Capítulo 1

Introducción

El ser humano no es consciente del proceso neuronal que tiene lugar en nuestro cerebro con el simple hecho de andar o coger un objeto. Se podría decir que tenemos un super ordenador conectado a los órganos sensoriales capaces de recoger muchísima información y procesarla en un tiempo récord.

Desde la antiguedad ya se estuvo pensando en reproducir las habilidades humanas en algún tipo de máquina, la noción de concebir la mente humana como algún tipo de mecanismo no es reciente es referida en célebres filósofos, sin embargo, no es hasta 1950 y con la noción de la computación cuando se introduce la IA (Inteligencia Artificial) por el científico Alan Turing en su artículo *Maquinaria Computacional e Inteligencia* y donde se empieza a coger interés por este campo que será el precursor de una gran cantidad de desarrollos e innovaciones.

En este proyecto abordaremos el problema de obtener un mapa 3D compacto del entorno que rodea a nuestro dispositivo a partir de la información aportada por un sensor RGBD, dando por conocida la información sobre su posición y orientación a través de algún método externo.

1.1. Visión artificial

Dentro de la IA, se conoce la visión artificial como el procesado que hace nuestro celebro de las imágenes. Que es actualmente un amplio mundo de investigación y desarrollo. Dispositivos como móviles que usan cámaras (reconocimiento facial).

bajo coste, permite extraer mucha información, procesamiento muy costoso, detección de bordes, features, texturas. Reconocimiento de formas, caras

Aplicaciones:

Medicina OCR Biometría Ojo de halcón (tenis) MediaPro: distancia recorrida por un futbolista

1.2. Técnicas de autolocalización

Las técnicas de autolocalización ha suscitado gran interés por los investigadores en los últimos años. Consiste en conocer la localización de la cámara en todo momento simplemente con las imágenes capturadas sin disponer de ninguna información extra.

El problema a sido abordado por dos comunidades distintas, por un lado la de visión artificial que denominó al problema como **structure from motion (SfM)**, donde la información es procesada por lotes, capaz de representar un objeto 2D a 3D con solo unas cuantas imágenes desde diferentes puntos de vista. Y por otro lado la comunidad robótica denominó al problema **SLAM** (*Simultaneous Localization and Mapping*) que trata de resolver el problema de una manera más compleja adaptando el funcionamiento de los sistemas en tiempo real.

1.2.1. Structure from Motion (SfM)

Las técnicas SfM se analizan generalmente de forma offline, las escenas se graban a través de un conjunto de imágenes y luego se procesa, lo que permite realizar optimizaciones para el cálculo de la trayectoria, como por ejemplo el llamado ajuste de haces.

Existen aplicaciones comerciales que utilizan estas técnicas como es el caso de la aplicación PhotoTourism (Noah Snavely y Szeliski, 2006) desarrollada por Microsoft. Que consiste en el cálculo de la posición 3D en la que fueron captadas las imágenes, por ejemplo de un monumento, para después extraer el modelo 3D con el que el usuario puede interactuar libremente (Figura 1.1).

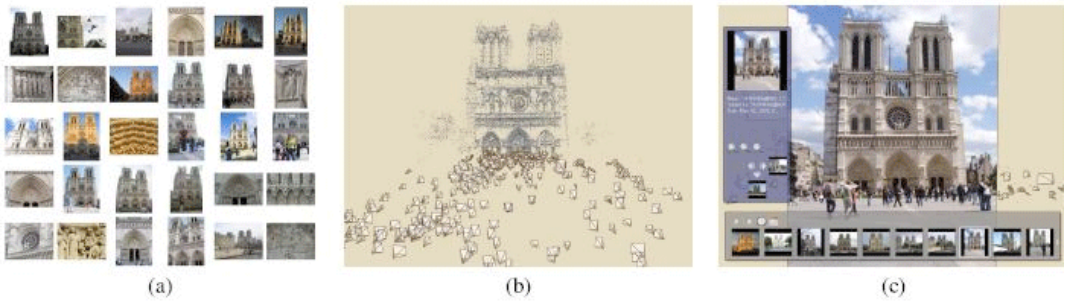

FIGURA 1.1: PhotoTourism: Se recogen una gran colección de imágenes (a), se reconstruyen los puntos 3D y los puntos de vista (b), por último la interfaz permite al usuario interactuar moviendose a través del espacio 3D mediante la transición entre fotografías.

1.2.2. Visual SLAM

En el problema conocido como *Simultaneous Localization and Mapping* (SLAM) busca resolver los problemas que plantea colocar un robot móvil en un entorno y una posición desconocidas, y que él mismo se encuentre capaz de construir incrementalmente un mapa de su entorno consistente y a la vez utilizar dicho mapa para determinar su propia localización.

La solución a este problema conseguiría hacer sistemas de robots completamente autónomos que junto con un mecanismo de navegación el sistema se encontrará con la capacidad para saber a dónde desplazarse, ser capaz de encontrar obstáculos y reaccionar ante ellos de manera inteligente.

La resolución al problema SLAM ha suscitado un gran interés en el campo de la robótica y ha sido resuelto teóricamente de diversas formas como es el caso del artículo

(Durrant-Whyte y Bailey, 2006). Y aunque algunas de ellas han obtenido buenos resultados en la práctica siguen surgiendo problemas a la hora de buscar el método más rápido o el que genere un mejor resultado con menos índice de fallo. La búsqueda de algoritmos y métodos que resuelvan estos problemas sigue siendo una tarea pendiente.

Odometría visual

Dentro de las familias de técnicas pertenecientes a las de Visual SLAM se encuentra la de odometría visual, que es la que abordaremos en este trabajo. Consiste en la estimación del movimiento de la cámara en tiempo real. Es decir, el cálculo de la rotación y traslación de la cámara a partir de dos imágenes simultáneas. Se trata de una técnica incremental ya que se basa en la posición anterior para calcular la nueva.

Este tipo de algoritmos se suelen utilizar técnicas de extracción de puntos de interés, cálculos de descriptores y algoritmos para el emparejamiento. Normalmente el proceso es el mismo, una vez calculados los puntos emparejados se calcula la matriz fundamental o esencial y descomponerlas mediante SVD para obtener la matriz de rotación y traslación (RT) (Scaramuzza y Fraundorfer, 2011. Fraundorfer y Scaramuzza, 2012).

Uno de los trabajos más importantes en el ámbito es el de monoSLAM de Davison[1] (Andrew J. Davison y Stasse, 2007) que propone resolver este problema con una única cámara RGB como sensor y realizar el mapeado y la localización simultáneamente. El algoritmo propuesto por Davison utiliza un filtro extendido de Kalman para estimar la posición y la orientación de la cámara, así como la posición de una serie de puntos en el espacio 3D. Para determinar la posición inicial de la cámara es necesario a priori dotar de información con la posición 3D de por lo menos 3 puntos. Después el algoritmo es capaz de situar la cámara en el espacio tridimensional y de generar nuevos puntos para crear el mapa y servir como apoyo a la propia localización de la cámara. En la Figura 1.2 se pueden ver unas capturas de pantalla sobre uno de los experimientos realizados.

Es importante destacar también la trascendencia que ha tenido el trabajo PTAM (Klein y Murray, 2007) que viene a solucionar uno de los principales problemas que tienen los algoritmos monoSLAM; el tiempo de cómputo, ya que aumenta exponencialmente con el número de puntos (Figura 1.3). Para ello se aborda el problema separando el mapeado de la localización, de tal modo que solo la localización deba funcionar en tiempo real, dejando así que el mapeado trabaje de una manera asíncrona. Este algoritmo parte de la idea de que solo la localización es necesaria que funcione en tiempo real. PTAM hace uso de *keyframes*, es decir, fotogramas clave que se utilizan tanto para la localización como para el mapeado y también de una técnica de optimización mediante ajuste de haces, como en SfM.

1.3. Realidad aumentada

[1] http://www.doc.ic.ac.uk/~ajd/

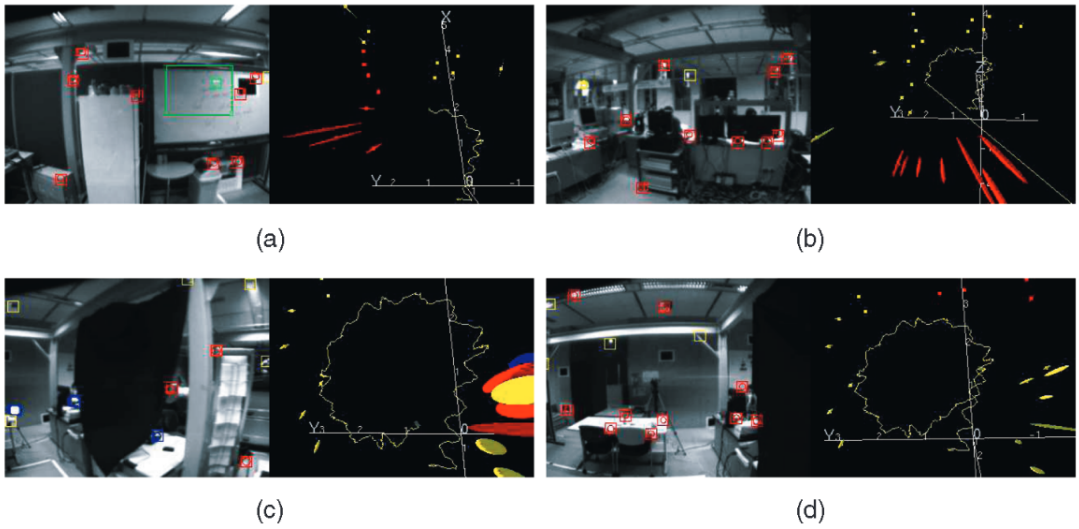

FIGURA 1.2: MonoSLAM: Un robot humanoide camina en una trayectoria circular de radio 0.75m. La estela amarilla muestra la trayectoria estimada del robot, y las elipses muestran los errores de localización.

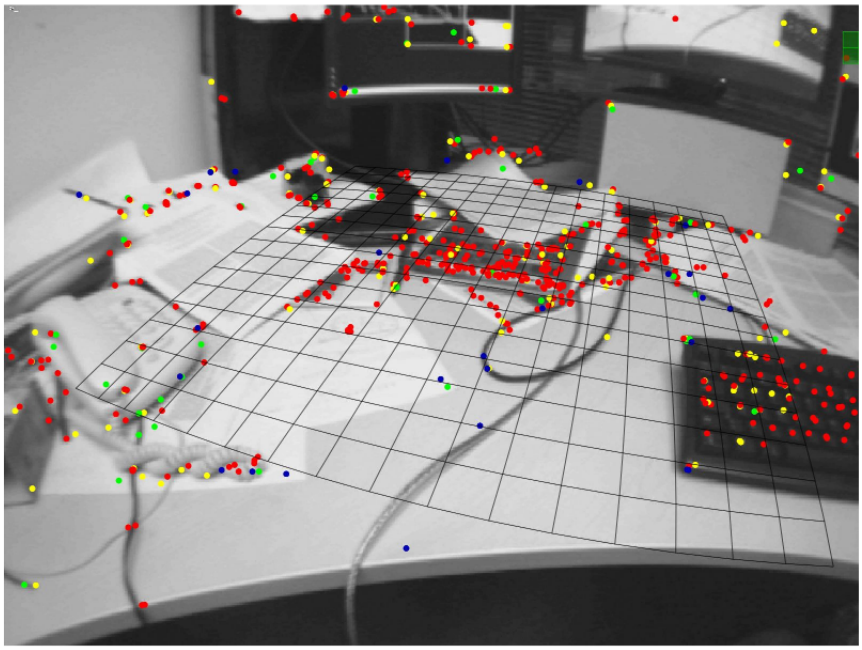

FIGURA 1.3: PTAM: Funcionamiento típico del sistema sobre un escritorio.

Capítulo 2

Infraestructura

En este capítulo se detallarán las herramientas base empleadas en la realización de este trabajo.

2.1. Sensores RGBD

Los sensores RGBD son capaces de captar a parte de las componentes roja, verde y azul de la luz, información de profundidad (o "D" depth en inglés). Es decir, por cada píxel asocia la información de color con su correspondiente componente de profundidad. Esta tecnología fue desarrollada por la empresa israelí **PrimeSense**. El sensor Kinect dispone también de un micrófono multiarray con el cual puede predecir de dónde proviene el sonido.

En el 2010 Microsoft sacó al mercado el sensor Kinect (Figura 2.2) para la consola de juegos Xbox 360 y Xbox One. Pronto se convirtió en uno de los dispositivos electrónicos más vendidos en todo el mundo después de su lanzamiento.

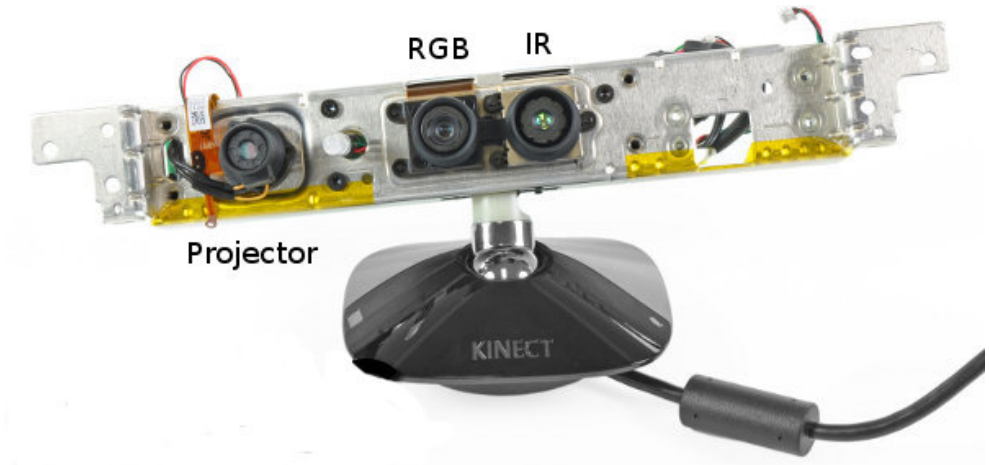

FIGURA 2.1: Sensor Microsoft Kinect

Este sensor salió al mercado a un precio mucho más reducido que algunos que existían antes que él por lo que el interés por este tipo de sensores se disparó y comenzaron a aparecen en diferentes áreas de la tecnología, como interfaces naturales de usuario (en inglés natural user interface, NUI), reconstrucción y realidad virtual o cartografía 3D.

CUADRO 2.1: Especificaciones técnicas del Asus Xtion PRO LIVE

Campo de visión:	58º H, 45º V, 70º D
Distancia de uso:	Entre 0.8m y 3.5m
Tamaño de la imagen de profundidad:	VGA (640x480) : 30 fps QVGA (320x240): 60 fps
Resolución:	SXGA (1280*1024)

El sensor utilizado en este trabajo es el **Asus Xtion PRO LIVE** que dispone de la misma tecnología comercializado por Asus, que proporciona profundidad, color y audio (utilizando un micrófono multiarray como el sensor Kinect). [1]

FIGURA 2.2: Asus Xtion PRO LIVE

Las especificaciones técnicas de este sensor se encuentran recogidas en la tabla 2.1

2.2. JDeRobot

JDeRobot es un proyecto desarrollado por el grupo de robótica de la Universidad Rey Juan Carlos [2]. Consiste en una plataforma de desarrollo de aplicaciones robóticas y de visión artificial. Está en su mayoría escrito en C++, donde disponen de una colección de componentes capaces de comunicarse a través de ICE middleware [3], los componentes pueden ejecutarse en diferentes ordenadores y pueden ser programados en diferentes lenguajes.

JdeRobot incluye numerosas herramientas, drivers, interfaces, librerías y tipos. Es *software* libre con licencia GPL y LGPL. También utiliza *software* de terceros como Gazebo, ROS, OpenGL, GTK y Eigen entre otros.

La versión de JdeRobot empleada ha sido la versión 5.4.0. A continuación se detallarán los componentes de JdeRobot que han sido de utilidad para la realización de este proyecto.

[1] https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/
[2] http://jderobot.org
[3] https://zeroc.com/products/ice

2.2.1. Biblioteca Progeo

Es una biblioteca de geometría proyectiva incluida en JdeRobot, que proporciona funciones muy útiles que relacionan puntos en dos y tres dimensiones.

Ha sido realmente útil en este trabajo para que a partir de puntos en dos dimensiones (píxeles) y su correspondiente información de profundidad (distancia), sacar los puntos relativos de la cámara en tres dimensiones.

Progeo usa el modelo de cámara **Pinhole**, en la Figura 2.3 se puede observar la representación geométrica de la retroproyección y la proyección. Este modelo es definido por unos parámetros intrínsecos y extrínsecos que definen la composición de todos los parámetros iniciales de configuración de la cámara. Los parámetros extrínsecos que establecen la posición 3D, foco de atención (foa) y roll, mientras que los parámetros intrínsecos determinan la distancia focal y el centro óptico o píxel central.

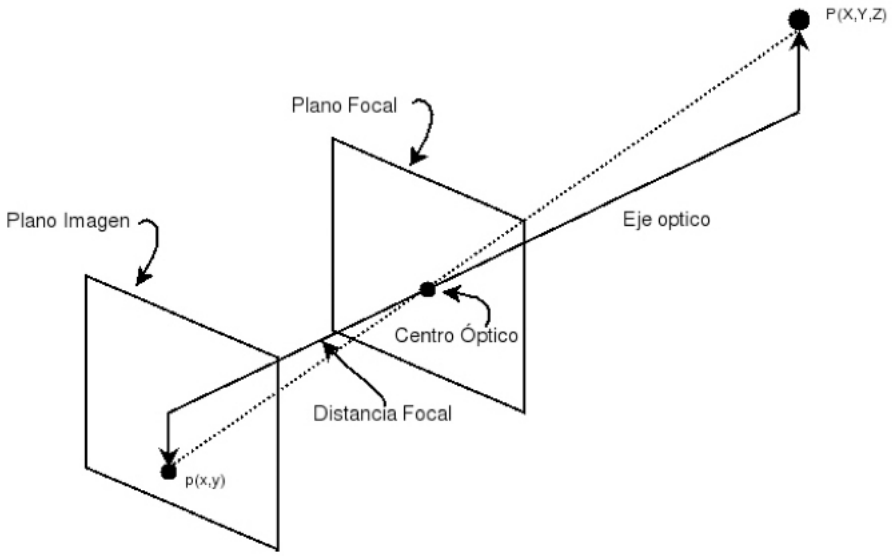

FIGURA 2.3: Modelo de cámara Pinhole

Las funciones que proporciona esta biblioteca son las siguientes:

- **Project**: Esta función permite proyectar un punto 3D del mundo al correspondiente pixel en 2D de la imagen de la cámara.
- **Backproject**: Esta función es capaz de a partir de las coordenadas de un píxel en 2D, obtener la línea de proyección que conecta la cámara y el foco con el rayo 3D que proyecta dicho píxel en el plano imagen. Con esto y conociendo la distancia real del punto 3D a calcular, se calcula las coordenas reales del punto 3D.
- **DisplayLine**: Esta función permite conocer si una línea definida por dos puntos en 2D es visible dentro del plano imagen.
- **Display_info**: Esta función muestra toda la información sobre la cámara utilizada.

2.2.2. Biblioteca parallelIce

Es otra librería incluída en JdeRobot, que soluciona el problema de latencia de información proveniente de los diferentes drivers, evitando la espera y proviniendo de un acceso asíncrono a una copia en local de las interfaces con muy bajo tiempo de procesado.

2.2.3. Servidor OpenniServer

OpenniServer es un driver que se comporta como un servidor y es capaz de proporcionar con un sensor RGBD (Kinect o Xtion), imágenes de color, de profundidad o nubes de puntos que son enviados a través de la interfaz ICE a un puerto específico, donde se pueden escuchar los datos. Este driver es el que se necesita para el funcionamiento de este trabajo ya que es desde donde se recogen tanto las imagenes de color (RGB) como las de profundidad (Depth) para su posterior procesado.

2.2.4. Herramienta RGBDViewer

Es una herramienta que permite enseñar la información proveniente de los sensores RGBD con openniServer como forma de visualización de los datos; imágenes RGB, DEPTH o nubes de puntos.

El funcionamiento corresponde a un hilo de ejecución llamado Control que se encarga de recolectar las imágenes provenientes del driver, una clase Shared para guardar y recoger los datos, y por último una clase Gui que se encargará de coger los datos (imágenes y nubes de puntos) guardados en Shared y mostrarlas.

Esta herramienta a servido como referencia para la realización de este trabajo. En la Figura 2.4 podemos ver una captura de pantalla con las diferentes visualizaciones.

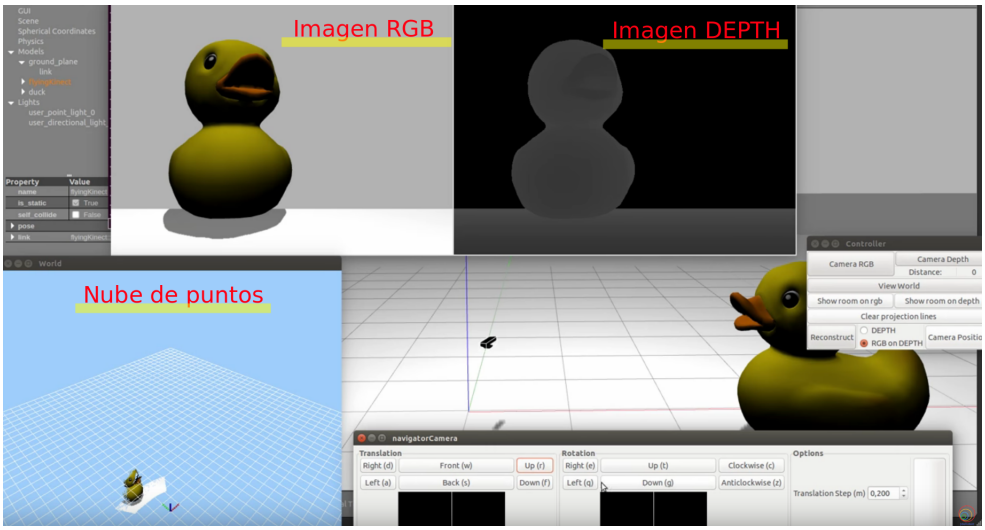

FIGURA 2.4: Captura de pantalla del con las tres vistas de los diferentes datos, imagen de color, de profundidad y nube de puntos.

2.2.5. Pose3D

Es una interfaz que define una posición en tres dimensiones (x, y, z, h) y una orientación con un cuaternión (q0, q1, q2, q3).

2.3. Biblioteca ICE de comunicaciones

ICE (Internet Communications Engine) es un RPC framework desarrollado por ZeroC con soporte en C++, C#, Java, JavaScript y Python entre otros. Se encuentra bajo doble licencia GNU GPL y código cerrado. Actúa como plataforma de comunicaciones y funciona bajo TCP/IP. [4]

En JdeRobot la podemos encontrar como librería y es utilizada como protocolo de comunicaciones entre los diferentes componentes de JdeRobot. En nuestro trabajo se ha usado la versión 3.5.1 y nos ha servido para establecer la comunicación entre el componente y el driver del sensor, recogiendo las imágenes de éste.

2.4. Biblioteca Point Cloud Library (PCL)

PCL es una librería desarrollada en C++ para el procesamiento de imágenes 2D/3D y nubes de puntos. Está publicada con licencia BSD y libre bajo usos comerciales y de investicación. Está financialmente soportada por un consorcio de companías comerciales y su propia organización sin ánimo de lucro, **Open Perception**. A parte de los donadores y contribuidores individuales que aportan al proyecto. [5]

Para simplificar el uso y el desarrollo, esta librería se encuentra dividida en módulos individuales de los que destacan el filtrado de puntos *outliers* o de ruido, estructuras de datos, estimación 3D, algoritmos para la detección de puntos de interés, combinación, segmentación, algoritmos para el reconocimiento de objetos.

En su página web disponen de mucha información y ejemplos prácticos que ayudan mucho a la compresión de todas las funcionalidades de esta librería. PCL también dispone de una librería i/o de entrada y salida para leer o crear nubes de puntos a partir de diferentes dispositivos, así como visualizadores 3D.

2.5. Biblioteca OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de código abierto que fue desarrollada para proporcionar una infraestructura común en aplicaciones de visión artificial y facilitar la inteligencia máquina, con mecanismos de aprendizaje y de interpretación de datos. Con licencia BSD da facilidades para su uso y su modificación bajo fines comerciales. [6]

[4] https://zeroc.com/products/ice
[5] http://pointclouds.org/
[6] http://opencv.org/

La librería contiene más de 2500 algoritmos. Estos algoritmos pueden ser usados para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas determinadas en vídeos, seguimiento del movimiento de cámaras, seguimiento de objetos, extraer modelos de objetos 3D, producir nubes de puntos a partir de cámaras, encontrar imágenes similares de un conjunto, juntar trozos de imágenes para producir una imagen final con más resolución, etc... OpenCV tiene más de 47 miles de usuarios en la comunidad, excediendo los 14 millones de descargas, la librería es usada ampliamente en empresas, grupos de investigación y organismos gubernamentales.

OpenCV a sido diseñada de forma eficiente y con un fuerte enfoque en aplicaciones de tiempo real. Escrita en C/C++, la librería obtiene las ventajas del procesamiento multi-núcleo. Dispone de interfaces en C++, C, Python, Java y MATLAB y es soportada por diferentes sistemas operativos como Windows, Linux, Android y Mac OS.

En este trabajo se ha utilizado la versión 2.4.8 y se ha usado a la hora de identificar puntos de interés y para los distintos métodos de emparejamiento. En la Figura 2.5 se puede apreciar un ejemplo de su uso en la detección de puntos de interés sobre un entorno real.

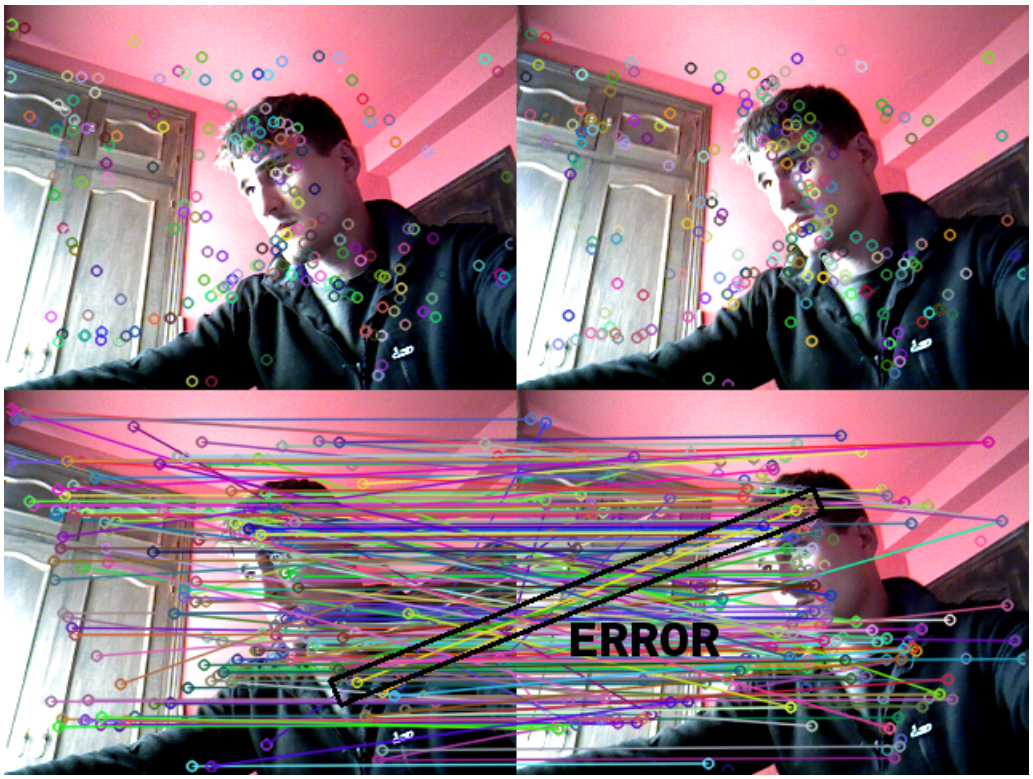

FIGURA 2.5: Detección y emparejamiento de puntos de interés con OpenCV usando SIFT.

2.6. Biblioteca Eigen

Eigen es una librería de algebra lineal que permite hacer operaciones aritméticas con matrices y vectores, a través de los operadores comunes de C++, tales como +, -, *

o a través de métodos especiales tales como dot(), cross(), etc... Para la clase *Matrix* (matrices y vectores) los operadores solo soportan operaciones de álgebra lineal. En Figura 2.6 se puede ver un ejemplo de como de simple es hacer una multiplicación y una división por un escalar. [7]

Example:

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;

int main()
{
  Matrix2d a;
  a << 1, 2,
       3, 4;
  Vector3d v(1,2,3);
  std::cout << "a * 2.5 =\n" << a * 2.5 << std::endl;
  std::cout << "0.1 * v =\n" << 0.1 * v << std::endl;
  std::cout << "Doing v *= 2;" << std::endl;
  v *= 2;
  std::cout << "Now v =\n" << v << std::endl;
}
```

Output:

```
a * 2.5 =
2.5   5
7.5  10
0.1 * v =
0.1
0.2
0.3
Doing v *= 2;
Now v =
2
4
6
```

FIGURA 2.6: Ejemplo de una multiplicación y división por un escalar con Eigen.

Eigen es *software* libre y desde la versión 3.1.1 tiene licencia MPL2 (LGPL3+ para las anteriores versiones). Se ha usado la versión 3.2.0 y ha sido de utilidad en este proyecto para realizar los cálculos de la matriz RT a través del los vectores de puntos 3D ya emparejados.

2.7. Biblioteca de interfaz gráfica GTK+

GTK+, o the GIMP Toolkit es una herramienta multiplataforma de creación de interfaces gráficas. Es multiplataforma y está escrito en C, pero a sido diseñado para tener soporte para un gran rango de lenguajes, tales como Perl y Python. GTK++ tiene una gran colección de *widgets* y interfaces para usar en la aplicación, tales como ventanas, botones, selectores, cajas de texto, etc.

La versión utilizada ha sido la 3.10.8. Es *software* libre y parte del proyecto GNU. Con licencia LGPL, permite que sea utilizado por todos los desarrolladores, incluyendo aquellos que están desarrollando un *software* privativo. GTK+ ha sido utilizado en muchos proyectos y en grandes plataformas. [8]

2.7.1. Glade

Glade es una *RAD tool* (Rapid Application Development Tool) que permite desarrollar de manera fácil y rápida interfaces de usuario en GTK+ para el entorno de escritorio GNOME. La interfaz gráfica diseñada en Glade es guardada en un XML

[7] http://eigen.tuxfamily.org/
[8] https://www.gtk.org/

que usando los objetos GTK+ de **GtkBuilder** pueden ser cargados y utilizados por aplicaciones de forma dinámica como se ha hecho en este trabajo. [9]

2.8. OpenGL

OpenGL es el principal entorno para el desarrollo de aplicaciones gráficas 2D y 3D interactivas. Desde 1992, OpenGL se ha convertido en la interfaz de aplicaciones gráficas más utilizada y soportada en la industria 2D y 3D, con miles de aplicaciones diponibles en diferentes plataformas. OpenGL ayuda al desarrollo de aplicaciones al incorporar un amplio conjunto de renderizado, mapeo de texturas, efectos especiales y otras potentes funciones de visualización. Se puede usar OpenGL en la mayoría de entornos de escritorio y diferentes plataformas. Es muy utilizada y conocida en la industria de los videojuegos.

Algunas de las ventajas de las que presume OpenGL son; que es un estándar de la industria, con soporte, multiplataforma y el único libre. Es estable, dispone de compatibilidad hacia atrás, escalable, fácil de usar y bien documentado. [10]

Se ha usado la librería **Mesa 3D Graphics** en linux que es una implementación de la especificación de OpenGL con código abierto [11].

OpenGL en este trabajo se ha usado para visualizar la posición de la cámara, su estela y la colección de nubes de puntos obtenida y procesada de las imágenes RGB y de profundidad. En la Figura 2.7 se puede apreciar una captura de pantalla con la posición de la cámara dibujada en el espacio tridimensional con el visualizador utilizado con OpenGL.

[9] https://glade.gnome.org/
[10] https://www.opengl.org/
[11] https://www.mesa3d.org/

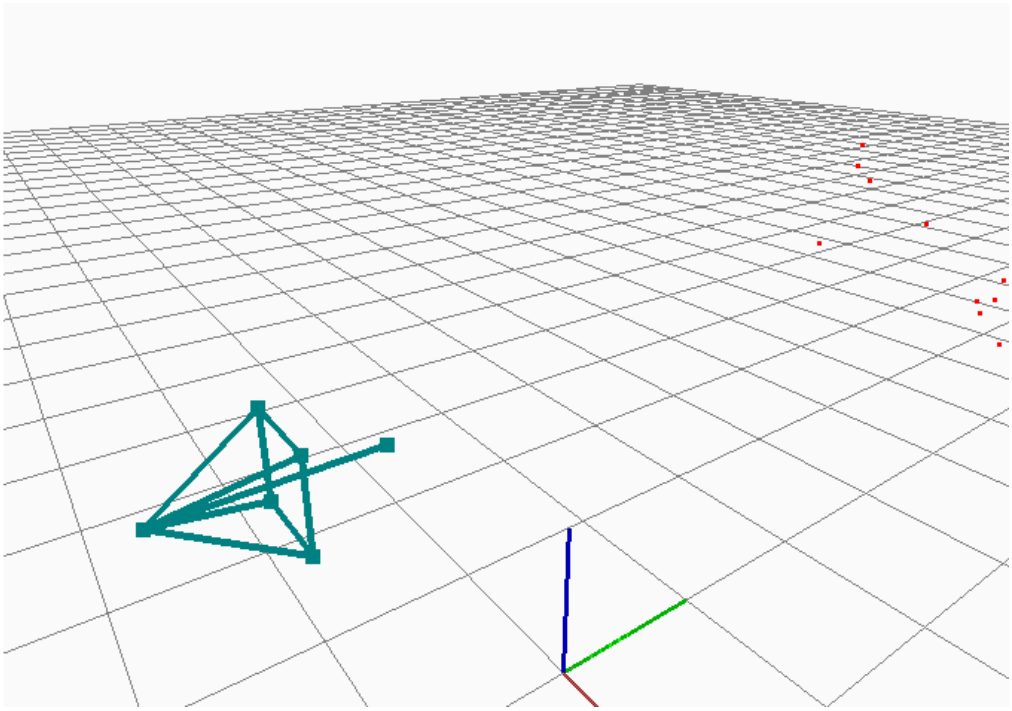

FIGURA 2.7: Captura de la posición de la cámara en el visualizador 3D con OpenGL.

Bibliografía

Andrew J. Davison Ian D. Reid, Nicholas D. Molton y Olivier Stasse (2007). «MonoSLAM: Real-Time Single Camera SLAM». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(6), págs. 1052-1067.

Durrant-Whyte, H. y T. Bailey (2006). «Simultaneous localization and mapping: part I». En: *IEEE Robotics Automation Magazine* 13, issue 2, págs. 99-110.

Fraundorfer, Friedrich y Davide Scaramuzza (2012). «Visual odometry: Part ii: Matching, robustness, optimization, and applications». En: *Robotics Automation Magazine, IEEE* 19(2), págs. 78-90.

Klein, G. y D. Murray (2007). «Parallel Tracking and Mapping for Small AR Workspaces». En: *6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, págs. 225-234.

Noah Snavely, Steven M. Seitz y Richard Szeliski (2006). «Photo tourism: Exploring photo collections in 3d». En: *SIGGRAPH Conference Proceedings*, págs. 835-846.

Scaramuzza, Davide y Friedrich Fraundorfer (2011). «Visual odometry [tutorial]». En: *Robotics Automation Magazine, IEEE* 18(4), págs. 80-92.