



UNIVERSIDAD DE JAÉN
*Escuela Politécnica
Superior de Jaén*

Trabajo Fin de Grado

SISTEMA DE CONTROL REMOTO Y NAVEGACIÓN AUTÓNOMA GUIADA POR VISIÓN PARA UN ROBOT HEXÁPODO

Alumno: Miguel Ángel Anguita Molina

Tutores: Dr. D. Javier Medina Quero
Dra. D^a Silvia M^a Satorres Martínez

Dpto: Informática

Septiembre, 2021



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Dr. D. Javier Medina Quero Y Dra. D^a Silvia M^a Satorres Martínez, tutores del Trabajo Fin de Grado titulado: Sistema de control remoto y navegación autónoma guiada por visión para un robot hexápodo, que presenta Miguel Ángel Anguita Molina, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2021

El alumno:

ANGUITA
MOLINA
MIGUEL
ANGEL -
77379789S
Firmado digitalmente por
ANGUITA
MOLINA MIGUEL
ANGEL -
77379789S
Fecha: 2021.09.13
05:10:01 +02'00'

Miguel Ángel Anguita Molina

Los tutores:

SATORRES
MARTINEZ
SILVIA -
26741651
B
Firmado digitalmente por
SATORRES MARTINEZ
SILVIA - 26741651B
Nombre de reconocimiento
(DN): cn=ES,
serialNumber=IDCES-26741
651B, givenName=SILVIA,
sn=SATORRES MARTINEZ,
cn=SATORRES MARTINEZ
SILVIA - 26741651B
Fecha: 2021.09.13 09:28:19
+02'00'

MEDINA
QUERO
JAVIER -
75131484Y
Firmado digitalmente por
MEDINA QUERO
JAVIER -
75131484Y
Fecha: 2021.09.13
11:36:02 +02'00'

Javier Medina Quero

Silvia M^a Satorres Martínez

“Part of the journey is the end”
-Tony Stark

A mis padres, gracias a ellos soy quien soy y sólo puedo expresar mi sincero agradecimiento por apoyarme siempre.

A mis abuelos, por darme su incondicional cariño, y a pesar de que a algunos no los tengo a mi lado, sé que siempre me están cuidando.

A mis amigos, tanto a los que conocía antes de entrar en la carrera, como a los que he conocido durante la misma, habéis sido y seréis mi fuente de inspiración para seguir superándome día a día.

Y por último, pero no menos importante, a mis tutores, Javi y Silvia, habéis sido mi guía y mi estímulo durante este TFG.

Gracias a todos vosotros, habéis conseguido que mi sueño se cumpla.

ÍNDICE

1. Introducción	1
1.1 Motivación	1
1.2 Propuesta	2
1.3 Objetivos	3
1.4 Planificación temporal	4
1.5 Estructura de la memoria	6
2. Control visual en robots	8
2.1 Visión por computador	8
2.1.1 Fases en un sistema de visión artificial	8
2.2 Control visual	31
2.2.1 Arquitectura del sistema de visión	33
2.2.2 Clasificación de los sistemas de control visual	38
2.3 Robots Hexápodos	43
2.4 Navegación en robots móviles	47
2.4.1 Navegación reactiva	47
2.4.2 Navegación autónoma basada en visión	50
2.5 Actualidad	52
3. Tecnologías software que se van a desarrollar	54
3.1 Control remoto mediante servicio API/REST	54
3.2 Marcadores fiduciaros	57
3.3 Navegación autónoma con A*	62
4. Hardware	66
4.1 Robot hexápodo	66
4.2 Incorporación de los nuevos elementos	73
5. Proceso ingeniería de software	81
5.1 Fases de la ingeniería del software	81
5.2 Especificación de requisitos	82
5.2.1 Requerimientos del sistema de control remoto mediante servicio API/REST	82
5.2.2 Requerimientos del sistema de reconocimiento de marcas ArUco	84
5.2.3 Requerimientos del sistema de navegación autónoma	85
5.2.4 Requerimientos de la interfaz de usuario	85
5.3 Análisis del sistema	86
5.4 Diseño del sistema	92
5.4.1 Formato de intercambio de información	92
5.4.2 Protocolo de comunicación entre componentes	94
5.5 Implementación del sistema	98
5.5.1 Preparar script Arduino para recibir información	98
5.5.2 Implementación del servicio API/REST	101
5.5.3 Implementación de reconocimiento de marcas ArUco	102

5.5.4 Implementación de navegación autónoma	106
5.5.5 Implementación de interfaz	110
5.6 Pruebas	118
5.6 Resultados	121
6. Conclusiones y líneas de trabajo futuras	122
6.1 Conclusiones	122
6.1.1 Robot Hexápodo	122
6.1.2 Servicio API/REST	122
6.1.3 Marcas ArUco	123
6.1.4 Interfaz gráfica	123
6.1.5 Conclusión final	123
6.2 Líneas futuras	124
6.2.1 Mover robot con sensor de ondas cerebrales	124
6.2.2 Realidad virtual con unity	124
6.2.3 Plataforma con cámara en perpendicular a las marcas	124
Anexos	126
Anexo 1: Configuración de RaspBerry	126
1.1 Instalación y configuración de RaspBian	126
1.2 Instalación de la cámara	129
1.3 Instalación de OpenCV	130
1.4 Instalación de librerías extras	134
Anexo 2: Manual de usuario	135
Anexo 3: ¿Por qué se calibra la cámara?	136
Bibliografía	138

ÍNDICE DE FIGURAS

Figura 1: Planificación para los meses de Enero, Febrero y Marzo.	5
Figura 2: Planificación para los meses de Abril, Mayo, Junio y Julio.	5
Figura 3: Planificación para los meses de Julio, Agosto y Septiembre.	5
Figura 4: Fases de un sistema de visión por computador	9
Figura 5: Niveles de cuantización	11
Figura 6: Operaciones aritmético lógicas	13
Figura 7: Operaciones geométricas en un imagen	14
Figura 8: Imagen original (arriba) frente a imagen con el histograma ecualizado (abajo)	15
Figura 9: Imagen original (arriba) frente a imágenes con ajustes de contraste	16
Figura 10: Secuencia de transformada en el filtrado de una imagen en el dominio de frecuencias	16
Figura 11: Tipos de filtros en el dominio de frecuencia	17
Figura 12: Imagen original frente a imagen con un filtro suavizado	18
Figura 13: Ejemplo de matrices usadas para los filtros	19
Figura 14: Imagen original (izquierda) frente a imagen tratada con el filtro de Canny (centro) y tratada con el filtro de Sobel (derecha).	20
Figura 15: Operaciones morfológicas que se le pueden realizar a una imagen	21
Figura 16: Umbralización de Otsu	23
Figura 17: Componentes principales de un sistema de control visual	34
Figura 18: Arquitecturas Eye-in-hand(izquierda) y Eye-to-hand (derecha)	35
Figura 19: Ejemplo de arquitectura Eye-to-hand	37
Figura 20: Ejemplo de arquitectura Eye-in-hand	37
Figura 21: “Ver y mover” estático.	38
Figura 22: “Ver y mover” dinámico	39
Figura 23: Control visual directo	40
Figura 24: “Ver y mover” dinámico basado en posición	41
Figura 25: Control visual directo basado en posición	42
Figura 26: Demostración de movimiento entre robot con patas y andante.	44
Figura 27: Distintos modelos de robots para hobbyistas	45

Figura 28: Robot Hexy desde el frente	46
Figura 29: Robot Vorpal desde el frente	46
Figura 30: Imagen de las tortugas robóticas Elsie y Elmer	48
Figura 31: Robot EV3 de Lego	49
Figura 32: Robot Roomba y ejemplo de trayectoria.	50
Figura 33: Ejemplo de servicio API/REST entre un cliente y un servidor.	55
Figura 34: Logo de la librería Pyserial	56
Figura 35: Ejemplos de marcadores fiduciaros.	58
Figura 36: Algoritmo de posicionamiento y representación de marcas ArUco	60
Figura 37: Robot Astrobees que hay en la ISS	61
Figura 38: Comparación de exploración entre Dijkstra (izquierda) y A* (derecha)	64
Figura 39: Robot hexápodo mostrando máxima altura de paso	66
Figura 40: Botones y Joysticks de mando de PS2	67
Figura 41: Robot a la máxima altura	70
Figura 42: Distintos tipos de rotaciones permitidas por el robot (6 tipos).	72
Figura 43: Distintos tipos de desplazamientos que puede hacer el robot	73
Figura 44: PCB sin modificar	74
Figura 45: PCB tras levantar el pegamento termofusible	74
Figura 46: Circuitería de la PCB	75
Figura 47: PCB tras desconectar el mando	76
Figura 48: Esquema de conexiones de RaspBerry pi Model 3	77
Figura 49: Conexión mediante GPIO del robot a la RaspBerry	77
Figura 50: Cámara NoIR V2	78
Figura 51: Robot hexápodo tras incorporar los nuevos elementos	79
Figura 52: Sistemas de referencia presentes en una arquitectura Eye-In-Hand	80
Figura 53: Ejemplo de URL	83
Figura 54: Caso de uso de control mediante API/REST	86
Figura 55: Caso de uso de visualizar marcas ArUco	87
Figura 56: Caso de uso de navegación autónoma	88
Figura 57: Caso de uso de control a través de interfaz	89
Figura 58: Caso de uso de simular navegación	90
Figura 59: Caso de uso del sistema general	91

Figura 60: Ejemplo XML	93
Figura 61: Ejemplo JSON	93
Figura 62: Conexión por USB de Arduino	95
Figura 63: Ejemplo de paquetes de control de MQTT	95
Figura 64: Ejemplo Arquitectura MQTT	97
Figura 65: Ejemplo protocolo pub/sub con broker	97
Figura 66: Logo de Mosquitto	98
Figura 67: IDE de Arduino	99
Figura 68: API de Flask cuando recibe un JSON	102
Figura 69: Marcas ArUco reconocidas	104
Figura 70: Ejemplo de imágenes que se utilizan para la calibración	105
Figura 71: Marcas reconocidas (izquierda) e ID, posición y rotación de las mismas (izquierda)	105
Figura 72: Distribución de las marcas en el suelo	106
Figura 73: Equivalencia de casillas y marcas	107
Figura 74: Flujograma de aproximación a una marca	109
Figura 75: Diagrama de secuencia del sistema de navegación	110
Figura 76: Terreno habilitado para moverse por él	111
Figura 77: Esquema de composición del mundo	112
Figura 78: Robots virtuales	113
Figura 79: Distintos ejemplos de estructuras incluidas en el mundo	114
Figura 80: Luces añadidas al mundo	115
Figura 81: Luz solar añadida en el mundo	116
Figura 82: Meteorito hecho con partículas	116
Figura 83: Humo hecho con partículas	116
Figura 84: Vista en primera persona	117
Figura 85: Vista en tercera persona	118
Figura 86: Cuadrícula elegida para las pruebas	119
Figura 87: Recorrido que tenía que realizar el robot	120
Figura 88: Pantalla principal de RaspBian	126
Figura 89: Lugar donde se encuentra el botón de configuración	127
Figura 90: Ops que hay que seleccionar para el funcionamiento del sistema	128

Figura 91: Conexión SCI de la RaspBerry	129
Figura 92: Comandos para actualización de RaspBian	130
Figura 93: Comando para instalar la librería picamera	130

ÍNDICE DE TABLAS

Tabla 1: Comandos para el modo movimiento del robot.....68

Tabla 2: Comandos para el modo rotación del robot.....69

1. Introducción

En este capítulo se va a hablar de la motivación que ha llevado a hacer esta propuesta, la propuesta en sí, los objetivos que se intentan conseguir, la planificación en el tiempo para el proyecto y la estructura de la que consta la memoria.

1.1 Motivación

Los recientes avances en la tecnología IoT para diversas aplicaciones desbordan la detección tradicional de los entornos que nos rodean. Está emergiendo muy rápidamente y se está convirtiendo en una fuente importante de intercambio de comunicación entre diferentes sensores, incluidos los sensores de visión a través de dispositivos inteligentes como los smartphones, etc [1], esto, junto a que, los nuevos dispositivos pueden ejecutar cualquier programa, de manera mucho mejor y más eficaz para hacer diferentes tareas [2], hace que esta propuesta de TFG sea de lo más interesante para profundizar en este campo.

También se va a utilizar visión por computador, que aunque sea una tecnología que data del 1960 de la mano de Larry Roberts, y a pesar de ser muy compleja y difícil, se encuentra en constante desarrollo con el surgimiento de nuevas tecnologías con el paso de los años [3], por lo que incluir este área en el trabajo aumentará el espectro de áreas trabajadas y ayudará a adquirir unos conocimientos mínimos sobre este tema.

Para aunar estos campos, se va a utilizar un servicio API/REST, se ha elegido este método porque el número de servicios que ofrecen interfaces de programación de aplicaciones (API) ha crecido rápidamente. Varios estudios señalan que los desarrolladores han pasado del protocolo simple de acceso a objetos (SOAP) o de la llamada a procedimiento remoto (RPC), a desplegar servicios web de transferencia de estado representativo (REST), como medio para que los consumidores utilicen sus servicios. Esto lo corroboran grandes sitios web como

Google, Facebook o Twitter, que ahora despliegan servicios REST para facilitar el acceso a sus valiosos recursos de datos, al tiempo que promover sus negocios [4].

Otras tecnologías que ayudarán a desarrollar este proyecto y las cuales también han motivado son:

- RaspBerry Pi 3 model B: ordenador de placa simple (SBC) de bajo coste la cual nos servirá para comunicarnos con el Arduino incorporado en el robot mediante puerto serie. Y a la cual se le conectará una cámara para la visión por computador.
- Lenguaje de programación Python: un lenguaje de programación potente y elegante que es fácil de leer y entender [5].
- Unity

Se van a usar metodologías diversas, pero que una vez unidas, van a permitir tener un conjunto de herramientas bastante amplio para trabajar con ellas y poder dar independencia a nuestro robot.

Además, otro de los atractivos que tiene este trabajo de fin de grado es que está orientado a utilizarse en talleres de exposición y de divulgación científica, para aumentar el interés por la informática y la electrónica.

Y por último, la multidisciplinariedad que conlleva este trabajo de fin de grado, es cuanto menos interesante, combinamos partes de robótica, con partes de sistemas y administración de redes, junto a inteligencia artificial, pasando por interfaces multimodales y finalizando por programación 3D, un trabajo fin de grado de lo más multitudinario y completo que se puede pedir.

1.2 Propuesta

Los propósitos que se persiguen abordar en este trabajo son:

- Implementación en Raspberry Pi con Python para la integración con la base de Arduino.
- Diseño y programación de algoritmos de reconocimiento de marcas.
- Diseño y programación de algoritmos de control para la navegación autónoma.
- Diseño y programación de una aplicación, interfaz de usuario, para controlar los movimientos del robot de forma manual.

1.3 Objetivos

Se empieza este proyecto estudiando el estado del arte en cuanto a visión por computador e IoT con sensores de visión, se estudiarán distintos tipos de sistemas que conforman los proyectos que se están implementando actualmente, los sistemas de comunicación que utilizan, qué información intercambian y como relacionan los componentes entre sí, que es de lo que se trata este trabajo de fin de grado, de conectar todos los elementos que se proponen para terminar con un sistema completo y funcional.

Una vez se han estudiado cómo están conectados esos componentes, se pasará a la integración de un dispositivo low cost, en este caso una RaspBerry, para su sincronización con Arduino, que es la base del robot. El objetivo de esta integración es que al ponerle una cámara a la RaspBerry, podamos tener un punto de vista de primera persona del robot.

A continuación, una vez se tengan conectados la RaspBerry y el Arduino, se tiene que desarrollar servicios remotos en Python que permitan controlar al robot mediante peticiones GET por WiFi.

Tras eso, hay que integrar un sensor de visión en el dispositivo low cost, como se ha mencionado anteriormente para recoger imágenes en tiempo real.

Una vez está finalizada la parte más hardware, obviando la parte de servicios remotos, se pasará a la parte software, la cual consistirá en desarrollar primeramente algoritmos de procesamiento de imagen que permitan localizar marcas en el entorno para conocer la ubicación del robot.

Una vez sea capaz de reconocer marcas, hay que desarrollar un generador de trayectorias para recorridos, que se integrará tanto con el procesamiento de imagen como con el servicio remoto para controlar las acciones de desplazamiento.

Para finalizar, se hará una aplicación, mejor llamada interfaz, para que el movimiento del robot sea interactivo y amigable para el usuario final.

En conjunto queda un sistema donde se podrá controlar al robot manualmente a través de una interfaz cómoda para el usuario o mandar al robot que haga una ruta a partir de unas marcas, todas estas acciones se verán reflejadas en la interfaz para mayor feedback con el usuario.

1.4 Planificación temporal

En este apartado se plantea la planificación temporal con la que se planea el proyecto, siendo las tareas:

1. Búsqueda de información sobre integración de sensores IoT de visión y control de robots.
2. Búsqueda de información sobre procesamiento de marcas de visión y técnicas de navegación autónoma
3. Implementación en Raspberry Pi con Python para la integración con la base de Arduino (encargada de proveer a los servos de capacidad para moverse conjuntamente, de forma ordenada e inteligente, permitiendo realizar movimientos en cada una de las extremidades).
4. Diseño y programación de algoritmos de reconocimiento de marcas.
5. Diseño y programación de algoritmos de control para la navegación autónoma.

- La distribución de esas tareas a lo largo del tiempo se puede ver en las Figuras 1-3.



1.5 Estructura de la memoria

La estructura de la memoria está compuesta por 6 capítulos los cuales se detallan a continuación:

En el capítulo 1 se realiza una introducción al proyecto, en la cual se expone la motivación del desarrollo de este proyecto, así como los objetivos a alcanzar en el mismo y la metodología para alcanzarlos.

En el capítulo 2 se muestra el estado del arte respecto a la visión por computador y robots hexápodos, mostrando los diferentes tipos de proyectos que se han aprovechado de estas tecnologías.

En el capítulo 3 se explican las tecnologías software que se van a utilizar en este proyecto, una explicación teórica para ponernos en situación y otra más práctica donde se explica cómo vamos a utilizarlas.

En el capítulo 4 se expone la parte hardware del proyecto, la base de la que parte que es el robot hexápodo, que funciones tiene y los límites que puede tener, para a continuación, explicar las modificaciones que hemos tenido que hacerle para que nos sirviera para este trabajo de fin de grado.

En el capítulo 5 se explican las distintas partes del proceso de ingeniería de software que se han llevado a cabo para formar este trabajo de fin de grado, concluyendo con un apartado de resultados donde se muestran los resultados obtenidos.

En el capítulo 6 se presentan las conclusiones personales del proyecto y propuestas de mejora para el futuro.

Para terminar está el anexo 1 donde se explica cómo configurar la RaspBerry para su funcionamiento y puesta en marcha para el proyecto, el anexo 2 donde se explica como poner en funcionamiento al sistema, el anexo 3 que explica por qué se calibra la cámara y finalmente la bibliografía recopilada en el estudio de este trabajo de fin de grado.

2. Control visual en robots

En este capítulo se encuentran trabajos relacionados con los conceptos básicos de los que consta este trabajo de fin de grado, al igual que ciertas explicaciones teóricas que hay que tener como base para entender el funcionamiento de las distintas partes que forman el proyecto.

2.1 Visión por computador

La visión por computador busca que un computador sea capaz de identificar imágenes capturadas y realizar ciertas operaciones, ya sea en las propias imágenes o en los mecanismos de control del sistema.

Dentro de la visión por computador, el control visual es el ámbito más utilizado en la industria, y en el que nos centraremos con mayor atención en este capítulo, a parte del estudio de los robots hexápodos, ya que en él se basan las aplicaciones que se encuentran en los diferentes estudios, investigaciones, desarrollos civiles y militares.

2.1.1 Fases en un sistema de visión artificial

Básicamente, el ser humano capta la luz a través de los ojos; y la información captada viaja al cerebro por el nervio óptico. El cerebro procesa la imagen, interpreta la escena y actúa en consecuencia. Los sistemas de visión artificial buscan reproducir este mismo comportamiento, para lo cual, se definen cuatro fases en el sistema, estas fases son:

1. Captura: Etapa en la que se adquiere la información, las imágenes.
2. Procesamiento: Etapa en la que mediante filtros o transformaciones se eliminan partes no deseadas de las imágenes, facilitando las etapas posteriores.
3. Segmentación: En esta etapa se aíslan los elementos que interesan del resto de la imagen para su comprensión.
4. Reconocimiento: En esta etapa se pretende distinguir los objetos segmentados mediante una serie de características que previamente se han establecido para diferenciarlos.

Las fases se pueden apreciar en la Figura 4.

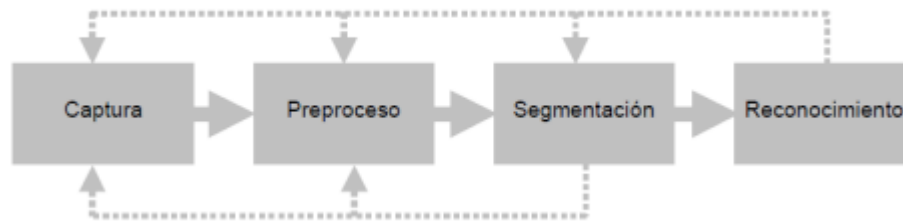


Figura 4: Fases de un sistema de visión por computador

Estas fases se siguen de manera secuencial en un sistema habitual de visión artificial, pero puede suceder que no sigan esa secuencia siempre, sino que se realimenten hacia atrás en el ciclo. Así, si la segmentación falla, se puede volver al procesado o a la captura para intentar arreglar el fallo.

2.1.1.1 Captura

La captura es el proceso en el que se obtiene información sobre una escena mediante un dispositivo óptico. Si la imagen es digital, además de la captura, hay que someterla a una digitalización, en la que transforma la información continua en información con todas sus componentes discretas.

De manera genérica, el proceso de captura se puede realizar mediante dos métodos. El primero se basa en el principio de la cámara oscura, mediante dispositivos pasivos; y el segundo método se basa en el escaneo de imágenes mediante dispositivos activos. También se puede capturar imágenes para el uso en el sistema de visión por computador mediante la construcción de imágenes sintéticas.

El principio de la cámara oscura se puede usar para obtener imágenes del mundo real y proyectarlas en un plano bidimensional. La cámara oscura se basa en la proyección de la imagen a través de una lente fina situada delante de un orificio, que hace que se obtenga una imagen invertida tanto vertical como horizontalmente de la escena que se está produciendo fuera de la cámara. La cámara es una caja cerrada y la luz entra en ella en el orificio que se le produce. La imagen obtenida se proyecta en la pared opuesta a la que tiene el orificio. Este dispositivo pasivo es un dispositivo muy conocido, utilizado en Bagdad por primera vez en el año 965 por el matemático árabe Alhacen, aunque no fue hasta 1550, cuando Cardan le aplicó una lente al orificio para aumentar la luminosidad y Kepler utilizó este principio para crear las bases del telescopio en 1604. Este principio es en el que se basa la cámara estenopeica, o cámara pinhole, precursor de la cámara fotográfica.

El otro método de captura de imágenes es el escaneo. En él, un elemento activo, como puede ser un haz de luz, recorre la escena que se desea capturar. Debido a esto, se necesitan como mínimo un haz de luz, un emisor de luz y un receptor o detector de luz, para que cuando el haz choque con la escena tras ser emitido, se pueda recoger con el detector y representar la escena tras la repetición de este proceso de manera continua. Los escáneres se usan con diferentes fines. Eso nos lleva a que un escáner de tambor, creado para dar representaciones 3D, pueda capturar imágenes de elementos bidimensionales y que escáneres-láser puedan capturar escenas tridimensionales aunque su aplicación fundamental sea la captura bidimensional.

De estos dos métodos, el primero, el que se basa en el modelo de cámara oscura, es el modelo que mejor se adapta a la mayoría de usos, debido a que es una forma muy sencilla y económica de capturar imágenes, a que la velocidad de captura es mucho mayor y a que el funcionamiento es bastante similar al del ojo humano. Es en el proceso de digitalización donde la cámara tiene una desventaja frente al escáner, ya que hasta hace relativamente poco tiempo, era un punto crítico, ya que actuaba como cuello de botella, pero con los desarrollos actuales y el auge de la fotografía digital y de las cámaras integradas en dispositivos de telefonía móvil, esta desventaja se ve disminuida casi al máximo.

La digitalización es el otro apartado que se necesita para capturar la imagen y poder pasar a la siguiente fase de procesamiento de imágenes. Es el paso del mundo analógico continuo al mundo digital discreto. Tiene a su vez dos pasos, que son el muestreo y la cuantización. El muestreo de una señal continua consiste en la discretización respecto a una variable, ya sea tiempo o espacio de manera general. El parámetro fundamental del muestreo es la frecuencia de muestreo, que es el número de veces que se mide un valor analógico por unidad de cambio.

En una imagen, el muestreo convierte la imagen continua en una matriz discreta de $N \times M$ píxeles. El píxel es la menor unidad homogénea en color de una imagen digital. El número de muestras por unidad de espacio sobre el objeto original lleva a la resolución espacial de la imagen, concepto definido por la distancia, sobre el objeto original, entre dos píxeles consecutivos. Ahora bien, la unidad más común de resolución espacial es el píxel por pulgada o DPI, medido siempre en el objeto original.

La siguiente operación en la digitalización es la cuantización de la señal, que es la discretización de los posibles niveles de los valores de cada píxel. Los niveles de esa cuantización se hacen en potencias de 2 para facilitar el almacenaje en el computador de esas imágenes, ya que el ordenador usa el byte como unidad de

memoria directamente direccionable. Los niveles posibles definen la resolución radiométrica y esos niveles pueden ser 2, 4, 16 o 256 habitualmente. En la figura 5 se pueden apreciar los distintos niveles de intensidad.

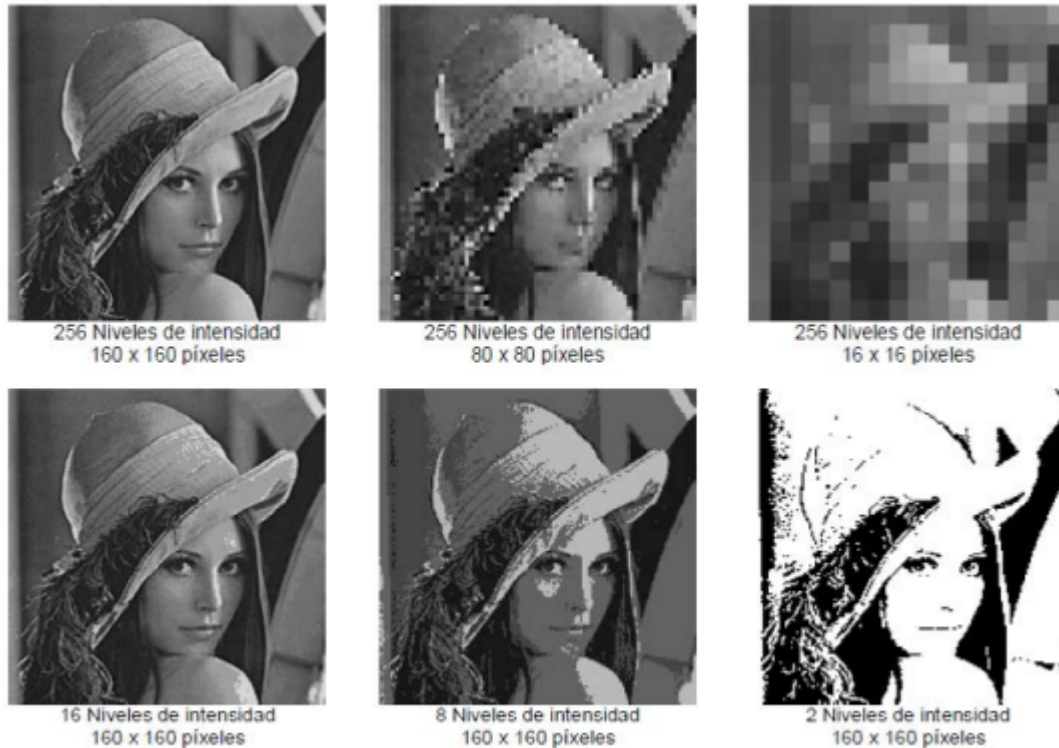


Figura 5: Niveles de cuantización

Cuando las imágenes tienen únicamente información sobre el brillo se habla de imágenes en niveles de gris. Se suelen utilizar hasta 256 niveles para representar desde el negro (nivel 0) al blanco (nivel 255).

Si se permiten dos niveles de cuantización solamente, estamos hablando de una imagen binaria o bitonal, y si nos vamos a imágenes en color, se suele usar 256 niveles de intensidad para cada uno de los canales que corresponden a los tres colores primarios, rojo (R), verde (G) y azul (B); obteniéndose de esta forma una imagen de 16 millones de colores ($256 \times 256 \times 256$). Estas imágenes se llaman imágenes en color real. En algunos casos se puede necesitar de mayor resolución radiométrica, llegando incluso a 4096 niveles. Además, las imágenes en color no solo se pueden cuantizar en el formato RGB, sino que se puede cuantizar en otros modelos de color, como el HSV (Hue, Saturation, Value (Matiz, Saturación, Valor)), utilizado por Apple e Intel para sus programas y librerías de programación como OpenCV, o el HSL (Hue, Saturation, Lightness (Matiz, Saturación, Luminosidad (o Intensidad))) que es el que usa, por ejemplo, Microsoft en sus programas incluidos en el sistema operativo Windows.

Estos dos procesos (muestreo y cuantización) se pueden realizar de manera uniforme, como se ha explicado, o de manera no uniforme, muestreando una parte de la imagen de forma más nítida, con una resolución espacial mayor, o realizando la cuantización de las imágenes con el uso de paletas, que son un conjunto de colores a los que se les asigna una referencia. Los píxeles de las imágenes que usan paletas contienen como valor de referencia al color de la paleta que quieren presentar. Cuando el número de colores de una imagen es pequeño, el uso de paletas permite un ahorro de memoria y simplificar operaciones al cambiar solo la paleta y no todos los píxeles de la imagen.

El proceso de digitalización requiere evaluar las dos resoluciones, la espacial y la radiométrica, que se precisan para representar de manera adecuada una imagen. O lo que es lo mismo, se necesita conocer la frecuencia con la que se muestrean los píxeles y la gama de colores permitida. A través del teorema de Shannon, al que fue convertida la conjetura de muestreo de Nyquist, se establece que la frecuencia mínima de muestreo para recuperar una señal sin errores debe ser al menos el doble de la máxima frecuencia con la que cambian los elementos que se quieran capturar en la señal. Este teorema, nos lleva a la conclusión de que si tenemos una imagen impresa a 200 DPIs, para su escaneo representando fielmente la imagen impresa necesitaremos al menos un muestreo de 400 DPIs.

También hay que tener en cuenta que dependiendo del uso que se vaya a hacer de la imagen se puede realizar la elección de parámetros de digitalización de manera menos objetiva. A la hora de usar una imagen para reconocimiento, nos puede ser más útil tener una gran variedad de niveles de intensidad pero una resolución espacial pequeña, pero para publicar un periódico en blanco y negro, con 16 niveles de intensidad, una resolución pequeña no será admisible.

2.1.1.2 Procesamiento

El procesamiento, o procesado de la imagen consiste en la realización de una serie de transformaciones con el objetivo primordial de facilitar la búsqueda de información para obtener las características de la imagen, o simplemente, mejorar la calidad de la imagen.

El procesado de imágenes va desde simples operaciones aritméticas o geométricas entre píxeles, hasta complejos filtros de contornos como los operadores Canny o Sobel. En este apartado se indicarán las principales operaciones que se realizan para procesar una imagen.

Operaciones aritmético-lógicas:

Las operaciones aritmético-lógicas son las operaciones más comunes que se pueden hacer a una imagen, pues se realizan leyendo los valores píxel a píxel y modificándose según se realiza.

Dentro de este grupo, están las siguientes operaciones:

- a. Conjunción: Operación lógica AND entre los bits de dos imágenes. Se suele usar para borrar píxeles.
- b. Disyunción: Operación lógica OR. Se usa para añadir píxeles a una imagen.
- c. Negación: Inversión de los bits que forman una imagen y con ella, se obtiene el negativo de la imagen.
- d. Suma: Suma de los valores de los píxeles de dos imágenes.
- e. Resta: Resta de los valores de los píxeles de dos imágenes.
- f. Multiplicación: Multiplicación de los valores de los píxeles de dos imágenes. Se usa para añadir textura a una imagen.
- g. División: División de los valores de los píxeles de una imagen entre los de otra imagen.

En la Figura 6, se observan las diferentes operaciones aritmético-lógicas realizadas en una imagen en blanco y negro para ver sus aplicaciones.

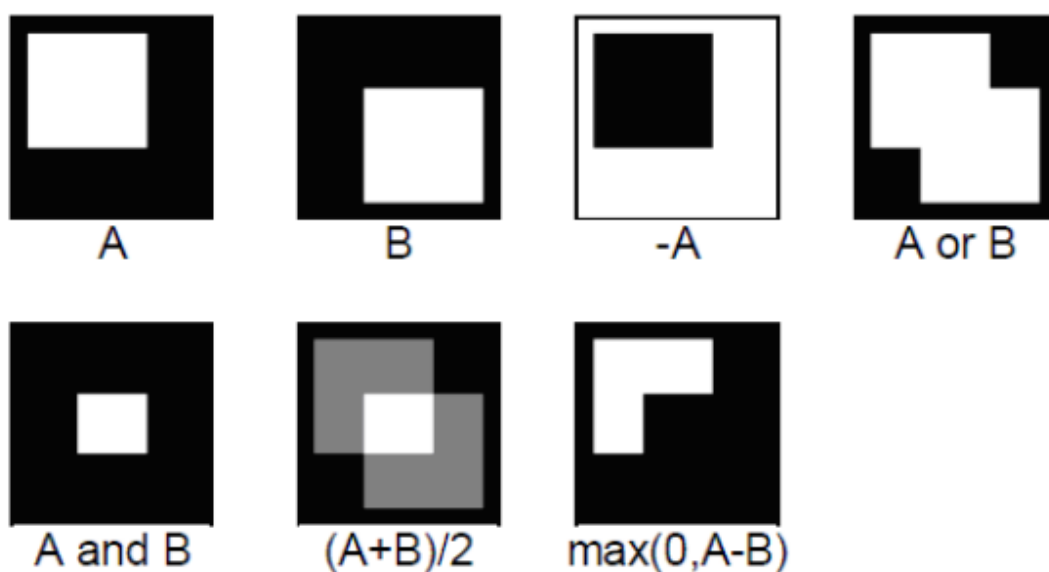


Figura 6: Operaciones aritmético lógicas

Operaciones geométricas

Las operaciones geométricas son operaciones en las que se modifica la posición de la imagen o su forma relativa. Expresadas en coordenadas homogéneas, estas operaciones se pueden expresar como una multiplicación de matrices. Las más frecuentes son

- a. Traslación: Movimiento de los píxeles de una imagen según un vector de desplazamiento.
- b. Rotación: Giro relativo de los píxeles de una imagen a un eje o a un punto, generalmente el origen de coordenadas.
- c. Escalado: Cambio del tamaño de una imagen al multiplicar las dimensiones de ésta por un factor de escala.

En la figura 7, la operación (a) es una traslación al origen. La operación (b) define el punto de rotación de la imagen. La operación (c) rota la imagen respecto al origen de coordenadas y la operación (d) vuelve a trasladar la imagen, ya girada, al punto inicial.

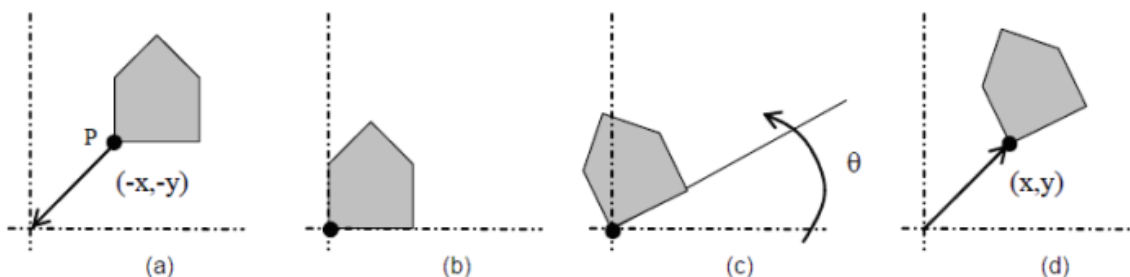


Figura 7: Operaciones geométricas en un imagen

Operaciones sobre el histograma

Las operaciones sobre el histograma son operaciones en las que se cambia el contraste y la luminosidad de la imagen. Primero habrá que hablar de qué es el histograma.

El histograma de una imagen es un diagrama de barras en el que se disponen en horizontal los niveles de cuantización y que cada barra del diagrama tiene una altura proporcional al número de píxeles que hay en cada nivel de cuantización. Los histogramas se suelen dar para imágenes en blanco y negro con 256 niveles de intensidad. Si se trata de una imagen a color, se da un histograma para cada una de las componentes RGB de la imagen. Además, el histograma se suele normalizar entre 0 y 1 para evitar dependencias del número de píxeles o de niveles.

Mediante una función de transferencia del histograma, las operaciones en éstos se visualizan de manera muy eficaz. Estas funciones de transferencia, al final, corresponden a aplicaciones, ya que cada punto del dominio tiene un único valor imagen. Las operaciones más comunes en el histograma son:

Ajuste de contraste: El contraste se define como la diferencia de intensidad pronunciada. Pues mediante el uso de una función de transferencia del histograma que aclare los niveles oscuros y oscurezca los claros, podremos aumentar el contraste del conjunto. Y si hacemos la operación contraria, reduciremos el contraste.

Ecualización del histograma: El ecualizado consiste en obtener un nuevo histograma de la imagen a partir del original, con una distribución uniforme de los niveles de intensidad.

En las Figura 8 y 9, se observan las diferentes operaciones de ecualizado de histograma y de ajuste de contraste:

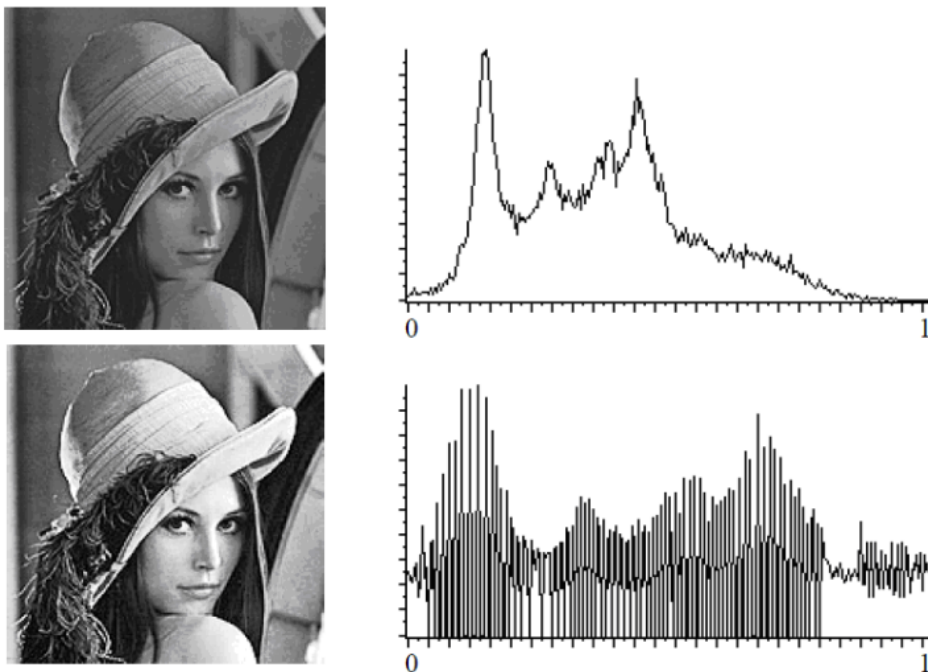


Figura 8: Imagen original (arriba) frente a imagen con el histograma ecualizado (abajo)

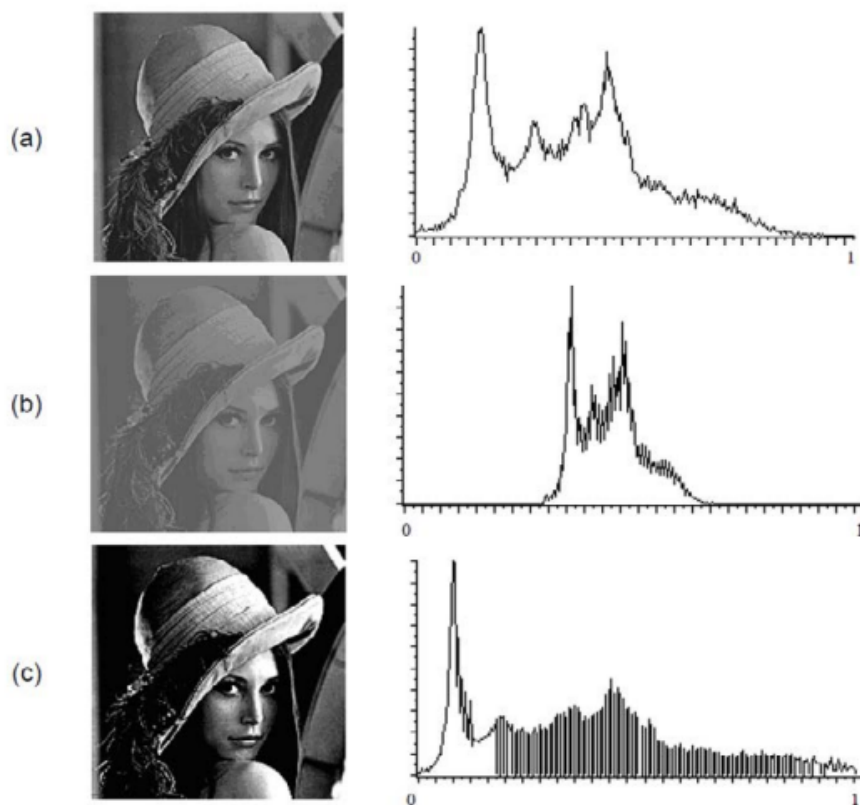


Figura 9: Imagen original (arriba) frente a imágenes con ajustes de contraste

Filtrado en el dominio de la frecuencia .

El filtrado de una imagen en el dominio de la frecuencia procesa una imagen aplicando la transformada de Fourier de la imagen y trabajando sobre el dominio de ésta. Para ello, se modifica el teorema de convolución de señales de forma que lo primero que se realiza es la transformada de Fourier de la imagen, después se aplica el filtro, y por último se realiza la transformada inversa de Fourier para devolverla al dominio espacial. Estos pasos se pueden ver en la Figura 10.

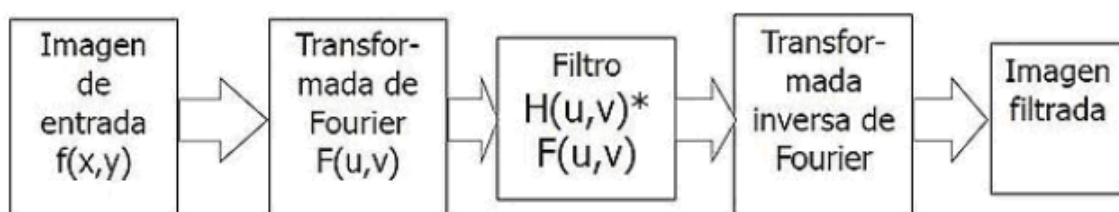


Figura 10: Secuencia de transformada en el filtrado de una imagen en el dominio de frecuencias

Se puede utilizar otras transformadas como la transformada rápida de Fourier, o la bidimensional, o la discreta; pero al aplicar cualquiera de estas, después hay que

aplicarle el filtro y luego devolver con la transformada inversa al dominio del espacio para obtener la imagen. Los filtros más comunes son:

Filtro pasa bajos: Atenúa las frecuencias altas y mantiene sin variación las frecuencias bajas. El resultado es un suavizado de las transiciones de la imagen, reduciendo el ruido, pues se atenúan los cambios fuertes de intensidad.

Filtro pasa altos: Atenúa las frecuencias bajas y mantiene sin variación las frecuencias altas. El resultado es una mejora en la detección de bordes y un refuerzo en los contrastes de la imagen.

Filtro pasa banda: Atenúa las frecuencias muy bajas y muy altas, manteniendo sin variación las frecuencias medias.

Como la multiplicación en el espacio de Fourier es idéntica a la convolución en el dominio espacial, estos mismos filtros podrían implementarse como filtros espaciales. Ejemplo de estos filtros en la Figura 11.

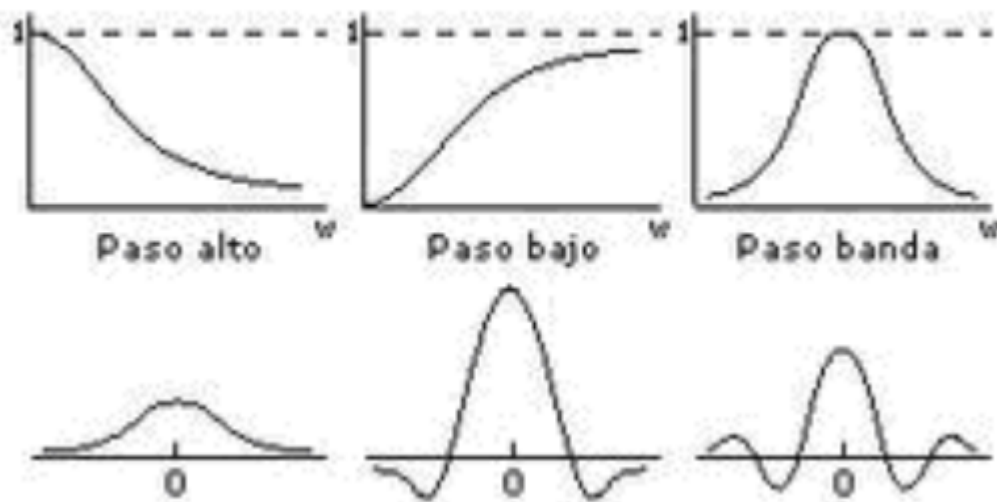


Figura 11: Tipos de filtros en el dominio de frecuencia

Estos filtrados tienen una serie de ventajas, como ser métodos sencillos de implementar, ser rápidos debido al uso del teorema de convolución, fácil asociación de concepto entre tonalidad y frecuencia; y proporcionar mucha flexibilidad en el diseño de soluciones, pero también tiene unos inconvenientes que son la necesidad de conocer varios campos para el desarrollo de una aplicación de procesamiento de imágenes y la imposibilidad de eliminación total del ruido de la imagen.

Filtrado en el dominio del espacio

El filtrado en el dominio del espacio se realiza directamente sobre la imagen. Normalmente se usa un filtro de convolución, aplicando una máscara de convolución, que al final reduce la operación a la multiplicación de los píxeles de la imagen por una matriz de convolución, que será la responsable del filtro. Existen otro tipo de filtros, que son los no lineales, que no se basan en la máscara de convolución. Para realizar el filtrado espacial, se realiza una convolución de la máscara sobre la imagen. Se sigue el Teorema de Convolución en el espacio:

$$g(x, y) = h(x, y) * f(x, y) \quad (1)$$

Cada píxel de la nueva imagen se obtiene del sumatorio de la multiplicación de la máscara por los píxeles contiguos:

$$g(x, y) = \sum \sum f(i, j)w(i, j) \quad (2)$$

Y se suele normalizar, dividiendo entre un valor constante, que se obtiene de la suma de los valores de la máscara empleada. Los filtros espaciales se dividen en los siguientes tipos:

Suavizado (Filtro pasa bajos): Utilizados para eliminar ruido o detalles de poco interés, estos filtros dependen mucho de la máscara utilizada. Pueden ser de promedio, promediando los píxeles contiguos con una matriz unidad; de mediana, de media, de pasa bajo en frecuencia, el del bicho raro y el gaussiano. El filtro de la mediana y el del bicho raro son filtros no lineales, el resto son lineales (Figura 12).



Figura 12: Imagen original frente a imagen con un filtro suavizado

Atenuación (Filtro pasa altos): Intensifica los bordes, los detalles y los cambios de alta frecuencia, y atenúa las zonas de tonalidad uniforme. Permite mejor identificación de objetos en la imagen, y es común la aparición de ruido.

Realce de bordes por desplazamiento y diferencia: Sustraer de la imagen original una copia desplazada de la misma. Su resultado es el realce de bordes existentes según la máscara de convolución utilizada sea horizontal, vertical o diagonal (horizontal/vertical).

Laplaciana: Este tipo de filtro realza los bordes en todas las direcciones, ya que trabaja con el operador laplaciano, que permite una mejora de resultados con un aumento de ruido en la imagen.

Filtrado de realce de bordes por gradiente direccional: Se emplea para destacar y resaltar con mayor precisión los bordes que se localizan en una dirección determinada. Trabaja con los cambios de intensidad existentes entre píxeles contiguos.

Obtención de Contornos: Como los filtros anteriores, estos detectores de contornos se basan en las diferencias de intensidad que se dan píxel a píxel. Requieren un menor coste computacional, ya que trabajan desde el operador gradiente en una máscara que hace que se obtenga el contorno de la imagen y se pueda clasificar las formas existentes en ella. Las más utilizadas son los filtros de Sobel (Horizontal y Vertical), el filtro de Prewitt, el de Roberts y el detector de contornos de Canny.

$$\begin{aligned} \text{Sobel: Horizontal: } & \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Vertical: } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\ \\ \text{Prewitt: Horizontal: } & \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Vertical: } \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \\ \\ \text{Robert Horizontal: } & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{Vertical: } \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \end{aligned}$$

Figura 13: Ejemplo de matrices usadas para los filtros



Figura 14: Imagen original (izquierda) frente a imagen tratada con el filtro de Canny (centro) y tratada con el filtro de Sobel (derecha).

Canny: Se basa en la primera derivada de una imagen que previamente se le ha realizado un filtro gaussiano (Figura 14).

Operaciones morfológicas

Las operaciones morfológicas tienen su razón de ser en la morfología matemática, basada en la teoría de conjuntos. En la morfología matemática, las imágenes binarias son conjuntos de puntos en dos dimensiones que representan los puntos activos de una imagen. Las imágenes en niveles de gris, son conjuntos en tres dimensiones, donde la tercera componente es el nivel de intensidad de gris.

Puesto que son operaciones basadas en la teoría de conjuntos, sus definiciones se darán también con esa notación. Y además, se darán las definiciones de estas operaciones para imágenes binarias, ya que son más sencillas de entender.

Las diferentes operaciones morfológicas que se pueden hacer son:

Dilatación: La dilatación de una imagen A con un elemento estructurante B consiste en examinar cada píxel de A, y si esta en valor 0, pasarlo al valor 1 si alguno de sus vecinos está en la imagen A al ser recorrida por la imagen B (o lo que es lo mismo, tiene valor 1). Normalmente se utilizan como vecinos los 8 píxeles que rodean al estudiado, aunque puede darse una conectividad de 4 píxeles, sin las diagonales.

Erosión: Es el proceso inverso a la dilatación. Pasa a valor 0 los píxeles con valor 1 si alguno de sus vecinos está con valor 0.

Apertura: Es la operación de erosionar y luego dilatar una imagen.

Cierre: Es la operación de dilatar y luego erosionar una imagen y por lo tanto, la operación inversa a la apertura.

Gradiente Morfológico: Es la operación resultante de restar la imagen erosionada de la imagen dilatada (Gradiente= Dilatación Erosión).

Top Hat: Esta operación se define como la resta de la imagen después de una apertura de la imagen original (Top Hat = Original Apertura). Esta operación aísla los píxeles de gran intensidad.

Black Hat: Esta operación se define como la resta de la imagen original de la imagen después de un cierre (Black Hat = Cierre Original) Esta operación aísla las zonas oscuras.

Todas estas operaciones se pueden ver en la Figura 15.

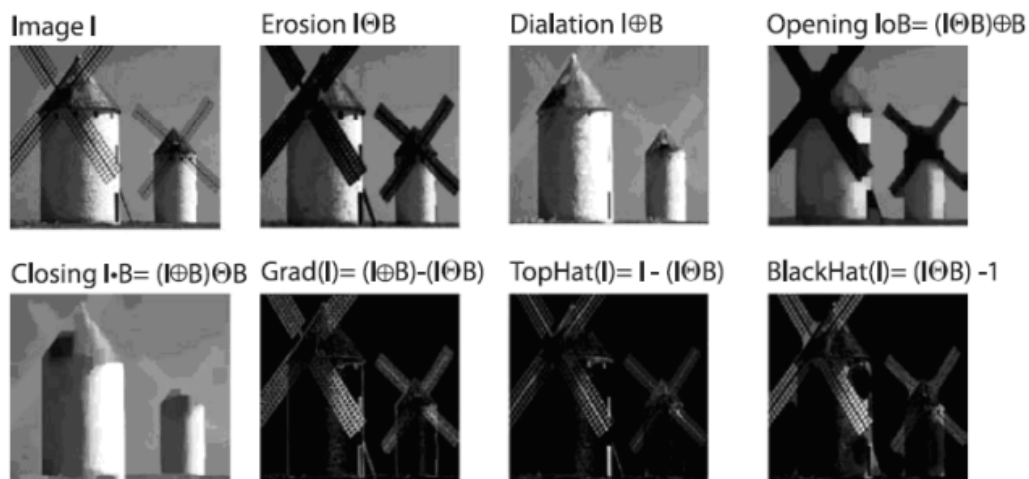


Figura 15: Operaciones morfológicas que se le pueden realizar a una imagen

2.1.1.3 Segmentación

La segmentación es la siguiente fase a considerar dentro de un sistema de visión por computador. La segmentación es un proceso que consiste en dividir una imagen en regiones homogéneas con respecto a una o más características como puede ser el brillo o el color, con el fin de facilitar un posterior análisis o reconocimiento. Esto hace que se puede utilizar tanto para localizar objetos como para encontrar los bordes de los mismos dentro de una imagen.

La segmentación asigna una etiqueta a cada píxel de la imagen de forma que los píxeles que comparten etiqueta tendrán ciertas características similares visuales. Este proceso termina cuando los objetos extraídos de la imagen se corresponden

unívocamente con las distintas regiones disjuntas a localizar en la misma. Se habla en este caso de segmentación completa, en oposición a la segmentación parcial, en la que no corresponde de manera unívoca y en la que se deberá hacer algún tipo de tratamiento posterior para conseguir la segmentación completa necesaria para completar el reconocimiento.

Los algoritmos de segmentación están basados en las discontinuidades de los niveles de gris, segmentando la imagen desde los cambios grandes de niveles de gris entre píxeles, como se hace en la detección de líneas, detección de bordes, detección de esquinas, de puntos aislados; o están basados en el efecto contrario, en la similitud de niveles de gris, al agrupar píxeles con características similares mediante umbralización, crecimiento de regiones, etc.

Existen numerosos métodos de segmentación, entre ellos:

Método del valor umbral

El método del valor umbral o umbralización es un método por el que se convierte una imagen de niveles de gris o de color en una imagen binaria, de manera que los objetos de interés tienen una etiqueta distinta al fondo de la imagen. Esta técnica requiere un coste computacional muy bajo, y por lo tanto, es una técnica rápida, pudiendo ser realizada en tiempo real en un computador.

El caso más simple de umbralización es el umbral único o fijo, en el que se establece un umbral fijo sobre el histograma, que marca la separación y es el punto clave del método. El umbral es el límite por el que se divide la imagen, asignando cada píxel a uno de los dos segmentos comparando el umbral y el nivel.

Los métodos del valor umbral son métodos de segmentación completos, pues cada píxel pertenece a uno de los dos segmentos y solo a uno de esos segmentos. La imagen final se calcula con respecto al umbral t de la siguiente manera: $T_{global}(g) = 0$ si $g < t$ y $T_{global}(g) = 1$ si $g \geq t$. Existen varias variantes de este método, como es la umbralización de banda, en la que se pone un límite inferior y uno superior; la multiumbralización, que consiste en dar varios umbrales a la imagen dando lugar a regiones diferentes y no a una imagen binaria, la semiumbralización, consistente en mantener la imagen tal cual está a partir del umbral, pero binarizarla si el valor del píxel es inferior al umbral, o la umbralización adaptativa, en la que se permite el cambio del valor umbral según las características del entorno del punto de estudio.

En la Figura 16 (b) vemos el histograma de niveles de gris de la figura (a). Para realizar la umbralización, se toma un nivel (c) a partir del cual, se binariza la imagen (d).

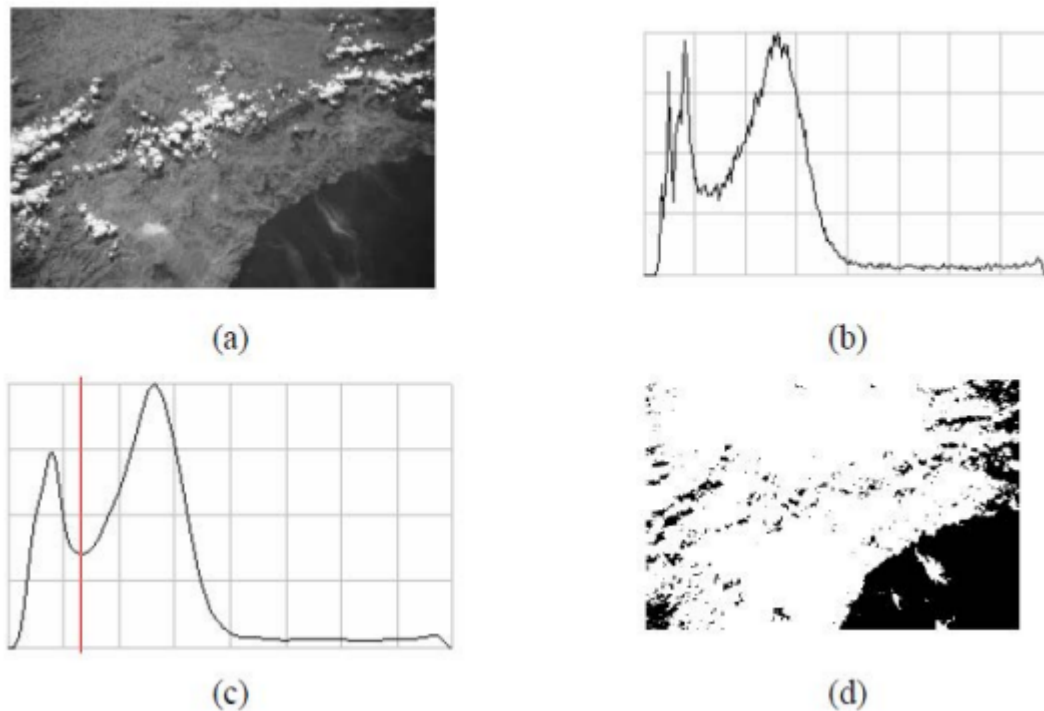


Figura 16: Umbralización de Otsu

La umbralización tiene su explicación matemática en el método de Otsu, llamado así por su creador Nobuyuki Otsu en 1979. Este método utiliza la varianza, calculando el valor umbral de manera que esa varianza sea la menor posible en cada segmento, pero que sea máxima entre segmentos distintos. Se calcula el cociente de ambas varianzas y busca un valor umbral que sea el que maximice ese cociente.

Transformación de Hough

La transformación de Hough es un algoritmo creado en 1962, que permite encontrar ciertas formas como rectas, círculos, etc, dentro de una imagen. La versión más simple es la que encuentra las líneas como características dentro de una imagen. Este algoritmo aprovecha la expresión del haz de rectas que pasan por un punto genérico (x_i, y_i) , cuya ecuación es:

$$(y_i = ax_i + b) \quad (3)$$

Siendo a y b parámetros variables. Su modo de operación es principalmente estadístico y consiste en que para cada punto que se desea averiguar si es parte de una línea se aplica una operación dentro de cierto rango, con lo que se averiguan las posibles líneas de las que puede ser parte el punto. Esto se continúa para todos los puntos en la imagen, y al final se determinan las líneas de la imagen. Las líneas

detectadas serán aquellas que tengan más puntos repetidos en el análisis, ya que la transformación se realiza buscando entre los vecinos del píxel analizado.

Métodos basados en el histograma

Los métodos basados en el histograma son métodos muy eficientes, pues solo requieren una única pasada por los píxeles de la imagen. Una vez se calcula el histograma, los picos y los valles de éste se usan para localizar grupos de píxeles en la imagen. Al refinar esta técnica se llega a un algoritmo recursivo de búsqueda en el histograma dividiendo la imagen en regiones cada vez más pequeñas. Este método tiene la desventaja de que al ir dividiendo en regiones cada vez menores, la identificación de picos y valles cada vez es más compleja.

Clustering o Método de agrupamiento

El clustering es una técnica de segmentación iterativa con la cual se puede dividir una imagen en un número determinado de regiones o clusters. El algoritmo básico es:

1. Escoger K centros de clusters, ya sea de forma aleatoria o basándose en algún método heurístico.
2. Asignar a cada píxel de la imagen el clúster que minimiza la varianza entre el píxel y el centro del cluster.
3. Recalcular los centros de los clusters haciendo la media de todos los píxeles del cluster.
4. Repetir los pasos 2 y 3 hasta que se consigue la convergencia (por ejemplo, los píxeles no cambian de clusters).

En este caso, la varianza es la diferencia absoluta entre un píxel y el centro del cluster. La diferencia se basa típicamente en color, la intensidad, la textura, y la localización del píxel, o una combinación ponderada de estos factores. El número K se puede seleccionar manualmente, aleatoriamente, o por una heurística.

Este algoritmo garantiza la convergencia, pero puede devolver una solución que no sea óptima. La calidad de la solución depende de la serie inicial de clusters y del valor de K. En estadística y aprendizaje automático, el algoritmo de las k-medias es un algoritmo de agrupamiento para dividir objetos en k grupos, donde $k < n$. Es similar al algoritmo de maximización de expectativas para las mezclas de gaussianas ya que ambos pretenden encontrar los centros de agrupaciones naturales de los datos.

El modelo requiere que los atributos del objeto correspondan a los elementos de un espacio vectorial. El objetivo es intentar alcanzar la mínima varianza total entre clusters, o, la función de error al cuadrado. El algoritmo de las k-medias fue

inventado en 1956. La forma más común del algoritmo usa una heurística de refinamiento conocida como el algoritmo de Lloyd. El algoritmo de Lloyd comienza dividiendo los puntos de entrada en k conjuntos iniciales, ya sea al azar o usando algunos datos heurístico y a continuación, calcula el punto medio o centro de gravedad de cada conjunto. Se construye una nueva partición, asociando cada punto con el centro de gravedad más cercano. Luego se recalculan los baricentros para los nuevos clusters, y el algoritmo repite alternando aplicación de estos dos pasos hasta que la converja, que se obtiene cuando los puntos ya no cambian de cluster (o los centros de gravedad ya no se modifican).

Método de crecimiento de regiones

Los métodos de crecimiento de regiones determinan zonas dentro de una imagen basándose en criterios de similaridad y proximidad entre los píxeles de la misma. En estas técnicas la homogeneidad (o falta de homogeneidad) entre regiones adyacentes es el criterio utilizado para unir (o dividir) regiones de la imagen. Dicha homogeneidad se puede definir a partir de criterios como: el nivel de gris medio, el color, la forma, etc. El resultado de la segmentación es una partición de la imagen en regiones homogéneas.

Las técnicas de segmentación basadas en contornos tratan de encontrar fronteras entre regiones. En general, las técnicas basadas en regiones trabajan mejor en imágenes con ruido, en donde los contornos son difíciles de localizar. La segmentación resultante de una detección de contornos y la basada en crecimiento de regiones, aplicadas a una misma imagen no producen normalmente el mismo resultado. Es posible combinar los resultados producidos por ambos tipos de segmentaciones. Las primeras técnicas de crecimiento de regiones fueron las del crecimiento a partir de semillas; en las que se dan una serie de semillas como entrada junto con la imagen. Las semillas marcan cada uno de los objetos que tienen que ser segmentados. Las regiones crecen iterativamente mediante la comparación de todos los píxeles vecinos no asignados a ninguna región.

Método del conjunto de nivel

La propagación de curvas es una técnica popular en el análisis de imágenes para la extracción de objetos, seguimiento de objetos, la reconstrucción en 3D, etc.

La idea central de este enfoque consiste en desarrollar una curva hacia el menor potencial de una función de coste que en su definición refleja la tarea a la que está dirigida e impone ciertas limitaciones de suavidad. Las técnicas de Lagrange se basan en la parametrización de acuerdo con alguna estrategia de muestreo y luego desarrollar cada elemento de acuerdo a la imagen y sus condiciones internas. Si bien esta técnica puede ser muy eficiente, también sufre varias limitaciones, como decidir sobre la estrategia de muestreo, la estimación de las propiedades

geométricas internas de la curva, el cambio de su topología, abordar los problemas de dimensiones superiores, etc. En cada caso, una ecuación en derivadas parciales llamada la ecuación del conjunto de nivel es resuelto por diferencia nita.

El método del conjunto de nivel fue propuesto inicialmente para realizar un seguimiento de interfaces móviles por Osher y Sethian en 1988 y se ha diseminado a través de varios dominios de imágenes a finales de los noventa. Se puede utilizar para hacer frente de manera eficiente al problema de la propagación de la curva o superficie de una manera implícita. La idea central consiste en representar la evolución del contorno usando una función con signo, donde su nivel cero corresponde al contorno actual. Entonces, de acuerdo a la ecuación de movimiento de las curvas de nivel, se puede obtener fácilmente un flujo similar de la superficie implícita que cuando se aplica al nivel de cero reflejará la propagación del contorno. El método del conjunto de nivel tiene numerosas ventajas como son que es un método implícito, intrínseco, sin parámetros, que ofrece una manera directa para estimar las propiedades geométricas de la estructura que evoluciona, y que puede cambiar la topología.

Métodos de particionamiento gráfico

Los métodos de particionamiento gráfico se pueden usar con eficacia en la segmentación de imágenes. En estos métodos, la imagen se modela como un grafo ponderado no dirigido. Por lo general, un píxel o un grupo de píxeles se asocian con los nodos y los pesos de las aristas definen la similitud entre los píxeles vecinos. El gráfico (imagen) se divide de acuerdo a un criterio de diseño para modelar "bien" los clusters. Cada una de las particiones de nodos (píxeles) da como salida de estos algoritmos los objetos segmentados que hubiese en la imagen. Algunos algoritmos populares de esta categoría son cortes normalizados, camino aleatorio, el mínimo corte, particionamiento isoperimétrico y árboles de expansión mínima.

Watershed o transformación divisoria

La transformación divisoria o watershed calcula las líneas divisorias o líneas de aguas. Una imagen en escala de grises puede ser vista como un relieve topográfico, donde se interpreta el nivel de gris de un píxel como su altura en el relieve. También, se puede considerar la magnitud del gradiente de una imagen como una superficie topográfica. Los píxeles que tienen las más altas intensidades de gradiente corresponden a las líneas divisorias, que representan los límites de las regiones. El agua puesta sobre cualquier píxel encerrado por una línea divisoria común fluye colina abajo a un mínimo de intensidad local común. Los píxeles que drenan a un mínimo común forman una cuenca, que representa un segmento de la imagen (un objeto).

Segmentación basada en modelos

La segmentación basada en modelos tiene como hipótesis fundamental que las estructuras de interés de la imagen tienen una forma geometría repetitiva. Por lo tanto, se puede buscar un modelo probabilístico para explicar la variación de la forma de la estructura y luego cuando se segmenta una imagen se imponen limitaciones para tomar la imagen como el modelo elegido a priori. Esta tarea implica:

1. La selección de los ejemplos de entrenamiento (ejemplos que se usan para probar los modelos).
2. La representación probabilística de la variación de los ejemplos seleccionados.
3. La inferencia estadística entre el modelo y la imagen. El estado del arte para la segmentación basada en el conocimiento implica la forma activa y los modelos de apariencia, contornos activos y una plantilla deformable y métodos basados en niveles.

Segmentación multiescala

Las segmentaciones de la imagen se calculan en múltiples escalas y a veces se propagada de gran escala a pequeña escala. Los criterios de segmentación pueden ser arbitrariamente complejos y se pueden tener en cuenta tanto criterios globales como locales.

Un requisito común es que cada región debe estar conectada en algún sentido. Los métodos de segmentación multiescala más importantes son el de segmentación jerárquica de señales unidimensionales y el de segmentación de imágenes

Segmentación semiautomática

En este tipo de segmentación, el usuario define las regiones de interés con clics del ratón y los algoritmos se aplicarán de forma que se elige el camino que mejor se ajusta al borde de la imagen. Hay bastantes métodos que utilizan este tipo de segmentación interactiva como las tijeras inteligentes o el SIOX

Redes neuronales de segmentación

Las redes neuronales de segmentación se basan en el procesamiento de pequeñas áreas de una imagen utilizando una red neuronal artificial o un conjunto de redes neuronales. Después de este proceso de decisión se construye un mecanismo que marca las áreas de una imagen de acuerdo a la categoría reconocida por la red neuronal.

Un tipo de red diseñada especialmente para esto es el mapa de Kohonen. Las redes neuronales de parejas de pulsos (PCNNs) son modelos neuronales propuesto por

modelos corteza visual de un gato y desarrollado para un alto rendimiento de procesamiento de imágenes biomiméticas.

El modelo de Eckhorn proporciona una herramienta sencilla y eficaz para estudiar la corteza visual de mamíferos pequeños, y pronto fue reconocido por ser una aplicación con un gran potencial en el procesamiento de imágenes. En 1994, el modelo de Eckhorn fue adaptado para ser un algoritmo de procesamiento de imágenes por Johnson, quien calificó este algoritmo como Pulse-Coupled Neural Network.

Aproximadamente desde el año 2000, PCNNs (por sus siglas en inglés) han sido utilizados para una variedad de aplicaciones de procesamiento de imagen entre las que se incluyen entre otras la segmentación de imágenes, la generación de características, la generación de funciones, la extracción de rostros, la detección de movimiento, la detección de regiones en crecimiento y la reducción de ruido.

Un PCNN es una red neuronal de dos dimensiones. Cada neurona en la red corresponde a un píxel en una imagen de entrada, recibiendo la información del color de su correspondiente píxel como un estímulo externo. Cada neurona se conecta con sus neuronas vecinas, recibiendo estímulos locales de ellas. Los estímulos externos y locales se combinan en un sistema de activación interna, que acumula los estímulos hasta que se excede un umbral dinámico, dando como resultado una salida de pulsos. A través de cálculos iterativos, las neuronas PCNN producen series temporales de impulsos de salidas. La serie temporal de impulsos de salidas contiene información de la imagen de entrada y puede ser utilizado para varias aplicaciones de procesamiento de imágenes, tales como la segmentación de la imagen y la generación de características. Comparado con los medios convencionales de procesamiento de imágenes, PCNNs tienen varias ventajas importantes, incluida la robustez frente al ruido, la independencia de las variaciones en los patrones geométricos de entrada, capacidad para pasar por pequeñas variaciones en los patrones de intensidad de entrada, etc.

Segmentación basada en el color

Como ya se ha visto anteriormente, el color se puede suponer como la unión de tres planos, tres canales, cada uno con sus niveles de intensidad en cada punto respecto las componentes de una base de color, normalmente la RGB. Una de las técnicas de segmentación basada en el color consiste en aplicar las técnicas anteriormente estudiadas a cada uno de los tres planos e integrar los resultados en la imagen, para que salga la imagen segmentada en color.

El clustering y la segmentación umbralizada en el modelo de color HSV son las técnicas que más se utilizan para segmentar basándose en el color.

Segmentación basada en la textura

En la segmentación basada en la textura, lo primero que tenemos que hacer es definir modelos de texturas para tratar la imagen como si fuese una función y no una colección de píxeles y así poder transformar una ventana de la imagen en un vector de características.

Los modelos de texturas se dividen en tres categorías, la primera basada en estructuras piramidales, la segunda en campos aleatorios y la tercera está basada en métodos estadísticos.

Dependiendo del número de texturas a reconocer y de si se tiene el conocimiento a priori de éstas, la segmentación basada en la textura estará clasificada en supervisada o no supervisada.

El objetivo de esta segmentación es el de reconocer regiones con un vector de características similares y asignarles la misma etiqueta. Es una segmentación compleja, pues necesita de la medida de la distancia entre esos vectores de características para discernir si son similares o no.

Segmentación basada en el movimiento

La segmentación basada en el movimiento se utiliza para discernir objetos animados de una secuencia de imágenes respecto de fondos estáticos. Esta técnica, conocida como sustracción de fondo, estudia la imagen resultante de la resta de dos imágenes consecutivas de la secuencia y deja los píxeles que se desplazan con valores distintos de cero al restar las imágenes, mientras que los que no se desplazan se mantienen con valores nulos al realizar la resta. Este tipo de segmentación depende muy fuertemente de la iluminación del entorno, pues la sustracción de fondo, al restar dos imágenes, se debe tomar un umbral en el que binarizar la imagen y que dependerá bastante de esa iluminación.

2.1.1.4 Reconocimiento

El reconocimiento es la última fase del sistema de visión artificial. En esta fase, el sistema interpreta nalmente la imagen, la clasifica y toma las decisiones pertinentes que previamente se le ha programado.

El reconocimiento tiene una primera parte de adquisición de datos, una extracción de características compleja y por último la toma de decisiones. Este proceso debe llevar al sistema de reconocimiento a asignar a cada objeto su categoría o clave que es el conjunto de entidades que comparten alguna característica que las diferencia del resto.

El reconocimiento lleva a la clasificación, entendida como la asignación de las diferentes partes del vector de características a grupos o clases, y basándose en las características extraídas, llegar al aprendizaje automático, desarrollando técnicas que permiten a las computadoras aprender, reconocer formas, tamaños, objetos u otras características, obtenidas desde patrones de estudio.

Los algoritmos de aprendizaje se caracterizan por partir de un conjunto de funciones discriminantes con unos valores en sus parámetros que hacen que con se pueda discriminar optimizando las funciones respecto al objeto a reconocer de forma acertada. El proceso termina cuando el conjunto de entrenamiento se clasifica correctamente. Los clasificadores utilizan alguno de estos procedimientos:

Geométricos, con los patrones con características que se pueden obtener mediante gráficos. En éste enfoque se emplea el cálculo de distancias, geometría de formas, vectores numéricos, puntos de atracción.

Estadísticos, que se basan en teorías probabilísticas y de estadística, como análisis de varianzas, covarianzas, dispersión, distribuciones, etc [32].

Los clasificadores, como base del sistema de reconocimiento, tienen una gran aplicación además de la visión artificial, donde se usan para reconocimiento de caracteres, ya sea caracteres escritos a mano o impresos, reconocimiento facial, reconocimiento de objetos y reconocimiento de huellas dactilares, como son:

- Previsión meteorológica: poder clasificar todos los datos meteorológicos según diversos patrones, y con el conocimiento a priori que tenemos de las diferentes situaciones que pueden aparecer nos permite crear mapas de predicción automática.
- Reconocimiento de voz: el análisis de la señal de voz se utiliza actualmente en muchas aplicaciones, un ejemplo claro son los teleoperadores informáticos.
- Aplicaciones en medicina: análisis de biorritmos, detección de irregularidades en imágenes de rayos X, detección de células infectadas, marcas en la piel.
- Interpretación de fotografías aéreas y de satélite: gran utilidad para propuestas militares o civiles, como la agricultura, geología, geografía, planificación urbana.
- Predicción de magnitudes máximas de terremotos.
- Reconocimiento de música: identificar el tipo de música o la canción concreta que suena.

2.2 Control visual

Inicialmente, los sistemas robóticos que integraban visión por computador trabajaban en bucle abierto, lo que se conoce como “ver y mover” (en la literatura anglosajona se referencia como “look and move”). Es decir, en primer lugar ver y reconocer el entorno con ayuda de un sistema de visión por computador, y posteriormente realizar el movimiento en función de la información capturada en la etapa previa. Una de estas primeras aplicaciones de guiado de robots por visión fue la desarrollada por Shirai e Inoue, quienes describen cómo puede utilizarse la realimentación visual para corregir la posición de un robot y aumentar la precisión. Esta primera aplicación consistía en utilizar realimentación visual para el guiado de un robot de forma que realice el agarre de un prisma y su posterior inserción en una caja. Durante esta década se desarrollaron las primeras aplicaciones de control visual para el agarre de objetos en movimiento por cintas transportadoras.

Una alternativa al planteamiento anterior es emplear control en bucle cerrado mediante realimentación de la información visual para controlar la localización del efector final del robot con relación a un objeto, lo que también se conoce en la literatura anglosajona como visual servoing.. Por tanto, el control visual está basado en el uso de información visual en la realimentación del bucle de control. Más concretamente, un sistema de control visual utiliza la información captada de una escena por una o más cámaras conectadas a un sistema de visión por computador, para controlar la localización del efector final del robot con relación a la pieza de trabajo.

En 1980 Sanderson y Weiss establecen una clasificación de los sistemas de control visual, describiendo asimismo el primer sistema de control visual basado en imagen. Desde la aparición de estos primeros sistemas de control visual a principios de los 80, su evolución ha sido lenta, sin embargo en los últimos años han aparecido crecientemente nuevos desarrollos debidos principalmente a las capacidades de procesamiento de los nuevos ordenadores personales que permiten procesar una escena a una frecuencia cada vez mayor.

Ya a finales de los 80 comienza a aplicarse el control visual a muy distintos ámbitos como puede ser el agrícola, en el que se encuentran aplicaciones para la recogida de frutas empleando realimentación visual, o la teleoperación de robots. Dentro de este último ámbito el control visual permite al operador humano especificar la tarea en términos de características en la imagen.

Otro ámbito de interés en control visual es el de la vigilancia. En muchas ocasiones el sistema de visión se localiza sobre una estructura móvil de forma que es capaz de mantener en el campo de visión un determinado objeto que se encuentra bajo vigilancia.

Durante los 90 y con el aumento de las capacidades de los sistemas de visión artificial aparecen un gran número de aplicaciones. Dentro de este nuevo conjunto de aplicaciones cabe mencionar algunas como el guiado de vehículos en las que se requiere una gran capacidad de procesamiento y una alta frecuencia en la realimentación visual. Otro ámbito en el que el control visual presenta una importancia creciente en la actualidad es el del guiado de robots aéreos. Aplicaciones dentro de este ámbito no han sido explotadas en gran medida, sin embargo, ya empiezan a surgir vehículos aéreos, como helicópteros, dotados de un sistema de visión que realimenta al controlador del vehículo de forma que permite realizar su guiado a partir de características del entorno, evitando obstáculos e incluso permitiendo realizar un aterrizaje automático. El desarrollo de sistemas de control visual en estos vehículos permite realizar aplicaciones como vigilancia, seguimiento, inspección, trabajo en lugares peligrosos, etc.

Con la miniaturización y el aumento de precisión de los componentes de los robots, van aumentando las posibilidades de la micromanipulación empleando realimentación visual procedente, por ejemplo, de microscopios. Dentro de este ámbito cabe destacar que cuando se observan microorganismos, los elementos desaparecen del campo de visión muy rápidamente. Se vienen aplicando en la actualidad sistemas de control visual con el objetivo de mantener en la imagen a lo

largo del tiempo ciertos microorganismos con el propósito de su posterior manipulación.

Esta mejora en el desempeño de los sistemas robóticos y de visión artificial ha propiciado la aparición de aplicaciones a robots submarinos. En este caso se dispone de vehículos submarinos guiados por visión o sistemas de manipulación submarina cuya tarea es guiada mediante realimentación visual.

Un ámbito de aplicación con mayor impacto social son las aplicaciones médicas. La alta precisión alcanzada con estos sistemas ha propiciado la aparición de sistemas para la realización de telecirugía así como robots que realizan tareas quirúrgicas de forma autónoma. A pesar de que estos sistemas aumentan el coste por intervención, la alta precisión junto con el pequeño espacio de trabajo requerido en, por ejemplo, cirugía cerebral, hace que estos sistemas se apliquen con éxito. Dentro de este ámbito y volviendo a las aplicaciones de guiado de vehículos son de destacar las investigaciones para aplicación de control visual para el guiado de sillas de ruedas para discapacitados.

En la actualidad, las mejoras de los tiempos de procesamiento de los ordenadores han permitido a las aplicaciones de robótica basadas en visión desarrollarse en tiempo real. Esto ha hecho posible incluir las características extraídas de las imágenes en un bucle de control. De hecho, en la actualidad encontramos aplicaciones muy diversas en el ámbito del control visual como puede ser posicionamiento preciso de un robot, control remoto o realimentación para teleoperación, micromanipulación, agarre de objetos en movimiento, sistemas de posicionamiento de un avión sobre la pista de aterrizaje, cámaras de seguimiento de misiles, recolección de frutas, robots que juegan al ping-pong, etc.

2.2.1 Arquitectura del sistema de visión

Uno de los factores a los que se debe el auge actual en los sistemas de control visual es el desarrollo de nuevos sistemas de visión. Estos nuevos sistemas han

propiciado que la información visual sea cada vez más utilizada, debido a la disminución de retardos de captura y procesamiento, lo que a su vez ha permitido el desarrollo de sistemas basados en distintas arquitecturas. En este apartado se describen las configuraciones más utilizadas para la ubicación de cámaras (una o varias) respecto al robot y al espacio de trabajo donde se desarrolla la tarea.

Supóngase una aplicación en la que un robot deba realizar el agarre de un objeto en movimiento. Los componentes más importantes en un sistema de control visual para una aplicación de seguimiento de este tipo son los mostrados en la Figura 17.

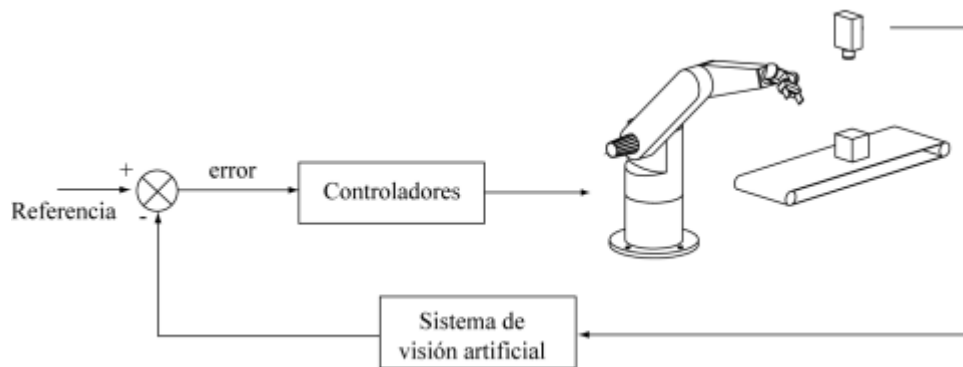


Figura 17: Componentes principales de un sistema de control visual

Se representa un esquema genérico en el que se observa el bloque denominado “sistema de visión artificial” que es el encargado de procesar las imágenes y realimentar constantemente al sistema con esta información. La entrada al sistema de control es una referencia, de manera que el error que anula el controlador es la diferencia entre el valor actual y el deseado. Este controlador se encarga de dirigir el manipulador hacia la pieza que se está siguiendo.

Es posible ubicar la cámara en diferentes posiciones. En muchas ocasiones se monta en el extremo del robot, en este caso observa el entorno en el que se realiza la tarea. Sin embargo, para la realización de otras tareas puede ser necesario que la cámara observe tanto al robot como a su entorno. En este último caso se suele utilizar un sistema de cámara fija o montada en un segundo robot.

Independientemente del tipo de configuración empleada, es necesario realizar la calibración de la cámara para determinar sus parámetros intrínsecos como distancia focal, precisión de píxel y punto principal. Por lo tanto, se puede afirmar que los sistemas de servovisión usan típicamente dos configuraciones de cámara: montada en el efector final del robot, o externa al robot.

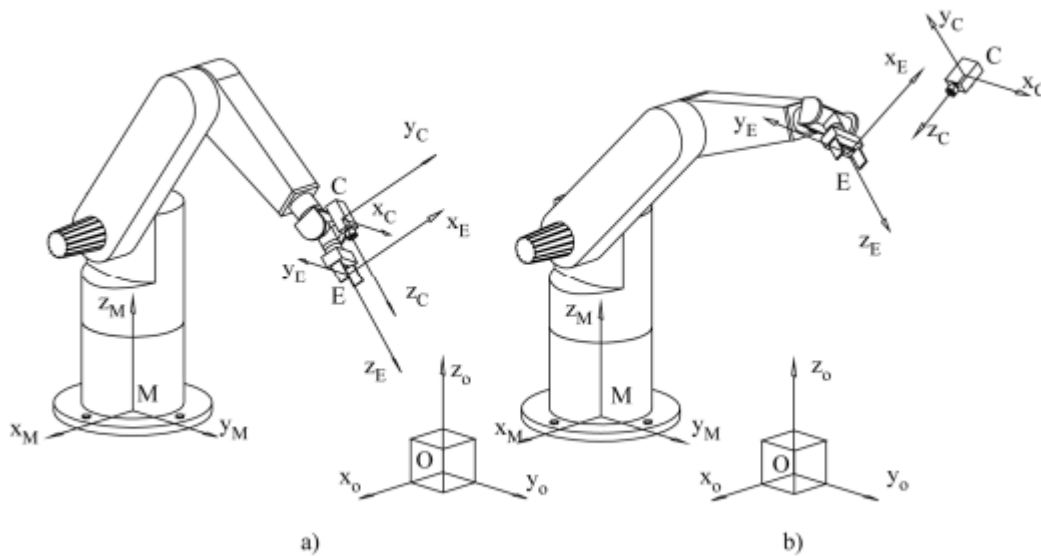


Figura 18: Arquitecturas Eye-in-hand(izquierda) y Eye-to-hand (derecha)

Cámara externa al robot (Eye- to-hand)

En este caso la cámara no tiene una relación mecánica con el robot que está siendo controlado visualmente, pero es conocida la relación entre la cámara y el sistema de coordenadas asociado a la base del robot. En principio, la vista obtenida es fija, salvo que la cámara esté ubicada sobre una estructura móvil. Una variante consiste en ubicar la cámara en el extremo de otro robot o bien emplear un cabezal con movimientos pan-tilt, con el propósito de conseguir visualizar el robot de la mejor forma. La cámara se posiciona de manera que el elemento terminal y los objetos situados en el espacio de trabajo del robot se encuentren en su campo de visión. Con esta configuración la información visual va a permitir obtener la localización tanto de un objeto situado en el espacio de trabajo como la del extremo del robot.

En la Figura 2-2.b se muestran los sistemas de coordenadas principales utilizados en un sistema de control visual en el que el sistema de visión está compuesto por una única cámara fija observando el espacio de trabajo y el extremo del robot. La tarea a realizar se va a indicar respecto al sistema de coordenadas del mundo o de referencia, M , situado en la base del robot. Se conoce la relación entre los sistemas de coordenadas de manera que se pueda comparar la localización del extremo del robot y la del objetivo a alcanzar utilizando un mismo sistema de coordenadas de referencia.

En el plano de la imagen de la cámara fija se obtiene una representación 2-D del espacio de trabajo a partir de la cual se extraen las características necesarias para el desarrollo de la tarea (posición de un objeto, su tamaño, centroide...). Estas características están expresadas respecto al sistema de la cámara, C , para referirlas respecto al sistema de coordenadas del Mundo, M , se necesitará disponer de la información acerca de la localización relativa entre la cámara y la base del robot, MT_c , que permita realizar la transformación de coordenadas.

Por otro lado, en una aplicación de control visual en la que el robot deba interactuar con los objetos del espacio de trabajo, será necesario disponer de información acerca de la localización del extremo del robot referida al sistema de coordenadas de referencia, M . Esta información podrá extraerse igualmente a partir de la información visual procedente del sistema de visión, realizando la suposición de que el sistema de visión capta información tanto del espacio de trabajo como del extremo del robot. Con esto ya se dispondrá de información relativa al robot y al espacio de trabajo referida a un único sistema de coordenadas, el del mundo, M . En ocasiones el número de cámaras puede ser mayor que uno para realizar una reconstrucción geométrica del entorno más fiable (Figura 19).

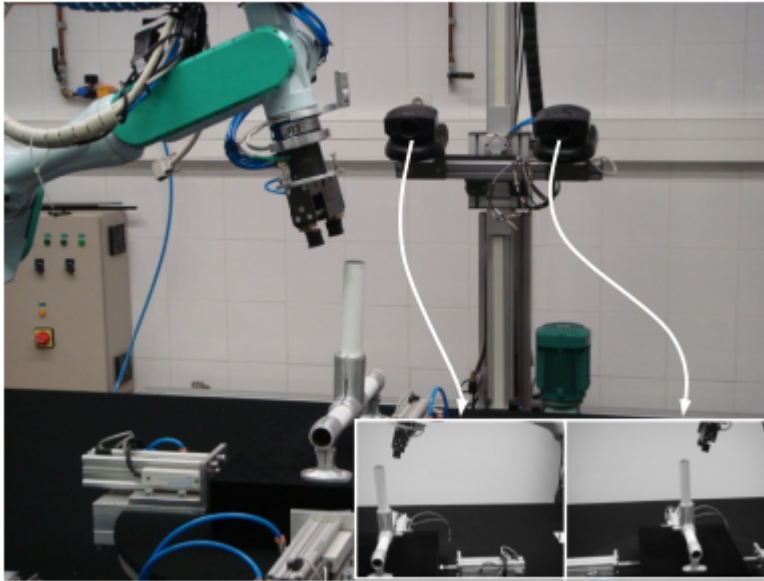


Figura 19: Ejemplo de arquitectura Eye-to-hand

Cámara en el extremo del robot(Eye-in-hand)

En esta solución, la cámara se encuentra solidaria al extremo del robot, de manera que los objetos del espacio de trabajo se encuentran dentro de su campo visual. En este caso el sistema de visión se emplea para determinar la localización de un objeto con respecto al extremo del robot (Figura 20).

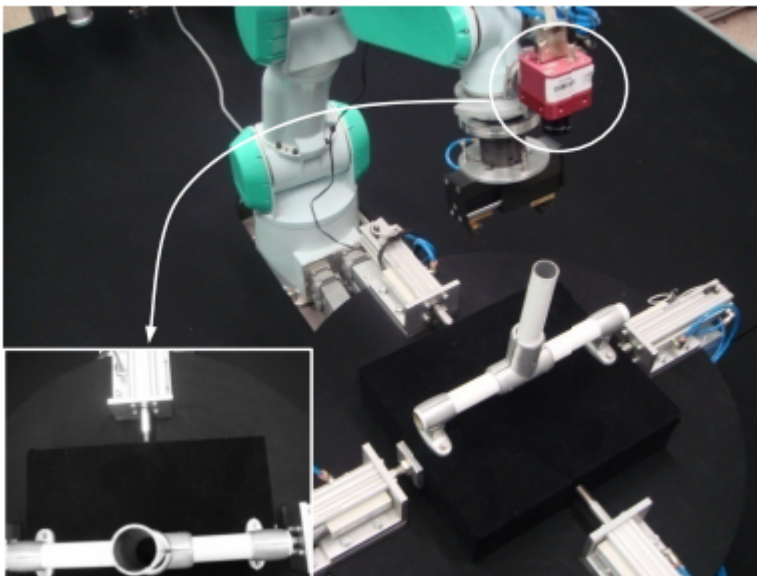


Figura 20: Ejemplo de arquitectura Eye-in-hand

2.2.2 Clasificación de los sistemas de control visual

En este apartado se va a mostrar una clasificación de los sistemas de control visual, en primer lugar atendiendo al tipo de arquitectura y enfoque del problema de control visual, posteriormente teniendo en cuenta la utilización de las características extraídas de la imagen y por último, se clasifican estos sistemas considerando el área observada por la cámara.

Ver y mover estático

Inicialmente, los sistemas robóticos que integraban visión por computador trabajaban en bucle abierto, en una configuración secuencial, primero ver y reconocer, y a continuación mover. Obviamente, la precisión de la operación a realizar depende directamente de la precisión del sensor de visión y del efector final del robot. El robot ejecuta las tareas suponiendo que ningún objeto en el entorno se modifica desde que se realiza la captura de la imagen hasta que se lleva a cabo la tarea. En esta arquitectura la única realimentación presente es la de los bucles internos de control de los servomotores (Figura 21).

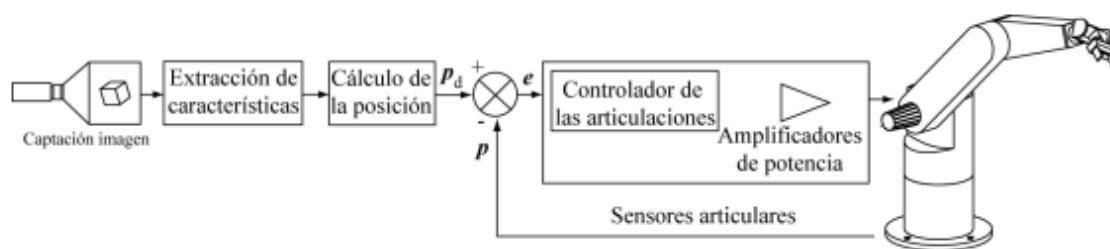


Figura 21: “Ver y mover” estático.

Supóngase una aplicación en la que un robot deba alcanzar la posición en la que se encuentra un objeto del espacio de trabajo. Para determinar la posición del objeto se utiliza un sistema de visión que indica en qué lugar se encuentra el objeto a alcanzar. Utilizando una estrategia de “ver y mover” estático, el robot se dirigiría a continuación hacia la posición indicada por el sistema de visión haciendo la suposición de que una vez alcanzada esa posición ya se habrá asimismo alcanzado

el objeto. Por lo tanto, no se comprueba, ni durante la trayectoria del robot ni una vez alcanzada la posición final, que efectivamente se haya alcanzado al objeto, con lo que se supone que, desde que se obtuvo la posición con el sistema de visión hasta que se alcanza la posición final, no se ha alterado la posición del objeto.

Ver y mover dinámico

Sobre el sistema “ver y mover” anterior, que se podría decir que es estático, se puede añadir una realimentación visual obteniendo otro tipo de sistema de control visual denominado “ver y mover” dinámico. En este tipo de arquitectura el sistema de visión es usado en un primer bucle de realimentación; existe un segundo lazo interno donde se realimenta la información articular (con una velocidad de muestreo mayor). Se realiza lo que se denomina un control visual indirecto, y frente al “ver y mover” estático, permite realimentarse visualmente mientras el robot se encuentra realizando la tarea (Figura 22).

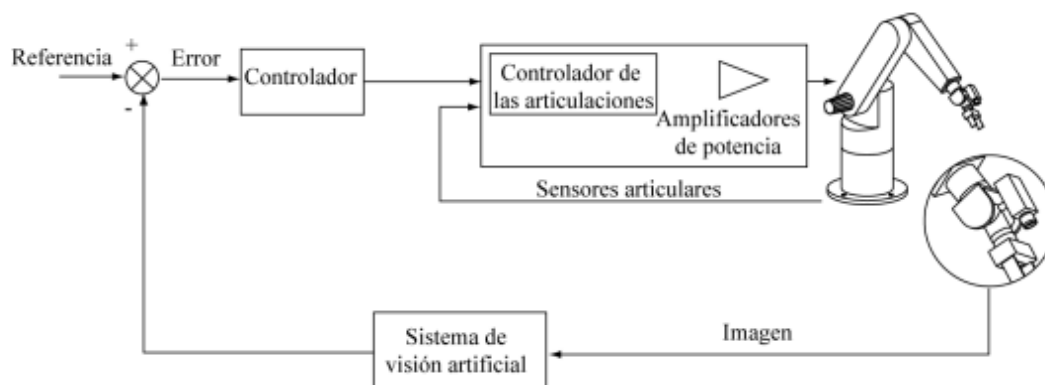


Figura 22: “Ver y mover” dinámico

Si se empleara una estrategia de tipo “ver y mover” dinámico, el sistema se realimentaría constantemente con la información visual. Esta continua realimentación permitiría al sistema corregir posibles errores en la posición del objetivo determinada en instantes anteriores o modificar la trayectoria del robot ante posibles movimientos o alteraciones en los objetos del espacio de trabajo.

Control visual directo

Una alternativa al planteamiento anterior es emplear control en bucle cerrado mediante realimentación de la información visual para controlar la localización del efector final del robot con relación a un objeto, lo que también se conoce como control visual directo. En este caso no se requiere el bucle de control a nivel articular, presentando sólo el bucle externo de control basado en la información visual, en el cual se calculan las consignas para cada una de las articulaciones del robot, esquema en la Figura 23.

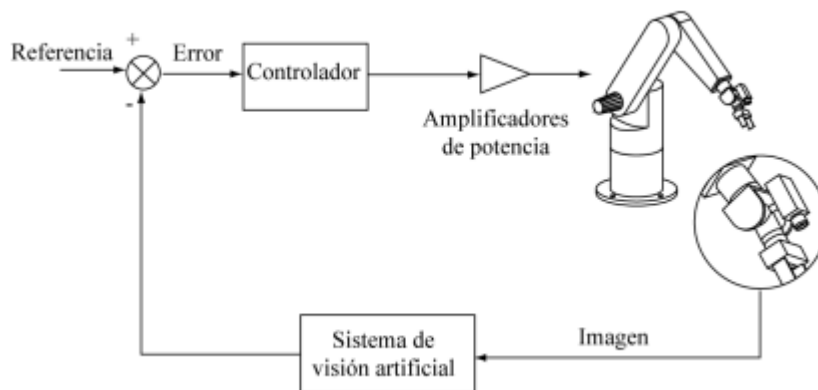


Figura 23: Control visual directo

Si se utilizara un control visual directo, al igual que en el caso de “ver y mover” dinámico el sistema se realimentará constantemente con información visual del entorno; sin embargo, en este caso no estará presente el controlador del robot. Este último regulador será sustituido por el controlador visual que utiliza información procedente del sistema de visión directamente para regular y estabilizar al sistema.

Control basado en posición y basado en imagen

Se suele introducir una segunda clasificación [Sanderson-1980] de los sistemas de control visual que se realiza dependiendo si el control se ejecuta basado en posición o basado en características extraídas de las imágenes. En los sistemas de control basados en posición se dispone de un modelo del espacio de trabajo de forma que, a partir de las características extraídas del sistema de visión, se estima la localización del objeto a alcanzar respecto al sistema de coordenadas de la cámara. Esta localización estimada se compara a su vez con la deseada o de referencia; la

diferencia entre ambas localizaciones será la entrada al regulador. A su vez, teniendo en cuenta la clasificación dada en los apartados anteriores, se pueden establecer las siguientes configuraciones para los sistemas de control visual basados en posición:

“Ver y mover” estático basado en posición. En el que se observa que las características extraídas son traducidas a una localización, p_d , en la que se desea que se posicione el robot.

“Ver y mover” dinámico basado en posición. En la realimentación del bucle de control se extraen las características observadas en la imagen a partir de las cuales se estima la localización del objetivo a alcanzar, C_p (posición), C_ϕ (orientación). La entrada de referencia al bucle de control es la localización deseada del robot, C_{dp} , $C_{d\phi}$. La diferencia entre ambas posiciones, referidas a un sistema común de referencia, es la entrada al regulador que se encargará de ir disminuyendo progresivamente esta distancia (Figura 24).

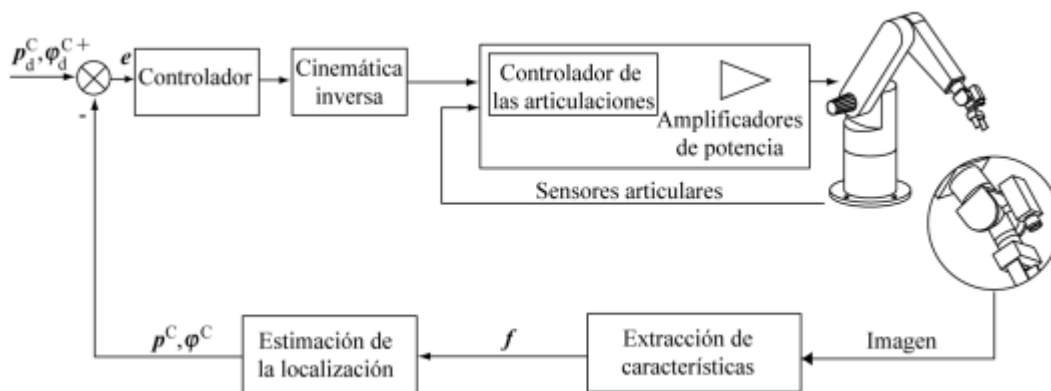


Figura 24: “Ver y mover” dinámico basado en posición

Se observa, por lo tanto, la necesidad en este tipo de sistemas de estimar la localización.

Control visual directo basado en posición. Este esquema, mostrado, utiliza al igual que en el caso anterior, un modelo del objetivo a alcanzar para determinar su

localización a partir de las características observadas por el sistema de visión. La entrada de referencia también será la localización deseada del robot. Sin embargo, en este caso el regulador se encargará de realizar la estabilización del robot (Figura 25).

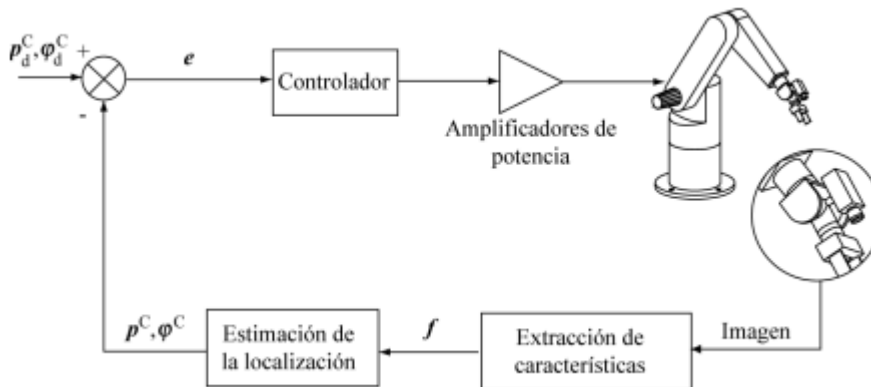


Figura 25: Control visual directo basado en posición

Por el contrario, en los sistemas de control basados en imagen, el control se realiza directamente a partir de las características extraídas por el sistema de visión, que representan la proyección del objetivo a alcanzar en el plano imagen. De esta manera, la entrada al regulador será una comparación entre las características observadas y las deseadas o de referencia.

Sistemas finales de bucle abierto y bucle cerrado

Por otra parte, según el área final observada por la cámara se pueden distinguir los sistemas de control visual en:

- Sistemas finales de bucle abierto o EOL (denominados en la literatura anglosajona como “end point open loop”). En ellos la cámara observa únicamente el objeto destino y no se obtiene información visual de la posición actual del robot. De esta manera, se determina y controla la posición del extremo del robot exclusivamente con una combinación de conocimiento interno de la posición del extremo del robot a partir de su cinemática y de la calibración cámara-objeto. Bajo estas consideraciones, la precisión en el

posicionamiento del robot depende de la precisión de la calibración y de la exactitud en la reconstrucción de la escena.

- Sistemas finales de bucle cerrado o ECL (denominados en la literatura anglosajona como “end point closed loop”). En ellos la cámara observa el objeto destino y el manipulador final del robot.

Un sistema EOL se puede convertir en un sistema ECL incluyendo la observación del efector final del manipulador. Aunque teóricamente parece preferible un sistema ECL a un EOL, la necesidad de seguir al efector final además del objeto destino, conlleva a que los problemas de visión tengan más peso en el diseño del controlador de un ECL, y a que los requerimientos de campo visual no puedan ser satisfechos siempre [33].

2.3 Robots Hexápodos

En cuanto a robots hexápodos, uno de los motivos del desarrollo de los robots caminantes es que pueden salvar obstáculos mayores que sus equivalentes con ruedas, esto es verdad hasta cierto punto, ya que estos últimos se pueden fabricar más grandes de manera que puedan solventarlos y tienen la ventaja de ser más simples que ellos. Sin embargo, los robots andantes gozan de mayor grado de movilidad y pueden operar en zonas estrechas e irregulares, donde no es posible usar un robot de mayores dimensiones, o de transportarlos hasta dicha zona. A medida que las condiciones de trabajo y los entornos se vuelven cada vez más peligrosos, es extremadamente conveniente eliminar al humano de estas situaciones potencialmente peligrosas y reemplazarlo con un robot diseñado específicamente para realizar una tarea determinada. Ejemplos de estas zonas son: Cuevas, minas, desastres naturales, bosques, barrancos, otros planetas etc... Los robots hexápodos son perfectos para estos lugares debido a las ventajas en su movilidad.

Una de las ventajas clave que gozan los robots andantes sobre los demás es que pueden moverse en cualquier dirección y girar en cualquier ángulo; esta capacidad de movimiento omnidireccional es la que premia a este tipo de robots frente al resto como se puede ver en la Figura 26.

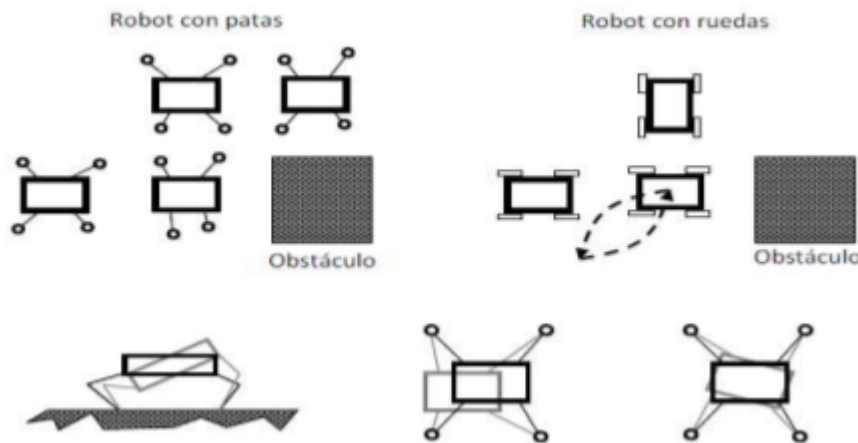


Figura 26: Demostración de movimiento entre robot con patas y andante.

En la actualidad, se encuentran robots hexápodos de todo tipo, los cuáles se pueden clasificar de muchas maneras, dependiendo del tipo de desempeño que vayan a realizar, del material del que estén fabricados, de su estructura, circular hexagonal, rectangular, etc... También dependiendo del tipo de actuadores que posean, o si son de software libre o cerrado, etc. Se pueden encontrar numerosos robots con distintas características. Sin embargo, el factor determinante es el desempeño que vaya a realizar el robot, ya que no se utilizaran los mismos materiales o actuadores para un robot de tipo hobby, que un robot que se utilice para el rescate en un desastre natural.

Con el avance actual de la robótica es normal encontrar soluciones sencillas que pueden ser fácilmente reproducibles por un determinado público. De esta idea surgen proyectos denominados DIY (do it yourself = hazlo por tí mismo). Destacar que en este ámbito puede haber personas que no sean profesionales del tema y solo quieran aprender (amateurs) o, por otra parte, personas con estudios más

específicos o profesionales que quieran enfrentarse a un reto, mejorar sus habilidades o poner en práctica sus conocimientos. Hay un gran abanico de materiales como plástico, fibra de vidrio, fibra de carbono o metal y actuadores tanto eléctricos, como hidráulicos o mecánicos, así como multitud de diseños y formas. Para el marco del hobbyista, el software suele ser libre para que puedas aprender de él, aunque no todas las plataformas ofrecen el mismo soporte en el que te explican cómo funciona el proyecto. Cabe destacar, como experiencia, que cuanto más complejo es el proyecto menos soporte hay detrás, aunque el software sea libre, ya que no añaden ninguna explicación del funcionamiento, más allá de alguna guía de montaje y uso. Aquí tenemos algunos modelos que parten de los 40 euros, como el KIKO.893, hasta más de los 1000 euros como el PhantomX AX Metal Hexapod Mark III, los cuales se pueden ver en la Figura 27.

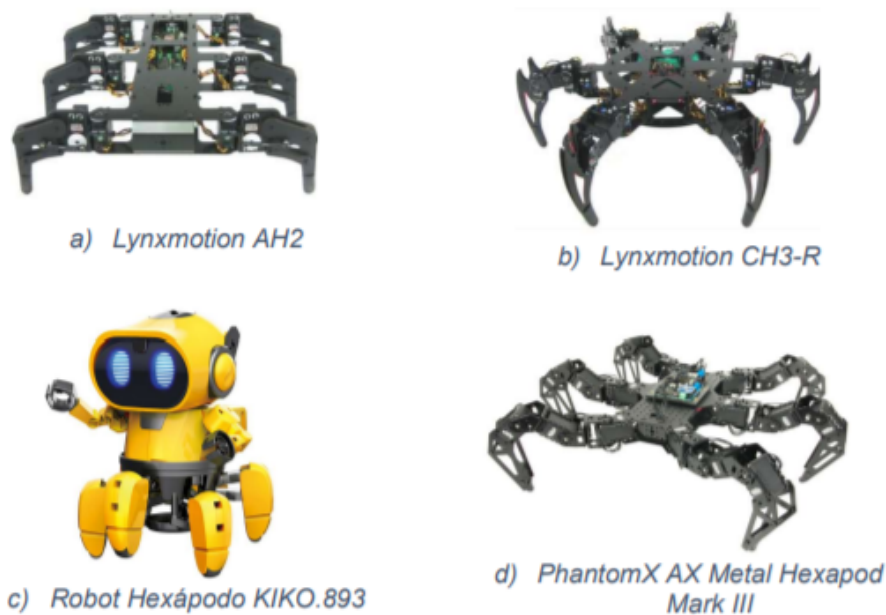


Figura 27: Distintos modelos de robots para hobbyistas.

Existen empresas o plataformas que apoyan la enseñanza de la robótica y/o la electrónica con ejemplos de robots hexápodos. Un ejemplo de estas empresas es Arbotics, la cual promueve la robótica con el robot hexápodo Hexy de una manera sencilla, proporcionando guías no sólo de montaje de piezas, sino también de montaje electrónico e incluso de programación. Arbotics también realiza charlas

explicativas sobre el funcionamiento de Hexy, promoviendo la robótica y la electrónica.

Como se ve, Hexy (Figura 28) es un robot básico formado por un sensor de ultrasonidos, 18 servomotores y piezas de plástico. Esto lo convierte en una solución muy accesible y barata para dar a conocer la robótica.



Figura 28: Robot Hexy desde el frente

Vorpal robotics es otra empresa que con su robot hexápodo Vorpal (Figura 29) proporciona similares características que el robot Hexy y además proporciona los archivos para imprimir en 3D las piezas que componen el robot, lo cual lo hace muy atractivo.



Figura 29: Robot Vorpal desde el frente

Además cabe destacar que este robot se mostró en El Hormiguero, un programa de la televisión española, donde mostraron su movimiento y algunos principios de su funcionamiento.

Con el surgimiento de la inteligencia artificial, hay grupos de desarrollo que están intentando aplicar el machine learning para mejorar la forma de andar del robot y que ésta se adapte según el terreno en el que se encuentre o a la pérdida de alguna extremidad. No obstante, estas investigaciones todavía están lejos de dar un resultado satisfactorio, por lo que todavía hay mucho que desarrollar en este ámbito.

2.4 Navegación en robots móviles

La navegación autónoma puede definirse “como la capacidad de un vehículo (robot) para desplazarse de una manera automática desde una posición inicial a otra final”. Desde la perspectiva humana, se cree que la navegación autónoma de los robots se fundamenta en la realización de un mapa que debe de seguir el robot desde un punto inicial a otro final. Sin embargo, es posible que los robots realicen dicha tarea sin la necesidad de poseer de antemano un mapa descriptivo del entorno o elaborarlo a medida que el dispositivo se va desplazando, ya que existe la alternativa de la navegación reactiva y cómo el robot es capaz de reaccionar directamente con el entorno para llevar a cabo el movimiento del mismo. A continuación, se va a proceder a explicar que se entiende por navegación reactiva y algunos de sus casos de uso.

A continuación, se va a proceder a explicar que se entiende por navegación reactiva y algunos de sus casos de uso.

2.4.1 Navegación reactiva

Como se ha explicado con anterioridad la navegación reactiva consta básicamente de dos etapas: una primera etapa de recopilación de información por parte de los sensores y otra etapa de actuación, la cual siempre va a depender de la información que haya sido recogida en la anterior etapa.

Un robot puede llevar a cabo una tarea compleja utilizando esta metodología y sin necesidad de tener un mapa y sin tener en cuenta donde se encuentra, solo sería

necesario superponer varias tareas de sentir-actuar (módulos) para poder llevar a cabo dicha tarea.

Su funcionalidad establece un flujo lineal de información entre los sensores y los actuadores, lo cual da lugar a una serie de ventajas como son:

- Reducción del tiempo de respuesta de cada módulo, lo que repercute en un tiempo de respuesta más rápido de todo el sistema completo.
- Mayor robustez a los errores y mayor facilidad de diseño de los módulos debido a que estos son independientes entre sí, lo que dota al sistema de una mayor estabilidad.

Hay una gran cantidad de robots que incorporan este tipo de sistema de navegación ya que, es el sistema ideal para sistemas que quieran gozar de una cierta flexibilidad a la hora de navegar ante entornos cambiantes y desconocidos, pues ofrece una elevada velocidad de respuesta con una capacidad computacional baja. (La carga computacional va a depender de la complejidad de la tarea a desarrollar).

Las tortugas “Elsie y Elmer” de William Grey Walter (Figura 30), de este modo fueron apodados los robots creados por William Grey Walter (1910-1977), un neurofisiológico americano, y este fue uno de los resultados entre varios robots que construyó para poder estudiar el comportamiento existente entre las interconexiones neuronales.

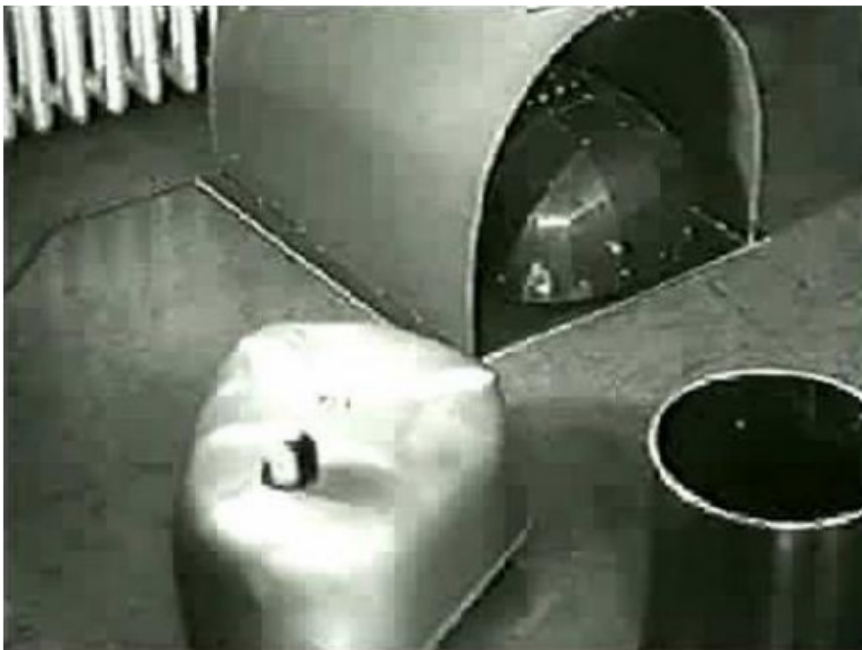


Figura 30: Imagen de las tortugas robóticas Elsie y Elmer

Las tortugas estaban formadas, cada una de ellas, por tres ruedas, dos receptores: uno sensible a la luz y otro al contacto; y un par de motores: uno para avanzar y otro para girar. Eran apodadas como tortugas debido a la forma de su chasis de plástico con forma de caparazón. Estas eran capaces de moverse hacia una fuente de luz, no intensa, si se encontraba con un nivel de batería óptimo. También tenían capacidad para evitar obstáculos gracias a su sensor de contacto, siempre y cuando, se encontraran en zonas donde la intensidad de luz no fuera muy baja, ya que, de lo contrario, el robot no accedería. En el caso de que se encontrara con un nivel bajo de batería, las tortugas estaban diseñada para buscar niveles de intensidad de luz intensa que, casualmente, estaban ligados a la “madriguera” del robot.

Valentino Braitenberg es un neurocientífico y cibernético austriaco-italiano que, en su libro “Vehículos: Experimentos en psicología sintética” (1986), describió el concepto de lo que actualmente se conoce como vehículos Braitenberg. Estos, se caracterizan por tener una combinación de sensores y actuadores en conexión directa, de manera, que son capaces de lograr objetivos.

Un claro ejemplo de este tipo de vehículos sería el descrito anteriormente, es decir, las tortugas “Elsie y Elmer”, aunque estas tienen la particularidad de que se crearon antes de que surgiera este término.

Un ejemplo actualizado de este tipo sería el famoso robot EV3 (Figura 31) de la compañía de Lego. Este posee un gran abanico de sensores que, junto con su software, permite un gran número de rutinas y tareas que se pueden ejecutar de manera autónoma.



Figura 31: Robot EV3 de Lego

El Roomba (Figura 32) es un robot aspirador fabricado y comercializado por la empresa iRobot. Es un ejemplo actual y que ha tenido un gran éxito a nivel internacional, reflejo de ello es el hecho de que se han vendido más de 5 millones de aspiradoras teniendo en cuenta las distintas versiones, del mismo, que han ido surgiendo.

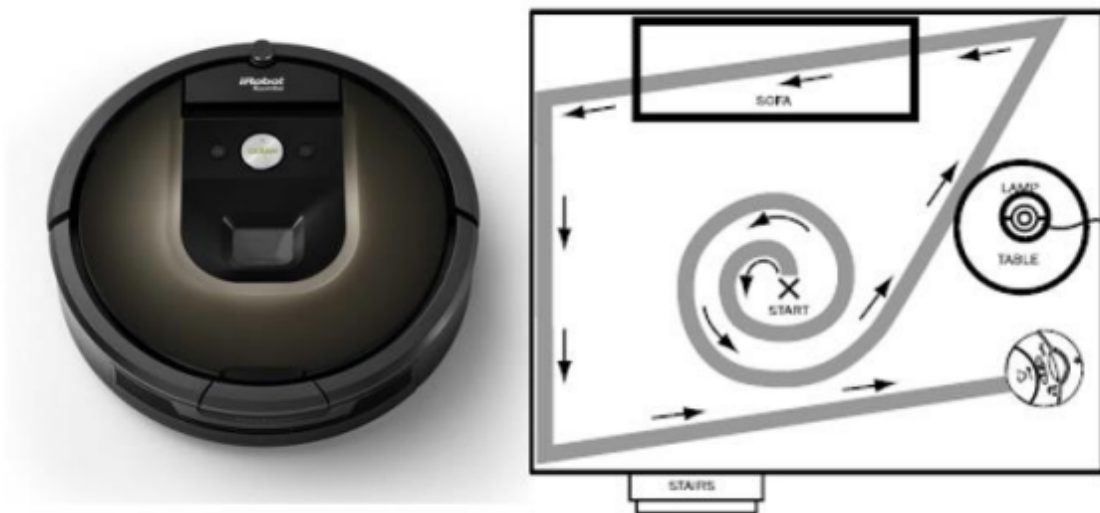


Figura 32: Robot Roomba y ejemplo de trayectoria.

Este dispositivo está formado por una serie de sensores de contacto e infrarrojos y dos ruedas independientes, las cuales, le permiten efectuar giros de 360°. Para su funcionamiento no sigue ningún mapa de las habitaciones donde opera, ya que resultaría imposible almacenar tanta cantidad de información en un robot de su tamaño. Su patrón de movimiento estándar (ya que dependiendo de la versión del dispositivo o los distintos modos de operación esto puede variar), se ejecuta en forma de espiral y a medida que va detectando, por ejemplo, diferencias de nivel debido a la existencia de una escalera, o una pared u obstáculos los cuales han sido detectados a su sensor de contacto, la trayectoria se irá modificando.

Además, se puede programar rutinas para en el caso de que detecte pared, el robot las vaya siguiendo hasta que haya pasado por todas las paredes que componen la habitación.

2.4.2 Navegación autónoma basada en visión

El algoritmo de navegación debe elaborar los comandos de movimiento del robot para que este se pueda desplazar por el entorno donde se encuentra. Para poder llevar a cabo dicho algoritmo de navegación autónoma, se suele dividir los problemas que presenta en tres cuestiones:

- Localización: ¿Dónde estoy?
- Planificación de las tareas: ¿Hacia dónde quiero ir?
- Planificación del movimiento: ¿Cómo puedo llegar ahí?

El problema de la localización consiste en que el robot debe tener la capacidad de saber, en todo momento, donde se encuentra respecto a un sistema que tome como referencia.

La planificación de las tareas, por su parte, se encarga del procedimiento a seguir para que se realicen unas determinadas acciones y, de esta forma, llegar a cumplir el objetivo marcado en un principio. Dicho objetivo puede ser la posición final a la que se quiere que llegue el robot. Así mismo, la planificación del movimiento se refiere a la generación automática de acciones para poder llevar a cabo las tareas planificadas anteriormente, por ejemplo, las acciones para esquivar los obstáculos que se encuentre el dispositivo antes de llegar a su posición final.

Antes de entrar en la navegación de robots basada en visión es importante que se entienda visión artificial o visión por computador como “la disciplina que estudia cómo reconstruir, interpretar y entender una escena en tres dimensiones a partir de imágenes en dos dimensiones siendo el objetivo último, de esto, crear un modelo que exceda las limitaciones de la visión humana usando un software y hardware determinados”. Resulta de gran importancia en este trabajo de fin de grado, ya que el único sensor empleado para adquirir información del entorno va a ser de este tipo.

En la navegación de robots basada en visión podemos distinguir dos ramas de aplicación diferentes, como son: la navegación basada en exteriores y la navegación basada en interiores. Como se ha indicado anteriormente, el presente TFG está enfocado a un sistema de navegación basado en interiores.

Dentro de los sistemas de navegación de interiores se puede tener tres posibles casos:

- Aproximaciones basadas en mapas: este tipo de navegación, consiste en proporcionar al robot un modelo o mapa topológico del entorno, a partir del cual, el sistema es capaz de construir un mapa.
- Navegación basada en la construcción de mapas: En este tipo de navegación, los robots exploran su entorno por medio de los sensores que estos tengan y, a través de ello, son capaces de construir una representación interna del mismo.

- **Navegación sin mapas:** En este caso, la navegación de este tipo de sistemas se consigue sin ninguna descripción del entorno. Los movimientos del robot son producidos por la extracción de la información de los elementos que se encuentran en el entorno.

2.5 Actualidad

Utilizar la visión por computador para dotar de visión a distintos sistemas es algo que se lleva estudiando desde 1960 donde Larry Roberts hizo una tesis donde discutía las posibilidades de extraer información geométrica en 3D a partir de imágenes en 2D, sin darse cuenta, con esa tesis se convirtió en el padre de la visión por computador, aunque siendo una tecnología que lleva tantos años con nosotros, hoy en día se sigue utilizando para multitud de campos. Un buen ejemplo que llegó a todo el mundo y que despertó una gran curiosidad por este campo fue el robot Spot de Boston Dynamics, este cuadrúpedo se está utilizando desde exploración en ambientes extremos [6], hasta para ayudar a combatir el COVID-19 [7].

Más novedoso, pero esta vez aprovechando la microgravedad, se tiene al Astrobee que es un nuevo sistema robótico de vuelo libre para realizar trabajos de Actividad Intravehicular (IVA) en la Estación Espacial Internacional (ISS). El sistema Astrobee incluye tres robots de vuelo libre, un muelle (para recargar la energía eléctrica y transferir grandes archivos de datos), y un sistema de datos en tierra que se utiliza para comunicación, control y transferencia de datos [8]. Estos robots utilizan marcas ArUco[19] para localizarse en el espacio, lo cual da confianza en que las tecnologías que se están utilizando son punteras y se están trabajando ahora mismo con ellas en entornos de alta tecnología.

Otro trabajo que utiliza visión por computador, es el rover Perseverance, el cual está realizando una misión para obtener muestras en Marte, este rover precisa de visión por computador desde su entrada en Marte, ya que, para poder aterrizar, necesitaba localizar mediante cámaras su posición y aterrizar en el lugar designado. El rover cuenta con 23 cámaras a bordo, algunas de ellas solamente sirven para mostrar

imágenes de Marte [11], pero las otras sirven, como he dicho anteriormente para utilizar la visión por computador.

Algo que se tiene más a bajo nivel son los coches autónomos, estos coches empezaron a conocerse por el 2012, y fue aquí en España donde se desarrolló uno de los pioneros y que más resultados favorables tenían. El desarrollo de este tipo de sistemas está motivado principalmente por el aumento del número de vehículos en las carreteras, que exige mejoras en términos de seguridad [9]. Ahora mismo el máximo exponente de estos vehículos autónomos viene de la mano de la empresa Tesla Motors, la cual ha desarrollado un piloto automático, que por ley tiene que ser semi automático, ya que el piloto tiene que tener en todo momento las manos sobre el volante, este piloto semi automático te da más confianza al volante, aumenta tu seguridad en la carretera y hace que la conducción en carretera sea más agradable. Aunque los coches sin conductor están todavía a unos años de distancia, el piloto semi automático de Tesla funciona de forma muy parecida a los sistemas que utilizan los pilotos de avión cuando las condiciones son claras [10].

Todos estos trabajos, cuentan con auto navegación gracias al control por visión, la cual se vale de la información que le llega por la cámara para localizarse y saber cómo actuar dependiendo de su situación.

3. Tecnologías software que se van a desarrollar

Para poder conformar el proyecto se necesita juntar las tecnologías mencionadas anteriormente, pero, a parte de eso, hay que especificar cómo se va a utilizar esas tecnologías, las cuales no funcionan por sí mismas, hay que incluir una funcionalidad para que nos sean útiles. Para ello hay que tener claro qué funcionalidad vamos a darle y cómo comunicar las distintas tecnologías entre sí.

3.1 Control remoto mediante servicio API/REST

En la actualidad existe una gran cantidad de proyectos o aplicaciones que disponen de una API REST, una nueva opción o estilo de uso de los Servicios Web (Web Services, WS), para la creación de servicios profesionales. Twitter, YouTube, los sistemas de identificación con Facebook, y cientos de empresas generan negocio gracias a REST y las APIs REST. REST es una interfaz entre sistemas que usa HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML, JSON, HTTP, etc. En los últimos años logró un gran impacto en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de utilizar y sobre todo por su eficiencia [14].

Pero cuando se piensa en servicios API/REST, se piensa en servidores, dentro de una sala refrigerada y esperando a que llegue la petición para devolver los datos que se piden como se ve en la Figura 33, en este trabajo se le da una vuelta de hoja y se va a hacer que el servidor que provee el servicio sea móvil.

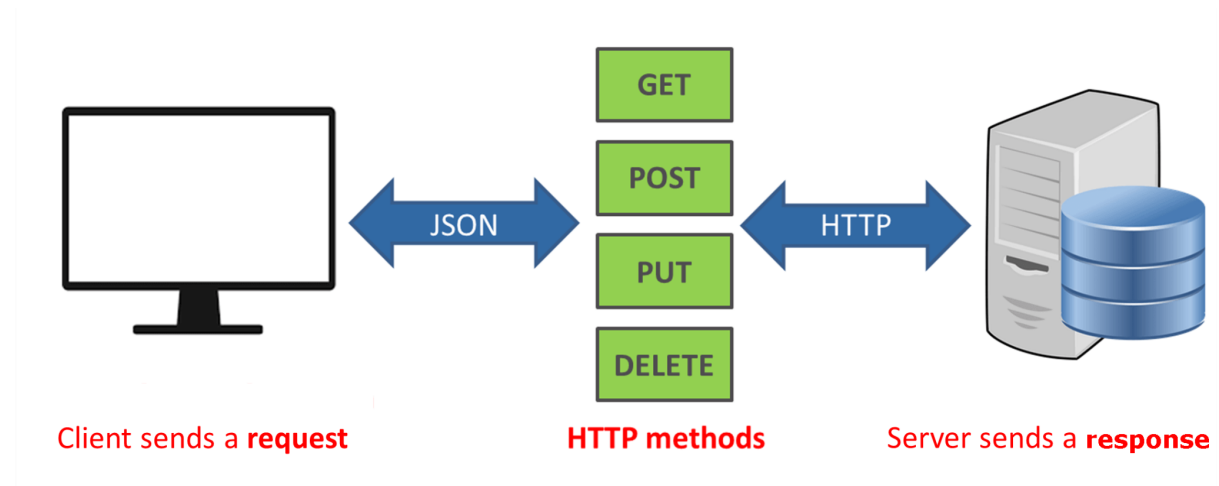


Figura 33: Ejemplo de servicio API/REST entre un cliente y un servidor.

Para hacer esto se necesita que nuestra RaspBerry haga de servidor y que dependiendo de lo que se le envíe en la petición, hará que el robot haga un movimiento u otro, esta funcionalidad se conseguirá con Flask, que es un micro framework de Python que proporciona la funcionalidad básica de un framework web y permite añadir más plug-ins por lo que la funcionalidad y el conjunto de características se pueden ampliar a un nuevo nivel. Flask se denomina micro framework de Python porque hace que el núcleo de funcionalidad simple pero extensible en términos de desarrollo. También se puede utilizar para ahorrar tiempo en la construcción de aplicaciones web. Flask utiliza Jinja Template Engine y el Werkzeug WSGI Toolkit. La estructura de Flask se divide en dos partes "Archivos estáticos y archivos de plantillas", el archivo de plantillas tiene todas las plantillas Jinja incluyendo páginas Html, mientras que los archivos estáticos tiene todos los códigos estáticos necesarios para el sitio web como el código CSS, código JavaScript y archivos de imagen [15].

Y también se necesita la librería Pyserial (Figura 34) [16] de Python, con el cual se podrá comunicar mediante puerto serie con nuestro Arduino y así enviarle la información que se necesite.



Figura 34: Logo de la librería Pyserial

Gracias a estos elementos se pueden hacer peticiones GET a la RaspBerry, esta actuará de servidor gracias a Flask y cuando reciba una petición, enviará la orden adecuada al Arduino mediante puerto serial, estos sistemas en los cuales interviene un Arduino y una RaspBerry con comunicación entre ellos, se les denomina Raspduino [17].

Finalmente, al utilizar peticiones GET como media de comunicación con el robot, cualquier lenguaje que pueda realizar peticiones GET sería posible utilizarlo para controlar al robot, por lo que lo convierte en una interfaz multimodal, las interfaces multimodales describen sistemas interactivos que buscan aprovechar las capacidades humanas naturales para comunicarse a través del habla, gestos, el tacto, la expresión facial y otras modalidades, aportando métodos más sofisticados de reconocimiento y clasificación de patrones de reconocimiento de patrones y clasificación a la interacción persona-ordenador. Aunque es poco probable que estos sistemas de escritorio y de interfaz gráfica de usuario, las interfaces multimodales adquieren cada vez más importancia gracias a los avances en el hardware y el software, y a las ventajas que pueden aportar a los usuarios y al encaje natural con el entorno informático móvil, cada vez más omnipresente [31].

3.2 Marcadores fiduciarios

Los marcadores fiduciarios son imágenes estilizadas compuestas por un conjunto de patrones distinguibles. En la actualidad, estos marcadores se utilizan para apoyar una amplia gama de aplicaciones prácticas. En algunos casos, por ejemplo, se adoptan para simplificar el proceso de reconocimiento de objetos o personas en movimiento. En otros casos se utilizan para guiar robots o vehículos en movimiento. También se utilizan para codificar una gran cantidad de información. En cualquier caso, se pueden distinguir dos tipos de información distinguir dentro de un marcador:

- Datos de reconocimiento (R-Data): es la parte del marcador que contiene información que favorece su reconocimiento dentro del entorno del mundo real, incluyendo el tamaño, la orientación y la distorsión.
- Datos de información (I-Data): el resto del marcador se compone tanto de datos sobre el objeto, datos sobre el destino y códigos de error para la corrección de estos datos.

En la parte izquierda de la Figura 10 se muestran tres ejemplos de marcadores fiduciarios comunes. El primero, Data Matrix (Figura 35a), fue diseñado para codificar gran cantidad de caracteres de datos. Originalmente, fue desarrollado para el programa del transbordador espacial, en el que había que hacer un seguimiento de millones de piezas. El segundo, MaxiCode (Figura 35b), fue diseñado por UPS (United Parcel Service) para clasificar y direccionar sus paquetes. Por último, el Código QR (Figura 35c), fue desarrollado por DENSO WAVE1 para apoyar diferentes operaciones logísticas. Estos marcadores tienen la característica común de que los R-data sólo llenan una pequeña parte de sus estructuras ya que el resto de los marcadores está diseñado para contener una gran cantidad de I-Data. Por ejemplo, dentro de un marcador Data Matrix los R-data están representados tanto por los dos bordes sólidos (inferior e izquierda), como por los dos bordes

discontinuos (arriba y derecha, alternando elementos blancos y negros). Este patrón de R-data requiere un examen minucioso para evitar interpretaciones erróneas o problemas de ambigüedad. Consideraciones similares pueden hacerse para los otros dos marcadores, MaxiCode y el código QR, cuyo patrón de R-data está representado por el conjunto de círculos concéntricos y por las tres formas cuadradas, respectivamente.

Una biblioteca de marcadores fiduciales adecuada para el desarrollo tiene que poseer algunas propiedades principales:

- El reconocimiento del marcador tiene que ser lo más independiente posible tanto de la perspectiva como de la distancia.
- El marcador tiene que contener una gran cantidad de R-data y una baja cantidad de I-data. Los únicos I-data de estos marcadores debe ser el número de identificación (ID). El resto del marcador debe ser un patrón simple de R-data fácilmente reconocible en diferentes contextos no colaborativos. [18]

Se han elegido los marcadores ArUco (Figura 35f) porque cumplen todos nuestros requisitos en cuanto a robustez, fiabilidad y versatilidad [19].



Figura 35: Ejemplos de marcadores fiduciales.

Además, su reconocimiento es muy rápido con un bajo nivel de ambigüedad en ambientes interiores, el reconocimiento de los marcadores ArUco en entornos exteriores es una tarea difícil. Este último aspecto es crucial para este tipo de sistemas. Normalmente, un marcador ArUco está representado por una matriz de 7

× 7 elementos, donde cada borde negro tiene un grosor de 1 elemento y el patrón interno es una submatriz de 5×5 elementos. Esto significa que el patrón interno se compone de 5 filas, cada una de las cuales con una longitud de 5 elementos binarios (es decir, negro o blanco). Los diseñadores han empleado una ligera modificación del Código Hamming para la codificación de cada marcador, donde se utilizan 2 bits para la información y 3 bits para la detección de errores. Como consecuencia, es posible generar hasta 1024 marcadores diferentes. La principal diferencia entre el Código Hamming original y su reformulación, es que el primer bit (paridad de los bits 3 y 5) se invierte. De este modo, el primer marcador ArUco (es decir, el marcador con ID = 0) puede codificarse con 100000 con el nuevo Código Hamming, en lugar de 00000 que sería con el original. La idea es evitar que un rectángulo completamente negro sea un marcador ID válido, con el objetivo de reducir la probabilidad de falsos positivos con objetos del entorno.

El algoritmo de posicionamiento y representación de las marcas consta de los siguientes pasos secuenciales:

1. Asignar los parámetros intrínsecos y extrínsecos de la cámara (el tamaño del sensor en píxeles y milímetros, la distancia focal del objetivo, los coeficientes de distorsión, la ubicación de la cámara).
2. Asignar las coordenadas bidimensionales de las esquinas del marcador de acuerdo con su imagen original.
3. Asignar las coordenadas tridimensionales de las esquinas del marcador de acuerdo con sus dimensiones físicas especificadas.
4. Asignar la posición 3D modelada del marcador utilizando el vector de traslación y rotación vector de traslación y rotación.
5. Cálculo de las nuevas coordenadas tridimensionales de las esquinas del marcador multiplicando las coordenadas iniciales por las matrices de rotación y traslación.
6. Proyección de los puntos tridimensionales de las esquinas del marcador en puntos bidimensionales en la imagen modelada utilizando los parámetros de la cámara.

7. Cálculo de la función de transformación de la perspectiva desde las coordenadas bidimensionales.
8. Renderización de la imagen del marcador en las nuevas coordenadas bidimensionales sobre la imagen modelada con el uso de la transformación de perspectiva.
9. Estimación de la posición del marcador 3D con el detector de marcadores en la imagen modelada.
10. Evaluación del error mediante la comparación de la posición estimada del marcador con una posición pre asignada en el paso 4.
11. Repetir los pasos 4-10 para otros marcadores [20].

El resultado final se puede ver en la figura 36.

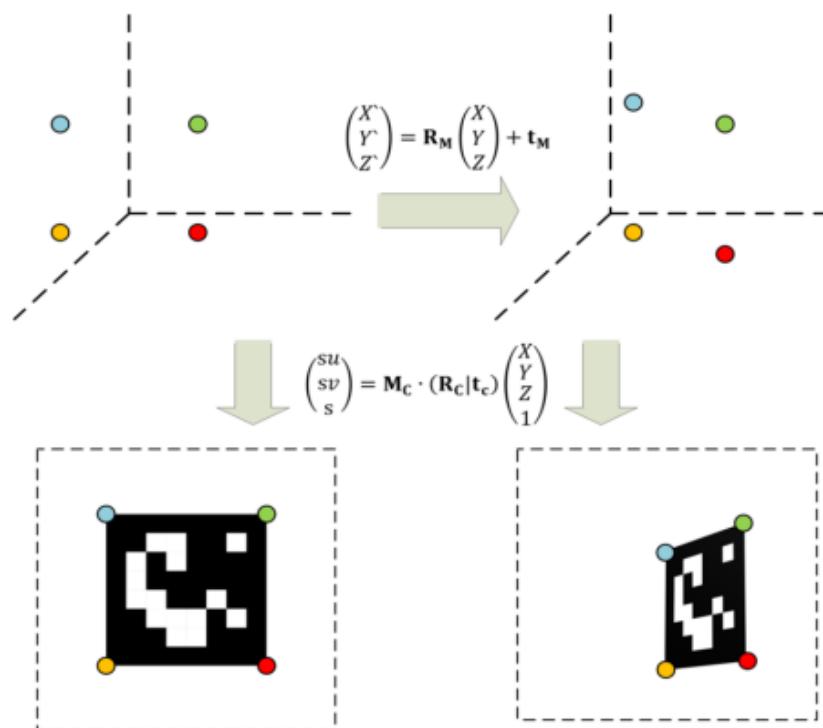


Figura 36: Algoritmo de posicionamiento y representación de marcas ArUco.

Actualmente, como se ha mencionado anteriormente, las marcas ArUco se están utilizando en varios proyectos de muy alto standing como es el Astrobees (Figura 37) [8].



Figura 37: Robot Astrobees que hay en la ISS

Para la implementación de esta tecnología usaremos Python, es una biblioteca libre de visión artificial. Fue desarrollada por Intel, en una primera versión que apareció en el mes de enero de 1999 y que se ha utilizado en infinidad de aplicaciones como pueden ser los sistemas de seguridad con detección de movimiento o aplicaciones de control de procesos donde se requiere reconocimiento de objetos. La gran aceptación de estas librerías se deben a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es una librería multiplataforma, existiendo versiones para los principales sistemas operativos. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo sencillo, eficaz, robusto y eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo y las propiedades que ofrecen esos lenguajes de programación para sistemas embarcados. OpenCV puede además utilizar un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

OpenCV se implementa como librería para códigos de programación en C, C++, Python, Java de manera muy sencilla y como se indicó anteriormente se dispone de multitud de material de información tanto en Internet como en manuales, lo que ha hecho que sea la más utilizada para sistemas de visión por computador. OpenCV permite un diseño muy sencillo de órdenes de procesado de imágenes, que serían operaciones bastante complejas si no estuviera instalada esta librería, como los detectores de borde de Sobel o de Canny, o la segmentación de imágenes, ya que habría que utilizar todo el potencial del cálculo matricial píxel a píxel de manera manual, lo cual es una gran fuente de posibles fallos de programación.

3.3 Navegación autónoma con A*

A* es un algoritmo de búsqueda ampliamente utilizado en la búsqueda de caminos y en el recorrido de grafos, el proceso de trazar un camino eficientemente entre puntos, llamados nodos. Es conocido por su rendimiento y precisión, y su uso está muy extendido. Peter Hart, Nils Nilsson y Bertram Raphael describieron por primera vez el algoritmo en 1968. Es una extensión del algoritmo de Edsger Dijkstra de 1959. A* consigue un mejor rendimiento (con respecto al tiempo) mediante el uso de la heurística. En 1964, Nils Nilsson inventó un enfoque basado en la heurística para aumentar la velocidad del algoritmo de Dijkstra. Este algoritmo se llamó A1. En 1967, Bertram Raphael mejoró notablemente este algoritmo, pero no pudo demostrar la optimalidad. Llamó a este algoritmo A2. Luego, en 1968, Peter E. Hart introdujo un argumento que demostraba que A2 era óptimo cuando se utiliza una heurística consistente con sólo pequeños cambios, su prueba del algoritmo también incluía una sección que mostraba que el nuevo algoritmo A2 era el mejor algoritmo posible dadas las condiciones. Así, denominó al nuevo algoritmo en sintaxis de estrella de Kleene para ser el algoritmo que comienza con A e incluye todos los números de versión posibles o A*.

Cuando A* recorre el grafo, sigue el camino de menor coste conocido, manteniendo una cola de prioridad ordenada de segmentos alternativos a lo largo del camino. Si,

en algún momento, un segmento del camino que se está recorriendo tiene un coste mayor que otro segmento del camino que se ha encontrado, se abandona el segmento del camino de mayor coste y se recorre el segmento del camino de menor coste en su lugar. Este proceso continúa hasta que se alcanza el objetivo. A* utiliza una búsqueda del mejor primero y encuentra un camino de menor coste desde un nodo inicial dado hasta un nodo de meta (de uno o más objetivos posibles).

Utiliza una función heurística de distancia + coste (normalmente denominada $f(x)$) para determinar el orden en que la búsqueda visita los nodos del árbol. La heurística de distancia + coste es una suma de dos funciones:

- La función de coste del camino, que es el coste desde el nodo inicial hasta el nodo actual (normalmente denotado $g(x)$)
- Y una "estimación heurística" admisible de la distancia a la meta (normalmente denominada $h(x)$).

La parte $h(x)$ de la función $f(x)$ debe ser una heurística admisible; es decir, no debe sobreestimar la distancia a la objetivo. Así, para una aplicación como el enrutamiento, $h(x)$ podría representar la distancia en línea recta a la meta, ya que es físicamente la menor distancia posible entre dos puntos o nodos. Si la heurística h satisface la condición adicional $h(x) \leq d(x,y) + h(y)$ para cada arista x, y del gráfico (donde d denota la longitud de esa arista), entonces h se denomina monótona, o consistente. En este caso, A* puede implementarse de forma más eficiente y A* es equivalente a ejecutar el algoritmo de Dijkstra con el coste reducido:

$$d'(x,y) := d(x,y) - h(x) + h(y) \quad (4)$$

La complejidad temporal de A* depende de la heurística. En el peor de los casos, el número de nodos expandidos es exponencial en la longitud de la solución (el

camino más corto), pero es polinomial cuando el espacio de búsqueda es un árbol, hay un único estado objetivo, y la función heurística h cumple la siguiente condición:

$$|h(x) - h^*(x)| = O(\log h^*(x)) \quad (5)$$

Donde h^* es la heurística óptima, el coste exacto para llegar de x a la meta. En otras palabras, el error de h no crecerá más rápido que el logaritmo de la "heurística perfecta" h^* que devuelve la verdadera distancia de x a la meta [21].

Aquí se examina el funcionamiento del algoritmo de Dijkstra y del algoritmo A* en el problema de búsqueda del camino más corto con obstáculos. La Figura 38 muestra las diferencias entre los métodos Dijkstra y A* en la búsqueda del camino más corto. Los colores del patrón muestra cómo el agente detectaba el objetivo [22].

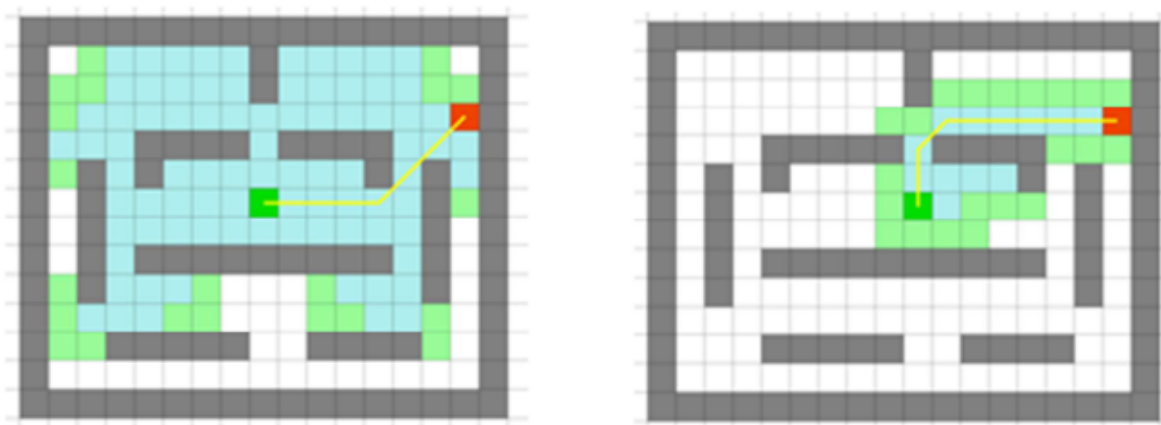


Figura 38: Comparación de exploración entre Dijkstra (izquierda) y A (derecha)*

Hay situaciones en las que A* puede no funcionar muy bien, por diversas razones. Los requisitos de tiempo más o menos real de los juegos, más las limitaciones de memoria y tiempo de procesador disponibles en algunos de ellos, pueden dificultar el buen funcionamiento de A*. Un mapa grande puede requerir miles de entradas en la lista de visitados y no visitados, y puede no haber espacio

suficiente para ello. La calidad de la búsqueda de A^* depende de la calidad de la heurística $h(n)$. Si h está muy cerca del coste real del camino restante su eficiencia será alta; en cambio, si es demasiado baja, su eficiencia es muy mala. De hecho, la búsqueda de amplitud primero es una búsqueda A^* , con h siendo trivialmente cero para todos los nodos, esto ciertamente subestima el coste del camino restante, y aunque encontrará el camino óptimo, lo hará lentamente [23].

4. Hardware

En este capítulo se habla de la base hardware de la que partimos, en concreto el robot hexápodo, y también los cambios que se le han tenido que realizar a nivel de hardware para empezar a trabajar con él en este proyecto.

4.1 Robot hexápodo

La base de este proyecto, un robot hexápodo desarrollado por un antiguo alumno de la EPSJ (Figura 39), el cual está dotado de distintos grados de libertad y distintos niveles de velocidad [12]. El robot contaba con un mando de PS2 incorporado que se comunicaba mediante una PCB diseñada por el autor del proyecto, la cual se conectaba al Arduino que está en el robot.

La principal idea de este trabajo es incorporar una RaspBerry al robot ya que, teniendo un Arduino que actúa como la médula espinal y los servomotores que actúan como los músculos, era de gran interés poner la RaspBerry como si fuera el cerebro del robot y poder controlar desde la RaspBerry el comportamiento del robot.



Figura 39: Robot hexápodo mostrando máxima altura de paso

El robot está alimentado por una batería de 6000 mAh que daba una corriente de 5.5V y 2A, suficiente para conectar la RaspBerry en paralelo y poder alimentar todos los componentes necesarios para su correcto funcionamiento.

En cuanto al control del robot, como se ha indicado anteriormente, se controla con un mando de PS2, siguiendo el siguiente esquema que está representado en la Figura 40:

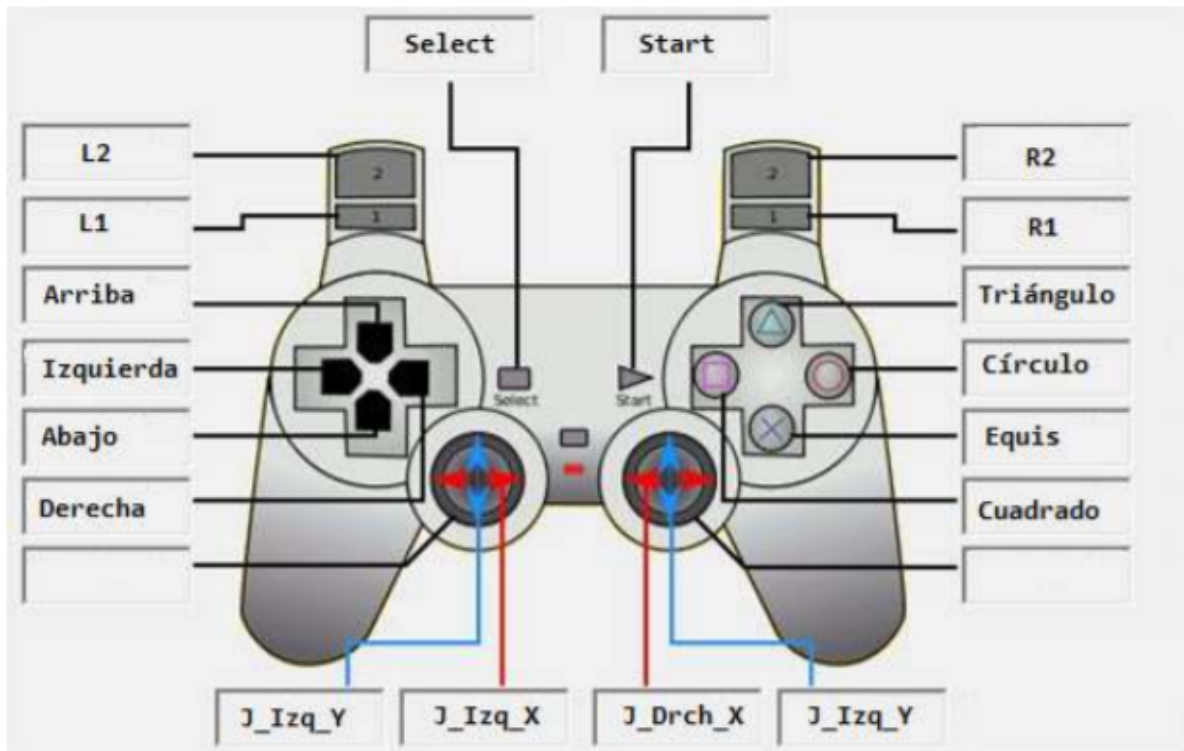


Figura 40: Botones y Joysticks de mando de PS2

El robot tiene varios modos de funcionamiento entre ellos el de desplazamiento y el de control de cuerpo, cada uno cuenta con sus propios controles que indico a continuación:

En el modo de desplazamiento el robot se mueve libremente por el suelo, pudiendo además girar y aumentar su altura respecto del suelo, los comandos se pueden ver en la Tabla 1.

Comando	Función
Triángulo	-
Círculo	-
Equis	-
Cuadrado	-
Arriba	-
Izquierda	Disminuimos los pasos de la secuencia
Abajo	-
Derecha	Aumentamos los pasos de la secuencia
Select	Cambiamos de secuencia
Start	Cambiamos de modo
R1	Aumentamos la altura del robot
R2	disminuimos la altura del robot
L1	Aumentamos la altura de los pasos
L2	disminuimos la altura de los pasos
Joystick izquierdo eje Y	Desplazamiento hacia delante (positivo) y hacia atrás (negativo)
Joystick izquierdo eje X	Desplazamiento a la izquierda(negativo) o derecha (positivo)
Joystick derecho eje Y	-
Joystick derecho eje X	Giro horario en el sentido positivo del eje, y antihorario en el sentido opuesto

Tabla 1: Comandos para el modo movimiento del robot

En el modo de control del cuerpo se maneja el giro y traslación del robot sobre los tres ejes coordenados. Los botones asignados para el control del cuerpo, representados en la Tabla 2, del robot son los siguientes:

Comando	Función
Triángulo	Traslación positiva del cuerpo sobre el eje Y
Círculo	Traslación positiva del cuerpo sobre el eje X
Equis	Traslación negativa del cuerpo sobre el eje Y
Cuadrado	Traslación negativa del cuerpo sobre el eje Y
Arriba	-
Izquierda	-
Abajo	-
Derecha	-
Select	-
Start	Cambiamos de modo
R1	Aumentamos la altura del robot
R2	disminuimos la altura del robot
L1	-
L2	-
Joystick izquierdo eje Y	Giro del cuerpo del robot sobre el eje Y
Joystick izquierdo eje X	Giro del cuerpo del robot sobre el eje X
Joystick derecho eje Y	-
Joystick derecho eje X	Giro del cuerpo del robot sobre el eje Z

Tabla 2: Comandos para el modo rotación del robot

Los botones más importantes son el de start y el de select, el primero permite cambiar entre los modos de movimiento del cuerpo y desplazamiento del robot; mientras que con el segundo se cambia el tipo de secuencia para el desplazamiento del robot. A parte de esto, con los joystick se controla la dirección y sentido del movimiento del robot.

El robot dota de una gran serie de movimientos y de rotaciones, las cuales funcionan hasta el fallo, por lo que hay que tener cuidado de no sobrepasar los límites de movilidad.

El fallo es provocado cuando se intenta ubicar a las patas en la posición la cual implica una altura mayor a la envergadura de estas, o que alguna de las articulaciones necesita orientarse a un cierto ángulo que está fuera de los límites

establecidos para el servomotor en cuestión. Para la altura máxima asignada, los servos no llegan a dar fallo, pero el robot comienza a moverse muy erráticamente a medida que se sobrepasan los 14 cm de altura. Esto evidencia que las patas están alcanzando su máxima envergadura (Figura 41) y no consiguen alcanzar su posición correctamente.



Figura 41: Robot a la máxima altura

El robot permite una gran altura de paso sin que este tenga que llegar al fallo, por lo que limitaremos esta altura simplemente porque no tiene lógica elevar tanto las patas. En las pruebas se observa que cuanto más incrementamos la altura de paso, el robot se vuelve más inestable. Esto es debido a que las patas tienen que recorrer más distancia en el mismo tiempo, incrementándose la inercia que adquieren estas y por lo tanto el balanceo del robot. También observamos que cuanto más dure el tiempo de ciclo, el robot es más estable a pesar de que incrementemos la altura de paso, esto se debe a que al aumentar el tiempo de ciclo, estamos disminuyendo la velocidad de desplazamiento del robot.

La distancia máxima de zancada está limitada a 8cm como máximo en el eje X y 10 cm en el eje Y. Por encima de estos límites las patas podrían chocar entre sí, debido a alguna combinación aleatoria en los controles, producida por un movimiento muy brusco de ambos joysticks a la vez. También se observa que aunque se seleccione una distancia de empuje de 8 y 10 cm para cada eje, físicamente se observa que la amplitud de movimiento es aproximadamente 1 cm o 1.5 cm inferior a la deseada.

Respecto al giro del cuerpo (rotación sobre sí mismo), el valor máximo está establecido en 30° por zancada para que las patas no choquen unas con otras en su movimiento. Respecto del modo de movimiento del cuerpo, estas rotaciones están limitadas a 30° , ya que si aumentamos más de este valor algunos servomotores llegan a sus límites, provocando el fallo. En la Figura 42 se puede ver toda la cantidad de giros se le permite hacer al cuerpo del robot.



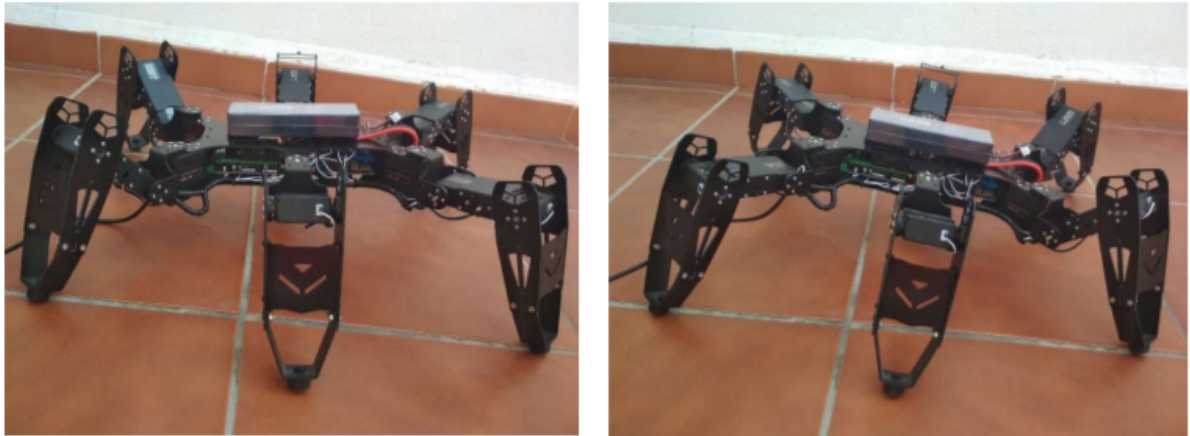


Figura 42: Distintos tipos de rotaciones permitidas por el robot (6 tipos).

Para la traslación del cuerpo en el plano xy, los valores máximos son 6cm de traslación para el eje X, en ambos sentidos y 8 cm en el eje Y en ambos sentidos. Si sobrepasamos estos valores las patas no conseguirán llegar hasta sus posiciones requeridas, provocando el fallo de los servomotores, como puede verse en la Figura 43.

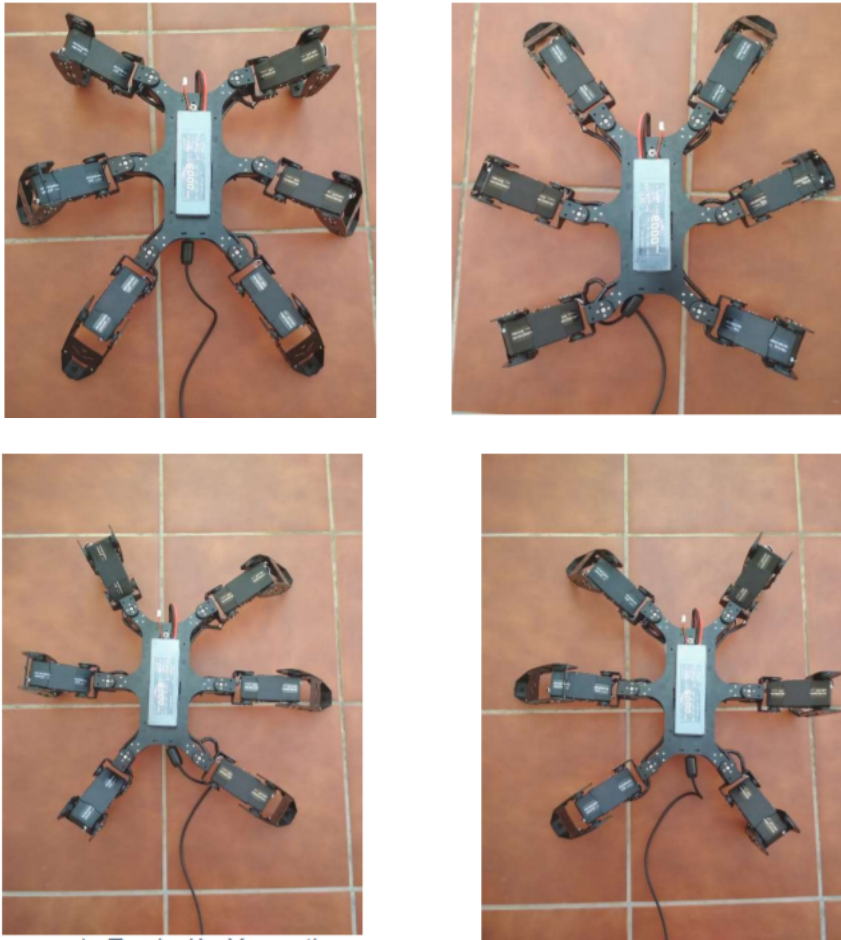


Figura 43: Distintos tipos de desplazamientos que puede hacer el robot

Cabe destacar la estabilidad con la que trabaja el robot y la robustez que tiene, estas características hacen que sea perfecto trabajar con él y modificarlo sin miedo a romperlo o hacerle algún daño.

4.2 Incorporación de los nuevos elementos

El primer cambio que hubo que hacerle al robot fue quitarle el mando, ya que si queríamos que fuese inalámbrico gracias a la RaspBerry, era imprescindible quitar este elemento. El mando estaba soldado a la PCB y asegurado con pegamento termofusible (Figura 44), el cual hubo que retirar primero para poder acceder a la PCB.



Figura 44: PCB sin modificar

Una vez retirado el pegamento que lo unía a la placa, se puede alcanzar a ver la conexión del mando con la PCB. El pegamento termofusible cumplía una función aislante y a la vez, para asegurar las conexiones ya que con el uso del robot y el paso del tiempo se habían desgastado un poco (Figura 45).

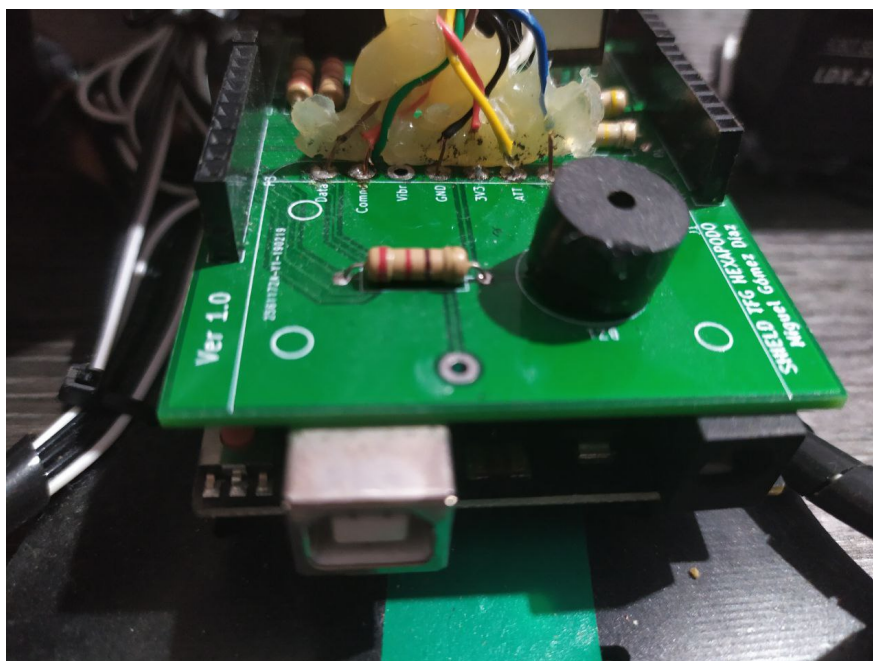


Figura 45: PCB tras levantar el pegamento termofusible

Antes de retirar los cables, hubo que comprobar las conexiones de la PCB con el Arduino para comprobar que si se desconecta el mando, no habría problemas. Para ello se comprobó en la memoria las conexiones y se pudo verificar que no tendría que haber ningún problema, ya que las conexiones más problemáticas, en estos casos que son las de Vcc y la GND, iban directas al Arduino por lo que no se quedaba ningún circuito sin cerrar (Figura 46).

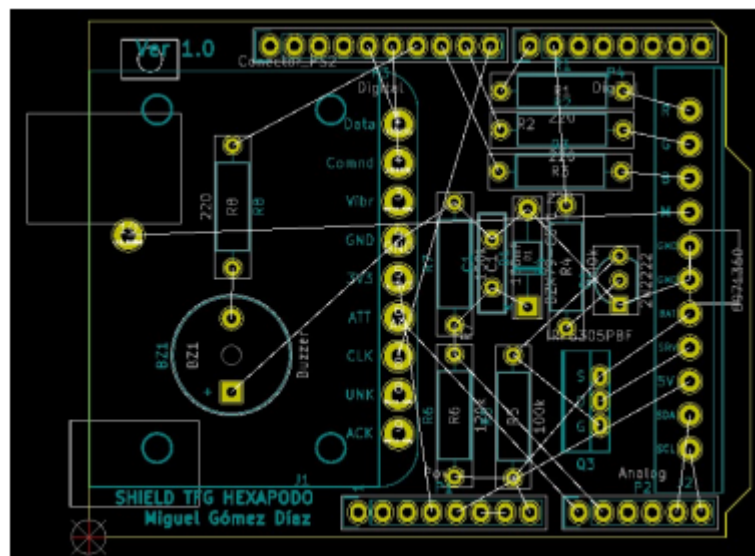


Figura 46: Circuitería de la PCB

Una vez se comprobó que se podía desconectar el mando con total seguridad, se quitaron los cables y se limpió la zona para que no quedara ningún contacto entre 2 conexiones (Figura 47).



Figura 47: PCB tras desconectar el mando

Una vez retirado el mando, solo quedaba sacar 2 conexiones, Vcc y GND, para alimentar la RaspBerry con las conexiones GPIO. Estas conexiones se sacaron mediante jumpers, los cuales van de las conexiones de 5.5V y GND de la derecha de la PCB hacia los pines 2 y 14 de la RaspBerry, como se ve en la Figura 48 está el esquema de conexiones y en la Figura 49 el resultado final.

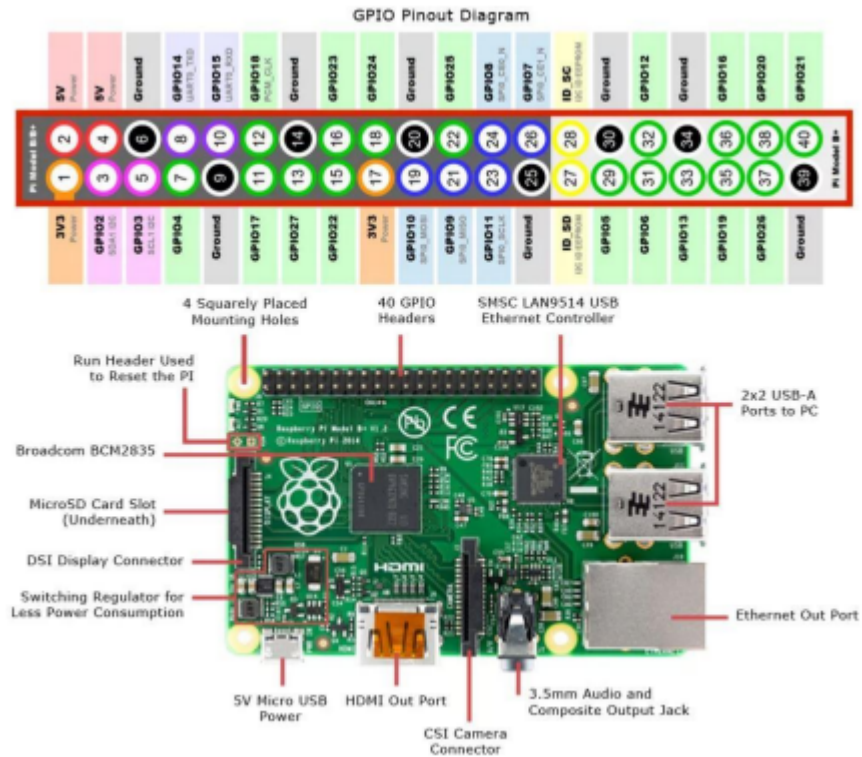


Figura 48: Esquema de conexiones de RaspBerry pi Model 3



Figura 49: Conexión mediante GPIO del robot a la RaspBerry

La RaspBerry ya estaba alimentada, lo siguiente era conectar la cámara, los ojos de este proyecto, la cámara es una RaspBerry Pi NoIR Camera Module V2, se trata de una cámara de alta definición que se puede usar tanto para vídeo como para capturar imágenes. Dispone de funciones de control automático como, por ejemplo, la detección de iluminación. Entre muchas de las ventajas hay que señalar que se conecta a la placa a través del puerto SCI, lo cual supone una ventaja si en un futuro necesitamos mejorar el presente proyecto y vamos a necesitar usar puertos USB, como se ve en la Figura 50.



Figura 50: Cámara NoIR V2

Existen dos tipos de este periférico conocidos como Daylight y Noir. Además de su pequeño tamaño posee 8 megapíxeles, gracias al sensor Sony IMX219 que lleva incorporado. También indicar, que es capaz de tomar imágenes fijas así como vídeos en alta definición [13].

Para proteger la cámara, se imprimió una carcasa en 3D con una base redonda la cual serviría para, además de proteger la cámara y darle estabilidad, poder dejarla en el ángulo adecuado para su funcionamiento. Tanto a la cámara como a la RaspBerry se le puso velcro por debajo para asegurarlas al robot, ocupando así la posición de la batería y la batería la pondremos en la base del robot, con sus debidas tiras de velcro en esa zona.

Para terminar la parte más hardware del proyecto, bastaba con conectar la RaspBerry por USB con el Arduino, para así tener comunicadas a las 2 partes mediante puerto serie, el resultado final se puede ver en la Figura 51.

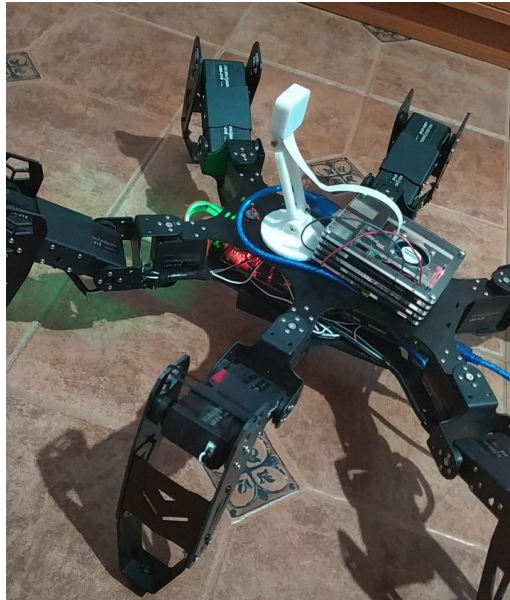


Figura 51: Robot hexápodo tras incorporar los nuevos elementos

Tras las modificaciones realizadas, tenemos una arquitectura de control por visión de Eye-in-hand (Figura 52), la cual hemos explicado anteriormente:

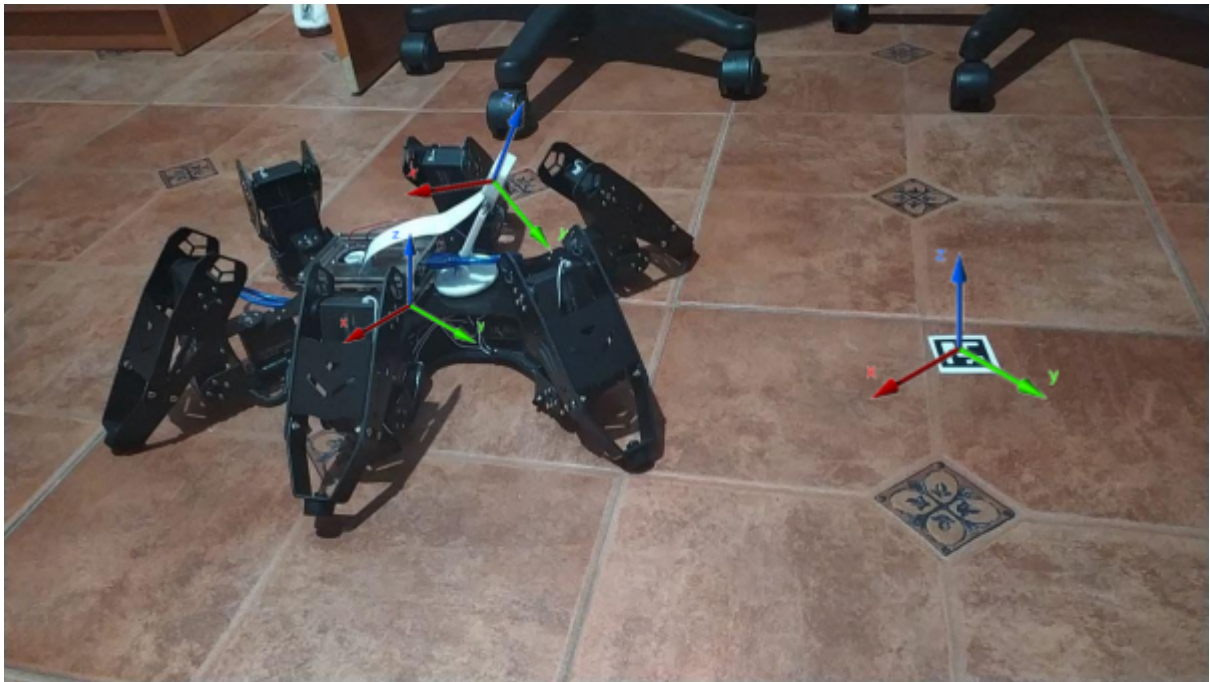


Figura 52: Sistemas de referencia presentes en una arquitectura Eye-In-Hand

5. Proceso ingeniería de software

Una de las definiciones que se pueden encontrar de ingeniería del software es una colección sistemática de buenas prácticas y técnicas de desarrollo de programas [24].

En el proceso de ingeniería de software de este trabajo se contará con los siguientes metodologías a desarrollar:

- Implementación en Raspberry Pi con Python para la integración con la base de Arduino (encargada de proveer a los servos de capacidad para moverse conjuntamente, de forma ordenada e inteligente, permitiendo realizar movimientos en cada una de las extremidades).
- Diseño y programación de algoritmos de reconocimiento de marcas.
- Diseño y programación de algoritmos de control para la navegación autónoma.
- Diseño y programación de una aplicación, interfaz de usuario, para controlar los movimientos del robot de forma manual y seleccionar funcionalidades de movimientos.

5.1 Fases de la ingeniería del software

Las fases que se encuentran en el proceso de ingeniería del software son las siguientes:

- Especificación de Requerimientos: Descripción completa del comportamiento del sistema que se va a desarrollar.
- Análisis del sistema: Especificación de los objetivos y limitaciones de nuestro sistema.
- Diseño del sistema: Definición de arquitectura, módulos, interfaces y datos para alcanzar los requisitos especificados.

- Implementación del sistema: Desarrollo del software anteriormente especificado que cumpla todos los requisitos.
- Pruebas: Verificar y validar que el sistema cumpla con nuestras perspectivas.

5.2 Especificación de requisitos

La definición de requisitos es la tarea de recopilar toda la información relevante que se utilizará para comprender una situación de un problema antes del desarrollo del sistema, la documentación de esta información se denomina especificación de requisitos. La forma y el contenido de la especificación de requisitos pueden tener un impacto tremendo en la tarea del software a lo largo de su vida útil [25].

5.2.1 Requerimientos del sistema de control remoto mediante servicio API/REST

5.2.1.1 Funcionales

Requerimiento funcional 1: Servicio API/REST:

El programa recibe, mediante la URL a la que se conecta el usuario, el movimiento que quiere que envíe al Arduino, coge la parte correspondiente de la URL, que en este caso lo llamamos slug (Figura 53) y lo envía por puerto serie al Arduino.

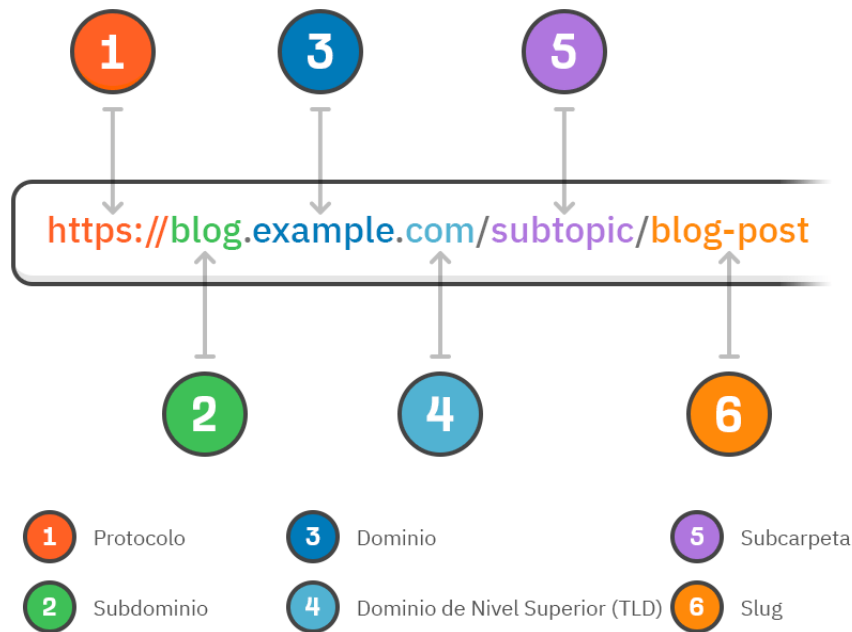


Figura 53: Ejemplo de URL

Requerimiento funcional 2: Procesamiento de información que llega por puerto serie

Al script de Arduino le llega una serie de información que le envía el servicio API/REST y este la procesa, dependiendo de lo que le llegue, el robot realizará un movimiento u otro.

5.2.1.2 No funcionales

- Requerimientos físicos (RaspBerry):
 - Procesador: 1.2 GHz
 - RAM: 1GB
 - HDD: 120GB
 - Conexión a la red
 - Conexión puerto serie
- Requerimientos físicos (Arduino):
 - Procesador: 16 MHz
 - RAM: 8KB
 - Flash: 256 KB
 - Conexión puerto serie

- Requerimientos de software:
 - Python 3.7.3
 - Software Arduino
 - Distribución GNU/Linux (Raspbian)
 - Servidor SSH

5.2.2 Requerimientos del sistema de reconocimiento de marcas ArUco

5.2.2.1 Funcionales

Requerimiento funcional 1: Reconocer marcas ArUco:

El script recibirá las imágenes a través de la cámara que tiene conectada la Raspberry, coge esas imágenes y las procesa, dando como resultado el identificador de la marca, la posición y la rotación. La imagen procesada se mostrará también por protocolo HTTP, para así poder visualizar si el algoritmo funciona correctamente.

5.2.2.2 No funcionales

- Requerimientos físicos (Cámara Raspberry):
 - Enfoque fijo de 8 megapíxeles
 - Compatible con 1080p, 720p60 y VGA90
 - Sensor de imagen CMOS Sony IMX219PQ
- Requerimientos físicos:
 - Marcas ArUco impresas
- Requerimientos de software:
 - Python 3.7.3

5.2.3 Requerimientos del sistema de navegación autónoma

5.2.3.1 Funcionales

Requerimiento funcional 1: Algoritmo A*:

El algoritmo le indicará al robot a qué marca tiene que ir, el robot irá hacia esa marca y una vez esté en ella, comunicará al algoritmo que ha llegado, en ese momento el algoritmo continúa su ejecución y le indica la siguiente marca a la que tiene que ir, así hasta que finalice el algoritmo.

5.2.3.2 No funcionales

- Requerimientos de software:
 - Python 3.7.3

5.2.4 Requerimientos de la interfaz de usuario

5.2.4.1 Funcionales

Requerimiento funcional 1: Mundo Virtual en Unity

Desde la interfaz gráfica en 3D se podrá controlar al robot, a la vez que si iniciamos el sistema de navegación autónoma, los movimientos se verán reflejados en ambos robots, tanto el de la vida real como el virtual.

5.2.4.2 No funcionales

- Requerimientos de software
 - Unity
 - Drivers de NVIDIA
- Requerimientos de hardware
 - Tarjeta Gráfica GTX 970 o mejor
 - Procesador Intel I7 o mejor
 - 8 GB de RAM o más
 - HDD 16 GB o superior

5.3 Análisis del sistema

El análisis del sistema es el proceso que descompone un sistema en sus componentes con el fin de definir qué tan bien interactúan estos componentes para cumplir con los requisitos establecidos.

Caso de uso 1: Sistema de control mediante servicio API/REST (Figura 54)

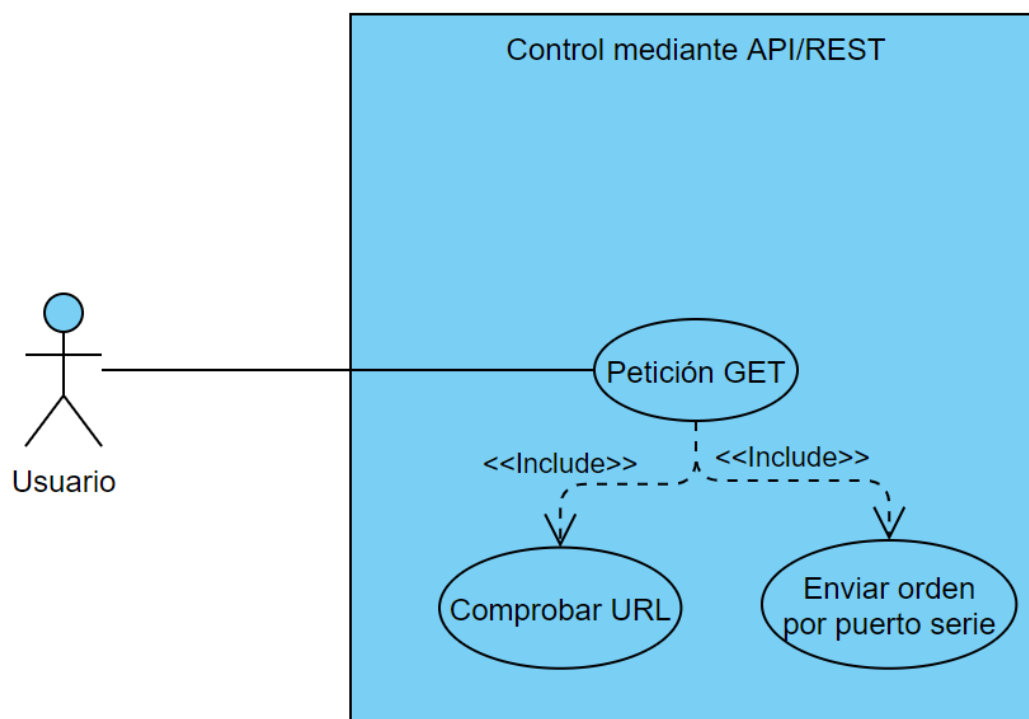


Figura 54: Caso de uso de control mediante API/REST

- Actores: Usuario
- Condiciones de entrada: Deben proporcionarse los parámetros de entrada:
 - Movimiento que queremos que haga el robot
- Eventos
 - El robot iniciará su rutina de encendido, ya que al detectar la comunicación por puerto serie, este reinicia. Tras eso se queda a la espera de órdenes.

- Cuando se envíe la orden, el robot hará el movimiento que le indiquemos nosotros.
- Excepciones
 - Si no se introduce una orden correcta, el robot no hará nada ya que solo detecta las órdenes que son correctas.

Caso de uso 2: Visualizar reconocimiento de marcas ArUco (Figura 55)

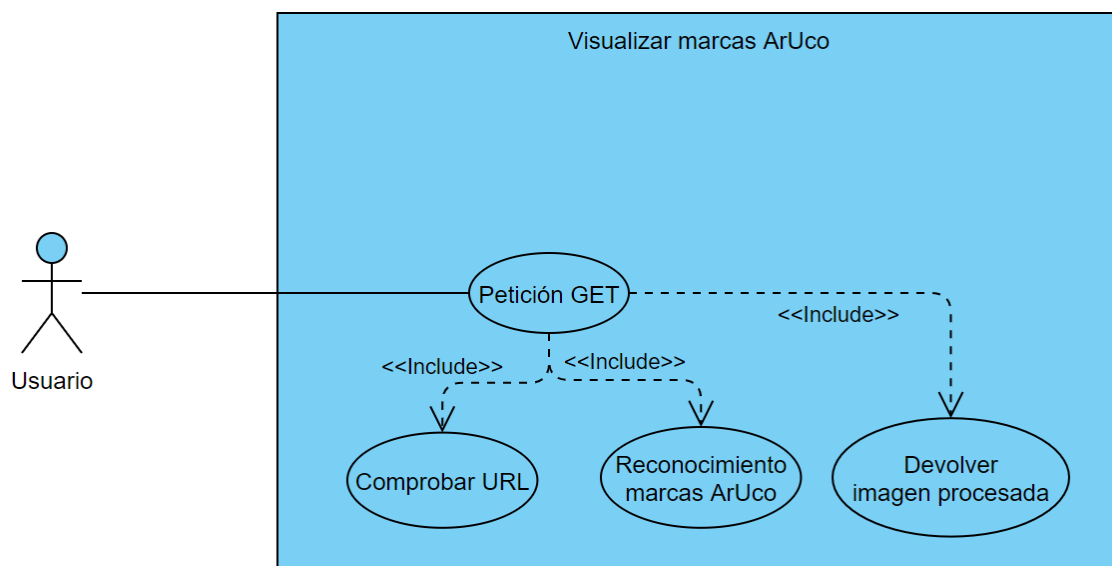


Figura 55: Caso de uso de visualizar marcas ArUco

- Actores: Usuario
- Condiciones de entrada: Deben proporcionarse los parámetros de entrada:
 - URL que nos da acceso a las imágenes procesadas
- Eventos:
 - En el navegador se visualizarán las imágenes obtenidas por la cámara después de haber sido procesadas, por lo que se nos mostrarán las marcas reconocidas.

- Se abrirá una terminal donde se indicará la información relativa a las marcas reconocidas.
- Excepciones
 - Si no se encuentra ninguna marca, en la consola indicará None.

Caso de uso 3: Sistema de navegación autónoma (Figura 56)

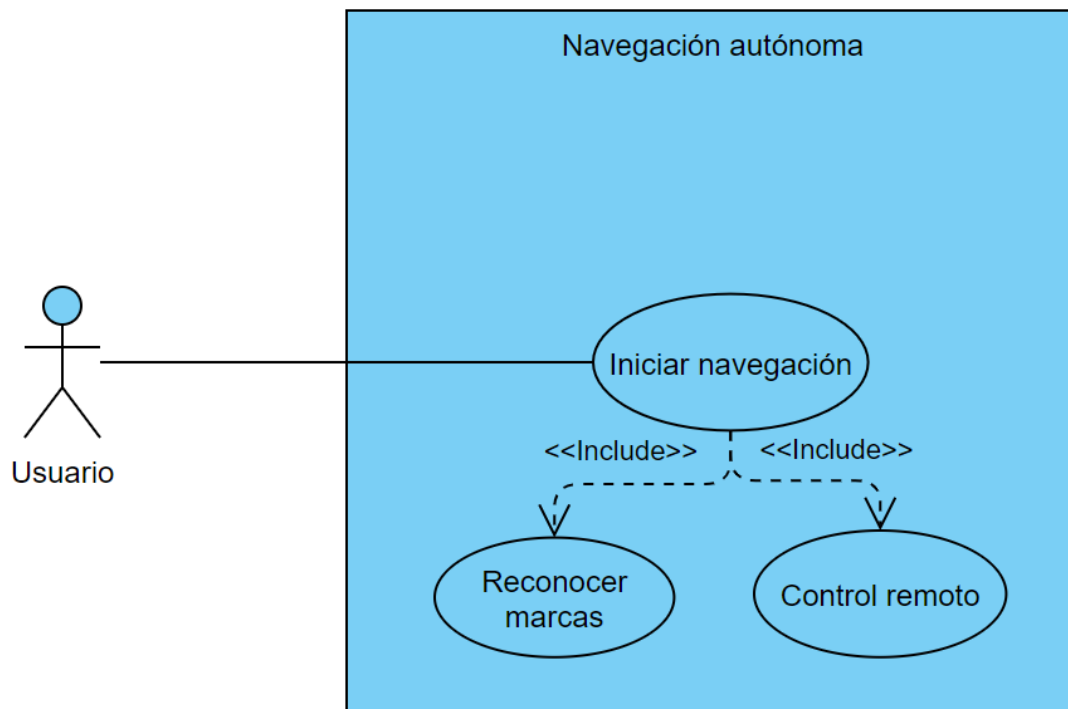


Figura 56: Caso de uso de navegación autónoma

- Actores: Usuario
- Condiciones de entrada:
 - El robot tiene que estar con el control remoto activado, es decir, con el servicio API/REST que nos permite controlarlo mediante GET, ya que el algoritmo necesita ese servicio para poder enviar las peticiones de movimiento.
- Eventos:
 - El robot comenzará a andar hacia la marca que le indique el algoritmo

- Una vez se encuentre encima de ella, se parará durante unos segundos
- Cuando finalice el algoritmo, el robot se quedará quieto.
- Excepciones
 - Si el algoritmo perdiera el control en algún momento, o dejara de funcionar la cámara, se encargaría el propio algoritmo de parar al robot para que no haga movimientos sin control.

Caso de uso 5: Interfaz de usuario (Figura 57)

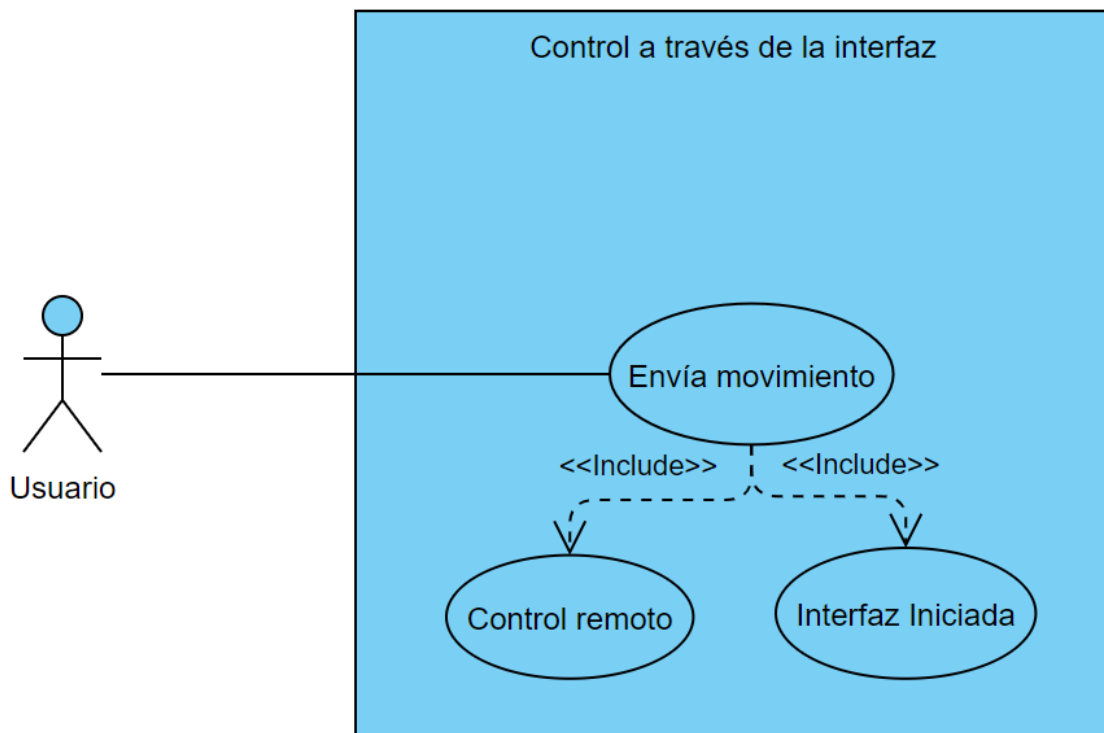


Figura 57: Caso de uso de control a través de interfaz

- Actores: Usuario
- Condiciones de entrada:
 - El sistema de control remoto tiene que estar iniciado, ya que la interfaz funciona gracias al servicio API/REST.
 - La interfaz tiene que estar iniciada para poder enviar las peticiones.
- Eventos:

- El robot se moverá a la vez que en la simulación.
- Mientras estén en funcionamiento los 2 elementos, se podrá mover al robot desde el teclado.
- Excepciones:
 - Si el servicio API/REST no está iniciado, la interfaz avisará de ello para así evitar usar la interfaz sin tener nada conectado.

Caso de uso 6: Simulación de la navegación del robot (Figura 58)

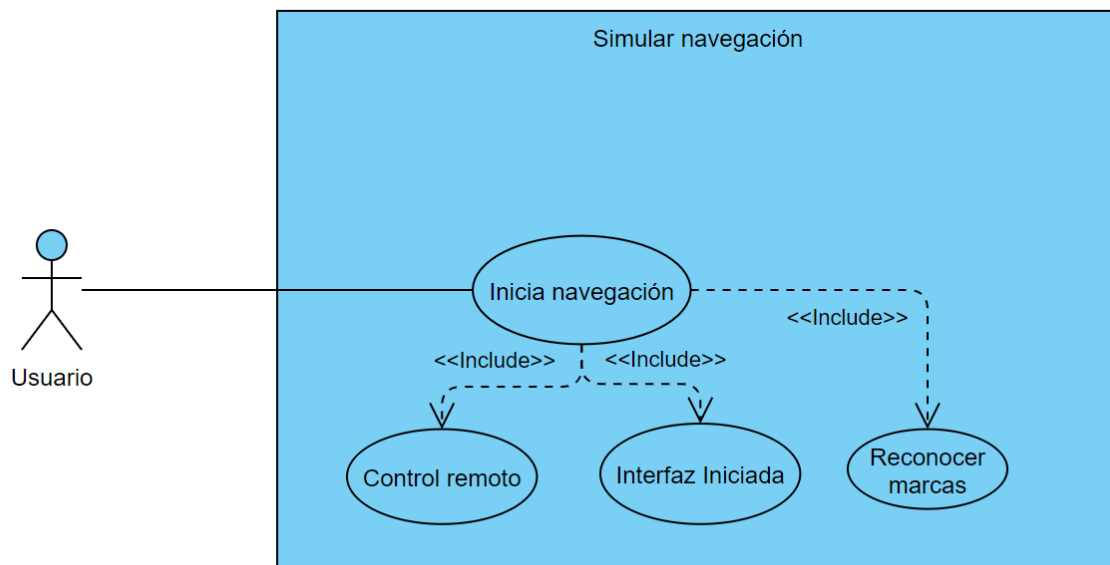


Figura 58: Caso de uso de simular navegación

- Actores: Usuario
- Condiciones de entrada:
 - Tienen que estar indicados tanto el servicio API/REST para el control remoto, como la interfaz, para que los movimientos que mande al algoritmo se vean reflejados en ambos sitios.
- Eventos:
 - El robot comenzará a moverse hacia la marca indicada, los movimientos que haga el movimiento en la vida real, se verán reflejados en la interfaz.
- Excepciones

- Si el servicio API/REST no está iniciado, la interfaz avisará de ello para así evitar usar la interfaz sin tener nada conectado.
- Si falla el servicio API/REST, el algoritmo seguirá ejecutándose en la interfaz y viceversa.

Caso de uso 7: Sistema general (Figura 59)

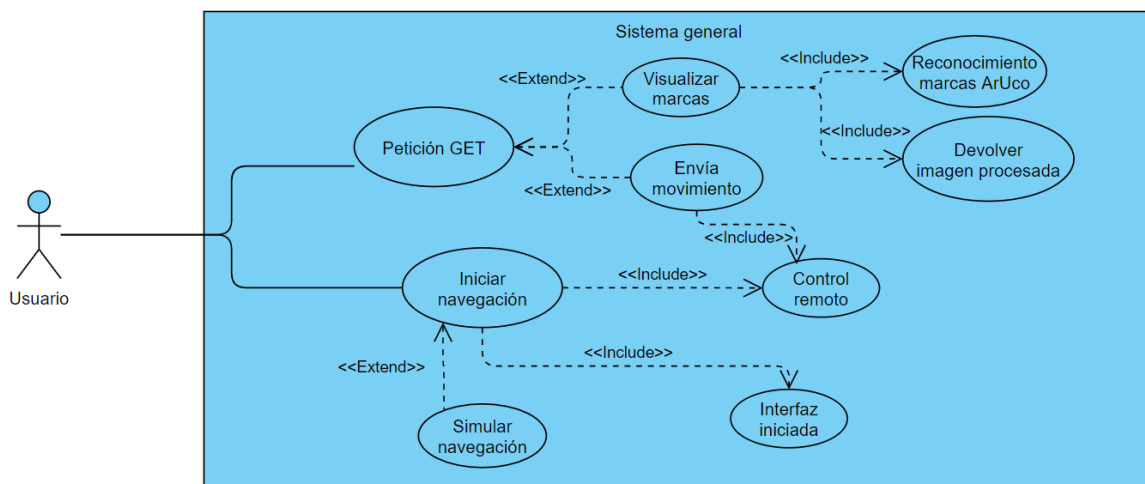


Figura 59: Caso de uso del sistema general

- Actores: Usuario
- Condiciones de entrada:
 - Como mínimo si queremos que el robot haga movimientos, tenemos que tener el servicio API/REST iniciado.
 - Si solamente se quiere visualizar las marcas reconocidas, basta con
- Eventos:
 - Dependiendo del GET que hagas podrás mover al robot o visualizar las marcas reconocidas.
- Excepciones:
 - Si el servicio API/REST no está iniciado, la interfaz avisará de ello para así evitar usar la interfaz sin tener nada conectado.
 - Si falla el servicio API/REST, el algoritmo seguirá ejecutándose en la interfaz y viceversa.

5.4 Diseño del sistema

El diseño del sistema es el proceso de diseñar los elementos de un sistema, como la arquitectura, los módulos y componentes, las diferentes interfaces de esos componentes y los datos que pasan por ese sistema.

5.4.1 Formato de intercambio de información

Los formatos de intercambio de datos pasaron de estar orientados al marcado y la visualización, a soportar la codificación de metadatos que describen los atributos estructurales de la información. Los requisitos de apoyo al intercambio de datos de las aplicaciones Java condujeron al desarrollo de formatos estándar de intercambio de datos. JSON y XML son dos de los más usados.

5.4.1.1 XML

El lenguaje de marcado extensible (XML) es un subconjunto del Lenguaje de Marcado Generalizado Estándar (SGML) y evolucionó como resultado de la complejidad de SGML. XML se considera el "santo grial" de la informática por su formato universal de representación de datos. La intención de un documento XML es evidente y está integrada en su estructura. Las consideraciones de diseño fundamentales de XML incluyen la simplicidad y la legibilidad humana. Entre los objetivos de diseño de XML, el W3C especifica que "XML debe ser directamente utilizable en Internet" y "los documentos XML deben ser legibles y razonablemente claros".

Los principales usos de XML son las llamadas a procedimientos remotos (RPC) y la serialización de objetos para la transferencia de datos entre aplicaciones. XML es un lenguaje utilizado para crear marcas definidas por el usuario, para documentos y esquemas de codificación. XML no tiene conjuntos de etiquetas predefinidos, y cada etiqueta válida es definida por un usuario, o a través de otro esquema automatizado. Un gran número de tutoriales y foros de usuarios proporcionan un amplio apoyo a

XML y han contribuido a crear una amplia base. XML es un formato de datos jerárquico definido por el usuario, como se puede ver en la Figura 60.

```
<name>  
  <first>John</first>  
  <last>Smith</last>  
</name>
```

Figura 60: Ejemplo XML

5.4.1.2 JSON

JSON está diseñado para ser un lenguaje de intercambio de datos que es legible para el ser humano y fácil de parsear y utilizar por los ordenadores. Se calcula que JSON se analiza hasta cien veces más rápido que XML en los navegadores pero, a pesar de sus afirmaciones sobre el rendimiento, los argumentos contra JSON incluyen la falta de soporte de espacios de nombres, falta de validación de la entrada y los inconvenientes de extensibilidad. Su conjunto de claves es independiente de todos los demás objetos, incluso excluyendo el anidamiento. Además, JSON utiliza contexto para evitar la ambigüedad [26], como puede verse en la Figura 61.

```
{  
  "firstname" : "John",  
  "lastname" : "Smith"  
}
```

Figura 61: Ejemplo JSON

Hay muchos papers que analizan qué algoritmo es más eficiente, pero por flexibilidad, comodidad y legibilidad, se eligió JSON, ya que tiene gran soporte en Python para usarlo como formato de comunicación.

5.4.2 Protocolo de comunicación entre componentes

5.4.2.1 Comunicación RaspBerry - Arduino por puerto serie

La comunicación asíncrona a través del puerto USB (Figura 62) intenta evitar la transmisión de un reloj de sincronización entre el emisor y el receptor. Cada byte de datos se transmite en un paquete o trama de bits. Las tramas se crean añadiendo bits de sincronización y paridad a los datos. La trama comienza con 1 bit de inicio, continúa con 5-9 bits de datos, seguidos de 0-1 bits de paridad y 1-2 bits de parada. La frecuencia de reloj del emisor y la frecuencia de reloj del receptor deben ser iguales es decir, deben utilizar la misma tasa de bits fija (periodos de reloj por segundo) o bps (bits por segundo) para que la comunicación pueda tener lugar. El baud más popular es el de 9600 bps. Cuanto más alta la tasa de bauds, más rápido se transmiten los datos, sin embargo, para la mayoría de los microcontroladores, la velocidad no superará los 115200 bps (por encima de esa velocidad la comunicación empieza a dar errores, ya que los relojes no proporcionan los periodos de muestreo adecuados). El protocolo es altamente configurable y una parte crítica es asegurarse de que tanto el transmisor y el receptor en el bus serie utilicen el mismo protocolo y los mismos parámetros [27].

Debido a su facilidad de uso y a que Python tiene una librería que facilita mucho la comunicación por puerto serie, hemos elegido este método para comunicar el cerebro con la médula espinal del proyecto.

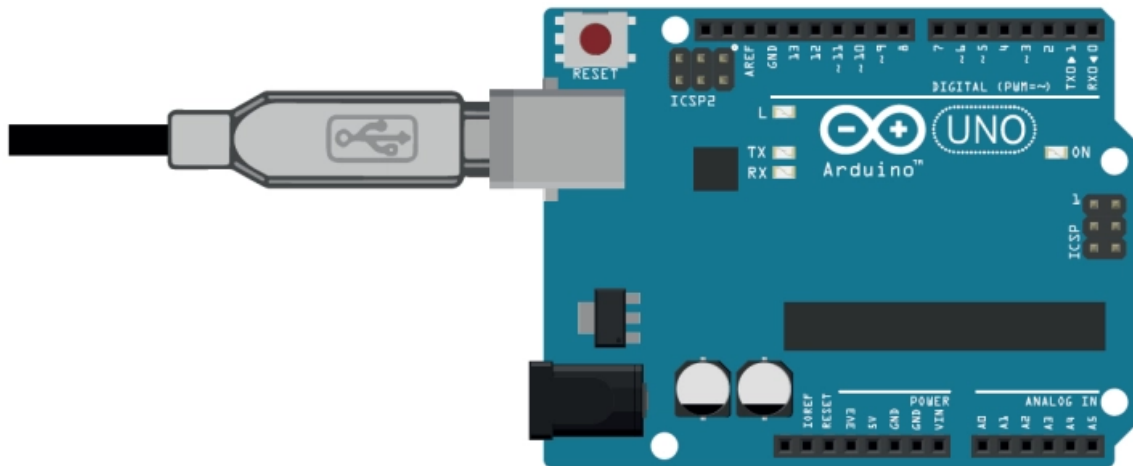


Figura 62: Conexión por USB de Arduino

5.4.2.2 Comunicación A* - Interfaz por MQTT

MQTT es un protocolo Push estandarizado de publicación/suscripción que fue lanzado por IBM en 1999. MQTT fue planeado para enviar datos de forma precisa bajo la condición de un largo retraso en la red y un bajo ancho de banda. En MQTT para la comunicación, el intercambio de una serie de paquetes de control de manera específica. Hay catorce paquetes de control, como se ve en la Figura 63. Cada uno de ellos contiene tres partes [28].

Fixed header present in all MQTT Control Packets
Variable header present in some MQTT Control Packets
Payload present in some MQTT Control Packets

Figura 63: Ejemplo de paquetes de control de MQTT

Conceptos básicos de MQTT que nos sirven para nuestro proyecto:

1) Publicar/suscribir: En el protocolo MQTT, el publicador publica mensajes y los usuarios se suscriben a topics que son comúnmente considerado como un modelo Publish/Subscribe. El suscriptor se suscribe a topics particulares que están relacionados con ellos y por ello recibe todos los mensajes que se publican en esos topics. Por otro lado, los clientes pueden publicar mensajes a los topics, de tal manera que permiten a todos los suscriptores acceder a los mensajes de esos temas.

2) Topics y suscripciones: En MQTT, los publicadores publican mensajes a topics que pueden ser considerados como objeto del mensaje. El suscriptor, por lo tanto, se suscribe a los topics para obtener mensajes específicos. Las suscripciones de topics pueden ser expresas, lo que restringe los datos que se recogen en un tema concreto.

Arquitectura del protocolo MQTT:

La arquitectura típica de MQTT puede dividirse en dos componentes principales como se muestra en la Figura 64. Cada componente se describe brevemente a continuación.

1) Cliente: El cliente puede ser un editor o un suscriptor y siempre establece la conexión de red con el servidor (Broker). Puede hacer lo siguiente:

- Publicar mensajes para los usuarios interesados.
- Suscribirse en el sujeto interesado para recibir mensajes.
- Darse de baja para extraer de los sujetos suscritos.
- Desvincularse del Broker.

2) Broker: El broker controla la distribución de la información y se encarga principalmente de recibir todos los mensajes del editor, filtrarlos, decidir quién está interesado en ellos y enviar los mensajes a todos los clientes suscritos. Puede hacer lo siguiente:

- Aceptar las solicitudes de los clientes.
- Recibe los mensajes publicados por los usuarios.

- Procesar diferentes peticiones como Suscripción y darse de baja de los Usuarios.
- Después de recibir los mensajes del editor los envía a los usuarios interesados.

Un ejemplo de funcionamiento se muestra en la Figura 65.

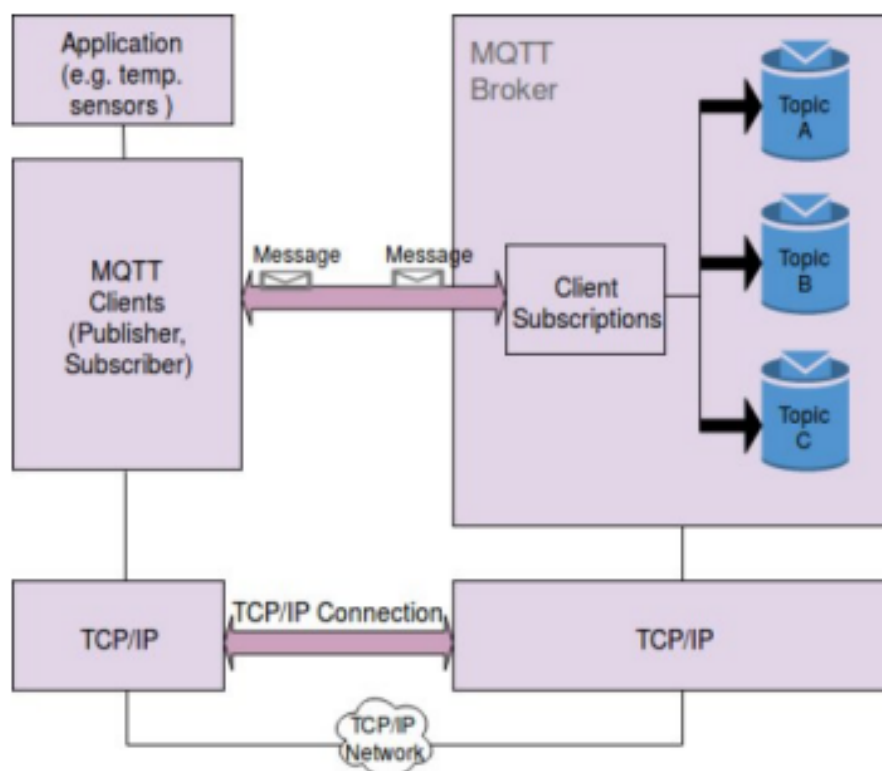


Figura 64: Ejemplo Arquitectura MQTT

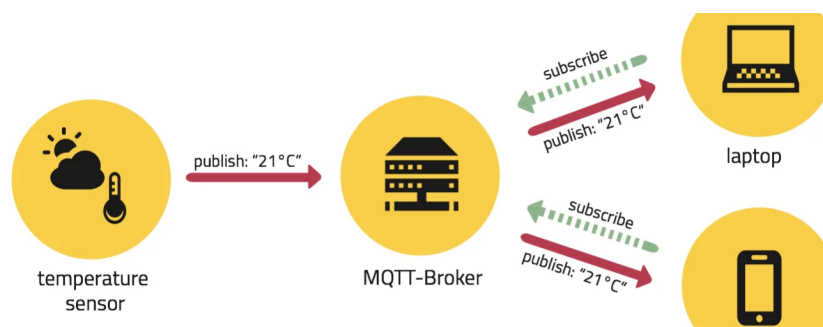


Figura 65: Ejemplo protocolo pub/sub con broker

Hay varios brokers que se pueden utilizar, en este caso va a ser Mosquitto (Figura 66) ya que es Open-Source y ofrece lo mismo que un broker de pago.



Figura 66: Logo de Mosquitto

5.5 Implementación del sistema

En este apartado se hará una especificación técnica con detalle de cada uno de los componentes que se han desarrollado para conseguir el objetivo marcado, tales como, algoritmos, entornos de desarrollo, dispositivos necesarios, configuraciones y scripts ejecutables.

Para la realización de estos apartados es imprescindible tener a punto la RaspBerry configurada, esta instalación y configuración se explica en el Anexo 1.

5.5.1 Preparar script Arduino para recibir información

El robot del que se parte tiene un excelente código que le permite realizar toda serie de movimientos y rotaciones en todos los ejes posibles, para este trabajo sólo se necesitaba que se moviera en las 4 direcciones ordinarias (adelante, atrás, izquierda y derecha) y además, que rotara en ambas direcciones para direccionar la cabeza y andar hacia adelante por donde quisiéramos.

Para realizar estos cambios se utilizó el Arduino IDE, es un software de código abierto que se utiliza principalmente para escribir y compilar el código en el módulo Arduino [29], además este sirve para subir el código a la placa, por lo que era el idóneo para usarlo, ejemplo de la interfaz en la Figura 67.

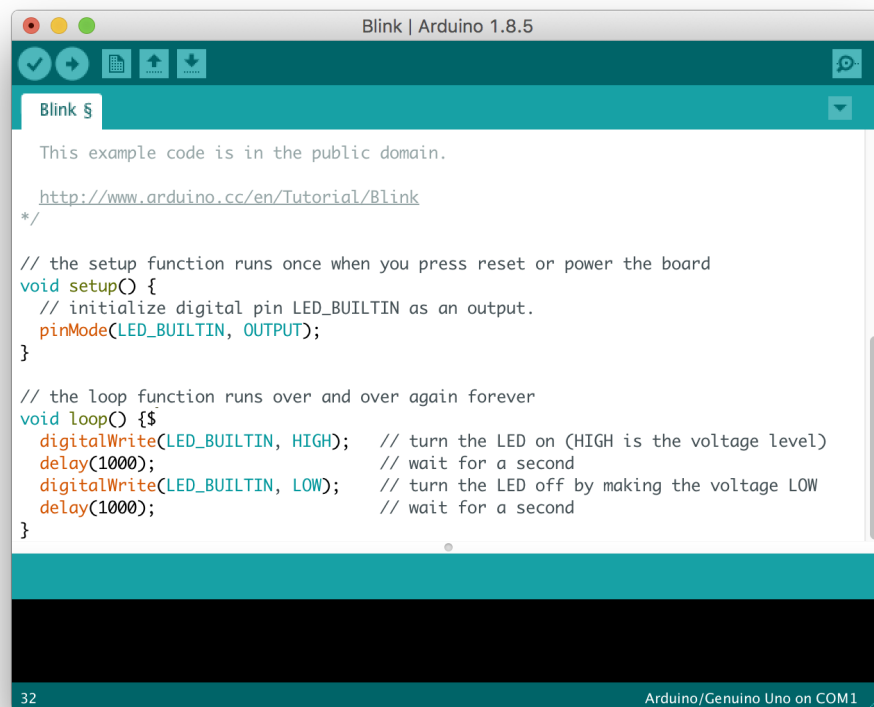


Figura 67: IDE de Arduino

Previamente se tuvo que hacer un trabajo de investigación sobre cómo funcionaba el código metido en el Arduino, ya que a partir de ese, se tenía que realizar las modificaciones oportunas para su requerimiento.

Se tenía la información de qué función realizaba cada botón y joystick del mando, interesaban la serie de movimientos que se ha indicado anteriormente por lo que se consultó y todos ellos se realizaban únicamente con los joystick, como se vió en la Tabla 1.

Lo primero que hubo que hacer fue conectar el Arduino por puerto serie al ordenador, para así poder monitorizar el puerto serie y ver que información circulaba por el código mientras se ejecutaba, tras poner unos cuantos `Serial.println` y probar a mover los joystick del mando mientras se ejecutaba, se llegó a la conclusión de que el código se quedaba a la espera de que el joystick pasara cierto rango para activar o desactivar el movimiento, no había progresión en la velocidad ni nada por el estilo, si pasaba cierto valor se movía y si no, no, por lo que esto facilitaba mucho más nuestro trabajo.

El valor que daba paso a la activación se comprobó gracias a la salida por el puerto serie, por lo que ya se sabía que valor había que manejar para poder activar el movimiento. Lo siguiente fue implementar un método en el script que estuviera siempre a la espera de lo que le llegara por el puerto serie, teniendo en cuenta como funcionaba un script de Arduino, se llegó a la conclusión de que lo mejor era crear una variable global en la cual se almacenara algún valor y que en cada iteración de la función `loop()` comprobase ese valor, dependiendo del valor haría un movimiento u otro jugando con los valores de paso mencionados anteriormente.

La implementación consistió en que la variable global almacenara un carácter, se le daba un valor el cual no activara nada, en este caso una "n", a continuación al comenzar la función `loop()`, se quedaba a la espera de que le llegara información por el puerto serie, si le llegaba un carácter distinto al actual, lo cambiaba, si no, no lo cambiaba, así se podía controlar al robot como se quisiera con solo enviar un carácter.

Los comandos que se eligieron para los movimientos seguirían el mapping WASD, popularizado por Quake, WASD (también conocido como Was-duh, WSAD o ASDW) es un conjunto de cuatro teclas en el lado izquierdo de un teclado de ordenador QWERTY o QWERTZ. W/S controla el avance y el retroceso y A/D controla izquierda y derecha, mapping el cual es utilizado en la mayoría de los juegos modernos [30], y hace más intuitivo el uso a la hora de trabajar con el robot. Además se añadió la Q para rotar a la izquierda y la E para rotar a la derecha.

Tan solo quedaba modificar un poco más el código para comprobar la variable global y dependiendo del carácter, realizar un movimiento u otro, tras hacer ingeniería inversa al código y ver donde estaba la función desde la cual se decidía el movimiento dependiendo del valor de los joystick del mando, solo hubo que modificar las sentencias condicionales donde comprobaban el valor de los joystick y cambiarlo por comprobar qué carácter había en la variable global.

5.5.2 Implementación del servicio API/REST

Para la implementación del servicio API/REST, se requirió de la librería Flask, mencionada anteriormente, el servicio constaba de una implementación muy sencilla, gracias a lo desarrollado en el apartado anterior, sólo se tenía que enviar un carácter dependiendo del movimiento que se quisiera que hiciera el robot.

Además Flask, tiene una propiedad que consiste en que se puede extraer partes de la URL con la que haces el GET al servicio, por lo que a través de la URL se podía enviar la información que se quisiera y el servicio lo trataría, por lo que se decidió que la URL para el control de robot fuera: `http://ip-raspberry:5000/robot/movimiento`.

En la parte llamada “movimiento” sería donde se indicara en formato WASD el movimiento que se quisiera que realizara el robot, por ejemplo:

Movimiento adelante: `http://ip-raspberry:5000/robot/w`

Movimiento atrás `http://ip-raspberry:5000/robot/s`

El servicio se encargaría de coger el carácter correspondiente al movimiento que se manda en la URL y enviarlo por puerto serie al Arduino. Mandando cualquier carácter que no fuera uno de control, el robot detiene su movimiento, para así evitar posibles errores a la hora de controlar el robot.

El servicio devuelve en formato JSON el movimiento que se ha enviado al robot, para así asegurarse de que el comando se ha enviado, este JSON se muestra en el navegador con la plantilla que viene por defecto en Flask (Figura 68).

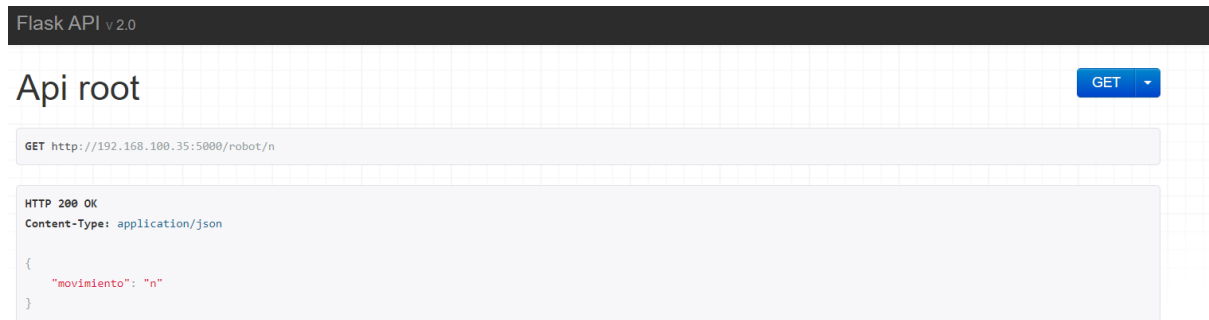


Figura 68: API de Flask cuando recibe un JSON

5.5.3 Implementación de reconocimiento de marcas ArUco

Para esta parte, se ha utilizado OpenCV, OpenCV no es más que las siglas en inglés de “Biblioteca de Visión por Computador de Código Abierto” y es una librería de código abierto para el tratamiento de imágenes que incluye varios cientos de algoritmos de visión por computador en tiempo real. Como se encuentra bajo licencia BSD permite su uso tanto para fines comerciales como académicos.

Del mismo modo que ocurre con Python, es multiplataforma al tener versiones para Windows, GNU/Linux, Mac OSX, iOS y Android. Está escrita en lenguaje C/C++ de manera optimizada y orientada a la eficiencia computacional con un enfoque especial en aplicaciones de tiempo real.

Posee una estructura modular, estando agrupadas la mayoría de las funciones de la biblioteca, en los siguientes módulos:

- Core: Módulo donde se definen las estructuras de datos básicas del resto de módulos.
- Imgproc: Módulo de procesamiento de imágenes donde nos podemos encontrar transformaciones geométricas de las imágenes, conversión del espacio de color, histogramas, etc.
- Video: Módulo para el análisis de video que incluye algoritmos de seguimiento de objetos, para la estimación del movimiento y extracción del fondo.

- Calib3d: Algoritmos básicos de geometrías de múltiple vista, como calibración de cámaras o reconstrucción 3D entre otros.
- Features2d: Detectores de características, descriptores y comparadores.
- Objdetect: Algoritmos de detección de objetos o instancias de clases predefinidas (por ejemplo: caras, nariz, ojos, personas, coches...)
- Highgui: Todo lo relacionado con la interfaz gráfica de OpenCV y el conjunto de funciones que permiten importar imágenes y vídeo.
- Gpu: Algoritmos de diferentes módulos de OpenCV con aceleración de GPU

La detección de las marcas tiene un procedimiento, el cual se ha comentado por encima en apartados anteriores, en este vamos a profundizar más:

1. Primero se aplica un umbral adaptativo hasta obtener los bordes de la marca.
2. Tras encontrar los bordes, se buscan los contornos de la marca, una vez encontrados, no solo se detectan los contornos de la marca, si no que se detecta una gran cantidad de bordes, de los cuales muchos de ellos son no deseados. Los siguientes pasos se encargan de eliminar esos contornos.
3. El primer paso para eliminar los bordes no deseados es eliminar los bordes con un pequeño número de puntos.
4. Seguidamente, se hace una aproximación poligonal de los contornos encontrados y se mantienen aquellos que tienen 4 esquinas. Se enumeran las esquinas en sentido contrario a las agujas del reloj.
5. Se eliminan los rectángulos demasiado cercanos
6. Para el identificador de la marca se elimina la perspectiva de proyección para obtener una vista frontal del área rectangular usando una homografía, a continuación se emplea el método Otsu para calcular el umbral óptimo que separará la marca del resto, los algoritmos de Otsu asumen una distribución bimodal y encuentra el umbral que maximiza la varianza extra-clase manteniendo baja la varianza dentro de la clase, después identifica el código

de la marca y por último, para los marcadores válidos, refina las esquinas mediante interpolación de subpíxeles.

Como resultado final nos queda la imagen de la Figura 69:

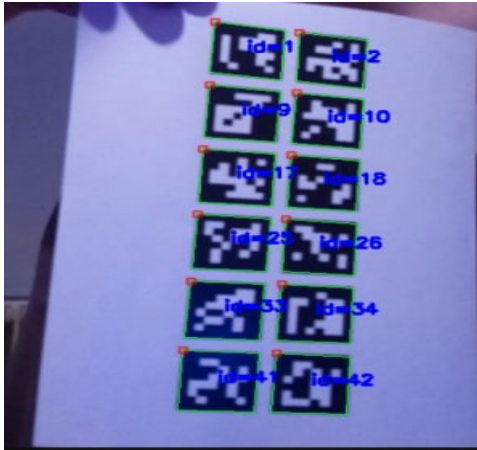


Figura 69: Marcas ArUco reconocidas

Una de las opciones que deja hacer esta librería es calibrar la cámara, esta opción es útil cuando se quiere estimar la posición de la cámara, para ello hay que ayudarse de una serie de imágenes de calibración, que se muestran en la Figura 70, las cuales procesa y con ello obtienen los parámetros intrínsecos y extrínsecos de la cámara (el tamaño del sensor en píxeles y milímetros, la distancia focal del objetivo, los coeficientes de distorsión, la ubicación de la cámara), dicha calibración está más detallada en el Anexo 3.



Figura 70: Ejemplo de imágenes que se utilizan para la calibración

Para poder visualizar las imágenes procesadas, se necesita la ayuda de Flask, se implementa una URL con la cual se pueden ver las imágenes procesadas en tiempo real a través del navegador, esta URL sería <http://ip-raspberry:5000/video>.

Además, se implementó que por terminal se mostrara la posición de las marcas reconocidas, esto ayudaría más adelante para la navegación autónoma (Figura 71).

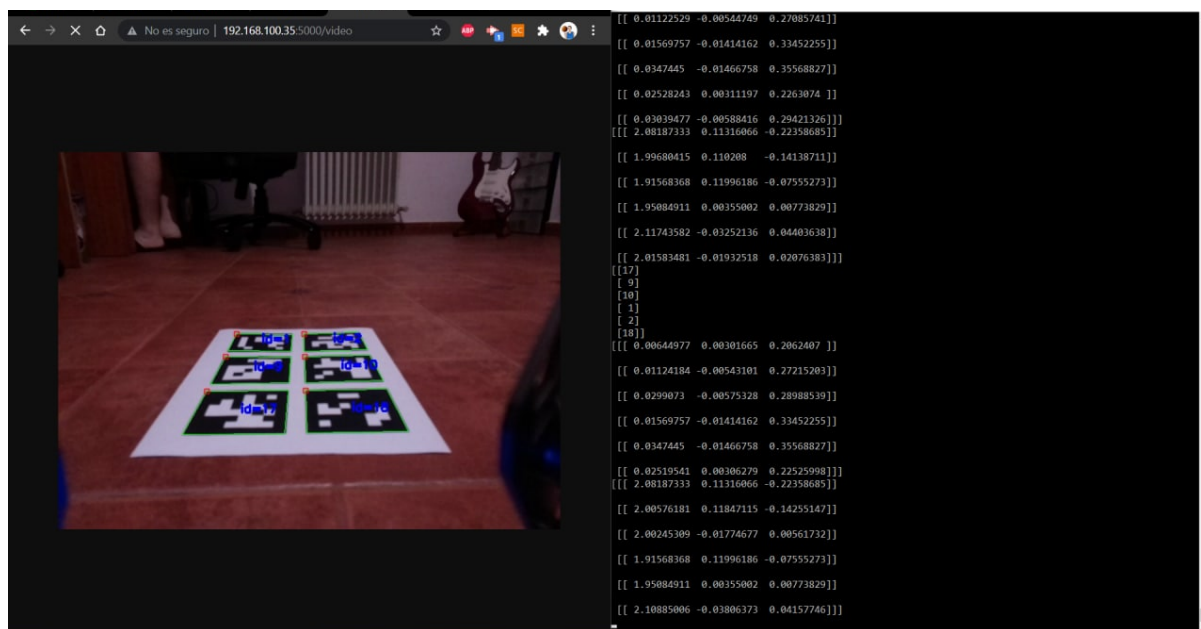


Figura 71: Marcas reconocidas (izquierda) e ID, posición y rotación de las mismas (izquierda)

5.5.4 Implementación de navegación autónoma

En esta parte se ponen a funcionar todas las partes a la vez simultáneamente, empezando por el algoritmo de navegación autónoma, como se mencionó en apartados anteriores, es un A*. Debido a cómo se planteó el funcionamiento de esta parte, se decidió dividir el algoritmo en 3 partes, un script que representara el mapa por donde va a moverse el robot, otro script que fuera el que hiciera el razonamiento sobre a qué posición del mapa tenía que ir y el último script, el que recibiría la posición a la que tiene que ir el robot y actuara en consecuencia enviando los comandos necesarios al robot para llegar a esa posición.

El mapa sería una matriz de 5x5 en un inicio, pero que se cambiaría después a 3x3 para ahorrar tiempo durante la fase de pruebas, cada posición de la matriz corresponde con una marca, al tener cada marca un identificador, cada casilla tendría un identificador de marca distinto, por lo que ya se podía saber dependiendo de a qué posición queríamos ir, a qué marca tenía que acercarse. En la Figura 72 se puede ver un ejemplo de como se distribuirían las marcas en el suelo y en la Figura 73 la correspondencia entre ID de marca y casilla.



Figura 72: Distribución de las marcas en el suelo

25	24	23	22	21
20	19	18	17	16
15	14	13	12	11
10	9	8	7	6
5	4	3	2	1

Figura 73: Equivalencia de casillas y marcas

El tamaño de la matriz, la posición y rotación inicial del robot y la casilla que tiene que alcanzar el robot se inicializan en el script del mapa, una vez inicializan los 3 scripts, funcionarían tal que:

El mapa le indica al script razonador la posición y la rotación del robot, además de la casilla meta, el razonador calcula la ruta que tiene que seguir mediante el algoritmo A*, una vez calculada la ruta, le indica al script que se encarga de mover al robot, a que casilla que tiene que ir y si tiene que rotar previamente o no, en este momento es cuando entra en juego el reconocimiento de las marcas, el script que se encarga del movimiento consulta qué marca corresponde a la casilla que tiene que ir, si tiene que rotar, rotaría hasta encontrar la marca y pararía. Tras eso, se centraría respecto a la marca para ir en línea recta hasta ella, durante el recorrido hasta la marca, va comprobando 2 medidas, la distancia y la rotación respecto a la marca a la que tiene que ir, la distancia nos sirve para comprobar cuando tenemos que parar al robot y la rotación, para ir corrigiendo al robot por si en un principio no se hubiera centrado lo suficiente, si en algún momento durante la rotación o la traslación no se localizara a

la marca que tiene que ir, el robot andaría un poco hacia atrás con el fin de volver a encontrar a la marca. Una vez llegue a la marca, le comunica al mapa que ha llegado a la posición con la rotación y la posición deseada, el mapa actualiza la posición del robot en la matriz y vuelve a mandar la información al script razonador, así hasta que se haya alcanzado la meta y se haya vuelto a la posición de inicio. Esto viene explicado con más detalle en el flujograma de la Figura 74 y en la Figura 75, el diagrama de secuencia general de comunicación.

Todo y cada uno de los movimientos implicados en la navegación autónoma los realiza el script ejecutor mediante peticiones GET al servicio API/REST, que tiene que estar activo antes de iniciar la autonavegación.

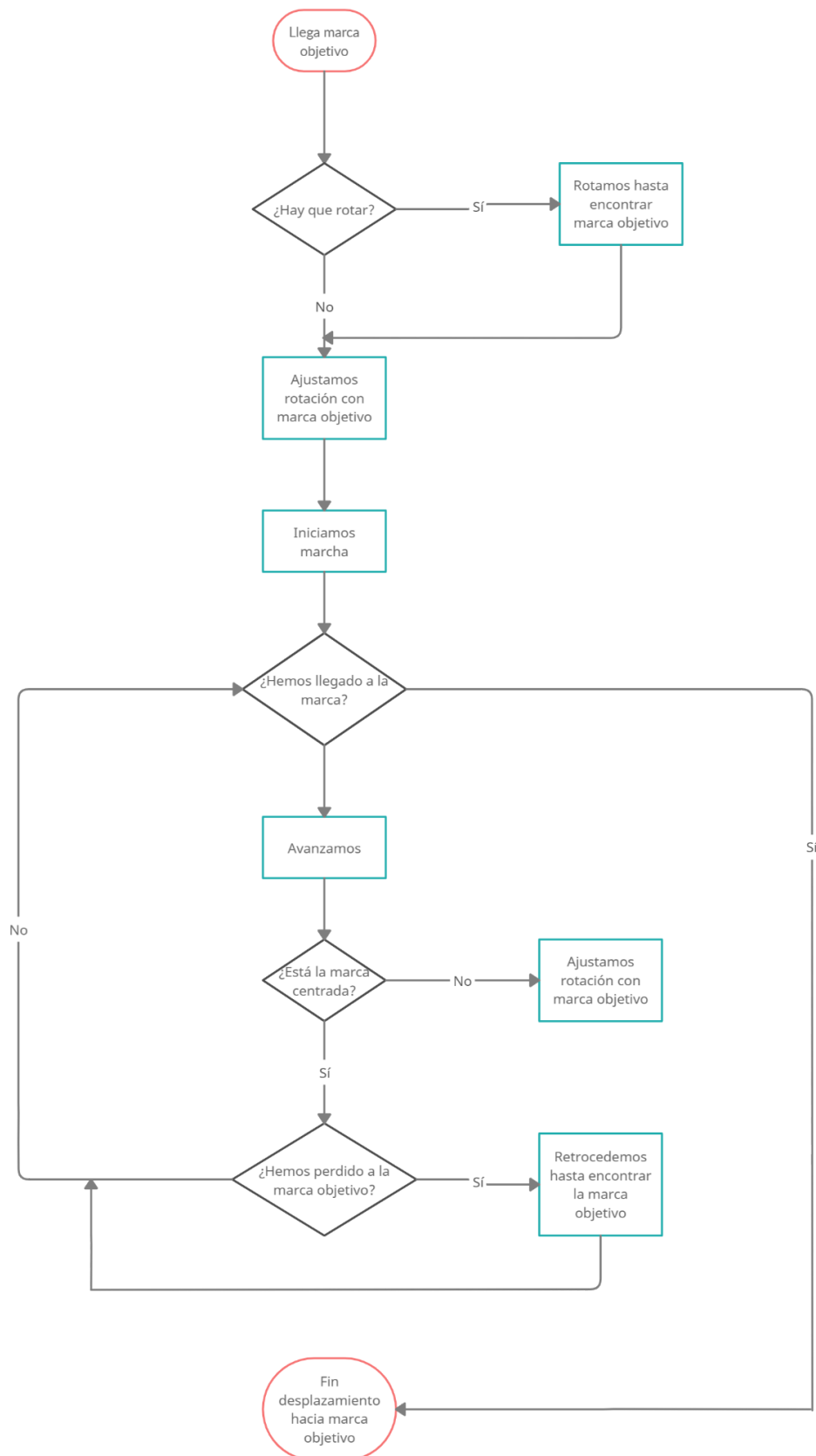


Figura 74: Flujograma de aproximación a una marca

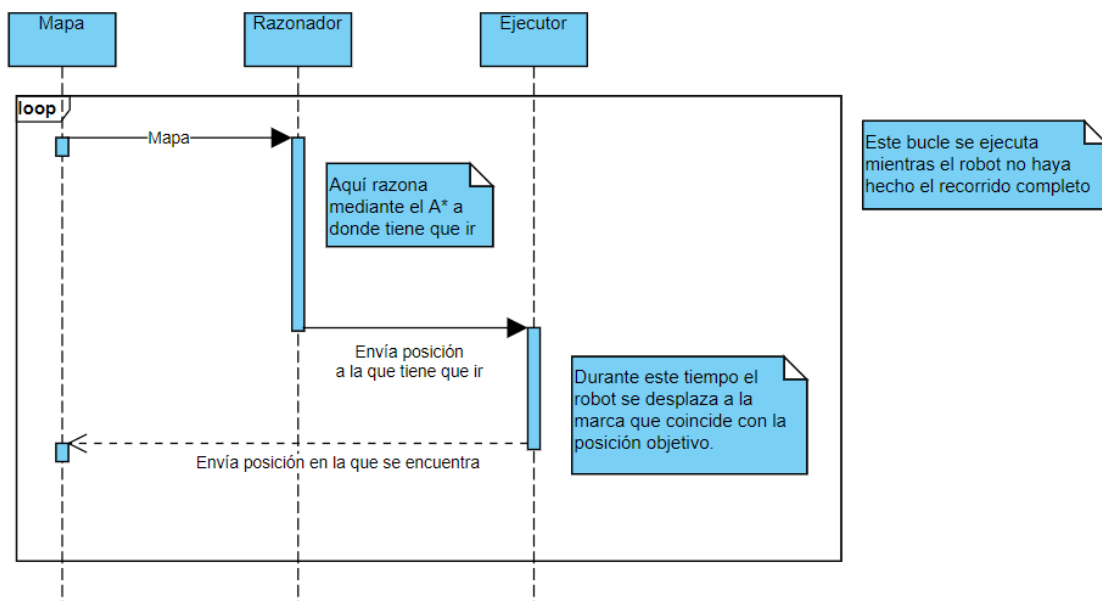


Figura 75: Diagrama de secuencia del sistema de navegación

5.5.5 Implementación de interfaz

Puesto que este proyecto está destinado a actividades de difusión que potencien el interés en la electrónica y en la informática, se decidió incluir una interfaz gráfica basada en una réplica en 3D de Marte, donde el robot tendría una serie de elementos que tiene que recolectar, para así hacer más llamativa la propuesta y no solo hacer una web con botones para controlar al robot, todo estaría enfocado a talleres y charlas educativas, se mostraría al alumnado el control de forma remota del robot hexápodo, pudiendo observar en el entorno simulado la superficie a la que estaría destinado para el análisis y extracción de diversos materiales.

La premisa para hacer más interesante la propuesta era:

Se ha enviado un robot explorador a Marte para extraer conocimiento sobre este planeta, por ejemplo, recoger materiales y analizarlos, recorrer áreas concretas, saber enfrentarse a situaciones inesperadas como falta de batería, caída de meteoritos, etc. Con el fin de entrenar a una posible inteligencia artificial o a una serie de conductores de ese robot explorador, se ha creado una simulación del entorno donde se destinaría al robot, que incluye una serie de bases, vegetación en

una bóveda controlada, cohetes, etc. Además, se presentan situaciones donde se deberá competir por ciertos materiales con otros robots, que pueden estar siendo controlados por un sujeto o por IA. De este modo, es posible entrenar a los sujetos encargados de nuestro robot y observar el comportamiento de este antes de enviarlo a esta superficie planetaria.

Lo interesante de este proyecto es que se puede controlar al robot tanto manualmente como con el controlador, ya que si se controla utilizando WASD, se verá reflejado tanto en la simulación como en el robot en la vida real, y si se ejecuta el algoritmo de navegación autónoma, al publicar la posición en la que se encuentra el robot, se moverá en ambos sitios a la vez.

El mundo simulado se localiza en la superficie de Marte, donde se encuentra un terreno con grandes variaciones y cráteres de diferentes tamaños, como se puede ver en la Figura 76. Los colores predominantes son cálidos debido a la luz del sol y el color desértico que posee el terreno.



Figura 76: Terreno habilitado para moverse por él

De toda la superficie solo hay explorada una zona, situada de forma aproximada en el centro del mapa. Es de las zonas más llanas del terreno, permitiendo que se hayan situado una base espacial, varias antenas, un cohete, una bóveda que

contiene vegetación y agua, etc. También hay una pequeña zona con meteoritos apilados. En la figura 77 se puede ver un esquema de cómo está formado el mundo.

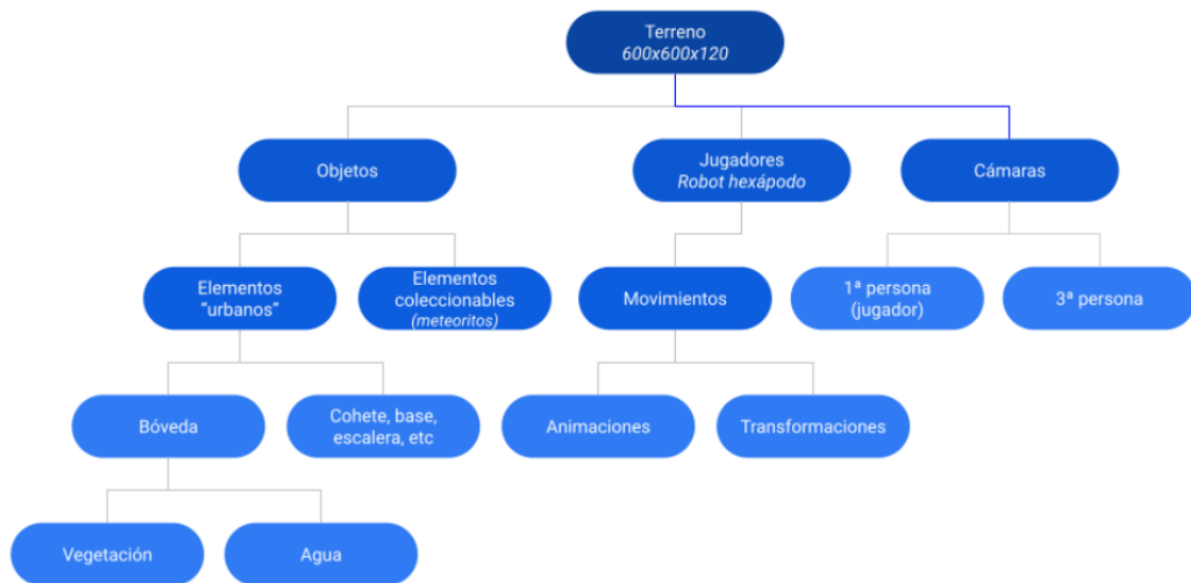


Figura 77: Esquema de composición del mundo

El robot se ha simulado como un robot escorpión, que si se compara con el robot en la vida real, se asemeja bastante por las patas y la posición de la cámara. También se han añadido otros robots con distinto color para simular que hay más agentes allí trabajando (Figura 78).

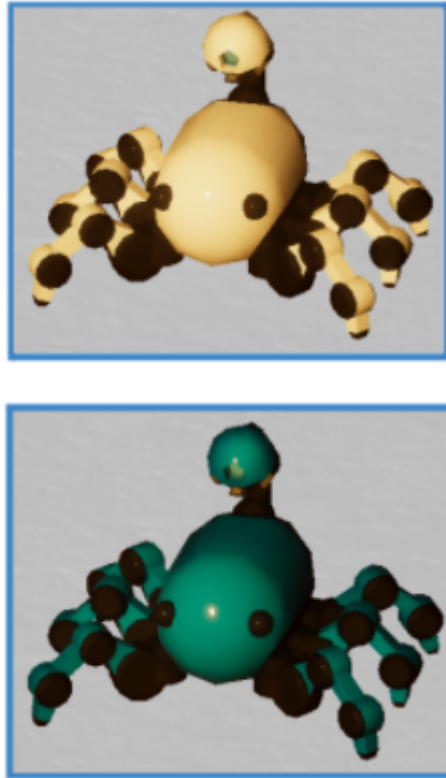


Figura 78: Robots virtuales

Para el terreno se ha hecho uso de un terreno base, se levantó con la herramienta que proporciona Unity y suavizando algunos picos con diferentes pinceles. Asimismo, se ha modificado el color al gusto para que se adecúe a la superficie de Marte, modificando el skybox por defecto por un dynamic physically based sky y añadiendo efecto niebla realista.

Todos los objetos han sido obtenidos de <https://assetstore.unity.com/>, y han sido modificados de acuerdo al estilo del proyecto general (adición de nuevos materiales, eliminación de módulos y adición de nuevas partes). Es importante destacar que TODOS los objetos se han convertido al formato HDRP para poder mostrarlos en alta definición, tal y como se ha realizado con el terreno como se puede apreciar en la Figura 79.

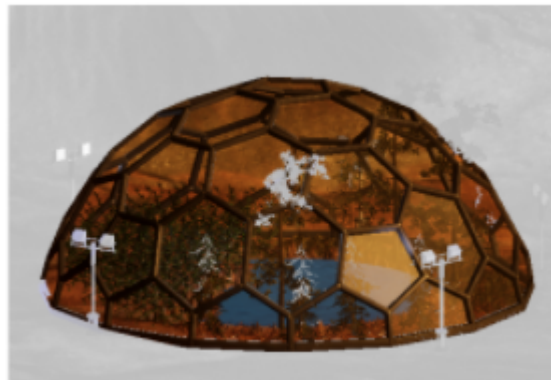
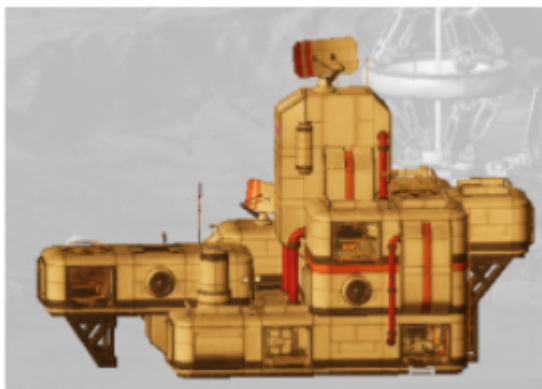
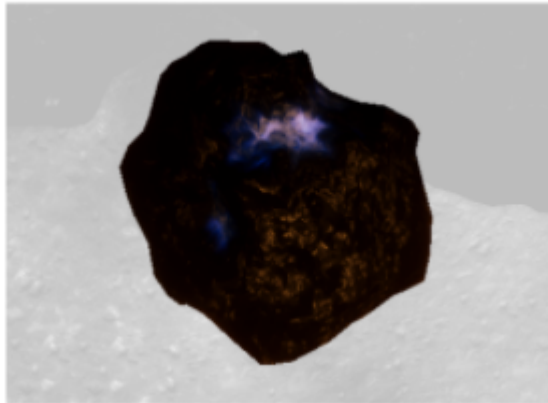
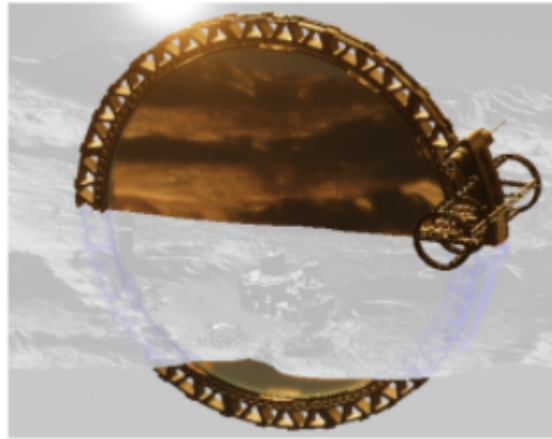
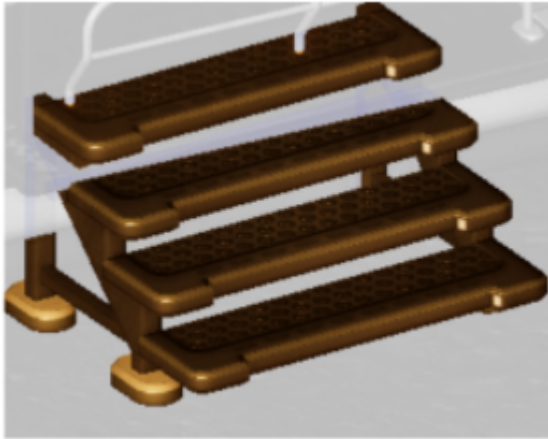


Figura 79: Distintos ejemplos de estructuras incluidas en el mundo

Para la cúpula hay 6 tipos diferentes de vegetación: Pinos altos, pinos pequeños, plantas

trepadoras, arbustos rojos, helechos, césped corto. Pequeño estanque con animación y flujo de agua asociado a la gravedad del planeta.

También hemos puesto una gran cantidad de luces por todo el mapa para mejorar la iluminación y darle un toque más realista (Figura 80).



Figura 80: Luces añadidas al mundo

Una de las luces más importantes es la luz direccional que simule la luz solar (Figura 81).

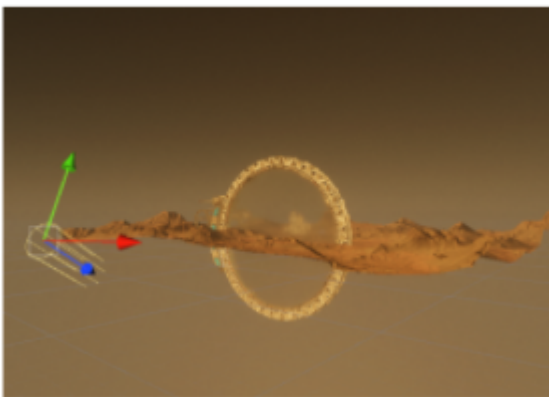


Figura 81: Luz solar añadida en el mundo

Para darle más realismo al mundo, se han añadido diversas animaciones en diferentes objetos, entre ellos, el robot hexápodo. Al moverse hacia arriba o hacia

abajo, las patas se mueven hacia delante una a una. Si es izquierda o derecha, se mueven primero las del lado hacia el que va a caminar. Al pasar por un objeto que recoger (meteorito coleccionable), se reproduce una animación. El anillo stargate y la torre eléctrica giran para darle más ambiente al mundo.

Junto a estas animaciones, se han añadido diversos sistemas de partículas como tormentas de arena, arena en suspensión levantada al caminar el robot, estrellas fugaces y humo que se genera en la salida de los ventiladores de la base espacial, estas partículas se pueden ver en las Figuras 82-83.

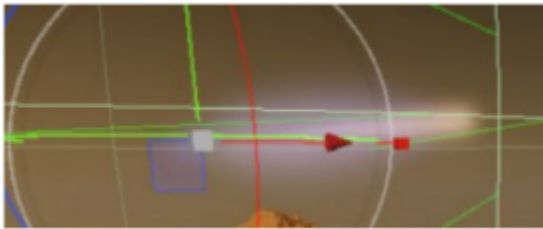


Figura 82: Meteorito hecho con partículas



Figura 83: Humo hecho con partículas

También se han incluido sonidos para que no sea todo tan sobrio, entre ellos, un tema de fondo, sonido del robot al caminar, los reactores de la base espacial suenan conforme te acercas y el sonido que hace el stargate al girar.

Además se han añadido distintas cámaras por las cuales se puede visualizar todo el mundo, en cada cámara, se puede mirar en 360° girando el ratón, las distintas cámaras son:

1ª persona: está la vista que tiene el personaje que se maneja, esta opción se selecciona con la tecla V (Figura 84):

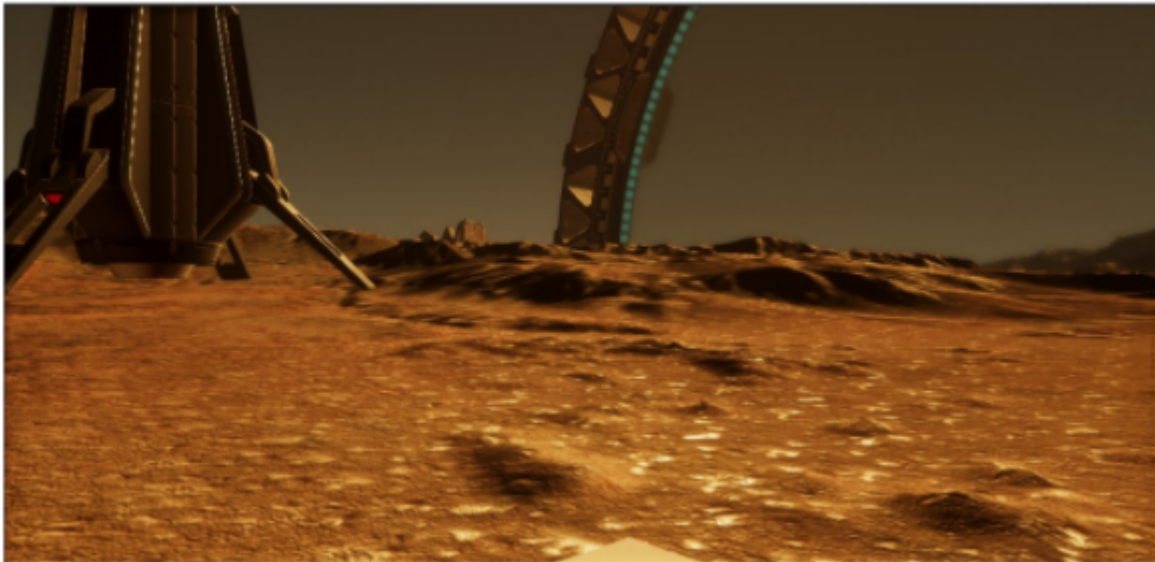


Figura 84: Vista en primera persona

3ª persona: se sigue al personaje pero lo se visualiza desde atrás con la tecla C (Figura 85), o bien lo se observa desde arriba todo el terreno, letra B.



Figura 85: Vista en tercera persona

Por último para finalizar la simulación, se añadió una simulación de horas de luz diarias, si se pasa el tiempo suficiente dentro de la simulación se ve un ciclo horario completo, esto se logró utilizando un “physically based sky” configurado en lugar del skybox por defecto, se han simulado el paso de las horas del día estableciéndose como dinámicas. Además, se ha fijado la exposición para que no cambie a nuboso u otro tipo de clima, pero se podría dejar de forma automática o detectando el actual a través de la cámara. También se ha añadido niebla para dar un efecto más realista.

Para finalizar la implementación de este apartado, se tuvo que implementar un script, con el cual se podía mover al robot y dependiendo del movimiento que se enviara al robot virtual, hiciera una petición GET al servicio API/REST de la Raspberry, con esto se conseguía que ambos robots se movieran simultáneamente.

5.6 Pruebas

Tras terminar la implementación del proyecto, hay que poner a prueba el proyecto completo para comprobar su funcionamiento y los posibles errores que puede dar, para ello había que irse a una superficie relativamente plana y poner las marcas en forma de matriz como se ha mencionado anteriormente, en este caso, se hizo con

una matriz de 3x3 para ahorrar tiempo en el proceso de ejecución, al reducir el tamaño de la matriz que íbamos a utilizar, había que seleccionar las marcas adecuadamente para que el robot no se confundiera de marcas al buscar la posición, esta selección se puede ver en la Figura 86.

25	24	23	22	21
20	19	18	17	16
15	14	13	12	11
10	9	8	7	6
5	4	3	2	1

Figura 86: Cuadrícula elegida para las pruebas

La marca con id=1, correspondía con la posición 0,0 del mapa, por lo que el robot tenía que comenzar desde esa marca, se puso como objetivo la marca 8, por como funciona el algoritmo, el robot tenía que empezar girado 90 grados a la derecha, para así saber cuando parar de girar y centrarse en la marca que le correspondía. La ruta que tenía que hacer el robot era la que corresponde con la Figura 87.

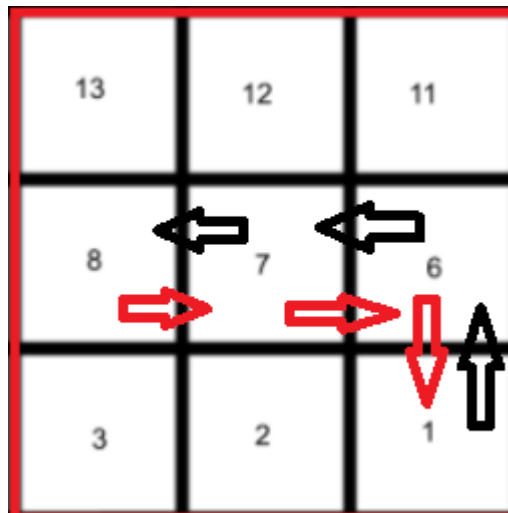


Figura 87: Recorrido que tenía que realizar el robot

El color negro simboliza la ida y el rojo, la vuelta, el primer fallo se hizo notar pronto, había que separar las marcas lo suficiente para que el robot pudiera localizarlas adecuadamente, tras prueba y error, se llegó a la conclusión de que la mejor distancia entre marcas era de 40cm.

El mayor de los problemas que se encontraba el robot era que mientras se desplazaba a la marca objetivo la perdía de vista, ya sea porque está demasiado cerca o porque simplemente el algoritmo de reconocimiento no la detecta, pero este fallo se subsanaba con la cláusula de que si no encontraba la marca objetivo en algún momento, se desplazara hacia atrás, así se podía asegurar de que ya fuera porque está demasiado cerca, o que el reconocimiento no había funcionado bien, se le daba otra oportunidad al algoritmo al rehacer los pasos.

Sin embargo una de las situaciones que más problemas ha traído fue que, mientras el robot se movía, sobre todo en las rotaciones, pisaba las marcas adyacentes a la marca donde estaba realizando la rotación, una de las soluciones fue pegar las marcas al suelo con cinta adhesiva, si en un futuro se desarrollara más este proyecto, una buena idea sería hacer una especie de plataforma con las marcas impresas, para así evitar movimientos innecesarios de estas.

La rotación de las marcas sobre su mismo eje no suponía a grandes rasgos un fallo, las primeras veces que se puso a funcionar al sistema, se ponían las marcas con la esquina superior izquierda en la misma orientación para todas las marcas, más tarde, se giraron algunas de ellas para ver si afectaban a la navegación, dependiendo de la ejecución, le costaba algunas veces centrarse en la marca adecuadamente, por lo que la mejor opción era colocar las marcas en el mismo sentido.

5.6 Resultados

Tras comprobar que el sistema de navegación funcionaba correctamente, faltaba ejecutar la simulación y ver si el robot funcionaba conjuntamente a ésta, al primer intento funcionó todo correctamente, lo único que hubo que hacer fue ajustar la velocidad de rotación del robot en la simulación para que se ajustara más a la velocidad real del mismo, el resultado impresiona bastante porque se está acostumbrado a ver simulaciones en la pantalla pero no que lo que sucede en la pantalla, suceda a la vez al lado en la vida real.

Una vez se tenía todas las variables a favor, pusimos a funcionar el robot 20 veces para ver el porcentaje de acierto, de las 20 ejecuciones, 17 llegó correctamente a la meta, 1 vez no se ajustó lo suficiente a la marca final (se quedó más atrás de lo necesario pero finalizó la ejecución correctamente) y las 2 que falta, perdió de vista a la marca objetivo y al moverse para atrás, se desplazó tanto que ya fue imposible que volviera a ver la marca que le correspondía.

Con un 85% de fiabilidad en el sistema, se puede dar por fiable al sistema, una posible mejora hubiera sido implementar una rutina que desarrollara más aún qué tendría que hacer el robot si no encontrara la marca objetivo.

6. Conclusiones y líneas de trabajo futuras

Para finalizar el trabajo, se va a dar unas conclusiones finales sobre el mismo y distintas líneas de trabajo futuras por si se sigue desarrollando sobre este robot.

6.1 Conclusiones

En este apartado se presentan las distintas conclusiones obtenidas con este trabajo, así como una conclusión final.

6.1.1 Robot Hexápodo

Desde que vi este robot hasta la finalización de este trabajo, me ha parecido fascinante el trabajo que hizo al autor del mismo, trabajar con él ha sido de las menores preocupaciones que he tenido, funcionamiento muy estable y código muy claro y legible, además del hardware, que a pesar de haber estado en varios eventos ya, sigue estando perfectamente, para ser la parte que más se salía de mi círculo de confort, ya que no he cursado nada de robótica, ha sido una de las partes que más he disfrutado porque después de tantos años escribiendo código en un pantalla, puedes ver cómo el código desarrollado funciona en la vida real y es una de las sensaciones más gratificantes que existen, si no fuera poco cuando te funciona un programa normal que no sale del ordenador.

6.1.2 Servicio API/REST

Profundizar en esta tecnología me ha ayudado a ver lo flexible que es, desde poder coger elementos que tenemos en la URL, hasta ver lo rápido que se puede montar un servicio de este tipo con unas pocas líneas de código. No ha sido la tecnología que más provecho le he sacado en este proyecto, pero sí la que más ha ayudado a facilitar su funcionamiento, tanto que utilizar un servicio API/REST para controlar el robot hace que tenga una interfaz multimodal, con la cual, cualquier sistema que pueda hacer peticiones GET podría comunicarse con el robot y controlarlo, por lo que estoy muy satisfecho de haberla usado.

6.1.3 Marcas ArUco

La visión por computador es una disciplina que siempre me ha llamado la atención, porque a grandes rasgos, es darle unos ojos a nuestro equipo y que procese lo que ven esos ojos, como si fuera nuestro propio cerebro, me parece una tecnología que todavía le queda mucho recorrido y que va a ser clave en los próximos años. Especificando más, las marcas ArUco, me han parecido lo más fascinante de este proyecto, el saber la posición de las marcas o de la cámara con solo ponerlas delante de la cámara y procesarlas me parece de lo más interesante. Como hemos visto al principio, se están usando en proyectos que están en la ISS(Estación Espacial Internacional), que posiblemente sea el centro donde más avanzada está la investigación, en resumen, haber adquirido conocimientos sobre esta disciplina, me ha parecido lo más útil de este proyecto.

6.1.4 Interfaz gráfica

Como apasionado por los videojuegos que me reconozco, haber trabajado con Unity me ha parecido una gran idea para ayudar a este proyecto a impulsar la curiosidad de los que en un futuro cojan este robot, una interfaz gráfica como esta hace mucho más atractivo al proyecto que si se hubiera quedado en una simple interfaz con botones en las direcciones en las que se mueve el robot. Otra disciplina en la cual, me alegro mucho de haber aprendido aunque sea unos mínimos de cómo funciona.

6.1.5 Conclusión final

Como conclusión final diría que no puedo irme más contento de haber podido aprender de tantas y tan diversas tecnologías, y haber sido capaz de juntarlas para formar este proyecto tan interesante y fascinante, espero que los usuarios que tengan la oportunidad de probarlo se queden tan alucinados como yo cuando funcionó por primera vez, en cuanto a funcionalidad, tanto en conjunto, como por separado, el sistema no podría haber quedado más estable, que si ponemos a funcionar todas las partes a la vez, el sistema funciona excepcionalmente, si cogemos y ponemos a funcionar las partes por separado, también funcionan sin

problema ninguno, por lo que un sistema tan estable y modular hacen las delicias de cualquiera que trabaje en esta disciplina.

6.2 Líneas futuras

En este apartado se presentan una serie de propuestas, las cuales se podrían desarrollar en próximos proyectos relacionados con este robot.

6.2.1 Mover robot con sensor de ondas cerebrales

Puesto que el servicio API/REST hace que el control del robot sea multimodal y que la idea del proyecto es que sea de exposición para talleres didácticos y demás, no se iba la idea de hacer que alguna interfaz cerebro-ordenador (llamados vulgarmente, sensor de ondas cerebrales), pudiera comunicarse con el robot y controlarlo mediante el control mental, esto haría que los usuarios que lo utilizaran quedaran alucinados, hay muchos modelos de interfaces cerebro-ordenador, algunas de ellas son compatibles con Python por lo que facilita mucho su implementación, se perdería la parte de navegación autónoma, pero como la cámara se puede desmontar fácilmente no supondría un problema.

6.2.2 Realidad virtual con unity

Relacionado con ponerse cascos en la cabeza, para un trabajo relacionado con los gráficos en 3D y realidad virtual, estaría muy interesante utilizar el proyecto de unity desarrollado y añadiendo realidad aumentada, que vieras a la vez el punto de vista del robot en la vida real y en la simulación.

6.2.3 Plataforma con cámara en perpendicular a las marcas

Uno de los grande problemas que se ha encontrado desarrollando el sistema es la lentitud con la que la cámara coge las imágenes, esto sumado a la velocidad del robot, hace que no sea muy preciso y por eso hubo que implementar la parte de que se moviera hacia atrás por si perdía la marca, este problema se solucionaría si se diseñara alguna especie de plataforma donde la cámara, en vez de estar en primera persona, estaría en tercera, justo encima de las marcas, eso junto a otra marca que

tuviera el robot colocada encima, haría un sistema mucho más preciso, esta arquitectura se ha mencionado en el capítulo 2, se trata de la arquitectura Eye-to-hand.

Anexos

En este capítulo se encuentran los distintos anexos a los que se hace referencia en el texto.

Anexo 1: Configuración de RaspBerry

1.1 Instalación y configuración de RaspBian

Para la instalación de RaspBian en la tarjeta SD existen dos vías: una es usando el archivo de imagen de Raspbian y la otra es usando NOOBS (Figura 88). Las cuales la podemos encontrar en la siguiente página:

<https://www.raspberrypi.org/downloads/>

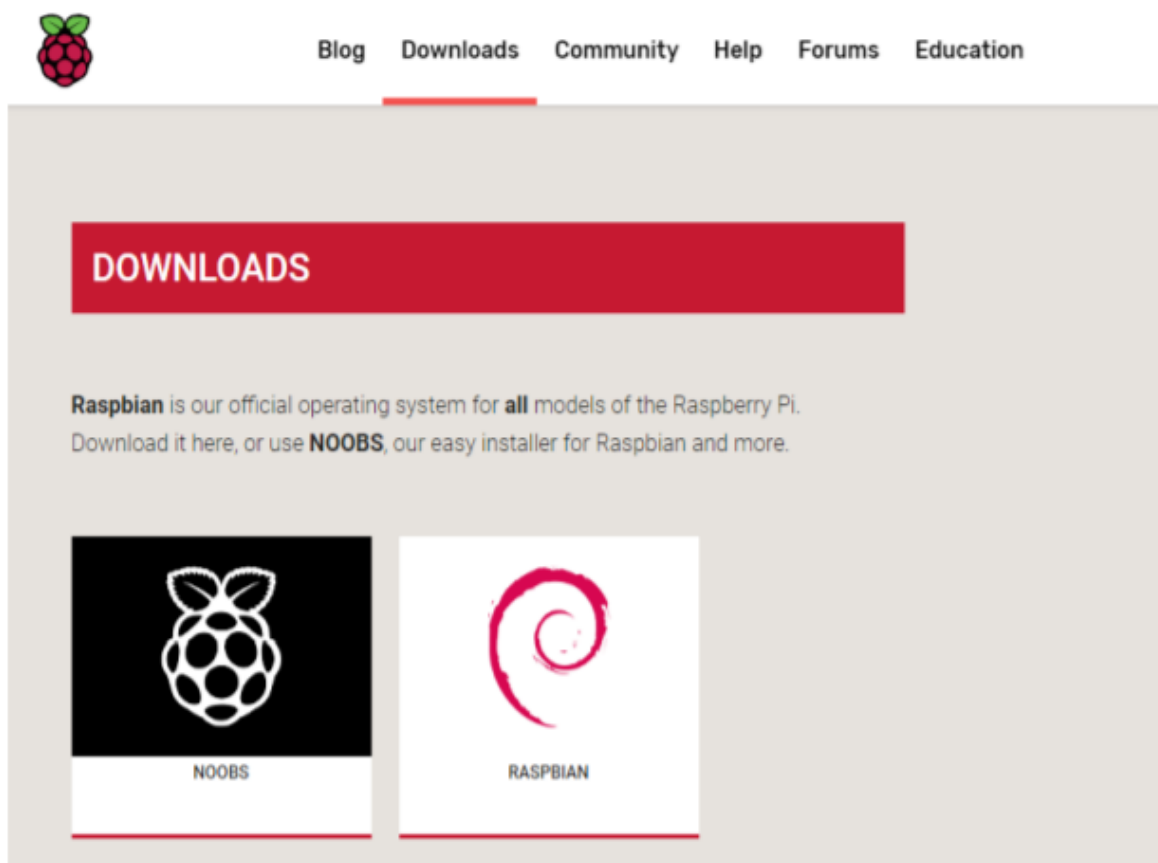


Figura 88: Pantalla principal de RaspBian

En nuestro caso, utilizamos la imagen ISO y gracias a la aplicación balenaEtcher podemos introducir el SO en una tarjeta SD, que funciona como disco duro en la RaspBerry. Una vez encendamos la RaspBerry con la tarjeta SD introducida, empezará a instalar el SO.

Cuando nos encontremos en el escritorio principal, se selecciona la opción de configuración (Figura 89)

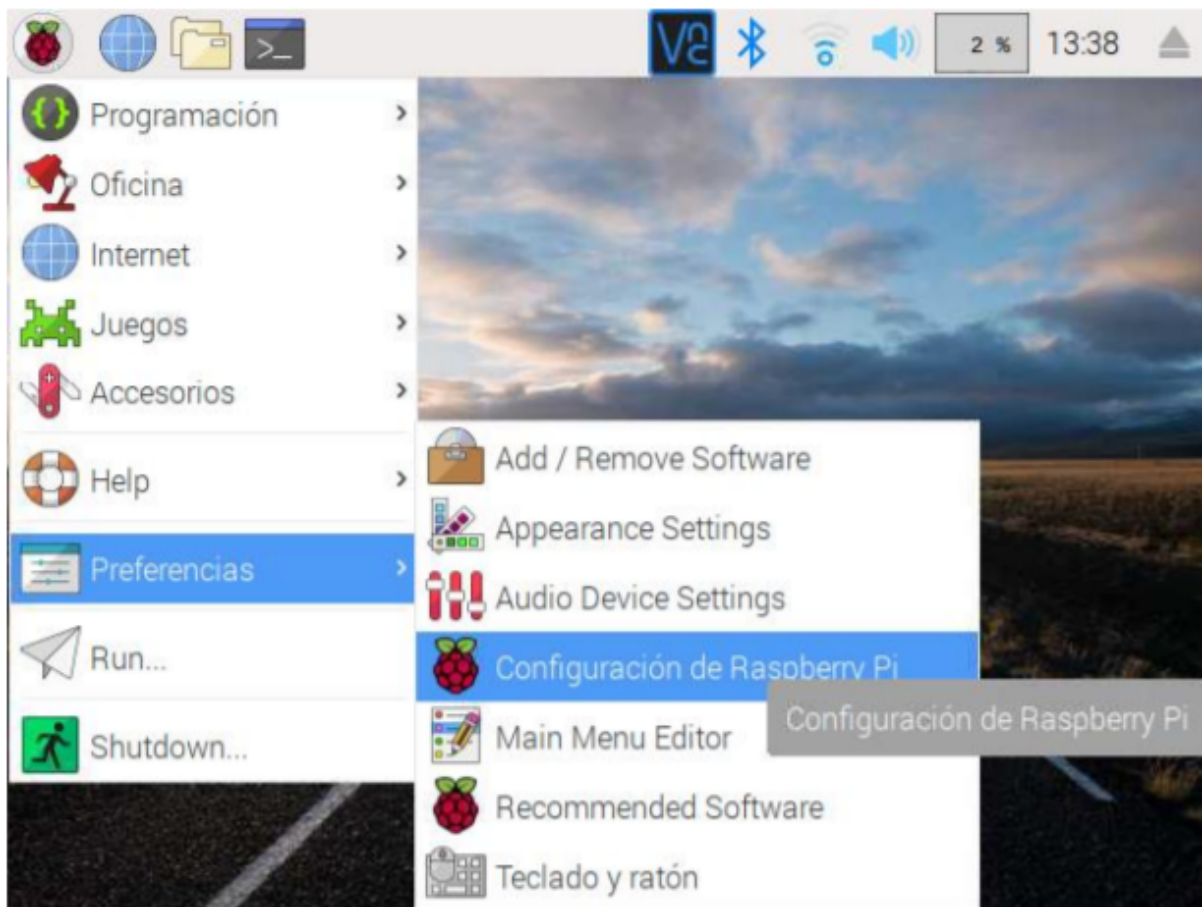


Figura 89: Lugar donde se encuentra el botón de configuración

A continuación, en la cual hay que desplazarse hasta el apartado interfaces. En dicho apartado, debe tener las opciones marcadas en la Figura 90 ya que serán de utilidad por distintos motivos como se explican a continuación:

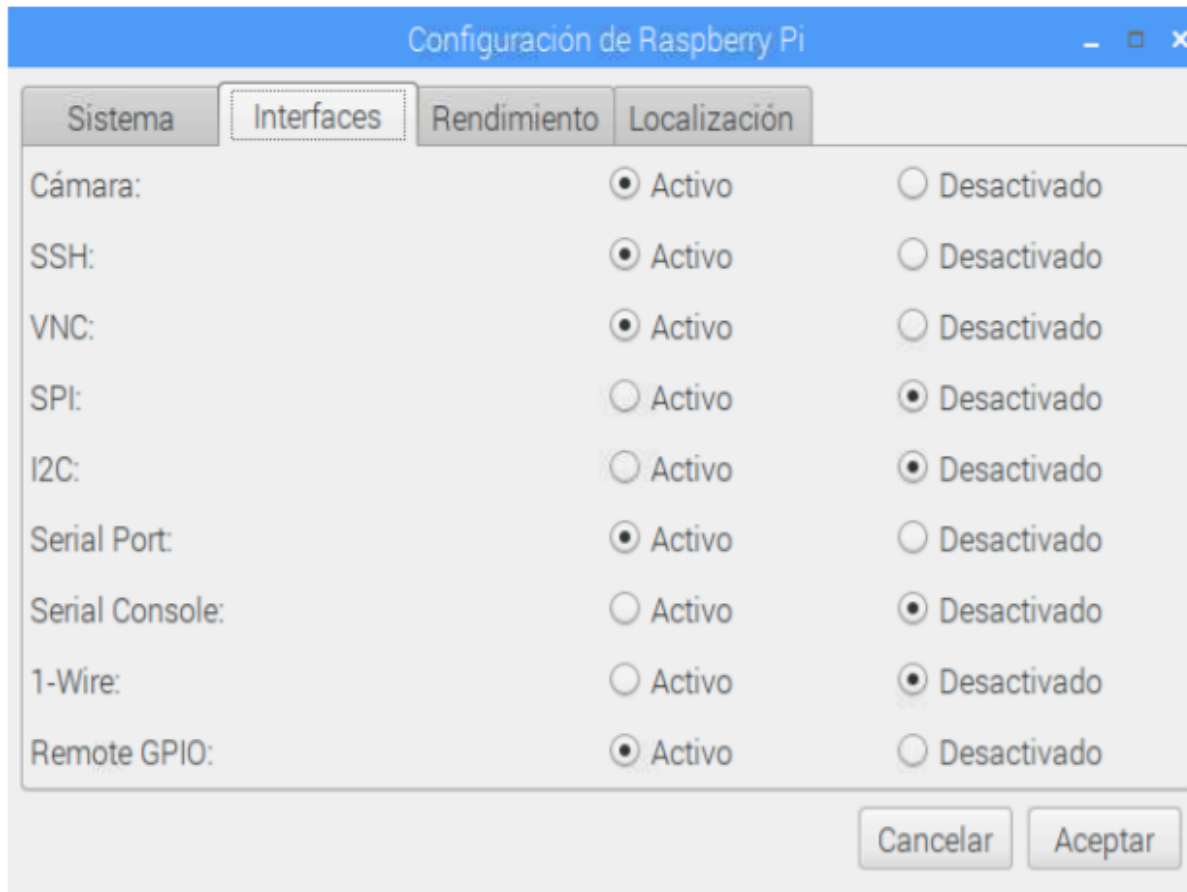


Figura 90: Opciones que hay que seleccionar para el funcionamiento del sistema

- Cámara: Esta opción es muy necesaria para el desarrollo de este TFG, ya que activa la interfaz SCI, que es donde se conecta la PiCam a la Raspberry Pi.
- SSH: La activamos para poder acceder a la Raspberry de forma remota desde un cliente SSH.
- Puerto Serie: Esta opción es habilitada, ya que a través del puerto serie se va a establecer la comunicación con el robot hexápodo, más concretamente con el módulo de Arduino que lo maneja.
- Remote GPIO: La marcamos ya que se va a usar pines de la placa para alimentar a la RaspBerry.

Los otros parámetros vienen activados por defecto.

1.2 Instalación de la cámara

En esta parte se va a explicar la instalación de la PiCam. Para ello la conexión se hace mediante un bus de cinta que irá conectado al conector SCI (Figura 91) como se explica a continuación. Se deben levantar la pestaña de color negro para que de esta forma se pueda introducir el cable. Este debe tener los cables metálicos mirando hacia afuera del puerto Ethernet.

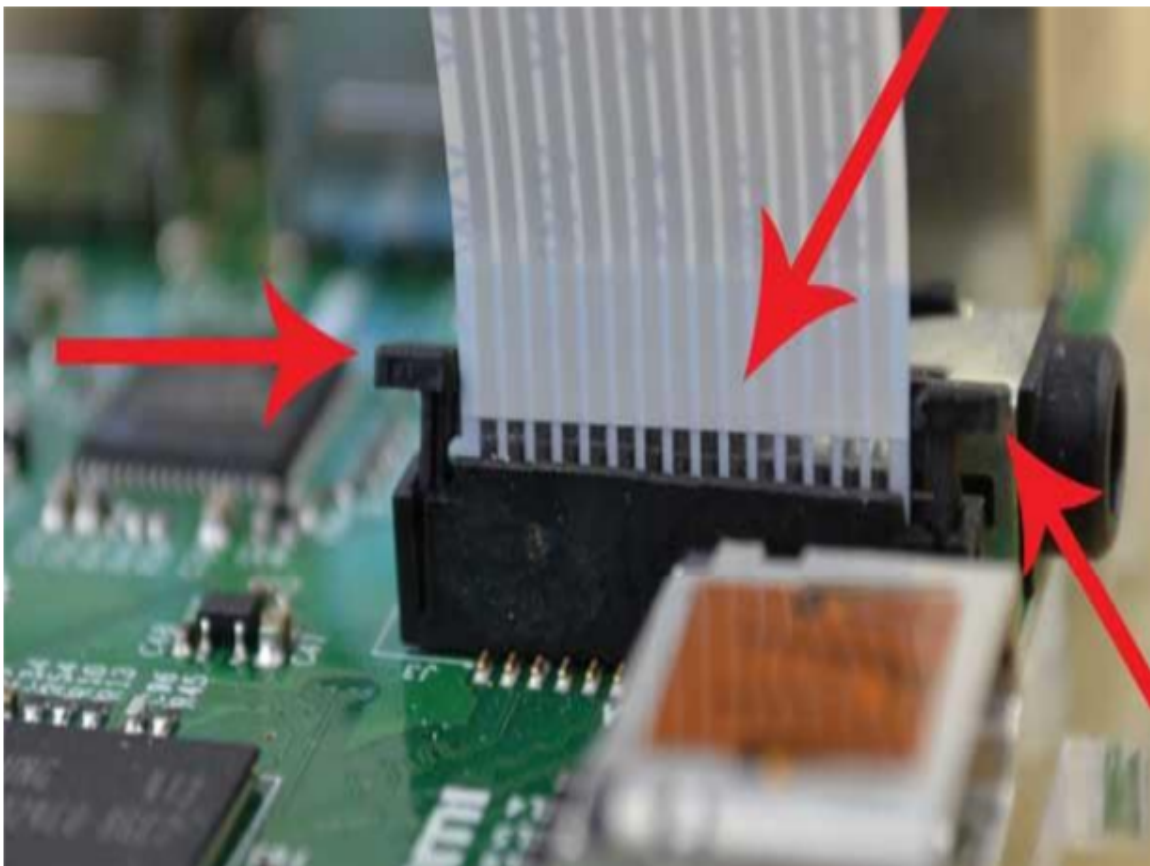
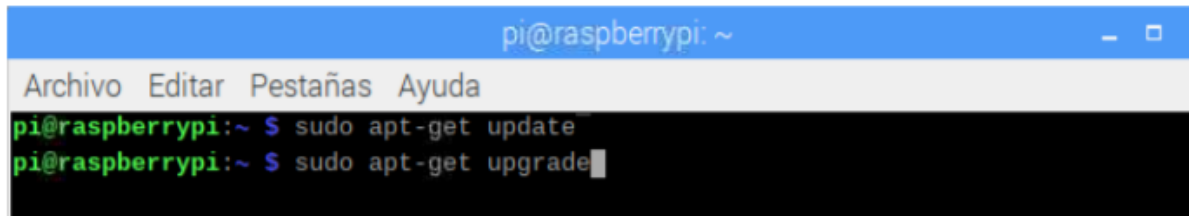


Figura 91: Conexión SCI de la RaspBerry

Cuando el conector se encuentre como en la imagen anterior, se bajarán la pestaña a su posición inicial, de manera que el cable quede bloqueado físicamente en su lugar correspondiente. Después de efectuar lo anterior, pasamos con el software.

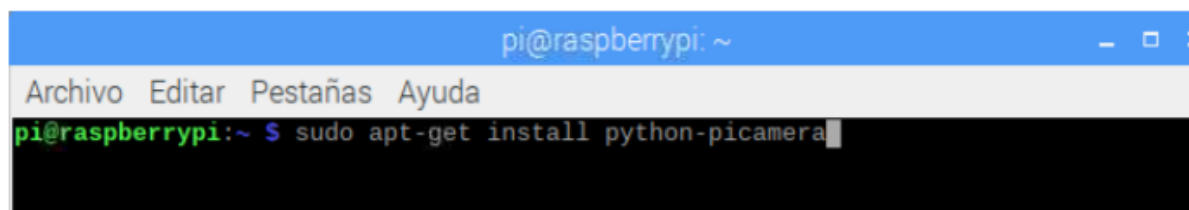
Una vez esté encendida la Raspberry, se ejecuta la consola de Raspbian y se debe introducir los siguientes comandos (Figura 92), los cuales son obligatorios introducir de manera que les dé tiempo a ejecutarse de manera completa cada uno por separado.



```
pi@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
pi@raspberrypi:~ $ sudo apt-get update  
pi@raspberrypi:~ $ sudo apt-get upgrade
```

Figura 92: Comandos para actualización de RaspBian

Los comandos anteriores se introducen para actualizar el software, en primer lugar, antes de instalar cualquier otro paquete en el sistema. A continuación, se instala la librería escribiendo el siguiente comando (Figura 93):



```
pi@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
pi@raspberrypi:~ $ sudo apt-get install python-picamera
```

Figura 93: Comando para instalar la librería picamera

1.3 Instalación de OpenCV

Para su instalación, se han de aplicar una serie de comandos que se deben seguir de manera ordenada. En primer lugar, se debe de hacer una actualización del software, tras reiniciarse la RaspBerry, tenemos que meter los siguientes comandos:

Instalar los paquetes de desarrollador CMAKE.

```
sudo apt-get install build-essential cmake pkg-config-y
```

Instalar los paquetes de I/O de imagen.

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev -y
```

Instalar los paquetes de I/O de video.

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev -y  
sudo apt-get install libxvidcore-dev libx264-dev -y
```

Instalar la biblioteca de desarrollo GTK para GUI básicas

```
sudo apt-get install libgtk2.0-dev libgtk-3-dev -y
```

Instalar paquetes de optimización.

```
sudo apt-get install libatlas-base-dev gfortran -y
```

A continuación, se instala Python y la librería numpy con los siguientes comandos:

```
sudo apt-get install python3 python3-setuptools python3-dev -y  
  
wget https://bootstrap.pypa.io/get-pip.py  
  
sudo python3 get-pip.py  
  
sudo pip3 install numpy
```

Descargamos la librería OpenCV junto con sus módulos extras:

```
cd ~  
  
wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.4.0.zip  
  
unzip opencv.zip  
  
wget -O opencv_contrib.zip  
  
https://github.com/Itseez/opencv_contrib/archive/3.4.0.zip  
  
unzip opencv_contrib.zip
```

Y, una vez descargada la librería junto con sus paquetes, se instala y se compila en Python con las siguientes opciones:

```
cd opencv-3.4.0  
  
mkdir build  
  
cd build  
  
cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D BUILD_opencv_java=OFF \  
-D BUILD_opencv_python2=OFF \  
-D BUILD_opencv_python3=ON \  
-D PYTHON_DEFAULT_EXECUTABLE=$(which python3) \  
-D INSTALL_C_EXAMPLES=OFF \  
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D BUILD_EXAMPLES=ON \  
-D OPENCV_EXTRA_MODULES_PATH=~/.opencv_contrib-3.4.0/modules \  
-D WITH_CUDA=OFF \  
-D BUILD_TESTS=OFF \  
-D BUILD_PERF_TESTS=OFF ..
```

Ahora se debe cambiar el tamaño del espacio para que, de esta manera, la Raspberry pueda compilar con los cuatro núcleos sin tener problemas de memoria.

Se abre / etc / dphys-swapfile y luego se edita la variable CONF_SWAPSIZE. Para ello en el terminal se escribe:

```
sudo nano /etc/dphys-swapfile
```

Y en la variable se coloca:

```
# CONF_SWAPSIZE=100  
CONF_SWAPSIZE=1024
```

Para guardar los cambios se debe de presionar las teclas Ctrl + O. Y para salir se presiona Ctrl + X. Escribimos las siguientes líneas para llevar a cabo esos cambios:

```
sudo /etc/init.d/dphys-swapfile stop  
sudo /etc/init.d/dphys-swapfile start
```

Finalmente, se compila todo lo anterior usando los cuatro núcleos de la placa. (Indicar que este proceso puede durar del orden de hora y media a dos horas).

```
make -j4
```

Y, para finalizar todo este proceso de instalación de la librería OpenCV y asegurar su correcto funcionamiento: (Este proceso tiene una duración similar al anterior)

```
sudo make install  
sudo ldconfig
```

Una vez se complete lo anterior, es recomendable hacer un reinicio del dispositivo antes de empezar a trabajar con esta librería [13].

1.4 Instalación de librerías extras

Para este proyecto hemos necesitado además las librerías:

Mosquitto

Flask

paho-mqtt

Las cuales se instalan con la herramienta pip de python.

Anexo 2: Manual de usuario

Este anexo explica como poner en marcha el sistema, el cual se puede ejecutar por partes o todo a la vez, para ellos vamos a distinguir:

- Control manual del robot a partir de peticiones GET:
 - Encendemos el robot
 - Se ejecuta el script servicioREST.py
 - Se controla mediante peticiones GET del formato:
 - <http://ip-raspberry:5000/robot/movimiento>
 - Movimiento sigue el mapping WASD
- Obtener imágenes obtenidas por la cámara y procesadas por el algoritmo:
 - Encendemos el robot
 - Se ejecuta el script marker_cam2.py
 - A través de <http://ip-raspberry:5000/video> podemos ver las imágenes
- Navegación autónoma sin simulación
 - Encendemos el robot
 - Ejecutamos el script servicioREST.py
 - Iniciamos mosquito
 - Ejecutamos el script PlayerRazoner.py con parámetro 1
 - Ejecutamos el script PlayerExecuter.py con parámetro 1
 - Ejecutamos el script map.py
- Control manual desde la simulación
 - Encendemos el robot
 - Ejecutamos el script servicioREST.py
 - Iniciamos mosquito
 - Iniciamos la simulación
 - Movimiento sigue el mapping WASD
- Sistema completo
 - Encendemos el robot
 - Ejecutamos el script servicioREST.py

- Iniciamos mosquito
- Iniciamos simulación
- Ejecutamos el script PlayerRazoner.py con parámetro 1
- Ejecutamos el script PlayerExecuter.py con parámetro 1
- Ejecutamos el script map.py

Todos los scripts están realizados para Python 3, por lo que tendremos que iniciarlos con python3. Todos los scripts se ejecutan en la Raspberry, salvo la simulación que se tiene que ejecutar en un ordenador a parte.

Anexo 3: ¿Por qué se calibra la cámara?

La calibración de una cámara es el proceso mediante el cual se obtienen los parámetros fundamentales de una cámara. Estos parámetros permiten determinar donde se proyecta un punto 3D del espacio en el sensor de la cámara. Los parámetros de esta se clasifican en intrínsecos y extrínsecos. Los primeros, son aquellos que describen el funcionamiento de una cámara, mientras que los segundos, definen la posición y orientación del cuadro de referencia de la cámara respecto al mundo real, es decir, dan la orientación externa de la cámara. Los parámetros intrínsecos son:

- f_x y f_y : Distancia focal de la lente de la cámara en ambos ejes. Normalmente expresados en píxeles.
- C_x y C_y : Centro óptico del sensor. Expresado en píxeles.
- k_1, k_2, p_1, p_2, k_3 : Coeficientes de distorsión.

En una cámara ideal, un punto (X, Y, Z) en el espacio se proyectaría en el píxel determinado por las siguientes ecuaciones:

$$x = -X \cdot f / Z + C_x \quad (6)$$

$$y = -Y \cdot f / Z + C_y \quad (7)$$

Sin embargo, las lentes de las cámaras distorsionan la escena, haciendo que la distancia de los diferentes puntos respecto al centro de la imagen varíe. Por lo tanto, si se desea conocer con exactitud la proyección de los diferentes puntos en la imagen se debe de considerar estos coeficientes de distorsión. La distorsión se puede descomponer en dos componentes, una componente radial y otra tangencial, ambas dependientes del ángulo y de la distancia respecto al centro de la imagen. Todo esto se contempla en los coeficientes de distorsión calculados en la calibración (k_1, k_2, p_1, p_2, k_3).

Si se desea conocer la proyección de un punto referido a un sistema de referencia arbitrario es necesario hacer uso de los parámetros extrínsecos. Los parámetros extrínsecos son básicamente las rotaciones 3D ($R_{vec} = \{R_x, R_y, R_z\}$) y las translaciones ($T_{vec} = \{T_x, T_y, T_z\}$) necesarias para traducir el sistema de referencia de la cámara al arbitrario. También se pueden tener en cuenta cambios de escala y cambios de perspectiva. Una forma de llevar a cabo este cambio de sistema de referencia es haciendo uso de las coordenadas homogéneas.

Bibliografía

- [1] Hussain, T., Muhammad, K., Khan, S., Ullah, A., Lee, M. Y., & Baik, S. W. (2019). Intelligent baby behavior monitoring using embedded vision in IoT for smart healthcare centers. *Journal of Artificial Intelligence and Systems*, 1(1), 110-124.
- [2] Othman, N. A., & Aydin, I. (2017, September). A new IoT combined body detection of people by using computer vision for security applications. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)* (pp. 108-112). IEEE.
- [3] Alonso, A. D. C., & Jara, E. A. M. (2016). Visión por computadora: identificación, clasificación y seguimiento de objetos. *FPUNE Scientific*, (10).
- [4] Neumann, A., Laranjeiro, N., & Bernardino, J. (2018). An analysis of public REST web service APIs. *IEEE Transactions on Services Computing*.
- [5] Python, I. (2019). Python.
- [6] Bouman, A., Ginting, M. F., Alatur, N., Palieri, M., Fan, D. D., Touma, T., ... & Agha-Mohammadi, A. A. (2020, October). Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2518-2525). IEEE.
- [7] Huang, H., Chai, P., Ehmke, C., Merewether, G., Dadabhoy, F., Feng, A., ... & Traverso, C. (2020). Agile mobile robotic platform for contactless vital signs monitoring. *TechRxiv, to be published, doi, 10*.
- [8] Bualat, M. G., Smith, T., Smith, E. E., Fong, T., & Wheeler, D. W. (2018). Astrobee: A new tool for ISS operations. In *2018 SpaceOps Conference* (p. 2517).

- [9] Godoy, J., Pérez, J., Onieva, E., Villagrà, J., Milanés, V., Haber, R. (2015). A driverless vehicle demonstration on motorways and in urban environments. *Transport*, 30(3), 253-263.
- [10] Ingle, S., & Phute, M. (2016). Tesla autopilot: semi autonomous driving, an uptick for future autonomy. *International Research Journal of Engineering and Technology*, 3(9), 369-372.
- [11] Maki, J. N., Gruel, D., McKinney, C., Ravine, M. A., Morales, M., Lee, D., ... & Algermissen, S. (2020). The Mars 2020 Engineering Cameras and microphone on the perseverance rover: A next-generation imaging system for Mars exploration. *Space Science Reviews*, 216(8), 1-48.
- [12] Gómez Díaz, M. (2019a, abril). *CONTROL DE UN ROBOT HEXÁPODO*. Jaén: Universidad de Jaén.
- [13] Guirado Cárdenas, J. A. (2018, octubre). *Navegación Autónoma de Robot Hexápodo*. Jaén: Universidad de Jaén.
- [14] Arsaute, A., Zorzán, F. A., Daniele, M., González, A., & Frutos, M. (2018). Generación automática de API REST a partir de API Java, basada en transformación de Modelos (MDD). In *XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste)*.
- [15] Aslam, F. A., Mohammed, H. N., Mohd, J. M., Gulamgaus, M. A., & Lok, P. (2015). Efficient way of web development using python and flask. *International Journal of Advanced Research in Computer Science*, 6(2), 54-57.
- [16] Liechti, C. (2016). PySerial documentation. *Versión: 2.6, Diciembre 2011*.
- [17] Mayer, I. R., Távira, J. V., Sánchez, R. F. M. Sistemas Embebidos para Robótica Móvil: Estudio Comparativo.

- [18] Avola, D., Cinque, L., Foresti, G. L., Mercuri, C., & Pannone, D. (2016, February). A practical framework for the development of augmented reality applications by using ArUco markers. In *International Conference on Pattern Recognition Applications and Methods* (Vol. 2, pp. 645-654). SciTePress.
- [19] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6), 2280-2292.
- [20] Poroykov, A., Kalugin, P., Shitov, S., & Lapitskaya, I. (2020). Modeling ArUco markers images for accuracy analysis of their 3D pose estimation. In *30th International Conference on Computer Graphics and Machine Vision (GraphiCon 2020). Part* (Vol. 2).
- [21] Nosrati, M., Karimi, R., & Hasanvand, H. A. (2012). Investigation of the *(star) search algorithms: Characteristics, methods and approaches. *World Applied Programming*, 2(4), 251-256.
- [22] Zikky, M. (2016). Review of A*(A star) navigation mesh pathfinding as the alternative of artificial intelligence for ghosts agents in the Pacman game. *EMITTER International journal of engineering technology*, 4(1), 141-149.
- [23] Stout, B. (1996). Smart moves: Intelligent pathfinding. *Game developer magazine*, 10, 28-35.
- [24] Mall, R. (2018). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd.
- [25] Greenspan, S. J., Mylopoulos, J., & Borgida, A. (1982, September). Capturing more world knowledge in the requirements specification. In *Proceedings of the 6th international conference on Software engineering* (pp. 225-234).

- [26] Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Comparison of JSON and XML data interchange formats: a case study. *Caine*, 9, 157-162.
- [27] Antal, T. A. (2018). Considerations on the serial pc-arduino uno r3 interaction, in java, using jdeveloper, for a 3r serial robot, based on the arduink library. *ACTA TECHNICA NAPOCENSIS-Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING*, 61(1).
- [28] Soni, D., & Makwana, A. (2017, April). A survey on mqtt: a protocol of internet of things (iot). In *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)* (Vol. 20).
- [29] Fezari, M., & Al Dahoud, A. (2018). Integrated Development Environment “IDE” For Arduino. *WSN applications*, 1-12.
- [30] Gkikas, K., Nathanael, D., & Marmaras, N. (2007, May). The evolution of FPS games controllers: how use progressively shaped their present design. In *Panhellenic Conference on Informatics (PCI)*.
- [31] Turk, M. (2014). Multimodal interaction: A review. *Pattern recognition letters*, 36, 189-195.
- [32] Ponce González, M. M. (2012). *Visión por computador para UAS*. Madrid: Universidad Politécnica de Madrid.
- [33] García Gómez, G. J. (2007, julio). SISTEMA DE CONTROL MULTISENSORIAL APLICADO A TAREAS DE MANIPULACIÓN AUTOMÁTICA. Alicante: Universidad de Alicante.