con

```
$ apt-get update
$ apt-get upgrade
$ apt-get dist-upgrade
```

y esperaremos a que se termine la actualización, que terminará en unas horas. Una vez actualizado el sistema, podemos bajarnos la version 2.4.9.1 de OpenCV de los repositorios que sí que nos permite compilar sin ningún problema los programas.

Apéndice

D Rosberry Pi

Esta guía explica como instalar ROS Indigo en una Raspberry Pi 2 con Raspbian Jessie. La guía a sido adaptada y traducida de la wiki de ROS [2].

D.1. Pre-requisitos

Primero vamos a hacer unas configuraciones y instalar dependencias necesarias para la correcta instalación de ROS.

D.1.1. Añadir los repositorios de ROS

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
  wheezy main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
$ wget https://raw.githubusercontent.com/ros/rosdistro/master/
  ros.key -O - | sudo apt-key add -
```

Ahora nos aseguramos de que el índice de repositorios de nuestro Debian esté actualizado:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

D.1.2. Instalar dependencias

```
$ sudo apt-get install python-setuptools python-pip python-
  yaml python-argparse python-distribute python-docutils
  python-dateutil python-setuptools python-six
$ sudo pip install rosdep rosinstall_generator wstool
  rosinstall
```

D.1.3. Inicializar rosdep

```
$ sudo rosdep init
$ rosdep update
```

D.2. Instalación

En este apartado vamos a descargar y compilar ROS Indigo.

D.2.1. Crear el espacio de trabajo de catkin

Para poder compilar los paquetes principales primero necesitamos crear un espacio de trabajo de catkin.

```
$ mkdir ~/ros_catkin_ws
$ cd ~/ros_catkin_ws
```

Después de esto descargaremos los paquetes principales. Para ello utilizaremos wstool, con wstool seleccionaremos la variante de ROS que queremos descargar. En un principio, se puede instalar cualquier variante, pero las probadas en la wiki son:

ROS-Comm: (Recomendada) Contiene ROS, librerías de comunicación y compilado. No tiene herramientas de interfaz de usuario.

```
$ rosinstall_generator ros_comm --rosdistro indigo --deps --
  wet-only --exclude roslisp --tar > indigo-ros_comm-wet.
  rosinstall
$ wstool init src indigo-ros_comm-wet.rosinstall
```

Desktop: ROS, rqt, rviz, y todas las librerías genéricas de robots.

```
$ rosinstall_generator desktop --rosdistro indigo --deps --wet
  -only --exclude roslisp --tar > indigo-desktop-wet.
  rosinstall
$ wstool init src indigo-desktop-wet.rosinstall
```

Con esto descargaremos todos los paquetes de la variante elegida y se guardarán en el directorio ~/ros_catkin_ws/src. El comando tardará unos minutos en descargar todos los paquetes principales de ROS. Con la opción -j8 podremos iniciar 8 descargas en paralelo.

En caso de que wstolls falle o se interrumpa, es posible continuar la descarga con:

```
$ wstool update -t src
```

D.2.2. Resolver dependencias

Antes de compilar el espacio de trabajo es necesario tener todas las dependencia requeridas. Utilizaremos para ello rosdep. Pero antes de empezar, instalaremos un par de dependencias que no están en los repositorios, así que tienen que ser instaladas manualmente.

D.2.2.1. Dependencias no disponibles en los repositorios de Jessie

Para Raspbian Jessie el paquete collada-dom-dev no se encuentra disponible. Dependiendo de que variante instalemos necesitaremos las siguientes librerías.

ROS-Comm: libconsole-bridge-dev y liblz4-dev

Desktop: libconsole-bridge-dev, liblz4-dev, liburdfdom-headers-dev, liburdfdom-dev y collada-dom-dev

Todas, excepto la mencionada al principio, pueden ser instaladas desde los repositorios. Para instalar collada-dom-dev lo haremos de la siguiente forma. Primero crearemos un nuevo directorio donde vamos a compilar (también instalaremos checkinstall y CMake).

```
$ mkdir ~/ros_catkin_ws/external_src
$ sudo apt-get install checkinstall cmake
$ sudo sh -c 'echo "deb-src␣http://mirrordirector.raspbian.org
  /raspbian/␣testing␣main␣contrib␣non-free␣rpi" >> /etc/apt/
  sources.list'
$ sudo apt-get update
```

Luego instalaremos la librería en cuestión (Se necesitará parchear collada_urdf tal y como se describe aquí.

```
$ cd ~/ros_catkin_ws/external_src
$ sudo apt-get install libboost-filesystem-dev libxml2-dev
$ wget http://downloads.sourceforge.net/project/collada-dom/
    Collada%20DOM/Collada%20DOM%202.4/collada-dom-2.4.0.tgz
$ tar -xzf collada-dom-2.4.0.tgz
$ cd collada-dom-2.4.0
$ cmake .
$ sudo checkinstall make install
```

Cuando check-install pregunte por algún cambio, cambiaremos collada-dom a collada-dom-dev, de la otra forma rosdep no encontrará la dependencia.

D.2.2.2. Resolviendo las dependencias con rosdep

```
$ cd ~/ros_catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro
    indigo -y -r --os=debian:jessie
```

Esto buscará todos los paquetes en el directorio src y encontrará todas las dependencias que estos tengan. Luego recursivamente instalará estas dependencias.

Puede que rosdep avise de que python-rosdep, python-catkin-pkg, python-rospkg y python-rosdistro no se instalaron. Se puede ignorar este error, por que ya fueron instalados anteriormente con pip.

D.2.3. Compilando el espacio de trabajo catkin

Una vez que se hayan completado las descargas y se hayan resuelto todas las dependencias, podemos compilar los paquetes.

Para ello llamamos a catkin_make_isolated:

```
$ sudo ./src/catkin/bin/catkin_make_isolated --install -
    DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/indigo
```

Esto instalará ROS en /opt/ros/indigo. Si elegimos la variante Desktop, en Raspberry Pi 2 catkin_make_isolated fallará en el paso de compilar rviz. Para compilarlo satisfactoriamente debemos utilizar:

```
$ .../ros_catkin_ws/build_isolated/rviz/make -j2
```

Ademas para rviz sera necesario aplicar este parche.

Ahora ROS debería estar instalado. Sólo queda hacer source a la instalación:

```
$ source /opt/ros/indigo/setup.bash
```

Apéndice

E

Paralelización del trabajo con OpenMP

OpenMP es una API utilizada para la programación multi-hilo. Esta API puede utilizarse en lenguajes como C/C++ y Fortran y esta pensada principalmente para procesadores multi-núcleo. Para ello se hace uso de directivas, con las que podremos indicar qué parte del código del programa debe ser ejecutada por varios flujos de ejecución. La sintaxis es muy sencilla de utilizar, es de la forma:

```
# pragma omp <directiva [clausula, ..]
{
  //Codigo paralelo
}
```

Como puede verse en la figura E.1, OpenMP utiliza un modelo fork-join, de forma que todo programa empieza con un único hilo llamado master que ejecutará el código hasta encontrarse con un bloque de código especificado como paralelo. Llegados a este punto el hilo master crea un grupo de hilos que ejecutarán ese bloque en paralelo. Cuando el

bloque se haya terminado, los hilos se sincronizarán, es decir se esperarán unos a otros y terminarán dejando solo a master ejecutar el resto del programa.

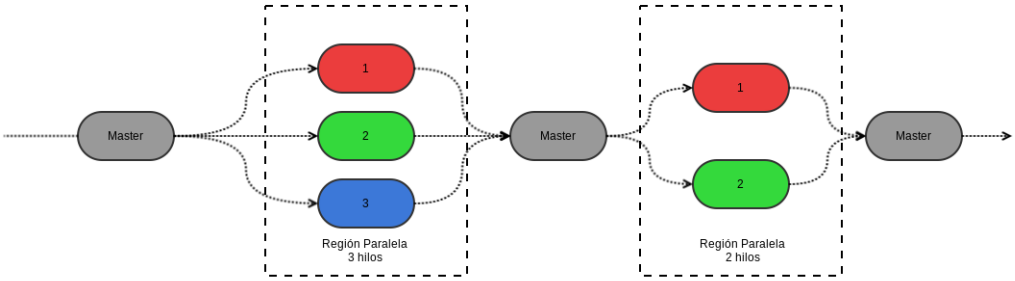


Figura E.1: Modelo *fork-join*

Se ha decidido paralelizar el trabajo, para aumentar el rendimiento del mismo e intentar mitigar cuellos de botella resultantes. Puede encontrarse mas información sobre tiempos y cuellos de botella encontrados en la sección 7.

Para ello, para cuatro hilos se ha seguido el planteamiento del diagrama de la figura E.2, que aunque no sea exactamente como se ha implementado, sigue la idea principal de la implementación.

La implementación real es la siguiente:

1. Primero se cargan todos los datos de la calibración, para poder rectificar las imágenes capturadas. Y se llega al bloque paralelo del programa

   ```
   #pragma omp parallel shared(pre_proccess_buff, disp_buff, t, go, rend), num_threads(4)
   ```

   Con la directiva *parallel* indicamos que a partir de ese punto comienza el bloque paralelo. La clausula *shared* indica las variables que serán compartidas entre los hilos. La directiva *num_threads* indica el numero total de hilos que se van a crear dentro del bloque.

2. Dentro del bloque paralelo se llega a otra directiva

   ```
   #pragma omp single nowait
   ```

   Los cuatro hilos se encontraran con esta directiva *single*, que indica que las instrucciones que encierra sólo serán ejecutadas por un único hilo. Con la cláusula *nowait* les decimos a los tres hilos restantes que no esperen al hilo que está trabajando y que sigan con el flujo de instrucciones. Esto se ha hecho así porque lo que ejecuta este hilo corresponde al primer bloque de la figura E.2, es decir, el hilo entrará en un bucle que parará cuando queramos terminar el programa e irá capturando imágenes y llenando el buffer.

3. Los tres hilos restantes siguen el flujo de ejecución del programa hasta que se encuentran con una directiva más.

   #pragma omp single

   Este código será ejecutado únicamente por un único hilo, y al no especificar la cláusula *nowait* que especificamos en el paso anterior, los dos hilos restantes se quedarán a la espera de este hilo. ¿Por qué se ha hecho esto? Pues porque este hilo va a entrar en otro bucle que acabará cuando queramos terminar el programa y se encargará de mirar si hay un par de imágenes dentro del buffer listas para ser utilizadas en la búsqueda de correspondencias. A medida que el buffer vaya llenándose, este hilo se encargará de lanzar tareas, que se ejecutarán en distintos hilos, para calcular el mapa de disparidad. Como este hilo no supone una carga para el procesador, se ha indicado un limite de 3 tareas de forma que puedan trabajar los 3 hilos simultáneamente. Para lanzar una tarea lo haremos con la directiva:

   #pragma omp task

Al ser un programa que se ejecuta en paralelo hay partes de la memoria que son compartidas (Ej: Buffer de imágenes). Por tanto hay que hacer uso de secciones críticas. Una sección crítica es una porción del programa en la que se accede a una parte de la memoria compartida, que sólo puede ser accedida por un único hilo de ejecución. Para indicar una sección critica dentro del programa se hace uso de la directiva:

```
# pragma omp critical(nombre_seccion)
{
  //Codigo de la seccion critica
}
```

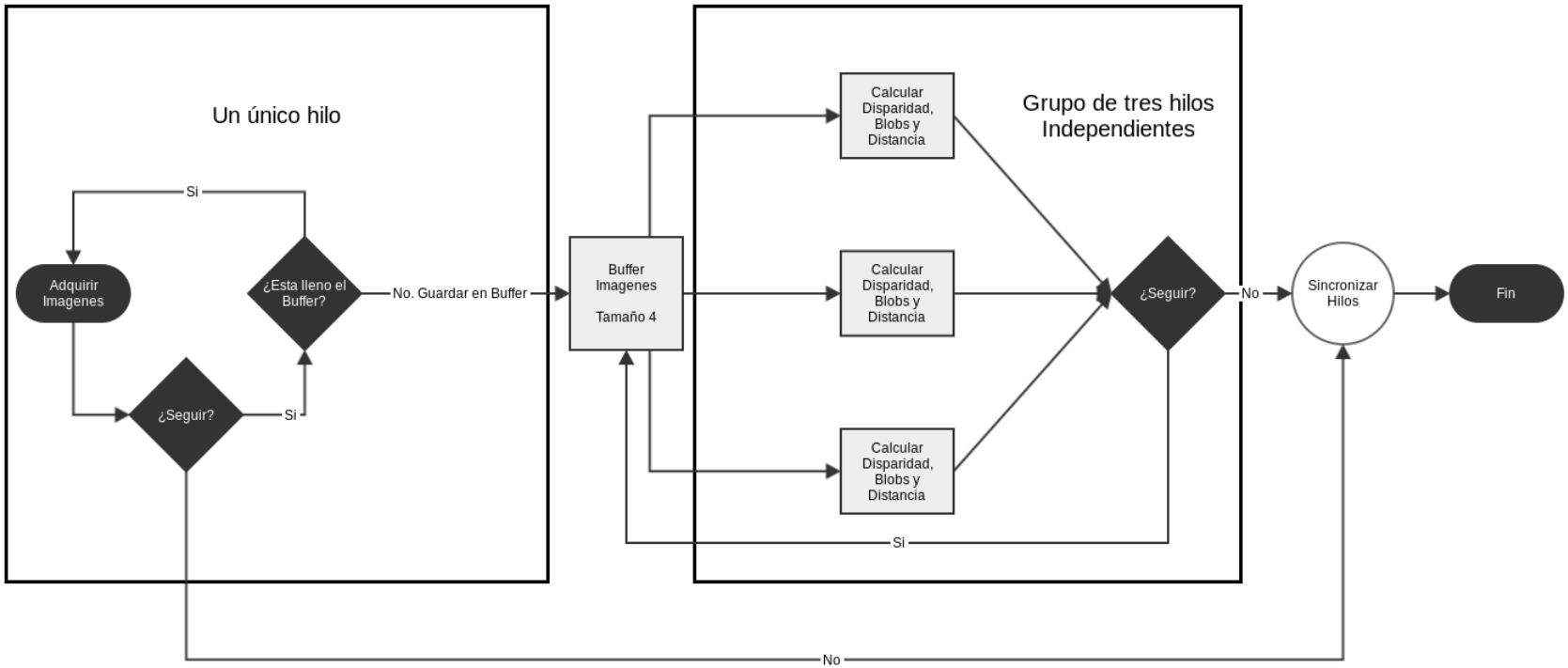


Figura E.2: Planteamiento de la paralelización del programa para 4 hilos

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom

to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be

included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Bibliografía

[1] Anonimo. Opencv documentation. http://docs.opencv.org/.

[2] Anonimo. Ros wiki. http://wiki.ros.org/. 97

[3] BOE. Xvi convenio de las tic. https://www.boe.es/boe/dias/2009/04/04/pdfs/BOE-A-2009-5688.pdf, 2009.

[4] D. Calin. Fundamental guide for stereo vision cameras in robotics. http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/, 2013.

[5] M. Hervás. Geometría de la formación de imágenes. http://manuelhervas.net/geometria-de-la-formacion-de-imagenes/, 2015.

[6] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341, 2008. 63

[7] D. Lee. Building and calibrating a stereo camera with opencv. https://erget.wordpress.com/2014/02/01/calibrating-a-stereo-camera-with-opencv/, 2014.

[8] C. McCormick. Stereo vision tutorial – part i. https://chrisjmccormick.wordpress.com/2014/01/10/stereo-vision-tutorial-part-i/, 2014.

[9] D. Mery. Visión por computador. *Santiago de Chile. Universidad Católica de Chile*, 2004. xi, xi, xi, 32, 34, 39

[10] M. Peris. Opencv: Stereo matching. http://blog.martinperis.com/2011/08/opencv-stereo-matching.html, 2011.

[11] J. Rambhia. Stereo calibration. http://www.jayrambhia.com/blog/stereo-calibration/, 2013.

[12] M. Shah. Fundamentals of computer vision1. 1997.

[13] J. Tarlea Jiménez. Sistema de posicionamiento de objetos mediante visión estéreo embarcable en vehículos inteligentes. *Trabajo de Grado (Ingeniero en Telecomunicaciones). Facultad de Ingeniería. Universidad de Alcalá, España, 104p*, 2009.

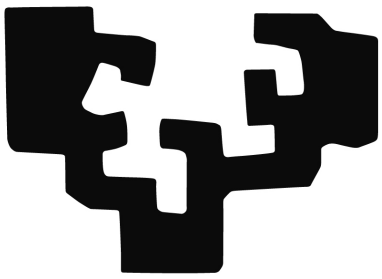
[14] M. T. Torriti. *Reconstrucción confiable de superficies usando rango de disparidad adaptivo*. PhD thesis, Tesis para obtener el grado de Magister en Ciencias de la Ingeniería. Santiago, Chile, 1998. xi, 37

[15] U. M. Torrontegi. Reconstrucción densa de modelos tridimensionales utilizando visión artificial. *Universidad del País Vasco / Euskal Herriko UnibertsitateaDepartamento de Ciencia de la Computacióne Inteligencia Artificial*, 2010.

[16] S. M. y Luca Nardelli. Opencvblobslib. http://opencvblobslib.github.io/opencvblobslib/.

[17] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000. 37

eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea