UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – SISTEMAS ELECTRÓNICOS

“SISTEMA DE CAPTURA DE POSTURAS DEL CUERPO HUMANO POR MEDIO DE VISIÓN DE MÁQUINA PARA ROBOT HUMANOIDE UTILIZANDO UN SISTEMA EMBEBIDO”

T E S I S

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN INGENIERÍA

PRESENTA:

ING. JUAN JOSÉ MOSCO LUCIANO

TUTOR:
Dr. JUAN MARIO PEÑA CABRERA, IIMAS-UNAM

CIUDAD UNIVERSITARIA, Cd. Mx., ENERO 2021

JURADO ASIGNADO

Presidente: Dr. Pérez Alcázar Pablo Roberto

Secretario: Dr. Rascón Estebané Caleb Antonio

1^{er}. Vocal: Dr. Peña Cabrera Juan Mario

2^{do}. Vocal: Dr. Lomas Barrie Víctor Manuel

3^{er}. Vocal: Dr. De La Rosa Nieves Saúl

Lugar o lugares donde se realizó la tesis:

Universidad Nacional Autónoma de México,

Ciudad Universitaria, Ciudad de México, México.

Instituto de Investigaciones en MatemáticasAplicadas y Sistemas(IIMAS).

Facultad de Ingeniería, UNAM

TUTOR DE TESIS:

Dr. Juan Mario Peña Cabrera

FIRMA

Quiero agradecer a la Universidad Nacional Autónoma de México y al posgrado de Ingeniería por brindarme un lugar en la institución.

Al CONACYT por ofrecerme apoyo durante mis estudios.

A mi tutor el Dr. Juan Mario Peña Cabrera por todo su apoyo y por darme la oportunidad de trabajar con el.

A mis profesores y sinodales que ayudaron a mi formación.

A mi Familia:

A mi madre, una mujer admirable e incansable que siempre me ha brindado su apoyo, cariño y sabiduría de forma incondicional, nunca terminare de agradecerte.

A mi hermano, por los momentos compartidos, por los consejos brindados y siempre haber sido una guía en mi formación profesional.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Resumen

En el presente trabajo se plantea la metodología para el desarrollo e implementación de un sistema que capture los movimientos de una persona mediante técnicas de visión artificial en un sistema embebido, con el fin de realizar el movimiento imitativo en un robot humanoide.

El cuerpo humano posee una gran cantidad de grados de libertad, en el proyecto, se simplifican en una interfaz de control implementada en un sistema embebido. Esta interfaz, permitirá que ciertos movimientos que realice la persona se vean reflejados en el humanoide.

El proyecto utiliza un robot humanoide con 20 grados de libertad totales. Para validar la metodología utilizada, se utilizan 4 de ellos, que abarcan los brazos y antebrazos en un plano bidimensional. Mediante un sistema de captura de movimientos basado en una cámara, se obtiene información de las posiciones corporales de la persona, las que determinaran los movimientos a imitar en el humanoide.

1.2. Objetivo

1.2.1. Objetivo general

Este trabajo tiene por objetivo diseñar e implementar un sistema escalable para capturar posturas del cuerpo humano mediante visión de máquina en un sistema embebido para el control imitativo en un robot humanoide.

1.2.2. Objetivos específicos

- Capturar las posturas de los movimientos con una cámara y obtener el gráfico del esqueleto de las mismas.
- Desarrollar los algoritmos de procesamiento que permitan obtener los ángulos de movimiento para el control del robot.
- Implementar los algoritmos de procesamiento y control en un sistema embebido.
- Teleoperación del robot humanoide.

1.3. Planteamiento del problema

A partir del surgimiento de las máquinas para el apoyo en la realización de las tareas humanas e industriales, y con el uso de robots antropomórficos y humanoides, ha surgido el interés de investigar la interacción de los humanos y robots para diversos fines. Actualmente existen varias aplicaciones de robots imitando movimientos humanos y realizando tareas humanas como el robot de Toyota T-HR3, que se controla desde un sistema maestro de maniobras que permite operar instintivamente todo el cuerpo del robot con controles vestibles que mapean los movimientos de manos, brazos y pies del robot [4]. Otro ejemplo de robot imitando los movimientos es el robot TEO, que mediante un sistema de visión permite teleoperarlo para aplicaciones de servicio [5]. De igual forma, en la industria aeroespacial y en la industria 4.0 se encuentran varias aplicaciones imitativas de los robots como el Skybot F-850 un robot ruso cosmonauta, o en Kawada industries donde se utilizan robots humanoides para tareas de manufactura [6].

Como se puede ver, la imitación en agentes roboticos tienen potenciales aplicaciones, donde algunas de las principales son aprendizaje por imitación, en el cual un agente trata de aprender a realizar una determinada tarea utilizando la información generada por un humano u otro agente, a menudo más experto, que realiza esa misma tarea [7], en la asistencia humana en tareas en una variedad de entornos, como el hogar, las instalaciones médicas, las obras de construcción y entornos hostiles para el ser humano como las zonas afectadas por desastres e incluso el espacio exterior.

Muchas de las investigaciones se han centrado en el uso de complejos sistemas inerciales para lograr diversas tareas como imitación, teleoperación, manufactura, etc. tanto en robots antropomórficos, como en robots humanoides. La desventaja de estos es que requieren de un *hardware* dedicado, en ocasiones costoso e invasivo.

La interacción con robots, ofrece diferentes alternativas tecnológicas para lograrlo, una de ellas es la visión artificial, al utilizarla, se puede hablar de un "modo sensorial de visión"que ofrece realimentación visual del entorno de una máquina robótica obteniendo información útil para llevar a cabo las acciones de tareas encomendadas en una aplicación, emulando así, la forma en que los humanos utilizamos este modo sensorial.

En el Laboratorio de Electrónica y Automatización para la Industria 4.0 del *IIMAS* se han realizado proyectos e investigaciones en celdas de manufactura flexibles, robots antropomórficos y dotar del modo sensorial de la visión a los mismos. El proyecto tiene la finalidad de sentar las bases de un método para conseguir los movimientos imitativos de un robot humanoide compatible con el marco de trabajo del laboratorio y teleoperar este robot mediante protocolos de internet de las cosas. Este trabajo de investigación permitirá posteriormente enseñar al robot movimientos, reconocer posturas para disparar rutinas, cooperación entre robots, etc.

1.4. Estado del arte

En esta sección se muestra una síntesis del trabajo previo en el tema de la tesis, obtenido de consultar artículos del estado del arte en los temas de sistemas de seguimiento visual, interacción con robots humanoides, visión de máquina en sistemas embebidos y teleoperación en robots humanoides.

En [8] J.M. GARCIA *etal*. capturan los movimientos de los miembros superiores de una persona en 3D mediante un sensor kinect, y mediante un algoritmo extraen los ángulos para teleoperación de un robot humanoide.

En [9] Emrehan Yavşan *etal*. realizan un *Framework* que captura los movimientos en 3D de una persona en tiempo real mediante el sensor Kinect, vectorizan los movimientos y por medio de métodos geométricos obtienen los ángulos, posteriormente con los datos angulares obtenidos entrenan una red neuronal para reconocer e imitar gestos y movimientos en un robot NAO.

en [10] Robinson Jiménez Moreno *etal*. por medio de una interfaz computacional y un sensor Kinect capturan los movimientos de una persona, realizan métodos de regresión lineal para una mayor estabilidad en los datos y con ellos controlan los movimientos imitativos en un agente bioloid.

En [11] Jonas Koenemann *etal*. realizan un sistema que permite a los robots humanoides imitar los complejos movimientos de todo el cuerpo de los humanos en tiempo

real. Utilizan datos humanos capturados con un sistema de captura de movimiento Xsens MVN para que un humanoide NAO imite los movimientos realizados.

En [12] Jeremy SEYSSAUD *etal*. controlan remotamente un robot DARwIN-OP para trabajar en ambientes nucleares no aptos o dañinos para un humano. Se instrumenta al robot con medidores radiológicos y un sistema embebido como computadora a bordo Ordroid XU3. La caminata del robot es controlada remotamente por un joystick, los movimientos de los brazos a través de una cámara ASUS y el gripper es controlado a través de un guante instrumentado.

En [13] Seung-Joon Yi *etal*. realizan un controlador para el robot humanoide DARwIN-OP basándose en el modelo cinemático del robot y de los movimientos humanos. Se obtiene información del operador mediante el sensor Kinect.

En [14] Alfabi Muhlisin Sakti *etal*. complementan el sistema de control del robot DARwIN-OP con un controlador PID para mejorar su marcha.

En [15] Hemant K.Baxi desarrollo una driver Bluetooth para el robot humanoide DARwIN-OP y una app para Android para controlarlo remotamente. Realizo un análisis cinemático de los movimientos del robot y programo un controlador para mejorar la marcha del robot.

En [16] Van Vuong Nguyen *etal*. realizan un sistema de captura de movimientos donde, los movimientos son capturados a través de un sensor Kinect, posteriormente se realiza un análisis cinemático. Este sistema es probado en el simulador Webots y posteriormente transferido desde la simulación al robot DARwIN-OP.

En [17] Alex W. Grammar M.S. *etal*. desarrollan una interfaz de control basada en señales electromiografícas que controlan 1 codo del robot DARwIN-OP. Implementan el control en un microcontrolador Arduino, el cual realiza las lecturas de la señal electromiografica y por medio de una interfaz USB le envia los datos al robot.

En [18] Ismail Almetwally *etal*. utilizan el sensor Kinect para capturar los movimientos de una persona, se guarda la posición inicial donde comenzaron los movimientos de la persona, realizan un análisis cinemático y con ello pueden teleoperar un robot NAO, mover sus brazos rotar, caminar, etc.

En [19] Petar Kormushev *etal*. implementan controladores en un robot humanoide Fujitsu HOAP-2 humanoide para adquirir nuevas habilidades motoras por la enseñanza de la cinética.

En [20] Christopher Stanton *etal*. teleoperan un robot humanoide NAO a través de un traje de captura de movimiento de cuerpo entero. El operador y el robot realizan una serie de movimientos sincronizados emparejados que capturar tanto la captura de movimiento del operador y los datos del actuador del robot y con ello entrenan una red neuronal para el aprendizaje de los movimientos.

En [21] Igor Rodriguez *etal*. realizan dos paqueterías para ROS para con ellas poder teleoperar al robot NAO. Estas paqueterías se pueden utilizar para teleoperación basada en habla a través de un servicio de Google y en teleoperación por movimientos capturados a través del sensor Kineckt.

En [22] Aditya Sripada *etal*. capturan los movimientos humanos por medio de un sensor Kinect y mediante ROS se realiza la comunicación entre las diversas partes de su sistema.

En [23] Emily-Jane Rolley-Parnell *etal*. desarrollaron un sistema llamado TelePose que es un método de teleoperación bimanual en tiempo real para controlar robots con extremidades, a través de sus efectores finales, en el espacio cartesiano. Mediante el sistema de detección de movimientos multi persona en tiempo real OpenPose, extraen el esqueleto del operador y con el sensor ASUS Xtion Pro extraen la profundidad para controlar y realizara los movimientos imitativos en un robot con extremidades.

En varios de estos trabajos se utilizan computadoras de propósito general, al igual que sensores y hardware especializado. En este trabajo de investigación se propone un método que simplifique el problema de visión desde la instrumentación en los marcadores hasta la implementación de los algoritmos desarrollados en una plataforma de computo dedicado, aprovechando algunas de sus características importantes como la concurrencia.

Capítulo 2

Marco teórico

2.1. Teleoperación

Se entiende por teleoperación a la acción que realiza un ser humano de controlar, manipular, gobernar un dispositivo o sistema a distancia. Existen una serie de definiciones relacionadas con este término y se muestran a continuación [24][25][26][27]

Operador: Un operador humano es la persona monitorea o acciona una máquina a operar.

Dispositivo teleoperado: puede ser un manipulador, un robot, un vehículo o dispositivo similar. Es la máquina que trabaja en la zona remota y que está siendo controlada por el operador.

Interfaz: conjunto de dispositivos que permiten la interacción del operador con el sistema de teleoperación. Se considera al manipulador maestro como parte de la interfaz, así como a los monitores de vídeo, o cualquier otro dispositivo que permita al operador mandar información al sistema y recibir información del mismo.

Control y canales de comunicación: conjunto de dispositivos que modulan, transmiten y adaptan el conjunto de señales que se transmiten entre la zona remota y la local. Generalmente se cuenta con una o varias unidades de procesamiento.

Sensores: conjunto de dispositivos que recogen la información, tanto de la zona local como de la zona remota, para ser utilizada por la interfaz y el control.

Un sistema de teleoperación se puede representar en 5 bloques de subsistemas: el operador, la interfaz, el medio de comunicación, el esclavo (sensores, actuadores,

etc.) y el ambiente y/o tareas. La figura 2.1. muestra los bloques de un sistema de teleoperación.

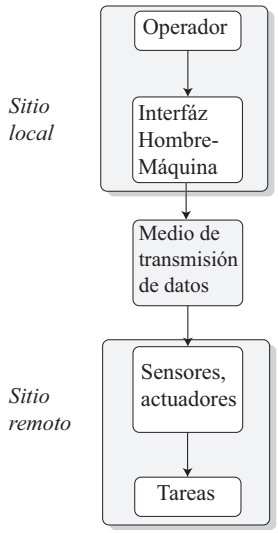


Figura 2.1: Sistema de teleoperación general.

2.2. Visión artificial

De los diferentes sentidos que posee el ser humano, la visión es utilizada en un 60 % en las actividades que realiza, obteniendo así por este medio la mayor información de su entorno. Esta condición, ha despertado un gran interés en las ciencias computacionales por la investigación y desarrollo de sistemas de visión artificial.

La visión artificial, es una disciplina que se refiere a la teoría y el desarrollo de sistemas tecnológicos que realizan funciones de adquisición y formación de imágenes, para extraer información de ellas que permitan la descripción de la estructura y propiedades de un mundo real tridimensional estático o dinámico, generalmente se hace a través de la adquisición de multiples imágenes bidimensionales [28].

Un sistema de visión está compuesto por diferentes bloques donde están implicados *hardware*, *software* y herramientas matemáticas. En la figura 2.2 se muestran los bloques clásicos de un sistema de visión.

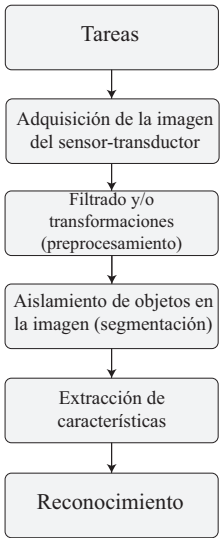


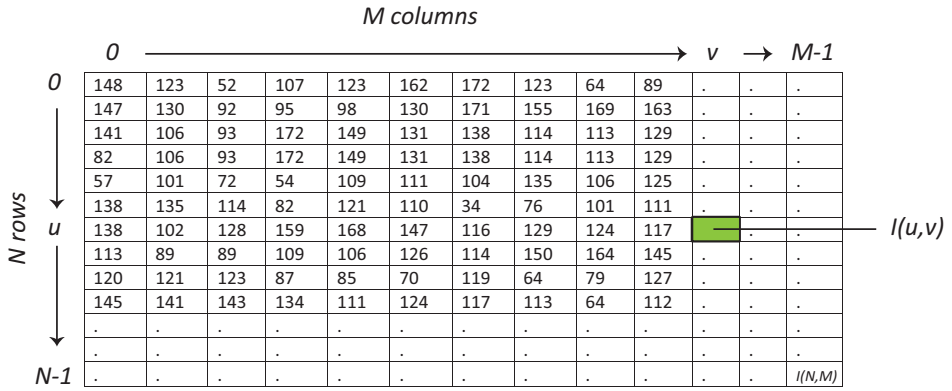
Figura 2.2: Sistema general de visión.

2.2.1. Adquisición de la imagen

Una imagen es la representación visual de un entorno u objeto donde convergen rayos luminosos. Para la formación de una imagen se tienen los siguientes elementos: el objeto o entorno, la fuente luminosa y el sistema de captura de la imagen que en la mayoría de los casos es una cámara, la cual consiste en un sistema óptico, sensores-transductores que reaccionen a los cambios de luz/color y el digitalizador para cuantificar las señales eléctricas del sensor-transductor. El resultado de estas tres etapas es una descripción de la imagen en la forma de una matriz $2D$ ordenada de números enteros [29]. En términos más formales de funciones discretas una imagen digital "I" es una función $2D$ que mapea desde el dominio de coordenadas enteras $N * M$ a un rango de posibles valores de píxeles "P" tal que:

$$I(u,v) \in P \quad \text{y} \quad u,v \in N,M \tag{2.1}$$

La figura 2.3 muestra la matríz de imagen.



148	123	52	107	123	162	172	123	64	89	.	.	.
147	130	92	95	98	130	171	155	169	163	.	.	.
141	106	93	172	149	131	138	114	113	129	.	.	.
82	106	93	172	149	131	138	114	113	129	.	.	.
57	101	72	54	109	111	104	135	106	125	.	.	.
138	135	114	82	121	110	34	76	101	111	.	.	.
138	102	128	159	168	147	116	129	124	117	.	.	.
113	89	89	109	106	126	114	150	164	145	.	.	.
120	121	123	87	85	70	119	64	79	127	.	.	.
145	141	143	134	111	124	117	113	64	112	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.

Figura 2.3: Matríz $I(u,v)$ de valores de una imagen.

Relaciones entre pixeles

El elemento mínimo de esta matriz de imagen es el pixel. El tamaño de una imagen se determina directamente a partir del ancho M (número de columnas) y la altura N (número de filas) de la matriz de imágenes I. La resolución de una imagen indica las dimensiones espaciales de una imagen con respecto al mundo real como *dots per inch* (dpi) *lines per inch* (lpi).

Un pixel p en la posición (u,v) siempre tiene a su lado un pixel adyacente al cual se le denomina "vecino". Al conjunto de vecinos alrededor de un pixel se le denomina vecindad [29]. Hay dos relaciones importantes de vecindad de pixeles las cuales son:

- vecindad de 4 (N_4): Los cuatro pixeles adyacentes a un pixel p dado en las direcciones verticales y horizontales $(u+1,v)$, $(u,v-1)$, $(u-1,v)$, y $(u,v+1)$
- vecindad de 8 (N_8): Los pixeles contenidos en N_4 más los pixeles adyacentes en las esquinas: $(u+1,v)$, $(u+1,v-1)$, $(u,v-1)$, $(u-1,v-1)$, $(u-1,v)$, $(u-1,v-1)$ y $(u,v+1)$, $(u+1,v+1)$.

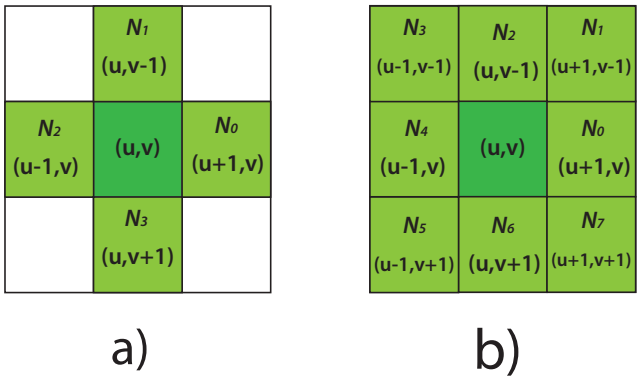


Figura 2.4: a)Vecindad de 4 pixeles. b)Vecindad de 8 pixeles.

Valores de los pixeles y tipos de color en imágenes

En una imagen, los pixeles toman valores de acuerdo a su intensidad y color, se codifican como palabras binarias de valor k, donde k es la información de la imagen acerca de ese color y brillo y con una posición espacial en la imagen. A continuación, se da una breve descripción de estos tres tipos principales de imágenes:

- Imagen binaria: Son imágenes de un solo canal o matriz de imagen donde sus pixeles solo pueden tomar dos valores, blanco o negro. En términos de bits, solo pueden tomar dos valores $(0/1)$ por pixel.
- Imagen en escala de grises: Es un tipo de imagen que consiste en una imagen de un solo canal que representa la intensidad de brillo de la imagen. Los valores típicos que pueden tomar cada pixel son 0 a $2^k - 1$ que en 8 bits serian de 0 a 255, donde 255 es el valor de intensidad más claro (blanco) y 0 el valor de intensidad más obscuro (negro).
- Imagen RGB: La mayoría de imágenes a color están compuestas por los tres componentes primarios rojo, verde y azul (RGB por sus siglas en ingles). Estas imágenes están compuestas por 3 canales o matrices de imagen, donde si el valor de cada color se codifica con 8 bits pueden tomar valores de 0 a 255 por cada color.

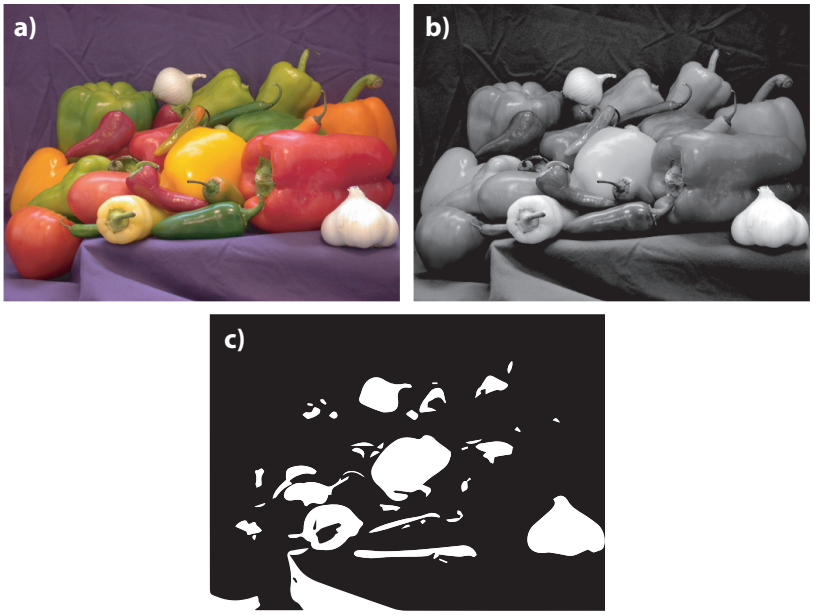


Figura 2.5: Tipos de imágenes. a)Imagen *RGB*. b)Imagen en escala de grises. c)Imagen binaria.

2.2.2. Etapas del sistema de visión

Preprocesado

El preprocesado de una imagen pretende reparar en la imagen desperfectos producidos por el *hardware*, o el ambiente: ruido, poco contraste o brillo, falta de ecualización, etc. Los algoritmos de preprocesado permiten modificar la imagen para eliminar ruido, transformarla geométricamente, mejorar la intensidad o el contraste, etc [28].

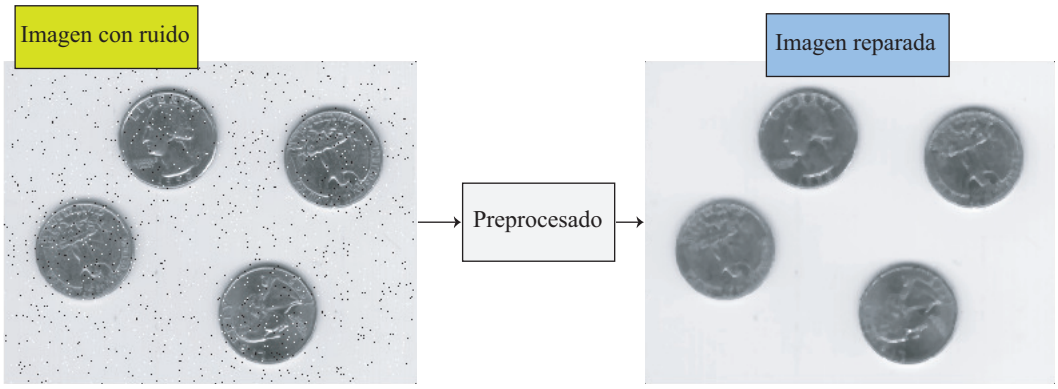


Figura 2.6: Ejemplo de reparación de una imagen en la etapa de preprocesado.

Segmentación y extracción de características

En el bloque de segmentación la imagen se divide en partes aislando los diferentes objetos de interés contenidos en la imagen por medio de diferentes técnicas. Esta es una de las partes más importantes en el procesamiento de imágenes ya que de esta etapa depende la extracción de características, por lo que si se realiza una correcta segmentación se facilitaran las etapas posteriores.

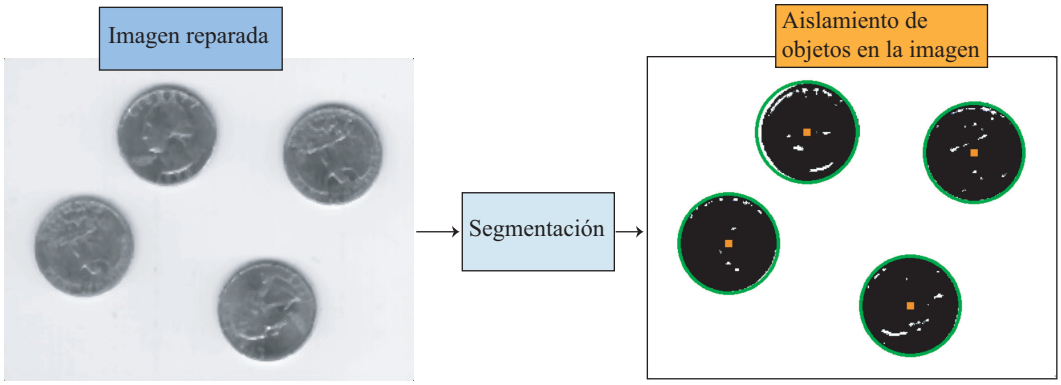


Figura 2.7: Aislamiento de objetos de interes en la imagen.

Una vez aisladas las regiones de interés, es decir, cuando se tienen ya los datos de los puntos de estas regiones, se procede a procesarlos para que queden de una forma adecuada y propia para la aplicación de la tarea a realizar. Con estos datos se puede reconocer los diferentes objetos que se encuentran en la imagen, realizar tareas de seguimiento de los objetos, extraer características útiles como centros geométricos, área, perímetros y otras características geométricas.

2.3. Sistemas Embebidos

Un sistema embebido se compone de diferentes elementos de *hardware* y *software* operando de manera integral, se distingue de otros sistemas computacionales porque es diseñado para realizar una función específica y no de uso general. Sin embargo, con los avances tecnológicos, es difícil tener una definición precisa de lo que es un sistema embebido. A continuación, se muestran algunas descripciones de sistema embebido [30][31][32] :

- Una combinación de hardware y software de computadora y tal vez partes adicionales, ya sea mecánicas o electrónicas diseñadas para realizar una función dedicada
- Sistema electrónico y de computación con las siguientes características:
  - Más limitado en la funcionalidad del hardware y/o software que una computadora personal (PC).
  - Diseñado para realizar una función dedicada.
  - Un sistema de computo con requisitos de calidad y fiabilidad superiores a los de otros tipos de sistemas de computo.
- Dispositivos electrónicos que incorporan microprocesadores en sus implementaciones. Los principales propósitos del microprocesador son simplificar el diseño del sistema y proporcionar flexibilidad.
- Un sistema basado en un microprocesador que está construido para controlar una función o rango de funciones y no está diseñado para ser programado por el usuario final de la misma manera que una PC. El usuario puede elegir la funcionalidad, pero no puede cambiar la funcionalidad del sistema añadiendo o sustituyendo *software*.

El modelo general de un sistema embebido se muestra en la figura 2.8, los sistemas embebidos se componen de tres capas: hardware, software de sistema y software de aplicación.

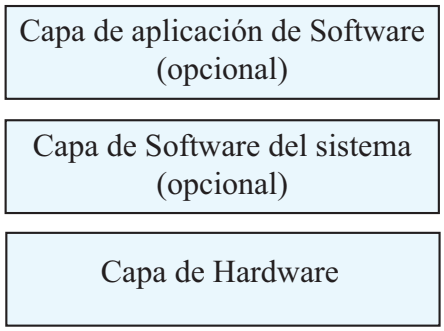


Figura 2.8: Modelo general de los sistemas embebidos.

2.3.1. FPGA como sistema embebido

FPGA es un dispositivo de arreglo de compuertas programable, que en general consta de 3 elementos básicos que son el bloque lógico configurable o *configurable logic block* (CLB), las interconexiones y los bloques de entrada/salida o *input output blocks* (I/O), como se ilustra en la Figura 2.9. Sin embargo, la arquitectura de un FPGA es diferente entre fabricantes. Los principales fabricantes de FPGA son Altera y Xilinx, cada uno ofrece modelos con ligeras diferencias en las arquitecturas, por ejemplo, el fabricante Xilinx aparte de ofrecer los 3 bloques básicos tambien ofrece un *Block* RAM (del inglés *Block Random Access Memory*) y un DSP Blocks (del inglés *Digital Signal Processing Blocks*)[33].

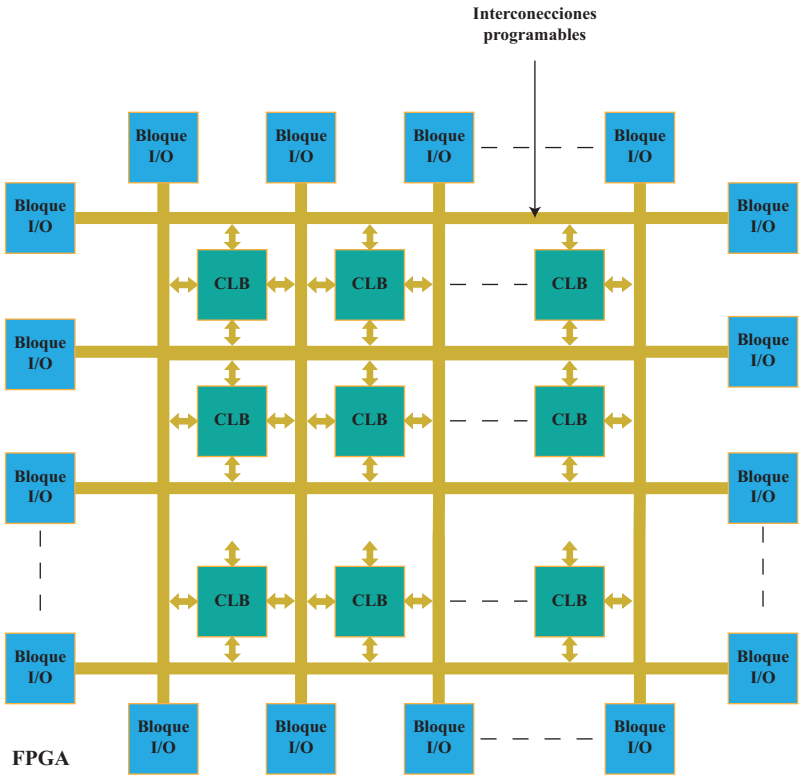


Figura 2.9: Modelo general de un FPGA.

A continuación, se definen cada uno de los principales elementos internos de un FPGA:

- CLB: son Elementos básicos en la arquitectura de una FPGA. Pueden ser configurados para realizar funciones lógicas. Estos bloques tienen dentro *slices* que a su vez se componen de flip flops y LUTs.
- I/O: entradas y salidas que comunican al FPGA con sistemas externos.
- LUT: una LUT (*look-up table*) es un tipo de memoria programable que se utiliza para generar funciones lógicas combinadas SOP (del inglés *sum of products*)
- RAM: bloques de memoria interna dentro del FPGA.
- DSP: El FPGA contiene funciones de memoria incorporada, así como funciones de procesamiento de señales digitales (DSP). Las funciones DSP, como los filtros digitales, se utilizan comúnmente en muchos sistemas.

Desde su introducción, el uso de los FPGA se ha incrementado en diversas aplicaciones donde destacan principalmente en comunicaciones, procesamiento de imágenes, ingeniería en control y redes [34][35]. Debido a este incremento de uso de los

FPGA, se ha buscado embeber varias funcionalidades de cómputo y electrónica en un solo chip surgiendo varias soluciones como los sistemas en chip (SoC por sus siglas en inglés), que son circuitos integrados que ofrecen soluciones integrando módulos que componen a una computadora además de otros sistemas electrónicos. Existen *SoC* con tecnología de FPGA y derivaciones de estos como los sistemas en chip multiprocesador (MPSoC por sus siglas en inglés), que son SoC con un procesador multinucleo integrado, o los Todo programable en un chip (APSoC por sus siglas en inglés), que son una forma de integrar en un chip la logica programable (PL por sus siglas en inglés) y un sistema de procesamiento (PS por sus siglas en inglés).

2.3.2. Internet de las cosas

Internet de las cosas (IoT, por sus siglas en ingles), es un término que lo acuño Kevin Ashton donde se refiere a sensores conectados a través de RFID [36]. Según la *Unión Internacional de Telecomunicaciones* (ITU por sus siglas en inglés), en su perspectiva de la normalización técnica, es una infraestructura mundial para la sociedad de la información que permita prestar servicios avanzados mediante la interconexión de elementos (físicos y virtuales) basados en las tecnologías de la información y las comunicaciones interoperables existentes y en evolución. La ITU recomienda que la arquitectura de la IoT consista en 4 capas y de capacidades de gestión y de seguridad relacionadas con estas cuatro capas: Capa de aplicación, Capa de apoyo a servicios y aplicaciones, Capa de red y Capa de dispositivo[1].

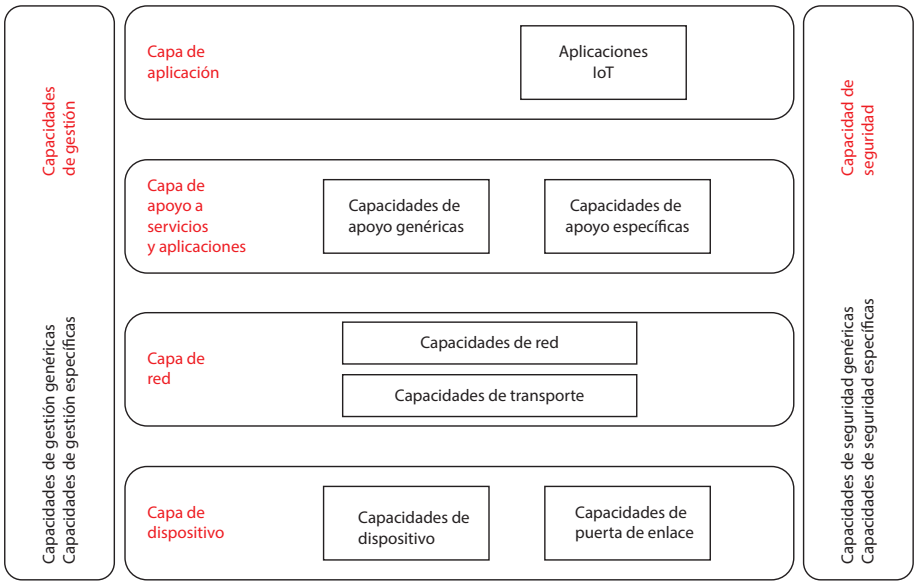


Figura 2.10: Modelo de referencia de IoT [1].

- Capa de aplicación:
  La capa de aplicación contiene aplicaciones IoT.
- Capa de soporte de servicios y aplicaciones:
  consiste en los siguientes dos grupos de capacidades:
  - Capacidades de soporte genéricas:
    Son capacidades comunes que pueden ser utilizados por diferentes aplicaciones de IoT, como el procesamiento o el almacenamiento de datos.
  - Capacidades de soporte específicas:
    Pueden consistir en diversos grupos de capacidades precisas que ofrecen distintas funciones de apoyo a las diferentes aplicaciones IoT.
- Capa de red:
  Consiste en los dos tipos siguientes de capacidades:
  - Capacidades de red:
    Proporcionan funciones de control pertinentes de la conectividad de la red, como las funciones de control del acceso y de los recursos de transporte, la gestión de la movilidad o la autenticación, autorización y contabilidad.
  - Capacidades de transporte:
    Se centran en proporcionar conectividad para el transporte de información de datos de servicios y aplicaciones específicas de IoT, así como en el transporte de información de control y gestión relacionada con la IoT.
- Capa de dispositivo:
  Se pueden clasificar lógicamente en dos tipos de capacidades:
  - Capacidades de dispositivo:
    Son, pero no se limitan a:
    - Interacción directa con la red de comunicación:
      Los dispositivos pueden reunir y cargar información directamente (es decir, sin utilizar capacidades de puerta de enlace).
    - Interacción indirecta con la red de comunicación:
      Los dispositivos son capaces de reunir y cargar información en la red de comunicación de forma indirecta, es decir, a través de las capacidades de la puerta de enlace.
    - Redes ad-hoc:
      Los dispositivos puede construir redes de manera ad-hoc en algunas circunstancias cuando sea necesario.

- Modo reposo y activo: Las capacidades de dispositivo deben disponer de mecanismos para pasar a los modos “reposo” y “activo” a fin de ahorrar energía.

- Capacidades de puerta de enlace:
  Son, pero no se limitan a:
  - Soporte de interfaces múltiples:
    En la capa de dispositivo, las capacidades de puerta de enlace soportan dispositivos conectados mediante diferentes tipos de tecnologías alámbricas e inalámbricas, tales como el bus de red de control de zona (CAN), ZigBee, Bluetooth o Wi-Fi. En la capa de red, las capacidades de puerta de enlace pueden comunicarse a través de diversas tecnologías, tales como la red telefónica pública conmutada (PSTN), las redes de segunda o tercera generación (2G o 3G), las redes LTE (evolución a largo plazo), Ethernet o las líneas digitales de abonado (DSL).
  - Conversión de protocolo:
    Se necesitan capacidades de puerta de enlace cuando las comunicaciones en la capa de dispositivo utilizan protocolos diferentes, por ejemplo, protocolos de tecnología ZigBee y Bluetooth y cuando cuando en la comunicación intervienen la capa de dispositivo y la de red y se utilizan protocolos diferentes en cada una, por ejemplo, el protocolo de tecnología ZigBee en la capa de dispositivo y el protocolo de tecnología 3G en la capa de red.

- Capacidades de gestión:
  Las capacidades de gestión IoT pueden clasificarse en capacidades genéricas y específicas.
  Las capacidades de gestión genéricas en IoT son esencialmente las siguientes:

  - Gestión de dispositivos, como activación y desactivación de dispositivos remotos, diagnóstico, actualización del firmware y/o del software, gestión del estado de trabajo del dispositivo;
  - Gestión de la topología de red local;
  - Gestión del tráfico y la congestión, como la detección de las condiciones de saturación de red y la aplicación de reserva de recursos para los flujos de datos esenciales para la vida o urgentes.

  Las capacidades de gestión específicas están estrechamente relacionadas con los requisitos específicos de la aplicación, por ejemplo, requisitos de control de la línea de transmisión por la red de suministro eléctrico inteligente.

- Capacidades de seguridad:
  Hay dos tipos de capacidades de seguridad: genéricas y específicas. Las capacidades de seguridad genéricas son independientes de la aplicación y son, entre otras:

  - En la capa de aplicación:
    autorización, autentificación, confidencialidad de datos de aplicación y protección de la integridad, protección de la privacidad, auditorías de seguridad y antivirus;
  - En la capa de red:
    autorización, autentificación, confidencialidad de datos de señalización y de datos de uso, y protección de la integridad de señalización;
  - En la capa de dispositivo:
    autentificación, autorización, validación de la integridad del dispositivo, control de acceso, confidencialidad de datos y protección de la integridad.

  Las capacidades de seguridad específicas están estrechamente relacionadas con los requisitos específicos de la aplicación, por ejemplo, los requisitos de seguridad para el pago con el móvil.

Protocolos de aplicación IoT

Para IoT han surgido varios protocolos encargados de la presentación y formato de los datos [37][38]. A continuación se hace una lista de los protocolos más importantes y usados en IoT:

- MQTT: *Message queue telemetry transport* (*MQTT*) es un estándar de OASIS. MQTT utiliza el modelo cliente/servidor, típicamente usa el protocolo TCP/IP en el nivel de transporte y su arquitectura incluye dos nodos de comunicación: los clientes y los servidores/*broker*. Los clientes pueden operar como publicadores o como suscriptores o ambos para enviar y recibir datos. Los mensajes se envían y reciben a través del *broker* que es un dispositivo central que actúa como distribuidor de los mensajes utilizando la dirección en la que se publica el mensaje que se denomina tópico (del inglés *topic*). Cada suscriptor debe suscribirse a uno o mas tópicos para recibir los mensajes subsiguientes que el publicador haya publicado sobre ese tema.
- CoAP: *Constrainded Aplication Protocol* (CoAP) es un protocolo que está diseñado para proporcionar permitir que las aplicaciones usen servicios RESTful con baja sobrecarga ya que estos servicios regularmente consumen recursos significativos. CoAP tiene 4 tipos de mensajes $confirmable$, $non-confirmable$,

piggyback, y *separate*, donde las primeras dos representan las transmisiones fiables y no fiables respectivamente. *Piggyback* es para la comunicación directa con el servidor mientras que *separate* viene con un mensaje separado del servidor que puede tardar algún tiempo.

- XMPP: *Extensible messaging and presence protocol* (*XMPP*) es un protocolo que originalmente surgió para aplicaciones de mensajería de chat. Está basado en el lenguaje XML y soporta tanto la arquitectura de publicación/suscripción y la de solicitud/respuesta. No tiene garantía de calidad de servicio y puede consumir recursos por lo cual es raramente usado en aplicaciones IoT.

- AMQP: *Advanced message queuing protocol* (*AMQP*) es otro protocolo estándar de OASIS, es muy similar a MQTT y su principal diferencia es que incluye un mecanismo de intercambio de mensajes con colas separadas para los respectivos suscriptores y asi distribuir el mensaje adecuadamente a su respectiva cola con el uso de su dirección virtual.

- DDS: *Data distribution service* (*DDS*) es un protocolo desarrollado por OMG, utiliza la arquitectura de publicación/suscripción y es principalmente utilizado para la comunicación M2M. Es un protocolo de alta fiabilidad ya que ofrece una variedad de criterios de calidad, entre ellos: seguridad, urgencia, prioridad, durabilidad, fiabilidad, etc.

Tabla 2.1: Principales características de los protocolos IoT.

Protocolo	Características principales			
	Protocolo de transporte	Arquitectura	Seguridad	QoS
MQTT	TCP	Pub/Sub	Opcional TLS/SSL	QoS 0 (en más de una vez) QoS 1 (en por lo menos una vez) QoS 2 (exactamente una vez)
CoAP	UDP	Req/Res	Opcional DTLS	NON CON
XMPP	TCP	Pub/Sub, Req/Res	TLS/SSL embebido	no
AMQP	TCP TCP	Pub/Sub Pub/Sub	TLS/SSL embebido Opcional TLS/SSL	A lo más una vez Al menos una vez Exactamente una vez
DDS	TCP/UDP TCP	Pub/Sub Pub/Sub	TLS/DTLS/ DDS sec., Opcional Opcional TLS/SSL	Casi 23 niveles de QoS

Capítulo 3

Diseño e implementación

En este capítulo se presenta la metodología de diseño y los pasos a seguir para lograr cada una de las etapas del sistema de captura de posturas. También se presentan los requerimientos del sistema, así como los criterios de selección de las herramientas de *software* (programas computacionales, algoritmos, lenguajes, etc.) y de *hardware* (dispositivos/sensores-transductores tarjetas electrónicas, circuitos, plataforma robótica, etc.). Para finalizar se muestra la integración e implementación del sistema completo.

3.1. Metodología

Con el fin de alcanzar los objetivos planteados para la realización de este proyecto, se propone la siguiente metodología:

- Requerimientos de *hardware* y *software* del sistema.
- Revisión bibliográfica del estado del arte
- Revisión teórica y técnica referente al proyecto.
- Diseñar la arquitectura general del sistema.
- Conocer la plataforma robótica a utilizar, así como los entornos de simulación robótica.
- Investigación de sistemas embebidos.
- Se define un área de trabajo para la captura de movimientos.
- Programación y prueba de los algoritmos de procesamiento de imagenes para comprobar su funcionalidad.

- Implementación de los algoritmos funcionales en el sistema embebido.
- Integración del sistema embebido con el humanoide.
- Pruebas de funcionamiento y obtención de resultados.

3.2. Requerimientos del sistema

Los sistemas electrónicos son un área multidisciplinaria con muchas vertientes, en las cuales están involucrados tanto el diseño, integración, así como el manejo de equipo electrónico y tecnológicas de la información. Los requerimientos del proyecto con respecto a dispositivos, tecnologías informáticas y de computo a utilizar se puede dividir en dos grupos, requerimientos de *hardware* y de *software*. Estos requerimientos están en función de los requerimientos generales del proyecto, es decir, que es lo que debe hacer el sistema de acuerdo a las necesidades de investigación del laboratorio de electrónica y automatización del IIMAS y a su infraestructura.

3.2.1. Requerimientos generales

Los requerimientos generales del sistema, deben satisfacer el comportamiento que se requiere para la operación del sistema (que se espera que haga). Estos requerimientos tienen que ver directamente con las investigaciones realizadas en el laboratorio de electrónica y automatización del IIMAS, y por consecuencia, de los objetivos planteados para este proyecto. Los requerimientos se enlistan a continuación:

- El sistema debe capturar el entorno visual donde se encuentra un sujeto de prueba realizando ciertos movimientos corporales.
- La información debe ser procesada en un sistema embebido.
- La información procesada se utilizará para teleoperar al robot por medio de un protocolo de internet de las cosas de transporte de mensajes ligero.
- Se realizará un almacenamiento de datos.
- Se hará uso de entornos de simulación robótica.
- El sistema debe ser escalable en cuanto a la modularidad y capacidad.

A partir de los requerimientos generales se puede realizar una revisión de arquitecturas y diagramas de bloques de los que están compuestos otros sistemas de *tracking* o seguimiento y de teleoperación para con ello bosquejar un modelo de arquitectura del sistema propio basado en el flujo de los requerimientos y la literatura.

3.2.2. Requerimientos de *hardware*

En general, los bloques que consitituyen un sistema electrónico son la entrada, que toma las variables del mundo físico por medio de sensores y/o transductores, el bloque de procesamiento que interpretan y manipulan la información eléctrica enviada por el bloque anterior y la salida donde se obtienen las señales eléctricas o datos de interés para llevar a cabo una acción.

Cámara

En el caso de los sistemas electrónicos de procesamiento de imágenes y de este proyecto, como bloque de entrada se requiere de una cámara, elemento de gran importancia que captura el entorno de donde se obtiene la región de interés para tener la información requerida para captura de movimientos. La cámara a usar en el proyecto es un requerimiento que forma parte de la infraestructura del laboratorio, la cual es una cámara *Logitech webcam c930*. La tabla 3.1 muestra sus principales características.

Tabla 3.1: Características de la cámara

Calidad de video	Cuadros por segundo	Campo visual	Tipo de conexión
HD 1080p	30fps	90°	USB 2.0 compatible con USB 3.0

Robot DARwIn-OP

Otro requerimiento que forma parte de la infraestructura del laboratorio es el robot humanoide DARwIn-OP. DARwIn-OP (del inglés Dynamic Anthropomorphic Robot with Intelligence–Open Platform)figura 3.8, es una plataforma robot-humanoide en miniatura asequible con una avanzada potencia de cálculo, sensores sofisticados, alta capacidad de carga útil y capacidad de movimiento dinámico para permitir cualquier actividad de investigación y educación [2].

Algunas de las características más importantes del robot son:

- Altura: 454,5 mm (17,89 pulgadas)
- Peso: 2,9 kg (6,4 lb)

- PC incorporado: Intel Atom Z530 de 1,6 GHz (32 bits), SSD flash de 4 GB
- Controlador de gestión (CM-730): ARM CortexM3 STM32F103RE 72 MHz
- 20 actuadores MX-28T (6 DOF pierna × 2 + 3 DOF brazo × 2 + 2 DOF cuello) con engranajes metálicos
- Bus de alta velocidad de 3 Mbit/s de Dynamixel para el control conjunto
- Giroscopio de 3 ejes, acelerómetro de 3 ejes, botón × 3, micrófono de detección × 2
- Funcionalidad versátil (puede aceptar periféricos heredados, actuales y futuros)

Figura 3.1: Robot DARwIn-OP [2].

Otro requerimiento del proyecto, para la etapa de comunicaciones y de procesamiento,

es que estas etapas deben ser implementadas en un sistema embebido con las siguientes características:

- Lógica programable (PL) para la aceleración de procesos de cómputo y un procesador (PS).
- Estar adaptado para el uso de programación en Python.
- Ser compatible con el protocolo de comunicación de la cámara (USB).
- Estándar de redes de computadora para acceder a internet (Ethernet/ Wi-Fi).
- Poder aplicar protocolos de comunicación de internet de las cosas (MQTT/MQTT-SN).
- Tener un sistema operativo basado en Linux.
- Bajo costo.
- Buena documentación y soporte

3.2.3. Requerimientos de *software*

Como parte de los objetivos del proyecto se busca que sea escalable para poder contribuir con futuras investigaciones del laboratorio de electrónica y automatización del IIMAS. Por esta razón se requiere hacer un almacenamiento histórico de los ángulos obtenidos en una base de datos.

También previo a una implementación física en el robot, se requiere probar la salida del sistema en un ambiente robótico de simulación.

En la parte de comunicaciones, se hará uso de un protocolo de internet de las cosas para realizar la teleoperación del robot.

Para la parte de *software* los requerimientos son los siguientes:

- Entorno de simulación física de robots por computadora que sea *software* libre, y que cuente con lenguaje o lenguajes multiplataforma y multiparadigma para la interoperabilidad.
- Infraestructura informática de *software* libre para el almacenamiento de datos.

3.3. Diseño del sistema

El desarrollo de la arquitectura general del sistema, define la estrastegia a seguir para lograr que cada uno de sus bloques sean funcionales. El sistema consta de 4 etapas principales, la entrada del sistema donde se define como abordar el problema de visión, el sistema embebido para la optimización y computo dedicado, el canal de comunicaciones para la transmisión de mensajes ligeros para la comunicación con las etapas posteriores y los objetivos de prueba para realizar los movimientos imitativos que son la simulación y el robot DARwIN-OP como se muestra en la figura 3.2.

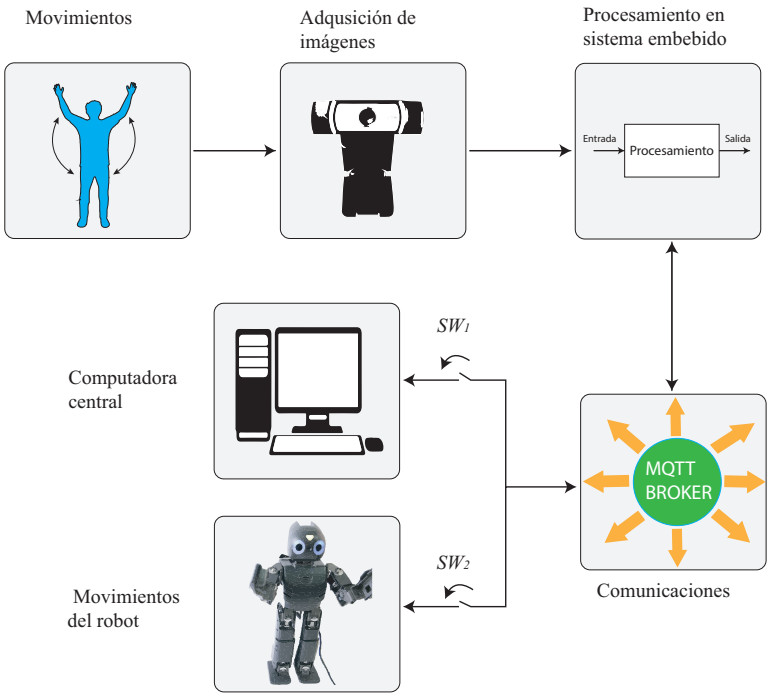


Figura 3.2: Diagrama de bloques general del sistema.

En la primera etapa, se captura la información cruda del entorno, en donde el operario, con marcadores colocados en partes específicas de los miembros superiores, se coloca en el área de trabajo para capturar con la cámara los movimientos realizados.

En la segunda etapa, la de procesamiento, son implementadas en un sistema embebido técnicas de procesamiento de imágenes tales como filtrado de color, filtrado de ruido, adelgazamiento, etc. En donde se obtienen puntos de interés con los que se calculan por medio de métodos geométricos los ángulos que controlan los movimientos del robot. En la etapa de comunicaciones la tarjeta debe conectarse a la red por medio

del protocolo de red Ethernet. Posteriormente se inicializa el servidor-*broker* y , por medio de este último, se envían los ángulos tanto al robot como a una computadora central, en la que se implementa un servidor WAMP para hacer un almacenamiento histórico de los ángulos que el sistema embebido envía por medio del protocolo de internet de las cosas MQTT. Con estos ángulos almacenados también se controla el movimiento del robot en el entorno de simulación robótica Webots.

En la figura 3.3 se desglosa cada bloque de la arquitectura general del sistema, y se muestran los subprocesos o subsistemas con los que cuenta.

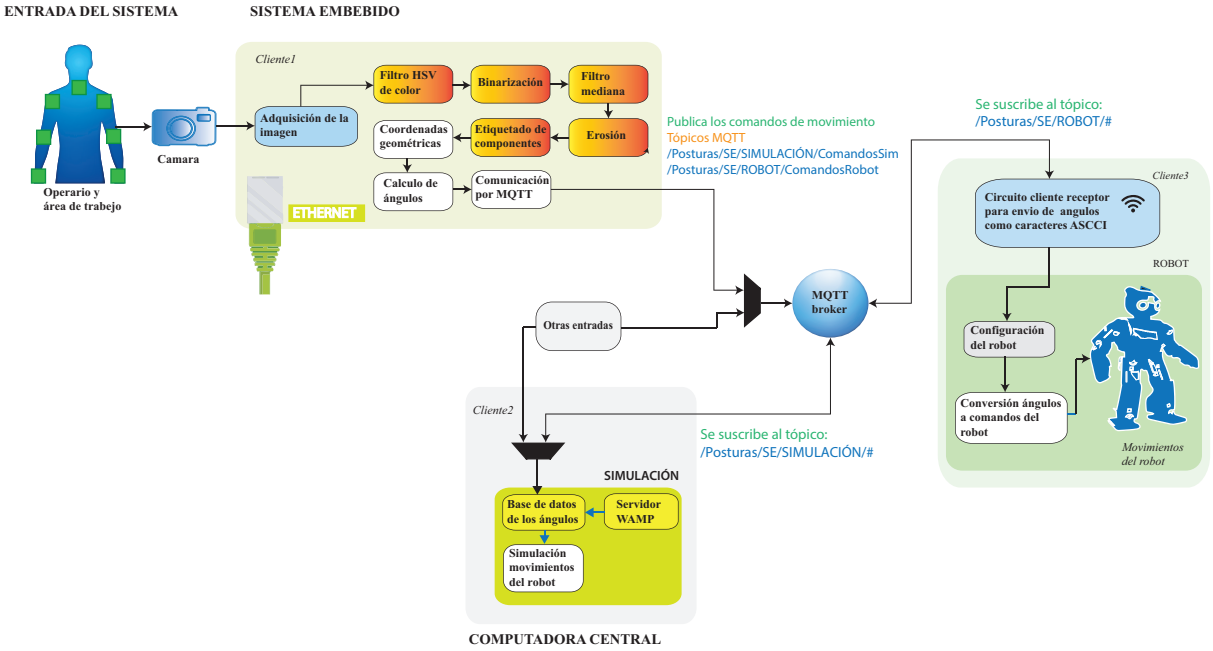


Figura 3.3: Arquitectura desglosada del sistema.

En el diagrama de la figura 3.3 , también se muestra un bloque llamado "*otros*". Este bloque indica otros clientes que pueden publicar o suscribirse a los tópicos, como la computadora central u otros clientes parte del laboratorio como la tarjeta Raspberry Pi.

3.3.1. Entrada del sistema

La etapa de entrada se compone del sujeto de prueba en el área de trabajo y la cámara que es el elemento que capturara el entorno.

Recordando los objetivos del proyecto, donde se plantea aportar una estrategia de captura de posturas del cuerpo humano, junto con el diseño e implementación de algoritmos de obtención de esqueleto de una imagen, se revisaron en la literatura algoritmos de procesamiento de imágenes, estrategias de estimación de pose y análisis cinemático para describir los movimientos del robot [39][40][41][42][24][43][28]. Con base en esta busqueda, se decidió simplificar el problema de seguimiento de visión colocando *markers* o marcas en las articulaciones de interés del cuerpo (manos, codos, hombros y pecho) para discriminar el entorno por técnicas de detección de color.

En diferentes trabajos realizados [44] donde utilizan la discriminación por color, se usan *markers* o *check points* de diferentes colores para la detección de puntos de interés. En este trabajo se optó por dar una variante utilizando todos los marcadores de color verde, basándose en la técnica de croma o *chroma Key*. Esta es una técnica audiovisual utilizada ampliamente tanto en cine, televisión y fotografía , con la cual se busca un color uniforme de fondo fácilmente detectable para posteriormente sustituir con una parte de otra imagen o video[45]. En este caso la idea es tener un color uniforme al igual que fácil de detectar, pero al contrario que la técnica de *chroma Key* (ver figura 3.4), no se busca sustituir con otra parte de video o imagen todo lo que se encuentre en color verde, sino solo detectarlo.

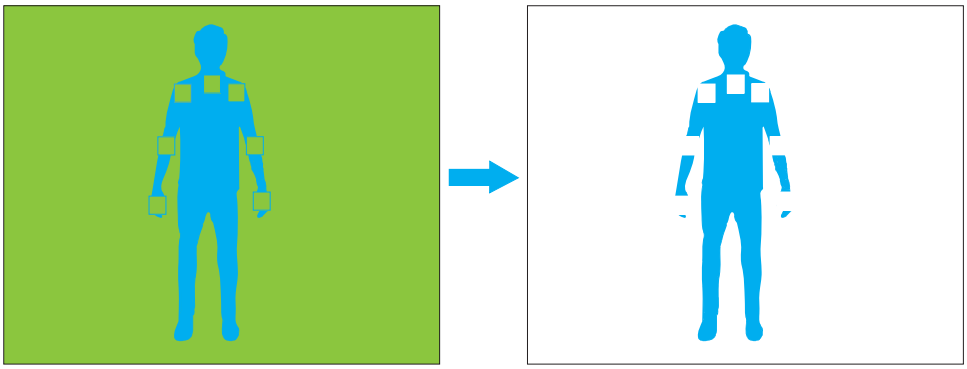


Figura 3.4: Ejemplo del funcionamiento de la técnica *Chroma Key* con fondo monocromático. Se puede observar cómo se sustituye todo lo que se encuentre en color verde.

Área de captura

Para definir el entorno de trabajo, se tomó como principio que la mayoría de los ambientes construidos por el hombre tienen restricciones, por lo cual las pruebas se harán en ambientes cerrados semi-controlados, como el laboratorio de electrónica y automa-

tización del IIMAS y un ambiente con fondo monocromatico. Se menciona ambiente "semi-controlado" ya que las técnicas de procesamiento le dan cierta robustez a los cambios de luz y a ciertos cambios de color en el entorno.

Una condición importante para el rango de visión de la cámara, es que este alcance a capturar todas las posturas a realizar por el sujeto de prueba, no es necesario capturar todo el cuerpo, solo la región de interés (parte del torso y miembros superiores) como se muestra en la figura 3.5.

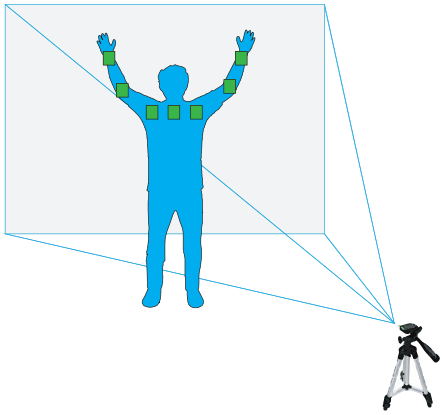


Figura 3.5: Operario en el área de trabajo dentro del rango de visión de la cámara.

3.3.2. Movimientos del operario

Una vez definida el área de trabajo, se procede a definir los movimientos a estudiar. Existen dos configuraciones llamadas codo arriba y codo abajo para los robots manipuladores antropomórficos como se muestra en la figura 3.6. Al elegir una de estas configuraciones se evitan ambigüedades en el estudio de los movimientos del brazo [46].

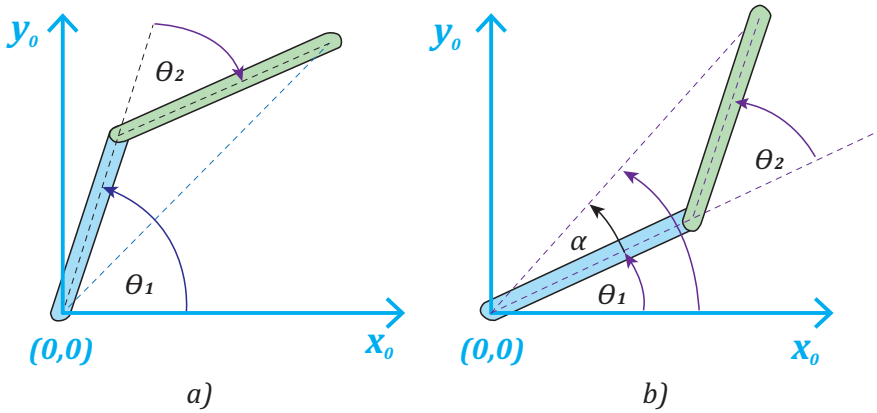


Figura 3.6: a)Configuración codo arriba. b)Configuración codo abajo.

Los movimientos que se definieron a imitar por el robot son los de las extremidades superiores, específicamente del hombro y el codo en un plano bidimensional x, y con la configuracion codo abajo, puesto que es el movimiento natural que tradicionalemnte realizan las personas, de igual manera se pondrán límites de apertura hacia arriba y hacia debajo de los brazos del operario debido a las limitaciones del robot. La figura 3.7 muestra la correspondencia de los movimientos a realizar por el operario con el robot.

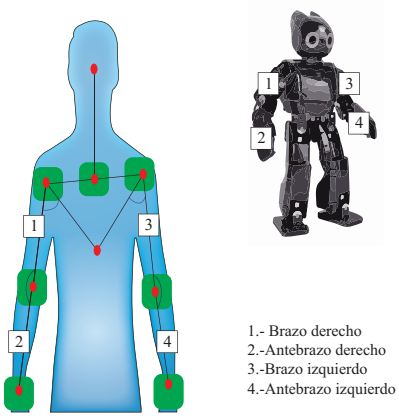


Figura 3.7: Correspondencia de los movimientos de las extremidades del robot y las del sujeto de prueba.

3.4. Elección de los elementos del sistema

En esta sección se mostrarán los criterios de selección de los elementos que conformarán el sistema. Con base en los requerimientos generales, de *hardware* y *software*, se hace una revisión y comparativa de las opciones existentes para la solución requerida.

3.4.1. Elección del hardware

El sistema const de 4 principales elementos de *hardware* que son, el sistema embebido, el robot, la cámara y una computadora central. La cámara y el robot, como se mencionó anteriormente, son requerimientos del sistema.

Elección del Sistema Embebido *(SE)*

Los requerimientos establecidos, definen la elección de la tarjeta electrónica a utilizar, pero también se tomaron en cuenta factores como el precio, el soporte, la documentación disponible, entre otros.

El lenguaje de programación es un aspecto importante ya que en la actualidad hay tendencias de uso y popularidad para ciertos lenguajes como lo muestran el ranking oficial de la IEEE *"The Top Programming Languages"* [47], que desde hace algunos años tiene posicionado a Python entre en el top y en los últimos 3 años ha ocupado el puesto número 1. Además, otros estudios como el realizado por la revista *EE Times* [48], muestran el reciente incremento de *Python* en el campo de los sistemas embebidos donde se encuentra en el lugar 3 de la lista de proyectos en los que actualmente desarrolladores están realizando sus proyectos y en los que están pensando en realizar, tan solo por debajo de c y c++.

Tabla 3.2: Los principales lenguajes de programación segun la IEEE 2019

Posición	Lenguaje	Usos principales
1	Python	Web, Empresarial/Científico, Embebido
2	Java	Web, Movil, Empresarial/Científico
3	C	Movil, Empresarial/Científico, Embebido
4	C++	Movil, Empresarial/Científico, Embebido
5	R	Empresarial/Científico

Tabla 3.3: Uso de lenguajes para sistemas embebidos

Posición	Lenguaje
1	C
2	C++
3	Python
4	Ensamblador
5	Java

Debido a estas tendencias, Python es ampliamente usado en el marco de trabajo del laboratorio de electrónica y automatización del IIMAS.

El proyecto requiere una tarjeta electrónica con parte PS y PL para acelerar el procesamiento, por lo cual debe ser una tarjeta FPGA del tipo SoC, PSoC, APSoC, otro requerimiento para el sistema embebido es que este adaptado para el lenguaje Python. Tomando en cuenta el lenguaje de desarrollo elegido y la tecnología del circuito integrado, se encontró que una de las marcas líderes, Xilinx tiene líneas las siguientes líneas de tarjetas: Arty A7: Artix-7, Zybo Z7, Nexys A7, ZedBoard, Arty S7, PYNQ-Z1, etc. Estas tarjetas tienen parte PS y PL, además de que se puede implementar en su parte PS un sistema operativo basado en Linux llamado Petalinux. Se eligió la tarjeta PYNQ-Z1 ya que esta esta adaptada para utilizar el *Framework* PYNQ. PYNQ es un proyecto de código abierto de Xilinx que hace mas asequible el uso de las plataformas Xilinx. Utilizando el lenguaje y las bibliotecas de Python, los desarrolladores pueden

aprovechar los beneficios de la lógica programable y los microprocesadores.

Algunas aplicaciones para las cuales esta optimizada la tarjeta son:

Visión por computadora.

Control industrial.

Internet de las cosas (IoT).

Drones.

Cifrado.

Aceleración informática embebida.

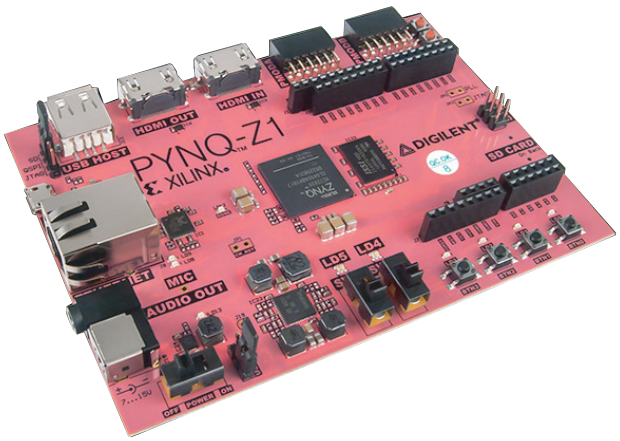


Figura 3.8: Tarjeta de desarrollo PYNQ-Z1.

Las especificaciones y características clave se muestran a continuación:

Convertidor analógico-digital en chip	Controladores periféricos de alto ancho
(XADC).	de banda: 1G Ethernet, USB 2.0, SDIO.
220 *slices* de DSP.	Controlador de memoria DDR3 con 8 canales
4 PLL y 4 MMCM.	DMA y 4 puertos esclavos AXI3 de alto rendimiento.
630 KB de bloques rápidos RAM.	Procesador Cortex-A9 de doble núcleo de 650MHz.
Lógica programable (PL) de la familia Artix-7	ZYNQ XC7Z020-1CLG400C.
Programable desde JTAG, *flash* Quad-SPI,	Un servidor web que alberga el entorno de
y tarjeta microSD.	diseño del Cuaderno Jupyter.
Controlador periférico de bajo ancho	El núcleo y los paquetes de IPython.
de banda: SPI, UART, CAN, I2C.	Linux.
	Biblioteca de *hardware* de base y API para la FPGA.

3.4.2. Elección del *software*

Los dos principales elementos de *software* que requiere el sistema son el entorno de simulación robótica y la base de datos para almacenar los ángulos.

Con respecto al *software* de simulación, en [49] se presenta una visión general de herramientas para simulación de dinámica de los robots, junto con una evaluación de los usuarios. Para robótica de humanoides colocan como la mejor elección a Gazebo, sin embargo, se eligió la herramienta Webots, que es una herramienta también bien evaluada en este trabajo, donde la sitúan como una herramienta de gama media. Las razones de la elección de esta herramienta es la variedad de lenguajes de desarrollo para la simulación entre los cuales se encuentra Python, se puede instalar en diversos sistemas operativos, es *open source* y dentro de la documentación oficial de los desarrolladores del robot DARwIN-OP se encuentra recomendado y documentado para su uso [2][50][51].

Para el almacenamiento de datos de los ángulos en la computadora central se optó por un servidor apache a través de una paquetería de software LAMP, WAMP, XAMP, etc. Esta paquetería es idónea ya que es *open source*, para sistemas tipo Unix, Windows, Mac, etc. y, permitirá escalabilidad[52].

3.4.3. Adquisición de la imagen

Cámara y adqusición

En los sistemas electrónicos instrumentados, el elemento fundamental para obtener las variables físicas del entorno es el sensor/transductor, el cual presenta un cambio ante una variable física o química para posteriormente convertir esta reacción en un tipo de energía de otra naturaleza (generalmente eléctrica). En el caso de un sistema de visión artificial o de seguimiento visual, uno de elementos de *hardware* necesario es la cámara, por medio de la cual se obtiene la información visual.

El protocolo de comunicación de la cámara a utilizar es USB 2.0 compatiblecon USB 3.0. En la tarjeta PYNQ-Z1 el periférico USB está conectado la parte PS, lo cual, la adquisición de la imagen se puede realizar por *software* en el programa. La figura 3.9 muestra la arquitectura del APSoC de la familia Zynq-7000 de la tarjeta PYNQ-Z1, donde se puede apreciar la conexión de la parte del sistema de procesamiento con la lógica programable y a su vez la interconexión con los periféricos. Se puede observar que el periférico USB está conectada a la parte de procesamiento.

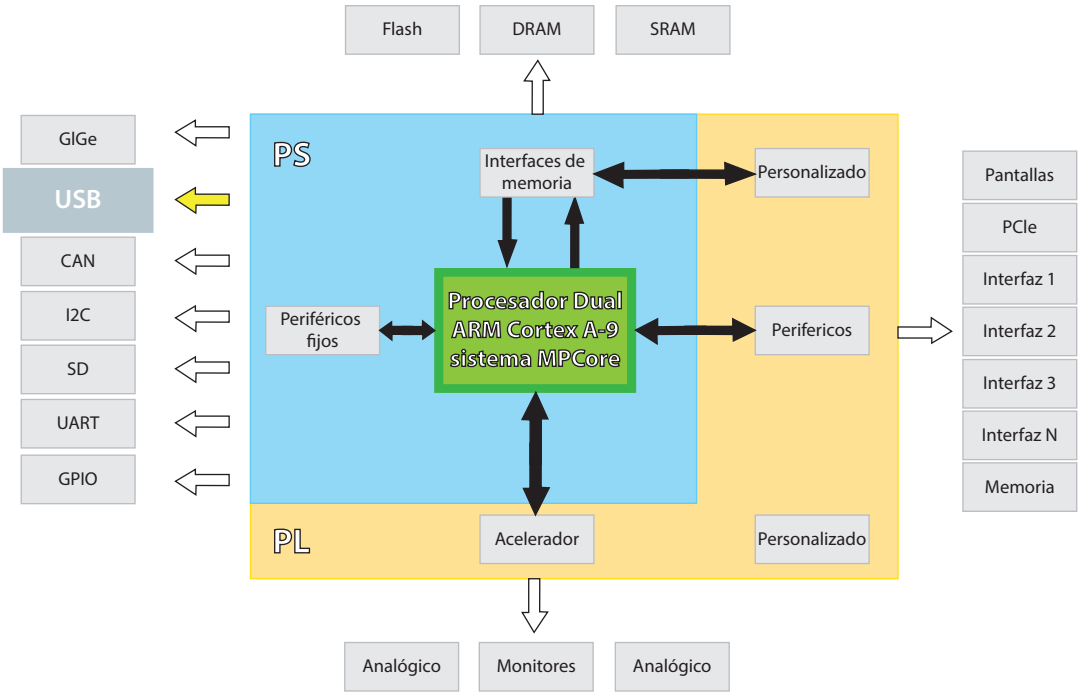


Figura 3.9: Arquitectura Zynq APSoC [3].

La tarjeta PYNQ-Z1 cuenta con un sistema operativo embebido basado en Linux, por lo que puede usarse el fragmento de código siguiente:

```
orig_img_path =
'/home/xilinx/jupyter_notebooks/Proyecto/Imagenes/Imagen.jpg'
!fswebcam --no-banner -r 640x480 --save {orig_img_path} -d
/dev/video0 2> /dev/null
```

La instrucción fswebcam es una aplicación de cámara web pequeña y simple para * nix (*unix-like*) o sistemas tipo Unix. Puede capturar imágenes de varias y diferentes fuentes y realizar una manipulación simple en la imagen capturada. La imagen se puede guardar como uno o más archivos PNG o JPEG. Los parametros que se fijan por default son el tamaño de “384x288", el formato de imágen JPEG, el formato de color RGB, etc., a menos que se indiquen otros valores.

Figura 3.10: Adquisición de la imágen.

Al adquirir de esta forma la imágen a través de la parte PS del APSoC Zynq, el Framework PYNQ permite tratar la imagen como un arreglo N-dimensional con la paquetería NumPy de Python [53].

3.4.4. Procesamiento

En esta sección se presentara el desarrollo e implementación de cada etapa dentro del procesamiento de la imagen utilizado en este proyecto.

Preprocesado (Filtro de color, binarización y filtros de ruido)

Para preparar la imagen para las etapas posteriores, se utilizaron técnicas de transformación de espacio de color y umbralización, filtrado, operaciones morfológicas, etc. El algoritmo comienza con un bucle ***Para***, donde se realizan 5 operaciones en un solo barrido de la imagen ImgRGB(i,j,c):

1. Las transformaciones de color RGB→HSV→RGB por medio de la función RGB_HSV_RGB*((R_d, G_n, B_l)* pasa los parametros R_d, G_n, B_l que son las componentes RGB de la imagen ImgRGB(i, j, c) y asi se filtra todo el fondo excepto los marcadores de color verde.

2. La binarización para simplificar la manipulación de pixeles en la imagen.

3. Se hace un prefiltrado de la imagen por medio de la función filtroMediana(i, j), donde se conservan los detalles de la imagen, es decir, no se difumina tanto.

4. Se erosiona la imagen por medio de la función Erosion(i, j) la cual degrada las islas de pixeles que no hayan sido eliminadas en la etapa anterior.

```
Para i ← 0 Hasta Fils + 11 Con Paso 1 Hacer
    Para j ← 0 Hasta Cols + 11 Con Paso 1 Hacer
        R_d, G_n, B_l ← ImgRGB(i,j,0), ImgRGB(i,j,1), ImgRGB(i,j,2)
                                  #Función RGB → HSV → RGB para filtro de color
        RGB(i,j,0), RGB(i,j,1), RGB(i,j,2) ← RGB_HSV_RGB((R_d, G_n, B_l))
                                                        #Binarización
        Si RGB(i,j,1) == 255 Entonces
            I(i,j) ← 0;
        En Caso Contrario
            I(i,j) ← 255;
        Fin Si
        filtroMediana(i,j);                          #Función filtro mediana
        Erosion(i,j);                                    #Función erosión
    Fin Para
Fin Para
```

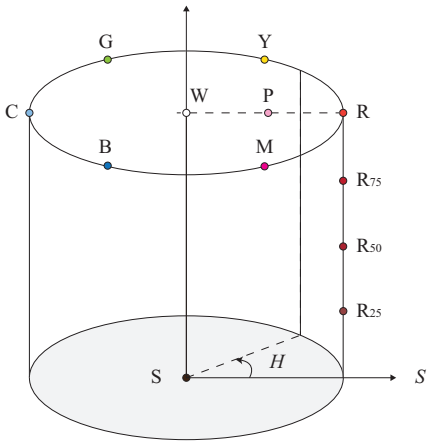
Figura 3.11: *Bucle* principal

Filtrado de color RGB→HSV→RGB

Para poder detectar el color se recurre a una técnica de transformación de espacio de color conocida como *Hue*, *Saturation*, *Value*/*Brightness* (HSV por sus siglas en inglés). Es una transformación no lineal del espacio de color, con la cual se deja en términos de los componentes de matiz o tonalidad (H), saturación (S) y valor de brillo (V) [54]. Se eligió esta técnica debido a que es robusta a cambios de iluminación además de que existen umbrales estandarizados para una amplia gama de colores [55][56][57].

La figura 3.12 a) muestra el espacio de color HSV como un cilindro con las coordenadas H como el ángulo, S como el radio, y V como la distancia a lo largo del eje vertical, que se extiende entre el punto negro S y el punto blanco W. En b) La tabla enumera los valores (R, G, B) y (H, S, V) de los puntos de color marcados en el cilindro. Los colores puros (compuestos de sólo uno o dos componentes) se encuentran en la pared exterior del cilindro $(S = 1)$, como lo ejemplifican los rojos gradualmente saturados (R25, R50, R75, R).

El algoritmo utilizado se muestra en la figura 3.13.



a)

Pt	Color	R	G	B	H	S	V
S	Negro	0.00	0.00	0.00	——	0.00	0.00
R	Rojo	1.00	0.00	0.00	0	1.00	1.00
Y	Amarillo	1.00	1.00	0.00	1/6	1.00	1.00
G	Verde	0.00	1.00	0.00	2/6	1.00	1.00
C	Cian	0.00	1.00	1.00	3/6	1.00	1.00
B	Azul	0.00	0.00	1.00	4/6	1.00	1.00
M	Magenta	1.00	0.00	1.00	5/6	1.00	1.00
W	Blanco	1.00	1.00	1.00	——	0.00	1.00
R_{75}	75%Rojo	0.75	0.00	0.00	0	1.00	0.75
R_{50}	50%Rojo	0.50	0.00	0.00	0	1.00	0.50
R_{25}	25%Rojo	0.25	0.00	0.00	0	1.00	0.25
P	Rosa	1.00	0.50	0.50	0	0.5	1.00

b)

Figura 3.12: a) Espacio de color HSV representado como un cilindro. b) Tabla de valores (R, G, B) y (H, S, V) de los puntos marcados en el cilindro.

Función RGB_HSV_RGB(r, g, b)

`#RGB a HSV`

$r, g, b \in [0, 1]$;

m_x, m_n, df $\leftarrow$ max(r, g, b), min(r, g, b), $m_x - m_n$;

Si $m_x == m_x$ **Entonces**

  $h \leftarrow 0$;

Si no Si $m_x == r$ **Entonces**

  $h \leftarrow mod\left((60)\left(\frac{g-b}{df}\right) + 360, 360\right)$;

Si no Si $m_x == g$ **Entonces**

  $h \leftarrow mod\left((60)\left(\frac{b-r}{df}\right) + 120, 360\right)$;

Si no Si mx $== b$ **Entonces**

  $h \leftarrow mod\left((60)\left(\frac{r-g}{df}\right) + 240, 360\right)$;

Si mx $== 0$ **Entonces**

  $s \leftarrow 0$;

Si no Entonces

  s $\leftarrow \frac{df}{m_x}$;

Fin Si

h, s, $v \leftarrow \frac{h}{2}, (s)(255), (m_x)(255)$;

`#Umbral`

Si $h \geq 25$ **y** $h \leq 110$ **y** $s \geq 110$ **y** $s \leq 255$ **y** $v \geq 140$ **y** $v \leq 255$ **Entonces**

  $h, s, v \leftarrow h, s, v$;

Si no Entonces

  $h, s, v \leftarrow 0, 0, 255$;

Fin Si

`#HSV a RGB`

$h, s, v \in [0, 1]$

$h', c_1, c_2 \leftarrow mod((6)(h), 6), |h'|, h' - c_1$

$x, y, z \leftarrow (1-s)(v), (1-(s)(c_2))(v), (1-(s)(1-c_2))(v)$;

Si no Si $h' == 0$ **Entonces**

  $r, g, b \leftarrow v, z, x$;

Si no Si $h' == 1$ **Entonces**

  $r, g, b \leftarrow y, v, x$;

Si no Si $h' == 2$ **Entonces**

  $r, g, b \leftarrow x, v, z$;

Si no Si $h' == 3$ **Entonces**

  $r, g, b \leftarrow x, y, v$;

Si no Si $h' == 4$ **Entonces**

  $r, g, b \leftarrow z, x, v$;

Si no Si $h' == 5$ **Entonces**

  $r, g, b \leftarrow v, x, y$;

Fin Si

$r, g, b \leftarrow$ min($(r)(255), 255$), min , min;

Fin Función

Figura 3.13: Función RGB-HSV-RGB

La función RGB_HSV_RGB() pasa las componentes RGB de la imagen, al espacio de color HSV como lo muestra el algoritmo. Cuando la imagen se encuentra en términos del espacio de color h, s y v se puede aplicar un umbral en donde se ponen en 0 todos los pixeles que no se encuentran entre este rango de valores proporcionados por la documentación de OpenCV [58]. Las ecuaciónes 3.2 muestra los umbrales aplicados.

$$
\begin{aligned}
h(i,j) &= \begin{cases} h(i,j), si & h_{max} \geq h(i,j) \geq h_{min} \\ 0, si & h(i,j) > h_{max} \\ 0, si & h(i,j) < h_{min} \end{cases} \\
s(i,j) &= \begin{cases} s(i,j), si & s_{max} \geq s(i,j) \geq s_{min} \\ 0, si & s(i,j) > s_{max} \\ 0, si & s(i,j) < s_{min} \end{cases} \\
v(i,j) &= \begin{cases} v(i,j), si & v_{max} \geq v(i,j) \geq v_{min} \\ 0, si & v(i,j) > v_{max} \\ 0, si & v(i,j) < v_{min} \end{cases}
\end{aligned}
\tag{3.1}
$$

Al aplicar los umbrales, se filtra todo lo que este en ese rango de valores, pero aún se encuentra en términos de matiz, saturación y valor, por lo que hay que hacer el proceso inverso de transformación de espacio de color a RGB.

Como salida final del sub-bloque de filtro RGB→HSV→RGB de color, se tiene una imagen RGB de w×h×3 como al inicio, pero con la diferencia que ahora solo se tienen las partes de interés. De esta imagen resultante, solo se ocupará una de las matrices, la G, que es el componente de color que se necesita, optimizando así la memoria.

Figura 3.14: Filtrado de los marcadores.

Si en la escena se llegaran a encontrar colores cercanos a los marcadores, el programa ajustara la ventana de los filtros en etapas posteriores para intentar eliminar aquellos pixeles no deseados. Si después del incremento no desaparecen los objetos no deseados, el programa envía valores por defecto al robot y despliega un mensaje alertando sobre una incorrecta detección de los marcadores.

Binarización

Las imágenes binarias son aquellas que solo tienen un bit de profundidad y 2 estados entre los cuales varían, 0 y 1. Este tipo de imágenes son en blanco y negro. Para convertir una imagen en una imagen binaria se debe tener previamente dicha imagen en escala de grises (8 bits), por lo que se toma solo una de las matrices, en este caso la matriz de color verde (RGB$(i, j, 1)$) y al tomar solo esta matriz, se le aplica un proceso de umbralización donde T es la el umbral como se muestra en la ecuación 3.2.

$$I(i,j) = \begin{cases} 0, si & RGB(i,j,1) > T \\ 1, si & RGB(i,j,1) < T \end{cases} \tag{3.2}$$

A partir de este proceso se obtiene una nueva imagen binaria de tamaño w×h×1 en blanco y negro como se muestra en la figura 3.15.

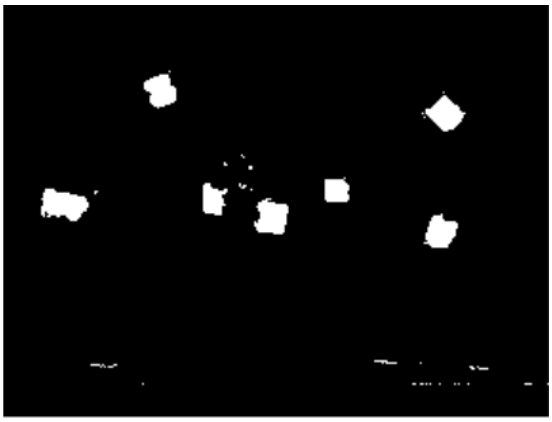


Figura 3.15: Imagen binaria.

Filtro mediana

El filtro mediana es un una técnica de filtrado digital no lineal el cual es ampliamente usado en la etapa de pre-procesamiento para obtener mejores resultados en las

siguientes etapas. Partiendo de que se tiene un arreglo ya sea en $1D$ o $2D$, el filtro mediana consiste en analizar entrada por entrada de la señal, en este caso pixel por pixel de la imagen, alrededor de estos valores se crea una ventana, regularmente de 3x3, con los vecinos próximos de cada pixel como en la figura 3.16, se realiza un ordenamiento de estos valores en un arreglo de $1D$ donde se colocan de menor a mayor y el valor de en medio es el que sustituye la entrada en la cual se formó una vecindad de pixeles.

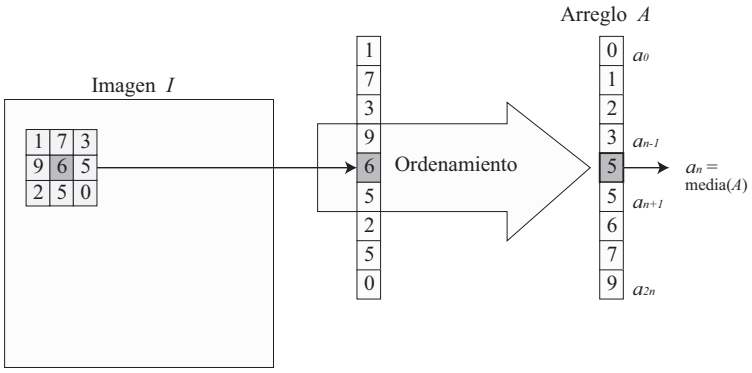


Figura 3.16: Demostración del calculo del filtro mediana de tamaño 3x3 ($n = 4$).

La mediana de un conjunto de 2_{n+1} valores $A = a_0, \ldots, a_{2n}$ puede ser definida como el valor central después del ordenamiento de a en una secuencia ordenada, es decir:

$$\text{mediana}\underbrace{(a_0, a_1, \ldots, a_{n-1}}_{\text{n valores}}, a_n, \underbrace{a_{n+1}, \ldots, a_{2n})}_{\text{n valores}} = a_n$$

donde $a_i \leq a_{i+1}$.

El algoritmo para el filtro mediana se muestra en la figura 3.18.

```
Función filtroMediana(i,j,kmedian)
    Si i ← (2)(rmedian) y j ← (2)(rmedian) Entonces
        u, v ← (i-(2)(rmedian))+rmedian-1, (j-(2)(rmedian))+rmedian-1 ;
        Si i <= (Fils-rmedian-2) y v <= (Cols-rmedian-2) Entonces
            Para i ← -rmedian Hasta rmedian+1 Con Paso 1 Hacer
                Para j ← -rmedian Hasta rmedian+1 Con Paso 1 Hacer
                    Amedian[kmedian] ← I[u+i, v+j];
                    Kmedian++;
                Fin Para
            Fin Para
            Ordenar(Amedian);
            I[u,v] ← Amedian[nmedian];
        Fin Si
    Fin Si
Fin Función
```

Figura 3.17: Función filtro mediana

En la figura 3.18 se puede ver una gran reducción de ruido con respecto a la imagen de la etapa previa después de aplicado el filtro. Cabe resaltar que los ejes originales de los marcadores se conservan, cosa que con otros filtros de espaciales no sucede.

Figura 3.18: Filtrado por medio del filtro mediana.

Erosión

Si bien la imagen binaria de la etapa anterior ha sido filtrada, pueden existir pequeñas islas de pixeles imperceptibles, por lo que hay que eliminarlos para que no perjudiquen

los resultados de las etapas siguientes.

Para eliminar esos pixeles de ruido sutiles restantes se utiliza una de las operaciones fundamentales de la morfológica matemática, la erosión. La erosión elimina los píxeles de los límites del objeto. La erosión reduce los contornos de los objetos, se utiliza para separar objetos que están unidos, donde los objetos se van separando unos de otros reduciéndose su grosor y diámetro.

Después de aplicar la operación morfológica de erosión, la imagen queda como se muestra en la figura 3.19, donde se han eliminado todos los pixeles aislados al igual que se reduce el área de la región de interés. Esta reducción del área principal es despreciable ya que la resolución de la imagen es de un tamaño suficientemente amplio, por lo tanto, asegura una buena densidad de pixeles por región para los bloques siguientes.

Figura 3.19: Filtrado por medio de la erosion.

Segmentación

La segmentación de una imagen digital es la etapa donde se divide en regiones la imagen, se les asigna una etiqueta a los pixeles para diferenciar los diversos objetos que hay en la imagen. al finalizar esta etapa, se tienen que conocer perfectamente los objetos que existen en la imagen para poder extraer las características de interés de cada uno de los objetos etiquetados.

Etiquetado de componentes

Llegado a este punto, en la imagen binaria solo se encuentran regiones que representan las articulaciones de los brazos, de las cuales se necesita extraer información útil. Para poder extraer las características de cada región de la imagen, es necesario identificar

las componentes conexas de cada región, en este caso tenemos 7 regiones que debemos identificar y reconocer por separado. Al proceso de asignar un valor numérico especifico a pixeles de cada región de la imagen se le conoce como etiquetado.

Para realizar el etiquetado de las regiones de interés de la imagen, se utilizará una técnica recursiva de etiquetado. La conectividad de los pixeles es la relación con la que los pixeles de una imagen se relacionan con sus vecinos, se pueden utilizar vecindades de 4 u 8 pixeles, por ejemplo, pero se optó por la vecindad de 8.

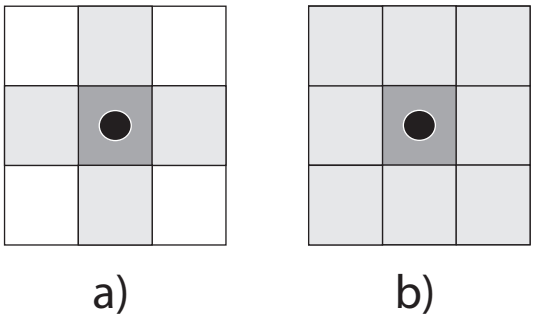


Figura 3.20: a)Vecindad de 4 pixeles. b)Vecindad de 8 pixeles.

Para este algoritmo, se asume que las regiones de interés en la imagen tienen el valor 1 (blanco) y el resto del fondo 0 (negro). El proceso iterativo es recursivo y se resume en 4 pasos:

- Se recorre la matriz de imagen hasta encontrar un 1, a este pixel se le asigna una etiqueta “L”
- Recursivamente le asigna esta etiqueta “L” a todos los componentes conexos.
- Se detiene si no hay más pixeles de valor 1 que sean conexos.
- Regresa al primer paso.

El resultado de la conexión de componentes se muestra en la figura 3.21 donde se puede apreciar que el primer componente encontrado se le asigna el valor 1 y el ultimo se le asigna el valor N, que en este caso es el número 7.

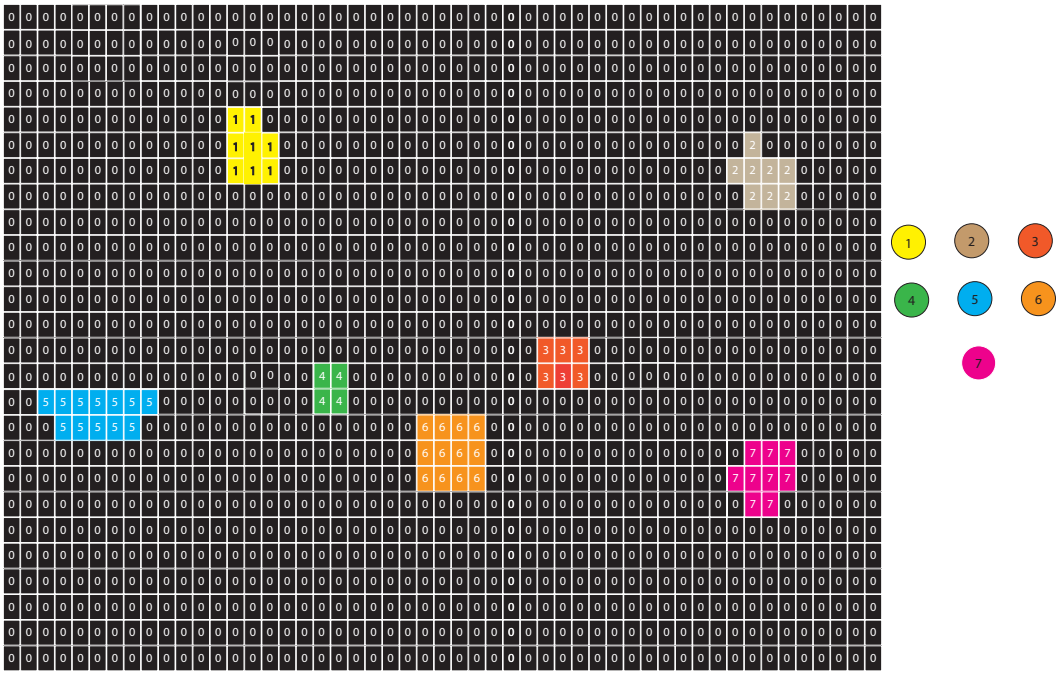


Figura 3.21: Etiquetado de componentes en la imagen.

Centroides cálculo de ángulos

A partir de los centros geométricos se obtienen los ángulos, ya que estos representan coordenadas espaciales del plano bidimensional x, y. No es necesario unir los centroides ya que es suficiente con las coordenadas para obtener la información angular.

La obtención de los ángulos se realiza por método geométrico, con la ley de senos y cosenos. Se utiliza esta ley ya que en las posiciones de los brazos se pueden trazar líneas imaginarias que forman triángulos.

La figura 3.22 muestra el semi-esqueleto con los triángulos que se forman uniendo los centroides encontrados.

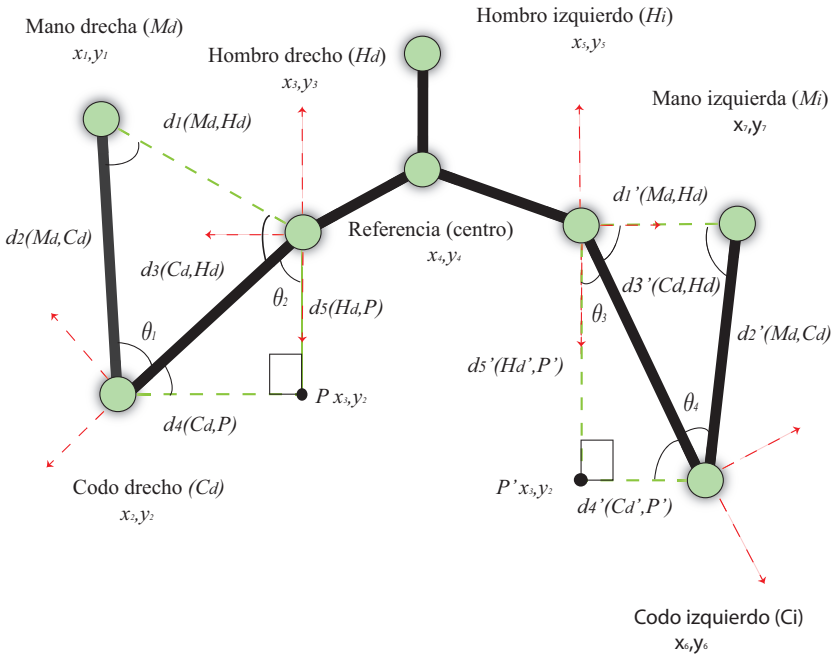


Figura 3.22: Geometría del esqueleto conformada por los centroides.

Las expresiones 3.3-3.9 son las que se programan para el cálculo de los ángulos de los miembros superiores del lado derecho.

$$d_1(M_d,H_d) = \sqrt{(x_{M_d} - x_{H_d})^2 + (y_{M_d} - y_{H_d})^2} \tag{3.3}$$

$$d_2(M_d,C_d) = \sqrt{(x_{M_d} - x_{C_d})^2 + (y_{M_d} - y_{C_d})^2} \tag{3.4}$$

$$d_3(C_d,H_d) = \sqrt{(x_{C_d} - x_{H_d})^2 + (y_{C_d} - y_{H_d})^2} \tag{3.5}$$

$$d_4(C_d,P) = \sqrt{(x_{C_d} - x_P)^2 + (y_{C_d} - y_P)^2} \tag{3.6}$$

$$d_5(H_d,P) = \sqrt{(x_{H_d} - x_P)^2 + (y_{H_d} - y_P)^2} \tag{3.7}$$

$$\theta_1 = \arccos\left(\frac{d_1 - d_2^2 - d_3^2}{-2(d_2)(d_3)}\right) \tag{3.8}$$

$$\theta_2 = \arccos\left(\frac{d_4 - d_3^2 - d_5^2}{-2(d_3)(d_5)}\right) \tag{3.9}$$

A partir de estas expresiones se pueden deducir las expresiones de los miembros superiores izquierdos.

Los ángulos se obtienen en términos de grados, para después, por medio de una conversión, pasar los valores a un rango manejado por el robot y por la simulación.

3.4.5. Comunicación por MQTT

Los procesos automatizados requieren mínima intervención humana, en donde está presente la interoperabilidad entre diferentes dispositivos, enviándose y recibiendo datos continuamente de sensores, procesos, etc.

En el caso de este proyecto, uno de los objetivos es la tele operación del robot, donde se usara protocolo M2M y de internet de las cosas MQTT.

El elemento central en el protocolo MQTT es el servidor-*broker*, que es el encargado de gestionar la información, se encarga de recibir como direccionar a los diferentes clientes que estén suscritos. En este caso el cliente que publica los datos de los ángulos es el sistema embebido. Un solo *broker* permite la gestión de diferentes clientes a través de los tópicos por lo que se establecerá una estructura jerárquica por niveles para la publicación-suscripción, y así poder dirigir el mensaje adecuadamente. La figura 3.23. muestra los niveles de tópicos con los que cuenta el proyecto.

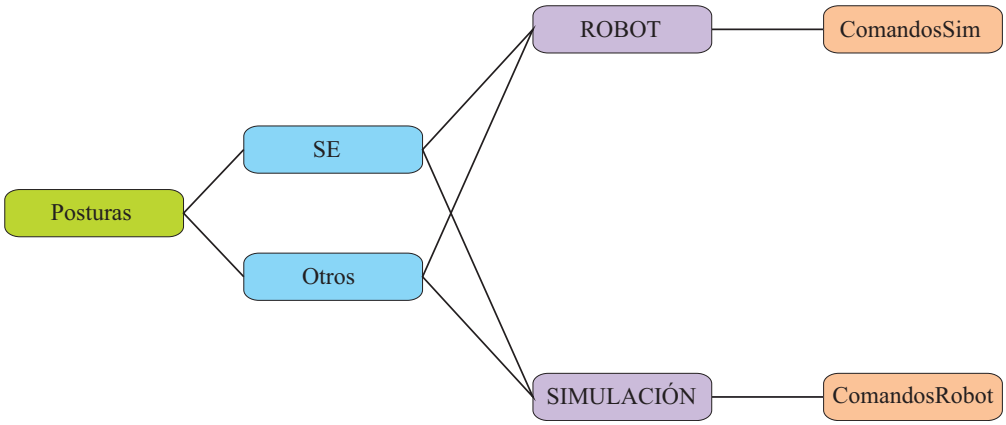


Figura 3.23: Estructura de los tópicos para publicar y suscribirse, donde se establece como usuario en el primer nivel a “Posturas”.

La estructura de los tópicos comienza por lo que se definió como usuario que en este caso es el primer nivel del tópico “Posturas”. El usuario posturas es el primer nivel al que se referirán los demás tópicos para este proyecto. En el segundo nivel se encuentran los tópicos sistema embebido “SE” y otras entradas “Otros”, donde el sistema embebido conformado por la tarjeta PYNQ-Z1, es el elemento de procesamiento principal que se encarga de publicar la información obtenida en el procesamiento y permite realizar pruebas de envío de datos o publicar desde algún otro elemento, para ello, el usuario se tendría que suscribir al tópico “Otros”.

3.5. Implementación

Con la estrategia de procesamiento, de extracción de información y de comunicaciones para la conexión del sistema embebido con el robot y la computadora central diseñadas, se procede a la implementación e integración del sistema completo. En esta sección se presentará la implementación de los algoritmos en el sistema embebido, los canales de comunicación, el servidor en la computadora central, el entorno de simulación física y los *drivers* de control para el robot en su entorno embebido de Linux.

3.5.1. Sistema Embebido

Los algoritmos de procesamiento y el protocolo MQTT serán implementados completamente en el sistema embebido, que contiene un circuito integrado Zynq que es un APSoC, lo cual nos permite integrar toda la parte programable, reconfigurable y de comunicaciones en un solo cihp.

En la tarjeta PYNQ-Z1 corre un sistema operativo de Linux embebido (Petalinux) en el procesador del dispositivo Zynq, donde a su vez corre el *Framework* PYNQ.

PYNQ permite la productividad de programación en un alto nivel de abstracción con Python, donde se pueden importar una gran cantidad de bibliotecas conocidas de Python, importar bibliotecas aceleradoras conocidas en este contexto como *Overlays*, que no son más que diseños realizados en la lógica programable (IP) exportados como bibliotecas de Python. PYNQ también utiliza como tecnología clave Jupyter Notebook, que es un GUI de cuadernillos que permite una programación interactiva. A continuación se mencionarán las bibliotecas, módulos y *Overlays* utilizados y en los que se basó el proyecto para su implementación.

La figura 3.24 muestra el *Overlay* base que permite que el *hardware* de los periféricos del dispositivo Zynq puedan importarse por *software* a través de Python, tales como los puertos HDMI de entrada y salida, entradas y salidas para el usuario, el micrófono, etc. Estos bloques IP tienen una interfaz con la parte de procesamiento del dispositivo Zynq.

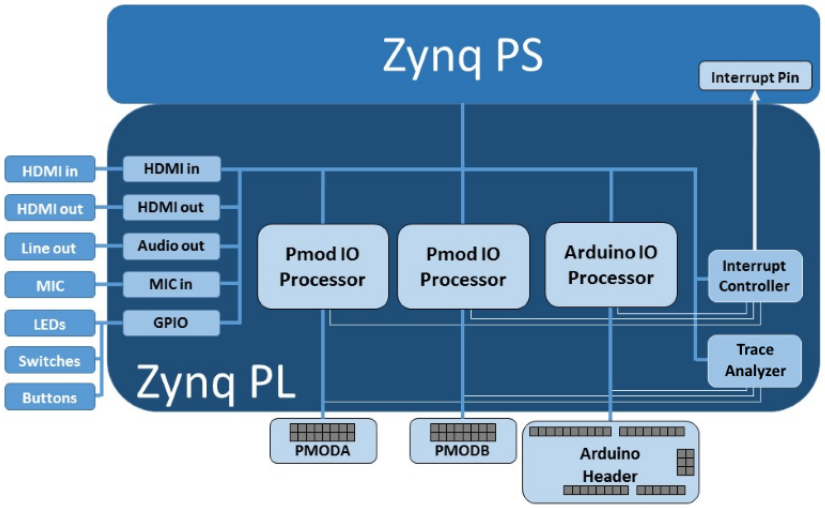


Figura 3.24: *Overlay* base de la tarjeta PYNQ-Z1.

El marco de PYNQ contiene aceleradores para visión computacional preinstalados basados en OpenCV. Estos aceleradores usan la arquitectura mostrada en la figura 3.25 donde usan la memoria compartida a través de movedores de datos como el IP AXI de acceso directo a la memoria (DMA por sus siglas en inglés) para transferir datos entre la parte PS y la parte PL del dispositivo Zynq, otros están completamente implementados en la parte de *software*.

Para la implementación de la estrategia de procesamiento en la tarjeta PYNQ-Z1 se combinaron los algoritmos desarrollados mostrados en la sección 3.4.4 en la parte PS y también haciendo uso de los módulos aceleradores preinstalados. La parte de binarización de la imagen fue completamente integrada en la parte *PL* de la tarjeta, haciendo uso de Vivado HLS siguiendo la arquitectura mostrada en la figura 3.25.

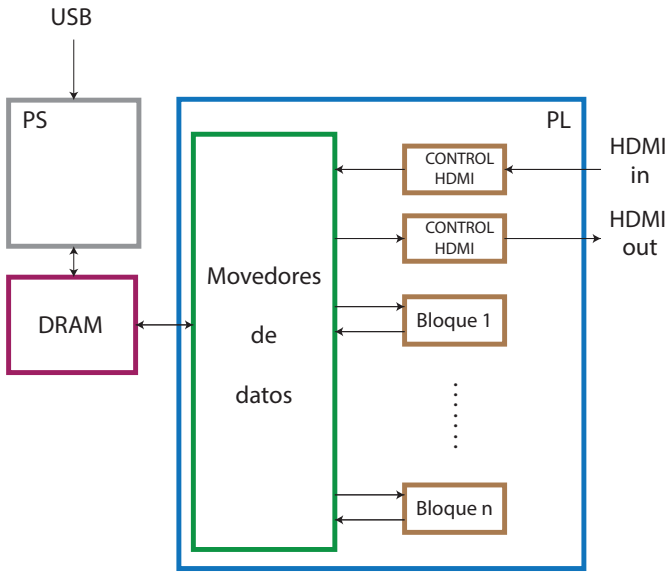


Figura 3.25: Diagrama de bloques de los *Overlays* de aceleración de visión computacional.

La figura 3.26 muestra el *Overlay* de redes que permite capacidades de redes en la tarjeta. En este *Overlay*, MQTT y MQTT-SN se implementa aprovechando bibliotecas de Python.

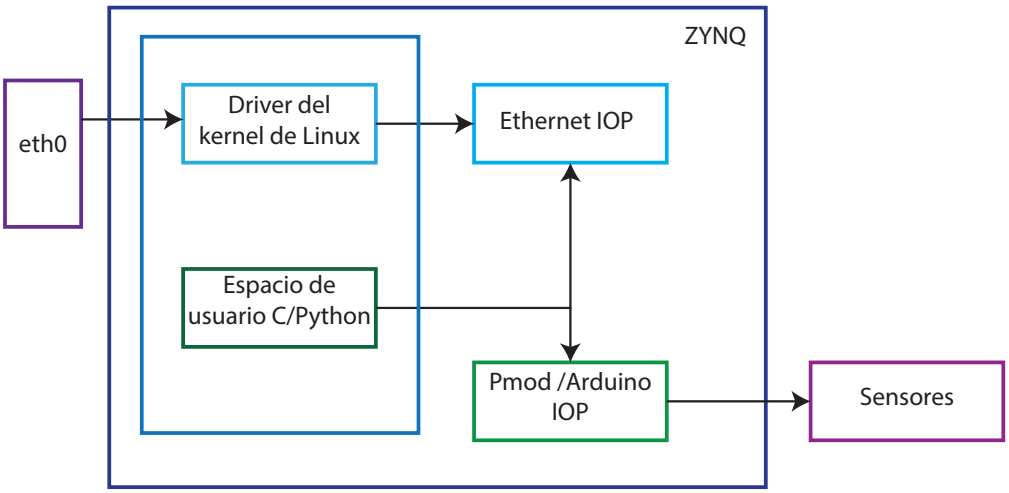


Figura 3.26: Diagrama de bloques del *Overlay* de redes.

3.5.2. Almacenamiento de ángulos en el servidor

Después de crear el *broker* que será el encargado de gestionar el envío de mensajes, se implementó el servidor Apache a través del *Uniform Server* una solución de servidor WAMP ligera y gratuita que permite ejecutar un servidor web en cualquier computadora basada en el sistema operativo Microsoft Windows. La figura3.27. muestra la base de datos que se implementó, la cual cuenta con 5 columnas pensando no solo en almacenar los datos que el robot interpretará como movimientos, sino también los niveles de tópico del esquema publicación-suscripción del bróker. Cabe mencionar que la computadora es un cliente que solo se suscribe al tópico de simulación, por lo que solo la simulación tendrá acceso a los datos almacenados en la base de datos.

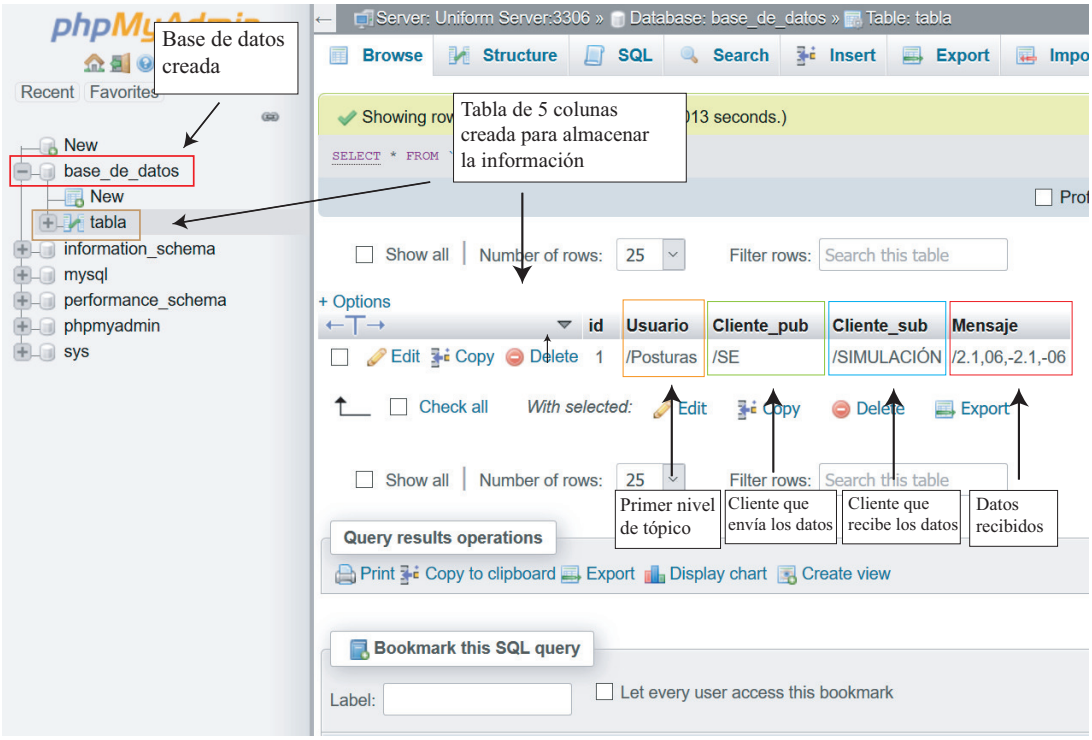


Figura 3.27: Base de datos implementada.

La computadora central es el cliente 2 la cual se suscribe al tópico “Simulación” para poder recibir los comandos de movimiento del robot para la simulación. Los mensajes del *broker* se reciben por medio de un *script* implementado en Python a través de la biblioteca Paho que proporciona implementaciones de cliente de código abierto de los protocolos de mensajería MQTT y MQTT-SN. Una vez recibidos los mensajes de *broker*, el script coloca los mensajes y los tópicos en la base de datos para que puedan

ser tomados por la simulación. La figura 3.28 muestra el proceso de almacenamiento de los mensajes en la base de datos

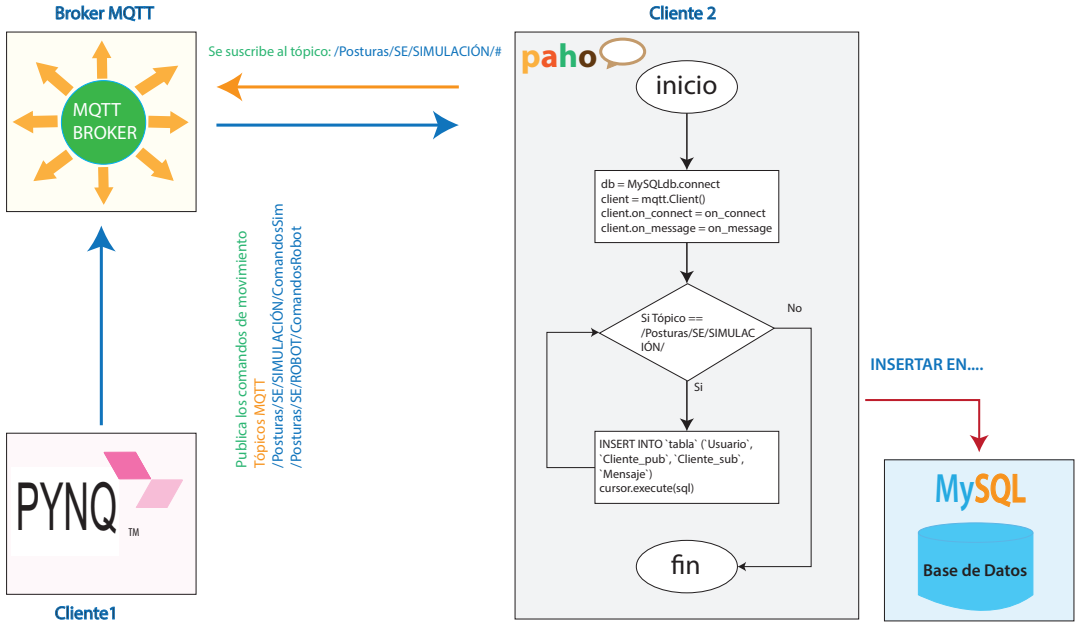


Figura 3.28: Resepción y almacenamiento de los mensajes MQTT en la base de datos.

3.5.3. Simulación

El entorno de simulación de Webots el usuario puede programar cada robot individualmente para que muestre el comportamiento deseado. Contiene una serie de interfaces para robots móviles reales, de modo que el robot simulado se comporta como se espera. Una simulación de Webots se compone de los siguientes elementos:

- Un archivo world de Webots (.wbt) que define uno o varios robots y su entorno. El archivo .wbt a veces depende de archivos externos de PROTO (.proto) y texturas.
- Uno o varios programas controladores para los robots (en C/C++/Java/Python/MATLAB).
- Un *plugin* de física opcional que puede ser usado para modificar el comportamiento de la física regular de Webots (en C/C++)

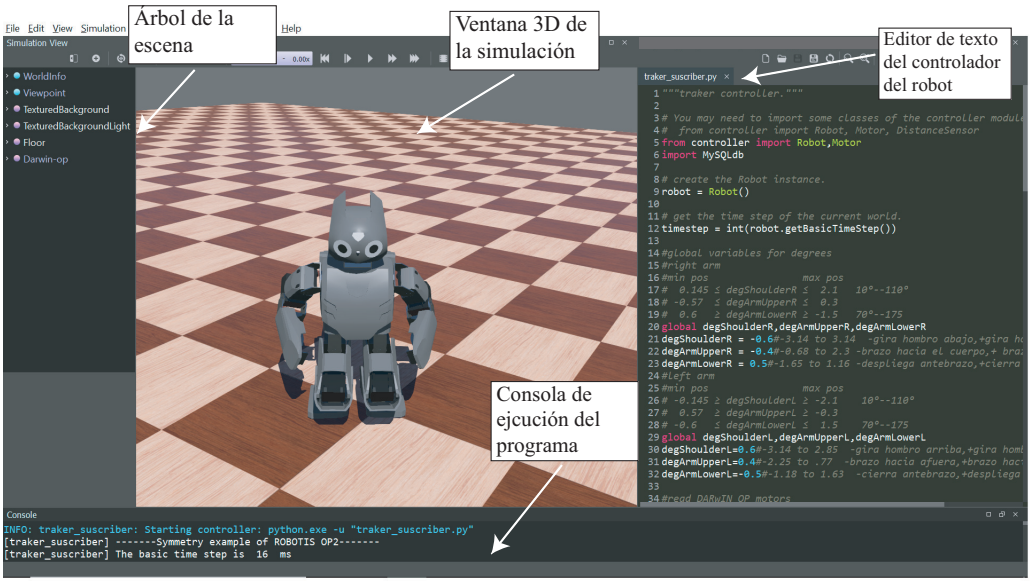


Figura 3.29: Interfaz gráfica del archivo de simulación de Webots.

En la figura 3.29 se muestran los elementos de la simulación. El árbol de la escena contiene la información que describe un archivo *world* simulado, incluyendo los robots y el medio ambiente, y su representación gráfica. En la ventana 3D de la simulación se interactúa con la simulación, que, en este caso, el controlador de la simulación (un *script* escrito en Python) toma los comandos de movimiento almacenados en la base de datos y con ello describirá el comportamiento del robot. La consola se encargará de mostrar errores o elementos de ejecución del programa.

La figura 3.30 muestra el proceso de toma de los comandos de movimientos de la base de datos por la simulación. El *script* inicializa las funciones de los motores, colocándole valores iniciales, posteriormente entra en un bucle donde constantemente recibirá los valores que se van actualizando en la base de datos.

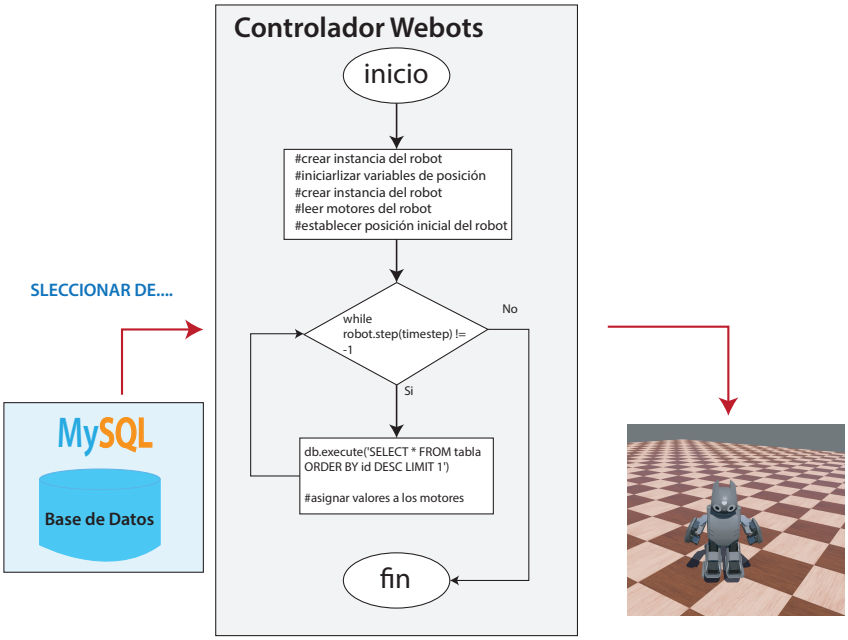


Figura 3.30: Proceso de selección de los datos para el control de la simulación.

Cada una de las extremidades superiores del robot se dividen en 3 partes en la simulación, degShoulderR/degShoulderL (hombro derecho/izquierdo), degArmUpperR/ degArmUpperL (brazo superior derecho/izquierdo) y degArmLowerR/ degArmLowerL (brazo inferior derecho/izquierdo). Estas funciones son las que reciben los datos obtenidos en el procesamiento, donde previamente deben someterse a una conversión ya que estas funciones no aceptan valores en términos de ángulos. La documentación muestra los valores máximos y mínimos que las funciones aceptan, sin embargo, estos valores representan giros máximos y mínimos de los motores donde pueden ser vueltas de más de 180° en las extremidades, por lo tanto, se establecieron rangos máximos y mínimos experimentales que correspondieran a los realizados por el operario tomando en cuenta las limitaciones en los grados de libertad del robot.

La tabla 3.4 muestra los valores acotados para el robot en la simulación, donde estos valores representa las posiciones máximas y mínimas de apertura de los miembros superiores.

Tabla 3.4: Posiciones máximas y mínimas de los miembros superiores del robot.

Brazo izquierdo			Brazo derecho		
Posición mínima	Extremidad	Posición máxima	Posición mínima	Extremidad	Posición máxima
0.145	$\leq \ degShoulderR \leq$	2.1	-0.145	$\geq degShoulderL \ \geq$	2.1
-0.57	$\leq degArmUpperR \leq$	0.3	0.57	$\geq degArmUpperL \geq$	-0.3
0.6	$\geq degArmLowerR \geq$	-1.5	-0.6	$\leq degArmLowerL \leq$	1.5

Para la conversión se asumió una correspondencia lineal entre el crecimiento de los valores del robot con respecto al crecimiento de los valores angulares del operario, por lo tanto, para realizar la conversión se utilizó de la ecuación de la recta 3.10 donde y_2 y x_2 representan los valores de posición máxima y y_1 y x_1 los valores de posición mínima.

$$y - y_1 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1) \tag{3.10}$$

Las expresiones obtenidas a partir de la ecuación 3.10 son:

$$degShoulderR = 0.01955(anguloH_d) - 0.0505 \tag{3.11}$$

$$degArmLowerR = 2 - 0.02(anguloC_d) \tag{3.12}$$

$$degShoulderL = 0.0505 - 0.01955(anguloH_i) \tag{3.13}$$

$$degArmLowerL = 0.02(anguloC_i) - 2 \tag{3.14}$$

Estas expresiones controlaran el movimiento del hombro derecho e izquierdo y del brazo inferior derecho e izquierdo. Los brazos superiores representan la apertura del brazo hacia el cuerpo o hacia afuera y, como se mencionó anteriormente, este caso de estudio se delimito a estudiar el plano bidimensional frontal, por lo tanto, no hay un ángulo de los brazos del operario que controle directamente esta extremidad. Se decidió controlar la apertura y cierre de los brazos superiores a través de la apertura y cierre de los hombros, de igual forma asumiendo una correspondencia lineal. Las expresiones para controlar estas extremidades son las siguientes:

$$degArmUpperR = 0.445013(degShoulderR) - 0.634527 \tag{3.15}$$

$$degArmUpperL = 0.445013(degShoulderL) + 0.634527 \tag{3.16}$$

3.5.4. Robot

En esta sección se presentara la comunicación realizada con el robot DARwIN-OP, las configuraciones y *drivers* realizados para poder obtener como resultado final los movimientos imitativos del humanoide.

Circuito electrónico de comunicación IoT

En trabajos como [12][59][16] se utiliza *software* de terceros o protocolos de red para transferir los comandos de movimiento. Para poder realizar la comunicación por MQTT, se optó por realizar un circuito externo que transmitiera vía USB la información recibida basándose en[15]. El circuito está conformado por un microcontrolador atmega32u que es un microcontrolador AVR de 32 bits, un regulador de voltaje, un convertidor de nivel y un ESP8266 que es un módulo WiFi. La figura 3.31 muestra el diagrama de conexión.

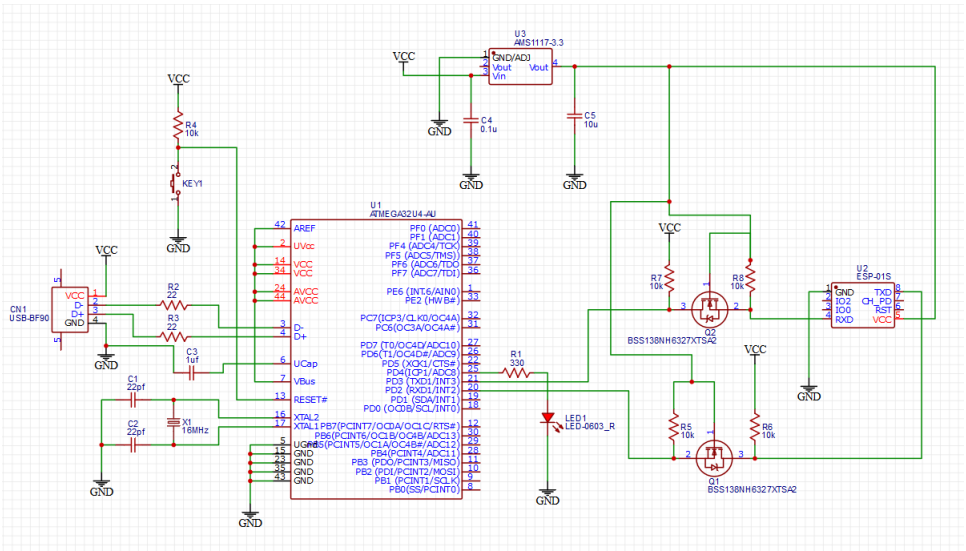


Figura 3.31: Diagrama de conexión del circuito IoT.

La tarjeta PYNQ-Z1, se conecta por medio del puerto Ethernet a la red. Posteriormente se inicializa el servidor-*broker* MQTT de la tarjeta para publicar los tópicos y los mensajes para que los usuarios se suscriban. El circuito IoT, a través del módulo WiFi ESP8266, se conecta a la red de forma inalambrica para suscribirse a los tópicos publicados por la tarjeta a través del servidor-*broker* . Este módulo manda serialmente el mensaje recibido al convertidor de nivel ya que el microcontrolador y el modulo manejan diferentes niveles de voltaje, una vez convertidos los niveles de voltaje, el

mensaje pasa al microcontrolador, este transforma el mensaje en caracteres ASCCI y los transmite por el protocolo USB al robot.

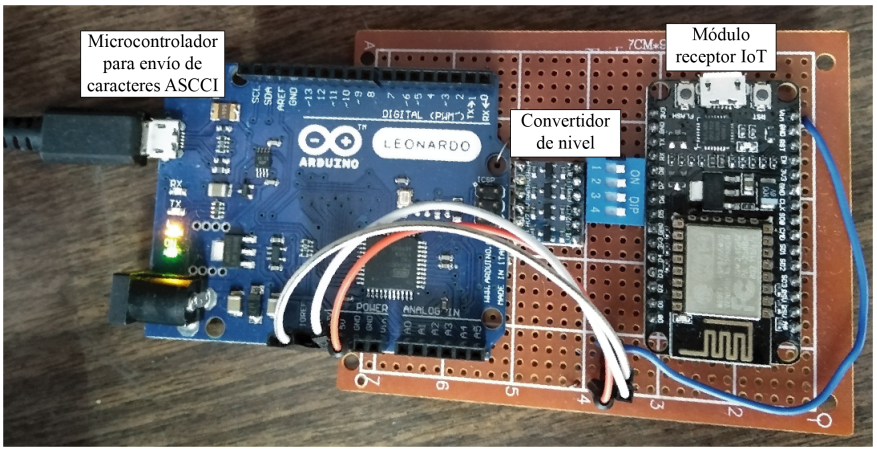


Figura 3.32: Dispositivo IoT.

Envío de comandos al Robot

El robot cuenta con un sistema operativo basado en el kernel de Linux con código fuente preinstalado. En este código fuente permite un *Framework* donde existen clases de interfaz para el control de la tarjeta controladora interna del robot. Cada plataforma obtiene sus clases correspondientes.

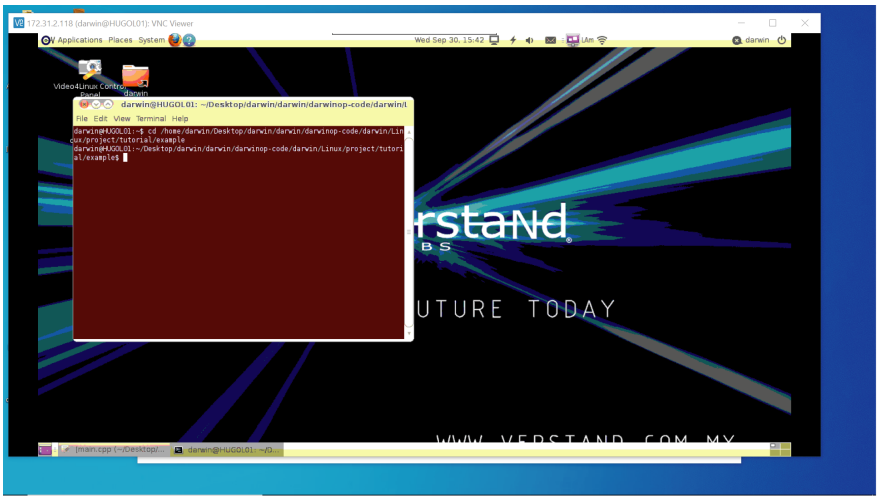


Figura 3.33: Entorno de desarrollo del Robot.

Figura 3.34: Integración del robot con el dispositivo IoT.

Para que el robot recibiera los comandos de movimiento, se implementó un *driver* de control de los movimientos basado en los ejemplos de la documentación oficial [2] donde se agregaron las instrucciones de recepción de los caracteres ASCCI.

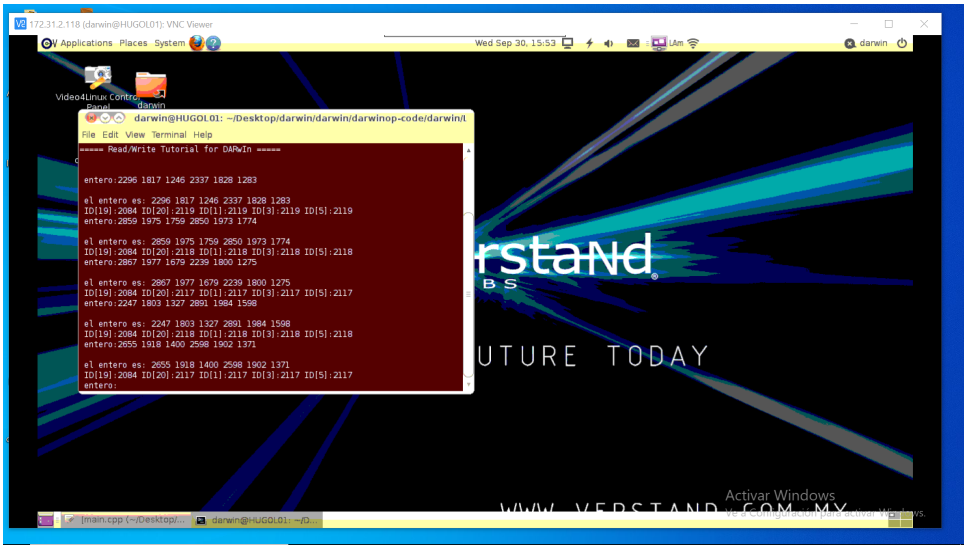


Figura 3.35: Ejecución del *driver* de recepción de caracteres ASCCI.

Capítulo 4

Pruebas y resultados

En esta sección se muestran las pruebas que se realizan del sistema completo como se puede observar en las figuras 3.2, 3.3 y 4.1 asi como los resultados obtenidos

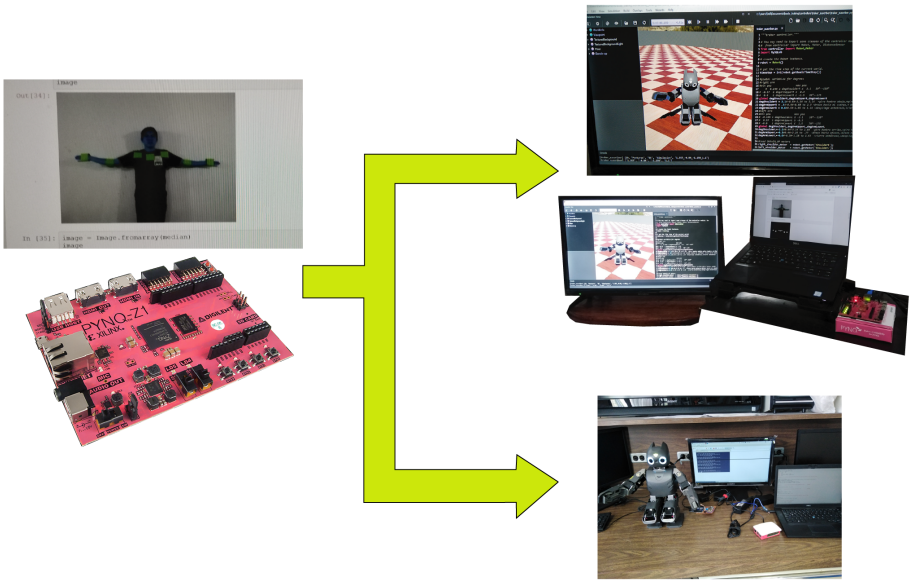


Figura 4.1: Integración del sistema.

Estas pruebas consisten en tomar 5 fotografías, que corresponden a todos los casos que se presentan en el análisis geométrico en un fondo de color uniforme blanco. La figura 4.2 muestra cómo se pueden agrupar los movimientos a realizar por el operario en 5 grupos de familias de movimientos, brazo-D arriba/brazo-I abajo, brazo-D abajo/brazo-I arriba, ambos brazos abajo, ambos brazos arriba, y por ultimo ambos brazos en medio.

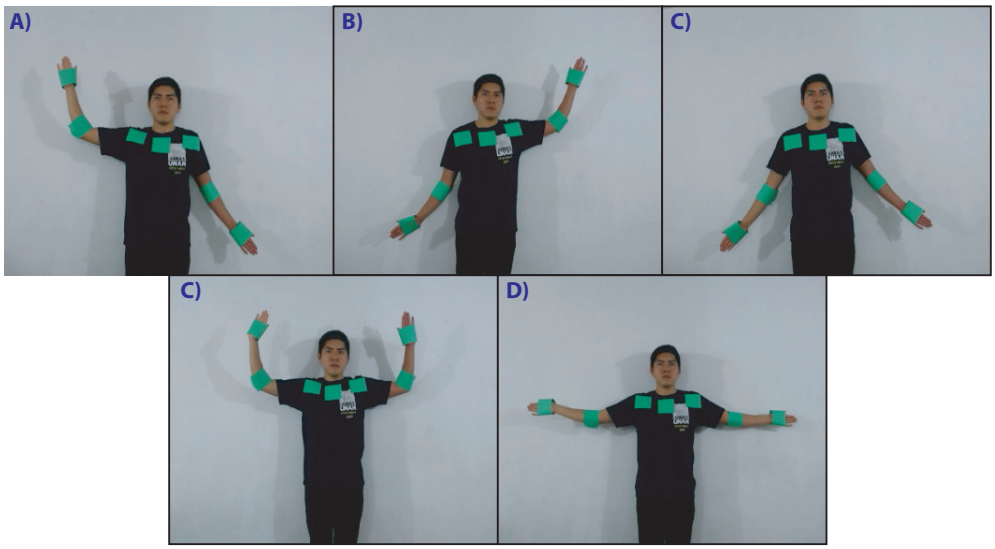


Figura 4.2: Posturas realizadas por el operario.

De las posturas realizadas por el operario de la figura 4.2, se obtienen los centros geométricos a partir de los cuales, por medio de la geometría del cuerpo, se obtienen los ángulos en términos de grados. La figura 4.3 muestra la imagen de salida con los centros geométricos obtenidos, las líneas rojas son líneas imaginarias que conforman el esqueleto. Las líneas punteadas grises muestran los triángulos por medio de los cuales se obtienen los ángulos de los hombros y los codos.

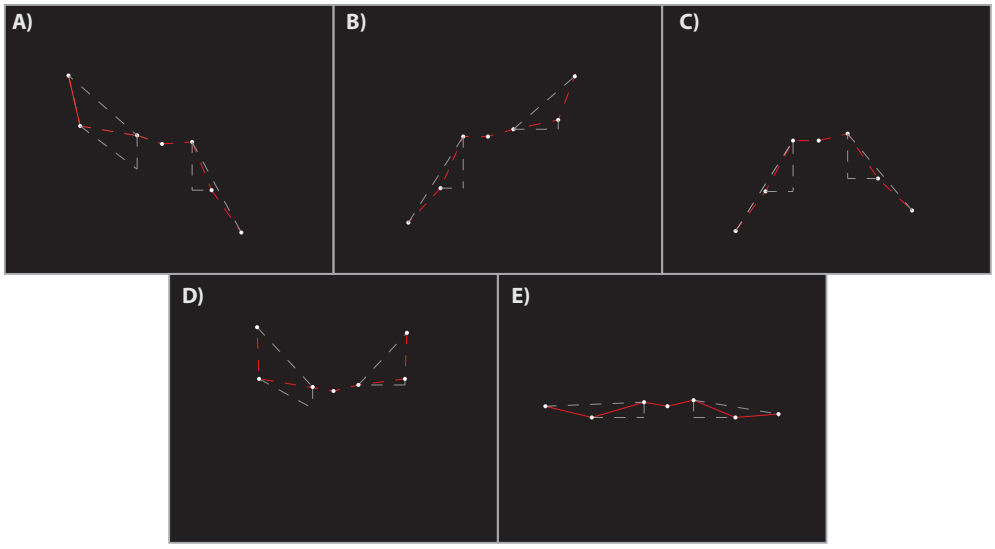


Figura 4.3: Centros geométricos de las posturas.

La imagen 4.4 muestra el resultado de las posturas realizadas por el operario reflejadas en la simulación y la imagen 4.5 las posturas realizadas por el robot.

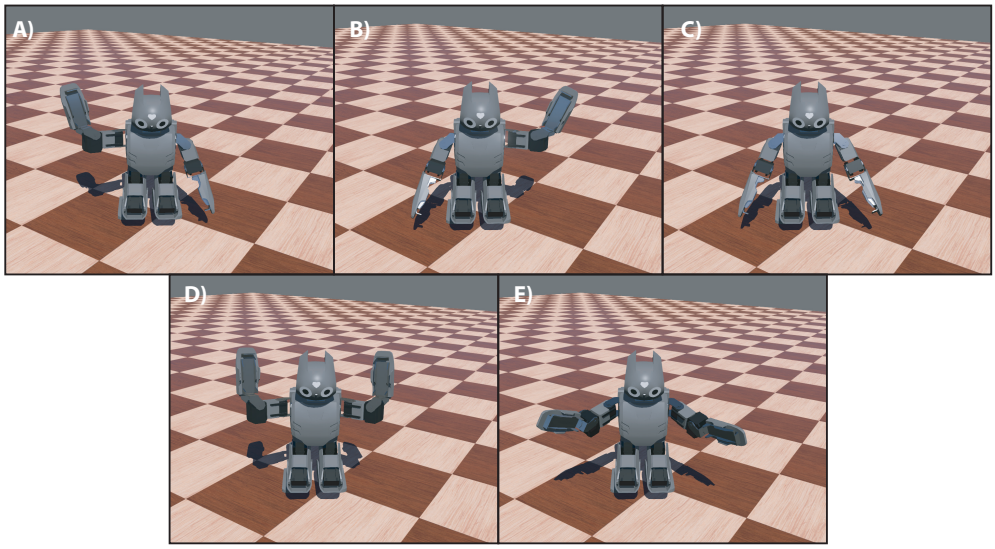


Figura 4.4: Posturas realizadas en la simulación

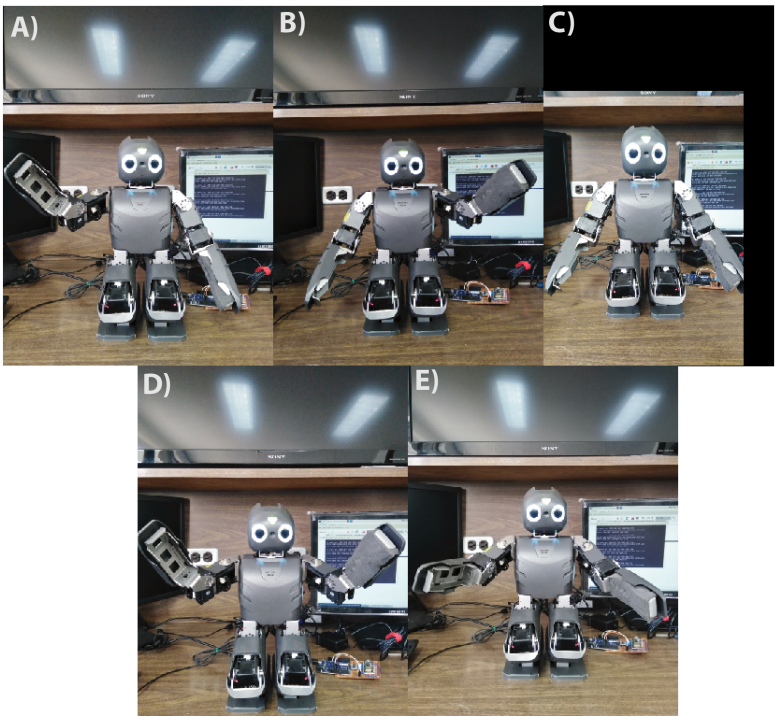


Figura 4.5: Posturas realizadas por el robot.

De los cinco casos presentados en las imágenes anteriores y de las familias derivadas de los mismos, se obtuvieron favorablemente los ángulos de los codos y de los hombros por medio del procesamiento. De igual manera, por medio de un *software* llamado ERGONOMICS RULER [60] que permite obtener ángulos en fotografías, se sometieron las fotografías de la figura 4.2. a medición para comparar los resultados con los arrojados por el sistema embebido.

Los datos obtenidos se muestran en la tabla 4.1. Esta tabla muestra los ángulos obtenidos para codo y hombro derecho, así como para codo y hombro izquierdo. Del *software* se obtuvieron los ángulos de la imagen original.

Tabla 4.1: Comparación de ángulos obtenidos mediante el Sistema Embebido y el Software

	Medición en SE					Medición Ángulos de fotografías en Software				
	A	B	C	D	E	A	B	C	D	E
Hombro Derecho	99.30°	23.19°	28.61°	98.13°	73.11°	97°	19°	30°	97°	68°
Hombro Izquierdo	21.99°	101.16°	33.69°	96.70°	66.03°	20°	99°	30°	85°	61°
Codo Derecho	111.83°	159.83°	171.30°	101.25°	149.61°	114°	161°	169°	106°	144°
Codo izquierdo	167.42°	122.72°	166.27°	99.04°	153.54°	164	123°	168°	97°	150°

Una vez realizadas las mediciones se procede a cuantificar la variación en los datos obtenidos en el sistema embebido y en el *software*. La tabla 4.2 muestra el promedio de los datos, la desviación estandar y el coeficiente de variación. Se compararon los 20 valores obtenidos en la tarjeta PYNQ contra los 20 obtenidos por el *software*.

Tabla 4.2: Resultados conseguidos en la obtención de los ángulos.

		Hombro Derecho	Hombro Izquierdo	Codo Derecho	Codo Izquierdo
Promedio de los datos	A	98.15°	20.99°	112.91°	165.71°
	B	21.09°	100.08°	160.41°	122.86°
	C	29.30°	31.84°	170.15°	167.13°
	D	97.56°	90.85°	103.62°	98.02°
	E	70.55°	63.51°	146.80°	151.77°
Desviación Estándar σ	A	1.15°	0.99°	1.08°	1.71°
	B	2.09°	1.08°	0.58°	0.14°
	C	0.69°	1.84°	1.15°	0.86°
	D	0.56°	5.85°	2.37°	1.02°
	E	2.55°	2.51°	2.80°	1.77 °
Coeficiente de Variación	A	1.17 %	4.73 %	0.96 %	1.03 %
	B	9.93 %	1.07 %	0.36 %	0.11 %
	C	2.37 %	5.79 %	0.67 %	0.51 %
	D	0.57 %	6.43 %	2.29 %	1.04 %
	E	3.62 %	3.95 %	1.91 %	1.16 %

Como se mencionó anteriormente, no se transmiten los datos en términos de ángulos, se realiza una conversión a términos de datos que la simulación y el robot entiendan. La tabla 4.3 muestra los valores obtenidos de la conversión para la simulación y el robot.

Tabla 4.3: Tabla de conversiones.

		degShoulderR	degShoulderL	degArmUpperR	degArmUpperL	degArmLowerR	degArmLowerL
Simulación	A	1.88	-0.38	0.25	0.63	-0.24	1.34
	B	0.39	-1.94	-0.63	-0.25	-1.2	0.46
	C	0.51	-0.61	-0.19	0.19	-1.42	1.32
	D	1.86	-1.84	0.25	-0.25	-0.02	-0.02
	E	1.37	-1.24	-0.19	0.19	-1.0	1.08
Robot DARwIN-OP	A	2867	2239	1977	1800	1679	1275
	B	2247	2891	1803	1984	1327	1598
	C	2296	2337	1817	1828	1246	1283
	D	2859	2850	1975	1973	1759	1774
	E	2655	2598	1918	1902	1400	1371

Se midió el tiempo de procesamiento con dos imágenes de diferente resolución, la primera de 384x288 y la segunda de 640x480. Para medir el tiempo promedio del procesamiento se realizaron 100 ciclos de repetición para cada una de las imágenes.

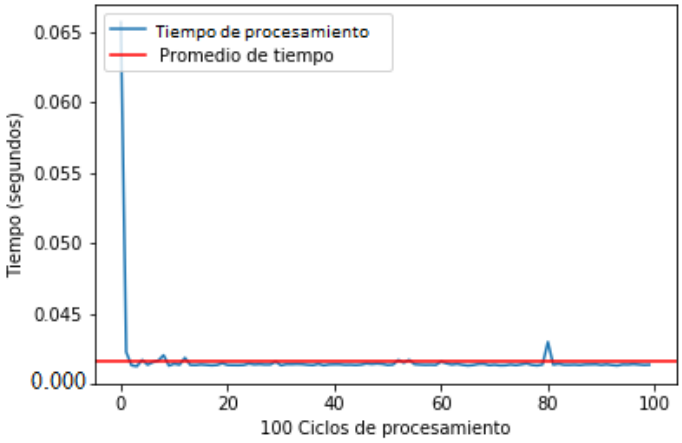


Figura 4.6: Tiempo de procesamiento en la imagen de 384x288

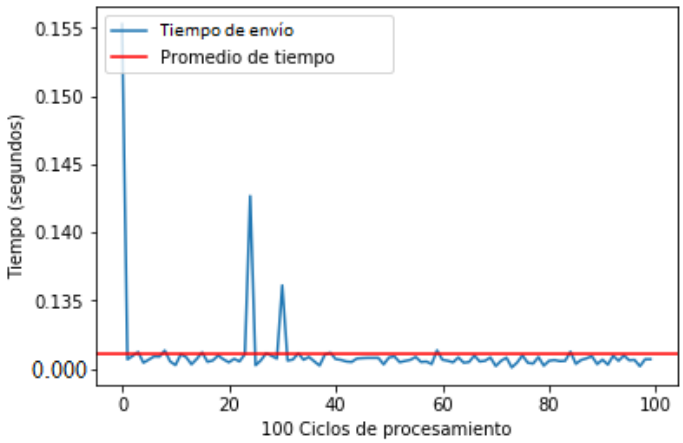


Figura 4.7: Tiempo de procesamiento en la imagen de 640x480

Para la imagen de 384x288 el tiempo promedio por ciclo fue de 0.04167 segundos mientras que para la imagen de 640x480 fue de 0.1310 segundos. Para imágenes de menor tamaño, como la primera imagen, el tiempo de procesamiento es un tiempo aceptable ya que se pueden procesar ≈ 24 imágenes por segundo o en términos de video, se pueden procesar a una tasa de 24 fotogramas por segundo (FPS por sus siglas en inglés) lo cual es un tiempo estándar de tasa de transmisión en cine y animación [61]. Para imágenes de mayor tamaño, como la segunda imagen, la tasa de procesamiento es de ≈ 8 imágenes por segundo, lo cual es un tiempo no tan favorable si se desea replicar los movimientos en tiempo real.

El tamaño de la imagen representa un evidente impacto de procesamiento y, sin embargo, a pesar del incremento en el tiempo, el procesamiento no rebasa los 0.2 segundos sin que todos los módulos de procesamiento sean ejecutados de forma concurrente. Se puede observar también muy pocas variaciones en la gráfica, donde los picos más significativos se encuentran en los primeros ciclos. Estas variaciones se deben a diferentes factores como el sistema operativo embebido de la tarjeta y como administra sus procesos, y también a que la ejecución de los programas se hace en cuadernillos interactivos de Jupyter Notebook que corren en el navegador. En estos cuadernillos los programas los maneja un Kernel que son procesos separados iniciados por la aplicación web del cuadernillo, que ejecuta el código de los usuarios en Python y devuelve el resultado a la aplicación web del cuadernillo.

Al igual que con las imágenes, se midió el tiempo de transmisión de paquetes de datos de los comandos de movimiento a través del *broker*. Las figuras 4.8 y 4.7 muestran la tasa de transmisión en 100 ciclos de repetición después del procesamifento de las imágenes.

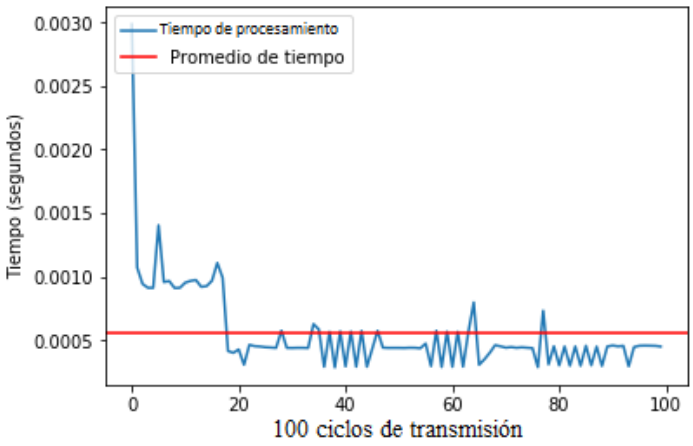


Figura 4.8: Tiempo de transmisión de datos en el procesamiento de la imagen de 384x288

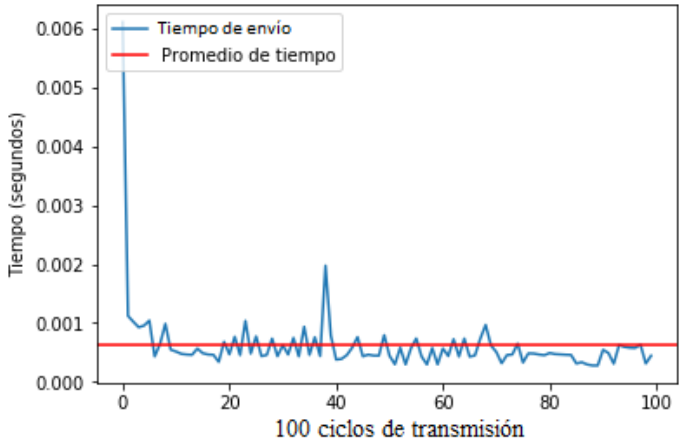


Figura 4.9: Tiempo de transmisión de datos en el procesamiento de la imagen de 640x480

El promedio de transmisión en el procesamiento de la imagen de 384x288 fue de 0.0005565 segundos mientras que para la imagen de 640x480 fue de .0006178 segundos. Para la primera imagen, en términos de tasa de transmisión, el promedio de transmisión de datos es de 1,796.9451 paquetes por segundo y para la segunda imagen fue 1,618.6468 paquetes por segundo. Esto resulta favorable en el uso del protocolo MQTT en cuanto a velocidad de transporte de mensajes de publicación/suscripción, ya que no se ve realmente afectado por el tiempo que consuma el extraer información de las imágenes además de que el retardo que agregaría al tiempo de procesamiento sería del orden de μ segundos.

Discusión

El objetivo del proyecto fue diseñar una estrategia de captura y procesamiento de los movimientos de una persona (operador) para lograr replicarlos en un robot humanoide. Para validar su funcionamiento, se tomaron 5 fotografías las cuales fueron suficientes ya que las posturas realizadas por el sujeto de prueba, muestran las posturas base de las cuales se derivan familias de movimientos para cada brazo. Estas posturas fueron replicadas en dos objetivos de prueba que eran la simulación y un robot humanoide, los cuales respondieron de forma esperada en la réplica de movimientos. La forma de imitación observada se muestra similar mas no exacta y esto es debido a las limitaciones en las articulaciones del robot y al modelado de los movimientos en el plano 2D.

Los ángulos obtenidos por el sistema embebido se compararon con un *software* profesional de medición de ángulos en fotografías para comparar la variación que había entre estos datos, en los cuales se observan bajos porcentajes en la variación de las mediciones (menores a 10%) [62]. Aunque se deben realizar más pruebas de validación experimental, las mediciones de los ángulos en el sistema embebido resultaron ser congruentes con las obtenidas por el *software*.

Con respecto a los tiempos de procesamiento y extracción de características de las imágenes, son menores para imágenes de menor tamaño ya que al incrementarlo, se ve impactado el tiempo de procesamiento. Es importante destacar que, a pesar de las características del sistema embebido con respecto a una computadora de propósito general, mostro tiempos aceptables aun sin tener todos los bloques de procesamiento implementados de forma concurrente en la parte PL.

El protocolo MQTT es el protocolo más utilizado para aplicaciones IoT. A pesar de que existen otros protocolos más rápidos para aplicaciones en tiempo real, este mostro una velocidad aceptable para el transporte de mensajes de publicación/suscripción ya que no agrego un retardo significativo posterior al procesamiento [63].

Capítulo 5

Conclusiones y trabajo a futuro

5.1. Conclusiones

Se lograron implementar exitosamente todos los bloques planteados para el sistema de captura de posturas del cuerpo humano con resultados favorables.

Se pueden destacar 4 grandes aspectos con los cuales se alcanzaron los objetivos planteados para este proyecto: la entrada del sistema donde se definió como abordar el problema de visión, el sistema embebido para la optimización y computo dedicado, el canal de comunicaciones de paquetería ligera para la comunicación con las etapas posteriores y los objetivos de prueba para realizar los movimientos imitativos que son la simulación y el robot DARwIN-OP.

Se pueden mencionar varios puntos favorables de la simplificación el problema de seguimiento de visión, como que el usar marcadores es un método de bajo costo ya que los marcadores no necesitan ser de materiales especiales o reflejantes, basta con usar hojas de papel del color a detectar (verde). otra ventaja es la adaptabilidad a diferentes cambios de luz y de color, donde en varios casos el entorno no afecta en la tarea de detección. Con respecto a los movimientos a realizar por el operador, no se presentó ninguna ambigüedad en la imitación por el robot y simulación debido a la delimitación del plano bidimensional y la configuración elegida para trabajar. Sin embargo, a pesar de las diferentes ventajas, caben resaltar algunas limitaciones en la estrategia de seguimiento y de delimitación del plano. La primera se presenta en las restricciones en las áreas de captura donde deben ser ambientes controlados cerrados ya que en ambientes abiertos se puede caer en el riesgo de que se presente algún elemento con colores cercanos al de los marcadores o en su defecto que estén en el rango de umbrales establecidos, el programa al detectar mas objetos en la escena ajusta los umbrles de los filtros para intentar eliminar los objetos no deseados, si después del incremento no

desaparecen los objetos no deseados, el programa envía valores por defecto al robot y despliega un mensaje alertando sobre una incorrecta detección de los marcadores. En cuanto a la parte del plano 2D de trabajo, limita en cierta parte el movimiento completo del brazo, aunque para este caso de estudio no representa mayores problemas.

En el SE se implementó toda la parte programable y reconfigurable, con lo cual se logró integrar todo en un chip (APSoC). La tarjeta PYNQ-Z1 con el framework PYNQ permitió utilizar como lenguaje de programación Python en el dispositivo Zynq de xilinx, del cual se pudieron aprovechar algunas de sus capacidades, como las de redes, la interoperabilidad, las de visión, etc. Al usar Jupyter como ambiente de desarrollo permitió una interfaz gráfica amigable con la cual se importaron módulos propios del dispositivo y de la lógica reconfigurable con lo cual se pudo retroalimentar de forma visual los resultados obtenidos de los módulos con las herramientas de Python. En la parte de procesamiento, al implementar el algoritmo completo en la parte PS (preprocesado, segmentación y extracción de características) o utilizando los aceleradores propios de PYNQ se obtienen los mismos resultados de información, sin embargo implementar todon en la parte PS no es eficiente en cuanto a tiempo de procesamiento. Se redujo bastante el tiempo de procesamiento al paralelizar los módulos de procesamiento del sistema de visión con los aceleradores que proporciona la tarjeta PYNQ. Al probar con dos tipos de resolución de imagen se pudo observar un ligero impacto en el tiempo de procesamiento . Los ángulos calculados por el SE se compararon con los obtenidos por el software 4.2 donde al comparar el error entre ambos datos obtenidos , se puede observar muy poca variación en los datos y esto es debido a un ligero movimiento en los marcadores al realizar los diferentes movimientos de las posturas a imitar por lo cual se pueden considerar datos fiables.

En el simulador Webots se corroboró el funcionamiento del sistema de captura de posturas. El modelo del robot DARwIN-OP que proporciona este simulador tiene como ventaja que cuenta con todos los grados de libertad del robot físico además de muchas de las más importantes funcionalidades como las cámaras, medición de sus acelerómetros, rutinas de caminata, además de poder programar la simulación con diferentes lenguajes y conectarse con software de terceros. El poder conectarse con otros softwares permitió a la computadora central suscribirse a los tópicos publicados por el SE y vincular la simulación con la base de datos que almacenaba los datos de movimiento recibidos de los tópicos. El robot físico se habilito como cliente IoT por medio de un circuito externo que se suscribía al tópico respectivo para poder enviar strings de caracteres ASCCI al robot.

5.2. Trabajo a futuro

El sistema de captura de posturas presentado en esta tesis tiene varias oportunidades para seguir desarrollando como se muestra a continuación:

En la parte de segmentación, detección y seguimiento, se puede agregar un sistema más robusto para detectar los marcadores o para detectar las articulaciones. Además, se busca que estas técnicas de procesamiento puedan ser completamente implementadas en hardware para paralelizar la mayoría de los procesos y solo ciertas tareas sean delegadas al software.

El actual sistema de captura de posturas se limita al estudio de las extremidades superiores en un plano 2D, por lo cual, se busca extender a mas extremidades del cuerpo con movimientos completos en el plano 3D aplicando análisis de la dinámica de los movimientos del cuerpo. Se pueden modelar los movimientos 3D siguiendo la misma metodología, observando la cantidad de pixeles acumulados (magnificación), sin embargo, resultaría ambiguo si las regiones de interés se erosionan de más, por lo cual se podría cambiar la metodología utilizando más cámaras y otro tipo de marcadores, agregando sensores de profundidad o inerciales como complemento al sistema de visión.

Los movimientos no solo buscan replicarse sino también buscan estudiarse patrones de comportamiento a través de la base de datos almacenada para con ellos reconocer gestos corporales como saludos, estiramientos y con ello disparar ciertas rutinas en el robot.

Se buscó para el sistema, dotar al robot de uno de los sentidos más importantes para los humanos como lo es la visión, sin embargo, seria particularmente interesante dotar de más sentidos como el tacto para reconocer superficies, el oído para reconocer sonidos y el lugar donde se producen, etc.

Apéndice A

Código de Python

A.1. Bucle principal

```
for i in range(0,Fils+11):
   for j in range(0,Cols+11):
      if i < Fils and j < Cols:
         Rd, Gn, Bl = ImgRGB[i,j,0], ImgRGB[i,j,1],
            ImgRGB[i,j,2]
#####llamado de funcion RGB_HSV_RGB
         RGB[i,j,0], RGB[i,j,1], RGB[i,j,2] = RGB_HSV_RGB(Rd,
            Gn, Bl)
#####Binarizacion
         if RGB[i,j,1] == 255:
            I[i,j] = 0;
         else:
            I[i,j] = 255;
#####funcion filtro mediana
         filtroMediana(i,j,kmedian)
#####funcion Erosion
         Erosion(i,j)
```

A.2. Función de filtrado de color RGB-HSV-RGB

```
1
2def RGB_HSV_RGB(r,g,b):
3#################### RGB - HSV
4    r = r/255
5    g = g/255
6    b = b/255
7####VALOR MAX
8    mx = r
9    if g > mx:
10       mx = g
11       if b > mx:
12          mx = b
13#####VALOR MIN
14   mn = r
15   if g < mn:
16       mn = g
17       if b < mn:
18          mn = b
19#####CONV
20   df = mx-mn
21   if mx == mn:
22       h = 0
23   elif mx == r:
24       h = (60*((g-b)/df) + 360)%360
25   elif mx == g:
26       h = (60*((b-r)/df) + 120)%360
27   elif mx == b:
```

```
28        h = (60*((r-g)/df) + 240)%360
29
30    if mx == 0:
31        s = 0
32    else:
33        s = df/mx
34    h = h/2#de 0-360 a 0-180
35    s = s*255#de 0-1 a 0-255
36    v = mx*255#de 0-1 a 0-255
37############  UMBRAL
38    if h >= 25 and h <= 102 and s >= 52 and s <= 255 and v >=
          72 and v <= 255:
39        h, s, v = h, s, v
40    else:
41        h, s, v = 0, 0, 255
42#################### HSV - RGB
43    h = (h*2)/360
44    s = s/255
45    v = v/255
46    hh = (6*h)%6
47    c1 = int(hh)
48    c2 = hh-c1
49    x = v*(1-s)
50    y = v*(1-c2*s)
51    z = v*(1-(1-c2)*s)
52    if hh == 0:
53        r, g, b = v, z, x
54    elif hh == 1:
55        r, g, b = y, v, x
```

```
56    elif hh == 2:
57       r, g, b = x, v, z
58    elif hh == 3:
59       r, g, b = x, y, v
60    elif hh == 4:
61       r, g, b = z, x, v
62    elif hh == 5:
63       r, g, b = v, x, y
64    r, g, b = min(int(r*255),255), min(int(g*255),255),
         min(int(b*255),255)
65    return r, g, b
```

A.3. Función filtro mediana

```
1 def filtroMediana(i,j,kmedian):
2    if i > 2*rmedian and j > 2*rmedian:
3       u, v = (i-2*rmedian)+rmedian-1, (j-2*rmedian)+rmedian-1
4       if u <= (Fils-rmedian-2) and v <= (Cols-rmedian-2):
5          for i in range(-rmedian,rmedian+1):
6             for j in range(-rmedian, rmedian+1):
7                Amedian[kmedian] = I[u+i, v+j]
8                kmedian = kmedian+1
9          Ordenar(Amedian,(2*rmedian+1)*(2*rmedian+1))
10         I[u,v] = Amedian[nmedian]
```

A.4. Función erosión

```
def Erosion(i,j):
    if i > 9 and j > 9:
        i2, j2 = i-7, j-7
        if i2 > LimVent_i and i2 < Fils-F and j2 > LimVent_j
            and j2 < Cols-C: #limites de la ventana prima
            (3,5,7...)
            Mn = I[i2,j2]
            Mx = Mn
            for i3 in range(i2-LimVent_i,i2+LimVent_i):
                for j3 in range(j2-LimVent_j,j2+LimVent_j):
                    if Mn > I[i3,j3]:
                        Mn = I[i3,j3]
                    I2[i2,j2] = Mn
```

A.5. Función de etiquetado de componentes

```
def CCL(x,y,c):
    global Etiqueta
    global I2
    dx = np.array([-1,0,1,1,1,0,-1,-1])
    dy = np.array([1,1,1,0,-1,-1,-1,0])
    Etiqueta[x,y] = c
    for i in range(0,8):
        nx = x+dx[i]
        ny = y+dy[i]
        if I2[nx,ny] and not(Etiqueta[nx,ny]):
            CCL(nx,ny,c)
```

```
12    return
```

A.6. Función cálculo de ángulos

```
1 def LeyCosenos(D):
2     #distancia entre dos puntos
3     a =
          math.sqrt(pow(int(D[0])-int(D[1]),2)+pow(int(D[3])-int(D[4]),2))
4     b =
          math.sqrt(pow(int(D[0])-int(D[2]),2)+pow(int(D[3])-int(D[5]),2))
5     c =
          math.sqrt(pow(int(D[1])-int(D[2]),2)+pow(int(D[4])-int(D[5]),2))
6     #ley de cosenos para encontrar angulos
7     num1 = pow(a,2)-pow(b,2)-pow(c,2)
8     dem1 = -2*b*c
9     A = math.acos(num1/dem1)
10    A = math.degrees(A)
11    num2 = pow(b,2)-pow(a,2)-pow(c,2)
12    dem2 = -2*a*c
13    B = math.acos(num2/dem2)
14    B = math.degrees(B)
15    num3 = pow(c,2)-pow(a,2)-pow(b,2)
16    dem3 = -2*a*b
17    C = math.acos(num3/dem3)
18    C = math.degrees(C)
19    return A, B, C
```

A.7. Bucle para encontrar los centros geométrico

```
for i in range(0,Fils):
   for j in range(0,Cols):
#lamado de la funcion de etiquetado de componentes
      if I2[i,j] and not(Etiqueta[i,j]):
         Componente = Componente+1
         CCL(i,j,Componente)
#centros geometricos
      if Etiqueta[i,j] == 1:
         NumPxl[0] = NumPxl[0]+1
         Sum_i[0] = Sum_i[0] + i
         Sum_j[0] = Sum_j[0] + j
         Ci[0] = round(Sum_i[0]/NumPxl[0])
         Cj[0] = round(Sum_j[0]/NumPxl[0])
      elif Etiqueta[i,j] == 2:
         NumPxl[1] = NumPxl[1]+1
         Sum_i[1] = Sum_i[1] + i
         Sum_j[1] = Sum_j[1] + j
         Ci[1] = round(Sum_i[1]/NumPxl[1])
         Cj[1] = round(Sum_j[1]/NumPxl[1])
      elif Etiqueta[i,j] == 3:
         NumPxl[2] = NumPxl[2]+1
         Sum_i[2] = Sum_i[2] + i
         Sum_j[2] = Sum_j[2] + j
         Ci[2] = round(Sum_i[2]/NumPxl[2])
         Cj[2] = round(Sum_j[2]/NumPxl[2])
      elif Etiqueta[i,j] == 4:
         NumPxl[3] = NumPxl[3]+1
```

```
28          Sum_i[3] = Sum_i[3] + i
29          Sum_j[3] = Sum_j[3] + j
30          Ci[3] = round(Sum_i[3]/NumPxl[3])
31          Cj[3] = round(Sum_j[3]/NumPxl[3])
32      elif Etiqueta[i,j] == 5:
33          NumPxl[4] = NumPxl[4]+1
34          Sum_i[4] = Sum_i[4] + i
35          Sum_j[4] = Sum_j[4] + j
36          Ci[4] = round(Sum_i[4]/NumPxl[4])
37          Cj[4] = round(Sum_j[4]/NumPxl[4])
38      elif Etiqueta[i,j] == 6:
39          NumPxl[5] = NumPxl[5]+1
40          Sum_i[5] = Sum_i[5] + i
41          Sum_j[5] = Sum_j[5] + j
42          Ci[5] = round(Sum_i[5]/NumPxl[5])
43          Cj[5] = round(Sum_j[5]/NumPxl[5])
44      elif Etiqueta[i,j] == 7:
45          NumPxl[6] = NumPxl[6]+1
46          Sum_i[6] = Sum_i[6] + i
47          Sum_j[6] = Sum_j[6] + j
48          Ci[6] = round(Sum_i[6]/NumPxl[6])
49          Cj[6] = round(Sum_j[6]/NumPxl[6])
```

A.8. Protocolo MQTT

```
1 def on_connect(client, userdata, flags, rc):
2   if rc == 0:
3       print("Conectado al broker")
```

```
4       global Connected
5       Connected = True
6   else:
7       print("Conexion fallida")
```

Apéndice B

Código de Vivado HLS

B.1. Binarización.cpp

```
#include " binarization .hpp"
void binarize ( axis_t *INPUT, axis_t *OUTPUT, unsigned int length){
// #pragma HLS INTERFACE s_axilite port=return bundle=CTRL
#pragma HLS INTERFACE s_axilite port=length bundle=CTRL
#pragma HLS INTERFACE axis depth=50 port=OUTPUT
#pragma HLS INTERFACE axis depth=50 port=INPUT
    for(unsigned int i=0; i<length; i++){
#pragma HLS PIPELINE
        axis_t cur = *INPUT++;
        if( cur.data != 255 && cur.data != 0 ){
            cur.data = 255;
            *OUTPUT++ = cur;
        }
        else{
            *OUTPUT++ = cur;
        }
    }
}
```

B.2. Binarización.hpp

```
1 #ifndef __BINARIZATION_HPP
2 #define __BINARIZATION_HPP
3 #include "ap_int.h"
4 struct axis_t {
5     int data;
6     ap_int<1>last;
7 };
8 void erode( axis_t *INPUT, axis_t *OUTPUT, unsigned int length);
9 #endif
```

Apéndice C

Sistema embebido

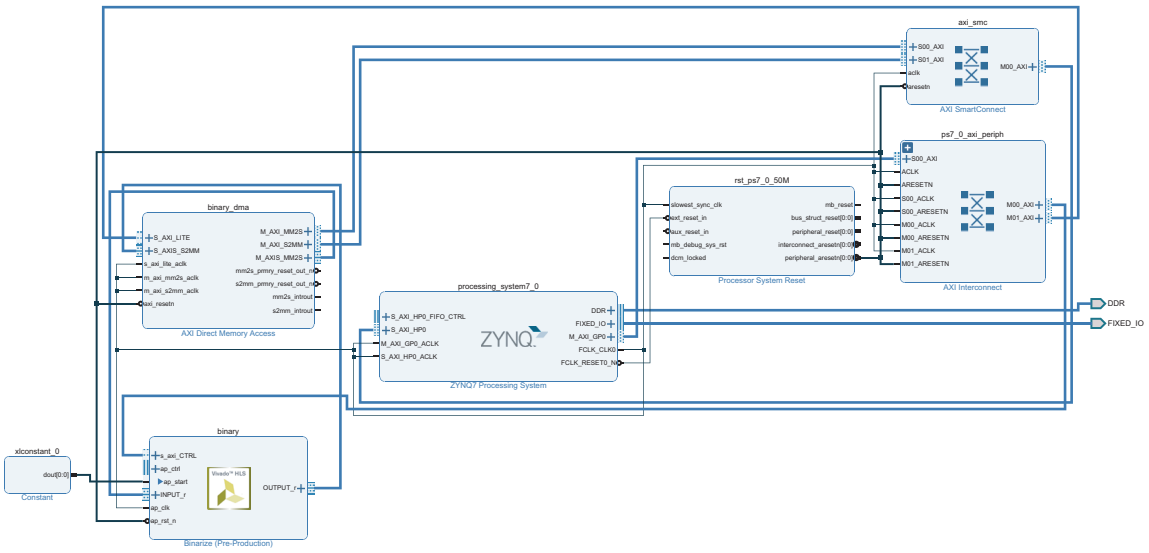


Figura C.1: Diagrama del modulo IP del *Overlay* de la binarización.

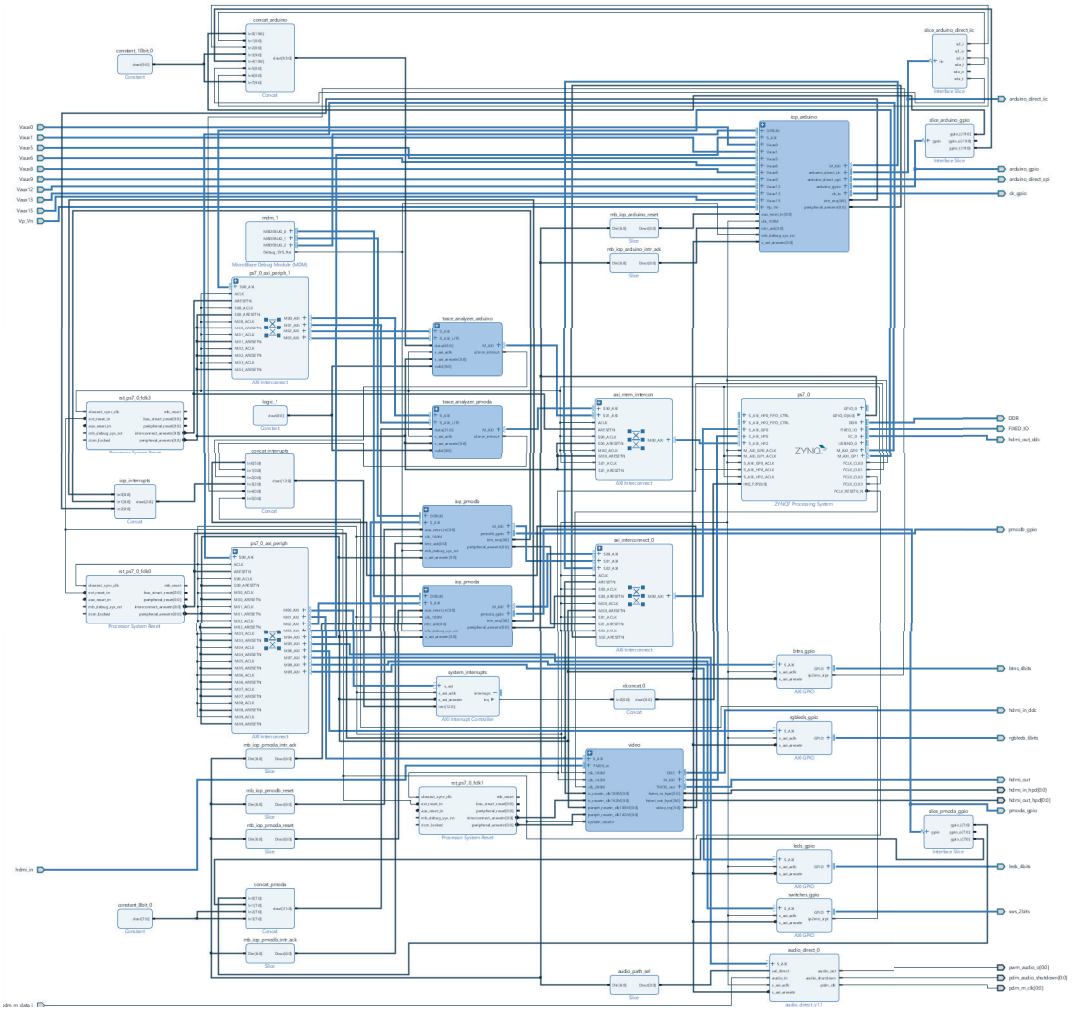


Figura C.2: Diagrama de los módulos IP del *Overlay* base del sistema embebido.

Bibliografía

[1] S. SECTOR and O. ITU, "Series y: Global information infrastructure, internet protocol aspects and next-generation networks next generation networks–frameworks and functional architecture models," *International Telecommunication Union, Geneva, Switzerland, Recommendation ITU-T Y*, vol. 2060, 2012.

[2] Copyright 2020 ROBOTIS. Powered by Jekyll & Minimal Mistakes., "Robotis e-manual," 2020. http://emanual.robotis.com/docs/en/platform/op/getting_started/#introduction, Last accessed on 2020-04-01.

[3] Copyright 2018, Xilinx, "Pynq overlays," 2020. https://pynq.readthedocs.io/en/v2.5.1/, Last accessed on 2020-04-01.

[4] T. G. Newsroom, "Toyota unveils third generation humanoid robot t-hr3," 2017.

[5] J. Hernandez-Vicen, S. Martinez, J. M. Garcia-Haro, and C. Balaguer, "Correction of visual perception based on neuro-fuzzy learning for the humanoid robot teo," *Sensors*, vol. 18, no. 4, p. 972, 2018.

[6] E. Bryndin, "Increase of safety use robots in industry 4.0 by developing sensitivity and professional behavioral skills," *American Journal of Mechanical and Industrial Engineering*, vol. 5, no. 1, pp. 6–14, 2020.

[7] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," *arXiv preprint arXiv:1905.13566*, 2019.

[8] J. M. García Haro *et al.*, "Algoritmo de captura de movimiento basado en visión por computador para la teleoperación de robots humanoides," *Actas de las XXXVIII Jornadas de Automática*, 2017.

[9] E. Yavşan and A. Uçar, "Gesture imitation and recognition using kinect sensor and extreme learning machines," *Measurement*, vol. 94, pp. 852–861, 2016.

[10] R. J. Moreno, F. A. E. Valcárcel, and D. A. Hurtado, "Control de movimiento de un robot humanoide por medio de visión de máquina y réplica de movimientos humanos," *INGE CUC*, vol. 9, no. 2, pp. 44–51, 2013.

[11] J. Koenemann, F. Burget, and M. Bennewitz, "Real-time imitation of human whole-body motions by humanoids," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2806–2812, IEEE, 2014.

[12] J. Seyssaud, J. Favrichon, K. Giraud-Esclasse, P. Girones, N. Mahjoubi, S. Bock, P. Capdepuy, and C. Moitrier, "An humanoid robot for inspections and cleaning tasks in nuclear glove box," 2015.

[13] S.-J. Yi, S. G. McGill, B.-T. Zhang, D. Hong, and D. D. Lee, "Active stabilization of a humanoid robot for real-time imitation of a human operator," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 761–766, IEEE, 2012.

[14] A. M. Sakti, D. F. Nugroho, Y. Rizky, A. Primahardika, H. Y. Saputra, F. A. Rahman, I. N. Ahmad, A. Athallah, M. R. Fuadin, M. I. Fikri, *et al.*, "Implementation of non-linear pid on darwin-op humanoid robot walking control system,"

[15] H. K. Baxi, *Teleoperative Control Plus Simulation and Analysis of Walking for the DARwIn-OP Humanoid Robot*. PhD thesis, Ohio University, 2016.

[16] V. V. Nguyen and J. Lee, "Full-body imitation of human motions with kinect and heterogeneous kinematic structure of humanoid robot," in *2012 IEEE/SICE International Symposium on System Integration (SII)*, pp. 93–98, 2012.

[17] A. W. Grammar and R. L. Williams, "Surface electromyographic control of a humanoid robot," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 55942, p. V06BT07A016, American Society of Mechanical Engineers, 2013.

[18] I. Almetwally and M. Mallem, "Real-time tele-operation and tele-walking of humanoid robot nao using kinect depth camera," in *2013 10th IEEE International Conference on networking, sensing and control (ICNSC)*, pp. 463–466, IEEE, 2013.

[19] P. Kormushev, D. N. Nenchev, S. Calinon, and D. G. Caldwell, "Upper-body kinesthetic teaching of a free-standing humanoid robot," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3970–3975, IEEE, 2011.

[20] C. Stanton, A. Bogdanovych, and E. Ratanasena, "Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning," in *Proc. Australasian Conference on Robotics and Automation*, vol. 8, p. 51, 2012.

[21] I. Rodriguez, A. Astigarraga, E. Jauregi, T. Ruiz, and E. Lazkano, "Humanizing nao robot teleoperation using ros," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 179–186, IEEE, 2014.

[22] A. Sripada, H. Asokan, A. Warrier, A. Kapoor, H. Gaur, R. Patel, and R. Sridhar, "Teleoperation of a humanoid robot with motion imitation and legged locomotion," in *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 375–379, IEEE, 2018.

[23] E.-J. Rolley-Parnell, D. Kanoulas, A. Laurenzi, B. Delhaisse, L. Rozo, D. G. Caldwell, and N. G. Tsagarakis, "Telepose: a teleoperation robotic system based on external range sensing–application on the centaur-like robot centauro,"

[24] P. F. Hokayem and M. W. Spong, "Bilateral teleoperation: An historical survey," *Automatica*, vol. 42, no. 12, pp. 2035–2057, 2006.

[25] E. Nuño and L. Basañez, "Teleoperación: técnicas, aplicaciones, entorno sensorial y teleoperación inteligente," *Dept. Control de Sistemas Industriales, Universidad Politécnica de Cataluña*, 2004.

[26] P. G. De Barros and R. W. Linderman, "A survey of user interfaces for robot teleoperation," 2009.

[27] S. Lichiardopol, "A survey on teleoperation," *Technische Universitat Eindhoven, DCT report*, vol. 20, pp. 40–60, 2007.

[28] A. González, F. Martínez, A. Pernia, F. Alba, M. Castejón, J. Ordieres, and E. Vergara, "Técnicas y algoritmos básicos de visión artificial," *La Rioja: Universidad de la Rioja*, 2006.

[29] W. Burger and M. J. Burge, *Digital image processing: an algorithmic introduction using Java*. Springer Science & Business Media, 2009.

[30] I. Sommerville, "Software engineering 9th edition," *ISBN-10*, vol. 137035152, p. 18, 2011.

[31] A. Malinowski and H. Yu, "Comparison of embedded system design for industrial applications," *IEEE transactions on industrial informatics*, vol. 7, no. 2, pp. 244–254, 2011.

[32] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440–459, 2014.

[33] T. L. Floyd, *Digital Fundamentals, 10/e*. Pearson Education India, 2010.

[34] J. G. Tong, I. D. Anderson, and M. A. Khalid, "Soft-core processors for embedded systems," in *2006 International Conference on Microelectronics*, pp. 170–173, IEEE, 2006.

[35] J. Romoth, M. Porrmann, and U. Rückert, "Survey of fpga applications in the period 2000–2015," *Technical Report*, 2017.

[36] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[37] T. Salman and R. Jain, "A survey of protocols and standards for internet of things," *arXiv preprint arXiv:1903.11549*, 2019.

[38] D. Glaroudis, A. Iossifides, and P. Chatzimisios, "Survey, comparison and research challenges of iot application protocols for smart farming," *Computer Networks*, vol. 168, p. 107037, 2020.

[39] N. N. Tai, T. Q. Vinh, *et al.*, "Chroma-key algorithm based on combination of k-means and confident coefficients," *International Journal of Information and Electronics Engineering*, vol. 4, no. 3, p. 184, 2014.

[40] A. Dutta, A. Mondal, N. Dey, S. Sen, L. Moraru, and A. E. Hassanien, "Vision tracking: A survey of the state-of-the-art," *SN Computer Science*, vol. 1, no. 1, p. 57, 2020.

[41] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1442–1468, 2013.

[42] J. Bandera, J. Rodriguez, L. Molina-Tanco, and A. Bandera, "A survey of vision-based architectures for robot learning by imitation," *International Journal of Humanoid Robotics*, vol. 9, no. 01, p. 1250006, 2012.

[43] A. C. Bovik, *Handbook of image and video processing*. Academic press, 2010.

[44] P. Muñoz Benavent, "Sistema de visión activo empotrado para robot humanoide," 2011.

[45] H. Agata, A. Yamashita, and T. Kaneko, "Chroma key using a checker pattern background," *IEICE TRANSACTIONS on Information and Systems*, vol. 90, no. 1, pp. 242–249, 2007.

[46] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons, 2020.

[47] Stephen Cass, "The top programming languages 2019," 2019. https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019, Last accessed on 2020-04-01.

[48] EETimes and Embedded, "ASPENCORE 2019 Embedded Markets Study," 2019.

[49] S. Ivaldi, J. Peters, V. Padois, and F. Nori, "Tools for simulating humanoid robot dynamics: A survey based on user feedback," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 842–849, 2014.

[50] M. David, "Mobile robot modeling simulation and programming," *BioRob & Cyberbotics Ecole polytechnique, Palaiseau*, 2013.

[51] Webots, "http://www.cyberbotics.com." Commercial Mobile Robot Simulation Software.

[52] Copyright © 2019 The Apache Software Foundation, Licensed under the Apache License, Version 2.0., "Apache," 2020. https://www.apache.org/, Last accessed on 2020-04-01.

[53] Copyright NumPy developers, "Numpy," 2020. https://numpy.org/, Last accessed on 2020-04-01.

[54] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, "Digital image processing using matlab, sec ed," *Gatesmark: Tata McGraw-Hill*, 2010.

[55] D. Bora, A. Gupta, and F. Khan, "Comparing the performance of l*a*b* and hsv color spaces with respect to color image segmentation," 06 2015.

[56] X. Wang, R. Hänsch, L. Ma, and O. Hellwich, "Comparison of different color spaces for image segmentation using graph-cut," in *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, vol. 1, pp. 301–308, IEEE, 2014.

[57] A. Jurio, M. Pagola, M. Galar, C. Lopez-Molina, and D. Paternain, "A comparison study of different color spaces in clustering based image segmentation," in *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pp. 532–541, Springer, 2010.

[58] Copyright 2011-2014, opencv dev team, "Miscellaneous image transformations. adaptivethreshold," 2019. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=hsv, Last accessed on 2020-04-01.

[59] S. Yi, B. Zhang, D. Hong, and D. D. Lee, "Active stabilization of a humanoid robot for impact motions with unknown reaction forces," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4034–4039, 2012.

[60] Ergonautas UPV, "Ergonomic ruler," 2020. www.ergonautas.upv.es, Last accessed on 2020-04-01.

[61] K. Rao, D. N. Kim, and J. J. Hwang, "Video coding standards," *The Netherlands: Springer*, pp. 51–97, 2014.

[62] T. Allahyari, A. Sahraneshin Samani, and H.-R. Khalkhali, "Validity of the microsoft kinect for measurement of neck angle: comparison with electrogoniometry," *International Journal of Occupational Safety and Ergonomics*, vol. 23, no. 4, pp. 524–532, 2017.

[63] Z. B. Babovic, J. Protic, and V. Milutinovic, "Web performance evaluation for internet of things applications," *IEEE Access*, vol. 4, pp. 6974–6992, 2016.