



# UNIVERSIDAD POLITÉCNICA DE JUVENTINO ROSAS

## TESIS

“Desarrollo de un sistema de seguridad mediante un módulo de tracción embebido controlado a través de visión artificial”

PARA OBTENER EL TÍTULO DE  
**INGENIERO EN REDES Y TELECOMUNICACIONES**

*Presenta:*  
**ZAMUDIO OLVERA DIEGO EMMANUEL**

*Asesores:*  
M.I. Luis Rey Lara González  
Ing. Miguel Arreguín Juárez

Juventino Rosas, Gto. 30 de Abril de 2021.

# Agradecimientos

Primero que nada me gustaría agradecer a mi universidad y a todo el personal que la conforma, ya que sin su apoyo, jamás hubiese llegado hasta este punto que hace 3 años y 8 meses sólo era un sueño.

-Agradezco encarecidamente a mi profesor y asesor en todos estos años, el M.I. Luis Rey Lara González por haberme enseñado tantas cosas a lo largo de todo este tiempo y por haberse acoplado a mi forma de trabajar. También por siempre confiar en mis capacidades y nunca dudar de que daría un resultado aceptable. Gracias por tantos regaños y enseñanzas, usted me enseñó que cada lección es un obsequio.

-También agradezco a mi mentora María de los Ángeles Arellano Vera por haberme compartido y confiado su proyecto, ya que sin ella jamás hubiera logrado todo esto. Muchas gracias por confiar en que podría sacar a adelante esto y por darme todas las herramientas que, en su momento, te costaron tiempo y esfuerzo conseguir.

-Agradezco a mi familia porque siempre me apoyó y creyó en mí ciegamente sin importar cómo me encontrara en ese momento y por siempre hacer hasta lo imposible para que yo pudiera seguir adelante.

-Especialmente a mi padre, que sin importar mi comportamiento, mi constante falta de cercanía con él y mi exigencia en cuanto a peticiones de todo tipo, siempre me apoyó sin negarse en ningún momento. Por haberse sacrificado durante tantos años para que yo lograra este objetivo en mi vida y por siempre encontrar la manera de solucionar los problemas que se me presentaban. Muchas gracias, papá y aunque no muy seguido lo digo, sabes que te amo mucho.

-A mi madre por seguir apoyándome incondicionalmente a pesar de la distancia y por nunca dudar de mis capacidades. Por haber tomado la difícil decisión de alejarse de sus hijos para poder darles un futuro mejor. Gracias por ser la mejor mamá del mundo, te amo.

-A mis 4 hermanos: Mauricio, Jesús, Christopher y Mario; ya que gracias a ellos tuve la motivación y apoyo moral que necesitaba en todo momento y por siempre verme como alguien capaz de cualquier cosa. Gracias por tantas risas y momentos felices juntos. Los amo.

-A mi tía Juana, mi segunda mamá. Que a pesar de que ya no se encuentre entre nosotros,

ella siempre creyó en mí y en que lograría cumplir todo lo que me propusiera sin dudar. Gracias por quererme como si fuera tu hijo, yo me sentí como si fuera tu hijo durante tantos años. Se logró, Tía.

-A mis dos mejores amigas:

- A Aranzazú, mi Pecosa. Por siempre estarme motivando y ayudando a que cumpliera con mis responsabilidades y por siempre tener tiempo para escuchar mis quejas y reclamos de la vida. Por siempre intentar ser su mejor versión de sí para mí y por haberme aguantado por tantos años. Muchas gracias, Pecosa.
- A Andrea, por ser la única persona que ha dejado cosas de lado por estar cuando más la necesitaba y por siempre escucharme en aquellas tardes viviendo juntos. Gracias por siempre ser la voz de la razón y por ayudarme siempre a no perder mi camino. Muchas gracias, Andy.

-A mis amigos Alan, Eduardo, Braulio, Alejandro y Jennifer por siempre sacarme una sonrisa, por ser tan incondicionales conmigo y ayudarme cada que se los pedía. Por tantos momentos juntos y por siempre escucharme cuando lo necesitaba.

-A todos mis amigos del "Grupo sin Tópico", ya que ellos han sido parte importante de mi desarrollo personal a lo largo de toda esta travesía llamada carrera universitaria. Mi vida ha sido más amena, interesante y alegre gracias a ellos y jamás tendré las palabras suficientes como para expresarles lo agradecido que estoy con ustedes.

-A mi mascota Neeko, que aunque no sepa realmente lo que hace, ha sido un gran apoyo emocional en todo este tiempo de contingencia, la he visto crecer y convivir diariamente con mi familia, además de que siempre está conmigo cuando estoy decaído. Muchas gracias, Neeko.

-A mi amiga más actual Vianney, que a pesar del poco tiempo que llevamos conociéndonos, siempre ha estado ahí para darme ánimos y palabras de aliento cada que le he contado mis metas y objetivos. Gracias por tu interés tan genuino y por ser un amor de persona.

-Y por último pero no menos importante, me gustaría darle gracias a todas aquellas personas que tal vez no mencione aquí textualmente pero que ellas saben lo mucho que confiaron en que podía lograrlo, en verdad gracias por tanto.

# Índice general

<b>1. Introducción</b>	<b>7</b>
1.1. Justificación . . . . .	8
1.2. Objetivos . . . . .	8
1.2.1. Objetivo general . . . . .	8
1.2.2. Objetivos específicos . . . . .	8
1.3. Estado del Arte . . . . .	9
1.3.1. Detección de movimiento mediante Python + OpenCV . . . . .	9
1.3.2. Freenect . . . . .	9
1.3.3. OpenNI . . . . .	10
1.3.4. Microsoft Kinect SDK . . . . .	10
1.3.5. Wavi Xtion . . . . .	10
1.3.6. OpenNI2 . . . . .	10
<b>2. Marco Teórico</b>	<b>11</b>
2.1. Kinect . . . . .	11
2.1.1. Componentes del Kinect . . . . .	12
2.1.2. Especificaciones de Hardware . . . . .	13
2.1.3. Kinect Xbox One . . . . .	14
2.1.4. Mejoras . . . . .	14
2.2. SDK Kinect . . . . .	16
2.3. Processing . . . . .	16
2.4. JAVA . . . . .	17
2.5. Arduino . . . . .	17
2.6. Raspberry PI . . . . .	18
2.6.1. Especificaciones técnicas . . . . .	19
2.6.2. GPIO's . . . . .	20
2.7. Puente H . . . . .	22

<b>3. Implementación</b>	<b>23</b>
3.1. Primera fase . . . . .	23
3.1.1. Raspberry PI y módulo de tracción . . . . .	23
3.2. Segunda fase . . . . .	25
3.2.1. Raspbian with Processing . . . . .	25
3.2.2. Configuración de Raspberry para el uso óptimo de Kinect . . . . .	25
3.3. Tercera fase . . . . .	25
3.3.1. Pruebas con processing y Kinect . . . . .	25
3.3.2. Desfragmentación del código . . . . .	25
3.3.3. Primeras pruebas. . . . .	26
<b>4. Resultados y Conclusión</b>	<b>28</b>
4.1. Resultados . . . . .	28
4.2. Conclusión . . . . .	30
4.2.1. Planes a futuro . . . . .	30
<b>Bibliografía</b>	<b>31</b>
<b>A. Código Módulo de tracción</b>	<b>32</b>
<b>B. Montura del SO Raspbian with Processing en Raspberry PI 3B</b>	<b>35</b>
B.0.2. Montura del SO utilizando Balena Etcher . . . . .	35
<b>C. Configuración de Raspberry y Kinect</b>	<b>39</b>
<b>D. Desfragmentación del código</b>	<b>41</b>
D.0.3. Código Completo. . . . .	47

# Resumen

Los sistemas de grabación mediante CCTV tienen la limitante de ser estáticos y sólo cubrir ciertas áreas de la zona que se desea vigilar, dando lugar a puntos ciegos en la grabación y a ángulos muy pobres.

Este proyecto tiene como objetivo principal mejorar los sistemas de vigilancia convencionales por uno más completo y eficaz, es decir, que pueda hacer más acciones que los actuales utilizando variedad de tecnologías y dispositivos como lo son Java, Kinect y Raspberry PI.

La metodología utilizada para realizar este proyecto fue una conocida como En cascada, la cual nos ayuda a visualizar de mejor manera el avance del proyecto mediante la experimentación y la tan conocida prueba y error, de esta forma se pudieron recabar datos importantes para retroalimentar activamente el desarrollo del presente proyecto.

En consecuencia a esto, se logró analizar, sintetizar y manipular los algoritmos que nos ofrece Kinect (Skeleton Tracking, Depth Threshold) dependiendo de las necesidades del proyecto, en este caso, para la toma de decisiones dependiendo de lo que detecten los sensores de Kinect. Además, se logró la conexión exitosa de la primera versión de Kinect con Raspberry PI y la activación de distintos dispositivos mediante el uso de los GPIO's de la tarjeta antes mencionada. Esto nos deja la pauta de poder realizar distintas tareas según sea requerido y, de esta manera, accionar distintos sistemas de seguridad en caso de estar frente a una amenaza, en este caso, de un intruso dentro del área que se deseé vigilar.

# Abstract

CCTV recording systems have the limitation of being static and only cover certain areas of the area to be monitored, resulting in blind spots in the recording and very poor angles.

The main objective of this project is to improve conventional surveillance systems for a more complete and efficient one, that is, to be able to do more than the current ones using a variety of technologies and devices such as Java, Kinect and Raspberry PI.

The methodology used to carry out this project was known as Cascade, which helps us to better visualize the progress of the project through experimentation and the well-known trial and error, in this way important data could be collected for feedback actively developing this project.

Consequently, it was possible to analyze, synthesize and manipulate the algorithms that Kinect offers us (Skeleton Tracking, Depth Threshold) depending on the needs of the project, in this case, for decision making depending on what the Kinect sensors detect. In addition, the successful connection of the first version of Kinect with Raspberry PI was achieved and the activation of different devices through the use of the GPIO's of the aforementioned card. This leaves us with the guideline of being able to carry out different tasks as required and, in this way, activate different security systems in case of being faced with a threat, in this case, from an intruder within the area to be monitored.

# **Capítulo 1**

## **Introducción**

Hoy en día la tecnología ha permitido nuevas formas de interactuar con los dispositivos digitales. Con el éxito y la producción en masa de las nuevas tecnologías en dispositivos digitales ha permitido que el humano tenga a su alcance éstas y pueda manipularlas a su conveniencia.

Este es el caso de tan conocido dispositivo llamado Kinect lanzado por la empresa Microsoft para la consola Xbox 360, con el cual actualmente se está manipulando no sólo para consolas de videojuegos, sino también para el desarrollo de proyectos debido a que contiene cámaras y un sensor infrarrojo que ofrece imágenes de profundidad, además de tener la capacidad de interpretar los movimientos que se registran en los objetos capturados por la cámara de Kinect en eventos que se pueden proyectar en pantalla y actualmente muchos desarrolladores han adaptado el dispositivo para que pueda ser utilizado desde un computadora gracias al adaptador a puerto USB, para trabajar con él y darle otros usos que no fueran únicamente para la consola de Microsoft Xbox 360.

## **1.1. Justificación**

En diferentes áreas ya sean públicas o particulares existe la necesidad de un sistema de vigilancia, esto hasta la fecha se limita principalmente a sistemas tradicionales como lo son cámaras de seguridad de circuito cerrado (CCTV), por el costo que conlleva algún componente adicional. Si a esto le agregamos la falta de movilidad en algunos de estos sistemas, encontraremos un gran área sin explotar, la cuál se podría mejorar con un sistema móvil de vigilancia.

## **1.2. Objetivos**

### **1.2.1. Objetivo general**

Desarrollar un módulo de tracción controlado con Raspberry mediante procesamiento de imagen a través de visión artificial.

### **1.2.2. Objetivos específicos**

- Generar una vía de comunicación entre Raspberry y Kinect v2.
- Desarrollar algoritmos para el procesamiento de imágenes.
- Desarrollar algoritmos para la comunicación entre Raspberry y Kinect v2.

## 1.3. Estado del Arte

### 1.3.1. Detección de movimiento mediante Python + OpenCV

En muchas aplicaciones basadas en la visión artificial, se utiliza la detección de movimiento. Por ejemplo, cuando queremos contar las personas que pasan por un determinado lugar. En este caso, lo primero que se hace es extraer las personas que hay en la escena. Existen diferentes técnicas, métodos o algoritmos que posibilitan la detección de movimiento. Al igual que en otras materias, en la visión artificial no hay casos genéricos. Dependerá de cada situación usar uno u otro. [2]

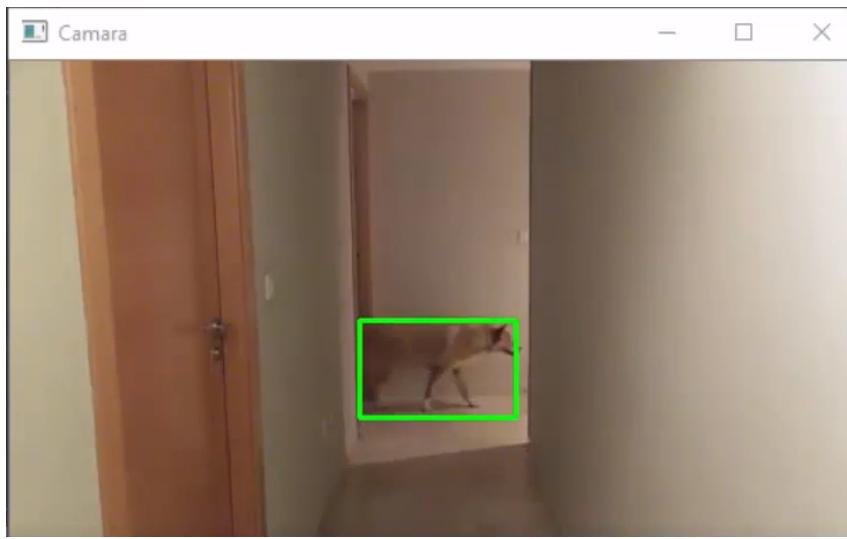


Figura 1.1: Detección de movimiento usando Python + OpenCV.

### 1.3.2. Freenect

En Noviembre de 2010, días después de llegar el kinect al mercado, libfreenect (el resultado del trabajo de H. Martín para el concurso que lanza Adafruit) aparece en escena y da pie OpenKinect como un proyecto abierto que ofrece un driver (la aplicación de bajo nivel que hace de interfaz entre el computador y un periférico) así como un conjunto de funciones para el desarrollador. También aparece una primera versión del grupo NUI, especialistas en proyectos relacionados con interfaces multitáctiles. [5]

### **1.3.3. OpenNI**

En Diciembre de 2010, el proyecto OpenNI ve la luz a partir de que PrimeSense libera sus drivers y una biblioteca de funciones capaz de detección de movimiento (NITE) [5]

### **1.3.4. Microsoft Kinect SDK**

En Febrero de 2012 aparece la primera versión del driver y el soporte de ejecución para un computador por parte de Microsoft. En Mayo aparece la primera versión del SDK. [5]

### **1.3.5. Wavi Xtion**

En 2012 Prime Sense y Asus unirían fuerzas para crear un dispositivo más pequeño en tamaño y consumo, el Wavi Xtion. [5]



Figura 1.2: Wavi-Xtion de Asus.

### **1.3.6. OpenNI2**

En Noviembre de 2013, aparece el proyecto OpenNI2 propiciado por Occipital (junto con el desarrollo del Structure Sensor) tras la adquisición de PrimeSense por Apple y el cierre del sitio web de OpenNI. [5]

# Capítulo 2

## Marco Teórico

En este capítulo se presentan los conceptos fundamentales para el desarrollo de este proyecto para su futura implementación.

### 2.1. Kinect

Kinect para Xbox 360, inicialmente conocido por el nombre clave Project Natal es un periférico para videojuegos que prescinde de mandos gracias a un sensor de detección de movimientos, creado por Microsoft y está previsto que sea utilizable en computadoras con el sistema operativo Windows 8 en adelante. Está basado en una cámara periférica que se conecta a la videoconsola Xbox 360 reconociendo los gestos del jugador, su rostro, voz, así como sus movimientos y los objetos estáticos dentro un campo visual.



Figura 2.1: Kinect

### 2.1.1. Componentes del Kinect

- Vídeo
  - Cámara CMOS de color
  - Cámara CMOS infrarrojo (IR)
  - Proyector de infrarrojos - 830nm, diodos láser de 60mW.
- Audio
  - Cuatro micrófonos
  - Control de inclinación
- Motor
  - Acelerómetro (3 ejes)
  - Procesadores y memoria
  - Chip PrimeSense PS1080-A2



Figura 2.2: Hardware del Kinect

## 2.1.2. Especificaciones de Hardware

Físicamente el sensor Kinect es una barra horizontal, de aproximadamente 23 centímetros, la cual está conectada a una pequeña base circular que posee un motor que permite el movimiento del sensor en el eje vertical aproximadamente unos  $27^{\circ}$ . El sensor está hecho para ser colocado longitudinalmente por encima o debajo de una pantalla de video. Posee 2 cámaras y un proyector infrarrojo. En la imagen siguiente se puede apreciar el primer componente de la izquierda es el proyector infrarrojo (IR Laser), el componente central es un Color Complementary Metal Oxide Semiconductor (CMOS), es decir una simple cámara RGB con una resolución de 640x480, 32bits de color y 30fps y finalmente el componente de la derecha es el IR CMOS o receptor IR con una resolución de 320x240, este sensor permite ver la habitación en 3D bajo cualquier condición de luz ambiental.

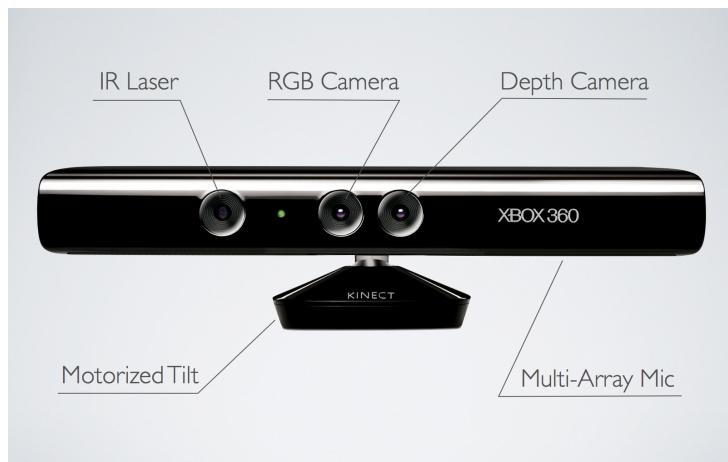


Figura 2.3: Cámaras y sensores del Kinect

Kinect posee además en la parte inferior un micrófono multiarray capaz de detectar voces y eliminar el sonido ambiente. El array del micrófono tiene cuatro cápsulas, y opera con cada canal procesando 16-bit de audio con un radio de frecuencia de 16 kHz. Para el correcto funcionamiento de las cámaras y sensores de Kinect se requiere de un espacio ideal de:

1.2m a - 3.5m de distancia entre el objetivo y el sensor Kinect.

Campo de visión horizontal de 58 grados.

Campo de visión vertical de 45 grados.

### 2.1.3. Kinect Xbox One

## Kinect for Windows v2



Figura 2.4: Cámaras y sensores del Kinect

### 2.1.4. Mejoras con respecto a antiguas versiones

Son muchas las diferencias que hay con su antecesor y muchas las posibilidades de desarrollo que se generan con ellas.

- Mayor campo de visión
  - 70° en horizontal (antes 57°) y 60 en vertical (antes 43°).
  - Esto permite poder detectar a más personas dentro de un mismo campo de visión. Hasta 6 personas pueden ser detectados simultáneamente.
  - Cabe destacar que esta versión de Kinect no tiene motor de inclinación.
- Mayor resolución
  - 1920 x 1080 Full HD (antes 640 x 480).
  - Permite detectar con más precisión todo el entorno.
  - Capacidad de diferenciar la orientación del cuerpo incluyendo sus manos y pudiendo diferenciar sus dedos.

- El Face tracking tiene mucho más detalle y permite
  - captar los gestos de la cara.
  - Más calidad de imagen.
- Mejora el rango de profundidad del sensor:
  - El rango de actuación pasa a ser de 0,5 a 4,5 metros.
- USB 3.0:
  - Al aumentar la velocidad de la comunicación con el ordenador los datos fluyen más rápido y esto disminuye la latencia del sensor. Pasa de 90ms a 60ms.
- Mejora de la captación de sonidos
  - Esta versión de Kinect viene dotada de una gran mejora en cuanto al reconocimiento de voz y la captación de sonidos. Se ha mejorado la eliminación del ruido ambiente y esto permite captar con más detalle las instrucciones vocales.
- Captación de movimiento a oscuras
  - Ahora Kinect v2 es capaz de reconocer y captar los movimientos, aunque la sala este a oscuras.
- Kinect 2 permite calcular/analizar la fuerza de nuestros músculos y medir el ritmo cardíaco.

## 2.2. SDK Kinect

El SDK (Software Development Kit) se trata de una librería que nos facilita diferentes funciones que nos ayudan a interactuar con el dispositivo Kinect.



Figura 2.5: Logo de la librería SDK.

## 2.3. Processing

Es un software flexible y un lenguaje para aprender a codificar dentro del contexto de las artes visuales. Desde 2001, Processing ha promovido la alfabetización de software dentro de las artes visuales y la alfabetización visual 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.



Figura 2.6: Software Processing

## 2.4. JAVA

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems.

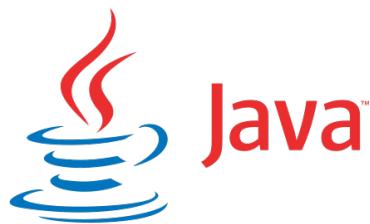


Figura 2.7: Logo de Java.

## 2.5. Arduino

Arduino es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador re-programable y una serie de pines hembra. Estos permiten establecer conexiones entre el microcontrolador y los diferentes sensores y actuadores de una manera muy sencilla (principalmente con cables jumper).

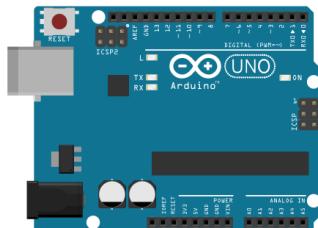


Figura 2.8: Tarjeta Arduino.

## 2.6. Raspberry Pi 3 B

Raspberry Pi es una computadora de placa reducida, una computadora de placa única o computadora de placa simple (SBC) de bajo costo desarrollado en el Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de informática en las escuelas.

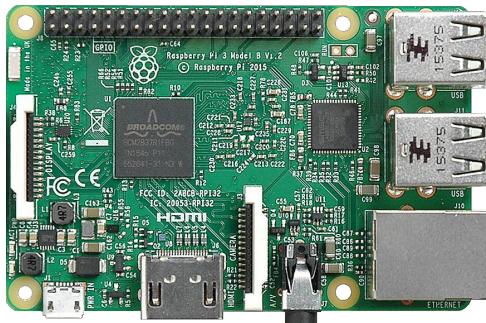


Figura 2.9: Raspberry PI 3B.

## 2.6.1. Especificaciones técnicas

- Procesador
  - • Quad Core 1.2GHz Broadcom BCM2837 ARM (64bit) CPU.
- Tarjeta de red
  - • BCM43438 LAN inalámbrica y Bluetooth Low Energy (BLE) a bordo.
- RAM
  - 1GB LPDDR2
- Sistema operativo
  - Linux.
- Dimensiones
  - 85 x 56 x 17mm
- Power
  - Fuente de alimentación Micro USB conmutada actualizada de hasta 2.5A
- Conectores
  - 100 Base Ethernet
  - GPIO extendido de 40 pines
  - 4 puertos USB 2
  - Salida de 4 polos estéreo y puerto de video compuesto
  - HDMI de tamaño completo
  - Puerto de cámara CSI para conectar una cámara Raspberry Pi
  - Puerto de pantalla DSI para conectar una pantalla táctil Raspberry Pi
  - Puerto micro SD para cargar su sistema operativo y almacenar datos

## 2.6.2. GPIO's

Los General Puporse Input Output (GPIO's) de Raspberry PI, se dividen en pines y la forma correcta de leerlos es la siguiente.

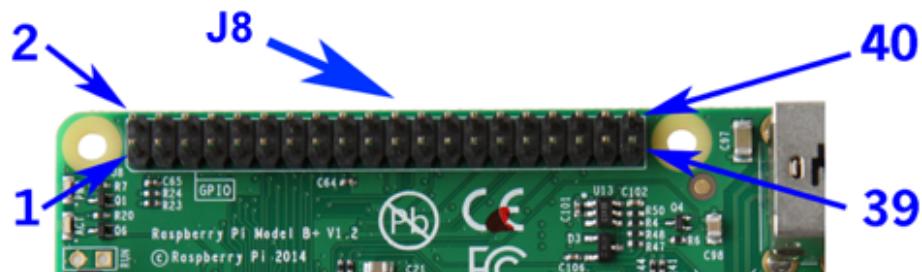


Figura 2.10: GPIO's de Raspberry PI B+ reales.

De estas terminales no todas son entradas o salidas digitales, algunas son específicamente para GND o voltaje.

Raspberry Pi 3 Model B (J8 Header)				
GPIO#	NAME		NAME	GPIO#
	3.3 VDC Power		2	5.0 VDC Power
<b>8</b>	GPIO 8 SDA1 (I2C)		4	5.0 VDC Power
<b>9</b>	GPIO 9 SCL1 (I2C)		6	Ground
<b>7</b>	GPIO 7 GPCLK0		8	GPIO 15 TxD (UART) <b>15</b>
	Ground		10	GPIO 16 RxD (UART) <b>16</b>
<b>0</b>	GPIO 0		12	GPIO 1 PCM_CLK/PWM0 <b>1</b>
<b>2</b>	GPIO 2		14	Ground
<b>3</b>	GPIO 3		16	GPIO 4 <b>4</b>
	3.3 VDC Power		18	GPIO 5 <b>5</b>
<b>12</b>	GPIO 12 MOSI (SPI)		20	Ground
<b>13</b>	GPIO 13 MISO (SPI)		22	GPIO 6 <b>6</b>
<b>14</b>	GPIO 14 SCLK (SPI)		24	GPIO 10 CE0 (SPI) <b>10</b>
	Ground		26	GPIO 11 CE1 (SPI) <b>11</b>
<b>30</b>	SDA0 (I2C ID EEPROM)		28	SCL0 (I2C ID EEPROM) <b>31</b>
<b>21</b>	GPIO 21 GPCLK1		30	Ground
<b>22</b>	GPIO 22 GPCLK2		32	GPIO 26 PWM0 <b>26</b>
<b>23</b>	GPIO 23 PWM1		34	Ground
<b>24</b>	GPIO 24 PCM_FS/PWM1		36	GPIO 27 <b>27</b>
<b>25</b>	GPIO 25		38	GPIO 28 PCM_DIN <b>28</b>
	Ground		40	GPIO 29 PCM_DOUT <b>29</b>

**Attention!** The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

Figura 2.11: Gráfica de GPIO's de Raspberry PI 3B+.

## 2.7. Puente H

Se realizó un diseño preliminar del módulo de tracción, este funciona con distintos componentes, entre ellos está un módulo L298N, con éste se logra controlar cuatro motores de corriente directa a 5V.

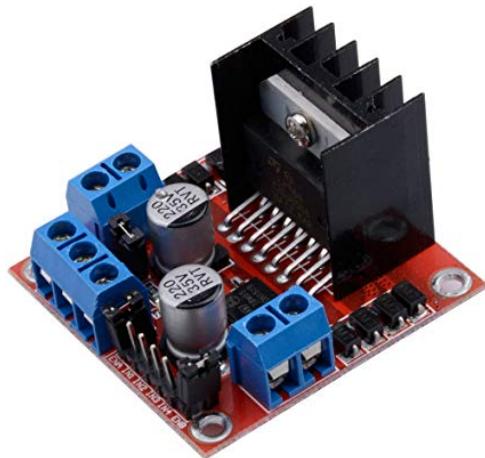


Figura 2.12: Módulo L298N.

Según el datasheet del componente, este puede manejar un voltaje de 44V corriente directa y 4 amperes, por lo cual funciona con un voltaje de 5 Volts. Esto resulta ideal para el proyecto, ya que, por sus capacidades, tiene mejor control al manejar todos los motores implementados.

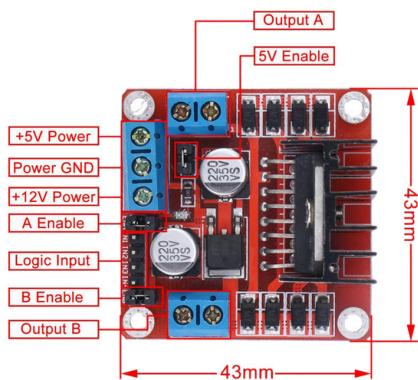


Figura 2.13: Entradas y salidas del L298N.

# Capítulo 3

## Implementación

### 3.1. Primera fase

#### 3.1.1. Raspberry PI y módulo de tracción

Para la realización del módulo de tracción se utilizó un cuerpo armable especial para proyectos electrónicos. Este cuenta con 4 motores de corriente directa que funcionan a 5V. Se le agregó el módulo L298N en una posición estratégica para tener un acceso rápido y cómodo a las terminales de este. En la parte superior del módulo de tracción colocamos la Raspberry Pi junto a una protoboard. Además, se agregaron unas baterías recargables de 7V en total, para encender la tarjeta Raspberry PI y una batería de 9V para la activación de los motores más un reductor de voltaje.

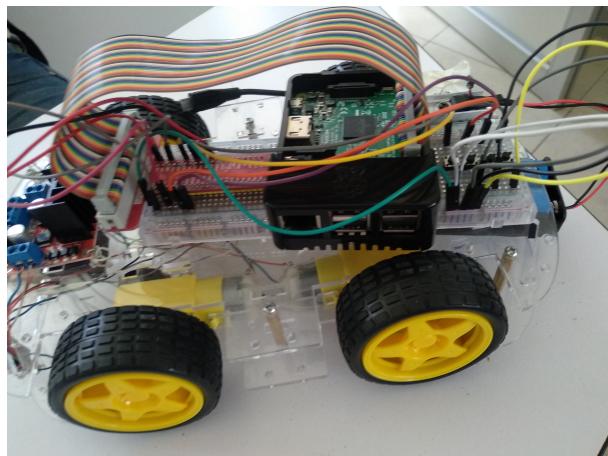


Figura 3.1: Resultado preliminar módulo de tracción.

El modulo L298N recibe señales a través de sus cuatro entradas, mediante el código desarrollado en IDE Processing utilizando lenguaje Java. Posteriormente se realizó la conexión directa del módulo, ya que las salidas de la Raspberry PI 3 B son digitales. Dicho módulo detecta cuando una entrada esta “Alta” o “Baja”, activando sus salidas con su respectiva configuración, en este caso se utilizaron cuatro GPIO'S para el control del módulo: GPIO 1, GPIO 2, GPIO 3 y GPIO 4.

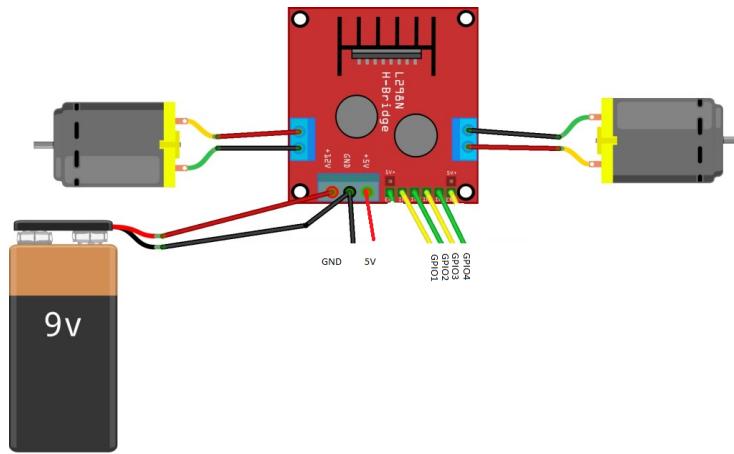


Figura 3.2: Diagrama de conexión del L298N

En el anexo A se muestra el código en Java usado para configurar los motores del módulo de tracción mediante el puente H L298N.

## **3.2. Segunda fase**

### **3.2.1. Raspbian with Processing**

El proceso de instalación del SO Raspbian with Processing se explica detalladamente en el anexo B.

### **3.2.2. Configuración de Raspberry para el uso óptimo de Kinect**

En el anexo C se explica detalladamente cómo realizar la configuración.

## **3.3. Tercera fase**

### **3.3.1. Pruebas con processing y Kinect**

Como se mencionaba anteriormente, existen diferentes algoritmos que fueron realizados para poder trabajar con el dispositivo Kinect, desarrollados por diferentes investigadores. Uno de estos utiliza la tecnología “Skeleton tracking”, en el cual se pueden observar diferentes librerías y funciones que permiten interpretar algunas articulaciones del cuerpo, así como seccionar partes del cuerpo, como mano derecha, mano izquierda, cabeza, piernas etc.

Una de las principales tareas que se realizaron fue descomponer el código y funciones para el entendimiento de cada una de sus partes, para que, de esta manera, se pudiera comprender éstas de una manera sencilla y eficaz para su posterior uso.

### **3.3.2. Desfragmentación del código**

En el anexo D se explica detalladamente la desfragmentación del código.

### 3.3.3. Primeras pruebas.

La tecnología de Skeleton Tracking es capaz de identificar el estado de las manos, no de la manera más precisa ya que sólo logra interpretar los pulgares y a los demás dedos como uno solo, pero siendo capaz de interpretar 3 estados de estas. Los cuales son:

- **Mano abierta, coloreando esta con un color verde.**



Figura 3.3: Detección de mano abierta con Kinect v2.

- **Mano haciendo un especie de lazo con los dedos, coloreándola de un color azul.**

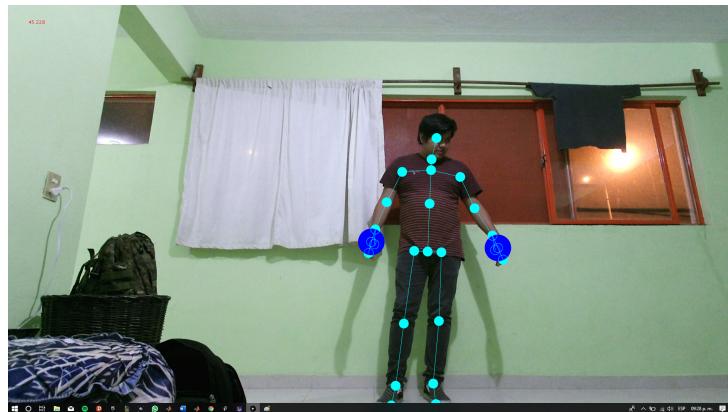


Figura 3.4: Detección de mano 'Lasso' con Kinect v2.

- Existe un tercer estado, si así puede llamarse, que es cuando kinect no es capaz de detectar o interpretar las manos de la persona, coloreando estas de un color blanco.

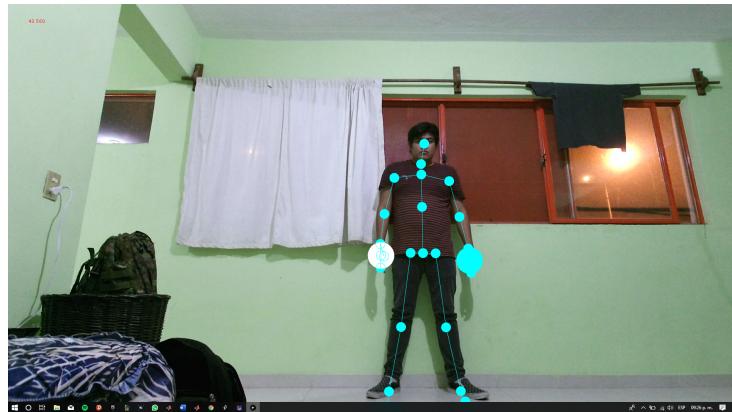


Figura 3.5: No detección de manos con Kinect v2.

Como puede ser visible las palmas de las manos cambian de color dependiendo la figura que formamos o el estado, con ayuda de esta y de forma independiente logramos agregar una integración de manera electrónica, con encendido de cuatro leds.

# Capítulo 4

## Resultados y Conclusión

### 4.1. Resultados

Gracias a la experimentación y a la metodología en cascada que se aplicó en este proyecto, se logró entender e indentificar cada parte de los scripts utilizados y, por consecuencia, se realizaron distintas pruebas con éstas, logrando así tomar decisiones dependiendo de lo que Kinect esté detectando.

Utilizando las diferentes posturas de manos se logró encender mediante un Arduino una serie de 3 leds. Estos encendían dependiendo de la postura de mano que Kinect lograra identificar.

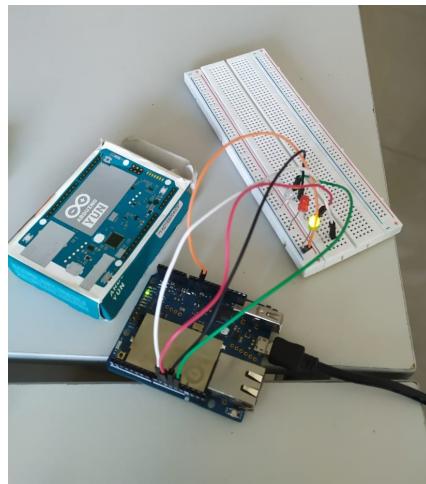


Figura 4.1: Activación de un led mediante la detección de manos con Kinect v2 'Mano Lasso'.

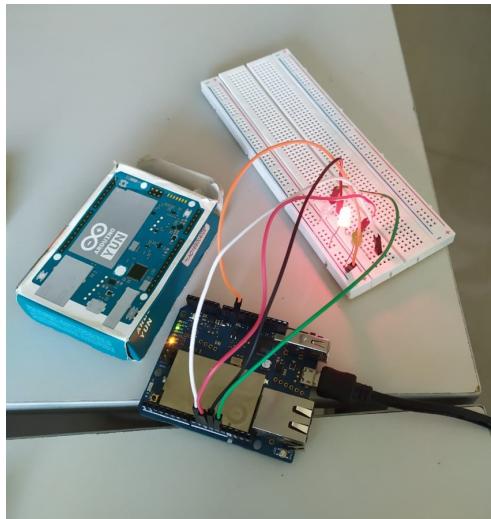


Figura 4.2: Activación de un led mediante la detección de manos con Kinect v2 'Mano Abierta'.

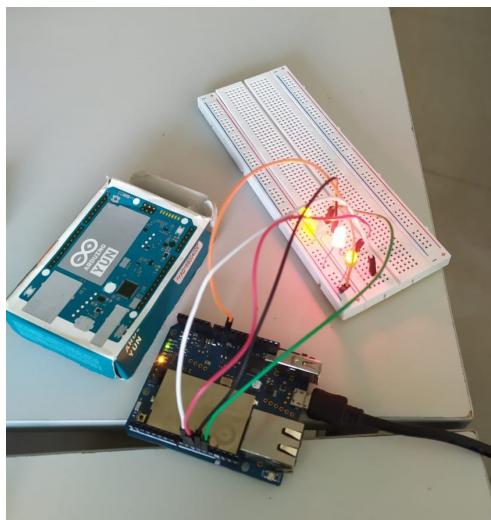


Figura 4.3: Activación de un led mediante la detección de manos con Kinect v2 'Mano No Detectada'.

## 4.2. Conclusión

El uso de un dispositivo como lo es Kinect para otros tipos de ámbitos además del entretenimiento y ocio es bastante amplio, sin contar con la posibilidad de utilizarlo en una plataforma de open source como lo es Raspberry.

Este pequeño avance da pauta a muchísimas posibilidades, como por ejemplo, tomar capturas de la imagen que da Kinect en el momento justo que detecte a un intruso delante de sus sensores y, posteriormente, enviarla mediante domótica hacia algún dispositivo final o guardarla dentro de un servidor.

Se encontraron varias limitantes en cuanto a la implementación, las cuales son:

- Una de las más obvias y por la que pasamos todos a lo largo de este año fue la pandemia, ya que a consecuencia de esta, no se pudo seguir trabajando con el sistema, puesto que este se encontraba dentro de la universidad y era muy complicado tener acceso a él.
- Instalar Kinect en “Raspberry PI 3B”, ya que varía en cuanto a paqueterías. Esto se pudo solucionar después de investigar distintos usos de este, probando distintos scripts y utilizando las partes que fueran útiles para llegar al objetivo.
- La tarjeta programable Raspberry PI 3B no es lo suficientemente potente para realizar las tareas que se le designaban, por lo cual, es necesario una actualización de hardware por una tarjeta programable Raspberry PI más reciente y/o con mayores capacidades.
- La capacidad de carga de los motores implementados es muy reducida. Estos no eran capaces de mover la suma del peso de todos los componentes de los que consta el sistema en sí.

### 4.2.1. Planes a futuro

Como planes a futuro se tiene contemplado adquirir el hardware requerido para que el sistema funcione de la manera más óptima posible, además, se trabajará en desarrollar un algoritmo para la toma de decisiones al momento de detectar a algún ser humano dentro del campo de visión de Kinect y a su vez crear una vía de comunicación entre el sistema y una red de área local para que, de esta manera, se genere un sistema de seguridad inteligente.

# Bibliografía

- [1] Fry B. & Reas C.. (2001). *Processing for PI*. Septiembre 2018, de Processing Sitio web: <https://pi.processing.org/>
- [2] Del Valle, L. (Julio 11, 2010). *Detección de movimiento con Raspberry Pi*. Noviembre 30, 2018, de Luis Del Valle Hernández Sitio web: [programarfacil.com](http://programarfacil.com)
- [3] Scott M.. (2015). *Introduction to Processing*. Marzo 2019, de Raspberry PI Foundation Sitio web: <https://projects.raspberrypi.org/en/projects/introduction-to-processing/10>
- [4] Cong R. & Winters R.. (2011). *How Does The Xbox Kinect Work*. Noviembre 2018, de Jameco Electronics Sitio web: <https://www.jameco.com/jameco/workshop/howitworks/xboxkinect.html>
- [5] Agustí i Melchor M . (2017). *Introducción. En Uso del Kinect en el computador desde el punto de vista del usuario(p.4)*. Valencia, España: Universitat Politècnica de València.
- [6] Murillo A. (2017). ¿Qué es el el SDK del dispositivo Kinect de Microsoft?. octubre, 2019, de Kinect Developers Sitio web: <http://www.kinectfordevelopers.com/es/2012/11/06/que-es-el-sdk-de-microsoft/>
- [7] Anónimo (2017), What is Raspberry Pi?, Course Hero, Inc. Octubre 2019, <https://www.coursehero.com/file/p2911ls/Raspberry-Raspberry-Pi-en-a-computer-board-reduced-computer-board/>
- [8] White Chris (2016)., Kinect V2 disconnects and reconnects on recent Intel chips – Solution., Microsoft. Febrero 2020, <https://social.msdn.microsoft.com/Forums/en-US/7a9f139d-85fe-4e68-9ad2-5456f8e1f173/kinect-v2-disconnects-and-reconnects-on-recent-intel-chips-solution?forum=kinectv2sdk&fbclid=IwAR0m2Gqobq6KX6fhhH3mM452hR8yD4BcqJX9SmE6oKWMd5r8wTlJeU1fTC>

## Anexo A

### Código Módulo de tracción

```
import com.pi4j.io.gpio.*;
import com.pi4j.io.gpio.event.GpioPinDigitalStateChangeEvent;
import com.pi4j.io.gpio.event.GpioPinListenerDigital;

public class Carro_fase_1 {

    public static void main(String args[]) throws InterruptedException {

        // Se crea un controlador para los gpio
        final GpioController gpio = GpioFactory.getInstance();

        // Se define al gpio pin #2 como una entrada cuando su resistor interno "pull down" es activado
        final GpioPinDigitalInput btn1 = gpio.provisionDigitalInputPin(RaspiPin.GPIO_01, PinPullResistance.PULLDOWN);
        final GpioPinDigitalInput btn2 = gpio.provisionDigitalInputPin(RaspiPin.GPIO_02, PinPullResistance.PULLDOWN);
        final GpioPinDigitalInput btn3 = gpio.provisionDigitalInputPin(RaspiPin.GPIO_03, PinPullResistance.PULLDOWN);
        final GpioPinDigitalInput btn4 = gpio.provisionDigitalInputPin(RaspiPin.GPIO_04, PinPullResistance.PULLDOWN);
        final GpioPinDigitalOutput pin1 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_05, "MyLED", PinState.LOW);
        final GpioPinDigitalOutput pin2 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_06, "MyLED", PinState.LOW);
        final GpioPinDigitalOutput pin3 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_07, "MyLED", PinState.LOW);
        final GpioPinDigitalOutput pin4 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_08, "MyLED", PinState.LOW);

        // configura los pines de entrada como apagados
```

```

btn1.setShutdownOptions(true);
btn2.setShutdownOptions(true);
btn3.setShutdownOptions(true);
btn4.setShutdownOptions(true);
// crea y registra el gpio pin listener
btn1.addListener(new GpioPinListenerDigital() {
@Override
public void handleGpioPinDigitalStateChangeEvent(
    GpioPinDigitalStateChangeEvent event) {
// Muestra el estado del pin en la consola
System.out.println("PRIMER_BOTON_1");
pin1.toggle();
pin2.low();
pin3.low();
pin4.toggle();

}

});

btn2.addListener(new GpioPinListenerDigital() {
@Override
public void handleGpioPinDigitalStateChangeEvent(
    GpioPinDigitalStateChangeEvent event) {
System.out.println("PRIMER_BOTON_2");
pin1.low();
pin2.toggle();
pin3.toggle();
pin4.low();

}

});

btn3.addListener(new GpioPinListenerDigital() {
@Override
public void handleGpioPinDigitalStateChangeEvent(
    GpioPinDigitalStateChangeEvent event) {
System.out.println("PRIMER_BOTON_3");
pin1.low();
pin2.toggle();
pin3.low();
pin4.toggle();

}

});

```

```

// 
btn4.addListener(new GpioPinListenerDigital() {
@Override
public void handleGpioPinDigitalStateChangeEvent(
    GpioPinDigitalStateChangeEvent event) {
System.out.println("PRIMER_BOTON_4");
pin1.toggle();
pin2.low();
pin3.toggle();
pin4.low();

}

});

System.out.println(".....texto_de_prueba_para_ver_la_respuesta_del_circuito
    aqui_en_la_consola.");

// mantiene el programa corriendo hasta que el usuario lo detenga con (CTRL
// -C)
while(true) {
Thread.sleep(500);
}
}
}

```

## Anexo B

# Montura del SO Raspbian with Processing en Raspberry PI 3B

Este anexo se centra en la descarga e instalación del sistema operativo Raspbian con el IDE Processing pre-instalado.

### B.0.2. Montura del SO utilizando Balena Etcher

Una vez montado el módulo de tracción junto a los demás componentes, se continuó con la instalación y configuración del IDE Processing dentro de la Raspberry PI, para esto se descargó un archivo .img de la página oficial Processing for PI y se montó en la memoria micro SD de la tarjeta.



Figura B.1: Página oficial de Processing for PI.

Para montar dicha imagen en la memoria micro SD se hizo uso del software Balena Etcher el cuál es un programa que nos facilita la montura de un archivo .img a una unidad de almacenamiento como lo son las memorias USB o las memorias microSD, la misma que se utiliza para montar un SO en Raspberry Pi.

El primer paso es abrir el programa y, posteriormente dar click en el botón que dice "Select Image".

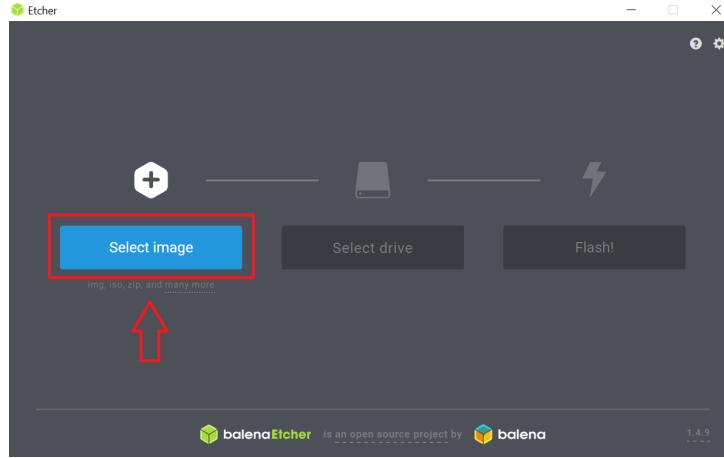


Figura B.2: Paso 1.Dar click en "Select image".

Una vez que se de click en el botón antes mencionado, se mostrará el administrador de archivos de nuestra PC, con este buscaremos el archivo .img que se desea montar en nuestra unidad de almacenamiento micro SD.

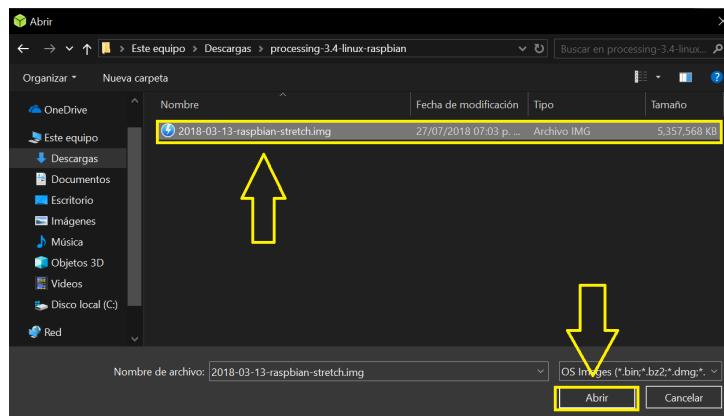


Figura B.3: Paso 2. Seleccionar la imagen que se va a montar en la micro SD.

Después de haber seleccionado la imagen, se activará el botón "Select Drive", el cuál debemos oprimir.

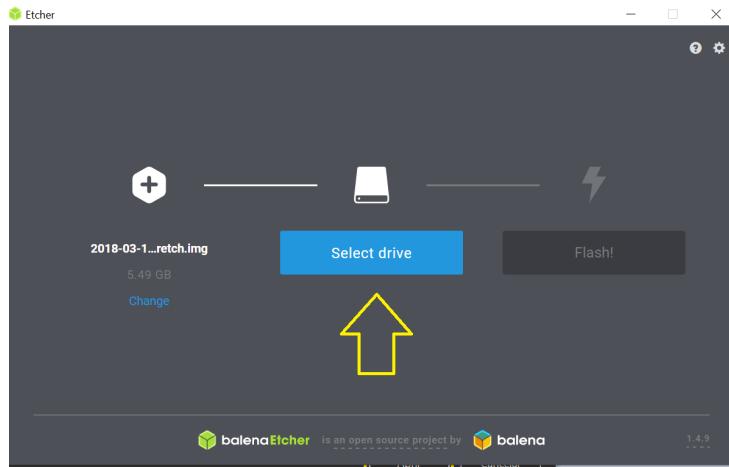


Figura B.4: Paso 3. Dar click en "Select Drive".

Se mostrará una lista de las unidades extraíbles disponibles que se pueden usar para montar el archivo .img. Se selecciona la unidad destinada para este propósito.

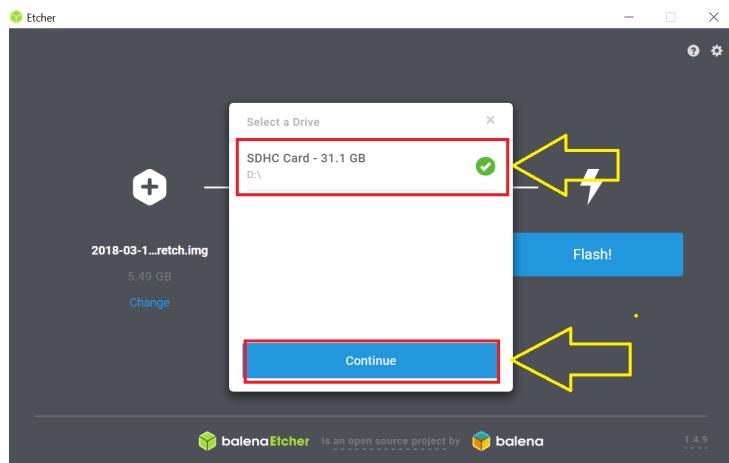


Figura B.5: Paso 4. Seleccionar la micro SD a utilizar en la Raspberry PI.

Por último, sólo queda dar click en el botón con la leyenda "Flash!" para empezar con el proceso de instalación del SO en la unidad microSD.

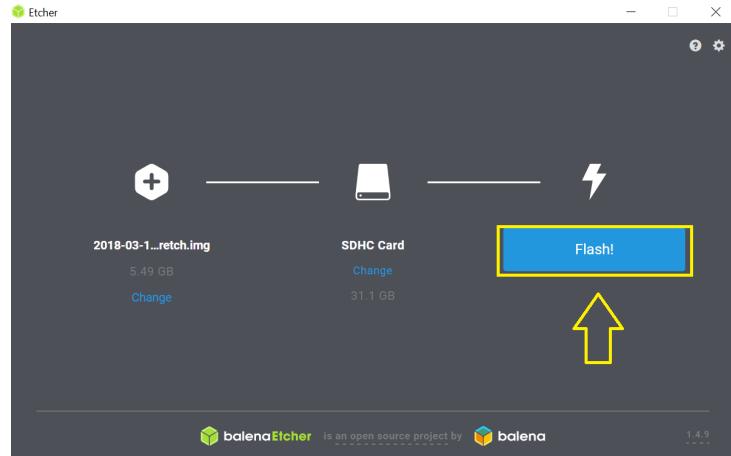


Figura B.6: Dar click en "Flash!" .

## Anexo C

# Configuración de Raspberry y Kinect

Una vez instalado Processing en la Raspberry PI, se deben descargar varias librerías pero sobretodo la librería “Open Kinect for Processing”. Dicha librería se encuentra en la página oficial de Processing for PI, en el apartado ”Technical”.

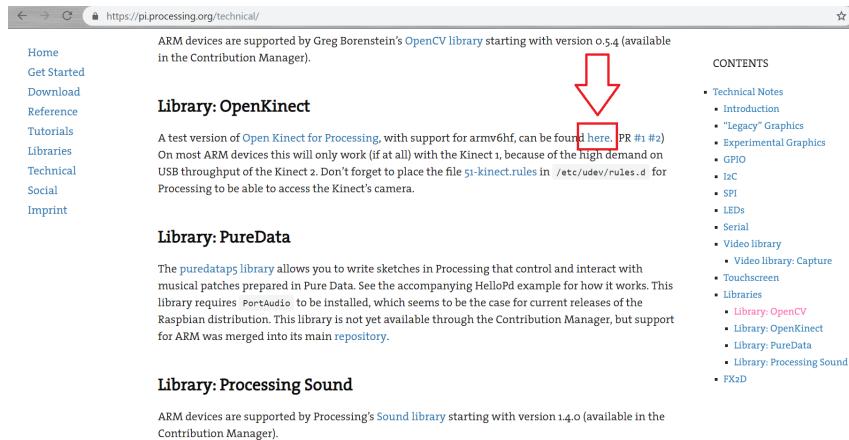


Figura C.1: Descarga de la librería Open Kinect for Processing.

Además, se debe crear un archivo .rules en la Raspberry PI dentro de la carpeta `/etc/udev/rules.d` con el nombre ”51-kinect.rules”, en el cual se debe escribir el siguiente código:

```
# ATTR{product}=="Xbox_NUI_Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
ATTR{idProduct}=="02b0", MODE=="0666"
# ATTR{product}=="Xbox_NUI_Audio"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
ATTR{idProduct}=="02ad", MODE=="0666"
```

```
# ATTR{product}=="Xbox_NUI_Camera"
SUBSYSTEM=="usb" , ATTR{idVendor}=="045e" ,
ATTR{idProduct}=="02ae" , MODE=="0666"

# Kinect for Windows
SUBSYSTEM=="usb" , ATTR{idVendor}=="045e" ,
ATTR{idProduct}=="02c2" , MODE=="0666"
SUBSYSTEM=="usb" , ATTR{idVendor}=="045e" ,
ATTR{idProduct}=="02be" , MODE=="0666"
SUBSYSTEM=="usb" , ATTR{idVendor}=="045e" ,
ATTR{idProduct}=="02bf" , MODE=="0666"
```

Una vez configurado todo lo anterior, se comenzó a realizar la codificación en el IDE Processing para el control de los motores con base a lo que el Kinect detectara usando su cámara de profundidad (Depth Camera).

## Anexo D

### Desfragmentación del código

Importación de librerías, en este apartado se puede observar las librerías con las cuales podemos manipular dos componentes, uno de ellos es el Kinect y el otro en la fase de pruebas es una librería para Arduino, que por el momento tendrá lugar para realizar diferentes pruebas que nos ayuden a avanzar de una manera eficiente el proyecto, teniendo en cuenta las facilidades que proporciona el Arduino para la manipulación de sus puertos seriales, con los cuales resulta una manera eficiente de cumplir con el objetivo.

```
import processing.serial.*;
import KinectPV2.KJoint;
import KinectPV2.*;
```

Declaración de dos variables de vital importancia, una de ellas para el manejo del puerto con atributos de serial en Arduino y la otra crea una variable con los atributos del Kinect v2.

```
KinectPV2 kinect;
Serial port;
```

Inicia función de configuración del Kinect, más adelante se detallará cada una de sus partes.

```
rintln(Serial.list());
```

Comando para visualizar todos los puertos del Arduino y asignarle un valor.

```
port = new Serial(this, Serial.list()[0], 9600);
```

Como cualquier cámara estas tienen una determinada definición, entes caso se abrirá una pestaña la cual contendría la definición dada por el programador y como lo indica tendrá y detectara colores RGB.

```
size(1920, 1080, P3D);
```

Inicia el objeto kinect

```
kinect = new KinectPV2(this);
```

se inicializa la clase Skeleton Color Map la cual grafica las diferentes partes del cuerpo que estén posicionados al alcance de la visión del Kinect.

```
kinect.enableSkeletonColorMap(true);
```

Inicializa camara RGB

```
kinect.enableColorImg(true);
```

¶

Inicia o enciende el kinect.

```
kinect.init();  
}
```

Inicializa el ciclo principal e infinito.

```
void draw() {
```

Le da valor de 0 (Color negro) al fondo de la ventana

```
background(0);
```

Inserta la imagen que obtiene la cámara RGB en la posición 0,0 (Eje zz eje "x")

```
Image(kinect.getColorImage(), 0, 0, width, height);
```

Crea un objeto con los atributos Skeleton la cual obtiene los componentes del objeto anterior

```
ArrayList<KSkeleton> skeletonArray = kinect.getSkeletonColorMap();
```

Grafica cada una de las extremidades individualmente, dependiendo de cuales estén en dentro del campo de visión del kinect.

```
for (int i = 0; i < skeletonArray.size(); i++) {  
    KSkeleton skeleton = (KSkeleton) skeletonArray.get(i);
```

Se decide si dentro del campo de visión existe alguna articulación, si llega a existir alguna se graficara, de lo contrario seguirá en proceso de búsqueda hasta que entre alguna en campo de visión.

```
if (skeleton.isTracked()) {
```

Obtiene número de articulaciones detectadas y las guarda en un arreglo.

```
KJoint[] joints = skeleton.getJoints();
```

Asigna un color diferente a cada uno de los usuarios los cuales pueden llegar a ser hasta 8.

```
color col = skeleton.getIndexColor();  
fill(col);  
stroke(col);  
drawBody(joints);
```

Se dibuja de diferente color la palma dependiendo del estado o reposo en el momento de captura.

```
drawHandState(joints[KinectPV2.JointType_HandRight]);  
drawHandState(joints[KinectPV2.JointType_HandLeft]);  
}  
}
```

Muestra en un texto la frecuencia de cuadros por segundo.

```
fill(255, 0, 0);  
text(frameRate, 50, 50);  
}
```

Grafica la espina dorsal.

```
void drawBody(KJoint[] joints) {  
    drawBone(joints, KinectPV2.JointType_Head, KinectPV2.JointType_Neck);  
    drawBone(joints, KinectPV2.JointType_Neck, KinectPV2.JointType_SpineShoulder);  
    drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.JointType_SpineMid);  
    drawBone(joints, KinectPV2.JointType_SpineMid, KinectPV2.JointType_SpineBase);  
    drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.JointType_ShoulderRight)  
    ;  
    drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.JointType_ShoulderLeft);  
    drawBone(joints, KinectPV2.JointType_SpineBase, KinectPV2.JointType_HipRight);  
    drawBone(joints, KinectPV2.JointType_SpineBase, KinectPV2.JointType_HipLeft);
```

Grafica las articulaciones del tronco.

```
drawBone(joints, KinectPV2.JointType_ShoulderRight, KinectPV2.JointType_ElbowRight);  
drawBone(joints, KinectPV2.JointType_ElbowRight, KinectPV2.JointType_WristRight);  
drawBone(joints, KinectPV2.JointType_WristRight, KinectPV2.JointType_HandRight);  
drawBone(joints, KinectPV2.JointType_HandRight, KinectPV2.JointType_HandTipRight);  
drawBone(joints, KinectPV2.JointType_WristRight, KinectPV2.JointType_ThumbRight);
```

Grafica las articulaciones del cuello.

```
drawBone(joints, KinectPV2.JointType_ShoulderLeft, KinectPV2.JointType_ElbowLeft);  
drawBone(joints, KinectPV2.JointType_ElbowLeft, KinectPV2.JointType_WristLeft);  
drawBone(joints, KinectPV2.JointType_WristLeft, KinectPV2.JointType_HandLeft);  
drawBone(joints, KinectPV2.JointType_HandLeft, KinectPV2.JointType_HandTipLeft);  
drawBone(joints, KinectPV2.JointType_WristLeft, KinectPV2.JointType_ThumbLeft);
```

Grafica las articulaciones de los hombros.

```
drawBone(joints, KinectPV2.JointType_HipRight, KinectPV2.JointType_KneeRight);  
drawBone(joints, KinectPV2.JointType_KneeRight, KinectPV2.JointType_AnkleRight);  
drawBone(joints, KinectPV2.JointType_AnkleRight, KinectPV2.JointType_FootRight);
```

Grafica las articulaciones de la parte baja del cuerpo.

```
drawBone(joints, KinectPV2.JointType_HipLeft, KinectPV2.JointType_KneeLeft);  
drawBone(joints, KinectPV2.JointType_KneeLeft, KinectPV2.JointType_AnkleLeft);  
drawBone(joints, KinectPV2.JointType_AnkleLeft, KinectPV2.JointType_FootLeft);  
  
drawJoint(joints, KinectPV2.JointType_HandTipLeft);  
drawJoint(joints, KinectPV2.JointType_HandTipRight);  
drawJoint(joints, KinectPV2.JointType_FootLeft);  
drawJoint(joints, KinectPV2.JointType_FootRight);
```

Grafica la cabeza y los pulgares.

```
drawJoint(joints, KinectPV2.JointType_ThumbLeft);  
drawJoint(joints, KinectPV2.JointType_ThumbRight);  
drawJoint(joints, KinectPV2.JointType_Head);  
}
```

Función para graficar todas las articulaciones detectadas.

```
void drawJoint(KJoint [] joints ,int jointType) {  
    pushMatrix();  
    translate(joints[jointType].getX() ,joints[jointType].getY() ,joints[jointType].  
    getZ());  
    ellipse(0, 0, 25, 25);  
    popMatrix();  
}
```

Forma la elipse que vemos en las articulaciones.

```
void drawBone(KJoint [] joints , int jointType1 , int jointType2) {  
    pushMatrix();  
    translate(joints[jointType1].getX() , joints[jointType1]  
    .getY() , joints[jointType1].getZ());  
    ellipse(0, 0, 25, 25);  
    popMatrix();  
    line(joints[jointType1].getX() , joints[jointType1].getY() , joints[jointType1].getZ() ,  
    joints[jointType2].getX() , joints[jointType2].getY() , joints[jointType2].getZ());  
}
```

CF

Función para graficar el estado de las manos.

```
void drawHandState(KJoint joint) {  
    noStroke();  
    handState(joint.getState());  
    pushMatrix();  
    translate(joint.getX() , joint.getY() , joint.getZ());  
    ellipse(0, 0, 70, 70);  
    popMatrix();  
}
```

Función para la detección del estado de la mano.

```
void handState(int handState) {  
    switch(handState) {
```

Mano abierta.

```
        case KinectPV2.HandState_Open:  
            fill(0, 255, 0);  
            size (200,200);
```

```
stroke (255,0,0);
strokeWeight(5);
fill(0,255,0);
rect (50,50,100,50);
port.write("A");
break;
```

Mano cerrada.

```
case KinectPV2.HandState_Closed:
fill(255, 0, 0);
port.write("B");
break;
```

Pulgar con índice.

```
case KinectPV2.HandState_Lasso:
fill(0, 0, 255);
port.write("C");
break;
```

94

Mano no detectada.

```
case KinectPV2.HandState_NotTracked:
fill(255, 255, 255);
port.write("D");
break;
}
```

### D.0.3. Código Completo.

```
import processing.serial.*;
import KinectPV2.KJoint;
import KinectPV2.*;

KinectPV2 kinect;
Serial port;

void setup() {
  println(Serial.list());
  port = new Serial(this, Serial.list()[0], 9600);
  size(1920, 1080, P3D);

  kinect = new KinectPV2(this);

  kinect.enableSkeletonColorMap(true);
  kinect.enableColorImg(true);

  kinect.init();
}

void draw() {
  background(0);

  image(kinect.getColorImage(), 0, 0, width, height);

  ArrayList<KSkeleton> skeletonArray = kinect.getSkeletonColorMap();

  for (int i = 0; i < skeletonArray.size(); i++) {
    KSkeleton skeleton = (KSkeleton) skeletonArray.get(i);
    if (skeleton.isTracked()) {
      KJoint[] joints = skeleton.getJoints();

      color col = skeleton.getIndexColor();
      fill(col);
      stroke(col);
      drawBody(joints);

      drawHandState(joints[KinectPV2.JointType_HandRight]);
      drawHandState(joints[KinectPV2.JointType_HandLeft]);
    }
  }

  fill(255, 0, 0);
}
```

```

text(frameRate, 50, 50);
}

void drawBody(KJoint[] joints) {
drawBone(joints, KinectPV2.JointType_Head, KinectPV2.JointType_Neck);
drawBone(joints, KinectPV2.JointType_Neck, KinectPV2.
    JointType_SpineShoulder);
drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.
    JointType_SpineMid);
drawBone(joints, KinectPV2.JointType_SpineMid, KinectPV2.
    JointType_SpineBase);
drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.
    JointType_ShoulderRight);
drawBone(joints, KinectPV2.JointType_SpineShoulder, KinectPV2.
    JointType_ShoulderLeft);
drawBone(joints, KinectPV2.JointType_SpineBase, KinectPV2.
    JointType_HipRight);
drawBone(joints, KinectPV2.JointType_SpineBase, KinectPV2.JointType_HipLeft
);

drawBone(joints, KinectPV2.JointType_ShoulderRight, KinectPV2.
    JointType_ElbowRight);
drawBone(joints, KinectPV2.JointType_ElbowRight, KinectPV2.
    JointType_WristRight);
drawBone(joints, KinectPV2.JointType_WristRight, KinectPV2.
    JointType_HandRight);
drawBone(joints, KinectPV2.JointType_HandRight, KinectPV2.
    JointType_HandTipRight);
drawBone(joints, KinectPV2.JointType_WristRight, KinectPV2.
    JointType_ThumbRight);

drawBone(joints, KinectPV2.JointType_ShoulderLeft, KinectPV2.
    JointType_ElbowLeft);
drawBone(joints, KinectPV2.JointType_ElbowLeft, KinectPV2.
    JointType_WristLeft);
drawBone(joints, KinectPV2.JointType_WristLeft, KinectPV2.
    JointType_HandLeft);
drawBone(joints, KinectPV2.JointType_HandLeft, KinectPV2.
    JointType_HandTipLeft);
drawBone(joints, KinectPV2.JointType_WristLeft, KinectPV2.
    JointType_ThumbLeft);

drawBone(joints, KinectPV2.JointType_HipRight, KinectPV2.
    JointType_KneeRight);
drawBone(joints, KinectPV2.JointType_KneeRight, KinectPV2.
    JointType_AnkleRight);
drawBone(joints, KinectPV2.JointType_AnkleRight, KinectPV2.
    JointType_FootRight);
}

```

```

drawBone(joints , KinectPV2.JointType_HipLeft , KinectPV2.JointType_KneeLeft)
;
drawBone(joints , KinectPV2.JointType_KneeLeft , KinectPV2.
JointType_AnkleLeft);
drawBone(joints , KinectPV2.JointType_AnkleLeft , KinectPV2.
JointType_FootLeft);

drawJoint(joints , KinectPV2.JointType_HandTipLeft);
drawJoint(joints , KinectPV2.JointType_HandTipRight);
drawJoint(joints , KinectPV2.JointType_FootLeft);
drawJoint(joints , KinectPV2.JointType_FootRight);

drawJoint(joints , KinectPV2.JointType_ThumbLeft);
drawJoint(joints , KinectPV2.JointType_ThumbRight);

drawJoint(joints , KinectPV2.JointType_Head);
i=0;
if (KJoint [ i]==null)
{
}

void drawJoint(KJoint [] joints , int jointType) {
pushMatrix();
translate(joints [jointType].getX() , joints [jointType].getY() , joints [
jointType].getZ());
ellipse(0, 0, 25, 25);
popMatrix();
}

void drawBone(KJoint [] joints , int jointType1 , int jointType2) {
pushMatrix();
translate(joints [jointType1].getX() , joints [jointType1].getY() , joints [
jointType1].getZ());
ellipse(0, 0, 25, 25);
popMatrix();
line(joints [jointType1].getX() , joints [jointType1].getY() , joints [
jointType1].getZ() , joints [jointType2].getX() , joints [jointType2].getY()
, joints [jointType2].getZ());
}

void drawHandState(KJoint joint) {
noStroke();
handState(joint.getState());
pushMatrix();
}

```

```
translate(joint.getX(), joint.getY(), joint.getZ());
ellipse(0, 0, 70, 70);
popMatrix();
}

void handState(int handState) {
switch(handState) {
case KinectPV2.HandState_Open:
fill(0, 255, 0);
port.write("A");
break;
case KinectPV2.HandState_Closed:
fill(255, 0, 0);
port.write("B");
break;
case KinectPV2.HandState_Lasso:
fill(0, 0, 255);
port.write("C");
break;
case KinectPV2.HandState_NotTracked:
fill(255, 255, 255);
port.write("D");
break;
}
}
```