Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Prototipo para estimación de la pose de un sistema móvil mediante múltiples sensores basados en visión artificial

Prototype for estimating the pose of a mobile system using multiple sensors based on artificial vision

Airán Omar Villacorta Betancor

La Laguna, 11 de septiembre de 2020

D. José Ignacio Estévez Damas, con N.I.F. 43.786.097-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Prototipo para estimación de la pose de un sistema móvil mediante múltiples sensores basados en visión artificial"

ha sido realizada bajo su dirección por **D. Airán Omar Villacorta Betancor** con N.I.F. 44.735.453-Q

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 11 de septiembre de 2020.

Agradecimientos

A mi pareja por su apoyo incondicional.

A mi familia por apoyarme en todas mis decisiones de la vida.

A D. José Ignacio Estévez Damas por su ayuda, disposición y labor docente durante el desarrollo del trabajo de fin de grado y de la carrera.

Licencia

Resumen

El objetivo principal de este proyecto es desarrollar una plataforma de experimentación para la estimación de la pose de un sistema móvil mediante múltiples sensores basados en visión artificial.

Las posibilidades que ofrece este tipo de sistemas de visión artificial utilizando componentes de bajo coste hacen de él un valor atractivo para su uso en el campo de la robótica.

Los experimentos realizados para el desarrollo se han llevado a cabo mediante la comparativa de varios sensores, para determinar cómo se asemejan o difieren interactuando en un mismo entorno. Para ello, capturan la pose y orientación de una plataforma móvil, para lo que se hace uso de dos sensores que utilizan visión artificial con técnicas diferentes: marcadores fiduciales y una cámara fija.

Palabras clave: posicionamiento, sensores, pose, visión artificial, Raspberry Pi, Android

Abstract

The main objective of this project is to develop an experimental platform for estimating the pose of a mobile system using multiple sensors based on artificial vision.

The possibilities offered by this type of artificial vision system using low-cost components make it an attractive value for use in the field of robotics.

The experiments carried out for the development have been carried out by comparing several sensors, to determine how they are similar or different interacting in the same environment. To do this, they capture the pose and orientation of a mobile platform, for which two sensors using artificial vision with different techniques are used: fiducial markers and a fixed camera.

Keywords: positioning, sensors, pose, artificial vision, Raspberry Pi, Android

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1.- Introducción

El siguiente proyecto se basa en el desarrollo de una plataforma de experimentación para la estimación de la pose de un sistema móvil mediante múltiples sensores basados en visión artificial.

El proyecto busca ir al fondo de las técnicas básicas, entender bien las fortalezas y debilidades de las técnicas. Por lo que se ha preferido una implementación a bajo nivel para ir recopilando aquellos problemas que puedan surgir.

Los sistemas de sensado basados en visión artificial presentan una serie de ventajas e inconvenientes. Entre sus ventajas tenemos las siguientes:

a) Costes reducidos en relación a otros sistemas como, por ejemplo, aquellos basados en telemetría láser.

b) Existencia de algoritmos capaces de detectar objetos sólo a partir de la información visual en tiempo real. Los avances tecnológicos en los sistemas de sensado, los procesadores y en la propia naturaleza de los algoritmos permite abordar tareas cada vez más complejas bajo severas restricciones temporales.

c) La información proveniente de los sensores basados en visión artificial puede permitir no sólo la estimación de la localización relativa del objeto, sino también su orientación.

d) El objeto a detectar no requiere modificaciones a priori, aunque con añadidos que suponen costes mínimos se puede mejorar mucho la precisión de los sistemas.

e) La detección puede ser posible en una gran variedad de rangos de distancias siempre que se den las condiciones de iluminación y de visualización adecuadas.

Entre las desventajas más importantes nos encontramos estas otras:

a) La calidad de la detección se degrada de manera importante si las condiciones del entorno varían, por ejemplo, cuando la iluminación cambia de forma importante o las condiciones de los objetos situados en las escena analizada varían.

b) La precisión de la detección puede no ser suficiente para muchas aplicaciones críticas.

c) La calidad de la detección y la precisión de la misma pueden variar dependiendo de la situación del objeto en la escena analizada. Por ejemplo, parte de la escena sensada puede sufrir “ocultaciones” temporales, el sistema que porta el sensor y que puede estar en movimiento se ve sometido a vibraciones o cambios bruscos de iluminación, etcétera.

Dado este conjunto de ventajas y desventajas es interesante investigar el aprovechamiento de un conjunto de sensores redundantes que puedan obtener información complementaria y cuyas propiedades los hagan más o menos ventajosos en diferentes situaciones.

1.1. Descripción general del prototipo

El prototipo está diseñado para localizar una plataforma móvil en un entorno y consta de:

1. **Cámara fija** situada a cierta altura conectada a un computador que denominaremos **estación base**. En este trabajo, la estación base se corresponde con una placa de bajo coste, modelo Raspberry Pi 3B. Será la encargada de albergar la aplicación principal, que recibirá los datos proporcionados por dos sensores, un sensor de localización a bordo ubicado en la plataforma móvil y un sensor de cámara fija conectada a la estación base y que ejecutará un sistema de visión artificial. Procesará y analizará los datos recibidos por los sensores y mostrará una simulación de comparación de los datos recibidos en tiempo real.

2. **Plataforma móvil** con elementos característicos ubicados en su geometría relativamente fáciles de detectar mediante visión artificial. Denominaremos a estos elementos **balizas visuales.** De esta manera la estación base puede localizar visualmente a la plataforma móvil utilizando la cámara fija orientada y algoritmos de visión artificial. Las balizas visuales utilizadas para este trabajo son pelotas de 150mm de diámetro de colores rojo, azul y verde, que se posicionarán en una posición concreta en la plataforma, esto posibilitará posteriormente poder realizar los cálculos oportunos para obtener información relevante. También incorporará un sensor de localización a bordo.

3. La plataforma móvil lleva a bordo una cámara y un sistema de procesamiento, por ejemplo un móvil, capaz de obtener información de localización desde la propia plataforma. Este sistema móvil comunica la información de localización a la estación base. Llamaremos a este equipo **sensor de localización a bordo**. En nuestro caso el sensor de localización de a bordo corresponde con un dispositivo móvil android que incorporará una aplicación desarrollada de visión artificial.

El objetivo de este trabajo es desarrollar un prototipo que integre dos tipos de sensores para la localización de un robot / sistema móvil operado en interiores que ofrezcan cierta complementariedad:

a) Un sensor externo al sistema móvil, que localice el mismo y establezca su orientación a partir de la detección visual del sistema móvil y los parámetros geométricos y físicos del montaje.

b) Un sensor a bordo del sistema móvil que tenga el mismo objetivo: determinar la orientación y localización del sistema, aprovechándose del conocimiento del entorno, del análisis visual de la escena y los parámetros geométricos y físicos del montaje.

Para el sensor externo se ha optado por un sistema de visión artificial compuesto por una cámara y un computador. En este prototipo, el sistema móvil sería modificado con balizas visuales, tres esferas de colores rojo, verde y azul, de unos 15 cm de diámetro de modo que sean detectables por el sensor desde un punto en fijo en estancias de tamaño considerable. Este sensor se situaría en una estancia típica en una posición más bien cercana al techo, con la cámara orientada hacia el suelo en un ángulo alfa de modo que salvo ocultaciones divise una porción lo más amplia posible del suelo.

El sensor a bordo del sistema móvil también estará basado en visión artificial, y buscará obtener una medida de su localización y orientación con una precisión aceptable para las necesidades de aplicaciones de robótica móvil. Esto requerirá la utilización de marcadores fiduciales en posiciones conocidas de la estancia. Los marcadores fiduciales consisten en patrones visuales conocidos por el sistema de visión artificial, de modo que a partir de la imagen “aparente” recogida por el sensor puede determinarse la posición y orientación del mismo respecto de la cámara que ha obtenido la imagen.

El sistema descrito requiere por tanto la intervención sobre la estancia: sistema de visión artificial externo al sistema móvil y colocación de marcadores fiduciales, como

sobre la plataforma móvil: balizas visuales. A cambio, obtenemos las siguientes ventajas:

a) Desde el punto de vista de las aplicaciones, tendríamos un sistema móvil localizable en interiores con aplicaciones en diferentes ámbitos como el de la logística.

b) Desde el punto de vista de la investigación, permitirá el ensayo de algoritmos que combinan los aspectos complementarios de estos sensores o determinan la existencia de fallos sistemáticos. La probabilidad de fallo por ocultación de los sensores serán diferentes siendo en muchas situaciones más habitual la ocultación de marcadores fiduciales en el campo de visión del sistema a bordo. Por contra, la calidad y precisión de la detección lograda por el sistema externo será menos robusta y dependerá en mayor medida de la iluminación, de los errores de los parámetros geométricos del sistema y de la propia posición del sistema.

Objetivos:

1. Crear un sistema de localización para la plataforma móvil basado en marcadores fiduciales. Dadas las características de los dispositivos móviles, con integración de cámaras, sensores y procesadores con suficiente capacidad de cómputo, esta podría ser una buena opción para implementar el sistema de localización “a bordo”.

2. Crear un sistema de localización externo a partir de balizas de colores. Se pueden utilizar diferentes tipos de ordenadores, aunque una opción recomendable es una plataforma compacta y de bajo coste como los computadores “Raspberry-Pi”[11] que además disponen de cámaras de calidad suficiente.

3. Estos sistemas deben comunicarse para centralizar la información y poder ensayar algoritmos que combinan los datos de los diferentes sensores. El sistema que reciba los datos puede estar tanto en la plataforma móvil como fuera de la misma. Dada la flexibilidad requerida se propone que los datos sean recogidos y analizados por una “Raspberry-Pi” mediante un sistema bluetooth.

4. Los datos recibidos de las plataformas de detección deben poder ser analizados bajo un sistema de referencia común de forma que sean directamente comparables.

1.2. Antecedentes y Estado actual del tema

En el desarrollo de la robótica, uno de los problemas más importantes en los que se interviene en casi todos los desarrollos e investigaciones, es el posicionamiento y la dirección de los objetos, ya sea a nivel industrial para su uso en maquinaria de fabricación, logística, sistemas autónomos, detección de situaciones concretas, etc.

Desde los comienzos, ha sido necesario conocer la posición y la dirección de los objetos que se desean controlar mediante sistemas robóticos. [3][4][5][6]

La tecnología ha avanzado en esa dirección proporcionando sistemas muy precisos en el ámbito de localización de objetos en el exterior y en el interior. Por ejemplo, el GPS permite localización en exteriores o la utilización de sistemas laser y algoritmos que permiten localizar a partir de la estructura del entorno y mapas. En el caso de interiores se suelen utilizar sensores internos al robot, como por ejemplo los que se basan en odometría y externos a él, y por lo tanto, surgen técnicas variadas adaptadas al tipo de sensor.

Haciendo uso de los sistemas basados en sensores externos existe la posibilidad de utilizar sistemas basados en cámaras que detectan mediante visión artificial al robot. Es el caso, de los sistemas de posicionamiento en interior (IPS). Son sistemas ideales para aplicaciones dentro de la logística gestionada por robots o realización de tareas concretas.

Otra técnica utilizada de localización de bajo coste y que evita que el robot tenga que hacer mediciones del entorno (técnica que se suele utilizar en algunos casos) es mediante mapas bidimensionales, captados por cámaras externas ubicadas en el techo de la habitación. que comparten información y que tras una serie de operaciones morfológicas, se puede estimar la ubicación de los objetos.

También podemos encontrar sistemas de localización basados en técnicas de calibración de cámaras (EMGM). Este tipo de técnicas abordan el problema modelando los cuatro bordes de la perspectiva generando una cuadrícula que aproxima el área del entorno, calculando sus intersecciones y usándose como una matriz tridimensional.

Haciendo uso de los sistemas basados en sensores internos ubicados en el propio robot, podemos nombrar algunas técnicas para la detección de la posición en el interior. Por ejemplo, mediante la detección desde el robot de marcadores fiduciales, que son patrones fácilmente detectables por una cámara y de los que se puede

deducir su orientación y ubicación. Esta técnica, que será usada en este proyecto, se basa en métodos trigonométricos, y en el conocimiento de los parámetros intrínsecos de la cámara [6]. Estos marcadores, contendrán un formato reconocido por el robot y el cual determinará cuando realice una detección de uno de ellos su posición y orientación respecto de esos marcadores detectados. Puede ser un sistema válido para su uso en entornos de logística o de trabajos concretos en interiores o de desarrollo propio.

1.3. Fundamentos Matemáticos

En este apartado del proyecto trataremos los fundamentos matemáticos que se han utilizado en los cálculos que aplicaremos al sistema que desarrollaremos para poder transformar los datos que obtenemos de los diferentes sensores a un sistema válido común que nos permita realizar comparaciones entre ellos, datos que obtenemos desde la librería Aruco[1] desde la cámara externa.

Para realizar los cálculos nos basamos en una librería que ha proporcionado el tutor la cual usaremos para obtener la localización y la orientación de lo que se detecte a través de los sensores de la cámara fija y del sensor de localización a bordo. Esta librería incluye la traducción en programación de las funciones que veremos en este capítulo y algunas más.

El sistema de referencia que utilizaremos para realizar los cálculos para posteriormente poder obtener comparaciones será el de la cámara fija conectada a la estación base (Raspberry Pi).

Sistema basado en una cámara externa fija.

A continuación expondremos los elementos esenciales de la cámara, con el fin de poder comprender cómo los utilizamos.

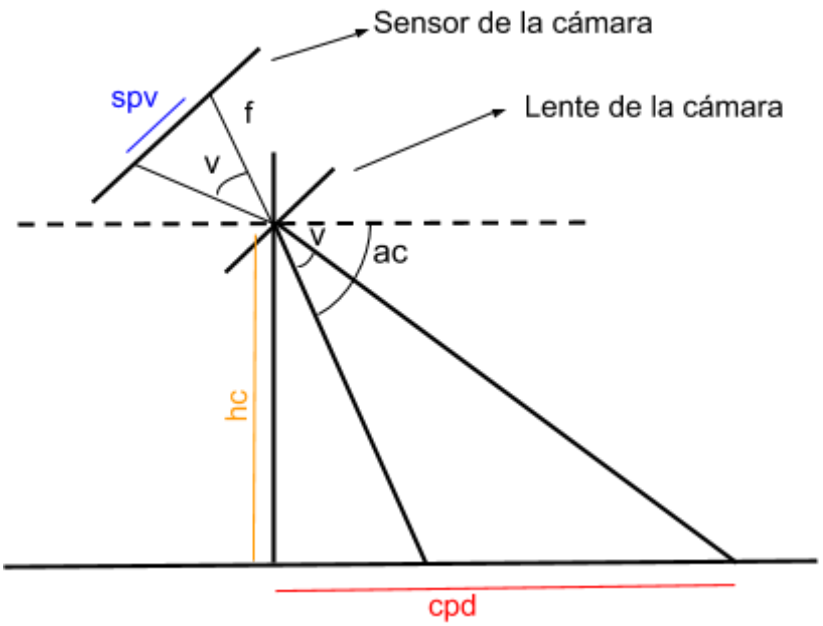

Figura 1.1. Elementos esenciales de la cámara.

αc = Inclinación de la cámara con $\alpha c = 0º$ cuando apunta al frente y $\alpha c = 90º$ cuando la cámara apunta al suelo.

f = distancia focal.
hc = distancia de la cámara respecto al suelo.
spv = distancia en el sensor respecto al punto central.
cpd = distancia del origen del rayo en el suelo respecto a la cámara.

Una captura por parte de la cámara para ser procesada obtiene una matriz de píxeles cuya dimensión depende de las características propias del sensor de la cámara. Una vez procesada esa imagen por el sistema de visión artificial, obtendremos una coordenadas en píxeles de la detección de los objetos de referencia o balizas en la imagen capturada.

En este punto, tenemos una posición de las balizas visuales respecto a la matriz de píxeles pero nos interesa conocer sus coordenadas en el espacio, en un sistema de referencia fijo para poder obtener la pose en el espacio.

Para ello tenemos que realizar una serie de operaciones con esos datos obtenidos por la cámara.

Previamente, hay que considerar ciertos aspectos para que los cálculos resultantes sean lo más precisos posible.

1. Tener en cuenta que el centro del origen (0,0) de la matriz de datos de la imagen capturada, no corresponde con el centro de la imagen que en nuestro caso corresponde con las coordenadas (640,360) en píxeles, y se ha de realizar un pequeño ajuste para que las detecciones capturadas sean correctas.

   Ajuste de la x en píxel (xp) con respecto a la x del centro de la imágen (cx).

   $$sph' = xp - cx$$

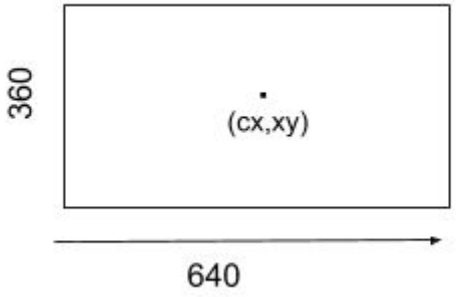

   Figura 1.2. Ajuste de la x en píxel con respecto a la x del centro de la imagen.

   Ajuste de la y en píxel (yp) con respecto a la y del centro de la imágen (cx).

   $$spv' = -(yp - yc)$$

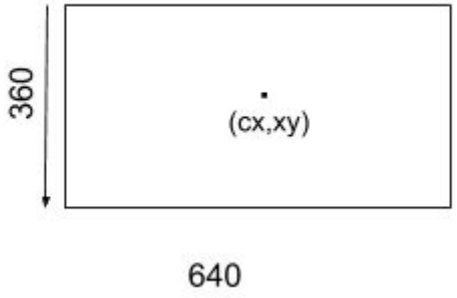

   Figura 1.3. Ajuste de la y en píxel con respecto a la y del centro de la imagen.

2. Formulamos las ecuaciones que nos permitirán obtener los datos geométricos de la x y de la profundidad (cpd).

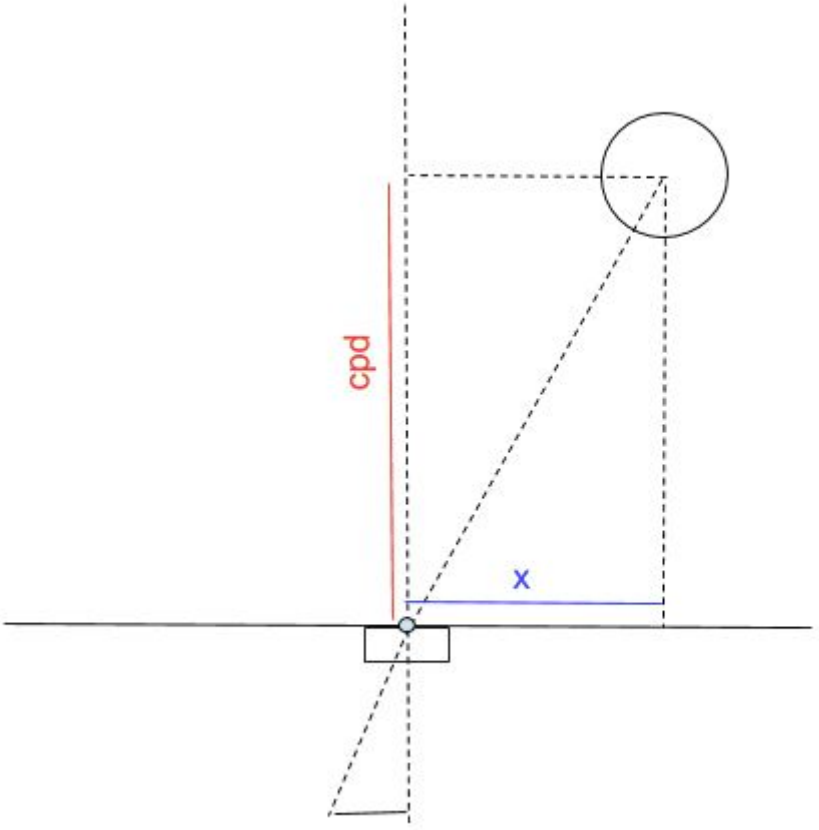

Figura 1.4. Datos geométricos de la x y la profundidad

La geometría de la x, proviene de las relaciones trigonométricas del sistema, siendo esta, una fórmula polinómica para adaptar las distorsiones de la cámara:

$$x = f(spv') \rightarrow x = c0 + c1 * spv' + c2 * spv'\hat{}2 + c3 * spv'\hat{}3$$

La geometría de profundidad (cpd), proviene de las relaciones trigonométricas del sistema:

$$cpd = k1 * \frac{(1 + k1 * k3 * spv')}{k2 - k3 * spv'}$$

donde:

$$k1 = hc$$
$$k2 = tan\,\alpha = \frac{hc}{cpd} = \frac{hc}{y\,píxel}$$

Con los algoritmos que procesan los datos recibidos desde el sistema de visión artificial, obtendremos los datos de profundidad geométrica (cpd) y la coordenada x geométrica, con esos datos, podemos tener la posición geométrica de la bola en el espacio.

La localización que obtenemos después de procesar los datos en base a una posición geométrica asume que el dispositivo móvil se encuentra en el centro de las balizas, es por ello, que debemos indicar en el parámetro correspondiente que su ubicación se ha modificado, en nuestro caso 160mm.

3. Para poder utilizar de manera adecuada los datos proporcionados por el sistema de visión artificial que hará uso de la cámara fija conectada a la estación base, ésta debe estar calibrada, este paso, corrige la distorsión que pueda tener la lente. Esta calibración nos devuelve unas constantes que nos permiten mediante una fórmula, calcular la posición estimada de cada una de las balizas detectadas.

   - Para calibrar el eje X se utilizará un ajuste polinómico mediante un algoritmo que estimará un valor función de una serie de medidas en píxeles captadas previamente por la cámara y que posteriormente utilizaremos para el cálculo de la coordenada X geométrica.

     $$x = c0 + c1 * sph' + c2 * sph'\hat{}2 + c3 * sph'\hat{}3$$

     Con el ajuste polinómico, obtenemos las constantes c0, c1, c2 y c3 que mejor resultado dan para esa serie de medidas previas que hemos captado.

   - Para calibrar el eje Y se utilizará una aproximación mediante un algoritmo que estimará un valor en función de una serie de medidas en píxeles captadas previamente por la cámara y que posteriormente utilizaremos para calcular la profundidad geométrica.

     El valor se denomina k3 y corresponde con:

     $\delta = 0.1$
     $\widehat{k3} = 0.1$
     for: cpd , spv' obtenemos $k3^{\,i}$
     $\overline{k3} = media(k3^{\,i})$
     $\widehat{k3} = \widehat{k3} + \delta * (\overline{k3} - \widehat{k3})$

dónde:

$cpd = coordenada\ y\ en\ píxel.$

Esta función se repetirá *n* veces y se realimenta con los datos obtenidos en la secuencia anterior, este algoritmo tiende a converger ajustando la fórmula.

Sistema basado en una cámara situada en un dispositivo móvil.

Aruco es una librería cuyas funciones se utilizan para detectar marcadores fiduciales. Con ello podemos estimar la posición de la cámara respecto al marcador.

Un marcador fiducial, es un marcador cuadrado compuesto por un borde negro ancho y una matriz binaria interna que determina su identificador, el borde negro permite una rápida detección en la imagen, y la codificación binaria, permite su identificación y la aplicación de técnicas de detección y corrección de errores.

El sistema de Aruco, cuándo realiza una captura de un marcador fiducial, devuelve la información de la pose del marcador mediante un vector de Rodrigues r^R para describir la rotación, un vector de traslación r^T y el identificador del marcador que ha detectado.

La traslación r^T es un vector de coordenadas del tipo $(\Delta x,\ \Delta y,\ \Delta z)$, es un sistema fácil de interpretar, ya que si observamos el vector tvecs, las coordenadas $\Delta x,\ \Delta y$ casi valen cero, mientras que Δz contiene el valor de la traslación que coincide con el de la distancia, aunque destacar que no siempre será así.

En OpenCV[15] se utiliza un sistema de codificación de las rotaciones basado en ángulo-eje.

- Sistema de Ángulo-Eje

Se utiliza un tipo de vectores de rotación llamados, vectores de Rodrigues. Funciona de la siguiente forma:

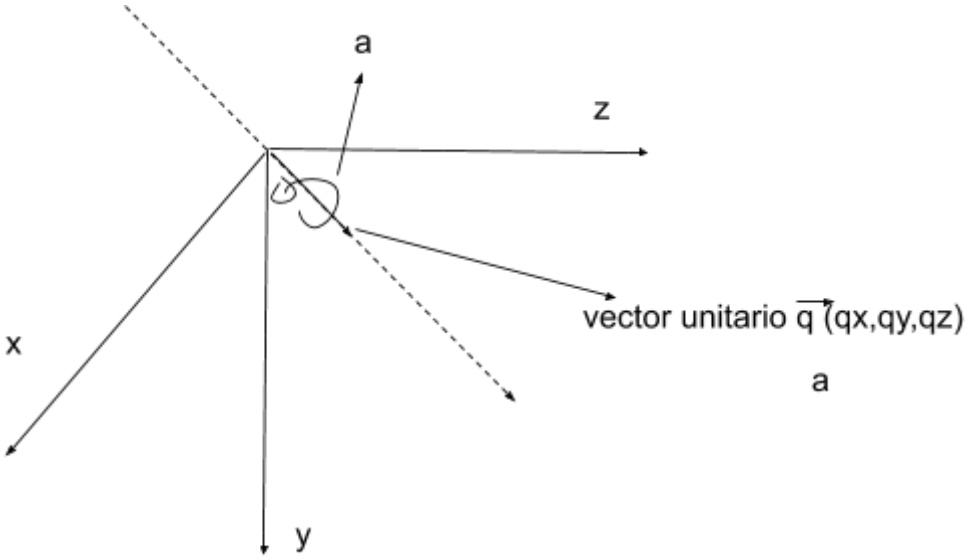

Figura 1.5. Vectores de rotación.

Una rotación no es más que un vector unitario indicando una dirección y un sentido del vector y un ángulo de giro (en el sentido de giro de un tornillo, sigue la regla de la mano derecha, el pulgar apunta donde gira el tornillo).

si denominamos al vector de rotación “q”, significa girar “a” entorno al (qx, qy, qz), una particularidad, es que solamente es necesario indicar la dirección y el sentido, por tanto, sólo es necesario tener dos coordenadas, ya que el módulo del vector (su tamaño) debe ser 1.

$$|q| = 1$$

Por tanto, el vector de Rodrigues no tiene cuatro magnitudes, sino solamente tiene tres.

$$r^R = (rx,\ ry,\ rz)$$

Entonces, si esa rotación queremos expresarla en función del vector de Rodrigues, que és lo que devuelve Aruco, lo único que deberíamos hacer es:

$\alpha = |r| \rightarrow$ el ángulo va a ser el módulo del vector
$(qx,\ qy,\ qz) = \frac{r}{|r|} \rightarrow$ las componentes normalizadas del vector van a ser las coordenadas.

Ejemplo de rotación de x expresada en función del vector de Rodrigues de un

ángulo π (Esto es, lo que nos devuelve Aruco, cuando tenemos la plataforma móvil frente al marcador)

Vector de Rodrigues : $(\pi,\ 0,\ 0)\ =\ r$

$|r| = \sqrt{\pi^{\,2} + 0^{\,2} + 0^2}\ =\ \pi\ \rightarrow\ \alpha = \pi$

$\frac{r}{|r|}\ =\ \frac{(\pi,0,0)}{\pi}\ =\ (1,0,0)$

Lo que significa, girar un ángulo de 180º alrededor de el eje colineal con el vector unitario (1, 0, 0) en torno al eje x, en el sentido que resulta de aplicar la regla de la mano derecha.

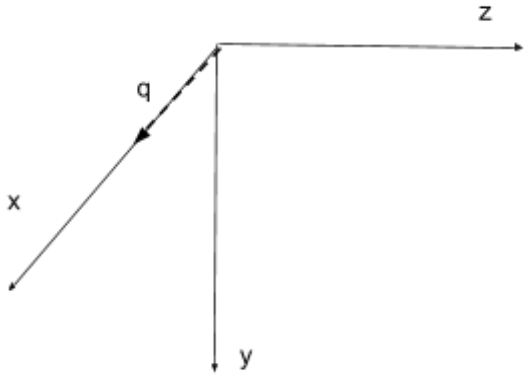

Figura 1.6. Rotación en términos del vector de Rodrigues.

Esta es la rotación típica que devuelve OpenCV en términos del vector de Rodrigues cuándo estima rotaciones de objetos, tal y cómo también hace Aruco.

Siendo de las formas que existen una de las más puras de expresar una rotación, con un vector y un ángulo.

Otra forma de expresar las rotaciones, es mediante matrices de rotación en un sistema de referencia fijo (método de Tait-Bryan). En ese sistema, la rotación se expresa como la matriz resultante del producto de otras tres matrices de rotación. Los factores representan la secuencia de rotaciones en torno a los ejes del sistema de referencias siguiendo un orden predeterminado, y siempre tomando el sistema de referencia sin rotar.

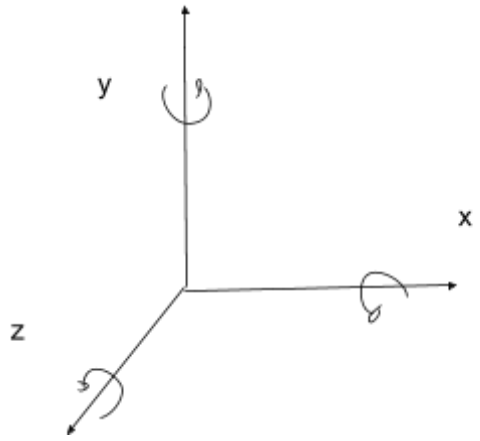

Figura 1.7. Rotación en torno a ejes fijos.

Se tiene un sistema de referencias y se definen las rotaciones en torno a ejes fijos. Significa que aunque hagamos una rotación, la sucesiva se realizará respecto al estado original.

Representadas mediante **matrices de rotación**.

El convenio que se sigue en los scripts y en el desarrollo es que las matrices de rotación transforman vectores columna, de modo que la aplicación de una rotación es como sigue:

$v' = R * r \rightarrow$ Donde tenemos primero la matriz de rotación y después el vector.

Una rotación γ en torno al eje x, viene dada por una matriz como la siguiente:

$$Rx(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & sin\gamma \\ 0 & -sin\gamma & cos\gamma \end{pmatrix}$$

Por ejemplo:

$\gamma = \pi$

$$Rx(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad v = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

se tiene que:

$$v' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

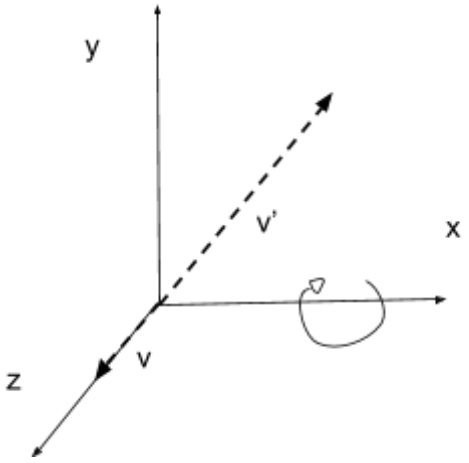

Figura 1.8. Vector unitario v'.

Si v es el vector unitario, v' sería el rotado 180º

Al igual que existe una rotación en x, existe también una rotación en z y en y.

$$Ry(\beta) = \begin{pmatrix} cos\beta & 0 & sen\beta \\ 0 & 1 & 0 \\ -sen\beta & 0 & cos\beta \end{pmatrix} \qquad Rz(\alpha) = \begin{pmatrix} cos\alpha & -sen\alpha & 0 \\ sen\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La manera de especificar una rotación R general en este sistema viene especificada por 3 ángulos. (La rotación en torno a cada eje)

$$R\,(\alpha,\ \beta,\ \gamma)\ =\ Rx(\gamma)\ Ry(\beta)\ Rz(\alpha)$$

Se define según el orden en el que se hagan las rotaciones, se escoge por convenio de primero se hace z, y, x

Existe una fórmula para pasar de vector de Rodrigues a matriz de rotación y

viceversa.

Existe una fórmula que permite realizar el paso de un sistema al otro. La propia librería de OpenCV tiene una función llamada Rodrigues que permite realizar este cambio, sin embargo nosotros utilizaremos las conversiones proporcionadas por la librería dada por el tutor.

Realizamos las conversiones y operaciones de paso de un sistema al otro:

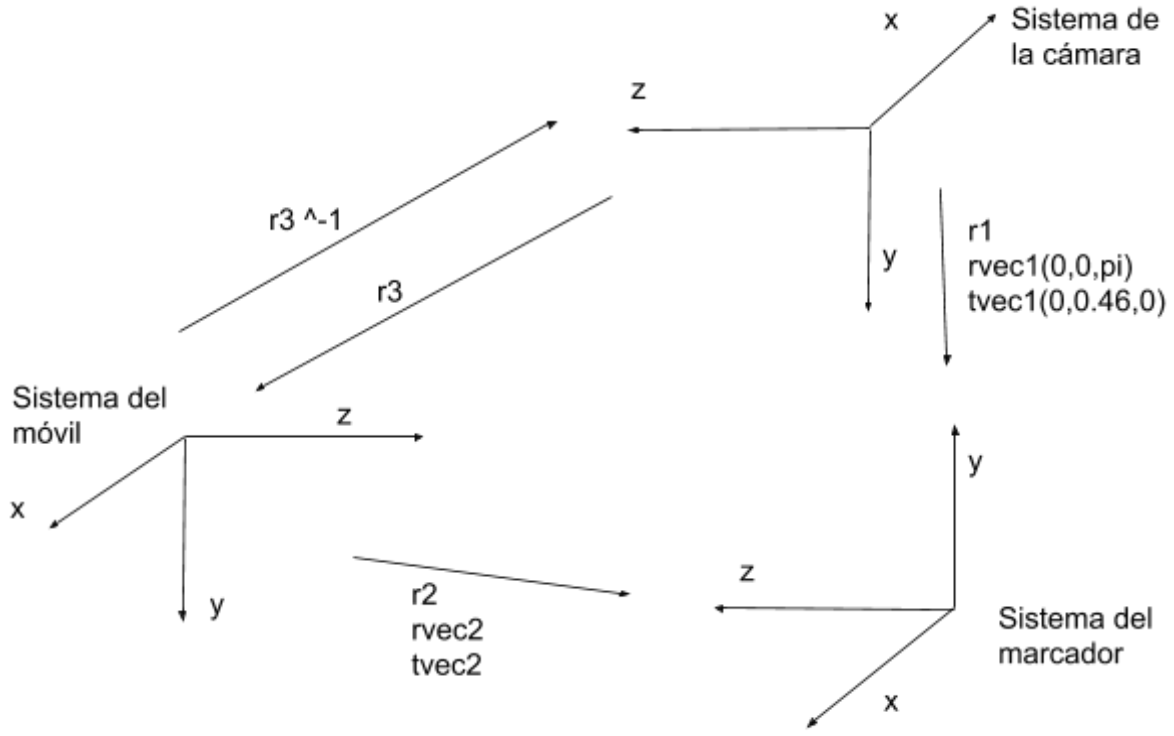

Figura 1.9. Operaciones de un sistema a otro.

En la figura se observan los diferentes sistemas de referencia que entran en juego. Recordemos que en el prototipo tenemos una cámara fija, una cámara situada en la plataforma móvil (con el sistema Aruco) y una colección de marcadores fiduciales. Además, hay que tener en cuenta que el convenio en OpenCV es utilizar sistemas de referencia orientados según la mano derecha (RHS), y con el eje z en el eje óptico de la cámara apuntando hacia afuera de la cámara y el eje y hacia abajo.

Lo que nos devuelve el sistema móvil (Aruco) es cómo pasar del sistema de referencia de la cámara situada en la plataforma móvil, al sistema del marcador, mediante una transformación r2 descrita por un vector de Rodrigues $r^R{}_2$ y un vector

de traslación rT_2. En el sistema del marcador el eje Z es perpendicular a la superficie y apunta hacia fuera del mismo y el eje Y apunta hacia la parte superior del marcador. Por tanto, la transformación de la que informa Aruco, se compone de dos partes: una primera consistente en una rotación en el sistema de referencia de la cámara que deja al marcador correctamente orientado, y una segunda que es una traslación hasta la posición geométrica que debe ocupar dicho marcador.

Lo que nos interesa es poder compartir la información de posicionamiento de la cámara fija con lo que nos da el móvil (Aruco). Para ello necesitamos conocer la situación de la plataforma móvil respecto a la cámara fija, es decir el vector de Rodrigues y el vector de traslación $r3 = (r^R{}_3, r^T{}_3)$.

Otra información que conocemos es como se encuentra el marcador respecto de la cámara, esos datos se encuentran parametrizados en el programa principal de la estación base y se genera un vector de Rodrigues con la siguiente información:

r1:
$r^R{}_1 = (0,\ 0,\ \pi)$
$r^T{}_1 = (0,\ 0.46,\ 0)$

r3 nos indica cómo está situado el móvil, si deshacemos la transformación de rotación de r3 hacia el otro lado, obtenemos la transformación inversa ya que todas las rotaciones pueden invertirse quedando $r3^{-1}$

Para obtener $r3$, podemos tener en cuenta que aplicar r1 desde el sistema de referencia de la cámara fija, es equivalente a aplicar primero $r3$ y luego $r2$: en ambos casos acabaremos transformando un vector del sistema de referencia de la cámara fija al sistema de referencia del marcador. Por ejemplo, en términos de matrices de rotación:

$R_2 * R_3 = R_1 \rightarrow$ Punto de partida

Para poder despejar R_3 multiplicamos ambos lados por $R^{-1}{}_2$ quedando:

$R_2^{-1} * R_2 * R_3 = R_2^{-1} R_1$

Cuando se multiplica un vector por su inversa obtenemos la matriz identidad por lo tanto podemos simplificar:

$R_3 = R_2^{-1} R_1$

Ahora, haciendo la inversa en los dos lados de la ecuación, obtenemos:

$$R_3^{-1} = (R_2^{-1} * R_1)^{-1}$$

La inversa de un producto de matrices es el producto cambiado de las inversas, es decir:

$$R_3^{-1} = R_1^{-1} * R_2$$

Y así obtenemos r3 y su inversa.

La orientación de la plataforma móvil en el sistema de referencia de la cámara fija puede obtenerse transformando un vector del sistema de referencia local. Por ejemplo, si $k_L = (0,0,1)^T$ es el vector unitario que apunta hacia el frente de la plataforma móvil, dicho vector en el sistema de referencia de la cámara fija sería $k_C = R_3^{-1} k_L$.

Todo esto se hace en términos de matrices de rotación, por tanto, tendremos que disponer en algún momento de r^R y r^T que es el vector de Rodrigues y el vector de traslación.

El punto de referencia es una ubicación en la geometría de la plataforma móvil. Su ubicación se establece mediante dos coordenadas (x,z) en el sistema de referencia de la cámara fija, asumiendo que la plataforma móvil está en su posición de referencia.

El punto de referencia del sensor de localización a bordo que estima la librería, no coincide con la ubicación real, es por ello, que se debe ajustar con los valores correctos para que el sistema conozca con detalle la geometría de la plataforma móvil y los elementos que la componen, balizas visuales y sensor de localización a bordo.

En el dibujo siguiente, se puede observar como se establece la geometría de la plataforma móvil y las balizas visuales. Cada componente se define por sus coordenadas (x,y) y cuyo origen es el (0,0).

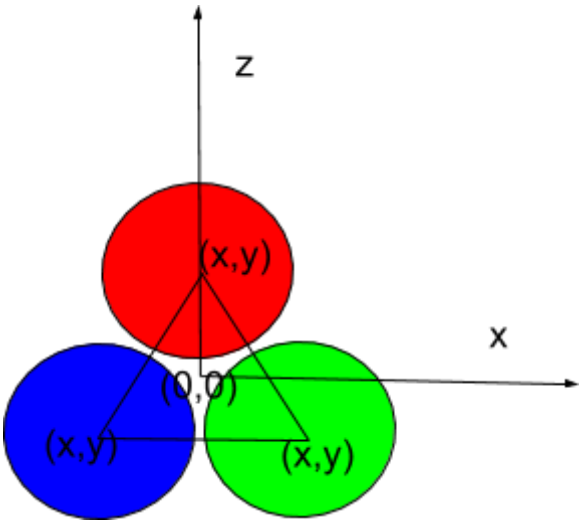

Figura 1.10. Posición de referencia de la plataforma móvil en el sistema de referencia de la cámara fija.

El punto de referencia del sensor de localización a bordo, el sistema por defecto asume que se encuentra en el origen, coordenadas (0,0), pero esto no es cierto, ya que según el diseño que se construyó de la plataforma móvil, el sensor de localización a bordo se encuentra justo delante de la baliza de color roja. Es por ello que debido a esto, se debe modificar el valor correspondiente a sus coordenadas para indicar al sistema que el punto de referencia del sensor en la plataforma ha cambiado, siendo ahora de (0,160). Quedando como se muestra en la figura siguiente:

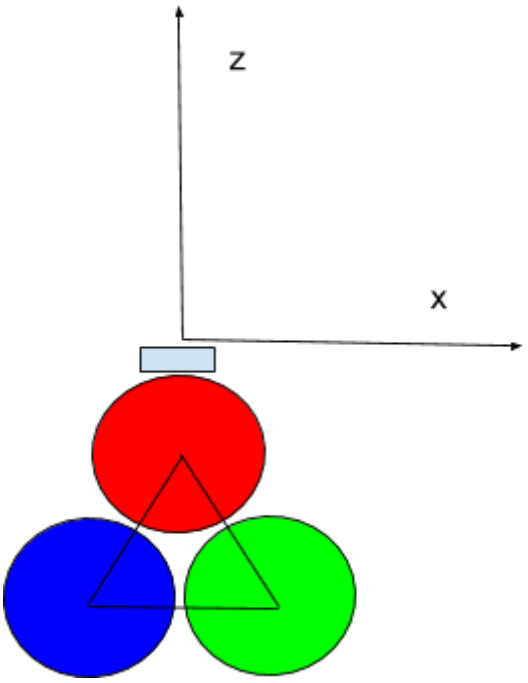

Figura 1.11 Posición de referencia del sensor de localización a bordo.

A la librería, le pasamos los datos de las balizas, se calcula una posición geométrica del punto de referencia, ese punto de referencia es el parámetro comentado anteriormente y que se le pasa a la función. Ese punto de referencia corresponde a la localización del dispositivo con el que se quiere comparar (el sensor de localización a bordo), que después de configurar el parámetro, se encuentra delante de las balizas.

El ángulo que nos devuelve la librería como 180º, sería el siguiente, si visualmente la plataforma móvil se encuentra en dirección a la cámara fija y con la baliza de color roja enfrentandola, ubicandose ambos en el mismo eje central.

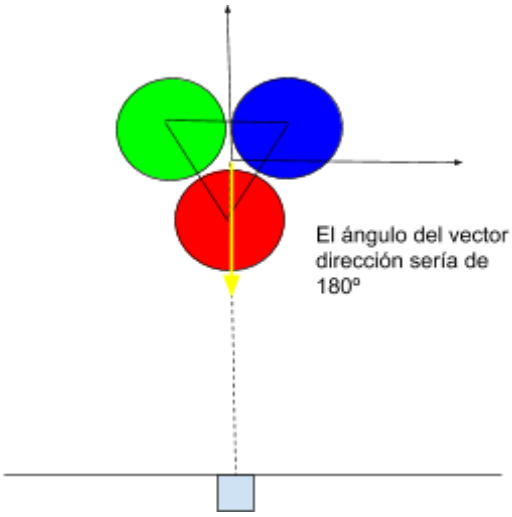

Figura 1.12. Posición de referencia respecto a la cámara fija.

En caso que la plataforma móvil estuviese direccionada con la baliza de color roja en la misma dirección que el sistema de visión artificial, el ángulo que tendría sería 0º.

Cálculo y orientación de la plataforma móvil.

Supongamos que r^R y r^B son dos vectores de posición en el sistema de referencia de la cámara fija indicando la ubicación de las bolas de color rojo y azul respectivamente en un momento determinado. Supongamos que ${r^R}_0$ y ${r^B}_0$, son las ubicaciones de dichas balizas cuando la plataforma está en su posición de referencia. La orientación de la plataforma puede determinarse a partir de la diferencia en los ángulos de los vectores r^{BR} y ${r^{BR}}_0$, que se definen como los vectores que van de la bola roja a la bola azul en la situación analizada y la posición de referencia mencionada. El sistema puede obtener así diferentes estimaciones del ángulo utilizando la variación detectada en las inclinaciones de r^{BR}, r^{GR} y r^{BG}.

Cálculo y posición de la plataforma móvil.

La localización del punto de referencia se puede obtener mediante cualquiera de las balizas detectadas, por ejemplo:

$$r^C = r^B + s^{CB}$$

donde r^C es el vector que indica la ubicación del punto de referencia y s^{CB} es el vector que va desde la bola azul hasta la posición de referencia. Para obtener este

vector la librería se basa en las posiciones de la bola azul y del punto de referencia cuando la plataforma está en su posición de referencia:

$$s^{CB}{}_0 = r^C{}_0 - r^B{}_0$$

y en el ángulo de rotación estimado de la plataforma móvil θ.

Entonces:

$$s^{CB} = R(\theta) s^{CB}{}_0$$

ya que se asume que el desplazamiento relativo del punto de referencia respecto de las balizas visuales es constante.

Esta interpretación, se hace de la misma forma desde la estimación de las balizas y desde los marcadores fiduciales, lo que permite poder realizar comparaciones de los ángulos directamente.

La estimación de ángulos realizada por la cámara que captura las balizas va a tener errores, ya que, una mínima diferencia en la construcción del sistema base, puede afectar en gran medida a un error mayor.

Cálculo y ajuste del tiempo.

Para el cálculo del tiempo, nos hemos basado en el sistema de tiempo del sensor de localización de a bordo de la estación base. Los tiempos que nos interesa ajustar son los recibidos a través del móvil. Para resolver este ajuste establecemos que:

t = *base de tiempos de la raspberry*
t' = *base de tiempos del móvil*

Presuponemos que la función que nos permite pasar un tiempo medio en el sensor de localización de a bordo en el tiempo equivalente es:

$$f(t')_{a \to b} = t' + \Delta$$

Para estimar el desfase entre las bases de tiempo, asumimos que el intervalo de tiempo entre el evento de envío de datos desde el sensor a bordo y el evento de recepción de los mismos en la estación base es de duración despreciables, con lo que:

$$t_{envío} = t_{recepción}$$

Respecto a la base de tiempos de la estación base:

$$f(t_{envío}')_{a \to b} = t_{envío}' + \Delta = t_{envío}$$

Si tenemos que:

$$t'_{envío} + \Delta = t_{recepción} \rightarrow \Delta = t_{recepción} - t'_{envío}$$

Cuándo recibimos los paquetes de datos, en cada dato del buffer, calculamos esa diferencia Δ de los datos del buffer que vengan desde el sensor de localización a bordo, y se calcula la media de todos los Δ obtenidos.

Por lo tanto, a los eventos del buffer llegados desde el el sensor de localización a bordo le sumamos el tiempo medio calculado Δ para ese buffer, y con eso, ya tendríamos todos los eventos en el mismo sistema de referencia de tiempos, el de la estación base, esto nos permite poder compararlos.

$$t' \rightarrow t' + \Delta$$

Este paso es necesario, ya que cada sistema se compone de un reloj de tiempo diferente y para poder comparar los datos, deben de estar todos en un sistema de tiempos común.

1.4. Planteamiento / Problemas a resolver

El siguiente proyecto se plantea de la siguiente forma:

Tenemos una aplicación principal que se encuentra en ejecución desde la estación base de la Raspberry Pi, siendo esa aplicación principal la encargada de recibir la información proporcionada por los sensores escogidos. Una vez reciba esa información, se encargará de analizarla bajo el mismo sistema de referencia adoptado y agruparla con el objetivo de que una serie de algoritmos la procesen y determinen su pose.

Los datos arrojados por los algoritmos desarrollados permitirán poder visualizar la información capturada de manera separada (sensor externo y sensor a bordo) y en

tiempo real, al igual que realizarán una comparación mostrando la diferencia entre ambos.

Con ellos podremos valorar la calidad de estimación de la pose del robot móvil realizada por diferentes sensores y visualizar los comportamientos según los pro y contras de los sensores utilizados.

Se posibilitará la inclusión de los datos capturados en un fichero externo para poder disponer de ellos y analizarlos externamente de cualquier otra forma.

Se dispondrá de un sensor ubicado en el robot móvil, éste dispondrá de una aplicación programada bajo android que calibre la cámara del dispositivo móvil y otra aplicación que realice capturas secuenciales y, determine la orientación y localización del robot, ambas aplicaciones realizarán una compartición de información, con el objetivo de que la cámara se encuentre calibrada para una detección eficaz. La información será enviada a la estación base por medio de algún sistema de comunicación inalámbrica.

Se dispondrá de un sensor externo al robot móvil, conectado directamente a la estación base, en nuestro proyecto, una placa de bajo coste. Este sensor será gestionado por una aplicación ubicada en la estación base que se encargará de monitorizar en tiempo real el espacio donde se encontrará el robot móvil, cuando lo detecte, se enviará la información a la aplicación principal para determinar su orientación y localización.

La información tratada se deberá mostrar en una interfaz gráfica con el fin de facilitar la visualización por parte del usuario.

Se fabricará una base móvil con los medios disponibles, que hará las funciones de robot móvil. Esta base contendrá las balizas de colores y el sensor móvil, deberá poder tener un tamaño adecuado a los objetos que debe soportar y una base firme para el sensor.

1.5. Estructura del documento

La memoria del Trabajo de Fin de Grado está compuesta por los siguientes capítulos:

- **Introducción:** En este capítulo se tratan los antecedentes, objetivos, alcance, fundamentos matemáticos y organización del TFG.

- **Herramientas y tecnologías:** En este capítulo, se describen las principales características del hardware, del software y de las tecnologías utilizadas para el desarrollo del proyecto.

- **Desarrollo de la aplicación**: En este capítulo, se implementará el desarrollo de las aplicaciones que componen el proyecto.

- **Experimentación**: En este capítulo, se realizará una serie de pruebas para evaluar los datos obtenidos por el sistema desarrollado.

- **Conclusiones**: Conclusiones del proyecto.

- **Summary and conclusions**: Conclusiones del proyecto en inglés.

- **Bibliografía**: Referencias bibliográficas consultadas para la realización del proyecto.

Capítulo 2.- Herramientas y tecnologías

A continuación se van a señalar las principales características del hardware, del software y de las tecnologías utilizadas para el desarrollo del proyecto.

2.1. Visión Artificial

El avance tan significativo surgido en la robótica en estas últimas décadas viene precedido de una mayor facilidad de información y de acceso a sistemas de bajo coste para su desarrollo y fabricación. Esto ha supuesto que las capacidades de las que podemos dotar a un sistema robotizado son casi infinitas, entre ellas, la visión artificial.

La visión artificial nace de la necesidad de dotar de capacidad visual a las máquinas. La tendencia siempre ha sido simular el comportamiento humano en los robots y una parte muy importante es la vista.

Visión artificial es la extracción de información del mundo real a través de imágenes haciendo uso de un sistema de cómputo.

Gracias a esta capacidad, un sistema robótico es capaz de determinar múltiples posibilidades obteniendo información de lo que tiene a su alcance, tales como: su posición, obstáculos a su alrededor, calcular distancias, reconocer objetos, incluso realizar tareas muy complejas que requieran de una precisión muy detallada. Los sectores entre los que hacen un uso muy activo de esta tecnología son: la alimentación, el transporte, la medicina, la logística, la tecnológica, etc.

Existen técnicas de visión artificial que permiten mediante una serie de operaciones obtener información tridimensional a partir de imágenes bidimensionales pertenecientes a una captura.

- **Descripción de elementos que componen sistema de visión artificial:**

Se trata de emular artificialmente el comportamiento del ojo humano, que captura la luz reflejada de su entorno de manera tridimensional, por un sistema de visión computarizado haciendo uso de una cámara conectada a un ordenador y cuyas capturas las obtiene de forma bidimensional.

Esto provoca una cantidad notable de pérdida de información de una escena capturada, que se tratará de mitigar aplicando una serie de técnicas para poder obtener información relevante del entorno, y poder determinar la orientación y localización del robot.

- **Representación de las imágenes por un sistema de visión artificial**

Las imágenes con las que trabajan los sistemas de visión artificial son bidimensionales, esto es debido a cómo funciona la captura de una imagen por el sensor de la cámara, ya que transforma la luz obtenida en una matriz de puntos de dimensiones M x N, donde cada uno de los elementos que lo componen los denominamos pixel.

- **Modelos matemáticos y técnicas que describen el comportamiento**

No existe una técnica óptima que sea aplicable a todos los casos de visión artificial, es por ello que se debe aplicar la más conveniente según el caso, para así poder tener el mejor resultado posible.

Encontramos dos grandes bloques dentro de las técnicas ópticas, las técnicas activas, que controlan la fuente de luz de las escenas ubicando puntos de iluminación conocidos y que reducen el problema de iluminación del entorno, y las técnicas pasivas, que no hace uso de esos métodos y que en algunos casos puede afectar a la exactitud de la medida.

Aun así, ambas técnicas se ven afectadas por la iluminación, teniendo que considerarse en el diseño.

Las técnicas ópticas pasivas se aplican a un mayor número de casos. En nuestro desarrollo utilizaremos un sistema de técnica de óptica pasiva.

Dentro de las principales limitaciones de estos sistemas al aplicar estas técnicas para la reconstrucción tridimensional desde escenas bidimensionales podemos nombrar:

- **Iluminación**. Es una de las principales limitaciones que nos encontramos, la parte más importante, una escena con una incorrecta iluminación provoca pérdida de información que se traduce en un mayor error en la reconstrucción. También interfieren otros factores que pueden comprometer la técnica escogida, cómo la variación de la luz en el tiempo o que algún objeto del entorno pueda inferir en un momento concreto variando la

iluminación. Para reducir este impacto, se puede hacer uso de técnicas para limitar el tiempo de exposición de las cámaras o modificar el espectro de luz adecuándose al uso propio.

- **Ocultación**. Producida por cambios en la superficie de las escenas que provoquen la ocultación de los objetos o que cambios bruscos de luz produzcan ocultaciones ocasionados por alteración de la forma o color del objeto o producidas por sombras de elementos que se encuentren en el entorno. Para corregir este tipo de errores, se puede proceder a añadir más cantidad de cámaras al entorno o modificando la posición de la cámara para poder captar toda la superficie de manera que se evite ese tipo de eventos.

- **Textura**. Cambios en las texturas de los objetos provocan que se determine una detección errónea del objeto sobre la escena capturada, para solucionar este tipo de casos, se suele procesar las capturas para reducir errores en las medidas (en nuestro caso, el cálculo del área del objeto)

- **Dispersión del patrón seleccionado**. Puede ocasionar que el patrón escogido para detectar el objeto en el entorno pueda provocar que se realicen detecciones espurias con el mismo patrón en la escena, esto hay que tenerlo muy en cuenta en este tipo de sistemas ya que, en caso contrario, estaríamos detectando objetos o secciones que no corresponden.

- **Reflexión**. Tanto la superficie del objeto a detectar cómo la escena dónde se realiza el trabajo, se debe controlar que no sean reflectantes, esto podría provocar que se hagan detecciones del patrón en ubicaciones que no corresponde. En el robot se aplica una capa de pintura mate para reducir el efecto.

- **Movimiento**. En casos que la frecuencia del sensor sea muy baja con respecto al movimiento del objeto en la escena, puede provocar desenfoque en las capturas obtenidas.

- **Componentes de un sistema de visión artificial para la reconstrucción 3D de una imagen 2D.**

La información geométrica del sistema de visión artificial, viene determinado por cómo se distribuyen los objetos en el espacio y sus distancias relativas en el entorno.

Según la distribución del entorno, las características de los objetos que lo componen y la configuración y ubicación de los dispositivos de captura, se establecerá unos parámetros determinados que validarán el sistema en función del problema a determinar y el entorno de trabajo.

- La iluminación

La iluminación es la forma que utilizan los sistemas de visión artificial para captar información relativa de los objetos contenidos en el entorno. Parte fundamental tanto en la captura, cómo en la elección posterior de cómo abordar el tratamiento y análisis de las imágenes capturadas, ya que determinará en gran medida, según los propósitos del análisis, el cómo abordar el problema y qué técnicas aplicar.

La iluminación puede ser tanto natural cómo artificial, en sistemas tan sensibles como estos, lo recomendable y siempre que se pueda, es utilizar sistemas artificiales que nos permitan en un entorno controlado, limitar los casos que generan fallas en el análisis.

- La cámara

Es un dispositivo cuya función es captar toda la luz reflejada por un objeto dentro de una escena tridimensional y que tras unas transformaciones matemáticas es capaz de generar una imagen bidimensional. Se conecta al sistema de visión artificial y es quien provee de la información captada.

Formada por un sensor fotosensible cuya finalidad es transformar la luz recibida en señal eléctrica. Los elementos a tener en cuenta en la elección de una cámara para según qué casos de uso podemos determinarlos por las características de su sensor: materiales de fabricación de su sensor (analógica o digital), su forma (lineal o matricial), dimensiones (píxeles con que genera la imagen), ruido (modificaciones en las imágenes generadas), entre otras.

- **Definición y comportamiento de la luz para la detección del color.**

Una parte muy importante en la detección de las balizas por parte del sistema de visión artificial pasa por el manejo y control de la luz, este apartado puede distorsionar de manera muy notable el espectro de luz que refleja las balizas.

En el sistema de captura de imágenes por parte de la estación base a través de la cámara que tiene conectada se realiza mediante detección de colores en las imágenes captadas.

Es el principal inconveniente de este tipo de sistemas de tracking ya que se ha de ajustar de una manera muy detallada el espectro de luz que se desea capturar. Es por ello que el sistema está diseñado para su uso en espacios de interior con las condiciones adecuadas.

OpenCV trabaja con el sistema de colores BGR (Otros sistemas suelen utilizar la convención normal de RGB), para poder obtener un rango mayor de configuración, utilizaremos el sistema HSV en sustitución de BGR.

El sistema HSV nos permite poder configurar el matiz, la saturación y el brillo que deseamos captar.

En la componente H, que determina el matiz, podemos configurar el espectro de color a captar.
En la componente S, que determina la saturación, podemos configurar la intensidad de más grisáceo a color más puro.

En la componente V, que determina el brillo, podemos configurar el brillo de más oscuro a más claro (negro-blanco)

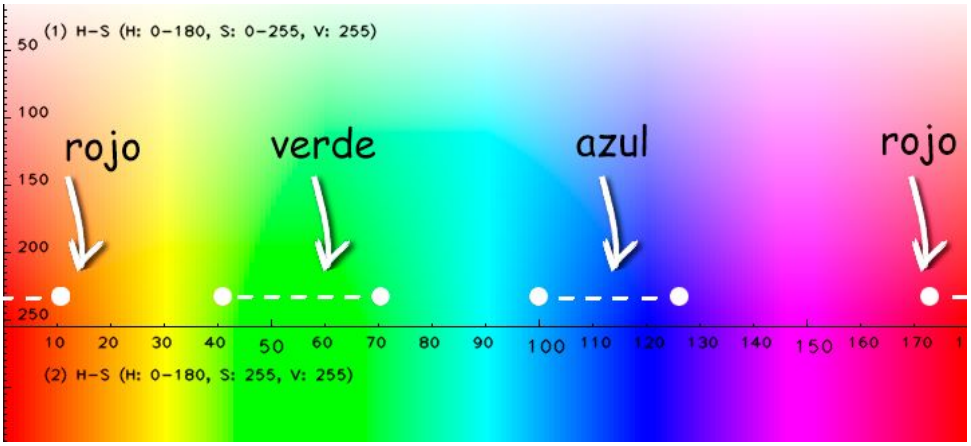

Figura 2.1. Espectro de colores sistema HSV

2.1.1. OpenCV

Es una biblioteca optimizada para visión artificial en tiempo real, multiplataforma, lo podemos encontrar para su uso en Windows, Linux, MacOS, iOS y Android. Compatible en desarrollos bajo los lenguajes de C++, Python[18] y Java.

Está publicada bajo licencia de código libre BSD, su estructura es modular y su versión extendida incluye muchas bibliotecas compartidas, cómo el caso de la librería Aruco. La biblioteca OpenCV contiene cientos de algoritmos de visión artificial, lo que la hace atractiva para su uso en cualquier desarrollo.

OpenCV facilita de manera muy considerable el estudio y la comprensión de las técnicas, al igual que el desarrollo por parte de los programadores, esto es gracias a que se encuentra desarrollado en una forma que aíslan a las personas que lo utilizan de la complejidad y de los cálculos matemáticos de los que hacen uso.

Dispone de los siguientes módulos generales:

- **Core funtionality (core):** Define las estructuras básicas de datos, la matriz multidimensional Mat y las funciones elementales utilizadas por el resto de módulos.

- **Image processing (imgproc):** Para el procesamiento de imágenes, filtrados lineales y no lineales, modificaciones geométricas, conversiones de espacios de color, etc.

- **Video analysis (video):** Para la detección y seguimiento de objetos, estimación de movimiento, sustracción de fondo, etc.

- **Camera calibration and 3D reconstruction (calib3d):** Para la calibración de la cámara, estimación de la pose de un objeto, elementos de reconstrucción 3D.

- **2D Features frameworks (features2d):** Para detectar características destacadas.

- **Object detection (objdetect):** Para detectar objetos, se puede hacer uso de las instancias predefinidas (caras, ojos, coches, etc.).

- **High-level GUI (highgui):** Una interfaz sencilla de utilizar para desarrollo de interfaces de usuario.

- **Video I/O (videoio):** Una interfaz sencilla para captura de video en tiempo real y códecs de videos para formato fichero.

Las librerías que componen la biblioteca de OpenCV, ofrece una gran cantidad de funciones que permite una gran cantidad de posibilidades dentro de la visión

artificial. Entre ellas destacan las transformaciones en imágenes o videos; detección, seguimiento y reconocimiento de objetos; captación de imágenes en tiempo real o desde un directorio determinado; etc.

Un problema frecuente de todas las cámaras son los fenómenos de distorsión que frecuentemente tienen las ópticas que incorporan, esto es un problema ocasionado por las lentes que utilizan y los rayos de luz, y que se debe abordar ya que en caso contrario las operaciones que se realicen sobre las imágenes darán resultados no correspondientes a la realidad.

La eliminación total de la distorsión es casi imposible, ya que se trata de un problema complejo, pero que posible de mitigar a unos niveles aptos para casi corregirlo. Esto se realiza mediante una serie de algoritmos que se aplican sobre unos patrones que determinan los parámetros de corrección para luego hacer uso de ellos.

Para tener una idea visual de este efecto, dentro de los diferentes tipos de distorsiones, podemos encontrar:

- Distorsión de Corsé
- Distorsión de Barril
- Sin Distorsión

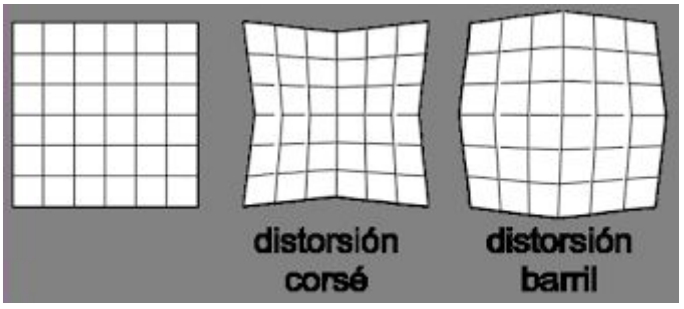

Figura 2.2. Tipos de distorsiones [7]

Es por ello de la necesidad casi imperativa de aplicar algún medio de calibración previa para poder hacer un uso adecuado de la visión artificial.

En el proyecto, utilizamos dos variantes de calibración. La primera variante es ofrecida por Aruco, basada en la calibración tradicional de OpenCV, el cual en vez

de utilizar un tablero de ajedrez para eliminar la distorsión de la cámara, utiliza un sistema de patrón a través de marcadores fiduciales. Y otra variante de calibración que se ha desarrollado con el fin de mostrar de una manera menos opaca el proceso de calibración. Todo ello podemos verlo más detenidamente en cada uno de los apartados 3.2.

En nuestro desarrollo podemos diferenciar según el sensor desde el que se observe diferentes usos de los métodos y algoritmos que nos proporciona OpenCV y Aruco. Entraremos más en detalle dentro de cada uno de los apartados del capítulo 3.

Instalación:

Para la instalación de OpenCV en Raspbian[17] se han realizado los siguientes pasos. La instalación se va a realizar bajo Python 2.7 que ya viene incorporado en la versión de Raspbian Strech.

La versión de OpenCV que instalaremos en la Raspberry Pi será la 4.4.0

Paso 1. Buscamos actualizaciones de los paquetes y el sistema y los actualizamos.

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

Paso 2. Añadimos los paquetes y librerías necesarios para la instalación de OpenCV.

Para permitir la carga de varios formatos de imágenes:

```
$ sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev
```

Para permitir la captura o lectura de video desde disco o en tiempo real:

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
```

Paso 3. Realizamos una ampliación de la memoria caché para ayudar en la ejecución de los hilos durante la compilación, este paso es momentáneo hasta que termine la compilación e instalación, ya que de no revertirlo podemos estropear la tarjeta SD. (Esto se debe a que la escritura/lectura en este tipo de tarjetas viene limitada a una cantidad determinada por el tipo de tecnología que utiliza. La

memoria caché es un espacio de memoria que copia el contenido de la memoria RAM a la memoria SD para liberarla y poder hacer uso de ella, el tamaño predeterminado de esta memoria caché es de 100MB, en nuestro caso lo ampliaremos a 2048MB).

- Editamos el fichero dphys-swapfile:
  $ sudo vi /etc/dphys-swapfile

- Comentamos la línea actual del parámetro CONF_SWAPSIZE=100 y añadimos debajo la siguiente:
  CONF_SWAPSIZE=2048

- Reiniciamos los servicios para que se actualicen los parámetros:
  $ sudo /etc/init.d/dphys-swapfile stop
  $ sudo /etc/init.d/dphys-swapfile start

Paso 4. Instalamos OpenCV.

(Añadimos el argumento --no-cache-dir para evitar que utilice la caché y acceda a buscar los datos a la fuente)
$ sudo pip install opencv-python --no-cache-dir

Paso 5. Verificamos la instalación.

Abrimos desde la terminal Python: $ python
Realizamos una importación de la librería cv2 de OpenCV: >>> import cv2
Comprobamos la versión: >>> cv2.__version__

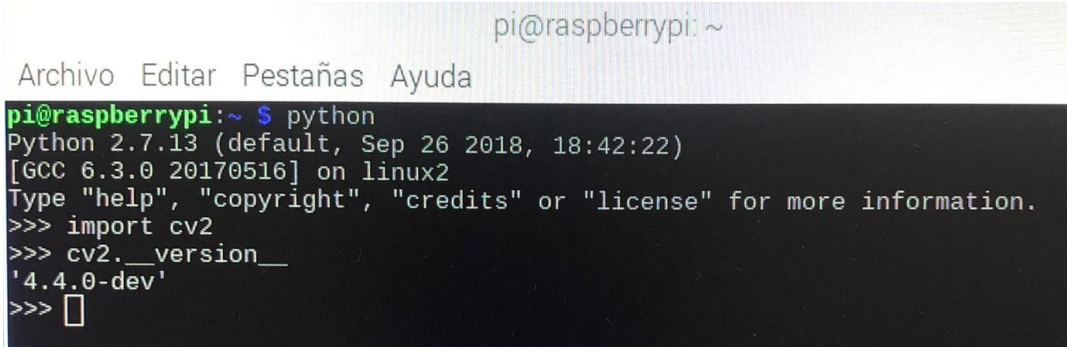

Figura 2.3. Terminal.

Paso 6. Revertimos los cambios de ampliación de la memoria caché.

Editamos nuevamente el fichero del Paso 3 y quitamos el comentario que insertamos y borramos la línea añadida, guardamos y volvemos a reiniciar los servicios.

Una vez instalado, ya podemos hacer uso de OpenCV para nuestro proyecto.

2.1.2. Aruco

Aruco es una librería de código abierto para detectar marcadores fiduciales en imágenes cuadradas. Podemos realizar estimaciones de pose respecto de los marcadores siempre que la cámara esté calibrada.

La biblioteca se encuentra escrita en C++ y se puede implementar junto con OpenCV en cualquier sistema que soporte OpenCV, podemos incluirlo dentro de las librerías de opencv_contrib o descargarlo desde la fuente y compilarlo.

- Marcadores Fiduciales.

Un marcador aruco, es un marcador cuadrado compuesto por un borde negro ancho y una matriz binaria interna que determina su identificador, esta matriz puede variar en tamaño permitiendo que entre mayor sea, más cantidad de información puede albergar. Entre más cantidad de datos contenga el marcador, más nitidez y precisión en la captura se ha de tener.

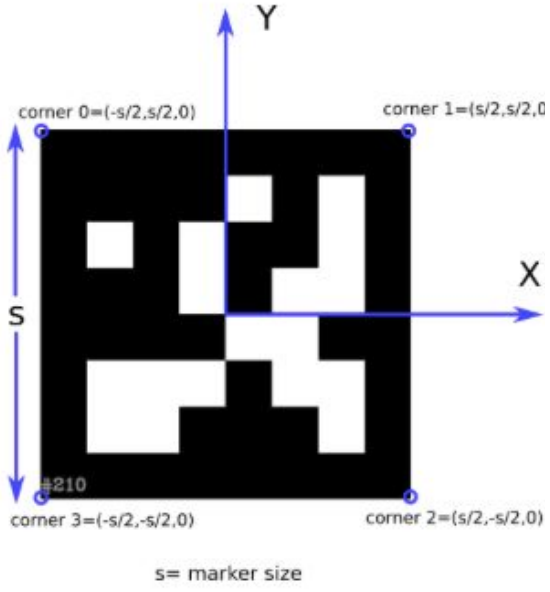

Figura 2.4 Ejemplo de marcador fiducial Aruco [2]

El borde negro facilita su rápida detección en la imagen y la codificación binaria, permite su identificación y la aplicación de técnicas de detección y corrección de errores.

Para una mejor detección se recomienda dejar un borde blanco en el exterior del marcador para una mejor detección de las esquinas.

El tamaño del marcador determina el tamaño de la matriz interna. En nuestro caso utilizaremos un marcador 4*4 de 100mm de lado compuesto por 16 bits.

Los marcadores se pueden generar haciendo uso de la función drawMarker(), una vez generados, se imprimen y colocan en el entorno de pruebas donde se utilizarán.

Existen numerosos tipos de marcadores, cada uno de ellos, pertenecen a un diccionario concreto. Cada biblioteca ha propuesto su propio conjunto de marcadores con el fin de diferenciarse de los demás, la librería Aruco es capaz de detectar los marcadores de otras bibliotecas.

Cada marcador, es un vector de cuatro puntos que corresponden con las esquinas de la imagen del marcador, una única identificación por cada marcador, un tamaño medido en metros y la rotación y traslación que determinan la pose del centro del marcador y la ubicación de la cámara.

Cómo ventajas podemos indicar la gran precisión que incorpora este tipo de sistemas, cómo inconveniente podemos destacar que este sistema puede sufrir ocultaciones puntuales de los marcadores o puede ser que en una posición concreta no detecte ningún marcador y no sea capaz de estimar su posición.

Existe otro fenómeno que se da cuando las detecciones se realizan en base a un sólo marcador fiducial, un marcador podría proyectarse en los mismos píxeles en dos ubicaciones de cámara diferentes. En general, la ambigüedad puede resolverse si la cámara está cerca del marcador. Sin embargo, a medida que el marcador se hace pequeño, aumentan los errores en la estimación de las esquinas aumentan y la ambigüedad surge como un problema.

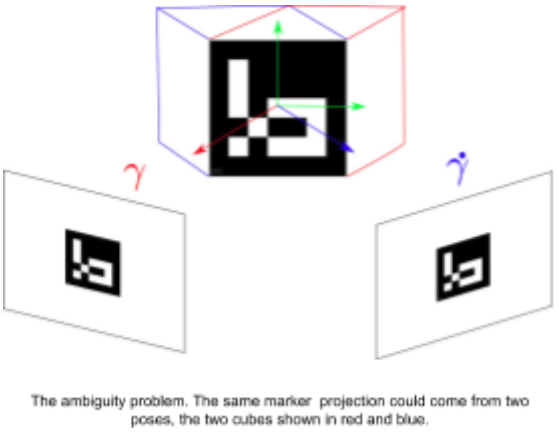

Figura 2.5 Ejemplo ambigüedad en marcador fiducial Aruco [2]

Es un buen sistema para combinarlo con algún otro tipo de sensor, en nuestro caso lo estimaremos con un sistema de visión artificial.

- Sistema de calibración

Ya se ha explicado en capítulos anteriores la necesidad de realizar una calibración de la cámara para poder tener los valores adecuados para realizar capturas y detecciones precisas.

En este caso veremos cómo es posible realizar una calibración utilizando la librería aruco. Esta calibración nos devolverá los parámetros que debemos ajustar a la hora de proceder a realizar las capturas.

El módulo de OpenCV permite realizar una calibración mediante un tablero de ajedrez usando las funciones calibrateCamera() pero podemos utilizar la librería Aruco que nos proporciona un sistema de calibración basado en las esquinas de los marcadores que es mucho más versátil que el sistema de OpenCV al permite oclusiones o vistas parciales, cosas que el sistema de OpenCV no permite.

La función proporcionada para realizar la calibración se llama calibrateCameraAruco(), y para una correcta calibración se recomienda realizar detecciones desde diferentes puntos de vistas.

- Detectar un marcador.

La función encargada de realizar las detecciones de marcadores se llama detecMarkers(), esta función devuelve un vector de traslación tvecs, un vector rotación rvecs y un identificador del marcador detectado.

Una vez detectado, podemos realizar ciertas operaciones sobre la imagen capturada, pintando los bordes de las esquinas del marcador, el sistema de ejes de coordenadas o pintando la información del identificador con el fin de visualmente identificar qué marcador es. Para ello utilizamos la función drawDetectedMarkers()

El sistema de ejes utilizado por Aruco y OpenCv en general, es el siguiente:

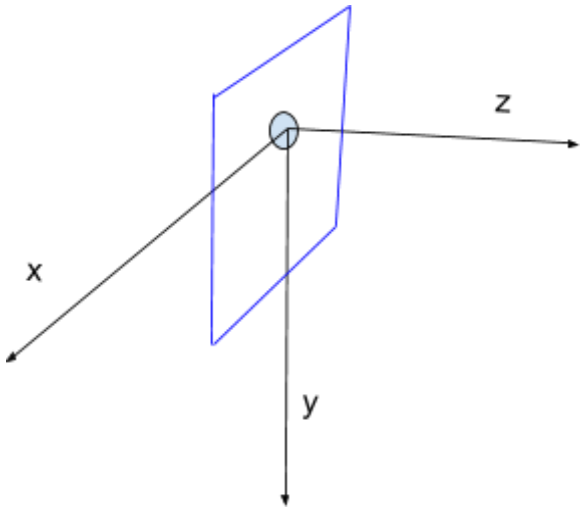

Figura 2.6. Sistema de ejes utilizado por Aruco.

Aruco cuándo realiza una captura de un marcador, devuelve la pose del marcador con un vector de rotación tvecs, un vector de traslación tvecs y el identificador del marcador que ha detectado.

Para indicar la pose del marcador, lo que hace Aruco es considerar que originalmente el marcador está en el origen de coordenadas del sistema de la propia cámara, con la imagen apuntando hacia afuera. Con el dibujo al revés apuntando hacia abajo, ya que el eje y se encuentra apuntando también hacia abajo.

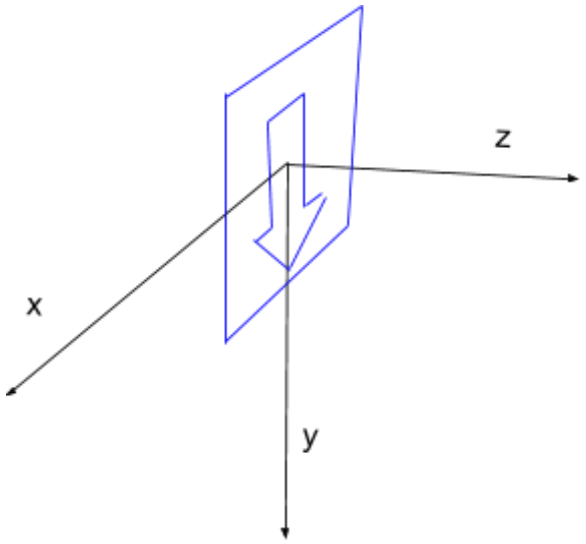

Figura 2.7. Sistema de ejes.

El sistema de referencias real de cómo se visualiza el marcador es:

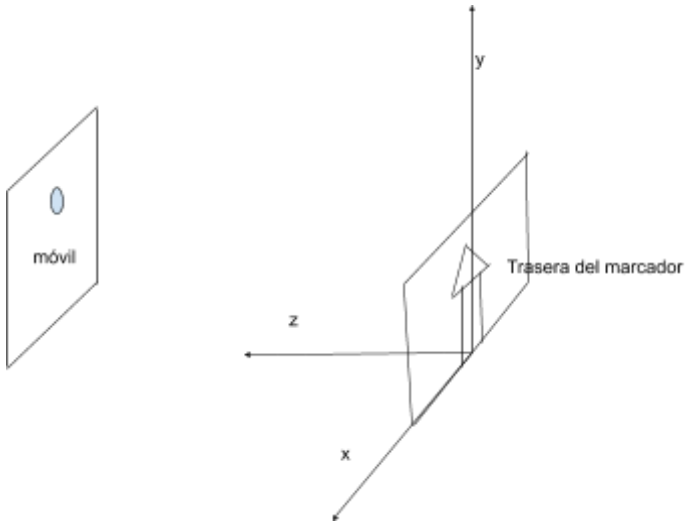

Figura 2.8. Sistema de referencias.

La librería de Aruco procesa esa información de dos pasos:

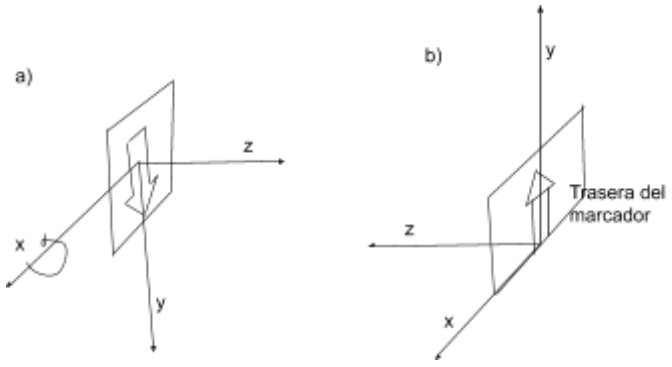

Figura 2.9. Procesamiento de información.

Tenemos que pasar de la posición a) cartulina mirando a la derecha, hacia b) cartulina mirando a la izquierda y modificando la orientación de la flecha de abajo hacia arriba.

Para pasar desde a) hasta b) se realizan 2 fases.

- Una primera fase, donde se realiza una una rotación in situ, en el propio sistema de la cámara. Rotamos el eje x en sentido contrario a las agujas del reloj un ángulo de 180º.

- Una segunda fase, donde una vez rotado se realiza una traslación de una cantidad determinada en el eje z, dando como resultado b).

Aruco nos devuelve la información cómo:

$rotación\, x = \pi$
$traslación\, z = \Delta z$ (positivo, ya que sale)

2.2. Estación base: Raspberry PI 3B

La Raspberry Pi es una placa de bajo coste que contiene un ordenador totalmente funcional, desde sus versiones más básicas encontramos la Raspberry Pi Zero con un costo sobre los 5,55€ (Desde su web oficial) hasta su última versión la Raspberry Pi 4B que incluye hasta 8GB de RAM con un coste de 83,99€. Esto posibilita que se pueda utilizar un ordenador de bajo coste en infinidad de proyectos, desde educativos, hasta soluciones empresariales.

Esta idea surge a través de Raspberry Pi Foundation en el Reino Unido, que tenían como objetivo el poner en manos de las personas de todo el mundo el poder de la informática y la creación digital. Cómo se comentó anteriormente, el bajo coste de estas placas ha permitido un amplio uso en el campo de la robótica, surgiendo a raíz de su popularidad, todo tipo de componentes que pueden ser agregados para desarrollar proyectos de todo tipo.

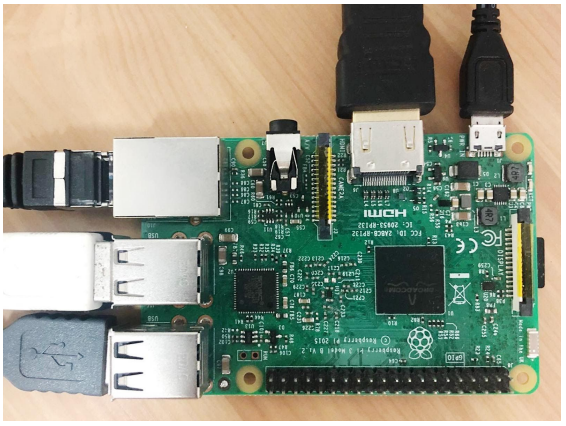

Figura 2.10. Raspberry Pi.

En nuestro desarrollo, hacemos uso de una placa de bajo coste, la Raspberry Pi modelo 3B, con un precio de 39,90€ y que incluye las siguientes características:

- Procesador Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- Cuenta con 1GB RAM
- Sistema de comunicación inalámbrica: BCM43438 wireless LAN y Bluetooth Low Energy (BLE) integrado
- Conexión de red 100 Base Ethernet
- 40-pin extended GPIO
- 4 puertos USB 2.0
- Salida HDMI
- Conexión del puerto cámara CSI para conectar la cámara de Raspberry Pi
- DSI display port para conectar a la Raspberry Pi una pantalla táctil.

- Un puerto Micro SD para cargar cualquier sistema operativo.

Se ha decidido escoger la placa Raspberry Pi como estación base por ser un sistema bastante potente que funciona muy bien y por un coste mínimo. Incorpora la posibilidad de tener un sistema operativo instalado que funciona muy bien con los periféricos que se le instalen. Es una buena forma de tener la potencia suficiente que se necesita cuando se trabaja con visión artificial y al mismo tiempo incorpora una serie de componentes propios de alta calidad que se integran de manera muy sencilla, entre ellos una cámara propia. Este tipo de placas se utilizan en infinidad de proyectos makers, de prototipado y robóticos.

2.1.1 Raspbian

Raspberry PI OS (previamente llamado Raspbian), es el sistema operativo oficial para todos los modelos desarrollados por Raspberry Pi.

En este caso se utiliza la versión Strech de Raspbian, se ha decidido utilizar esta versión concreta para la estación base del proyecto porque trabaja de manera estable con los componentes tanto de software como de hardware utilizados en el mismo.

Después de un análisis previo para determinar los diferentes medios de comunicación disponibles a utilizar y del software que se quería implementar, se ha podido observar que en algunas versiones de raspbian se comportaba en determinados casos de manera no estable.

También se valoró el uso de linux bajo una versión de ubuntu para la Raspberry Pi, pero después de un análisis exhaustivo se prefirió utilizar la versión de raspbian por contener una documentación más completa y encontrarse en un estado más estable por ser la distribución más utilizada y completa de todas las existentes para Raspberry Pi.

Una característica que agrega Raspbian respecto a otros sistemas linux disponibles, es que incorpora una aplicación propia que permite una configuración bastante sencilla de los componentes más importantes de la placa. La aplicación “raspi-config” permite realizar ajustes de manera gráfica de los dispositivos que dispone la placa, la memoria utilizada, configuración del arranque, etc. El cual en

otros sistemas se realiza de manera individual modificando los ficheros de configuración de cada uno de los dispositivos.

2.1.2. Python

Para el desarrollo del programa principal, la aplicación que gestiona el sensor externo, todos los algoritmos de los procesos que realizan los cálculos y la calibración de la cámara externa se harán con Python. Esto es debido a las grandes capacidades que nos proporciona este lenguaje, en temas de comunicaciones, visión artificial, herramientas para el tratamiento de los datos, interfaces de usuario, cálculos matemáticos, etc.

Python es un lenguaje de programación interpretado sin necesidad de ser compilado, que se caracteriza por su baja curva de aprendizaje respecto a otros lenguajes, notable reputación en el uso científico, multiparadigma usando la orientación de objetos, de tipado dinámico permitiendo la comprobación de las variables en tiempo de ejecución, multiplataforma ya que lo podemos usar bajo cualquier sistema Windows, Linux, MacOS, etc., integrado por defecto en Raspbian, con una gran cantidad de bibliotecas muy completas y robustas enfocadas al cálculo matemático, numerosas bibliotecas centradas en la gestión del sistema de red, puertos seriales y gestión de bluetooth, se integra muy bien con la visión artificial.

Dispone de librerías potentes de las que haremos uso, como:

- **Numpy**: Potente herramienta de computación numérica, proporciona casi todos los elementos necesarios para realizar los cálculos que necesitamos.

- **Tkinter**: Nos proporciona la capacidad de diseñar interfaces de usuario amigables y funciones de interacción con la interfaz.

- **Threading**: Módulo que permite trabajar el paralelismo basado en hilos.

- **Pandas**: Proporciona herramientas para la manipulación y el análisis de datos.

- **Time**: Proporciona funciones para trabajar con el tiempo.

- **Pathlib**: Utilizado para el manejo de ficheros de datos orientado a objetos.

- **Numpy.Polynomial**: Provee de objetos para tratar con polinomios.

- **Socket**: Contiene librerías para manejar los canales de comunicaciones, en la misma máquina o entre varias, utilizado mucho en la programación de redes.

- **Cv2:** Para hacer uso de la librería de OpenCV que usaremos para la visión artificial.

- **DBus:** Librería que permite la comunicación entre procesos y servicios, permite notificar servicios con nombres determinados para la interacción con ellos, una vez publicado, este no cambia y queda disponible para que otro se conecte y haga uso del mismo.

2.1.3. Tarjeta microSD

Para la estación base compuesta por la Raspberry Pi, se ha hecho uso de una tarjeta microSD con las siguientes características:

- Marca: SAMSUNG
- Tipo de memoria: MicroSDHC
- Clase de Velocidad: Clase 10
- Lectura: 95 Mb/s
- Escritura: 20 Mb/s

Figura 2.11. Tarjeta SD.

2.3. Sensor de localización de a bordo: dispositivo móvil basado en Android con cámara y comunicación Bluetooth.

La utilización del dispositivo móvil, viene determinado por ser un dispositivo de fácil acceso, disponible en cualquier vivienda hoy día, pero a su vez es una parte crítica del desarrollo y planteamiento del proyecto. El dispositivo móvil utilizado para la realización del proyecto es un Huawei P Smart. Modelo FIG-LX1, con las características que se especifican a continuación:

- Número de compilación 9.1.0.115 (C109E6R1P1)
- Versión Android 9.0 Pie
- Cámara trasera: 13MP+2MP, f/1.8

El dispositivo móvil que se ha utilizado en este proyecto es el que se disponía debido a las circunstancias extraordinarias actuales y que ha determinado en gran medida las versiones de las plataformas de desarrollo a utilizar. Desde las versiones de compilación de Android Studio[9] para el código nátivo, la compilación de código Java, compiladores externos, la utilización de versiones determinadas de OpenCV y Aruco, el tratamiento de la privacidad y permisos por parte de los servicios y ficheros compartidos desde las propias aplicaciones Android, las conexiones y usos disponibles para las transferencias de datos o incluso en el desarrollo de las interfaces de las aplicaciones, entre otras. Es por ello que el proyecto dependía en gran medida de las características y posibilidades que permitía el dispositivo móvil disponible.

Adicionalmente, el uso del dispositivo móvil es de gran importancia por contener una cantidad notable de sensores integrados, giroscopios, acelerómetros, cámaras, etc. En nuestro caso, nos hemos centrado en el uso de la cámara y la detección de los marcadores fiduciales proporcionados por Aruco desde OpenCV, pero sería conveniente tener en cuenta los otros tipos de sensores de cara a futuras actualizaciones del proyecto.

2.3.1. Aplicaciones de código nativo

También desarrollaremos código nativo, que es una particularidad que permite el desarrollo en Android Studio.

Gracias a esta posibilidad podemos implementar código desarrollado en C++/C para el uso de aquellas librerías que se encuentran disponibles en ese lenguaje y en otros, adicionalmente, podemos acceder gracias al código nativo al hardware de un modo más amplio que el utilizado bajo el lenguaje de disponible por Android

Developer en Java, ya que la gran mayoría de funcionalidades vienen limitadas a un uso concreto y con esta posibilidad podemos acceder a un control total del hardware del dispositivo móvil.

Todo ello incrementa las posibilidades, podemos aumentar el rendimiento de las aplicaciones desarrolladas, sobre todo las que necesitan de mucho proceso computacional como lo es la visión por computación o incluso los videojuegos.

Para entender cómo funciona código nativo en Android Studio, es conveniente conocer cómo se establece la unión entre el código nativo y el código Java, al igual que los procesos que intervienen entre ellos.

Existen las bibliotecas compartidas y estáticas, el NDK es el encargado de hacer las compilaciones necesarias de las bibliotecas para el código nativo. También encontramos la interfaz nativa de Java (JNI), que es el encargado de la conexión y entendimiento entre el código nativo desarrollado en C / C++ y el código propio de Java. Luego tenemos la Interfaz binaria de aplicación (ABI), encargada de traducir a lenguaje máquina el código, permite compatibilidades en sistemas ARM, x86, x86_64, etc.

Para nuestro proyecto, hemos realizado una configuración específica de Android Studio para poder permitir integrar las bibliotecas que utilizaremos para el desarrollo de la visión artificial en el dispositivo móvil. Esta configuración viene determinada por la versión del sistema operativo que tiene el móvil.

Las librerías utilizadas de OpenCV en su versión 4.0.1 y Aruco en su versión 3.1.12 han sido compiladas bajo el software CMAKE, en su versión 3.18.0-rc3 que se incluye en configuración dentro de Android Studio. Para la compilación de las librerías externas se hizo uso de Visual Studio 16 2019.

Las librerías .so se han incorporado al proyecto dentro de la estructura src/main/ cómo una nueva carpeta llamada jniLibs/armeabi-v7a

Las librerías de OpenCV que incluyen la biblioteca de Aruco se ha agregado al proyecto como un nuevo módulo, posteriormente se han incluido en las dependencias de la aplicación para que estén disponibles para su uso.

Se tiene que incluir también un fichero, llamado CMakeLists.txt, donde se especificarán las versiones mínimas para realizar la compilación, ubicación de las bibliotecas agregadas, ubicación y nombre de los ficheros que contendrán el código nativo, linkeo y ubicación de los directorios de las librerías de las que haremos uso.

2.4. Cámara externa.

Se utilizó una webcam para la cámara externa conectada a la Raspberry pi (a partir de ahora estación base) y que será la encargada de realizar las detecciones del robot utilizando visión artificial.

Puesto que la calidad del sistema de visión artificial depende en buena medida de la cámara, sería interesante probar otras cámaras, por ejemplo la cámara específicamente diseñada para la Raspberry Pi.

Las características de la webcam son las siguientes:

- Marca: Creative Labs
- Modelo: VF0770
- HD 720p Video (1280*720)
- 30 FPS
- F/2.4 con un intervalo de longitud focal de f=1.92mm
- Tamaño del sensor 1.0 megapixel

La cámara externa estará ubicada en una plataforma conectada a la estación base con una inclinación y una distancia determinada y conocida.

2.5. Bluetooth

Bluetooth es una tecnología para redes inalámbricas desarrollada por el Institute of Electrical and Electronics Engineers (Estados Unidos). Esta tecnología permite la interconexión de dispositivos de corto alcance capacitando la transferencia de información entre ellos utilizando la radiofrecuencia en la banda de trabajo ISM a 2.4GHz. Funciona bajo redes WPAN, para conexiones de dispositivos cercanos entre sí. Entre las características más destacables encontramos el buen comportamiento en lugares cerrados, no necesita de infraestructura extra para crear la conexión, en las últimas versiones encontramos sistemas de bajo consumo y es buen aliado en sincronización de equipos próximos.

La Raspberry Pi 3B viene incorporada con varios sistemas de comunicación inalámbrica, entre ellos dispone de wifi 2.4GHz IEEE 802.11.b/g/n y bluetooth 4.1 (Low Energy).

A la hora de desarrollar el proyecto se ha tenido que escoger una tecnología común entre el dispositivo móvil y la raspberry pi, se ha determinado, que el sistema de comunicaciones bluetooth es muy apropiada para el tipo de desarrollo por ser un estándar muy utilizado en sistemas robóticos y que a diferencia de las plataformas de comunicación de tipo red, las bluetooth no requieren de infraestructura alguna ya que dispone de infraestructura propia para la conexión cliente/servidor sin necesidad de utilizar elementos externos, se comporta de manera estable en la comunicación entre las aplicaciones móviles Android y la placa raspberry pi.

El estándar Bluetooth se encuentra diseñado para dispositivos de conexión de corto alcance y de bajo consumo, cosa que por las versiones del dispositivo móvil y la raspberry pi comparten. Las conexiones se realizan bajo radiofrecuencia, lo cual posibilita que los dispositivos se encuentren en diferentes zonas dentro de su radio de actuación.

Para la comunicación entre el dispositivo móvil y la estación base se utilizará el protocolo bluetooth RFCOMM (Comunicación por radio frecuencia). El protocolo RFCOMM es un protocolo de transmisión de información, desarrollado sobre el protocolo L2CAP. El protocolo RFCOMM es un protocolo de conexión basado en puertos de entrada/salida, emula la comunicación entre los dispositivos a través de un puerto serie, lo que posibilita trabajar con estos sistemas de manera muy sencilla.

2.6. Herramientas de desarrollo

2.6.1. Android Studio

Android Studio es un entorno de desarrollo diseñado por Google, es el entorno oficial para el desarrollo de todos los productos de Android, desde aplicaciones móviles, aplicaciones para el coche, aplicaciones para el reloj o cualquier dispositivo que contenga Android como sistema operativo.

Este entorno nació como sustitución del IDE Eclipse. Su primera versión de Android Studio oficial estable se publicó a finales de 2014. La versión estable actual es la 4.0 que es la que utilizamos en el desarrollo de nuestro proyecto.

Se ha vuelto tan popular, gracias a que incluye numerosas herramientas que facilitan el desarrollo de software, se basa en el software IntelliJ IDEA de la compañía JeBrains, que lo publicó bajo licencia Apache 2.0. Muchos otros entornos de desarrollo se basan en ese software.

Android Studio se encuentra disponible para todas las plataformas, nosotros utilizaremos la versión de Windows 10.

Actualmente, se encuentra diseñado y preparado para el desarrollo en el nuevo lenguaje de programación utilizado por Google denominado Kotlin, pero en nuestro caso nos hemos decantado por utilizar Java por ser un lenguaje del que se dispone de numerosa documentación y por ya tener conocimientos previos del lenguaje.

Android Studio trae todas las herramientas necesarias casi de manera nativa para la programación y testeo de aplicaciones para dispositivos multiplataforma, ya sea móvil, reloj, televisores, etc. También permite la emulación de un terminal con unas características específicas, dando la posibilidad al desarrollador de ajustar diferentes dispositivos para las pruebas, al igual que integración con sistemas de control de versiones, etc.

Dentro de las características más importantes nos encontramos con:

- Build Systems: Contiene un sistema de compilado fundamentado en Gradle, el cual es un conjunto de herramientas que dan soporte en la creación de conjuntos de paquetes de software. Construido con la misma filosofía de Apache Ant y Apache Maven.

- Maven-based build dependencies: Es una parte fundamental en el desarrollo en Java, se trata de una herramienta que gestiona las dependencias y compilaciones en proyectos bajo el lenguaje de Java. Su formato basado en XML, permite a los desarrolladores describir y comprender los entresijos de cualquier proyecto descrito en Java.

- Build variants and multiple-APK generation: Permite compilar y generar desde un mismo desarrollo, diferentes compilados para múltiples dispositivos, desde móviles o tablets, hasta versiones diferentes dentro del mismo sistema, a estas versiones diferentes se las denomina (Flavors o Sabores).

- Editor gráfico de las vistas: Permite visualizar tanto gráficamente como mediante su código las vistas de cada pantalla de la aplicación de desarrollo, facilitando su diseño.

- Soporte NDK (Native development kit - Sistema de desarrollo nativo): Una de las partes más importantes y por la cual usamos Android a través de Android Studio para nuestro proyecto. Permite instalar y utilizar bibliotecas escritas en C / C++ al igual que en otros lenguajes, una vez compiladas y agregadas al proyecto para el dispositivo en cuestión se puede hacer uso de ellas, posibilitando un desarrollo más completo.

2.6.2. Raspberry Pi Imager for Windows

Es una utilidad oficial proporcionada por la plataforma de Raspberry Pi que lanzaron en marzo de 2020, que nos facilita la incorporación del sistema de Raspberry PI OS (previamente llamado Raspbian) en cualquier modelo de raspberry pi. Es una herramienta multiplataforma, en nuestro caso, la utilizaremos en Windows.

Como ventaja a destacar:

- Realiza una comprobación de la instalación en la SD y en la imágen del sistema utilizado.

- Dispone de una opción que permite borrar los datos de la SD para permitir posteriores usos o instalaciones.

2.6.3. Windows 10

Para el desarrollo de la gran parte del proyecto, se ha hecho uso del sistema operativo de Microsoft Windows 10. En él se han instalado todos los programas y entornos necesarios para el desarrollo, tales como, Android Studio, Sublime Text, Python, Visual Studio, CMake, Git[14], utilidades para la gestión de la SD usada en la Raspberry, etc.

Para la descarga e instalación de Windows 10, se ha procedido a hacer uso del servicio DreamSpark que ofrece la Universidad de La Laguna en conjunto con Microsoft, el cual deja a disposición de los estudiantes software y servicios de la marca de manera gratuita, todo ello para fines de aprendizaje, enseñanza e investigación.

2.6.4. Github

Antes de entrar en la explicación de qué es Github, es necesario conocer algunos conceptos previos.

Un **sistema de control de versiones** es un sistema que controla, desde que lo activamos sobre una carpeta, los cambios que se realicen en las subcarpetas y ficheros incluidos en el.

Existen varias operaciones que se realizan para indicar modificaciones y marcarlas, enviar cambios a un servidor para que quede disponible para otras personas, etcétera.

Un **repositorio**, por tanto, es aquella carpeta que incluye un sistema de control de versiones.

- Para marcar un cambio, se realiza la operación de commit sobre el repositorio. Esto permite indicar cuándo se ha realizado un cambio junto a una nota que suele ser a modo informativo. Pudiendo volver en cualquier momento a un commit anterior, o lo que es lo mismo, a una versión anterior del proyecto.

- Para recibir cambios realizados por otros miembros del grupo de desarrollo, utilizamos la operación pull.

- Para enviar nuestros cambios al servidor para que queden reflejados para el resto de desarrolladores utilizamos la operación push.

Github es una herramienta que nos permite ubicar nuestros repositorios en internet. Esto es muy útil cuando se quiere tener una copia en otro lugar del proyecto o para desarrollo colaborativo. No es la única herramienta que existe, pero es una de las más famosas. Existen otras cómo: Bitbucket, Gitlab, etc.

El proyecto se gestiona bajo el control de versiones de **Git**, utilizando como plataforma de gestión colaborativa de los repositorios la herramienta de Github.

Es necesario destacar que la integración con Android Studio, Raspberry PI y Windows 10, es bastante sencilla y permite una clara gestión y utilización de Github.

Los enlaces a los proyectos se encuentran a continuación:

- Aplicación sistema móvil de calibración: https://github.com/airanvillacorta/AppCalibracion3

- Aplicación sistema móvil de captura: https://github.com/airanvillacorta/AppCapture

- Aplicación Estación base con cámara: https://github.com/airanvillacorta/base-station

Capítulo 3.- Desarrollo de la Aplicación

Para el desarrollo de la aplicación tenemos por un lado el sensor de a bordo y por otro la estación base con una cámara. Por lo tanto, trabajaremos con dos tipos de sensores diferentes que presentan diferentes problemas en su uso:

Por una parte, tenemos un sensor basado en visión artificial, que es muy poco preciso, que depende bastante de la fuente de luz del entorno, etc.

Por otro lado, tenemos un sensor bastante más preciso, que se basa en la detección de patrones concretos con una precisión muy notable. Este tipo de sensor, aún siendo mucho más preciso, puede tener fallos muy graves, cómo la ocultación momentánea de algún patrón que se encuentre observando o que el entorno en el que se desenvuelve no esté bien preparado y pueda dejar de localizar esos patrones.

Pensamos en ambos casos para su uso en una habitación preparada y adaptada al uso.

Para nuestro desarrollo, disponemos de dos aplicaciones principales, el sensor de a bordo que trabaja bajo Android y una aplicación desarrollada en Python que se encontrará ejecutándose en la estación base (Raspberry Pi).

Se hará uso de una librería básica proporcionada por el tutor del proyecto que incluye funciones para obtener la localización y la orientación de lo que se detecte a través de los sensores de la cámara y del móvil. Esta librería incluye la traducción en programación de las funciones básicas a realizar con los datos obtenidos capturados, estas funciones podemos encontrarlas también en OpenCV, pero la usamos con el fin de comprender el funcionamiento del sistema. Esta librería permite hacer transformaciones entre la información recibida por aruco, de la cámara y al contrario. Es una librería que se encuentra en desarrollo y que en líneas futuras se seguirá desarrollando.

Incluye funciones para:

- Crear matrices de rotación en los ejes x y z.
- Devolver un vector de Rodrigues junto a un ángulo y un vector unitario en torno al que se realiza una rotación.
- Calcular la matriz de rotación a partir de un vector de Rodrigues.

- Pasar de una matriz de rotación a un vector de Rodrigues.

- Composición de dos rotaciones consecutivas con vectores de Rodrigues.

- Una función para obtener la pose, a la que le pasamos un vector de Rodrigues y una traslación desde el punto de vista del sensor móvil junto a un vector de Rodrigues del marcador y un vector de traslación respecto de la cámara y nos devuelve un vector de Rodrigues, un vector traslación y una matriz de rotación en el mismo sistema de la cámara.

- Una función de proyección de la rotación en un plano, que considera todos los casos de orientación del vector en función de sus coordenadas, considera los casos especiales y nos retorna el ángulo Theta.

- Una función que nos permite visualizar los datos que vamos generando en las sesiones de observación.

- Una función que devuelve el ángulo de un vector entre $-\pi \, y \, \pi$

- Una función que normaliza cualquier ángulo entre $0 \, y \, 2 * \pi$

- Dos funciones de las que obtenemos la orientación del sistema y su posición a partir de la posición medida de las balizas visuales y de sus correspondientes localizaciones en la posición de referencia.

3.1. Sistema de sensado de a bordo

El sistema de sensado de a bordo está compuesto por un terminal móvil bajo Android. La funcionalidad principal de este sensor, será haciendo uso de visión por computación utilizando OpenCV, más concretamente de la librería Aruco, de realizar detecciones de marcadores fiduciales ubicados alrededor del entorno de pruebas y enviarlos a una estación base que se encargará de utilizar esa información recibida para determinar, en la habitación donde se realizan los test, la posición y la dirección de la base de pruebas sobre la que se encuentra ubicado el sensor.

Como cualquier sistema que haga uso de cámaras, estas deberán de ser calibradas previamente para obtener datos coherentes con la realidad.

Este tipo de sistemas de detección incluye una serie de pros y contras que determinan la calidad y la eficacia cuando se aplica a un problema concreto.

La detección a través de marcadores fiduciales tiene como ventaja que sus capturas son muy precisas si las comparamos por ejemplo con el otro sensor, que utiliza un sistema de detección artificial basado en detección visual de balizas y que tiene muchos factores que intervienen en la calidad de la detección. La detección a través de marcadores fiduciales aporta una ventaja cuando se utiliza en interiores, ya que, en entornos controlados donde se verifique que sea cual sea la ubicación de la estación de pruebas hay suficientes marcadores en el campo de visión, en el caso ideal, siempre detecta un marcador, posibilitando conocer su pose y su orientación.

Como inconveniente, podemos destacar, que en espacios abiertos, no controlados, puede producirse situaciones en las que el sistema no es capaz de encontrar algún marcador y por tanto no podría conocerse en esos intervalos su posición. También puede producirse ocultaciones puntuales del sensor, por verse situado en línea con algún objeto que oculte el marcador, por tanto, aún controlando todo el entorno, pueden producirse ocultaciones fortuitas que no permitan conocer su pose y orientación.

Ya que uno de los objetivos del desarrollo es poder experimentar una comparación de los datos obtenidos por sensores, es ideal disponer de sistemas de detección diferentes con diferentes ventajas e inconvenientes, que complementandose, se consiga ajustar la detección consiguiendo minimizar los inconvenientes, en pro de las ventajas que ambos aportan a la detección.

Se han desarrollado dos aplicaciones bajo Android, una primera aplicación que se utilizará para realizar una calibración y que generará un fichero con los parámetros para calibrar la cámara del dispositivo móvil y que compartirá con la segunda aplicación que será la encargada de realizar las detecciones de los marcadores fiduciales y enviarlos mediante bluetooth a la estación base de la Raspberry Pi. Todo se ha desarrollado haciendo uso del entorno de desarrollo de Android Studio, junto con el sistema de desarrollo de aplicaciones nativas de Android NDK.

3.1.1. Aplicación de Calibración (AppCalibration).

3.1.1.1. Funcionamiento general

Cuando se inicia la aplicación, se carga la interfaz y permanece a la espera que se utilicen las funciones que mostramos a continuación.

1. **Capture.** Esta función realiza detecciones de marcadores fiduciales y haciendo uso de los datos devueltos los almacena a esperas que indiquemos

que queremos calibrar la cámara. Estas detecciones se realizan en un promedio de 10 capturas para determinar que se ha detectado el marcador, esto es una restricción que se ha añadido para evitar sobrecargar el sistema por la gran cantidad de cálculos que se realizan durante la detección. Lo ideal en este punto, tal y como indican los desarrolladores de la librería de Aruco, es realizar capturas desde diferentes ángulos y posiciones con el fin de tener unos datos de calibración óptimos.

2. **Calibrate.** Una vez se han realizado varias capturas, se puede hacer uso de la función de calibrar, esto genera un fichero de texto con los parámetros intrínsecos y los coeficientes de distorsión de la cámara.

   Este fichero será el que se comparta con la segunda aplicación y que será la encargada de realizar las detecciones en el funcionamiento normal del sensor.

   A continuación, mostramos el fichero que utilizamos con los datos de calibración ya incluidos.

```
%YAML:1.0
---
calibration_time: "Sat Apr 25 13:17:08 2020"
image_width: 1280
image_height: 720
flags: 0
camera_matrix: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 1.2388857047573306e+03, 0., 6.2907517301799135e+02, 0.,
       4.9380841660650435e+02, 3.4282601897972847e+02, 0., 0., 1. ]
distortion_coefficients: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data: [ 2.6648489671759714e+00, 2.5295206089027623e+01,
       7.4183652960375035e-01, -1.9758136100564092e-01,
       -2.1641351628720531e+02 ]
avg_reprojection_error: 2.3656559859472737e-01
```

Figura 3.1. Datos de calibración de Aruco.

Destacar que la calibración en una cámara, sólo es necesario realizarse una sola vez.

3. **Discard.** Se dispone de otra función que permite desechar todas las capturas realizadas, esto nos proporciona la posibilidad de poder descartar las capturas realizadas desde el comienzo de las detecciones sin necesidad de guardar esos datos en el fichero.

En la figura 3.1 se puede apreciar el diseño de interfaz de usuario de la aplicación de calibración y en las figura 3.2 y 3.3, el prototipo de la aplicación.

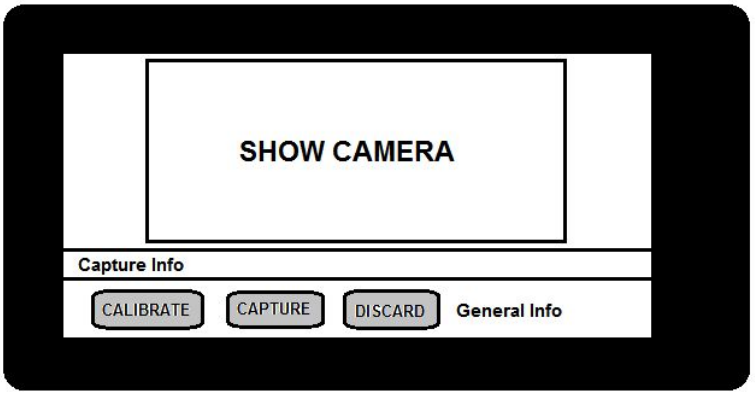

Figura 3.2. Diseño de interfaz de calibración.

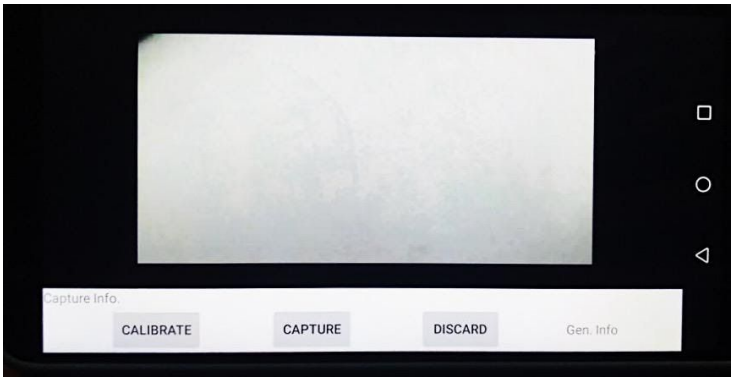

Figura 3.3. Resultado de la implementación del interfaz de usuario.

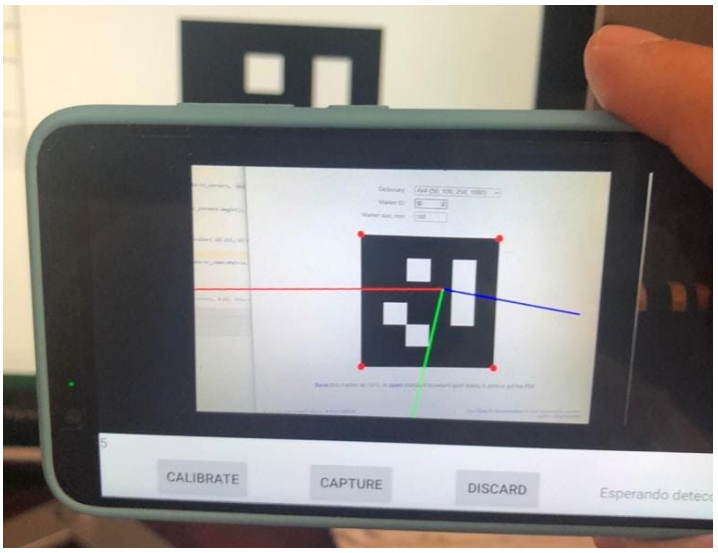

Figura 3.4. Aplicación de calibración en funcionamiento.

A continuación mostramos el proceso de datos de captura para la calibración desde la API de Aruco.

Método que hace de traductor entre Java y C++, realiza una llamada a la función calibrate() del objeto Aruco.

```
extern "C"
JNIEXPORT jboolean JNICALL
Java_com_example_appcalibracion3_MainActivity_calibrate(JNIEnv *env, jobject thiz) {
    // TODO: implement calibrate()
    bool result=aState->calibrate();
    return result;
}
```

Función que realiza la calibración, estos datos de calibración se guardarán en un fichero para que puedan ser utilizados por la aplicación que realizará las capturas. Esta función comprueba si se ha realizado alguna detección de algún marcador fiducial en el frame actual, en el caso que sí, se encarga de realizar las detecciones oportunas para obtener los valores que se usarán en la calibración, como el tamaño de la imagen, los coeficientes de distorsión, la matriz de la cámara, el ratio, etc.

```
bool ArucoState::calibrate(){
    bool result=false;
    if(c_allIds.size()<1)
        return false;
    if(calibrationFlags & CALIB_FIX_ASPECT_RATIO){
        c_cameraMatrix=Mat::eye(3,3,CV_64F);
        c_cameraMatrix.at<double>(0,0)=aspectRatio;
    }

    vector< vector<Point2f> > allCornersConcatenated;
    vector<int> allIdsConcatenated;
    vector<int> markerCounterPerFrame;
    markerCounterPerFrame.reserve(c_allCorners.size());
    for(unsigned int i=0;i<c_allCorners.size();i++){
        markerCounterPerFrame.push_back((int)c_allCorners[i].size());
        for(unsigned int j =0;j<c_allCorners[i].size();j++){
            allCornersConcatenated.push_back(c_allCorners[i][j]);
            allIdsConcatenated.push_back(c_allIds[i][j]);
        }
    }

    double repError;

    repError=aruco::calibrateCameraAruco(allCornersConcatenated,allIdsConcatenated,
        markerCounterPerFrame,board,imgSize,c_cameraMatrix,c_distCoeffs,c_rvecs,c_tvecs,calibrationFlags);

    bool saveOk = ArucoState::saveCameraParams(cameraFilePath,imgSize,aspectRatio,calibrationFlags,c_cameraMatrix,
            c_distCoeffs,repError);

    return saveOk;
}
```

3.1.2. Aplicación de Captura (AppCapture).

3.1.2.1. Funcionamiento general

Cuando se inicia la aplicación, se lee el fichero compartido por la aplicación de calibración y carga los parámetros en el sistema. A continuación carga la interfaz y permanece a la espera que se utilice la función de capturar, que será la encargada de ir realizando detecciones de los marcadores fiduciales desde la base de pruebas y enviarlos a la estación base.

La manera en la que realiza las detecciones viene determinada, al igual que en el caso de la calibración, por un sistema que determina que cada 10 detecciones visuales del marcador obtenga los datos de la pose. Esto se hace con el fin de no sobrecargar el sistema, por la gran cantidad de operaciones de cómputo que realiza en su funcionamiento.

Una vez se realiza la detección y captura del marcador fiducial, los datos obtenidos correspondientes al vector de traslación, el vector de rotación y el identificador del marcador, se envían al hilo principal de la aplicación. Esto se realiza mediante unas funciones que realizan las traducciones entre el código java, que maneja la

aplicación de android, y el código nativo, desarrollado en C++ y que contiene las funciones de OpenCV y Aruco.

Una vez tenemos la información en el hilo principal, se transforma para adecuarlo al sistema de referencia común que espera la estación base. Paralelamente a la captura y envío se realizan toma de medidas de tiempos, que se adjuntará a la información a enviar y que permitirá al sistema que procesa los datos agrupar los eventos pertenecientes a un determinado intervalo para poder compararlos con los obtenidos por el otro sensor.

Ahora, teniendo toda la información necesaria para ser enviada, se procede a realizar el envío mediante la conexión bluetooth establecida previamente con la estación base.

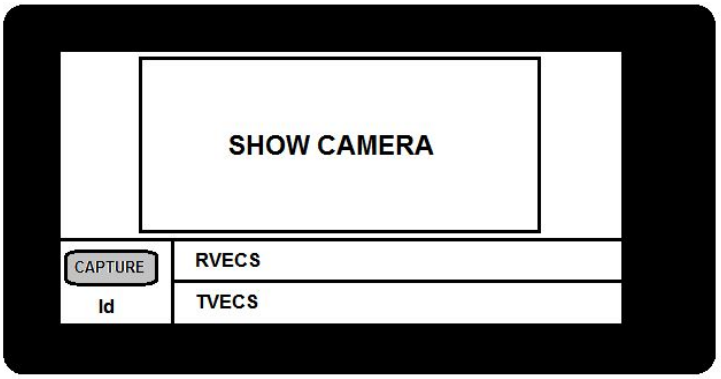

Figura 3.5 Diseño del interfaz de usuario de la aplicación de captura del prototipo

Figura 3.6. Resultado de la implementación del interfaz de usuario de la aplicación de captura

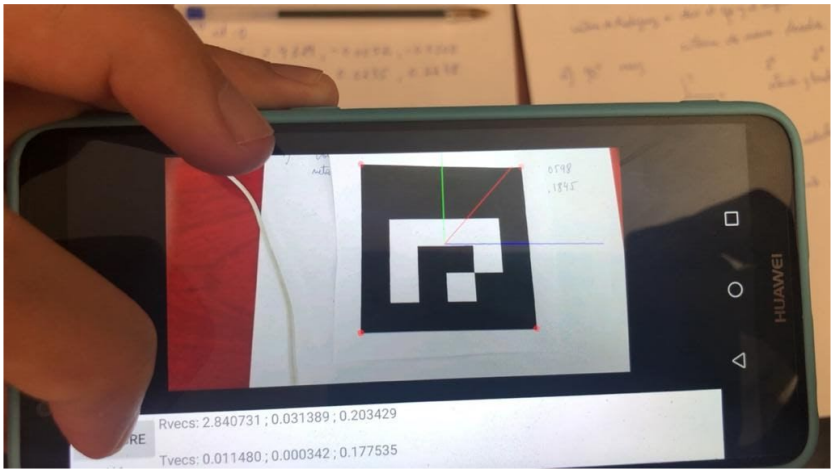

Figura 3.7 El prototipo de la aplicación de captura en funcionamiento

A continuación mostramos el proceso recogida de información de los datos de calibración y su posterior configuración en la aplicación.

La siguiente función es la encargada de configurar los parámetros de calibración que se encuentran en el fichero generado por la aplicación de calibración.

```
bool ArucoState::setCameraFromFileCamera(const char* df){
    FileStorage fs;

    //std::ifstream fs;
    try {
        fs.open(this->_cameraPath, FileStorage::READ);
        //fs.open(df,std::ifstream::in);
    }
    catch(cv::Exception e){
        //if(fs.fail())
        // return false;

        if(!fs.isOpened())
            return false;
    }

    fs["camera_matrix"] >> this->c_cameraMatrix;
    fs["distortion_coefficients"] >> this->c_distCoeffs;

    return true;
}
```

Mostramos como es el proceso desde la API de detección de un marcador fiducial y la obtención de los datos del mismo.

Primero se verifica si se han detectado marcadores fiduciales conocidos, en caso que si, se realiza una estimación de su pose, se generan unas variables this->tRvecs, this->tTvecs, this->Id, que serán las encargadas de traspasar esa información del código en C++ al código de Java. Para mostrar visualmente al usuario que se encuentre observando la pantalla del sensor de localización a bordo hacemos uso de la función drawAxis() que dibuja en la pantalla los datos recogidos, tal y cómo se muestra en el código siguiente.

```
bool ArucoState::detect(Mat &imGray, Mat &imRGBa){
   bool result=false;
   Mat imCopy;

   aruco::detectMarkers(imGray, this->dictionary,this->c_corners, this->c_ids, this->params, this->c_rejected);
   if(this->refindStrategy)
       aruco::refineDetectedMarkers(imGray, this->board, this->c_corners, this->c_ids, this->c_rejected);

    //image.copyTo(imageCopy);

   if(c_ids.size()>0) {
       imRGBa.copyTo(imCopy);

       vector<Vec3d> rvecs, tvecs; // Airán.

       aruco::estimatePoseSingleMarkers(this->c_corners, 0.1, this->c_cameraMatrix, this->c_distCoeffs, rvecs, tvecs);
       int j=0;

       this->tRvecs=rvecs;
       this->tTvecs=tvecs;
       this->Id=c_ids[0];

       for(vector< vector<Point2f> >::iterator it=c_corners.begin(); it!=c_corners.end(); ++it ){
           vector<Point2f> corners =*it;
           for (int i=0;i<4;i++){
               Point pt1 = corners[i];
               circle(imRGBa,pt1,10,Scalar(255,0,0),-1,8);
           }

           aruco::drawAxis(imRGBa, this->c_cameraMatrix, this->c_distCoeffs, rvecs[j], tvecs[j], 0.1); // Airán.
           j++;
       }

       // Airán
       /*vector<Vec3d> rvecs, tvecs;
       aruco::estimatePoseSingleMarkers(this->c_corners, 0.05, this->c_cameraMatrix, this->c_distCoeffs, rvecs, tvecs);
       for(int i=0; i<this->c_ids.size(); i++){
           aruco::drawAxis(imRGBa, this->c_cameraMatrix, this->c_distCoeffs, rvecs[i], tvecs[i], 0.1);
       }*/



       //aruco::drawDetectedMarkers(imCopy, c_corners, c_ids,Scalar(255,0,0));
       imgSize=imRGBa.size();
       //imRes=imageCopy.clone();
       result = true;
   }


   return result;
}
```

Mostramos la función que recoge el vector de traslación del marcador fiducial detectado desde la API de Aruco en C++ y lo envía al código de Java para que posteriormente pueda ser utilizado por sus funciones. Se recogen los datos de la matriz tTvecs[] y se parsean para ser enviados en un formato String tal y como requiere la implementación del código.

```
extern "C" JNIEXPORT jstring JNICALL
Java_com_example_appcapture_MainActivity_stringFromTvecs(
        JNIEnv* env,
        jobject /* this */) {
    char buf2[2048]; // need a buffer for that
    memset(buf2, 0, sizeof(buf2));
    double a = aState->tTvecs[0].val[0];
    double b = aState->tTvecs[0].val[1];
    double c = aState->tTvecs[0].val[2];
    sprintf(buf2,"%lf;%lf;%lf;",a,b,c);

    //for(int i=0;i<(aState->tRvecs).size();i++){

    const char* p = buf2;
    return env->NewStringUTF(p);
}
```

3.1.2.2. Transferencia de datos mediante Bluetooth

La transferencia de datos desde el sensor móvil hacia la estación base, se realiza mediante Bluetooth.

La estación base, implementa un servidor que se mantiene a la espera, hasta que algún dispositivo que previamente haya sido emparejado, se conecte y comience a enviar información.

La función que realiza todo el procedimiento de habilitación del servidor por parte de la estación base, se puede observar en el fichero server_mvl.py del código de la estación. En ella hace uso de la clase socket, para poder establecer las conexiones oportunas. Destacar, que se encuentran parametrizados los valores de la dirección MAC, el puerto por el que se comunican o el tamaño máximo de la información recibida.

En el sistema sensor de a bordo, se ha desarrollado una clase, que permite facilitar el proceso de las conexiones. Esa clase se llama BlueConnect y en ella, podemos crear una conexión utilizando los parámetros que previamente hayamos configurado en la estación base.

Utilizando la función conectar(), automáticamente se crea un socket de conexión con el servidor de la estación base y se establece un enlace para enviar los datos capturados.

Para el tratamiento de la conexión y envío de los datos por Bluetooth desde el sensor de localización a bordo a la estación base, se ha creado una Clase que en Java que realiza todo el procedimiento de manera más sencilla.

Atributos y constructor de la clase.

```
public class BlueConnect {

    // Atributos.
    private static String btserveraddress;
    private static UUID uuid;

    private BluetoothDevice device;
    private BluetoothAdapter btAdapter;
    private BluetoothSocket btSocket;

    private OutputStream outStream;
    private boolean conexion;

    // Constructor.
    public BlueConnect(String btserveraddress, String uuid){
        this.btserveraddress = btserveraddress;
        this.uuid = UUID.fromString(uuid);
        btAdapter = null;
        btSocket = null;
        OutputStream outStream = null;
        conexion = false;
    }
```

Métodos encargados de enviar la información, uno envía una cadena de texto ya procesada y la otra envía un dato Serializado (Para futuras actualizaciones del desarrollo).

```
// Método de envio texto.
public boolean writetostream(String mensaje){
    if(outStream != null && conexion == true) {
        try {
            byte[] msgBffr = mensaje.getBytes();
            outStream.write(msgBffr);
            Log.d("Bluetooth", "OK: mensaje enviado");
            return true;
        } catch (IOException e) {
            Log.d("Bluetooth", "Error: mensaje no enviado");
        }
    }
    return false;
}
// Método de envio texto.
public boolean writetostreambytes(byte[] byteSend){
    if(outStream != null && conexion == true) {
        try {
            byte[] msgBffr = byteSend;
            outStream.write(msgBffr);
            Log.d("Bluetooth", "OK: mensaje enviado");
            return true;
        } catch (IOException e) {
            Log.d("Bluetooth", "Error: mensaje no enviado");
        }
    }
    return false;
}
```

Método de conexión, en este método se realiza todo el procedimiento de conexión con la estación base, los parámetros utilizados se establecen desde la aplicación principal que hace uso de la clase a través de su constructor a la hora de crear el objeto.

```
// Métodos de conexión.
public boolean conectar(){
    btAdapter = BluetoothAdapter.getDefaultAdapter();
    device = btAdapter.getRemoteDevice(btserveraddress);
    Log.d("Bluetooth","Conectando...");

    if(createSocket()){
        if(createConnect()){
            if(createBridge()){
                if(outStream != null) {
                    conexion = true;
                    Log.d("Bluetooth","OK: Conexión establecida. Listo para enviar datos");
                    return true;
                }else{
                    Log.d("Bluetooth","Error: Conexión NO establecida. outStream es NULL");
                    return false;
                }
            }else{
                conexion = false;
                return false;
            }
        }else{
            conexion = false;
            return false;
        }
    }else{
        conexion = false;
        return false;
    }
}
private boolean createSocket(){
    try {
        btSocket = device.createRfcommSocketToServiceRecord(uuid);
        Log.d("Bluetooth","OK: Socket de conexión");
        btAdapter.cancelDiscovery();
        return true;
    } catch(IOException e) {
        Log.d("Bluetooth","Error: Socket de conexión");
    }
    btAdapter.cancelDiscovery();
    return false;
}
private boolean createConnect(){
    try{
        btSocket.connect();
        Log.d("Bluetooth","OK: Se pudo crear conexión btSocket");
        return true;
    } catch(IOException e){
        Log.d("Bluetooth","Error: No se pudo crear conexión btSocket");
    }
    return false;
}
private boolean createBridge(){
    try{
        outStream = btSocket.getOutputStream();
        Log.d("Bluetooth","OK: Se puedo crear un outputstream");
        return true;
    } catch(IOException e){
        Log.d("Bluetooth","Error: No se puedo crear un outputstream");
    }
    return false;
}
```

Método que cierra la conexión establecida.

```
// Método desconexión.
public boolean closeConnection(){
    try {
        outStream.close();
        btSocket.close();
        return true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```

Líneas futuras de desarrollo.

Como líneas futuras de desarrollo y actualización de la aplicación de captura de marcadores fiduciales para determinar la pose de un sistema móvil, se han previsto el desarrollo de una serie de clases.

- Una clase que permite generar un tipo determinado de objeto con la información a enviar.

- Una clase que serializa el objeto con el fin de hacer más eficiente su propagación por la red.

Estas clases dependen de la implementación en una línea futura de desarrollo, de la serialización en la recepción de la estación base y la traducción de objetos Java en Python.

3.2. Estación base con cámara (Raspberry Pi)

3.2.1. Funcionamiento general de la aplicación

La estación base con cámara desarrollada sobre una Raspberry Pi, contiene la aplicación principal del sistema. La aplicación principal está desarrollada en varios módulos que se reparten las siguientes funciones:

- Detección y localización de balizas visuales (patrón visual). Se trata de un sistema externo (static localization) que hace un seguimiento (tracking) de la base de pruebas que contiene las balizas formadas por tres pelotas de color (rojo, azul y verde) y el sensor móvil (ego-localization).

- Servidor de recepción de información del sensor móvil ubicado sobre la base de pruebas mediante un sistema de conexión bluetooth.

- Un programa principal que hace la recepción de los datos del sistema de ego-localización y del sistema de localización estática, realiza los cálculos de posicionamiento y orientación, visualiza los cálculos realizados mediante una interfaz gráfica y realiza una registro de los datos capturados en un sistema de almacenamiento.

- Un sistema de calibración de la cámara fija, que dándole una serie de capturas realizadas por la cámara nos devuelve los parámetros necesarios

para calibrar y evitar problemas de distorsiones, etc. Esos datos devueltos, debemos posteriormente agregarlos al fichero de *paremeters.py*

- Contiene un fichero llamado *paremeters.py* donde se pueden establecer todos los parámetros necesarios para el ajuste del sistema.

La aplicación se ejecuta mediante el fichero *menu.py*, en él se establecen un menú que permite una serie de opciones:

1. Data simulation visualization. (Realiza la simulación con los datos recogidos en tiempo real)

2. Display and save simulation data. (Igual que el modo anterior guardando los datos en un fichero llamado *salida-simulacion.txt*)

3. Camera calibration. (Realiza los cálculos para la calibración de la cámara utilizando los datos dados en los ficheros "*data.csv*" y "*datosxy.csv*" y muestra los valores en la terminal, para configurarlos, el usuario debe modificar esos valores en el fichero *parameters.py* y volver a arrancar el programa)

4. Show Parameters. (Muestra el fichero *parameters.py*)

5. Exit

3.2.2. Funcionamiento interno de la aplicación

En la figura 3.8 se muestra el esquema de funcionamiento de la aplicación ubicada en la estación base.

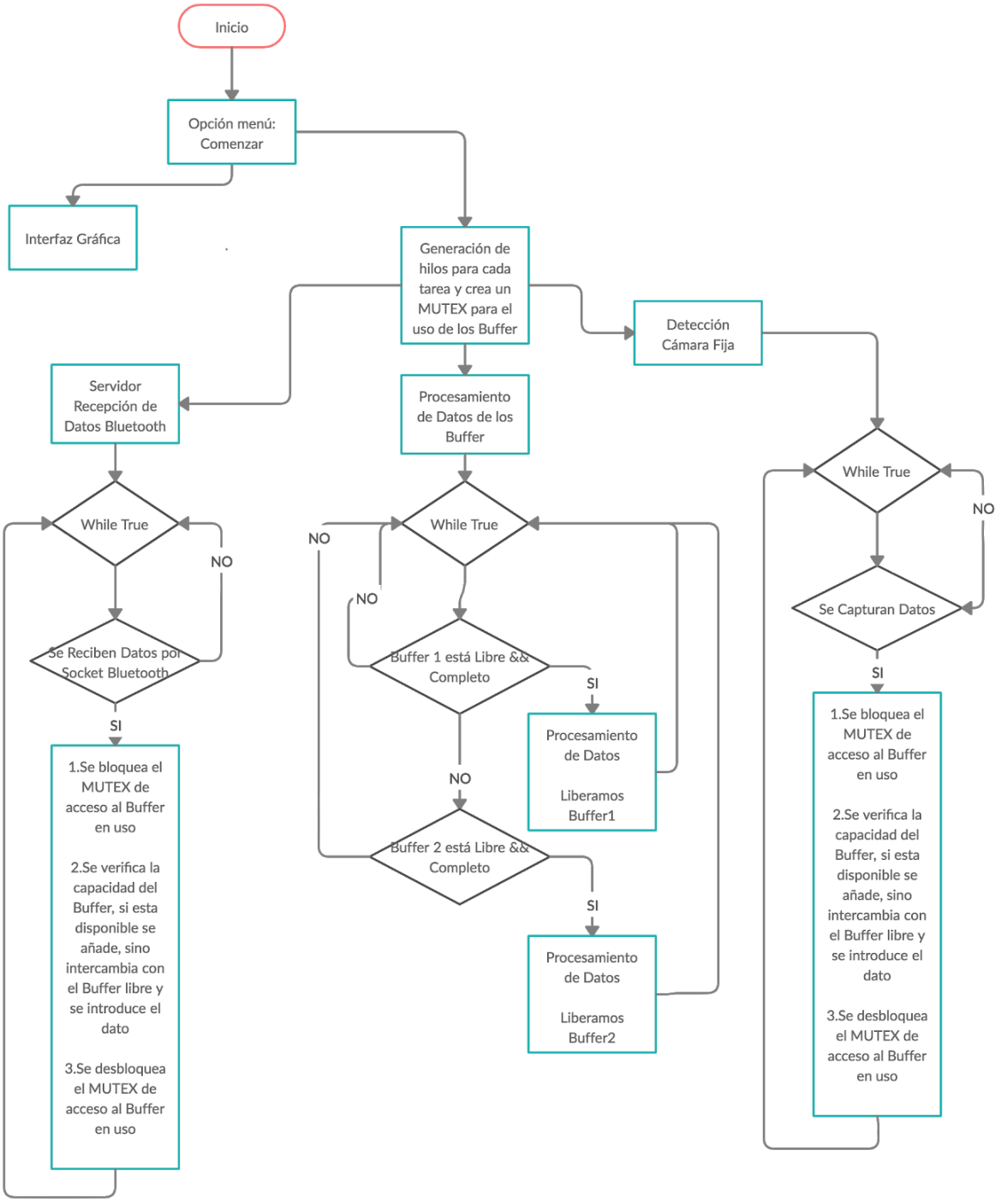

Figura 3.8. Esquema programa principal.

Cuando se indica que comience la captura y simulación de los datos, se activa el servidor de recepción bluetooth y el sistema de captura de visión artificial.

Cada vez que se recibe un dato, sea del sensor móvil o de la visión artificial por parte de la cámara, este se guarda como un objeto de datos dentro de unos buffers que se llenarán según van recibiendo hasta el tamaño previsto del buffer (este tamaño es un parámetro que puede ser modificado)

Cuando el buffer se completa, queda disponible para su procesamiento, no sin antes, habilitar otro buffer para que se sigan recibiendo los datos. El sistema dispone de dos buffers que se van intercambiando, mientras uno captura, el otro en caso que esté lleno, se procesa, cuando éste termina el procesado de datos, vuelve a quedar disponible para cuando se requiera.

El procesamiento de los datos realiza previamente un ajuste de tiempos, utilizando como referencia el sistema de tiempo de la Raspberry Pi. Esto es necesario para los datos recibidos desde el sistema sensorial de a bordo, ya que cada sistema utiliza un reloj de tiempo diferente y para poder realizar comparaciones de datos, se debe de estar en un sistema de tiempo común. Cada sistema puede tener una desviación de tiempo de unos milisegundos ya que utilizan cronómetros diferentes. Para poder realizar comparaciones, tienen que estar capturados con el mismo cronómetro, pero al no tener un sistema común de tiempos, la raspberry y el dispositivo móvil utilizan cronómetros diferentes, que incluso yendo al mismo ritmo, pueden marcar valores distintos. Para poder ajustar esta desviación, se busca un evento común que compartan ambos sistemas y que se asume que sean simultáneos o casi simultáneos, como es el caso del envío y recepción de los datos. La diferencia de tiempos entre la recepción y el envío del dato nos indica cuánto se ha de corregir un cronómetro respecto al otro.

Un intervalo de tiempo Δ se usa para determinar qué medidas pertenecen a la misma situación, actualmente de 5000ms ya que es un prototipo estático, y se ha estimado como un valor aceptable para hacer medidas de validación iniciales. En caso que el prototipo fuese dinámico, habría que bajar ese intervalo de tiempo para garantizar que el conjunto de eventos dentro del intervalo se correspondan a la misma situación.

Una vez agrupados, estos datos se procesan y se comparan con las funciones de conversiones según mostrados en el Capítulo 1.2 haciendo uso de la librería proporcionada por el docente.

Destacar que el sistema de detección de balizas tiene una serie de restricciones que podemos observar a continuación:

- Con 1 bola detectada, la máxima información que podemos obtener de ella es la posición de la bola.

- Con 2 bolas o más detectadas, sería suficiente para calcular la posición y la orientación.

Una vez procesado el buffer, los resultados devueltos serán mostrados en una interfaz gráfica con el fin de poder tener una referencia visual de cómo se comportan los datos capturados mediante la simulación y poder determinar posibles situaciones que puedan generar los sensores utilizados, al igual que podríamos estimar que tan buena es la precisión de un sistema en comparación con el otro, ya que se determina la diferencia entre ambos sistemas.

Se dispone en el menú de la aplicación una opción que permite guardar cada uno de los buffers capturados a un fichero para un análisis posterior.

Mostramos el código del proceso de extracción de información de los buffers (nombrado en las líneas anteriores) y de su envío para que puedan ser procesados por la función correspondiente. Se podrá observar que incluye un apartado que indica que si la variable save==True se proceda adicionalmente a guardar la información capturada en un fichero de salida, esto es una función de una de las opciones del menú, Figura 3.9.

```
def simulation_data():
    global save

    if save == True:
        archivo = open("salida-simulacion.txt", "w")

    while True:
        if buffers.buf_lib == 1 and len(buffers.l1) == 10:
            #print "Buffer 1 - "
            i = 0
            arreglo = np.zeros((10,17), dtype=float)
            while i < 10: # Recorremos el buffer.
                if save == True:
                    archivo.write(buffers.l1[i].imprimir())
                arreglo[i] = buffers.l1[i].arreglo()
                i+=1

            process_data.simulation(arreglo)
            buffers.l1 = []
            buffers.buf_lib = -1

        elif buffers.buf_lib == 2 and len(buffers.l2) == 10:
            #print "Buffer 2 - "
            i = 0
            arreglo = np.zeros((10,17), dtype=float)
            while i < 10: # Recorremos el buffer.
                if save == True:
                    archivo.write(buffers.l2[i].imprimir())
                arreglo[i] = buffers.l2[i].arreglo()
                i+=1

            process_data.simulation(arreglo)
            buffers.l2 = []
            buffers.buf_lib = -1

        if param.close:
            if save == True:
                archivo.close()
                save = False
            break
```

Figura 3.9. Código extracción de datos de los buffers.

3.2.3. Módulo de calibración

Para poder utilizar de manera adecuada los datos proporcionados por el sistema de visión artificial que hará uso de la cámara conectada, ésta debe estar calibrada, este paso, corrige la distorsión que pueda tener la lente. Esta calibración nos devuelve unas constantes que nos permiten mediante una fórmula, calcular la posición estimada de cada una de las balizas detectadas. En nuestro caso, determinaremos que se trata de una cámara que no tiene distorsión. Es uno de los problemas que destacamos de este sistema que utilizamos.

Tenemos dos sistemas de calibración. En el caso de la cámara externa se trató de utilizar un algoritmo de calibración diferente al ofrecido por OpenCV, adaptado a los métodos de localización utilizados. Se realizó así para probar la precisión de un planteamiento de la localización a partir de simples fórmulas trigonométricas y asumiendo un modelo simple de la cámara (pin-hole) sin considerar las distorsiones. Se mantuvo este planteamiento para el desplazamiento en profundidad, pero se tuvo que recurrir después a un ajuste polinómico para el desplazamiento horizontal.

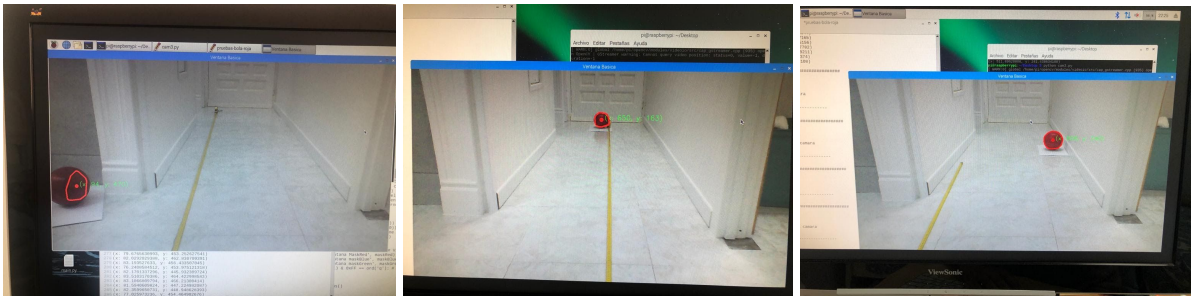

Figura 3.10. Mediciones realizadas para la calibración de la cámara.

El sistema propio de calibración de la cámara de visión artificial tendrá el siguiente funcionamiento.

Para calibrar el eje X, se utilizará un ajuste polinómico mediante un algoritmo que estimará los coeficientes de un polinomio de grado 3, a partir de un valor función de una serie de medidas en píxeles captadas previamente por la cámara y que posteriormente utilizaremos para el cálculo de la coordenada X geométrica.

La función que contiene el algoritmo se llama calibration_x() dentro del fichero *calibrate_camera.py.* Este algoritmo se alimenta de una cantidad grande de datos

que previamente se han capturado de una de las balizas haciendo uso de la cámara y que se guardaron en un fichero, estos datos contienen las coordenadas x e y medidas en el entorno con un metro y las coordenadas x e y devueltas por el sistema de visión.

Después de realizar los correspondientes ajustes del algoritmo, este devuelve una serie de parámetros que se corresponden con la configuración estimada para una correcta calibración y posteriores capturas.

A continuación mostramos una gráfica donde se puede apreciar la existencia de una cierta desviación de los datos recogidos respecto de los detectados por el sistema de visión artificial, es el problema principal de este tipo de visión artificial.

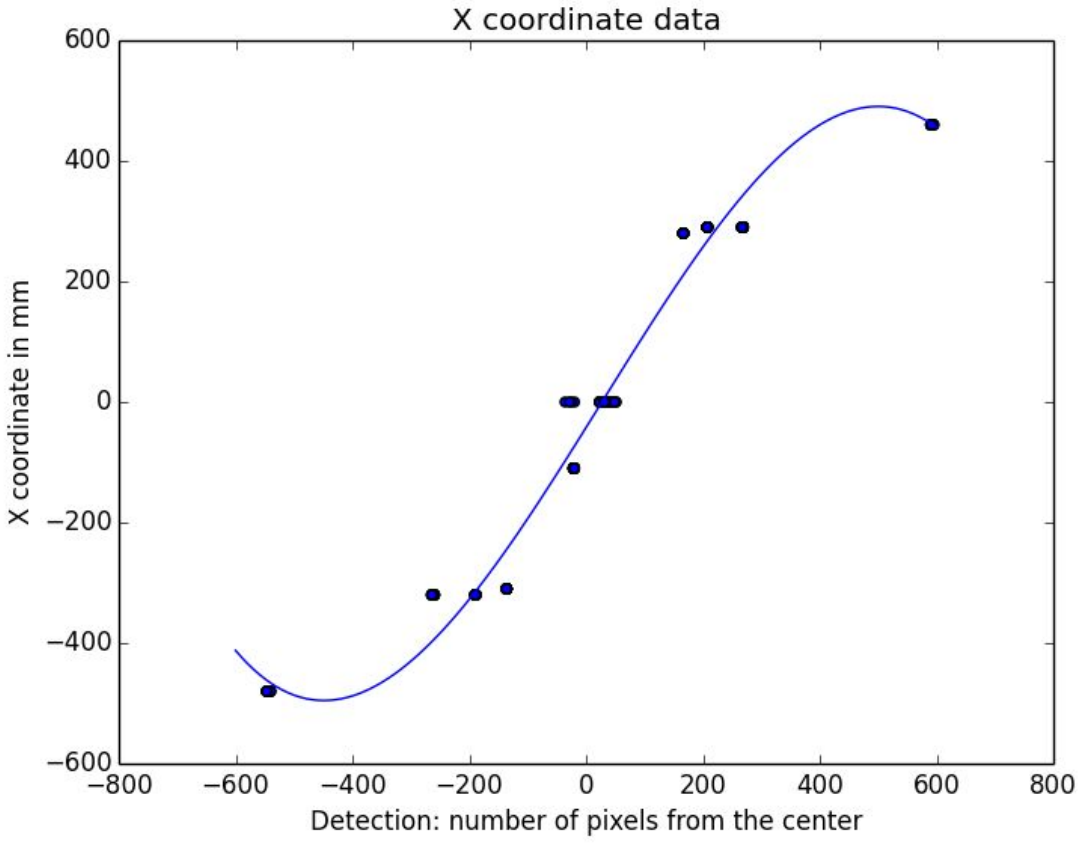

Figura 3.11. Datos calibración eje x.

Para calibrar el eje Z o de profundidad, se utilizará una aproximación mediante un algoritmo que estimará un valor en función de una serie de medidas en píxeles captadas previamente por la cámara y que posteriormente utilizaremos para calcular la profundidad geométrica.

La profundidad geométrica es calculada a partir del modelo descrito en el apartado de fundamentos matemáticos, y la calibración en este caso consiste en obtener la constante K3, que es la que da cuenta de las características intrínsecas de la cámara (focal y relación entre el número de píxel y su situación espacial en el detector de la cámara)

La función que contiene el algoritmo se llama calibration_y() dentro del fichero *calibrate_camera.py.* Este algoritmo se alimenta de una cantidad grande de datos que previamente se han capturado de una de las balizas haciendo uso de la cámara y que se guardaron en un fichero, estos datos contienen las coordenadas y medidas en el entorno con un metro y las coordenadas y devueltas por el sistema de visión.

El algoritmo aplica los datos capturados (píxel, profundidad desde la base de la cámara) para obtener estimaciones de la constante k3 que se puede despejar de la fórmula.

A continuación mostramos una gráfica donde se puede apreciar cómo a medida que las distancias crecen el error tiende a ser mayor, en cambio, con medidas recogidas en corto alcance menor de 2000mm la detección es bastante buena, es el problema principal de este tipo de visión artificial.

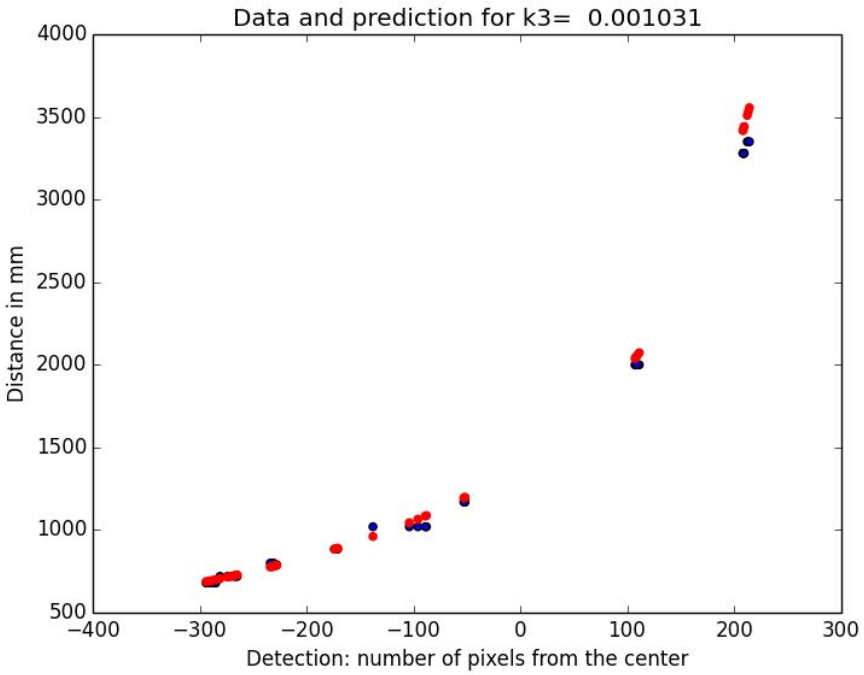

Figura 3.12. Datos calibración eje z.

3.2.4. Módulo de visión artificial

El fichero donde se desarrolla el sistema de visión artificial de la estación base es el denominado *server_camera.py* en el hacemos uso de la librería de OpenCV para realizar el tracking de la base de pruebas utilizando el sistema de balizas.

Este sistema de visión artificial, como se ha comentado ya en capítulos anteriores, es poco preciso debido a la necesidad de un sistema de iluminación muy concreto, se está utilizando para un entorno de pruebas cerrado y con una iluminación controlada. Por ello se utilizará en sustitución del sistema de colores típico BGR de OpenCV el sistema basado en HSV.

En OpenCV sus componentes pueden tomar los siguientes valores:

H: 0 - 179
S: 0 - 255
V: 0 - 255

Para la detección de las balizas nos basaremos en detectar los colores de las bolas (rojo, azul y verde), se ha establecido un margen bastante amplio que garantiza dentro del entorno de pruebas poder detectar con facilidad las balizas.

el espectro de colores que nos interesan captar serían:

Para el rojo: Este color tiene una particularidad ya que se encuentra en dos posiciones diferentes del espectro, al principio y al final, para poder definirlo de manera correcta debemos generar dos componentes y unirlas con el fin de captar el objeto correctamente.

- Para el valor rojo de la parte más a la izquierda más oscuro [1,150,10]
- Para el valor rojo de la parte más a la izquierda más claro [8,255,255]
- Para el valor rojo de la parte más a la derecha más oscuro [175,150,10]
- Para el valor rojo de la parte más a la derecha más claro [179,255,255]

Para el verde: Usaremos los siguientes datos.

- Para el color verde más oscuro [100,100,20]
- Para el color verde más claro [125,255,255]

Para el azul: Usaremos los siguientes datos.

- Para el azul verde más oscuro [100,100,20]
- Para el azul verde más claro [125,255,255]

Para realizar una mejor detección se ha hecho uso de transformaciones morfológicas de las imágenes con el fin de poder eliminar todo tipo de ruido que pueda generarse y hacer una detección más eficaz, ya que se intenta capturar tres colores a la vez.

Se utilizó una dilatación de la captura en todos los colores (esta operación permite ampliar el área del objeto capturado), pero sólo hizo falta eliminar el ruido exterior del color verde (esto se realiza con la morfología de opening, elimina todas las coincidencias del color dispersas por la imagen), ya que es un color del que se componen muchos otros y tiene tendencia a generar más ruido.

Figura 3.13. Dilation [8]

Figura 3.14. Opening [8]

Luego hacemos uso de una función que dándole las máscaras de color detectado, realiza una operación de busqueda de un área superior a una cantidad determinada

que se puede ajustar, dibuja un contorno circular (por tratarse de balizas de bolas) del mismo color del que se encuentran pintadas.

Se determina las coordenadas x e y de la matriz de píxeles capturados y devuelve ese conjunto de valores al programa principal que realiza el procesamiento de los datos.

Mostramos a continuación el código que realiza estas detecciones que determina las coordenadas x e y de los datos capturados Figura 3.15.

```
def dibujar(mask, color, font, frame):
        contornos,_=cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        xx=-1
        yy=-1
        for c in contornos:
                area = cv2.contourArea(c)
                if area > 1500:
                        M = cv2.moments(c)
                        if M['m00'] == 0 : M['m00'] = 1
                        x = int(M['m10']/M['m00'])
                        y = int(M['m01']/M['m00'])
                        xx = float(M['m10']/M['m00'])
                        yy = float(M['m01']/M['m00'])
                        nuevoContorno = cv2.convexHull(c)
                        cv2.circle(frame, (x,y), 7, color, -1)
                        cv2.putText(frame, '(x: ' + str(x) + ', y: ' + str(y) + ')', (x+10,y), font, 0.75, (0,255,0), 1, cv2.LINE_AA)
                        cv2.drawContours(frame, [nuevoContorno], 0, color, 3)
                        return xx, yy
        return xx, yy
```

Figura 3.15. Código para extraer coordenadas x e y..

Podemos ver de manera visual cómo es la detección de las balizas de la plataforma móvil y sus coordenadas.

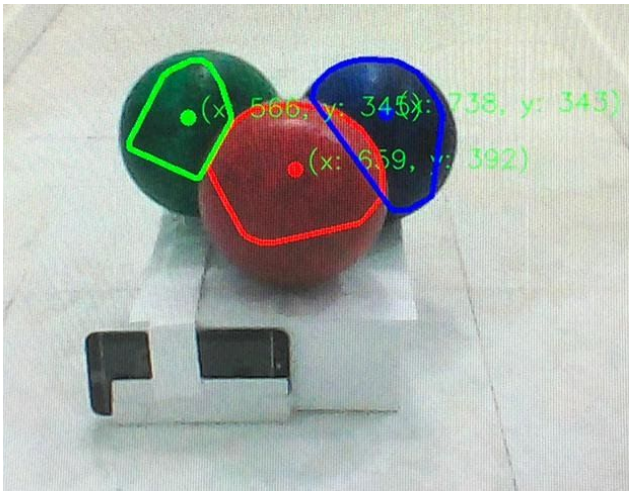

Figura 3.16. Detección de balizas.

3.2.5. Interfaz de usuario

Para el diseño de la interfaz, se ha hecho uso de de una librería llamada Tkinter que proporciona la capacidad de diseñar interfaces de usuario amigables y funciones de interacción con la interfaz.

Se creó una interfaz que utiliza unas variables compartidas con el sistema que realiza los cálculos matemáticos correspondientes a determinar la pose y la orientación tanto de la base de pruebas cómo de la cámara de visión artificial, que dado un cierto tiempo configurable actualiza los valores de la interfaz gráfica con los valores generados por el sistema de procesamiento de datos.

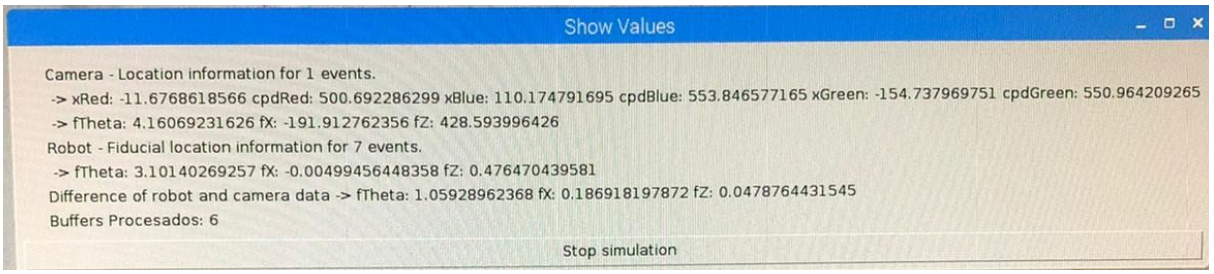

Figura 3.17 Interfaz gráfica.

Capítulo 4.- Experimentación

Se han podido realizar pruebas de validación. Los datos que hemos utilizado en la comparativa son los que obtenemos tras el procesamiento de los buffer.

Para el desarrollo de la parte de experimentación se ha tenido que fabricar una plataforma móvil para la sujeción de las balizas visuales de colores y un soporte para el sensor de localización que se ubicará sobre él. Los materiales utilizados han sido cartones y folios reciclados, unidos por cinta de precintar transparente. La unión de las balizas visuales se ha realizado con unos palos de madera. Las balizas visuales se han fabricado con pelotas de corcho blancas y han sido pintadas con pinturas acrílicas sobre una base negra para matizar el color (roja, azul y verde).

Figura 4.1. Plataforma móvil.

Parte experimental.

Para la parte experimental se ha realizado un montaje haciendo uso de las partes que intervienen en el desarrollo. A continuación se mostrarán unas imágenes del conjunto donde podemos ver la plataforma móvil con las balizas de colores y su sensor de localización a bordo y en frente la posición del marcador fiducial y la ubicación de la cámara fija.

El montaje tiene unos parámetros que serán de utilidad en el desarrollo de los cálculos para determinar la detección y posición de la plataforma móvil, estos

parámetros son el: ángulo de la cámara fija apuntando hacia el suelo denominado “alfa” y la altura de la cámara fija respecto de la base denominado “hc”

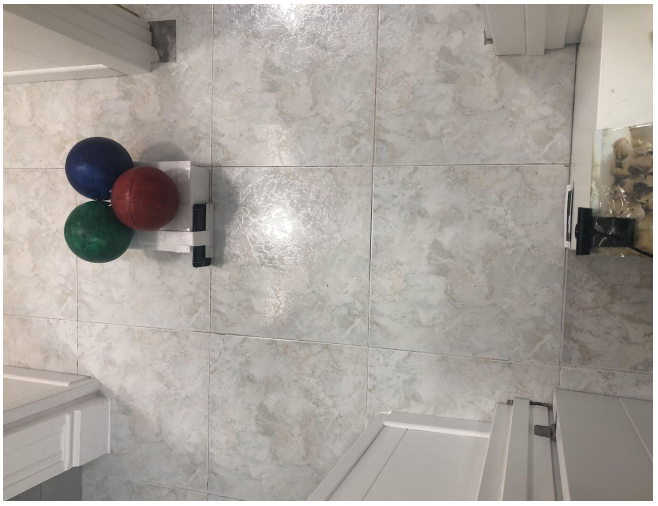

Figura 4.2. Vista 1 del montaje en el entorno.

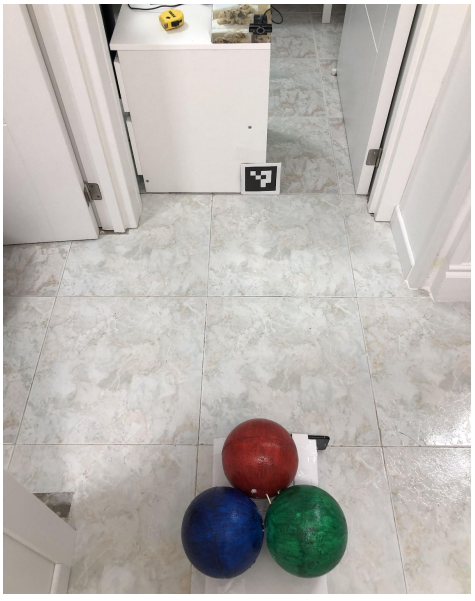

Figura 4.3. Vista 2 del montaje en el entorno.

Leyenda: Los datos de coordenadas se encuentran medidos en milímetros y los ángulos en radianes.

FxC = Coordenada Geométrica X de la cámara fija.
FxM = Coordenada Geométrica X del sensor de abordo.
FzC = Coordenada Geométrica Z de la cámara fija.
FzM = Coordenada Geométrica Z del sensor de abordo.
FThetaC = Ángulo de dirección obtenido de la cámara fija.
FThetaM = Ángulo de dirección obtenido del sensor de abordo.

Prueba 1: Correspondiente a la ubicación de la plataforma móvil ubicada frente a la cámara fija, en la zona central.

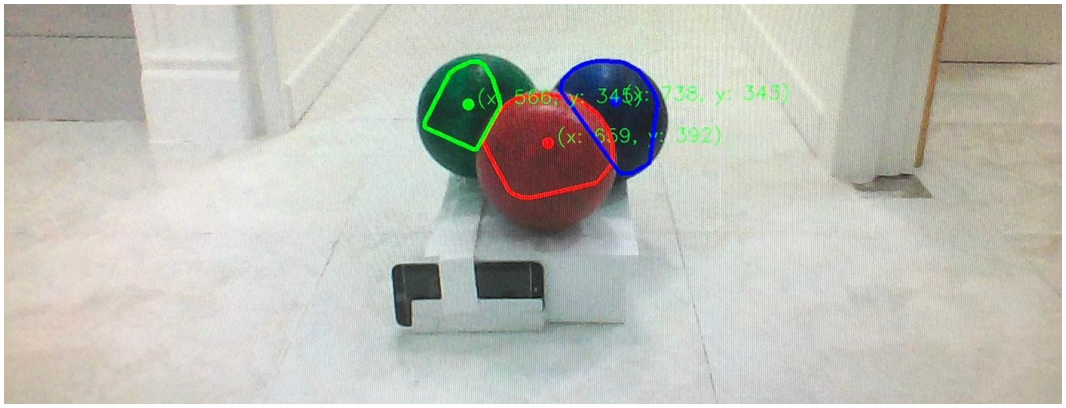

Figura 4.4. Imagen correspondiente a la prueba 1.

Prueba 1	FxC	FxM	FzC	FzM	FThetaC	FThetaM
Buffer 1	-2.	-4.	4137	476	2	3
Buffer 2	-17	-14	429	476	3	3
Buffer 3	-11	-21	449	476	3	3

Tabla 4.1. Datos de la prueba 1.

Conclusiones:

Podemos observar que los datos obtenidos son coherentes, se aproximan bastante, vemos que en este tipo de situaciones afecta en mayor medida al sistema de visión artificial de la cámara fija, la detección de la posición en esas distancias tiende a ser más eficaz el sensor basado en marcadores fiduciales.

Prueba 2: Correspondiente la ubicación de la plataforma móvil ubicada a la derecha de la cámara fija.

Figura 4.5. Imagen correspondiente a la prueba 2.

Prueba 2	FxC	FxM	FzC	FzM	FThetaC	FThetaM
Buffer 1	504	425	453	472	2	3
Buffer 2	514	425	458	473	2	3
Buffer 3	498	425	446	473	2	3

Tabla 4.2. Datos de la prueba 2.

Conclusiones:

Podemos observar que los datos obtenidos son coherentes pero se visualiza que la distorsión es mayor, hay más diferencia con respecto al sensor basado en

marcadores, se visualiza cómo a medida que se acerca a los extremos de la lente la detección empeora, la regresión en los extremos apenas existe. Puede deberse también a que los parámetros de calibración no son tan exactos. Aún así, en distancias cortas y en según que casos sigue teniendo unos valores aceptables. En este tipo de situaciones, el sensor basado en marcadores fiduciales no tiene problemas alguno y sus datos se pueden considerar como referencia para ver en cuánto se desvía la otra medida.

Prueba 3: Correspondiente la ubicación de la plataforma móvil ubicada a la izquierda de la cámara fija.

Figura 4.6. Imagen correspondiente a la prueba 3.

Prueba 3	FxC	FxM	FzC	FzM	FThetaC	FThetaM
Buffer 1	-490	-401	448	474	2	3
Buffer 2	-492	-401	446	474	2	3
Buffer 3	-531	-402	461	474	3	3

Tabla 4.3. Datos de la prueba 3.

Conclusiones:

Podemos observar que los datos obtenidos son prácticamente idénticos a los de la prueba 2, incluso podemos observar una desviación mayor en los ejes FxC y FxM, el cálculo de la profundidad es aceptable, tiene una desviación pero dentro de la media de las demás medidas.

Prueba 4: Correspondiente la ubicación de la plataforma móvil ubicada a la derecha de la cámara fija.

Figura 4.7. Imagen correspondiente a la prueba 4.

Prueba 4	FxC	FxM	FzC	FzM	FThetaC	FThetaM
Buffer 1	512	-370	432	493	2	3
Buffer 2	464	-369	428	493	3	3
Buffer 3	482	-372	425	491	3	3

Tabla 4.4. Datos de la prueba 4.

Conclusiones:

En este caso, podemos ver que la plataforma móvil tiene ahora una inclinación aproximada de unos 45º respecto a una línea paralela a la cámara fija. Podemos observar que los datos obtenidos tienen una diferencia notable en el valor de FxM respecto de FxC, indicando no sólo valores que difieren mucho sino además en una posición casi simétrica en las dos medidas. En este caso, podemos observar un fenómeno bastante inusual, el sensor basado en marcadores visuales está fallando.

Es un fallo debido a un problema típico en el uso de un sólo marcador fiducial para realizar la detección de la pose, es un problema que se advierte en la documentación de Aruco [2]. Cuándo se realiza la captura del marcador fiducial, se generan dos proyecciones del marcador fiducial en la cámara y que son simétricas, esto provoca que no se pueda conocer en primera instancia cuál es la correcta. Este tipo de fallos o ambigüedades, se solucionan haciendo uso de varios marcadores, ya que al tener dos detecciones una corrige la proyección errónea de la otra. En este tipo de situaciones y teniendo un margen de error mayor por el sistema que aplica, tiende a ser más preciso el sistema de visión artificial de la cámara fija.

Prueba 5: Correspondiente la ubicación de la plataforma móvil ubicada a la izquierda de la cámara fija.

Figura 4.8.Imagen correspondiente a la prueba 5.

Prueba 5	FxC	FxM	FzC	FzM	FThetaC	FThetaM
Buffer 1	-451	338	441	480	2	3
Buffer 2	-426	336	441	482	2	3
Buffer 3	-478	332	420	489	3	3

Tabla 4.5. Datos de la prueba 5.

Conclusiones:

Para esta prueba nos encontramos un caso idéntico que la prueba 4, un caso de fallo por ambigüedad por parte del sensor que se basa en marcadores fiduciales. Conociendo que disponemos de otro sensor de la cámara fija, se podría estudiar la posibilidad de realizar redundancia de los sensores, contemplando los casos particulares, se podría obtener lo mejor de cada sistema y que uno de soporte al otro cuando se encuentra en una situación en la que tienda a fallar. Ya que si se conoce a priori que en esa situación un sensor falla, se puede aplicar una corrección que lo mitigue.

Conclusiones

En primer lugar, destacar que se han cubierto los objetivos del proyecto pero ha habido más dificultades de las esperadas en este tipo de proyecto debido a la diversidad de la tecnología que se ha utilizado, ya que requiere un mayor esfuerzo de aprendizaje y es necesario establecer mecanismos para que las diferentes partes del proyecto funcionen conjuntamente, como por ejemplo métodos de comunicación y estructuras que agreguen la información como el sistema de buffers
conlleva una complejidad mayor de la esperada.

En general, el uso y experiencia de las herramientas utilizadas en el desarrollo del proyecto han sido bastante buenas. Por destacar alguna parte negativa, podría señalar la configuración del entorno completo de desarrollo para adaptarlo a las características técnicas de los elementos que íbamos a usar. Además, he podido comprobar las posibilidades de uso que nos dá el desarrollo de aplicaciones nativas en Android, ya que nos permite obtener toda la potencia que nos proporciona un terminal móvil.

En segundo lugar, durante el desarrollo del proyecto hemos visto que los sistemas basados en visión artificial, especialmente aquellos que tratan de detectar elementos, dependen mucho de las condiciones del entorno. ¿Puede ser que esto haga necesario la utilización de sistemas redundantes que cubran los diferentes escenarios de funcionamiento?

En referencia a la capacidad de los sensores se ha comprobado que, efectivamente, en espacios cerrados y controlados pueden servir para localizar la plataforma móvil.

Finalmente en relación a posibles líneas futuras de trabajo, podría considerarse incluir algoritmos que combinan información de ambos sensores, técnicas más avanzadas de detección de las balizas visuales, así como técnicas más avanzadas de marcadores fiduciales.

Este tipo de sistemas, en la práctica, pueden ser utilizados en el marco de la logística (Amazon), control de sistemas de fabricación, sillas de ruedas robotizadas, etc.

Summary and Conclusions

Firstly, it should be noted that the project objectives have been covered but there have been more difficulties than expected in this type of project due to the diversity of the technology used, as it requires a greater learning effort and it is necessary to establish mechanisms for the different parts of the project to work together, such as communication methods and structures that aggregate the information such as the buffer system involves a greater complexity than expected.

In general, the use and experience of the tools used in the development of the project have been quite good. To highlight some negative aspects, I could point out the configuration of the whole development environment to adapt it to the technical characteristics of the elements we were going to use. In addition, I have been able to check the possibilities of use that gives us the development of native applications in Android, since it allows us to obtain all the power that a mobile terminal provides.

Secondly, during the development of the project we have seen that systems based on artificial vision, especially those that try to detect elements, depend a lot on the conditions of the environment. Could it be that this makes it necessary to use redundant systems that cover the different operating scenarios?

With regard to the capacity of the sensors, it has been proven that, in closed and controlled spaces, they can indeed be used to locate the mobile platform.

Finally, in relation to possible future lines of work, consideration could be given to including algorithms that combine information from both sensors, more advanced techniques for detecting visual beacons, as well as more advanced fiducial marker techniques.

This type of system, in practice, can be used in the framework of logistics (Amazon), control of manufacturing systems, robotic wheelchairs, etc.

Presupuesto

El cálculo del presupuesto se ha basado en el cronograma de trabajo de este proyecto y en una estimación de las horas de trabajo y los costes correspondientes para su desarrollo.

Tipo de actividad	Coste/Hora	Horas	Coste
Análisis y Diseño	5€	25h	125€
Desarrollo	10€	120h	1.200€
Placa Raspberry Pi 3B			39,90€
Tarjeta microSD			
Dispositivo Móvil			80€
Webcam			50€
3 Balizas			9€
Pintura			4€
Total			1.514,90€

Presupuesto del proyecto.

Bibliografía

[1] Aruco. Sitio web oficial de Aruco:
https://www.uco.es/investiga/grupos/ava/node/26

[2] Documentación oficial de Aruco. Recurso web:
https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit

[3] Youssef N. Naggar, Ayman H. Kassem, Mohamed S. Bayoumi, "A Low Cost Indoor Positioning System Using Computer Vision", International Journal of Image, Graphics and Signal Processing(IJIGSP), Vol.11, No.4, pp. 8-25, 2019.DOI: 10.5815/ijigsp.2019.04.02

[4] Jae Hong Shim, Young Im Cho. "A Mobile Robot Localization via Indoor Fixed Remote Surveillance Cameras". Sensors, 2016

[5] Emin Kusku, Aaron Rabaabah. "Mobile Robot Location Via Efficient Calibration Technique of a Fixed Remote Camera". International Journal of Science and Informatics, 2012.

[6] Andrej Babinec et al. "Visual Localization of Mobile Robot Using Artificial Markers". Procedia Engineering, 2014.

[7] Tipos de distorsión. Imagen de un pdf:
https://www.fing.edu.uy/if/cursos/intr_optica/Material/aberraciones.pdf

[8] Transformaciones morfológicas. Imagen web:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[9] Android Studio. Wikipedia:
https://es.wikipedia.org/wiki/Android_Studio

[10] Developer Android Studio. Recurso web:
https://developer.android.com/studio

[11] Raspberry Pi. Recurso web:
https://www.raspberrypi.org/

[12] Artículo sobre Bluetooth. Wikipedia:
https://es.wikipedia.org/wiki/Protocolos_Bluetooth#RFCOMM

[13] OpenCV OCR. Recurso web:
http://opencvpython.blogspot.com.es/2012/04/simple-digit-recognition-ocr-in-opencv.html

[14] Git. Recurso web:
https://git-scm.com/book/es/v2

[15] OpenCV. Documentación web:
https://docs.opencv.org/4.4.0/

[16] Desarrollo nativo. Wikipedia:
https://es.wikipedia.org/wiki/Desarrollo_de_programas_para_Android#Native_development_kit_-_Sistema_de_desarrollo_nativo

[17] Raspbian. Recurso web:
https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

[18] Python. Documentación web:
https://wiki.python.org/moin/BeginnersGuide

[19] Julio Molleda Meré. “Técnicas de visión por computador para la reconstrucción en tiempo real de la forma 3D de productos laminados”. Tesis doctoral. Universidad de Oviedo. 2008

[20] Delia Johana Muñoz. “Corrección de la distorsión de imágenes en equipos ópticos por medio de splines y técnicas de análisis de imágenes”. Scientia et technica. 2007