UNIVERSIDAD DE PIURA

FACULTAD DE INGENIERÍA

Sistema de posicionamiento *indoor* para el guiado de robots móviles implementado en *Robot Operating System* (ROS)

Tesis para optar el Título de
Ingeniero Mecánico - Eléctrico

Jayro Zaid Paiva Mimbela

Asesor:
Mgtr. Ing. Juan Carlos Soto Bohórquez

Piura, octubre de 2019


UNIVERSITAS STUDIORUM PIURENSIS

A mis padres y hermana, por el sacrificio y apoyo constante.


UNIVERSITAS STUDIORUM PIURENSIS

Resumen Analítico-Informativo

Sistema de posicionamiento *indoor* para el guiado de robots móviles implementado en *Robot Operating System* (ROS)
Jayro Zaid Paiva Mimbela
Asesor(es): **Mgtr. Ing. Juan Carlos Soto Bohórquez**
Tesis.
Ingeniero Mecánico Eléctrico
Universidad de Piura. Facultad de Ingeniería.
Piura, octubre de 2019

Palabras claves: espacios inteligentes, posicionamiento en interiores, visión por computador, seguimiento y detección de marcadores, ROS

Introducción: Actualmente la tecnología de posicionamiento global (GPS) es empleada tanto a nivel miliar como comercial; sin embargo, uno de los principales problemas de este sistema es su mal funcionamiento en espacios cerrados, imposibilitando su uso en aplicaciones como la robótica de servicio en oficinas, laboratorios, hospitales, almacenes, hogares, etc.

Metodología: Para dar solución al problema identificado, se plantea el desarrollo de un sistema de posicionamiento *indoor* basado en el concepto de espacios inteligentes. Para ello, se desarrolla un sistema de detección y seguimiento que en complemento con un sistema de conversión de espacios 2D a 3D, permite obtener la posición de un objeto en el espacio a través de un marcador. Finalmente, las pruebas y validación del sistema son llevadas a cabo en un entorno construido para los fines de la presente tesis.

Resultados: De 121 puntos empleados para pruebas y validación, el sistema posiciona al marcador con un error aceptable, además de reconstruir fielmente la trayectoria real que sigue.

Conclusiones: Se consigue desarrollar, con un error aceptable, un sistema capaz de posicionar objetos en el espacio a través de marcadores, y de reconstruir fielmente cualquier trayectoria que siga. El error del sistema es producto de emplear parámetros de la cámara aproximados, y de no considerar la distorsión del lente óptico en la formulación matemática para obtener la posición. El alcance del sistema depende del rango visual de las cámaras. El sistema desarrollado abre el camino a estudios biomecánicos empleando captura de movimiento, desarrollo de sistemas de control de trayectorias, reconstrucción geomorfológica, escaneo de piezas mecánicas o partes biomédicas, etc.

Fecha de elaboración del resumen: 17 de octubre de 2019

Analytical-Informative Summary

Indoor positioning system for mobile robot guidance implemented in Robot Operating System (ROS)
Jayro Zaid Paiva Mimbela
Advisor: Mgtr. Ing. Juan Carlos Soto Bohórquez
Thesis
Mechanical Electrical Engineer
Universidad de Piura. Facultad de Ingeniería.
Piura, october 2019

Keywords: intelligent spaces, indoor positioning, computer vision, marker detection and tracking, ROS

Introduction: Currently, global positioning technology (GPS) is used both at military and commercial level; however, one of the main problems of this system is its malfunction in closed spaces, making it impossible to use it in applications such as service robotics in offices, laboratories, hospitals, warehouses, homes, etc.

Methodology: To solve the problem identified, is proposed the development of an indoor positioning system based on the concept of intelligent spaces. For this, is developed a detection and tracking system that in complement with a 2D to 3D space conversion system, allows to obtain the position of an object in the space through a marker. Finally, the tests and validation of the system are carried out in an environment built for the purposes of this thesis.

Results: Of 121 points used for testing and validation, the system positions the marker with an acceptable error, in addition to faithfully reconstruct the actual trajectory that follows.

Conclusions: It gets developed, with an acceptable error, a system capable of positioning objects in space through markers and faithfully reconstruct any trajectory that follows. The system error is the product of using approximate camera parameters, and of not considering the distortion of the optical lens in the mathematical formulation to obtain the position. The scope of the system depends on the visual range of the cameras. The developed system opens the way to biomechanical studies using motion capture, development of trajectory control systems, geomorphological reconstruction, scanning of mechanical or biomedical parts, etc.

Summary date: october 17, 2019

Prefacio

Un estudiante de ingeniería, a lo largo de su vida académica, cursa una serie de asignaturas que le brindan el conocimiento necesario para poder afrontar los retos laborales a futuro. La presente tesis, en mención a estas asignaturas, surge como un trabajo semestral cuyo desarrollo en profundidad despierta en el autor un sumo interés en robótica, siendo el principal motivo para llevar el trabajo realizado a un trabajo de tesis, puesto que llevarlo a cabo amerita ahondar en temas cuya relación con esta ciencia es muy estrecha, así como el aprendizaje de herramientas computacionales, tales como *Robot Operating System*, visión artificial, C++, Python, etc.

Por último, el autor desea que el material puesto a disposición sea de utilidad para aquellos que tengan la intención de llevar a cabo proyectos que impliquen un estudio profundo en robótica, ya que su desarrollo a nivel regional y nacional es mínimo. Por ello, la importancia de dar los primeros pasos para que pueda ser vista como una alternativa de solución a los distintos problemas que tienen lugar en la región y el país. El autor anima al lector a ser partícipe de una de las ciencias más desarrolladas en los últimos años, y de mayor trascendencia e impacto en el mundo.


UNIVERSITAS STUDIORUM PIURENSIS

Tabla de contenido


UNIVERSITAS STUDIORUM PIURENSIS

Lista de tablas


UNIVERSITAS STUDIORUM PIURENSIS

Lista de figuras

Introducción

El sistema GPS (*Global Positioning System*), diseñado por el Departamento de Defensa de los Estados Unidos con fines militares, permite calcular la posición de un objetivo, en toda la extensión de la tierra, a través del cálculo de distancias del objetivo a un mínimo de tres satélites cuya localización es conocida (García, y otros, 2000). Actualmente, a pesar de tener origen militar, está siendo empleado a nivel comercial en aplicaciones, tales como el transporte de personas y mercancías, agricultura de precisión, topografía, operaciones de rescate, etc. Con respecto a la precisión, que es del orden de los metros, puede mejorar usando dispositivos multifrecuencia y aplicando correcciones diferenciales (Blog CARTIF, 2016).

Sin embargo, uno de los principales problemas de este sistema es su mal funcionamiento en espacios cerrados, puesto que la estructura que los rodea no permite que las señales satelitales puedan atravesar y localizar al objetivo (Cliement, 2015); en consecuencia, se imposibilita su uso en aplicaciones como la robótica de servicio en instalaciones como oficinas, laboratorios, hospitales, almacenes, hogares, etc. La robótica de servicio, según ISO TC 184/SC2, contempla el uso de robots que operan de forma parcial o totalmente autónoma con el fin de brindar bienestar a los seres humanos (AER, s.f.).

En vista a este inconveniente, es que a lo largo de los años han surgido sistemas denominados como sistemas de posicionamiento en interiores (IPS), los cuales buscan nuevas alternativas para la localización de objetivos específicos en espacios cerrados que, a diferencia de los sistemas de posicionamiento global, emplean diversas tecnologías (Blog CARTIF, 2016). Ejemplo de ello son (Santiago Pé, 2007), (Fernández Lorenzo, 2005), (Pizarro, Santiso, & Mazo, 2006), (Martín, Melcón, & Tapia, 2009), (Losada Gutiérrez C. , 2010) (Losada Gutiérrez C. , 2010), (Losada Gutiérrez, Mazo, Palazuelos, Pizarro, & Marron, 2011), (Martínez Novo, 2015), (Calvillo Ardila, 2017).

La presente tesis, en relación con los mencionados, tiene como objetivo desarrollar una alternativa de solución al posicionamiento de robots móviles u objetivos específicos en espacios cerrados. El fin de su desarrollo es que pueda ser empleado en robótica de servicio, ya sea usando robots terrestres o aéreos.

En el Capítulo 1 se realiza el estudio de las bases conceptuales centradas en los *espacios inteligentes* y el estado del arte de los trabajos previos. El Capítulo 2 describe el hardware, software y algoritmos a emplear para el desarrollo de un sistema de detección y seguimiento. Complementario al sistema mencionado, el Capítulo 3 describe los algoritmos y conceptos a emplear para desarrollar un sistema que permite obtener la posición de un objeto a partir de dos imágenes. El Capítulo 4 describe el proceso de integración de los sistemas a desarrollar en *Robot Operating System* (ROS). En el Capítulo 5 se muestran las pruebas llevadas a cabo, y la validación del sistema en un entorno de pruebas construido y, finalmente, se presentan las conclusiones, planteamiento de futuros trabajos y recomendaciones.

Capítulo 1
Aspectos Generales

1. Introducción

Este primer capítulo abarca el estudio teórico sobre los espacios inteligentes, mismo que proporciona la teoría base para llevar a cabo la presente tesis. Se detalla cómo están estructurados y los tipos de sistemas de posicionamiento, que dependiendo del sensor empleado para la obtención de la posición, se les pueden implementar. Más adelante, se realiza un estudio más detallado acerca de los sistemas de posicionamiento basados en la detección de objetos tridimensionales en el espacio mediante cámaras, y se presenta el estado del arte con un listado de los trabajos realizados por diversos autores, los cuales guardan una estrecha relación con la presente tesis.

2. Espacios inteligentes

Se define a los espacios inteligentes como ambientes o espacios que tienen la capacidad de percibir y entender los posibles eventos que ocurran en su interior. Estos espacios se encuentran equipados con diversos tipos de sensores, tales como cámaras, ultrasonidos, micrófonos, etc.; siendo los encargados de la percepción de los eventos. Los eventos percibidos se analizan y provocan una reacción del espacio inteligente ante ellos, de tal modo que las acciones que tome el mismo son comunicadas a través de actuadores, pantallas, portavoces, proyectores, etc.

Dicho esto, se puede decir que los espacios inteligentes poseen la capacidad de observar a los usuarios (personas) que se encuentran en su interior, analizar las acciones que realizan y dar una respuesta de acción en base a ellas. Sin embargo, los espacios inteligentes no solo se centran en personas, sino que también son capaces de brindar soporte a robots móviles, puesto que debido a la interacción que tienen con el espacio, son dotados de una mayor inteligencia que les permite poder cumplir la función de actuador en un espacio inteligente, brindando servicio a los usuarios.

Se concluye que un espacio inteligente es un espacio o ambiente que deja de ser pasivo para ser un espacio sensitivo con capacidad de actuar frente a diversos eventos que tengan lugar en su interior; por ello, que el concepto de espacios inteligentes sea aplicado en distintos entornos, tales como oficinas, hospitales, almacenes, fábricas, hogares, etc. (Joo-Ho & Hiroshi, 2002).

2.1. Estructura del espacio inteligente. Para comprender la estructura de los espacios inteligentes es necesario definir los siguientes conceptos (Santiago Pé, 2007):

2.1.1. *Entorno*. Entidad fija en el espacio en donde tienen lugar una serie de eventos, los cuales son analizados y comprendidos por el espacio inteligente. A su vez, el entorno está dividido en las denominadas *capas funcionales.*

2.1.1.1. *Capa pasiva de percepción.* Sistema comprendido por una serie de sensores como cámaras, sensores láser, ultrasonidos, etc.; los cuales permiten obtener información sobre los eventos que ocurren dentro del entorno. Los sensores pueden encontrarse fijos en el espacio o unidos a la siguiente capa funcional.

2.1.1.2. *Capa de interacción.* Capa comprendida por un número de agentes controlables que tienen la capacidad de interactuar físicamente con el entorno. Si bien esta capa es utilizada para cumplir con los objetivos para los cuales fue diseñado el espacio inteligente, permite que pueda ser empleada como capa activa de percepción; es decir, una capa de percepción comprendida de sensores que no se encuentran fijos en el espacio, por ejemplo, un robot móvil. De ser así, los agentes controlables disponen de un sistema sensorial que les permite recoger información adicional del entorno para ser usada como complemento a la ya brindada por la capa pasiva de percepción.

2.1.1.3. *Capa de comunicación.* Red de comunicación que establece una conexión entre el entorno, agentes controlables; y eventualmente, *agentes autónomos* (personas). Representa el sistema nervioso de los espacios inteligentes; a través de ella, la capa de percepción (pasiva o activa) y de interacción establecen una conexión con la siguiente capa.

2.1.1.4. *Capa de inteligencia.* Sistema encargado del procesamiento y gestión de toda la información recogida por la capa de percepción (pasiva o activa), generando una base de datos que permite el intercambio de esta información con los agentes que pertenecen a la capa de interacción.

2.1.2. Mundo de percepción. La capa de percepción e interacción guarda una relación con los eventos que ocurren dentro del entorno, los cuales pueden ser llevados a cabo por los siguientes agentes (Santiago Pé, 2007):

2.1.2.1. *Agentes autónomos.* Agentes con libertad de ejercer cualquier tipo de actividad en el entorno, debido a que éste no tiene control sobre ellos. Pueden llegar a ser un obstáculo no estático que tenga que ser evitado por los agentes controlables.

2.1.2.2. *Agentes estáticos.* Como su nombre menciona, permanecen inertes en el entorno; sin embargo, deben ser parte de la información recogida por la capa de percepción con el fin de desarrollar una mejor interacción y cumplimiento de los objetivos del espacio inteligente.

2.1.2.3. *Agentes controlables.* Agentes conocidos por el entorno a través de la capa de comunicaciones. Forman parte de la capa de interacción y son vistos por la capa de percepción como un ente que cuenta con una firma o apariencia, debido a que están presentes en el entorno.

En la Figura 1 se muestra la estructura de un espacio inteligente cuya capa de percepción está compuesta por una red de cámaras; y que a su vez, cuenta con la presencia de un robot móvil como agente controlable y de una persona como usuario o agente autónomo.

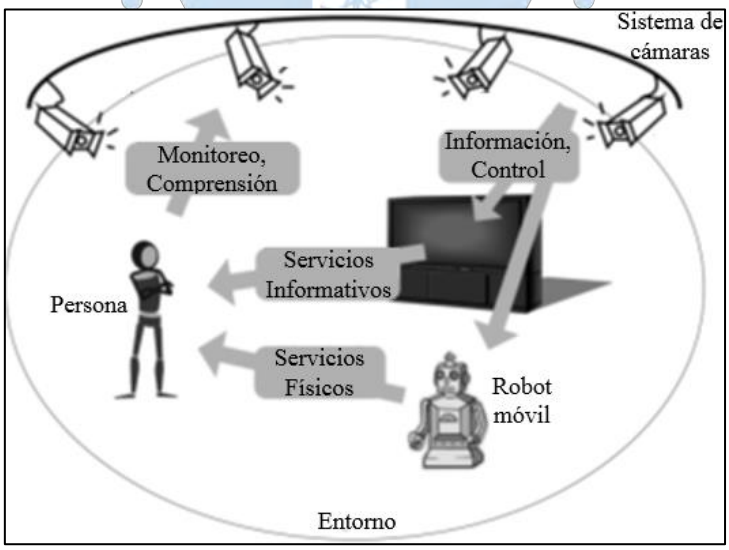

Figura 1. Estructura de un espacio inteligente que considera a un robot móvil como agente controlable.
Fuente: (Joo-Ho & Hiroshi, 2002).
Elaboración: Propia.

En conclusión, según la estructura expuesta y en relación con el objetivo de la presente tesis, el espacio inteligente a desarrollar consta de una capa pasiva de percepción cuya información del entorno viaja a través de la capa de comunicación hasta la capa de inteligencia, en donde el resultado del procesamiento de la información recibida sea la posición en el espacio de un robot móvil que pueda llegar a ser un agente controlable.

2.2. Sistemas de posicionamiento en espacios inteligentes. Como se ha expuesto, parte esencial de la estructura de los espacios inteligentes es la capa de percepción; esto es, el sistema de sensores que permite obtener información de los eventos que tienen lugar en el entorno, la cual es empleada para obtener la posición de un agente controlable en el espacio. A continuación, se hace mención de los sistemas de posicionamiento que generalmente son implementados para cumplir con este objetivo (Joo-Ho & Hiroshi, 2002).

2.2.1. Sistema de posicionamiento con cámaras. El campo de la visión artificial viene ya desarrollándose desde la década de los 60 con la idea de establecer conexión entre una cámara de video y un computador; por ello, ya en la actualidad los sistemas de detección de humanos y objetos mediante cámaras han sido ampliamente estudiados.

Las cámaras representan un sensor económico y de fácil implementación, razón por la cual han sido empleadas con frecuencia en diversos trabajos de investigación sobre espacios inteligentes; sin embargo, la complejidad de los posibles espacios a observar, los efectos causados por la variación de luminosidad, la pequeña resolución a medida que las cámaras se alejan del objetivo a detectar, la necesidad de una previa calibración, etc.; hacen que la sola implementación de estos sensores no permita el desarrollo de un sistema robusto y de fácil uso para la detección de humanos y objetos. Para conseguir un sistema de posicionamiento robusto, es necesario, adicional a las cámaras, un grupo sensorial extra (Joo-Ho & Hiroshi, 2002).

2.2.2. Sistema de posicionamiento con ultrasonidos, Indoor GPS, etc. Sistemas basados en diversos portadores de señales como ondas de radio o de luz, los cuales a menudo suelen ser llamados “GPS de interiores” debido a la similitud que tienen con el sistema GPS. En analogía con los satélites, estos sistemas de posicionamiento usan dispositivos externos con ubicación conocida para determinar la posición absoluta de un objetivo.

Sistemas similares existen, pero basados en sensores ultrasonido, tales como *Active Bat*[1], *Cricket*[2] y el sistema de posicionamiento de zonas de *Furukawa Electric*, *Ltd*., instalado en el Instituto de Ciencia Industrial de la Universidad de Tokio. La principal característica de este tipo de sistemas es la precisión relativamente alta que ofrecen; sin embargo, suelen ser uno de los sistemas menos atractivos para la implementación en entornos reales debido a su precio bastante elevado; necesidad de instalar múltiples dispositivos en el espacio, al igual que una calibración precisa de los mismos; y, como sucede en los sistemas de radiofrecuencia, la necesidad del uso de etiquetas.

2.2.3. Sistema de posicionamiento con buscadores de alcance láser. Los buscadores de alcance láser o LIDARs son dispositivos capaces de determinar la distancia a un objeto mediante el uso de rayos láser. Estos dispositivos ya llevan regular tiempo en el mercado, pero debido a la aparición de ejemplares con un precio relativamente más bajo, se hizo más frecuente su uso, en especial en tareas como mapeo y topografía. Así mismo, al no ser necesario el uso de etiquetas y ser un sistema de fácil instalación, se ha convertido en un sensor estándar a implementar a bordo de robots móviles.

Si bien parecen ser una buena opción para la detección de objetos en espacios inteligentes, los mismos presentan problemas de oclusión[3] debido al tipo de distribución que tienen en el espacio, que es de unos 20 cm por encima del suelo, haciendo que la posición de un ser humano deba estimarse en función de sus dos piernas, lo cual resulta complejo cuando se realiza una exploración. En la Tabla 1 se presenta un cuadro comparativo entre los distintos sistemas de posicionamiento que pueden ser implementados en un espacio inteligente según el sensor empleado.

Tabla 1. Comparación entre sistemas de posicionamiento según el sensor empleado.

Sensor	Ventajas	Desventajas
Cámara	Barato. Fácil de instalar.	No robusto en ambientes complejos.
RFID	Barato.	No Preciso. Se necesita de etiquetas.

[1] *Active Bat* es un sistema inalámbrico de ubicación en interiores de baja potencia con una precisión de hasta 3cm. Está basado en el principio de trilateración, y en múltiples receptores ultrasónicos, incrustados en el techo, que miden el tiempo de vuelo.
[2] Cricket es un sistema de ubicación en interiores para entornos informáticos, proporciona información detallada de ubicación (identificadores de espacio, coordenadas de posición y orientación) para aplicaciones que se ejecutan en computadoras de mano, computadoras portátiles y nodos de sensores.
[3] Obstrucción parcial del objetivo a detectar.

Sensor	Ventajas	Desventajas
Ultrasonido, iGPS	Preciso.	Bastante caro. Se necesita de etiquetas.
Sensores de Presión	No intrusivo. Preciso.	Bastante caro.
IMU	Medición directa de la velocidad y orientación	Acumulación del error. Debe estar equipado a bordo.
LIDARs	Preciso.	Oclusiones. Bastante caro.

Fuente: (Joo-Ho & Hiroshi, 2002).

2.3. Posicionamiento de objetos tridimensionales mediante cámaras. Descritos los sistemas de posicionamiento generalmente usados en espacios inteligentes, se opta por la implementación de un sistema de posicionamiento mediantes cámaras debido a que es de bajo costo y brinda las prestaciones suficientes para cumplir con el objetivo de la presente tesis. El sistema debe permitir la detección, seguimiento y posicionamiento final de objetos tridimensionales a partir de una serie de imágenes recopiladas del entorno, teniendo como objetivo conocer, en tiempo real, la ubicación en el espacio de un robot móvil que actúa en el entorno como agente controlable.

Dicho esto, como primer paso es necesario encontrar un método que permita detectar la proyección de un objeto en la imagen capturada por una cámara. Algunos de estos métodos se mencionan a continuación:

2.3.1. Detección de objetos basado en marcas artificiales. En el campo de la visión artificial es una técnica clásica el uso de marcas artificiales, extendiéndose su uso en campos como la fotogrametría[4]. Para cumplir con el objetivo que se persigue mediante esta técnica, las marcas deben tener las siguientes características (Santiago Pé, 2007):

- Cada marca artificial proyectada en el plano imagen debe ser unívoca, de manera que no existan problemas al determinar, en base a la proyección en la imagen, a cuál de las marcas pertenece la proyección.

- Una marca artificial debe ser extraída de manera sencilla y robusta en la imagen.

[4] Técnica que permite la obtención de mapas y planos de una extensión grande de terreno por medio de la fotografía área.

- El diseño de las marcas artificiales debe ser tal que garantice una suficiente precisión en los cálculos realizados en el proceso de detección y localización.

Generalmente, la distancia relativa entre marcas es un dato conocido; por ello, la razón de su uso es asegurar un proceso de correspondencia y detección robusta. Con respecto al tipo de marcas artificiales empleadas, se pueden distinguir dos tendencias:

2.3.1.1. *Marcas artificiales puntuales.* Basado en el uso de marcas que corresponden a regiones circulares. Una de las ventajas que ofrecen es la relativa apariencia invariante que presentan ante deformaciones proyectivas, y una muy estable localización de la marca a partir de la obtención de su centroide.

2.3.1.2. *Marcas artificiales planares.* En este caso, las marcas no están basadas en una correspondencia puntual, sino que consisten de planos de los cuales se realiza un cálculo directo de su posición.

2.3.2. Detección de objetos mediantes marcas naturales. De lo mencionado en **2.3.1**, se hace notar que se requiere un cierto grado de intervención en el objeto a detectar, además de ser necesaria una previa calibración; sin embargo, el detalle de mayor importancia es que la técnica no hace uso de la información propia que la imagen puede aportar de modo natural. Por ello, uno de los desafíos en los sistemas de visión artificial es realizar un seguimiento a partir de la información visual que la propia geometría del objeto brinda, lo cual representa generalmente un tiempo de procesamiento mayor (Santiago Pé, 2007).

2.3.2.1. *Métodos basados en un modelo previo.* Como su nombre menciona, parten de un modelo previo del objeto a detectar a partir del cual se buscan métodos que permitan encontrar una relación entre los puntos con las características primitivas que definen el modelo de la imagen capturada, tales como puntos, líneas y volúmenes. En base a esta relación se genera información que permite distinguir unos puntos de otros a partir de las características que estos presentan, haciendo posible calcular la correspondencia entre el modelo y la imagen a analizar, y dotando al sistema con la capacidad de obtener la posición y orientación del objeto.

Una posible división de estos métodos, en base al tipo de características que se busca en la imagen y que es utilizada para generar los modelos, se presenta a continuación:

- *Métodos basados en detección de bordes.* Como su nombre menciona, estos métodos están basados en la detección de bordes. Normalmente son usados en los sistemas de visión artificial debido a su eficiencia computacional y su fácil implementación, además de ser métodos que no presentan problemas frente a variaciones de iluminación.

- *Métodos basados en ajuste de plantilla.* Métodos basados en el ajuste de una plantilla bidimensional genérica cuando se ve sometida a un conjunto de deformaciones afines. El fin de estos algoritmos es encontrar, para una cierta imagen, los parámetros que definan la transformación que se debería aplicar a una plantilla para obtener el mismo resultado que en la imagen.

Una vez que se obtiene la plantilla es posible recuperar su pose a través de la homografía que define el plano que la forma y los parámetros intrínsecos de la cámara. Para la detección no son usadas las características locales del objeto, sino que es tomada toda la imagen que corresponde a la plantilla como fuente de información para la detección. Generalmente, este tipo de métodos presentan limitaciones, como el ser poco robusto ante variaciones de iluminación y posibles oclusiones en la plantilla.

- *Métodos basados en la extracción de puntos.* Métodos basados en el uso de marcas naturales localizadas, dejando de lado marcas globales como contornos o plantillas. Al estar basados en puntos individuales, estos algoritmos no presentan problemas frente a cambios de iluminación, oclusiones o errores de correspondencia. Para la obtención de estos puntos el enfoque se basa en la información que contiene el plano imagen, más que en la apariencia real del objeto. Fundamentalmente, estos métodos deben hacer frente a los siguientes problemas:

  - *Identificación de puntos de interés.* Para la identificación de los puntos de interés se utiliza la información que aporta la textura de una región de la imagen situada en torno a los puntos; sin embargo, no todos los puntos proyectados del objeto pueden ser detectados en otras imágenes, puesto que la textura que rodea a un punto puede sufrir deformaciones proyectivas, cambios de iluminación y ruido. Por ello, el proceso de identificación consiste en la detección de puntos proyectados del objeto que presenten ciertas características que los conviertan en candidatos ideales para su detección en otras imágenes.

o *Correspondencia entre imágenes.* Para la detección del objeto y obtención de la trasformación que sufre, es necesario un método de correspondencia entre los puntos de interés que son extraídos de una imagen y los puntos de interés del objeto. Generalmente, estos métodos tienen algoritmos de estimación robusta que les permite descartar falsas correspondencias.

- *Métodos basados en descriptores invariantes.* La base de estos métodos es la misma que de los basados en la extracción de puntos, la diferencia radica en que cada punto de interés tiene asociado un descriptor local, el cual busca extraer información de un área localizada de la imagen como la textura alrededor de un punto de interés, de tal modo que se define un método de codificación que no varíe frente a variaciones de iluminación y transformaciones afines. Los descriptores deben ser definidos de manera que a su vez permitan separar dos áreas que no pertenezcan a un mismo punto.

Habiendo optado ya por el tipo de sistema de posicionamiento a implementar en el espacio inteligente; ahora, de los dos métodos de detección detallados en el apartado **2.3** del presente capítulo, se define la utilización de la metodología basada en el uso de marcas artificiales. Más detalles acerca del desarrollo del sistema de detección y seguimiento bajo esta metodología de detección son descritos en el Capítulo 2.

3. Estado del Arte

La tesis doctoral con el título "Sistema de posición absoluto de un robot móvil utilizando cámaras externas", presentada por Ignacio Fernández Lorenzo, desarrolla un trabajo dentro del campo de las aplicaciones de los espacios inteligentes para el guiado de robots móviles. El método de detección está basado en el uso de marcas artificiales, proponiendo el autor la utilización de balizas circulares configuradas en forma de T, elaboradas a partir de diodos de infrarrojo, tal y como se muestra en la Figura 2. Las balizas son ubicadas a bordo de los robots, de tal modo que no solo facilitan su localización, incluso en posibles condiciones de extrema luminosidad ambiental, sino que simplifica los algoritmos de procesamiento de imágenes (Fernández Lorenzo, 2005).

Figura 2. Configuración de balizas en T.
Fuente: (Fernández Lorenzo, 2005).

El artículo con el título "Localización Simultánea a la Reconstrucción de Robot Móviles en Espacios Inteligentes mediante Múltiples Cámaras", presentada por Daniel Pizarro Perez, Enrique Santiso Gómez y Manuel Mazo Quintas, propone el uso de cámaras calibradas y fijas en el entorno para tareas de posicionamiento de robots móviles. Esta red de cámaras conforma la capa de percepción del sistema inteligente, capaz de reconocer robots presentes en su área de percepción visual, ofreciendo una solución al problema de obtención de la pose de robots móviles de estructura rígida. Así mismo, proponen una solución en ausencia de balizamiento activo a bordo del robot haciendo uso de un algoritmo definido como un proceso bayesiano de inferencia secuencial, el cual permite la obtención simultánea de la pose del objeto al mismo tiempo que es adquirida la información tridimensional del mismo. En la Figura 3 se muestra una simulación del sistema de posicionamiento propuesto (Pizarro, Santiso, & Mazo, 2006).

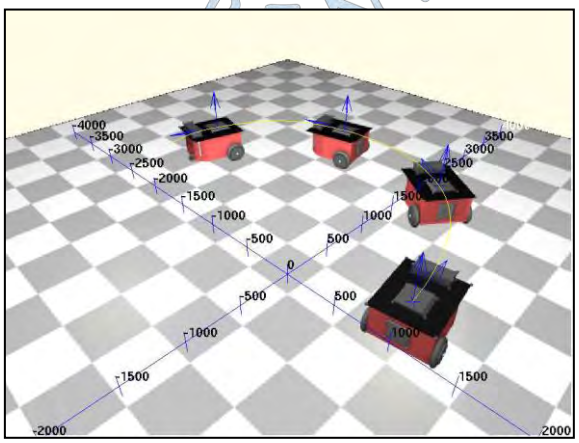

Figura 3. Simulación 3D de un espacio inteligente
Fuente: (Pizarro, Santiso, & Mazo, 2006).

La tesis de maestría con el título "Sistema de detección de objetos mediante cámaras. Aplicación en Espacios Inteligentes", presentada por Amaia Santiago Pé, aborda el problema de obtención de la posición y orientación de objetos tridimensionales en un entorno cerrado, proponiendo un sistema de cámaras calibradas y conectadas entre sí con el fin de formar un espacio inteligente. El objetivo de la autora es el desarrollo de un sistema capaz de realizar tareas de búsqueda y reconocimiento de objetos en el espacio, empleando un método de detección basado en marca naturales. El sistema detecta al objeto a partir de un modelo geométrico, que contiene puntos identificados en la imagen, a través de características pseudo-invariantes a la deformación proyectiva, de tal modo que el sistema permanece invariante frente a cambios de inmunización, oclusiones y cambios de perspectiva. Un resultado de las pruebas del sistema de detección se muestra en la Figura 4 (Santiago Pé, 2007).

Figura 4. Ejemplo de detección utilizando el modelo de cámara afín realizado en el Espacio Inteligente del Departamento de Electrónica de la Universidad Alcalá.
Fuente: (Santiago Pé, 2007).

El trabajo de curso de la asignatura de Sistemas Informáticos de la Facultad de Informática de la Universidad Complutense de Madrid, con el título "Identificación óptica de la posición y orientación de un vehículo aéreo no tripulado", presentada por Beatriz Martín Guadaño, Ana Melcón Sanjuán y Daniel Tapia Maganero, tiene como objetivo principal el desarrollo de un sistema de posicionamiento en espacios cerrados a través de visión estereoscópica. El sistema es capaz de hacer seguimiento a un objeto móvil en tiempo real empleando un sistema de detección basado en marcas artificiales; en este caso, dos marcas ópticas del mismo color a bordo del móvil, las cuales son detectadas haciendo uso de OpenCV[5]. En la Figura 5 se muestra la configuración del espacio de pruebas (Martín, Melcón, & Tapia, 2009).

[5] OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones.

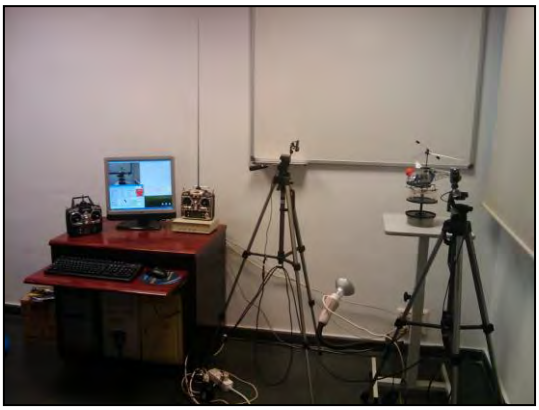

Figura 5. Entorno de pruebas.
Fuente: (Martín, Melcón, & Tapia, 2009).

La tesis doctoral con el título "Segmentación y Posicionamiento 3D de Robots Móviles en Espacios Inteligentes mediante redes de Cámaras Fijas", presentada por Cristina Losada Gutiérrez, surge con el objetivo de segmentar, identificar y posicionar en el espacio a robots móviles en un espacio inteligente a partir de información del entorno. La información es adquirida por medio de una o varias cámaras sincronizadas y calibradas, las cuales están ubicadas en posiciones fijas del entorno de actuación de los robots. Para cumplir con este objetivo, la autora propone una solución basada en la minimización de una función objetivo, la cual depende de tres grupos de variables: los contornos que definen la segmentación sobre el plano imagen, los parámetros de movimiento 3D y la profundidad de cada punto de la escena al plano imagen. Cabe mencionar que el sistema está basado en la detección de marcas naturales. En la Figura 6 se aprecia el entorno de pruebas del espacio inteligente (Losada Gutiérrez C. , 2010).

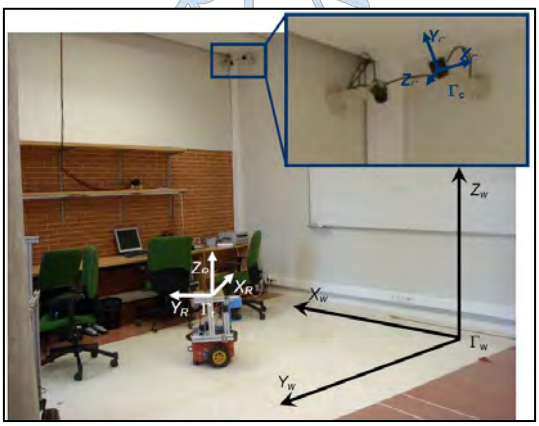

Figura 6. Espacio inteligente del Departamento de Electrónica de la Universidad de Alcalá.
Fuente: (Losada Gutiérrez C. , 2010).

El grupo de investigación GEINTRA, perteneciente al departamento de Electrónica de la Universidad de Alcalá (UAH), propone un sistema para el posicionamiento y guiado autónomo de robots móviles a partir de la información brindada por un sistema de cámaras estáticas cuya posición fija y estratégica le permite al sistema abarcar todo el campo de acción de los robots. El método de detección está basado en marcas naturales, las cámaras recogen información de todo el cuerpo del robot, obteniendo la posición en el espacio a partir de los contornos (que definen la segmentación sobre el plano imagen de cada cámara), las componentes de velocidad en el espacio (lineal y angular de cada robot móvil) y la profundidad (distancia de cada punto que pertenece al robot móvil a cada una de las cámaras). Así mismo, el sistema desarrollado tiene los algoritmos implementados en Matlab; por ello, necesita un tiempo de alrededor de siete segundos para la segmentación y estimación de la posición en el espacio; por lo tanto, no le permite al sistema obtener la posición en tiempo real. En la Figura 7 podemos observar la distribución de las cámaras en el entorno de trabajo de los robots (Losada Gutiérrez, Mazo, Palazuelos, Pizarro, & Marron, 2011).

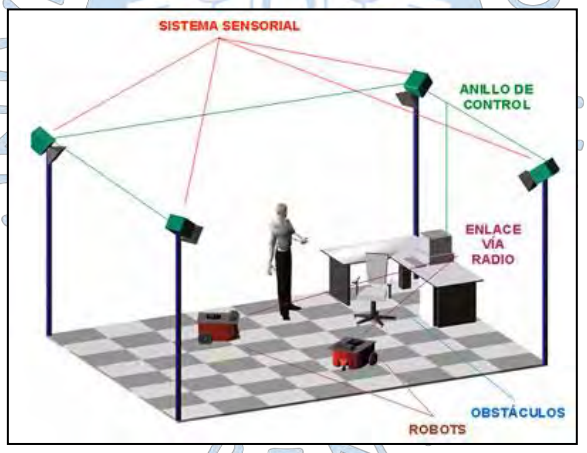

Figura 7. Configuración de espacio inteligente para el posicionamiento y guiado de robots móviles.
Fuente: (Losada Gutiérrez, Mazo, Palazuelos, Pizarro, & Marron, 2011).

La tesis de pregrado con el título “Sistema de posicionamiento para un drone”, presentada por Elías Rodriguez Martín, plantea el desarrollo de un sistema de posicionamiento basado en odometría visual para un drone cuyas tareas de navegación tienen lugar en espacios cerrados. El autor busca dar solución al problema de alcance de la señal GPS en interiores, centrándose

en el desarrollo de un módulo ROS[6] que se encarga de capturar una secuencia de imágenes del suelo, estimando la posición relativa respecto al mismo a partir de la detección del desplazamiento entre las imágenes. En la Figura 8 se pueden observar las diferentes componentes del sistema e interconexiones (Martínez Novo, 2015).

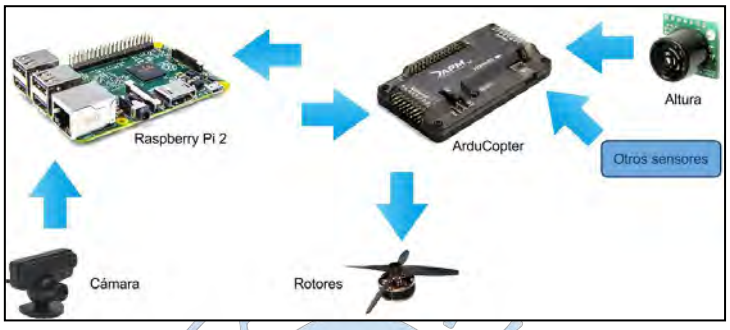

Figura 8. Interconexión de las componentes de un sistema de posicionamiento para dron.
Fuente: (Martínez Novo, 2015).

La tesis de maestría con el título "Posicionamiento de interiores de un dron por método multicámara", presentada por José Antonio Calvillo Ardilla, al igual que los otros mencionados y en relación a la presente tesis, ofrece una alternativa tecnológica para la obtención de la posición de un robot móvil en espacios cerrados; en este caso, el robot móvil es un dron cuyo entorno está implementado con dos cámaras y sin el uso de algún tipo de sensor adicional. Para su localización, el sistema de detección está basado en el uso de marcas artificiales puntuales, incorporando al dron un LED en la parte superior del mismo. El LED incorporado es detectado por medio de un método de reconocimiento basado en la identificación del punto de máxima luminosidad presente en el entorno; para ello, el autor hace uso de la biblioteca libre de visión artificial OpenCV. En la Figura 9 se puede observar al dron implementado con un LED en la parte superior, marca artificial puntual empleada para su localización (Calvillo Ardila, 2017).

En Suiza; *Research D'Andrea*, uno de los grupos de investigación de la Escuela Politécnica Federal de Zúrich dirigido por el Dr. D'Andrea Raffaelo, lleva realizando una serie de proyectos de investigación en cooperación con drones que cumplen con misiones de vuelo en espacios cerrados a partir de la posición de los mismos. La información se obtiene a partir de un sistema

[6] *Framework* o entorno de trabajo flexible orientado a la escritura de software para robots.

de posicionamiento en el espacio, proporcionado por *OptiTrack*[7], que está compuesto por una red de cámaras que emiten luz infrarroja y de una serie de marcadores reflectivos que reflejan la luz de vuelta hacia la fuente de emisión; permitiendo la detección, seguimiento y posicionamiento de todos los marcadores implementados a bordo de los vehículos aéreos que se encuentran en el campo visible de las cámaras. En la Figura 10 se observa a un dron que está por desplazarse del punto A al punto B a partir de la posición en el espacio de los marcadores (ETH Zurich, s.f.).

Figura 9. Implementación de un LED en la parte superior de un dron para la determinación de su posición.
Fuente: (Calvillo Ardila, 2017).

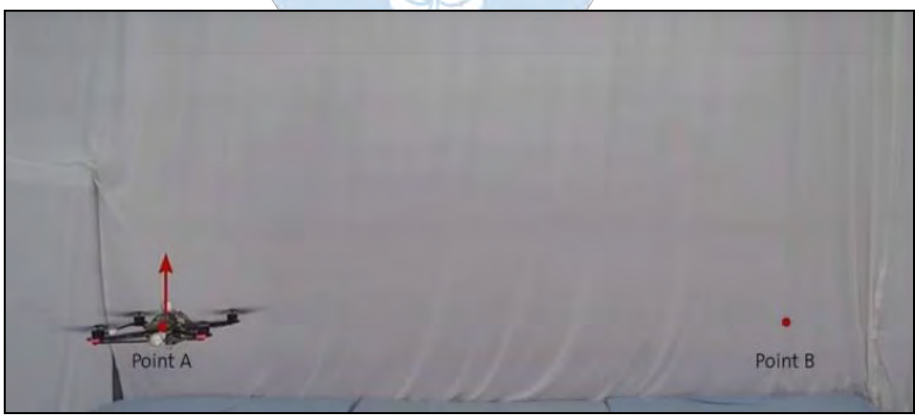

Figura 10. Trabajo de investigación - *Research D'Andrea*.
Fuente: (ETH Zurich, s.f.)

[7] *OptiTrack* es el proveedor de sistemas de captura de movimiento más grande del mundo; ofrece un seguimiento óptico de alto rendimiento a los precios más asequibles de la industria, y su línea de productos incluye software de captura de movimiento, cámaras de rastreo de alta velocidad, así como servicios de ingeniería por contrato.


UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 2
Sistema de detección y seguimiento

1. Introducción

Como se menciona en el apartado **2.3** del Capítulo 1, el primer paso para cumplir con el objetivo de la presente tesis es desarrollar un sistema que detecte la proyección de un objeto, sin perderlo de vista, en la imagen capturada por una cámara. En función a este propósito, el presente capítulo describe el hardware, software y algoritmos de visión artificial a emplear para el desarrollo de un sistema de detección y seguimiento que lleve a cabo esta operación.

2. Descripción del Hardware

2.1. Cámaras. Los espacios inteligentes, en base a la teoría descrita, se encuentran estructurados en entorno y mundo de percepción; a su vez, el entorno está dividido en capas funcionales de las cuales la capa de percepción es la comprendida por una serie de sensores cuya función es obtener información del entorno. A continuación, se detalla al sensor elegido para ser parte de esta capa.

Las cámaras son dispositivos que a través de un sistema óptico capturan las imágenes del entorno que son proyectadas en el sensor que los compone, transfiriéndolas a un sistema electrónico capaz de interpretarlas, almacenarlas y/o visualizarlas. Dependiendo de la configuración de los elementos fotosensibles en el sensor, pueden ser clasificados en dos tipos: cámaras matriciales y cámaras lineales (INFAIMON, s.f.).

2.1.1. ***Cámaras Lineales.*** Este tipo de cámaras construyen la imagen línea a línea a partir de un barrido lineal del objeto a través de sensores que poseen un rango de 512 a 12000 elementos (píxeles). Para ello, la cámara se desplaza manteniendo estático al objeto a capturar o viceversa. En la Figura 11 se muestra cómo se lleva a cabo este proceso.

Si bien este tipo de cámaras fue desarrollado para su uso en aplicaciones de inspección de materiales fabricados en continuo, tales como tela, planchas metálicas, etc., se están implementando actualmente en procesos productivos que requieren de una alta resolución y/o velocidad, tales como tareas de comprobación, medición y clasificación de objetos que se mueven rápidamente (INFAIMON, s.f.) (IDS Imaging Development Systems GmbH, s.f.).

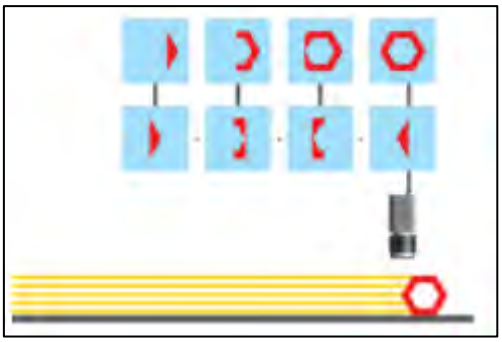

Figura 11. Barrido de una cámara lineal.
Fuente: (INFAIMON, s.f.).

2.1.2. ***Cámaras Matriciales.*** Este tipo de cámaras, conocidas como cámaras de área, poseen un sensor que cubre un área o está formado por una matriz de pixeles, el cual genera una imagen cuya área generalmente tiene una relación de aspecto 4:3. Esta relación, ya presentada por las cámaras vidicón[8], era usada en los formatos de cine y televisión; sin embargo, en la actualidad las cámaras y los formatos de televisión y cine están adaptados a los nuevos formatos de alta definición 16:9.

2.1.3. ***Sensor de imagen.*** Con respecto a los sensores de la cámara, pueden ser CCD (*charge coupled device*) o CMOS (*complementary metal oxide semiconductor*). A continuación, se detalla a los mismos (INFAIMON, s.f.):

2.1.3.1. CCD (*charge coupled device*). Estos sensores actualmente son los más usados, utilizan un material que posee sensibilidad a la luz con el fin de convertir los fotones en carga eléctrica. Una gran cantidad de diodos sensibles se posicionan formando una matriz cuyo registro de desplazamiento transfiere la carga de cada pixel, de tal modo que forman la señal de video. Su tamaño está definido en pulgadas, pero su tamaño real está basado en la relación de los primeros CCD con tubos vidicón; por ello, no tiene relación con el tamaño que viene especificado. Los formatos más comunes en la actualidad son de 1/3”, ½” y 2/3” (INFAIMON, s.f.).

[8] Cámaras cuyo sistema de captación de imágenes estaba compuesto por un tubo de material fotoconductor, el cual representa una resistencia variable según la luminosidad incidente.

2.1.3.2. CMOS (*complementary metal oxide semiconductor*). En los últimos años los sensores y cámaras de este tipo de tecnología se han popularizado debido a la notoria mejora en la calidad de imagen, además de ser indispensables en sistemas que demandan una alta velocidad. Los sensores de este tipo emplean un sustrato inmaterial sensible a luz, pero a diferencia de los sensores CDD, operan a través de un método de acceso aleatorio para transmitir la información del pixel, en lugar de utilizar registros de desplazamiento. Los sensores CMOS incluyen en su substrato al área activa del pixel y el espacio necesario para el chip que se aloja en el propio circuito. La principal ventaja de estos sensores es su velocidad, siendo capaces de funcionar a cientos o incluso miles de imágenes por segundo (INFAIMON, s.f.).

En relación a las cámaras, en la presente tesis se opta por el uso de la cámara matricial de sensor CMOS de la marca Microsoft, siendo el sensor que comprende el sistema sensorial de la capa de percepción del espacio inteligente a desarrollar. En la Figura 12 se muestra una imagen de la cámara y en la Tabla 2 se enlistan sus especificaciones técnicas.

Figura 12. Cámara Microsoft.

Fuente: (Microsoft, s.f.).

Tabla 2. Especificaciones técnicas de la cámara empleada.

GENERAL	
Marca	Microsoft
Modelo	Life Cam Studio
ESPECIFICACIONES	
Interfaz	USB
Resolución	1080x720 (HD), 30 fps
Sensor	CMOS
Profundidad de enfoque	10 cm - infinito

Fuente: (Microsoft, s.f.).

Elaboración: Propia.

2.2. Ordenador Portátil. La capa de inteligencia es la encargada de procesar toda la información recogida por el sistema sensorial de la capa de percepción, además de permitir que pueda ser intercambiada con los agentes del entorno; por ello, la importancia de su definición. Dicho esto, en la presente tesis se opta por el uso de un ordenador portátil como sistema encargado de procesar y gestionar la información recogida en la capa de percepción. Cabe mencionar que los algoritmos de visión artificial y cálculos matemáticos para la conversión de espacios 2D a 3D, tienen lugar en esta capa funcional. En la Figura 13 se muestra una imagen del ordenador portátil y en la Tabla 3 se enlistan sus especificaciones técnicas.

Figura 13. Ordenador portátil.
Fuente: (amazon, s.f.)

Tabla 3. Especificaciones técnicas del ordenador empleado.

GENERAL	
Marca	Lenovo
Modelo	Z50-70 i72G
ESPECIFICACIONES TÉCNICAS	
Procesador	Intel Core i5-4210U 1.70Ghz
Memoria RAM	8GB
Disco Duro	1TB 5400 RPM
Tarjeta de video	NVIDIA GeForce GT 840M con 4GB DDR3 dedicado

Fuente: (amazon, s.f.).

Elaboración: Propia.

2.3. Marcador Reflectivo. En el apartado **2.3** del Capítulo 1 se hace mención de dos metodologías para la detección de objetos mediante cámaras; así mismo, al término de sus descripciones se define la utilización de la metodología basada en el uso de marcas artificiales. A continuación, se describe la marca artificial a emplear en la presente tesis.

Los marcadores reflectivos son esferas cubiertas con un material reflectante que les brindan la capacidad de reflejar la luz de vuelta hacia la fuente que la emite, sin importar el ángulo de incidencia[9]. Estas son las macas artifíciales que se implementan a bordo de robots móviles y que el sistema de posicionamiento debe detectar, seguir y posicionar en el entorno para los fines que sean convenientes. En la Figura 14 se muestra una imagen del marcador reflectivo.

Figura 14. Marcador reflectivo *OptiTrack.*
Fuente: (OptiTrack, s.f.).

Para tener una idea más clara del efecto que tiene el material reflectivo que cubre los marcadores, en la Figura 15.a se muestra una imagen del marcador tomada con un dispositivo que no cuenta con una fuente de luz, mientras que en la Figura 15.b se muestra una imagen del mismo marcador, pero implementando al dispositivo de captura con una fuente de luz.

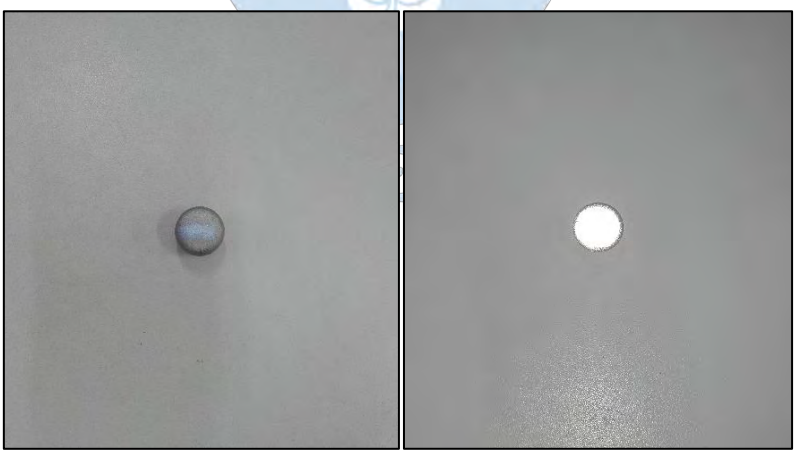

Figura 15. Imagen de un marcador reflectivo sin una fuente de luz (a). Imagen de un marcador reflectivo con una fuente de luz (b).
Fuente: Elaboración Propia.

[9] Punto de reflexión donde se ubica la normal de luz sobre algún objeto reflectivo cóncavo o convexo.

En función a la característica reflectiva de los marcadores, el sistema de detección y seguimiento es desarrollado, de tal modo que sea capaz de detectar el punto de máxima luminosidad en las imágenes recogidas por las cámaras implementadas en el entorno del espacio inteligente. Para entender esto, en la Figura 16 se muestra una imagen tomada con un dispositivo que cuenta con una fuente de luz LED; como se observa, existen tres objetos de los cuales uno es el marcador reflectivo. En esta situación, el sistema debe ser capaz de discriminar a los objetos adicionales y detectar solo al marcador reflectivo, el cual representa al objeto de máxima luminosidad en la imagen. Cabe mencionar que el uso de este marcador evita que el sistema presente problemas de detección debido al reflejo de la luz natural presente en el entorno de actuación, el cual es uno de los principales problemas que surgen en el desarrollo de aplicaciones con visión artificial.

Figura 16. Característica reflectante de un marcador *OptiTrack*.
Fuente: Elaboración Propia.

2.4. Sistema de iluminación. Uno de los puntos más críticos en toda aplicación con visión artificial es la iluminación, puesto que es la luz reflejada de los objetos lo que capturan las cámaras; en consecuencia, la finalidad de la iluminación es controlar la forma del objeto a ser visto. Cabe mencionar que la luz es reflejada de manera distinta dependiendo del objeto; es decir, no es lo mismo iluminar una bola de acero que una hoja de papel blanco; por lo tanto, el sistema de detección y seguimiento debe adaptarse al objeto a iluminar. Se puede concluir que una decisión de vital importancia es la elección del sistema de iluminación, el cual depende de la aplicación a desarrollar.

En relación a la iluminación se pueden diferenciar cuatro tipos de sistemas: Iluminación mediante fibra óptica, mediante fluorescentes, mediante diodos LED y mediante láser. Se debe tener en cuenta que cada una presenta sus ventajas y desventajas; por ello, se debe definir cuál usar dependiendo de la aplicación. Los mismos se detallan a continuación (Aplicación práctica de la visión artificial, 2012):

2.4.1. Iluminación mediante fibra óptica. Este tipo de iluminación es hasta ahora la que más intensidad de luz proporciona de entre todos los usados en aplicaciones con visión artificial. Básicamente, su principio de funcionamiento está basado en la conducción de luz proveniente de una lámpara halógena, de xenón, de haluro metálico o LED, que se encuentra en una fuente de iluminación, a través de un haz de fibras ópticas que termina en un adaptador específico dependiendo del tipo de aplicación. Estos adaptadores pueden tener forma circular, lineal, puntual o de panel; además de ser de distintas dimensiones. En la Figura 17 se muestra la imagen de un haz de fibra óptica (INFAIMON, s.f.).

Figura 17. Haz de fibra óptica.
Fuente: (INFAIMON, s.f.).

2.4.2. *Iluminación por fluorescente*. El tubo fluorescente es una lámpara de vapor de mercurio, que se encuentra a baja presión, utilizada para iluminación industrial. La ventaja que tiene con respecto a las otras, como las incandescentes, es su eficiencia energética. Son diseñadas para proporcionar el máximo de intensidad al menos por 7000 horas, representado una productividad mucho mayor.

Este tipo de iluminación es empleada en aplicaciones cuyo entorno requiere de mucha luz y ninguna presencia de sombras; por ejemplo: Análisis bilógicos, inspección y microscopía,

ensamblaje, inspección de circuitos, industria, laboratorios, visión industrial, fotografía, control de calidad, robótica, etc. En la Figura 18 se muestra la imagen de un fluorescente circular (INFAIMON, s.f.) (Aplicación práctica de la visión artificial, 2012).

Figura 18. Fluorescente circular.
Fuente: (INFAIMON, s.f.).

***2.4.3. Iluminación mediante diodos* LED.** Los diodos LED (*Light-Emitting Diode*) son dispositivos semiconductores capaces de emitir luz de espectro reducido cuando son polarizados de forma directa (unión p-n) y circula por los mismos una corriente eléctrica. Actualmente, son muchas las aplicaciones que hacen uso de este tipo de iluminación debido a su reducido tamaño, eficiencia energética y bajo precio. Así mismo, proporcionan una intensidad de iluminación relativamente alta a un costo accesible, además de tener un periodo de vida de aproximadamente 100 mil horas. En la Figura 19 se muestra la imagen de algunos ejemplares de sistemas de iluminación LED (INFAIMON, s.f.).

Figura 19. Sistemas de iluminación LED.
Fuente: (INFAIMON, s.f.).

2.4.4.*Iluminación mediante láser*. También conocida como iluminación mediante luz estructurada, es un tipo de iluminación que cuenta con una fuente monocromática que presenta distintos patrones de luz, tales como puntos, líneas, rejillas, etc. Para ser utilizada es necesario contar con un sistema de seguridad que evite daños a los operarios que están trabajando cerca a esta iluminación. Las líneas láser son también utilizadas para indicar el trazado por dónde se debe ajustar un proceso, por ejemplo, en aplicaciones de corte de madera o roca. En la Figura 20 se muestra la imagen de un dispositivo láser (INFAIMON, s.f.).

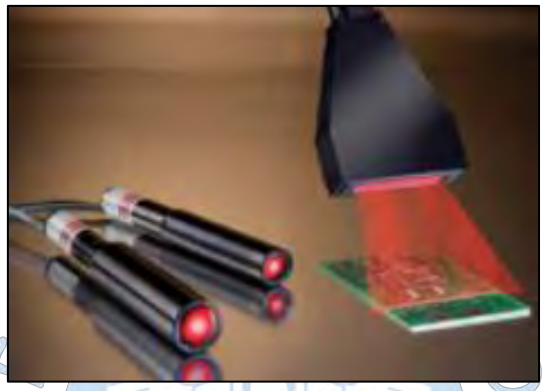

Figura 20. Iluminación mediante láser.
Fuente: (INFAIMON, s.f.).

Descritos los distintos sistemas de iluminación, en la presente tesis se opta por la utilización de un sistema de iluminación LED, puesto que en la Figura 16 se observa el comportamiento deseado del marcador bajo este tipo de iluminación. Dicho esto, en la Figura 21 se muestra el sistema de iluminación conformado por tiras de diodos LED y un marco de acrílico, al igual que una imagen del sistema implementando en una de las cámaras a emplear.

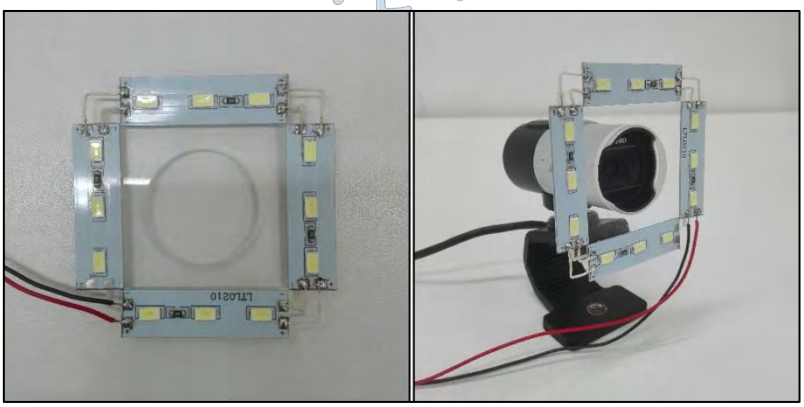

Figura 21. Sistema de iluminación a implementar en las cámaras.
Fuente: Elaboración propia.

3. Descripción del Software

3.1. Ubuntu. Sistema operativo de código abierto para computadoras basado en la arquitectura de Debian. La razón de su elección se debe al hecho de que el *framework* a usar en la presente tesis (ROS), cuenta con librerías que hasta la fecha están orientadas solo para el sistema operativo Ubuntu GNU/Linux. Si bien ya se están adaptando a otros sistemas operativos, aún son considerados como experimentales (Open Source Robotics Foundation, s.f.).

3.2. OpenCV. Biblioteca libre de visión artificial desarrollada originalmente por Intel. La primera versión alfa aparece en el mes de enero de 1999, siendo utilizada desde un inicio en infinidad de aplicaciones, tales como como sistemas de seguridad con detección de movimiento o aplicaciones que implementan el reconocimiento de objetos para el control de procesos. Esto se debe a que OpenCV fue publicada bajo la licencia BSD, permitiendo que pueda ser usada de manera libre para propósitos comerciales y de investigación bajo las condiciones que se expresan en la licencia (OpenCV team, s.f.). Al ser multiplataforma se extiende a versiones como GNU/Linux, Mac OS X, Windows y Android; contando con más de 500 funciones que abarcan un gran número de áreas en el campo de visión artificial, por ejemplo, el reconocimiento de objetos y facial, inspección de productos de fábrica, escaneo médico, seguridad, interfaces de usuario, calibración de cámaras, visión robótica, etc.; además de contar con una librería completa de uso general para aprendizaje automático (*Machine Learning Library*). Con respecto a su estructura; OpenCV es modular, teniendo como principales módulos a los siguientes (Rodríguez Bazaga, s.f.):

- ***Core*:** Módulo que contiene las estructuras de datos básicas a utilizar por el resto de módulos. Así mismo, cuenta con funciones básicas para procesamiento de imágenes (Rodríguez Bazaga, s.f.) (Geeky Theory, s.f.).

- ***Highhui*:** Módulo que proporciona una interfaz de usuario, códec[10] de imagen y video, y capacidad para captura de los mismos. Si se necesita capacidades de UI (*User Interface*) más complejas, se debe hacer uso de *frameworks* como: Qt, *WinForms*, etc. (Rodríguez Bazaga, s.f.).

[10] Dispositivo o programa hardware que permite codificar o decodificar una señal o flujo de datos digitales.

- ***Imgproc*:** Módulo que contiene algoritmos básicos para procesamiento de imágenes, tales como funciones para el filtrado lineal y no lineal, transformaciones afines, conversión del espacio de color, histogramas, etc. (Geeky Theory, s.f.).

- ***Video*:** Módulo para el análisis de video que incluye algoritmos para la estimación de movimientos, extracción de fondos y seguimiento de objetos (Geeky Theory, s.f.).

- ***Objdetect*:** Módulo que contiene algoritmos para detección y reconocimiento de objetos e instancias de clases predefinidas, tales como caras, ojos, personas, coches, etc. (Geeky Theory, s.f.).

4. Desarrollo del sistema de detección

En esta parte del capítulo se describe cómo llegar a desarrollar el sistema de detección. Se detallan algunas de las funciones que ofrece la biblioteca Open CV que son empleadas en el procesamiento de imagen para cumplir con el objetivo del sistema. Cabe mencionar que la biblioteca a emplear es de código abierto.

4.1. Procesamiento de imagen para la detección del marcador. Para empezar con el desarrollo del sistema de detección, es necesario realizar un estudio previo sobre reconocimiento e identificación de objetos, uno de los temas con mayor tiempo invertido en el estudio de técnicas y algoritmos. El fin de los algoritmos estudiados es que simplifiquen cada vez más el trabajo de detección; por ello, en lugar de intentar reconocer cada mínima zona que compone una imagen, se realiza el reconocimiento según un fin específico, por ejemplo, la detección de un objeto con un determinado color, detección de caras, detección de personas, etc. Generalmente, las técnicas que existen para la detección de objetos se pueden clasificar en 4 metodologías, las mismas son descritas a continuación (Pérez González, 2016):

4.1.1. *Detectores de regiones*. Metodología basada en la generación de regiones con rasgos similares que sean de interés. Para la identificación de estas regiones, se requiere que los algoritmos analicen cada punto de la imagen, llegando a ser algoritmos de alto coste computacional por el considerable tiempo que les toma recorrer cada pixel; sin embargo, los resultados que ofrece son excelentes.

4.1.2. ***Substracción de fondo.*** Metodología basada en la eliminación del fondo de una imagen, destacando los objetos que no permanecen fijos de un *frame* a otro. Esta metodología resulta muy útil para sistemas de visión que permanecen fijos.

4.1.3. ***Machine Learning.*** Metodología también denominada como aprendizaje automático, campo de estudio que brinda al computador la capacidad de aprender sin algún tipo de programación previa. Para ello, el algoritmo aprende en base a una serie de ejemplos en donde se detecta a un mismo objeto en una serie de imágenes. La idea es que el algoritmo pueda ser capaz de seguir identificado al objeto en imágenes que no forman parte de los ejemplos empleados en el aprendizaje. Sin embargo, el problema de esta metodología es la necesidad de un nuevo proceso de aprendizaje para un nuevo objeto a identificar, además de la alta potencia de procesamiento requerida en el análisis de los ejemplos.

4.1.4. ***Segmentación.*** Metodología que consiste en la partición de una imagen digital en múltiples regiones formadas por un conjunto de pixeles, denominadas también como *superpíxeles*. El fin de la segmentación es poder simplificar la imagen, facilitando el análisis de la misma al tener como resultado una imagen con regiones de características similares.

Las regiones adyacentes son significativamente distintas en relación a características, tales como el color, intensidad, textura, etc. Al no existir una solución general para la segmentación de una imagen, se cuenta con distintos algoritmos de segmentación que proporcionan cada uno un resultado distinto; por ello, en función del resultado que se desea obtener se hace la elección del algoritmo a usar. Algunos de estos son: Segmentación basada en el agrupamiento (*Clustering)*, segmentación basada en el histograma, segmentación basada en la detección de bordes, segmentación basada en un valor umbral (*Threshold*), segmentación basada en modelos, etc.

De las metodologías descritas para la detección de objetos, se opta por la metodología de segmentación, en específico la basada en un valor umbral (*Threshold*), ya que la misma es de bajo coste computacional en comparación con las otras y brinda las prestaciones suficientes para cumplir con el objetivo del sistema de detección.

4.2. Segmentación bajo el método del valor umbral *(Threshold).* Con respecto a la segmentación basada en un valor umbral, se cuenta con 5 modos de operar. Los mismos son mencionados a continuación, siendo src(x,y) la intensidad de los píxeles de la imagen original, dst(x,y) la intensidad de los píxeles de la imagen de salida, y *thresh* el valor umbral (Threshold (Umbralización), s.f.):

- ***Umbral binario.*** Este modo de operación de umbral se expresa como:

$$\text{dst}(x,y) = \begin{cases} \text{Valor máximo} \rightarrow \text{si } \text{src}(x,y) > \textit{thresh} \\ 0 \qquad\qquad \rightarrow \text{el resto de píxeles} \end{cases}$$

Según la expresión, si el valor de intensidad de los píxeles (src(x,y)) se encuentra por encima del valor umbral (*thresh*), pasan a tener un valor igual al máximo valor de intensidad dependiendo de la escala de color de trabajo; de lo contrario, los píxeles pasan a tener un valor de intensidad igual a 0. En la Figura 22 se representa lo mencionado, en donde la línea azul representa a un valor fijo de umbral.

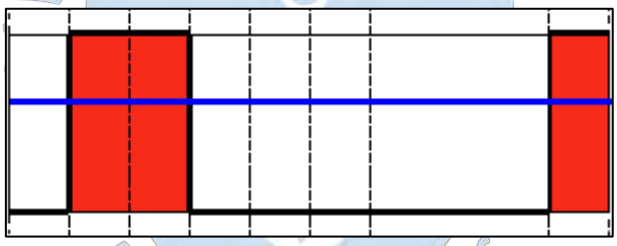

Figura 22. Representación gráfica del umbral binario.
Fuente: (Threshold (Umbralización), s.f.).

- ***Umbral binario invertido.*** Este modo de operación de umbral se expresa como:

$$\text{dst}(x,y) = \begin{cases} 0 \qquad\qquad \rightarrow \text{si } \text{src}(x,y) > \textit{thresh} \\ \text{Valor máximo} \rightarrow \text{el resto de píxeles} \end{cases}$$

Según la expresión, si el valor de intensidad de los pixeles (src(x,y)) se encuentra por encima del valor umbral (*thresh*), pasan a tener un valor de intensidad igual a 0; de lo contrario, los píxeles pasan a tener un valor igual al máximo valor de intensidad dependiendo de la escala de color de trabajo. En la Figura 23 se representa lo mencionado, en donde la línea azul representa a un valor fijo de umbral.

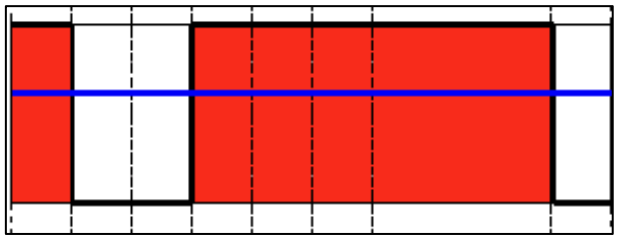

Figura 23. Representación gráfica del umbral binario invertido.

Fuente: (Threshold (Umbralización), s.f.).

- ***Truncar.*** Este modo de operación de umbral se expresa como:

$$\text{dst}(x,y)=\begin{cases} thresh & \rightarrow \text{si } \text{src}(x,y) > thresh \\ \text{src}(x,y) & \rightarrow \text{el resto de píxeles} \end{cases}$$

Según la expresión, si el valor de intensidad de los pixeles (src(x,y)) se encuentra por encima del valor umbral (*thresh*), pasan a tener un valor de intensidad igual al valor umbral designado; de lo contrario, los píxeles permanecen con el mismo valor de intensidad. En la Figura 24 se representa lo mencionado, en donde la línea azul representa a un valor fijo de umbral.

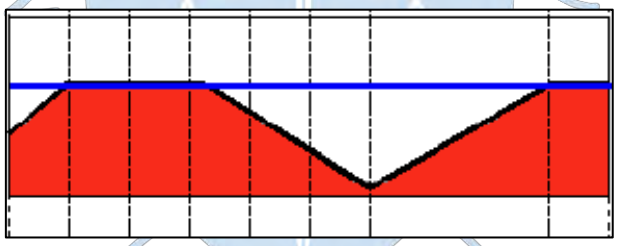

Figura 24. Representación gráfica del umbral trunco.

Fuente: (Threshold (Umbralización), s.f.).

- ***Umbral a cero.*** Este modo de operación de umbral se expresa como:

$$\text{dst}(x,y)=\begin{cases} \text{src}(x,y) & \rightarrow \text{si } \text{src}(x,y) > thresh \\ 0 & \rightarrow \text{el resto de píxeles} \end{cases}$$

Según la expresión, si el valor de intensidad de los pixeles (src(x,y)) se encuentra por encima del valor umbral (*thresh*), permanecen con el mismo valor de intensidad; de lo contrario, los píxeles pasan a tener un valor de intensidad igual a 0. En la Figura 25 se representa lo mencionado, en donde la línea azul representa a un valor fijo de umbral.

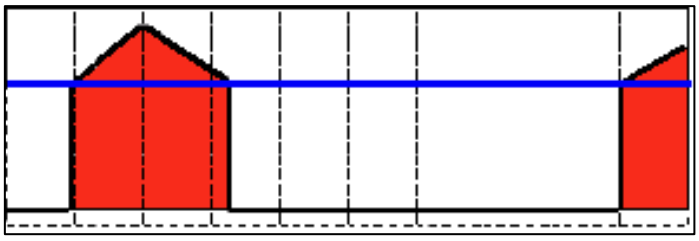

Figura 25. Representación gráfica del umbral a cero.
Fuente: (Threshold (Umbralización), s.f.).

- ***Umbral a cero invertido.*** Este modo de operación de umbral se expresa como:

$$\text{dst}(x,y) = \begin{cases} 0 & \rightarrow \text{si } \text{src}(x,y) > \textit{thresh} \\ \text{src}(x,y) & \rightarrow \text{el resto de píxeles} \end{cases}$$

Según la expresión, si el valor de intensidad de los píxeles (src(x,y)) se encuentra por encima del valor umbral (*thresh*), pasan a tener un valor de intensidad igual a 0; de lo contrario, los píxeles permanecen con el mismo valor de intensidad. En la Figura 26 se representa lo mencionado, en donde la línea azul representa a un valor fijo de umbral.

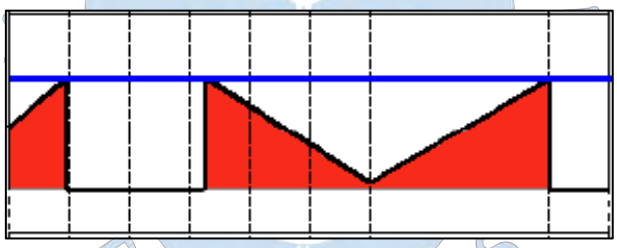

Figura 26. Representación gráfica del umbral a cero invertido.
Fuente: (Threshold (Umbralización), s.f.).

En relación a los modos de operación descritos, se opta por el modo umbral binario debido a su amplio uso en distintas aplicaciones en detección de objetos con resultados aceptables. En cuanto a la característica en común de los superpíxeles, se opta por el color; es decir, el marcador debe ser tal que su color lo diferencie notablemente del fondo, separándolo de éste y siendo el único visible, característica que presenta el marcador y que se corrobora en el apartado **2.3** del presente capítulo.

Se concluye que la metodología a emplear para el sistema de detección es: "Metodología de segmentación, basada en el color, empleando el método del valor umbral bajo el modo umbral

binario". A continuación, se detallan las funciones a emplear para llevar a cabo la metodología de segmentación definida cuya aplicación está orientada al sistema de detección.

4.2.1. ***Conversión del espacio de color (BGR → HSV).*** Las imágenes capturadas por una cámara haciendo uso de la biblioteca de OpenCV generalmente están almacenadas en el espacio de color BGR; es decir, pueden ser consideradas como la suma de tres matrices: *Blue* (Azul), *Green* (Verde) y *Red* (Rojo), con valores que van del 0 al 255. En la Figura 27 se muestra como una imagen es representada como la suma de estas tres matrices (Pérez González, 2016), en donde cada pequeña caja que se observa representa a un pi xel de la imagen. En una imagen real, estos píxeles son tan pequeños que el ojo humano no es capaz de poder diferenciarlos.

Como se menciona, las imágenes capturadas son almacenadas en el espacio de color BGR; sin embargo, no es el espacio de color adecuado para la metodología de segmentación definida. En su lugar, el espacio HSV es el más adecuado para la segmentación de imágenes basadas en el color; por ello, cambiar el espacio de color de BGR a HSV es el primer paso para aplicar la metodología (Shermal, s.f.).

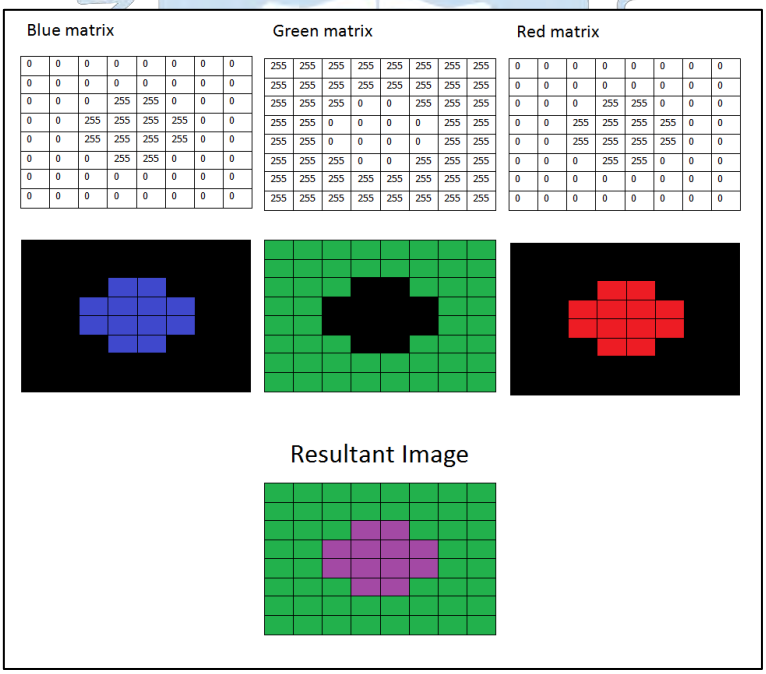

Figura 27. Imagen representada como la suma de tres matrices.
Fuente: (Shermal, s.f.).

El espacio de color HSV, al igual que el BGR, consiste de tres matrices: *Hue* (Matiz), *Saturation* (Saturación) y *Value* (Valor de intensidad), con valores en OpenCV que van de 0 a

180 para *Hue*, y de 0 a 255 para *Saturation* y *Value*. Este espacio de color permite una segmentación mucho más eficiente, ya que tiene en consideración las condiciones de luminosidad e intensidad de color de la imagen, aspectos que el espacio BGR no tiene en cuenta. Esto es posible separando la componente "chroma", la cual aporta información del color y es la única también empleada en el espacio BGR, de la componente "luma", la cual aporta la intensidad del color. La aplicación de la metodología de segmentación bajo el espacio de color HSV llega a ser de mucha utilidad en los sistemas de visión artificial, puesto que se consiguen sistemas robustos frente a cambios de iluminación. Cabe mencionar que existen muchos más espacios de color, tales como YcbCr, Lab, etc.; sin embargo, en la presente tesis se hace uso del espacio HSV por ser la conversión de BGR a HSV, una de las más extendidas y de fácil implementación al encontrarse disponible en la biblioteca de OpenCV. En la Figura 28 se muestra al espacio HSV, la cual generalmente es representada como un cono invertido haciendo uso de coordenadas cilíndricas. A continuación se detallan sus componentes (Pérez González, 2016) (Wan Mahani & Shahrul Nizam, 2017):

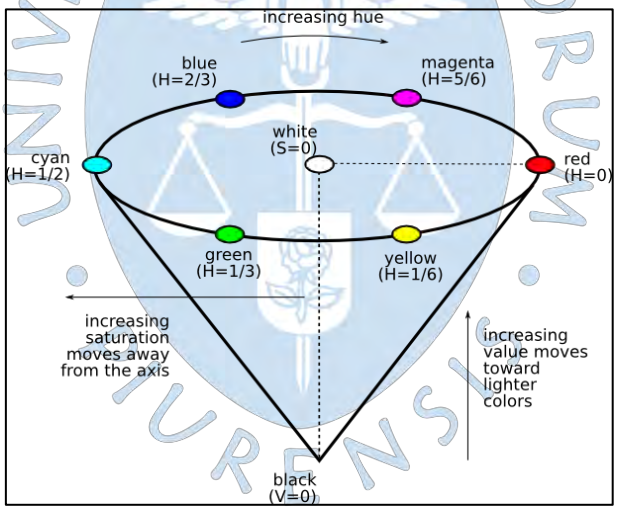

Figura 28. Representación gráfica de la escala de color HSV.

Fuente: (Wan Mahani & Shahrul Nizam, 2017).

- ***Hue* (Matiz):** Es representado como un ángulo entre 0° y 360°, de tal modo que cada ángulo barrido corresponde a un color. Al tratarse de una imagen de 8 bits, OpenCV realiza un escalamiento a valores enteros que van de 0 a 180; para ello, se aplica la siguiente conversión: H→H/2 (360→180).

- ***Saturation* (Saturación):** Se define como la distancia al eje blanco-negro, con valores que van del 0% al 100%. En este caso, OpenCV realiza un escalamiento a valores enteros que van de 0 a 255.

- ***Value* (Valor):** Se define como la altura en el eje blanco-negro, con valores que van del 0% al 100%, en donde 0% representa al color negro (vértice del cono). Una vez más, OpenCV realiza un escalamiento a valores enteros que van de 0 a 255.

Conocida la necesidad de cambiar el espacio de color de BGR a HSV, se busca una función que permita cambiar el espacio de color de una imagen. De las funciones disponibles en la biblioteca de OpenCV, *cvtColor* es la que permite llevar a cabo esta operación.

```
C++: void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0
```

Parámetros:

- ***InputArray src*:** Imagen de entrada.

- ***OutputArray dst*:** Imagen de salida que debe tener el mismo tamaño y profundidad de la imagen de entrada.

- ***int code*:** Código de conversión del espacio de color, tales como *COLOR_BGR2BGR2BGRA*, *COLOR_BGR2HSV*, *COLOR_RGB2HSV*, *COLOR_BGR2GRAY*, *COLOR_BGR2YCrCb*, etc.

- ***Int dstCn*:** Número de canales de la imagen de salida. Si es 0, el número de canales se deriva automáticamente de la imagen de entrada y el código de conversión.

4.2.2. *Umbralización.* Con las imágenes capturadas en el espacio de color adecuado, el siguiente paso a realizar es el método de umbralización en sí. El fin de este método es llevar una imagen que se encuentra en un determinado espacio de color, a una imagen binaria de acuerdo a los parámetros que sean asignados.

Como se menciona en el apartado **4.2** del presente capítulo, el modo de umbralización a emplear es el binario, de tal modo que la lógica de operación es la siguiente: Si el valor de intensidad del pixel que se analiza se encuentra por encima de un valor umbral (*threshold)* asignado, toma un valor de 255; es decir, se torna de color blanco; de lo contrario, toma un valor de 0, tornándose de color negro. Este proceso separa las regiones correspondientes al objeto a detectar; en este caso, del marcador reflectivo (Pérez González, 2016).

Mencionado el proceso de umbralización a llevar a cabo, se busca una función que permita comprobar si un valor de intensidad de pixel se encuentra en un rango determinado por el color a detectar. De las funciones disponibles en OpenCV, *inRange* es la que permite llevar a cabo esta operación, de tal modo que si la intensidad de un pixel se encuentra dentro del rango asignado, toma un valor de 255; de lo contrario, toma un valor mínimo de 0. Como resultado, el objeto detectado se muestra de color blanco en la imagen de salida, mostrando de color negro las regiones que no interesan. A continuación, la función es detallada para su uso en el lenguaje de programación de C++ (Opencv dev team, s.f.):

```
C++: void inRange(InputArray src, InputArray lowerb, InputArray upperb,
OutputArray dst)
```

Parámetros:

- ***InputArray src*:** Imagen de entrada.

- ***InputArray lowerb*:** Límite inferior. Si lowerb = Scalar(X, Y, Z), los píxeles que llegan a tener valores menores a (X, Y, Z) para HUE, SATURATION y VALUE respectivamente, se tornan de color negro en la imagen de salida.

- ***InputArray upperb*:** Límite superior. Si upperb = Scalar(X, Y, Z), los píxeles que llegan a tener valores mayores o iguales a (X, Y, Z) para HUE, SATURATION y VALUE respectivamente, se tornan de color negro en la imagen de salida.

- ***OutputArray dst*:** Imagen de salida que debe tener el mismo tamaño y profundidad de la imagen de entrada.

Para entender esto, en la Figura 29 se muestra la imagen resultante de un cubo de rubik tras llevar a cabo un proceso de segmentación similar al definido para la presente tesis. Como se puede deducir de la imagen, el valor umbral escogido es el valor de intensidad correspondiente al color azul, puesto que se observa que en la imagen de salida son las piezas azules las que se tornan de color blanco, permaneciendo en la imagen binaria resultante. Sin embargo, se puede observar que existe presencia de ruido[11], punto a tratar en la siguiente operación a realizar.

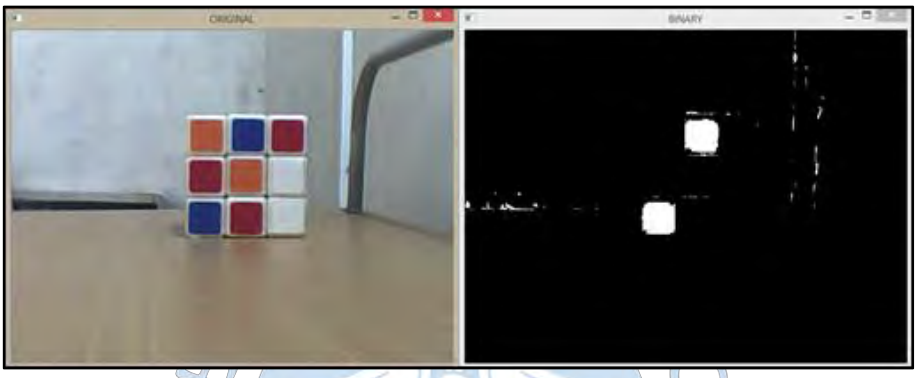

Figura 29. Imagen binaria resultante después de llevar a cabo un proceso de umbralización.
Fuente: (Marín Abrego, s.f.).

4.2.3. ***Operaciones Morfológicas.*** Se pueden definir como procesos no lineales ejecutados sobre un número de píxeles, tomando en cuenta los adyacentes con el objetivo de modificar la estructura o forma de los mismos en la imagen. Las aplicaciones más frecuentes son las siguientes (Marenco Jiménez, 2009). (González Díaz & Real Jurado, 2003):

- **Pre-procesamiento de imágenes:** Supresión de ruido, simplificación de formas, etc.

- **Destacar la estructura de los objetos:** Extracción del esqueleto de una imagen, detección de objetos, ampliación, reducción, etc.

- **Descripción cualitativa de objetos:** Área, perímetro, diámetro, etc.

Tomando en cuenta las aplicaciones mencionadas y la necesidad de suprimir el ruido presente en una imagen, tal como se observa en la Figura 20, se buscan funciones que permitan

[11] El ruido se define como una variación aleatoria de la información del brillo o color en las imágenes de naturaleza digital, causada por el dispositivo que captura la imagen y que no corresponde con la realidad.

realizar esta operación. Dicho esto, de las distintas funciones que existen, se opta por el uso de las funciones *erode* (erosión) y *dilale* (dilatación), las cuales son muy usadas en los sistemas de visión artificial porque permiten reducir en gran medida el ruido presente en las imágenes capturadas.

Cabe mencionar que la manera más sencilla de ser empleadas es sobre imágenes binarias (imágenes en blanco y negro), detalle a favor, ya que se cuenta con imágenes de este tipo después de la umbralización. A continuación son detalladas las funciones mencionadas:

4.2.3.1. *Erosión binaria.* Esta transformación morfológica resulta de comprobar si un elemento estructurante B está completamente incluido dentro de un conjunto X; de no ser así, el resultado de la transformación es el conjunto vacío, desapareciendo de la imagen los objetos que sean menores al elemento estructurante; esto es (Platero, 2008):

$$\varepsilon_B(X) = X \ominus B = \{x \mid B_x \subseteq X\}$$

Como se menciona, con el paso de un elemento estructurante sobre la imagen, son borrados aquellos objetos menores a éste; sin embargo, también causa la degradación de los objetos que aún permanecen en la imagen de salida. Por ello, se define que la transformación morfológica de erosión binaria supone una continua degradación de la imagen, corriendo con el riesgo de eliminar todos los objetos si es aplicada de manera consecutiva. La erosión binaria llega a ser una transformación antiextensiva[12], esto es (Platero, 2008):

$$\varepsilon_B(X) \subseteq X$$

Una segunda interpretación de la erosión supone tomar el valor mínimo de la imagen en el entorno de vecindad, el cual es definido por el elemento estructurante. En la Figura 30 se puede observar el efecto de degradación al aplicar la transformación morfológica de erosión binaria a una determinada imagen.

[12] Una transformación es antiextensiva cuando el resultado de la transformación está incluido en el conjunto de la entrada.

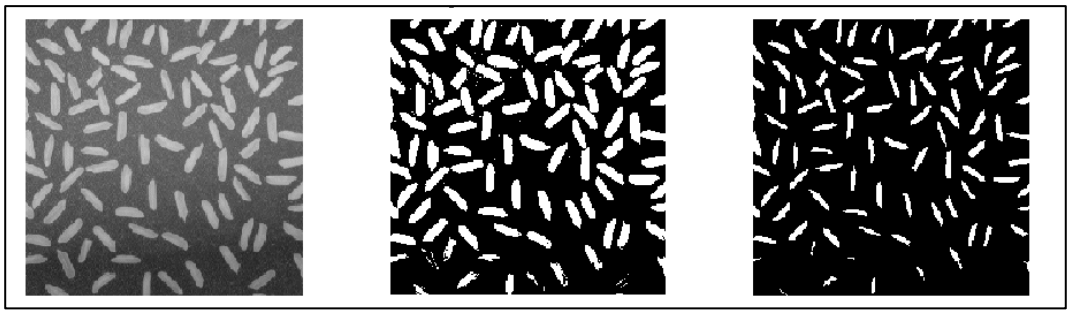

Figura 30. Efecto de degradación en una imagen al aplicar una transformación morfológica de erosión binaria con disco de radio 2 como elemento estructurante.
Fuente: (Platero, 2008).

Descrito lo que implica la aplicación de esta transformación morfológica, se hace mención de *erode*, función disponible en OpenCV que permite llevar a cabo esta operación. A continuación, es detallada para su uso en el lenguaje de programación de C++ (Opencv dev team, 2019):

```
C++: void erode(InputArray src, OutputArray dst, InputArray Kernel, Point
anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT,
const Scalar& borderValue=morphologyDefaultBorderValue())
```

Parámetros:

- ***InputArray src*:** Imagen de entrada.

- ***OutputArray dst*:** Imagen de salida que debe tener el mismo tamaño y profundidad de la imagen de entrada.

- ***InputArray Kernel*:** Elemento estructurante usado para la erosión de la imagen de entrada.

- ***Point anchor*:** Posición del anclaje dentro del núcleo. Si Point anchor = Point (-1.-1), el centro del núcleo es tomado como la posición del anclaje.

- ***int iterations*:** Número de veces que es aplicada la transformación.

- ***int borderType*:** Método de extrapolación de píxeles en una condición de contorno.

- ***const Scalar& borderValue*:** Valor de los píxeles en el contorno si borderType = BORDER_CONSTANT.

4.2.3.2. *Dilatación binaria.* Esta transformación morfológica, dual a la erosión, resulta de comprobar si al menos algún elemento del conjunto estructurante B está contenido en el conjunto X, cuando B se desplaza sobre el conjunto X:

$$\delta_B(X) = X \oplus B = \{x \mid X \cap B_x \neq \emptyset\}$$

Como se menciona, el conjunto X no se ve modificado con el paso de un elemento estructurante dentro del mismo; sin embargo, en la frontera del conjunto, al desplazarse B, el conjunto resultado se ve expandido. Por ello, se define que la transformación morfológica de dilatación binaria supone una continua degradación de la imagen, corriendo con el riesgo de hacer coincidir el conjunto dilatado con la imagen si es aplicada de manera consecutiva. La dilatación binaria llega a ser una transformación extensiva[13]:

$$\delta_B(X) \supseteq X$$

Una segunda interpretación de la dilatación supone tomar el valor máximo de la imagen en el entorno de vecindad, el cual es definido por el elemento estructurante (Platero, 2008). En la Figura 31 se puede observar el efecto de degradación al aplicar la transformación morfológica de dilatación binaria a una determinada imagen.

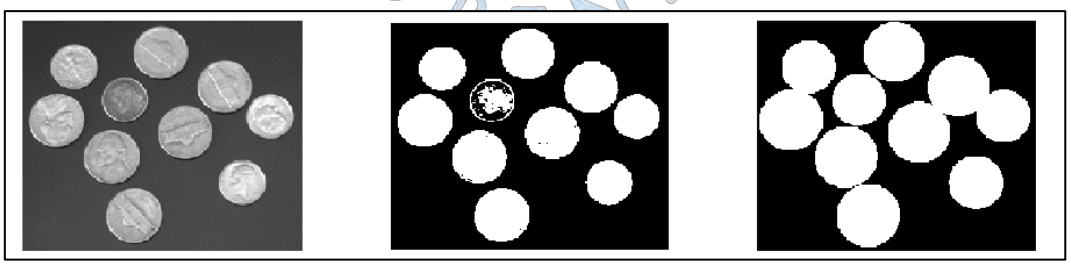

Figura 31. Efecto de degradación en una imagen al aplicar una transformación morfológica de dilatación binaria con disco de radio 5 como elemento estructurante.
Fuente: Platero, C. (2005).

[13] Una transformación es extensiva si el conjunto entrada está incluido en el conjunto salida.

Descrito lo que implica la aplicación de esta transformación morfológica, se hace mención de *dilate*, función disponible en OpenCV que permite llevar a cabo esta operación. A continuación, es detallada para su uso en el lenguaje de programación de C++ (Opencv dev team, 2019):

```
C++: void dilate(InputArray src, OutputArray dst, InputArray Kernel, Point
anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT,
const Scalar& borderValue=morphologyDefaultBorderValue())
```

Parámetros:

- ***InputArray src*:** Imagen de entrada.

- ***OutputArray dst*:** Imagen de salida que debe tener el mismo tamaño y profundidad de la imagen de entrada.

- ***InputArray Kernel*:** Elemento estructurante usado para la dilatación de la imagen de entrada.

- ***Point anchor*:** Posición del anclaje dentro del núcleo. Si Point anchor = Point (-1.-1), el centro del núcleo es tomado como la posición del anclaje.

- ***int iterations*:** Número de veces que es aplicada la transformación.

- ***int border Type*:** Método de extrapolación de píxeles en una condición de contorno.

- ***const Scalar&borderValue*:** Valor de los píxeles en el contorno si borderType = BORDER_CONSTANT.

Para entender el efecto de las transformaciones de erosión y dilatación binaria, en la Figura 32 se muestra el resultado tras aplicar las funciones de *erode* y *dilate* a la imagen binaria mostrada en la Figura 29. Como se observa, el ruido ha sido suprimido. Se hace evidente entonces la importancia de considerar la aplicación de estas transformaciones morfológicas en los sistemas de visión artificial para la detección de objetos.

Figura 32. Resultados tras la aplicación de la función *erode* y *dilate* a una imagen binaria.
Fuente: (Marín Abrego, s.f.).

5. Desarrollo del sistema de seguimiento

En esta parte del capítulo se describe cómo llegar a desarrollar el sistema de seguimiento. Se detallan algunas de las funciones que ofrece la biblioteca de OpenCV que son empleadas en el procesamiento de imagen para cumplir con el objetivo del sistema. Cabe mencionar que la biblioteca a emplear es de código abierto.

5.1. Procesamiento de imagen para el seguimiento del marcador. El problema de seguimiento puede ser definido como un problema de estimación de trayectoria para un objeto que se encuentra en constante movimiento dentro de una imagen plana; por lo tanto, el sistema debe mostrar la posición del objeto en cada *frame* capturado en video. Las funciones a emplear para cumplir con este objetivo se detallan a continuación (Pérez González, 2016):

5.1.1. *Identificación de contornos.* Del sistema de detección se obtiene como resultado una imagen binaria como la que se muestra en la Figura 32. Cabe recordar que la característica principal de la misma es que el objeto detectado, y posterior a seguir, se muestra de color blanco, mientras que las regiones que no interesan de color negro. Dicho esto, se puede pensar que solo es necesario buscar una función capaz de seguir la región de color blanco en la imagen binaria; sin embargo, esto es solo conveniente cuando se trata de objetos estáticos, puesto que al tratarse de objetos en movimiento, como es el caso de la presente tesis, se generan matrices de grandes dimensiones con las que son difícil tratar. Por ello, es necesaria una función intermedia que permita reducir la información a una cantidad suficiente que simplifique los cálculos y permita realizar un seguimiento en tiempo real.

De las funciones disponibles en OpenCV, *findContours* es la que permite simplificar la información del objeto a detectar a través de una reducción de la región detectada (mostrada de

color de blanco), a una nueva formada por los contornos de la misma. La aplicación de esta función a una imagen binaria representa una considerable reducción de los datos a procesar para el seguimiento de un objeto (Pérez González, 2016). A continuación, la función es detallada para su uso en el lenguaje de programación de C++ (Opencv dev team, s.f.):

```
C++: void findContours(InputOutputArray image, OutputArrayOfArrays contours,
OutputArray hierarchy, int mode, int method, Point offset=Point())
```

Parámetros:

- ***InputOutputArray image*:** Imagen binaria de entrada.

- ***OutputArrayOfArrays contours*:** Contornos detectados y almacenados como un vector de puntos.

- ***OutputArray hierarchy*:** Vector opcional de salida que contiene información sobre la topología de la imagen; es decir, la manera en cómo están relacionados los contornos detectados. Así mismo, contiene el mismo número de elementos que de contornos detectados.

- ***int mode*:** Modo de recuperación de los contornos. Se cuenta con los siguientes:

  - ***CV_RETR_EXTERNAL*:** Se recupera solo los contornos externos.

  - ***CV_RETR_LIST*:** Se recuperan todos los contornos sin haber establecido alguna relación jerárquica.

  - ***CV_RETR_CCOMP*:** Se recuperan los contornos sin ninguna modificación en sus relaciones jerárquicas. Cabe mencionar que es el modo de recuperación a emplear.

  - ***CV_RETR_TREE*:** Usando este modo se reconstruye una jerarquía completa de contornos anidados.

- ***int method*:** Métodos de aproximación de contornos. Se cuenta con los siguientes:

- ***CV_CHAIN_APPROX_NONE*:** Método que almacena todos los puntos del contorno sin realizar ninguna aproximación.

- ***CV_CHAIN_APPROX_SIMPLE*:** Método que comprime los segmentos horizontales, verticales y diagonales con el fin de reducir los puntos del contorno. Cabe mencionar que debido a sus buenos resultados, con un mínimo de datos del contorno, es el método de aproximación a emplear.

- ***CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS*:** Método que aplica un algoritmo de detección de puntos dominantes en curvas digitales, propuesto por Cho-Huak Teh y Roland T.Chin (Teh & Chin, 1989).

- ***Point offset*:** Desplazamiento opcional de cada punto identificado del contorno.

5.1.2.*Aproximación de curvas.* Si bien la función de detección de contornos ha reducido la cantidad de información a procesar, es posible reducirla aún más. De las funciones disponibles en OpenCV, *approxPolyDP* es la que permite aproximar una curva o polígono a otra curva o polígono pero con una menor cantidad de vértices que los originales, representando un tiempo de procesamiento menor para futuros cálculos.

La función está basada en el algoritmo matemático de Ramer-Douglas-Peucker (Ramer, 1972), capaz de reducir el número de puntos en una curva, de tal modo que partiendo de una curva compuesta por segmentos se obtiene una curva aproximada, similar a la original, pero con un número menor de puntos. Esto es posible definiendo condiciones que permitan agrupar los puntos en función de la máxima distancia entre la curva original y la curva aproximada. Además del algoritmo de Ramer-Douglas-Peucker (RDP), existen más algoritmos que permiten simplificar una curva, tales como el de Visvalingam-Whyatt, Reumann-Witkam, Opheim, etc; sin embargo, se opta por el algoritmo RDP debido a que se encuentra implementado en la biblioteca de OpenCV y ofrece muy buenos resultados (Pérez González, 2016). A continuación, la función es detallada para su uso en el lenguaje de programación de C++ (Opencv dev team, s.f.):

```
C++: void approxPolyDP(InputArray curve,OutputArray approxCurve,
double epsilon, bool closed)
```

Parámetros:

- ***InputArray curve*:** Curva de entrada compuesta por un vector de puntos 2D.

- ***OutputArray approxCurve*:** Curva aproximada del mismo tipo que la curva de entrada.

- ***double epsilon*:** Parámetro que especifica la precisión de aproximación. Es la distancia máxima entre la curva de entrada y su aproximación.

- ***bool closed*:** Parámetro que solo puede tomar dos valores: *True* y *False*. Si es *True* la curva aproximada es cerrada; es decir, sus primeros y últimos puntos están conectados; en caso contrario, la curva aproximada es abierta.

Para entender el resultado de aplicar la función *approxPolyDP*, en la Figura 33 se muestra una imagen binaria con el contorno de los objetos a seguir, previamente identificados, de color azul. Al aplicar la función de aproximación se obtiene como resultado la misma imagen binaria, pero con los contornos formados por segmentos que consideran un menor número de puntos, sin alejarse de la tendencia de los contornos originales.

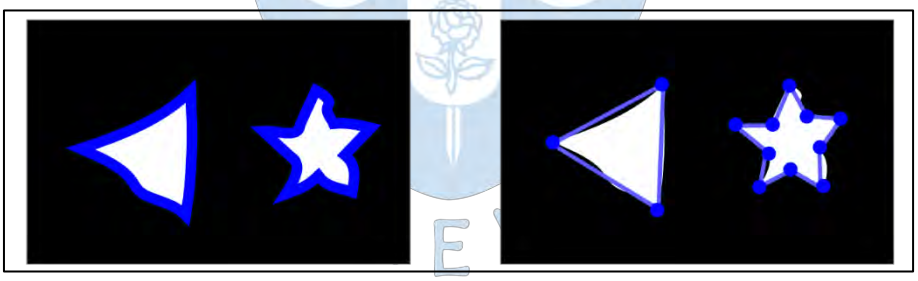

Figura 33. Resultados tras aplicar la función *approxPolyDP* a una imagen binaria con contornos identificados.
Fuente: (Robologs, 2015).

5.1.3.*Circulo circunscrito.* Finalmente, tras aplicar una función de identificación de contornos y aproximación de los mismos, se busca una función que permita obtener las coordenadas del marcador a emplear en el plano imagen.

Para ello, de las funciones disponibles en OpenCV, *minEnclosingCircle* permite encontrar el círculo de área mínima que encierra a un conjunto de puntos en un plano imagen, de tal modo

que las coordenadas del centro del círculo trazado representan las coordenadas del marcador. A continuación, la función es detallada para su uso en el lenguaje de programación de C++ (Opencv dev team, s.f.):

```
C++: void minEnclosingCircle(InputArray points, Point2f& center,
float& radius)
```

Parámetros:

- ***InputArray points*:** Vector de entrada compuesto por puntos 2D.
- ***Point2f& center*:** Centro del círculo encontrado.
- ***float& radius*:** Radio del círculo encontrado.

Para entender los resultados al aplicar la función *minEnclosingCircle*, se muestra en la Figura 34 la imagen binaria de un objeto cuya forma circular representa al marcador a emplear en la presente tesis. Al aplicar la función, tras una previa identificación de contornos, se tiene como resultado el trazado de un círculo que encierra a los puntos del contorno; obteniendo, a través de las coordenadas del centro del círculo, un medio para seguir la trayectoria del marcador.

Figura 34. Resultados tras aplicar la función *minEnclosingCircle* a una imagen binaria con contornos identificados.
Fuente: Elaboración propia.

Cabe mencionar que el uso de esta función, en lugar de una función de reconocimiento de figuras circulares, evita que el sistema presente problemas ante la aparición de falsos círculos

generados por efecto del reflejo de la luz del ambiente en la superficie del objeto detectado. Un ejemplo de lo mencionado se muestra en la Figura 35, en donde la región detectada cuenta con un área circular en su interior producto del reflejo de la luz en el objeto. De usar una función de reconocimiento de círculos, el sistema hubiera detectado dos círculos; es decir, dos marcadores, causando resultados erróneos. Sin embargo, con la aplicación de la función detallada se puede observar que el sistema solo traza un único círculo que comprende al círculo interior, evitando futuros errores de posicionamiento del marcador en el espacio.

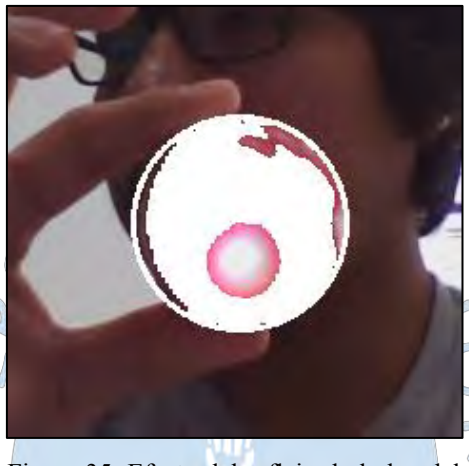

Figura 35. Efecto del reflejo de la luz del ambiente en la superficie de un objeto.
Fuente: Elaboración propia.

Las pruebas del sistema de detección y seguimiento son llevadas a cabo y mostradas en el Capítulo 5.

Capítulo 3
Sistema de conversión de espacios 2D a 3D

1. Introducción

En el capítulo anterior se detalla el desarrollo de un sistema de detección y seguimiento mediante cámaras, siendo el primer paso para cumplir con el objetivo de la presente tesis. En el presente capítulo, se hace una descripción de los algoritmos y conceptos empleados para el desarrollo de un sistema que, en complemento con el sistema antes mencionado, permita obtener la posición en el espacio de un objeto a partir de su posición en el plano imagen de las cámaras.

2. Modelo de cámara

Uno de los puntos más importantes en los múltiples sistemas de visión artificial es el estudio de las relaciones que vinculan las coordenadas tridimensionales en el espacio con las coordenadas bidimensionales en el plano imagen de las cámaras. Estas relaciones dependen de los siguientes parámetros (Fernández Lorenzo, 2005):

- **Parámetros extrínsecos:** Describen la posición y orientación de la cámara con respecto a un sistema de coordenadas de referencia definido.

- **Parámetros intrínsecos:** Describen las características internas de la cámara, tales como distancia focal, inclinación de los píxeles, distorsión de las lentes, etc.

En la Figura 36 se pueden observar las relaciones existentes que vinculan la coordenada 3D de un objeto y la coordenada del mismo en un plano 2D visto desde la cámara (Yun, Park, Choi, & Lee, 2004).

En vista de la importancia de los parámetros mencionados, en esta parte del capítulo se estudian distintos modelos de cámaras, de los cuales se elige el más indicado a emplear dependiendo del tipo de entorno de trabajo y de la precisión que se desea obtener en las medidas. Cabe mencionar que la precisión está estrechamente ligada con la exactitud con la que son obtenidos los parámetros del modelo (Fernández Lorenzo, 2005).

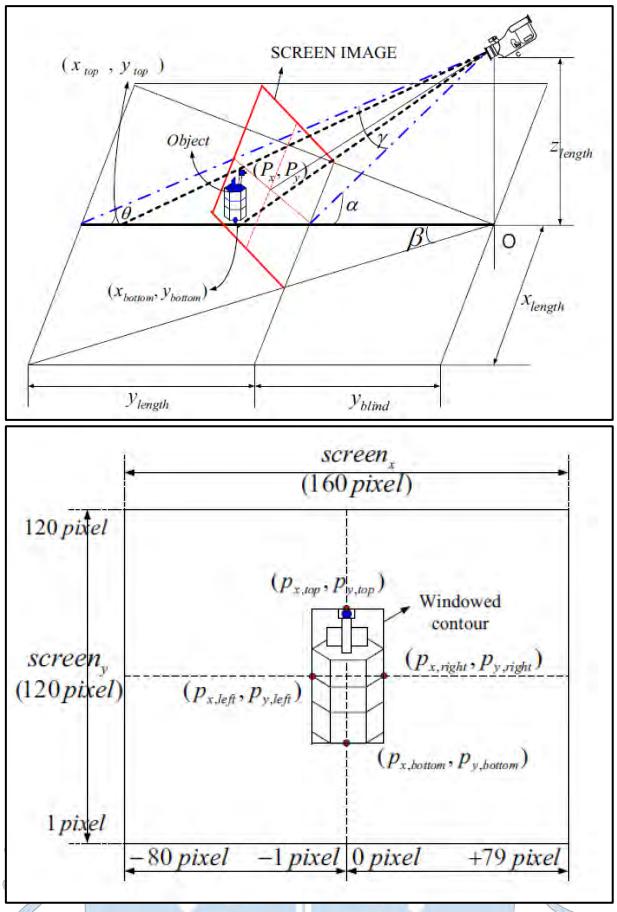

Figura 36. Modelamiento para la correspondencia entre el plano imagen 2D y la coordenada 3D de un móvil (a). Representación en coordenadas pixélicas del plano imagen visto desde la cámara (b).
Fuente: (Yun, Park, Choi, & Lee, 2004).

2.1. Modelos sin distorsión. Modelos basados en algoritmos matemáticos que aproximan el proceso de formación de una imagen haciendo la hipótesis que los lentes empleados en la óptica son ideales; es decir, sin presencia de aberraciones ópticas[14] (Fernández Lorenzo, 2005).

2.1.1. *Modelo de proyección perspectiva o cónica.* Este modelo de cámara es el más empleado debido a su sencillez y buenos resultados en aplicaciones prácticas. Comúnmente, es conocido como modelo *pin-hole* o modelo de lente delgado. En la Figura 37 se presenta la configuración geométrica del modelo que sitúa al plano imagen en el centro óptico **O**, de tal modo que la imagen no aparece invertida (Fernández Lorenzo, 2005) (Prieto, 2014).

[14] Defecto óptico que da lugar a desviaciones en el sistema óptico, produciendo falta de nitidez o alteraciones en las imágenes capturadas.

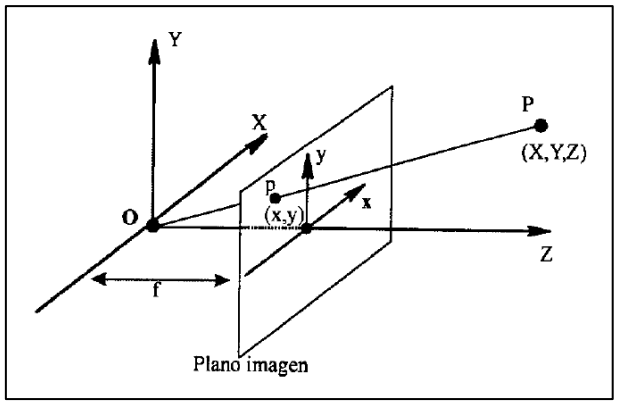

Figura 37. Modelo de proyección perspectiva o cónica.
Fuente: (Fernández Lorenzo, 2005).

Como se observa, el modelo proyecta en un plano imagen un punto **P**, de coordenadas (X,Y,Z), que se encuentra en el espacio; siendo **p**, de coordenadas (x,y), el punto proyectado con un escalamiento **λ** que depende de la distancia focal **f** y de la distancia del objeto al plano imagen o focal. Referenciando ambos puntos a un sistema de coordenadas que se ubica en la propia cámara, se obtienen las siguientes expresiones (Fernández Lorenzo, 2005) (Calvillo Ardila, 2017):

$$x = \lambda X \qquad y = \lambda Y \qquad f = \lambda Z \tag{3.1}$$

De (3.1) se obtienen finalmente las relaciones de proyección:

$$x = f\frac{X}{Z} \qquad y = f\frac{Y}{Z} \tag{3.2}$$

En coordenadas homogéneas, artificio matemático empleado para describir de forma lineal la proyección de un punto en el plano imagen, se expanden las coordenadas 2D y 3D agregando una dimensión cuyo valor es uno (Fernández Lorenzo, 2005) (Calvillo Ardila, 2017):

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

2.1.2. ***Modelo de proyección ortográfica.*** Este modelo de cámara es una simplificación del modelo de proyección perspectiva que linealiza las ecuaciones de proyección, siendo ideal cuando es posible considerar que la distancia del objeto a la cámara es infinita. Supone que los rayos procedentes del objeto son paralelos al lente óptico; es decir, perpendiculares al plano imagen. Este modelo, por ejemplo, puede ser empleado en imágenes tomadas desde un avión o por un satélite, de tal modo que la información de profundidad es omitida. Las relaciones de proyección son las siguientes (Fernández Lorenzo, 2005):

$$x = X \qquad y = Y \tag{3.4}$$

2.1.3. ***Modelo de proyección débil.*** Este modelo de cámara, también denominado modelo de proyección ortográfica escalada, es una segunda simplificación del modelo de proyección perspectiva. Supone que todos los puntos se encuentran en un plano objeto paralelo al plano imagen y situado a una distancia fija ($Z = Z_0$). Por lo tanto, también es posible usar este modelo en imágenes tomadas desde un avión o por un satélite, en donde la distancia de la cámara al objeto es mucho mayor que las dimensiones del objeto. Las relaciones de proyección son las siguientes:

$$x = f\frac{X}{Z_0} \qquad y = f\frac{Y}{Z_0} \tag{3.5}$$

2.1.4. ***Modelo de proyección paraperspectiva.*** En el modelo de proyección débil se proyecta un punto del entorno sobre un plano objeto haciendo uso de rectas paralelas al eje óptico; sin embargo, esta consideración causa errores de aproximación que están en función de la distancia del objeto al eje óptico, aumentando el error a medida que aumenta la distancia.

En vista a este problema, el modelo de proyección paraperspectiva permite disminuir el error haciendo que las rectas de proyección sean paralelas a una recta que une el centroide del objeto, que pertenece al entorno y de coordenadas $[X_G, Y_G, Z_G]$, con el centro óptico; esta nueva recta es denominada recta de proyección central. Este modelo sigue empleando un plano objeto, de tal modo que las rectas de proyección de todos los puntos se proyectan paralelas a la recta de proyección central hasta este plano. Posteriormente, los puntos son proyectados del plano de profundidad media al plano imagen de forma perspectiva, siendo las relaciones de proyección las siguientes (Fernández Lorenzo, 2005):

$$x = f\frac{X}{Z_G} - \frac{X_G}{Z_G^2}Z + \frac{X_G}{Z_G} \qquad y = f\frac{Y}{Z_G} - \frac{Y_G}{Z_G^2}Z + \frac{Y_G}{Z_G} \tag{3.6}$$

2.2. Modelos con distorsión. Como se menciona en el apartado **2.1** del presente capítulo, los modelos mencionados se encuentran bajo la hipótesis de que las lentes empleadas en la óptica no tienen presencia de aberraciones ópticas; sin embargo, en la vida real las ópticas introducen aberraciones que desvían la proyección de los puntos del entorno en el plano imagen. Estas aberraciones son causadas por la naturaleza del medio óptico, puesto que, en función de la longitud de onda, presentan un comportamiento distinto ante los fenómenos de refracción[15] y dispersión[16]. En consecuencia, se introducen nuevos parámetros intrínsecos en la descripción del modelo interno de la cámara (Fernández Lorenzo, 2005).

2.2.1. ***Modelo de proyección perspectiva con distorsión.*** Considerando la distorsión en el modelo de proyección perspectiva, se modifica la posición de los puntos en el plano imagen de tal forma que no siguen las ecuaciones de proyección (3.2). En este caso, el punto **P** se proyecta en un punto **p′** que se encuentra distanciado del punto **p** una magnitud que está en función de su posición en el plano imagen. En la Figura 38 se puede observar lo mencionado (Fernández Lorenzo, 2005).

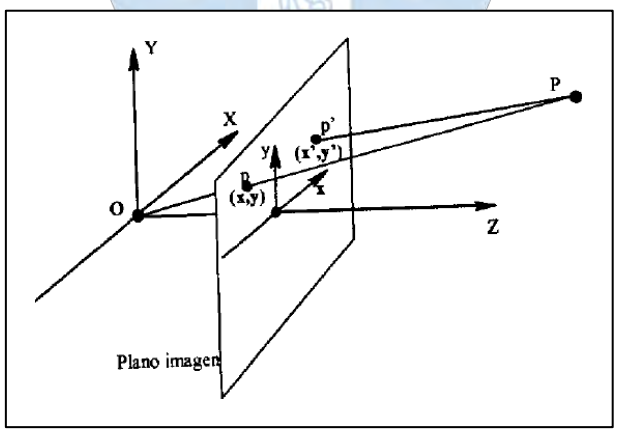

Figura 38. Modelo de proyección perspectiva con distorsión.
Fuente: (Fernández Lorenzo, 2005).

[15] Cambio en la dirección de un rayo de luz al pasar de un medio óptico a otro.
[16] Separación de las ondas con distinta frecuencia al atravesar un material.

De la Figura 38 se identifica a (x,y) como las coordenadas del punto **p** y a (x′,y′) como las coordenadas del punto **p′**, de tal modo que se obtienen las siguientes expresiones:

$$\begin{aligned} x' &= x + \delta_x(x,y) = x + \delta_{xr}(x,y) + \delta_{xt}(x,y) \\ y' &= y + \delta_y(x,y) = y + \delta_{yr}(x,y) + \delta_{yt}(x,y) \end{aligned} \tag{3.7}$$

De las ecuaciones (3.7), los términos $\boldsymbol{\delta_x}$ y $\boldsymbol{\delta_y}$ resumen de forma general la distorsión, la cual se puede modelar en función del tipo de distorsión que predomina. Por otro lado, los valores que componen la componente radial $\boldsymbol{(\delta_{xr},\delta_{yr})}$ y tangencial $\boldsymbol{(\delta_{xt},\delta_{yt})}$ son modelados con una serie de infinitos términos. Sin embargo, para muchas aplicaciones es suficiente con el uso de dos términos para cada una, de tal modo que la distorsión puede modelarse usando las siguientes expresiones:

$$\begin{aligned} \delta_{xr} &= x(k_1r^2 + k_2r^4) \\ \delta_{yr} &= y(k_1r^2 + k_2r^4) \\ \delta_{xt} &= 2p_1xy + p_2(r^2 + 2x^2) \\ \delta_{yt} &= p_1(r^2 + 2y^2) + 2p_2xy \\ r &= \sqrt{(x^2 + y^2)} \end{aligned} \tag{3.8}$$

A partir de (3.8) es posible expresar las expresiones (3.7) como:

$$\begin{aligned} x' &= x + x(k_1r^2 + k_2r^4) + 2p_1xy + p_2(r^2 + 2x^2) \\ y' &= y + y(k_1r^2 + k_2r^4) + p_1(r^2 + 2y^2) + 2p_2xy \end{aligned} \tag{3.9}$$

En donde $\mathbf{k_1}$ y $\mathbf{k_2}$ son los coeficientes que modelan la distorsión radial y $\mathbf{p_1}$ y $\mathbf{p_2}$ los que modelan la distorsión tangencial. Las expresiones (3.9) pueden ser expresadas de forma matricial como:

$$p' = p + \delta(p) \tag{3.10}$$

En donde **δ(p)** representa la distorsión del punto **p**.

2.2.2. ***Corrección de la distorsión.*** Para poder emplear el modelo sin distorsión visto en el apartado **2.1.1** del presente capítulo, es necesario hacer una corrección de la distorsión que introduce la óptica de la cámara, puesto que afecta de manera significativa a la obtención de información métrica precisa. Para ello, se plantea una aproximación de las coordenadas del punto **p** a partir de las coordenadas del punto **p′**.

Giorgio Toscani y Olivier Faugeras proponen un método que permite convertir una imagen con distorsión a una sin distorsiones a partir de la aproximación de **p** en función de **p′** (Faugeras & Toscani, 1987), empleando una transformación bilineal en pequeñas regiones de la imagen. Una propuesta más simple que la anterior es la planteada por Trond Melen, la cual propone el uso de la siguiente expresión (Fernández Lorenzo, 2005) (Melen, 1994):

$$p \approx p' - \delta(p) \tag{3.11}$$

La expresión (3.11) puede convertirse en la siguiente función recursiva:

$$\begin{aligned} p &= p' - \delta(p) \\ p &= p' - \delta(p' - \delta(p)) \\ p &= p' + \delta(p' - \delta(p' - \delta(p))) \\ p &= p' + \delta(p' + \delta(p' - \delta(p' - \delta(p)))) \end{aligned} \tag{3.12}$$

En la práctica, con 2 o 3 iteraciones ya es suficiente para corregir la distorsión, incluso en casos extremos. Por otro lado, Janne Heikkilä (Heikkilä & Silvén, 1997) propone un modelo alternativo al propuesto por Melen, el cual reduce el tiempo de cómputo desarrollando en series de Taylor la función **δ(p)** en la expresión (3.11) alrededor de **p**, obteniendo lo siguiente (Fernández Lorenzo, 2005):

$$p \approx p' - \frac{\delta(p')}{4k_1r^2 + 6k_2r^4 + 8p_1y + 8p_2x + 1} \tag{3.13}$$

Un modelo más elaborado pero de mayor coste computacional, propone emplear un algoritmo que resuelve la corrección de la distorsión entre la imagen original y la corregida mediante un proceso de inversión de la imagen original (Fernández Lorenzo, 2005).

De los modelos de cámara descritos, se opta por el uso del modelo de proyección perspectiva, o más conocido como modelo *pin-hole*, por ser uno de los modelos más usado en aplicaciones de visión artificial, además de ser de fácil implementación y brindar buenos resultados. A continuación se realiza un estudio al detalle del mismo.

3. Modelo de cámara *Pin-Hole*

Este modelo de cámara, como se menciona en el apartado **2.1.1** del presente capítulo, es el modelo más empleado para modelar la formación de imágenes en una cámara digital. Este no es un modelo físico de la cámara, ni de los transductores que conforman al sensor de la misma (CDD o CMOS); sin embargo, es usado por las ventajas que proporciona la sencillez de sus ecuaciones que modelan el comportamiento físico de la cámara, además de proporcionar una suficiente precisión para ser empleada en aplicaciones robóticas u otras disciplinas (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018).

Según este modelo, la imagen obtenida se define a partir del esquema que se muestra en la Figura 39, en donde el punto **C** es el centro de la cámara, o centro óptico, y el punto **P**, que pertenece al entorno, se proyecta en el plano imagen o focal a una distancia **f** del centro de la cámara, esta distancia se denomina distancia focal (Calvillo Ardila, 2017).

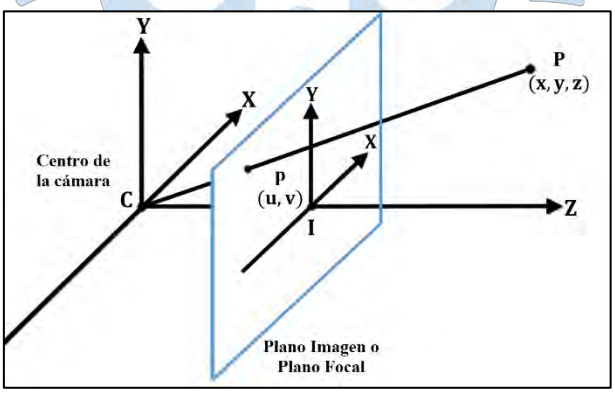

Figura 39. Modelo de cámara *Pin-Hole.*
Fuente: (Calvillo Ardila, 2017)

Además de esta convención del modelo, se consideran los siguientes puntos (Calvillo Ardila, 2017):

- En el plano imagen o focal, el eje X es horizontal y crece en dirección hacia la derecha, el eje Y es vertical y crece en dirección hacia arriba y el eje Z es perpendicular y crece en dirección del centro de la cámara hacia el plano focal.

- El eje óptico pasa por el centro de la cámara **C** y es perpendicular al plano focal.

- El punto **I** se encuentra en la intersección del plano imagen y el eje óptico, de tal modo que en una foto **P** está cerca del centro de la imagen, pero no exactamente en el centro de la cámara.

- Si el centro de la cámara se escoge como origen para el sistema de referencia de **P**, de coordenadas (x,y,z); los ejes de este punto son paralelos a X y Y; y el eje Z se considera como el eje óptico, se cumple con lo siguiente:

$$\frac{x}{z} = \frac{u}{f} \qquad \frac{y}{z} = \frac{v}{f} \tag{3.14}$$

3.1. Matriz de cámara. Cuando se observa una fotografía se es capaz de interpretar distancias en las tres dimensiones dentro de la misma, por ejemplo, en la foto que se muestra en la Figura 40 se intuye que el muro se encuentra a unos 6 cm, aproximadamente, del zapato de la persona que aparece de pie en la foto. Esta estimación es posible gracias a una correspondencia realizada entre los puntos en el espacio y los puntos en la fotografía, la cual se puede modelar a través de una transformación lineal que emplea una matriz, de dimensiones 3x4, que permite llevar un punto del espacio 3D al 2D. La transformación lineal expresada en coordenadas homogéneas y asumiendo que el origen de coordenadas se encuentra en el centro de la cámara, se puede expresar con la siguiente ecuación (Calvillo Ardila, 2017) (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = [P] \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{3.15}$$

Según la expresión (3.15), para obtener la proyección de un punto en el espacio, de coordenadas (x_c,y_c,z_c), es necesario convertirlo a coordenadas homogéneas y resolver la matriz de cámara definida, de tal modo que se obtiene lo siguiente:

$$\lambda u = fx_c \rightarrow u = \frac{fx_c}{\lambda}$$
$$\lambda v = fy_c \rightarrow v = \frac{fy_c}{\lambda} \tag{3.16}$$
$$\lambda = z_c$$

Finalmente, el resultado de (3.16) es el siguiente:

$$u = f\frac{x_c}{z_c} \qquad v = f\frac{y_c}{z_c} \tag{3.17}$$

Cabe mencionar que la expresión (3.17) coincide con la relación de proyección mencionada en (3.2). Como se observa, al tener (3.15) solución única, para proyectar un punto del entorno en el plano imagen de la cámara, considerando que el origen de coordenadas se encuentra en el centro de la cámara, basta con definir la matriz de cámara haciendo uso del valor de la distancia focal y las coordenadas homogéneas del punto a proyectar.

Figura 40. Ejemplo de estimación de distancias en las tres dimensiones.
Fuente: (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018).

3.2. Parámetros intrínsecos. Como se menciona en el apartado **2** del presente capítulo, son aquellos que definen la geometría interna y óptica de la cámara. Los mismos pueden ser expresados por una matriz cuyos componentes son los siguientes (Calvillo Ardila, 2017) (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

- **f_x y f_y:** Estas componentes proporcionan información sobre las dimensiones de los píxeles, los cuales representan la matriz de sensores de luz, generalmente de forma cuadrada, que conforman una cámara digital.

- **c_x y c_y:** Representan las coordenadas, en píxeles, del centro del plano imagen o focal, el cual es definido como la intersección del plano imagen y el eje óptico.

- **s:** Denominado *skew*, representa la inclinación del pixel; sin embargo, suele tener un valor muy pequeño o casi cero.

En la Figura 41 se muestran todas las componentes mencionadas:

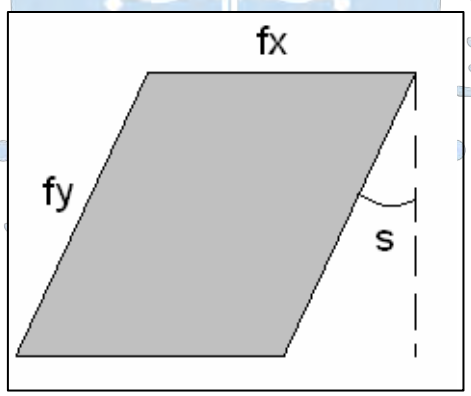

Figura 41. Geometría de un pixel.
Fuente: (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018).

Finalmente, los parámetros intrínsecos pueden ser expresados en forma matricial como:

$$A = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

3.3. Parámetros extrínsecos. Como se menciona en el apartado **2** del presente capítulo, son aquellos que relacionan el sistema de coordenadas del mundo real con el sistema de referencia de la cámara, puesto que pueden no coincidir al estar la cámara desplazada y rotada en el espacio. Por lo tanto, se puede dividir a estos parámetros en un vector de traslación y un vector de rotación (Calvillo Ardila, 2017) (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018).

3.3.1. ***Vector de traslación.*** Vector que describe el desplazamiento de la cámara en el espacio. Para ser definido, es necesario conocer los desplazamientos de la cámara a lo largo de las tres dimensiones (X,Y,Z). Puede ser expresada de la siguiente manera:

$$t = [t_x, t_y, t_z] \tag{3.19}$$

3.3.2. ***Matriz de rotación.*** Matriz cuyo fin es alinear, en la misma dirección y sentido, los ejes X, Y y Z de los sistemas de referencia del mundo real y de la cámara. A su vez, la matriz está compuesta por 3 matrices de rotación.

Estos corresponden a la rotación con respecto al eje X, especificada con un ángulo α; la rotación con respecto al eje Y, especificada con un ángulo β; y la rotación con respecto al eje Z especificada con un ángulo θ. En la Figura 42 se muestran los ángulos de rotación mencionados.

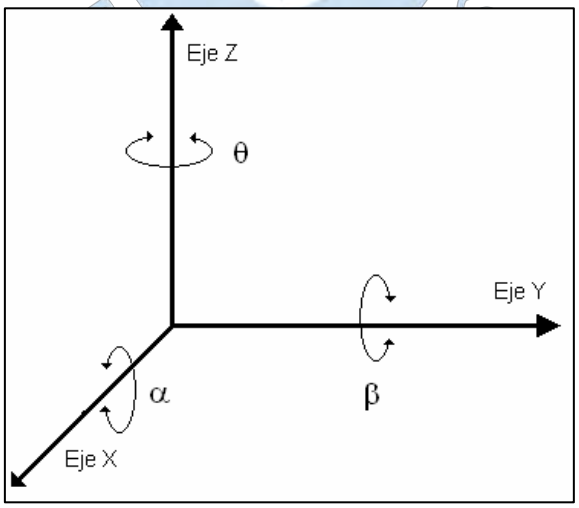

Figura 42. Ángulos de rotación con respecto a los ejes X, Y y Z.
Fuente: (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018).

En relación a las 3 matrices de rotación, pueden ser expresadas de la siguiente forma:

$$R_\alpha = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \text{sen}(\alpha) \\ 0 & -\text{sen}(\alpha) & \cos(\alpha) \end{bmatrix}$$
$$R_\beta = \begin{bmatrix} \cos(\beta) & 0 & -\text{sen}(\beta) \\ 0 & 1 & 0 \\ \text{sen}(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad (3.20)$$
$$R_\theta = \begin{bmatrix} \cos(\theta) & -\text{sen}(\theta) & 0 \\ \text{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Con el fin de contar con solo una matriz de rotación **R** que facilite su utilización, es posible multiplicar las matrices (3.20) y obtener lo siguiente:

$$R = \begin{bmatrix} \cos(\beta)\cos(\theta) & -\cos(\beta)\text{sen}(\theta) & -\text{sen}(\beta) \\ \text{sen}(\alpha)\text{sen}(\beta)\cos(\theta) + \cos(\alpha)\text{sen}(\theta) & -\text{sen}(\alpha)\text{sen}(\beta)\,\text{sen}(\theta) + \cos(\alpha)\cos(\theta) & \text{sen}(\alpha)\cos(\beta) \\ \cos(\alpha)\text{sen}(\beta)\cos(\theta) - \text{sen}(\alpha)\text{sen}(\theta) & -\cos(\alpha)\text{sen}(\beta)\,\text{sen}(\theta) - \text{sen}(\alpha)\cos(\theta) & \cos(\alpha)\cos(\beta) \end{bmatrix} \quad (3.21)$$

De manera simplificada, se puede denotar a la matriz de rotación **R** como:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{21} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (3.22)$$

Finalmente, la matriz de parámetros extrínsecos considerando al vector de traslación **t** y a la matriz de rotación **R**, es expresada como:

$$[R \quad t] = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{21} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{array}\right] \quad (3.23)$$

Si bien la ecuación (3.15) permite hallar la proyección de un punto que pertenece al entorno en el plano imagen de la cámara, la misma es formulada suponiendo que el sistema de referencia del entorno es el mismo para la cámara. Sin embargo, no es el caso del sistema propuesto en la presente tesis; por ello, es necesario agregar a la ecuación las matrices que representan el alineamiento de ambos sistemas de referencia, de tal modo que se tiene la siguiente ecuación (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

$$\lambda m = A[R \quad t]M \tag{3.24}$$

Donde:

- $M = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$: Coordenadas homogéneas de un punto 3D en el espacio.

- $m = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$: Coordenadas homogéneas del mismo punto pero proyectado en el plano imagen 2D de la cámara.

- λ: Escalar mencionado en el apartado **2.1.1** del presente capítulo.

- $[R \quad t]$: Matriz de parámetros extrínsecos.

- A: Matriz de parámetros intrínsecos de la cámara.

Reemplazando las matrices (3.18) y (3.23), la ecuación (3.24) queda expresada como:

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} f_x & s & u_x \\ 0 & f_x & v_x \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{21} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.25}$$

Finalmente, si se realiza el producto entre la matriz de parámetros intrínsecos y la matriz de parámetros extrínsecos, la ecuación (3.25) puede ser denotada de la siguiente forma:

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.26}$$

4. Calibración de cámaras

Proceso que determina los parámetros intrínsecos y extrínsecos de la cámara; es decir, sus respectivas matrices. Cabe recordar que estas matrices intervienen en las ecuaciones que relacionan los puntos del entorno con los puntos proyectado en el plano imagen. Para ello, se

cuenta con distintas metodologías que permiten llevar a cabo este proceso, siendo algunas las que mencionan a continuación (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

4.1. Método de Fougeras. El proceso de calibración bajo este método permite conocer los parámetros intrínsecos y extrínsecos de la cámara, solo si es conocido un punto en el espacio y su respectiva proyección en el plano imagen. Para ello, se parte de la ecuación (3.25) en donde es posible calcular las coordenadas **u** y **v** asumiendo que $\boldsymbol{\lambda = 1}$, de tal modo que se obtiene el siguiente sistema de ecuaciones (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

$$\begin{aligned} u &= \frac{m_{11}x_i + m_{12}y + m_{13}z_i + m_{14}}{m_{31}x_i + m_{32}y_i + m_{33}z_i + m_{34}} \\ v &= \frac{m_{21}x_i + m_{22}y + m_{23}z_i + m_{24}}{m_{31}x_i + m_{32}y_i + m_{33}z_i + m_{34}} \end{aligned} \tag{3.27}$$

A partir de (3.27), el proceso de calibración consiste en encontrar los valores de $\mathbf{m_{ij}}$, por lo que se replantea el sistema de ecuaciones para un i-ésimo conjunto de puntos:

$$\begin{aligned} m_{11}x_i + m_{12}y_i + m_{13}z_i + m_{14} - \left(m_{31}x + m_{32}u_iy_i + m_{33}u_iz_i\right) &= u_im_{34} \\ m_{21}x_i + m_{22}y_i + m_{23}z + m_{24} - \left(m_{31}v_ix_i + m_{32}v_iy_i + m_{33}v_iz_i\right) &= v_im_{34} \end{aligned} \tag{3.28}$$

Sin embargo, para un futuro procesamiento computacional, es conveniente que (3.28) esté expresado en su forma matricial, tal como se muestra a continuación:

$$\begin{bmatrix} & & & & & & \vdots & \vdots & \vdots & & & \\ x_i & y_i & z_i & 1 & 0 & 0 & 0 & 0 & -u_ix_i & -u_iy_i & -u_iz_i \\ 0 & 0 & 0 & 0 & x_i & y_i & z_i & 1 & -v_ix_i & -v_iy_i & -v_iz_i \\ & & & & & & \vdots & \vdots & \vdots & & & \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} \vdots \\ u_im_{34} \\ v_im_{34} \\ \vdots \end{bmatrix} \tag{3.29}$$

Partiendo de (3.29), el método de Fougeras plantea resolver esta ecuación matricial asumiendo que $\mathbf{m_{34} = 1}$, puesto que se trata de un valor no nulo suponiendo que los sistemas de

referencia del entorno y de la cámara no coinciden, además de representar solo la componente **Z** del vector de traslación para el alineamiento de los sistemas de referencia.

Sin embargo, la desventaja de este método es que se obtiene un resultado escalado cuyo valor de escalamiento es una incógnita. Según lo mencionado, como resultado se tiene una ecuación matricial que puede ser resuelta mediante mínimos cuadrados:

$$KM = U \tag{3.30}$$

$$M = (K^T K^T)^{-1} U \tag{3.31}$$

Donde:

- $K = \begin{bmatrix} \vdots & \vdots & \vdots & & & & & & & \\ x_i & y_i & z_i & 1 & 0 & 0 & 0 & 0 & -u_i x_i & -u_i y_i & -u_i z_i \\ 0 & 0 & 0 & 0 & x_i & y_i & z_i & 1 & -v_i x_i & -v_i y_i & -v_i z_i \\ & & & & \vdots & \vdots & \vdots & & & & \end{bmatrix}$

- $M = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix}$

- $U = \begin{bmatrix} \vdots \\ u_i \\ v_i \\ \vdots \end{bmatrix}$

4.2. Método de Zhang. Proceso de calibración basado en la observación de un patrón plano de damero de ajedrez en distintas posiciones; a diferencia del método de Fougeras, para emplear este método no es necesario conocer un punto en el espacio con su correspondiente proyección, ni conocer las distintas posiciones de la cámara al tomar las capturas del patrón.

Por ello, se considera como una técnica versátil en lugar de otras en donde es necesaria una ardua preparación del entorno, ya que los puntos deben forman un plano y ser conocidos. En la Figura 43 se muestra el patrón empleado (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

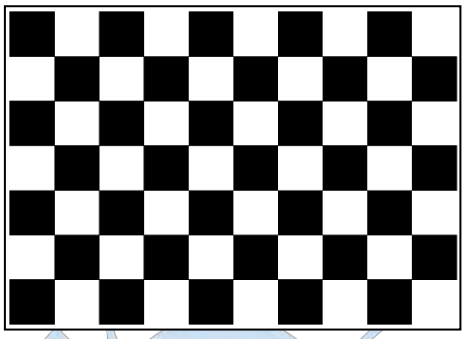

Figura 43. Patrón plano de damero de ajedrez para calibración.
Fuente (Bustamante Mejia & López Varona, 2014).

La ecuación empleada para la obtención de los parámetros intrínsecos y extrínsecos es la (3.25), pero considerando que el patrón plano se encuentra sobre $\mathbf{Z = 0}$ en el sistema de referencia del entorno. Denotando la columna i-ésima de la matriz de rotación por $\mathbf{r_i}$, se obtiene la siguiente ecuación:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A[r_1 \quad r_2 \quad r_3 \quad t] \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = A[r_1 \quad r_2 \quad t] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.32}$$

Donde **A** es la matriz de parámetros intrínsecos y **H** es conocida como la matriz de homografía[17]. De una manera más simplificada, (3.32) puede ser denotada de la siguiente manera:

$$\lambda m = HM \tag{3.33}$$

Donde:

$$H = [h_1 \quad h_2 \quad h_3] = A[r_1 \quad r_2 \quad t] \tag{3.34}$$

[17] Transformación proyectiva que determina una correspondencia entre dos figuras geométricas planas.

Al ser **R** una matriz de rotación, los vectores que la componen deben cumplir con las restricciones de ortonormalidad, de tal modo que $\mathbf{r_1}^T.\mathbf{r_2} = \mathbf{0}$ y $\mathbf{r_1}^T.\mathbf{r_1} = \mathbf{r_2}^T.\mathbf{r_2}$; por lo tanto, al extraer los vectores de rotación a partir de (3.34), obtenemos lo siguiente:

$$h_1^T A^{-T} A^{-1} h_2 = 0 \tag{3.35}$$

$$h_1^T A^{-T} A^{-1} h_1 = h_2^T A^{-T} A^{-1} h_2 \tag{3.36}$$

Las expresiones (3.35) y (3.36) son para una homografía, dos restricciones básicas de los parámetros intrínsecos. Para dar solución a este problema, se propone una solución analítica seguida de una optimización lineal. La matriz $\mathbf{A^{-T}A^{-1}}$ está compuesta por los parámetros intrínsecos de la cámara (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018) (Zhang, 1998):

$$B = A^{-T}A^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{c}{\alpha^2\beta} & \frac{cv_0 - u_0\beta}{\alpha^2\beta} \\ -\frac{c}{\alpha^2\beta} & \frac{c^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{c(cv_0 - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{cv_0 - u_0\beta}{\alpha^2\beta} & -\frac{c(cv_0 - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(cv_0 - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix} \tag{3.37}$$

Al ser **B** una matriz simétrica puede ser definida por un vector de 6 elementos, tal como se muestra a continuación:

$$b = [B_{11} \quad B_{12} \quad B_{22} \quad B_{13} \quad B_{23} \quad B_{33}]^T \tag{3.38}$$

A partir de (3.37), y asumiendo que $\mathbf{H_i} = [h_{i1} \quad h_{i2} \quad h_{i3}\ ]^T$, podemos obtener lo siguiente:

$$h_i^T B h_i = v_{ij}^T b \tag{3.39}$$

Donde:

$$v_{ij}^T = [h_{i1}h_{j1} \quad h_{i1}h_{j2} + h_{i2}h_{j1} \quad h_{i2}h_{j2} \quad h_{i3}h_{j1} + h_{i1}h_{j3} \quad h_{i3}h_{j2} + h_{i2}h_{j3} \quad h_{i3}h_{j3}] \tag{3.40}$$

Con las expresiones obtenidas es posible escribir las restricciones de los parámetros intrínsecos en dos ecuaciones homogéneas, las cuales están en función de **b**:

$$\begin{bmatrix} v_{12}{}^{T} \\ (v_{11} - v_{22})^{T} \end{bmatrix} b = Vb = 0 \tag{3.41}$$

Donde **V** es una matriz de dimensión 2**n**x6, de modo que si **n** ≥ 3 existe una solución general cuya única solución de **b** está definida con un factor de escala. Si **n** = 2, es posible añadir a la expresión (3.37) la restricción **c** = 0.

Hallado el valor de **b**, los parámetros intrínsecos de la cámara, que componen la matriz **A**, pueden ser hallados usando las siguientes expresiones (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

$$\begin{aligned} v_0 &= \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{12}B_{22} - B_{12}{}^2} \\ \lambda &= B_{33} - \frac{[B_{13}{}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]}{B_{11}} \\ \alpha &= \sqrt{\frac{\lambda}{B_{11}}} \\ \beta &= \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}{}^2}} \\ c &= -\frac{B_{12}\alpha^2\beta}{\lambda} \\ v_0 &= \frac{cv_0}{\beta} - \frac{B_{13}\alpha^2}{\lambda} \end{aligned} \tag{3.42}$$

Conocida la matriz de parámetros intrínsecos **A**, la matriz de parámetros extrínsecos puede ser hallada usando las siguientes expresiones (Zhang, 1998):

$$\begin{aligned} r_1 &= \lambda A^{-1}h_1 \\ r_2 &= \lambda A^{-1}h_2 \\ r_3 &= r_1 x r_2 \\ t &= \lambda A^{-1}h_3 \\ \lambda &= \frac{1}{\|A^{-1}h_1\|} = \frac{1}{\|A^{-1}h_2\|} \end{aligned} \tag{3.43}$$

Con respecto a la distorsión radial, esta puede ser modelada por un polinomio de dos términos, siendo **(u,v)** la coordenada ideal de un pixel y **($\breve{u}$,$\breve{v}$)** su coordenada real en la imagen. Así mismo, **(x,y)** y **($\breve{x}$,$\breve{y}$)** son estas mismas coordenadas pero normalizadas en la imagen, de tal modo que se cumplen las siguientes expresiones (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018) (Zhang, 1998):

$$\begin{aligned}\breve{x} &= x + x\left[k_1(x^2+y^2) + k_2(x^2+y^2)^2\right] \\ \breve{y} &= y + y\left[k_1(x^2+y^2) + k_2(x^2+y^2)^2\right]\end{aligned} \tag{3.44}$$

Donde $\mathbf{k_1}$ y $\mathbf{k_2}$ son los coeficientes de la distorsión radial. Considerando el punto principal de la cámara como el centro de la distorsión radial **($\breve{\upsilon} = \upsilon_0 + \alpha\breve{x} + c\breve{y}$, $\breve{v} = v_0 + \beta\breve{y}$)** y asumiendo que $\mathbf{c = 0}$, tenemos lo siguiente (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

$$\breve{\upsilon} = \upsilon + (\upsilon - \upsilon_0)\left[k_1(x^2+y^2) + k_2(x^2+y^2)^2\right] \tag{3.45}$$

$$\breve{v} = v + (v - v_0)\left[k_1(x^2+y^2) + k_2(x^2+y^2)^2\right] \tag{3.46}$$

Para estimar los coeficientes de la distorsión radial, se expresa (3.45) y (3.46) en forma matricial, de tal modo que tenemos lo siguiente:

$$\begin{bmatrix}(\upsilon - \upsilon_0)(x^2+y^2) & (\upsilon - \upsilon_0)(x^2+y^2)^2 \\ (v - v_0)(x^2+y^2) & (v - v_0)(x^2+y^2)^2\end{bmatrix}\begin{bmatrix}k_1 \\ k_2\end{bmatrix} = \begin{bmatrix}\breve{\upsilon} - \upsilon \\ \breve{v} - v\end{bmatrix} \tag{3.47}$$

Para **m** puntos que se encuentran en **n** imágenes, es posible juntarlas y obtener 2**mn** ecuaciones, o expresarlas en forma de matriz como:

$$Dk = d \tag{3.48}$$

Donde:

$$k = [k_1, k_2]^T \tag{3.49}$$

Y cuya solución puede ser hallada mediante mínimos cuadrados:

$$k = (D^T D)^{-1} d \tag{3.50}$$

4.3. Otros patrones de calibración para el método de Zhang. Si bien el procedimiento de calibración según el método de Zhang está basado en la captura de un patrón plano de damero de ajedrez en distintas posiciones, no es el único tipo de patrón que es posible emplear. A continuación se detallan otros tipos de patrones usados (Fernández Lorenzo, 2005).

4.3.1. *Plantilla 3D.* Patrón pensado en la captura de distintos puntos con diferentes alturas a partir de una sola imagen. Una ventaja en comparación con un patrón plano, ya que solo se necesita de una captura de imagen para obtener los suficientes puntos para llevar a cabo el proceso de calibración. En la Figura 44 se muestra una plantilla 3D como patrón (Fernández Lorenzo, 2005) (Sierra Álvarez, 2012).

Figura 44. Plantilla 3D.
Fuente: (Sierra Álvarez, 2012).

4.3.2. *Plantilla plana con círculos.* Patrón que consta de una serie de círculos, tal como se muestra en la Figura 45. Su ventaja radica en una mejor detección de los círculos independientemente de la perspectiva; así mismo, también es posible usar un modelo basado en luminancia, de tal modo que se pueda obtener el centro de los círculos con muy buena precisión.

Este tipo de patrones, damero de ajedrez o plantilla con círculos, resultan de gran utilidad en los algoritmos multi-imagen en donde la precisión de la calibración no guarda relación con la precisión en la elaboración de los patrones, sino en la detección exacta de un mismo punto en todas las capturas que forman parte de la secuencia. Cabe mencionar que la captura de imagen del patrón en distintas posiciones y ángulos permite considerar el factor de profundidad (Fernández Lorenzo, 2005) (Bustamante Mejia & López Varona, 2014).

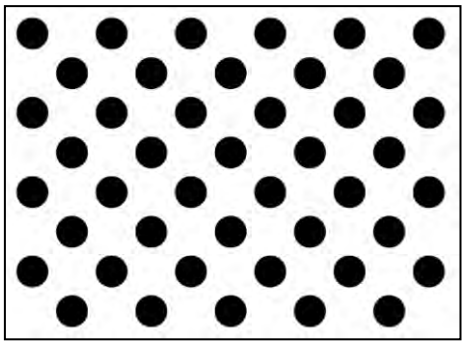

Figura 45. Patrón plano con círculos.
Fuente: (Bustamante Mejia & López Varona, 2014).

4.3.3. ***Plantilla esférica.*** Patrón pensado para no solo llevar a cabo el proceso de calibración a una sola cámara, sino a un conjunto de ellas mediante una esfera que puede ser capturada en simultáneo debido a su forma tridimensional. Por lo tanto, es un método que permite obtener los datos de calibración de manera rápida, además de agregar nuevas restricciones a los parámetros de calibración en vista al conjunto de cámaras con las que se cuenta (Fernández Lorenzo, 2005).

Detalladas las posibles metodologías para llevar a cabo el proceso de calibración, se opta por el método de Zhang basado en un patrón de damero de ajedrez. El procedimiento llevado a cabo es presentado en el Capítulo 5.

5. Visión estereoscópica

La visión estereoscópica básicamente es un proceso que permite obtener la tercera dimensión de los objetos, que pertenecen a un entorno, a través de la distancia comprendida entre los mismos y un sistema de referencia. Parte del modelo estereoscópico biológico, mostrado en la Figura 46(a), en donde el distanciamiento relativo de los ojos permite determinar la profundidad de los objetos o tercera dimensión a través de un proceso de triangulación, el cual emplea dos imágenes generadas por el mismo objeto del entorno en cada ojo. Esto es posible debido a la separación que existe entre ellos, haciendo que las imágenes en cada ojo se muestren desplazadas según la distancia entre los mismos y el objeto (Guerrero Hernández & Pajares Martinsanz, 2011).

Si la imagen de un par de objetos vista por ambos ojos fuera superpuesta en una sola imagen, se obtiene una imagen como la mostrada en la Figura 46(b), puesto que en el entorno la estrella

está más cercana a los ojos que el triángulo. La separación relativa que existe entre los objetos en la imagen vista por cada ojo es denominada disparidad (Guerrero Hernández & Pajares Martinsanz, 2011).

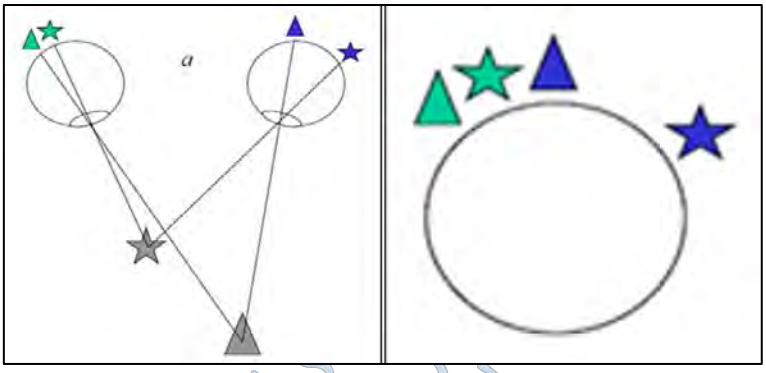

Figura 46. Modelo estereoscópico biológico (a). Superposición de las imágenes vistas por cada ojo (b).
Fuente: (Guerrero Hernández & Pajares Martinsanz, 2011).

Para poder trasladar el modelo biológico a un modelo artificial, se implementa un sistema compuesto por dos cámaras separadas entre sí, de tal modo que se puedan capturar las respectivas imágenes del par estéreo. El proceso está basado en la captura de dos imágenes, que pertenecen al mismo entorno, teniendo en cuenta que cada imagen es capturada dependiendo de dónde se encuentre posicionada la cámara; en consecuencia, las imágenes capturadas también presentan un distanciamiento entre sí, principio básico de la visión estereoscópica que hace posible la obtención de la tercera dimensión del objeto.

5.1. Geometría canónica. Es la geometría más empleada en los sistemas de visión estereoscópica artificial, la cual contempla a dos cámaras cuyos ejes ópticos (Z_I y Z_D) son paralelos y están separados a una distancia conocida como línea base. En la Figura 47, la línea base es señalada como **b**; así mismo, los centros de proyección de las dos cámaras (O_I y O_D) tienen sus ejes ópticos perpendiculares y líneas de exploración o epipolares paralelas a **b**. Las líneas epipolares se definen como las líneas que unen un mismo punto, que pertenece al entorno, en las imágenes capturadas por ambas cámaras (Guerrero Hernández & Pajares Martinsanz, 2011).

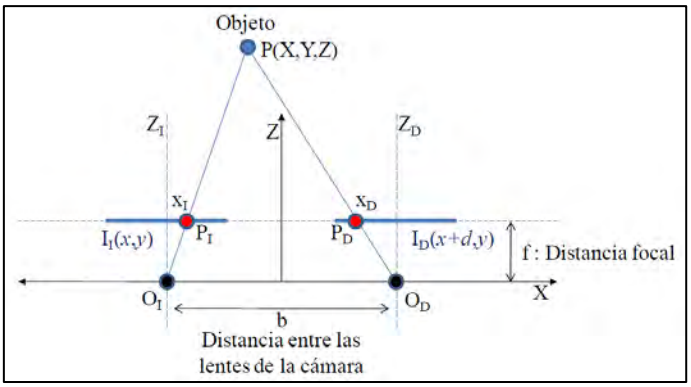

Figura 47. Geometría de un sistema de visión estereoscópica diseñado con sus ejes ópticos paralelos, desde una vista superior. Fuente: (Guerrero Hernández & Pajares Martinsanz, 2011).

Otro detalle importante es que bajo esta geometría, con ejes ópticos paralelos, los centros ópticos solo se ven desplazados horizontalmente; es decir, un mismo punto capturado en ambas imágenes solo se ve afectado en su componente horizontal, tal como se muestra en la Figura 46 con $I_I(x,y)$ y $I_D(x + d,y)$, siendo I_I y I_D los planos imagen o focales. En la Figura 48, se muestra un par de imágenes capturadas desde un par estéreo de cámaras que se encuentran en un mismo eje horizontal, de tal modo que solo presentan un desplazamiento en el sentido horizontal y no en el vertical. Esta captura de imágenes se puede dar por dos procedimientos (Guerrero Hernández & Pajares Martinsanz, 2011):

- Capturando las imágenes a partir de dos cámaras desplazadas una con respecto a la otra que, a la vez, se encuentren alineadas en un mismo eje.

- Capturando las imágenes con una sola cámara que se desplaza tomando la captura de imagen de un punto a otro.

Figura 48. Imágenes capturadas desde un par estéreo de cámaras. Fuente: (Guerrero Hernández & Pajares Martinsanz, 2011).

En ambos procedimientos, el diseño de la geometría puede contemplar que las cámaras tengan sus ejes ópticos paralelos o convergentes; sin embargo, el modelo más empleado en las aplicaciones de visión artificial es el comprendido por un sistema de cámaras cuyos ejes ópticos son paralelos.

5.1.1. ***Método de reconstrucción.*** Bajo esta geometría, los pasos a seguir para reconstruir puntos que pertenecen a dos vistas son los siguientes (Barranco Gutiérrez, Martínez Díaz, & Gómez Torres, 2018):

Siendo **M** el producto de la matriz de parámetros intrínsecos y extrínsecos, mencionado en la ecuación (3.26), de la cámara izquierda y **M′** el mismo producto de parámetros pero de la cámara derecha, es posible conocer la posición de un punto **P = (X,Y,Z,1)** que se encuentra en el entorno, cuyas coordenadas proyectadas en el plano imagen de cada cámara son **p = (u,v)** y **p′ = (u′,v′)**, a partir del siguiente procedimiento:

$$M=\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}$$
$$M'=\begin{bmatrix} m_{11}' & m_{12}' & m_{13}' & m_{14}' \\ m_{21}' & m_{22}' & m_{23}' & m_{24}' \\ m_{31}' & m_{32}' & m_{33}' & m_{34}' \end{bmatrix} \tag{3.51}$$

Conocidas las matrices **M** y **M′**, multiplicamos a las mismas por el punto **P**, de tal modo que se obtienen las siguientes ecuaciones:

$$u = \frac{m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}}$$
$$v = \frac{m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} \tag{3.52}$$

$$u' = \frac{m_{11}'X_i + m_{12}'Y_i + m_{13}'Z_i + m_{14}'}{m_{31}'X_i + m_{32}'Y_i + m_{33}'Z_i + m_{34}'}$$
$$v' = \frac{m_{21}'X_i + m_{22}'Y_i + m_{23}'Z_i + m_{24}'}{m_{31}'X_i + m_{32}'Y_i + m_{33}'Z_i + m_{34}'} \tag{3.53}$$

Las ecuaciones (3.52) y (3.53) se puede agrupar y reordenar en su forma matricial como:

$$\begin{aligned}(um_{31} - m_{11})X + (um_{32} - m_{12})Y + (um_{33} - m_{13})Z &= (m_{14} - um_{34})\\ (vm_{31} - m_{21})X + (vm_{32} - m_{22})Y + (vm_{33} - m_{23})Z &= (m_{24} - um_{34})\\ (u'm_{31}' - m_{11}')X + (u'm_{32}' - m_{12}')Y + (u'm_{33}' - m_{13}')Z &= (m_{14}' - u'm_{34}')\\ (v'm_{31}' - m_{21}')X + (v'm_{32}' - m_{22}')Y + (v'm_{33}' - m_{23}')Z &= (m_{24}' - v'm_{34}')\end{aligned} \tag{3.54}$$

Como resultado, se obtiene un sistema de ecuaciones de la forma $\mathbf{Ax = b}$ que puede ser resuelto multiplicando ambos miembros de la ecuación por $\mathbf{A^T}$:

$$A^TAx = A^Tb \tag{3.55}$$

Finalmente, a partir de (3.55), el vector solución **x** puede ser despejado, de tal modo que se obtienen las coordenadas de un punto en el entorno con la siguiente expresión:

$$x = (A^TA)^{-1}A^Tb \tag{3.56}$$

5.2. Geometría convergente. A pesar de que la geometría canónica es la más empleada en los sistemas de visión estereoscópica, existe la posibilidad de que los ejes ópticos no se encuentren paralelos, sino que se entrecrucen entre sí, tal como se muestra en la Figura 49. Esta geometría en particular se denomina geometría convergente. A continuación, se detallan algunas de sus nociones básicas (Sánchez Pérez).

5.2.1. *Epipolo.* Punto que resulta de la intersección de la línea que une a los dos centros de cada cámara con el plano imagen o focal. En la Figura 50 son denotados como **e** y **e′**.

5.2.2. *Plano epipolar.* Es el plano que contiene a la línea base, de tal modo que existe una familia de planos epipolares conocido como abanico epipolar, tal como se muestra en la Figura 51 (Prieto, 2014).

5.2.3. *Línea epipolar.* Es la intersección de un plano epipolar con el plano imagen o focal de la cámara. En la Figura 52 son denotados como **l** y **l′**.

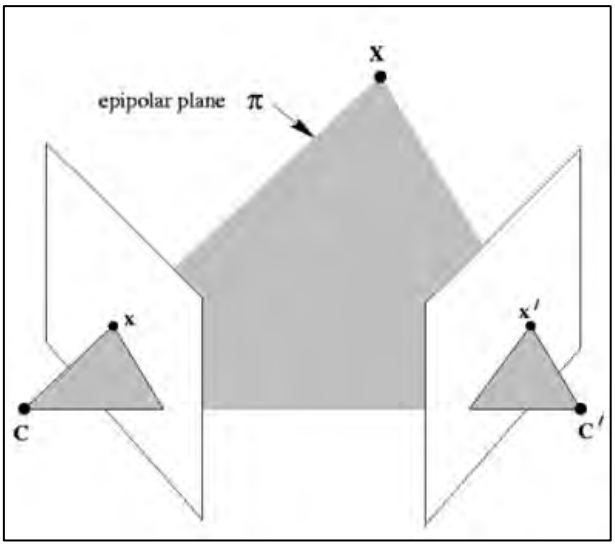

Figura 49. Geometría de un sistema de visión estereoscópica diseñado con sus ejes ópticos convergentes.
Fuente: (Sánchez Pérez).

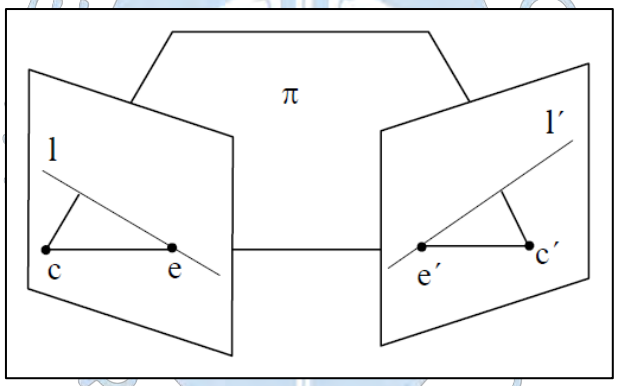

Figura 50. Epipolo.
Fuente: (Sánchez Pérez).

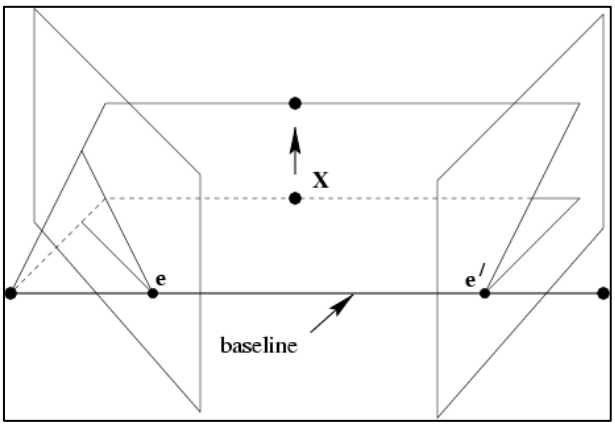

Figura 51. Abanico epipolar.
Fuente: (Prieto, 2014).

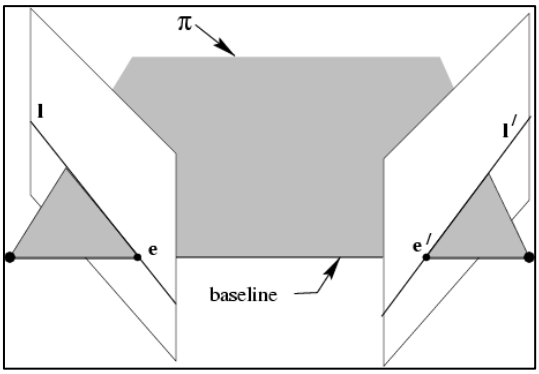

Figura 52. Líneas epipolares.
Fuente: (Prieto, 2014).

5.2.4. ***Matriz fundamental.*** Es la representación algébrica de la geometría epipolar, la cual relaciona cada punto de una imagen con una línea epipolar existente en la otra; por lo tanto, a cada punto **x** le corresponde una línea epipolar $\mathbf{l}'$ existente en la otra imagen, tal que cada punto **x′**, correspondiente al punto **x**, debe caer en $\mathbf{l}'$. Esta relación proyectiva de puntos a líneas es representada por la matriz fundamental **F**, de dimensiones 3x3. (Prieto, 2014) (Sánchez Pérez).

5.2.4.1. Deducción geométrica. Para definir la relación de un punto en una imagen con su correspondiente línea epipolar en la otra, tal como se muestra en la Figura 53, se pueden seguir estos dos pasos (Prieto, 2014):

- Se relaciona al punto **x** con algún punto **x′** que debe caer en la línea epipolar $\mathbf{l}'$.
- Se obtiene la línea epipolar $\mathbf{l}'$ como una línea que une al punto **x′** al epipolo **e′**.

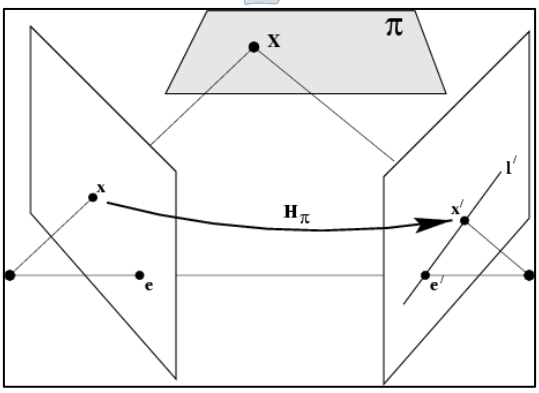

Figura 53. Deducción geométrica de la matriz fundamental.
Fuente: (Prieto, 2014).

Donde $\mathbf{H}_\pi$ es la relación de transferencia de una imagen a otra a través de un plano $\boldsymbol{\pi}$, de tal modo que se cumple con las siguientes relaciones:

$$x' = H_\pi x \qquad l' = e'x' \tag{3.57}$$

De (3.57) se puede obtener la siguiente relación:

$$l' = e'x' = [e']H_\pi x = Fx \tag{3.58}$$

Donde **F** es la matriz fundamental, la cual puede ser escrita como:

$$F = [e']H_\pi \tag{3.59}$$

5.2.4.2. Deducción algebraica. La matriz fundamental en función de las matrices de proyección **P** y $\mathbf{P}'$ de dos cámaras, tal como se muestra en la Figura 54, puede ser obtenida de la siguiente manera (Prieto, 2014):

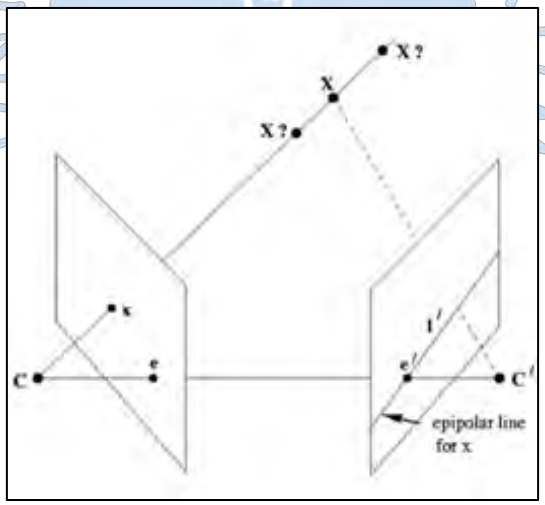

Figura 54. Deducción algebraica de la matriz fundamental.
Fuente: (Prieto, 2014)

Se re-proyecta al punto **x** por **P**, de tal modo que:

$$PX = x \tag{3.60}$$

Con lo cual surge una familia de soluciones de un parámetro:

$$X(\lambda) = P^{+} + \lambda C \tag{3.61}$$

Donde $\mathbf{P}^{+}$ es la seudoinversa de **P**; es decir, $\mathbf{P}^{+}\mathbf{P} = \mathbf{I}$; **C** es el centro de la cámara, tal que $\mathbf{PC} = \mathbf{0}$; y $\boldsymbol{\lambda}$ es un parámetro escalar. Así mismo, la línea epipolar se define como:

$$I' = (P'C)(P'P^{+})x \tag{3.62}$$

Donde $\mathbf{P'C}$ es el epipolo en la segunda imagen, de tal modo que la matriz fundamental puede ser hallada con la siguiente expresión:

$$I' = [e'](P'P^{+})x = Fx \tag{3.63}$$

Donde $\mathbf{F} = [\mathbf{e'}](P'P^{+})$

5.2.4.3. Condición de correspondencia. Para un par de puntos que tienen una correspondencia en dos imágenes ($x \leftrightarrow x'$), la matriz fundamental cumple con lo siguiente (Sánchez Pérez):

$$x'^{T}Fx = 0 \tag{3.64}$$

Por lo tanto, es posible calcular la matriz **F** a partir de puntos con correspondencia entre las imágenes.

5.2.4.4. Matriz esencial. La matriz esencial es el caso particular en el que las coordenadas de la matriz fundamental están normalizadas, de tal modo que una matriz de cámara se puede descomponer como $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$, donde **K** es igual a identidad.

Si se conoce a la matriz de calibración, los puntos proyectados se pueden representar como:

$$\hat{x} = K^{-1}x \tag{3.65}$$

Donde $\mathbf{x} = \mathbf{PX} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}$, de tal modo que:

$$\hat{x} = [R|t]X \tag{3.66}$$

Si se considera el par de matrices $\mathbf{P = [I|0]}$ y $\mathbf{P'} = [\mathbf{R}|\mathbf{t}]$, la matriz fundamental que se obtiene es la denominada esencial, la cual tiene la siguiente forma:

$$E = [t]R \tag{3.67}$$

Y que cumple con lo siguiente:

$$\hat{x}'^{T} E \hat{x} = 0 \tag{3.68}$$

Si se reemplaza en (3.68) los puntos originales, se obtiene:

$$\hat{x}'^{T} (K'^{T} E K^{-1}) x = 0 \tag{3.69}$$

Finalmente, la relación con la matriz fundamental es la siguiente:

$$E = K'^{T} F K \tag{3.70}$$

5.2.5. ***Método de reconstrucción.*** Bajo esta geometría, los pasos a seguir para reconstruir puntos que pertenecen a dos vistas, son los siguientes (Prieto, 2014) (Sánchez Pérez):

Se calcula la matriz fundamental de los puntos correspondientes:

$$\hat{x}'^{T} F x = 0 \tag{3.71}$$

Se calculan las matrices de las cámaras a partir de la fundamental:

$$P = [I|t] \qquad P' = \left[[e']F|e'\right] \tag{3.72}$$

Finalmente, para cada punto correspondiente ($x_i \leftrightarrow x_i'$), se calculan las coordenadas de un punto en el entorno con las siguientes expresiones:

$$P = [I|t] \qquad P' = \left[[e']F|e'\right] \tag{3.73}$$

5.3. Estimación de las coordenadas de un punto mediante análisis vectorial. Si bien en los apartados **5.1.1** y **5.2.5** del presente capítulo se mencionan las metodologías de reconstrucción de puntos según la geometría empleada; para la presente tesis, se propone una metodología basada en la intersección de vectores, la cual se detalla a continuación partiendo del bosquejo que se presenta en la Figura 55.

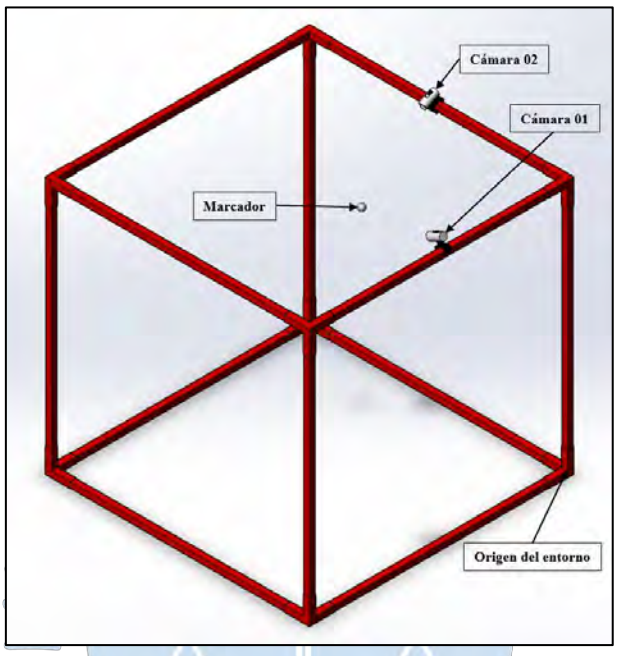

Figura 55. Esquema para la obtención de las coordenadas de un punto en el entorno mediante análisis vectorial.
Fuente: Elaboración propia.

En primer lugar, se generan los vectores unitarios $\bar{\mathbf{u}}$ y $\bar{\mathbf{v}}$ a partir de (3.14), pero esta vez tomando en cuenta el *offset* del punto principal, puesto que el origen de coordenadas del plano imagen no está necesariamente en el punto principal, obteniéndose así las siguientes expresiones:

$$\frac{x}{z} = \frac{u - c_x}{f} \qquad \frac{y}{z} = \frac{v - c_y}{f} \tag{3.74}$$

De (3.74) se pueden despejar las coordenadas del marcador en el entorno, de tal modo que se obtiene lo siguiente:

$$x = \left(\frac{u - c_x}{f}\right) z \qquad y = \left(\frac{v - c_y}{f}\right) z \qquad z = z \tag{3.75}$$

Asumiendo que **z = 1**, las coordenadas del marcador normalizadas son las siguientes:

$$K_{xc1} = \left(\frac{u - c_x}{f}\right) \qquad K_{yc1} = \left(\frac{v - c_y}{f}\right) \qquad K_{zc1} = 1$$
$$K_{xc2} = \left(\frac{up - c_x}{f}\right) \qquad K_{yc2} = \left(\frac{vp - c_y}{f}\right) \qquad K_{zc2} = 1 \tag{3.76}$$

Donde $\overline{\mathbf{K_{c1}}} = (K_{xc1}, K_{yc1}, K_{zc1})$ es el vector posición normalizado con respecto a la cámara 1; $\overline{\mathbf{K_{c2}}} = (K_{xc2}, K_{yc2}, K_{zc2})$ con respecto a la cámara 2; y **u**, **v**, **up** y **vp** son las coordenadas en píxeles del marcador en el plano imagen de la cámara 1 y 2 respectivamente. Para continuar, es necesario rotar a los vectores normalizados con el fin de alinear los ejes del plano imagen de cada cámara con los ejes del entorno, de tal modo que los ejes ahora se encuentran tal como se muestra en la Figura 56.

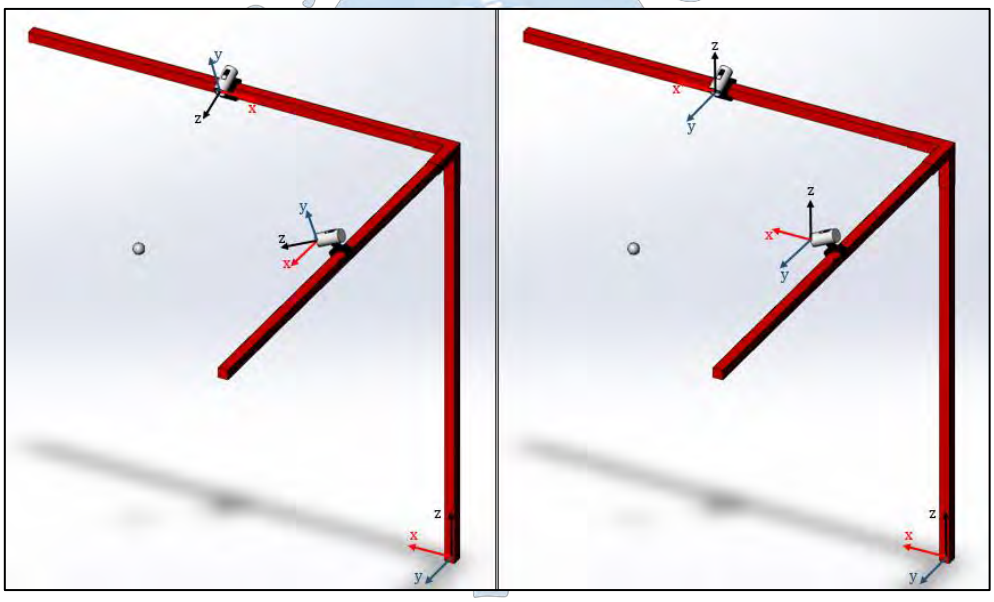

Figura 56. Alineamiento de los ejes del plano imagen de cada cámara con los ejes del entorno.
Fuente: Elaboración propia.

Para ello, se multiplica a los vectores $\overline{\mathbf{K_{c1}}}$ y $\overline{\mathbf{K_{c2}}}$ por las matrices de rotación correspondiente a cada cámara, tal como se muestra a continuación:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{21} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} K_{xc1} \\ K_{yc1} \\ K_{zc1} \end{bmatrix} = \begin{bmatrix} K_{xrc1} \\ K_{yrc1} \\ K_{zrc1} \end{bmatrix} \qquad \begin{bmatrix} r_{11}' & r_{12}' & r_{13}' \\ r_{21}' & r_{22}' & r_{21}' \\ r_{31}' & r_{32}' & r_{33}' \end{bmatrix} \begin{bmatrix} K_{xc2} \\ K_{yc2} \\ K_{zc2} \end{bmatrix} = \begin{bmatrix} K_{xrc2} \\ K_{yrc2} \\ K_{zrc2} \end{bmatrix} \tag{3.77}$$

Donde $\overline{\mathbf{K_{rc1}}} = (K_{xrc1}, K_{yrc1}, K_{zrc1})$ es el vector posición normalizado y rotado con respecto a la cámara 1, y $\overline{\mathbf{K_{rc2}}} = (K_{xrc2}, K_{yrc2}, K_{zrc2})$ con respecto a la cámara 2. Con las coordenadas normalizadas rotadas, es posible obtener un vector unitario cuyo origen está en el punto principal de la cámara y es paralelo al vector que une al centro de la cámara con el marcador. Siendo $\bar{\mathbf{u}}$ el vector unitario con origen en el punto principal de la cámara 1 y $\bar{\mathbf{v}}$ el vector unitario con origen en el punto principal de la cámara 2, los cuales se muestran en la Figura 57.

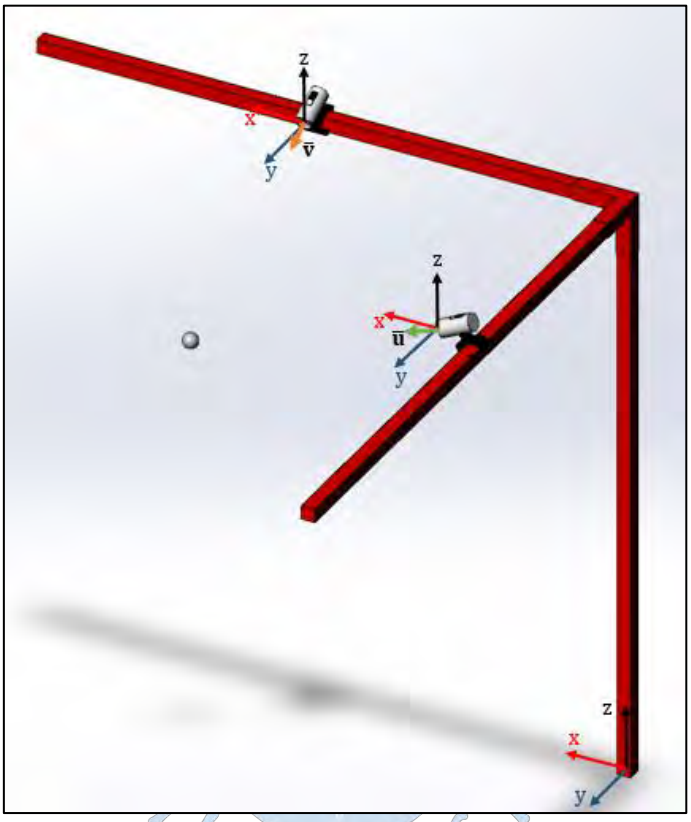

Figura 57. Vectores unitarios con origen en el punto principal de cada cámara.
Fuente: Elaboración propia.

Donde $\bar{\mathbf{u}}$ y $\bar{\mathbf{v}}$ se obtienen con las siguientes expresiones:

$$\bar{\mathbf{u}} = \left(\frac{K_{xrc1}}{\sqrt{K_{xrc1}^2 + K_{yrc1}^2 + K_{zrc1}^2}}, \frac{K_{yrc1}}{\sqrt{K_{xrc1}^2 + K_{ryc1}^2 + K_{zrc1}^2}}, \frac{K_{zrc1}}{\sqrt{K_{xrc1}^2 + K_{yrc1}^2 + K_{zrc1}^2}} \right)$$

$$\bar{\mathbf{v}} = \left(\frac{K_{xrc2}}{\sqrt{K_{xrc2}^2 + K_{yrc2}^2 + K_{zrc2}^2}}, \frac{K_{yrc2}}{\sqrt{K_{xrc2}^2 + K_{yrc2}^2 + K_{zrc2}^2}}, \frac{K_{zrc2}}{\sqrt{K_{xrc2}^2 + K_{yrc2}^2 + K_{zrc2}^2}} \right) \quad (3.78)$$

Luego de obtener los vectores unitarios, se crean dos vectores que representan al vector que une el punto principal de cada cámara con el marcador; para ello, los vectores unitarios son multiplicados con un escalar, distinto para cada vector, de tal modo que los vectores resultantes son los que muestran en la Figura 58.

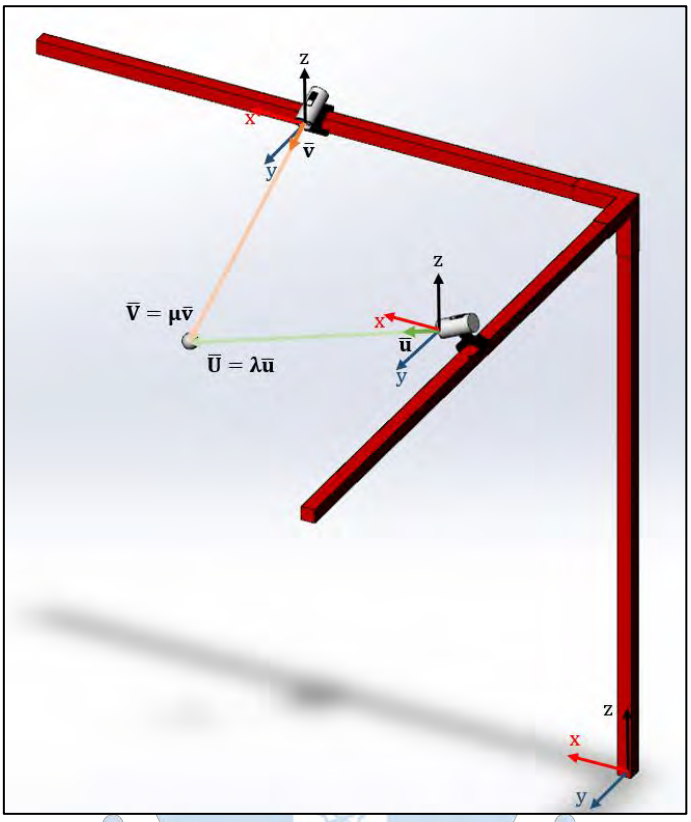

Figura 58. Vectores que unen el punto principal de cada cámara con el marcador.
Fuente: Elaboración propia.

Donde **λ** y **μ** son los escalares que multiplican a $\bar{\mathbf{u}}$ y $\bar{\mathbf{v}}$.

Ahora, se busca un vector cuya intersección con los vectores $\bar{\mathbf{U}}$ y $\bar{\mathbf{V}}$ permita hallar las coordenadas del marcador; en este caso, se hace uso de un vector que resulta del producto vectorial de estos vectores, el cual es perpendicular a ambos y pasa por el centroide del marcador.

En la Figura 59 se muestra a este vector denotado como $\overline{\mathbf{W}}$, el cual por ser resultado de dos vectores escalados, presenta también un valor de escalamiento **ρ**.

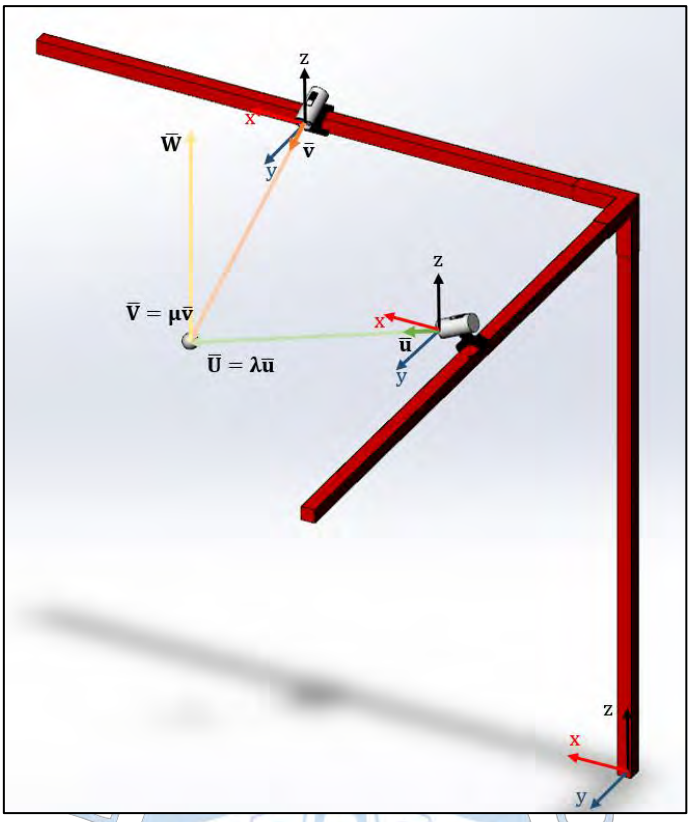

Figura 59. Vector $\bar{W}$, resultado del producto vectorial de $\bar{U}$ y $\bar{V}$.
Fuente: Elaboración propia.

Para obtener finalmente las coordenadas del marcador es necesario hallar el valor de escalamiento **λ** o **μ**; puesto que ambos vectores, luego de ser trasladados con respecto al origen del entorno, representan el vector posición del marcador.

Para ello, se plantea la siguiente ecuación, la cual representa la intersección de los vectores $\bar{U}$, $\bar{V}$ y $\bar{W}$:

$$\begin{aligned} u_x\lambda - v_x\mu - w_x\rho &= t_{2x} - t_{1x} \\ u_y\lambda - v_y\mu - w_y\rho &= t_{2y} - t_{1y} \\ u_z\lambda - v_z\mu - w_z\rho &= t_{2z} - t_{1z} \end{aligned} \tag{3.79}$$

Donde $\bar{T} = \bar{t_2} - \bar{t_1} = (t_{2x},t_{2y},t_{2z}) - (t_{1x},t_{1y},t_{1z})$, el cual se señala en la Figura 60.

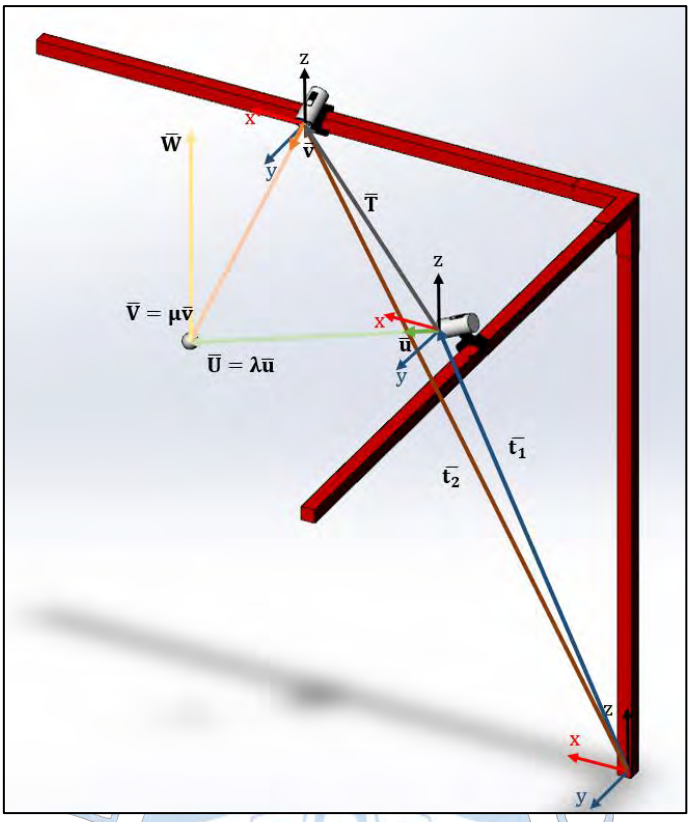

Figura 60. Vectores que intervienen en la obtención del vector posición del marcador en el entorno.
Fuente: Elaboración propia.

De (3.79) se obtienen los valores de escalamiento **λ** y **μ**, por lo tanto, existen dos posibles expresiones que permiten obtener las coordenadas del marcador, las cuales son:

$$\begin{aligned} S_x &= u_x\lambda + t_{1x} \qquad S_y = u_y\lambda + t_{1y} \qquad S_z = u_z\lambda + t_{1z} \\ R_x &= v_x\mu + t_{2x} \qquad R_y = v_y\mu + t_{2y} \qquad R_z = v_z\mu + t_{2z} \end{aligned} \tag{3.80}$$

De (3.80) finalmente se define que las coordenadas del marcador en el entorno se obtienen a partir de la siguiente expresión, la cual tiene en consideración que las rectas no llegan a intersectarse exactamente en un mismo punto, sino que se entrecruzan. En la Figura 61 se hace alusión a esta consideración.

$$x = \frac{S_x + R_x}{2} \qquad y = \frac{S_y + R_y}{2} \qquad z = \frac{S_z + R_z}{2} \tag{3.81}$$

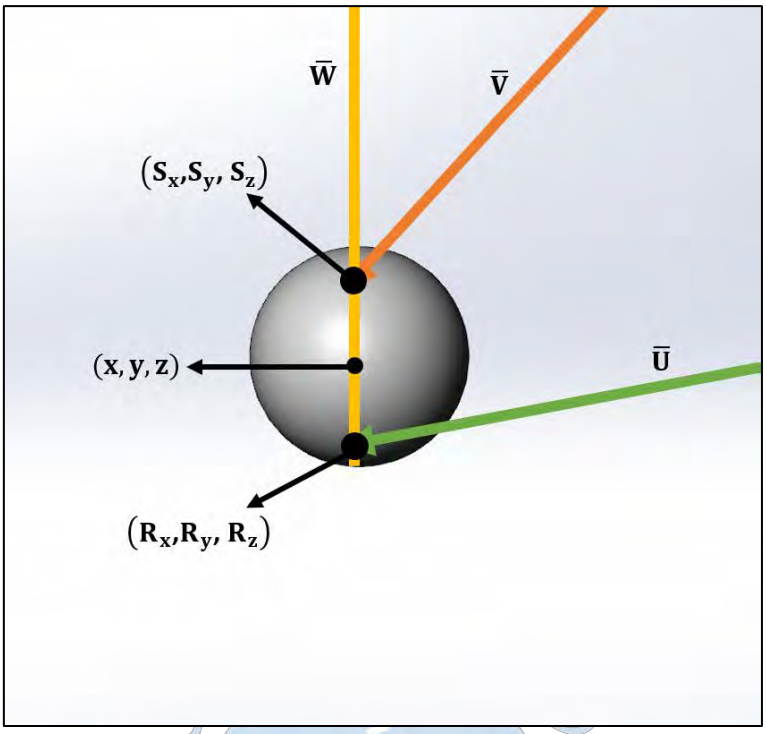

Figura 61. Obtención de las coordenadas del marcador

Fuente: Elaboración propia.

A partir de esta metodología y en complemento con el sistema de detección y seguimiento, detallado en el Capítulo 2, se cuenta con el sistema de posicionamiento *indoor* completo. Los resultados del sistema tras emplear la metodología propuesta y obtención de los parámetros extrínsecos e intrínsecos de las cámaras son llevados a cabo y mostrados en el Capítulo 5.

Capítulo 4

Integración del sistema de posicionamiento *indoor* en ROS

1. Introducción

Descritos en los capítulos 2 y 3 los subsistemas que componen finalmente al sistema de posicionamiento *indoor*, se describe en el presente capítulo el proceso de integración de los mismos, y de un sistema adicional que permite reconstruir la trayectoria recorrida por el marcador en una interfaz gráfica. A continuación, se detalla al metasistema operativo[18] empleado para este proceso, al igual que el procedimiento que se lleva a cabo.

2. *Robot Operating System* (ROS)

ROS es un *framework* o entorno de trabajo flexible desarrollado originalmente en el 2007 con el nombre de *Switchyard* por el Laboratorio de Inteligencia de Stanford como parte del proyecto STAIR[19]. Su desarrollo principal ocurre en *Willow Garage*[20] para luego transferir su administración a *Open Source Robotic Foundation* (OSFR)[21] (Cuevas Castañeda, 2016).

ROS está orientado a la escritura de software para robots que simplifica la tarea de crear un comportamiento complejo y robusto en múltiples aplicaciones robóticas a partir de una colección de herramientas, librerías y convenciones. Así mismo, fue creado con el fin de incentivar el trabajo colaborativo, de tal modo que es posible desarrollar sistemas independientes que lleguen a ser subsistemas de un sistema mucho más complejo. Por ejemplo, un sistema de mapeo *indoor*, un sistema de navegación a partir de un mapeo previo, y un sistema de visión artificial que reconozca objetos, se pueden integrar para formar un sistema más complejo (Open Source Robotics Foundation, s.f.).

2.1. Conceptos ROS. ROS cuenta con tres niveles de conceptos: nivel de sistema de archivos, nivel de gráfico computacional y nivel de comunidad.

---

[18] Necesita ser instalado sobre otro sistema operativo.
[19] Robot que puede navegar en entornos domésticos y de oficina, recoger e interactuar con objetos y herramientas, y conversar de manera inteligente, de tal modo que pueda brindar ayuda a las personas.
[20] Fue una incubadora de empresas y laboratorio de investigación robótica que cerró sus puertas en junio del 2014.
[21] Fundación que trabaja de la mano con la industria, la academia y el gobierno para crear y dar soporte a software de código abierto y hardware para uso en robótica orientado a la investigación, educación y desarrollo de productos.

2.1.1. ***Nivel de sistema de archivos ROS.*** En este nivel se encuentran principalmente los recursos de ROS. Los mismos de detallan a continuación (Open Source Robotics Foundation, 2014):

2.1.1.1. *Paquetes.* Los paquetes son la unidad más básica en el que puede estar organizado software en ROS; es decir, es la estructura más granular[22] que se puede construir y lanzar. Puede contener procesos ejecutables (nodos), librerías dependientes de ROS, conjuntos de datos, archivos de configuración, o cualquier otro elemento que sea útil en conjunto.

2.1.1.2. *Meta-paquetes.* Los meta-paquetes son paquetes especiales que solo sirven para representar a un conjunto de paquetes que están relacionados.

2.1.1.3. *Manifiesto de paquetes.* Los manifiestos (*package.xml*) proporcionan metadatos[23] sobre un paquete; esto es, información que incluye el nombre, versión, descripción, información de la licencia, dependencias; u otra información relevante sobre el paquete.

2.1.1.4. *Repositorios.* Los repositorios son una colección de paquetes que comparten en común un sistema VCS[24]. Los paquetes que comparten este sistema son de la misma versión, permitiéndoles ser lanzados juntos haciendo uso de la herramienta de lanzamiento automático *catkin bloom*. Es posible que los repositorios cuenten con un solo paquete.

2.1.1.5. *Archivos msg.* Definen la estructura de los datos para los mensajes que se envían en ROS. Esta información se almacena en: *mi_paquete/msg/tipo_de_mensaje.msg.*

2.1.1.6. *Archivos srv.* Definen la estructura de los datos de solicitud y respuesta para los servicios en ROS. Esta información se almacena en: *mi_paquete/srv/tipo_de_servicio.srv.*

[22] Vinculación de información simple que al vincularse con otras, están dotadas de un valor implícito distinto y de mayor nivel conceptual.
[23] Información sobre las propiedades de una cierta cantidad de datos, al igual que información sobre la estructura de los mismos.
[24] Del inglés *Version Control System* (Sistema de control de versiones), se define como un sistema que permite gestionar las modificaciones de un proyecto de software.

2.1.2. ***Nivel de gráfico computacional ROS.*** Se define como gráfico computacional a la red *peer-to-peer*[25] de procesos ROS, la cual procesa datos en paralelo. Los conceptos básicos a este nivel son los siguientes (Open Source Robotics Foundation, 2014):

2.1.2.1. *Nodos.* Los nodos se definen como una serie de procesos que llevan a cabo algún tipo de cálculo en la red del sistema. ROS está diseñado para ser modular a una escala de nodos, de tal modo que el sistema de control de un robot está compuesto por muchos de ellos. Por ejemplo, un sistema robótico puede estar compuesto por un nodo que controla un buscador láser, un nodo encargado del control en los motores de las ruedas, un nodo que realiza la localización, un nodo que planifica la ruta, etc. Para escribir un nodo ROS se hace uso de la biblioteca de cliente ROS, tales como *roscpp* o *rospy*.

2.1.2.2. *Maestro.* El maestro ROS provee registro de los nombres y búsqueda del resto de conceptos que se encuentran en este nivel. Sin el maestro los nodos no son capaces de comunicarse entre sí; es decir, de intercambiar mensajes o invocar servicios.

2.1.2.3. *Servidor de parámetros.* El servidor de parámetros permite almacenar datos en una ubicación central. En la actualidad forma parte del maestro.

2.1.2.4. *Mensajes.* Los mensajes son una estructura de datos que comprende campos escritos, al igual que el estándar de tipos de datos primitivos, tales como enteros, flotantes, booleanos, etc. Así mismo, pueden incluir estructuras y vectores que se encuentren arbitrariamente anidadas[26], tal como las estructuras que caracterizan al lenguaje C. Cabe recalcar que los mensajes son el medio a través del cual los nodos son capaces de comunicarse.

2.1.2.5. *Tópicos.* Los mensajes provenientes de los nodos, que conforman la red del sistema, se enrutan a través de un sistema de transporte del tipo publicación/suscripción, de tal modo que un nodo envía un mensaje publicando al mismo en un tópico determinado.

[25] Red de ordenadores cuya característica es que todos o algunos aspectos funcionan sin clientes ni servidores fijos; en lugar de ello, funcionan como una serie de nodos que se comportan como iguales entre sí; estos es, que actúan a la vez como clientes y servidores respecto a los otros nodos de la red.
[26] Estructuras que se encuentran dentro de otra estructura.

Dicho esto, un tópico llega a ser el nombre empleado para poder identificar el contenido de un mensaje; por ello, un nodo que tiene interés en un cierto tipo de datos debe suscribirse al tópico indicado.

Pueden existir múltiples nodos publicadores y suscriptores para un solo tópico, al igual que puede haber un solo nodo capaz de publicar y/o suscribirse a múltiples tópicos. Esto es posible debido a que los nodos no tienen conciencia de la existencia de otros nodos; en consecuencia, es posible desacoplar el consumo de información con la producción de la misma.

Se puede entender a un tópico como un bus de mensajes, el cual cuenta con un nombre y está disponible para que cualquier nodo sea capaz de conectarse a él para el envío o recepción de mensajes, siempre y cuando sean del tipo correcto.

2.1.2.6. *Servicios.* El modelo publicación/suscripción llega a ser un paradigma de comunicación muy flexible; sin embargo, su transporte unidireccional de muchos a muchos no es la más apropiada para las interacciones solicitud/respuesta que suelen ser requeridas en un sistema distribuido[27]. Este tipo de interacción se realiza a través de servicios, los cuales están definidos mediante un par de estructuras de mensajes, una para la solicitud y otra para la respuesta, de tal modo que un nodo proveedor ofrece un servicio bajo un nombre y un cliente hace uso del mismo enviando el mensaje de solicitud y esperando una respuesta.

2.1.2.7. *Bags.* Son un formato de almacenamiento y reproducción para datos que se envían en mensajes ROS, llegando a ser una importante herramienta para el almacenamiento de datos. Por ejemplo, el de sensores que pueden presentar dificultad en su recopilación, pero que son necesarios para el desarrollo y validación de algún algoritmo.

En relación al maestro ROS, actúa como un servidor de nombres en este nivel, almacenando información del registro de servicios y de los tópicos para los nodos ROS. Los nodos se comunican con el maestro para hacer un reporte de su información de registro, de tal modo que a medida que se comunican con el maestro pueden recibir información de otros nodos que se encuentran registrados, dándoles la capacidad de hacer conexiones según les convenga. Cuando la información llega a cambiar, el maestro también es capaz de realizar devoluciones de llamada

[27] En computación, un sistema distribuido es una colección de computadores que se encuentran separados físicamente pero las cuales se encuentran conectadas entre sí por una red de comunicaciones.

a estos nodos, permitiéndoles crear conexiones dinámicas a medida que se ejecutan nuevos nodos. En la Figura 62 se muestra un esquema de los conceptos que forman parte de este nivel (Achmad, Priyandoko, & Daud, 2017).

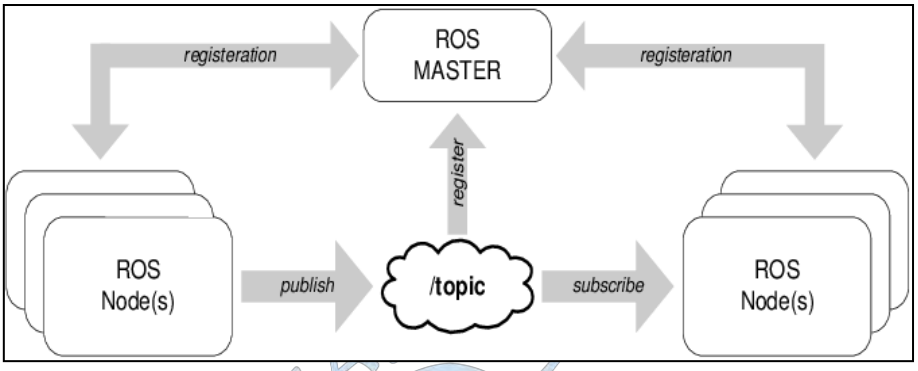

Figura 62. Conceptos ROS a nivel de gráfico computacional.
Fuente: (Achmad, Priyandoko, & Daud, 2017).

2.1.3. ***Nivel de comunidad ROS.*** A este nivel se encuentran una serie de recursos ROS que permiten a distintas comunidades hacer un intercambio de software y conocimientos. Los mismos son detallados a continuación (Open Source Robotics Foundation, 2014):

2.1.3.1. *Distribuciones.* Las distribuciones de ROS son una colección de pilas[28] versionadas que pueden ser instaladas. De manera análoga a las distribuciones de Linux, facilitan la instalación de una colección de software y mantienen versiones coherentes en un conjunto de software.

2.1.3.2. *Repositorios.* ROS está basada en una red federada de repositorios de código en donde es posible que distintas instituciones puedan desarrollar y lanzar sus propios componentes de software robóticos.

2.1.3.3. *El Wiki de ROS.* Se trata del foro principal para documentar información sobre ROS, de tal modo que cualquier persona es capaz de registrarse y contribuir con una documentación propia. Por ejemplo, correcciones o actualizaciones, tutoriales, etc.

[28] Manera en que están organizados los paquetes en ROS.

2.1.3.4. *Lista de correos.* Canal principal de comunicación para estar al tanto de las nuevas actualizaciones de ROS, así como un foro en donde es posible hacer preguntas acerca del software.

2.1.3.5. *Respuestas de ROS.* Sitio web en donde se encuentran las respuestas a una serie de preguntas relacionadas con ROS (https://answers.ros.org/questions/).

2.1.3.6. *Blog.* Blog de ROS que proporciona actualizaciones periódicas del software, así como fotos y videos (http://www.ros.org/news/).

2.2. Comandos en ROS. Las siguientes herramientas están instaladas en *$ROS_ROOT* y deben ser agregadas a la variable PATH como parte del proceso de instalación (Open Source Robotics Foundation, 2015):

2.2.1. *rosbag.* Comando que permite realizar distintas operaciones que incluyen la reproducción, grabación y validación, con archivos *bag*.

2.2.2. *rosbash.* No es una línea de comando, sino un conjunto de comandos y funciones; por ello, se requiere extraer el contenido del archivo *rosbash*. Además, proporciona los comandos *roscd* y *rosed*, al igual que la funcionalidad de autocompletado de comandos para los mencionados, *rosmake* y *rosrun*.

```
source $ROS_ROOT/tools/rosbash/rosbash
```

2.2.3. *roscd.* Comando que permite cambiar de manera directa un directorio actual al directorio de un paquete o pila[29]; para ello, solo es necesario conocer el nombre del paquete o fila sin necesidad de conocer una ruta especifica.

Uso:

```
roscd nombre_paquete[/subdirectorio]
```

[29] Conjunto de paquetes.

Ejemplo:

```
roscd roscpp/include
```

2.2.4.*rosclean.* Comando que limpia recursos, creados por ROS, del sistema de archivos. Por ejemplo, los archivos de registro.

2.2.5.*roscore.* Colección de nodos y programas que son requisito fundamental de un sistema basado en ROS, de tal modo que si se omite su ejecución, los nodos no son capaces de comunicarse entre sí. Al ejecutarse inicia un maestro ROS, un servidor de parámetros ROS y un nodo de registro *rosout.*

Uso:

```
roscore
```

2.2.6.*rosdep.* Comando que permite instalar dependencias del sistema.

Uso:

```
rosdep install NOMBRE_PAQUETE
```

2.2.7.*rosed.* Comando que permite editar de manera directa un archivo que se encuentra dentro de un determinado paquete; para ello, solo es necesario conocer el nombre del paquete sin necesidad de conocer una ruta específica.

Uso:

```
rosed nombre_del_paquete nombre_del_archivo
```

Ejemplo:

```
rosed roscpp ros.h
```

Si el nombre del archivo no se encuentra de manera única dentro del paquete, se despliega un menú con una serie de posibles archivos a editar. Cuando se ejecuta este comando se abre el editor definido en el entorno de variables *$EDITOR* o, por defecto, en Vim[30].

2.2.8. *roscreate-pkg.* Es parte del paquete *roscreate*. Crea los archivos *Manifest, CMakeLists, Doxygen* y otros archivos comunes que son necesarios para la creación de un nuevo paquete en ROS.

Uso:

```
roscreate-pkg nombre_del_paquete
```

Para comprobar que se ha creado el paquete, se puede usar *roscd* de la siguiente manera:

```
roscd nombre_del_paquete
```

También es posible definir las dependencias del paquete a crear:

```
roscreate-pkg nombre_del_paquete dependencia_1 dependencia_2 dependencia_3
```

Ejemplo:

```
roscreate-pkg mi_paquete roslib roscpp std_msgs
```

Para las dependencias de uso común como *roscpp* y *rospy*, *roscreate-pkg* puede crear adicionalmente estructuras de directorio de uso común.

2.2.9. *roscreate-stack.* Comando que crea o actualiza una pila involucrando la creación de un archivo *stack.*xml, siendo la manera más conveniente de actualización automática.

[30] Vim es una versión mejorada del editor de texto vi, el cual está presente en todos los sistemas UNIX.

Uso:

```
roscreate-stack
```

2.2.10. ***rosrun.*** Comando que permite ejecutar un archivo ejecutable de un determinado paquete sin la necesidad de estar en el directorio en donde se encuentra el archivo.

Uso:

```
rosrun nombre_paquete archivo_ejecutable
```

Ejemplo:

```
rosrun roscpp_tutorials talker
```

2.2.11. ***roslaunch.*** Comando que permite lanzar un conjunto de nodos a partir de un archivo de configuración XML.

Uso:

```
roslaunch nombre_paquete archivo.launch
```

2.2.12. ***rosmake.*** Comando que permite construir un paquete, facilitando la construcción de paquetes que cuentan con dependencias.

Ejemplo:

```
rosmake move_base
```

En el ejemplo citado, *rosmake* determina las dependencias del paquete *move_base* y las dependencias de las dependencias, asegurando que todas las dependencias sean construidas antes de construir *move_base.*

2.2.13. ***rosmsg.*** Comando que permite mostrar información sobre un determinado mensaje ROS. Cuenta con las siguientes opciones:

2.2.13.1.*rosmsg show.* Muestra información sobre el tipo de mensaje.

2.2.13.2.*rosmsg list.* Muestra una lista de todos los mensajes.

2.2.13.3.*rosmsg package.* Muestra una lista de todos los mensajes en un paquete.

2.2.13.4.*rosmsg packages.* Muestra una lista de todos los paquetes con mensajes.

2.2.13.5.*rosmsg users.* Muestra los archivos de código que utiliza un determinado mensaje.

2.2.14. ***rosnode.*** Comando que permite mostrar información sobre un determinado nodo. Cuenta con las siguientes opciones:

2.2.14.1.*ronode info.* Muestra información sobre un determinado nodo que incluye sus publicaciones o suscripciones.

2.2.14.2.*ronode kill.* Finaliza algún nodo que se haya colgado.

2.2.14.3.*ronode list.* Muestra una lista de todos los nodos que se encuentran en ejecución.

2.2.14.4.*ronode machine.* Muestra una lista de los nodos que se están ejecutando en una determinada máquina.

2.2.14.5.*ronode ping.* Realiza ping[31] a un determinado nodo de manera repetitiva; es decir, comprueba su conectividad.

2.2.15. ***rospack.*** Comando que permite obtener información sobre un determinado paquete que se encuentra disponible en el sistema de archivos.

[31] Este comando es utilizado para comprobar el estado de la red.

2.2.16. ***rosparam.*** Comando que permite obtener y configurar los valores del servidor de parámetros desde líneas de comando empleando texto codificado YAML[32].

2.2.17. ***rossrv.*** Comando que permite mostrar información sobre un determinado servicio. Cuenta con las mismas opciones que *rosmsg.*

2.2.18. ***rosservice.*** Implementa una serie de comandos que permiten conocer los servicios que están en ejecución, al igual que el nodo de procedencia; así mismo, también permiten conocer información sobre un servicio determinado, tales como el tipo, URI[33] y argumentos.

2.2.19. ***rosstack.*** Comando que permite obtener información sobre una determinada pila que se encuentra disponible en el sistema, llegando a ser equivalente a *rospack.*

2.2.20. ***rostopic.*** Comando que permite mostrar información sobre un determinado tópico. Cuenta con las siguientes opciones:

2.2.20.1.*rostopic bw.* Muestra el ancho de banda[34] utilizado por un determinado tópico.

2.2.20.2.*rostopic echo.* Muestra los mensajes que son publicados en un determinado tópico.

2.2.20.3.*rostopic find.* Permite encontrar un determinado tópico por su tipo.

2.2.20.4.*rostopic hz.* Muestra la tasa de publicación de un tópico. La tasa que se reporta por defecto es la tasa promedio durante todo el tiempo de ejecución de *rostopic*.

2.2.20.5.*rostopic info.* Muestra la información de un determinado tópico.

2.2.20.6.*rostopic list.* Muestra una lista de los tópicos que están en ejecución.

---

[32] Formato que es empleado para guardar objetos de datos con estructura de árbol.
[33] Identificador de recursos uniforme o URI, se define como una cadena de caracteres que identifica los recursos de una red de forma unívoca.
[34] Cantidad de información o de datos que es posible enviar a través de una conexión de red en un periodo de tiempo.

2.2.20.7.*rostopic pub.* Permite publicar datos en un determinado tópico.

2.2.20.8.*rostopic type.* Muestra el tipo de un determinado tópico.

2.2.21. *rosversion.* Comando que reporta la versión de una pila.

2.3. Herramientas gráficas en ROS.

2.3.1. *rqt_bag.* Herramienta gráfica que permite visualizar los datos en archivos *bag*.

2.3.2. *rqt_deps.* Herramienta que permite generar un archivo en formato PDF de las dependencias.

2.3.3. *rqt_graph.* Herramienta que muestra en un gráfico interactivo los nodos y tópicos en estado activo, tal como se muestra en la Figura 63.

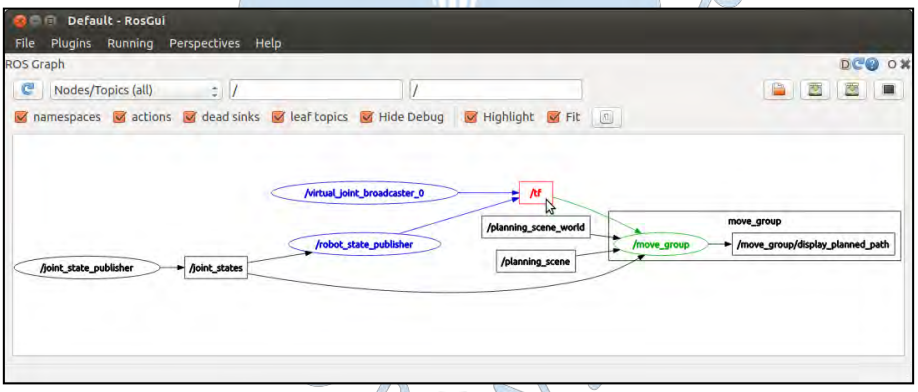

Figura 63. Gráfico interactivo generado con la herramienta *rqt_graph*.
Fuente: (Open Source Robotics Foundation, 2015).

2.3.4. *rqt_plot.* Herramienta que permite graficar los datos numéricos de un tópico en todo el tiempo que está activo.

3. Proceso de integración del sistema de posicionamiento *indoor*

En esta parte del capítulo se detalla el proceso de integración del sistema de posicionamiento en ROS, desde la creación de una carpeta de trabajo hasta la creación de los archivos ejecutables de cada subsistema. Cabe recalcar que lo descrito en adelante supone que ya se tiene instalado ROS. Una guía de instalación se encuentra en: http://wiki.ros.org/indigo/Installation/Ubuntu.

3.1. Creación del espacio de trabajo. Un espacio de trabajo es una carpeta en donde se puede modificar, crear o instalar paquetes *catkin*[35], los cuales pueden contener hasta cuatro espacios distintos, cumpliendo cada uno con una función particular en el desarrollo del software. Para la creación del espacio de trabajo en la presente tesis, se ejecutan en una terminal de Ubuntu los siguientes comandos (Open Source Robotics Foundation, 2019):

```
$ mkdir -p ~/tesis/src
$ cd ~/tesis/
$ catkin_make
```

catkin_make es un comando que permite trabajar con espacios de trabajo *catkin*; una vez ejecutado, crea un archivo *CMakeLists.txt* en la carpeta creada *src*. Así mismo, si se navega dentro del directorio actual, se puede notar la creación de dos carpetas adicionales: *devel* y *build*, en donde la primera contiene una serie de archivos *setup.*sh.* Antes de continuar, se debe obtener el nuevo archivo *setup.*sh.* Si se escriben y ejecutan los comandos mencionados, aparece lo que se muestra en la Figura 64.

```
$ source devel/setup.bash
```

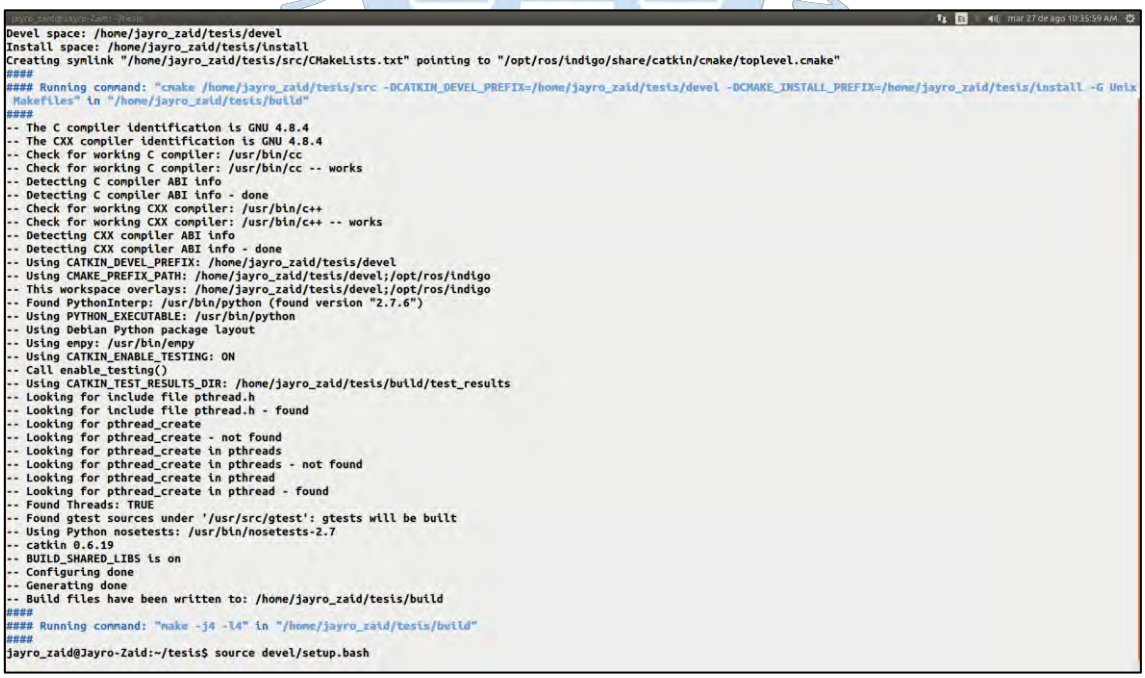

Figura 64. Creación del espacio de trabajo desde una terminal de Ubuntu.

Fuente: Elaboración propia.

[35] Sistema de compilación oficial de *ROS*.

3.2. Creación del paquete *catkin*. Un paquete puede ser considerado como *catkin* si cumple con los requisitos que se mencionan a continuación (Open Source Robotics Foundation, 2019):

- El paquete debe contener un archivo *package.xml* que proporcione meta información sobre el mismo.

- El paquete debe contener un archivo *CMakeLists.txt* que es utilizado por *catkin.*

- Cada paquete debe tener su propia carpeta; es decir, no existen paquetes anidados, ni más de uno que se encuentren en el mismo directorio.

Para la creación del paquete, primero se debe estar en el directorio del espacio de trabajo creado previamente; para ello, se escribe el siguiente comando en una terminal:

```
$ cd ~/tesis/src
```

Una vez en el directorio adecuado, se hace uso del comando *catkin_create_pkg*, el cual requiere de un nombre para el paquete y, de manera opcional, una lista de dependencias, tal como se muestra a continuación:

```
$ catkin_create_pkg nombre_paquete dependencia1 dependencia2 dependencia3
```

Para esta tesis, se crea un paquete con las dependencias *std_msgs*, *rospy* y *roscpp.*

```
$ catkin_create_pkg tesis_jayro_zaid_paiva_mimbela std_msgs rospy roscpp
```

Al ejecutar este comando, se crea una carpeta llamada *tesis_jayro_zaid_paiva_mimbela*, la cual contiene un archivo *package.xml* y un archivo *CMakeLists.txt* que están completados con la información parcial que proporciona *catkin_create_pkg*, requisitos que debe cumplir el paquete para ser considerado como *catkin.*

Ahora, se necesita construir el paquete creado en el espacio de trabajo; para ello, es necesario moverse al directorio del espacio de trabajo y ejecutar el comando *catkin_make*, tal como se muestra a continuación:

```
$ cd ~/tesis
$ catkin_make
```

Para agregar el espacio de trabajo al entorno de ROS, se necesita obtener el archivo de configuración generado; para ello, se ejecuta el siguiente comando:

```
$ . ~/tesis/devel/setup.bash
```

Si se escriben y ejecutan los comandos mencionados, aparece lo que se muestra en la Figura 65.

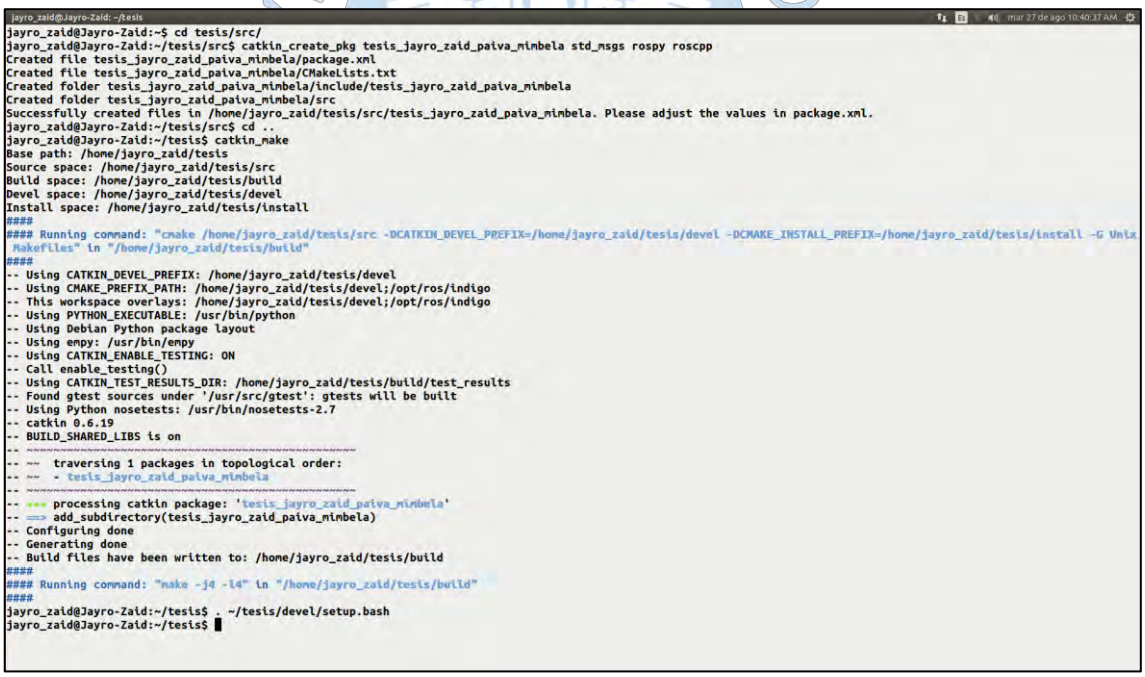

Figura 65. Creación del paquete *catkin* desde una terminal de Ubuntu.

Fuente: Elaboración propia.

3.3. Creación de los mensajes ROS. Los archivos *msg* son archivos de texto simple que describen el tipo de dato permitido para ser almacenado en un determinado mensaje ROS, los cuales se encuentran en la carpeta *msg* de un paquete. En relación a los tipos de mensaje permitidos, se tienen los siguientes:

- *int8, int16, int32, int64*
- *float32, float64*
- *string*
- *time, duration*
- *array*

Como paso común para la creación de los mensajes a ser empleados por los subsistemas que componen el sistema de posicionamiento, se debe crear una carpeta de nombre *msg* dentro del directorio del paquete; para ello, se ejecutan los siguientes comandos:

```
$ roscd tesis_jayro_zaid_paiva_mimbela
$ mkdir msg
```

Creada la carpeta, hace falta asegurar que los archivos *msg* puedan convertirse en código fuente para C++, Python u otros lenguajes de programación; para ello, se abre el archivo *package.xml*, originado al crearse el paquete, y se verifica que no se encuentren comentadas las siguientes líneas:

```
<build_depend> message_generation </build_depend>
<exec_depend> message_runtime </exec_depend>
```

Donde *message_generation* es necesaria para la compilación, mientras que *message_runtime* es necesaria para el tiempo de ejecución. Para la generación de los mensajes, es necesario agregar la dependencia *message_generation* a la función de llamada *find_package*, la cual se encuentra en el archivo *CMakeLists.txt*, originado al crearse el paquete. A continuación, se muestra el bloque de código de la función con los cambios realizados:

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

Del mismo modo, se asegura de estar exportando la dependencia *message_runtime*; para ello, se agrega la dependencia a la función *catkin_package*, la cual también se encuentra en *CMakeLists.txt*. A continuación, se muestra el bloque de código de la función con los cambios realizados:

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

Por último, se asegura de estar llamando a la función *generate_message*(); para ello, se verifica que no se encuentren comentadas las líneas que se muestran a continuación:

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

3.3.1.*Mensaje ROS para el sistema de detección y seguimiento.* Para la creación del mensaje ROS que contiene la información brindada por el nodo del sistema de detección y seguimiento, lo primero a realizar es la búsqueda del siguiente bloque de código en *CMakeLists.txt:*

```
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

Ubicado el bloque de código, se deben remover los signos numerales, de tal modo que dejen de ser comentarios, y reemplazar los archivos *Message1.msg* y *Message2.msg* por los archivos a emplear en la presente tesis. El bloque de código, con los cambios realizados, debe verse como el siguiente, en donde *pixeles.msg* es el archivo creado para esta tesis:

```
add_message_files(
  FILES
  pixeles.msg
)
```

El contenido de *pixeles.msg* es el que se muestra a continuación:

```
float64 u
float64 v
float64 up
float64 vp
```

Ahora, al igual que el caso anterior, se ubica y descomenta el siguiente bloque de código:

```
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )
```

Luego, se deben agregar los paquetes de los que dependen los mensajes a crear, en este caso, *std_msgs*:

```
 generate_messages(
   DEPENDENCIES
   std_msgs
 )
```

Por último, se tiene que volver a construir al paquete; para ello, se escriben y ejecutan los siguientes comandos:

```
$ roscd tesis_jayro_zaid_paiva_mimbela
$ cd ../..
$ catkin_make install
$ cd -
```

Al ser ejecutado, debe aparecer lo que se muestra en la Figura 66. Como se menciona en el apartado **3.3** del presente capítulo, los archivos *msg*, al construirse el paquete, se convierten en un archivo de código fuente; en este caso, se ha creado un archivo con el nombre *pixeles.h*, localizado en la ruta: *~/tesis/devel/include/tesis_jayro_zaid_paiva_mimbela/*, cuyo contenido es el que se muestra en el Apéndice A-1 de la presente tesis.

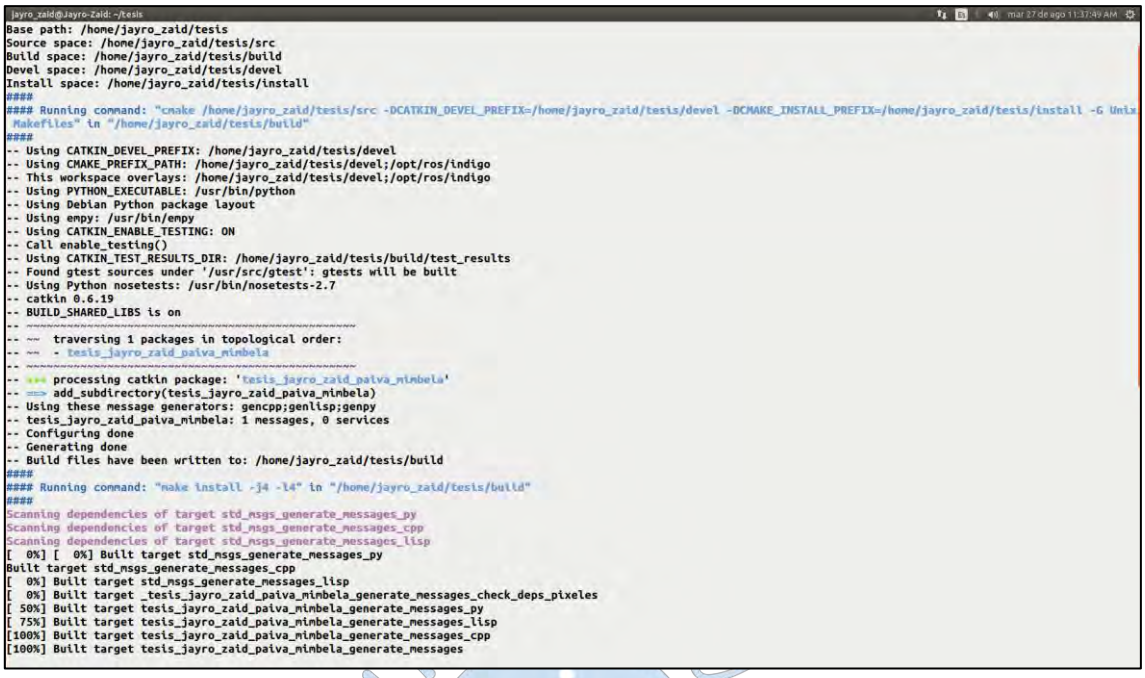

Figura 66. Creación del código fuente *pixeles.h* desde una terminal de Ubuntu.

Fuente: Elaboración propia.

3.3.2. ***Mensaje ROS para el sistema de conversión de espacios 2D a 3D.*** Para la creación del mensaje ROS que contiene la información brindada por el nodo del sistema de conversión de espacios 2D a 3D, se realizan los mismos pasos que se siguieron en el apartado **3.3.1** del presente capítulo, pero con la diferencia que el nuevo archivo *msg* a agregar es *espaciales.msg*, tal como se muestra a continuación:

```
add_message_files(
  FILES
  pixeles.msg
  espaciales.msg
)
```

El contenido de *espaciales.msg* es el que se muestra a continuación:

```
float64 x
float64 y
float64 z
```

Luego de volver a construir al paquete, debe aparecer lo que se muestra en la Figura 67.

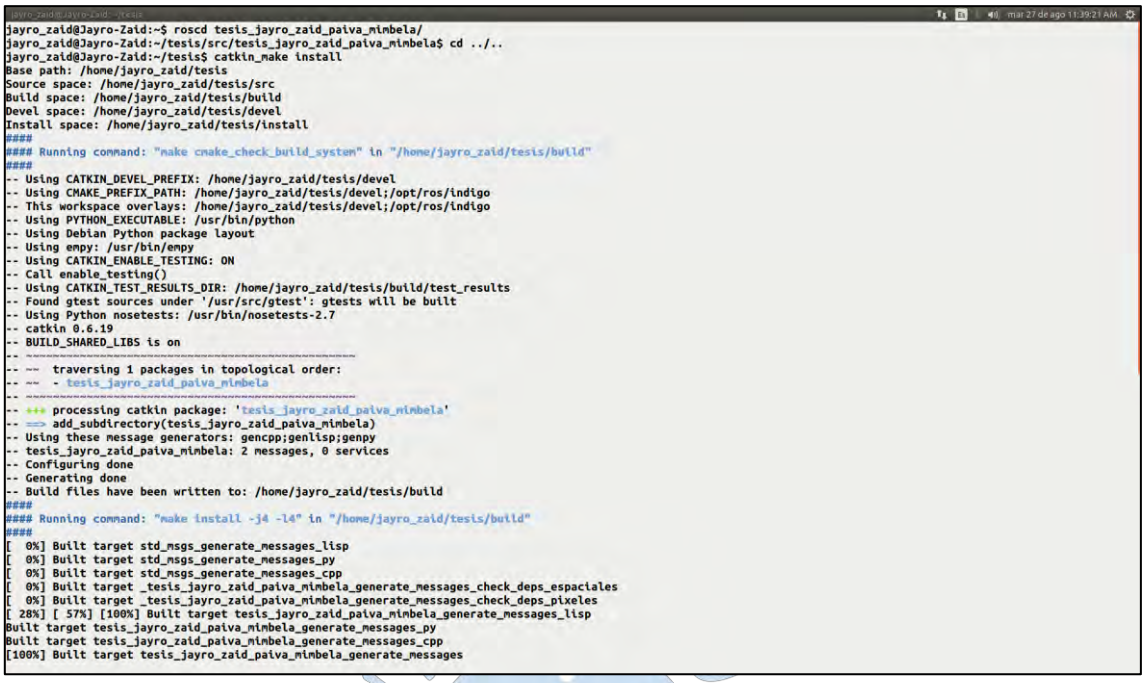

Figura 67. Creación del código fuente *espaciales.h* desde una terminal de Ubuntu.

Fuente: Elaboración propia.

Igual que en el apartado **3.3.1** del presente capítulo, el archivo *espaciales.msg* se convierte en un archivo de código fuente; en este caso, se ha creado un archivo con el nombre *espaciales.h*, localizado en el mismo directorio que *pixeles.h*, cuyo contenido es el que se muestra en el Apéndice A-2 de la presente tesis.

3.4. Creación de los nodos del sistema de posicionamiento *indoor*. Los nodos, como se define en el apartado **2.1.2** del presente capítulo, es el término para un ejecutable que está conectado a la red ROS. A continuación, se procede a crear los nodos correspondientes a los subsistemas que conforman el sistema de posicionamiento *indoor*.

3.4.1. *Nodo del sistema de detección y seguimiento.* Se crea el archivo *src/sistema_deteccion_seguimiento.cpp* dentro del paquete creado en el apartado **3.2** del presente capítulo. Parte del código que comprende este archivo, y que sigue la metodología detallada en el apartado **4.2** del Capítulo 2, se explica a continuación. El código completo se encuentra en el Apéndice B-1 de la presente tesis.

En primer lugar se incluyen las librerías correspondientes; en este caso, las librerías de OpenCV y de ROS, tal como se muestra a continuación:

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "ros/ros.h"
```

Con las librerías agregadas, se inicializa ROS especificando el nombre del nodo que representa este primer ejecutable, procurando que sea único en el sistema. Para esta tesis, el primer ejecutable recibe el nombre de: **sistema_deteccion_seguimiento**. A continuación, se muestra la línea de código que realiza esta acción:

```
init(argc, argv, "sistema_deteccion_seguimiento");
```

Inicializado ROS, se crea un objeto de la clase *NodeHandle*, tal como se muestra en la siguiente línea de código:

```
NodeHandle n;
```

Ahora, se crea un primer tópico llamado *coordenadas_pixeles* en donde se especifica la posibilidad de publicar mensajes del tipo *pixeles* a través de él; con ello, el maestro es capaz de comunicar al resto de nodos que existe información publicada en el tópico creado. La línea de código que realiza esta acción es la siguiente:

```
Publisher pub = n.advertise<pixeles>("coordenadas_pixeles",1000);
```

El segundo argumento indica la cantidad de mensajes que puede almacenar antes de comenzar con la publicación de los mismos. El método *advertise*(), de la clase *NodeHandle*, retorna un objeto del tipo *Publisher* que cuenta con el método *publish*(), el cual permite publicar mensajes en el tópico *coordenadas_pixeles*.

Luego, se aperturan las cámaras creando un objeto de la clase *VideoCapture*, uno por cada cámara:

```
VideoCapture camara_01(1);
VideoCapture camara_02(2);
```

Obtenidas las imágenes de ambas cámaras, se llevan del espacio de color BGR al HSV, para luego ser binarizadas bajo el método del valor umbral, detallado en el apartado **4.2** del Capítulo 2, y aplicar las transformaciones morfológicas de *erode* y *dilate*, detalladas en el apartado **4.2.3** del capítulo en mención. A continuación, se muestra el bloque de código que realiza este proceso:

```
//Creación del elemento estructural
Mat element = getStructuringElement(MORPH_RECT, Size(15,15));

//Conversión de BGR a HSV – Cámara 01
cvtColor(Imgc1, hsvImgc1, CV_BGR2HSV);
inRange(hsvImgc1,  Scalar(lowHc1,  lowSc1,  lowVc1),  Scalar(highHc1,  highSc1,
highVc1), binImgc1);

//Aplicación de filtros
erode(binImgc1, binImgc1, element);
dilate(binImgc1, binImgc1, element);

//Conversión de BGR a HSV – Cámara 02
cvtColor(Imgc2, hsvImgc2, CV_BGR2HSV);
inRange(hsvImgc2,  Scalar(lowHc2,  lowSc2,  lowVc2),  Scalar(highHc2,  highSc2,
highVc2), binImgc2);

//Aplicación de filtros
erode(binImgc2, binImgc2, element);
dilate(binImgc2, binImgc2, element);
```

Obtenidas las imágenes binarias de ambas cámaras, se procede con la identificación de los contornos; para ello, se emplean las siguientes líneas de código:

```
findContours(cam1, contoursc1b, hierarchyc1b, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
findContours(cam2, contoursc2b, hierarchyc1b, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
```

Identificados los contornos, se aproximan a un conjunto de líneas con el fin de simplificar el procesamiento de la imagen. Realizada esta aproximación, se encuentra al mínimo círculo circunscrito, de tal modo que la posición del mismo representa la posición del marcador en pixeles; esto es, la posición del marcador en el plano imagen (2D) de ambas cámaras, lógica detallada en el apartado **5.1.3** del Capítulo 2. A continuación, se muestra el bloque de código que realiza este proceso, en donde lo más importante son las variables **u**, **v**, **up** y **vp**, puesto que representan las coordenadas del marcador en pixeles, la cual es la información de interés a publicar en el tópico *coordenadas_pixeles*.

```
//Para la cámara 1
for (size_t i = 0; i < contoursc1b.size(); i++)
{
 approxPolyDP(Mat(contoursc1b[i]), contours_polyc1[i], 3, true);
 minEnclosingCircle(contours_polyc1[i], centerc1[i], radiusc1[i]);
}

for (size_t i = 0; i< contoursc1b.size(); i++)
{
 Scalar color = Scalar(255, 255, 255);
 drawContours(cam1, contours_polyc1, (int)i, color, 1, 8, vector<Vec4i>(), 0,
Point());
 circle(cam1, centerc1[i], (int)radiusc1[i], color, 2, 8, 0);
 circle(Imgc1, centerc1[i], (int)radiusc1[i], color, 2, 8, 0);
 putText(cam1,     format("(%0.1f,%0.1f)",     centerc1[i].x,     centerc1[i].y),
centerc1[i], CV_FONT_NORMAL, 1, color, 1);

 u = centerc1[i].x;
 v = centerc1[i].y;
}

//Para la cámara 2
for (size_t i = 0; i < contoursc2b.size(); i++)
{
 approxPolyDP(Mat(contoursc2b[i]), contours_polyc2[i], 3, true);
 minEnclosingCircle(contours_polyc2[i], centerc2[i], radiusc2[i]);
}

for (size_t i = 0; i< contoursc2b.size(); i++)
{
 Scalar color = Scalar(255, 255, 255);
 drawContours(cam2, contours_polyc2, (int)i, color, 1, 8, vector<Vec4i>(), 0,
Point());
 circle(cam2, centerc2[i], (int)radiusc2[i], color, 2, 8, 0);
 circle(Imgc2, centerc2[i], (int)radiusc2[i], color, 2, 8, 0);
 putText(cam2,     format("(%0.1f,%0.1f)",     centerc2[i].x,     centerc2[i].y),
centerc2[i], CV_FONT_NORMAL, 1, color, 1);

 up = centerc2[i].x;
 vp = centerc2[i].y;
}
```

Para hacer efectiva la publicación de la información de interés de este primer nodo, se crea el objeto de una clase adaptada que generalmente es generada por un archivo *msg*, tal como se muestra en el siguiente bloque de código:

```
coordenadas_pixeles msg;
msg.u=u;
msg.v=v;
msg.up=up;
msg.vp=vp;
```

Finalmente, con el método *publish*() se publica la información correspondiente, de tal modo que pueda ser empleada por cualquier nodo que se encuentre en la red y se suscriba al tópico *coordenadas_pixeles*. La línea de código que permite realizar esta acción es la siguiente:

```
pub.publish(msg);
```

En resumen, el proceso llevado a cabo tras lanzar este primer nodo es el siguiente:

- Inicialización del sistema ROS.

- Se anuncia al maestro ROS la posibilidad de publicar mensajes del tipo *pixeles* en el tópico *coordenadas_pixeles*.

- Se entra en un ciclo de publicación de mensajes en el tópico *coordenadas_pixeles.*

Para construir este primer nodo, se le realizan algunas modificaciones al archivo *CMakeLists.txt*, creado al construir el paquete; el mismo debe ser igual al bloque que se muestra a continuación:

```
cmake_minimum_required(VERSION 2.8.3)
project(tesis_jayro_zaid_paiva_mimbela)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs
message_generation OpenCV
)

## Declare ROS messages
add_message_files(FILES pixeles.msg)

## Generate added messages and services with any dependencies listed here
generate_messages(DEPENDENCIES std_msgs)

## catkin specific configuration
catkin_package(CATKIN_DEPENDS roscpp rospy std_msgs message_runtime)

## Build
include_directories(${catkin_INCLUDE_DIRS} ${OpenCV_INCLUDE_DIRS})
```

```
## Declare a C++ executable
add_executable(sistema_deteccion_seguimiento
src/ sistema_deteccion_seguimiento.cpp)

## Add cmake target dependencies of the executable
add_dependencies(sistema_deteccion_seguimiento
tesis_jayro_zaid_paiva_mimbela_generate_messages_cpp)

## Specify libraries to link a library or executable target against
target_link_libraries(sistema_deteccion_seguimiento
${catkin_LIBRARIES} ${OpenCV_LIBRARIES})
```

Finalmente, se vuelve a construir el paquete para crear el primer archivo ejecutable, el cual se encuentra, por defecto, en la siguiente ruta: *~/tesis/devel/lib/tesis_jayro_zaid_paiva_mimbela.* Para ello, se ejecutan los siguientes comandos:

```
$ cd ~/catkin_ws
$ catkin_make
```

Al ser ejecutados, debe aparecer lo que se muestra en la Figura 68.

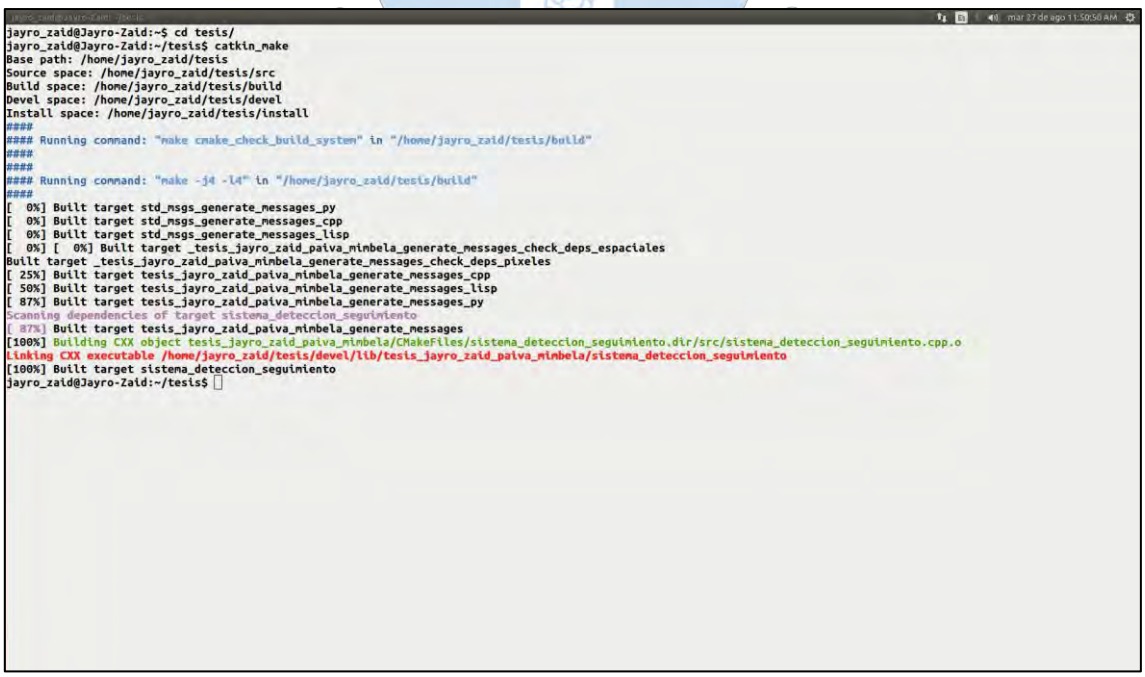

Figura 68. Creación del nodo del sistema de detección y seguimiento desde una terminal de Ubuntu.

Fuente: Elaboración propia.

3.4.2. ***Nodo del sistema de conversión de espacios 2D a 3D.*** Igual que en el apartado **3.4.1** del presente capítulo, se crea el archivo *src/sistema_conversion_2d_3d.cpp* dentro del paquete creado. Parte del código que comprende este archivo, y que sigue la metodología detallada en el apartado **5.3** del Capítulo 3, se explica a continuación. El código completo se encuentra en el Apéndice B-2 de la presente tesis.

En primer lugar se incluyen las librerías correspondientes; en este caso, se vuelven a incluir las librerías de OpenCV y de ROS, tal como se muestra a continuación:

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "ros/ros.h"
```

Con las librerías agregadas, se definen las matrices intrínsecas y extrínsecas de cada cámara, las cuales son necesarias para el cálculo de la posición del marcador en el entorno. A continuación, se muestra el bloque de código que realiza esta acción:

```
//Matriz de rotación - Cámara 01
float R1[3][3]=
{
   {Q11,Q21,Q31},
   {Q12,Q22,Q32},
   {Q13,Q23,Q33}
};

//Matriz de traslación - Cámara 01
float t1[3][1]=
{
   {1.5},
   {53.0},
   {106.0}
};

//Matriz de rotación - Cámara 02
float R2[3][3]=
{
   {R11,R21,R31},
   {R12,R22,R32},
   {R13,R23,R33}
};

//Matriz de traslación - Cámara 02
float t2[3][1]=
{
   {46.0},
   {1.5 },
   {106.0}
};
```

```
//Definición de matrices intrínsecas
//Cámara 01
float A1[3][3]=

{
   {641.005175542343,0.00000000000000,314.264418608371},
   {0.00000000000000,640.456610088887,240.340845427825},
   {0.00000000000000,0.00000000000000,1.00000000000000}
};

//Cámara 02
float A2[3][3]=

{
   {633.130740940056,0.00000000000000,318.85831503788},
   {0.00000000000000,633.27789680685 ,231.268076567509},
   {0.00000000000000,0.00000000000000,1.00000000000000}
};
```

Definidas las matrices, es necesario construir las funciones que permitan llevar a cabo la metodología de reconstrucción que se sigue. A continuación se muestran los bloque de código de las funciones *prodX*() y *vectorunitario*():

```
void prodX( float cx , float cy, float cz,float dx , float dy, float dz, float
&ux, float &uy, float &uz)
{
   ux = cy*dz - cz*dy;
   uy = cz*dx - cx*dz;
   uz = cx*dy - cy*dx;
}

void vectorunitario( float c1 , float c2, float c3, float &u1, float &u2, float
&u3)
{
   u1=(c1/sqrt(pow(c1,2)+pow(c2,2)+pow(c3,2)));
   u2=(c2/sqrt(pow(c1,2)+pow(c2,2)+pow(c3,2)));
   u3=(c3/sqrt(pow(c1,2)+pow(c2,2)+pow(c3,2)));
}
```

Así mismo, se construye una función que permita dar solución al sistema de ecuaciones (3.79), planteado en el Capítulo 3, puesto que su solución conlleva a obtener la posición del marcador en el entorno. Para dar solución al sistema de ecuaciones, se opta por el método de gauss. A continuación, se muestra parte de la función *gauss*():

```
void gauss(float m[3][4],float &sol1, float &sol2, float &sol3)
{
   m[0][1] = m[0][1]/m[0][0];
   m[0][2] = m[0][2]/m[0][0];
   m[0][3] = m[0][3]/m[0][0];
   m[0][0] = m[0][0]/m[0][0];
   ...
   ...
   ...
```

```
   ...
   ...
   m[0][0] = m[0][0] - m[0][1]*m[1][0];
   m[0][2] = m[0][2] - m[0][1]*m[1][2];
   m[0][3] = m[0][3] - m[0][1]*m[1][3];
   m[0][1] = m[0][1] - m[0][1]*m[1][1];

   m[2][0] = m[2][0] - m[2][1]*m[1][0];
   m[2][2] = m[2][2] - m[2][1]*m[1][2];
   m[2][3] = m[2][3] - m[2][1]*m[1][3];
   m[2][1] = m[2][1] - m[2][1]*m[1][1];

   m[2][3] = m[2][3]/m[2][2];
   m[2][2] = m[2][2]/m[2][2];

   m[0][0] = m[0][0] - m[0][2]*m[2][0];
   m[0][1] = m[0][1] - m[0][2]*m[2][1];
   m[0][3] = m[0][3] - m[0][2]*m[2][3];
   m[0][2] = m[0][2] - m[0][2]*m[2][2];

   m[1][0] = m[1][0] - m[1][2]*m[2][0];
   m[1][1] = m[1][1] - m[1][2]*m[2][1];
   m[1][3] = m[1][3] - m[1][2]*m[2][3];
   m[1][2] = m[1][2] - m[1][2]*m[2][2];

   sol1 = m[0][3];
   sol2 = m[1][3];
   sol3 = m[2][3];
}
```

Otra función necesaria es la función de devolución de llamada[36], la cual es llamada cuando un nuevo mensaje llega al tópico al que está suscrito este segundo nodo. La función es la que muestra a continuación:

```
void llamada( const pixeles::ConstPtr& msg)
{
 u  = msg->u;
 v  = msg->v;
 up = msg->up;
 vp = msg->vp;
}
```

Ahora, se inicializa ROS especificando el nombre del nodo que representa a este segundo ejecutable. Para esta tesis, el segundo ejecutable recibe el nombre de **sistema_conversion_2d_3d**. A continuación, se muestra la línea de código que realiza esta acción:

```
init(argc, argv,"sistema_conversion_2d_3d");
```

[36] Función que es usada como argumento de otra función, de modo que cuando se llame a esta última se llame a la primera.

Inicializado ROS, se crea un objeto de la clase *NodeHandle*, tal como se muestra en la siguiente línea de código:

```
NodeHandle n;
```

Ahora, se especifica que este nodo va a estar suscrito al tópico *coordenadas_pixeles*, de tal modo que permite a ROS llamar a la función *llamada*() cada vez que llegue un mensaje. Cabe recordar que la información que se publica en este tópico son las coordenadas del marcador en el plano imagen de cada cámara (**u, v, up, vp**), las cuales son variables que forman parte del sistema de ecuaciones con el que se estima la posición del marcador en el entorno. La línea de código que realiza esta acción es la siguiente:

```
Subscriber sub=n.subscribe("coordenadas_pixeles",1000,llamada);
```

Como se menciona, la información recogida del tópico *coordenadas_pixeles* es empleada para obtener la posición del marcador en el entorno (**x, y, z**); sin embargo, esta última información también es necesaria para reconstruir la trayectoria recorrida del mismo. Por ello, es necesario publicar esta información con el fin de ser recogida por un nodo que cumpla con este propósito. Dicho esto, se crea un segundo tópico, llamado *coordenadas_espaciales*, en donde se especifica la posibilidad de publicar mensajes del tipo *espaciales* a través de él. La línea de código que realiza esta acción es la siguiente:

```
Publisher pub=n.advertise<espaciales>("coordenadas_espaciales",1000);
```

Con la información recogida del tópico *coordenadas_pixeles*, se muestra el bloque de código que realiza las operaciones propuestas en la metodología de reconstrucción:

```
kxc1= -1*(u-A1[0][2])/A1[0][0];
kyc1= -1*(v-A1[1][2])/A1[1][1];
kxc2= -1*(up-A2[0][2])/A2[0][0];
kyc2= -1*(vp-A2[1][2])/A2[1][1];

kxr1= (Q11*kxc1)+(Q12*kyc1)+(Q13*1);
kyr1= (Q21*kxc1)+(Q22*kyc1)+(Q23*1);
kzr1= (Q31*kxc1)+(Q32*kyc1)+(Q33*1);
```

```
kxr2= (R11*kxc2)+(R12*kyc2)+(R13*1);
kyr2= (R21*kxc2)+(R22*kyc2)+(R23*1);
kzr2= (R31*kxc2)+(R32*kyc2)+(R33*1);

vectorunitario(kxr1,kyr1,kzr1,ux,uy,uz);
vectorunitario(kxr2,kyr2,kzr2,vx,vy,vz);
prodX(ux,uy,uz,vx,vy,vz,wx,wy,wz);

float g[3][4] = {ux,-vx,-wx,t2[0][0]-t1[0][0],
                 uy,-vy,-wy,t2[1][0]-t1[1][0],
                 uz,-vz,-wz,t2[2][0]-t1[2][0]};
```

Definido el sistema de ecuaciones, se hace uso de la función, previamente creada, *gauss*(). La función es llamada como se muestra a continuación:

```
gauss(g, lambda, nu, t);
```

De la solución del sistema de ecuaciones se obtienen los valores de **λ**, **μ** y **p**, a partir de los cuales se obtienen las coordenadas del marcador con el siguiente bloque de código:

```
Sx = ux*lambda + t1[0][0];
Sy = uy*lambda + t1[1][0];
Sz = uz*lambda + t1[2][0];
Rx = vx*nu + t2[0][0];
Ry = vy*nu + t2[1][0];
Rz = vz*nu + t2[2][0];

float x=(Sx+Rx)/2;
float y=(Sy+Ry)/2;
float z=(Sz+Rz)/2;
```

Del último bloque de código lo más importante son las variables **x**, **y** y ***z***, puesto que representan a las coordenadas del marcador en el entorno, cumpliendo con el objetivo de la presente tesis. Esta información es publicada en el tópico *coordenadas_espaciales*, creado previamente, volviendo a usar el objeto de una clase adaptada, tal como se muestra en el siguiente bloque de código:

```
espaciales msg;
msg.x=x;
msg.y=y;
msg.z=z;
pub.publish(msg);
```

Finalmente, con el método *publish*() se publica la información correspondiente, de tal modo que pueda ser empleada por cualquier nodo que se encuentre en la red y se suscriba al tópico *coordenadas_espaciales.* La línea de código que permite realizar esta acción es la siguiente:

```
pub.publish(msg);
```

Adicionalmente, se agrega una función necesaria cuando se está recibiendo devoluciones de llamada; es decir, cuando el nodo se encuentra suscrito a un determinado tópico; si es omitida, las devoluciones no son llamadas. La función es llamada como se muestra a continuación:

```
spinOnce();
```

En resumen, el proceso llevado a cabo tras lanzar este segundo nodo es el siguiente:

- Inicialización del sistema ROS.
- Suscripción al tópico *coordenadas_pixeles*.
- Cuando un mensaje llega, se llama a la función de devolución de llamada, y se recoge la información del tópico al que está suscrito el nodo.
- Se obtienen, a partir de la información recogida, las coordenadas del marcador en el entorno.
- Se anuncia al maestro ROS la posibilidad de publicar mensajes del tipo *espaciales* en el tópico *coordenadas_espaciales.*
- Se entra en un ciclo de publicación de mensajes en el tópico *coordenadas_espaciales.*

Para construir este segundo nodo, al igual que en el apartado **3.4.1** del presente capítulo, se le realizan algunas modificaciones al archivo *CMakeLists.txt*. Cabe mencionar que este archivo lo comparten en común el primer y segundo nodo, el cual debe ser igual al bloque se muestra a continuación:

```
cmake_minimum_required(VERSION 2.8.3)
project(tesis_jayro_zaid_paiva_mimbela)

## Find catkin and any catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs message_generation
OpenCV
)

## Declare ROS messages
  add_message_files(FILES pixeles.msg espaciales.msg)

## Generate added messages and services with any dependencies listed here
  generate_messages(DEPENDENCIES std_msgs)

## catkin specific configuration
catkin_package(CATKIN_DEPENDS roscpp rospy std_msgs message_runtime)

## Build
 include_directories(${catkin_INCLUDE_DIRS} ${OpenCV_INCLUDE_DIRS})

## Declare a C++ executable
add_executable(sistema_deteccion_seguimiento
src/sistema_deteccion_seguimiento.cpp)
add_executable(sistema_conversion_2d_3d src/sistema_conversion_2d_3d.cpp)

## Add cmake target dependencies of the executable
add_dependencies(sistema_deteccion_seguimiento
tesis_jayro_zaid_paiva_mimbela_generate_messages_cpp)
add_dependencies(sistema_conversion_2d_3d
tesis_jayro_zaid_paiva_mimbela_generate_messages_cpp)


## Specify libraries to link a library or executable target against
target_link_libraries(sistema_deteccion_seguimiento ${catkin_LIBRARIES}
target_link_libraries(sistema_conversion_2d_3d ${catkin_LIBRARIES}
```

Finalmente, se vuelve a construir el paquete para crear el segundo archivo ejecutable, el cual se encuentra, por defecto, en la siguiente ruta: *~/tesis/devel/lib/tesis_jayro_zaid_paiva_mimbela.* Para ello, se ejecutan los siguientes comandos:

```
$ cd ~/catkin_ws
$ catkin_make
```

Al ser ejecutados, debe aparecer lo que se muestra en la Figura 69.

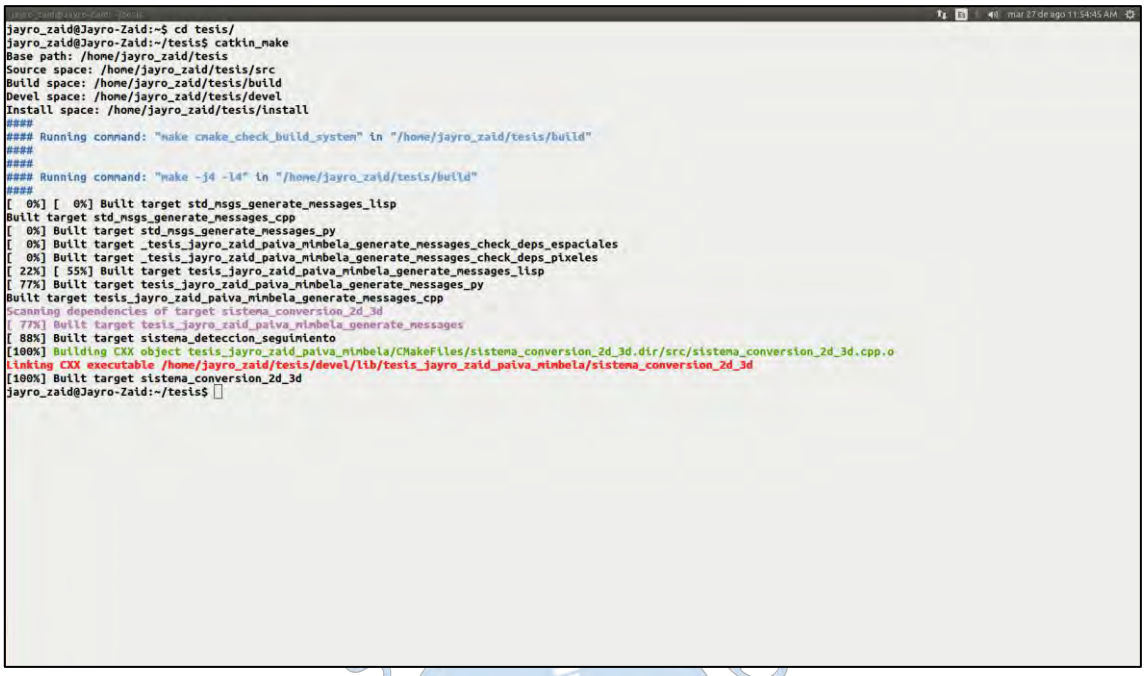

Figura 69. Creación del nodo del sistema de conversión de espacios 2D a 3D desde una terminal de Ubuntu. Fuente: Elaboración propia.

3.4.3. ***Nodo del sistema de reconstrucción de trayectorias.*** Los nodos previamente creados tienen una característica en común, ambos son escritos empleando el lenguaje de programación de C++; sin embargo, el nodo que representa a este tercer ejecutable está escrito en el lenguaje de programación de Python. Dicho esto, se puede pensar que es imposible comunicar a este nodo con los otros dos, pero lo cierto es que sí es posible comunicar nodos escritos en distinto lenguajes. Esto es posible gracias a que los códigos fuente (de extensión *.h)*, creados a partir de un archivo *msg*, permiten ser usados para mensajes en todos los lenguajes compatibles en ROS. A continuación, se describe la creación de este último nodo:

En primer lugar, se crea una carpeta llamada *scripts* en el directorio del paquete *tesis_jayro_zaid_paiva_mimbela*; y luego, se crea el archivo *reconstruccion_trayectorias.py* dentro de ella. Al igual que los nodos anteriores, parte del código se explica a continuación. El código completo se encuentra en el Apéndice B-3 de la presente tesis.

Para empezar, todo nodo ROS escrito en Python tiene la declaración que se muestra a continuación, la cual asegura el que el código sea ejecutado como un script[37] de Python:

[37] Programas generalmente pequeños para tareas específicas que son interpretados línea a línea en tiempo real para su ejecución.

```
#!/usr/bin/env python
```

Esta línea de código asegura el que el código sea ejecutado como un script[38] de Python.

Como en los otros nodos, se incluyen las librerías correspondientes; en este caso, se incluyen las librerías de *matplotlib* y *mplot3D* para la interfaz de reconstrucción; y *rospy,* la librería de ROS en Python, tal como se muestra a continuación:

```
import rospy
import sys
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import mpl_toolkits.mplot3d.axes3d as p3
import Image
```

Al igual que en el apartado **3.4.2** del presente capítulo, es necesario definir una función de devolución de llamada. En Python, se define tal como se muestra a continuación:

```
def llamada(msg):
    global x, y, z
    x=msg.x
    y=msg.y
    z=msg.z
```

Ahora, se inicializa ROS especificando el nombre del nodo que representa este tercer ejecutable. Para esta tesis, el tercer ejecutable recibe el nombre de **sistema_reconstruccion_trayectorias**. A continuación, se muestra la línea de código que realiza esta acción:

```
rospy.init_node("sistema_reconstruccion_trayectoria")
```

Como también se menciona en el apartado **3.4.2** del presente capítulo, el segundo nodo se encuentra publicando la posición del marcador en el tópico *coordenadas_espaciales,* siendo la información que necesita el presente nodo para cumplir con la función de reconstruir la

[38] Programas generalmente pequeños para tareas específicas que son interpretados línea a línea en tiempo real para su ejecución.

trayectoria recorrida por el marcador; por ello, es necesario suscribirse al tópico mencionado. En Python, la línea de código que realiza esta acción es la siguiente:

```
sub = rospy.Subscriber("coordenadas_espaciales", espaciales, llamada)
```

Recogida la información necesaria; es decir, la posición del marcador en el entorno, ya se puede diseñar la interfaz de reconstrucción. Para ello, se crea el entorno visual en donde se muestra la trayectoria recorrida por el marcador en plano tridimensional **XYZ**, y los planos **XY**, **XZ** y **YZ**.

```
fig = plt.figure(1,figsize=(10,15),
dpi=100,facecolor='#00467A',edgecolor='k',constrained_layout=False)

ax1 = fig.add_subplot(1, 2, 1,projection="3d")
ax2 = fig.add_subplot(2, 4, 3)
ax3 = fig.add_subplot(2, 4, 4)
ax4 = fig.add_subplot(2, 4, 7)
```

Para la animación de la trayectoria recorrida, se crea la siguiente función:

```
def animate(i):

    ax1.scatter([x], [y], [z], color="b", s=10)
    ax2.scatter([x], [y], color="b", s=10)
    ax3.scatter([x], [z], color="b", s=10)
    ax4.scatter([y], [z], color="b", s=10)
```

Definida la función, se emplean las siguientes líneas de código para mostrar la animación en los planos mencionados:

```
ani = animation.FuncAnimation(fig, animate, interval=1000)
plt.show()
```

Adicionalmente, se añade una función que evita que el nodo salga de ejecución hasta que realmente sea detenido:

```
rospy.spin()
```

En resumen, el proceso llevado a cabo tras lanzar este último nodo es el siguiente:

- Inicialización del sistema ROS.

- Suscripción al tópico *coordenadas_espaciales*.

- Cuando un mensaje llega, se llama a la función de devolución de llamada y se recoge la información del tópico al que está suscrito el nodo.

- Se obtienen, a partir de la información recogida, las coordenadas del marcador en el entorno.

- Se muestra en una interfaz gráfica con la reconstrucción de la trayectoria recorrida.

Para construir este último nodo, se debe convertir al archivo *sistema_reconstruccion_trayectorias* en un archivo ejecutable. Para ello, se ejecuta el siguiente comando:

```
chmod + x reconstruccion_trayectoria.py
```

Finalmente, se vuelve al directorio del espacio de trabajo *catkin* y se construye el paquete ejecutando los siguientes comandos:

```
$ cd ~ / tesis
$ catkin_make
```

Al ser ejecutados, debe aparecer lo que se muestra en la Figura 70.

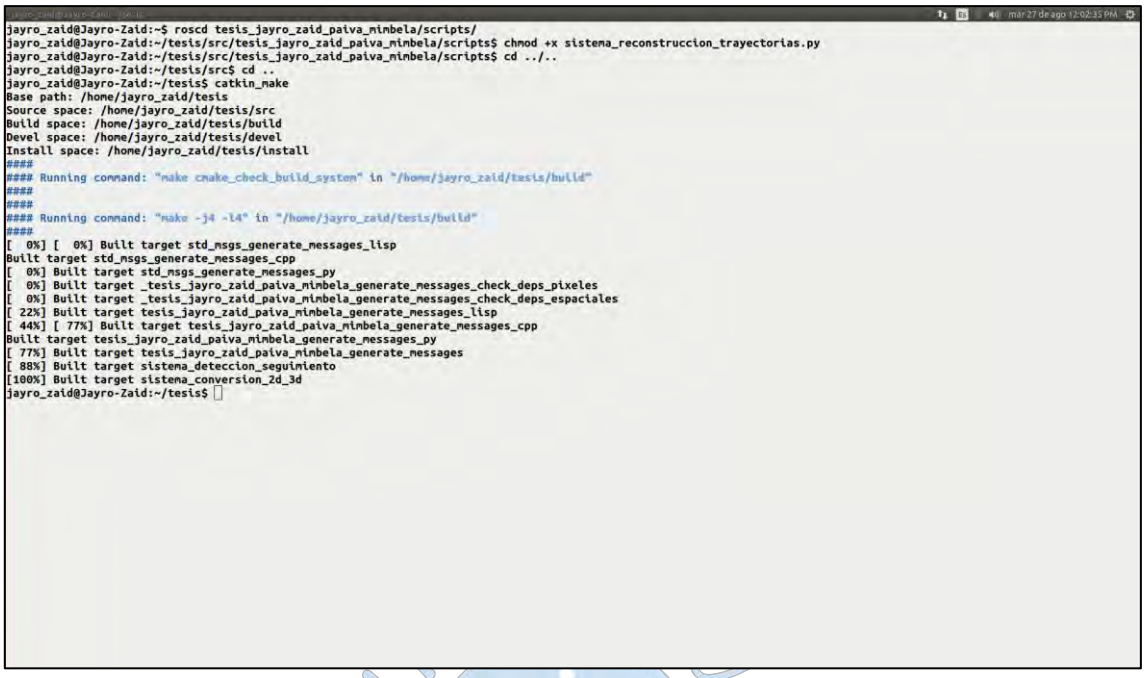

Figura 70. Creación del nodo del sistema de reconstrucción de trayectorias desde una terminal de Ubuntu. Fuente: Elaboración propia.

Las pruebas del sistema integrado son llevadas a cabo y mostradas en el siguiente capítulo de la presente tesis.


UNIVERSITAS STUDIORUM PIURENSIS

Capítulo 5

Pruebas y resultados del sistema de posicionamiento *indoor*

1. Introducción

En este último capítulo se describe al entorno en donde es implementado el sistema de posicionamiento desarrollado, y son llevadas a cabo las pruebas respectivas. Se detalla el proceso de calibración de las cámaras y el procedimiento que se sigue para validar al sistema. Por último, se muestran los resultados junto a un análisis de los mismos.

2. Entorno de pruebas

En la búsqueda de un espacio que brinde las facilidades para implementar al sistema desarrollado y llevar a cabo las pruebas del mismo, se desarrolla un entorno de pruebas que consta en primer lugar de una estructura metálica, la cual se muestra en la Figura 71.

Figura 71. Estructura metálica.
Fuente: Elaboración propia.

En segundo lugar se adiciona una malla cuadricular particionada en cuadros de 1 cm que es colocada en el plano **Z = 0**; es decir, en el plano **XY** del entorno. Debido a esta partición de la malla, la cual se muestra en la Figura 72, el error por defecto del sistema está medido en cm. Por último, se procede con la implementación de las cámaras junto al sistema de iluminación que se menciona en el apartado **2.4** del Capítulo 2.

Figura 72. Malla cuadricular para orientación dentro del entorno de pruebas.
Fuente: Elaboración propia.

El entorno de pruebas completamente implementado se muestra en la Figura 73, en donde se señala su origen de coordenadas. El fin de las etiquetas que están ubicadas en puntos arbitrarios del plano **XY** se detalla más adelante.

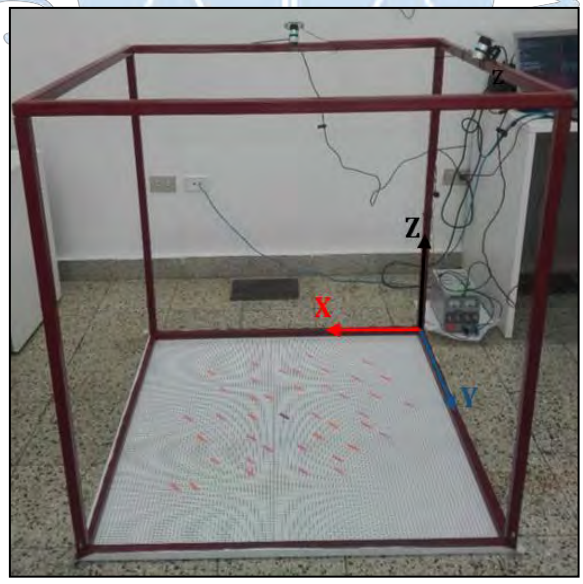

Figura 73. Entorno de pruebas completamente implementado.
Fuente: Elaboración propia.

3. Calibración de las cámaras

El proceso de calibración se lleva a cabo en *Camera Calibrator Toolbox*, una de las herramientas incluidas en Matlab[39]. Una imagen de la interfaz de inicio de esta herramienta se muestra en la Figura 74.

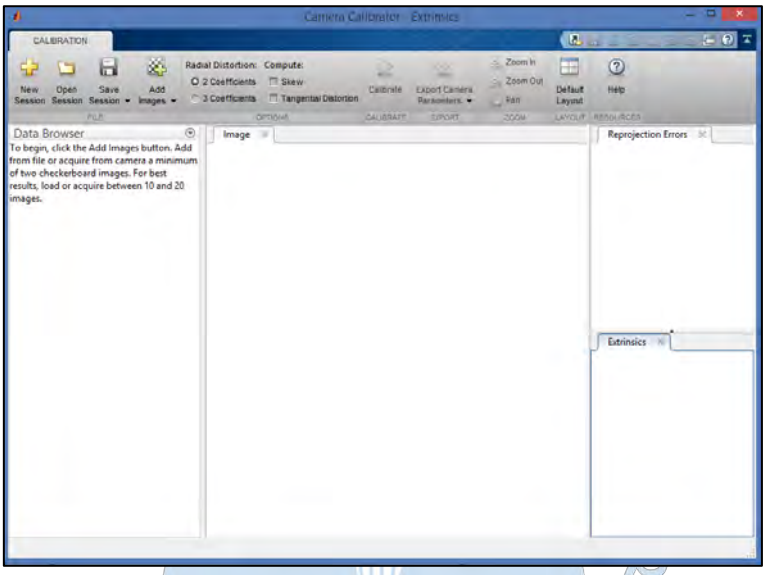

Figura 74. Interfaz de inicio de la herramienta *Camera Calibrator*.
Fuente: Elaboración propia.

Camera Calibrator necesita de una cierta cantidad de fotos del patrón de calibración en distintas posiciones, siendo el procedimiento mencionado en el apartado **4.2** del Capítulo 3. En las Figuras 75 y 76 se muestran algunas de las fotos que se ingresaron para poder hallar las matrices intrínsecas de ambas cámaras, las cuales son las siguientes:

$$\mathbf{A1} = \begin{bmatrix} 641.01 & 0 & 314.26 \\ 0 & 640.46 & 240.34 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{A2} = \begin{bmatrix} 633.13 & 0 & 318.86 \\ 0 & 633.28 & 231.27 \\ 0 & 0 & 1 \end{bmatrix}$$

Donde **A1** y **A2** son las matrices de parámetros intrínsecos de la cámaras 1 y 2 respectivamente. Cabe recordar que los parámetros que conforman esta matriz son esenciales para el cálculo de la posición del marcador en el entorno; por ello, la importancia de una adecuada calibración.

[39] Software de entorno numérico.

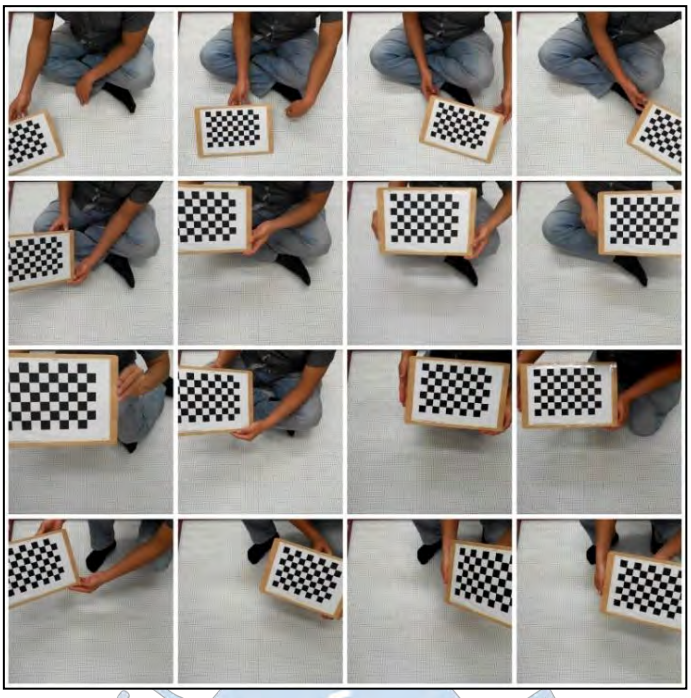

Figura 75. Imágenes del patrón tomadas con la cámara 01.
Fuente: Elaboración propia.

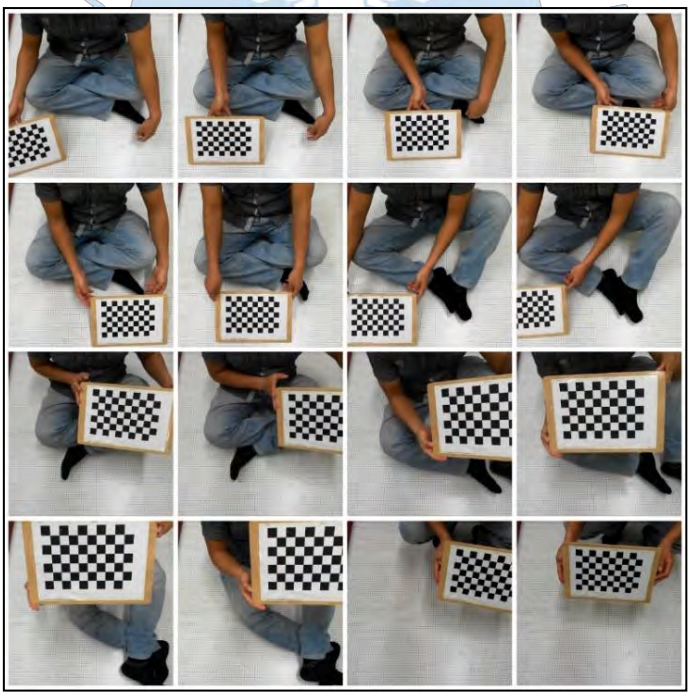

Figura 76. Imágenes del patrón tomadas con la cámara 02.
Fuente: Elaboración propia.

Con respecto a las matrices extrínsecas, las matrices de rotación son halladas a partir de los siguientes ángulos:

$$\alpha_1 = 22.5° \quad \beta_1 = \pi° \quad \theta_1 = \frac{\pi}{2}°$$
$$\alpha_2 = 22.5° \quad \beta_2 = \pi° \quad \theta_2 = 0°$$

Donde los subíndices 1 y 2 hacen referencia a las cámaras respectivas. Cabe mencionar que los ángulos son aproximados, puesto que son obtenidos considerando que el eje **X** del plano imagen de ambas cámaras es paralelo a los ejes **X** y **Y** del entorno de pruebas.

En relación a los vectores de traslación, se mide cuanto es la distancia que existe entre los ejes del entorno y los ejes de cada plano imagen. Los vectores aproximados, en donde los subíndices 1 y 2 hacen referencia a las cámaras respectivas, son los siguientes:

$$t_1 = (1.5, 53, 106)$$
$$t_2 = (46, 1.5, 106)$$

4. Procedimiento para la validación del sistema

Al no contar con un referente con el cual validar y obtener los errores de posicionamiento, se plantea ubicar al marcador en distintos puntos del entorno y comparar la posición calculada por el sistema con la obtenida a partir de mediciones empíricas, consideradas como las reales en adelante. Para obtener las coordenadas reales del marcador, se hace uso del instrumento de medición que se muestra en la Figura 77, el cual consta de un soporte que aloja a una serie de varillas de madera cuyas dimensiones son conocidas y varían entre los 20 cm y 70 cm.

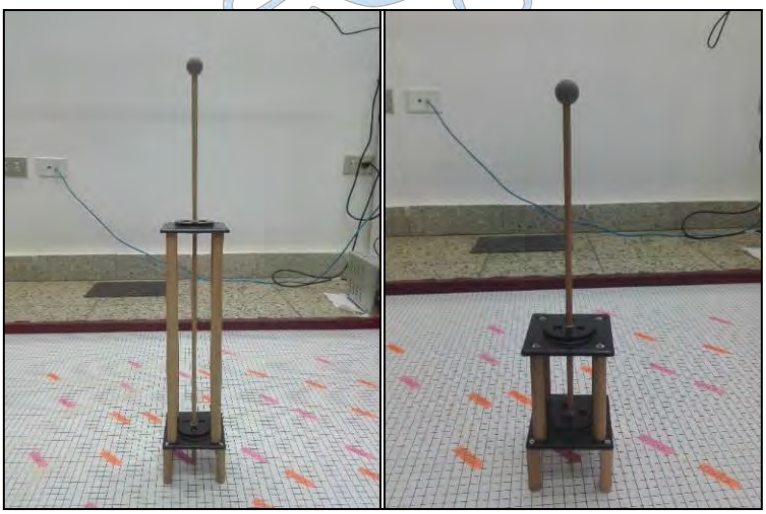

Figura 77. Instrumento de medición empírico.

Fuente: Elaboración propia.

La medición es llevada a cabo de tal modo que ubicando al instrumento en una de las etiquetas que se mencionan en el apartado 2 del presente capítulo, se calculan las coordenadas **x** y **y** del marcador. Una distribución más exacta de las etiquetas se muestra en la Figura 78.

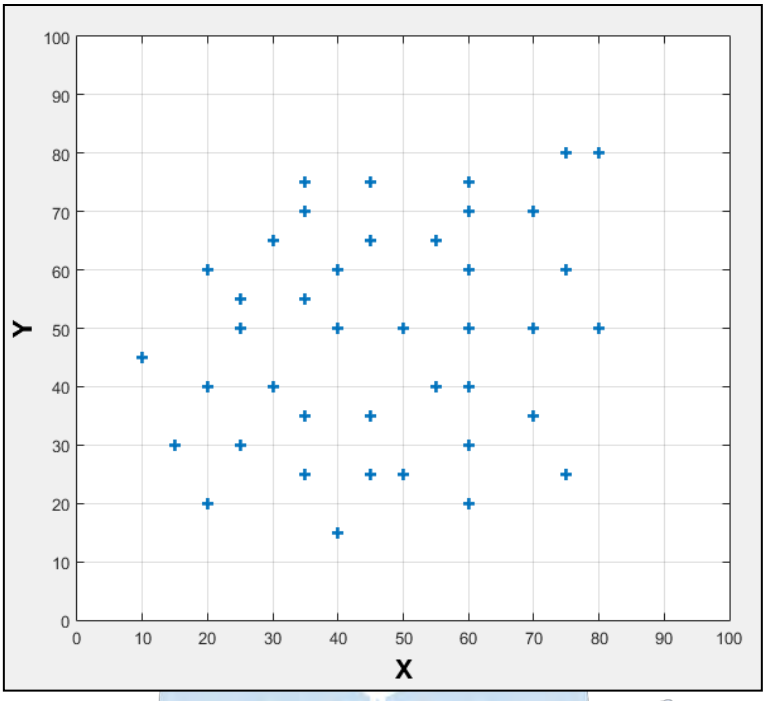

Figura 78. Distribución de las etiquetas en el plano XY del entorno de pruebas.
Fuente: Elaboración propia.

Con respecto a la coordenada **z**, esta es igual al largo de la varilla que se está usando. Para entender lo mencionado, en la Figura 79 se muestra al instrumento de medición ubicado en puntos conocidos del entorno, de tal modo que la validación consiste en comparar la posición que está calculando el sistema con la posición real que se obtiene bajo el procedimiento descrito. El soporte en el instrumento de medición asegura que la varilla se encuentre completamente perpendicular al plano **XY**, evitando validaciones erróneas en la relación a coordenadas **x** y **y**. Cabe mencionar que debido a la naturaleza del sensor empleado en esta capa de percepción, las coordenadas **x**, **y** y **z** se ven condicionadas por el rango visual de las cámaras, puesto que la condición para calcular la posición del marcador en el entorno es que ambas estén observando al mismo. Dicho esto, la restricción más notable tiene lugar en la coordenada **z** con un valor máximo de validación de 70 cm.

Figura 79. Instrumento de medición en distintos puntos del entorno.

Fuente: Elaboración propia.

5. Resultados

En esta parte del capítulo finalmente se llevan a cabo las pruebas del sistema de posicionamiento *indoor*, verificando el correcto funcionamiento de los subsistemas que lo componen. Para ello, se consideran tres puntos arbitrarios, mostrados en la Tabla 4, cuya posición real se obtiene según el procedimiento del apartado **4** del presente capítulo. Cabe recordar que los subsistemas, al ser integrados en ROS, reciben el nombre de nodos. El primer nodo representa al sistema de detección y seguimiento, el segundo al sistema de conversión de espacios 2D a 3D y el tercero al sistema de reconstrucción de trayectorias.

Tabla 4. Coordenadas reales de tres puntos arbitrarios.

	Coordenada Real		
Puntos	**X**	**Y**	**Z**
1	60.0	40.0	20.8
2	55.0	40.0	20.8
3	45.0	35.0	40.8

Fuente: Elaboración propia.

5.1. Resultados del sistema de detección y seguimiento. Este primer nodo al ser ejecutado genera 4 ventanas, tal como se muestra en las Figuras 80, 81 y 82. En el lado (a) de las figuras no solo se puede observar al marcador en ambas cámaras, sino que la región detectada es trazada de color blanco; en consecuencia, el que se muestre completamente blanco el marcador es un indicador de detección satisfactoria. En el lado (b) lo que se muestra es el resultado tras aplicar el procesamiento mencionado en los apartados **4.2** y **5.1** del Capítulo 2.

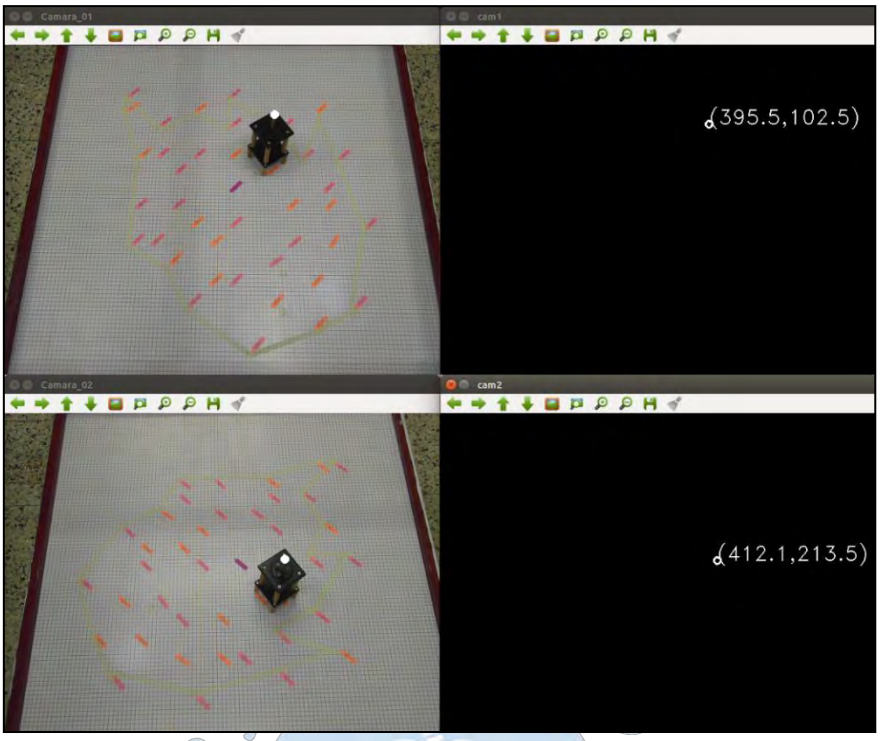

Figura 80. Detección del marcador en ambas cámaras (a). Seguimiento del marcador detectado (b). - Punto 1.

Fuente: Elaboración propia.

Figura 81. Detección del marcador en ambas cámaras (a). Seguimiento del marcador detectado (b). - Punto 2.

Fuente: Elaboración propia.

Figura 82. Detección del marcador en ambas cámaras (a). Seguimiento del marcador detectado (b). - Punto 3.

Fuente: Elaboración propia.

Además de las ventanas que se muestran, se genera una salida de consola en donde se muestran las coordenadas en píxeles calculadas, tal como se muestra en la Figuras 83, 84 y 85. Un resumen de lo calculado por el subsistema se muestra en la Tabla 5.

Posición en pixeles del marcador	
Posición del marcador 1	Posición del marcador 2
u=395.5	up=412.1
v=102.5	vp=213.5

Figura 83. Coordenadas en píxeles del marcador – Punto 1.

Fuente: Elaboración propia.

Posición en pixeles del marcador	
Posición del marcador 1	Posición del marcador 2
u=396.9	up=378.0
v=131.2	vp=213.5

Figura 84. Coordenadas en píxeles del marcador – Punto 2.

Fuente: Elaboración propia.

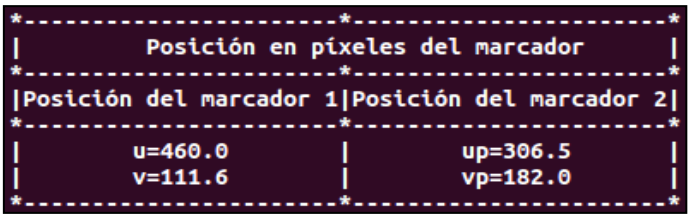

Figura 85. Coordenadas en píxeles del marcador – Punto 3.

Fuente: Elaboración propia.

Tabla 5. Coordenadas en píxeles de tres puntos arbitrarios

	Coordenadas en plano imagen (píxeles)			
	Cámara 01		Cámara 02	
Puntos	**u**	**p**	**up**	**vp**
1	395.5	102.5	412.1	213.5
2	396.9	131.2	378.0	213.5
3	460.0	111.6	306.5	182.0

Fuente: Elaboración propia.

5.2. Sistema de conversión de espacios 2D a 3D. Este segundo nodo al ser ejecutado calcula, a partir de la información brindada por el primer nodo y la metodología del apartado **5.3** del Capítulo 3, la posición del marcador en el entorno. El resultado se muestra a través de una salida de consola, tal como se muestra en las Figuras 86, 87 y 88. Un resumen comparativo entre las coordenadas reales y las calculadas por este subsistema se muestra en la Tabla 6.

Tabla 6. Coordenadas reales vs coordenadas calculadas

	Coordenadas reales			Coordenadas calculadas		
Puntos	**X**	**Y**	**Z**	**X**	**Y**	**Z**
1	60.0	40.0	20.8	60.0	40.0	20.8
2	55.0	40.0	20.8	55.0	40.0	20.5
3	45.0	35.0	40.8	44.9	35.2	40.6

Fuente: Elaboración propia.

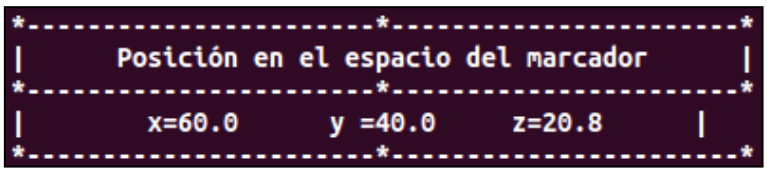

Figura 86. Impresión en consola de las coordenadas calculadas por el sistema *indoor* – Punto 1.

Fuente: Elaboración propia.

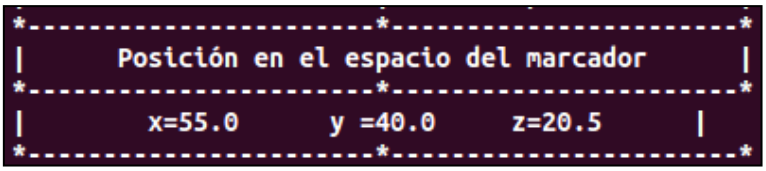

Figura 87. Impresión en consola de las coordenadas calculadas por el sistema *indoor* – Punto 2.

Fuente: Elaboración propia.

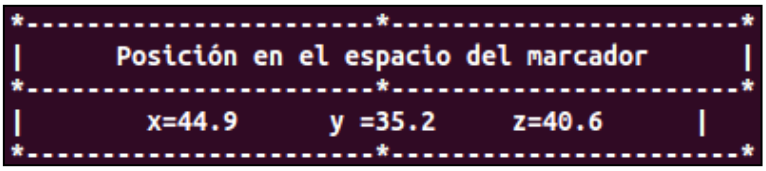

Figura 88. Impresión en consola de las coordenadas calculadas por el sistema *indoor* – Punto 3.

Fuente: Elaboración propia.

Siguiendo este procedimiento, en la Tabla 7 se muestra un cuadro comparativo entre las coordenadas reales y las calculadas de un total de 121 puntos medidos en el entorno, según el apartado **4** del presente capítulo, para validar el correcto cálculo de la posición del marcador.

Tabla 7. Coordenadas reales vs coordenadas calculadas por el sistema *indoor*.

	Coordenadas reales			**Coordenadas calculadas**		
Puntos	**X**	**Y**	**Z**	**X**	**Y**	**Z**
1	50.0	50.0	0.8	50.0	49.6	1.4
2	40.0	50.0	0.8	40.0	49.5	1.1
3	45.0	35.0	0.8	45.3	34.6	0.8
4	60.0	70.0	0.8	59.8	69.3	2.4
5	60.0	50.0	0.8	60.1	49.7	1.5
6	35.0	25.0	0.8	35.1	24.5	0.0
7	30.0	65.0	0.8	29.8	64.6	1.5
8	80.0	50.0	0.8	80.0	49.5	2.2
9	75.0	25.0	0.8	75.7	24.6	1.2
10	20.0	20.0	0.8	19.9	19.2	-0.4
11	35.0	55.0	0.8	35.5	54.6	1.2
12	50.0	50.0	20.8	49.9	49.9	20.9
13	60.0	40.0	20.8	60.0	40.0	20.8
14	60.0	70.0	20.8	59.6	69.4	21.7
15	45.0	65.0	20.8	44.6	64.5	21.2
16	35.0	55.0	20.8	34.6	54.8	20.7
17	45.0	35.0	20.8	45.0	34.9	20.3

	Coordenadas reales			Coordenadas calculadas		
Puntos	**X**	**Y**	**Z**	**X**	**Y**	**Z**
18	60.0	50.0	20.8	59.7	49.9	21.1
19	50.0	25.0	20.8	50.2	24.9	20.1
20	35.0	35.0	20.8	34.7	34.8	20.2
21	40.0	50.0	20.8	39.8	49.9	20.7
22	50.0	50.0	25.8	49.8	50.0	26.0
23	55.0	40.0	25.8	54.9	40.1	25.8
24	45.0	25.0	25.8	45.1	25.1	25.4
25	40.0	50.0	25.8	39.8	50.0	25.9
26	20.0	20.0	25.8	19.6	19.8	24.9
27	25.0	30.0	25.8	24.8	29.9	25.0
28	60.0	70.0	25.8	59.7	69.2	26.6
29	45.0	75.0	25.8	44.6	74.4	26.8
30	50.0	25.0	25.8	50.1	24.9	25.4
31	35.0	35.0	25.8	34.8	34.8	25.4
32	50.0	50.0	30.8	50.0	50.0	31.4
33	45.0	35.0	30.8	45.1	35.1	30.9
34	60.0	40.0	30.8	60.1	40.0	31.3
35	35.0	25.0	30.8	35.0	25.0	30.6
36	25.0	55.0	30.8	24.5	55.0	30.9
37	20.0	60.0	30.8	19.4	59.8	31.0
38	30.0	65.0	30.8	29.3	64.6	31.3
39	25.0	30.0	30.8	24.7	30.0	30.2
40	45.0	25.0	30.8	45.0	24.9	30.4
41	50.0	25.0	30.8	50.0	25.0	30.5
42	50.0	50.0	35.8	49.6	50.0	36.3
43	40.0	50.0	35.8	39.8	49.8	36.2
44	45.0	35.0	35.8	45.1	34.8	35.9
45	60.0	50.0	35.8	59.8	49.7	36.4
46	25.0	30.0	35.8	25.2	29.8	35.5
47	20.0	40.0	35.8	19.9	39.9	35.6
48	60.0	60.0	35.8	59.8	59.6	36.7
49	35.0	35.0	35.8	34.9	34.9	35.8
50	30.0	40.0	35.8	30.1	40.1	35.8
51	25.0	55.0	35.8	25.0	54.9	36.1
52	50.0	50.0	40.8	49.8	50.1	41.3
53	45.0	35.0	40.8	45.0	35.2	41.0
54	35.0	35.0	40.8	34.7	34.9	40.8
55	35.0	25.0	40.8	34.9	25.0	40.6
56	40.0	15.0	40.8	40.3	15.0	40.6
57	60.0	20.0	40.8	60.3	20.1	41.0
58	60.0	30.0	40.8	60.1	30.1	41.1

	Coordenadas reales			Coordenadas calculadas		
Puntos	**X**	**Y**	**Z**	**X**	**Y**	**Z**
59	60.0	60.0	40.8	59.6	59.8	41.7
60	44.0	50.0	40.8	43.8	50.1	41.3
61	55.0	40.0	40.8	54.8	40.2	41.2
62	50.0	50.0	45.8	49.7	49.8	46.3
63	35.0	55.0	45.8	34.6	54.8	46.3
64	25.0	55.0	45.8	24.9	55.2	46.2
65	25.0	50.0	45.8	24.9	50.0	46.1
66	20.0	40.0	45.8	20.0	40.1	45.7
67	30.0	40.0	45.8	29.9	40.3	45.9
68	45.0	35.0	45.8	45.2	35.3	46.0
69	55.0	40.0	45.8	54.9	40.3	46.2
70	50.0	25.0	45.8	50.3	25.3	45.9
71	45.0	50.0	45.8	44.8	50.2	46.3
72	50.0	50.0	50.8	50.1	50.2	51.3
73	40.0	50.0	50.8	40.0	50.2	51.2
74	45.0	35.0	50.8	45.2	35.2	51.0
75	35.0	35.0	50.8	34.7	35.3	50.8
76	20.0	40.0	50.8	20.1	39.7	50.8
77	50.0	25.0	50.8	50.3	25.3	51.0
78	50.0	30.0	50.8	49.8	30.3	51.0
79	45.0	30.0	50.8	44.8	30.2	50.9
80	36.0	42.0	50.8	36.0	42.3	51.0
81	40.0	42.0	50.8	40.1	42.2	51.1
82	45.0	35.0	55.8	45.1	35.2	55.9
83	45.0	25.0	55.8	45.0	25.4	55.8
84	35.0	25.0	55.8	34.7	25.3	55.7
85	20.0	40.0	55.8	19.8	40.3	55.8
86	35.0	44.0	55.8	35.4	44.4	56.0
87	35.0	35.0	55.8	34.6	35.2	55.8
88	45.0	29.0	55.8	45.1	29.3	55.8
89	40.0	29.0	55.8	40.1	29.2	55.8
90	35.0	44.0	55.8	34.6	44.0	56.0
91	30.0	44.0	55.8	29.8	44.3	55.9
92	35.0	35.0	60.8	34.5	34.6	60.8
93	30.0	40.0	60.8	30.0	39.7	60.9
94	20.0	40.0	60.8	19.8	39.6	60.8
95	25.0	30.0	60.8	24.9	29.9	60.7
96	28.0	33.0	60.8	27.5	32.8	60.7
97	37.0	29.0	60.8	36.5	28.7	60.8
98	40.0	41.0	60.8	40.0	41.2	61.0
99	35.0	41.0	60.8	34.9	41.3	61.0

	Coordenadas reales			Coordenadas calculadas		
Puntos	**X**	**Y**	**Z**	**X**	**Y**	**Z**
100	33.0	36.0	60.8	32.9	36.1	60.9
101	32.0	29.0	60.8	32.1	29.7	60.7
102	35.0	35.0	65.8	34.5	35.4	65.9
103	25.0	30.0	65.8	24.9	30.6	65.8
104	24.0	37.0	65.8	23.7	37.3	65.8
105	30.0	35.0	65.8	29.9	35.4	65.9
106	32.0	30.0	65.8	31.9	30.4	65.8
107	32.0	29.0	65.8	31.9	29.4	65.8
108	24.0	34.0	65.8	23.7	34.2	65.8
109	24.0	31.0	65.8	23.9	31.4	65.8
110	23.0	37.0	65.8	22.9	37.1	65.8
111	24.0	35.0	65.8	23.6	35.2	65.8
112	27.0	32.0	70.8	26.9	32.1	70.8
113	28.0	32.0	70.8	28.0	32.2	70.9
114	31.0	32.0	70.8	30.9	32.1	70.9
115	32.0	32.0	70.8	31.9	32.1	70.9
116	33.0	33.0	70.8	32.6	33.1	70.9
117	32.0	34.0	70.8	32.0	33.8	70.9
118	25.0	34.0	70.8	24.9	33.7	70.8
119	26.0	32.0	70.8	26.3	32.0	70.8
120	27.0	32.0	70.8	27.2	32.1	70.8
121	28.0	33.0	70.8	28.3	32.8	70.9

Fuente: Elaboración propia.

Para poder interpretar los resultados que se muestran en la Tabla 7, se reconstruye la trayectoria definida por los puntos calculados y se compara con la trayectoria que definen los puntos reales, de tal modo que la tendencia que presente la trayectoria calculada con respecto a la real es un indicador del correcto funcionamiento del sistema desarrollado. Las reconstrucciones por eje se muestran en las Figuras 89, 90 y 91.

De las gráficas mostradas se puede adelantar que el sistema desarrollado está cumpliendo con el objetivo trazado en la presente tesis; sin embargo, para tener una idea cuantitativa del desempeño del sistema, más adelante se analizan los errores del mismo.

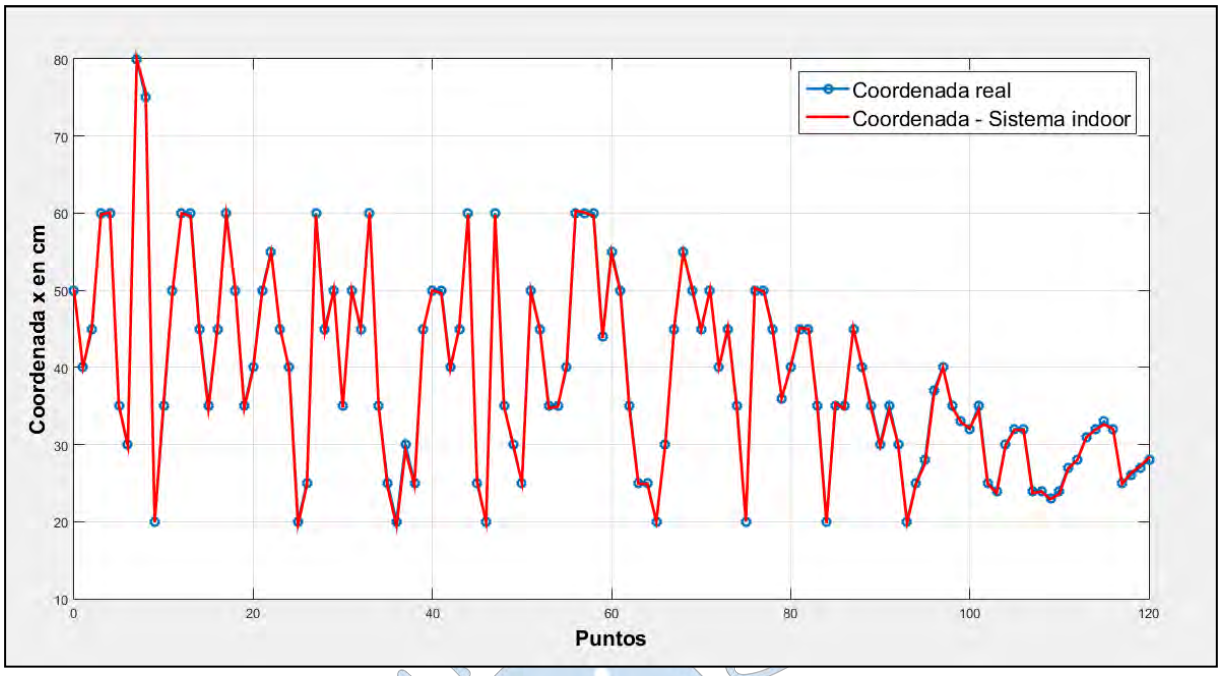

Figura 89. Trayectoria calculada por el sistema vs trayectoria real – Eje X.

Fuente: Elaboración propia.

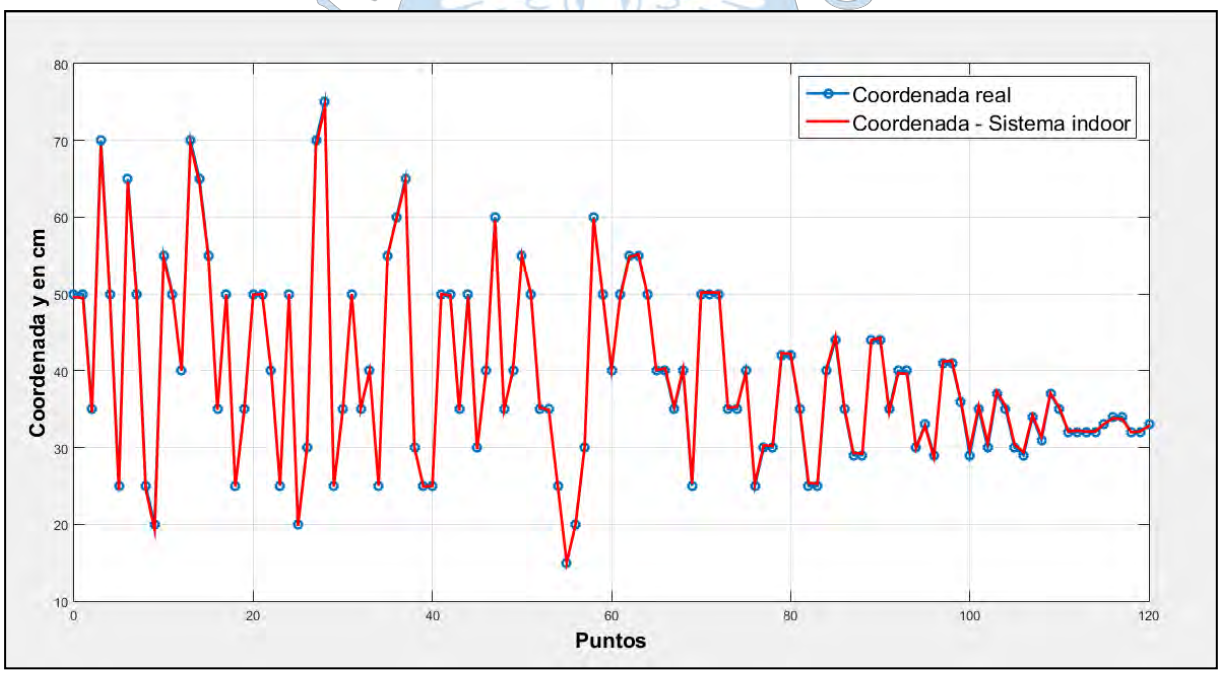

Figura 90. Trayectoria calculada por el sistema vs trayectoria real – Eje Y.

Fuente: Elaboración propia.

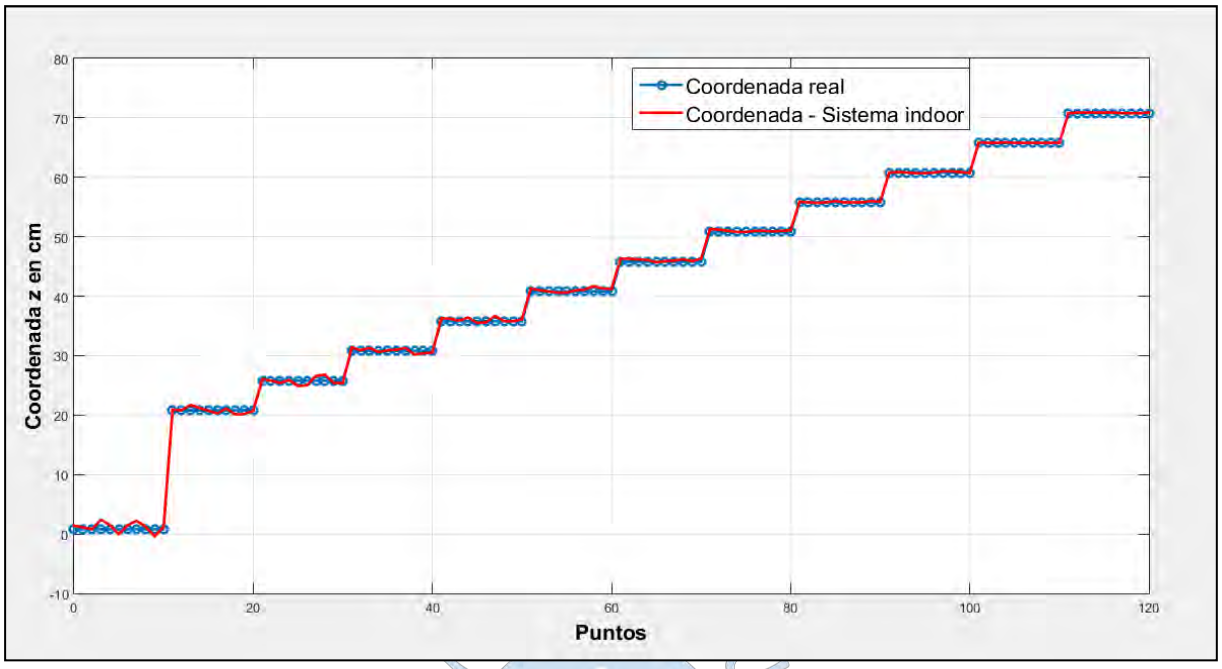

Figura 91. Trayectoria calculada por el sistema vs trayectoria real – Eje Z.

Fuente: Elaboración propia.

5.3. Sistema de reconstrucción de trayectorias. Hasta el momento, la posición del marcador es calculada e impresa en consola; sin embargo, una manera más intuitiva de saber que se está calculando correctamente la posición del marcador es reconstruyendo la trayectoria que sigue. Por ello, este tercer nodo al ser ejecutado tiene como salida una interfaz de reconstrucción de trayectorias, la cual se muestra en la Figura 92.

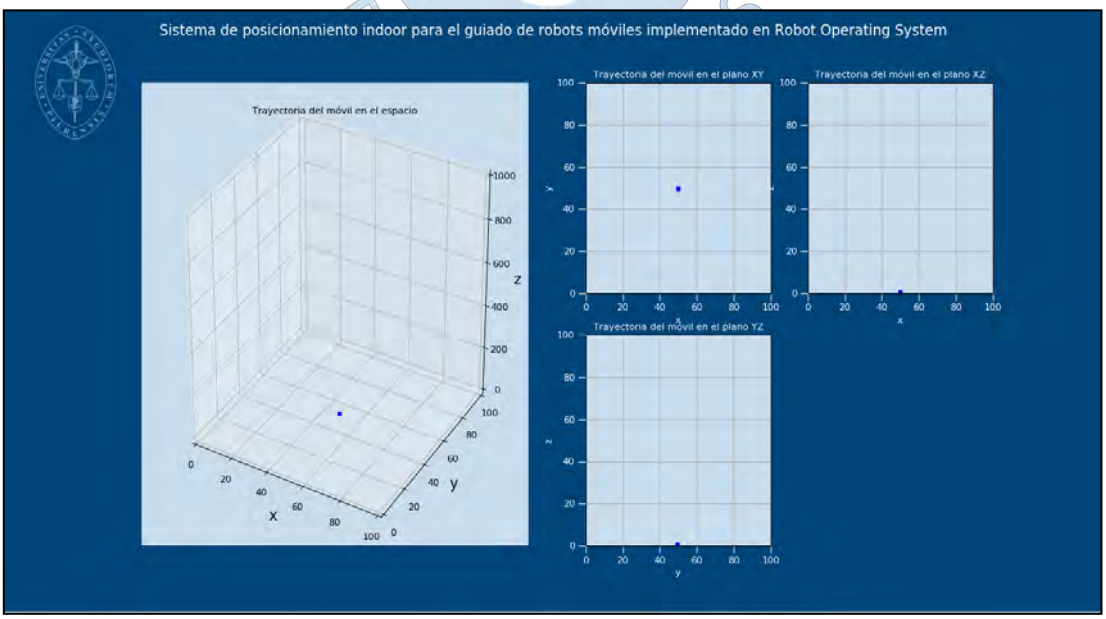

Figura 92. Interfaz gráfica de reconstrucción de trayectorias.

Fuente: Elaboración propia.

En esta interfaz se reconstruye la trayectoria que sigue el marcador tanto en plano tridimensional, que simula al entorno de pruebas, como en los planos **XY**, **YZ** y **XZ**. A modo de prueba, en las Figuras 93, 94 y 95 son reconstruidas tres trayectorias que son el resultado de generar una trayectoria a mano alzada dentro del entorno de pruebas.

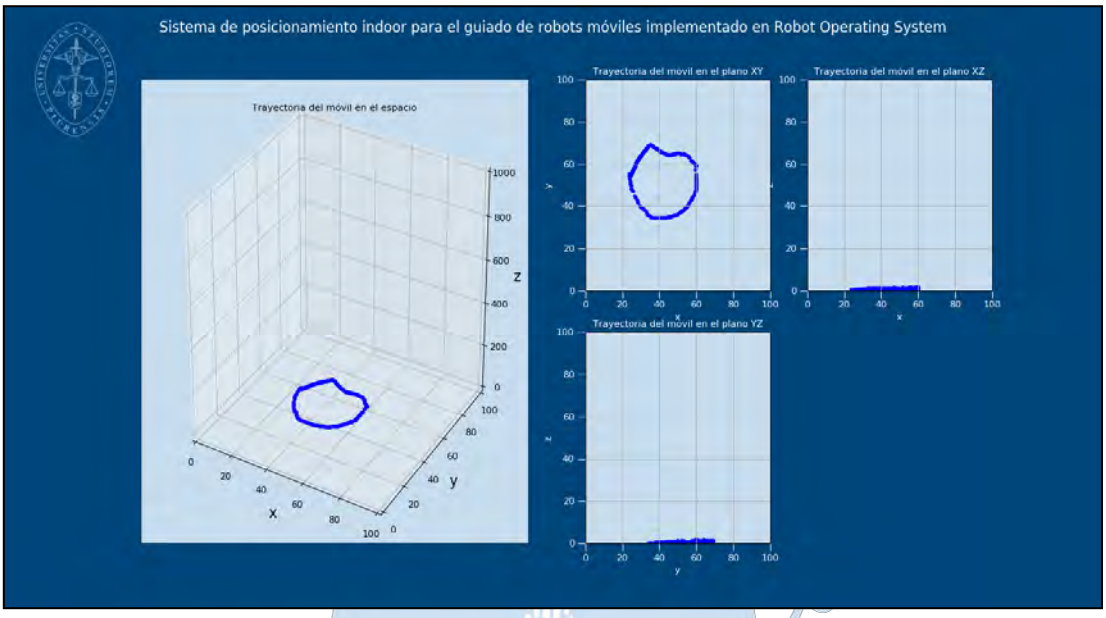

Figura 93. Trayectoria a mano alzada reconstruida.

Fuente: Elaboración propia.

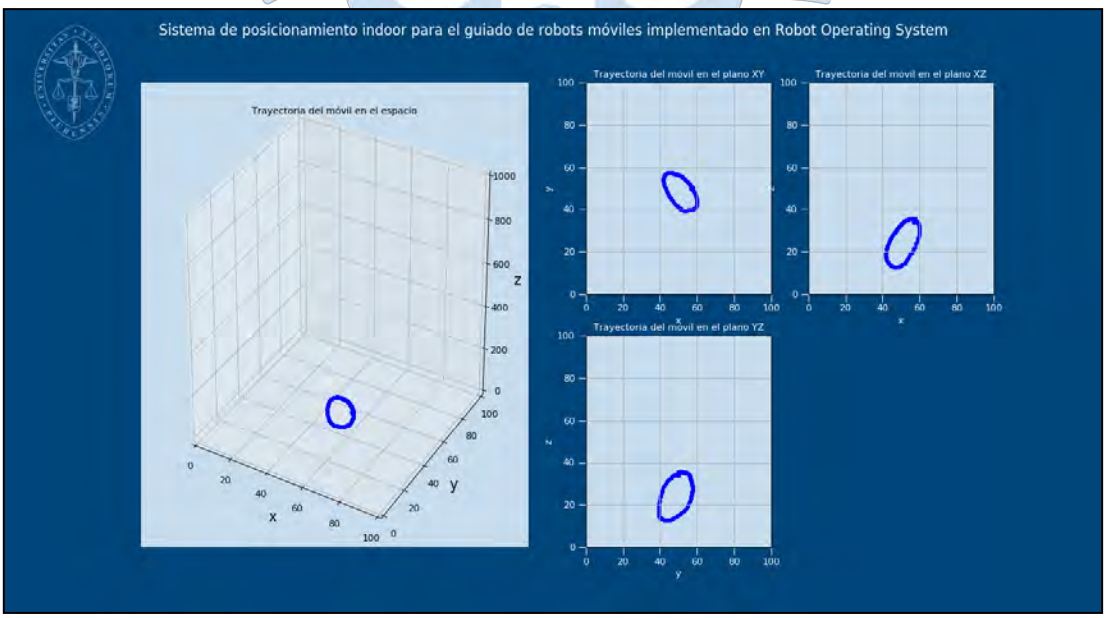

Figura 94. Trayectoria a mano alzada reconstruida.

Fuente: Elaboración propia.

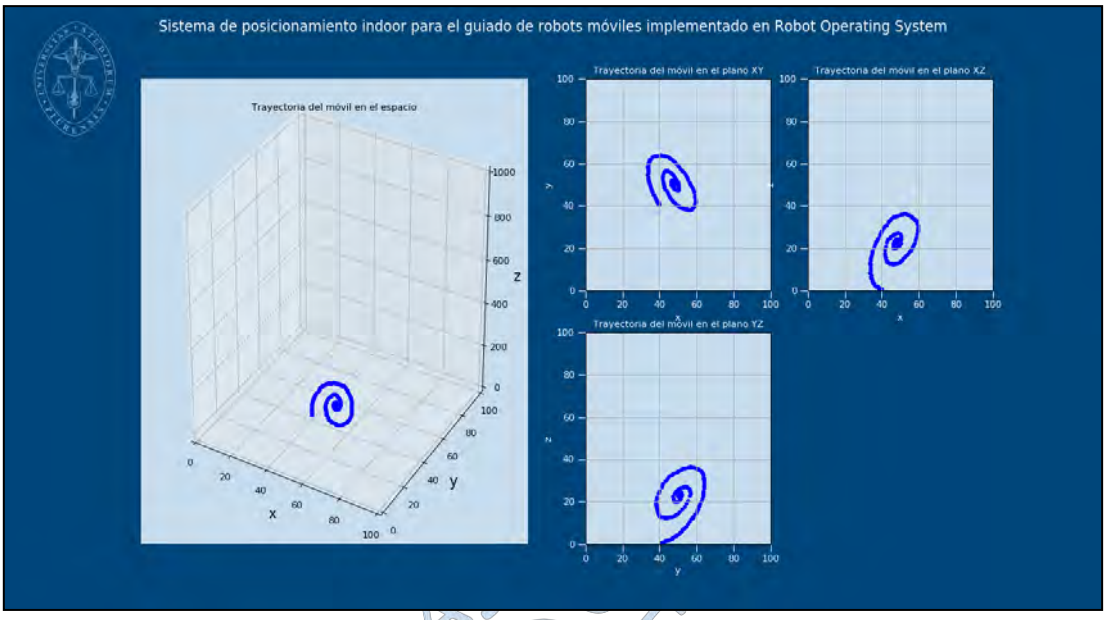

Figura 95. Trayectoria a mano alzada reconstruida.

Fuente: Elaboración propia.

6. Análisis de errores

En este último apartado se calculan y analizan los errores del sistema de posicionamiento en los tres ejes, los cuales son resumidos en la Tabla 8.

6.1. Error en el eje X. En la Figura 96 se muestra el histograma del error de posicionamiento en el eje **X** para 121 puntos reconstruidos cuyos valores se encuentran en el intervalo de $[-0.8, 0.8]$ medidos en cm. Así mismo, para una distribución normal, el error medio es de 0.09 cm y la desviación estándar es de 0.23675 cm; por lo tanto, el 99.7% de los errores se encuentran en el rango de [-0.616035, 0.804465] medidos en cm. Dicho esto, se puede adelantar a concluir que el error máximo en el eje **X** es de 0.8 cm, con un error absoluto medio de 0.2 cm.

6.2. Error en el eje Y. En la Figura 97 se muestra el histograma del error de posicionamiento en el eje **Y** para 121 puntos reconstruidos cuyos valores se encuentran en el intervalo de $[-0.8, 0.8]$ medidos en cm. Así mismo, para una distribución normal, el error medio es de 0.02 cm y la desviación estándar es de 0.29098 cm; por lo tanto, el 99.7% de los errores se encuentran en el rango de [-0.857238, 0.888642] medidos en cm. Dicho esto se puede adelantar a concluir que el error máximo en el eje **Y** es de 0.9 cm, con un error absoluto medo de 0.2 cm.

6.3. Error en el eje Z. En la Figura 98 se muestra el histograma del error de posicionamiento en el eje **Z** para 121 puntos reconstruidos cuyos valores se encuentran en el intervalo de [−2 ,1.5] medidos en cm. Así mismo, para una distribución normal, el error medio es de -0.12 cm y la desviación estándar es de 0.40436 cm; por lo tanto, el 99.7% de los errores se encuentran en el rango de [-1.33705, 1.08911] medidos en cm. Dicho esto, se puede adelantar a concluir que el error máximo en el eje **Z** es de 1 cm, con un error absoluto medio de 0.3 cm.

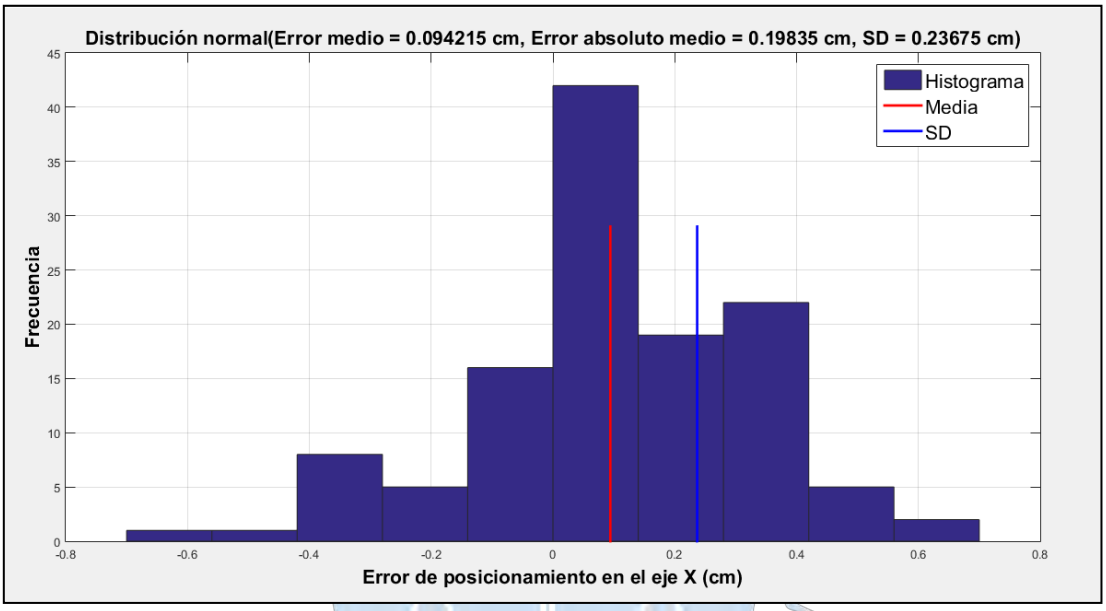

Figura 96. Histograma del error en el eje X en una muestra de 121 puntos del entorno.

Fuente: Elaboración propia.

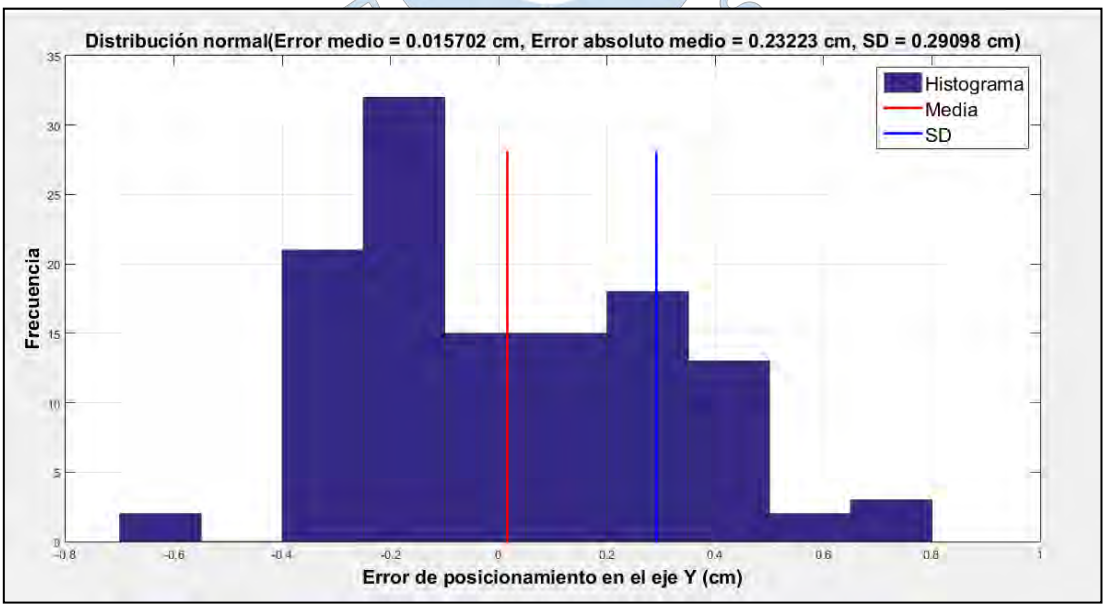

Figura 97. Histograma del error en el eje Y en una muestra de 121 puntos del entorno.

Fuente: Elaboración propia.

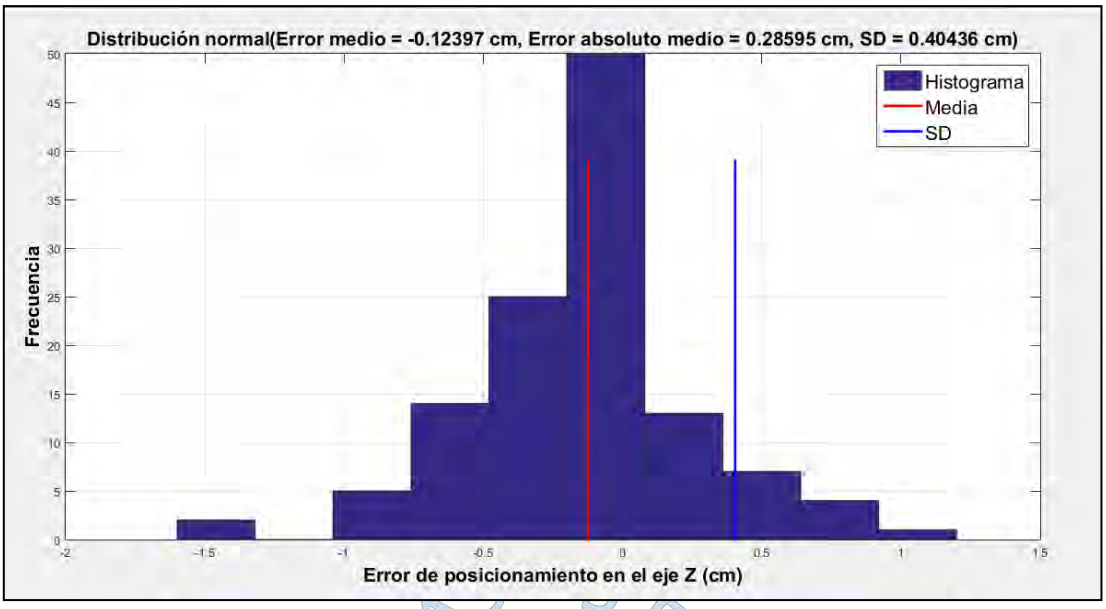

Figura 98. Histograma del error en el eje Z en una muestra de 121 puntos del entorno.

Fuente: Elaboración propia.

Tabla 8. Errores en los tres ejes del sistema de posicionamiento *indoor* desarrollado.

	Error máximo	**Error medio**	**Error medio absoluto**
X	0.8 cm	0.09 cm	0.2 cm
Y	0.9 cm	0.02 cm	0.2 cm
Z	1 cm	-0.12 cm	0.3 cm

Fuente: Elaboración propia.

Conclusiones

1. Se ha desarrollado un sistema de posicionamiento *indoor* que cumple con el objetivo de la presente tesis, siendo capaz de posicionar objetos en el espacio a través de marcadores.

2. El sistema permite reconstruir cualquier trayectoria seguida por un objeto en un determinado entorno de trabajo, tal como se muestra en las Figuras 93, 94, 95 del apartado **5.3** del Capítulo 5.

3. Los errores de posicionamiento en los tres ejes, los cuales se muestran en la Tabla 8, son aceptables. Por lo tanto, es posible usar al sistema como sensor en un fututo trabajo de control de trayectorias para robot móviles.

4. Los factores causantes del error en el cálculo de la posición son los siguientes:

- Consideración de parámetros aproximados de posición y rotación de las cámaras con respecto al origen de coordenadas del entorno de pruebas, puesto que forman parte de la formulación matemática para el cálculo de la posición del marcador.

- Distorsión radial y tangencial no considerada en la formulación matemática del apartado **5.3** del Capítulo 3; en consecuencia, el error aumenta a medida que el marcador se aleja del centro del plano imagen.

5. El entorno de pruebas es limitado debido a que el sistema está sujeto al rango visual de las cámaras, imposibilitando el cálculo de la posición del marcador en todos los puntos del entorno. Por ello, es necesario adicionar cámaras que cubran los puntos ciegos.

6. El sistema de detección responde según lo esperado, sin importar los cambios de iluminación del ambiente.

7. Debido a la velocidad de captura de las cámaras (30 fps), no es posible reconstruir completamente la trayectoria del marcador si va a alta velocidad.

8. A partir de la presente tesis se plantea el desarrollo de los siguientes trabajos:

- Estudio biomecánico de la marcha bípeda para el desarrollo de exoesqueletos mediante un sistema de captura de movimiento.

- Diseño e implementación de un sistema de control de trayectorias para vehículos aéreos o terrestres.

- Propuesta de un sistema de reconstrucción geomorfológica.

- Desarrollo y puesta en marcha de un escáner para la reconstrucción de piezas mecánicas o partes biomédicas.

Recomendaciones

Para mejoras en el sistema de posicionamiento desarrollado en la presente tesis y con otros fines de investigación, se recomienda lo siguiente:

- Contar con un entorno de pruebas de mayores dimensiones.

- Usar iluminación infrarroja en lugar de iluminación LED, de tal modo que sea imperceptible para las personas.

- Aumentar el número de cámaras con el fin de no dejar puntos ciegos en el entorno.

- Trabajar con cámaras que ofrezcan una mayor velocidad de captura.

- Considerar en la formulación matemática para el cálculo de la posición, la distorsión radial y tangencial del lente óptico.


UNIVERSITAS STUDIORUM PIURENSIS

Referencias Bibliográficas

Achmad, H., Priyandoko, G., & Daud, M. (2017). Tele-Operated Mobile Robot for 3D Visual Inspection Utilizing Distributed Operating System Platform. *International Journal of Vehicle Structures & Systems, 9*(3). doi:http://dx.doi.org/10.4273/ijvss.9.3.12

AER. (n.d.). Retrieved Agosto 14, 2019, from https://www.aer-automation.com/mercados-emergentes/robotica-de-servicio/

amazon. (n.d.). Retrieved Agosto 14, 2019, from https://www.amazon.es/Lenovo-Z50-70-Port%C3%A1til-I5-4210U-Windows/dp/B00Q88Q3X6

Barranco Gutiérrez, A. I., Martínez Díaz, S., & Gómez Torres, J. L. (2018). *Visión estereoscópica por computadora con Matlab y OpenCV*. (Lulu.com, Ed.)

Blog CARTIF. (2016, Setiembre 29). Retrieved Agosto 13, 2019, from https://blog.cartif.com/es/sistemas-de-geolocalizacion-en-interiores/

Bustamante Mejia, J., & López Varona, R. (2014, Marzo 1). Calibración de Cámara Termográfica Fluke TI-32. *Scientia et Technica, 19*(1), 59-66. doi:http://dx.doi.org/10.22517/23447214.8857

Calvillo Ardila, J. A. (2017). *Posicionamiento en interiores de un dron por método multicámara.* Trabajo fin de máster, Universitat Oberta de Catalunya.

Cliement, M. (2015, Enero 22). *El mundo*. Retrieved Agosto 12, 2019, from https://www.elmundo.es/economia/2015/01/22/54bff268268e3efa718b4583.html

Cuevas Castañeda, C. C. (2016). Ros-gazebo. una valiosa Herramienta de Vanguardia para el Desarrollo de la Robótica. *Journal specializing in engineering, 10*. doi:https://doi.org/10.22490/25394088.1593

ETH Zurich. (n.d.). *Institute for Dynamic Systems and Control*. Retrieved from Research D'Andrea: https://idsc.ethz.ch/research-dandrea.html

Faugeras, O. D., & Toscani, G. (1987). Camera Calibration for 3D Computer Vision. *l Workshop on Industrial Applications of Machine Vision and Machine Intelligence: Seiken Symposium*, *1*, pp. 240-247. Tokyo.

Fernández Lorenzo, I. (2005). *Sistema de posicionamiento absoluto de un robot móvil utilizando cámaras externas.* Tesis doctoral, Universidad de Alcalá, Departamento de Electrónica.

García, L., Pozo Ruz, A., Ribeiro, A., Sandoval, F., García Alegre, M., & Guinea, G. (2000). Sistema GPS: Descripción, análisis de errores, aplicaciones y futuro. *Mundo electrónico*(306), 54-59.

Geeky Theory. (n.d.). *Geeky Theory*. Retrieved Agosto 14, 2019, from https://geekytheory.com/opencv-en-linux

González Díaz, R., & Real Jurado, P. (2003). *Morfología y Análisis de Imágenes Digitales*. Retrieved Agosto 14, 2019, from http://grupo.us.es/gtocoma/pid/morfologia/documentacion/morfologia.htm

Guerrero Hernández, J. M., & Pajares Martinsanz, G. (2011). Técnicas de procesamiento de imágenes estereoscópicas. *CES Felipe II*(13).

Heikkilä, J., & Silvén, O. (1997). A four-step camera ca!ibration procedure with. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2*, pp. 1106-1112. doi:10.1109/CVPR.1997.609468

IDS Imaging Development Systems GmbH. (n.d.). *iDS*. Retrieved Agosto 14, 2019, from https://es.ids-imaging.com/technical-article/es_tech-article-linescan-mode.html

INFAIMON. (n.d.). Retrieved Febrero 06, 2019, from https://www.infaimon.com/enciclopedia-de-la-vision/

Joo-Ho, L., & Hiroshi, H. (2002). Intelligent Space - concept and contents. *Advanced Robotics, 16*(3), 265-280. doi:10.1163/156855302760121936

Losada Gutiérrez, C. (2010). *Segmentación y posicionamiento 3D de robots móviles en espacios inteligentes mediante redes de cámaras fijas.* Tesis doctoral, Universidad de Alcalá, Departamento de Electrónica, Madrid.

Losada Gutiérrez, C., Mazo, M., Palazuelos, S., Pizarro, D., & Marron, M. (2011). Posicionamiento 3D de robots móviles en un espacio inteligente mediante cámaras fijas. *XVIII Seminario Anual de Automática y Electrónica Industrial (SAAEI 2011)*, (pp. 783-788).

Marenco Jiménez, M. D. (2009). *Desarrollo y puesta en funcionamiento de una herramienta avanzada de segmentación aplicada a imágenes de ecografía.* Trabajo fin de grado, Universidad de Sevilla.

Marín Abrego, C. (n.d.). *Tutor de Programación*. Retrieved Agosto 14, 2019, from http://acodigo.blogspot.com/2016/04/seguimiento-de-objetos-por-color.html

Martín, B., Melcón, A., & Tapia, D. (2009). *Identificación óptica de la posición y orientación de un vehículo aéreo no tripulado.* Trabajo de curso, Universidad Complutense de Madrid.

Martínez Novo, J. (2015). *Sistema de posicionamiento para un drone mediante odometría visual.* Trabajo de fin de grado.

Melen, T. (1994). *Geometrical modelling and calibration of video cameras for underwater navigation.* Doctoral dissertation, Norwegian University of Science and Technology, Trondheim.

Microsoft. (n.d.). Retrieved Agosto 14, 2019, from https://www.microsoft.com/accessories/es-es/products/webcams/lifecam-studio/q2f-00009#specsColumns-testCarousel

Open Source Robotics Foundation. (2014, Julio 21). *ROS.org*. Retrieved Agosto 18, 2019, from ROS/Concepts: http://wiki.ros.org/ROS/Concepts

Open Source Robotics Foundation. (2015, Agosto 13). *ROS.org*. Retrieved Agosto 18, 2019, from ROS/CommandLineTools: http://wiki.ros.org/ROS/CommandLineTools

Open Source Robotics Foundation. (2019, Julio 16). *ROS.org*. Retrieved Agosto 18, 2019, from ROSTutorials: http://wiki.ros.org/ROS/Tutorials

Open Source Robotics Foundation. (n.d.). *About ROS*. Retrieved Agosto 18, 2019, from http://www.ros.org/about-ros/

Open Source Robotics Foundation. (n.d.). *ROS.org*. Retrieved Agosto 14, 2019, from https://wiki.ros.org/Distributions

Opencv dev team. (2019, Agosto 14). *OpenCV documentation*. Retrieved from Eroding and Dilating: https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

Opencv dev team. (n.d.). *OpenCV documentation*. Retrieved Agosto 14, 2019, from Operations on Arrays: https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html

Opencv dev team. (n.d.). *OpenCV documentation*. Retrieved Agosto 14, 2019, from Structural Analysis and Shape Descriptors: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

OpenCV team. (n.d.). *OpenCV*. Retrieved Agosto 14, 2019, from https://opencv.org/

OptiTrack. (n.d.). *OptiTrack*. Retrieved Agosto 14, 2019, from Markers: https://optitrack.com/products/motion-capture-markers/

Pérez González, C. (2016). *Detección y seguimiento de objetos por colores en una plataforma Raspberry Pi.* Trabajo fin de grado, Universidad Politécnica de Madrid, Automática, Ingeniería Eléctrica y Electrónica e Informática Industrial.

Pizarro, D., Santiso, E., & Mazo, M. (2006). Localizacion Simultanea a la Reconstruccion de Robots Moviles en Espacios Inteligentes mediante Multiples Camaras. *Seminario Anual de Automatica, Electronica Industrial e Instrumentacion (SAAEI´06)*, *1*, pp. 1-6. Gijón.

Platero, C. (2008). *Robótica y Visión Artificial*. Retrieved Agosto 14, 2019, from http://www.elai.upm.es/webantigua/spain/Asignaturas/Robotica/InfoRobotica.htm

Prieto, F. (2014). *Geometría y parámetros de las cámaras - Visión estéreo.* Universidad Nacional de Colombia.

Procesamiento De Imágenes en OpenCV. (n.d.). Retrieved Agosto 14, 2019, from https://sites.google.com/site/cg05procesamientodeimagenes/home/threshold-umbralizacion

Ramer, U. (1972, Noviembre). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing, 1*(3), 244-256. doi:10.1016/S0146-664X(72)80017-0

Robologs. (2015, Enero 25). Retrieved Agosto 14, 2019, from Detección de triángulos con OpenCV y Python: https://robologs.net/2015/01/25/deteccion-de-triangulos-con-opencv-y-python/

Rodríguez Bazaga, A. (n.d.). *Oficina de Software Libre* . Retrieved Febrero 22, 2019, from Universidad de La Laguna: https://osl.ull.es/software-libre/opencv-libreria-vision-computador/

Sánchez Pérez, J. (n.d.). *Visión tridimensional.* Curso de doctorado, Universidad de Las Palmas de Gran Canaria, Departamento de Informática y Sistemas.

Santiago Pé, A. (2007). *Sistema de detección de objetos mediante cámaras. Aplicación en Espacios Inteligentes.* Trabajo fin de máster, Universidad de Alcalá, Departamento de Electrónica, Alcalá de Henares.

Shermal, F. (n.d.). *OpenCV Tutorial C++*. Retrieved Agosto 14, 2019, from https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html

Sierra Álvarez, S. (2012). *Sistema de medición de objetos basado en visión artificial.* Trabajo fin de grado, Universidad EAFIT, Medellín.

Teh, C.-H., & Chin, R. (1989, Agosto). On the Detection of Dominant Points. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 11*(8), 859-872. doi:10.1109/34.31447

Visión Artificial. (2012, Febrero). Retrieved Agosto 14, 2019, from http://visionartificial.fpcat.cat/prototipos-didacticos/control-de-estampacion-en-frio/unidades-didacticas/

Wan Mahani , A., & Shahrul Nizam, Y. (2017, Abril). Modified excess green vegetation index for uneven illumination. *International Journal of Current Research, 9*(4), 48656-48661.

Yun, J. Y., Park, J. W., Choi, H. S., & Lee, J. M. (2004). Absolute Positioning System for Mobile Robot Navigation in an Indoor Environment ., (pp. 1448-1451). Bangkok.

Zhang, Z. (1998). *A Flexible New Technique for Camera.* Technical Report. Retrieved from http://research.microsoft.com/~zhang


UNIVERSITAS STUDIORUM PIURENSIS

Apéndices


UNIVERSITAS STUDIORUM PIURENSIS

Apéndice A. Archivos de cabecera

A.1. Archivo de cabecera pixeles.h

```
// Generated by gencpp from file tesis_jayro_zaid_paiva_mimbela/pixeles.msg
// DO NOT EDIT!

#ifndef TESIS_JAYRO_ZAID_PAIVA_MIMBELA_MESSAGE_PIXELES_H
#define TESIS_JAYRO_ZAID_PAIVA_MIMBELA_MESSAGE_PIXELES_H

#include <string>
#include <vector>
#include <map>

#include <ros/types.h>
#include <ros/serialization.h>
#include <ros/builtin_message_traits.h>
#include <ros/message_operations.h>

namespace tesis_jayro_zaid_paiva_mimbela
{
template <class ContainerAllocator>
struct pixeles_
{
 typedef pixeles_<ContainerAllocator> Type;

 pixeles_()
   : u(0.0)
   , v(0.0)
   , up(0.0)
   , vp(0.0)  {
   }

 pixeles_(const ContainerAllocator& _alloc)
   : u(0.0)
   , v(0.0)
   , up(0.0)
   , vp(0.0)  {
(void)_alloc;
   }

  typedef double _u_type;
 _u_type u;

  typedef double _v_type;
 _v_type v;

  typedef double _up_type;
```

```
  _up_type up;

   typedef double _vp_type;
  _vp_type vp;

  typedef boost::shared_ptr<
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> > Ptr;

  typedef boost::shared_ptr<
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> const> ConstPtr;

}; // struct pixeles_

typedef ::tesis_jayro_zaid_paiva_mimbela::pixeles_<std::allocator<void> > pixeles;

typedef boost::shared_ptr< ::tesis_jayro_zaid_paiva_mimbela::pixeles > pixelesPtr;
typedef boost::shared_ptr< ::tesis_jayro_zaid_paiva_mimbela::pixeles const>
pixelesConstPtr;

// constants requiring out of line definition

template<typename ContainerAllocator>
std::ostream& operator<<(std::ostream& s, const
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> & v)
{
ros::message_operations::Printer<
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >::stream(s, "", v);
return s;
}

} // namespace tesis_jayro_zaid_paiva_mimbela

namespace ros
{
namespace message_traits
{

// BOOLTRAITS {'IsFixedSize': True, 'IsMessage': True, 'HasHeader': False}
// {'tesis_jayro_zaid_paiva_mimbela':
['/home/jayro_zaid/tesis/src/tesis_jayro_zaid_paiva_mimbela/msg'], 'std_msgs':
['/opt/ros/indigo/share/std_msgs/cmake/../msg']}

// !!!!!!!!!!! ['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
'__getattribute__', '__hash__', '__init__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', '_parsed_fields', 'constants', 'fields', 'full_name', 'has_header', 'header_present',
'names', 'package', 'parsed_fields', 'short_name', 'text', 'types']

template <class ContainerAllocator>
struct IsFixedSize< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
```

```
  : TrueType
  { };

template <class ContainerAllocator>
struct IsFixedSize< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> const>
  : TrueType
  { };

template <class ContainerAllocator>
struct IsMessage< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
  : TrueType
  { };

template <class ContainerAllocator>
struct IsMessage< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> const>
  : TrueType
  { };

template <class ContainerAllocator>
struct HasHeader< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
  : FalseType
  { };

template <class ContainerAllocator>
struct HasHeader< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> const>
  : FalseType
  { };

template<class ContainerAllocator>
struct MD5Sum< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
{
  static const char* value()
  {
    return "855eac937cbc33bbd15e647835614b0e";
  }

  static const char* value(const
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator>&) { return value(); }
  static const uint64_t static_value1 = 0x855eac937cbc33bbULL;
  static const uint64_t static_value2 = 0xd15e647835614b0eULL;
};

template<class ContainerAllocator>
struct DataType< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
{
  static const char* value()
  {
    return "tesis_jayro_zaid_paiva_mimbela/pixeles";
  }
```

```
  static const char* value(const
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator>&) { return value(); }
};

template<class ContainerAllocator>
struct Definition< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
{
  static const char* value()
  {
    return "float64 u\n\
float64 v\n\
float64 up\n\
float64 vp\n\
\n\
\n\
\n\
\n\
";
  }

  static const char* value(const
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator>&) { return value(); }
};

} // namespace message_traits
} // namespace ros

namespace ros
{
namespace serialization
{

  template<class ContainerAllocator> struct Serializer<
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
  {
    template<typename Stream, typename T> inline static void allInOne(Stream& stream, T m)
    {
      stream.next(m.u);
      stream.next(m.v);
      stream.next(m.up);
      stream.next(m.vp);
    }
    ROS_DECLARE_ALLINONE_SERIALIZER
  }; // struct pixeles_

} // namespace serialization
} // namespace ros

namespace ros
{
```

```
namespace message_operations
{
template<class ContainerAllocator>
struct Printer< ::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator> >
{
  template<typename Stream> static void stream(Stream& s, const std::string& indent, const
::tesis_jayro_zaid_paiva_mimbela::pixeles_<ContainerAllocator>& v)
  {
    s << indent << "u: ";
    Printer<double>::stream(s, indent + "  ", v.u);
    s << indent << "v: ";
    Printer<double>::stream(s, indent + "  ", v.v);
    s << indent << "up: ";
    Printer<double>::stream(s, indent + "  ", v.up);
    s << indent << "vp: ";
    Printer<double>::stream(s, indent + "  ", v.vp);
  }
};

} // namespace message_operations
} // namespace ros

#endif // TESIS_JAYRO_ZAID_PAIVA_MIMBELA_MESSAGE_PIXELES_H
```

A.2. Archivo de cabecera espaciales.h

```
// Generated by gencpp from file tesis_jayro_zaid_paiva_mimbela/espaciales.msg
// DO NOT EDIT!

#ifndef TESIS_JAYRO_ZAID_PAIVA_MIMBELA_MESSAGE_ESPACIALES_H
#define TESIS_JAYRO_ZAID_PAIVA_MIMBELA_MESSAGE_ESPACIALES_H

#include <string>
#include <vector>
#include <map>

#include <ros/types.h>
#include <ros/serialization.h>
#include <ros/builtin_message_traits.h>
#include <ros/message_operations.h>

namespace tesis_jayro_zaid_paiva_mimbela
{
template <class ContainerAllocator>
struct espaciales_
{
  typedef espaciales_<ContainerAllocator> Type;

  espaciales_()
```

```
   : x(0.0)
   , y(0.0)
   , z(0.0)  {
   }
  espaciales_(const ContainerAllocator& _alloc)
   : x(0.0)
   , y(0.0)
   , z(0.0)  {
  (void)_alloc;
   }

   typedef double _x_type;
  _x_type x;

   typedef double _y_type;
  _y_type y;

   typedef double _z_type;
  _z_type z;

  typedef boost::shared_ptr<
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> > Ptr;
  typedef boost::shared_ptr<
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> const> ConstPtr;

}; // struct espaciales_

typedef ::tesis_jayro_zaid_paiva_mimbela::espaciales_<std::allocator<void> > espaciales;

typedef boost::shared_ptr< ::tesis_jayro_zaid_paiva_mimbela::espaciales > espacialesPtr;
typedef boost::shared_ptr< ::tesis_jayro_zaid_paiva_mimbela::espaciales const>
espacialesConstPtr;

// constants requiring out of line definition

template<typename ContainerAllocator>
std::ostream& operator<<(std::ostream& s, const
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> & v)
{
ros::message_operations::Printer<
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >::stream(s, "", v);
return s;
}

} // namespace tesis_jayro_zaid_paiva_mimbela

namespace ros
{
namespace message_traits
{
```

```
// BOOLTRAITS {'IsFixedSize': True, 'IsMessage': True, 'HasHeader': False}
// {'tesis_jayro_zaid_paiva_mimbela':
['/home/jayro_zaid/tesis/src/tesis_jayro_zaid_paiva_mimbela/msg'], 'std_msgs':
['/opt/ros/indigo/share/std_msgs/cmake/../msg']}

// !!!!!!!!!!! ['__class__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__',
'__getattribute__', '__hash__', '__init__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', '_parsed_fields', 'constants', 'fields', 'full_name', 'has_header', 'header_present',
'names', 'package', 'parsed_fields', 'short_name', 'text', 'types']

template <class ContainerAllocator>
struct IsFixedSize< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
  : TrueType
  { };

template <class ContainerAllocator>
struct IsFixedSize< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>
const>
  : TrueType
  { };

template <class ContainerAllocator>
struct IsMessage< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
  : TrueType
  { };

template <class ContainerAllocator>
struct IsMessage< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>
const>
  : TrueType
  { };

template <class ContainerAllocator>
struct HasHeader< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
  : FalseType
  { };

template <class ContainerAllocator>
struct HasHeader< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>
const>
  : FalseType
  { };

template<class ContainerAllocator>
struct MD5Sum< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
{
  static const char* value()
  {
    return "4a842b65f413084dc2b10fb484ea7f17";
```

```
  }

  static const char* value(const
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>&) { return value(); }
  static const uint64_t static_value1 = 0x4a842b65f413084dULL;
  static const uint64_t static_value2 = 0xc2b10fb484ea7f17ULL;
};

template<class ContainerAllocator>
struct DataType< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
{
  static const char* value()
  {
    return "tesis_jayro_zaid_paiva_mimbela/espaciales";
  }

  static const char* value(const
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>&) { return value(); }
};

template<class ContainerAllocator>
struct Definition< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
{
  static const char* value()
  {
    return "float64 x\n\
float64 y\n\
float64 z\n\
";
  }

  static const char* value(const
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>&) { return value(); }
};

} // namespace message_traits
} // namespace ros

namespace ros
{
namespace serialization
{

  template<class ContainerAllocator> struct Serializer<
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
  {
    template<typename Stream, typename T> inline static void allInOne(Stream& stream, T m)
    {
      stream.next(m.x);
      stream.next(m.y);
```

```
    stream.next(m.z);
  }

  ROS_DECLARE_ALLINONE_SERIALIZER
 }; // struct espaciales_

} // namespace serialization
} // namespace ros

namespace ros
{
namespace message_operations
{

template<class ContainerAllocator>
struct Printer< ::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator> >
{
 template<typename Stream> static void stream(Stream& s, const std::string& indent, const
::tesis_jayro_zaid_paiva_mimbela::espaciales_<ContainerAllocator>& v)
 {
  s << indent << "x: ";
  Printer<double>::stream(s, indent + "  ", v.x);
  s << indent << "y: ";
  Printer<double>::stream(s, indent + "  ", v.y);
  s << indent << "z: ";
  Printer<double>::stream(s, indent + "  ", v.z);
 }
};

} // namespace message_operations
} // namespace ros

#endif // TESIS_JAYRO_ZAID_PAIVA_MIMBELA_MESSAGE_ESPACIALES_H
```


UNIVERSITAS STUDIORUM PIURENSIS

Apéndice B. Nodos

B.1. Nodo 1 – Sistema de detección y seguimiento

```
/*
TESIS: "Sistema de posicionamiento indoor para el guiado de robots móviles implementado en Robot Operating System (ROS)"
Autor: Jayro Zaid Paiva Mimbela
UNIVERSIDAD DE PIURA-2019
*/

//Librerías incluidas
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "ros/ros.h"
#include "tesis_jayro_zaid_paiva_mimbela/pixeles.h"
#include <stdio.h>

using namespace cv;
using namespace ros;
using namespace std;
using namespace tesis_jayro_zaid_paiva_mimbela;

//Declaración de variables
//Imagen Original
Mat Imgc1, Imgc2;
//Imagen en rango HSV
Mat hsvImgc1, hsvImgc2;
//Imagen binaria
Mat binImgc1, binImgc2;
//Coordenadas en píxeles con respecto a cada cámara
float u,v,up,vp;

int main(int argc, char** argv)
{

    init(argc, argv, "sistema_deteccion_seguimiento");
    NodeHandle n;
    Publisher pub = n.advertise<pixeles>("coordenadas_pixeles",10);

    VideoCapture camara_01(0);
        if(camara_01.isOpened()==false)
        {
            printf("ERROR: Falló la conexión de la cámara 01\n");
            return 1;
        }
```

```
    VideoCapture camara_02(1);
        if(camara_02.isOpened()==false)
        {
            printf("ERROR: Falló la conexión de la cámara 02\n");
            return 1;
        }

    // Configuración H
    int lowHc1 = 0, lowHc2 = 0;
    int highHc1 = 180, highHc2 = 180;
    // Configuración S
    int lowSc1 = 0, lowSc2 = 0;
    int highSc1 = 255, highSc2 = 255;
    // Configuración V
    int lowVc1 = 234, lowVc2 = 234;
    int highVc1 = 255, highVc2 = 255;

    while(ok() && camara_01.isOpened() && camara_02.isOpened())
    {
        bool blnFrameReadSuccessfullyc1 = camara_01.read(Imgc1);
        bool blnFrameReadSuccessfullyc2 = camara_02.read(Imgc2);

        if (!blnFrameReadSuccessfullyc1 || Imgc1.empty())
        {
            printf("ERROR: No se pueden leer los frames de la cámara 01\n");
            return 1;
        }

        if (!blnFrameReadSuccessfullyc2 || Imgc2.empty())
        {
            printf("ERROR: No se pueden los leer frames de la cámara 02\n");
            return 1;
        }

        Mat element = getStructuringElement(MORPH_RECT, Size(3,3));

        //Transición de espacio de color de BGR a HSV - Cámara 01
        cvtColor(Imgc1, hsvImgc1, CV_BGR2HSV);
        inRange(hsvImgc1, Scalar(lowHc1, lowSc1, lowVc1), Scalar(highHc1, highSc1,
highVc1), binImgc1);
        //Aplicación de transformaciones morfológicas
        erode(binImgc1, binImgc1, element);
        dilate(binImgc1, binImgc1, element);

        //Transición de espacio de color de BGR a HSV - Cámara 02
        cvtColor(Imgc2, hsvImgc2, CV_BGR2HSV);
        inRange(hsvImgc2, Scalar(lowHc2, lowSc2, lowVc2), Scalar(highHc2, highSc2,
highVc2), binImgc2);
       //Aplicación de transformaciones morfológicas
        erode(binImgc2, binImgc2, element);
```

```
        dilate(binImgc2, binImgc2, element);

        //Declaración de variables para la identificación de contornos.
        int largest_areac1 = 0, largest_areac2 = 0;
        int largest_contour_indexc1 = 0, largest_contour_indexc2 = 0;

        // Vector para almacenar el contorno
        vector<vector<Point> > contoursc1a, contoursc2a;
        vector<Vec4i> hierarchyc1a, hierarchyc2a;
        findContours(binImgc1, contoursc1a, hierarchyc1a, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);
        findContours(binImgc2, contoursc2a, hierarchyc2a, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);

        // Iteración a través de cada contorno
        for (int i = 0; i< contoursc1a.size(); i++)
        {
            //Se encuentra el área del contorno
            double a = contourArea(contoursc1a[i], false);
            if (a>largest_areac1)
            {
                largest_areac1 = a;
                //Almacenamiento del índice del contorno más grande
                largest_contour_indexc1 = i;
            }
        }

        // Iteración a través de cada contorno
        for (int i = 0; i< contoursc2a.size(); i++)
        {
            //Encuentra el área del contorno.
            double a = contourArea(contoursc2a[i], false);
            if (a>largest_areac2)
            {
                largest_areac2 = a;
                //Almacenamiento del índice del contorno más grande
                largest_contour_indexc2 = i;
            }
        }

        // Color del contorno
        Scalar color(255, 255, 255);

        drawContours(Imgc1, contoursc1a, largest_contour_indexc1, color, CV_FILLED, 8,
hierarchyc1a);
        drawContours(binImgc1, contoursc1a, largest_contour_indexc1, color, CV_FILLED, 8,
hierarchyc1a);
        drawContours(Imgc2, contoursc2a, largest_contour_indexc2, color, CV_FILLED, 8,
hierarchyc2a);
```

```
        drawContours(binImgc2, contoursc2a, largest_contour_indexc2, color, CV_FILLED, 8,
hierarchyc2a);

        Mat cam1(480, 640, CV_8U, Scalar::all(0)), cam2(480, 640, CV_8U, Scalar::all(0));
        drawContours(cam1, contoursc1a, largest_contour_indexc1, color, CV_FILLED, 8,
hierarchyc1a);
        drawContours(cam2, contoursc2a, largest_contour_indexc2, color, CV_FILLED, 8,
hierarchyc2a);

        //Declaración de variables para la aproximación de curvas
        vector<vector<Point> > contoursc1b, contoursc2b;
        vector<Vec4i> hierarchyc1b, hierarchyc2b;
        findContours(cam1, contoursc1b, hierarchyc1b, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
        findContours(cam2, contoursc2b, hierarchyc1b, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
        vector<vector<Point> > contours_polyc1(contoursc1b.size()),
contours_polyc2(contoursc2b.size());
        vector<Point2f>centerc1(contoursc1b.size()), centerc2(contoursc2b.size());
        vector<float>radiusc1(contoursc1b.size()), radiusc2(contoursc2b.size());

        //Para la cámara 01
        for (size_t i = 0; i < contoursc1b.size(); i++)
        {
            approxPolyDP(Mat(contoursc1b[i]), contours_polyc1[i], 3, true);
            minEnclosingCircle(contours_polyc1[i], centerc1[i], radiusc1[i]);
        }

        for (size_t i = 0; i< contoursc1b.size(); i++)
        {
            Scalar color = Scalar(255, 255, 255);
            drawContours(cam1, contours_polyc1, (int)i, color, 1, 8, vector<Vec4i>(), 0,
Point());
            circle(cam1, centerc1[i], (int)radiusc1[i], color , 2, 8, 0);
            circle(Imgc1, centerc1[i], (int)radiusc1[i], color, 2, 8, 0);
            putText(cam1, format("(%0.1f,%0.1f)", centerc1[i].x, centerc1[i].y), centerc1[i],
CV_FONT_NORMAL, 1, color, 1);

            u = centerc1[i].x;
            v = centerc1[i].y;

        }

        //Para la camara 2
        for (size_t i = 0; i < contoursc2b.size(); i++)
        {
            approxPolyDP(Mat(contoursc2b[i]), contours_polyc2[i], 3, true);
            minEnclosingCircle(contours_polyc2[i], centerc2[i], radiusc2[i]);
        }
        for (size_t i = 0; i< contoursc2b.size(); i++)
```

```
        {
            Scalar color = Scalar(255, 255, 255);
            drawContours(cam2, contours_polyc2, (int)i, color, 1, 8, vector<Vec4i>(), 0,
Point());
            circle(cam2, centerc2[i], (int)radiusc2[i], color, 2, 8, 0);
            circle(Imgc2, centerc2[i], (int)radiusc2[i], color, 2, 8, 0);
            putText(cam2, format("(%0.1f,%0.1f)", centerc2[i].x, centerc2[i].y), centerc2[i],
CV_FONT_NORMAL, 1, color, 1);

            up = centerc2[i].x;
            vp = centerc2[i].y;
        }

        pixeles msg;
        msg.u=u;
        msg.v=v;
        msg.up=up;
        msg.vp=vp;

        printf("\n *----------------------*----------------------*");
        printf("\n |        Posición en píxeles del marcador      |");
        printf("\n *----------------------*----------------------*");
        printf("\n |Posición del marcador 1|Posición del marcador 2|");
        printf("\n *----------------------*----------------------*");
        printf("\n |       u=%0.1f       |       up=%0.1f      |",u,up);
        printf("\n |       v=%0.1f       |       vp=%0.1f      |",v,vp);
        printf("\n *----------------------*----------------------*\n");

        pub.publish(msg);

        //Cración de barras para el control de color.
        namedWindow("Control de color C1", CV_WINDOW_AUTOSIZE);
        createTrackbar("LowH", "Control de color C1", &lowHc1, 180); //Hue (0 - 179)
        createTrackbar("HighH", "Control de color C1", &highHc1, 180);
        createTrackbar("LowS", "Control de color C1", &lowSc1, 255); //Saturation (0 - 255)
        createTrackbar("HighS", "Control de color C1", &highSc1, 255);
        createTrackbar("LowV", "Control de color C1", &lowVc1, 255); //Value (0 - 255)
        createTrackbar("HighV", "Control de color C1", &highVc1, 255);

        namedWindow("Control de color C2", CV_WINDOW_AUTOSIZE);
        createTrackbar("LowH", "Control de color C2", &lowHc2, 180); //Hue (0 - 179)
        createTrackbar("HighH", "Control de color C2", &highHc2, 180);
        createTrackbar("LowS", "Control de color C2", &lowSc2, 255); //Saturation (0 - 255)
        createTrackbar("HighS", "Control de color C2", &highSc2, 255);
        createTrackbar("LowV", "Control de color C2", &lowVc2, 255); //Value (0 - 255)
        createTrackbar("HighV", "Control de color C2", &highVc2, 255);

        //Visualización de las cámaras
        //Cámara 01
        namedWindow("cam1", CV_WINDOW_AUTOSIZE);
```

```
        namedWindow("Imagen Binaria C1", CV_WINDOW_AUTOSIZE);
        namedWindow("Camara_01", CV_WINDOW_AUTOSIZE);
        imshow("cam1", cam1);
        imshow("Imagen Binaria C1", binImgc1);
        imshow("Camara_01", Imgc1);

        //Cámara 02
        namedWindow("cam2", CV_WINDOW_AUTOSIZE);
        namedWindow("Imagen Binaria C2", CV_WINDOW_AUTOSIZE);
        namedWindow("Camara_02", CV_WINDOW_AUTOSIZE);
        imshow("cam2", cam2);
        imshow("Imagen Binaria C2", binImgc2);
        imshow("Camara_02", Imgc2);

        if((waitKey(30))==27)
        {
            printf("El usuario ha presionado ESC\n");
            break;
        }

        spinOnce();

    }

    return 0;

}
```

B.2. Nodo 2 – Sistema de conversión de espacios 2D a 3D

```
/*
TESIS :"Sistema de posicionamiento indoor para el guiado de robots móviles implementado
en Robot Operating System (ROS)"
Autor: Jayro Zaid Paiva Mimbela
UNIVERSIDAD DE PIURA-2019
*/

//Librerías empleadas
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "ros/ros.h"
#include "tesis_jayro_zaid_paiva_mimbela/pixeles.h"
#include "tesis_jayro_zaid_paiva_mimbela/espaciales.h"
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <fstream>
```

```
using namespace std;
using namespace cv;
using namespace ros;
using namespace tesis_jayro_zaid_paiva_mimbela;

//Declaración de variables
const float pi = 3.1416f;
float kxc1,kyc1,kxr1,kyr1,kzr1;
float kxc2,kyc2,kxr2,kyr2,kzr2;
float lambda,nu,t;
float Sx,Sy,Sz;
float Rx,Ry,Rz;;
float vx,vy,vz;
float ux,uy,uz;
float wx,wy,wz;
float u,v,up,vp;

//Definición de matrices extrínsecas
float alpha1=(-22.5/180.0)*pi;
float beta1=-pi;
float gamma1=-pi/2.0;
float alpha2=(-22.5/180.0)*pi;
float beta2=-pi;
float gamma2=0.0;

float Q11= cos(beta1)*cos(gamma1);
float Q12= (sin(alpha1)*sin(beta1)*cos(gamma1)) - (cos(alpha1)*sin(gamma1));
float Q13= (cos(alpha1)*sin(beta1)*cos(gamma1))+(sin(alpha1)*sin(gamma1));
float Q21= cos(beta1)*sin(gamma1);
float Q22= (sin(alpha1)*sin(beta1)*sin(gamma1))+(cos(alpha1)*cos(gamma1));
float Q23= (cos(alpha1)*sin(beta1)*sin(gamma1))-(sin(alpha1)*cos(gamma1));
float Q31=  -sin(beta1);
float Q32= sin(alpha1)*cos(beta1);
float Q33= cos(alpha1)*cos(beta1);

float R11= cos(beta2)*cos(gamma2);
float R12= (sin(alpha2)*sin(beta2)*cos(gamma2)) - (cos(alpha2)*sin(gamma2));
float R13= (cos(alpha2)*sin(beta2)*cos(gamma2)) + (sin(alpha2)*sin(gamma2));
float R21= cos(beta2)*sin(gamma2);
float R22= (sin(alpha2)*sin(beta2)*sin(gamma2))+(cos(alpha2)*cos(gamma2));
float R23= (cos(alpha2)*sin(beta2)*sin(gamma2))-(sin(alpha2)*cos(gamma2));
float R31=  -sin(beta2);
float R32= sin(alpha2)*cos(beta2);
float R33= cos(alpha2)*cos(beta2);

//Matriz de rotación - Cámara 01
float R1[3][3]=

{
   {Q11,Q21,Q31},
```

```
    {Q12,Q22,Q32},
    {Q13,Q23,Q33}
};

//Matriz de traslación - Cámara 01
float t1[3][1]=

{
    {1.5},
    {53.0},
    {106.0}
};

//Matriz de rotación - Cámara 02
float R2[3][3]=

{
    {R11,R21,R31},
    {R12,R22,R32},
    {R13,R23,R33}
};

//Matriz de traslación - Cámara 02
float t2[3][1]=

{
    {46.0},
    {1.5 },
    {106.0}
};

//Definición de matrices intrínsecas
//Cámara 01
float A1[3][3]=

{
    {641.005175542343,0.00000000000000,314.264418608371},
    {0.00000000000000,640.456610088887,240.340845427825},
    {0.00000000000000,0.00000000000000,1.00000000000000}
};

//Cámara 02
float A2[3][3]=

{
    {633.130740940056,0.00000000000000,318.85831503788},
    {0.00000000000000,633.27789680685 ,231.268076567509},
    {0.00000000000000,0.00000000000000,1.00000000000000}
};
```

```
void gauss(float m[3][4],float &sol1, float &sol2, float &sol3)
{
    m[0][1] = m[0][1]/m[0][0];
    m[0][2] = m[0][2]/m[0][0];
    m[0][3] = m[0][3]/m[0][0];
    m[0][0] = m[0][0]/m[0][0];

    m[1][1] = m[1][1] - m[1][0]*m[0][1];
    m[1][2] = m[1][2] - m[1][0]*m[0][2];
    m[1][3] = m[1][3] - m[1][0]*m[0][3];
    m[1][0] = m[1][0] - m[1][0]*m[0][0];

    m[2][1] = m[2][1] - m[2][0]*m[0][1];
    m[2][2] = m[2][2] - m[2][0]*m[0][2];
    m[2][3] = m[2][3] - m[2][0]*m[0][3];
    m[2][0] = m[2][0] - m[2][0]*m[0][0];

    m[1][2] = m[1][2]/m[1][1];
    m[1][3] = m[1][3]/m[1][1];
    m[1][1] = m[1][1]/m[1][1];

    m[0][0] = m[0][0] - m[0][1]*m[1][0];
    m[0][2] = m[0][2] - m[0][1]*m[1][2];
    m[0][3] = m[0][3] - m[0][1]*m[1][3];
    m[0][1] = m[0][1] - m[0][1]*m[1][1];

    m[2][0] = m[2][0] - m[2][1]*m[1][0];
    m[2][2] = m[2][2] - m[2][1]*m[1][2];
    m[2][3] = m[2][3] - m[2][1]*m[1][3];
    m[2][1] = m[2][1] - m[2][1]*m[1][1];

    m[2][3] = m[2][3]/m[2][2];
    m[2][2] = m[2][2]/m[2][2];

    m[0][0] = m[0][0] - m[0][2]*m[2][0];
    m[0][1] = m[0][1] - m[0][2]*m[2][1];
    m[0][3] = m[0][3] - m[0][2]*m[2][3];
    m[0][2] = m[0][2] - m[0][2]*m[2][2];

    m[1][0] = m[1][0] - m[1][2]*m[2][0];
    m[1][1] = m[1][1] - m[1][2]*m[2][1];
    m[1][3] = m[1][3] - m[1][2]*m[2][3];
    m[1][2] = m[1][2] - m[1][2]*m[2][2];

    sol1 = m[0][3];
    sol2 = m[1][3];
    sol3 = m[2][3];
}
```

```
void prodX( float cx , float cy, float cz,float dx , float dy, float dz, float &ux, float &uy, float
&uz)
{
    ux = cy*dz - cz*dy;
    uy = cz*dx - cx*dz;
    uz = cx*dy - cy*dx;
}

void vectorunitario( float c1 , float c2, float c3, float &u1, float &u2, float &u3)
{
    u1=(c1/sqrt(pow(c1,2)+pow(c2,2)+pow(c3,2)));
    u2=(c2/sqrt(pow(c1,2)+pow(c2,2)+pow(c3,2)));
    u3=(c3/sqrt(pow(c1,2)+pow(c2,2)+pow(c3,2)));
}

void llamada( const pixeles::ConstPtr& msg)
{
    u = msg->u;
    v = msg->v;
    up = msg->up;
    vp = msg->vp;
}

int main(int argc, char** argv)
{
    init(argc, argv,"sistema_conversion_2d_3d");
    NodeHandle n;
   Subscriber sub=n.subscribe("coordenadas_pixeles",10,llamada);
    Publisher pub=n.advertise<espaciales>("coordenadas_espaciales",10);

    Rate loop_rate(10);

    while(ok())
    {

        kxc1= -1*(u-A1[0][2])/A1[0][0];
        kyc1= -1*(v-A1[1][2])/A1[1][1];
        kxc2= -1*(up-A2[0][2])/A2[0][0];
        kyc2= -1*(vp-A2[1][2])/A2[1][1];

        kxr1= (Q11*kxc1)+(Q12*kyc1)+(Q13*1);
        kyr1= (Q21*kxc1)+(Q22*kyc1)+(Q23*1);
        kzr1= (Q31*kxc1)+(Q32*kyc1)+(Q33*1);

        kxr2= (R11*kxc2)+(R12*kyc2)+(R13*1);
        kyr2= (R21*kxc2)+(R22*kyc2)+(R23*1);
        kzr2= (R31*kxc2)+(R32*kyc2)+(R33*1);

        vectorunitario(kxr1,kyr1,kzr1,ux,uy,uz);
        vectorunitario(kxr2,kyr2,kzr2,vx,vy,vz);
```

```
        prodX(ux,uy,uz,vx,vy,vz,wx,wy,wz);

        float g[3][4] = {ux,-vx,-wx,t2[0][0]-t1[0][0],
                         uy,-vy,-wy,t2[1][0]-t1[1][0],
                         uz,-vz,-wz,t2[2][0]-t1[2][0]};

        gauss(g, lambda, nu, t);

        Sx = ux*lambda + t1[0][0];
        Sy = uy*lambda + t1[1][0];
        Sz = uz*lambda + t1[2][0];
        Rx = vx*nu + t2[0][0];
        Ry = vy*nu + t2[1][0];
        Rz = vz*nu + t2[2][0];

        float x=(Sx+Rx)/2;
        float y=(Sy+Ry)/2;
        float z=(Sz+Rz)/2;

        printf("\n *----------------------*----------------------*");
        printf("\n |        Posición en píxeles del marcador      |");
        printf("\n *----------------------*----------------------*");
        printf("\n |Posición del marcador 1|Posición del marcador 2|");
        printf("\n *----------------------*----------------------*");
        printf("\n |       u=%0.1f        |       up=%0.1f       |",u,up);
        printf("\n |       v=%0.1f        |       vp=%0.1f       |",v,vp);
        printf("\n *----------------------*----------------------*");
        printf("\n |     Posición en el espacio del marcador      |");
        printf("\n *----------------------*----------------------*");
        printf("\n |       x=%0.1f      y =%0.1f     z=%0.1f      |",x,y,z);
        printf("\n *----------------------*----------------------*\n");

        espaciales msg;
        msg.x=x;
        msg.y=y;
        msg.z=z;
        pub.publish(msg);

        spinOnce();
        loop_rate.sleep();
    }

    return 0;
}
```

B.3. Nodo 3 – Sistema de reconstrucción de trayectorias

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
```

```
'''
TESIS :"Sistema de posicionamiento indoor para el guiado de robots móviles implementado
en Robot Operating System (ROS)"
Autor: Jayro Zaid Paiva Mimbela
UNIVERSIDAD DE PIURA-2019
'''

#Librerías incluidas
import rospy
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import mpl_toolkits.mplot3d.axes3d as p3
import Image
from tesis_jayro_zaid_paiva_mimbela.msg import espaciales

im=Image.open('/home/jayro_zaid/tesis/src/tesis_jayro_zaid_paiva_mimbela/udep.png')
height = im.size[1]

def llamada(msg):
    global x, y, z
    x=msg.x
    y=msg.y
    z=msg.z

rospy.init_node("sistema_reconstruccion_trayectorias")

sub = rospy.Subscriber("coordenadas_espaciales",espaciales,llamada)

fig = plt.figure(1,figsize=(10,15),
dpi=80,facecolor='#00467A',edgecolor='k',constrained_layout=False)

fig.suptitle(u'Sistema de posicionamiento indoor para el guiado de robots móviles
implementado en Robot Operating System', fontsize=15, color='w')

fig.figimage(im, xo=10, yo=550, origin='lower')

ax1 = fig.add_subplot(1, 2, 1,projection="3d")
ax2 = fig.add_subplot(2, 4, 3)
ax3 = fig.add_subplot(2, 4, 4)
ax4 = fig.add_subplot(2, 4, 7)

ax1.set_xlabel('x',fontsize=15)
ax1.set_ylabel('y',fontsize=15)
ax1.set_zlabel('z',fontsize=15)
ax1.set_title(u'Trayectoria del móvil en el espacio', fontsize=10)
ax1.set_facecolor('#c7dced')
ax1.set_adjustable('datalim', True)
ax1.set_xlim(0.0,100.0)
ax1.set_ylim(0.0,100.0)
ax1.set_zlim(0.0,1000.0)
```

```
ax2.set_xlabel('x',fontsize=10, color='w')
ax2.set_ylabel('y',fontsize=10, color='w')
ax2.grid(True)
ax2.set_title(u'Trayectoria del móvil en el plano XY',fontsize=10, color='w')
ax2.tick_params(labelcolor='w',size=8,color='w')
ax2.set_facecolor('#c7dced')
ax2.set_xlim(0.0,100.0)
ax2.set_ylim(0.0,100.0)

ax3.set_xlabel('x',fontsize=10, color='w')
ax3.set_ylabel('z',fontsize=10, color='w')
ax3.grid(True)
ax3.set_title(u'Trayectoria del móvil en el plano XZ',fontsize=10, color='w')
ax3.tick_params(labelcolor='w',size=8,color='w')
ax3.set_facecolor('#c7dced')
ax3.set_xlim(0.0,100.0)
ax3.set_ylim(0.0,100.0)

ax4.set_xlabel('y',fontsize=10, color='w')
ax4.set_ylabel('z',fontsize=10, color='w')
ax4.grid(True)
ax4.set_title(u'Trayectoria del móvil en el plano YZ',fontsize=10, color='w')
ax4.tick_params(labelcolor='w',size=8,color='w')
ax4.set_facecolor('#c7dced')
ax4.set_xlim(0.0,100.0)
ax4.set_ylim(0.0,100.0)

def animate(i):

    ax1.scatter([x], [y], [z], color="b", s=10)
    ax2.scatter([x], [y], color="b", s=10)
    ax3.scatter([x], [z], color="b", s=10)
    ax4.scatter([y], [z], color="b", s=10)

while not rospy.is_shutdown():
    ani = animation.FuncAnimation(fig, animate, interval=10)
    plt.show()

rospy.spin()
```