UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERIA ELECTRÓNICA

“DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL MEDIANTE SENSOR KINECT, ARDUINO Y COMUNICACION WIFI PARA CONTROLAR UN BRAZO ROBÓTICO DE 4GDL.”

Trabajo de titulación previo a la obtención del

Título de **Ingeniero Electrónico**

AUTORES:

- **ROBERTO LUIS MOREIRA SANCHEZ**
- **CRISTHIAN DARIO VASQUEZ ARRIAGA**

TUTOR:

MSC. MONICA MARIA MIRANDA RAMOS

GUAYAQUIL – ECUADOR

2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros **Roberto Luis Moreira Sánchez** con cédula de identificación N° **1206702613** y **Cristhian Dario Vasquez Arriaga** con cédula de identificación N° **0803230051;** manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 17 de enero del año 2022.

(f)Roberto Luis Moreira Sánchez
C.I: 1206702613

(f)Cristhian Dario Vasquez Arriaga
C.I: 0803230051

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros **Roberto Luis Moreira Sanchez** con cédula de identificación N° **1206702613** y **Cristhian Dario Vasquez Arriaga** con cédula de identificación N° **0803230051**, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del **Proyecto técnico: "Desarrollo de un sistema de visión artificial mediante sensor Kinect, Arduino y comunicación Wifi para controlar un brazo robótico de 4GDL."**, el cual ha sido desarrollado para optar por el título de: **INGENIERO ELECTRÓNICO**, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 17 de enero del año 2022.

(f)Roberto Luis Moreira Sánchez
C.I: 1206702613

(f)Cristhian Dario Vasquez Arriaga
C.I: 0803230051

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo Mónica María Miranda Ramos con documento de identificación N° 0917271785, docente de la **Universidad Politécnica Salesiana**, declaro que bajo mi autoría fue desarrollado el trabajo de titulación: **"DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL MEDIANTE SENSOR KINECT, ARDUINO Y COMUNICACION WIFI PARA CONTROLAR UN BRAZO ROBÓTICO DE 4GDL."**, realizado por **Roberto Luis Moreira Sanchez** con cédula de identificación N° **1206702613** y por **Cristhian Dario Vasquez Arriaga** con cédula de identificación N° **0803230051** obteniendo como resultado final el trabajo de titulación bajo la opción **Proyecto técnico**, que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 17 de enero del año 2022.

Atentamente,

Ing. Mónica María Miranda Ramos, MSC
C.I: 0917271785

DEDICATORIA

Le dedico este proyecto a mi Dios todopoderoso por darme la fuerza y sabiduría para realizar mis metas. A mis padres por apoyarme cuando lo necesitaba y darme valor para lograr mis metas. A mi hermana por su ayuda incondicional. A mi abuela por haberme criado y formado con amor y estar siempre conmigo. A mi abuelo que fue un modelo a seguir en mi vida que desde el cielo me sigue guiando y cuidando.

Roberto Moreira

DEDICATORIA

A Dios por su gracia y por la majestuosidad de todo lo que coloca en mi vida, a mis padres Carlos y Clara; a mis suegros Luis y Gladys, a mi amada esposa Diana y mi pequeña hija Dariana, por estar presentes en cada etapa de mi vida, han sido mi mayor motivación y a ellos les dedico cada logro personal y profesional que he alcanzado.

Cristhian Vasquez

AGRADECIMIENTO

En primera instancia quiero agradecer a Dios por brindarme de fuerza y la sabiduría necesaria para culminar todas mis metas propuestas. También agradezco a mis padres por darme el apoyo económico y emocional. A mis abuelos por haberme formado con buenos valores y conocimientos. A la Universidad Politécnica Salesiana por haberme formado profesionalmente. A mi tutora de tesis, la Ing. Mónica Miranda por su paciencia y su guía en este proceso de titulación.

Roberto Moreira

AGRADECIMIENTO

Me gustaría agradecer en estas líneas la ayuda que muchas personas y colegas me han prestado durante el proceso de investigación y redacción de este trabajo. En primer lugar, quisiera agradecer a Dios que me ha dado la fortaleza para culminar mis estudios, mis padres que me han ayudado y apoyado en todo, mis suegros que son como mis segundos padres, a mi amada esposa por ser un pilar fundamental en mi vida y a mi hija quien es la mayor inspiración de mis actos.
A todos ellos quienes fueron los que me ayudaron de una manera desinteresada, gracias infinitas por toda su bondad y buena voluntad.

Y por último y no menos importante a la Universidad Politécnica Salesiana por ser la sede de todo el conocimiento adquirido en estos años.

Cristhian Vasquez

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE PROYECTO	TEMA DE PROYECTO DE TITULACIÓN
2022	MOREIRA SANCHEZ ROBERTO LUIS VASQUEZ ARRIAGA CRISTHIAN DARIO	MSC. MONICA MIRANDA	"DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL MEDIANTE SENSOR KINECT, ARDUINO Y COMUNICACION WIFI PARA CONTROLAR UN BRAZO ROBÓTICO DE 4GDL."

El proyecto tiene como objetivo realizar el control a distancia de un brazo robótico de cuatro grados de libertad mediante el uso de un sensor Kinect que permite la detección de las articulaciones humanas, gracias a esos datos se puede realizar el análisis de coordenadas y de las articulaciones, además calcular el respectivo ángulo de giro. Esta información será enviada al módulo ESP8266 que estará conectado a la tarjeta Arduino y por medio de comunicación Wifi y el protocolo de comunicación MQTT, se podrán receptar los datos y efectuar los respectivos movimientos en el brazo robótico.

El protocolo MQTT no guarda datos en un servidor, los datos que son enviados solo se usaran si están siendo receptados, lo que proporciona rapidez a la transmisión de datos, y evitando retrasos al momento de realizar un control en el movimiento requerido en el momento. Gracias al control a distancia no se requiere que un operador este presente en el área de trabajo, pudiendo realizar trabajos en zonas donde los estándares de seguridad no sean óptimos para una persona.

El proyecto consta de dos zonas de trabajo, en la primera la persona que realiza el movimiento debe de estar ubicada frente del sensor Kinect, preferible en una zona bien iluminada y con una camisa ya sea manga corta o de un solo color. El sensor está conectado a una computadora mediante el adaptador que posee una entrada para el sensor y una salida USB para la computadora. En el PC se realiza el respectivo análisis y cálculo de datos que serán enviados a la segunda zona de trabajo en donde se ubica el Arduino que controla el brazo robótico, esta placa se conecta a internet gracias al módulo Wifi Esp8266, en esta zona se realiza el trabajo que se quiere realizar usando los movimientos emulados de la primera zona.

Palabras Claves: Kinect, MQTT, Arduino, Control, Robótica, Articulaciones, Coordenadas.

ABSTRACT

YEAR	STUDENTS	PRJ. DIRECTOR	SUBJECT
2022	MOREIRA SANCHEZ ROBERTO LUIS VASQUEZ ARREAGA CRISTHIAN DARIO	MSC. MONICA MIRANDA	"DEVELOPMENT OF AN ARTIFICIAL VISION SYSTEM THROUGH KINECT SENSOR, ARDUINO AND WIFI COMMUNICATION TO CONTROL A 4GDL ROBOTIC ARM."

The objective of the project is to remotely control a robotic arm with four degrees of freedom using a Kinect sensor that allows the detection of human joints, thanks to these data, coordinate and joint analysis can be performed, also calculate the respective angle of rotation. This information will be sent to the ESP8266 module that will be connected to the Arduino board and through Wi-Fi communication and the MQTT communication protocol, the data can be received, and the respective movements made in the robotic arm.

The MQTT protocol does not store data on a server, the data that is sent will only be used if it is being received, which provides speed in the transmission of data, and avoiding delays when performing a control on the movement required at the time. Thanks to the remote control, an operator is not required to be present in the work area, being able to carry out work in areas where safety standards are not optimal for a person.

The project consists of two work areas, in the first the person who performs the movement must be in front of the Kinect sensor, preferably in a well-lit area and with either a short-sleeved or a single-color shirt. The sensor is connected to a computer through the adapter that has an input for the sensor and a USB output for the computer. The respective analysis and calculation of data is carried out on the PC and will be sent to the second work area where the Arduino that controls the robotic arm is located, this board is connected to the internet thanks to the Esp8266 Wifi module, in this area it is carried out the work to be done using the emulated movements of the first zone.

Keywords: Kinect, MQTT, Arduino, Control, Robotic, Articulations, Coordinates.

ABREVIATURAS

MQTT: Message Queue Telemetry Transport.

IOT: Internet of Things.

GDL: Grados de libertad.

WIFI: Wireless Fidelity

USB: Universal Serial Bus

SDK: Software Development Kit

INDICE GENERAL

INDICE DE FIGURAS

INTRODUCCION

El documento está compuesto por distintos capítulos en donde se aborda; la problemática del proyecto, se explica el origen de lo que se busca resolver, así como lo que se desea obtener como resultado final, teniendo presente los antecedentes; el marco teórico, donde se explica las bases técnicas que ayudan a comprender el funcionamiento del proyecto y el cómo será realizado; la implementación, en la que se explica el cómo fue desarrollado el proyecto, incluyendo la instalación de los respectivos softwares, la conexión de los hardware y la programación respectiva para el debido funcionamiento; finalmente los resultados obtenidos, que son analizados para corroborar que los objetivos propuestos fueron cumplidos.

En la industria existen distintos ambientes no tan seguros para los operadores, por ser expuestos a gases nocivos o están en un entorno propenso a accidentes o incluso si se está pasando por una cuarentena a causa de una pandemia global, para ello se pensó una manera en la que un operador pueda realizar sus labores, como si estuviera presente en su lugar de trabajo en tiempo real, pero sin estarlo.

El uso de brazos robóticos en la industria se ha ido extendiendo a lo largo de las últimas décadas, pero poseen de un control complejo para realizar distintas tareas, debido a esto, se desarrolló un programa que, permita de manera sencilla e intuitiva, controlar el brazo robótico a la distancia, inclusive pudiendo comunicarse entre diferentes ciudades. Con todo esto en mente se utiliza el sensor Kinect que es capaz de detectar las coordenadas de las articulaciones, gracias a su cámara y sensor infrarrojo que detecta profundidad.

Lo primero que se implementó es la estación emisora, que es donde se realizara el movimiento, se requiere de una laptop en la que deberá estar instalado el software de programación Processing con los respectivos drivers necesarios para poder comunicarse con el sensor, aparte de las respectivas librerías que se usan para el análisis de datos y la comunicación por medio del protocolo de comunicación MQTT, además de eso el sensor Kinect con su respectivo adaptador para conectarse a la PC por medio del puerto USB. En esta estación se realiza el inicio del programa, la adquisición de las coordenadas de las articulaciones, el cálculo de los grados de giro entre articulaciones y el envío respectivo hacia la estación receptora.

En la estación receptora, estará un módulo ESP8266 que ayuda con la conexión a internet, el cual recepta, por medio del protocolo MQTT, los grados de giro de las articulaciones. El módulo está conectado a una tarjeta Arduino que comanda los movimientos

del brazo robótico Adeept. En la programación del Arduino se instalaron las librerías del protocolo MQTT, la del módulo ESP8266 y servo. Las articulaciones controladas son; el hombro que posee dos movimientos: arriba-abajo y adelante-atrás; codo que posee un movimiento: arriba-abajo; y mano, que sirve para abrir o cerrar.

1. EL PROBLEMA

1.1. Descripción del problema.

En la industria como en laboratorios químicos, biológicos y reactivos se realiza la investigación de patógenos peligrosos, compuesto con alta radioactividad y elaboración de nuevos compuestos altamente nocivos o inestables, estos sitios representan un peligro para la salud, se deben ejecutar todas las acciones necesarias a una distancia segura para el operador, pero con un movimiento espontaneo, estas maniobras pueden ser realizadas por Robots los cuales pueden comandarse remotamente. En este trabajo se propone usar, un brazo robótico Adeept 4-DOF, además de establecer una comunicación inalámbrica entre la persona y el robot.

1.2. Antecedentes.

A lo largo de la historia los procesos de alto riesgo tanto en la industria como en laboratorios han representado un gran peligro. Sobre todo, para quienes laboraran en ambientes donde se usan sustancias toxicas y se realizan procedimientos peligrosos. Situación que ha llevado incluso hasta la muerte de una gran cantidad de personas. Esto es debido a la complejidad de dichos procesos donde se requiere de personal humano para la toma de decisiones ante situaciones inesperadas. Hoy en día en la industria y laboratorios existen robots que hacen procedimientos de forma precisa, pero no realizan acciones espontaneas.

Según (Sanderson, 2019), el uso de brazos robóticos químicos por parte de la Universidad de Liverpool se ha programado y probado para que funcione en la planta de innovación de materiales de la universidad, donde el director de la planta, Andy Cooper, está trabajando para encontrar un catalizador. Puede hacer que el proceso de extracción de hidrógeno del agua sea más eficiente. Muestra la utilidad de los robots en los laboratorios para un manejo eficiente de ciertas sustancias sin exponer al personal humano.

Con el avance de red 5G se ha desarrollado un experimento, el cirujano Kais Assadullah Rona realizo una cirugía a distancia con un brazo robótico a una banana, con este nuevo sistema de conectividad que hace posible la operación de herramientas quirúrgicas en remoto permitiendo salvar vidas a distancia (Sanitaria, 2020).

Ante la necesidad de realizar procedimientos donde requiera precisión y además la capacidad humana en la toma de decisiones ante situaciones presentadas, tomamos como

referencia los robots operados por doctores para la realización de cirugías complejas de alta precisión y en donde la manipulación humana directa representa un riego para el paciente (Orbaugh Antillón, 2021). Con base a esto, observamos la necesidad de tener un brazo robótico operado mediante el movimiento natural del brazo de una persona que se encuentre en cualquier parte de mundo, siempre y cuando tenga acceso a internet. Dada la situación, los conocimientos adquiridos en la carrera universitaria, permiten desarrollar un prototipo que cumpla con los requerimientos.

1.3. Importancia y alcances.

El desarrollo de este prototipo demuestra la replicación del movimiento del brazo realizada por el operador en un brazo robótico que está limitado a 4 grados de libertad, con estos precedentes se podría realizar manipulación de equipos de laboratorios, procesos en la industria, aplicaciones en la medicina, solventando la necesidad y compromiso de disminuir los índices de riesgos laboral, como por ejemplo problemas de exposición química o en caso de existir un virus de alto contagio que representan un riesgo para el ser humano, en el campo de la medicina puede ser usado para la realización de intervenciones quirúrgicas desde cualquier parte del mundo siempre que posea una conexión a internet.

Mediante el sensor Kinect se utiliza un sistema de visión artificial que realiza el reconocimiento y análisis del movimiento de un brazo humano y mediante comunicación Wifi, pueda enviar datos por un servidor de mensajes de código abierto hacia el brazo robótico para poder realizar la réplica del movimiento.

1.4. Delimitación del problema.

1.4.1 Temporal.

El desarrollo tuvo una duración de doce meses a partir de la aprobación de este.

1.4.2 Espacial.

El proyecto al ser un prototipo y contar con 2 estaciones; una de análisis de imágenes y emisión de datos, y otra de recepción de datos y trabajo del brazo robótico; se realizó la construcción del proyecto en la ubicación domiciliaria de los autores, una estación está ubicada en la Av. Babahoyo y Av. Juan León Mera, en el cantón Montalvo-Los Ríos, tal como se muestra en la Figura 1. y otra estación se ubica en la parroquia Posorja de la provincia del Guayas, tal como se muestra en la Figura 2.

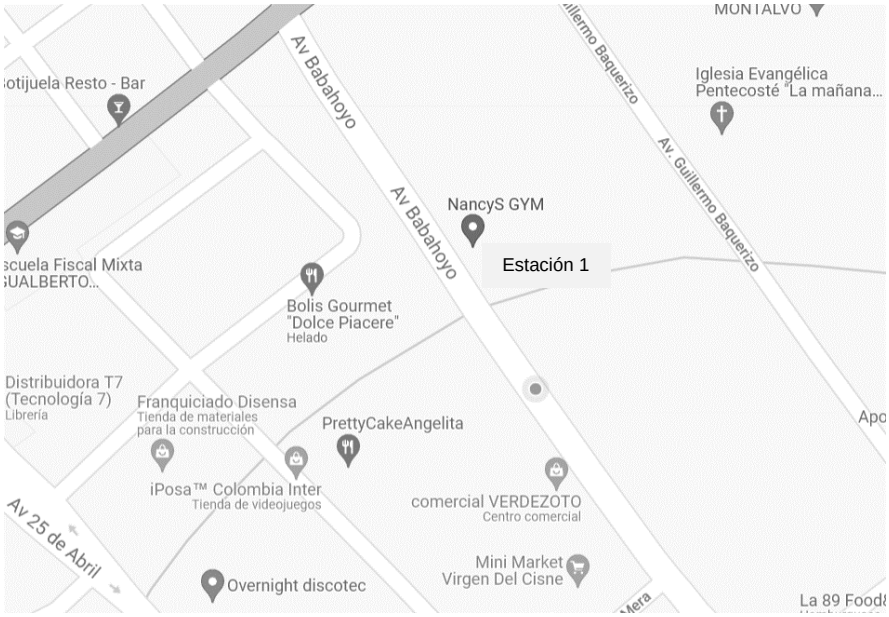


Figura 1. Ubicación de la estación 1 imagen tomada de Google Maps.

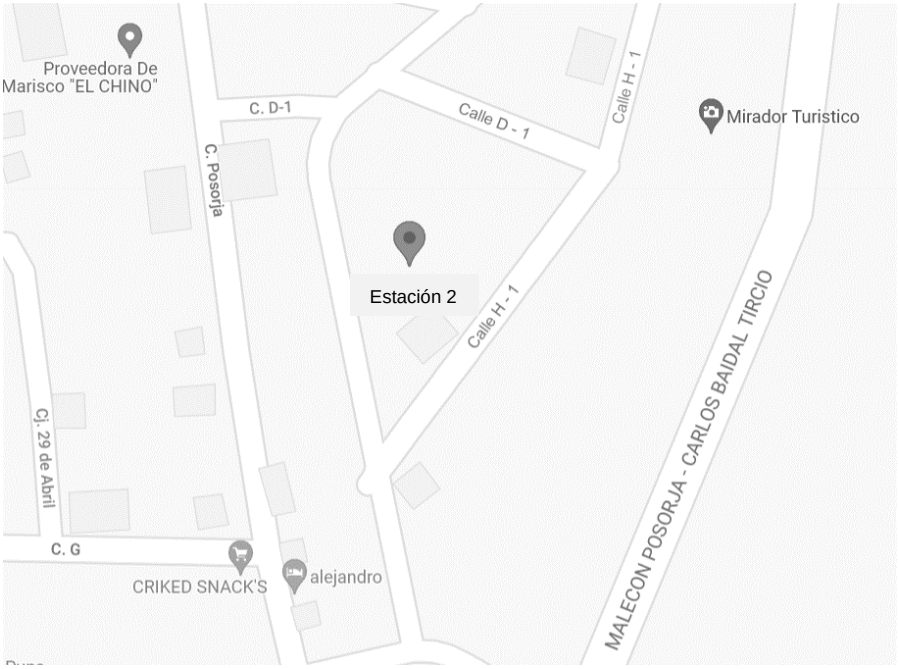


Figura 2. Ubicación de la estación 2, imagen tomada de Google Maps.

1.4.3 Académica.

El proyecto propuesto cumple con las medidas solicitadas por la Universidad Politécnica Salesiana basado en su grado investigativo y modelo de presentación para proyectos de titulación, además se aplican los conocimientos adquiridos durante todo el proceso académico en materias como: Programación, Robótica, Electrónica digital.

1.5. Objetivos.

1.5.1 Objetivo General.

Desarrollar un sistema que sea capaz de controlar en tiempo real los movimientos de un brazo robótico de 4 grados de libertad, mediante el uso de visión artificial haciendo uso de Kinect y Arduino con comunicación Wifi.

1.5.2 Objetivo Específicos.

- Identificar las coordenadas de las diferentes partes de un brazo humano, como codo, hombro y mano; mediante el sensor Kinect y el software Processing y así poder determinar los grados de giro que realizarán los servos.

- Desarrollar un programa que permita la comunicación Wifi entre la estación de análisis de movimientos y la del brazo robótico.

- Emular los movimientos de un brazo humano y verlo reflejado en un brazo robótico de 4 grados de libertad.

2. MARCO TEORICO REFERENCIAL

2.1. Sensor Kinect del xbox360.

Esta cámara salió al mercado en noviembre del 2010 por Microsoft, diseñado especialmente para uso en consolas de videojuegos, en particular, como periférico de la videoconsola Xbox 360 de Microsoft. Su principal innovación es que los usuarios pueden controlar e interactuar con la consola sin necesidad de tocar ningún control, a través de una interfaz de usuario natural basada en gestos y comandos de voz. En la Figura 3 se observa el dispositivo Kinect para la xbox360 (Salvatore, Osio, & Morales., 2014).

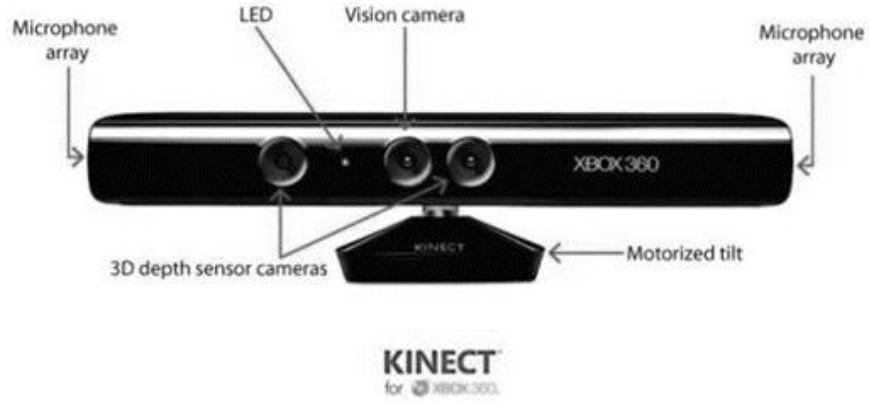


Figura 3. Sensor Kinect (ResearchGate, 2022).

2.2. Adaptador Kinect-USB.

El adaptador USB de Kinect entabla una conexión del sensor a una computadora ayudando que la información sea procesada fácilmente, este dispositivo se introdujo al mercado con la llegada de Xbox One S para así utilizarlo en la consola Slim y después en la Xbox One X, siendo esta una de la consola más potente con 4K y HDR. Cabe mencionar que este adaptador no es fabricado por la empresa Microsoft, pudiéndose apreciar su forma y la de sus terminales en la Figura 4 (Microsoft, 2020).

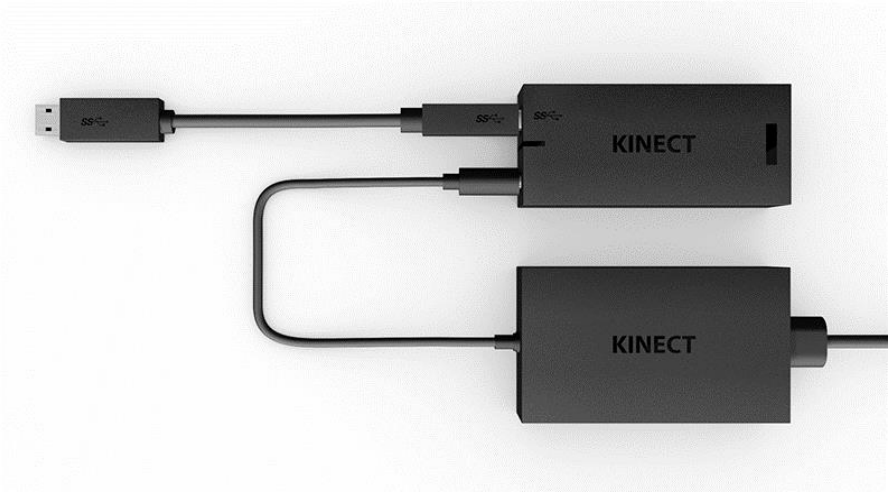


Figura 4*.* Adaptador de Kinect para PC con Windows 10 (Microsoft, 2020).

2.3. Visión artificial.

La visión artificial es la forma en que las maquinas obtienen información del mundo a su alrededor, mediante cámaras recolectan imágenes que pasan por un proceso de ajuste para que sean entendible por las maquinas, las cuales hablan un lenguaje matemático donde transforman los colores en vectores de color y posición pasando a procesar obteniendo histogramas, gráficos estadísticos, umbrales y patrones (Germán & Esteban, 2014).

(Marcos, y otros, 2006) Afirma que mediante dispositivos ópticos se provee imágenes a las maquinas dotándolos de visión artificial para mejorar la comprensión y deducción automática de estructuras y propiedades tridimensionales, como también colores y movimientos donde dicha información es de gran utilidad para el control de procesos.

Para llevar a cabo un funcionamiento básico de visión artificial se debe de seguir una cantidad de etapas previas para poder realizar una tarea, esto se aprecia en la Figura 5.

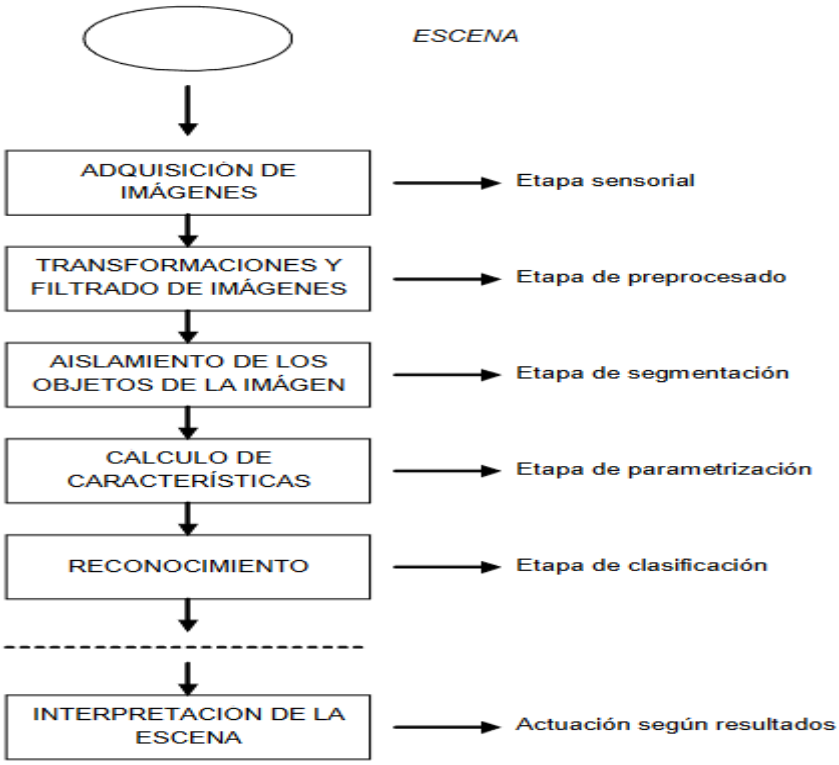


Figura 5. Diagrama de bloques de las etapas de un sistema de visión artificial (Marcos, y otros, 2006).

2.4. Componentes de un sistema de visión artificial.

Para la obtención de un sistema de visión artificial se necesitan de hardware que son mínimamente requeridos para realizar una aplicación básica. El funcionamiento de estos se muestra en la Figura 6.

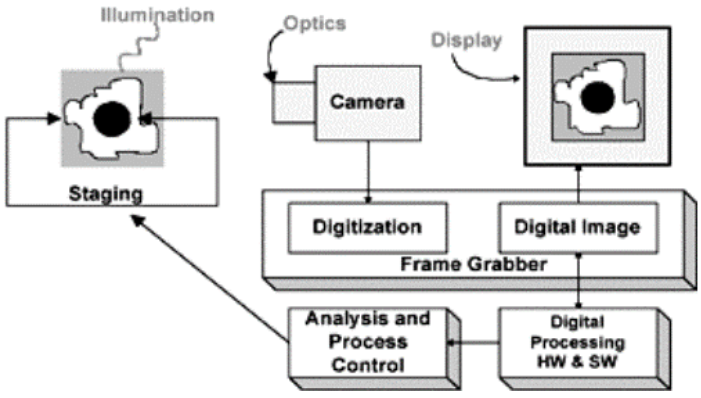


Figura 6. Diagrama de bloques de un sistema de visión artificial (Marcos, y otros, 2006).

2.4.1. Sensor óptico.

El Este sensor podría ser una cámara color o monocromo la cual crea una imagen completa del dominio del problema cada 1/30 segundos. Los sensores también puede ser cámara scanner que crea una línea en cada momento. En esta situación el desplazamiento del objeto por la línea del scanner (o al revés) crea la imagen bidimensional. La naturaleza del sensor y la imagen que genera vienen determinadas por la aplicación (Marcos, y otros, 2006).

2.4.2. Tarjeta de adquisición de imagen.

La señal de video obtenida en el subsistema anterior es digitalizada para luego pasar al computador (Marcos, y otros, 2006).

2.4.3. Computador.

Cuando la imagen es digitalizada se almacenada en la memoria de un computador para que pueda ser procesada y manipulada por programa (Marcos, y otros, 2006).

2.4.4. Monitor de video.

Permite realizar la visualización de imágenes o escenas captadas como los resultados del procesamiento de estas imágenes (Marcos, y otros, 2006).

2.5. Processing.

Processing es un cuaderno de bocetos de software flexible y un lenguaje para aprender a codificar dentro del contexto de las artes visuales. Desde 2001, Processing ha promovido la alfabetización en software dentro de las artes y la alfabetización visuales dentro de la tecnología. Hay decenas de miles de estudiantes, artistas, diseñadores, investigadores y aficionados que utilizan Processing para el aprendizaje y la creación de prototipos. En la Figura 7 observamos el entorno de programación del software (Processing Foundation, 2020).

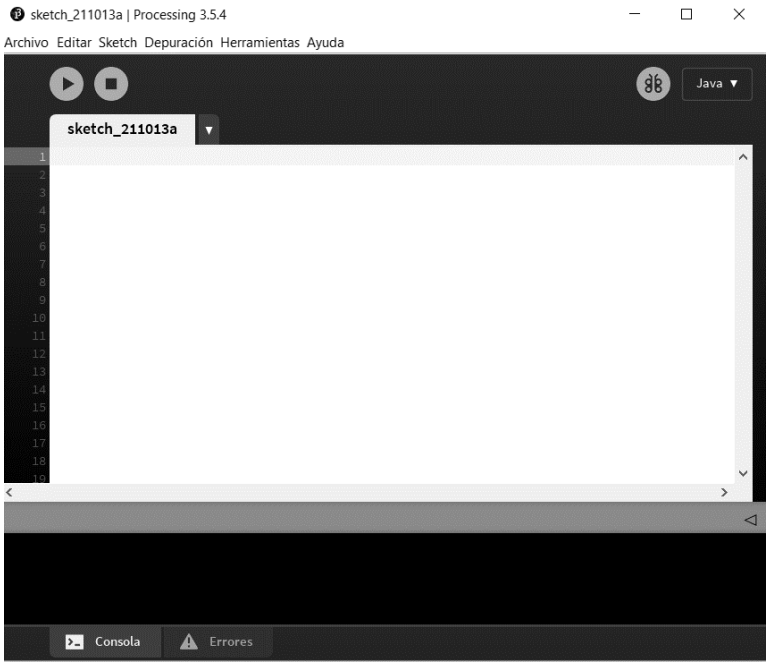


Figura 7. Entorno de programación del software Processing.

2.6. Simple Open NI.

Mediante esta librería se realiza la comunicación del sensor Kinect con el software Processing para poder estudiar la información que capta del sensor. Con esta librería se realiza el reconocimiento de las articulaciones que permiten obtener los diferentes ángulos de giro.

2.7. OpenCV.

Es una librería de procesamiento de imágenes por computadora, lo que permite clasificar y distinguir diferentes aspectos de una imagen, como brillo, profundidad, rasgos faciales colores, entre otros.

2.8. Driver.

Son controladores utilizados para interactuar con hardware conectados a pc que mediante programas informáticos le permiten al sistema operativo interactuar con diferentes equipos y elementos externos mediante una interfaz de usuario (Martínez Ramírez, SesrisnelbaTorres Manzo, & Alberto Palacio., 2018).

2.9. SDK Kinect para Windows.

El Kit de desarrollo de software es un driver que permite al sensor Kinect ser reconocido por Windows, con el fin de poder ser usado para la aplicación que lo requiera (Microsoft, 2020).

2.10. Protocolo de comunicación MQTT.

Es un protocolo de mensajería estándar de OASIS para Internet de las cosas (IoT). Que permite el envío y la recepción información con calidad y seguridad, mensajería de publicación / suscripción extremadamente liviana que es ideal para conectar dispositivos remotos con una huella de código pequeña y un ancho de banda de red mínimo. El MQTT es protocolo que utiliza wifi para diferenciar entre distintos dispositivos por lo cual se utiliza en una amplia variedad de industrias, como la automotriz, la fabricación, las telecomunicaciones, el petróleo, el gas, etc. En la Figura 8 apreciamos la arquitectura de red con la que funciona el protocolo (MQTT, 2021).

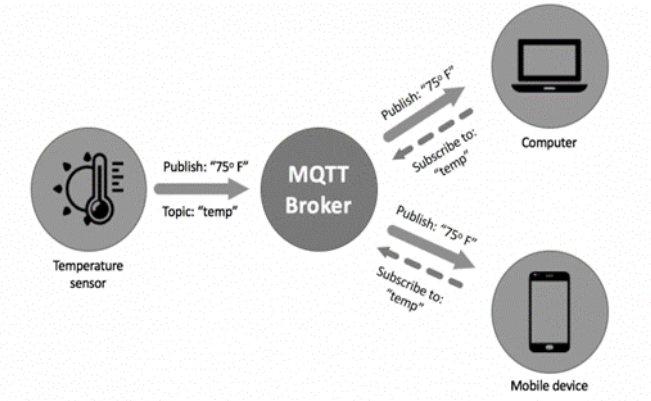


Figura 8. Arquitectura de red del protocolo MQTT (MQTT, 2021).

2.11. Módulo ESP-01.

El ESP8266EX de Espressif ofrece una solución WiFi SoC altamente integrada para satisfacer las necesidades de los usuarios, la demanda de eficiencia energética ininterrumpida, un diseño compacto y un rendimiento confiable en la industria de Internet de las cosas. Con una red WiFi completa e independiente, el ESP8266EX puede funcionar como una aplicación independiente o como esclavo del MCU anfitrión. Cuando el ESP8266EX aloja la aplicación, se inicia rápidamente desde la memoria flash. La caché de alta velocidad incorporada aumenta el rendimiento del sistema y optimiza la memoria del sistema. Además, ESP8266EX se puede aplicar a cualquier diseño de microcontrolador como un adaptador WiFi

a través de interfaces SPI / SDIO o UART. La Figura 9 muestra la forma del componente (Espressif Systems, 2020).

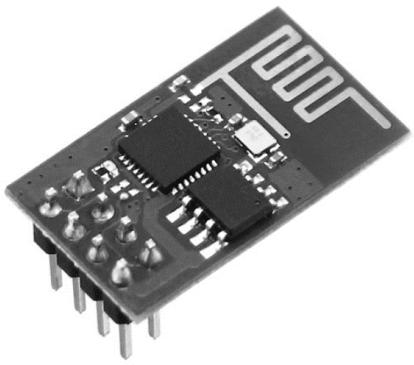


Figura 9. Módulo ESP-01 (Espressif Systems, 2020).

2.12. Adaptador USB a ESP8266 ESP-01.

Este módulo hace posible que la conexión USB directa del módulo WiFi ESP8266 ESP-01 a un pc para realizar la programación y uso de este dispositivo, utiliza el circuito integrado CH340, para la conversión USB – Serial (UART). El modelo que se utilizo es como el de la Figura 10 (UNIT ELECTRONIC, 2021).

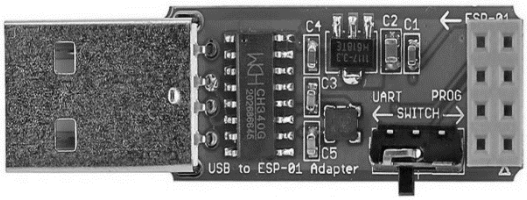


Figura 10. Adaptador USB a ESP8266 (UNIT ELECTRONIC, 2021).

2.13. Arduino.

Es un pequeño ordenador personal diseñado para la creación de proyectos de aprendizaje e investigación que está basada en hardware y software libre, flexible y fácil de programar para creadores y desarrolladores, esta placa consta de un microcontrolador reprogramable y una serie de pines tanto de entrada como de salida para realizar la recepción y emisión de información para el control de sensores y actuadores, tal como se puede observar en la Figura 11. Desde un punto de vista completamente práctico veremos cómo configurarla y usarla para desarrollar programas y sistemas electrónicos propios. Se invertirá esfuerzo considerable en comunicar la Arduino UNO con el mundo físico (ARDUINO.cl, 2021).

Figura 11. Arduino Uno (ARDUINO.cl, 2021).

2.14. Robótica.

Un robot es una máquina en la que están incorporados componentes mecánicos, eléctricos, electrónicos y de comunicación ya que están equipados con un sistema informático para el control en tiempo real, la percepción del entorno y la programación.

En robótica industrial, es fundamental aportar flexibilidad a los procesos de producción manteniendo la productividad obtenida con una máquina automatizada especializada.

A lo largo de ochenta y nueve años, muchas máquinas se han diseñado con la intención no de reemplazar la actividad directa de los trabajadores en la línea de producción, sino de realizar tareas que representan un riesgo para la salud los obreros, como la manipulación de objetos pesados o de gran tamaño, para las industria la utilización de robot ha significado un gran avance que genera beneficios económicos, un robot puede trabajar sin descanso las 24 horas los 7 días de la semana por lo que produce más a bajo costo (Baturone, 2005).

2.15. Brazo robótico.

Es un tipo de brazo mecánico, normalmente programable, con funciones parecidas a las de un brazo humano; este puede ser la suma total del mecanismo o puede ser parte de un robot más complejo. Las partes de estos manipuladores o brazos son interconectadas a través de articulaciones que permiten, tanto un movimiento rotacional (tales como los de un robot articulado), como un movimiento translación al o desplazamiento lineal, como se aprecia en la Figura 12 (Pérez, 2013).

Figura 12. Brazo robótico adeept de 4 dof (Adeept.com, 2021).

2.16. Estructura de robots manipuladores.

Para (Baturone, 2005) los robots manipuladores son, esencialmente, brazos articulados. De forma más precisa, un manipulador industrial convencional es una cadena cinemática abierta formada por un conjunto de eslabones o elementos de la cadena interrelacionados mediante articulaciones o pares cinemáticos. Las articulaciones permiten el movimiento relativo entre los eslabones.

En la robótica se encuentran con varias clases de articulaciones, en la Figura 13 se muestran los más usados y los grados de libertad, movimientos independientes que pueden realizar las articulaciones, que poseen.

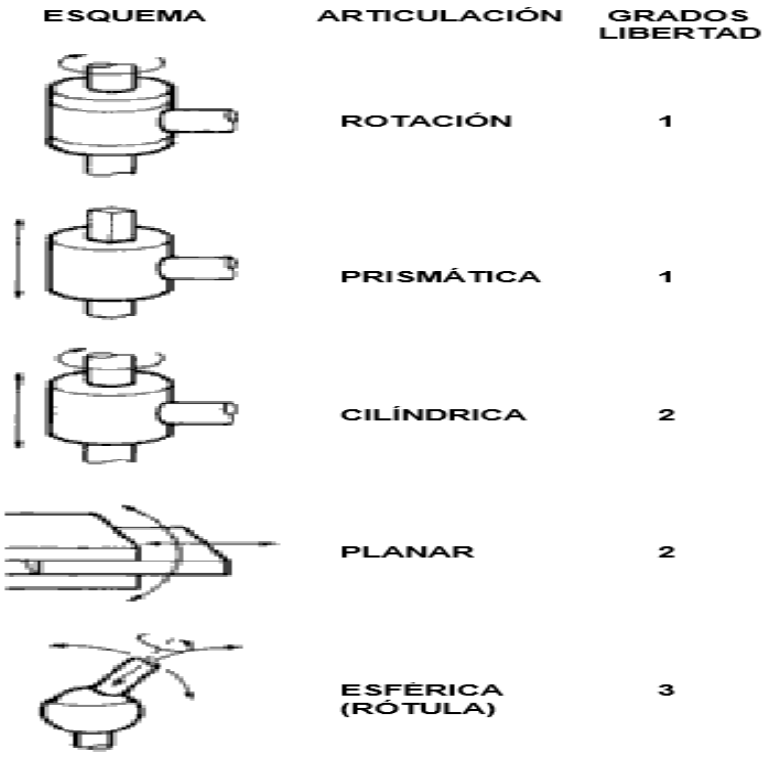


Figura 13. Tipos de articulaciones (Baturone, 2005).

La estructura típica de un robot manipulador consiste en un brazo compuesto por elementos con articulaciones entre ellos. En el último enlace se coloca un órgano terminal o efector final, tal y como una pinza o un dispositivo especial para realizar operaciones. El espacio de trabajo es el volumen encerrado por las superficies que determinan los puntos a los que accede el manipulador con su estructura totalmente extendida y plegada. En la Figura 14 observamos las configuraciones más usadas en la robótica móvil.

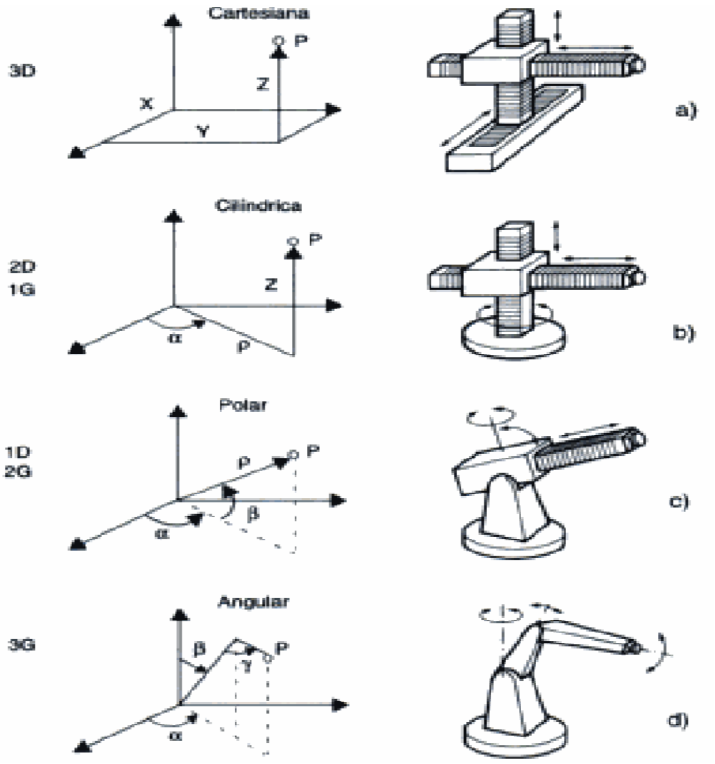


Figura 14. Tipos de configuraciones de robótica móvil (Baturone, 2005).

2.16.1. Configuración cartesiana.

Posee tres articulaciones prismáticas, cuya posición se especifica mediante coordenadas cartesianas. No poseen un volumen de trabajo apto para espacios cerrados, debido a que es muy reducido comparado a las otras configuraciones. Es normalmente utilizados para transportar cargas pesadas.

2.16.2. Configuración cilíndrica.

Posee dos articulaciones prismáticas y una de rotación, cuya posición se especifica mediante coordenadas cilíndricas. Son utilizadas en el centro de varias máquinas para aportar su servicio.

2.16.3. Configuración polar o esférica.

Posee dos articulaciones de rotación y una prismática, cuya posición se especifica mediante coordenadas polares. Posee un amplio volumen de trabajo solo inferior a la configuración de tipo angular.

2.16.4. Configuración angular,

Posee tres articulaciones de rotación, cuya posición se especifica mediante coordenadas angulares. Tiene la mejor movilidad en espacios cerrados, útil para trabajos de construcción. Su mayor uso es en la industria, para trabajos de manipulación y armados de piezas.

2.16.5. Configuración Scara.

Posee dos articulaciones de rotación con respecto a 2 ejes paralelos y un desplazamiento perpendicular. Es mayormente usado para labores de montaje, su área de trabajo depende de la longitud de la articulación perpendicular, y los 360 grados de las articulaciones de rotación.

2.17. Vectores posición.

Para representar la ubicación de un elemento con respecto a otro, se usan puntos ubicados en un espacio con 3 planos perpendiculares entre sí. Todos los planos se interceptan entre sí en el punto de origen o punto de referencia, las coordenadas son las distancias recorridas por un punto con respecto a los planos correspondientes, demuestra que tan distante se encuentre con respecto al origen. Los planos están compuestos por 2 ejes de referencia, en un espacio en 2 dimensiones (x,y), y 3 ejes de referencia en un espacio de 3 dimensiones(x,y,z) (Ayala & Barragan, 2013).

2.17.1. Distancia entre puntos.

La distancia entre dos puntos en un plano es la magnitud del vector dirección de ambos puntos. Si se tiene un punto A(ax,ay,…an) y B(bx,by,…bn), según (Kolman & Hill, 2006), el vector AB es:

$$A-B=(bx-ax),\ (by-ay),\ldots(bn-an) \qquad (1)$$

Por consiguiente, la distancia es determinada por la fórmula de magnitud de un vector:

$$D=|AB|=\sqrt[2]{(\mathbf{bx}-\mathbf{ax})^{\mathbf{2}}+(\mathbf{by}-\mathbf{ay})^{\mathbf{2}}\ldots+(\mathbf{bn}-\mathbf{an})^{\mathbf{2}}} \quad (2)$$

2.17.2 Angulo entre vectores.

Según (Kolman & Hill, 2006), el ángulo "Ɵ" entre 2 vectores (U y V) está dada por la siguiente ecuación:

$$Cos(Ɵ)=\frac{\boldsymbol{U.V}}{|\boldsymbol{U}|*|\boldsymbol{V}|} \quad (3)$$

Despejando el ángulo "Ɵ" se obtiene:

$$Ɵ=\boldsymbol{Cos^{-1}}\left(\frac{\boldsymbol{U.V}}{|\boldsymbol{U}|*|\boldsymbol{V}|}\right) \quad (4)$$

3. MARCO METODOLOGICO.

3.1. Funcionalidad.

El presente proyecto presenta una solución desarrollada por 2 etapas, una de emisión y análisis de datos, y la segunda de recepción y ejecución de movimientos. En la Figura 15 observamos un diagrama de funcionamiento detallando el proceso de manera general.

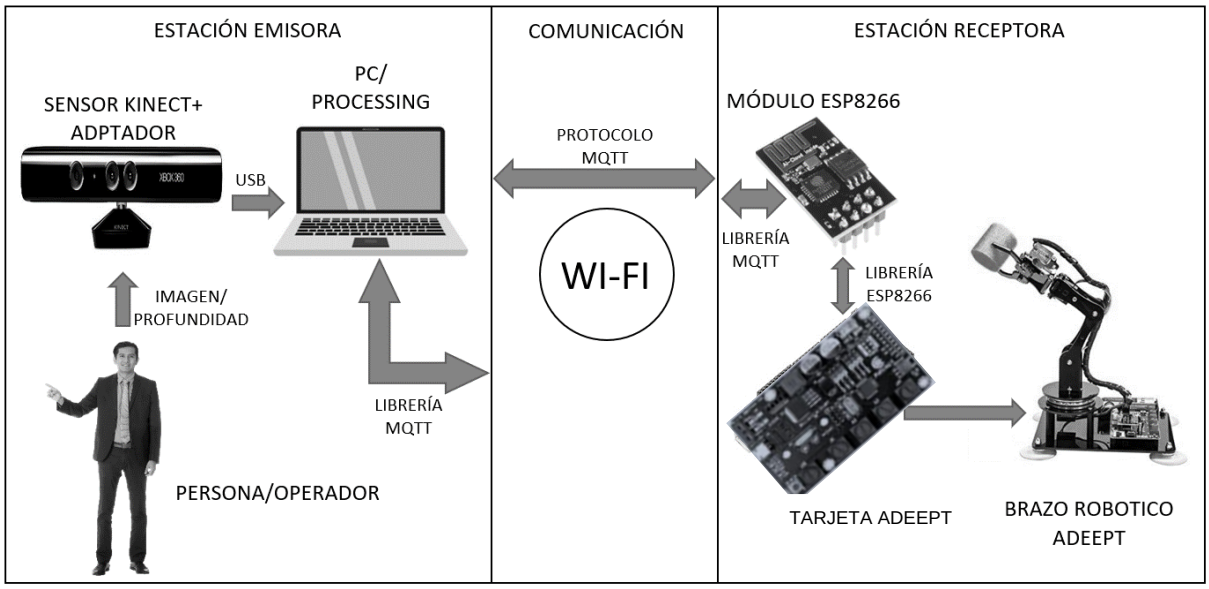


Figura 15. Diagrama de funcionamiento.

El proceso comienza cuando se inicializa el programa y el operador se ubica a una distancia de aproximadamente 1.5 metros del sensor Kinect y realiza el gesto que permite arrancar con la comunicación a la espera que la estación receptora esté disponible. Una vez conectadas el sensor Kinect envía los datos a la PC, tanto imagen como profundidad, por medio del adaptador USB. Esta información es recibida gracias a los drivers que fueron previamente instalados del SDK de Kinect, y es procesada por el software Processing por medio de las librerías SimpleOpenNI y OpenCV. Las librerías permiten detectar las ubicaciones de las articulaciones, entregando sus coordenadas en 3 dimensiones, siendo necesarias para esta solución, los hombros, el codo, la mano y los dedos. Con las coordenadas de las articulaciones se pueden hallar los vectores entre puntos, y entre vectores se puede calcular los ángulos de giro y de apertura de mano. Una vez determinado los grados de giro se procede a enviarlo a la estación receptora por medio del protocolo MQTT. Se determina un nombre de host para que pueda ser recibido del otro lado, para ello se hace uso de su respectiva librería.

En la estación receptora, el Arduino es capaz, con las respectivas librerías, de acceder a la comunicación WIFI gracias al módulo ESP8266, y recibe los datos por medio del protocolo MQTT, siempre y cuando se conecte al host correspondiente a la estación emisora. Se adquieren los datos de los grados de giro y de las articulaciones a las que corresponden, y son enviados a sus respectivos servos para poder realizar los movimientos del brazo Adeept. En la Figura 16 se observa el diagrama de flujo que resume el proceso realizado.

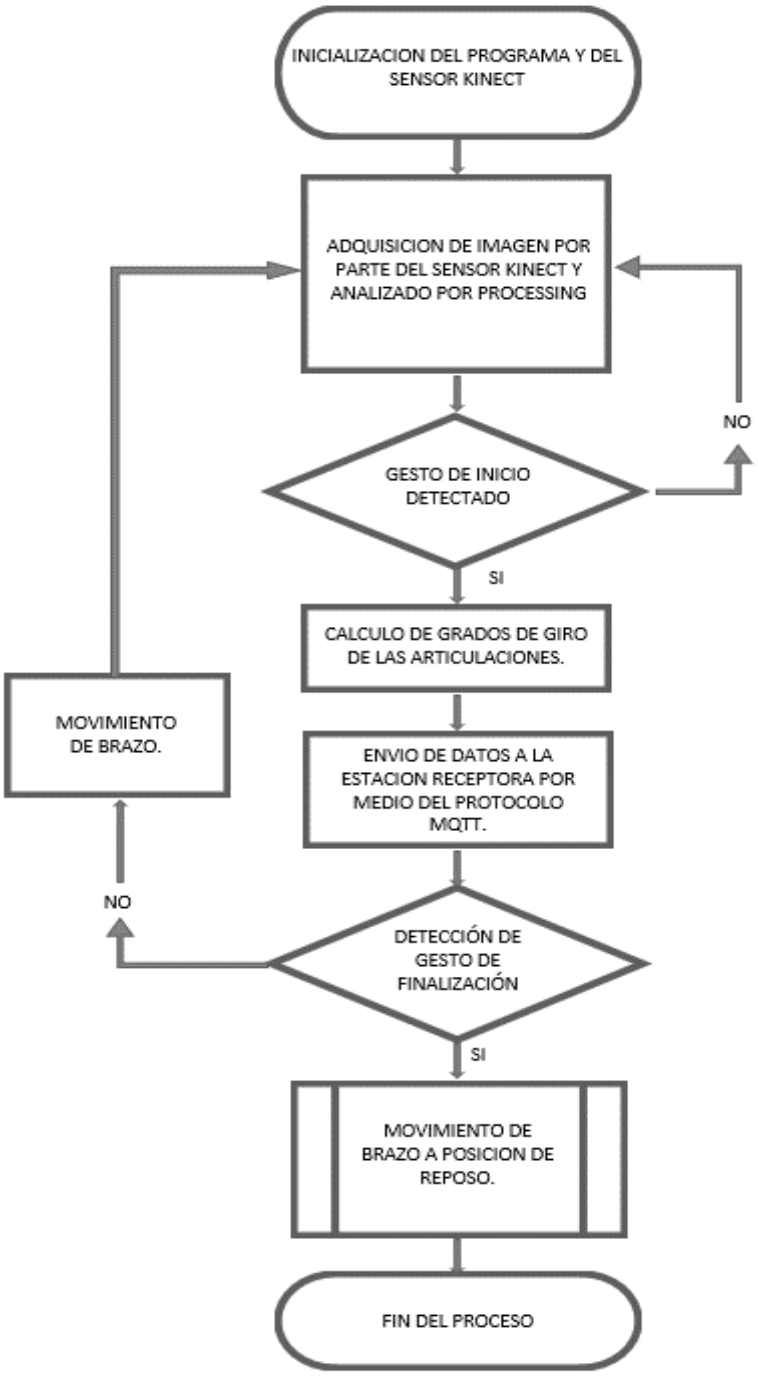


Figura 16. Diagrama de flujo del proceso.

En la Figura 17 se observa el diagrama de bloques del proceso.

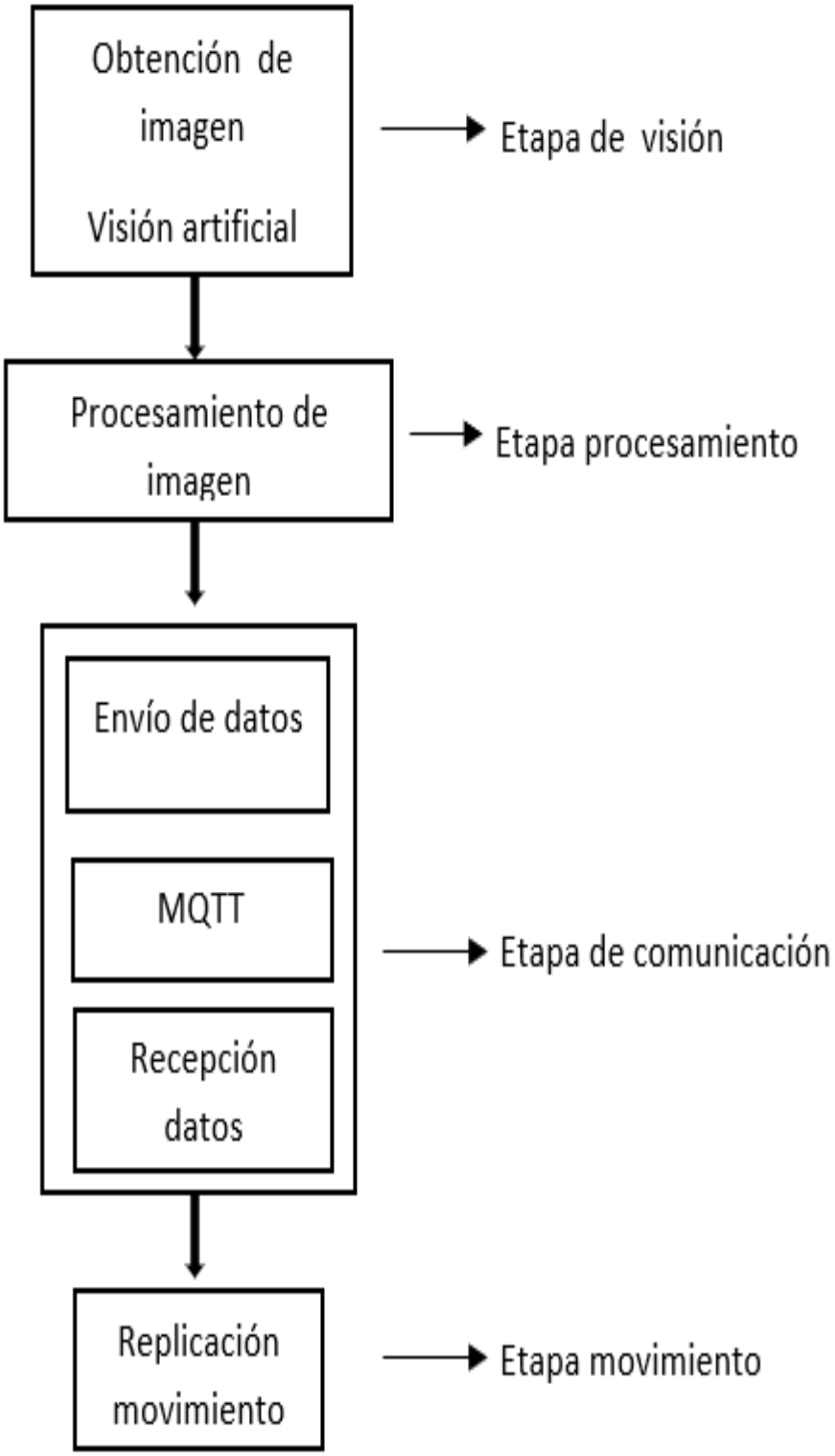


Figura 17. Diagrama de bloques del proceso.

3.2. Herramientas necesarias.

Las herramientas necesarias para el desarrollo del proyecto dependen de la estación en la que se usan. Los precios de los componentes también se adjuntan basándose en el valor en el que fueron adquiridos, estos datos se muestran en la Tabla 1.

Tabla 1
Listado de materiales

Descripción	Cantidad	Valor unitario ($)	Valor final ($)
Estación emisora			
Computadora	1	1000	1000
Adaptador Kinect a USB	1	40	40
Sensor Kinect	1	60	60
Led de alta intensidad	3	3	9
Resistencias (330 ohm)	3	0.2	0.6
Botón	1	0.5	0.5
Batería 9v	1	1	1
Guantes	1	2	2
Estación receptora			
Brazo robótico Adeept	1	115	115
Modulo Esp8266	1	3	3
Adaptador USB-esp8266	1	4	4
Impresión 3D de extensión	2	5	10
Cable de conexión jumper	6	0.2	1.2
		Total:	1246.3

Las especificaciones mínimas de la computadora son: Sistema operativo Windows 8 (x64 bit) en adelante, 2GB de RAM, Procesador de doble núcleo a 3.2 GHz o más. La computadora que se uso es de características superiores.

También se requiere del software que debe ser debidamente instalado en el ordenador para la correcta comunicación, análisis y procesamiento de datos que nos proporcionan los hardware.

3.2.1. Processing

Para descargar el software se busca en la página oficial de Processing, “https://processing.org/”, en las opciones de descargas se escoge la opción para el sistema operativo que se requiera, en este caso Windows 64-bit, tal y como se aprecia en Figura 18 y Figura 19.

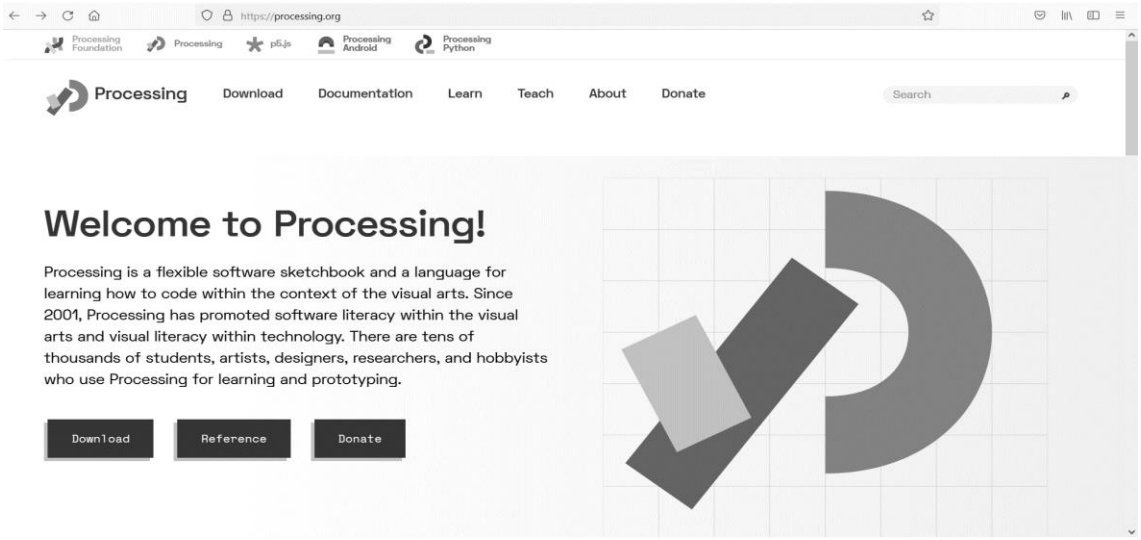


Figura 18. Página de Processing.

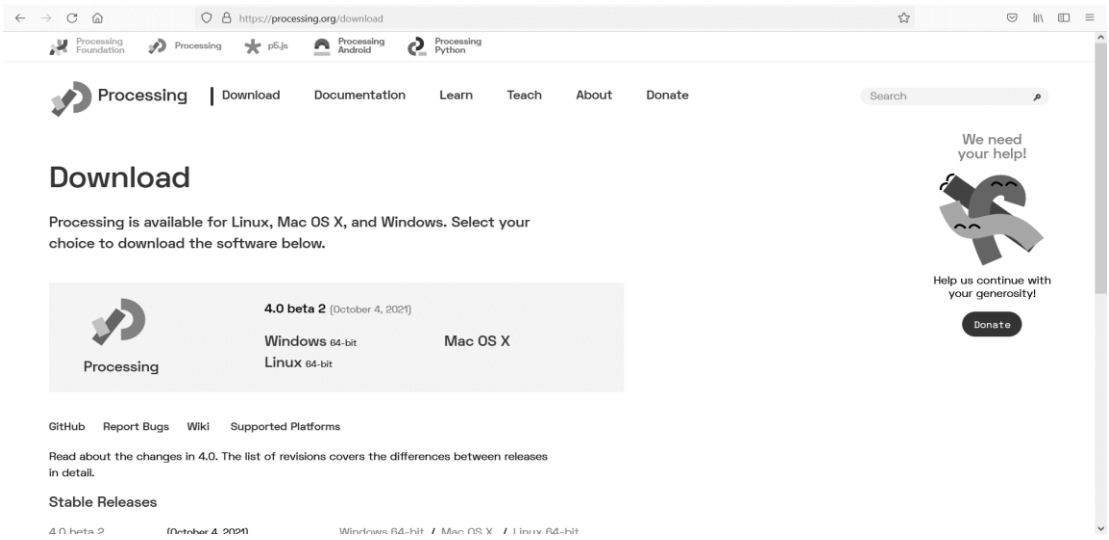
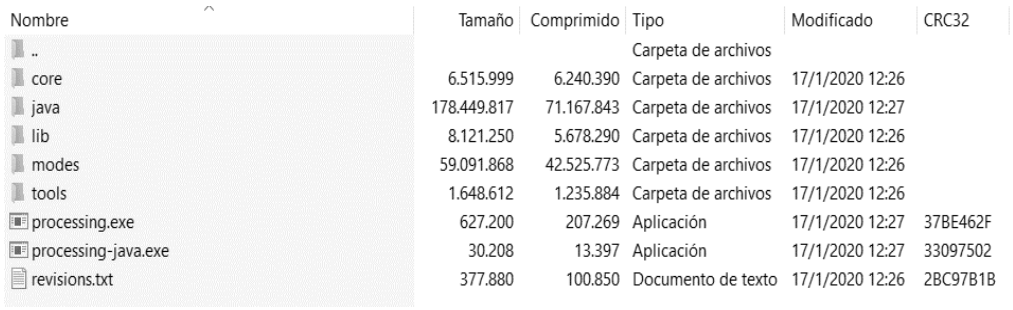


Figura 19. Página de descarga de Processing.

Actualmente Processing cuenta con la versión 4.0, para el proyecto se usa la versión 3.5.4. Una vez descargado se procede a ejecutar el archivo para empezar a instalar. En la Figura 20 se muestran los archivos descargados.



Nombre	Tamaño	Comprimido	Tipo	Modificado	CRC32
..			Carpeta de archivos		
core	6.515.999	6.240.390	Carpeta de archivos	17/1/2020 12:26	
java	178.449.817	71.167.843	Carpeta de archivos	17/1/2020 12:27	
lib	8.121.250	5.678.290	Carpeta de archivos	17/1/2020 12:26	
modes	59.091.868	42.525.773	Carpeta de archivos	17/1/2020 12:26	
tools	1.648.612	1.235.884	Carpeta de archivos	17/1/2020 12:26	
processing.exe	627.200	207.269	Aplicación	17/1/2020 12:27	37BE462F
processing-java.exe	30.208	13.397	Aplicación	17/1/2020 12:27	33097502
revisions.txt	377.880	100.850	Documento de texto	17/1/2020 12:26	2BC97B1B

Figura 20. Carpeta descargada de Processing.

El proceso de instalación es inmediato, puesto a que solo requiere del ejecutable, una vez ejecutado se muestra el IDE del software, tal y como se aprecia en la Figura 7.

En Processing existen dos maneras de importar librerías, desde el mismo software o de manera externa. Las librerías externas se descargan y se pegan dentro de la carpeta “libraries” tal y como se muestra en la Figura 21, siguiendo la ruta desde la carpeta de instalación: “processing-3.5.4\modes\java\libraries”.

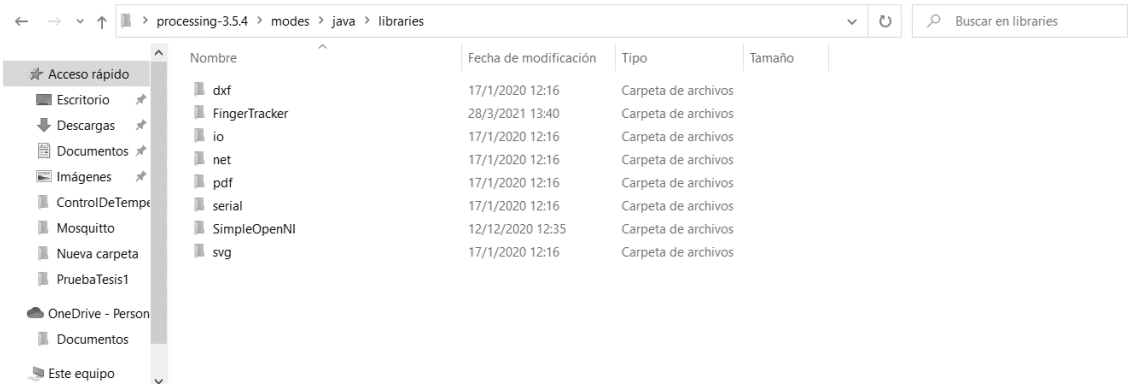


Figura 21. Carpeta de librerías de Processing.

La otra manera de importar librerías es buscándola dentro del software en donde se facilitan librerías que se encuentren disponibles, tal es el caso de MQTT. En la Figura 22 se observa como es el acceso y en la Figura 23 se muestra la ventana de búsqueda.

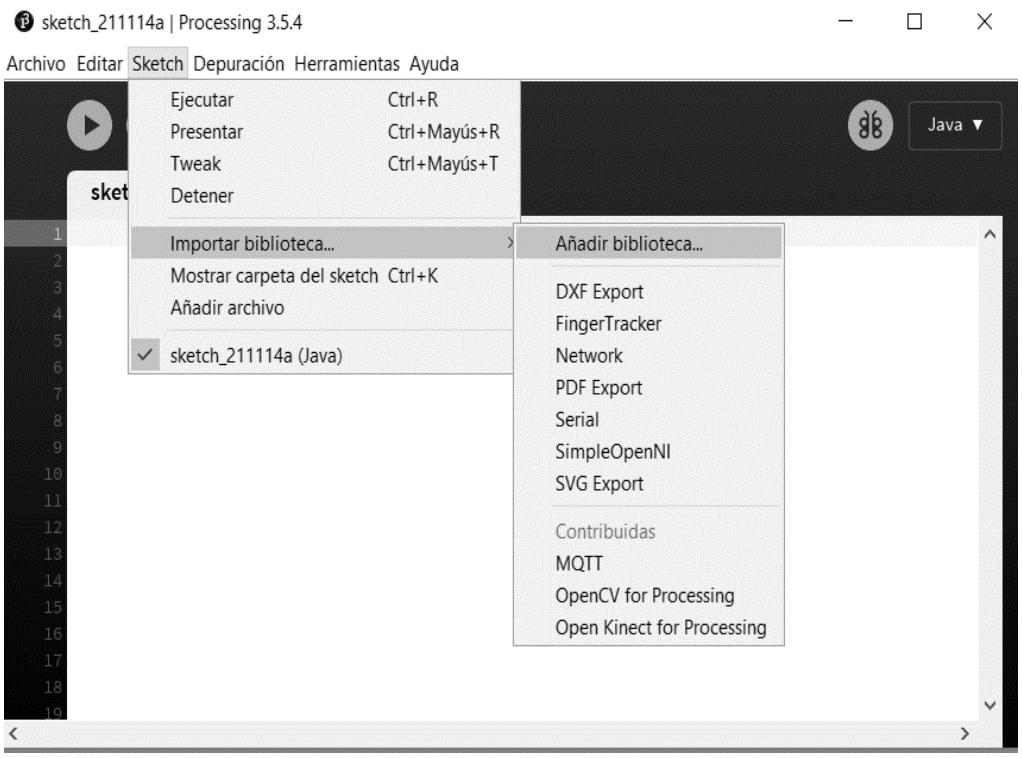


Figura 22. Importación de librerías desde el Processing.

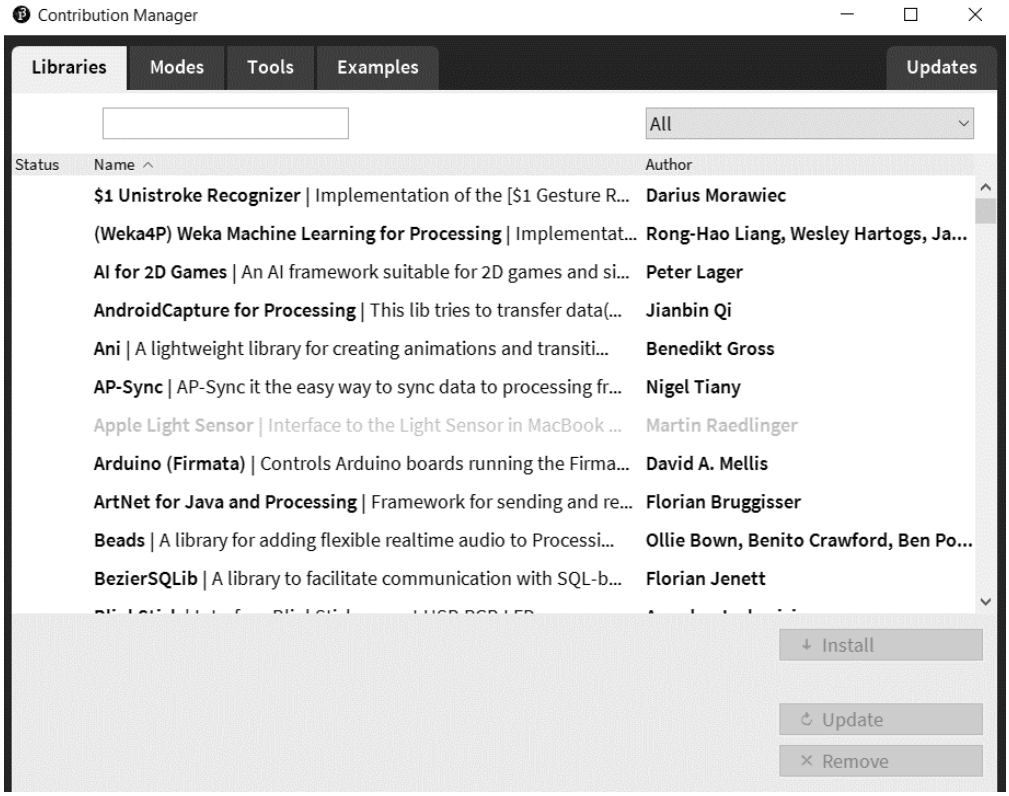


Figura 23. Ventana de búsqueda de librerías de Processing.

3.2.2. Librería SimpleOpenNI de Processing.

Para poder interpretar la información del Kinect se usa la librería SimpleOpenNI. Para implementarlo en el Processing se debe descargar directamente desde su github: "https://github.com/mqvlm/mqvlm.github.io/blob/master/code/simpleOpenNI/simpleOpenNI.pde". La carpeta descargada debe de ser colocada en la carpeta de librerías de Processing tal y como se muestra en la Figura 21.

3.2.3. Librería OpenCV de Processing.

Para poder detectar la abertura de la mano detectando el punto más brillante, se usa la librería OpenCV, arreglando el limitante que poseía la librería SimpleOpenNI. Para implementarlo en el Processing se lo debe buscar en el software Processing tal y como se muestra en la Figura 24.

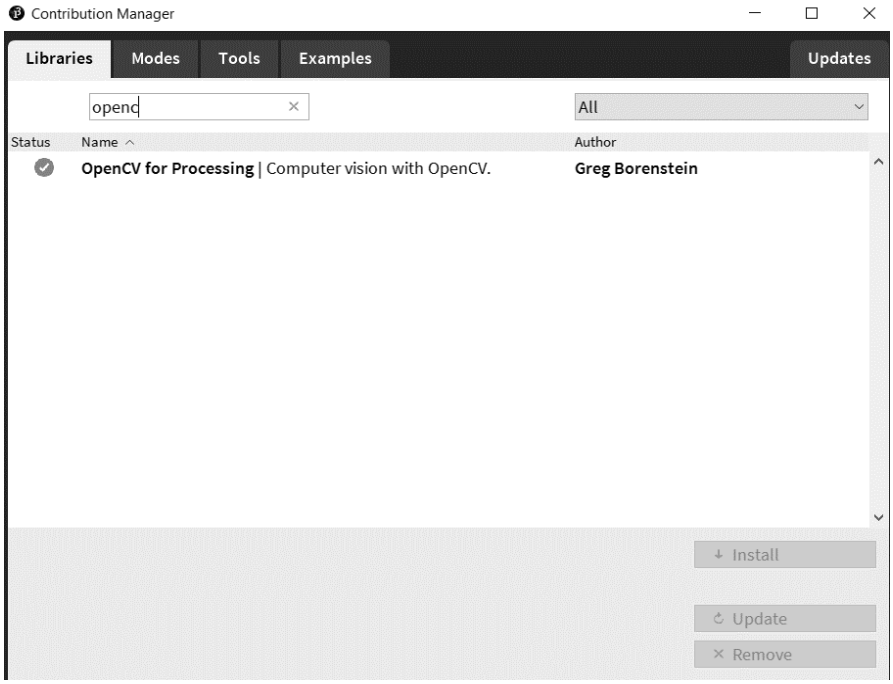


Figura 24. Ventana de búsqueda de la librería OpenCV de Processing.

3.2.4. Librería MQTT de Processing.

Esta librería permite conectarse a un bróker MQTT a través del mosquitto, permitiendo comunicar a varios equipos que estén conectados al mismo bróker. En la Figura 25 se aprecia como se encuentra disponible en el software Processing.

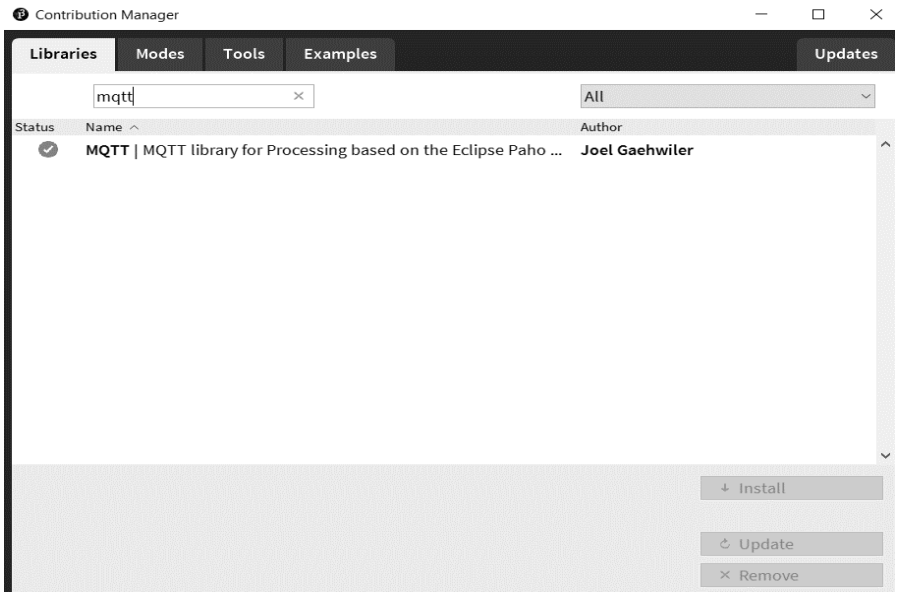


Figura 25. Ventana de búsqueda de la librería MQTT de Processing**.**

3.2.5. Mosquitto.

Mosquito es un protocolo de mensajería que permite comunicar con otros dispositivos a través de un bróker, ya sea público o privado. Para descargarlo se busca en la página oficial de mosquitto: “https://mosquitto.org/”, tal y como se aprecia en la Figura 26. En la sección de descargas se escoge lo que es requerido por el sistema operativo, en este caso, Windows 64 bits, la versión usada es la 2.0.13. En la Figura 27 se observa la ventana de descargas y el archivo que se descarga.

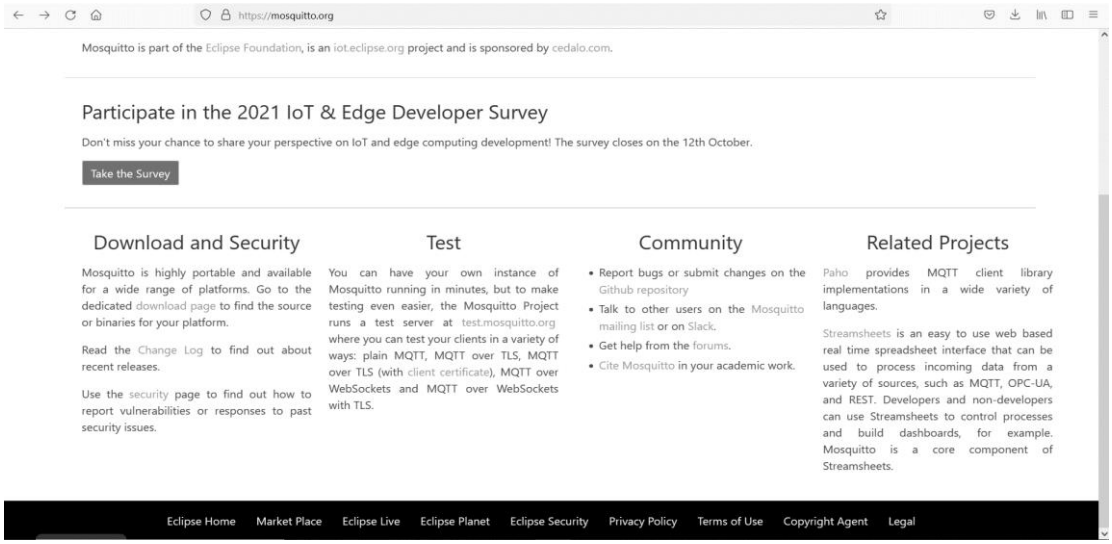


Figura 26. Página de Mosquitto***.***

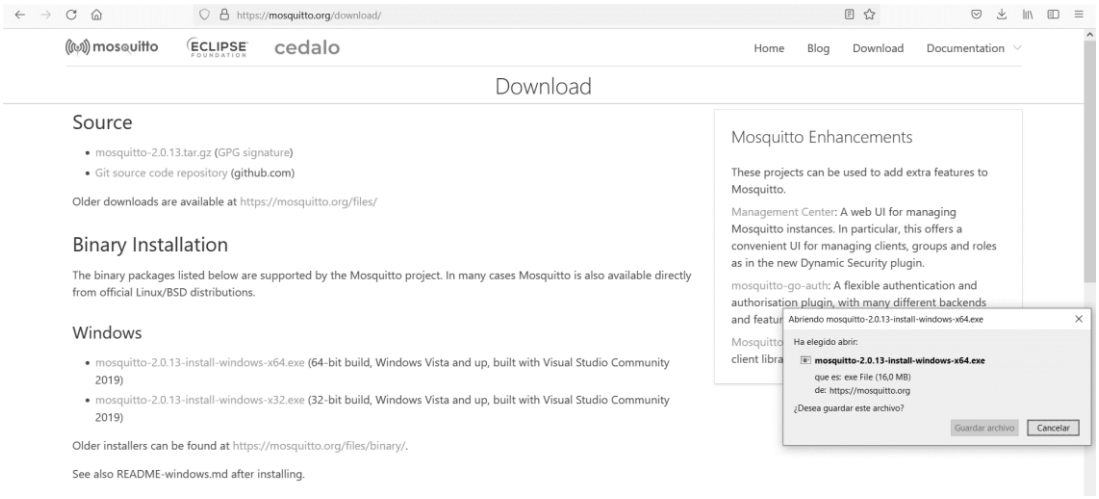


Figura 27. Página de descarga de Mosquitto.

El archivo descargado es un ejecutable el cual permite iniciar el proceso de instalación. En la ventana de instalación se procede a seleccionar “siguiente” hasta finalizar. En la Figura 28 se aprecia el proceso de instalación y la dirección en la que será ubicada dentro del ordenador.

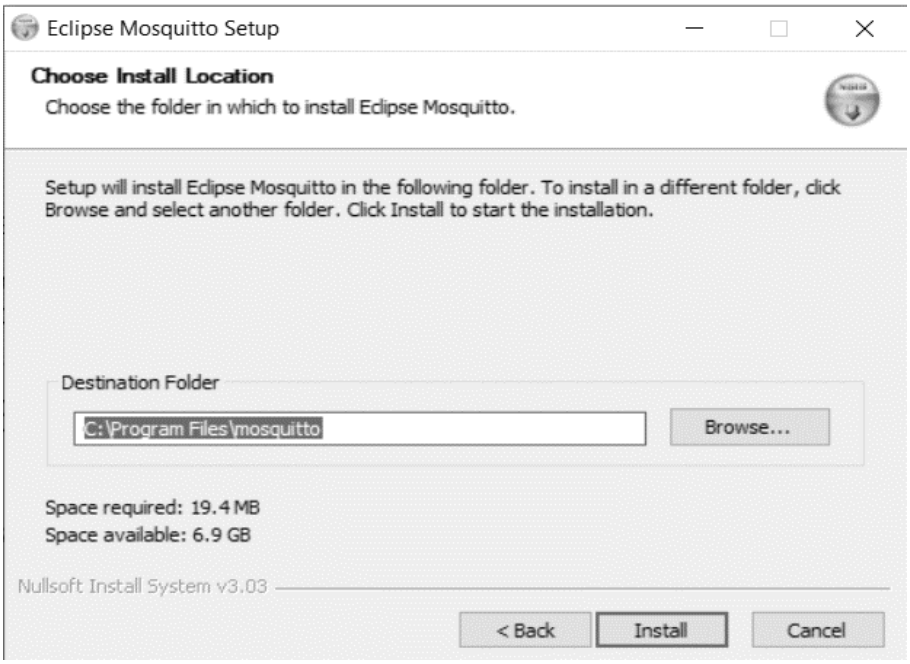


Figura 28. Ventana de instalación de Mosquitto.

Para verificar que se instala de manera correcta, se comprueba la carpeta mosquitto en la dirección en la que se instaló: “C:\Program Files\Mosquitto” tal como se muestra en la Figura 29.

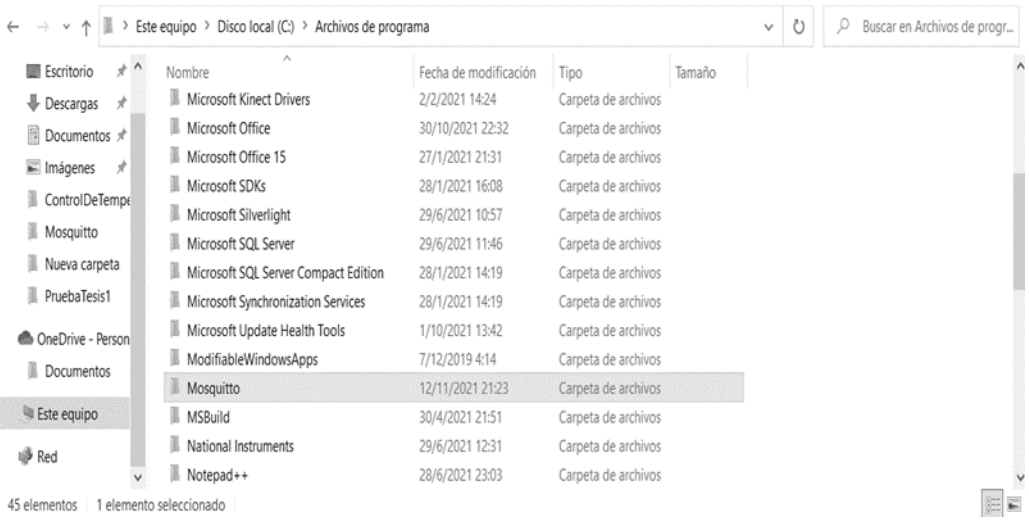


Figura 29. Ubicación de Mosquitto en el ordenador.

3.2.6. Drivers de Kinect.

Los siguientes drivers son necesarios para comunicar el sensor Kinect con el ordenador, mediante el adaptador Kinect a USB, sin estos, el ordenador no reconoce el Kinect como dispositivo.

3.2.6.1. SDK V18.

Para descargar el SDK (Software Development Kit) se busca en la página de Microsoft: “https://www.microsoft.com/en-us/download/details.aspx?id=40278”. En la Figura 30 se muestra la página de descarga.

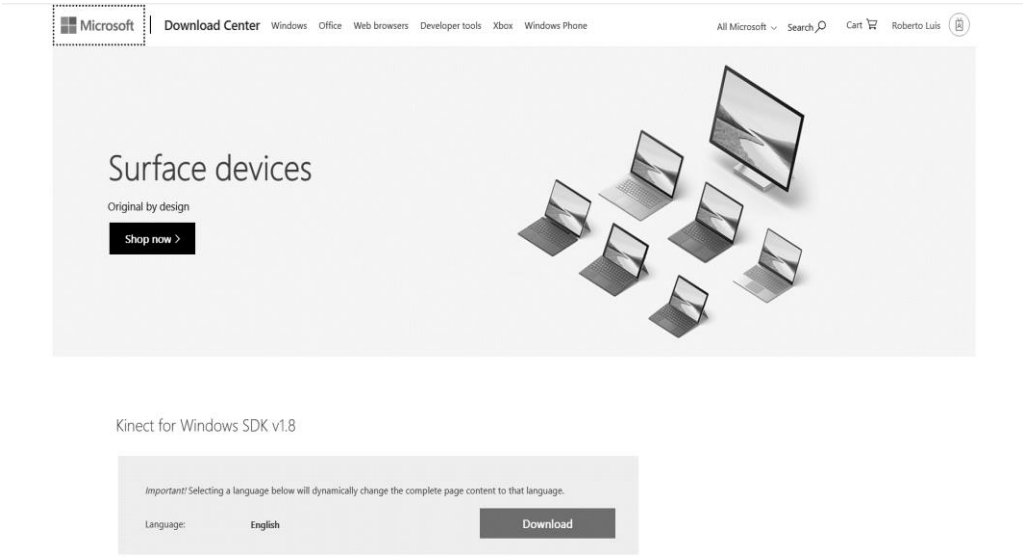


Figura 30. Página de descarga de SDK de Microsoft.

Finalizada la descarga se ejecuta el archivo y se procede con el proceso de instalación dando los respectivos permisos y se selecciona “Instalar”. En la Figura 31 se observa la ventana de instalación.

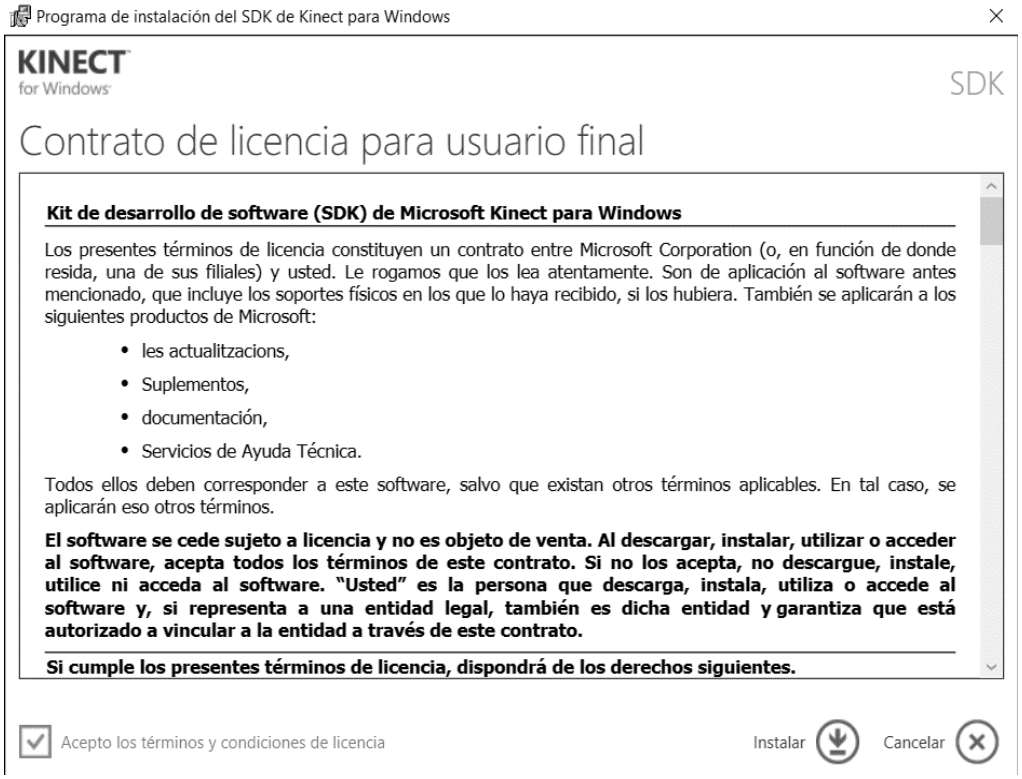


Figura 31. Ventana de instalación de SDK de Microsoft.

3.2.6.2. Developer Toolkit V18.

Para descargar el Developer Toolkit se busca en la página de Microsoft: “https://www.microsoft.com/en-us/download/details.aspx?id=40276”. En la Figura 32 se muestra la página de descarga.

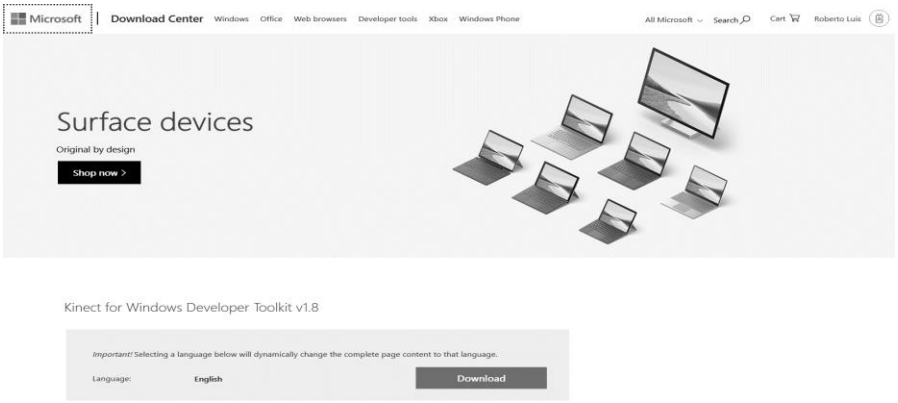


Figura 32. Página de descarga de developer toolkit de Microsoft.

Finalizada la descarga se ejecuta el archivo y se procede con el proceso de instalación dando los respectivos permisos y se selecciona “Instalar”. En la Figura 33 se observa la ventana de instalación.

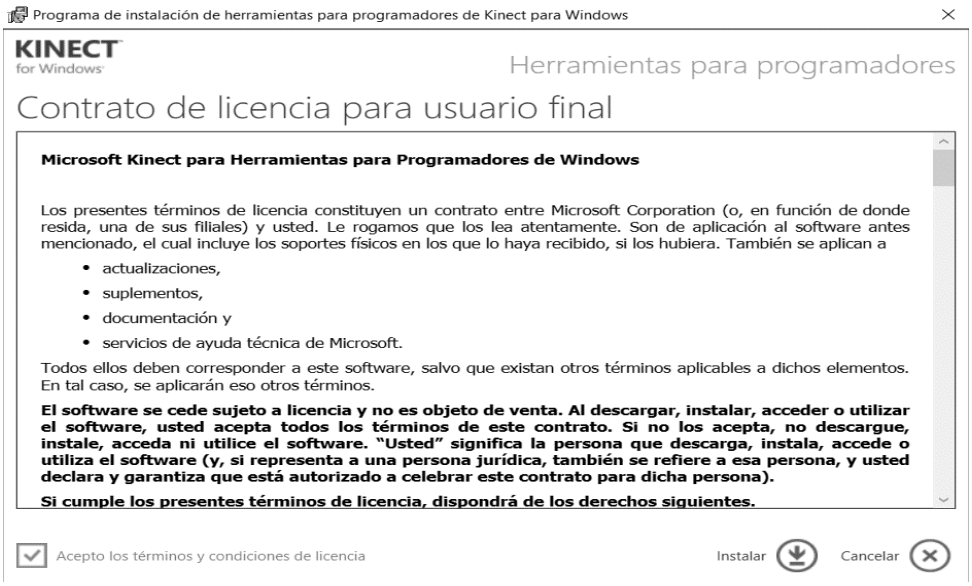


Figura 33. Ventana de instalación de developer toolkit de Microsoft.

3.2.6.3. Runtime V18.

Para descargar el Runtime se busca en la página de Microsoft: “https://www.microsoft.com/en-us/download/details.aspx?id=40277”. En la Figura 34 se muestra la página de descarga.

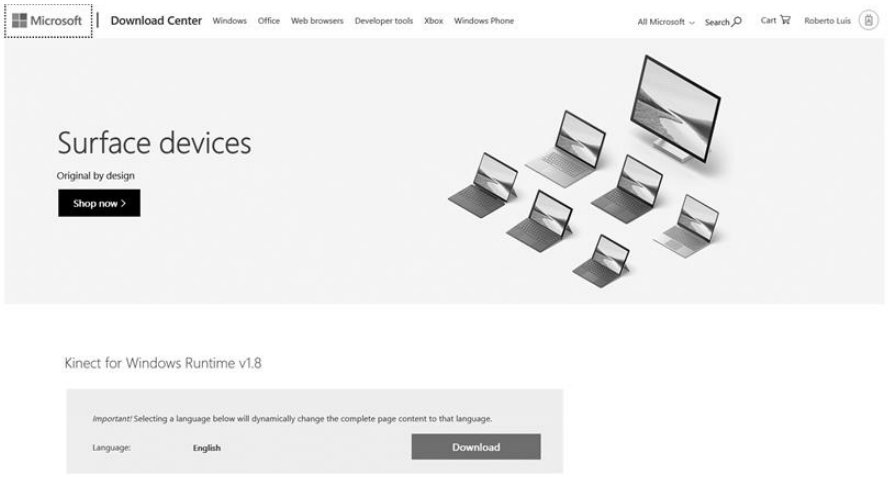


Figura 34. Página de descarga de runtime de Microsoft.

Finalizada la descarga se ejecuta el archivo y se procede con el proceso de instalación dando los respectivos permisos y se selecciona “Instalar”. En la Figura 35 se observa la ventana de instalación.

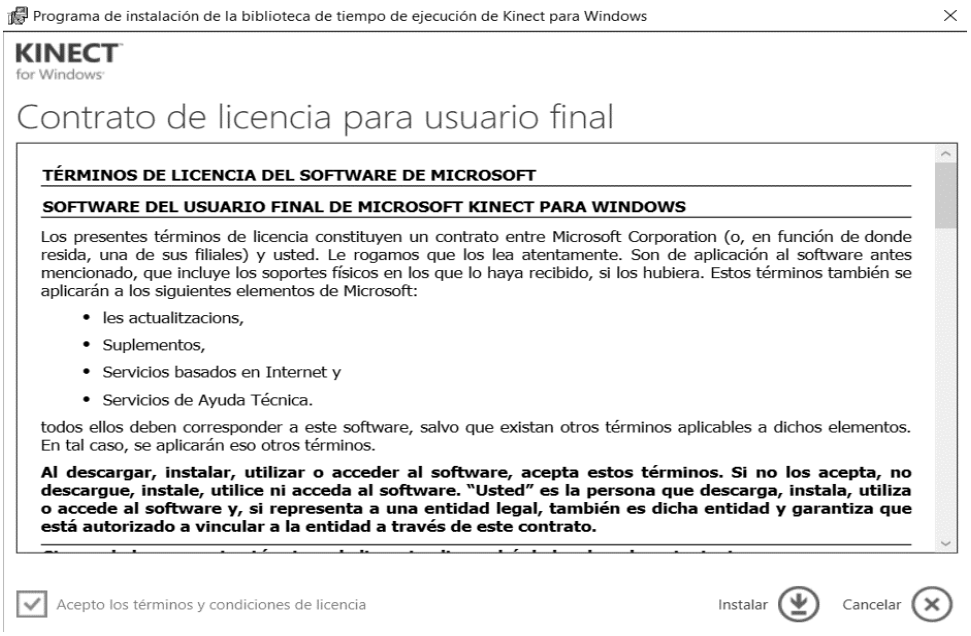


Figura 35. Ventana de instalación de runtime de Microsoft.

3.2.7. OpenNi.

Para descargar OpenNI se dirige a: “https://structure.io/openni”. En la Figura 36 se muestra la página de descarga.

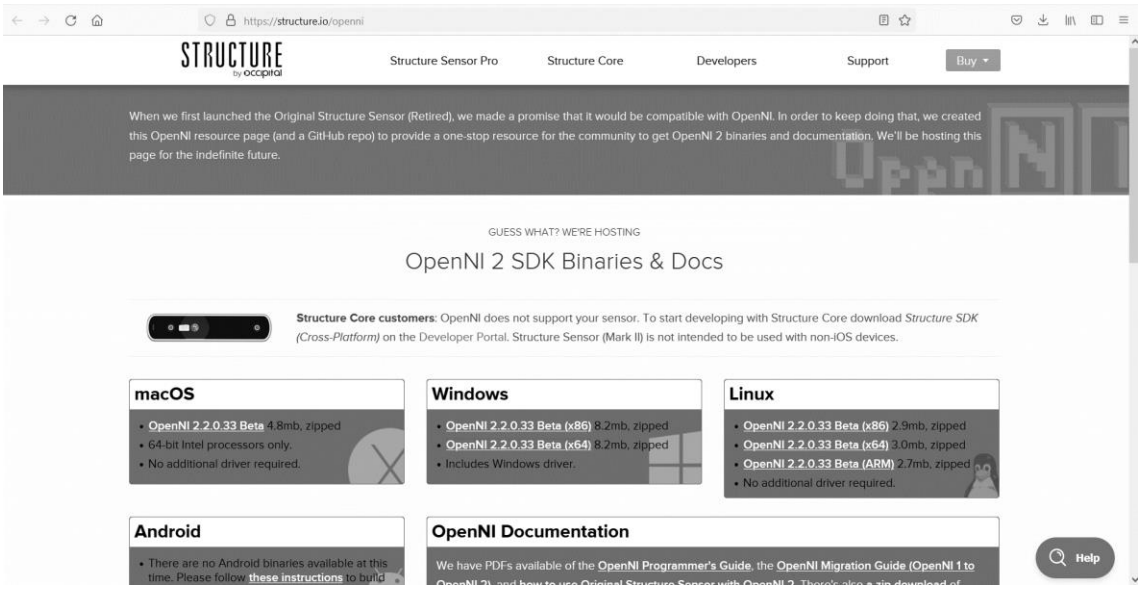


Figura 36. Página de descarga de OpenNI.

Finalizada la descarga se ejecuta el archivo y se continua con la instalación dando los respectivos “Install” y “Next” y se finaliza con “Finish”. En la Figura 37 y Figura 38 se muestra la ventana de instalación.

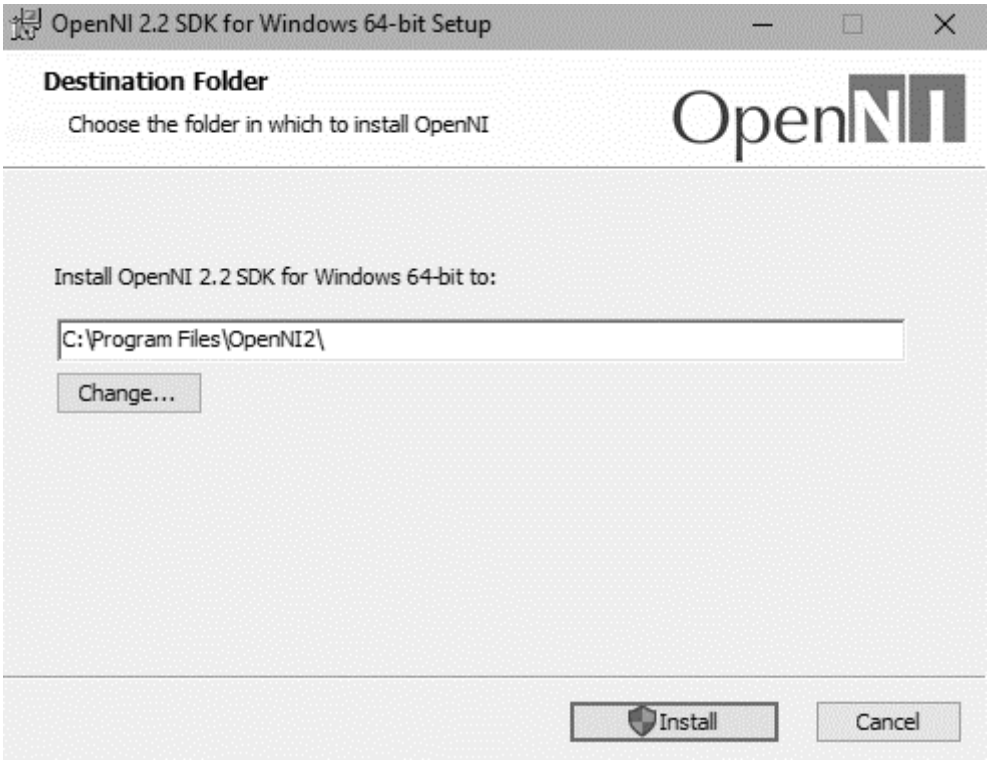


Figura 37. Ventana de instalación de OpenNI.

Figura 38. Ventana final de instalación de OpenNI.

3.2.8. Nite 2.2.

Para descargar NITE nos dirigimos a: “https://primesense-nite2-for-windows.software.informer.com/2.2b/”. En la Figura 39 se muestra la página de descarga.

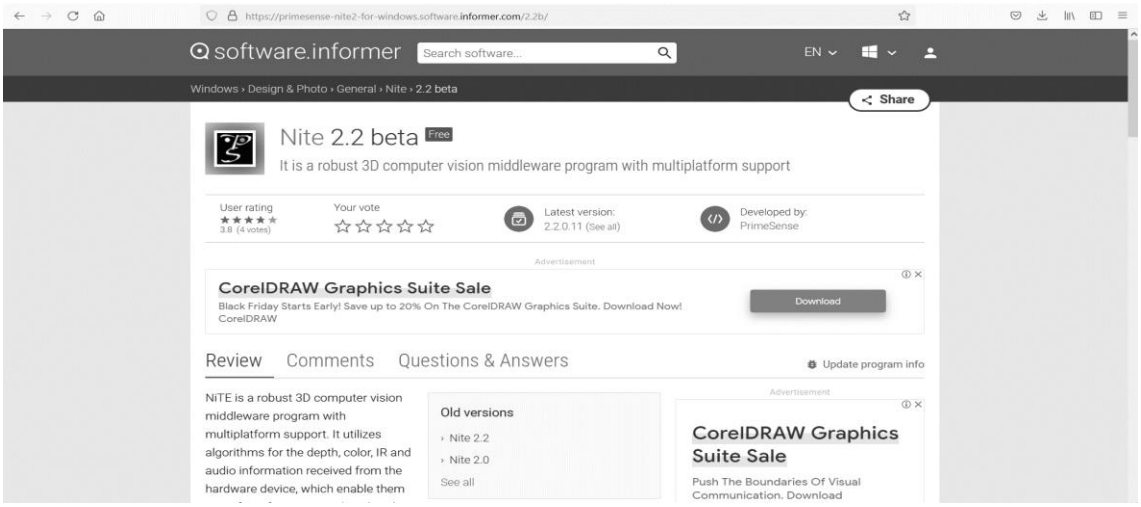


Figura 39. Página de descarga de NITE.

Finalizada la descarga se ejecuta el archivo y se continua con la instalación dando los respectivos “Next” y se finaliza con “Finish”. En la Figura 40 y Figura 41 se muestra la ventana de instalación.

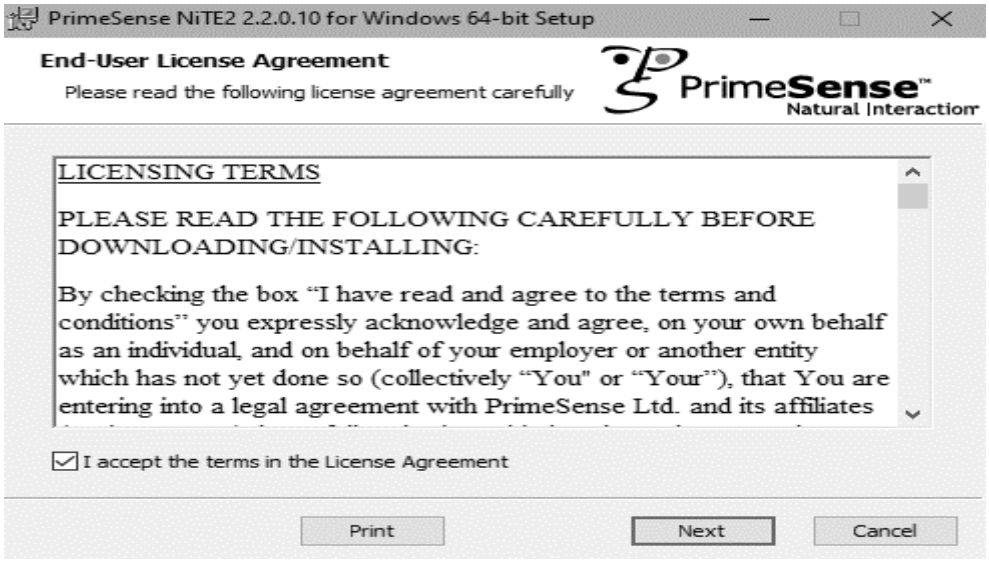


Figura 40. Ventana de instalación de OpenNI.

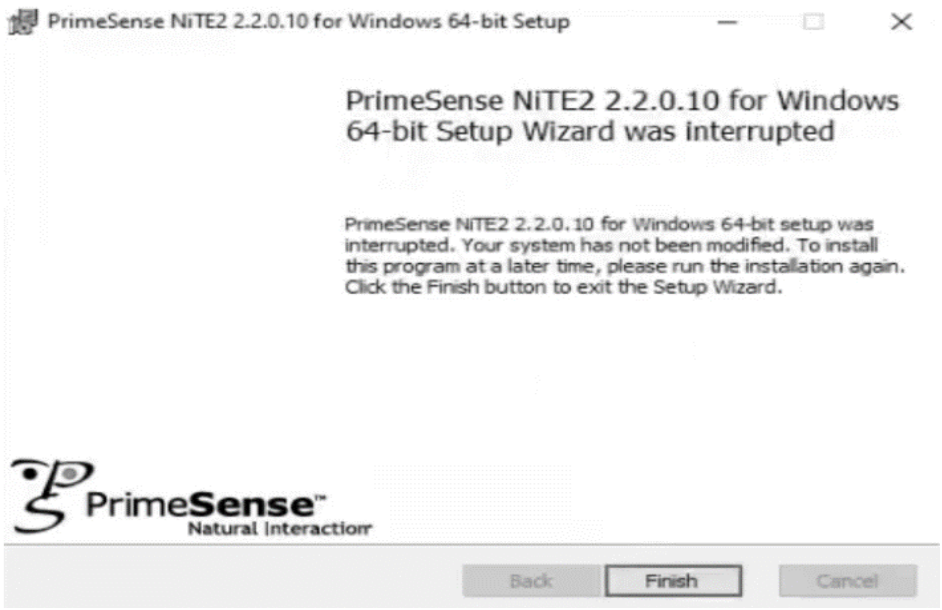


Figura 41. Ventana final de instalación de NITE.

3.2.9. Arduino.

Para descargar el software se busca en la página oficial de Arduino, “https://www.arduino.cc/en/software”, en las opciones de descargas se escoge la opción para el sistema operativo que se requiera, en este caso Windows 64-bit, tal y como se aprecia en Figura 42 y Figura 43.

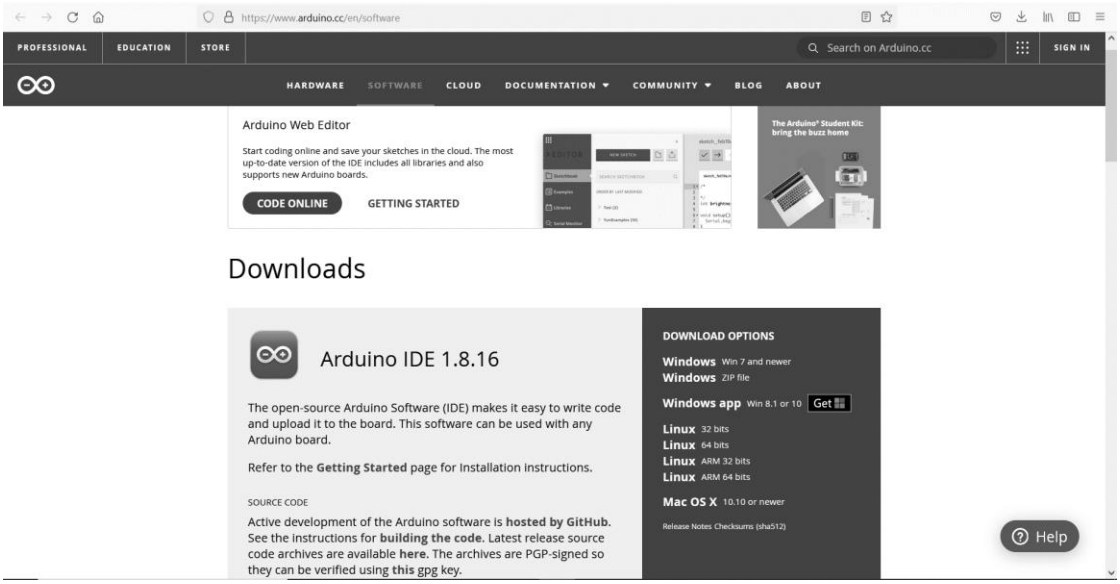


Figura 42. Página de Arduino.

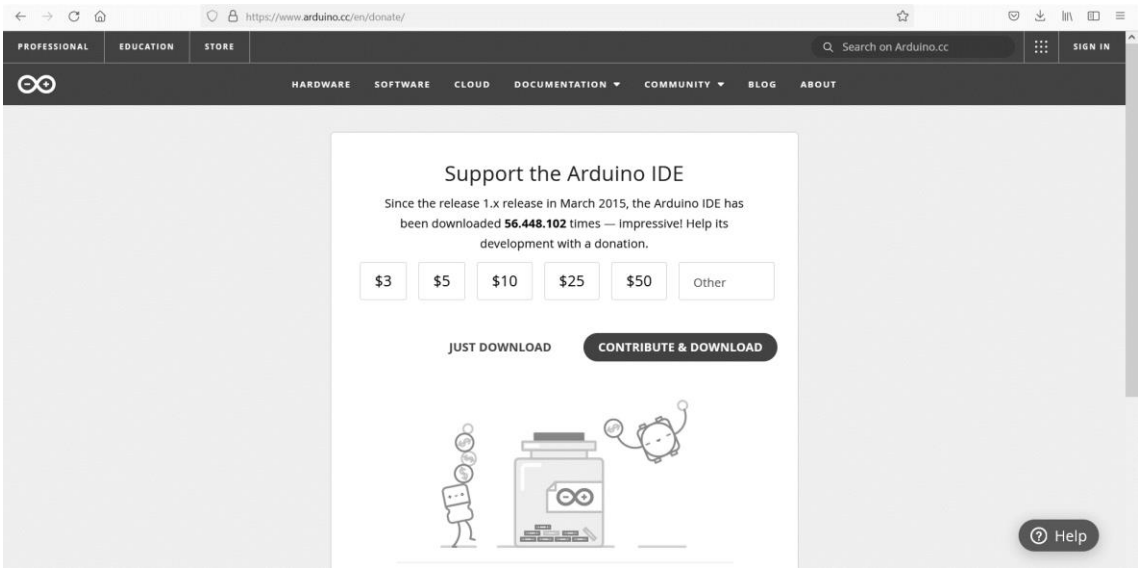


Figura 43. Página de descarga de Arduino.

Una vez descargado se procede a ejecutar el archivo para empezar a instalar. Se acepta los términos y condiciones del software y se selecciona “Next” en todas las ventanas emergentes, finalmente se escoge el lugar de instalación, en la Figura 44 y Figura 45 se observa el proceso de instalación.

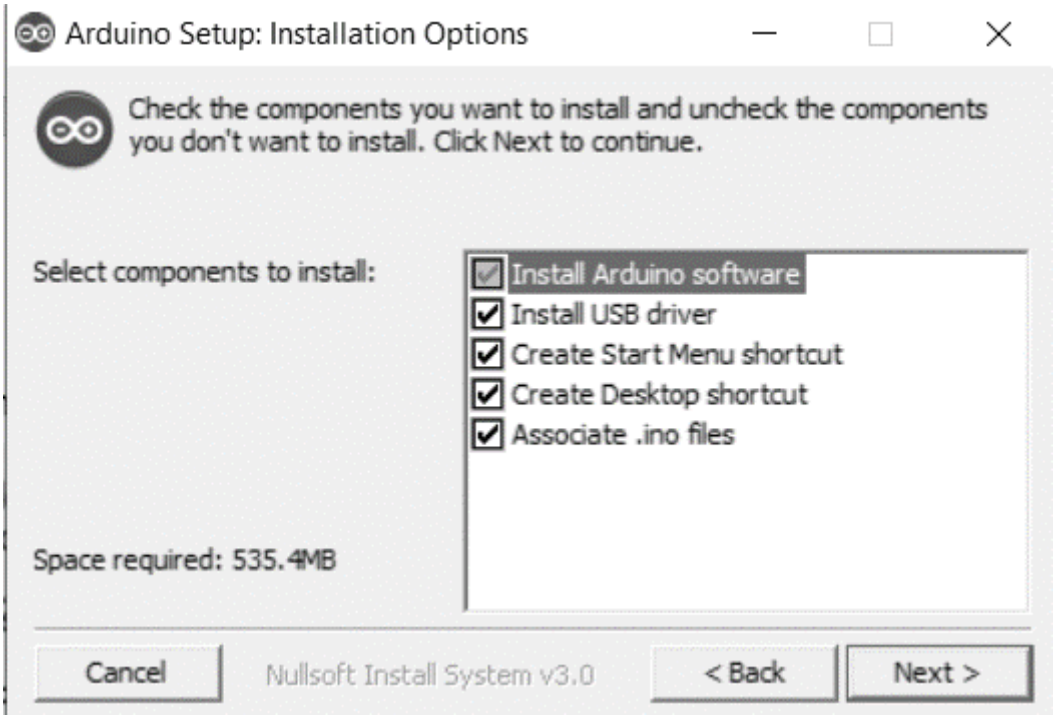


Figura 44. Ventana de instalación de Arduino.

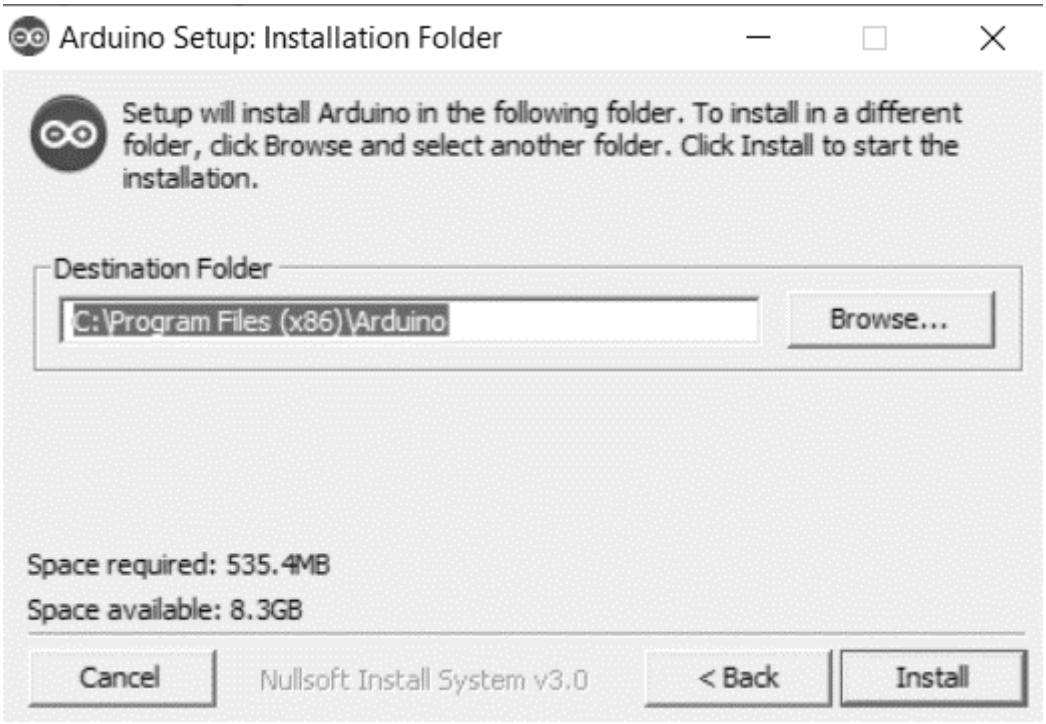


Figura 45. Ventana final de instalación de Arduino.

Una vez concluido se puede trabajar en el software, en la Figura 46 se aprecia la ventana del IDE de Arduino.

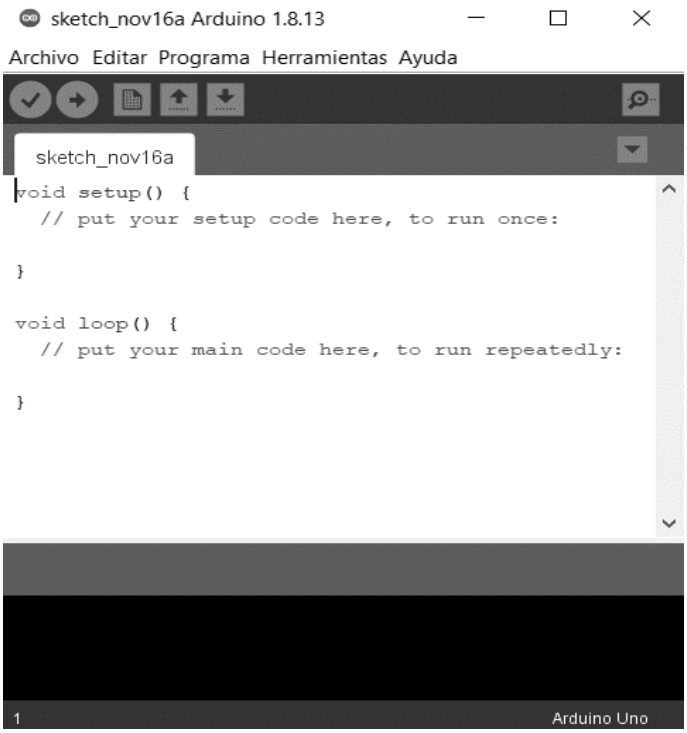


Figura 46. IDE de Arduino.

3.2.10. Librerías de Arduino

Las librerías instaladas por defecto con la instalación del IDE de Arduino las podemos observar en la siguiente dirección “C:\Program Files (x86)\Arduino\libraries” como se muestra en la Figura 47, mientras que las instaladas externamente en “C:\Users \Documents\Arduino\libraries” como se observa en Figura 48.

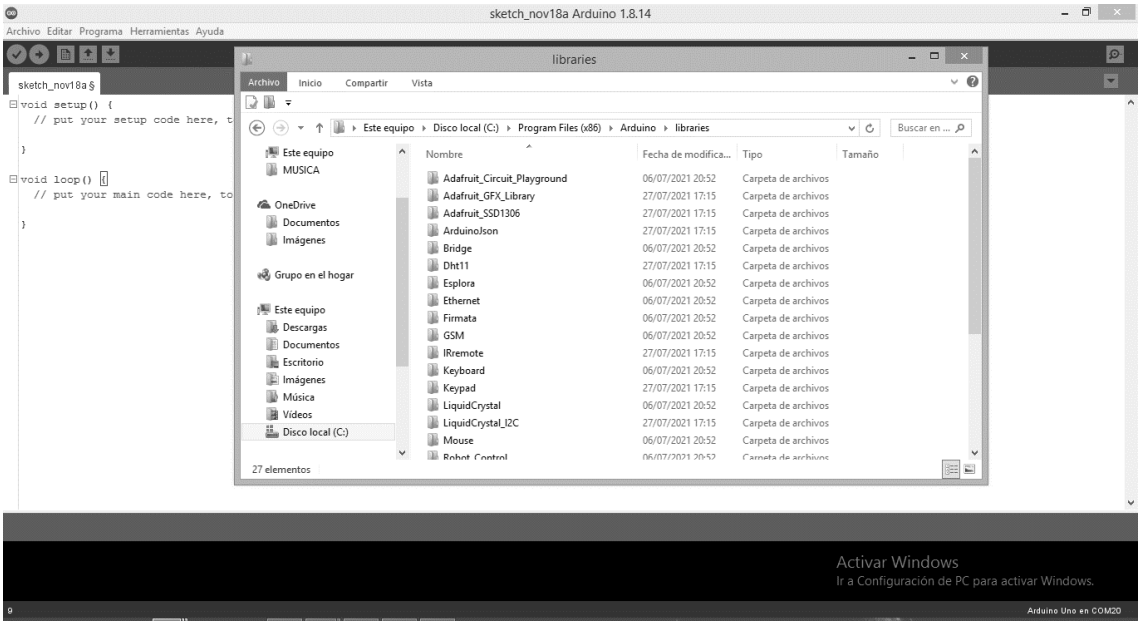


Figura 47. Ubicación de librerías instaladas con Arduino.

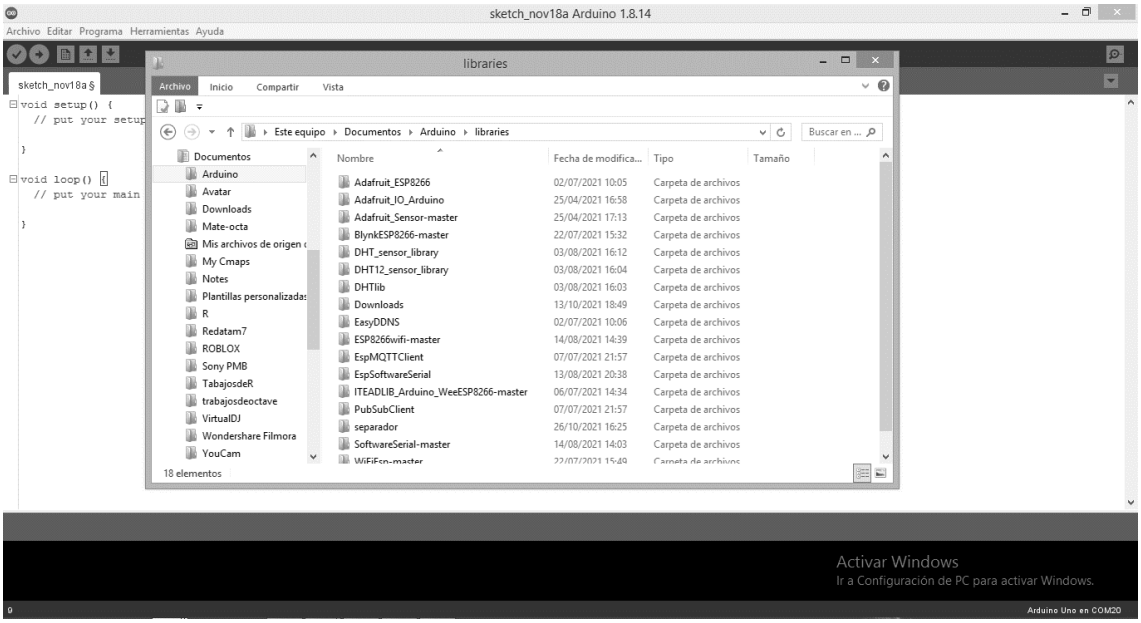


Figura 48. Ubicación de librerías de Arduino instaladas externamente.

Existen varias formas de instalar librerías de las cuales abordaremos 2, las más sencillas esto depende de si las librerías son de fuentes externas, son creadas por el fabricante.

La primera forma es descargando la librería de manera externa y añadirla en el IDE de Arduino, a través de “Programa/Añadir fichero”, tal y como se muestra en la Figura 49, después se selecciona la librería que se descargó previamente, en la Figura 50 se observa como es el proceso.

Figura 49. Añadir fichero en Arduino.

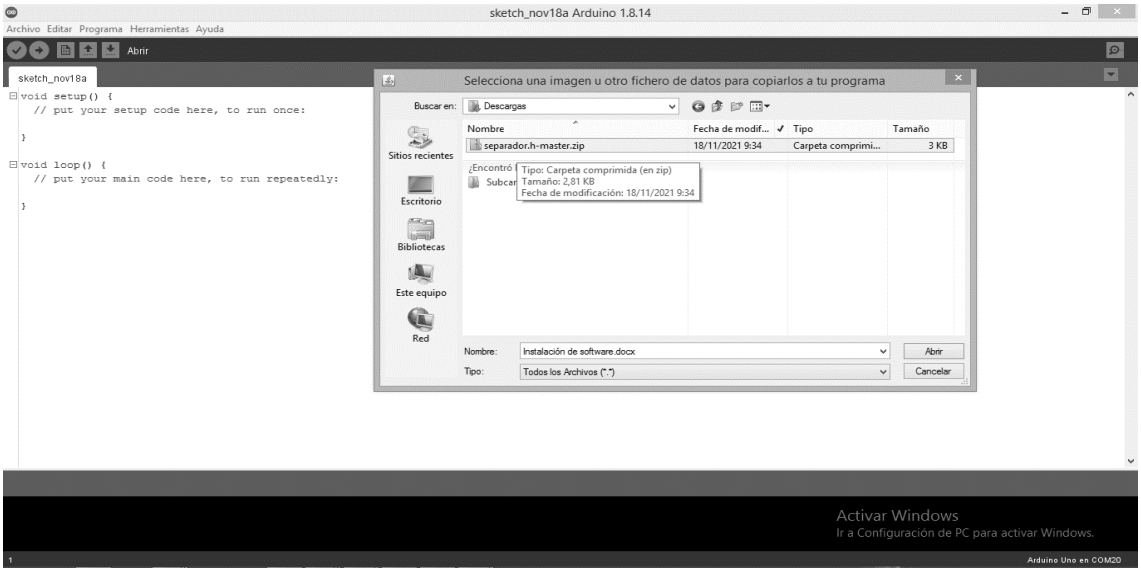


Figura 50. Fichero descargado para ser añadido.

La segunda forma es buscar la librería en el IDE de Arduino, mediante “Programa/Incluir librería/Administrar bibliotecas”, esto se visualiza en la Figura 51, inmediatamente se abre la venta del gestor de librerías, tal y como se muestra en la Figura 52, en donde se busca la librería que se desea implementar.

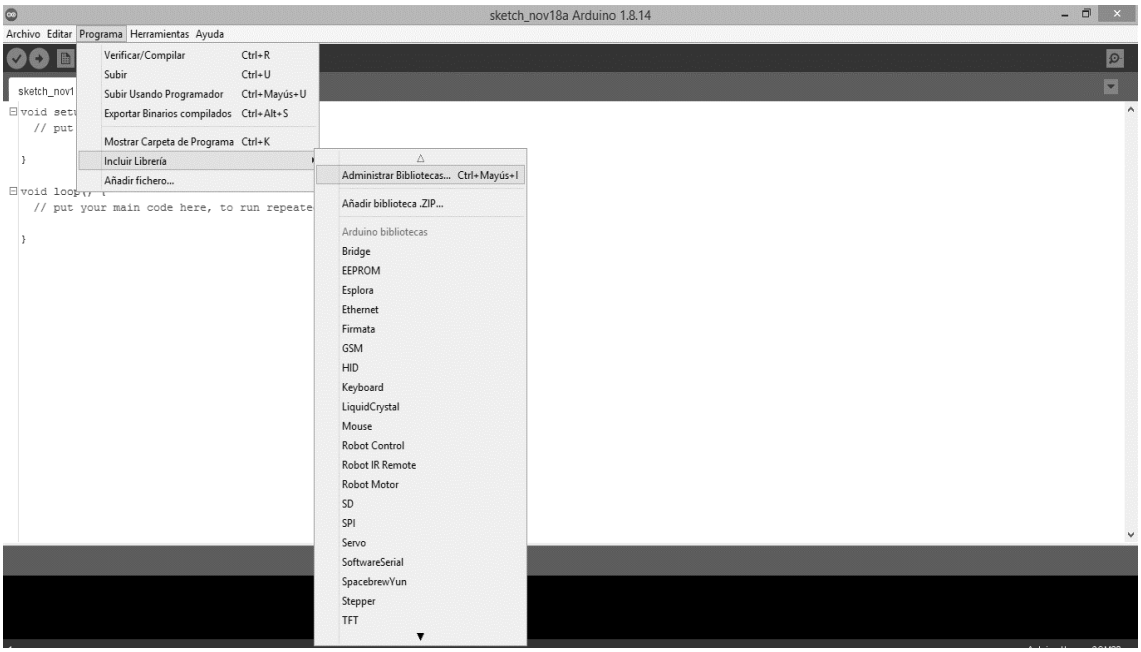


Figura 51. Abertura del gestor de librerías en Arduino.

Figura 52. Gestor de librerías en Arduino.

3.2.10.1. Esp8266 Wifi.

La conexión a una red wifi se puede realizar con la configuración de módulo ESP8266 y el Arduino, por esta razón necesitamos que se reconozca esta tarjeta. Esta librería se instala de la segunda forma expuesta, para incluir la librería ESP8266Wifi se necesitan configurar las preferencias del IDE Arduino para poder instalarla, lo cual se puede ver en la Figura 53.

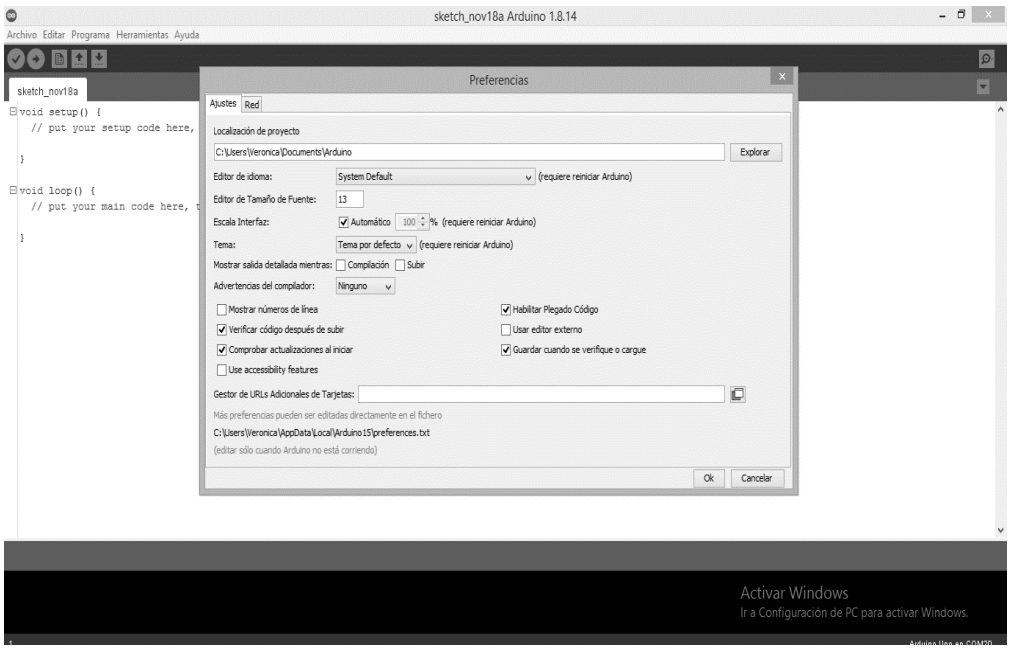


Figura 53. Gestor de librerías en Arduino.

En este punto se agrega este link en gestor de URLs adicionales de tarjetas “http://arduino.esp8266.com/stable/package_esp8266com_index.json” para que el Arduino pueda acceder a la librería ESP8266Wifi descargándola e instalándola directamente del IDE.

3.2.10.2. Adafruit_GFX.

Esta librería proporciona toda la clase central de donde derivan la mayoría de las librerías de gráficos utilizadas en pantallas OLED, la descarga de esta librería se puede realizar desde la página de github: “https://github.com/adafruit/Adafruit-GFX-Library” obteniendo el archivo Zip se la añade mediante la primera forma para añadirla el fichero a nuestro banco de librerías en el IDE de Arduino.

3.2.10.3. Adafruit_SSD1306.

La utilización de pantallas OLED monocromático con controladores SSD1306 requieren de esta librería para su funcionamiento, la cual se obtiene descargándola directamente de github: “https://github.com/adafruit/Adafruit_SSD1306” obteniendo el archivo Zip se la añade mediante la primera forma para añadirla el fichero a nuestro banco de librerías en el IDE de Arduino.

3.2.10.4. Separador.h.

Separa los datos que están delimitados por comas en distinto elementos para ser usados de forma independiente, se descarga directamente de la página github: “https://github.com/jams1416/separador.h/blob/master/README.md” obteniendo el archivo Zip se la añade mediante la primera forma para añadirla el fichero a nuestro banco de librerías en el IDE de Arduino.

3.3. Desarrollo de estación emisora.

En esta parte del proyecto se realiza la captura de los movimientos que efectúa la persona frente al sensor Kinect, los datos que son enviados al ordenador son las coordenadas del brazo que están constantemente cambiando según el movimiento de la persona, para conectar el sensor Kinect con el ordenador se usa un adaptador USB. Una vez se reciben los datos de las coordenadas, se realiza el análisis de esta mediante el software Processing, que haciendo uso de las librerías SimpleOpenNI y OpenCV, se puede calcular los ángulos de las articulaciones y con la librería MQTT se puede enviar la información hacia la estación receptora al bróker MQTT en el que estén conectados ambas estaciones. En la Figura 54 se observa un esquema del funcionamiento.

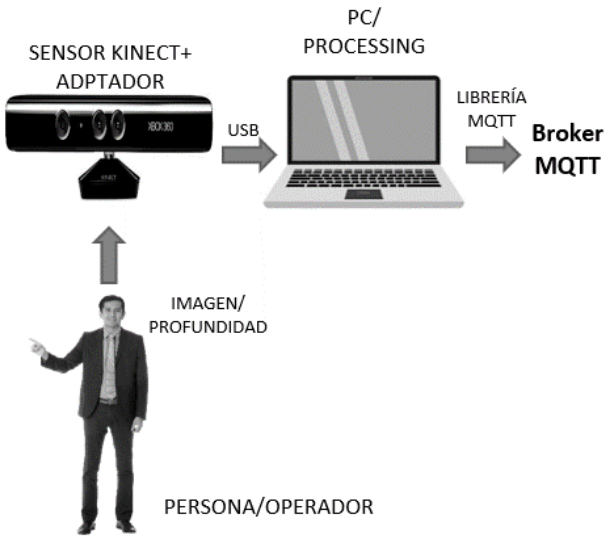


Figura 54. Esquema de funcionamiento de la estación emisora.

3.3.1. Estructura de la estación.

Esta estación está conformada por el ordenador, el sensor Kinect y el adaptador Kinect-USB. El adaptador posee 3 terminales, la primera es una alimentación de 100-240 voltios alternos a 0.3 A, la segunda en la entrada que se conecta al sensor Kinect y el ultimo es el terminal USB que se conecta al ordenador, en la Figura 55 se observa la ubicación de los elementos y su conexión.

Figura 55. Conexión de los elementos de la estación emisora.

El sensor Kinect debe de estar a una altura aproximada de 1.3 metros, debido a que podrá captar mejor la imagen y profundidad de la persona en frente. La persona se ubica a una distancia entre los 1.5 metros y 2.5 metros, para que las coordenadas de las articulaciones sean detectadas de manera correcta. En la Figura 56 se observa a la persona en frente del sensor Kinect y sus distancias respectivas.

Figura 56. Distancia entre sensor Kinect y una persona.

La librería de SimpleOpenNI posee una restricción al momento de detectar la articulación de la mano, es decir, no detecta su comportamiento. Para solucionar esto se decantó por fabricar un guante que se ilumine cuando se cierra la mano, de esta manera el sensor Kinect, mediante la librería OpenCV, detecta el punto más brillante cerca de la mano derecha, y así se determina la abertura y cierre del brazo robótico.

Para fabricar el guante se utiliza, leds de alta intensidad, resistencias de 330 ohm, botón y batería de 9 voltios. Los leds se conectan en serie con las respectivas resistencias de protección, y cada led con resistencia se conecta en paralelo, luego los terminales se conectan los respectivos terminales positivos y negativos de la batería, el terminal positivo de la batería se conecta en serie a un botón para que permita el paso de la corriente y alimente el circuito cuando se cierre, en la Figura 57 se observa el circuito de conexión de. En la Figura 58 y Figura 59 se aprecia el proceso de fabricación del guante.

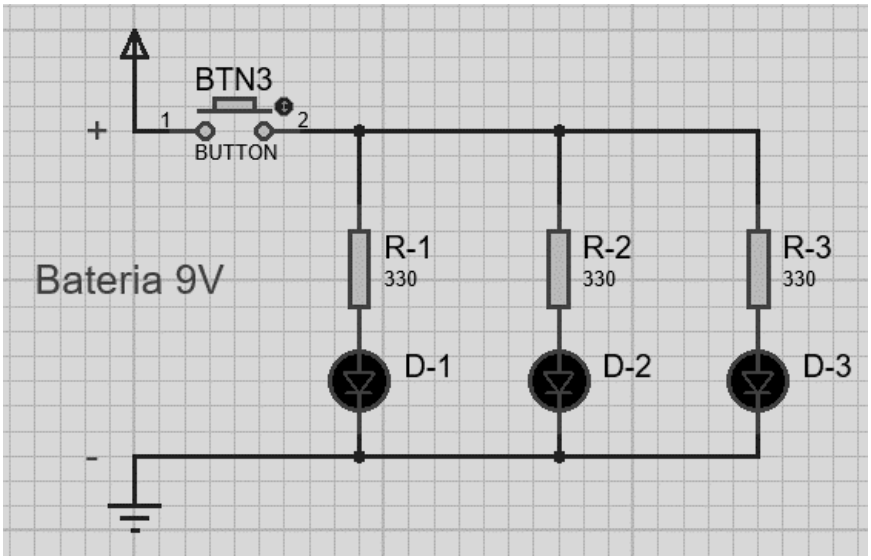


Figura 57. Circuito de guante de iluminación.

Figura 58. Proceso de pegado de circuito a guante de tela.

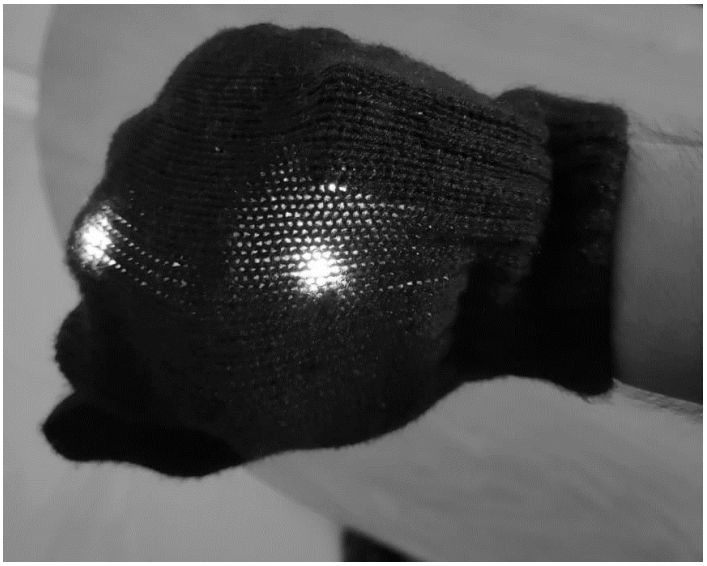


Figura 59. Guante de iluminación terminado.

3.3.2. Desarrollo de la programación.

Toda la programación de la estación emisora se desarrolló en el software Processing. Para empezar con la programación, se llama a las librerías que se usan, en este caso, MQTT, SimpleOpenNI y OpenCV. Para ello se usa la sentencia “import” más el nombre de la carpeta que contenga la librería, las librerías deberán estar dentro de la carpeta de librerías de Processing. Si se usa una carpeta dentro de otra, se las separa con”.”, si se quiere usar todo los elementos que contenga la carpeta se finaliza con “.*”. En la Figura 60 se muestra cómo se realizó la importación de librerías.

```
1 //Declaracion de librerias
2 import mqtt.*;//Importar libreria MQTT
3 import gab.opencv.*;//Importar libreria simple Opencv
4 import SimpleOpenNI.*;//Importar libreria simple OpenNi
```

Figura 60. Importación de librerías en Processing.

Se crean los objetos de las librerías para poder usar los métodos correspondientes a cada una de ellas. Se crean los objetos “client”, que es tipo MQTTclient, que sirve para realizar la conexión y envío de datos al bróker MQTT; “kinect”, que es tipo SimpleOpenNI, que permite utilizar la información recibida del sensor Kinect, y “cv”, que es tipo OpenCV; que sirve para analizar la imagen y detectar el punto más brillante. En la Figura 61 se aprecia la declaración de objetos para el proceso.

```
6 //Declaracion de objetos
7 MQTTClient client;//Creamos objeto client
8 SimpleOpenNI kinect;//Creamos objeto Kinect
9 OpenCV cv;//Creamos objeto openCV
```

Figura 61. Declaración de objetos en Processing.

Se declaran las variables globales que se usan en todo el programa. Los tipos de datos de las variables son “PFont”, para editar la fuente de la letra; “PImage”, para insertar imágenes en la ventana de procesos; “float”, para datos decimales, en este caso los ángulos calculados del brazo derecho, se los inicializa con el valor de “0”; “int”, para indicar valores enteros, en este caso se usan para asignar valores de “0” y “1” que indican si se está efectuando alguna acción en concreto, se los inicializa con el valor de “0”, y “String”, para datos de tipo carácter, que se usa para comunicar en letra el estado de una acción determinada. En la Figura 62 se aprecia las variables declaradas.

```
12 //Declaracion de variables
13 PFont fuente;//Fuente de letra
14 PImage logotipo;//Imagen de presentacion
15 float RightshoulderAngle=0;//Angulo de hombro derecho
16 float RightelbowAngle=0;//Angulo de codo derecho
17 float codo=0;//Angulo ajustado de codo derecho
18 float RightShoulderAngleHorizontal=0;//Angulo de Hombro2
19 float hombro=0;//Angulo ajustado de Hombro2 derecho
20 int hands=0;//Estado de mano
21 String hand;//Carater de mano
22 int conexion=0;//Determina cuando se conecta
23 String estado="Desconectado";//Caracter de conexion o desconexion
```

Figura 62. Declaración de variables en Processing.

Los programas en Processing se inicializan en el “void setup()”, que es en donde se ejecuta la primera acción del programa. En la Figura 63 se observa el “void setup()” desarrollado, aquí se establece el tamaño que tendrá la ventana de proceso, que es de 1280x680 pixeles; con “cv” se crea un nuevo objeto OpenCV y se indica su área de trabajo, que es de 640x480 pixeles; con “Kinect” se crea un nuevo objeto SimpleOpenNI, que habilita la generación de usuarios que formen esqueletos junto a las cámaras de color y profundidad, también se deshabilta la opción de espejo; con “client” se crea un nuevo objeto MQTTClient y se realiza la conexión con el bróker MQTT, en este caso se usa el bróker gratuito de mosquito “mqtt://test.mosquitto.org”, en la conexión se establece un nombre de usuario que debe ser único para evitar colisiones, el nombre de usuario a usar es "moreiravasquez261021"; se carga la imagen en la variable de tipo PImage “logotipo” con la función “loadImage()”, el argumento de la función es el nombre de la imagen “.JPG”, que debe de estar en la misma ubicación del archivo, y se carga la fuente en la variable de tipo PFont “fuente” con la función “createFont()”, el argumento de la función es el tipo y tamaño de fuente, Processing ya posee internamente varias fuentes, y para usar en el texto usa la función “textFont()”, cuyo argumento es la fuente recientemente cargada.

```
25 //Inicializar el programa
26 void setup() {
27   size(1280, 680);//Establecemos el tamaño de la pantalla
28   cv=new OpenCV(this,640,480);//Determinanos que el objeto cv es un objeto OpenCV y su area de trabajo
29   kinect = new SimpleOpenNI(this);//Determinanos que el objeto Kinect es un objeto SimpleOpenNI
30   kinect.enableDepth();//Habilita la camara de profundidad
31   kinect.enableRGB();//Habilita la camara de imagen a color
32   kinect.enableUser();//Habilita la generacion de articulaciones
33   kinect.setMirror(false);//Inhabilitamos la funcion de espejo
34   client = new MQTTClient(this);//Creamos un nuevo cliente para MQTT
35   client.connect("mqtt://test.mosquitto.org", "moreiravasquez261021");//Estableces el servidor al que se conectara y el nombre de usuario
36   logotipo=loadImage("Logo.JPG");//Cargamos la imagen del logotipo
37   fuente= createFont("Georgia", 32);//Creamos la fuente Georgia de tamaño 32 para su posterior uso
38   textFont(fuente);//Escogemos la fuente del texto que se mostrara
39 }
```

Figura 63. Inicialización del programa en el Void setup.

Después de inicializarse se procede a ejecutar repetidamente los procesos deseados, en Processing esto se realiza en el “void draw()”, que es en donde se ejecutan el resto de acciones del programa en bucle. En la Figura 64 se observa el “void draw()” desarrollado, aquí se establece el fondo de color azul con la función “background()”, su argumento es el código de color en RGB, en este caso es 0,0,255 del azul; con la función “rect()” se inserta un rectángulo sin bordes y de color amarillo mostaza en la zona inferior de la ventana; se inserta la imagen en la ventana usando la función “image()”, cuyos argumentos son, la variable “logotipo” previamente cargada, la ubicación en x,y y el tamaño que ocupara; se actualiza la imagen recibida del sensor Kinect en cada iteración con el método “update()” del objeto “kinect”; se analiza la imagen a color recibida por el sensor Kinect con el método “loadImage()” del objeto “cv” ; se inserta la imagen a color captada por el sensor Kinect en la posición (0,0) de la ventana, al repetirse las iteraciones se aprecia como un video; se crea “userList”, un nuevo arreglo de datos de tipo “IntVector”, en donde se guarda la cantidad de usuarios detectados; se detecta un nuevo usuario con el método “getUsers()” del objeto “kinect” y la cantidad se guarda en la variable “userList”; se establece el tamaño de letra en 40 con la función “textSize()”; se muestra la variable “estado”, en la posición (1035,475) de la ventana con la función “text()”; se crea la primera sentencia “if()”, en donde si existe solo un usuario detectado ejecute lo interno; se guarda el primer usuario de “userList” en la variable de tipo “entera “userId”, y se crea la segunda sentencia “if()”, que consultara si se inició con el tracking del esqueleto de la persona o “userId”, esto se realiza con el método “isTrackingSkeleton()” del objeto ”kinect”, cuando se cumple la condición se ejecuta las funciones internas:

- “conectar()”, determina el estado de conexión del ordenador con el bróker MQTT.
- “DibujarEsqueleto()”, dibuja el esqueleto y articulaciones sobre la imagen recibida por el sensor Kinect.
- “abertura()”, determina si la mano está abierta o cerrada dependiendo del punto más brillante detectado.
- “GradosDeLibertad()”, calcula los ángulos entre articulaciones del brazo derecho, los muestra en la ventana y los envía por medio de MQTT al bróker conectado.

```
//Funcion de trabajo
void draw() {
  background(0,0,255);//Fondo azul
  noStroke();//Sin borde
  fill(255,219,88);//Color amarillo mostaza
  rect(0,480,1280,200);//Rectangulo
  image(logotipo,640,0,640,240);//Insertamos una imagen de presentacion del proyecto
  kinect.update();//Actualiza las imagenes del Kinect en cada iteraciòn.
  cv.loadImage(kinect.rgbImage());//OpenCv analiza la imagen a color proporcionada por el Kinect
  image(kinect.rgbImage(), 0, 0);//Dibujar la imagen que percibe el sensor Kinect
  IntVector userList = new IntVector();//Creo una lista para usuarios
  kinect.getUsers(userList);//Determino el usuario por la cantidad de lista
  textSize(40);//Tamaño de texto 40
  fill(255);//color blanco
  text(estado,1030,475);//Muestra el estado de conexion
    if (userList.size() == 1) {//Cuando se detecta al menos 1 usuario
      int userId = userList.get(0);//Guarda el numero de usuario
      if (kinect.isTrackingSkeleton(userId)&&conexion==1) {//Cuando detecta a una persona=TRUE
        conectar(userId);
        DibujarEsqueleto(userId);//Uso drawSkeleton para dibujar el esqueleto del usuario
        abertura(userId);//Uso del punto mas brillante para determinar si la mano esta abierta o cerrada
        GradosDeLibertad(userId);//Uso ArmsAngle para determinar los angulos de las articulaciones
      }
    }
}
```

Figura 64. Ejecución del programa en bucle en el Void draw.

La librería de SimpleOpenNI posee funciones internas que se ejecutan al detectar o perder un nuevo usuario, para poder editarlas basta con declarar la función y añadir el proceso deseado. En la Figura 65 se muestra las funciones de detección de usuario. La primera función es la de detección de un nuevo usuario, “onNewUser()”, cuyo argumento es un objeto de tipo SimpleOpenNI, y una variable tipo entero que indica el número de usuario, dentro de la función ordena que envíe un mensaje de “Start”, asigna un valor de 1 a la variable “conexión”, asigna el string “Conectado” a la variable “estado”, y se inicia el análisis del esqueleto del usuario “userId” con el método “startTrackingSkeleton()”, indicando que se detectó un usuario. La segunda función es la de pérdida de usuario, “onLostUser()”, cuyo argumento es una variable tipo entero que indica el número de usuario, dentro de la función ordena que asigne un valor de 0 a la variable “conexión”, asigna el string “Desconectado” a la variable “estado”, y se envía un mensaje que se perdió el usuario “userId.

```
//Funcion de deteccion de usuario
void onNewUser(SimpleOpenNI kinect, int userID) {
  println("START");
  conexion=1;
  estado="Conectado";
  kinect.startTrackingSkeleton(userID);//Inicio de trackeo de esqueleto
}

//Funcion de perdida de usuario
void onLostUser (int userId){
 conexion=0;
 estado="Desconectado";
 println("Usuario: "+ userId); //Mensaje de usuario perdido
}
```

Figura 65. Funciones internas de detección de usuario.

La librería de mqtt posee funciones internas que se ejecutan al conectarse, desconectarse o recibir un mensaje, para poder editarlas basta con declarar la función y añadir el proceso deseado. En la Figura 66 se muestra las funciones de conexión con el bróker MQTT. La primera función es “clientConnected()”, esta función se ejecuta cuando se establece conexión con el bróker MQTT, dentro de la función ordena que envíe un mensaje de “Conectado a MQTT”. La segunda función es “messageReceived()”, cuyos argumentos son una variable tipo string que indica el topic hacia donde envía el mensaje y una variable de tipo byte que es el mensaje recibido. La tercera función es “connectionLost()”, esta función se ejecuta cuando se pierde la conexión con el bróker MQTT, dentro de la función ordena que envíe un mensaje de “Desconectado a MQTT”.

```
//Funcion de conexion de cliente
void clientConnected() {
  println("Conectado a MQTT");//Mostramos el mensaje cuando se halla conectado
}

//Funcion de mensaje recibido
void messageReceived(String topic, byte[] payload) {
  println("new message: " + topic + " - " + new String(payload));
}

//Funcion de conexion perdida
void connectionLost() {
  println("Desconectado a MQTT");//Se muestra este mensaje cuando se pierde la conexion
}
```

Figura 66. Funciones internas de conexión con el broker MQTT.

En la Figura 67 se observa la estructura de la función “conectar”. Esta función permite desconectar la estación emisora del broker mqtt para que el brazo robótico ya no efectúe movimiento. La acción se realiza cuando se detecta que la mano izquierda se encuentra en un lugar determinado. El algoritmo de la función consiste en crear 2 “PVector”, uno en donde se receptan las coordenadas detectadas por el Kinect y otra en donde se guardan las coordenadas que puedan ser usadas en el programa. Se detecta las coordenadas usando el método “getJointPositionSkeleton()” del objeto ”kinect”, cuyos argumentos son: el usuario, la articulación que desea detectar y la variable “PVector” donde se desea guardar. Para que estas coordenadas sean escaladas y llevadas a un plano en el que sea más fácil de interpretar se usa el método “convertRealWorldToProjective()” del objeto ”kinect”, cuyos argumentos son: la variable “PVector” detectada por el kinect y la variable “PVector” donde se desea guardar. Una vez detectada se establece si la coordenada de la mano izquierda se encuentra en una posición establecida, en este caso, la esquina superior izquierda. En caso de cumplir se ordena que se asigne un valor de 0 a la variable “conexión”, asigna el string “Desconectado” a la variable “estado”.

```
129 //Funcion conexion-desconexion
130 void conectar(int userId) {
131   PVector leftHand = new PVector();//Generamos el vector de mano izquierda
132   kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, leftHand);//Guardamos la
133   //posicion de la articulacion en el vector creado
134   PVector manoIzq = new PVector();//Creamos el vector de coordenadas
135   kinect.convertRealWorldToProjective(leftHand, manoIzq);//Determina el vector posicion con
136   //coordenadas que podamos usar
137   if(manoIzq.x<80 && manoIzq.y<80){//Si la ubicacion de la mano izquierda se encuentra en una
138   //zona determinada realiza lo siguiente
139     conexion=0;//mandamos a desconectar
140     estado="Desconectado";//mandamos a desconectar
141   }
142 }
```

Figura 67. Función conectar().

En la Figura 68 se muestra la función “DibujarArticulacion”, que permite dibujar con un círculo rojo sobre las articulaciones detectadas por el sensor Kinect, sus argumentos son el usuario y la articulación que se desea representar. El algoritmo de la función consiste en crear en crear 2 “PVector”, uno en donde se receptan las coordenadas detectadas por el Kinect y otra en donde se guardan las coordenadas que puedan ser usadas en el programa. Se detecta las coordenadas usando el método “getJointPositionSkeleton()” del objeto ”kinect”, cuyos argumentos son: el usuario, la articulación que desea detectar y la variable “PVector” donde se desea guardar. Este método también retorna una variable de tipo “float” que indica la claridad con la que se detecta la articulación, esta debe ser como mínimo el valor de “0.5” para ser representada. Para que estas coordenadas sean escaladas y llevadas a un plano en el que sea más fácil de interpretar se usa el método “convertRealWorldToProjective()” del objeto ”kinect”, cuyos argumentos son: la variable “PVector” detectada por el kinect y la variable “PVector” donde se desea guardar. Finalmente se dibuja un círculo de radio 5 en las coordenadas en “X” y “Y” de la articulación detectada.

```
151 //Funcion dibujar puntos de articulacion(Convierte coordenadas del Kinect
152 //a coordenadas de la pantalla)
153 void DibujarArticulacion(int userId, int jointID) {//Recibe la articulacion que
154 //se desea graficar la coordenada
155   PVector joint = new PVector();//Creamos nuevo vector joint
156   float confidence = kinect.getJointPositionSkeleton(userId, jointID, joint);//Guardamos
157   //la ubicacion en una variable
158   if (confidence < 0.5) {//Si es menor a 0.5 no grafica la articulacion
159     return;
160   }
161   PVector convertedJoint = new PVector();//Creamos el vector de coordenadas
162   kinect.convertRealWorldToProjective(joint, convertedJoint);//Determina el vector posicion
163   //con coordenadas que podamos usar
164   ellipse(convertedJoint.x, convertedJoint.y, 5, 5);//Dibuja un circulo en la posicion dada
165 }
```

Figura 68. Función DibujarArticulación().

En la Figura 69 se muestra la función “DibujarEsqueleto”, que permite dibujar con una línea negra las extremidades entre las articulaciones detectadas por el sensor Kinect, sus argumentos son el usuario. El algoritmo de la función consiste en crear en crear 2 “PVector”, uno en donde se receptan las coordenadas detectadas por el Kinect y otra en donde se guardan las coordenadas que puedan ser usadas en el programa. Se traza una línea entre las coordenadas de las articulaciones usando el método “drawLimb()” del objeto ”kinect”, cuyos argumentos son: el usuario, y las 2 articulaciones que se deseen unir. En este caso se representa las extremidades comprendidas entre el hombro, codo y mano derecha. Se da color al trazo con “stroke()”, para color negro el argumento es “0”. Se da grosor de tamaño 5 al trazo con “strokeWeight()”. Después de trazar las extremidades, se cambia el color de trazo a rojo con “fill()”, para proceder a usar la función “DibujarArticulacion”, y dibujar las articulaciones del hombro, codo y mano derecha.

```
160 //Funcion dibujar esqueleto
161 void DibujarEsqueleto(int userId) {
162   stroke(0);//Color de trazo negro
163   strokeWeight(5);//De tamaño 5
164   //drawLimb permite dibujar lineas entre 2 puntos(funcion que usa las coordenadas propias
165   //del SimpleOpenNI)
166   //entre hombro derecho y codo derecho
167   kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER, SimpleOpenNI.SKEL_RIGHT_ELBOW);
168   //entre codo derecho y mano derecha
169   kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW, SimpleOpenNI.SKEL_RIGHT_HAND);
170   noStroke();//Quitar color de trazo
171   fill(255, 0, 0);//Rellenamos de color rojo el trazo
172   //Usamos la funcion drawJoint para dibujar los puntos de articulaciones
173   DibujarArticulacion(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER);//Hombro derecho
174   DibujarArticulacion(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW);//Codo derecho
175   DibujarArticulacion(userId, SimpleOpenNI.SKEL_RIGHT_HAND);//Mano derecha
176 }
```

Figura 69. Función DibujarEsqueleto().

En la Figura 70, se observa la función “distancia()”, que retorna el cálculo de la distancia entre 2 puntos en 2D, haciendo uso de la ecuación 2. Sus argumentos son 2 “PVector” que guardan las coordenadas del punto más brillante y de la mano derecha.

```
122 //Funcion Distancia entre dedos y mano
123 float distancia(PVector luz, PVector mano){
124   //Calculamos la distancia con la formula de distancia entre puntos en 3D
125   return sqrt(sq(luz.x-mano.x)+sq(luz.y-mano.y));//retornamos la distancia entre el dedo y la mano
126 }
```

Figura 70. Función distancia().

En la Figura 71 se muestra la función “abertura()”. Esta función permite controlar la abertura de la pinza del brazo robótico, detectando si la distancia entre el punto más brillante, proveniente del guante, y la mano derecha es cercana . El algoritmo de la función consiste en crear 3 “PVector”, uno en donde se receptan las coordenadas detectadas por el Kinect, otra en donde se guardan las coordenadas que puedan ser usadas en el programa y una donde se guardan las coordenadas del punto más brillante. Se detecta las coordenadas usando el método “getJointPositionSkeleton()” del objeto ”kinect”, cuyos argumentos son: el usuario, la articulación que desea detectar y la variable “PVector” donde se desea guardar. Para que estas coordenadas sean escaladas y llevadas a un plano en el que sea más fácil de interpretar se usa el método “convertRealWorldToProjective()” del objeto ”kinect”, cuyos argumentos son: la variable “PVector” detectada por el Kinect y la variable “PVector” donde se desea guardar. Se detecta el punto más brillante con el método “max()” del objeto “cv”. Una vez detectada las coordenadas se calcula la distancia entre ambas con la función “distancia()” y se lo guarda en una variable de tipo “float”. Si la distancia es menor a 50 se asigna un valor de “0” a la variable “hands”, y asigna el string “Close” a la variable “hand”, caso contrario se asigna un valor de “1” a la variable “hands”, y asigna el string “Open” a la variable “hand”. Para percibir el punto más brillante detectado se dibuja un círculo de radio 20, color rojo, grosor de trazo 4 y sin relleno.

```
97  //Funcion de abertura de mano
98  public void abertura(int userId){
99    PVector rightHand = new PVector();//Generamos el vector de mano izquierda
100   kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, rightHand);//Guardamos
101   //la posicion de la articulacion en el vector creado
102   PVector manoDer = new PVector();//Creamos el vector de coordenadas
103   kinect.convertRealWorldToProjective(rightHand, manoDer);//Determina el vector posicion con
104   //coordenadas que podamos usar
105
106   PVector luz = cv.max();//Determina la localizacion del punto mas brillante
107   float dist=distancia(luz,manoDer);//Calculamos la distancia entre la mano derecha y el
108   //punto mas brillante
109   if(dist<50){//Determinamos si el numero de dedos es menor o igual a 0
110     hand="Close";//Determinamos que la variable hand esta cerrada
111     hands=0;//Determinamos que la variable hands esta cerrada asignando un "0"
112   }
113   else{
114     hand="Open";//Sino determinamos que la variable hand esta abierta
115     hands=1;//Sino determinamos que la variable hands esta abierta asignando un "1"
116   }
117   stroke(255, 0, 0);//Color de trazo rojo
118   strokeWeight(4);//Grosor 4
119   noFill();//Sin relleno
120   ellipse(luz.x, luz.y, 20, 20);//Circulo en la ubicacion de la luz
121 }
```

Figura 71. Función abertura().

En la Figura 72 se observa la función “CalcularAngulos()”, que retorna el cálculo de los ángulos entre los vectores formados por las coordenadas de las articulaciones. Sus argumentos son tres “PVector” que guardan las coordenadas de la articulación en donde se interceptan las otras 2 articulaciones ubicadas a los extremos. Para hallar los vectores se usa la ecuación 1, que consiste en restar, haciendo uso de la función “sub()”, las coordenadas de los extremos con el centro para hallar los 2 vectores, el de la extremidad de referencia y la extremidad que va variando su abertura. Si se ubica al revés se genera el complemento del ángulo. Una vez encontrados los vectores se usa la ecuación 4 para calcular los ángulos de giro, y para retornar en grados se usa la función “degrees()”.

```
181 //Calcular angulo entre vectores
182 //Se requiere de las coordenadas del vector origen y los extremos
183 float CalcularAngulos(PVector one, PVector two, PVector centro){
184   float angulo=0;//Iinicializamos la variable a retornar
185   //Formula de calulo de angulo a=acos(A.B/|A|.|B|)
186   PVector vector1=PVector.sub(centro,one);//vector desde el extremo 1 al origen
187   PVector vector2=PVector.sub(centro,two);//vector desde el extremo 2 al origen
188   float productoPunto=(vector1.x*vector2.x)+(vector1.y*vector2.y);//Producto punto entre los 2 vectores
189   float magnitud1=sqrt(sq(vector1.x)+sq(vector1.y));//Magnitud del vector1
190   float magnitud2=sqrt(sq(vector2.x)+sq(vector2.y));//Magnitud del vector2
191   angulo=acos(productoPunto/(magnitud1*magnitud2));//Coseno inverso de la division de productos
192   return degrees(angulo);//retornamos la variable angulo en grados
193 }
```

Figura 72. Función CalcularAngulos().

En la Figura 73 se muestra la función “GradosDeLibertad()”, en donde se desarrolla el cálculo de los ángulos, muestreo en la interfaz y envío de datos a través de MQTT. En esta parte del algoritmo se crean 6 “PVector”, en donde se receptan las coordenadas detectadas por el Kinect. Se detecta las coordenadas usando el método “getJointPositionSkeleton()” del objeto ”kinect”, cuyos argumentos son: el usuario, la articulación que desea detectar y la variable “PVector” donde se desea guardar. Las articulaciones que se detectan son: Hombro derecho, hombro izquierdo, cadera derecha, codo derecho, mano derecha y mano izquierda. Para el cálculo de los ángulos se necesitan solo 2 coordenadas de la articulación, para ello se crean nuevos “PVector” en donde se guarden solo las coordenadas (x,y) de: mano derecha, codo derecho, hombro derecho, hombro izquierdo y cadera derecha; y las coordenadas (x,z) de la mano derecha. Una vez formado los nuevos vectores se procede a usar la función “CalcularAngulos()”, en donde calcula los ángulos entre los vectores formados por las articulaciones, estas son:

- El ángulo entre el vector referencia formado por el hombro derecho y la cadera derecha, y el vector móvil formado por el hombro derecho y el codo derecho.

- El ángulo entre el vector referencia formado por el hombro derecho y la codo derecho, y el vector móvil formado por el codo derecho y la mano derecha.

- El ángulo entre el vector referencia formado por el hombro izquierdo y el hombro derecho, y el vector móvil formado por el hombro derecho y la mano derecha.

```
//Funcion del proceso(Al ser calculo de angulos es indiferente la coordenadas de donde se realice)
public void GradosDeLibertad(int userId) {
  //Vector posicion de las articulaciones
  PVector rightHand = new PVector();//Generamos el vector de mano derecha
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, rightHand);
  PVector rightElbow = new PVector();//Generamos el vector de codo derecho
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW, rightElbow);
  PVector rightShoulder = new PVector();//Generamos el vector de hombro derecho
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER, rightShoulder);
  PVector rightHip = new PVector();//Generamos el vector de cadera derecha
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HIP, rightHip);
  PVector leftShoulder = new PVector();//Generamos el vector de hombro izquierdo
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER, leftShoulder);
  PVector leftHand = new PVector();//Generamos el vector de mano izquierda
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, leftHand);

  //Vector en coordenadas 2D
  PVector rightHand2D = new PVector(rightHand.x, rightHand.y);//Coordenadas X,Y de la mano derecha
  PVector rightElbow2D = new PVector(rightElbow.x, rightElbow.y);//Coordenadas X,Y del codo derecho
  PVector rightShoulder2D = new PVector(rightShoulder.x, rightShoulder.y);//Coordenadas X,Y del hombro derecho
  PVector rightHip2D = new PVector(rightHip.x, rightHip.y);//Coordenadas X,Y de la cadera
  PVector leftShoulder2D = new PVector(leftShoulder.x, leftShoulder.y);//Coordenadas X,Y del hombro izquierdo
  PVector rightHand2DHorizontal = new PVector(rightHand.x, rightHand.z);//Coordenadas X,Z del hombro derecho

  //Calculo de angulos haciendo uso de la funcion CalcularAngulos
  RightshoulderAngle=CalcularAngulos(rightHip2D,rightElbow2D,rightShoulder2D);//Angulo entre hombro y codo
  RightelbowAngle=CalcularAngulos(rightShoulder2D,rightHand2D,rightElbow2D);//Angulo entre codo y mano
  //Angulo entre 2 planos del hombro derecho
  RightShoulderAngleHorizontal = CalcularAngulos(leftShoulder2D,rightHand2DHorizontal,rightShoulder2D);
```

Figura 73. Cálculo de ángulos en la función GradosDeLibertad().

En la Figura 74, se muestra el escalamiento de los ángulos calculados, tomando como referencia los datos experimentales y los datos que necesitan los servomotores, además de restricciones que evitan que el servo obtenga valores no deseados y nada coherentes. Para el escalamiento se usa la función “map()”, cuyos argumentos son: la variable a escalar, el mínimo inicial, el máximo inicial, el mínimo deseado y el máximo deseado.

```
235  //Rango de datos de angulo entre hombro y codo
236  if (RightshoulderAngle <= 0) {//Si el angulo es menor o igual a 0
237    RightshoulderAngle=10;//Se asigna el valor de 10
238  }
239
240  //Rango de datos de angulo entre codo y mano
241  codo=map(RightelbowAngle,50,180,5,150);//Se escala los datos recibidos a datos que se ajusten al servo del brazo
242  if (codo <= 20) {//Si el angulo es menor o igual a 20
243    codo=20;//Se asigna el valor de 10
244  }
245  if (codo > 180) {//Si es mayor a 180
246    codo=170;//Se asigna el valor de 170
247  }
248
249  //Rango de datos de angulo entre hombro y mano en plano x,z
250  //Se escala los datos recibidos a datos que se visualicen de mejor forma
251  hombro=map(RightShoulderAngleHorizontal,70,115,45,170);
```

Figura 74. Escalamiento de ángulos de la función GradosDeLibertad() para su uso en servos.

En la Figura 75 se aprecia como se muestran las variables en la interfaz. En el lado derecho de la ventana se muestran, con una letra de tamaño 40 y de color blanco, los ángulos calculados: ángulo en (x,y) del hombro derecho, ángulo en (x,z) del hombro derecho, ángulo en (x,y) del codo derecho y la abertura de la mano . Y en la parte inferior se muestran, con letra de tamaño 30 y color negro, las coordenadas de: hombro derecho, hombro izquierdo, cadera derecha, codo derecho, mano derecha y mano izquierda.

```
260  //Escritura de datos
261  fill(255);//Rellenamos de color blanco la letra a usar
262  scale(1);//Escala 1
263  textSize(40);//Tamaño de texto 40
264  //Mostramos todos los angulos
265  text("ANGULOS DE OPERACION",700,280);
266  text("Hombro: "+ int(RightshoulderAngle) +", "+ int(hombro)+ "\n" + "Codo: " + int(RightelbowAngle)+ "\n" + "Mano: " + hand,650,340);
267
268  fill(0);//color negro
269  text("COORDENADAS",480,520);
270  textSize(30);//Tamaño de texto 30
271  text("Hombro Derecho= X: "+ int(rightShoulder.x) +", Y: "+ int(rightShoulder.y)+", Z: "+ int(rightShoulder.z)+ "\n"
272  + "Codo Derecho= X: "+ int(rightElbow.x) +", Y: "+ int(rightElbow.y)+", Z: "+ int(rightElbow.z)+ "\n"
273  + "Mano Derecha= X: "+ int(rightHand.x) +", Y: "+ int(rightHand.y)+", Z: "+ int(rightHand.z),10,570);
274  text("Hombro Izquierdo= X: "+ int(leftShoulder.x) +", Y: "+ int(leftShoulder.y)+", Z: "+ int(leftShoulder.z)+ "\n"
275  + "Cadera Derecha= X: "+ int(rightHip.x) +", Y: "+ int(rightHip.y)+", Z: "+ int(rightHip.z)+ "\n"
276  + "Mano izquierda= X: "+ int(leftHand.x) +", Y: "+ int(leftHand.y)+", Z: "+ int(leftHand.z),650,570);
```

Figura 75. Impresión de las coordenadas y ángulos en la interfaz.

En la Figura 76, se aprecia el envío de datos, después de un delay de 200 milisegundos, mediante el método “publish()” del objeto “client”. Se envía los cuatro grados de libertad y el estado de conexión.

```
289   //Envio por MQTT
290   delay(200);//Retardo de 200 ms para que no generar un cambio de angulo
291   //Envio de datos(4 articulaciones), y estado de conexion hacia la estacion receptora
292   //por medio de la funcion publish de MQTT
293   client.publish("moreiravasquez26102021",""+int(RightshoulderAngle)+","+int(RightelbowAngle)+
294   ","+int(hombro)+","+int(hands)+","+int(conexion));
295 }
296
```

Figura 76. Envío de datos a travez de MQTT.

En la Tabla 2 se aprecia la lista de articulaciones y su traducción, que pueden ser detectadas mediante el sensor Kinect y que están disponibles en la librería SimpleOpenNI.

Tabla 2
Listado de articulaciones de la librería SimpleOpenNI

Librería	**Descripción**
SKEL_HEAD	Cabeza
SKEL_LEFT_ELBOW	Codo izquierdo
SKEL_LEFT_FINGERTIP	Punta del dedo de la mano izquierda
SKEL_LEFT_FOOT	Pie izquierdo
SKEL_LEFT_HAND	Mano izquierda
SKEL_LEFT_HIP	Cadera izquierda
SKEL_LEFT_KNEE	Rodilla izquierda
SKEL_LEFT_SHOULDER	Hombro izquierdo
SKEL_NECK	Cuello
SKEL_ RIGHT _ELBOW	Codo derecho
SKEL_ RIGHT _FINGERTIP	Punta del dedo de la mano derecha
SKEL_ RIGHT _FOOT	Pie derecho
SKEL_ RIGHT _HAND	Mano derecha
SKEL_ RIGHT _HIP	Cadera derecha
SKEL_ RIGHT _KNEE	Rodilla derecha
SKEL_RIGHT_SHOULDER	Hombro derecho
SKEL_TORSO	Torso

3.4. Desarrollo de estación receptora.

El desarrollo de la recepción de datos y la replicación de los movimientos ejecutados por un brazo robótico son semejantes a la persona que se encuentra en la estación emisora, para llevar a cabo esto se utilizó el módulo ESP8266 el cual permite la conexión a una red wifi, mediante la conexión a internet nos enlazamos al servidor mosquitto con un topic específico el cual debe ser el mismo para poder recibir los datos de la estación emisora en el arduino UNO controlando de esta manera los servomotores para replicación de los movimientos, en la Figura 77 se aprecia la estructura se esta estación.

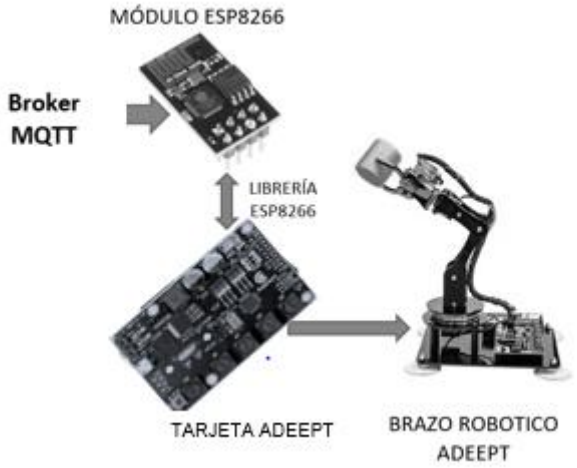


Figura 77. Esquema de funcionamiento de la estación receptora.

3.4.1. Estructura de la estación.

La estructura original del brazo adeept fue modificada para dar una apariencia más real a la forma del brazo humano, de esta manera se aprecian mejor los movimientos de la articulación del hombro, este cambio fue realizado en la pieza como se muestra en la Figura 78, el diseño se realizó en el software AUTOCAD e impresas en 3D con dimensiones precisas, el plano de la pieza se las puede observar en el anexo B y anexo C.

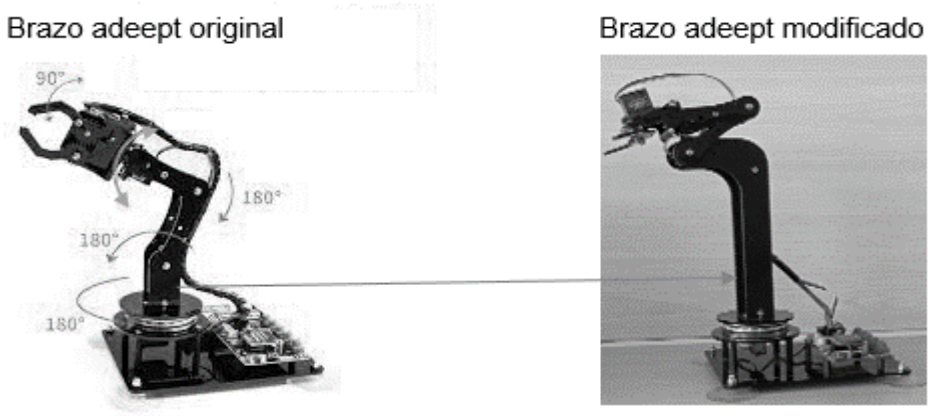


Figura 78. Mejora del brazo adeept.

3.4.2. Desarrollo de la programación.

En la configuración del módulo ESP8266 por medio del IDE de arduino se usa la librería esp8266wifi dando acceso a esta tarjeta, luego se incluye PubSubClient para recibir y enviar datos al servidor MQTT, con esto se declaran como constantes el nombre y la contraseña de la red a conectar, como también la dirección del servidor, en este caso es “test.mosquitto.org”, en la Figura 79 se observa la conexión a la red la universidad, en este caso se omitió la contraseña porque esta red es abierta.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// ingreso de voleres de red y servidor al deseo conectarme
const char* ssid = "UPS_ESTUDIANTES";
//const char* password = "eclipse2009";
const char* mqtt_server = "test.mosquitto.org";
```

Figura 79. Librerías, credenciales de red y servidor.

La función “setup_wifi” realiza la conexión a la red, para esto se configura el módulo en modo STA o en modo estación guardando el nombre de la red en la memoria flash no volátil que, ante un fallo de conexión, permita al esp8266 volver a conectarse automáticamente a la última red, esta programación se observa en la Figura 80.

```
void setup_wifi() {

  delay(10);
  // Empezamos la coneccion a la red wifi
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //Colocamos en modo estacion y verificamos el estado de esp8266
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid);
```

Figura 80. Función de acceso a red.

Obteniendo la conexión a la red wifi se procede a enlazar al servidor especificando la dirección y el puerto del servidor MQTT necesario para conectarse, y así, tener el flujo de datos desde la estación emisora hasta la receptora, esto se observa en la Figura 81.

```
void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}
```

Figura 81. Enlace al servidor MQTT.

Completado el código de programación se selecciona la tarjeta adecuada. En la pestaña Herramientas del IDE de Arduino escogiendo como placa a “Generic ESP8266 Module”, se ubica el puerto en el que se conectó el adaptador USB a ESP8266 ESP-01 en modo de programación y conectado el módulo ESP-01, en la Figura 82 se muestra la elección de placa y puerto USB. Finalizada la carga del programa al módulo este es retirado del adaptador y ubicado en la placa adeept V3.0.

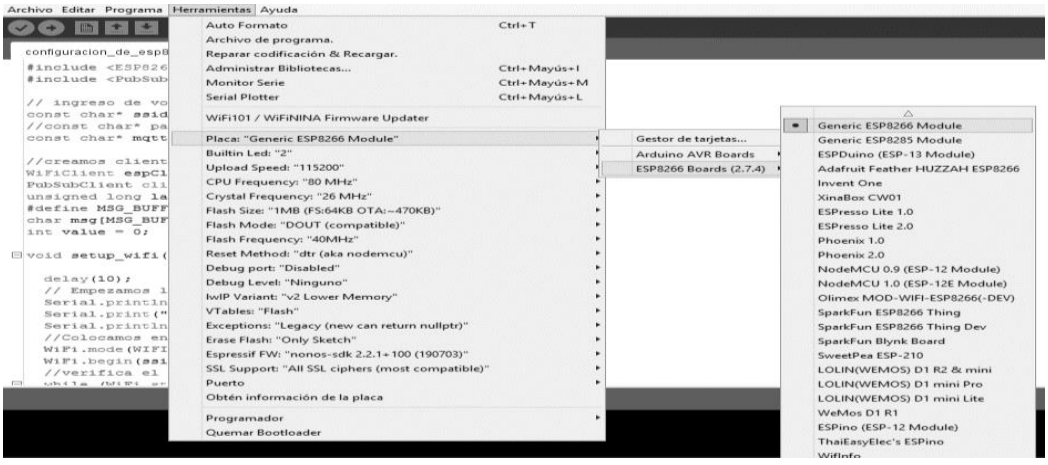


Figura 82. Selección de tarjeta Generic ESP8266 Module.

En la programación de la placa adeept V3.0 se utilizó el IDE de Arduino ya que está utiliza el microprocesador ATMEL MEGA 328P, siendo este el mismo que utiliza la palca Arduino UNO. Se incluye las librerías necesarias para el uso de los servos, separación de los datos que ingresan por puerto serial y las que utiliza la pantalla OLED en la visualización de los nombres, esto se aprecia en la Figura 83.

```
#include <Separador.h>
#include <Servo.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET     4
```

Figura 83. Librerías utilizadas.

La estación emisora envía un paquete de datos separados por comas estos ingresan por el monitor serial, siendo leídos mediante la función “Serial.read” para ser almacenados en una variable de tipo String, aplicando la librería “Separador” se divide la variable String en cinco datos, los cuatro primeros corresponden a los ángulos de giro de cada servo motor y el último dato es para fijar en posición de desconexión al momento en que decidamos terminar el movimiento del brazo, para que los servo motores puedan interpretar el valor del giro se realiza una conversión de dato tipo String a dato tipo Float, esta programación se visualiza en la Figura 84.

```
String datosrecividos = inString;
//////////separacion de datos por coma//////////////////
String e1 = s.separa(datosrecividos, ',', 0);
String e2 = s.separa(datosrecividos, ',', 1);
String e3 = s.separa(datosrecividos, ',', 2);
String e4 = s.separa(datosrecividos, ',', 3);
String e5 = s.separa(datosrecividos, ',', 4);
//////////conversion de string en flotante////////////////
float angulo1 = e1.toFloat();
float angulo2 = e2.toFloat();
float angulo3 = e3.toFloat();
float angulo4 = e4.toFloat();
float angulo5 = e5.toFloat();
if(angulo5==0){
  servo1.write(90);
  servo2.write(90);
  servo3.write(90);
  servo4.write(0);
  Serial.println(datosrecividos);
}else {
  ///////////////asigna un valor a servo/////////////
  servo1.write((angulo1));
  servo2.write(angulo2);
  servo3.write(angulo3);
  servo4.write(pinza);
```

Figura 84. Separación y conversión de datos.

4. RESULTADOS

La interfaz desarrollada en Processing nos muestra el video captado por el sensor Kinect, además de unos puntos y líneas dibujados sobre el brazo derecho que indican las articulaciones del brazo. También se aprecia los ángulos calculados que realizan los servos para emular el movimiento, estos son:

- El ángulo del hombro que gira de arriba-abajo.
- El ángulo del hombro que realiza de derecha-izquierda.
- El ángulo de abertura del codo con respecto al brazo.
- Estado de abertura de la mano.

También se aprecian las coordenadas de las articulaciones usadas para el cálculo de los ángulos de giro, estas son:

- El hombro izquierdo.
- La cadera derecha.
- El hombro derecho.
- El codo derecho.
- La mano derecha.

El estado de conexión con el broker MQTT se muestra en la sección inferior derecha de la interfaz.

En la Figura 85 se observa la interfaz mostrando los ángulos realizados al alzar el brazo perpendicular al tronco, en la Figura 86 se muestra la comparativa entre el operador y el brazo robótico.

Figura 85. Movimiento de brazo a posición perpendicular al piso.

Figura 86. Primera comparativa entre operador y brazo robótico.

En la Figura 87 se observa la interfaz mostrando los ángulos realizados al alzar el brazo perpendicular y doblar el codo paralelo al tronco, en la Figura 88 se muestra la comparativa entre el operador y el brazo robótico.

Figura 87. Movimiento de brazo con posición del codo a 90 grados.

Figura 88. Segunda comparativa entre operador y brazo robótico.

En la Figura 89 se observa la interfaz mostrando los ángulos realizados al colocar el brazo en posición de reposo, en la Figura 90 se muestra la comparativa entre el operador y el brazo robótico.

Figura 89. Brazo en posición de reposo.

Figura 90. Tercera comparativa entre operador y brazo robótico.

En la Figura 91 se observa la interfaz mostrando los ángulos realizados al mover el brazo de derecha a izquierda extendiendo el brazo hacia adelante, en la Figura 92 se muestra la comparativa entre el operador y el brazo robótico.

Figura 91. Movimiento del brazo hacia adelante.

Figura 92. Cuarta comparativa entre operador y brazo robótico.

En la Figura 93 se observa la interfaz mostrando los ángulos realizados al extender el brazo hacia adelante y flexionando el codo, en la Figura 94 se muestra la comparativa entre el operador y el brazo robótico.

Figura 93. Movimiento del brazo hacia adelante con flexión de codo.

Figura 94. Quinta comparativa entre operador y brazo robótico.

En la Figura 95 se observa la interfaz mostrando los ángulos realizados al extender el brazo hacia adelante y flexionando el codo, además de usar el guante para provocar el punto más brillante cerca de la ubicación de la mano derecha, indicando así que se debe cerrar la pinza del brazo robótico, en la Figura 96 se muestra la comparativa entre el operador y el brazo robótico.

Figura 95. Movimiento de brazo con cierre de mano.

Figura 96. Sexta comparativa entre operador y brazo robótico.

5. ANALISIS DE RESULTADOS

En la Figura 85 se observa cómo se detecta la ubicación de las articulaciones, esto se representa en los puntos rojos dibujados encima. Al unir esos puntos obtenemos vectores, representados por líneas de color negro, gracias a eso se calcula el ángulo producido entre vectores mostrados en la interfaz. En la Figura 86 se observa como el brazo robótico adopta la misma ubicación que el brazo del operador.

En la Figura 97 se muestra como los ángulos calculados se aproximan con los realizados por el operador, además del punto brillante estar lejos de la mano derecha, indicando que debe de estar abierta.

Figura 97. Comparativa de ángulos entre la realidad contra lo calculado.

En la Figura 98 se muestra como al realizar un movimiento del codo hacia arriba se genera un cambio al instante de los ángulos calculados, procediendo el brazo robótico a emular el mismo movimiento, tal y como se aprecia en la Figura 88. También se muestra como la similitud es aproximada a la realidad, teniendo en cuenta el brazo no está elevado totalmente, produciendo un ángulo menor a 90 grados, así como tener un pequeño movimiento hacia adelante generando un ángulo menor a 180 entre el brazo derecho y el pecho.

Figura 98. Comparativa de ángulos entre la realidad contra lo calculado después de realizar un movimiento.

En la Figura 99 se aprecia como la ubicación del punto más brillante está cerca de la zona de la mano derecha, lo que se interpreta como un cierre de la pinza del brazo robótico, lo cual se aprecia en la interfaz, con el mensaje de “Close” y también en la Figura 96, en donde se muestra la similitud entre el operador y el brazo robótico.

Figura 99. Comparativa de ángulos entre la realidad contra lo calculado después de cerrar la mano.

En la Figura 94 se observa como el operador tiene el brazo hacia adelante, mostrando similitud con el brazo robótico, y en la Figura 92 se observa la transición del brazo al realizar un giro hacia la izquierda teniendo el brazo en frente, pasando de tener 136 grados a tener 92 grados, demostrando el emulado del brazo robótico. En la Figura 100 se muestra la transición y cambio de los ángulos al realizar el movimiento.

Figura 100. Transición de movimiento al realizar un giro hacia la izquierda.

El envío de datos se realiza de manera óptima y veloz, con un mínimo retraso de tiempo, esto permite que el movimiento del brazo del operador sea replicado casi al instante.

6. CONCLUSIONES

- Para el desarrollo del programa se tomaron los datos brindados por el sensor Kinect y fueron enviados al ordenador por medio del adaptador, siendo utilizadas en el software processing haciendo uso de las librerías correspondientes.

- Las coordenadas detectadas de las articulaciones fueron analizadas y usadas para el cálculo de los grados de giro que realiza el servomotor.

- Los grados que realizan los servos son plasmados en una interfaz gráfica en donde se especifica el grado del servo de la articulación a la que pertenece y el estado de abertura de la mano. también se observa el estado de conexión.

- El video captado por el sensor Kinect es mostrado en la interfaz gráfica, dibujando por encima las articulaciones que realiza el brazo.

- Se ejecuta una función de desconexión al detectar la mano izquierda en una coordenada específica, al desconectar se dejará de enviar datos hacia el bróker MQTT.

- El movimiento que realiza el brazo robótico en la estación receptora es muy similar al realizado por el operador en la estación emisora por lo cual se cumple con los objetivos planteados.

- Con la modificación del brazo adeept se logró obtener una mejor apreciación de la replicación de los movimientos.

- Mediante la programación correcta del módulo esp-01 aseguramos una conexión estable a una wifi y servidor obteniendo una buena comunicación entre la estación emisora y la estación receptora con un desfase de no más de 500ms.

- Mediante las pruebas realizadas podemos determinar que este prototipo es escalable, siendo posible llevarlo a la industria utilizando un control en brazo industrial, donde se necesita la utilización de fuerza.

7. RECOMENDACIONES.

- Previo al inicio del funcionamiento, se debe verificar el estado de conexión de la estación emisora y receptora, que esté energizado y en posición inicial.

- Se debe tener una iluminación adecuada, lo suficiente para que se reconozca bien las articulaciones, y que también se pueda apreciar el punto brillante del guante. tener cuidado de no poseer brillos de luz cerca del operador porque puede cegar o confundir al programa. en estos casos se recomienda ubicarse en una zona sin brillo y colocar una luz en la parte inferior izquierda que ayuda al programa a identificar un solo punto brillante, ignorando los puntos de alrededor, gracias a eso, al momento del uso del guante este será reconocido rápidamente.

- Se recomienda tener una buena velocidad de internet y movimientos suaves del brazo, para evitar los retrasos y movimientos bruscos.

- Se recomienda realizar la desconexión con la estación receptora antes de salir de la visión de la cámara.

- Se debe evitar usar prendas de colores variados o con diseños de figuras humanoides, para evitar una confusión en la detección de las articulaciones.

- Verificar que la cámara y el adaptador estén conectados de manera correcta, caso contrario generará un error.

- Revisar el estado de los servomotores, estos pueden generar movimientos erróneos si los servos no funcionan de manera correcta.

- Modificar la estructura del brazo adeept para dar más similitud a un brazo humano apreciando de mejor manera los movimientos.

- Ingresar correctamente las credenciales de la red a la que se desea conectar, la velocidad del internet es vital para obtener movimientos fluidos sin retardos.

- Utilizar cables con longitud adecuada para facilitar el movimiento y giro de las articulaciones.

- Para facilitar la programación del módulo esp-01 se recomienda usar un adaptador USB a esp8266 esp-01.

8. REFERENCIAS BIBLIOGRÁFICAS

Adeept.com. (2021). *Adeept*. Obtenido de https://www.adeept.com/adeept-arduino-compatible-diy-5-dof-robotic-arm-kit-for-arduino-uno-r3-steam-robot-arm-kit-with-arduino-and-processing-code_p0118.html

Andrew W. Senior, J. H. (2004). *Guide to Biometrics.*

Anil K. Jain, A. A. (2011). *Introduction to Biometrics.*

Arduino. (2008). *Arduino Nano.* Obtenido de arduino.cc: https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf

ARDUINO.cl. (2021). Obtenido de https://arduino.cl/producto/arduino-uno/

Ayala, M. M., & Barragan, J. E. (2013). Representación geométrica de las coordenadas. Lat. Am. J. Phys. Educ.

B. Navya Rupa, G. K.-h. (2015). Test Report Generation Using JSON. *International Journal of Software Engineering and Its Applications*.

Banzi, M. (2008). *Getting Started with Arduino.*

Batini, C. (s.f.). *Conceptual Database Design, An Entity-relationship Approach.* 1992: Benjamin/Cummings.

Baturone, A. O. (2005). *Robótica: Manipuladores y Robots Móviles.* Marcombo.

Ben Everard, E. U. (2014). *Learning Computer Architecture with Raspberry Pi.*

Blum, C. B. (2013). *Sams Teach Yourself Python Programming for Raspberry Pi in 24 Hours.*

Boulgouris, N. V. (2009). *Biometrics: Theory, Methods, and Applications.*

Boxall, J. (2013). *Arduino Workshop: A Hands-On Introduction with 65 Projects.*

Brock Craft, J. E. (2015). *Raspberry Pi Projects For Dummies.*

C.Calderón, M. C. (2014). Desarrollo de una Aplicación Cliente/Servidor para un Wall View en base a la. *REVISTA EPN*, 7. Obtenido de https://revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/155/pdf

Ceballos, E. L. (2013). Diseño web adaptativo o responsivo. *Revista Digital Universitaria*.

Elio, E. M. (2015). *Microcontrolador Arduino.* Universidad Cristobal Colón.

Espressif Systems. (2020). Obtenido de https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

Fernando Bizzarro, A. H. (2016). *The V-Dem Party Institutionalization Index: a new global indicator (1900-2015).*

Germán, C. M., & Esteban, J. A. (2014). *DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA QUE.* UNIVERSIDAD POLITÉCNICA SALESIANA.

Guillermo Diez-Andino Sancho, R. M. (2003). Desarrollo de un servidor HTTP para dispositivos móviles en J2ME. *Departamento. Ingeniería Telemática - Universidad Carlos III de Madrid*. Obtenido de http://www.it.uc3m.es/celeste/papers/ServidorHTTP.pdf

Javier, L. S. (2001). Base de Datos Distribuidas Estudio de Actualización de Réplicas. *Facultad de Informática – UNLP*. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/3872/Documento_completo__.pdf-PDFA1b.pdf?sequence=1

Jayesh Umre, K. B. (2014). Comparative performance analysis of MySQL and SQLite relational database management systems in Windows 10 environments. *International Journal of Latest Trends in Engineering and Technology*.

Jimy Alexander Cortés Osorio, F. A. (2010). *Sistemas de seguridad basados en biometría.*

Kilicdagi, A. (2014). *Laravel Design Patterns and Best Practices.*

Kolman, B., & Hill, D. R. (2006). Algebra lineal. Pearson Educación.

Marcos, A. G., Ascacíbar, F. J., Espinoza, A. V., Elías, F. A., Limas, M. C., Meré, J. B., & González, E. P. (2006). *Técnicas y Algoritmos Básicos de Visión Artificial.* Universidad de La Rioja.

Margolis, M. (2011). *Arduino Cookbook.*

María F. Maldonado, A. C. (2008). Implementación de un sistema Web para manejo de datos meteorológicos del Laboratorio de Energías Alternativas y Eficiencia Energética de la Escuela Laboratorio de Energías Alternativas y Eficiencia Energética de la Escuela Politécnica Nacional. Obtenido de http://bibdigital.epn.edu.ec/bitstream/15000/4923/1/PAPER%20Implementaci%C3%B3n%20de%20un%20sistema%20Web%20para%20manejo%20de%20datos%20meteorol%C3%B3gicos%20del%20Laboratorio%20de%20Energ%C3%ADas%20Alterna.pdf

Martínez Ramírez, J. M., SesrisnelbaTorres Manzo, W., & Alberto Palacio., L. (enero-marzo de 2018). ¿POR QUÉACTUALIZAR EL SOFTWARE DE LA COMPUTADORA PERSONAL? *REDEL. Revista Granmense de Desarrollo Local*.

Microsoft. (2020). *Microsoft, Hardware & networking*. Obtenido de Set up Kinect for Windows v2 with a Kinect Adapter for Windows 10 PC: https://support.xbox.com/en-US/help/hardware-network/kinect/kinect-for-windows-v2-setup-with-adapter

Mishra, A. (2014). Critical Comparison Of PHP And ASP.NET For Web Development. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 3,*. Obtenido de http://www.ijstr.org/final-print/july2014/Critical-Comparison-Of-Php-And-Aspnet-For-Web-Development.pdf

Molloy, D. (2016). *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux.* Obtenido de https://www.element14.com/community/community/raspberry-pi?src=raspberrypi

Monk, S. (2012). *Programming the Raspberry Pi: Getting Started with Python.*

MQTT. (2021). Obtenido de https://mqtt.org/

Mulloy, B. (2012). *Web API Design, Crafting Interfaces that Developers Love.* apigee.

Nguyen, Q. H. (2015). Building a web application with LARAVEL 5. *Oulu University of Applied Sciences*.

Oracle. (2011). *PHP Scalability and High Availability Database Resident Connection Pooling and Fast Application Notification.*

Orbaugh Antillón, D. (2021). Diseño e implementación de un sistema de control para un brazo robótico asistencial de 6 grados de libertad para procedimientos de cirugías para el diagnóstico de epilepsia. *Universidad del Valle de Guatemala*.

Peck, A. (2017). *Jumpstarting the Raspberry Pi Zero W.*

Pérez, J. (2013). *upaep.mx.* Obtenido de https://upaep.mx/micrositios/coloquios/coloquio2013/memorias/Mesa%204%20Mec%20y%20Bio/4.-Meliton%20Fernando%20Santiago.pdf

Philbin, C. A. (2013). *Adventures in Raspberry Pi.*

Processing Foundation. (2020). Obtenido de Processing Foundation: https://processing.org/

Purdum, J. J. (2012). *Beginning C for Arduino: Learn C Programming for the Arduino.*

ResearchGate. (5 de Enero de 2022). *ResearchGate*. Obtenido de https://www.researchgate.net/figure/Architecture-of-Microsoft-Kinect-sensor_fig2_282477283

Ross, A. A. (2006). *Handbook of Multibiometrics.*

Salvatore, J., Osio, J., & Morales., M. (2014). Detección de objetos utilizando el sensor Kinect. *LACCEI Latin American and Caribbean Conference for Engineering and Technology*.

Sanderson, K. (2019). Automatización: Química dispara a la Luna. *TECHNOLOGY FEATURE*.

Sanitaria. (11 de Noviembre de 2020). *REDACCIÓN MÉDICA*. Obtenido de https://www.redaccionmedica.com/virico/noticias/5g-operar-platano-distancia-tecnologia-covid-conspiracion-6836

Scott Trent, M. T. (2008). *Performance Comparison of PHP and JSP as Server-Side Scripting Languages.* Obtenido de https://www.researchgate.net/publication/225161349_Performance_Comparison_of_PHP_and_JSP_as_Server-Side_Scripting_Languages

Spectator, C. (2016). *Comparativa de rendimiento de la red entre distintos proveedores Cloud desde diferentes localizaciones geográficas.*

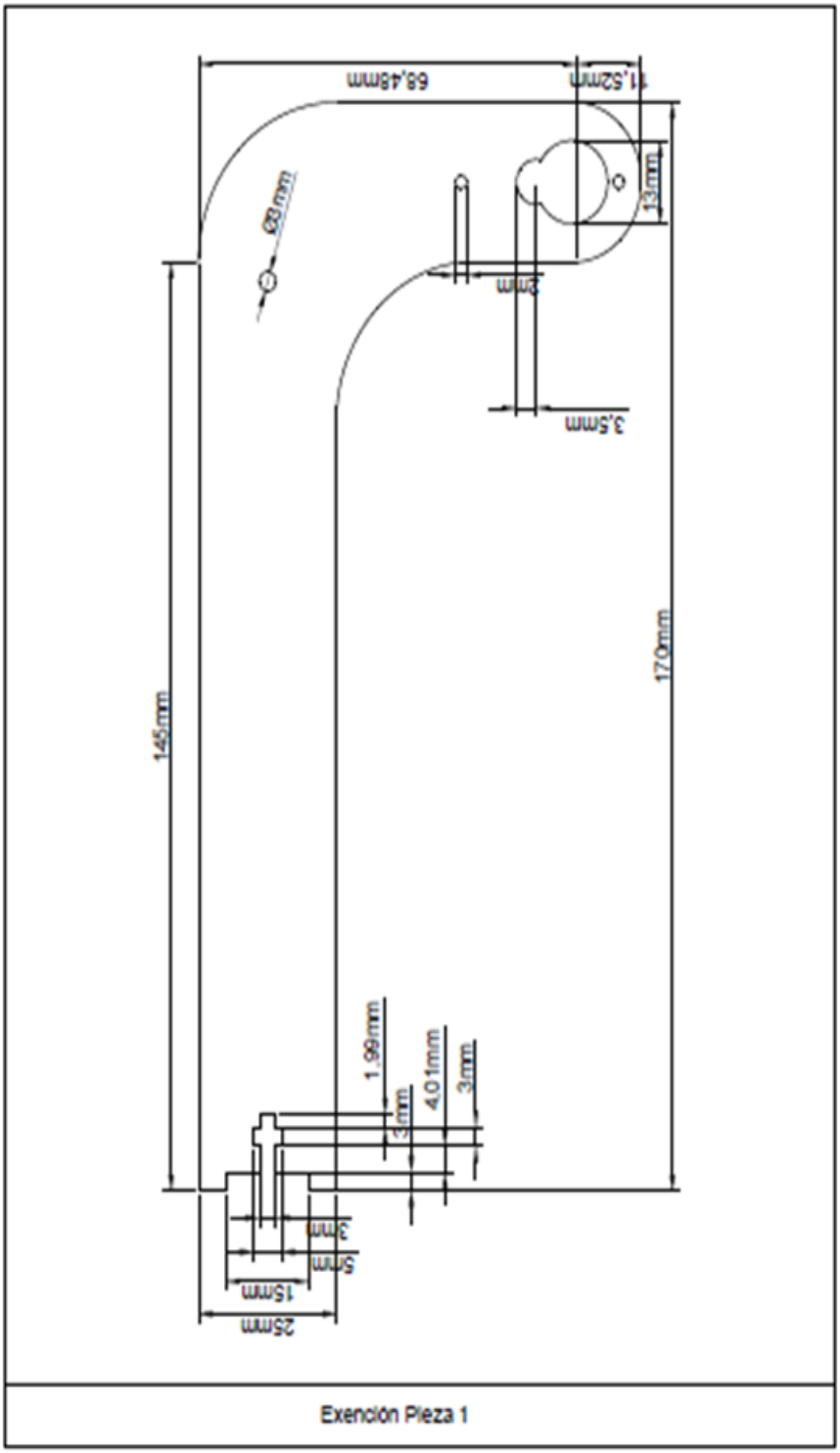
UNIT ELECTRONIC. (2021). Obtenido de https://uelectronics.com/producto/interfaz-usb-a-esp8266-programador/

9. ANEXOS

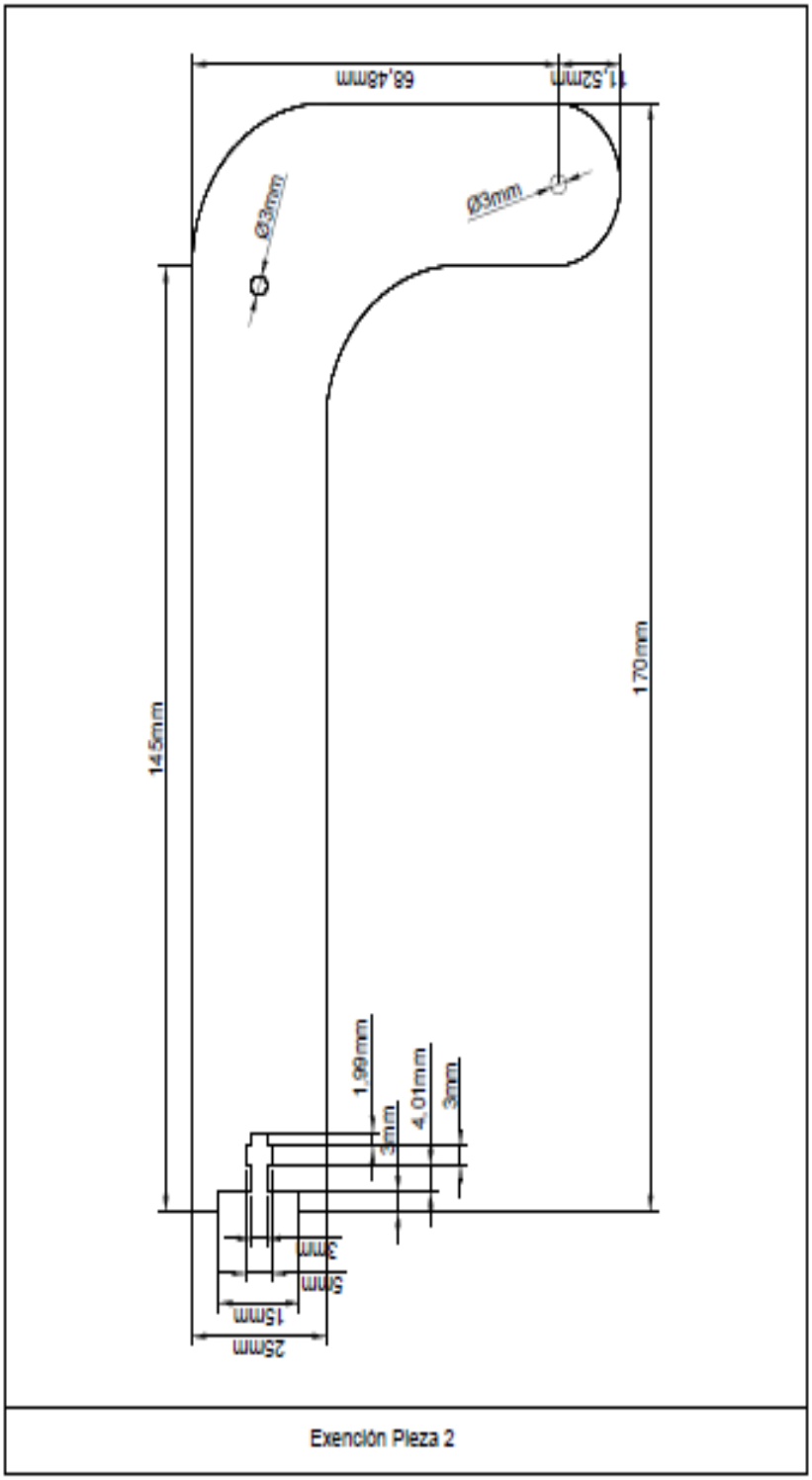
ANEXO A. CRONOGRAMA DE DURACIÓN DEL PROYECTO.

	ACTIVIDADES	MESES									
		1	2	3	4	5	6	7	8	9	10
	PRESENTACIÓN DEL ANTEPROYECTO										
	CORRECCIÓN DEL ANTEPROYECTO										
	APROVACIÓN DEL ANTEPROYECTO										
	BUSQUEDA Y ADQUISICIÓN DE EQUIPOS										
	IMPLEMENTACIÓN DE ESTRUCTURA										
	DESARROLLO Y PROGRAMACIÓN PROCESSING-ARDUINO										
	COMPROBACIÓN DE FUNCIONAMIENTO										
	DOCUMENTACIÓN										
	PRESENTACIÓN DE PROYECTO										

ANEXO B. DISEÑO DE PRIMERA EXTENSION DEL BRAZO EN AUTOCAD.



ANEXO C. DISEÑO DE SEGUNDA EXTENSION DEL BRAZO EN AUTOCAD.



ANEXO D. PROGRAMACION DEL MODULO ESP-01.

```
////////////////////PROYECTO TECNICO//////////////////////////////
//////////CONTROL DE BRAZO ROBOTIVO MEDIANTE WIFI//////////
///////AUTORES: CRISTHIAN VASQUEZ & ROBERTO MOREIRA///////
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
// ingreso de voleres de red y servidor al deseo conectarme
const char* ssid = "UPS_ESTUDIANTES";
//const char* password = "";
const char* mqtt_server = "test.mosquitto.org";
//creamos cliente wifi
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE	(50)
char msg[MSG_BUFFER_SIZE];
int value = 0;
void setup_wifi() {
  delay(10);
  // Empezamos la coneccion a la red wifi
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //Colocamos en modo estacion y verificamos el estado de esp8266
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid);
  //verifica el estado de coneccion
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  //inicializa el generador de números pseudoaleatorios, haciendo que
  //comience en un punto arbitrario en su secuencia aleatoria. Esta secuencia, aunque muy
larga y aleatoria, es siempre la misma.
  randomSeed(micros());
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
  //funcion que devuelve la llamada
void callback(char* topic, byte* payload, unsigned int length) {
  //Serial.print("Message arrived [");
  //Serial.print(topic);
  //Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
 Serial.println();
}
  //funcion de reconeccion
```

```
void reconnect() {
  // bucle de reconeccion
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Crea un random cliente ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    //interntar conectarse
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Suscribirse a un topic
      client.subscribe("moreiravasquez26102021");
      } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // esperar 5 segundos para volver a conectar
      delay(5000);
    }
  }
}
void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
```

ANEXO E. PROGRAMACION EN ARDUINO.

```
//////////////////PROYECTO TECNICO//////////////////////////////
//////////CONTROL DE BRAZO ROBOTIVO MEDIANTE WIFI//////////
///////AUTORES: CRISTHIAN VASQUEZ & ROBERTO MOREIRA////////
#include <Separador.h>
#include <Servo.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET     4
Separador s;
Adafruit_SSD1306 display(128, 64, &Wire, OLED_RESET);
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;
float pinza;
int PULSOMIN = 300;
int PULSOMAX = 1000;
String inString = "";
void setup() {
  ///////////////comunicacion con pantalla OLED///////////////
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  ////Establece el color de visualización de la fuente///////
  display.setTextColor(WHITE);
  //////////////////Limpiar pantalla/////////////////////
  display.clearDisplay();
  ///////////Establecer el tamaño de la fuente///////////////
  display.setTextSize(1.8);
  ////////Establecer la ubicación de la pantalla/////////////
  display.setCursor(30, 3);
  ////////////Mostrar palabras en pantalla/////////////////
  display.println("UNIVERSIDAD");
  display.setCursor(2, 15);
  display.println("POLITECNICA SALESIANA");
  display.setCursor(35, 30);
  display.println("AUTORES");
  display.setCursor(10, 43);
  display.println("CRISTHIAN VASQUEZ");
  display.setCursor(15, 55);
  display.println("ROBERTO MOREIRA");
  display.display();
  ///////////velocidad de comunicacion serial//////////////////
  Serial.begin(115200);
  ///////////asigna variable y pin a cada servo/////////////////
  servo1.attach(9);//conecta el servo1 en el pin 9 al objeto servo
  servo2.attach(6);//conecta el servo1 en el pin 6 al objeto servo
  servo3.attach(10);//conecta el servo1 en el pin 10 al objeto servo
  servo4.attach(3);//conecta el servo1 en el pin 3 al objeto servo
  }
```

```
void loop() {
  if (Serial.available() > 0) {
    int inChar = Serial.read();
    if (inChar != '\n') {
      inString += (char)inChar;
    } else {
     /////asignacion de datos a una variable////////////////////
      String datosrecividos = inString;
      //////////separacion de datos por coma////////////////////
      String e1 = s.separa(datosrecividos, ',', 0);
      String e2 = s.separa(datosrecividos, ',', 1);
      String e3 = s.separa(datosrecividos, ',', 2);
      String e4 = s.separa(datosrecividos, ',', 3);
      String e5 = s.separa(datosrecividos, ',', 4);
      //////////conversion de string en flotante////////////////
      float angulo1 = e1.toFloat();
      float angulo2 = e2.toFloat();
      float angulo3 = e3.toFloat();
      float angulo4 = e4.toFloat();
      float angulo5 = e5.toFloat();
      if(angulo5==0){
        servo1.write(90);
        servo2.write(90);
        servo3.write(90);
        servo4.write(0);
        Serial.println(datosrecividos);
      }else {
        ///////////////asigna un valor a servo/////////////
        servo1.write((angulo1));
        servo2.write(angulo2);
        servo3.write(angulo3);
        servo4.write(pinza);
        Serial.println(datosrecividos);
        //////////control de abrir y cerrar pinza/////////
         if (angulo4 == 0) {
          pinza = 70.00;
        } else if (angulo4 == 1) {
          pinza = 0.00;
        }
      }
       //Serial.println(datosrecividos);
        inString = "";
      }
    }
  }
```

ANEXO F. PROGRAMACION EN PROCESSING.

```
 //Declaracion de librerias
import mqtt.*;//Importar libreria MQTT
import gab.opencv.*;//Importar libreria simple Opencv
import SimpleOpenNI.*;//Importar libreria simple OpenNi

//Declaracion de objetos
MQTTClient client;//Creamos objeto client
SimpleOpenNI kinect;//Creamos objeto Kinect
OpenCV cv;//Creamos objeto openCV

//Declaracion de variables
PFont fuente;//Fuente de letra
PImage logotipo;//Imagen de presentacion
float RightshoulderAngle=0;//Angulo de hombro derecho
float RightelbowAngle=0;//Angulo de codo derecho
float codo=0;//Angulo ajustado de codo derecho
float RightShoulderAngleHorizontal=0;//Angulo de Hombro2
float hombro=0;//Angulo ajustado de Hombro2 derecho
int hands=0;//Estado de mano
String hand;//Carater de mano
int conexion=0;//Determina cuando se conecta
String estado="Desconectado";//Caracter de conexion o desconexion

//Inicializar el programa
void setup() {
  size(1280, 680);//Establecemos el tamaño de la pantalla
  cv=new OpenCV(this,640,480);//Determinanos que el objeto cv es un objeto OpenCV y su
area de trabajo
  kinect = new SimpleOpenNI(this);//Determinanos que el objeto Kinect es un objeto
SimpleOpenNI
  kinect.enableDepth();//Habilita la camara de profundidad
  kinect.enableRGB();//Habilita la camara de imagen a color
  kinect.enableUser();//Habilita la generacion de articulaciones
  kinect.setMirror(false);//Inhabilitamos la funcion de espejo
  client = new MQTTClient(this);//Creamos un nuevo cliente para MQTT
  client.connect("mqtt://test.mosquitto.org", "moreiravasquez261021");//Estableces el servidor
al que se conectara y el nombre de usuario
  //client.connect("mqtt://public:public@public.cloud.shiftr.io",
"MoreiraVasquez261021");//Estableces el servidor al que se conectara y el nombre de usuario
  logotipo=loadImage("Logo.JPG");//Cargamos la imagen del logotipo
  fuente= createFont("Georgia", 32);//Creamos la fuente Georgia de tamaño 32 para su
posterior uso
  textFont(fuente);//Escogemos la fuente del texto que se mostrara
}
//Funcion de trabajo
void draw() {
  background(0,0,255);//Fondo azul
  noStroke();//Sin borde
  fill(255,219,88);//Color amarillo mostaza
  rect(0,480,1280,200);//Rectangulo
```

```
  image(logotipo,640,0,640,240);//Insertamos una imagen de presentacion del proyecto
  kinect.update();//Actualiza las imagenes del Kinect en cada iteraciòn.
  cv.loadImage(kinect.rgbImage());//OpenCv analiza la imagen a color proporcionada por el
Kinect
  image(kinect.rgbImage(), 0, 0);//Dibujar la imagen que percibe el sensor Kinect
  IntVector userList = new IntVector();//Creo una lista para usuarios
  kinect.getUsers(userList);//Determino el usuario por la cantidad de lista
  textSize(40);//Tamaño de texto 40
  fill(255);//color blanco
  text(estado,1030,475);//Muestra el estado de conexion
    if (userList.size() == 1) {//Cuando se detecta al menos 1 usuario
      int userId = userList.get(0);//Guarda el numero de usuario
      if (kinect.isTrackingSkeleton(userId)&&conexion==1) {//Cuando detecta a una
persona=TRUE
        conectar(userId);
        DibujarEsqueleto(userId);//Uso drawSkeleton para dibujar el esqueleto del usuario
        abertura(userId);//Uso del punto mas brillante para determinar si la mano esta abierta o
cerrada
        GradosDeLibertad(userId);//Uso ArmsAngle para determinar los angulos de las
articulaciones
      }
    }
 }

//Funcion de deteccion de usuario
void onNewUser(SimpleOpenNI kinect, int userID) {
  println("START");
  conexion=1;
  estado="Conectado";
  kinect.startTrackingSkeleton(userID);//Inicio de trackeo de esqueleto
}
//Funcion de perdida de usuario
void onLostUser (int userId){
 conexion=0;
 estado="Desconectado";
 println("Usuario: "+ userId); //Mensaje de usuario perdido
}
//Funcion de conexion de cliente
void clientConnected() {
  println("Conectado a MQTT");//Mostramos el mensaje cuando se halla conectado
}
//Funcion de mensaje recibido
void messageReceived(String topic, byte[] payload) {
  println("new message: " + topic + " - " + new String(payload));
}

//Funcion de conexion perdida
void connectionLost() {
  println("Desconectado a MQTT");//Se muestra este mensaje cuando se pierde la conexion
}

//Funcion de abertura de mano
public void abertura(int userId){
```

```
  PVector rightHand = new PVector();//Generamos el vector de mano izquierda
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND,
rightHand);//Guardamos
  //la posicion de la articulacion en el vector creado
  PVector manoDer = new PVector();//Creamos el vector de coordenadas
  kinect.convertRealWorldToProjective(rightHand, manoDer);//Determina el vector posicion
con
  //coordenadas que podamos usar
  PVector luz = cv.max();//Determina la localizacion del punto mas brillante
  float dist=distancia(luz,manoDer);//Calculamos la distancia entre la mano derecha y el
  //punto mas brillante
  if(dist<50){//Determinamos si el numero de dedos es menor o igual a 0
    hand="Close";//Determinamos que la variable hand esta cerrada
    hands=0;//Determinamos que la variable hands esta cerrada asignando un "0"
  }
  else{
    hand="Open";//Sino determinamos que la variable hand esta abierta
    hands=1;//Sino determinamos que la variable hands esta abierta asignando un "1"
  }
  stroke(255, 0, 0);//Color de trazo rojo
  strokeWeight(4);//Grosor 4
  noFill();//Sin relleno
  ellipse(luz.x, luz.y, 20, 20);//Circulo en la ubicacion de la luz
}
//Funcion Distancia entre dedos y mano
float distancia(PVector luz, PVector mano){
  //Calculamos la distancia con la formula de distancia entre puntos en 3D
  return sqrt(sq(luz.x-mano.x)+sq(luz.y-mano.y));//retornamos la distancia entre el dedo y la
mano
}
//Funcion conexion-desconexion
void conectar(int userId) {
  PVector leftHand = new PVector();//Generamos el vector de mano izquierda
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND,
leftHand);//Guardamos la
  //posicion de la articulacion en el vector creado
  PVector manoIzq = new PVector();//Creamos el vector de coordenadas
  kinect.convertRealWorldToProjective(leftHand, manoIzq);//Determina el vector posicion con
  //coordenadas que podamos usar
  if(manoIzq.x<80 && manoIzq.y<80){//Si la ubicacion de la mano izquierda se encuentra en
una
  //zona determinada realiza lo siguiente
    conexion=0;//mandamos a desconectar
    estado="Desconectado";//mandamos a desconectar
  }
}
//Funcion dibujar puntos de articulacion(Convierte coordenadas del Kinect
//a coordenadas de la pantalla)
void DibujarArticulacion(int userId, int jointID) {//Recibe la articulacion que
//se desea graficar la coordenada
  PVector joint = new PVector();//Creamos nuevo vector joint
  float confidence = kinect.getJointPositionSkeleton(userId, jointID, joint);//Guardamos
  //la ubicacion en una variable
```

```
  if (confidence < 0.5) {//Si es menor a 0.5 no grafica la articulacion
    return;
  }
  PVector convertedJoint = new PVector();//Creamos el vector de coordenadas
  kinect.convertRealWorldToProjective(joint, convertedJoint);//Determina el vector posicion
  //con coordenadas que podamos usar
  ellipse(convertedJoint.x, convertedJoint.y, 5, 5);//Dibuja un circulo en la posicion dada
}
//Funcion dibujar esqueleto
void DibujarEsqueleto(int userId) {
  stroke(0);//Color de trazo negro
  strokeWeight(5);//De tamaño 5
  //drawLimb permite dibujar lineas entre 2 puntos(funcion que usa las coordenadas propias
  //del SimpleOpenNI)
  //entre hombro derecho y codo derecho
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_RIGHT_ELBOW);
  //entre codo derecho y mano derecha
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
SimpleOpenNI.SKEL_RIGHT_HAND);
  noStroke();//Quitar color de trazo
  fill(255, 0, 0);//Rellenamos de color rojo el trazo
  //Usamos la funcion drawJoint para dibujar los puntos de articulaciones
  DibujarArticulacion(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER);//Hombro derecho
  DibujarArticulacion(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW);//Codo derecho
  DibujarArticulacion(userId, SimpleOpenNI.SKEL_RIGHT_HAND);//Mano derecha
}
//Calcular angulo entre vectores
//Se requiere de las coordenadas del vector origen y los extremos
float CalcularAngulos(PVector one, PVector two, PVector centro){
  float angulo=0;//Iinicializamos la variable a retornar
  //Formula de calulo de angulo a=acos(A.B/|A|.|B|)
  PVector vector1=PVector.sub(centro,one);//vector desde el extremo 1 al origen
  PVector vector2=PVector.sub(centro,two);//vector desde el extremo 2 al origen
  float productoPunto=(vector1.x*vector2.x)+(vector1.y*vector2.y);//Producto punto entre los 2
vectores
  float magnitud1=sqrt(sq(vector1.x)+sq(vector1.y));//Magnitud del vector1
  float magnitud2=sqrt(sq(vector2.x)+sq(vector2.y));//Magnitud del vector2
  angulo=acos(productoPunto/(magnitud1*magnitud2));//Coseno inverso de la division de
productos
  return degrees(angulo);//retornamos la variable angulo en grados
}
//Calcular angulo entre vectores
//Depende de lo requerido seria extremo,centro,referencia
float AnguloEntreVectores(PVector A, PVector B, PVector referencia){//Recibe 2 vectores de
coordenadas y un vector de referencia
  PVector movil = PVector.sub(B,A);//Vector entre los 2 vectores de coordenada
  return degrees(PVector.angleBetween(movil,referencia));//Calcula en angulo en grados
entre el vector coordenadas y el vector orientacion
}
//Funcion del proceso(Al ser calculo de angulos es indiferente la coordenadas de donde se
realice)
public void GradosDeLibertad(int userId) {
```

```
  //Vector posicion de las articulaciones
  PVector rightHand = new PVector();//Generamos el vector de mano derecha
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, rightHand);
  PVector rightElbow = new PVector();//Generamos el vector de codo derecho
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW, rightElbow);
  PVector rightShoulder = new PVector();//Generamos el vector de hombro derecho
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
rightShoulder);
  PVector rightHip = new PVector();//Generamos el vector de cadera derecha
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HIP, rightHip);
  PVector leftShoulder = new PVector();//Generamos el vector de hombro izquierdo
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
leftShoulder);
  PVector leftHand = new PVector();//Generamos el vector de mano izquierda
  //Guardamos la posicion de la articulacion en el vector creado
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, leftHand);

  //Vector en coordenadas 2D
  PVector rightHand2D = new PVector(rightHand.x, rightHand.y);//Coordenadas X,Y de la
mano derecha
  PVector rightElbow2D = new PVector(rightElbow.x, rightElbow.y);//Coordenadas X,Y del
codo derecho
  PVector rightShoulder2D = new PVector(rightShoulder.x, rightShoulder.y);//Coordenadas
X,Y del hombro derecho
  PVector rightHip2D = new PVector(rightHip.x, rightHip.y);//Coordenadas X,Y de la cadera
  PVector leftShoulder2D = new PVector(leftShoulder.x, leftShoulder.y);//Coordenadas X,Y del
hombro izquierdo
  PVector rightHand2DHorizontal = new PVector(rightHand.x, rightHand.z);//Coordenadas X,Z
del hombro derecho

  //Calculo de angulos haciendo uso de la funcion CalcularAngulos
  RightshoulderAngle=CalcularAngulos(rightHip2D,rightElbow2D,rightShoulder2D);//Angulo
entre hombro y codo
  RightelbowAngle=CalcularAngulos(rightShoulder2D,rightHand2D,rightElbow2D);//Angulo
entre codo y mano
  //Angulo entre 2 planos del hombro derecho
  RightShoulderAngleHorizontal =
CalcularAngulos(leftShoulder2D,rightHand2DHorizontal,rightShoulder2D);

  //Rango de datos de angulo entre hombro y codo
  if (RightshoulderAngle <= 0) {//Si el angulo es menor o igual a 0
    RightshoulderAngle=10;//Se asigna el valor de 10
  }

  //Rango de datos de angulo entre codo y mano
  codo=map(RightelbowAngle,50,180,5,150);//Se escala los datos recibidos a datos que se
ajusten al servo del brazo
  if (codo <= 20) {//Si el angulo es menor o igual a 20
```

```
  codo=20;//Se asigna el valor de 10
 }
 if (codo > 180) {//Si es mayor a 180
  codo=170;//Se asigna el valor de 170
 }

 //Rango de datos de angulo entre hombro y mano en plano x,z
 //Se escala los datos recibidos a datos que se visualicen de mejor forma
 hombro=map(RightShoulderAngleHorizontal,70,115,45,170);

 //Escritura de datos
 fill(255);//Rellenamos de color blanco la letra a usar
 scale(1);//Escala 1
 textSize(40);//Tamaño de texto 40
 //Mostramos todos los angulos
 text("ANGULOS DE OPERACION",700,280);
 text("Hombro: "+ int(RightshoulderAngle) +", "+ int(hombro)+ "\n" + "Codo: " +
int(RightelbowAngle)+ "\n" + "Mano: " + hand,650,340);
 fill(0);//color negro
 text("COORDENADAS",480,520);
 textSize(30);//Tamaño de texto 30
 text("Hombro Derecho= X: "+ int(rightShoulder.x) +", Y: "+ int(rightShoulder.y)+", Z: "+
int(rightShoulder.z)+ "\n"
 + "Codo Derecho= X: "+ int(rightElbow.x) +", Y: "+ int(rightElbow.y)+", Z: "+ int(rightElbow.z)+
"\n"
 + "Mano Derecha= X: "+ int(rightHand.x) +", Y: "+ int(rightHand.y)+", Z: "+
int(rightHand.z),10,570);
 text("Hombro Izquierdo= X: "+ int(leftShoulder.x) +", Y: "+ int(leftShoulder.y)+", Z: "+
int(leftShoulder.z)+ "\n"
 + "Cadera Derecha= X: "+ int(rightHip.x) +", Y: "+ int(rightHip.y)+", Z: "+ int(rightHip.z)+ "\n"
 + "Mano izquierda= X: "+ int(leftHand.x) +", Y: "+ int(leftHand.y)+", Z: "+
int(leftHand.z),650,570);

 //Se escala los datos recibidos a datos que se ajusten al servo del brazo
 hombro=map(RightShoulderAngleHorizontal,70,115,170,45);

 if (hombro <= 45) {//Si el angulo es menor o igual a 45
  hombro=45;//Se asigna el valor de 45
 }
 if (hombro >= 180) {//Si el angulo es mayor o igual a 180
  hombro=180;//Se asigna el valor de 180
 }

 //Envio por MQTT
 delay(200);//Retardo de 200 ms para que no generar un cambio de angulo
 //Envio de datos(4 articulaciones), y estado de conexion hacia la estacion receptora por
medio de la funcion publish de MQTT

client.publish("moreiravasquez26102021",""+int(RightshoulderAngle)+","+int(RightelbowAngl
e)+","+int(hombro)+","+int(hands)+","+int(conexion));
}
```