DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

TEMA: NAVEGACIÓN AUTÓNOMA DE MINI-DRONES PARA MULTI-ROBOTS SLAM 3D EN ENTORNOS ESTÁTICOS CON ALGORITMOS DE INTELIGENCIA DE EJAMBRE

AUTOR: TAYUPANTA ZÚÑIGA, WILMER EDUARDO

DIRECTOR: ING. ERAZO SOSA, ANDRÉS SEBASTIÁN

SANGOLQUÍ

2018

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICACIÓN

Certifico que el trabajo de titulación, ***"NAVEGACIÓN AUTÓNOMA DE MINI-DRONES PARA MULTI-ROBOTS SLAM 3D EN ENTORNOS ESTÁTICOS CON ALGORITMOS DE INTELIGENCIA DE ENJAMBRE"*** fue realizado por el señor ***Tayupanta Zúñiga, Wilmer Eduardo*** el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 24 de agosto de 2018

Ing. Erazo Sosa, Andrés Sebastián

DIRECTOR

C.C.: 1720400082

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORÍA DE RESPONSABILIDAD

Yo, ***Tayupanta Zúñiga, Wilmer Eduardo***, declaro que el contenido, ideas y criterios del trabajo de titulación: ***"NAVEGACIÓN AUTÓNOMA DE MINI-DRONES PARA MULTI-ROBOTS SLAM 3D EN ENTORNOS ESTÁTICOS CON ALGORITMOS DE INTELIGENCIA DE ENJAMBRE"*** es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, 24 de agosto de 2018

Tayupanta Zúñiga, Wilmer Eduardo

C.C.: 1722588520

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

Yo, ***Tayupanta Zúñiga, Wilmer Eduardo***, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **"*NAVEGACIÓN AUTÓNOMA DE MINI-DRONES PARA MULTI-ROBOTS SLAM 3D EN ENTORNOS ESTÁTICOS CON ALGORITMOS DE INTELIGENCIA DE ENJAMBRE*"** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 24 de agosto de 2018

Tayupanta Zúñiga, Wilmer Eduardo

C.C.: 1722588520

DEDICATORIA

A mi novia Sharon que me ha sabido comprender, brindar todo su cariño para mi felicidad, siempre ha creído en mis capacidades y que ha estado presente en los buenos y malos momentos. Le dedico todo mi esfuerzo de corazón pues gracias a ella todo esto se ha hecho posible y que aún queda mucho porque lograr juntos.

A mi madre por toda la paciencia y sus consejos que han sido pilares de lo que soy ahora, porque siempre ha estado conmigo brindando sus consejos.

A mis abuelitos por ayudar incondicionalmente a mi familia y han sabido comprender las situaciones que hemos pasado.

A todos ellos les dedico el presente trabajo de titulación.

AGRADECIMIENTO

Agradezco a mis padres, por toda la paciencia y el cariño con el que me educaron. Gracias por escuchar cada idea y saberla guiar para que se cumpla, a mi madre por estar incondicionalmente a mi lado y a mis hermanos por todo los consejos y momentos que han sido difíciles de caminar.

A mi novia Sharon, por estar ahí cuando más lo necesite y saberme escuchar mis ideas, comprender mi forma de pensar y que sepa que le agradezco por todo lo que ha significado para mí, por todas las palabras y sonrisas que me brinda.

Gracias a mis primos por sus consejos y su tiempo que compartieron conmigo, a mis abuelitos que ahora se encuentran muy lejos, a mi abuelita Laura por apoyar y brindar todo de si a mi familia.

Por último, pero no menos importante al Ing. Andrés Erazo por toda la guía y brindar sus conocimientos para que este tema de tesis llegue a su final.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente trabajo muestra un estudio sobre reconstrucción de entornos a través de cámaras monoculares de mini drones y el diseño de controlador de vuelo para estos últimos, la utilización de extracción y seguimiento de características entre imágenes hacen posible la reconstrucción de entornos a través del modelo de cámara pinhole, que utiliza matrices de transformación para convertir puntos en coordenadas de pixeles en puntos en coordenadas externas. Las cámaras tienen propiedades que ayudan a la reconstrucción de entornos, estos valores son calculados a través de calibración de cámara por medio de un modelo pre definido como un tablero de ajedrez. Los parámetros obtenidos son los parámetros intrínsecos de la cámara que corrigen matemáticamente la distorsión causada por el lente, de igual forma se necesita los parámetros extrínsecos de la cámara para realizar la reconstrucción 3D. Para ello se llevó a cabo reconocimiento de objetos a través de redes neuronales convolucionales, para posteriormente ubicarlos en el espacio. La utilización de marcadores y la librería OpenCV permite la obtención de la posición y orientación de un objeto en el espacio a través de marcadores ArUco. Estos valores son necesarios para realizar seguimiento visual del objeto, para implementar algoritmos de control de vuelo de los mini drones.

PALABRAS CLAVE:

- **CÁMARA PINHOLE**
- **RED NEURONAL CONVOLUCIONAL**
- **PARÁMETROS INTRÍNSECOS Y EXTRÍNSECOS**
- **MARCADOR ARUCO**

ABSTRACT

The present work shows a study on reconstruction of environments through mini-drone monocular cameras and the design of flight controller for the latest ones, the use of extraction and tracking of features between images make possible the reconstruction of environments through the pinhole camera model, which uses transformation matrices to convert points into pixel coordinates into points in external coordinates. The cameras have properties that help the reconstruction of environments, these values are calculated through camera calibration by means of pre-defined models such as a chessboard. The parameters obtained are the intrinsic parameters of the camera that mathematically correct the distortion caused by the lens; in the same way you need the extrinsic parameters of the camera to perform the 3D reconstruction. For this, recognition of objects was carried out through convolutional neural networks, to later locate them in space. The use of markers and the OpenCV library allows obtaining the position and orientation of an object in space through ArUco markers. These values are necessary to perform visual tracking of the object, to implement flight control algorithms of the mini drones.

KEYWORDS:

- **CAMERA PINHOLE**
- **CONVOLUTIONAL NEURAL NETWORKS**
- **INTRINSIC AND EXTRINSIC PARAMETERS**
- **ARUCO MARKER**

CAPÍTULO I

INTRODUCCIÓN

1.1 Antecedentes

En la actualidad la robótica se ha inspirado en la naturaleza tomando en cuenta el comportamiento de las especies animales, un comportamiento notable es la capacidad de cooperación para lograr un objetivo común; es así como se han desarrollado algoritmos para que los robots puedan cumplir con dicho objetivo (Archimowicz, Martínez, Monzón, & Canetti, 2011).

La robótica de enjambre emplea conceptos de inteligencia de enjambre en sistemas multi robot autónomos con un enfoque distribuido (Bayındır, 2016), para la optimización de tareas como la exploración de áreas, rescate, recolección, mantenimiento, etc. El termino inteligencia de enjambre fue empleado por Beni and Wang para describir sistemas de robótica celular (Beni & Wang, 1993), este término se refiere a la inteligencia colectiva que surge de la interacción de varios individuos autónomos (Bonabeau, Dorigo, & Theraulaz, 1999).

Algunos de los estudios sobre robótica de enjambre son de: optimización darwiniana de enjambres de partículas (RDPSO), que emplea algoritmos evolutivos sobre los no evolutivos (Couceiro, Vargas, Rocha, & Ferreira, 2014); utilización de módulos de abstracción de tareas (TAM), que hace que una tarea multi robot compleja sea abstraída en subtareas de un solo robot (Brutschy et al., 2015); módulos generales, donde cada robot debe tener funciones básicas como la detección, comunicación y movimiento donde los robots intercambian y propagan información entre sí para la cooperación a nivel de enjambre (Tan & Zheng, 2013).

Un tema importante de investigación en la Visión por Computadora es la reconstrucción de objetos tridimensionales a partir de imágenes bidimensionales (Azevedo, Tavares, & Vaz, 2009),

el principal objetivo es utilizar las transformaciones proyectivas de un par de imágenes para determinar mapas de profundidad (Hartley & Zisserman, 2004). Para ello se debe seguir una secuencia de pasos que son: se determina puntos de interés entre el par de imágenes, posteriormente se busca correspondencia de los puntos, se utiliza las matrices de rotación para describir el movimiento entre las imágenes para su posterior reconstrucción (Grandón-Pastén, Aracena-Pizarro, & Tozzi, 2007).

Modelos como el Pinhole emplea cámaras para obtener un conjunto de características de interés que ubican a un objeto en el entorno (Murphy & Robin, 2000), este método relaciona los parámetros de la imagen proyectada y los parámetros intrínsecos de la cámara, con posición y medidas de los objetos observados (Huang, Wang, Dissanayake, & Frese, 2009), posteriores investigaciones presentan la generación de mapas 3D a base de la optimización de mínimos cuadrados con la utilización de filtros de información de extendido (EIF), a través del suavizado e interacción para la mejora de la exactitud (Huang et al., 2009).

El método LSD-SLAM permite la construcción de mapas en base a la alineación directa de la imagen, construyendo en el entorno 3D en tiempo real en una CPU utilizando fotogramas clave con mapas de profundidad, esto se logra a través del filtrado y comparación sobre un gran número de componentes estéreo en base a pixeles (Engel, Schöps, & Cremers, 2014). Otro método emplea cámaras RGB-D para captura de imágenes RGB para la construcción de mapas 3D, la combinación de las características visuales y alineación basada en la forma para la optimización de la posición y así lograr mapas consistentes (Henry et al., 2010).

El termino SLAM es el acrónimo de sus siglas en inglés Simultaneous localization and mapping (localización y mapeado simultáneos). SLAM se emplea para construir un mapa de un entorno desconocido por uno o varios robots, mientras navega por el mismo (Riisgaard & Blas, 2005), una

amplia gama de algoritmos SLAM son los enfoques para la generación de un gráfico de posición y orientación, que consiste en estados para la navegación del robot. Cada estado del robot contiene la posición y los datos del sensor (Deutsch, Liu, & Siegwart, 2016).

1.2 Justificación e importancia

El desarrollo de las tecnologías hace que los vehículos aéreos no tripulados (UAV) requieran menos supervisión humana, lo que ha llevado el interés de los ministros de defensa de 60 países para la vigilancia militar, como es el caso de los Estados Unidos, Argentina, chile, entre otros países y grupos militares como Hizbolá o Hamás.

Los Estados Unidos ha invertido más de 600 millones de dólares para el desarrollo e investigación de la autonomía de los UVAs (Packer & Reeves, 2013), Argentina ha desarrollado modelos UAV Lipán M3 que cuenta con autonomía de cinco horas, recepción de video en alta calidad y telemetría de largo alcance (Infodefensa.com, 2011), la empresa Elbit vendió a Chile el modelo Elbit Hermes 900 que cuenta con autonomía de vuelo de más de 30 horas, con capacidad de multi visión, realizar misiones ISTAR (Inteligencia, vigilancia, adquisición de objetos y reconocimiento) (Elbit Systems, 2014).

En la mayoría de ocasiones un único robot no puede lograr la robustez y eficacia que se puede lograr con la cooperación de diferentes robots (Merino, Capitán Jesús, & Ollero Anibal, 2009), cabe mencionar que los robots poseen comportamientos simples para la resolución de tareas, con mayor velocidad y precisión (Archimowicz et al., 2011; Bulla Cruz et al., 2014).

La investigación de robótica de enjambre se inspira en sistemas biológicos como las colonias de insectos, bandadas de aves, entre otras. Estas investigaciones apoyan al desarrollo de nuevas reglas para la solución de problemas que podrían ser difíciles de resolver en base a las técnicas tradicionales (Barca & Sekercioglu, 2013), diversos grupos de investigación como el Laboratorio

de “Bioingeniería e Inteligencia en Sistemas Autónomos” de la Universidad de Bristol, en Inglaterra, desarrolla estudios sobre inteligencia de enjambre de robots donde se centra en la comprensión de que las culturas presentes en la naturaleza evolucionan a partir de comportamientos similares (Bristol, 2006), el proyecto “Swarm-bots” desarrolla sus estudios en base a técnicas de auto ensamblado y auto organización de robots (Gro, Bonani, Mondada, & Dorigo, 2006).

Los enjambres de robots pueden realizar exploración de áreas rápidamente, el nivel de robustez es mayor ante fallos ya que si un agente falla otros pueden asumir el trabajo, la distribución de carga de trabajo es distribuida entre sus miembros para la realización de tareas aéreas específicas y la realización de varias tareas simultáneamente con robots que realizan tareas simples (Barca & Sekercioglu, 2013). Es necesario la imitación de estrategias colaborativas como algoritmos de exploración, evasión de obstáculos y planificación de rutas mediante la utilización de algoritmos de inteligencia de enjambre (Martínez Puerta & Vallejo Jiménez, 2016).

Sin embargo, existen ocasiones en las que no se cuenta con la información del entorno en el cual el robot se desplazara libremente. Durante los últimos años el desarrollo de sistemas de estéreo visión, sensores láser y la posibilidad de integrar técnicas de visión por computadora para las investigaciones de SLAM 3D (Gil, Reinoso, Payá, & Ballesta, 2007), lo cual ha facilitado la construcción de mapas en base a la ayuda de la información proporcionada por los sensores (Andrade & Martin, 2007).

El método Pinhole tiene como enfoque la relación matemática que existe las coordenadas de un punto en el espacio y su proyección en el plano de imagen, es una aproximación que genera un mapa a partir de una imagen en 2D. Es por ello que el método Pinhole es utilizado frecuentemente para la representación del espacio 3D (Sturm, 2014).

El Proyecto de Investigación “Localización de TNT y pólvora de base doble a través de censado químico en un entorno controlado mediante robótica cooperativa”, no cuenta con un mapa del entorno donde se va a realizar la búsqueda de la pólvora, emplea algoritmos de búsqueda a base de las señales provenientes de los sensores químicos sin tomar en cuenta de los elementos que se encuentren en la zona de búsqueda. Dadas las condiciones que anteceden, se necesita un sistema de robótica con inteligencia de enjambre para la generación de mapas 3D con tolerancia a fallas y gran precisión.

1.3 Alcance del proyecto

El presente proyecto de investigación tiene como finalidad el desarrollo de un algoritmo de inteligencia de enjambre, que permita la localización y mapeo simultáneos de un entorno controlado con la utilización de mini drones, relacionado al proyecto de investigación “Localización de TNT y pólvora de base doble a través de censado químico en un entorno controlado mediante robótica cooperativa”.

Para ello se realizará simulaciones con el software V-REP, la utilización de ROS para el control de los robots y con la programación en lenguaje C++ y Python. Los mini drones deben poseer cámaras monoculares para la adquisición de imágenes del entorno para su posterior procesamiento para lograr la generación de mapas 3D.

Posteriormente se implementará con modelos reales de mini drones que poseerán cámaras y una duración de vuelo de 5 minutos. Se emplearán 3 robots los cuales contarán con funciones básicas para lograr el cumplimiento del objetivo.

Las imágenes obtenidas serán transmitidas a una computadora que servirá para el procesamiento, así como el control de la navegación de mini drones.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar un sistema SLAM 3D en base a mini drones con algoritmos de inteligencia de enjambre con ROS.

1.4.2 Objetivo específico

- Implementar algoritmos de procesamiento de imágenes para reconstrucción del entorno.
- Implementar el algoritmo más óptimo para el control de mini drones físicos.
- Realizar un análisis de error de los resultados obtenidos en la ubicación de objetos en el entorno.

CAPÍTULO II

ESTADO DEL ARTE

2.1 Modelo de cámara pinhole

Es un modelo básico que se emplea en la visión por computadora, se basa en el concepto de cámara estenopeica que es una cámara cerrada con un pequeño orificio (pinhole) por el cual cada haces de luz que se refleja en el objeto pase por el orificio, de manera que cada punto de la imagen es proyectado en el llamado plano de imagen (**Figura 1**); de tal forma que existe una correspondencia entre la superficie visible y el área que se genera (2D) en el plano de la imagen (Sturm, 2014).

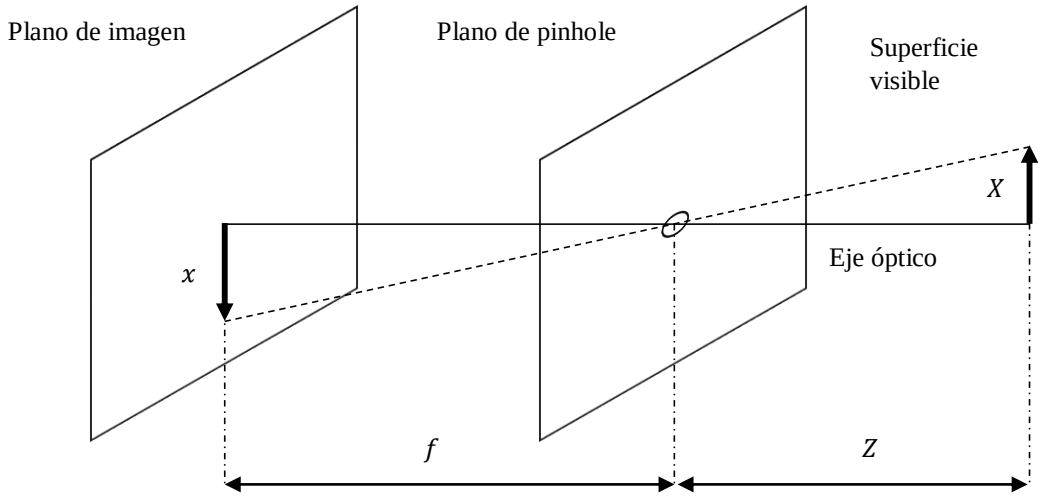


Figura 1. Modelo Pinhole

Lo que conlleva a trabajar con una imagen invertida, pero se puede trabajar con un modelo simplificado donde el plano de la imagen se encuentra delante del centro de proyección (**Figura 2**), el plano de la imagen es en donde se proyecta lo observado en la superficie visible, el centro de proyección es el sistema de coordenadas que se encuentra asociado a la cámara y el eje óptico es un rayo que se origina en el centro de proyección y atraviesa de manera perpendicular el centro del

plano de la imagen (Calderón, 2012). Se definen tres sistemas de coordenadas: el sistema de coordenadas de la cámara (unidades métricas) (X_c, Y_c, Z_c) se emplea la notación x para definir un punto en el sistema, el sistema de coordenadas de la imagen (unidades de pixel) (x, y) se emplea la notación u para definir un punto en el sistema, el sistema de coordenadas externo o del mundo (unidades métricas) (X_e, Y_e, Z_e) se emplea la notación X para definir un punto en el sistema, el sistema de coordenadas de la cámara se encuentra relacionado por una matriz de rotación y de traslación $[R|t]_{3x4}$ con el sistema de coordenadas externo (Brendstrup, 2013).

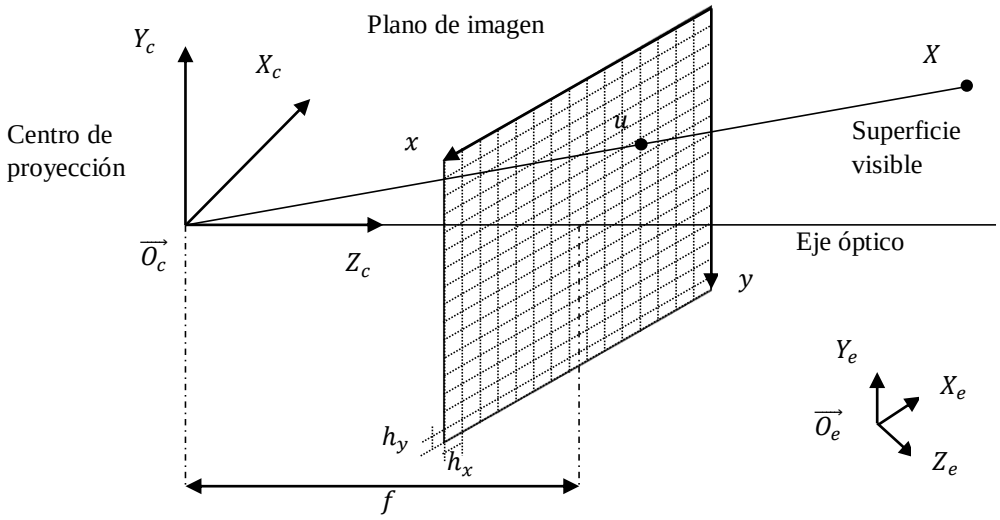


Figura 2. Sistema de referencia

Donde f es la distancia entre el centro de proyección y el plano de imagen (unidades métricas) llamada distancia focal, se divide en dos valores horizontal $f_x = \frac{f}{h_x}$ y vertical $f_y = \frac{f}{h_y}$; los valores de h_x y h_y son las dimensiones físicas de un pixel y los valores de centro de proyección tanto horizontal c_x y vertical c_y (Calderón, 2012). Para relacionar un punto en coordenadas externas con su proyección en el plano de la imagen se tiene que: dado un punto en coordenadas externas $X = (X, Y, Z)$, su proyección en el plano de la imagen $u = (x, y)$, los valores de la distancia focal y centro de proyección se tiene que (Stricker, 2012):

$$x = f_x\left(\frac{X}{Z}\right) + c_x$$

$$y = f_y\left(\frac{Y}{Z}\right) + c_y$$

2.2 Parámetros de la cámara

Para modelar las propiedades físicas de la cámara se plantean los parámetros intrínsecos y extrínsecos de la cámara.

2.2.1 Parámetros intrínsecos

Los parámetros intrínsecos determinan el funcionamiento de la cámara, se emplean los valores como la distancia focal, centro de proyección y el eje óptico, dichos parámetros definen un punto en la imagen con respecto al centro de proyección de la cámara. Los valores varían dependiendo de la cámara , estos valores se definen como la matriz de parámetros intrínsecos de la cámara K (Jiménez, 2009):

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

2.2.1.1 Distorsión

Los elementos ópticos de la cámara causan distorsiones geométricas que puede ser modeladas de manera radial y tangencial, se corrigen de forma no lineal aplicando un offset a las coordenadas reales de un punto en la imagen este proceso se llama undistortion (**Figura 3**) (Brendstrup, 2013).

La distorsión radial identifica que los objetos que se encuentran alejados al eje óptico tienden a redondearse y la distorsión tangencial identifica que el lente no se encuentra alineado al eje óptico (OpenCV Open Source Computer Vision, 2011):

(a) (b)

Figura 3. (a) Imagen sin corrección (b) Imagen corregida
Fuente: (Calderón, 2012)

El valor de r es la distancia del pixel al eje óptico:

$$r = \sqrt{(x - c_x)^2 + \left(y - c_y\right)^2}$$

Distorsión radial:

$$x_{corregido} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corregido} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Distorsión tangencial:

$$x_{corregido} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{corregido} = y + [p_1(r^2 + 2y^2) + p_2 xy]$$

Estos parámetros de distorsión radial y tangencial pueden ser agrupados en un solo vector de distorsión d:

$$d = [k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3]$$

2.2.2 Parámetros extrínsecos

Los parámetros extrínsecos no dependen de las propiedades internas de la cámara, determinan la rotación y traslación del centro de proyección de la cámara con respecto al sistema de coordenadas externo. Se emplea una matriz $[R|t]$ para definir los parámetros (Jiménez, 2009):

$$[R|t] = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{bmatrix}$$

Para intercambiar un punto visto desde el sistema de coordenadas de la cámara x al punto visto desde el sistema de coordenadas externo X y viceversa se emplean las ecuaciones (Ramírez, 2012):

$$x = RX + t$$

$$X = R^T(x - t)$$

2.3 Calibración de la cámara

Para obtener una estimación de los parámetros intrínsecos de la cámara se emplea el proceso de calibración, ya que el fabricante no proporciona dichos valores. El método empleado para la calibración se basa en el análisis de imágenes que contengan un patrón de calibración conocido (Calderón, 2012).

OpenCV provee funciones de calibración que utilizan un conjunto de imágenes que contengan un patrón de calibración que es similar a un tablero de ajedrez ya que permite detectar fácilmente sus vértices (OpenCV Open Source Computer Vision, 2011). Al igual ROS provee un paquete para la calibración de la cámara (**Figura 4**) (ROS Perception, 2009).

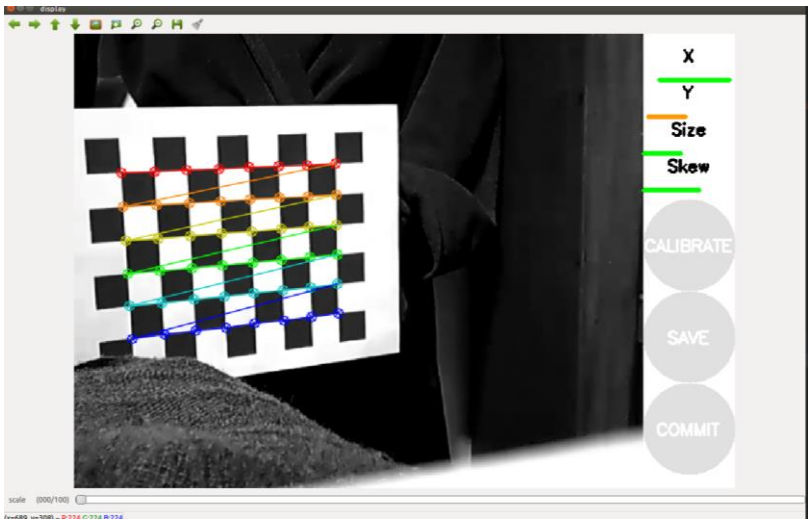


Figura 4. Calibración de cámara

2.4 Matrices de transformación

Las transformaciones de proyección de 3D a 2D se emplean coordenadas homogéneas, para puntos 2D es de tres componentes y para puntos 3D es un vector de cuatro componentes (Gallardo, 2011):

2.4.1 Conversión a coordenadas homogéneas

Coordenadas homogéneas de la imagen:

$$u = (x, y) \rightarrow u' = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Coordenadas homogéneas externas:

$$X = (X, Y, Z) \rightarrow X' = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2.4.2 Conversión desde coordenadas homogéneas

Coordenadas homogéneas de la imagen:

$$u' = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow u = \left(\frac{x}{w}, \frac{y}{w}\right)$$

Coordenadas homogéneas externas:

$$X' = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \rightarrow X = \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right)$$

2.4.3 Matriz de proyección

Para relacionar un punto del sistema de coordenadas externo X con un punto en el sistema de coordenadas de la imagen u se emplea la siguiente ecuación lineal (Gallardo, 2011):

$$u = PX$$

Siendo la matriz de proyección:

$$P = K[R|t]$$

2.5 Extracción de características de una imagen

Para la extracción de características de una imagen se debe tener en consideración el contenido de una imagen que puede ser característico en una región o aproximada a ella, este proceso se llama detección de puntos clave (Howse, Joshi, & Beyeler, 2016).

2.5.1 Detección de esquinas

Métodos como Harris y Stephens que desarrollaron un método de detección de esquinas que se basa en derivadas parciales de la imagen en escala de grises para generar valores propios (**Figura 5a**) (Harris & Stephens, 1988), este método funciona de manera satisfactoria en ciertos casos a diferencia del método de Shi-Tomasi que es un método más robusto que emplea funciones de puntuación diferentes, este último método es empleado para rastrear características de una imagen (**Figura 5b**) (Shi, 1994).

(a) (b)

Figura 5. (a) Método de Harris y Stephens (b) Método de Shi-Tomasi

2.5.2 Transformación de características invariable de escala (SIFT)

SIFT es el acrónimo de Scale invariant feature transform (transformación de características invariable de escala) es uno de los métodos más populares en visión por computadora, es empleado

para extracción de puntos clave y construir descriptores de características. Pero conlleva un esfuerzo computacional mayor, lo que conlleva a que es lento para aplicaciones en tiempo real (Howse et al., 2016).

El método SIFT emplea pirámides para la reducción de muestreo de la imagen tomando diferencia Gaussiana (OpenCV Open Source Computer Vision, 2015), para el cálculo de la invariancia de escala utiliza un filtro Gaussiano para construir los niveles sucesivos de la pirámide con lo que analiza el punto actual y los compara en con sus vecinos para obtener un valor máximo lo que significa que es un punto clave; para el cálculo de la invariancia de rotación se toma cada punto clave y se le asigna una orientación, se analiza el punto clave con sus vecinos y se obtiene la magnitud y la dirección del gradiente (Figure 6) (Lowe, 2004).

También es empleado para buscar el mismo punto clave en otra imagen, esto se logra con el cálculo de la invariancia de escala y la invariancia de rotación. Cabe mencionar que el método SIFT no es libre para el uso comercial, OpenCV cuenta con una extensión de SIFT pero de forma de contribución de la comunidad (OpenCV Open Source Computer Vision, 2015).

Figura 6. Método SIFT para detección de puntos clave

2.5.3 Funciones robustas aceleradas (SURF)

SURF es el acrónimo de Speeded up robust features (funciones robustas aceleradas) es un método tres veces más rápido que SIFT, emplea filtros de caja simple que se acercan al filtro Gaussiano (Bay, Tuytelaars, & Gool, 2008), al igual que el método SIFT no es libre para el uso comercial, OpenCV cuenta con una extensión de SURF pero de forma de contribución de la comunidad.

El método tiene la ventaja que el filtro de caja es calculado fácilmente empleando imágenes integrales, utiliza la matriz de Hess para detectar descriptores basados en la distorsión y puede ser utilizado en paralelo para diferentes escalas (**Figure 7**) (Mordvintsev & Abid, 2013).

Figura 7. Método SURF para detección de puntos clave

2.5.4 Características de la prueba de segmento acelerado (FAST)

Fast es el acrónimo de Features from accelerated segment test (características de la prueba de segmento acelerado) es un método rápido que no requiere mucho esfuerzo computacional puede ser empleado tanto en dispositivos móviles como aplicaciones en tiempo real porque no requiere altos requisitos en el sistema empleado, cabe recalcar que solo es empleado para la detección de

puntos clave para posteriormente realizar el cálculo de descriptores con los métodos anteriormente mencionados (Howse et al., 2016; Rosten & Drummond, 2006).

Para un detector de alta velocidad se ha empleado una máquina de aprendizaje para la detección de esquinas donde se tiene un conjunto de imágenes, se emplea el algoritmo FAST encada imagen para la detección de puntos característicos, posteriormente se almacenan 16 pixeles que se encuentran a su alrededor para el cálculo de un vector característico que cuenta con tres estados (más oscuro, similar, más brillante), se selecciona al pixel con mayor información que es una esquina, esto se aplica recursivamente hasta que la entropía sea cero (Figure 8) (Rosten & Drummond, 2006).

Figura 8. Método FAST para detección de puntos clave

2.5.5 Características básicas binarias robustas e independientes (BRIEF)

BRIEF es el acrónimo de Binary robust independent elementary features (características básicas binarias robustas e independientes) es un método rápido para el cálculo de descriptores de características. A diferencia de los métodos SIFT y SURF no utiliza mucha memoria, el método selecciona un parche de imagen suavizado donde selecciona un conjunto de pixeles proporcionando una alta tasa de reconocimiento si no existe mucha rotación en el plano (**Figura 9**). El método no es libre para el uso comercial, OpenCV cuenta con una extensión de BRIEF pero de forma de contribución de la comunidad (Calonder, Lepetit, Strecha, & Fua, 2010).

Figura 9. Método BRIEF para detección de puntos clave

2.5.6 Orientado FAST y Rotado BRIEF (ORB)

ORB fue lanzado oficialmente por OpenCV Labs, es un método que no requiere mucho esfuerzo computacional, tiene mayor rendimiento y de código abierto. ORB es la combinación de los métodos FAST y BRIEF, pero con varias modificaciones que mejoran el rendimiento. Emplea FAST para el cálculo de los puntos clave, posteriormente aplica el método de Harris para la detección de esquinas de los N primeros puntos clave y utiliza pirámides para las características a múltiple escala (OpenCV Open Source Computer Vision, 2018). Una desventaja del método por la utilización de FAST produce que no puede calcular la orientación, por lo que realiza el cálculo del centroide con mayor ponderación dentro del parche con la esquina en el centro, la dirección que se describe desde la esquina al centroide es la orientación. Se emplea el cálculo tanto para el par (x, y) que se encuentran en la región que describe el radio donde el valor de r es el tamaño descrito del parche (Rublee, Rabaud, Konolige, & Bradski, 2011).

Como se ha mencionado al emplear BRIEF para la detección de descriptores, no tiene buenos resultados, para solucionar esto emplea la dirección de los puntos clave para cualquier conjunto de

características para el par (x, y) (OpenCV Open Source Computer Vision, 2018; Rublee et al., 2011).

Figura 10. Método ORB para detección de puntos clave

2.6 Correspondencia de puntos entre imágenes

Se refiere a la capacidad de encontrar el mismo punto en una segunda imagen, métodos como RANSAC (RANdom SAmple Consensus) permite identificar correspondencias entre conjuntos coincidentes (vecino más cercano) a pesar de existir varios puntos candidatos, es utilizado para la estimación de estructura y movimiento y el reconocimiento de objetos (Zhang & Kosecka, 2006).

RANSAC es un algoritmo iterativo para identificar los parámetros de un modelo matemático de un conjunto de datos (Niedfeldt & Beard, 2015), para encontrar correspondencias emplea una transformación de los puntos, así en cada iteración va segregando los puntos con mayor error, generando un único inlier que es el punto cuyo error es menor a un umbral, este inlier corresponde al punto de la segunda imagen. (López Radcenco, 2010).

Adicionalmente existe el método de flujo óptico Lucas-Kanade donde el flujo óptico se describe como un patrón de movimiento aparente de los objetos entre imágenes consecutivas, el cálculo es

realizado de forma vectorial 2D donde la medición del movimiento es un vector de desplazamiento que muestra el movimiento del primer cuadro al segundo, es por ello que la aplicación de Lucas-Kanade crea una matriz 3x3 de cada punto de interés del primer cuadro que muestre movimiento y lo busca en el segundo cuadro, existe la función en OpenCV que permite encontrar el flujo óptico a través del método Lucas-Kanade cv2.calcOpticalFlowPyrLK(), que recibe como parámetros la imagen del primer, segundo cuadro en escala de grises y los puntos que se hayan considerado como puntos clave (OpenCV Open Source Computer Vision, 2017).

2.7 Visión estereotípica

La visión estereotípica toma como referencia el modelo biológico del desplazamiento que existe entre los ojos, el observador mira de manera simultánea dos imágenes del mismo objeto, el desplazamiento que existe entre los dos ojos genera una percepción diferente del objeto lo que da una idea de cómo está estructurada la escena (Cárdenas, Morales, & Caycedo, 2015).

Como las imágenes se encuentran ligeramente desplazadas se puede calcular la diferencia relativa de la posición del objeto, esto se denomina disparidad (**Figura 11**) (Miguel, Hernández, & Martinsanz, 2011).

El cálculo de la disparidad funciona cuando el centro de proyección de las dos imágenes es paralelo entre sí (Fernández et al., 2010).

2.8 Geometría epipolar

Para solventar el problema que existe cuando los centros de proyección no son paralelos (que se encuentran en diferente posición y rotación una con respecto a la otra $[R|t]$) se emplea la geometría epipolar, la geometría epipolar es aquella que relaciona los planos de las imágenes con sus planos de proyección tomando en cuenta los centros de proyección que se unen mediante una

línea imaginaria que es la línea base (**Figura 12**), las proyecciones dependen de los parámetros intrínsecos y extrínsecos de la cámara (Javier & Quintana, 2011).

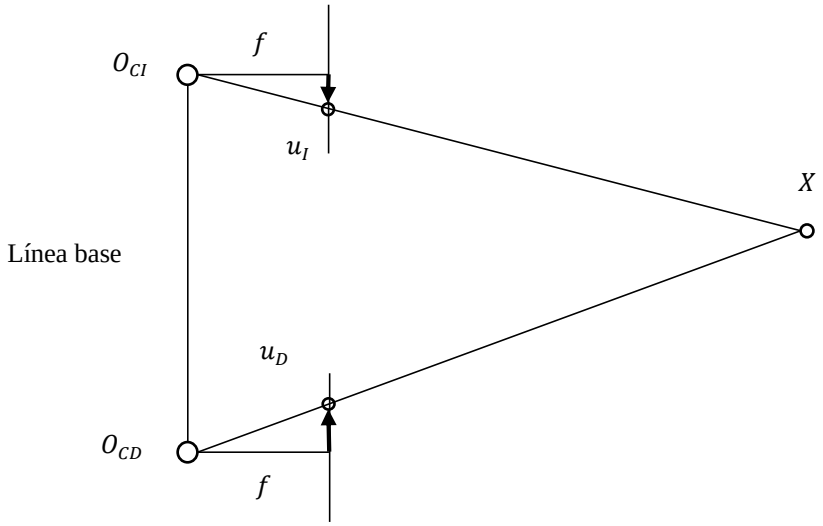


Figura 11. Visión estereotípica

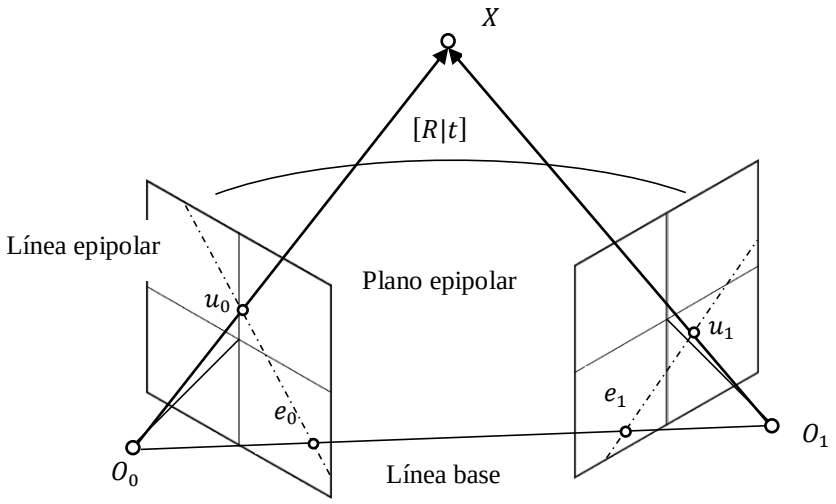


Figura 12. Geometría epipolar

El punto X en coordenadas externas tiene una proyección en las dos imágenes u_0 y u_1, los puntos en las imágenes que genera la línea base e_0 y e_1 se denominan epipolos. El plano que se genera entre el punto X y los epipolos se llama plano epipolar; y las líneas que se generan entre los epipolos y los puntos de proyección se llama línea epipolar l (Calderón, 2012).

Como se puede observar tanto en la imagen izquierda el punto u_0 es el punto correspondiente de X en la imagen derecha u_1, por lo que se debe tener en consideración lo siguiente (Calderón, 2012; Javier & Quintana, 2011):

- Cada punto en coordenadas externas se encuentra en un plano epipolar que cruza cada imagen en la línea epipolar.
- El punto de proyección de una imagen debe corresponder a un único punto de la siguiente imagen que se encuentre en la línea epipolar, se conoce como restricción epipolar.

2.8.1 Matriz fundamental

La matriz fundamental relaciona el sistema de coordenadas de la imagen de las dos cámaras (Brendstrup, 2013), la correspondencia de un puntos entre las dos imágenes se representa mediante el punto u_0 que genera una línea epipolar l_1 en la segunda imagen, el punto u_1 debe estar en la línea epipolar l_1. Este proceso genera un mapa proyectivo que es una correlación singular de puntos a líneas, es representado mediante una matriz F llamada matriz fundamental (Hartley & Zisserman, 2004).

$$u_0 \rightarrow l_1$$

2.8.1.1 Derivación geométrica

La matriz fundamental se puede derivar de la correspondencia de puntos de la primera imagen con respecto a la segunda imagen, generando una homografía H que envía cada punto de la primera imagen a la segunda imagen $u_1 = Hu_0$, empelando este concepto se calcula la línea que contiene al epipolo de la segunda imagen (Javier & Quintana, 2011).

$$l_1 = e_1 \times u_1 = e_1 \times Hu_0 = Fu_0$$

Donde la matriz fundamental representa la función para el cambio del plano proyectivo de la primera imagen al conjunto de líneas epipolares empleando el epipolo (Hartley & Zisserman, 2004):

$$F = e_1 \times H$$

2.8.1.2 Derivación algebraica

La matriz fundamental se puede derivar con la utilización de las matrices proyectivas P de las dos cámaras para cada punto $u = PX$, donde para el punto $u_0 = P_0X$ genera una semi recta en el espacio que intercepta al punto en coordenadas externas, del punto en coordenadas externas se puede expresar como $X = P_0^{-1}u_1$ en consecuencia la línea epipolar que contiene al punto genera una homografía $H = P_1P_0^{-1}$ con respecto a las matrices de proyección de las cámaras $u_1 = P_1P_0^{-1}u_0$ donde (Hartley & Zisserman, 2004; Javier & Quintana, 2011):

$$l_1 = e_1 \times u_1 = e_1 \times (P_1P_0^{-1})u_0 = Fu_0$$

$$F = e_1 \times P_1P_0^{-1}$$

Siendo $P_0 = K_0[I|0]$ y $P_1 = K_1[R|t]$ se descompone en la siguiente matriz:

$$F = K_1^{-T}RK_0^T[K_0R^Tt]_x$$

2.8.2 Matriz esencial

La matriz esencial relaciona el sistema de coordenadas de la cámara de las dos cámaras, se expresa como (Brendstrup, 2013):

$$E = R[t]_x$$

Donde R y t representan los parámetros extrínsecos de la cámara, además $[t]_x$ se denota como:

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

2.8.3 Relación entre la matriz fundamental y la esencial

Al utilizar la matriz fundamental tiene como restricción que al multiplicar un punto en coordenadas de la imagen de la primera imagen con la matriz fundamental y su correspondiente en la segunda imagen el resultado debe ser cero, esto se debe a que el punto pertenece a una línea epipolar deforma que al reemplazar el valor del punto en la ecuación de la línea el resultado será (Brendstrup, 2013; Hartley & Zisserman, 2004):

$$u_1^T l_1 = 0$$

Como $l_1 = Fu_0$

$$u_1^T F u_0 = 0$$

Además se cuenta con la restricción de la matriz esencial que al multiplicar un punto en coordenadas de la cámara de la primera imagen con la matriz esencial y su correspondiente en la segunda imagen el resultado debe ser cero (Brendstrup, 2013; Hartley & Zisserman, 2004):

$$x_1^T E x_0 = 0$$

De tal forma la matriz esencial se puede expresar en términos de la matriz fundamental como y viceversa (Hartley & Zisserman, 2004):

$$E = K_1^T F K_0$$

$$F = K_1^{-T} E K_0^{-1}$$

2.8.4 Descomposición en valores singulares (SVD)

Como se analizó la matriz esencial se deriva de las propiedades extrínsecas de la cámara, pero al no poseer estos valores se puede calcular a partir de la relación con la matriz fundamental. Este cálculo se realiza mediante la descomposición en valores singulares de la matriz esencial, SVD es un método de factorización de modelo ("Descomposición en Valores singulares(SVD)," 2010):

$$A = U\Sigma V^T$$

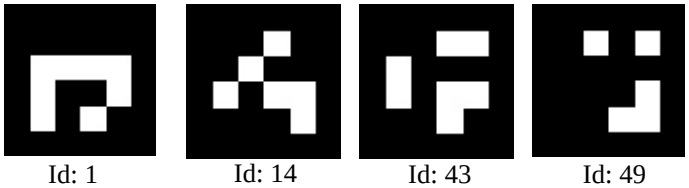
Donde las matrices U y V son ortogonales de la forma $m \times n$ y $n \times n$ respectivamente, Σ es una matriz diagonal $n \times n$ y la matriz $A = E$, para el cálculo de la matriz V se debe determinar la base ortogonal compuesta de los auto vectores de $A^T A$, para el cálculo de U depende de los auto vectores de $A^T A$ que no son nulos por lo tanto se pueden normalizar los valores para que sean ortonormales, el cálculo de la matriz Σ tiene una forma de bloques de los valores singulares de A (Calabuig, Garcia, & Sánchez, 2015).

El resultado son cuatro soluciones posibles y solo una de ellas es la matriz de la cámara, un método para confirmar el resultado de la factorización SVD en la que todos los puntos clave se encuentre en ambas cámaras (Howse et al., 2016).

2.9 Marcadores ArUco

Es de mucha importancia en aplicaciones de visión por computadora el uso de la estimación de postura por ejemplo realidad aumentada, navegación de robots etc. Para ello se emplea la librería de OpenCV ArUco (Garrido, Muñoz, Madrid, & Marín, 2014), que es una librería popular para la detección de marcadores.

Los marcadores ArUco son recuadros que se encuentran compuestos por una matriz binaria interna que corresponde a un identificador y un borde negro para una detección y codificación rápida del recuadro (**Figura 13**). Los marcadores pueden ser generados de manera computacional dependiendo de las dimensiones de la matriz y el tamaño del marcador en bits.



Id: 1 Id: 14 Id: 43 Id: 49

Figura 13. Marcadores ArUco

2.10 Sistema operativo robótico (ROS)

ROS es el acrónimo de robot operating system (sistema operativo robótico) es un entorno de software de robots flexible que contiene varias herramientas para la disminuir la tarea de representar un comportamiento robótico (Santos, Portugal, & Rocha, 2013).

Tiene la característica utilizar un diseño modular y distribuido, lo que se refiere a que se pueden crear múltiples paquetes (nodos) que interactúan al recibir y enviar mensajes para ejecutar una tarea satisfactoria (Cashmore et al., 2015).

2.10.1 ROS máster

ROS está estructurado de forma que existe un nodo máster que permite que los diferentes nodos puedan interactuarse entre sí, de forma que no se debe especificar la ruta especifica en que los nodos se comuniquen entre ellos (Wasowski, 2018).

2.10.2 Nodos

Los nodos son archivos ejecutables dentro de un paquete creado en ROS, emplean una biblioteca tipo cliente para publicar o suscribir mensajes entre nodos (Wasowski, 2018).

2.11 Aprendizaje profundo

Desde el 2010 toma fuerza el termino aprendizaje profundo respaldado por Nvidia (**Figura 14**), es un subconjunto de la inteligencia artificial y el aprendizaje automático; además es empleado para varios avances en la tecnología como autos que se manejan por si solos, asistentes de voz, etc. Su principal característica es que puede aprender automáticamente sin utilizar reglas que sean introducidas por el ser humano, tiene un alto grado de precisión ya que se puede introducir una gran cantidad de datos al inicio y durante el proceso (NVIDIA, 2010).

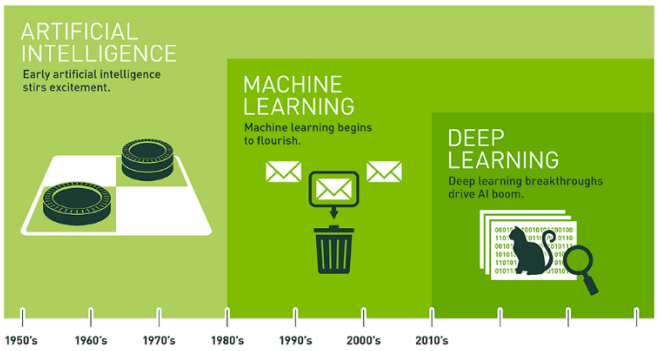


Figura 14. Cronología del desarrollo de la IA
Fuente: (NVIDIA, 2010)

El aprendizaje profundo está compuesto por modelos computacionales multi capa que permite aprender con varios niveles de abstracción, se emplea algoritmos de retro propagación que permite que la maquina pueda cambiar sus parámetros internos a partir de la capa anterior. Uno de los ejemplos empleados en visión por computadora son las redes neuronales convolucionales (LeCun, Bengio, & Hinton, 2015).

2.12 Redes neuronales convolucionales (CNNs)

La red neuronal convolucional (convolutional neural networks) es una red neuronal con una o múltiples capas del tipo feed-forward (Bakker, 2017), la diferencia de las redes neuronales convencionales es que emplean operaciones de convolución. Tiene un gran desempeño en la visión por computadora para la detección o categorización de objetos, clasificación de imágenes, etc. (Loncomilla, 2016). Por ende tienen mayor eficiencia al trabajar con imágenes multi escala al optimizar recursos computacionales a diferencia de las redes neuronales convencionales (Perez, Serrano, Acha, Serrano, & Linares, 2013).

La estructura básica de una red neuronal convolucional cuenta con una capa de reducción de parámetros para extraer las características comunes, una capa convolucional y una capa clasificadora para el resultado final de la red (Pacheco, 2017).

La capa de reducción es aquella en la cual se normaliza la entrada para que sea empleada en la red, se utilizan los valores de ancho, largo y los canales que cuenta la imagen (Xie, Wang, Wei, Wang, & Tian, 2016).

La capa convolucional es aquella que es empleada como filtro llamada comúnmente campo receptivo, esta capa se ajusta para la extracción de características. La capa consta con cuatro hiper parámetros (Xie et al., 2016):

- Numero de filtros.
- Dimensión.
- Stride que sirve para el salto de pixel dentro del filtro.
- Pad que sirve para iniciar el recorrido del campo.

La capa de clasificación es empleada para abstraer la información de la entrada, se emplea la función de activación softmax para el resultado final de la red (Perez et al., 2013).

Para implementar este tipo de redes neuronales se debe de contar con varias características de hardware para la optimización de recursos del sistema, una de las ventajas que existe en el mercado son las GPU (unidad de procesamiento gráfico) con núcleos CUDA que provee la empresa Nvidia en sus tarjetas gráficas (Bergstra, Breuleux, & Bastien, 2010).

2.12.1 Arquitectura unificada de dispositivos de computo (CUDA)

CUDA es el acrónimo de Compute unified device architecture (arquitectura unificada de dispositivos de computo) es una tecnología desarrollada por Nvidia para aprovechar el cálculo en paralelo de las tarjetas GPU, es empleada en varios estudios como el procesamiento de imágenes, simulación de fluidos, etc. (NVIDIA Corporation, 2018b).

La arquitectura que presentan las GPU con núcleos CUDA permiten realizar co procesamiento compartido de la CPU con la GPU, esta arquitectura permite a cada núcleo contar con recursos como memoria y registros, lo que permite que el procesamiento se centre en el mismo chip sin necesidad de enviarla a la CPU (NVIDIA Corporation, 2018b; Pérez, Cámara, & Sánchez, 2016).

Se emplean lenguajes de alto nivel para la programación de los núcleos de la GPU, dentro de la programación se utiliza el kernel que es aquel que permite la ejecución del proceso en paralelo como hilos que pueden ser ubicados de manera organizada dentro de bloques distribuidos o en forma de malla. Este tipo de organización permite tener varias dimensiones que al ejecutarse interactúan entre hilos. (Pérez et al., 2016).

2.13 TensorFlow

Es una biblioteca de código abierto desarrollada por Google Brain y liberada para el público en general en el 2015 para el cálculo numérico de alto rendimiento; es utilizado ampliamente en campos de la ciencia, la investigación de aprendizaje profundo y automático tiene una arquitectura flexible que puede ser implementado en CPU, GPU o TDU. Emplea gráficos de flujo de datos que representan la computación, el estado compartido y las operaciones que cambian de estado (Abadi et al., 2016; Tang, 2010).

Al emplear flujos de datos los vértices actualizan el estado mutable o representan los cálculos que posee, los bordes contienen matrices multidimensionales llamadas tensores que se encuentran entre nodos, de esta forma se presentan diferentes sistemas de paralización. Un ejemplo de flujo de datos (**Figura 15**) es una tubería de entrenamiento que permite leer los datos de entrada, realizar un pre procesamiento, posteriormente el entrenamiento y un estado que sirve como punto de control, este esquema es representado en sub gráficos. De esta forma permite realizar

actualizaciones in situ de manera que se propaga la actualización de forma paralela (Abadi et al., 2016).

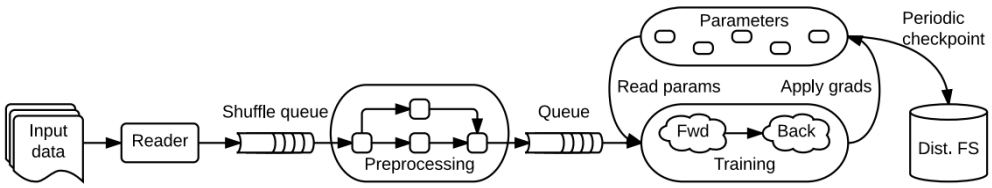


Figura 15. Flujo de datos
Fuente: (Abadi et al., 2016)

2.13.1 Elementos del gráfico de flujo de datos

Se describe a continuación a detalle cada uno de los elementos de un gráfico de flujo de datos:

2.13.2 Tensores

Son datos que representan las entradas o los resultados de una operación de estructura multi dimensional con tipos de datos como int32, float64 o string. Los tensores pueden tener una o varias dimensiones para codificar datos dispersos con diferentes números de elementos (Abadi et al., 2016).

2.13.3 Operaciones

Las operaciones toma uno o varios tensores de entrada y producen uno o varios tensores de salida, puede tener varios atributos para determinar su comportamiento. Una operación puede variar en el tiempo de compilación dependiendo del tipo de dato esperado (Abadi et al., 2016).

2.13.4 Operaciones con estado: variables

Dentro de las operaciones se puede tener un estado mutable que se emplea para leer o escribir un búfer mutable. Una variable contiene un identificador que se emplea para referenciar el búfer como un tensor denso (Abadi et al., 2016).

2.13.5 Operaciones con estado: colas

Dentro de la programación de TensorFlow se admiten varios tipos de colas como la FIFOQueue que tiene el orden primero en entrar primero en salir, contiene varios tensores para el acceso simultaneo, la implementación de otros tipos de colas puede garantizar que los tensores sean presentados de forma adecuada. Además las colas contienen identificadores que puede ser empleado en operaciones (Abadi et al., 2016).

CAPÍTULO III

IMPLEMENTACIÓN Y RESULTADOS

3.1 Introducción

El sistema se encuentra estructurado con tres mini drones con cámara WIFI y una cámara general (**Figura 16**) para la detección y ubicación en el espacio de diez esferas. Se emplea detección de objetos en las imágenes capturadas por los mini drones para ubicar a las esferas y para la ubicación e identificación de los mini drones se emplea las imágenes capturadas por la cámara general que buscara los marcadores ArUco.

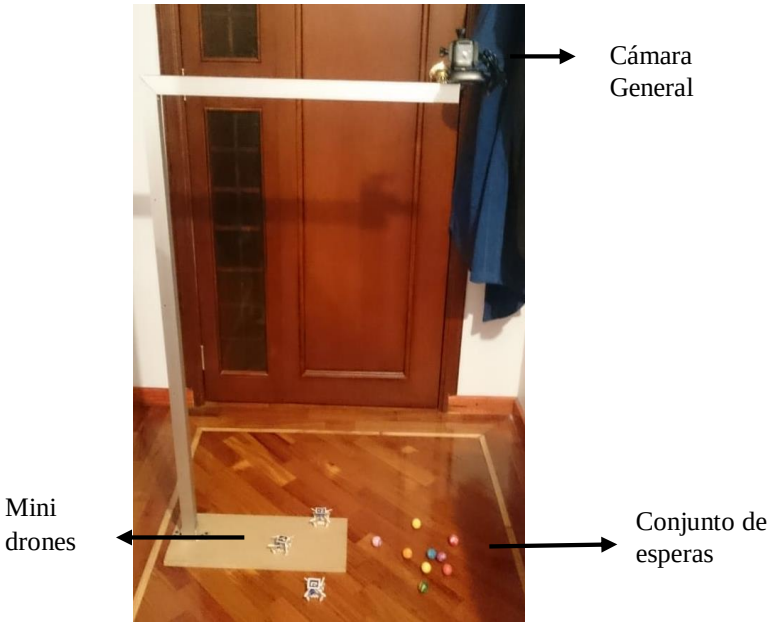


Figura 16. Modelo del sistema

Cada imagen obtenida por el primer mini drone es emparejada con el siguiente mini drone generando una visión estereotípica de dos vistas para mayor soporte se generó un tercer emparejamiento con el tercer mini drone. El proceso de emparejamiento se refiera a la capacidad de extraer características de las imágenes de los mini drones y buscar su correspondencia en las siguientes imágenes, como se ha utilizado redes neuronales para la detección de las esferas se

segmenta el análisis de correspondencia a solo las esferas, se empleó el método ORB para extracción de características por ser un método rápido y robusto al implementar. Una vez ubicadas las esferas correspondientes entre las imágenes se procede al proceso de reconstrucción estereotípica utilizando los parámetros intrínsecos de las cámaras utilizando el proceso de triangulación (**Figura 17**). Para el desarrollo del trabajo de investigación se empleó el IDE Pycharm para compilar código Python versión 2.7.

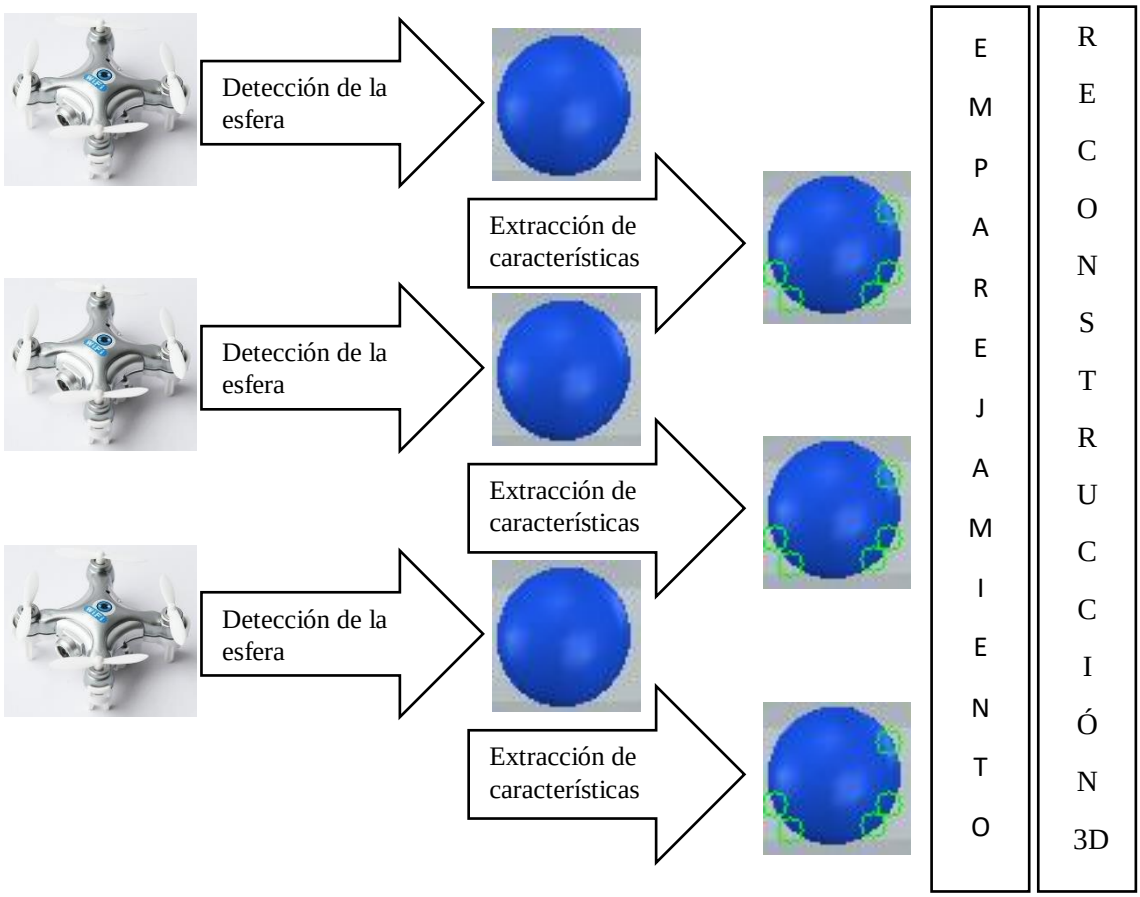


Figura 17. Esquema de desarrollo del proyecto

3.2 Mini drone

El modelo Cheerson cx-10w (**Figura 19**) es un mini drone que permite ser controlado por WIFI mediante una aplicación Android, además de la incorporación de una cámara de 0.3MP que permite

volar en primera persona (FPV, firs person view). Además, incorpora un giroscopio de 6 ejes para proveer un mejor control de estabilidad (**Tabla 1**).

Tabla 1
Parámetros técnicos

Dimensiones	42x42x25 mm
Diámetro de la cuchilla	30 mm
Tiempo de carga	30 minutos
GYRO	Si
Cámara	0.3 MP
Peso	17g
Batería	150 mAh 3.7V Lipo
Tiempo de vuelo	4 minutos
Distancia Wifi	1.15 a 18 m

Fuente: (Cheerson Hobby Technology Company, 2016)

La aplicación Android está diseñada de forma que la comunicación entre el drone y el Smartphone se lleve a cabo mediante sockets. Para inicializar la comunicación entre el drone y el Smartphone la aplicación envía una secuencia de datos (**Anexo 1**), para mantener la comunicación se envía una secuencia de datos (**Anexo 2**) cada segundo.

La comunicación entre el mini drone y el Smartphone es de uno a uno por lo que solo un dispositivo puede acceder al control de él, además el video es enviado a la aplicación en formato h264 que es un formato de compresión de video.

3.3 Distribución de red

Como la comunicación entre el mini drone y el dispositivo de control es uno a uno se plantea la siguiente distribución de la Red de comunicación (**Figura 18**). Cada mini drone cuenta con comunicación WIFI por la cual recibe los parámetros de vuelo y envía el video en formato h264, estos parámetros serán controlados mediante una Raspberry Pi 3.

Cada Raspberry Pi estará conectada a un drone mediante WIFI y a un router mediante ETHERNET. La red WIFI del mini drone no cuenta con clave de acceso y por ende se asigna una

IP DHCP para cada dispositivo, la comunicación ETHERNET contará con direcciones IP estáticas que servirá para compartir las imágenes provenientes del drone a una laptop que cuenta con las capacidades para realizar un procesamiento de la red neuronal que se ha creado. Cabe destacar que los mini drones no cuentan con un estándar WIFI en sí, sino una variante que no permite un alcance mayor a 5m. Se utiliza a las Raspberry Pi 3 para comunicación de uno a uno con los mini drones ya que el sistema WIFI de los mini drones no puede conectarse a un router y generar una red con los demás mini drones como es el caso de los modelos Parrot y ArDrone que cuentan con comunicación WIFI con soporte a telnet, donde se puede manipular los valores de red de los drones.

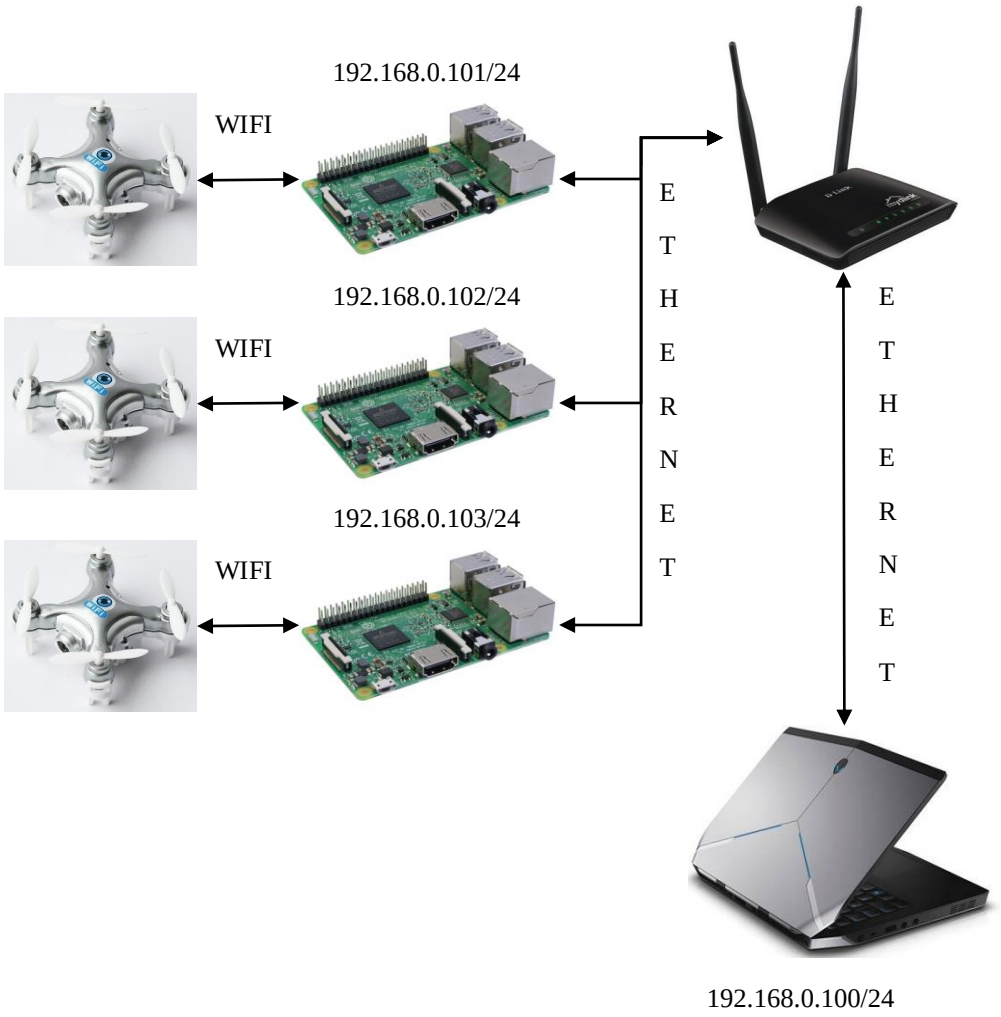


Figura 18. Distribución de la red

Figura 19. Mini drone cheerson cx10-w
Fuente: (Cheerson Hobby Technology Company, 2016)

3.3.1 Comunicación con los mini drones

Cada mini drone se comunicará mediante WIFI a una sola Raspberry Pi 3. Para que la comunicación sea prolongada se ha modificado la aplicación JAVA de (Ciano, 2017) donde emplea protocolos FFPLAY, FFMPEG para video que se encuentra en formato h264 y sockets para el control de vuelo.

Al realizar la conexión entre el mini drone y la Raspberry Pi, el mini drone asigna una dirección IP DHCP a la Raspberry Pi, pero al realizar la recepción y envío de datos para configurar al mini drone emplea la dirección IP 172.16.10.1, para mantener la comunicación se emplea un Heartbeat descrita (**Anexo 2**) en el puerto 8888. De esta manera se establecerá una comunicación prolongada tanto para el envío de datos de inicialización y recepción de video.

Como la Raspberry Pi no cuenta con gran capacidad de procesamiento las imágenes son enviadas a la computadora principal mediante comunicación entre sockects (**Tabla 2**).

Tabla 2
Direcciones IP y Puertos de los Sockets

Dirección IP de cada Raspberry Pi	IP de la computadora principal	Puerto
192.168.0.101	192.168.0.100	8890
192.168.0.102	192.168.0.100	8891
192.168.0.103	192.168.0.100	8892

La aplicación JAVA (**Figura 20 Anexo 3**) cuenta con dos botones:

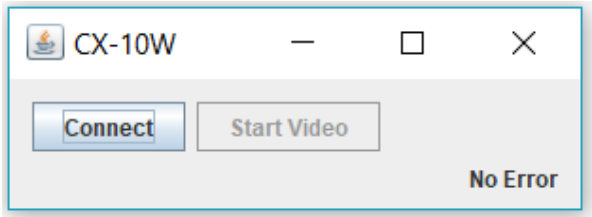


Figura 20. Aplicación JAVA de inicialización de comunicación

El primer botón Connect se encuentra desbloqueado para realizar la comunicación entre el drone y la computadora principal, una vez realizada la comunicación se desbloqueará el botón Start Video que inicializará el socket para la transmisión de video a la computadora principal, el socket se encuentra configurado como cliente de forma que al iniciar los nodos ROS. Para el video de los mini drones se debe iniciar el socket servidor de lo contrario mostrará error (**Figura 21**).

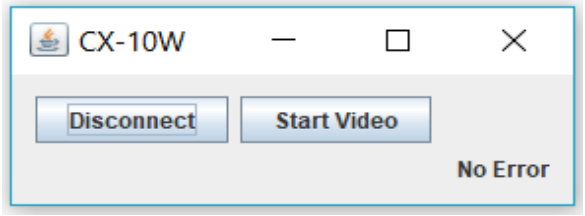


Figura 21. Comunicación inicializada

En la consola se puede observar que al enviar los protocolos de inicialización el mini drone responde en formato hexadecimal para verificar la conexión, lo cual es mostrado por consola, la respuesta del mini drone no cambiar al realizar múltiples conexiones a la red WIFI (**Figura 22**).

Una vez inicializada la conexión se puede observar el envío del Heartbeat que sirve para mantener la conexión, el mini drone procesa la primera conexión y envía un mensaje de confirmación, esta respuesta se muestra por consola y es actualizada la verificación de la conexión cada segundo (**Figura 23**). Por ende, esta primera parte es esencial para mantener la comunicación con los mini drones para su posterior utilización en el desarrollo de la tesis.

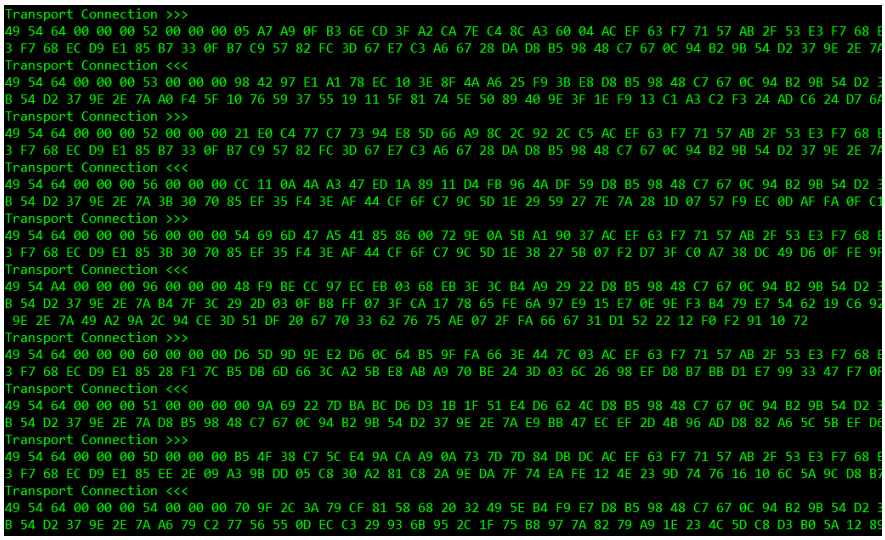


Figura 22. Envío y recepción de datos del mini drone

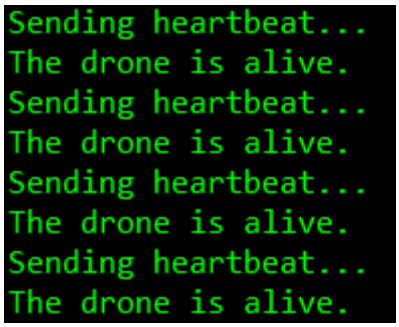


Figura 23. Envío de Heartbeat

3.4 Distribución de nodos ROS

Gracias a que ROS provee un diseño modular y distribuido, se plantea el siguiente sistema de nodos desarrollado en lenguaje Python, se ubican los nodos dentro de la carpeta src (**Figura 24**).

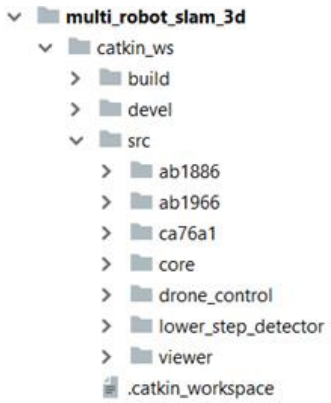


Figura 24. Distribución de nodos en el IDE Pycharm

3.4.1 Nodos de recepción y envío de imágenes

Como se había mencionado anteriormente la comunicación con los mini drones con las Raspberry Pi 3 es de uno a uno, los sockets están configurados como clientes y son comunicados a la computadora principal para transformarlos en mensajes para los nodos ROS (**Anexo 4**), cada nodo ha sido nombrado como el SSID de la red WIFI de cada mini drone (**Figura 25**).

El mensaje que se envía desde los nodos ROS que contienen las imágenes al nodo principal se describe (**Tabla 3**). Para que no se realice retardos en el envío de las imágenes se ha empleado una compresión en formato JPG.

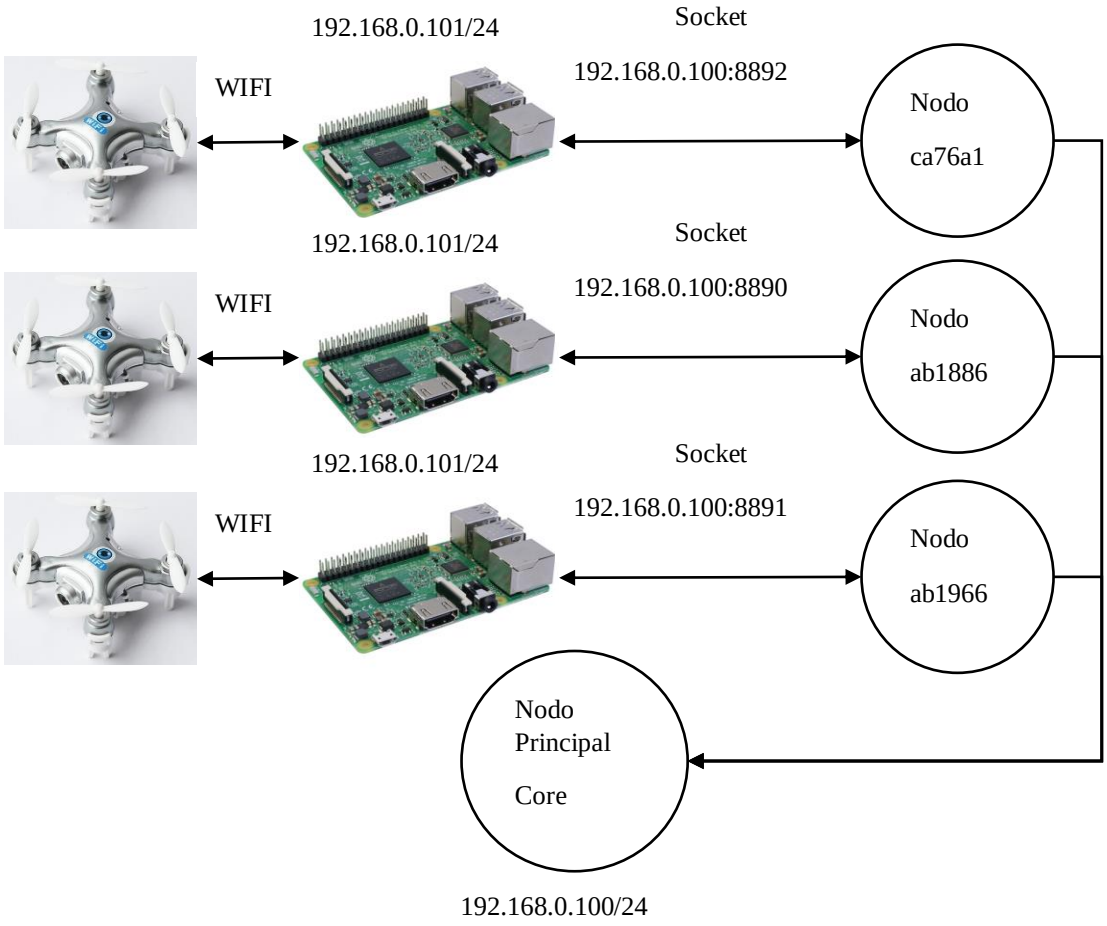


Figura 25. Distribución de nodos ROS

Tabla 3
Nodos ROS

Nombre del Nodo	Nombre del tema que se envía	Tipo
ab1966	camera/image_raw_0	CompressedImage
ca76a1	camera/image_raw_1	CompressedImage
ab1886	camera/image_raw_2	CompressedImage

3.4.1.1 Nodo ab1966

Dentro del nodo correspondiente al primer mini drone, se presenta la estructura (**Figura 26**) donde como primer paso se asigna la dirección del nodo principal ROS_MASTER_URI=192.168.0.100 con el puerto TCP_PORTS=8892 como resultado se genera la siguiente dirección 'tcp://ROS_MASTER_URI:TCP_PORTS' para recibir las imágenes desde la aplicación JAVA. Además, se asigna un id=0 que corresponde al identificador del mini drone en el sistema.

Una vez asignadas las variables para la recepción de las imágenes a través de un socket se utiliza la función de OpenCV de captura de video cv2.VideoCapture donde se le asigna la dirección del socket cv2.VideoCapture('tcp://ROS_MASTER_URI:TCP_PORTS').

Para la transformación de las imágenes a mensajes ROS se debe crear un tema que sirve para enviar el mensaje (camera/image_raw_0) con el tipo CompressedImage. Además, se asignó una tasa de velocidad de envío de 20hz.

En el proceso de compresión de la imagen a formato JPG se debe configurar de acuerdo al tipo de mensaje que se envía, como se describe a continuación: msg: Crea un objeto CompressedImage , msg.header.stamp: se refiere a un atributo de encabezado del mensaje, se envía el tiempo actual de ejecución de la sentencia y es empleado para la sincronización de los demás nodos, msg.format: corresponde al formato que se desea realizar la compresión en este caso se asigna en JPEG,

msg.data: corresponde a la imagen, se utiliza una función de OpenCV para la codificación de la imagen en formato JPEG.

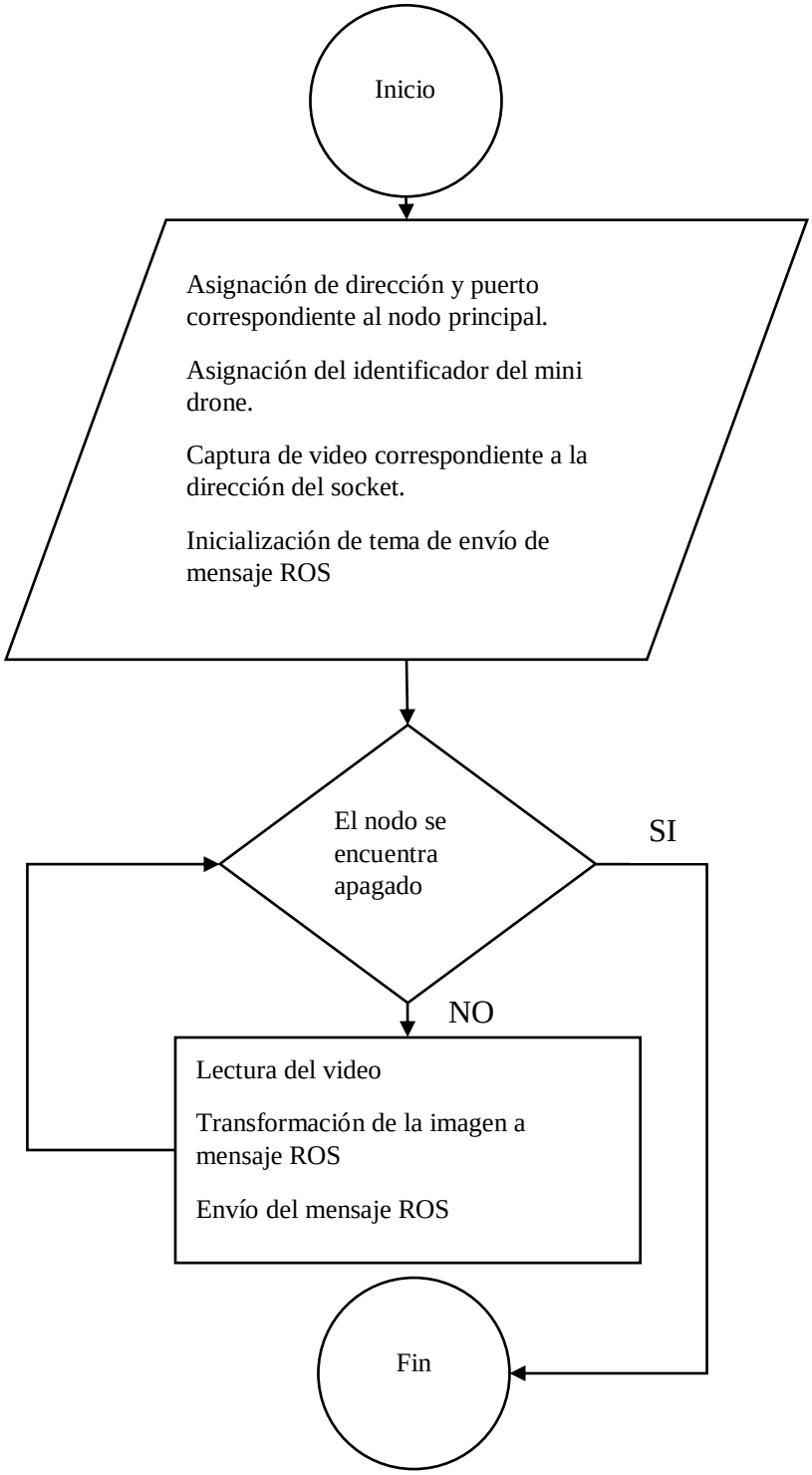


Figura 26. Diagrama de flujo de transformación de socket a mensaje ROS

3.4.1.2 Nodo ca76a1

Dentro del nodo correspondiente al segundo mini drone, se presenta la estructura (**Figura 26**) donde como primer paso se asigna la dirección del nodo principal ROS_MASTER_URI=192.168.0.100 con el puerto TCP_PORTS=8890 como resultado se genera la siguiente dirección 'tcp://ROS_MASTER_URI:TCP_PORTS' para recibir las imágenes desde la aplicación JAVA. Además, se asigna un id=1 que corresponde al identificador del mini drone en el sistema.

Una vez asignadas las variables para la recepción de las imágenes a través de un socket se utiliza la función de OpenCV de captura de video cv2.VideoCapture donde se le asigna la dirección del socket cv2.VideoCapture('tcp://ROS_MASTER_URI:TCP_PORTS').

Para la transformación de las imágenes a mensajes ROS se debe crear un tema que sirve para enviar el mensaje (camera/image_raw_1) con el tipo CompressedImage. Además, se asignó una tasa de velocidad de envío de 20hz.

En el proceso de compresión de la imagen a formato JPG se debe configurar de acuerdo al tipo de mensaje que se envía, como se describió anteriormente.

3.4.1.3 Nodo ab1886

Dentro del nodo correspondiente al segundo mini drone, se presenta la estructura (**Figura 26**) donde como primer paso se asigna la dirección del nodo principal ROS_MASTER_URI=192.168.0.100 con el puerto TCP_PORTS=8891 como resultado se genera la siguiente dirección 'tcp://ROS_MASTER_URI:TCP_PORTS' para recibir las imágenes desde la aplicación JAVA. Además, se asigna un id=2 que corresponde al identificador del mini drone en el sistema.

Una vez asignadas las variables para la recepción de las imágenes a través de un socket se utiliza la función de OpenCV de captura de video cv2.VideoCapture donde se le asigna la dirección del socket cv2.VideoCapture('tcp://ROS_MASTER_URI:TCP_PORTS').

Para la transformación de las imágenes a mensajes ROS se debe crear un tema que sirve para enviar el mensaje (camera/image_raw_2) con el tipo CompressedImage. Además, se asignó una tasa de velocidad de envío de 20hz.

En el proceso de compresión de la imagen a formato JPG se debe configurar de acuerdo al tipo de mensaje que se envía, haciendo que la imagen sea enviada a la computadora central por medio del nodo core, el cual recibe el mensaje de las Raspberry Pi 3.

3.4.1.4 Nodo Principal Core

El nodo principal se emplea para la reconstrucción y ubicación en el espacio las esferas. El proceso (**Figura 27**) como primer paso se realiza la asignación de los valores de los parámetros intrínsecos de la cámara, posteriormente se sincroniza los mensajes ROS (camera/image_raw_0, camera/image_raw_1, camera/image_raw_2) correspondientes a cada mini drone, una vez sincronizados los nodos se realiza el proceso para que las imágenes se encuentren preparadas sin distorsión empleando la función de OpenCV undistortion() que utiliza los parámetros intrínsecos de la cámara.

Posteriormente las imágenes sin distorsión son analizadas por la red neuronal que se ha entrenado previamente ubicando a las esferas en recuadros dentro de la imagen. Se realiza extracción y seguimiento de puntos clave de cada par de imágenes, de manera que se pueda realizar la triangulación mediante el modelo epipolar (el proceso se describe a detalle en la sección 3.6). por último, se grafica la ubicación de las esferas en el espacio.

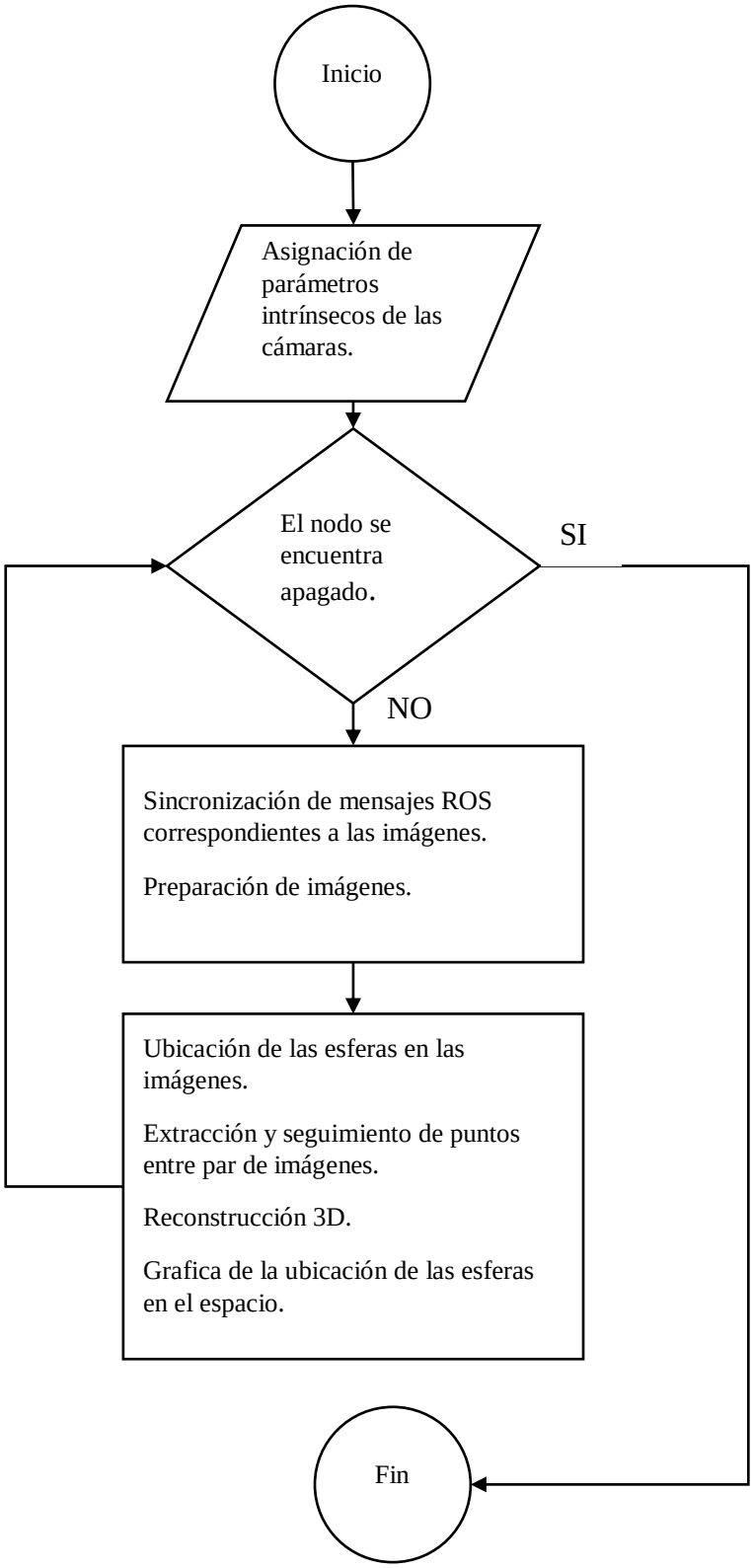


Figura 27. Diagrama de flujo del nodo Core

3.5 Detección de objetos

Con el fin de detectar objetos dentro de las imágenes se ha utilizo la API proporcionada por Google de detección de objetos TensorFlow, la API facilita el entrenamiento de una red neuronal convolucional en base a una red entrenada previamente con un conjunto de datos COCO (Objetos comunes en contexto), conjunto de datos OpenImages y el conjunto de datos Kitti.

Se empleará la API para crear una red neuronal convolucional para detectar objetos: las esferas, se delimita en la clase shpere. Para lo cual se debe considerar los siguientes parámetros:

3.5.1 Capacidad del sistema

Para la utilización de la API se debe contar con la instalación de TensorFlow. La instalación puede utilizar la arquitectura de la CPU o GPU en lenguaje Python, de manera que se empleara la arquitectura GPU ya que el sistema tiene una tarjeta gráfica Nvidia GeForce GTX 970M con núcleos CUDA (**Tabla 4**).

Tabla 4

Especificaciones del motor GTX 970M

Núcleos CUDA	1280
Reloj base (MHz)	924 + Boost
Memoria RAM	3072 MB

Fuente: (NVIDIA Corporation, 2018a)

3.5.2 Modelo pre entrenado

Dentro del repositorio de la API se encuentran modelos pre entrenados, cada modelo cuenta con información de tiempo de ejecución en imágenes de 600x600, estos tiempos fueron tomados con la tarjeta gráfica Nvidia GeForce GTX TITAN X. Además, cuenta con el rendimiento del detector medido en mAP (medida de precisión promedio) que es la capacidad de que la red neuronal es sensible a detectar los objetos de interés en cuadros delimitados. Se listan los modelos de cada conjunto de datos (**Tabla 5-7**).

Tabla 5

Modelos pre entrenados con COCO

Nombre del modelo	Velocidad (ms)	COCO mAP	Salidas
ssd_mobilenet_v1_coco	30	21	Cajas
ssd_mobilenet_v2_coco	31	22	Cajas
ssdlite_mobilenet_v2_coco	27	22	Cajas
ssd_inception_v2_coco	42	24	Cajas
faster_rcnn_inception_v2_coco	58	28	Cajas
faster_rcnn_resnet50_coco	89	30	Cajas
faster_rcnn_resnet50_lowproposals_coco	64		Cajas

Fuente: (COCO, 2017)

Tabla 6

Modelos pre entrenados en Kitti

Nombre del modelo	Velocidad (ms)	COCO mAP	Salidas
faster_rcnn_resnet101_kitti	79	87	Cajas

Fuente: (Wu, 2017)

Tabla 7

Modelos pre entrenados con OpenImages-trained

Nombre del modelo	Velocidad (ms)	COCO mAP	Salidas
faster_rcnn_inception_resnet_v2_atrous_oid	727	37	Cajas
faster_rcnn_inception_resnet_v2_atrous_lowproposals_oid	347		Cajas

Fuente: (Wu, 2017)

Se selecciona el modelo ssd_mobilenet_v1_coco porque presenta una velocidad de treinta (ms) y con precisión de veinte y un (mAP) que se considera perfecto para computadoras, cabe destacar si se desea implementar en dispositivos móviles se debe verificar el modelo pre entrenado.

3.5.3 Conjunto de datos de entrenamiento

Se cuenta con un conjunto de datos personalizados con mil imágenes de la clase esfera, se marcan manualmente la ubicación de los objetos en cada imagen con el programa labelImage (**Figura 28**), las marcaciones son gravadas en archivos xml que cuenta con el nombre del archivo, tamaño de la imagen, la clase a la que pertenece y la ubicación de la esfera en la imagen (xmin, ymin, xmax, ymax) estos valores se encuentran en pixeles.

Figura 28. Datos de entrenamiento

Posteriormente se generó archivos tfrecord de las imágenes para que sean las entradas a la red neuronal utilizando el código de la API de detección de objetos create_tf_record.py (**Anexo 6**).

python create_tf_record.py --data_dir=models/sphere_model --output_dir=data/sphere --label_map_path=data/sphere/sphere_label_map.pbtxt --faces_only=True

Donde:

- data_dir: corresponde a la dirección donde se encuentran las imágenes de entrenamiento.
- output_dir: corresponde a la dirección donde se va a almacenar los datos tfrecord.
- label_map_path: corresponde a los identificadores de la clase que se va a entrenar.
- faces_only: se configura en True ya que se desea delimitar los objetos con recuadros.

3.5.4 Configuración de la tubería de entrenamiento

La API cuenta con archivos de configuración ssd_mobilenet_v1_sphere.config (**Anexo 5**) en los cuales se debe configurar la ubicación del modelo pre entrenado, los datos de prueba y entrenamiento, marcar el número de clases que detectara la red neuronal y finalmente el tamaño de lote para un ciclo de evaluación. El tamaño del lote depende mucho de la capacidad de memoria

ram que cuenta la GPU. Por ende, se ha implementado una configuración de lote de cinco imágenes por ciclo.

3.5.5 Entrenamiento

El proceso de entrenamiento se llevó a cabo durante aproximadamente dos horas donde se generaron 50 mil pasos con una perdida aproximada promedio de 2.468. El grafico de pérdida total tiende a cero, para mejor rendimiento se empleó la última versión de la API de TensorFlow para detección de objetos. El proceso de entrenamiento puede ser retomado con un nuevo grupo de imágenes ya que TensorFlow permite almacenar puntos de control del entrenamiento.

Se utilizó el código de la API de detección de objetos model_main.py (**Anexo 7**) para el entrenamiento de la red neuronal.

PIPELINE_CONFIG_PATH=models/sphere_model/ssd_mobilenet_v1_sphere.config.

MODEL_DIR=models/sphere_model/.

NUM_TRAIN_STEPS=50000.

NUM_EVAL_STEPS=2000.

python model_main.py --pipeline_config_path = ${PIPELINE_CONFIG_PATH} --model_dir = ${MODEL_DIR} --num_train_steps = ${NUM_TRAIN_STEPS} --num_eval_steps = ${NUM_EVAL_STEPS} --alsologtostderr.

Donde los valores configurados como PIPELINE_CONFIG_PATH, MODEL_DIR, NUM_TRAIN_STEPS, NUM_EVAL_STEPS se describen a continuación:

- PIPELINE_CONFIG_PATH: corresponde a la dirección donde se encuentra almacenado el archivo de configuración ssd_mobilenet_v1_sphere.config.
- MODEL_DIR: corresponde a la dirección donde se almacenarán los datos de la red neuronal.

- NUM_TRAIN_STEPS: corresponde al número de iteraciones de entrenamiento.
- NUM_EVAL_STEPS: corresponde al número de iteraciones de evaluación.

Además, al utilizar la API v1.9 presenta el valor de mAP con valores iguales a 0.6094 (medida de precisión promedio) del entrenamiento.

3.5.6 Prueba de la red neuronal

TensorBoard permite verificar como se va ubicando los objetos dentro de la imagen en cada iteración (**Figura 29**).

Figura 29. Proceso de entrenamiento

TensorFlow permite exportar el grafico de inferencia donde se toma en cuenta el punto de control y la configuración de la tubería para ser utilizado posteriormente en la aplicación con el código de la API export_inference_graph.py (**Anexo 8**).

PIPELINE_CONFIG_PATH=models/sphere_model/ssd_mobilenet_v1_sphere.config.

CHECKPOINT_PATH=models/sphere_model/model.ckpt-50000.

OUTPUT_DIRECTORY=inference_graph/sphere_inference_graph.

python export_inference_graph.py --input_type image_tensor --pipeline_config_path ${PIPELINE_CONFIG_PATH} --trained_checkpoint_prefix ${CHECKPOINT_PATH} --output_directory ${OUTPUT_DIRECTORY}.

Donde:

- PIPELINE_CONFIG_PATH: corresponde a la dirección donde se encuentra almacenado el archivo de configuración ssd_mobilenet_v1_sphere.config.
- CHECKPOINT_PATH: corresponde a la dirección del archivo de punto de control del entrenamiento.
- OUTPUT_DIRECTORY: corresponde a la dirección donde se va a almacenar la red neuronal.

Una vez exportado el grafico de inferencia se ha utilizado para la detección de una esfera en una nueva imagen que no se haya utilizado en el entrenamiento, se muestra cajas delimitadas donde se encuentra el objeto de interés (**Figura 30**).

Figura 30. Detección de objetos

3.6 Reconstrucción 3D

Para el sistema de reconstrucción 3D se empleó el nodo principal Core (**Anexo 9**) donde se realiza el proceso de reconstrucción a partir de imágenes monoculares. Como primer paso se realizaron pruebas del funcionamiento de la obtención de las imágenes provenientes de las cámaras del simulador V-REP (**Figura 31**).

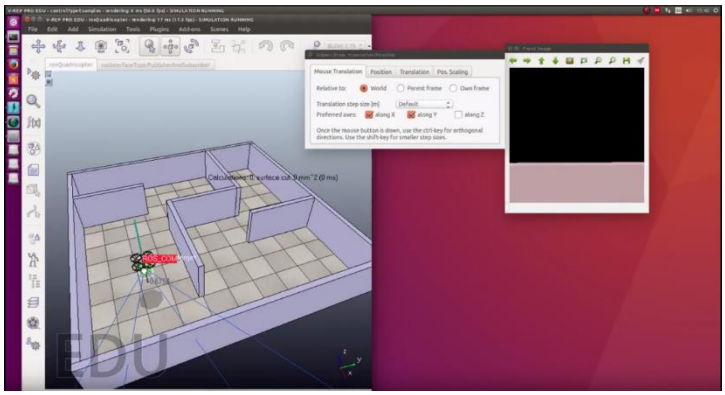


Figura 31. Obtención de imágenes del simulador V-REP

Se realizó un estudio con el modelo LSD-SLAM (Engel et al., 2014), donde se crearon paquetes de los nodos publicados por los autores del artículo, los nodos ROS que son utilizados en el artículo son llamados Core y Viewer. Se desarrolló un entorno en el simulador V-REP (**Figura 32**) para probar el funcionamiento de los nodos. Donde se obtuvieron los parámetros intrínsecos de la cámara del simulador V-REP.

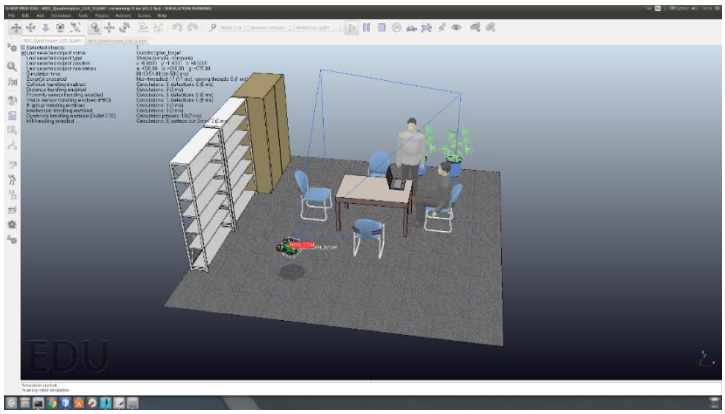


Figura 32. Desarrollo de entorno en el simulador V-REP

El nodo Core es empleado para el procesamiento de imágenes monoculares y posteriormente enviado para el nodo Viewer (**Figura 33-34**). El nodo Viewer proporciona su propio entorno para la ubicación de la nube de puntos en el espacio. Emplea su propio optimizador de error para la correcta ubicación de los puntos.

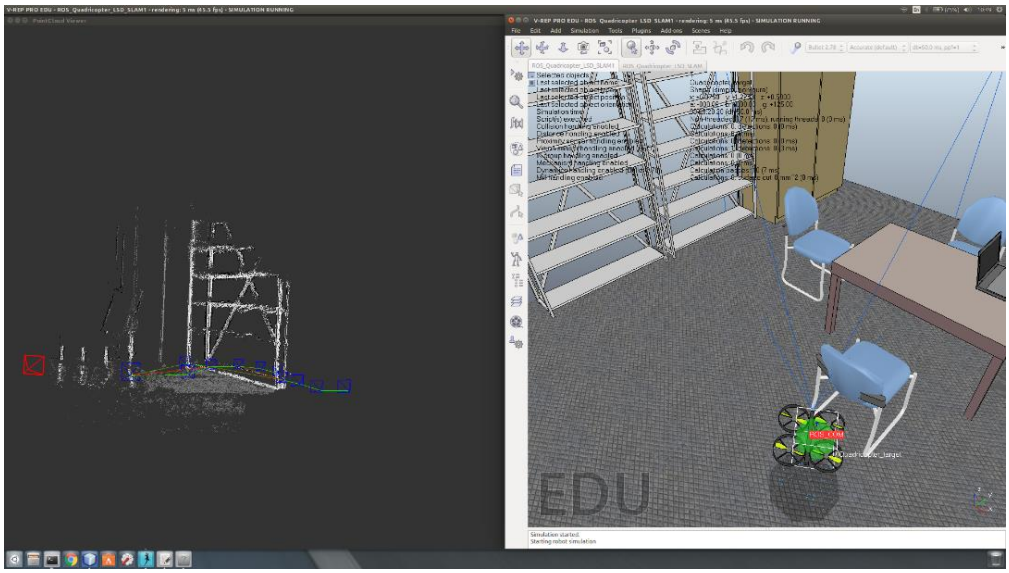


Figura 33. Prueba de funcionamiento de LSD-SLAM

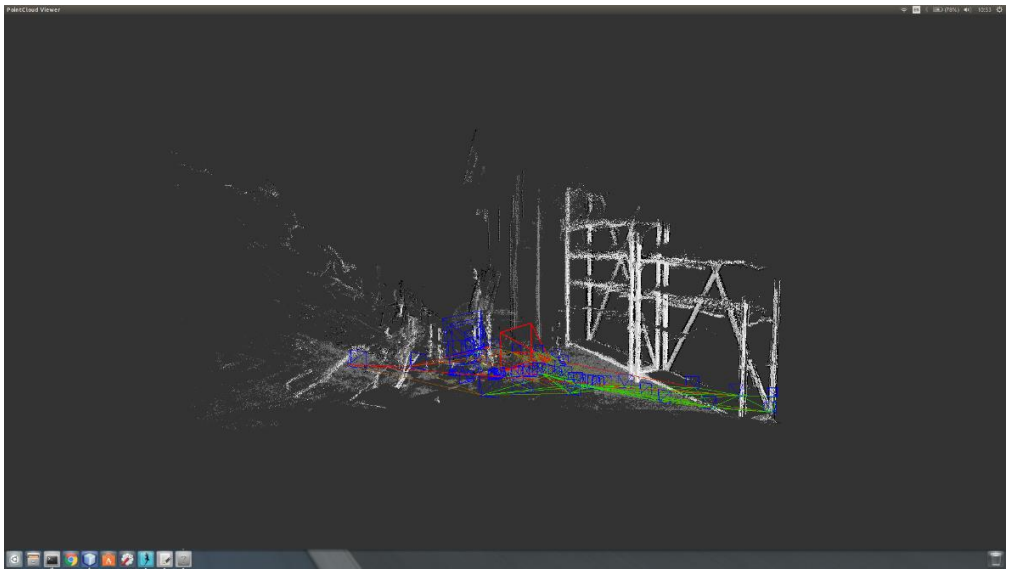


Figura 34. Visualizador Vewer para la ubicación de la nube de puntos generado por el nodo

Como el sistema LSD-SLAM cuenta con su propio visualizador que cuenta con optimización de error, no es posible utilizarlo para exportar la información al visualizador de ROS RVIZ. Para solventar el problema del visualizador se utilizó el modelo de visión estereotípica donde se desarrolló algoritmos para la extracción de características de las imágenes, correspondencia entre puntos y finalmente la utilización de la geometría epipolar para ubicar las esferas en el espacio.

3.6.1 Calibración de las cámaras

Se utilizó el nodo de calibración de cámaras monoculares de ROS donde se empleó una imagen de tablero de ajedrez de dimensiones conocidas de 8x6 cuadros es decir nueve cuadrados de ancho y siete de alto de 108 mm de dimensión, el nodo utiliza los vértices que se encuentran en el interior del tablero de ajedrez (**Anexo 10**). Se ejecutó el nodo para las tres cámaras correspondientes a los mini drones y para la cámara del simulador V-REP (**Figura 35**):

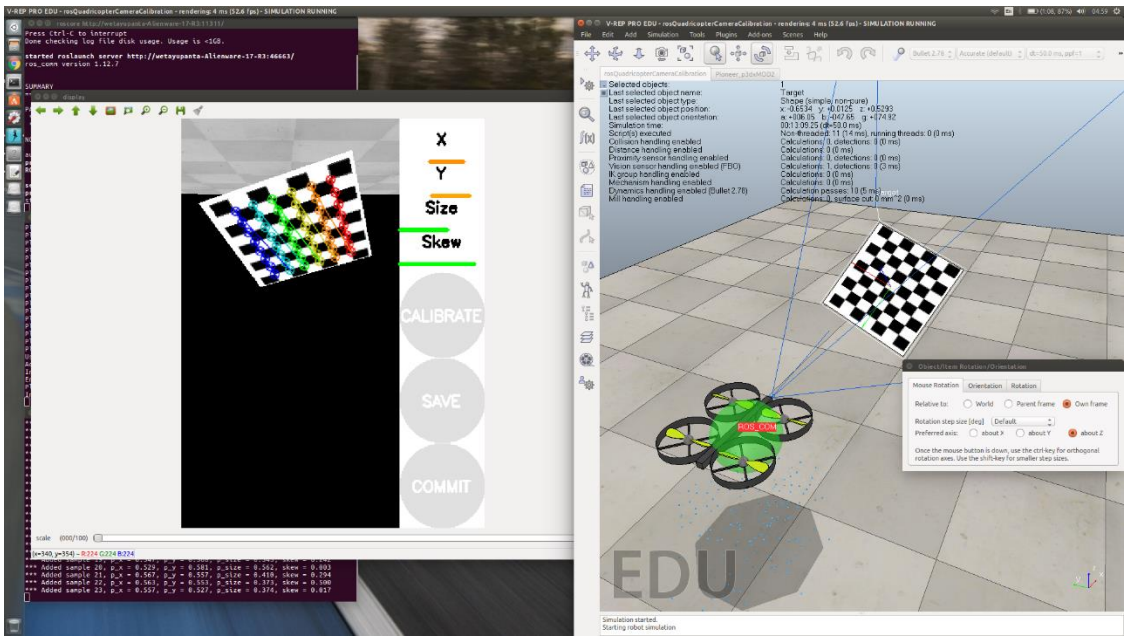


Figura 35. Calibración de cámara monocular del simulador V-REP

- Cámara de mini drone ab1966: rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/camera/image_raw_0 camera:=/camera –no-service-check.
- Cámara de mini drone ca76a1: rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/camera/image_raw_1 camera:=/camera –no-service-check.
- Cámara de mini drone ab1886: rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/camera/image_raw_2 camera:=/camera –no-service-check.

Donde se obtuvieron los parámetros intrínsecos de las cámaras en un archivo de extensión YALM (**Anexo 11**):

3.6.1.1 Parámetro intrínseco de la cámara del mini drone ab1966

Matriz de la cámara:

$$K = \begin{bmatrix} 898.189565 & 0 & 378.022643 \\ 0 & 947.954499 & 308.477699 \\ 0 & 0 & 1 \end{bmatrix}$$

Valores de distorsión:

$$d = [-0.01075 \quad 0.064771 \quad 0.000523 \quad 0.002831 \quad 0.0]$$

Matriz de la cámara rectificada:

$$K = \begin{bmatrix} 899.228638 & 0 & 379.330164 \\ 0 & 950.960205 & 308.610501 \\ 0 & 0 & 1 \end{bmatrix}$$

3.6.1.2 Parámetro intrínseco de la cámara del mini drone ca76a1

Matriz de la cámara:

$$K = \begin{bmatrix} 1150.728113 & 0 & 354.636720 \\ 0 & 1219.258611 & 299.976169 \\ 0 & 0 & 1 \end{bmatrix}$$

Valores de distorsión:

$$d = [0.117662 \quad -0.319429 \quad 0.000118 \quad -0.004803 \quad 0.0]$$

Matriz de la cámara rectificada:

$$K = \begin{bmatrix} 1162.650757 & 0 & 352.761695 \\ 0 & 1235.460327 & 299.982356 \\ 0 & 0 & 1 \end{bmatrix}$$

3.6.1.3 Parámetro intrínseco de la cámara del mini drone ab1866

Matriz de la cámara:

$$K = \begin{bmatrix} 896.424529 & 0 & 368.695443 \\ 0 & 947.863893 & 266.670713 \\ 0 & 0 & 1 \end{bmatrix}$$

Valores de distorsión:

$$d = [0.022055 \quad -0.134515 \quad -0.001724 \quad 0.006630 \quad 0.0]$$

Matriz de la cámara rectificada:

$$K = \begin{bmatrix} 896.194153 & 0 & 371.711197 \\ 0 & 950.958862 & 265.987625 \\ 0 & 0 & 1 \end{bmatrix}$$

3.6.1.4 Parámetro intrínseco de la cámara del simulador V-REP

Matriz de la cámara:

$$K = \begin{bmatrix} 229.45164632 & 0 & 128.0 \\ 0 & 229.45164632 & 128.0 \\ 0 & 0 & 1 \end{bmatrix}$$

Valores de distorsión:

$$d = [0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]$$

3.6.2 Extracción y correspondencia de características de una imagen

Para el proceso de extracción de características de una imagen se realiza a través del nodo ROS Core que previamente fue explicado. Como los mensajes ROS se encuentran sincronizados se realiza una rectificación de las imágenes con los parámetros intrínsecos de cada cámara, se realizaron pruebas del proceso tanto para el simulador V-REP y con los mini drones físicos (**Figura 36-37**). Para el proceso de sincronización de los nodos para el video de los mini drones físicos se empleó una función interna de ROS que permite que la sincronización sea aproximada ya que si se realizaba una sincronización por defecto se bloqueaba el hilo que genera el nodo Core. Por ende, la recepción de imágenes no es muy tardía en el nodo Core, este tiempo se puede mejorar al emplear computadoras en vez que las tarjetas Raspberry Pi 3, que cuentan con recursos limitados.

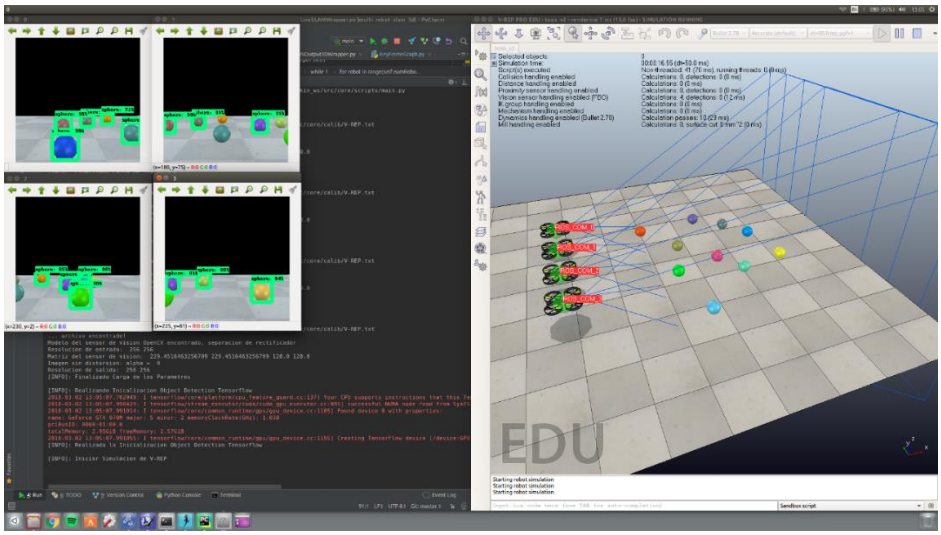
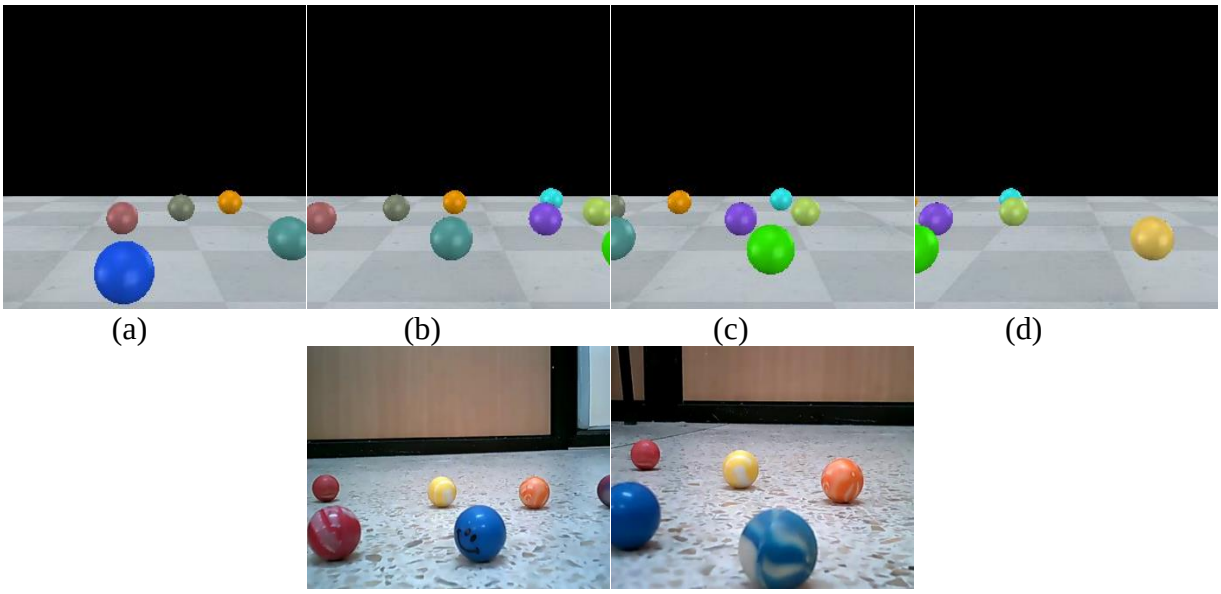


Figura 36. Prueba con el simulador V-REP

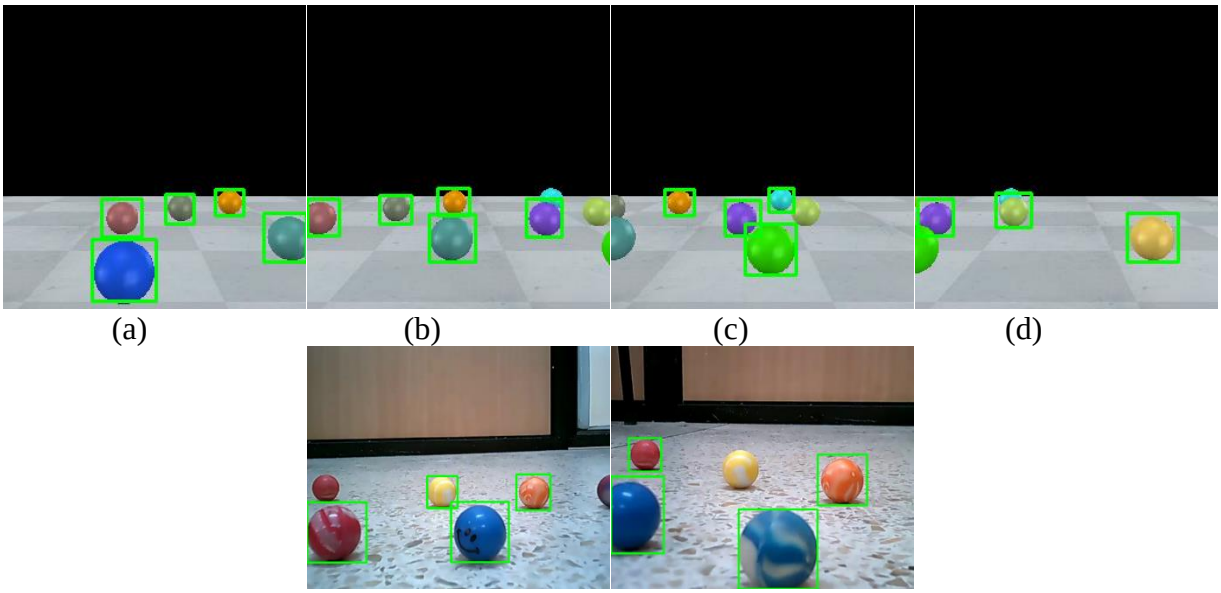


(a) (b) (c) (d)

(e) (f)

Figura 37. Imágenes rectificadas. Primer par de imágenes obtenidas del simulador V-REP

Posteriormente se realiza la ubicación de las esferas en las imágenes a través de la red neuronal entrenada con la API de TensorFlow (**Figura 38**), los cuadros delimitadores de las esferas en las imágenes (xmin, ymin, xmax, ymax) sirven para segmentar la extracción de puntos clave a solo las esferas, de forma que el resto de la imagen no es analizada (**Anexo 12**).



(a) (b) (c) (d)

(e) (f)

Figura 38. Cuadros delimitadores

Para la extracción de puntos clave (**Figura 38**) se emplea la función de OpenCV goodFeaturesToTrack(), que permite que encontrar N esquinas dentro de la imagen por el método Shi-Tomasi, esta función requiere que la imagen se encuentre en escala de grises y con los parámetros que se desea:

- Número máximo de bordes: 100.
- Nivel de calidad: 0.3 que denota la calidad mínima de esquinas que se desea encontrar.
- Distancia mínima: 7 que se refiere a la distancia euclidiana mínima entre bordes detectados.

Con esta información se encuentran los bordes de las esferas, los valores que se encuentren debajo del nivel de calidad se rechazan.

Como el método de cámara estereotípica se basa en el análisis de la imagen correspondiente a la cámara izquierda se realiza primero la búsqueda de cada borde en esa imagen (**Figura 39**).

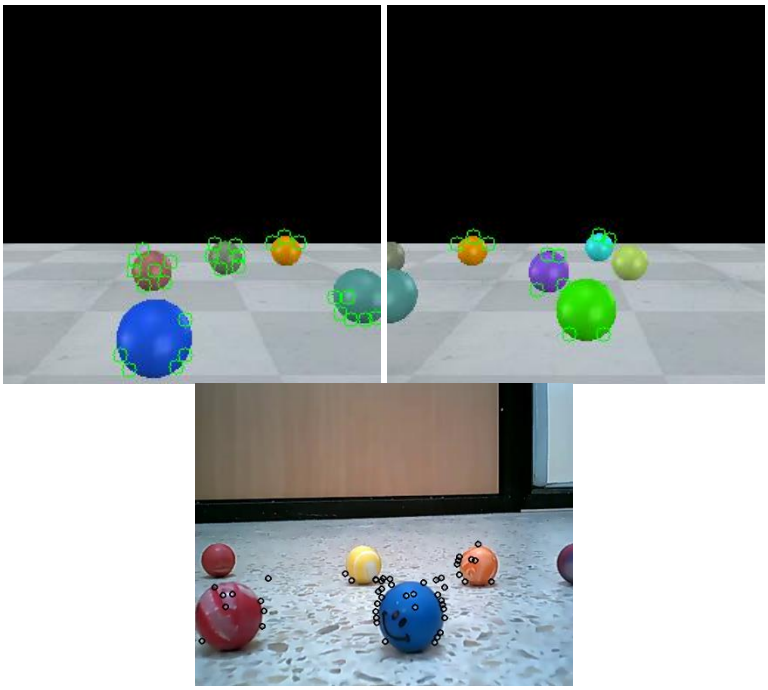


Figura 39. Extracción de puntos clave de las cámaras izquierdas

Una vez encontrados los bordes de cada esfera se busca si existe correspondencia con la imagen de la cámara derecha (**Figura 40**), para ello se utiliza el método de flujo óptico de Lucas-Kanade. De esta forma permite la búsqueda de esferas similares entre el par de imágenes.

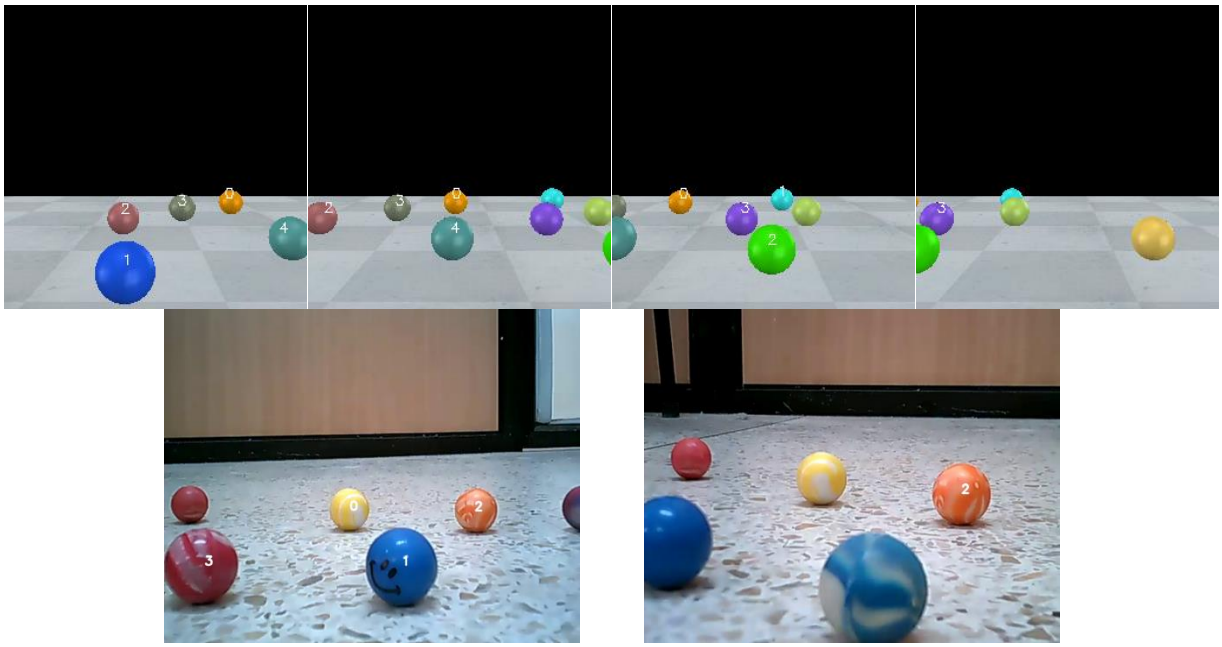


Figura 40. Correspondencia de esferas entre pares de imágenes, se numera cada esfera

3.6.3 Geometría Epipolar

Una vez identificados los puntos correspondientes entre un par de imágenes, se realiza el cálculo de la matriz fundamental a partir de la función de OpenCV cv2.findFundamentalMat().

Donde la matriz fundamental cumple con la ecuación:

$$u_1^T F u_0 = 0$$

La función toma como valores de entrada los puntos correspondientes entre el par de imágenes y el método de correspondencia que es el método RANSAC. Los valores de retorno de la función es la matriz fundamental y una matriz de N dimensiones donde se establece con valores 0 a los puntos que no sean correspondientes y 1 a los valores que son correspondientes. Posteriormente se realiza el cálculo de la matriz esencial con la ecuación con los parámetros intrínsecos de cada cámara:

$$E = K_1^T F K_0$$

Una vez realizado el cálculo de la matriz esencial se procede a descomponer en los componentes de rotación y traslación $[R|t]$ empleando la descomposición de valores singulares SVD.

$$A = U \Sigma V^T$$

La librería numpy de Python permite realizar descomposiciones SVD, donde el resultado de la descomposición genera cuatro soluciones posibles, es por ello que se realiza la verificación de cada solución donde cada punto correspondiente se encuentran frente de ambas cámaras, pero para ello se debe normalizar y homogeneizar las coordenadas de la imagen para obtener las coordenadas homogéneas externas:

$$u = PX$$

$$P = K[R|t]$$

$$u = K[R|t]X$$

Donde $[R|t]$:

$$[R|t]_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Esto se debe a que la proyección se toma desde la cámara izquierda hacia la cámara derecha:

$$X = K^{-1}u$$

$$u' = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$X' = K^{-1}u' = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

El resultado de la verificación de las matrices nos genera una segunda matriz $[R|t]$ que corresponde a la cámara de la derecha:

$$[R|t]_1 = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{bmatrix}$$

Al obtener las matrices con los parámetros extrínsecos de las cámaras se procede a calcular las matrices de proyección de ambas cámaras:

$$P_0 = K_0[R|t]_0$$

$$P_1 = K_1[R|t]_1$$

Para el cálculo del punto en el espacio, para ello se debe calcular la disparidad de los puntos en cada imagen para aplicar las siguientes ecuaciones:

$$P = P_1 P_0^{-1}$$

$$u' = PX'$$

$$P^{-1} = P^T[PP^T]^{-1}$$

$$X' = P^{-1}u'$$

Para ubicar las esferas y las cámaras de los mini drones se debe utilizar las propiedades intrínsecas de la cámara (**Figura 41-42**).

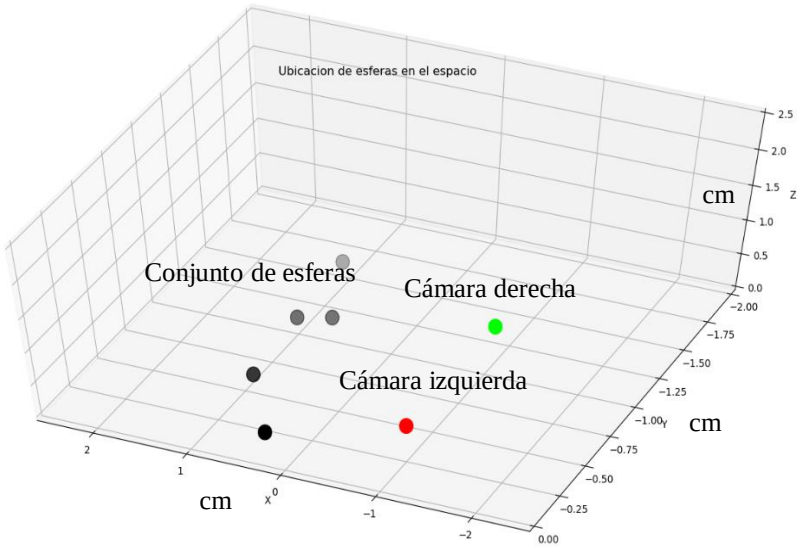


Figura 41. Ubicación de las esferas en el espacio simulador V-REP

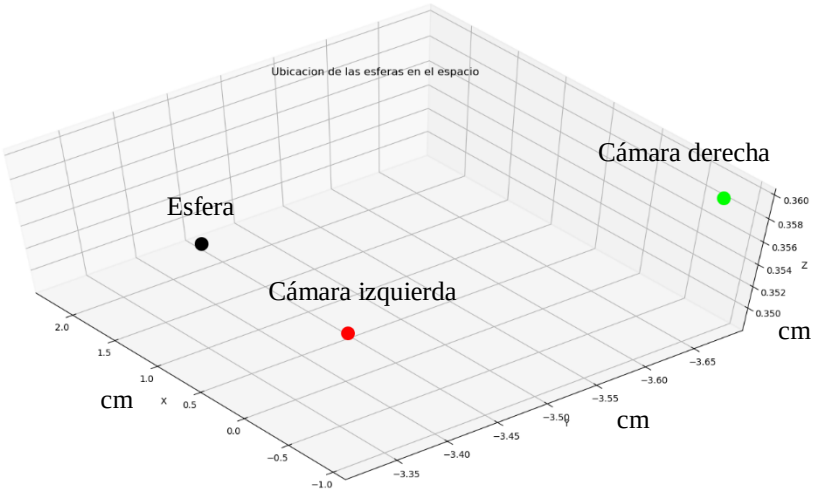


Figura 42. Ubicación de las esferas en el espacio

3.7 Control de vuelo de los mini drones

Para el control de vuelo de los mini drones se empleó seguimiento por visualización mediante una cámara general que se encuentra ubicada en la parte superior de los mini drones, los mini drones cuentan con marcadores ArUco (**Figura 43**).

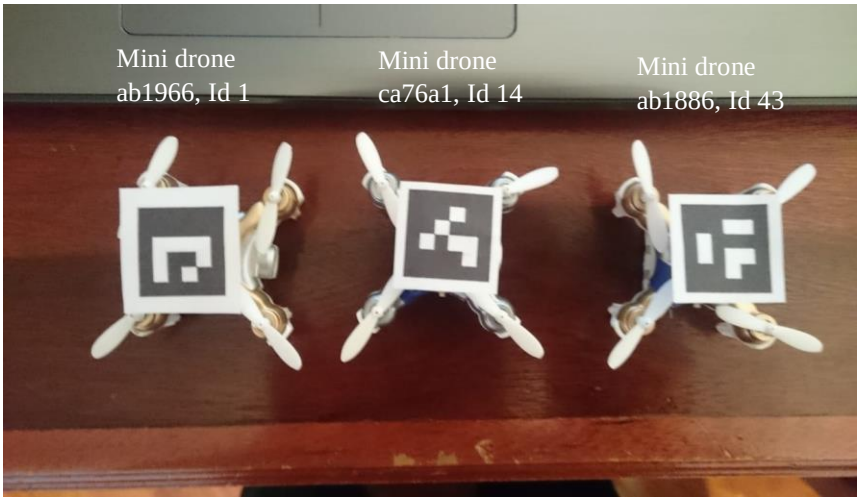


Figura 43. Marcadores ArUco

Los marcadores ArUco fueron generados a partir de la librería de OpenCV Aruco, los marcadores fueron generados a con una matriz binaria de 4x4 con tamaño de 80 bits. Los marcadores que se escogieron fueron con el identificador 1 (mini drone ab1966), 14 (mini drone ca76a1) y 43 (mini drone ab1886) respectivamente de la (**Figura 43**).

3.7.1 Calibración de la cámara general

Se utilizó el nodo ROS para calibración de cámaras monoculares (**Anexo 13**) que se mencionó en la sección 3.6.1.

3.7.1.1 Parámetro intrínseco de la cámara general

Matriz de la cámara:

$$K = \begin{bmatrix} 802.925027 & 0 & 311.180790 \\ 0 & 811.706111 & 332.692223 \\ 0 & 0 & 1 \end{bmatrix}$$

Valores de distorsión:

$$d = [0.128950 \quad 0.011042 \quad 0.009671 \quad -0.007468 \quad 0.0]$$

Matriz de la cámara rectificada:

$$K = \begin{bmatrix} 829.063110 & 0 & 308.061199 \\ 0 & 838.634033 & 333.766999 \\ 0 & 0 & 1 \end{bmatrix}$$

3.7.2 Estimación de postura del marcador

Como se mencionó anteriormente los marcadores ArUco sirve para la estimación de postura del marcador dentro del sistema de coordenadas externas (**Figura 44**), para ello se utilizan los parámetros intrínsecos de la cámara.

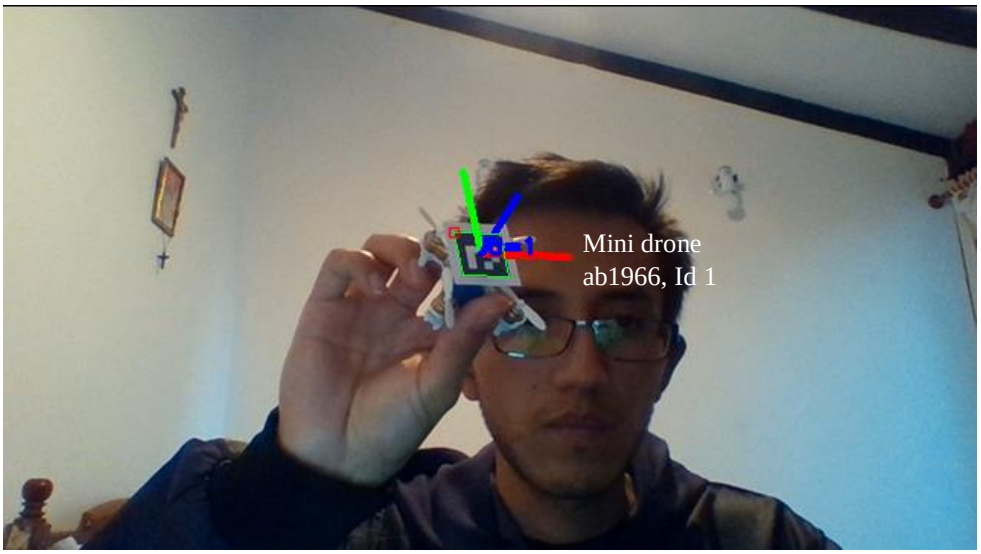


Figura 44. Estimación de postura del marcador

Como se puede apreciar (**Figura 44**) la librería ArUco permite el grafico de las coordenadas en el marcador al utilizar la cámara general (**Figura 45**), se identifica cada marcador dependiendo de su id.

Esta identificación es de suma importancia ya que dependiendo del marcador se obtiene la posición del mini drone en el espacio y es utilizado para el controlador de vuelo.

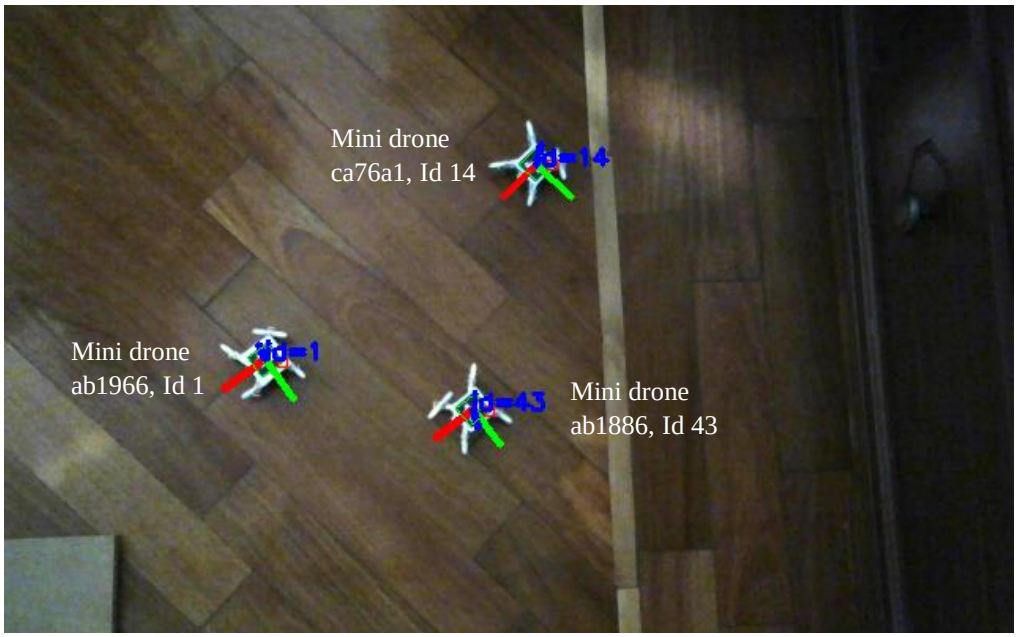


Figura 45. Cámara general para la detección de marcadores ArUco

3.7.3 Distribución de nodos ROS

Desde el nodo drone_control que se encuentra en la computadora principal se generan los mensajes (drone/pose_0, drone/pose_1, drone/pose_2) de tipo Pose que son los valores de rotación y traslación del mini drone en el sistema de coordenadas externo (**Anexo 14**). Estos mensajes son enviados a cada Raspberry Pi que cuenta con nodos ROS con el nombre del SSID de la red WIFI que genera el mini drone, estos nodos sirven para el control de vuelo de los mini drones (**Tabla 8**).

Se empleó un diseño de controlador a través de seguimiento visual con la estimación de pose de los marcadores ArUco, que provee los valores de la posición y orientación del mini drone con respecto a la cámara general.

Tabla 8

Nodos ROS

Nombre del Nodo	Nombre del tema	Tipo
drone_control	drone/pose_0, drone/pose_1, drone/pose_2	Pose
ab1966	drone/pose_0	Pose
ca76a1	drone/pose_1	Pose
ab1886	drone/pose_2	Pose

3.7.3.1 Nodo drone_control

El nodo drone_control se encuentra en la computadora principal y es aquel en donde se realiza la detección y seguimiento del marcador ArUco correspondiente a cada mini drone, ya que se obtienen los valores de la posición y rotación del mini drone, estos valores son enviados al correspondiente mini drone de acuerdo al identificar del marcador (**Figura 46-47**).

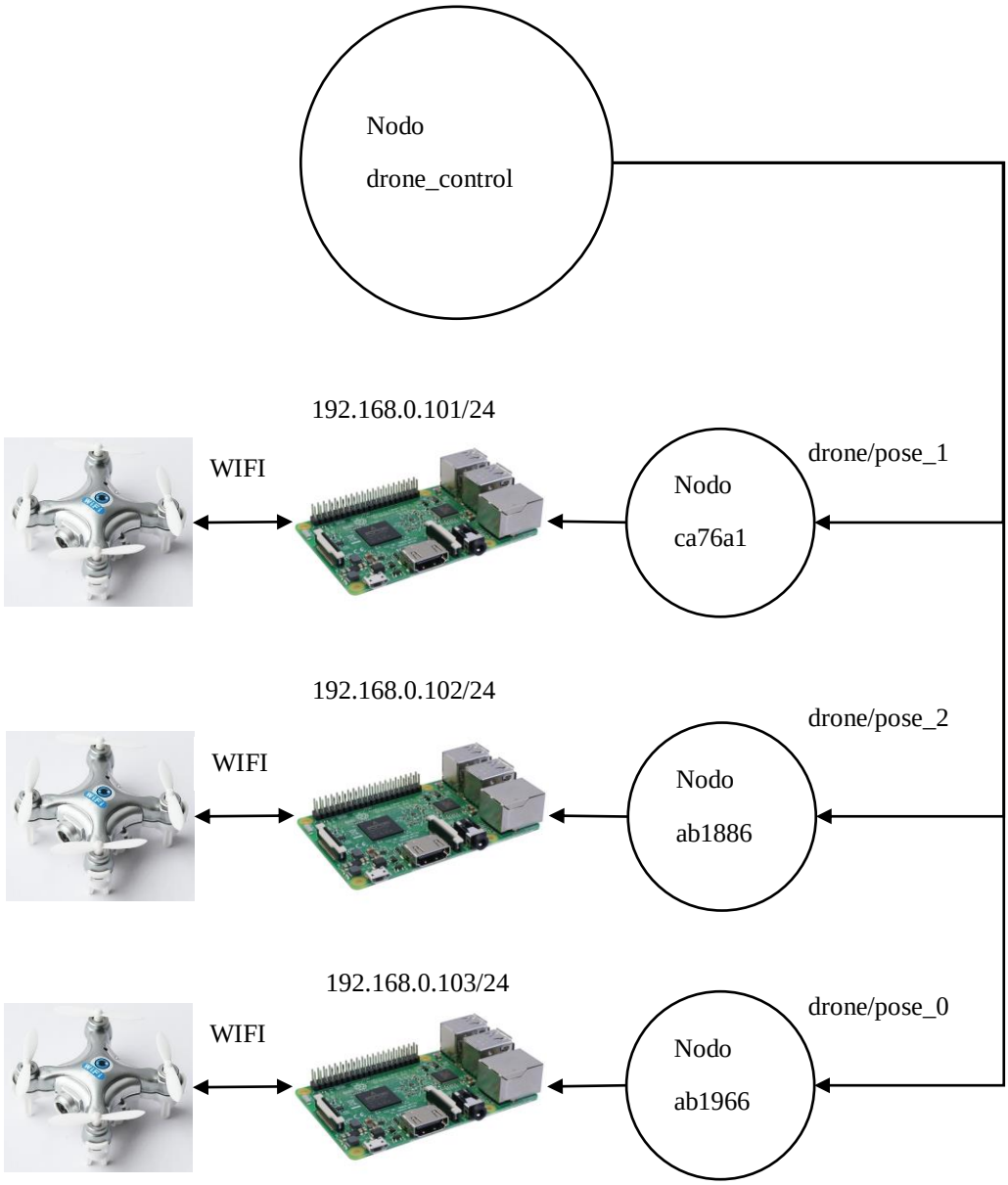


Figura 46. Distribución de nodos ROS

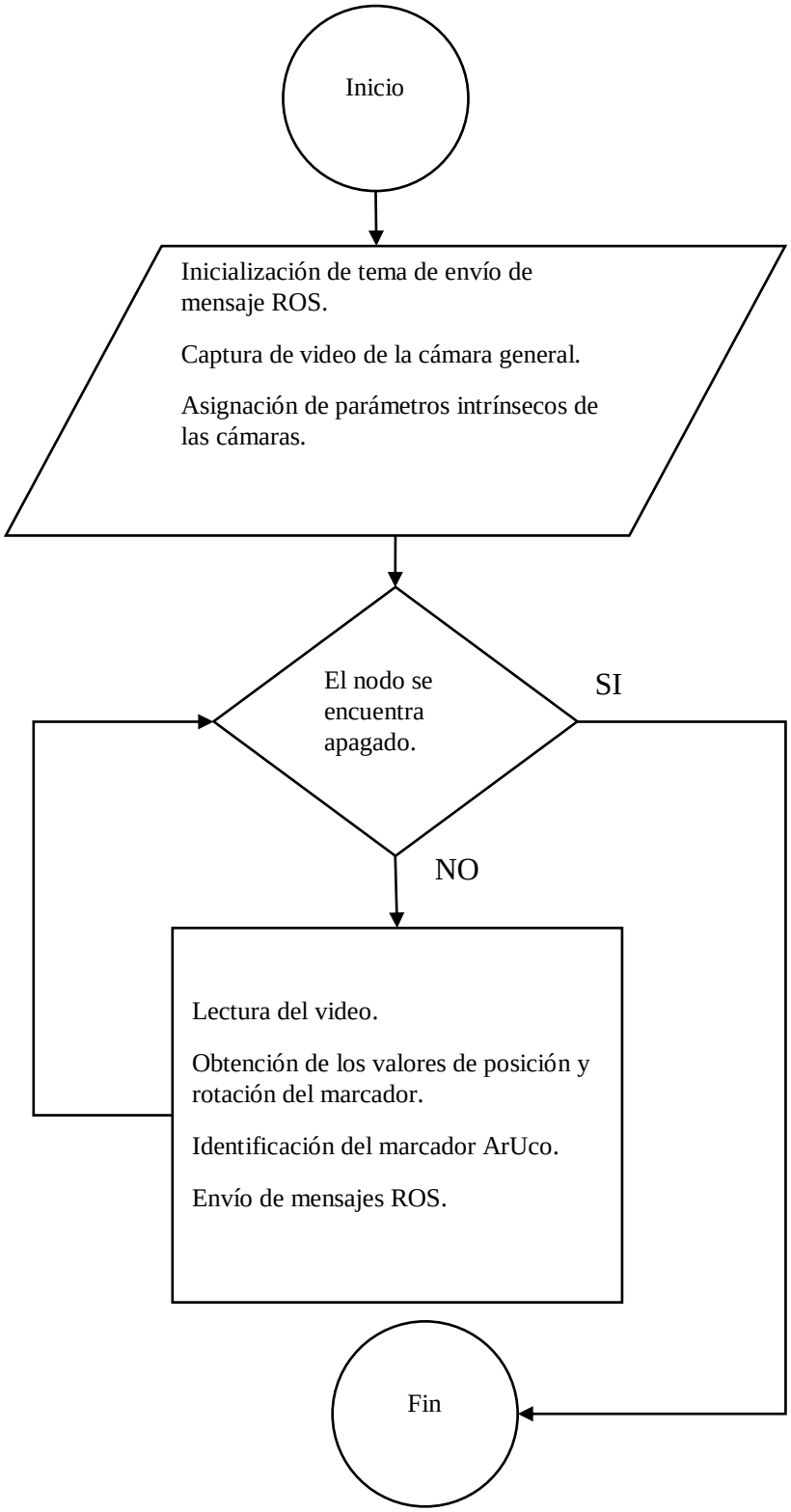


Figura 47. Diagrama de flujo del nodo ROS drone_control

3.7.3.2 Nodo ab1966, ca76a1 y ab1886

Dentro de cada nodo recibe el mensaje ROS (drone/pose_0, drone/pose_1, drone/pose_2) respectivamente, estos valores son almacenados dentro de la Raspberry Pi para que el sistema siempre cuente con la posición actual del mini drone (**Figura 48**), una vez almacenado la posición y rotación del marcador se ejecuta el controlador de vuelo del mini drone.

Dentro de cada nodo se establece una conexión vía socket a cada mini drone con la dirección IP 176.16.10.1 con el puerto 8895, para el envío de los valores de throttle, pitch, roll, yaw donde los rangos de cada variable se describen en la (**Tabla 9**). Además, se puede apreciar que los valores de thottle para el control de altura van desde 0 a 255 siendo su valor inicial 0, así como los valores de pitch, roll, yaw que tienen el rango similar a throttle, pero el valor inicial es 127, que dependiendo de los valores de roll y yaw se puede obtener un giro si es mayor a 127 a la derecha y si es menor gira hacia la izquierda. Para el desplazamiento hacia adelante se utilizó el valor de pitch, donde si los valores eran mayores que 127 se desplazaba hacia adelante y si fueron menores se desplaza hacia atrás. Se generaron pruebas con los rangos establecidos en los cuales se llegó a la conclusión que el controlador interno no es óptimo para el sistema requerido, además de contar con giroscopio interno.

Tabla 9

Valores para el control de cada mini drone

Nombre	Rango	Valor de inicio	Rotación
Throttle	0 a 255	0	
Pitch	0 a 255	127	>127 giro hacia adelante <127 giro hacia atrás
Roll	0 a 255	127	>127 giro derecha <127 giro izquierda
Yaw	0 a 255	127	>127 giro derecha <127 giro izquierda

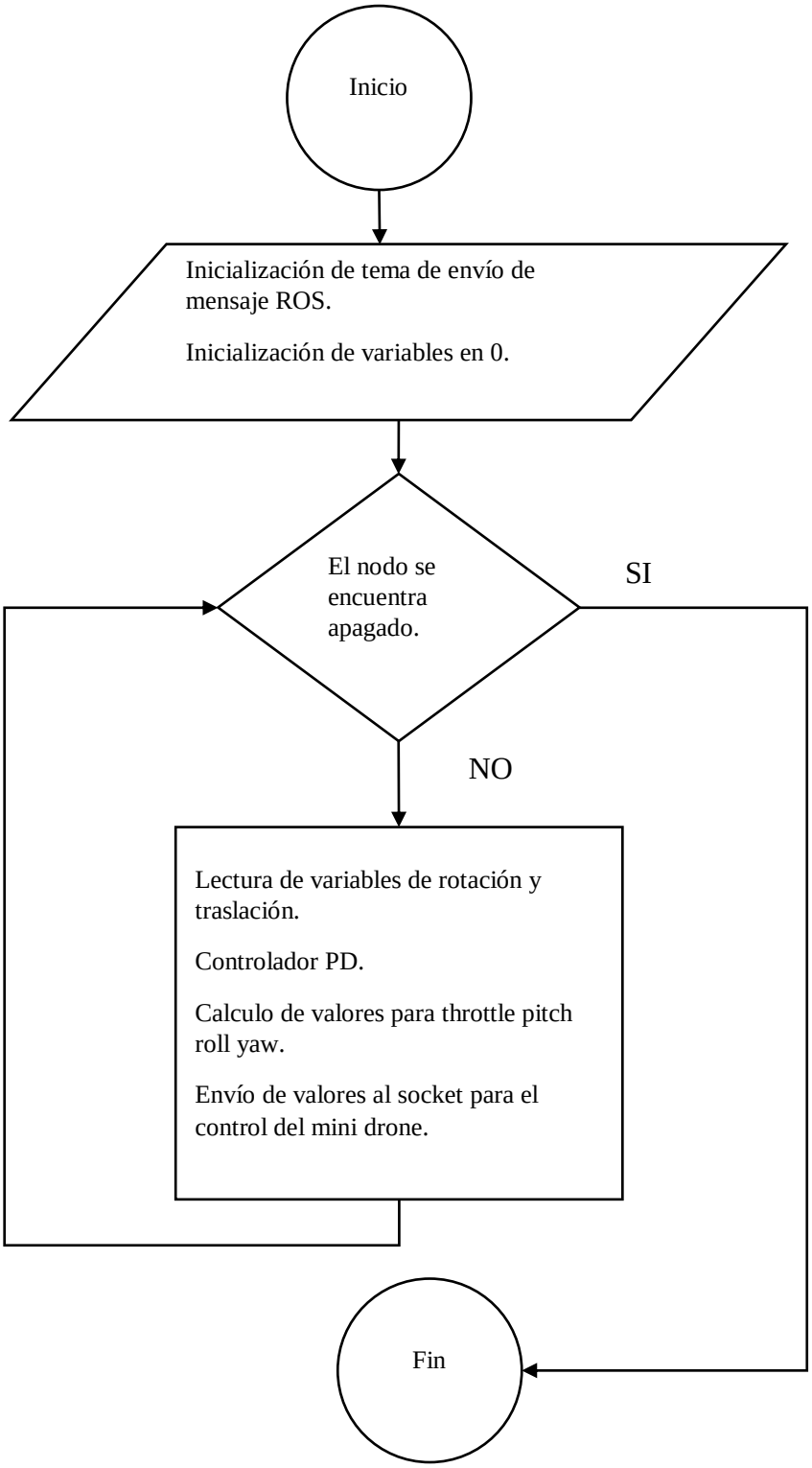


Figura 48. Diagrama de flujo de los nodos ab1966, ca76a1 y ab1886

3.7.4 Controlador PD

El diseño del controlador se basó en (Ericsson, 2015), donde se tomó como referencia la función de transferencia del controlador PD para el sistema de forma (**Figura 49**):

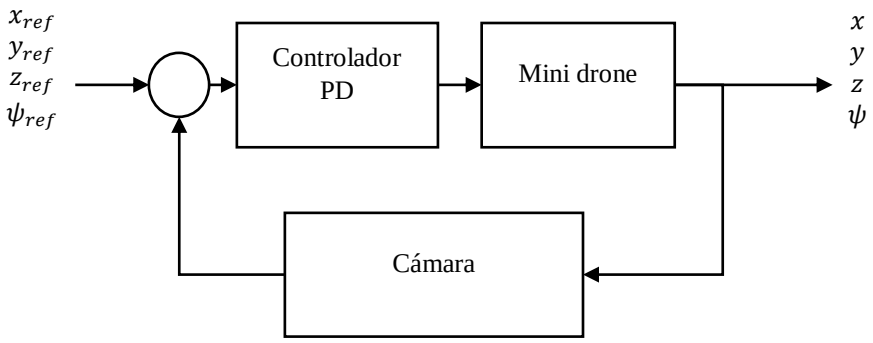


Figura 49. Diagrama de bloques del sistema

$$u(t) = K_p e(t) + K_d \dot{e}_\gamma(t)$$

Transformación al dominio de la frecuencia y reemplazo de $K_d s$ por el filtro:

$$\frac{NK_d s}{N + K_d s}$$

Donde:

$$s = \frac{2(z-1)}{h(z+1)}$$

Dando como resultado la siguiente expresión:

$$U(z) = K_p E(z) + \frac{NK_d \frac{2}{h}(z-1)}{N(z+1) + K_d \frac{2}{h}(z-1)} E_\gamma(z)$$

Resolviendo la expresión para la señal de control es:

$$u(n+1) = K_p e(n+1) + \frac{B}{A}\Big(K_p e(n) - u(n)\Big) + \frac{C}{A}\Big(e_\gamma(n+1) - e_\gamma(n)\Big)$$

Siendo los valores de A, B, C:

$$A = \left(N + K_d \frac{h}{2}\right)$$

$$B = \left(N - K_d \frac{h}{2}\right)$$

$$C = NK_d \frac{h}{2}$$

Este cálculo genera la señal de control tanto para posicionamiento en los ejes x, y, z para cada iteración.

Para el cálculo del controlador del ángulo yaw para lo cual se plantea la señal de control para el ángulo yaw.

$$u_{yaw}(t) = K_p e(n)$$

Para el cálculo de todos los valores de K_p y K_d de las acciones de control se ha realizado un ajuste por prueba y error donde los valores se asignan como x, y, z, yaw (**Tabla 10**).

Tabla 10
Valores para el controlador PD

Nombre del Nodo	**Valores de K_p**	**Valores de K_d**
ab1966	x: 0.168 y: 0.23 z: 0.4 ψ: 0.6	x: 4.6 y: 3.763 z: 6.0
ca76a1	x: 0.168 y: 0.23 z: 0.6 ψ: 0.6	x: 4.6 y: 3.763 z: 6.0
ab1886	x: 0.0668 y: 0.023 z: 0.03 ψ: 0.012	x: 0.66 y: 0.363 z: 6.0

3.7.4.1 Retroalimentación visual

El controlador está diseñado de manera que solo requiere el ángulo yaw ψ que se obtiene del marcador ArUco, ya que con los valores de posición t_x y t_y se puede calcular los valores de pitch y roll para el impulso del mini drone en los ejes x e y. Con el ángulo yaw se describe una nueva matriz de rotación para que el cálculo no afecte a los valores de pitch y yaw.

$$R_{3x3} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La cámara general siempre se encuentra estable se toma como el Sistema de coordenadas externo, para la asignación de del punto de referencia x, y, z.

$$x_{ref} = \begin{bmatrix} x_{ref} \\ y_{ref} \\ z_{ref} \end{bmatrix}$$

Para el cálculo del error:

$$\hat{e} = x_{ref} - \begin{bmatrix} x \\ y \\ -z \end{bmatrix} = \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ z - z_{ref} \end{bmatrix}$$

Para estar en el mismo sistema de coordenadas de la cámara se debe rotar al error y generar el cambio de signo en el eje y.

$$e = R_{3x3} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \hat{e}$$

$$e = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ z - z_{ref} \end{bmatrix}$$

Para que el cálculo no influencie al punto de referencia se ha definido un error:

$$e_\gamma = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \gamma x_{ref} - x \\ \gamma y_{ref} - y \\ z - \gamma z_{ref} \end{bmatrix}$$

De igual forma se define el error del ángulo yaw:

$$\widehat{R_{3x3}} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\psi_{ref} & -\sin\psi_{ref} & 0 \\ \sin\psi_{ref} & \cos\psi_{ref} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Donde:

$$e = atan2(\hat{r}_{2,1}, \hat{r}_{1,1})$$

3.8 Visualización en el entorno RVIZ de ROS

Adicionalmente se generó un lanzador para que ejecute el nodo drone_control con el visualizador RVIZ donde muestra la posición de los mini drones en el espacio, graficados como esferas (**Figura 40-52**). Para ello se empleó un retardo de 1 segundo entre el lanzamiento de los dos nodos (**Anexo 15**).

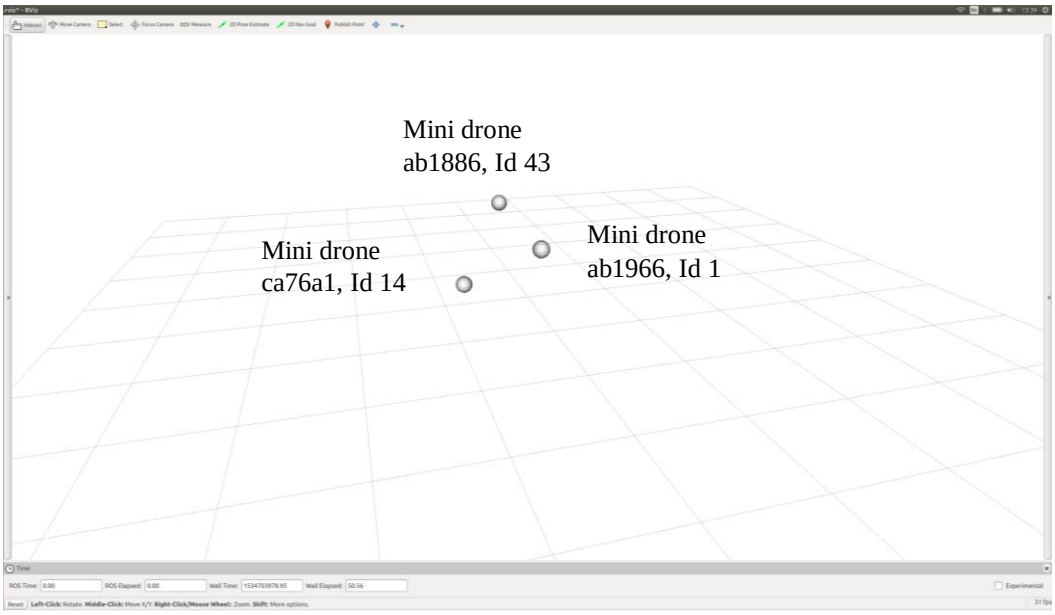


Figura 50. Visualizador RVIZ para la ubicación de los mini drones en el espacio

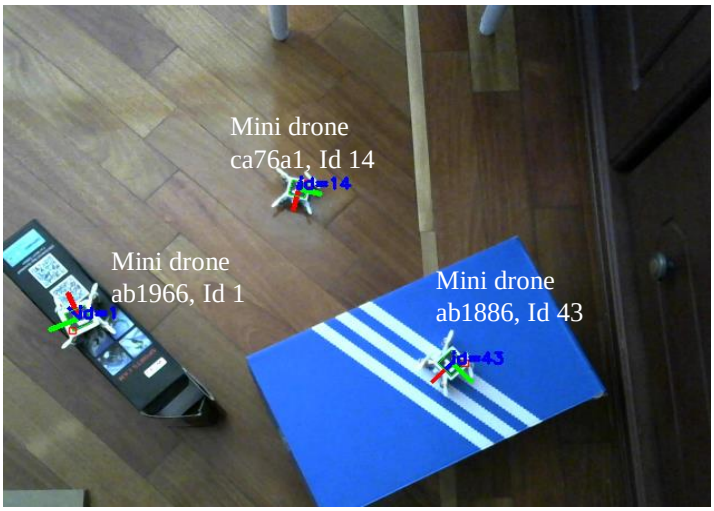


Figura 51. Cámara general para ubicación de marcadores ArUco

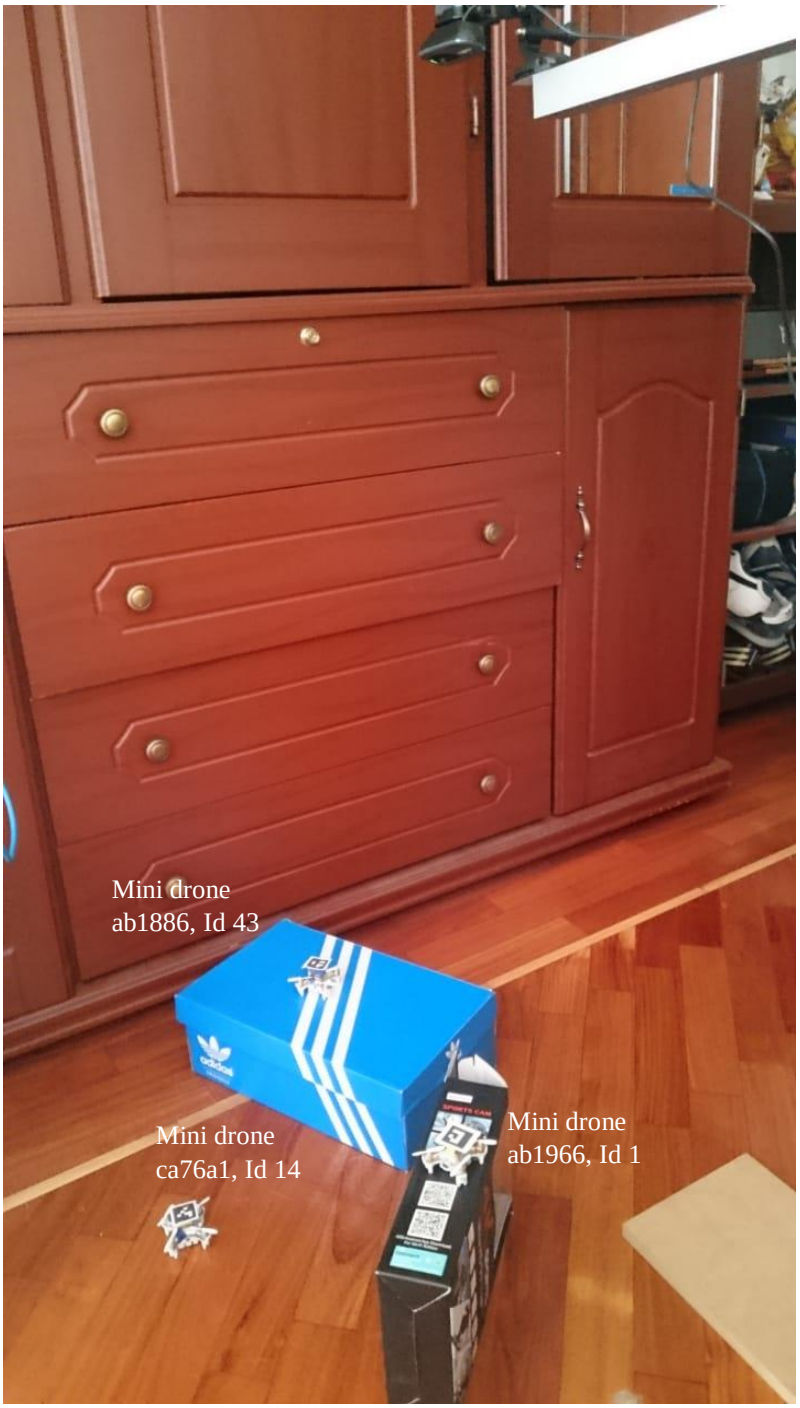


Figura 52. Vista vertical de la ubicación de los mini drones

CAPÍTULO IV

ANÁLISIS

4.1 Detección de Objetos

Dependiendo de la versión de la API para la detección de objetos, se obtuvo que para el proceso de entrenamiento que se llevó a cabo durante ocho días donde se generaron 110.4 k pasos con una perdida aproximada promedio de 1.797 (**Figura 53**), como se muestra en la (**Figura 54**). La pérdida total tiende a tener picos, esto se debe al desarrollo de la librería TensorFlow ya que se empleó la API v1.8. El proceso de entrenamiento puede ser retomado con un nuevo grupo de imágenes ya que TensorFlow permite almacenar puntos de control del entrenamiento.

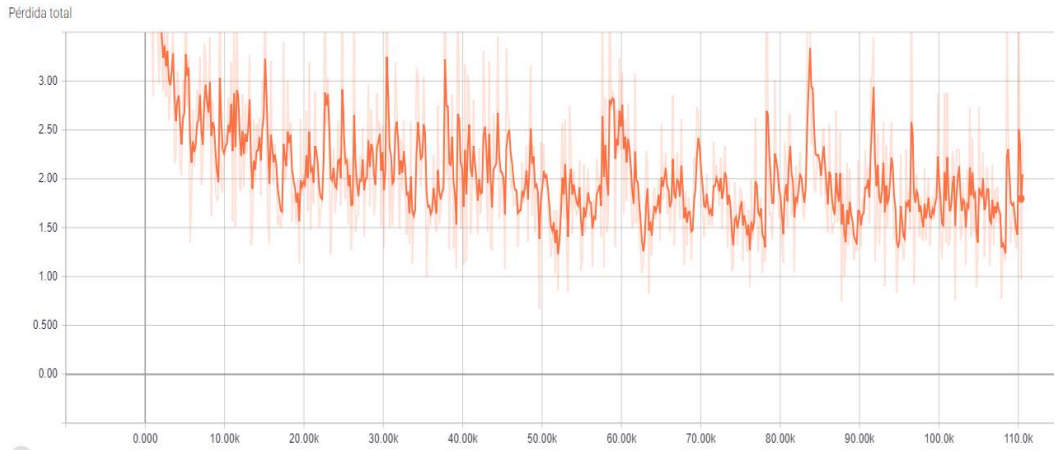


Figura 53. Gráfico de inferencia

Nombre	Alisado	Valor	Paso	Hora	Relativo
.	1.797	0.9729	110.4k	Martes 3 de julio, 16:23:32	8d 2h 7m 15s

Figura 54. Valores presentados por TensorBoard

Al emplear la API v1.9 se obtuvo mejores resultados con 50 mil iteraciones (**Figura 55**) con perdida aproximada promedio de 2.468 en dos horas (**Figura 56**).

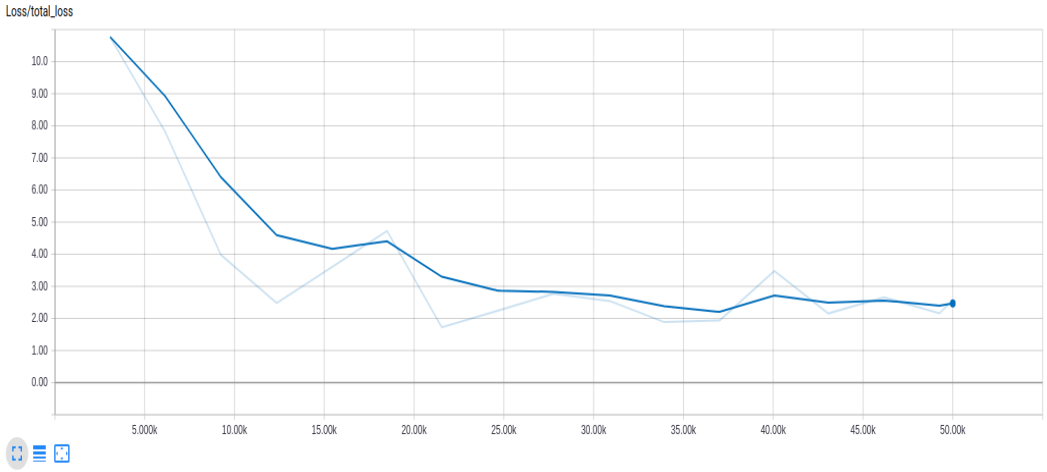
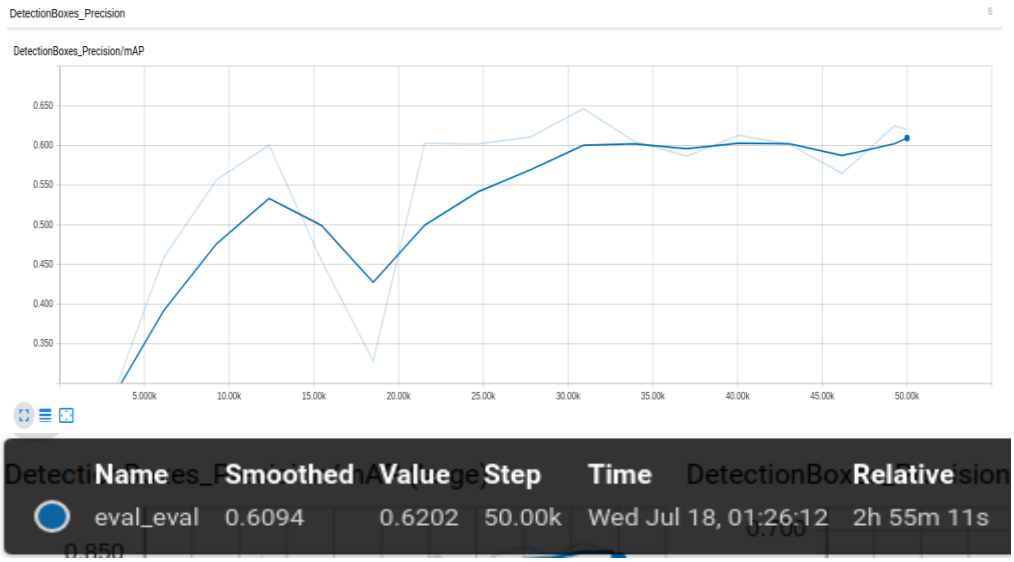


Figura 55. Gráfico de inferencia

Name	Smoothed	Value	Step	Time	Relative
eval_eval	2.468	2.571	50.00k	Wed Jul 18, 01:26:12	2h 55m 11s

Figura 56. Valores presentados por TensorBoard

Además, al utilizar la última versión de la API nos genera un rendimiento del detector de la medida de precisión promedio de 0.6094 mAP (**Figura 57**). Que es una medida para validar la capacidad de que la red neuronal cuando éstas detectar los objetos de interés en cuadros delimitados.



Name	Smoothed	Value	Step	Time	Relative
eval_eval	0.6094	0.6202	50.00k	Wed Jul 18, 01:26:12	2h 55m 11s

Figura 57. Medida de precisión promedio

4.1.1 Tiempo de predicción de las esferas

Para el cálculo de tiempo de predicción se utilizó a las imágenes (**Figura 58)**. No se tomó en cuenta el tiempo que existe al iniciar el proceso de adquisición de imágenes y sincronización de los nodos ROS.

Es por ello que solo se midió el tiempo de predicción de esferas en las imágenes, al crear una sesión de TensorFlow que se encuentra previamente cargada la red neuronal y clase que corresponde a la esfera. En cada iteración se ejecuta la sesión de TensorFlow con operaciones de OpenCV para la predicción de las esferas con un tiempo de ejecución de:

$$0.06645\ segundos$$

Figura 58. Ubicación de esferas

4.1.2 Precisión de predicción

Para la imagen obtenida del mini drone ab1966 (**Figura 58 (a)**) se predijo 3 esferas de las 7 esferas visibles de la imagen y para la imagen obtenida del mini drone ca76a1 (**Figura 58 (b)**) se predijeron 4 esferas de las 7 esferas visibles de la imagen, esto se debe a que las esferas no son apreciadas completamente es decir solo se muestra un pedazo de la esfera, esto puede ser mejorado con agregar mayores imágenes en el conjunto de datos de entrenamiento de la red neuronal.

Los valores de predicción de las esferas para la imagen (**Figura 58 (a)**) y (**Figura 58 (b)**) se describen (**Tabla 11**).

Tabla 11
Precisión de predicción

Nombre del mini drone	**Numero de esfera encontradas**
Figura 57 (a) mini drone ab1966	3 esferas con (81%, 99%, 99%)
Figura 57 (b) mini drone ca76a1	4 esferas con (94%, 99%, 98%, 92%)

Como se observa la detección de las esferas en las imágenes tiene un porcentaje mayor al 80%, además no se detectan a las esferas que no se encuentren completas.

4.2 Emparejamiento de imágenes

Para el emparejamiento de imágenes que se muestran (**Figura 58)**, se obtuvo las imágenes (**Figura 9**). Se muestra que existe correspondencia entre las esferas ubicadas (**Figura 58 (a))** y (**Figura 58 (b)**). Se aprecia que la esfera marcada con 0 y 2 (**Figura 59 (a)**) se marca como correspondiente en la misma esfera (**Figura 59 (b)**). La esfera marcada como 1 es la correcta porque existe correspondencia entre las dos imágenes, corresponde a la esfera azul.

Figura 59. Correspondencia de esferas entre pares de imágenes, se numera cada esfera

Cuando sucede este tipo de eventos que marcan a la misma esfera dos veces, se elimina la esfera para el grafico. Posteriormente se realiza el grafico de la ubicación de los mini drones y las esferas

en el espacio (**Figura 60**) marcando a la esfera de color negro, al mini drone ab1966 de color rojo y al mini drone ca76a1 de color verde.

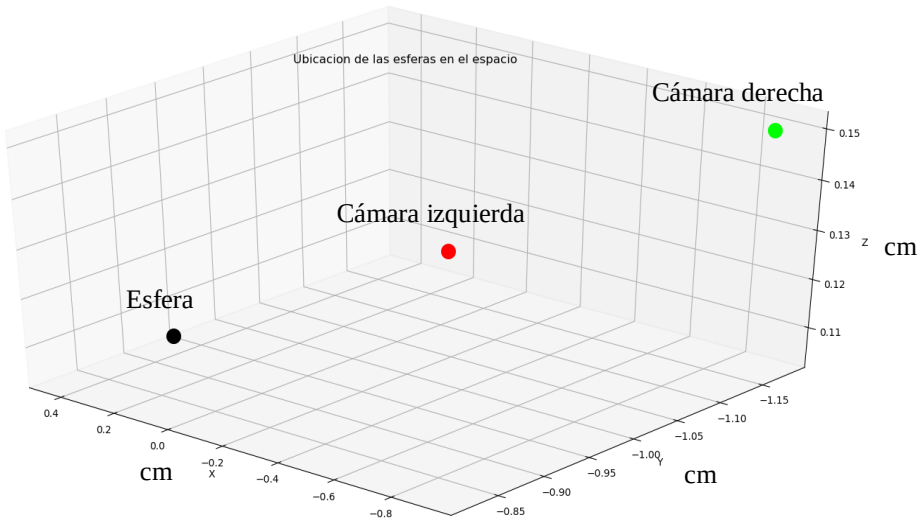


Figura 60. Ubicación de las esferas y los mini drones

4.3 Error de calibración de cámaras

Como cualquier cálculo se genera errores en el proceso, para ello se realiza el cálculo de error de re-proyección que es una estimación de los parámetros intrínsecos de las cámaras. Para esto se ha calculado la matriz de la cámara y la distorsión tanto tangencial y rotacional causadas por el lente.

4.3.1 Cámara mini drone ab1966

Dentro del conjunto de imágenes que almacenó el nodo ROS de calibración de cámaras monoculares se verificó que existe un error, que es la media aritmética de los errores que existe para todas las imágenes obtenidas en la calibración:

0.071468

Lo que demuestra que los parámetros intrínsecos de obtenidos del nodo de calibración son correctos ya que el valor se acerca a cero.

Con un tiempo de ejecución para calcular el error:

$$95.1430\ segundos$$

4.3.2 Cámara mini drone ca76a1

El error de calibración obtenido es de:

$$0.093661$$

Lo que demuestra que los parámetros intrínsecos de obtenidos del nodo de calibración son correctos con un valor cercano a cero.

El tiempo de ejecución para calcular el error fue de:

$$42.515999\ segundos$$

4.3.3 Cámara mini drone ab1886

El error de calibración obtenido es de:

$$0.07489015$$

Lo que demuestra que los parámetros intrínsecos del nodo de calibración son correctos con un valor cercano a cero.

Se obtuvo un tiempo de ejecución para calcular el error:

$$118.359\ segundos$$

4.3.4 Cámara general

El error de calibración se situó en: 0.023683

Lo que demuestra que los parámetros intrínsecos del nodo de calibración son correctos ya que el valor se acerca a cero.

Los datos se obtuvieron con un tiempo de ejecución para calcular el error de:

$$101.2940 \text{ segundos}$$

4.4 Tiempo de ejecución de los nodos ROS

El tiempo de procesamiento de todo el sistema desarrollado en los nodos ROS (**Tabla 12**) es igual a 0.4381528 segundos al realizar una iteración. Lo que demuestra que no existe una sobrecarga masiva de datos para el procesamiento en la computadora ya que el sistema se encuentra distribuido en las tarjetas Raspberry Pi 3.

Se observa que el nodo con mayor tiempo de procesamiento es el nodo core, es aquel que realiza la sincronización de los nodos provenientes de los mini drones, la predicción de las esferas en las imágenes que toma un tiempo de 0.06645 segundos para su posterior extracción de puntos clave y correspondencia entre puntos clave.

Los demás nodos correspondientes a las tarjetas Raspberry Pi 3 no generan mucho procesamiento ya que se cuenta con limitaciones por su hardware.

Tabla 12
Tiempo de ejecución de los nodos

Nombre del Nodo	**Tiempo de ejecución para realizar el proceso [s]**
core	0.1795
drone_control	0.045511
ab1966	0.04837
ca76a1	0.0495018
ab1886	0.04882
Tiempo total de ejecución	**0.4381528**

4.5 Error de posición y rotación

El sistema para el cálculo de error se calcula de acuerdo a la posición de los mini drones que se muestran (**Figura 45**). Para ello los valores calculados se debe pasar a sistema de coordenadas externas cumpliendo con la siguiente ecuación:

$$X' = P^{-1}u'$$

Como la cámara tiene un alcance de 77.5x56 cm y está ubicada a 100 cm del suelo. Para obtener los valores reales de las imágenes se creó un plano con los ejes x e y (**Figura 61**).

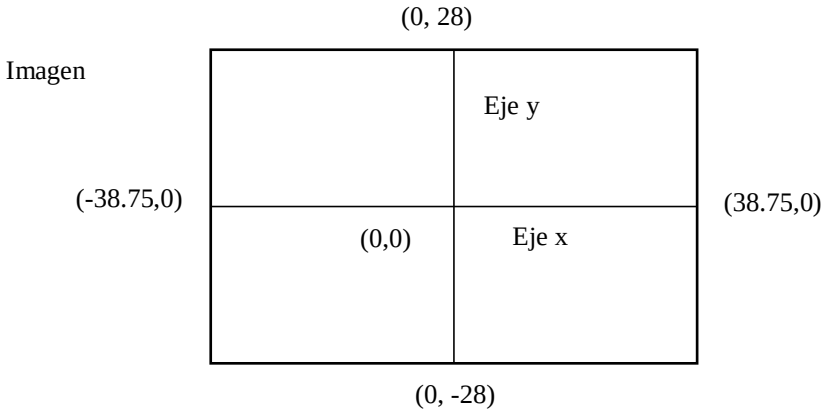


Figura 61. Sistema de coordenadas externo

Se realizaron pruebas para la posición de los mini drones en el espacio donde se tomó en consideración el sistema planteado (**Figura 61**) donde la imagen corresponde a la imagen tomada de la cámara general ubicada en la parte superior para realizar el seguimiento visual de los mini drones.

Para el cálculo del error (**Tabla 13**) se generó un plano cartesiano donde se obtuvo la posición real del mini drone. Se ubicó al mini drone de acuerdo al alcance de las dimensiones de la cámara que fueron de 77.5 cm en el eje x y 56 del eje y. Para el eje z se realizaron mediciones manuales con metro desde la ubicación de la cámara superior hacia el suelo.

Tabla 13

Error de posición de los mini drones en el espacio

Nombre del mini drone	**Valor real [cm]**	**Valor calculado sistema de coordenadas de la cámara**	**Error relativo**
ab1966	X = -27.678571 Y = -5.090909 Z = 80	X = -22.983306 Y = -5.120252 Z = 88.497229	X = 0.169635 Y = 0.005763 Z = 0.106215
ca76a1	X = -5.535714 Y = 8.654545 Z = 96.3	X = -5.535137 Y = 5.354209 Z = 88.405226	X = 0.000104 Y = 0.381341 Z = 0.081981
ab1886	X = 11.041428 Y = -11.2	X = 10.317258 Y = -10.248623	X = 0.065586 Y = 0.084944

CONTINÚA

	Z = 85	Z = 88.468016	Z = 0.040800
Error relativo promedio de coordenadas			**X = 0.07844** **Y = 0.157349** **Z = 0.076332**

Como el ángulo ψ se necesita para el controlador de vuelo de los mini drones. Se analizó el error de rotación con respecto al ángulo yaw ψ (**Tabla 14**) de los códigos ArUco de acuerdo (**Figura 62**).

Como se muestra el error relativo promedio de la rotación sobre el eje z de los tres mini drones es de 0.02222.

Como se muestra (**Figura 60**), existe solo una esfera de ubicación en el espacio. Para ello se realizó el cálculo del error (**Tabla 15**). La posición real de la esfera fue calculada con la ayuda de la cámara general que puede observar todo lo que se encuentra debajo suyo (**Figura 60**). El error de posición de la esfera en el espacio es cercano a cero.

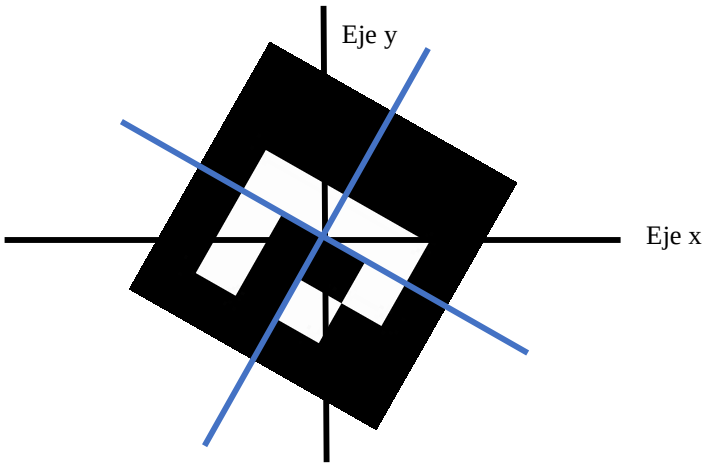


Figura 62. Angulo real ψ del marcador ArUco

Tabla 14

Error de rotación de los mini drones en el espacio

Nombre del mini drone	**Valor real de la rotación de los ejes [rad]**	**Valor calculado de la rotación de los ejes [rad]**	**Error relativo**
ab1966	Z = 2.04561	Z = 2.04052	Z = 0.00248

CONTINÚA

ca76a1	Z = -1.84636	Z = -1.78633	Z = 0.03251
ab1886	Z = -2.51364	Z = -2.43402	Z = 0.03167
Error relativo promedio			Z= 0.02222

Tabla 15.

Error de posición de la esfera en el espacio de la Figura 59

Nombre	**Valor real [cm]**	**Valor calculado [cm]**	**Error relativo**
Esfera 1	X = 0.446213	X = 0.422954	X = 0.052125
	Y = -0.963104	Y = -0.92677316	Y = 0.037722
	Z = 0.110326	Z = 0.104832	Z = 0.049772

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

6.1 Detección de objetos

Se implementó una red neuronal convolucional para la detección de esferas en imágenes obtenidas por los mini drones, dentro de un entorno controlado de tres dimensiones. Esto se logró gracias a la API de TensorFlow que se basa en redes previamente entrenadas con un gran conjunto de datos.

Para el proceso de entrenamiento de la red neuronal convolucional se necesitan imágenes con diferentes niveles de iluminación, distancias de los objetos y que las imágenes sean de la mayor calidad posible.

TensorFlow puede ser implementado en sistemas CPU y GPU, en sistemas de CPU el tiempo que toma el proceso de predicción de los objetos es aproximadamente de 0.53 segundos a diferencia que los sistemas GPU que es de aproximadamente 0.02 segundos. Además, demuestra que el tiempo de predicción es el 37% de todo el tiempo que toma una iteración el nodo core ya que la computadora principal cuenta con procesador Intel core i7 con 16 GB de RAM.

La API de TensorFlow para la detección de objetos en la versión 1.8 tuvo una pérdida total de 0.097 pero el tiempo de entrenamiento fue de 8 días, a diferencia que la versión 1.9 que en alrededor de tres horas se obtuvo una pérdida total de 2.468 y la curva de la perdida tuvo un mejor descenso sin generar picos en la gráfica de pérdida total.

La medida de precisión promedio del entrenamiento es de 0.6094 mAP, esto puede ser mejorado con mayor entrenamiento con diferentes imágenes de la red neuronal convolucional, además de aumentar el número de iteraciones y el número de lotes por iteración.

6.2 Reconstrucción 3D

Para la ubicación de las esferas en el espacio se analizó tanto los parámetros intrínsecos de cada cámara, para ello se utilizó el nodo ROS para la calibración de cámaras monoculares donde se obtuvieron tanto las matrices de la cámara, distorsión de la cámara y las matrices de la cámara rectificada, el error de proyección fue calculado con los parámetros encontrados lo que significa que tuvo un error promedio de las cámaras de 0.08779.

La matriz rectificada de la cámara permite corregir la distorsión causada por el lente, esta distorsión se compensa con la matriz de distorsión radial y tangencial del lente.

Los métodos de extracción de características de una imagen como SIFT y SURF pueden extraer los puntos clave y los descriptores de características. Estos últimos denotan la rotación entre un par de imágenes, pero consumen demasiados recursos computacionales a diferencia que el método ORB, el cual emplea el método FAST para la extracción de puntos clave para posteriormente aplicar el método BRIEF para los descriptores de características.

Al realizar diferentes pruebas de extracción de puntos clave se muestra que el empleo del método ORB es el adecuado para sistemas que requieren respuesta a tiempo real, para rastrear puntos clave entre imágenes el método de Flujo Óptico Lucas-Kanade, que emplea el patrón de movimiento aparente de los objetos entre dos imágenes consecutivas. Para ello se calcula el flujo óptico que es un método muy efectivo cuando el movimiento es pequeño, pero si el movimiento es grande los puntos son eliminados.

Se empleó el modelo de visión estereotípica para la triangulación de puntos entre dos imágenes obteniéndose un mayor acercamiento en las posiciones reales tanto de los mini drones con un error relativo promedio de posición de los mini drones de X = 0.07844, Y = 0.157349, Z = 0.076332.

Para el cálculo del error de rotación del ángulo yaw ψ muestra un error relativo promedio de 0.0222 y el cálculo del error de posición de las esferas es de X = 0.052125, Y = 0.037722, Z = 0.049772

Lo que demuestra que existe un error menor a 10 en los ejes x y z a diferencia del eje y que muestra un error alrededor de 0.16. En el caso de la ubicación de las esferas.

Cuando la imagen de la cámara derecha se encuentran con una rotación muy grande con respecto de la izquierda no se puede obtener un resultado de la matriz fundamental para ello se debe cambiar el método de extracción de punto clave.

Para el cálculo de la matriz de rotación y traslación de la cámara derecha se utilizó el método de descomposición de valores singulares (SVD) donde las cuatro soluciones posibles son analizadas con cada punto clave se encuentre al frente de cada cámara. Estos valores suelen ser variables ya que se pierden respuestas con el análisis de las cuatro soluciones, nos permite concluir que la aplicación de la descomposición de valores singulares para la obtención de las matrices de rotación de la cámara derecha se consideran apropiadas para la aplicación de reconstrucción 3D.

Se debe tomar en cuenta el cambio de coordenadas con las matrices de rotación provenientes de cada sistema para obtener valores reales y graficar las esferas en el espacio.

6.3 Controlador de vuelo de los mini drones

Para el control de vuelo de los mini drones se implementó un controlador con seguimiento visual, implementándose marcadores ArUco con una matriz binaria de 4x4 de tamaño de 80 bits para ser ubicados en la parte superior de los mini drones.

Cuando el mini drone se mueve con mayor velocidad se pierde la ubicación del mini drone en el espacio, esto se produce debido al tamaño y movimiento de los marcadores; lo que afecta al algoritmo de control. Se puede mejorar el rendimiento de la detección del marcador utilizando una cámara con mayor capacidad de definición ya que en el trabajo se empleó una cámara de 720 P.

Al contar los mini drones con controladores de vuelo internos no permiten el correcto funcionamiento del controlador de seguimiento por visión, además que los valores obtenidos para el controlador fueron asignados por prueba y error, se puede utilizar otro tipo de controlar que no necesite una gran cantidad de cálculo de parámetros para el control de vuelo, se recomienda para trabajos futuros emplear controladores difusos.

El visualizador RVIZ permite ubicar los drones en el espacio y así poder observar el comportamiento de los drones en él, donde se pudo observar que la estimación de postura del mini drone se desplaza en el eje z al igual que los ángulos de giro.

6.4 Nodos ROS

La aplicación de nodos ROS emplea protocolos TCP/IP y funcionan como hilos en varios lenguajes de programación además dentro del nodo la recepción de los mensajes cuenta con un hilo independiente al del nodo.

En la generación de código en ROS el lenguaje C++ requiere una mayor cantidad de asignación de variables en el archivo package.xml, a diferencia de Python que cuenta con la librería rospy que simplifica la asignación de variables.

Una desventaja que cuenta los nodos ROS es que en ciertos casos se requiere un retardo de inicio de nodo entre ellos, esto se debe a la sincronización de los nodos ya que dentro de cada nodo se utiliza una tasa de envío que es medida en hertzios. Para la aplicación actual este valor es de 20 Hz para la codificación de las imágenes a mensajes ROS y de 1000 Hz para el control de vuelo de los mini drones.

REFERENCIAS BIBLIOGRÁFICAS

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Brain, G. (2016). *TensorFlow: A system for large-scale machine learning*. Recuperado de https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

Andrade, F., & Martin, L. (2007). *SLAM Estado del arte*. Recuperado de https://www.fing.edu.uy/inco/grupos/mina/pGrado/pgSLAM/documentos/eda.pdf

Archimowicz, Y., Martínez, S., Monzón, P., & Canetti, R. (2011). *EdRo Un enjambre de robots*. Recuperado de Universidad de la República https://iie.fing.edu.uy/publicaciones/2011/AM11/AM11.pdf

Azevedo, T. C. S., Tavares, J. M. R. S., & Vaz, M. A. P. (2009). *3D Object reconstruction from uncalibrated images using an off-the-shelf camera*. In advances in computational vision and medical image processing (pp. 117–136). Dordrecht: Springer Netherlands. https://doi.org/10.1007/978-1-4020-9086-8_7

Bakker, I. den. (2017). *Introduction - python deep learning cookbook*. Recuperado el 2 de Julio de 2018, de https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781787125193/3/ch03lvl1sec26/introduction

Barca, J. C., & Sekercioglu, Y. A. (2013). *Swarm robotics reviewed*. Robotica, 31(03), 345–359. https://doi.org/10.1017/S026357471200032X

Bay, H., Tuytelaars, T., & Gool, L. Van. (2008). *SURF: Speeded Up Robust Features*. Recuperado de http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

Bayındır, L. (2016). *A review of swarm robotics tasks*. Neurocomputing, 172, 292–321. https://doi.org/10.1016/J.NEUCOM.2015.05.116

Beni, G., & Wang, J. (1993). *Swarm intelligence in cellular robotic systems*. In robots and biological systems: Towards a new bionics (pp. 703–712). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-58069-7_38

Bergstra, J., Breuleux, O., & Bastien, F. (2010). *Theano: compilador matemático de CPU y GPU en python. Proc. 9th python In* . Recuperado de https://www.researchgate.net/profile/Razvan_Pascanu/publication/228832149_Theano_A_CPU_and_GPU_math_compiler_in_Python/links/004635314aa30be30d000000/Theano-A-CPU-and-GPU-math-compiler-in-Python.pdf

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence : from natural to artificial isystems*. Oxford University Press.

Brendstrup, P. (2013). *Reconstrucción 3D desde secuencias de video*. Recuperado de http://www2.famaf.unc.edu.ar/institucional/biblioteca/trabajos/638/16846.pdf

Bristol, U. (2006). *PhD robotics and autonomous systems -FARSCOPE (EPSRC Centre for doctoral training) Faculty admissions cycle*. Recuperado de www.brl.ac.uk

Brutschy, A., Garattoni, L., Brambilla, M., Francesca, G., Pini, G., Dorigo, M., & Birattari, M. (2015). *The TAM: abstracting complex tasks in swarm robotics research*. Swarm Intelligence, 9(1), 1–22. https://doi.org/10.1007/s11721-014-0102-6

Bulla Cruz, N., Nedjah, N., de Macedo Mourelle, L., Bonilla, A., N., B. C., L., e M. M., & A., B. (2014). *Agrupamiento espacial eficiente; 2 clases para róbotica de enjambre*. In 2014 IEEE Biennial Congress of Argentina (ARGENCON) (pp. 180–185). IEEE. https://doi.org/10.1109/ARGENCON.2014.6868493

Calabuig, J., Garcia, L., & Sánchez, E. (2015). *Álgebra lineal y descomposición en valores singulares*. Modelling in Science Education and Learning, 8(2), 133. https://doi.org/10.4995/msel.2015.4010

Calderón, D. (2012). *Generación de mapas de profundidad a partir de imágenes estéreo utilizando registro no rígido*. Recuperado de http://repositorio.uchile.cl/handle/2250/111904

Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). *BRIEF: Binary Robust Independent Elementary Features*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15561-1_56

Cárdenas, E., Morales, L., & Caycedo, A. (2015). *La estereoscopía, métodos y aplicaciones en diferentes áreas del conocimiento*. Recuperado de http://www.scielo.org.co/pdf/recig/v13n16/v13n16a10.pdf

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., … Carreras, M. (2015). *ROSPlan: Planning in the Robot operating system. Proceedings of international conference on automated planning and scheduling (ICAPS)*. Recuperado de https://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/viewFile/10619/10379

Cheerson Hobby Technology Company. (2016). *CX-10W WIFI*. Recuperado el 4 de Julio de 2018, de http://www.cheersonhobby.com/en-US/Home/ProductDetail/84

Ciano, O. (2017). *wifi_china_drone_controller*. Recuperado de https://github.com/Otacon/wifi_china_drone_controller

COCO. (2017). *COCO - Common objects in context*. Recuperado el 3 de Julio de 2018, de http://cocodataset.org/#detection-eval

Couceiro, M. S., Vargas, P. A., Rocha, R. P., & Ferreira, N. M. F. (2014). *Benchmark of swarm robotics distributed techniques in a search task*. Robotics and Autonomous Systems, 62(2), 200–213. https://doi.org/10.1016/J.ROBOT.2013.10.004

Deutsch, I., Liu, M., & Siegwart, R. (2016). A *framework for multi-robot pose graph SLAM. In 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR) (pp. 567–572)*. IEEE. https://doi.org/10.1109/RCAR.2016.7784092

Elbit Systems. (2014). *Hermes TM 900 - Elbit Systems*. Recuperado el 25 de Junio de 2018, de http://elbitsystems.com/product/hermes-900-5/

Engel, J., Schöps, T., & Cremers, D. (2014). *LSD-SLAM: Large-Scale Direct Monocular SLAM (pp. 834–849)*. Springer, Cham. https://doi.org/10.1007/978-3-319-10605-2_54

Ericsson, M. (2015). *Department of Automatic Control Visual Tracking and Control of a Quadrocopter*. Recuperado de

http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8593547&fileOId=8593552

Fernández, A., Gabriel, C., Rivera, S., Moreno García, J., Rivera, G. S., & Moreno García Son Colaboradores, J. (2010). *Simulación de la visión estereoscópica. Parte i: aspectos físicos y geométricos*. Recuperado de http://www.dsi.uclm.es/personal/AntonioFdez/download/papers/journal/ENSAYOS-aspectos.pdf

Gallardo, D. (2011). *Modelo y calibración de cámara*. Recuperado de http://www.jtech.ua.es/vision/2011/

Garrido, S., Muñoz, R., Madrid, F., & Marín, M. (2014). *Automatic generation and detection of highly reliable fiducial markers under occlusion*. Pattern Recognition, 47(6), 2280–2292. https://doi.org/10.1016/j.patcog.2014.01.005

Gil, A., Reinoso, O., Payá, L., & Ballesta, M. (2007). *Influencia de los parámetros de un filtro de partículas en la solución al problema de SLAM*. Recuperado de http://www.ewh.ieee.org/reg/9/etrans/ieee/issues/vol06/vol6issue1March2008/6TLA1_03GilAparicio.pdf

Grandón-Pastén, N., Aracena-Pizarro, D., & Tozzi, C. L. (2007). *Reconstrucción de objeto 3D a partir de imágenes calibradas*. Ingeniare. Revista Chilena de Ingeniería, 15(2), 158–168. https://doi.org/10.4067/S0718-33052007000200006

Gro, R., Bonani, M., Mondada, F., & Dorigo, M. (2006). *Autonomous self-assembly in swarm-bots*. IEEE Transactions on Robotics, 22(6), 1115–1130. https://doi.org/10.1109/TRO.2006.882919

Harris, C., & Stephens, M. (1988). *A combined corner and edge detector*. https://doi.org/10.5244/C.2.23

Hartley, R., & Zisserman, A. (2004). *Multiple view geometry in computer vision*. Recuperado de http://cvrs.whu.edu.cn/downloads/ebooks/Multiple View Geometry in Computer Vision (Second Edition).pdf

Henry, P., Henry, P., Krainin, M., Herbst, E., Ren, X., & Fox, D. (2010). *RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments*. In the 12th international symposium on experimental robotics (ISER). Recuperado de http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.226.91

Howse, J., Joshi, P., & Beyeler, M. (2016). *Opencv: computer vision projects with python*. Recuperado el 28 de Junio de 2018, de https://www.packtpub.com/mapt/book/application_development/9781787125490/9/ch05lvl1sec047/what-are-keypoints%253f

Huang, S., Wang, Z., Dissanayake, G., & Frese, U. (2009). *Iterated SLSJF: A sparse local submap joining algorithm with improved consistency*. Recuperado de https://pdfs.semanticscholar.org/6e41/e7840bf378709098b624ed299d637c6fc63e.pdf

Infodefensa.com. (2011). *El Ejército argentino presenta formalmente los UAV Lipan M3 y simulador de tiro SITARAN II - Noticias Infodefensa América*. Recuperado el 25 de Junio de 2018, de http://www.infodefensa.com/latam/2011/12/12/noticia-el-ejercito-argentino-presenta-formalmente-los-uav-lipan-m3-y-simulador-de-tiro-sitaran-ii.html

Javier, C., & Quintana, C. (2011). *Determinación de distancias entre objetos de una imagen*. Recuperado de http://bdigital.unal.edu.co/4204/1/CarlosJavierCastroQuintana.2011.pdf

Jiménez, E. (2009). *Medición de distancias por medio de procesamiento de imágenes y triangulación, haciendo uso de cámaras de video*. Recuperado de http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/jimenez_c_e/capitulo3.pdf

LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning*. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Loncomilla, P. (2016). *Deep learning: Redes convolucionales*. Recuperado de https://ccc.inaoep.mx/~pgomez/deep/presentations/2016Loncomilla.pdf

López Radcenco, M. (2010). *Introducción al reconocimiento de patrones*. Recuperado de https://eva.fing.edu.uy/file.php/514/ARCHIVO/2010/TrabajosFinales2010/informe_final_Lopez.pdf

Lowe, D. G. (2004). *Distinctive image features from scale-invariant keypoints*. International Journal of Computer Vision. Recuperado de http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

Martínez Puerta, J. J., & Vallejo Jiménez, M. M. (2016). *Comparación de estrategias de navegación colaborativa para robótica móvil*. Reecuperado de http://repositorio.autonoma.edu.co/jspui/handle/11182/935

Merino, L., Capitán Jesús, & Ollero Anibal. (2009). *Robótica cooperativa e integración con sensores en el ambiente. Aplicaciones en entornos urbanos*. Luis Merino 1, Jesús Capitán 2, Aníbal Ollero 2 - PDF. Recuperado el 25 de Junio de 2018, de http://docplayer.es/4974138-Robotica-cooperativa-e-integracion-con-sensores-en-el-ambiente-aplicaciones-en-entornos-urbanos-luis-merino-1-jesus-capitan-2-anibal-ollero-2.html

Miguel, J., Hernández, G., & Martinsanz, G. P. (2011). *Técnicas de procesamiento de imágenes estereoscópicas*. Recuperado de http://www.cesfelipesegundo.com/revista/articulos2011/Guerrero, J.M.pdf

Mordvintsev, A., & Abid, K. (2013). *Introduction to SURF (Speeded-Up Robust Features) — Opencv-python tutorials 1 documentation*. Recuperado el 28 de Junio de 2018, de http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html?highlight=surf

Murphy, & Robin. (2000). *Introduction to AI robotics*. Recuperado de http://www.profesaulosuna.com/data/files/ROBOTICA/ROBOTICS EBOOKS/Introduction to AI Robotics.pdf

Niedfeldt, P., & Beard, R. (2015). *Convergence and complexity analysis of recursive-RANSAC: A new multiple target tracking algorithm*. IEEE Transactions on Automatic Control, 1–1. https://doi.org/10.1109/TAC.2015.2437518

NVIDIA. (2010). *Deep learning | NVIDIA developer*. Recuperado el 27 de Junio de 2018, de https://developer.nvidia.com/deep-learning

NVIDIA Corporation. (2018a). *GeForce GTX 970M | specifications | GeForce*. Recuperado el 3 de Julio de 2018, de https://www.geforce.com/hardware/notebook-gpus/geforce-gtx-970m/specifications

NVIDIA Corporation. (2018b). *Procesamiento paralelo CUDA | Qué es CUDA |NVIDIA*. Recuperado el 3 de Julio de 2018, de http://www.nvidia.es/object/cuda-parallel-computing-es.html

OpenCV Open Source Computer Vision. (2011). *Opencv: camera calibration with Opencv*. Recuperado el 27 de Junio de 2018, de https://docs.opencv.org/3.4.0/d4/d94/tutorial_camera_calibration.html

OpenCV Open Source Computer Vision. (2015). *Opencv: Introduction to SIFT (Scale-Invariant Feature Transform)*. Recuperado el 28 de Junio de 2018, de https://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html

OpenCV Open Source Computer Vision. (2017). *Opencv: Optical flow*. Recuperado el 18 de Agosto de 2018, de https://docs.opencv.org/3.4.0/d7/d8b/tutorial_py_lucas_kanade.html

OpenCV Open Source Computer Vision. (2018). *Opencv: ORB (Oriented FAST and Rotated BRIEF)*. Recuperado el 30 de Junio de 2018, de https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html

Pacheco, A. (2017). *Tesis que presenta*. Recuperado de http://www.ctrl.cinvestav.mx/~yuw/pdf/MaTesMLP.pdf

Packer, J., & Reeves, J. (2013). *Romancing the drone: Military desire and anthropophobia from SAGE to swarm*. Canadian Journal of Communication, 38(3). https://doi.org/10.22230/cjc.2013v38n3a2681

Pérez, C., Cámara, J., & Sánchez, P. (2016). *Introducción a la programación en CUDA*. Recuperado de http://riubu.ubu.es/bitstream/10259/3933/1/Programacion_en_CUDA.pdf

Perez, J., Serrano, C., Acha, B., Serrano, T., & Linares, B. (2013). *Red neuronal convolucional rápida sin fotogramas para reconocimientos de dígitos*. Recuperado de http://digital.csic.es/handle/10261/84753

Ramírez, H. (2012). *Modelado Cinemática de Robots*. Recuperado de http://www.utm.mx/~hugo/robot/Robot2.pdf

Riisgaard, S., & Blas, M. R. (2005). *SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping By the "dummies"*. Recuperado de https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

ROS Perception. (2009). *How to calibrate a monocular camera*. Recuperado el 27 de Junio de 2018, de http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

Rosten, E., & Drummond, T. (2006). *Machine learning for high-speed corner detection*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11744023_34

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). *ORB: an efficient alternative to SIFT or SURF*. Retrieved from http://www.willowgarage.com/sites/default/files/orb_final.pdf

Santos, J. M., Portugal, D., & Rocha, R. P. (2013). *An evaluation of 2D SLAM techniques available in robot operating system*. In 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR) (pp. 1–6). IEEE. https://doi.org/10.1109/SSRR.2013.6719348

Shi, J. (1994). *Good features to track*. Recuperado de http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf

Stricker, D. (2012). *Epipolar Geometry*. Recuperado de https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec05.pdf

Sturm, P. (2014). *Pinhole camera model*. In Computer Vision (pp. 610–613). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-31439-6_472

Tan, Y., & Zheng, Z. (2013). *Research advance in swarm robotics*. Defence Technology, 9(1), 18–39. https://doi.org/10.1016/J.DT.2013.03.001

Tang, Y. (2010). *TensorFlow*. Recuperado el 27 de Junio de 2018, de https://www.tensorflow.org/

Wasowski, A. (2018). *Documentation - ROS wiki*. Recuperado el 3 de Julio de 2018, de http://wiki.ros.org/

Wu, N. (2017). *Tensorflow detection model zoo*. Recuperado de https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Xie, L., Wang, J., Wei, Z., Wang, M., & Tian, Q. (2016). *DisturbLabel: Regularizing CNN on the Loss Layer*. Recuperado de http://arxiv.org/abs/1605.00055

Zhang, W., & Kosecka, J. (2006). *Generalized RANSAC framework for relaxed correspondence problems*. In Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06) (pp. 854–860). IEEE. https://doi.org/10.1109/3DPVT.2006.67