UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE CIENCIAS EMPRESARIALES

Escuela Ingeniería Civil en Informática

Estudio y aplicación de la librería OpenCV sobre la arquitectura ARM,

para el control de agentes robóticos móviles usando visión artificial

Memoria para optar al título de Ingeniero Civil en Informática

AUTOR:

ESTEBAN MITCHEL CEA HIDALGO

Profesor Guía: Miguel Rodrigo Pincheira Caro
Profesor Guía: Juan Carlos Figueroa Durán

Chillán, Marzo de 2018

AGRADECIMIENTOS

A mis padres

Por darme siempre las herramientas necesarias para aprender, estudiar y desarrollar nuevos conocimientos, darme amor y tranquilidad en los momentos complejos y por estar siempre disponibles para todo.

A mi polola e hija

Por estar en cada etapa de mi vida, por ayudarme a despejar mi mente, darme la tranquilidad de que todo saldría bien y por cada momento de felicidad que he vivido a su lado.

A mis profesores guía

Por apoyarme en mi idea de no hacer un proyecto común, dandome ideas, información y risas para superar los problemas.

RESUMEN

En la actualidad, existen diversas herramientas en el mundo de la robótica que son utilizadas de manera frecuente para desarrollar soluciones apoyadas por la informática, contando con una renovación frecuente de tecnologías y materiales cada día más accesible al público menos conocedor.

Se plantea como objetivo en este informe, estudiar y analizar la librería OpenCV, que actualmente es el conjunto de librerías con más uso dentro del software para análisis de imágenes utilizando visión artificial, que cuenta con compatibilidad con varios lenguajes de programación como lo son C, C++, Java y Python; siendo este último el más utilizado para el desarrollo de prototipos.

OpenCV cuenta con compatibilidad con diversas plataformas, tales como Windows, macOS, iOS, Android y Linux; esta última será en la cual nos enfocaremos, específicamente en la arquitectura ARM presente en Raspberry 3, detallando su instalación y preparación para poder utilizar de forma funcional las librerías, y las características por las cuales ha sido propuesta para realizar los prototipos.

Los prototipos que se desarrollan son la identificación de señales de tránsito, tomando como referencia el signo pare; y también el reconocimiento de las líneas presentes en la calle, para apoyar la conducción autónoma de un vehículo.

Se documentan las características más relevantes de las librerías utilizadas, (de los prototipos propuestos) a través de la implementación de códigos sencillos de comprender; así también se demuestra que podemos utilizar estas librerías en casos reales, en un ambiente más complejo, como lo es el manejo de un automóvil real.

ABSTRACT

At present, there are several tools in the world of robotics that are used frequently to develop solutions supported by computer science, counting on a frequent renewal of technologies and materials every day more accessible to the public.

The aim of this project is to study and analyze the OpenCV library, which is currently the most used libraries in the software for image analysis using artificial vision, which has compatibility with several programming languages such as C, C ++, Java and Python; being the latter the most used on development of prototypes.

OpenCV has compatibility with various platforms, such as Windows, macOS, iOS, Android and Linux; we will focus on Linux, specifically on the ARM architecture present in Raspberry 3, detailing its installation and preparation to be able to use, and the characteristics why it has been proposed to make the prototypes.

The prototypes developed are the identification of traffic signals, taking as reference the stop sign; and the recognition of the lines present on the street, to support the autonomous driving of a vehicle.

The most relevant characteristics of the libraries used will be documented through the implementation of simple codes, this show that we can use these libraries in real cases, in a more complex environment, such as the handling of a real car.

INDICE GENERAL

INDICE DE IMÁGENES

1. Introducción

1.1. Introducción general

Este proyecto tiene como propósito obtener el título profesional de Ingeniero Civil en Informática, demostrando que el autor tiene las aptitudes y conocimientos necesarios para optar a dicho título a través de la indagación llamada "ESTUDIO Y APLICACIÓN DE LA LIBRERÍA OPENCV SOBRE LA ARQUITECTURA ARM, PARA EL CONTROL DE AGENTES ROBÓTICOS MÓVILES USANDO LA VISIÓN ARTIFICIAL", la cual consiste en comprender, aplicar y documentar el funcionamiento de la librería OpenCV en su versión 3.0.0, siendo actualmente una de las librerías de visión artificial más utilizadas, para posteriormente documentarla y realizar un prototipo que muestres las características destacables presentes en OpenCV.

La indagación se centra en describir las características de OpenCV, sus alcances y limitaciones, así también las cualidades técnicas necesarias para utilizar la librería.

Al final de la revisión se presentan las conclusiones obtenidas del estudio y análisis, a su vez se entregarán sugerencias para poder utilizar e implementar de forma efectiva la librería OpenCV.

Finalmente, se desarrolla un prototipo funcional a baja escala utilizando OpenCV como librería principal, Python como lenguaje de programación y RaspberryPi3 como plataforma de desarrollo, para proporcionar una base teórica y práctica donde nuevos integrantes del Grupo de Robótica de la Universidad del Bío-Bío puedan basarse para comenzar en el uso de OpenCV.

Este informe está compuesto por cinco capítulos, siendo el primero donde se detallan los objetivos tanto generales como específicos propuestos, así también se incluye la metodología usada para el desarrollo y por último las limitaciones presentes en el informe y el prototipo.

El segundo capítulo detalla el marco teórico de las tecnologías utilizadas, una breve explicación de su funcionamiento y características de estas para facilitar la contextualización y familiarización con el lector.

En el tercer capítulo se detallan a modo de instructivo, los pasos a seguir para poder instalar el sistema operativo en la RaspberryPi3, instalar y utilizar de forma correcta la

librería OpenCV, como así también la integración de la librería de Python y activación de PiCamera.

El capítulo cuatro se muestran el desarrollo y los resultados del prototipo desarrollado, así también la creación y utilización de la inteligencia artificial para que el prototipo se considere un agente autónomo.

En el capítulo cinco se habla sobre las conclusiones obtenidas del estudio y una serie de recomendaciones para utilizar de buena forma la plataforma y librerías presentes en el prototipo.

1.2. Especificaciones del estudio.

Primero se investigan las plataformas en las cuales se puede desarrollar utilizando la librería OpenCV, determinando que ARM, específicamente RaspberryPi3 es la más adecuada dado las características que se busca cumplir, como lo son movilidad en el agente robótico, y portabilidad.

Cuando se tiene definida la plataforma, se comienza a buscar información sobre la instalación de un sistema operativo que cumpla con la compatibilidad de integración con OpenCV y Python, y que cuente con la capacidad de utilizar el accesorio PiCamera[1], que será la principal fuente de entrada de información al sistema. Acorde a estas necesidades se define el sistema Raspbian, una distribución de Linux de uso libre basado en Debian Jessie, que se orienta a la enseñanza de la informática.

1.3. Especificaciones de los prototipos.

Los prototipos son desarrollados para un agente robótico móvil, que utiliza las cualidades de OpenCV para guiarlo por el ambiente, esto mediante el accesorio PiCamera, utilizado como el principal receptor de información desde el ambiente. Específicamente se desarrollan dos funcionalidades principales, la primera consta de reconocimiento de señales de tránsito, pudiendo realizar acciones en base a su detección; la segunda consta en el reconocimiento de las líneas presentes en las calles, las cuales nos permiten conocer la posición actual del vehiculo respecto de estas, esto nos permite guiar a un automóvil que funcione de forma autónoma, obteniendo información su ambiente (calle, ruta) para seguir su camino sin excederse de los límites de esta y respetando la señaletica.

[1] Módulo de Raspberry Pi, utilizado para capturar imágenes y videos.

1.4. Objetivos.

a) Generales:

Documentar y probar las propiedades de la librería de visión artificial OpenCV sobre la arquitectura ARM, y su uso en el control de agentes autónomos robóticos.

b) Específicos:

- Indagar la librería OpenCV y sus principales funcionalidades.
- Determinar, respecto de la indagación, cuales funciones se adecuan a las necesidades del proyecto para la detección y reconocimiento de objetos.
- Implementar un ejemplo con la librería, utilizando el lenguaje Python, para lograr que un agente robótico móvil de los disponibles en el grupo de robótica, sea capaz de reconocer objetos y evitar obstáculos.
- Documentar la librería y sus propiedades, así también la implementación del prototipo para poder su posterior uso en proyectos robóticos similares.

1.5. Metodología

Para el proyecto se ha decidido utilizar una metodología incremental (Somerville, 2011), pero con enfoque orientado a investigación; se opta por esta metodología ya que contamos con el foco principal de la indagación y su desarrollo, pero con el paso de las iteraciones se le añaden a los prototipos algunas de las funcionalidades investigadas, que son de relevancia para alcanzar el objetivo del informe; así también se optimiza la documentación en cada iteración, evitando efectuarla al termino del desarrollo de los prototipos, pudiendo omitir información importante.

Las etapas de la metodología elegida son:

- **Estudio de la Librería OpenCV:** Consiste en analizar, estudiar y destacar el contenido disponible en la documentación oficial de la librería.
- **Determinar aspectos a utilizar:** Ya conocido el contenido de la librería, se determina cuáles son los puntos destacables y que se utilizarán para desarrollar el prototipo funcional.

- **Implementación del prototipo:** Consiste en elaborar pequeños prototipos con las características más relevantes encontradas en la etapa anterior, implementando un modelo de baja escala en un ambiente controlado.
- **Documentación de la librería y el prototipo:** Se detallan los elementos relevantes de la librería, se documenta la creación del prototipo y las complejidades encontradas durante su construcción.
- **Conclusiones:** Ya logrados los resultados del experimento, el investigador debe aplicar su criterio científico para recomendar qué modelo ocupar o descartar en las situaciones vividas a lo largo del estudio. (Sampieri, 1998)

1.6. Alcances y limitaciones

Los alcances que tiene este informe se centran en el uso de la librería OpenCV y sus aplicaciones dentro del campo de la visión artificial en robótica.

OpenCV cuenta con una gran cantidad de prestaciones, pero este trabajo de título se limita a detallar y documentar los recursos utilizados durante la construcción y prueba de los prototipos.

Así también se describen las técnicas utilizadas para la creación y uso de la inteligencia artificial que hace a este prototipo un agente robótico móvil autónomo.

1.7. Definiciones y abreviaturas

- Robot: "Máquina automática programable capaz de realizar determinadas operaciones de manera autónoma y sustituir a los seres humanos en algunas tareas, en especial las pesadas, repetitivas o peligrosas; puede estar dotada de sensores, que le permiten adaptarse a nuevas situaciones". (Press, 2017).
- Haar: Clasificador en cascada, entrenado para reconocimiento de objetos, teniendo como base cientos de muestras, tanto positivas como negativas.
- Red neuronal: Es un procesador distribuido en paralelo de forma masiva con una propensión natural a almacenar conocimiento experimental y convertirlo en disponible para su uso. (Ripley, 2008).
- OpenCV: Open Source Computer Vision; librería de código abierto, que simplifica la implementación de prototipos apoyados por visión artificial.

- RoS: Robot Operating System; sistema operativo que cuenta con un conjunto de librerías y herramientas que permiten simplificar el desarrollo de robots. (Stanford, 2017).
- Arduino: Placa de hardware y software libre, utilizada en proyectos de robótica dado la fácil accesibilidad de sus componentes y su comunidad extensa. (Team, 2017).
- GPIO: General Purpose Input/Output, Entrada/Salida de Propósito General; es el puerto principal de conexión presente en Raspberry Pi, donde podemos controlar el comportamiento (ya sea de entrada o salida de datos) de los motores o sensores conectados a estos pines.
- ROI: Region of Interest / Región de interés; sección definida por el usuario en el cual se realizará el análisis de la información contenida en ella.
- FPS: Frame per Second – Cuadros por segundo; velocidad en la que un dispositivo muestra o captura imágenes.
- NOOBS: New Out Of Box Software; herramienta que facilita la instalación de distribuciones de Linux.
- Raspbian: Distribución de Linux basado en Debian, adaptado para Raspberry Pi.
- APT: Advanced Packaging Tool / Herramienta Avanzada de Empaquetado; biblioteca de funciones en C++ que permiten la gestión de paquetes, ya sea para instalación o desintalación de programas o librerías, presentes en las distribuciones de Linux.
- SUDO: Super User DO, súper usuario de forma temporal; utilidad que permite ejecutar programas con privilegios de administrador de forma temporal.

2. Marco teórico.

2.1. Introducción

En este capítulo se presentan las características relevantes encontradas en OpenCV, específicamente para el área de agentes robóticos móviles autónomos, generando una estructura detallada de las funciones prestadas por la librería, funciones que son utilizadas en la construcción del prototipo propuesto. Cabe mencionar que existen más métodos dentro de la librería, pero se detallan solo los que toman parte en el desarrollo.

2.2. Funciones principales

OpenCV cuenta con ciertas funciones principales que son la base para la utilización de la librería, tales como la lectura y escritura de imágenes, acceso a sus propiedas, etc. Y que estas características en conjunto con el lenguaje Python se utilizan para efectuar operaciones más complejas (OpenCVTeam, Funciones principales, 2017).

2.2.1. Operaciones básicas en imágenes

En esta sección se aprenden diversas técnicas para:

1. Lectura y escritura de imágenes: Se utilizan los métodos cv2.imread('Nombre_imagen'), donde cv2 es un objeto de OpenCV, seguido de imread para leer una imagen presente en el directorio desde el cual se está ejecutando el código; así también se utiliza cv2.imwrite('Nombre_imagen'), para guardar la imagen en dicho directorio. En caso de querer leer o escribir archivos en otros directorios, se debe proporcionar la ruta especifica donde se leerá o guardará.
2. Acceder a las propiedades de la imagen: Dentro de las propiedades de la imagen se obtienen los siguientes datos: el número de filas (altura en pixeles), columnas (ancho en pixeles) y canales; tipo de imagen, número de pixeles.
   Se destaca el uso del método img[2].shape, que retorna un arreglo en base al objeto img, que contiene el número de filas, columnas y canales[3] ; y el método img.size, que retorna el número de pixeles presentes en la imagen.

[2] Objeto que contiene una imagen previamente capturada mediante cv2.imread()
[3] En caso de que la imagen sea en colores, si es una imagen en escala de grises no se muestra este atributo

3. Establecer una ROI[4]: En determinados casos se necesita utilizar determinadas regiones dentro de una imagen, ya sea para detección de objetos, demarcar partes importantes, etc. Se utiliza en conjunto con la librería Numpy[5], y en este caso seleccionaremos una región que será copiada en otra parte de la imagen; mediante el siguiente código:

```
1   pelota = img[280:340, 330:380]
2   img[273:333, 100:160] = pelota
```

Figura 1: Código ROI

Fuente: Cea, 2017

En la primera línea 1 de la figura 1, definimos que tendremos objeto pelota que será un extracto de *img*, dado por las coordenadas [280:340, 330:380], para luego en la línea 2 insertar en img en las coordenadas [273:333, 100:160] lo que tenemos guardado en el objeto pelota. Siendo el resultado obtenido mostrado en la figura 2.

Figura 2. Ejemplo ROI.

Fuente: https://docs.opencv.org/3.0-beta/_images/roi.jpg.

[4] Region of interest, Región de interés.
[5] Librería que permite indexar archivos, específicamente se utiliza en este caso para acceder mediante coordenadas a un sitio específico en la imagen.

4. Separar y unir imágenes: A veces, necesitaremos trabajar de forma separada los diferentes canales presentes en la imagen. Para dicha tarea, contamos con dos métodos principales, presentados en la figura 3.

```
1   b,g,r = cv2.split(img)
2   img = cv2.merge((b,g,r))
```

Figura 3: Código separar y unir.

Fuente: Cea, 2017.

Donde la línea 1, toma como referencia img que es una imagen, cv2 es un objeto de OpenCV al cual se le invoca el método split que se utiliza para separar sus canales, que son guardados en las variables "*b*", "g" y "*r*"; siendo "*b*" para el canal azul, "*g*" para el canal verde, y "*r*" para el canal rojo, presentes en la imagen. De la misma forma se pueden volver a unir mediante el uso de cv2.merge, pasándole como parámetros las variables "*b, g, r*"; cabe destacar que estas funciones trabajan en formato BGR, y no RGB como se acostumbra, así que hay que tener especial cuidado al utilizarlas, ya que la imagen se distorsiona al ingresarlos en diferente orden.

2.2.2. Operaciones aritméticas en imágenes

En esta sección se utilizan dos atributos principales, un ROI y una imagen, lo que se realiza es una fusión de la imagen con la ROI, sin alterar el contenido de ambas.

La función utilizada dentro de esta categoría es cv2.bitwise_and('*imagen'*, '*ROI*'); donde cv2 es un objeto de OpenCV; en la línea 1, se define una serie de puntos que se utilizan para definir la *ROI*, en la linea 3 del código se transforma la imagen a escala de grises mediante el método cvtColor, al cual se le pasa como parametros la imagen a transformar y la constante cv2.COLOR_BGR2GRAY que define que se cambia del espacio de color BGR a escala de grises; luego en la línea 4, esta imagen se transforma en una matriz de ceros y unos; en la línea 5 se define que se ignore el color blanco dado por el código 255; el método cv2.fillPoly que se aprecia en la linea 6, realiza en base a esta imagen transformada a ceros y unos, tomar el área definida por el atributo *points*, e ignorar todos los elementos que no sean de color blanco, para luego, mediante el método cv2.bitwise_and(), unir ambas sin modificarlas. En específico el código mostrado en la figura 4, fue utilizado para el reconocimiento de las líneas de las calles.

```
1  points = np.array([[(0, 240),(320, 240), (226,100), (94,100)]], dtype = np.int32)
2  #trasnformacion imagen a escala de grises
3  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
4  ROI = np.zeros_like(gray_image)
5  ignore_mask_color = 255
6  cv2.fillPoly(ROI, points, ignore_mask_color)
7  masked_image = cv2.bitwise_and(gray_image, ROI)
```

Figura 4: bitwise_and().

Fuente: Cea, 2017.

2.3. Procesamiento de imágenes

En esta sección se explican las funciones principales con las que cuenta OpenCV para procesar las imágenes, ya sean obtenidas desde un archivo, o desde una cámara en tiempo real (OpenCVTeam, Procesamiento de imagenes, 2017).

2.3.1. Cambio en el espacio de color

El espacio de color está dado por los canales de la imagen a procesar, puede ser BGR[6] como se muestra en la figura 5, HSV[7] como se muestra en la figura 6, o Gray[8] como se muestra en la figura 7.

Para realizar algunas operaciones, es necesario ir pasando de un espacio de color a otro, y OpenCV nos aporta el método cv2.cvtColor('imagen', 'transformación'); donde imagen es el archivo que será sometido al cambio de su espacio de color; y transformación es una de las siguientes opciones:

1. cv2.COLOR_BGR2HSV: desde BGR hacia HSV.
2. cv2.COLOR_BGR2GRAY: desde BGR hacia escala de grises.
3. cv2.COLOR_HSV2GRAY: desde HSV hacia escala de grises.
4. cv2.COLOR_GRAY2HSV: desde escala de grises hacia HSV.
5. cv2.COLOR_GRAY2BGR: desde escala de grises hacia BGR.
6. cv2.COLOR_HSV2BGR: desde HSV hacia BGR.

[6] Blue, Green, Red - Azul, Verde, Rojo
[7] Hue, Saturation, Value – Matiz, Saturación, Valor
[8] Escala de grises

Los espacios de color están vistos desde perspectivas diferentes. El espacio BGR o RGB está toma como base que los colores están formados desde la mezcla por adición de los colores rojo, azul y verde, siendo estos los colores primarios y su representación tridimencional se puede apreciar en la figura 5.

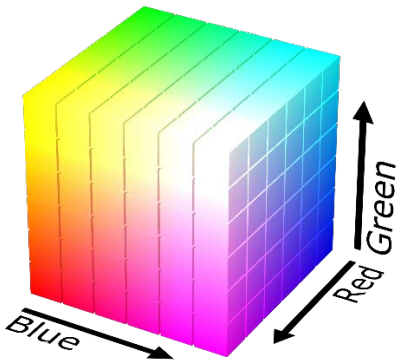

Figura 5: Espacio BGR.

Fuente: Cea, 2017.

El espacio de color HSV o HSB[9] (ambos corresponden al mismo espacio de color), define los colores en base a sus componentes, siendo estos el matíz, saturación y valor o brillo (V o B), a diferencia del espacio RGB, HSV no toma de forma lineal los colores para combinarlos, si no que utiliza una progresión del color basado en RGB, por tanto variar uno de los componente de el, hace que el color cambie, la respresentación tridimencional se muestra en la figura 6.

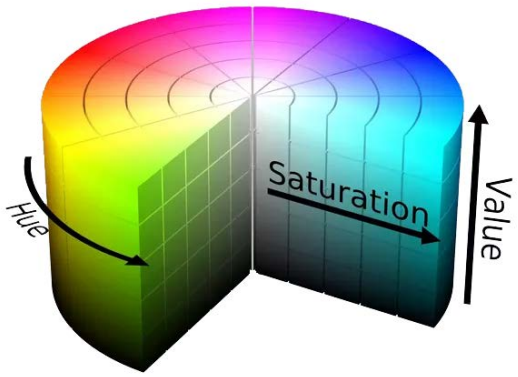

Figura 6: Espacio HSV.

[9] Hue, Saturation, Brightness – Matiz, Saturación, Brillo

Fuente: Cea, 2017.

El espacio de escala de grises, no tiene representación tridimencional, ya que en el solo influyen dos variables, como lo son el blanco y negro, y las tonalidades que se pueden encontrar en imágenes usan este espacio de color, están representadas en la figura 7.

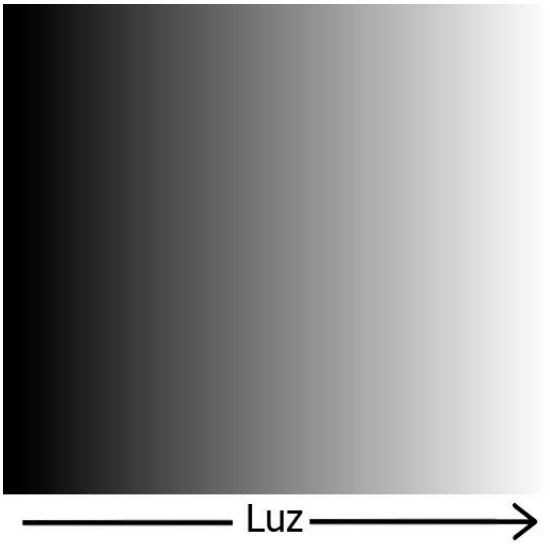

Figura 7: Espacio escala de grises.

Fuente: Cea, 2017.

2.3.2. Transformación geométrica

La transformación geométrica tiene como finalidad adaptar las imágenes, para poder realizar las tareas necesarias de mejor forma, adaptándose a las necesidades que se tengan. Se tiene como método principal a cv2.resize('imagen', 'factores', 'tipo'); donde imagen es el archivo a transformar; factores son los que darán cuanto crece o decrece la imagen; y tipo, puede tomar tres contantes, siendo estas cv2.INTER_CUBIC y cv2.INTER_LINEAR para ampliar, entre ellas se diferencian en que INTER_LINEAR realiza el reescalado de la imagen pixel a pixel, en cambio, INTER_CUBIC realiza el reescalado en fracciones de 4x4 pixeles, generando una imagen menos definida; y por último cv2.INTER__AREA para reducir.

En la figura 8 se muestra el código para ampliar una imagen al doble de su tamaño inicial.

```
1    res = cv2.resize(img,(2*width, 2*height), interpolation = cv2.INTER_CUBIC)
```

Figura 8: Ejemplo resize.

Fuente: Cea, 2017.

2.3.3. Detección de bordes canny

La detección de bordes canny, es una función que es útil cuando necesitamos delimitar donde se dan los mayores cambios de colores, y para esto se pueden utilizar cualquier tipo de imagen, ya sean BGR, HSV o Escala de grises, en la figura 9 se muestra el código para detectar los bordes mediante el método Canny.

```
1    edged = cv2.Canny( image, lower, upper)
```

Figura 9: Canny código.

Fuente: Cea, 2017.

Donde *image* es el archivo de referencia, *lower* y *upper* son los valores mínimo y máximo respectivamente, de diferencia entre colores.

En la figura 10 se puede ver un ejemplo de la funcionalidad de esta operación, en el lado izquierdo la imagen original, y en el lado derecho luego de aplicar la detección canny.

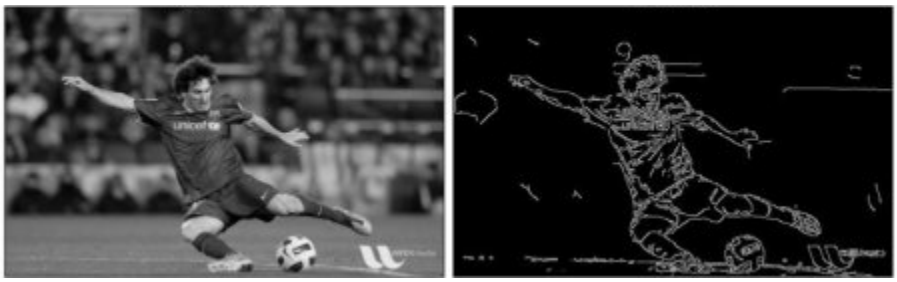

Figura 10: Ejemplo Canny.

Fuente: https://docs.opencv.org/3.0-beta/_images/canny1.jpg.

2.4. Video análisis

En la sección de video análisis, se encuentran métodos que proporcionan diversas utilidades para efectuar un análisis en tiempo real de videos, ya sean desde archivos hasta capturados en vivo con una cámara (OpenCVTeam, Video análisis, 2017).

2.4.1. Sustracción de fondo

Específicamente se llama sustracción de fondo a la operación que tiene por finalidad tomar una imagen como base que puede ser definido como el primer frame[10], o una imagen predefinida mediante un archivo; para luego compararlo con los siguientes frames capturados por la cámara y obtener las diferencias entre ellos. En la figura 11 apreciamos dos partes principales, la primera es la definición de las variables que estaremos utilizando durante el ejercicio.

```
1   cap = cv2.VideoCapture('vtest.avi')
2
3   fgbg = cv2.createBackgroundSubstractorMOG()
4
5   while(1):
6       ret, frame = cap.read()
7
8       fgmask = fgbg.apply(frame)
9
10      cv2.imshow('frame', fgmask)
11      k = cv2.waitKey(1) & 0xFF
12      if k == ord("q"):
13          break
```

Figura 11: Sustracción de fondo.

Fuente: Cea, 2017.

[10] Imagen capturada según la capacidad de la cámara, por ejemplo, una cámara que graba a 30fps captura 30 imágenes por segundo.

Donde cap en la línea 1 es una variable que referencia al archivo que se analiza; y en la línea 2, fbgb contiene el primer frame del video, en este caso se utiliza la figura 12; la segunda parte comienza en la linea 5, donde se define un bucle infinito que solo se interrumpe cuando se presione la letra "q" desde el teclado para finalizar la ejecución; y al interior de este bucle, se realiza la lectura del frame actual en la línea 6, donde a la variable cap se le aplica el metodo read() , para guardar su contenido en las variables ret y frame; la ejecución continúa en la línea 8 aplicando la diferencia entre ambas imágenes tomando como base a la variable definida en la línea 3 fgbg a la cual se le aplica el método apply respecto de frame, y se guarda a su vez esto en fgmask, esta variable contiene el resultado de la operación y genera el resultado visto en la figura 13, esto se muestra en el sistema mediante el método escrito en la línea 10 del código en la figura 11.

El algoritmo de sustracción de fondo, lo que realiza es una comparación directa entre la imagen base, en este caso la figura 12, y la imagen capturada actualmente mediante la camara o el siguiente frame del video; identificando que elementos variaron entre ellos y destacandolos. Este método se puede aplicar para el análisis de vehículos que se circulan por una calle mediante una cámara de vigilancia, en la puerta de algún edificio para contar las personas que transitan por ella. Pero hay que tener en consideración que no se pueden distinguir de manera exacta que es precisamente lo que está cambiando, así que el mejor uso que se le puede dar, es para identificar en que lugar está la diferencia y marcarla en la imagen que se está mostrando mediante el método imshow[11] en la línea 10 del ejemplo.

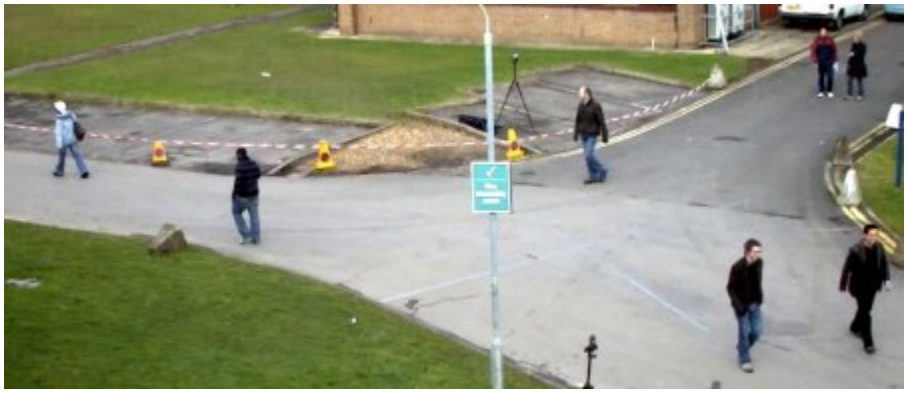

Figura 12: Base ejemplo sustracción de fondo.

Fuente: https://docs.opencv.org/3.0-beta/_images/resframe.jpg.

[11] Método de OpenCV que despliega una interfaz con la imagen pasada por parametro.

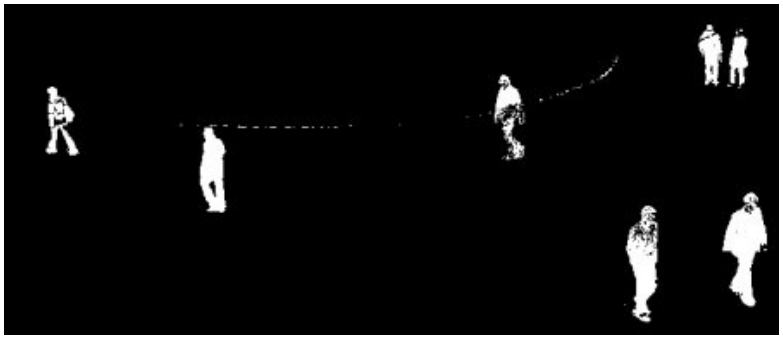

Figura 13: Ejemplo sustracción de fondo aplicado.

Fuente: https://docs.opencv.org/3.0-beta/_images/resmog.jpg.

2.5. Calibración de cámara

OpenCV proporciona un apartado completo para poder calibrar la cámara, buscando bajar el ruido de las imágenes capturadas, para evitar inconsistencias e información defectuosa (OpenCVTeam, Calibración de cámara, 2017).

2.5.1. Calibración

Dado la naturaleza de los cristales curvos presentes en los lentes de las cámaras, es normal que las imágenes que son capturadas por ellos tengan leves distorsiones, haciendo que una línea recta, no lo sea realmente, para lo cual OpenCV brinda el método cv2.calibrateCamera(), que evita la distorsión de las líneas por los lentes de la cámara. En la figura 14 se logra apreciar la imagen sin corrección y en la figura 15 la imagen corregida.

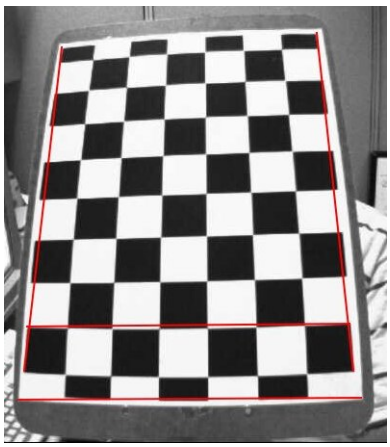

Figura 14: Sin calibración.

Fuente: https://docs.opencv.org/3.0-beta/_images/calib_radial.jpg.

Figura 15: Con calibración.

Fuente: https://docs.opencv.org/3.0-beta/_images/calib_result.jpg.

2.6. Machine learning

En esta sección, se detalla las máquinas de aprendizaje, vistas desde el punto de su entrenamiento, y su uso como clasificador. Dichas máquinas son un sub-campo de las ciencias de la computación y una rama de la inteligencia artificial, que busca dotar al computador de técnicas para aprender, determinadas mediante programas capaces de generar comportamientos en base a ejemplos positivos y negativos (OpenCVTeam, Machine learning, 2017).

2.6.1. Entrenamiento en cascada

El entrenamiento en cascada para una red neuronal es uno de los métodos más sencillos para dotar de comportamientos al computador, ya que se utilizan ejemplos positivos y negativos para su entrenamiento. Los ejemplos positivos utilizados son imágenes donde se pueda apreciar de forma clara el elemento a identificar, visto desde diferentes perspectivas, para poder tener una muestra más amplia de él; así también los ejemplos negativos son lo opuesto, imágenes donde no se encuentre el elemento a identificar, se recomienda utilizar ejemplos negativos que tengan relación al problema, no seleccionamos imágenes negativas del mar, si estamos en un problema vial. Luego se hace una serie de combinaciones entre ellas, generando imágenes que tienen en su interior el elemento sobre los ejemplos negativos y se genera un documento de texto, que contiene las coordenadas en donde se encuentra el elemento a buscar y su tamaño, para luego ser sometidas a una serie de combinaciones y comandos para realizar el entrenamiento de la red neuronal; generando un archivo con

extensión XML[12] que contendrá las posibles entradas del sistema, y la identificación del elemento. Dicho entrenamiento será ejemplificado y documentado en profundidad en el capítulo cuatro de este informe ya que es un tema bastante extenso.

2.7. Detección de objetos

La detección de objetos es uno de los puntos importante presentes en OpenCV, ya que permite una integración rápida y fácil teniendo una red neuronal entrenada. A modo de ejemplo, OpenCV cuenta con un tutorial para utilizar dichas redes neuronales, proporcionando una ya entrenada, para reconocimiento de cara y ojos (OpenCVTeam, Detección de objetos, 2017).

2.7.1. Haar Cascade Classifiers

Dentro de la detección de objetos, Haar Cascade Classifiers nos sirve para encontrar objetos en tiempo real desde una transmisión desde una cámara. OpenCV proporciona el método cv2.CascadeClassifier('*archivo.xml*') que se puede ver gráficamente en la línea 1 de la figura 16; donde archivo es un documento con extensión xml de una red neuronal entrenada. Luego este objeto se le aplica el método detectMultiScale('*imagen*') en la línea 2 del código; donde *imagen* es el frame actual obtenido desde la cámara; a su vez, "*x, y*" son la coordenada de donde comienza el elemento encontrado, y "*w, h*" son anchura y altura del elemento, es importante mencionar que detectMultiScale puede encontrar mas de un elemento en la imagen, por tanto en la línea 4 decimos que para cada elemento encontrado en *detect*, obtenemos su *x, y, w, h*. Luego en la línea 5 utilizamos el método cv2.rectangle para demarcar el elemento; y cv2.putText en la línea 6 para escribir la alerta detect en la imagen mostrada en ese momento.

```
1  classifier_cascade = cv2.CascadeClassifier('classifier.xml')
2  detect = classifier_cascade.detectMultiScale(image)
3
4  for (x,y,w,h) in detect:
5      cv2.rectangle(image, (x+5, y+5),(x+w-5, y+h-5), (255,0,0),2)
6      cv2.putText(image, 'detect', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.7,(0,0,255),2)
```

Figura 16: Utilización de Cascade Classifiers.

Fuente: Cea, 2017.

[12] eXtensible Markup Language, lenguaje de marcado extensible.

3. Instructivo.

3.1. Introducción

En este capítulo se detallan los pasos a seguir para trabajar con la librería, la instalación del sistema operativo, complementos necesarios para trabajar con Python, proceso de activación de la cámara en el sistema operativo y activación del complemento de VNC para utilizar escritorio remoto.

3.2. Detalle Raspberry Pi 3

La Raspberry Pi 3 es la tercera generación de Raspberry Pi, sucesor de la Raspberry Pi 2 Modelo B, reemplazándola en febrero de 2016. Sus características de hardware son:

1. CPU: ARMv8 1.2GHz, quad-core a 64-bit
2. 802.11n Wireless LAN
3. Bluetooth 4.1 de bajo consumo
4. 4 puertos USB
5. 40 pines GPIO
6. Salida HDMI
7. Puerto Ethernet
8. Jack de audio 3.5mm
9. Interfaz de cámara
10. Interfaz de pantalla
11. Puerto de memoria Micro SD (reverso)
12. Entrada de poder
13. 1 Gb de memoria RAM

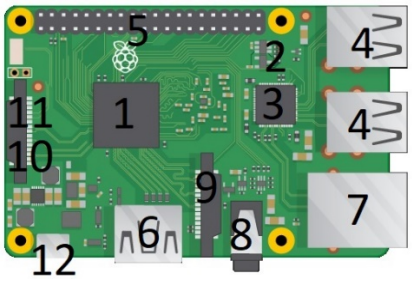

Figura 17: Hardware Raspberry Pi 3.

Fuente: Cea, 2017.

3.3. GPIO

En esta sección hablaremos sobre el GPIO (General Purpose Input/Output), que como su nombre lo indica, es un sistema de entrada y salida para multiples usos, que está presente en la Raspberry Pi 3.

El GPIO es la interfaz hacia el mundo de la Raspberry, permitiendo la conexión de sensores de diversos tipos, ya sean de intensidad luminica, sensores de ultrasonido; tambien se pueden conectar actuadores tales como motores de corriende continua mediante un integrado, servomotores; y muchos otros perifericos extras.

El GPIO cuenta exclusivamente con pines “unbuffered”, esto quiere decir que no cuentan con protección, por tanto hay que tener cuidado con los voltajes e intensidades que se conectan a el para no dañar los componentes electronicos de la placa o los perifericos.

GPIO cuenta con tres tipos diferentes de pines; estos son los de voltaje, que pueden entregar 5v o 3.3v; los pines de tierra que hacen de negativo, y los pines GPIO, que son los que nosotros podemos programar para determinadas tareas. En la figura 18 podemos apreciar la distribución de los pines en la Raspberry Pi 3.

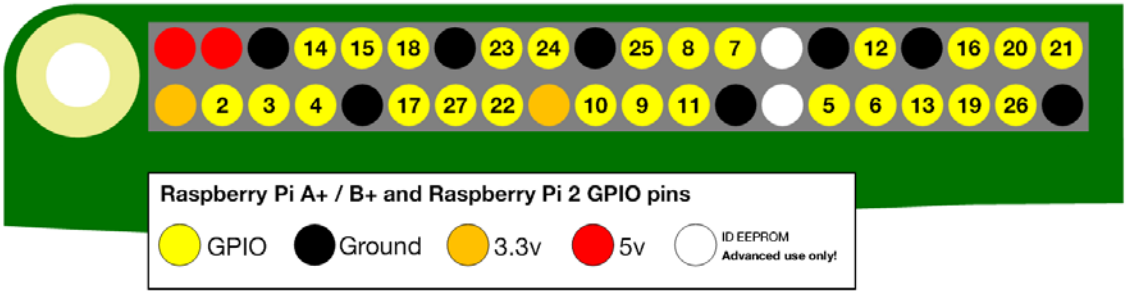

Figura 18: Pines GPIO.

Fuente: https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/images/gpio-numbers-pi2.png

A diferencia de sus versiones anteriores, la Raspberry Pi 3 cuenta con 40 pines, siendo 26 programables, 2 de 5v, 2 de 3.3v, 8 de tierra, y 2 para usos avanzados. Los pines programables pueden ser definidos como entrada o salida digital serial, y para la lectura desde sensores análogos, es necesario incluir un conversor analógico digital.

3.4. Instalación sistema operativo

En esta sección se habla sobre la instalación del sistema operativo en la Raspberry Pi 3, se decide utilizar NOOBS[13] proporcionado por los desarrolladores de Raspberry, ya que cuenta con una instalación más sencilla que hacerlo desde una imagen de Raspbian[14] .

3.4.1. Preparación micro SD

Lo primero que hay que hacer es formatear la memoria Micro SD en formato FAT32, acceder al sitio de descarga oficial[15], y descargar la última versión del NOOBS disponible en formato zip como se muestra en la figura 19.

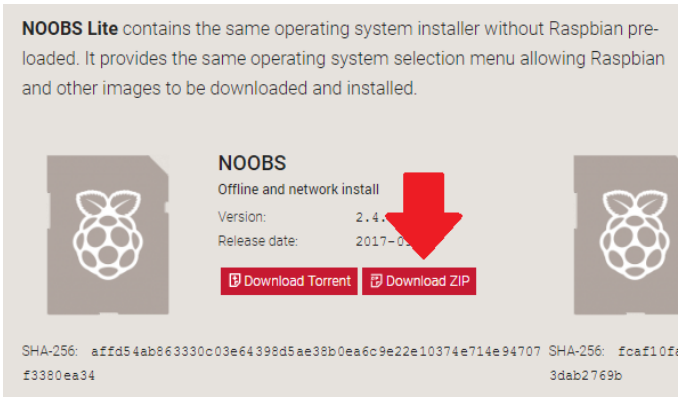

Figura 19: Descarga NOOBS.

Fuente: https://www.raspberrypi.org/learning/software-guide/images/download-zip.png.

3.4.2. Instalación

El siguiente paso es descomprimir el fichero descargado en la raíz de la Micro SD, y luego conectarla en el puerto de la Raspberry Pi 3. La Raspberry comenzará a reconocer los archivos nuevos instalados en la memoria, y procederá a mostrar una interfaz que cuenta con varios sistemas operativos compatibles con Raspberry Pi 3, en este caso seleccionar la primera opción que es Raspbian, ejemplificada en la figura 20.

[13] New Out Of Box Software
[14] Adaptación de Debian para Raspberry
[15] https://www.raspberrypi.org/downloads/

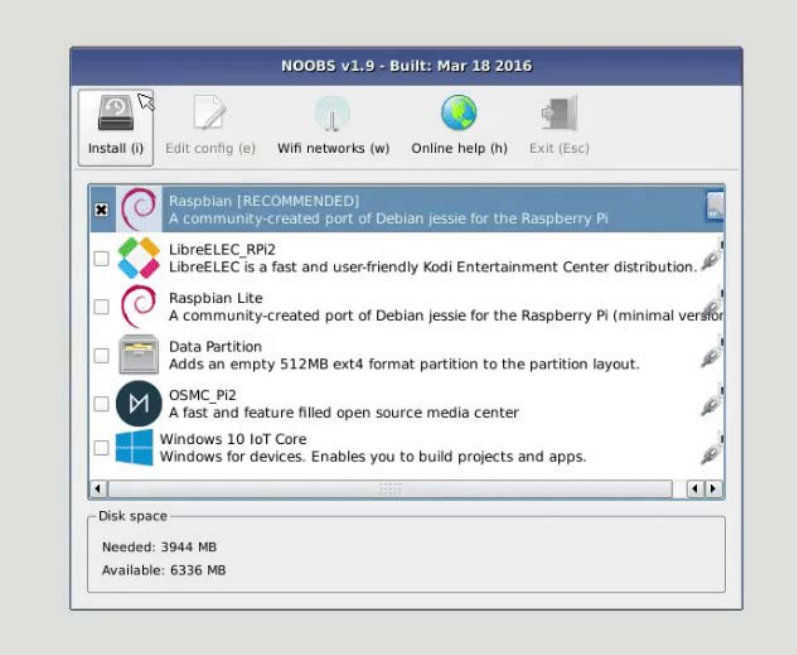

Figura 20: Interfaz de instalación.

Fuente: Cea, 2017.

Luego de seleccionar el sistema operativo deseado, presionamos el botón Install y confirmar la selección como se muestra en la figura 21.

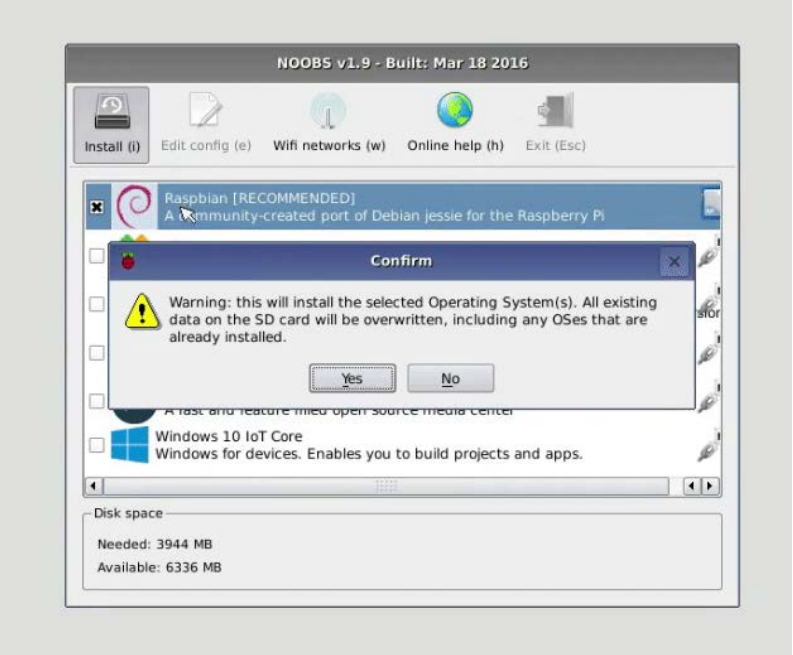

Figura 21: Confirmación instalación.

Fuente: Cea, 2017.

El siguiente proceso es esperar a que el sistema operativo se instale correctamente en la memoria Micro SD, donde se aprecia la interfaz mostrada en la figura 22, al finalizar el proceso se muestra una notificación de que se ha instalado correctamente, tal como se ve en la figura 23, y se carga el sistema operativo Raspbian que se puede apreciar en la figura 24.

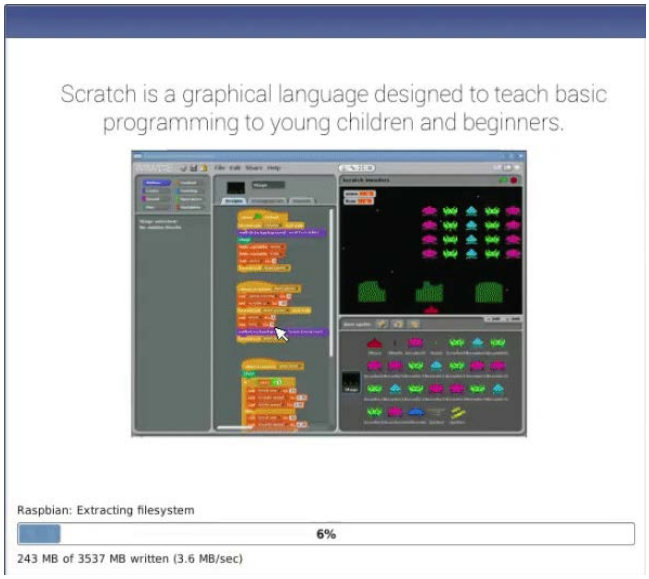

Figura 22: Proceso instalación.

Fuente: Cea, 2017.

Figura 23: Instalación exitosa.

Fuente: Cea, 2017.

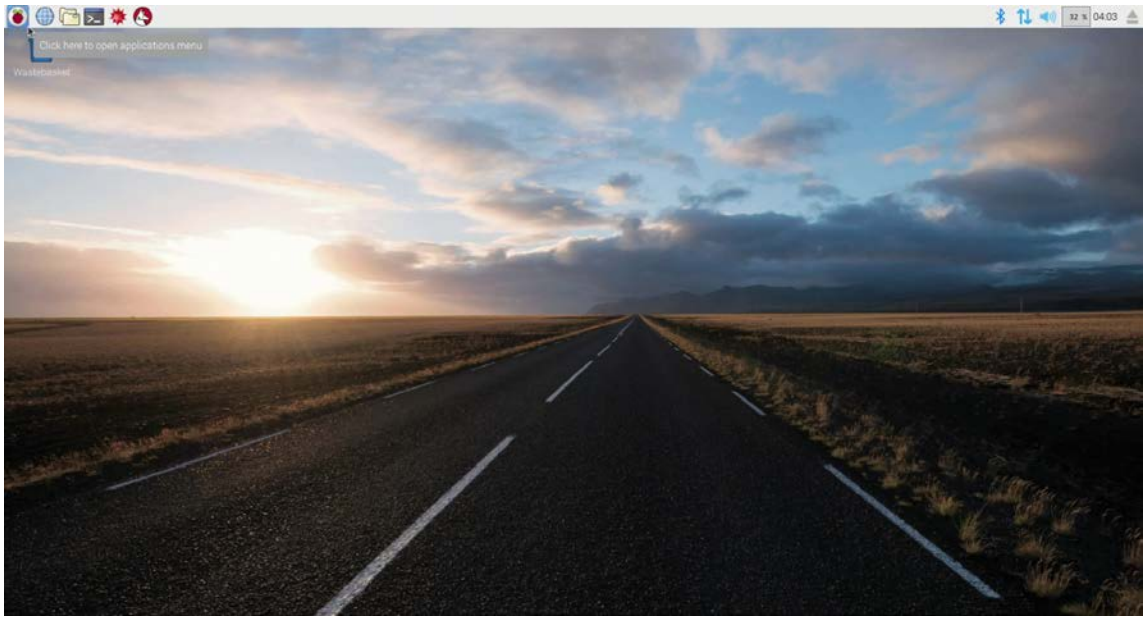

Figura 24: Raspbian.

Fuente: Cea, 2017.

3.5. Instalación Python

La distribución de Raspbian descargada desde la página oficial de Raspberry, incluye Python 3.0, por lo cual no es necesario instalarlo ya que es la versión base que se utiliza, y tiene compatibilidad con las librerías extras que se necesitan en el desarrollo del prototipo.

A pesar de estar incluido, se menciona el método de instalación para contar con la versión actualizada de Python en su versión 3.6.3. Para esto se utiliza APT[16], ya que simplifica en gran medida la instalación y desinstalación de complementos o programas en el entorno de Raspbian.

[16] Advanced Packaging Tool (Herramienta Avanzada de Empaquetado)

Lo primero que se realiza es abrir una consola del sistema operativo e ingresar el comando de la figura 25.

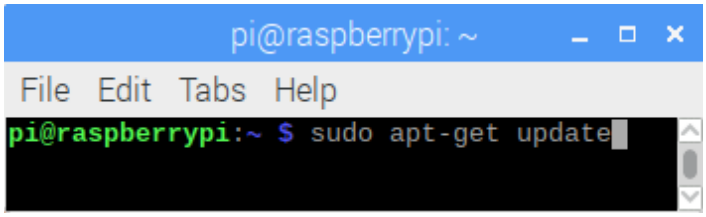

Figura 25: sudo apt-get update.

Fuente: Cea, 2017.

Con el cual se comprueba si es que hay alguna actualización disponible. La cual se instala con el comando mostrado en la figura 26 para mantener el sistema actualizado y bajar la probabilidad de fallos por contar con librerias o software desactualizado.

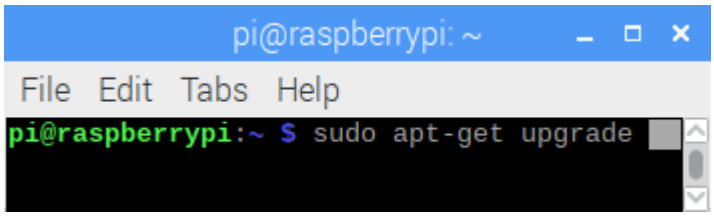

Figura 26: sudo apt-get upgrade.

Fuente: Cea, 2017.

Luego se continúa con el comando detallado en la figura 27, con esto se dispone del complemento PiCamera para la utilización de la cámara en el proyecto.

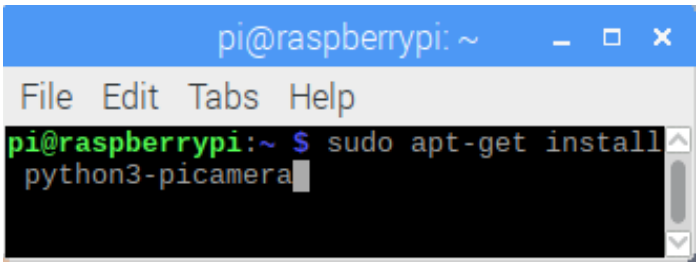

Figura 27: sudo apt-get install python3-picamera.

Fuente: Cea, 2017

3.6. Activación cámara

Lo que se realiza en este paso es la activación de la cámara, ya que, si no se realiza, al ejecutar cualquier programa esta no funcionará de forma correcta o simplemente no se puede acceder a ella.

Primero se abre un terminal, en el cual se ingresa el siguiente comando de la figura 28.

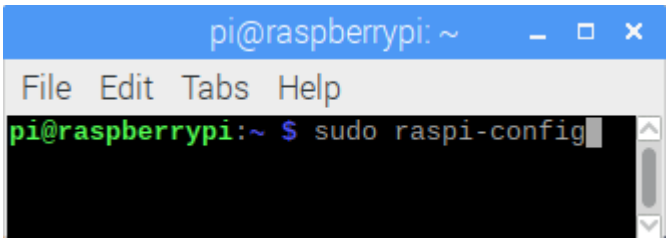

Figura 28: sudo raspi-config.

Fuente: Cea, 2017.

Al ingresar el comando, el sistema abre la interfaz presente en la figura 29.

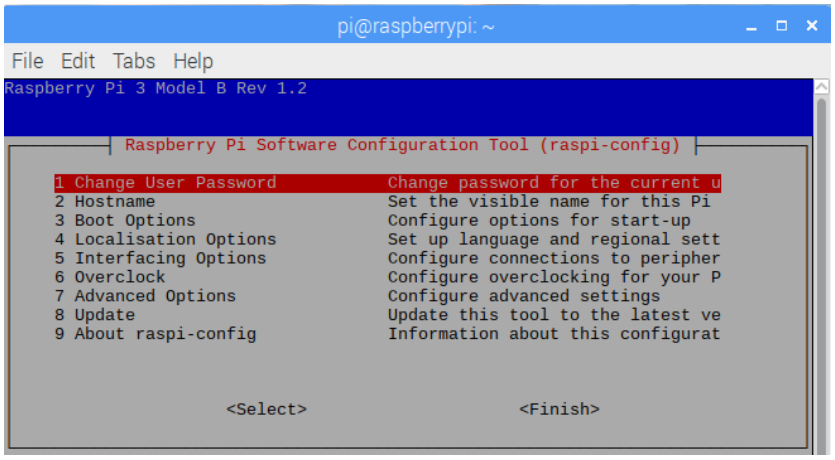

Figura 29: Selección opciones de interfaz.

Fuente: Cea, 2017.

Seleccionamos la opción 5 y presionamos la tecla enter para entrar al siguiente menú.

En el menú desplegado que se aprecia en la figura 30, se selecciona la opción P1 con enter, y se confirma la operación para activar el módulo de la cámara.

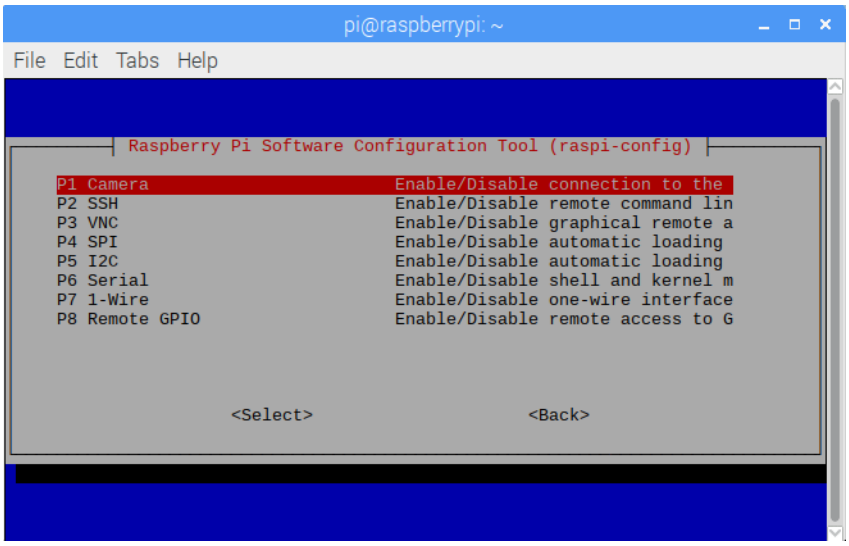

Figura 30: Activar cámara.

Fuente: Cea, 2017.

Luego de activar la opción, se recomienda reiniciar el sistema, para cargar desde el inicio del sistema operativo la compatibilidad con la cámara, evitando que la Raspberry no la reconozca.

3.7. Activación VNC

3.7.1. Configuración desde RaspberryPi

Para activar la aplicación de VNC, que permite utilizar cualquier dispositivo conectado dentro de la red como monitor de la Raspberry, evitando tener que mantener una pantalla en todo momento conectada a la placa. Se ejecuta desde una consola el comando de la figura 31.

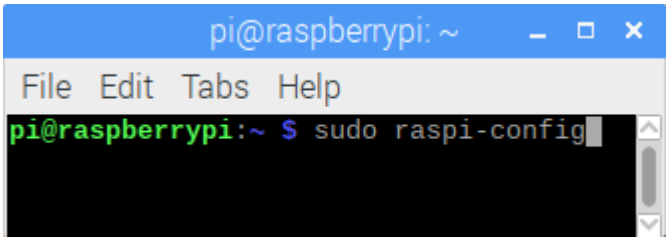

Figura 31: sudo raspi-config.

Fuente: Cea, 2017.

Luego de ingresar el comando, el sistema despliega la interfaz que se muestra en la figura 32, en donde se selecciona la opción 5.

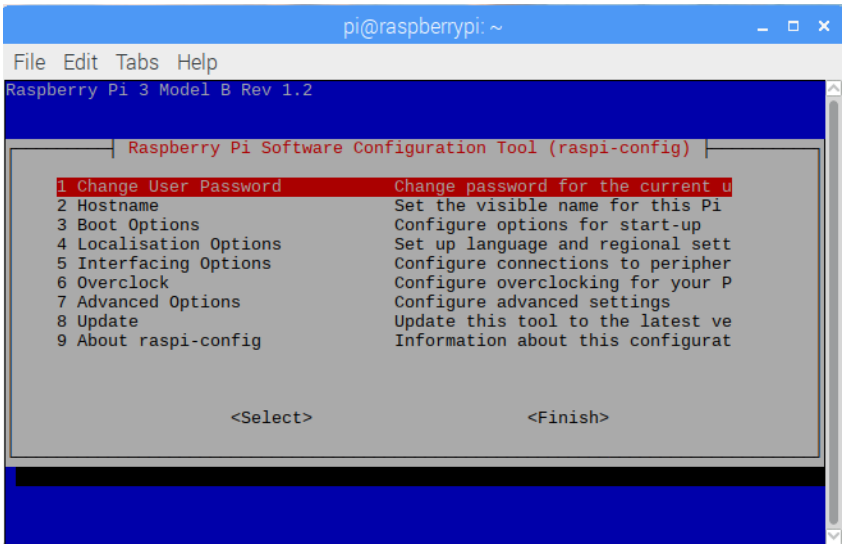

Figura 32: Selección opciones de interfaz.

Fuente: Cea, 2017.

Dentro del menú desplegado en la figura 33, se selecciona la opción P3 VNC, para activar la aplicación que permite utilizar un escritorio remoto.

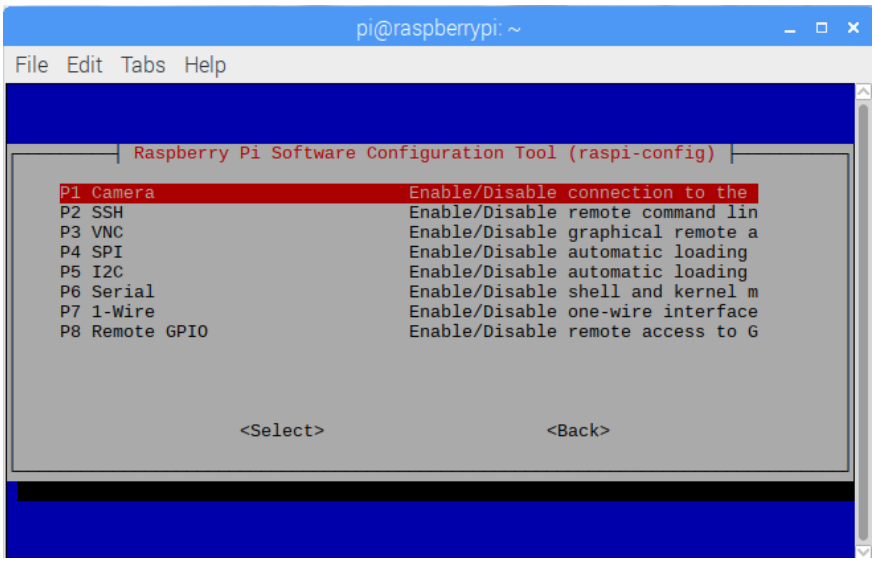

Figura 33: Selección VNC.

Fuente: Cea, 2017.

Se activa el botón de la aplicación VNC de color azul, en la barra de tareas del sistema operativo, este lo se ve en la figura 34, se hace doble click en el icono para ingresar a la aplicación.

Figura 34: Icono VNC.

Fuente: Cea, 2017.

Se abre la interfaz que se muestra en la figura 35, donde se muestra nuestra dirección IP en la red, la que usaremos posteriormente para conectar desde cualquier dispositivo, ya sea Windows, o macOS; que tenga la aplicación VNC.

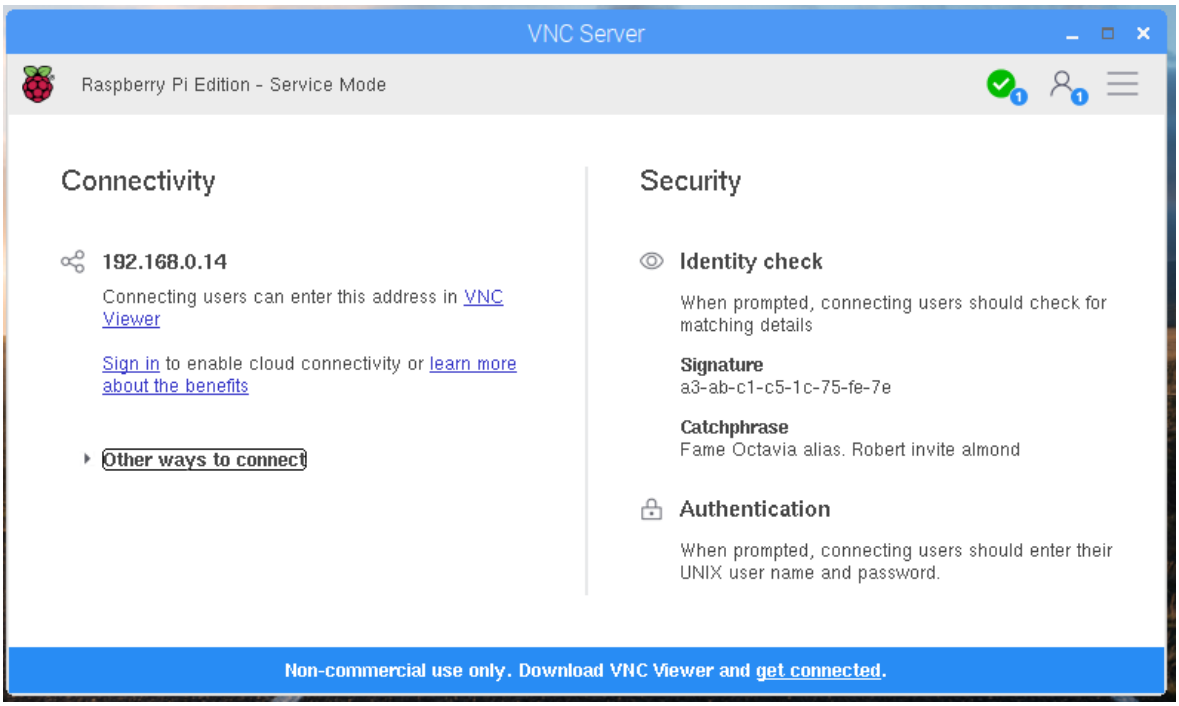

Figura 35: Interfaz VNC.

Fuente: Cea, 2017.

3.7.2. Configuración desde Windows

VNC cuenta con una aplicación oficial desarrollada para Windows, que se puede descargar desde su sitio web oficial[17] visto en la figura 36, el sitio sugiere descargar la versión compatible con el sistema operativo desde donde se está trabajando, y se realiza mediante el botón de descarga. En la barra superior se pueden seleccionar diversos sistemas operativos permitiendo la descarga de la aplicación correspondiente a dicho sistema.

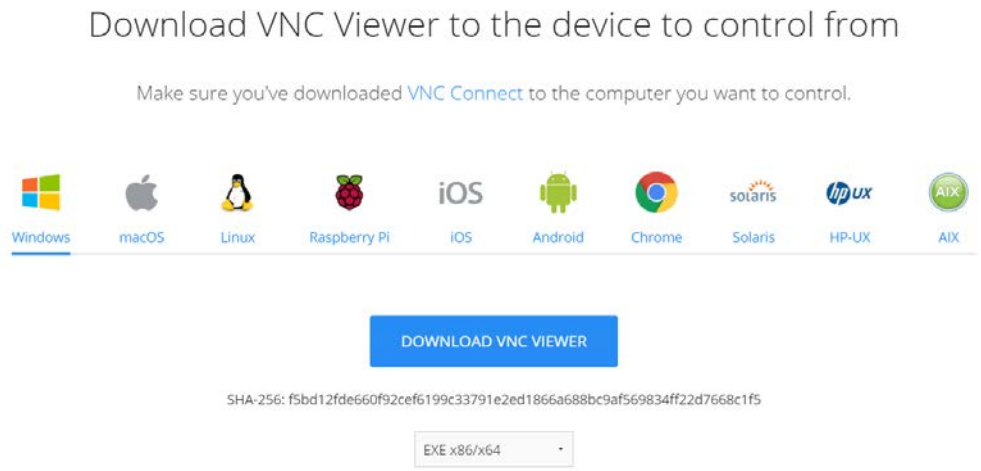

Figura 36: VNC sitio oficial.

Fuente: Cea, 2017.

Se inicia la instalación del software VNC Viewer, seleccionando el idioma como lo se aprecia en la figura 37.

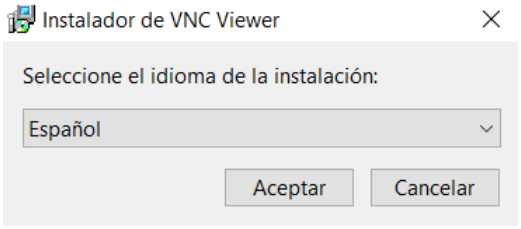

Figura 37: Intalación VNC.

Fuente: Cea, 2017.

[17] https://www.realvnc.com/en/connect/download/viewer/

Seleccionamos el botón siguiente, presente en la figura 38, se aceptan los terminos y condiciones detallados en la figura 39.

Figura 38: Instalación VNC.

Fuente: Cea, 2017.

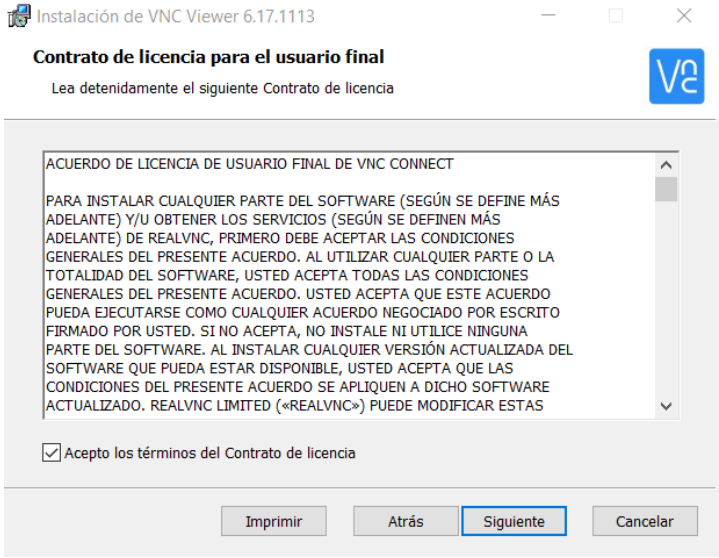

Figura 39: Terminos y condiciones VNC.

Fuente: Cea, 2017.

Se selecciona la ruta de instalación del software, y se continúa con el botón siguiente de la figura 40. Y se comienza la instalación presionando el botón Instalar de la figura 41.

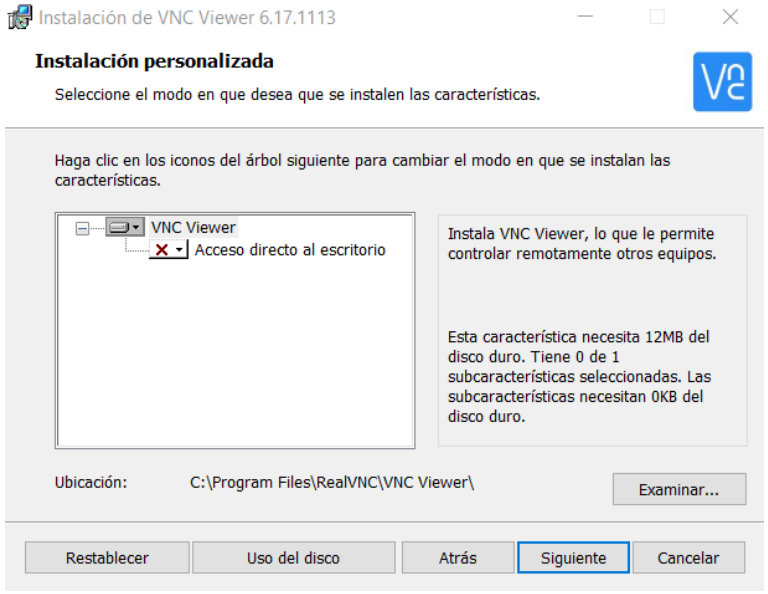

Figura 40: Ruta instalación VNC.

Fuente: Cea, 2017.

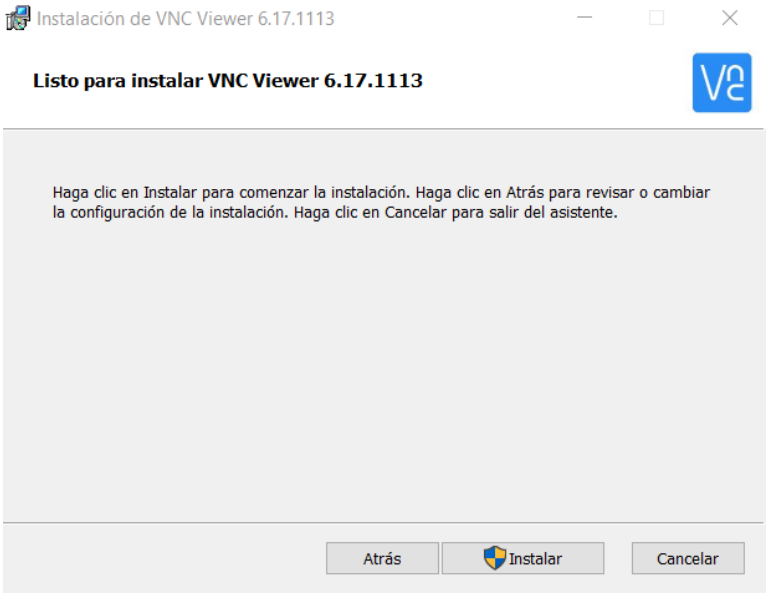

Figura 41: Instalar VNC.

Fuente: Cea, 2017.

Lo siguiente es finalizar el proceso de instalación presionando el botón finalizar mostrado en la figura 42. Se inicia el programa desde el acceso directo creado en el escritorio y se despliega la interfaz de la figura 43.

Figura 42: Instalación completa.

Fuente: Cea, 2017.

Figura 43: Interfaz VNC.

Fuente: Cea, 2017.

En la ventana de VNC, se selecciona en el menú superior la opción Archivo, y luego Nueva conexión. Esto despliega la ventana presentada en la figura 44. En esta ventana se ingresa la dirección IP proporcionada en VNC desde la Raspberry Pi 3 en la sección VNC Server, y se le asigna un nombre para su identificación, luego se acepta mediante al botón en la parte inferior y se crea un acceso directo a la Raspberry Pi 3.

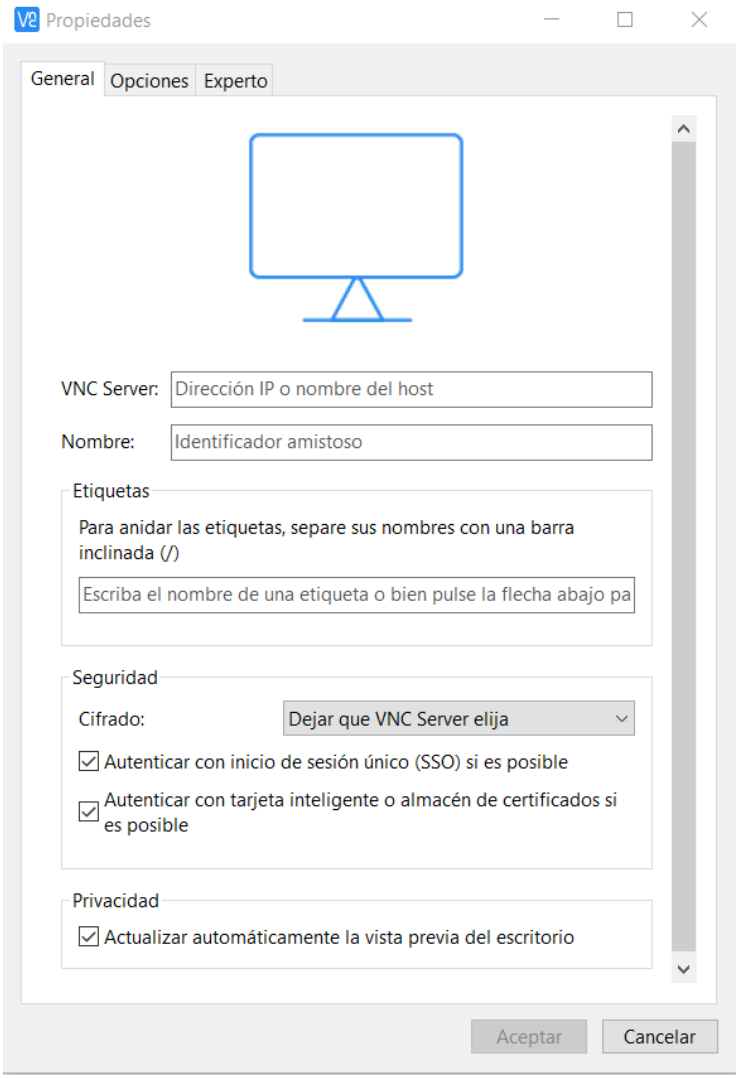

Figura 44: Nueva conexión.

Fuente: Cea, 2017.

Desde la ventana principal de VNC, se selecciona el icono de la Raspberry, que se muestra en la figura 45.

Figura 45: Acceso directo RaspberryPi.

Fuente: Cea, 2017.

Al hacer doble click, se despliega la ventana de la figura 46, en donde se ingresa el nombre de usuario y la contraseña, estas por defecto son, Nombre de usuario: pi; Contraseña: raspberry.

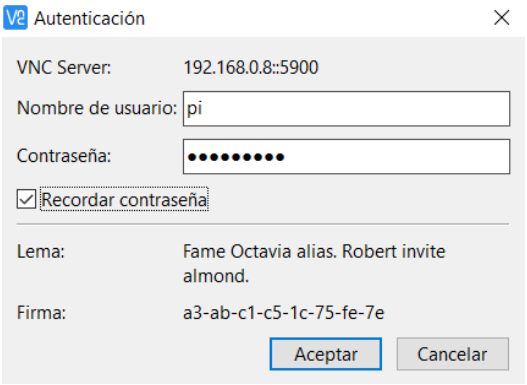

Figura 46: Datos conexión.

Fuente: Cea, 2017.

Se hace click en aceptar, y establecemos la conexión con la Raspberry Pi 3, permitiendo controlarla desde nuestro computador, utilizando nuestro teclado y mouse para realizar acciones en ella como se aprecia en la figura 47.

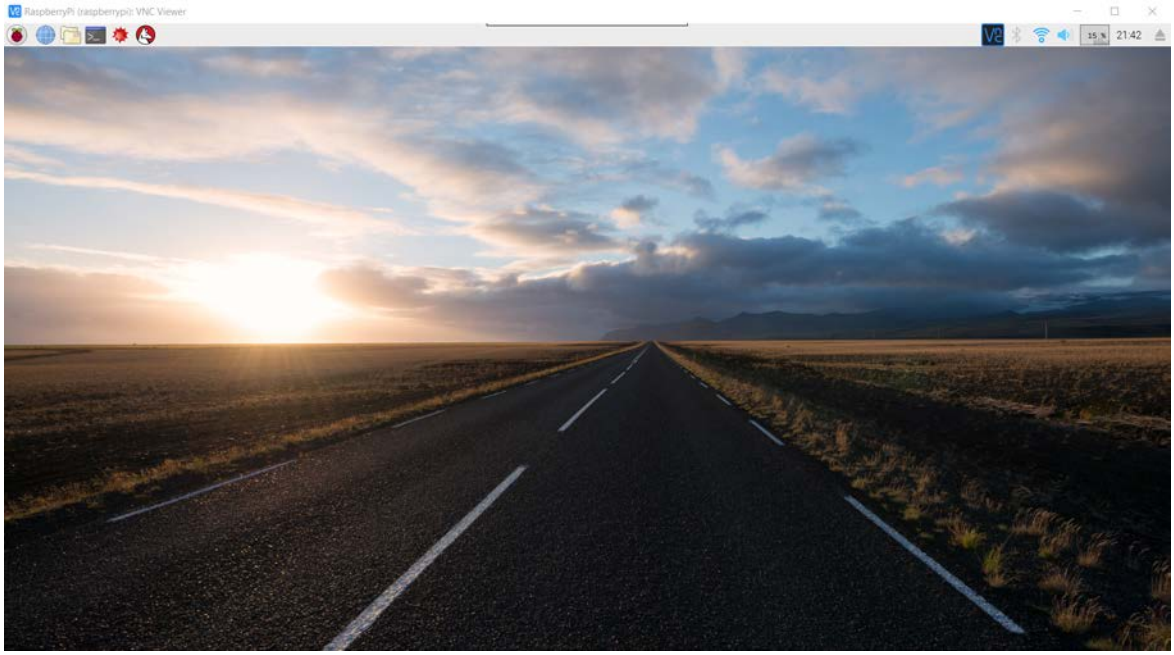

Figura 47: Vista desde aplicación VNC desde Windows.

Fuente: Cea, 2017.

3.7.3. Configuración desde Android

VNC tambien cuenta con una aplicación de smartphone, que se encontra en AppStore desde iOS, y tambien en PlayStore desde dispositivos Android, en la figura 48 se muestra la interfaz en un dispositivo Android, en donde se ingresa la IP que muestra VNC server desde la Raspberry. Y se le asigna el nombre con el que se guarda la configuración.

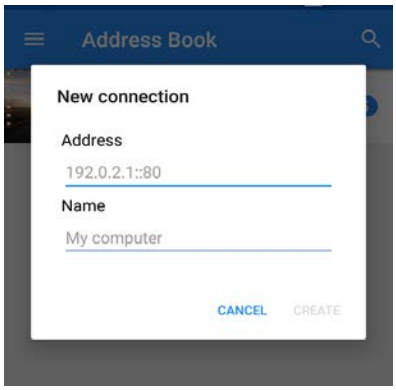

Figura 48: Configuración VNC celular.

Fuente: Cea, 2017.

Nos solicita ingresar el usuario y contraseña tal como se aprecia en la figura 49 por defecto los datos son “pi” en usuario, y “raspberry" en contraseña, se puede seleccionar la opción Remember password para guardar los datos, para no volver a ingresarlos cada vez que se necesite utilizar la aplicación.

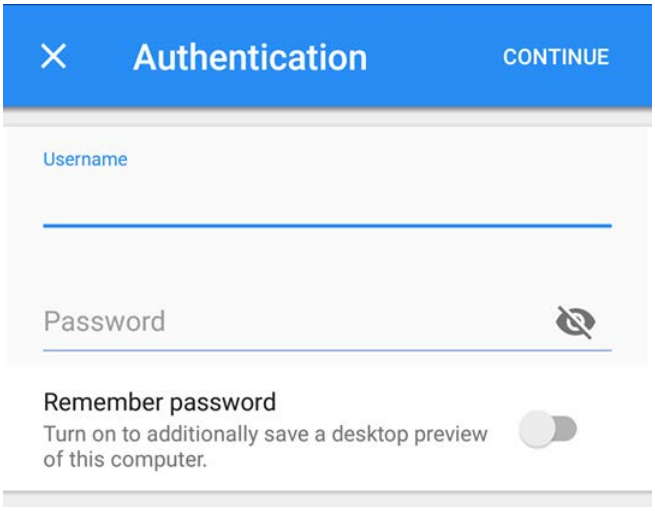

Figura 49: Datos VNC cliente.

Fuente: Cea, 2017.

Como resultado de configurar la aplicación movil, se obtiene como resultado la visualización de la Raspberry Pi en el smartphone o computador. En la figura 50 se puede ver la conexión desde el dispositivo Android de manera funcional

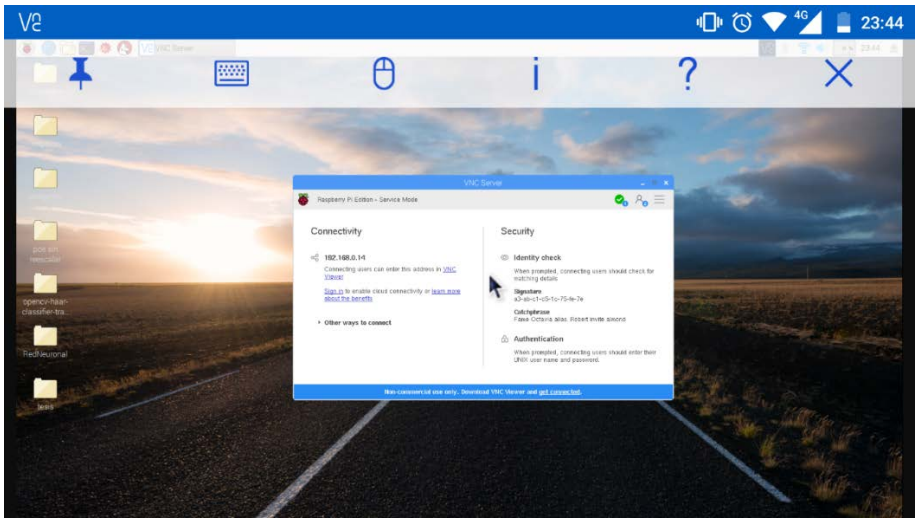

Figura 50: Conexión desde Android.

Fuente: Cea, 2017.

4. Prototipo.

4.1. Introducción

En este capítulo se detalla la construcción del prototipo, que muestra las funciones seleccionadas que aportan con su función.

Se destaca que se intentó realizar un prototipo que pudiera utilizar todas las funciones en conjunto, pero dada la capacidad de computo con la que cuenta la Raspberry Pi 3, no fue posible su realización, por lo cual se opta por realizar prototipos con las funcionalidades por separado.

Se puede efectuar un prototipo en conjunto, utilizando más de una Raspberry Pi 3, distribuyendo la carga de procesamiento entre ellas, y sincronizándolas para trabajar en conjunto; así también trabajar con un arduino en el control de servomotores. De igual forma, se puede utilizar el sistema RoS [18] que logra utilizar de mejor forma los recursos disponibles al ser un sistema operativo diseñado para la robótica, pero del cual no se detalla en profundidad ya que no concierne al tema principal de este trabajo.

4.2. Materiales

Para la construcción de este prototipo se utilizan los siguientes componentes:

- Raspberry Pi 3 – Figura 51.
- PiCamera v1.3 – Figura 52.
- Auto a control remoto – Figura 53.
- Protoboard – Figura 54.
- Cables varios.
- Teclado.
- Mouse.
- Pantalla.
- Celular.

[18] http://www.ros.org/ - Robot Operating System

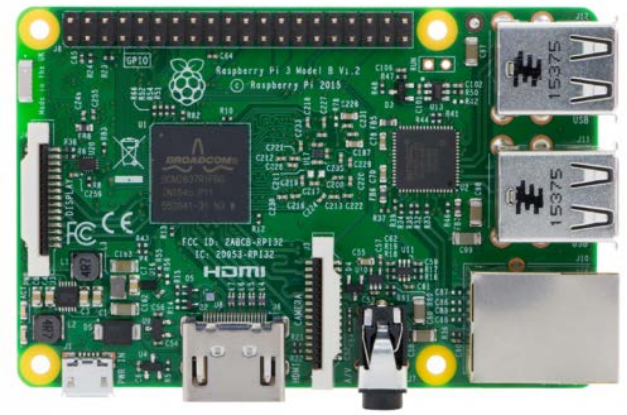

Figura 51: Raspberry Pi 3.

Fuente: https://www.raspberrypi.org/.

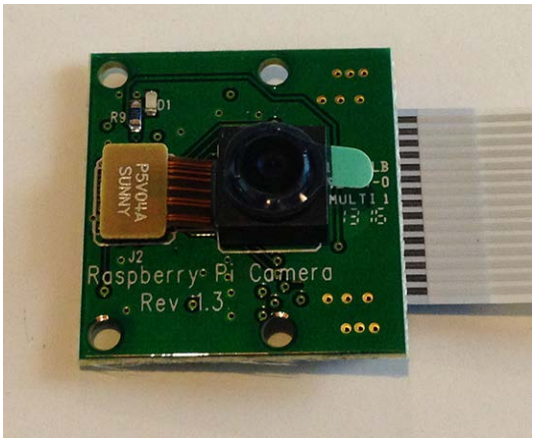

Figura 52: Raspberry Pi Camera.

Fuente: Cea, 2017.

Figura 53: Auto a control remoto.

Fuente: Cea, 2017.

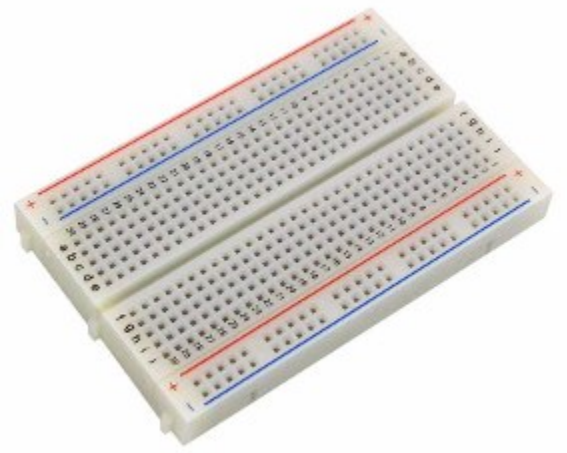

Figura 54: Protoboard.

Fuente: Cea, 2017.

4.3. Construcción

La construcción del prototipo comienza con la conexión de la PiCamera en el puerto correspondiente en la Raspberry Pi 3. Luego se toman las medidas de dichos componentes, para adquirir un auto a control remoto que tenga la capacidad interna para albergar la Raspberry Pi en su interior.

Se le realizan modificaciones externas al auto, para poder tener un mejor acceso a la Raspberry, evitando tener que desarmar el vehiculo cada vez que fuera necesario utilizar la placa; para lo cual se efectúan cortes en la puerta trasera del auto para dejar libre acceso a los puertos USB y Ethernet. Se adaptan partes internas del chasis del auto para poder crear una plataforma de sujeción, que es ubicada en la parte posterior del vehiculo, calzando con los cortes efectuados en la puerta.

Se realiza un pequeño corte en la parte superior del armazón, para instalar una pieza diseñada e impresa en 3d como soporte de la cámara, esta cuenta con un ángulo de 25,5° de inclinación, así la cámara apunta en dirección al suelo, contando con aproximadamente 3 veces el largo del vehiculo de distancia en la captura de la imagen.

Se incluye una protoboard[19] en conjunto con un circuito integrado L293D, que permite controlar 2 motores dc[20], los cuales se utilizan para el control de movimiento del agente autónomo.

En la parte frontal del vehiculo se aprecia un sensor ultrasónico modelo HC-SR04, que puede ser utilizado para la detección de objetos que pueden llegar a entorpecer la ruta seleccionada por el auto.

El integrado L293D y el sensor ultrasónico HC-SR04 tienen una conexión directa con la Raspberry Pi 3, conectándose en los pines definidos en el software mediante el GPIO[21].

4.4. Implementación de ejemplos

En esta sección se detalla el proceso de implementación de los ejemplos desarrollados para las diversas funcionalidades que se decidieron documentar.

4.4.1. Captura imagen con cámara.

Capturar una imagen con la cámara es el primer paso que se debe realizar para verificar el funcionamiento de la cámara, el lenguaje de programación Python y los complementos instalados para su funcionamiento.

En la figura 55 se aprecia el código para tomar y guardar una fotografía de forma sencilla, empezando por la sección donde se importan las librerías en las lineas 1 y 2, se cuenta con time que facilita implementar contadores de forma muy sencilla, preguntando al sistema la diferencia de tiempo desde que se solicitó la operación; y luego picamera, que es la librería con la que se conecta directamente al hardware de la cámara.

Se configura la resolución de la cámara en 640x480 pixeles en la línea 5, posteriormente comienza una pre visualización de lo que está captando la cámara en ese momento con la línea 6, y luego en la línea 7 se realiza una espera de dos segundos para efectuar la captura de la fotografía, siendo almacenada con el nombre que se le pasa al método en la línea 8 del código.

[19] Tablero de pruebas, que tiene orificios para conectar elementos de electrónica
[20] Direct current (Corriente continua)
[21] General Purpose Input/Output, Entrada/Salida de Propósito General

```
1  import time
2  import picamera
3
4  with picamera.PiCamera() as camera:
5      camera.resolution = (640, 480)
6      camera.start_preview()
7      time.sleep(2)
8      camera.capture('captura.jpg')
9
```

Figura 55: Captura imagen con cámara.

Fuente: Cea, 2017.

En la figura 56 se aprecia el resultado de la ejecución del código para capturar imágenes desde la cámara.

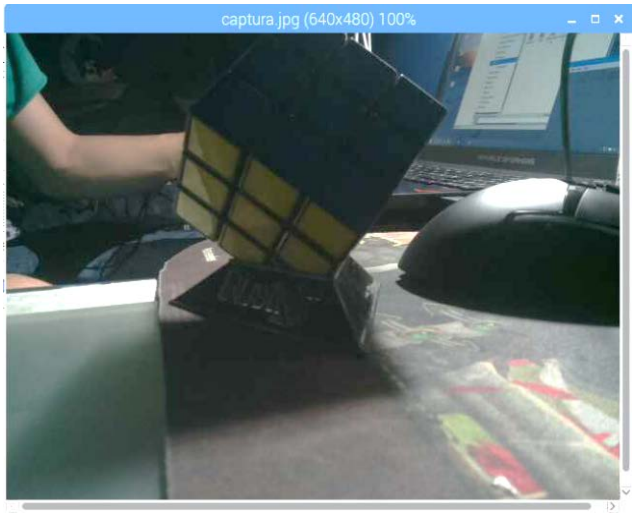

Figura 56: Captura de imagen con cámara.

Fuente: Cea, 2017.

Es importante mencionar que, para realizar una captura correcta, hay que poner atención en el parámetro que le se le pasa al método capture, ya que si no contiene una extensión compatible no será guardada.

Las extensiones compatibles son jpg, png, gif, bmp, yuv, rgb, rgba, bgr, raw; todas pertenecientes a tipos de imagen, siendo las más comunes jpg, png y bmp.

4.4.2. Captura continua con cámara.

La captura continua es la base funcional del prototipo, siendo esta la que va entregando cada uno de los frames necesarios para realizar el análisis correspondiente de cada elemento que se busca clasificar dentro del frame. En la figura 57 se muestra el código realizado para esta tarea.

```
1  from picamera.array import PiRGBArray
2  from picamera import PiCamera
3  import time
4  from time import sleep
5  import cv2
6
7  #Configuracion de camara
8  camera = PiCamera()
9  camera.resolution = (320,240)
10 camera.framerate = 15
11 rawCapture = PiRGBArray(camera, size=(320,240))
12
13 for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port= True):
14     #frame
15     image = frame.array
16
17     #muestra frame
18     cv2.imshow("Frame", image)
19
20     key = cv2.waitKey(1) & 0xFF
21
22     rawCapture.truncate(0)
23
24     if key == ord("q"):
25         break
```

Figura 57: Captura continua con cámara.

Fuente: Cea, 2017.

Entre la línea 1 y la línea 5 se encuentra el código necesario para realizar el importe de las librerías necesarias; luego en la línea 8 se declara que la variable *camera* será la que contendrá los datos recolectados por la PiCamera , luego se define la resolución en la línea 9, y la cantidad de frames por segundo que se capturan en la línea 10, la línea 11 define que *rawCapture* tendrá la información completa de la imagen capturada, no tan solo la imagen procesada, pudiendo acceder mediante a esta a cada uno de los canales presentes en la imagen. En el ciclo for definido en la línea 13, se decalara que la captura de imágenes se efectuará de manera continua, y la información recopilada por la camará será guardada en la variable *rawCapture*, en formato bgr utilizando el puerto de video disponible en la placa de la Raspberry Pi, se ejecuta esta fracción de código hasta que no existan más frames disponibles capturados por la cámara. Dentro del ciclo lo primero que se hace es asignarle a *image* el frame actual capturado, del cual solo se utiliza la imagen en bruto y no la información sobre

sus canales, ni formato de imagen. Para luego ser mostrados con cv2.imshow("Nombre_ventana", image) en la línea 18 del código, donde *Nombre_ventana* es el título con el que se quiere mostrar la interfaz, e *imagen* es el frame a mostrar. Luego como se muestra en la línea 24 se puede detener la ejecución del código en cualquier momento presionando la tecla "q".

4.4.3. Entrenamiento red neuronal.

En esta sección se explica el proceso de entrenamiento necesario para crear una red neuronal funcional, enfocada en el proceso de reconocimiento de imágenes.

En este caso se toma como ejemplo el entrenamiento realizado para la detección de la señal de transito pare, con la salvedad de que la señal estará en inglés dado que fue mucho mas sencillo conseguir más muestras de imágenes con el signo pare en este idioma dado el elevado número de muestras necesarias.

El primer paso es conocer de qué trata una red neuronal, este es un sistema conexionista, que, dado ciertas entradas, realiza procesos internos y entrega una respuesta; todo esto siendo imperceptible para el usuario.

Sabiendo esto, comienza el proceso de entrenamiento; en principio hay que tener claro un principio fundamental, este es un sistema que se basa en realizar en aprender a discriminar entre muestras positivas y muestras negativas.

Las muestras positivas son aquellas donde se aprecia en totalidad el objeto a identificar, estas muestras deben contener al objeto visto desde distintos puntos de vista y en diferentes condiciones de luminosidad, para generar un entrenamiento más certero abarcando no solo la forma, si no que incluyendo la intensidad del color, la interferencia de la luz en el objeto y la posición de este en el ambiente. En cambio, las muestras negativas son imágenes que no tienen al objeto a identificar; se recomienda utilizar imágenes que sean acorde al problema, en este caso, se utilizan muestras negativas de calles con edificios, calles sin edificios, o diversos paisajes que contengan calles, ya que son parte del contexto de vialidad.

La colección de muestras es un proceso que requiere mucho tiempo, en este ejemplo se utilizaron 2000 muestras negativas, contra 1000 muestras positivas, a su vez, cada muestra

positiva se le aplica una transformación desde sus ocho perfiles, para obtener más muestras desde la misma imagen como se aprecia en la figura 58.

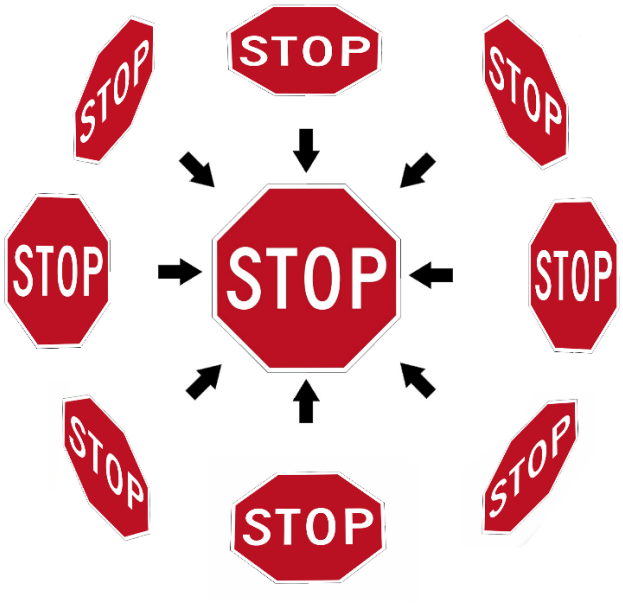

Figura 58: Distorción de imagen desde ocho perfiles.

Fuente: Cea, 2017.

El siguiente paso es crear un documento de texto, que contiene el nombre de las muestras positivas; y otro documento de texto que contenga el nombre de las muestras negativas. Para este propósito, desde la Raspberry se puede utilizar el comando de la figura 59.

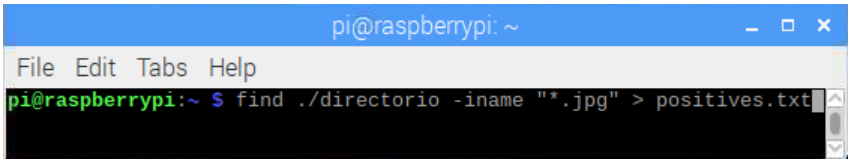

Figura 59: Comando find positives.

Fuente: Cea, 2017.

Este comando lo que hace es recuperar el nombre de todos los archivos presentes en el directorio con la extensión jpg y guardarlos en el documento positives.txt. Para las muestras negativas se utiliza el comando de la figura 60, realizando el mismo procedimiento que para

las muestras positivas pero guardandolas en el documento negatives.txt. Estos documentos se crearán en donde se ejecute el comando desde el terminal.

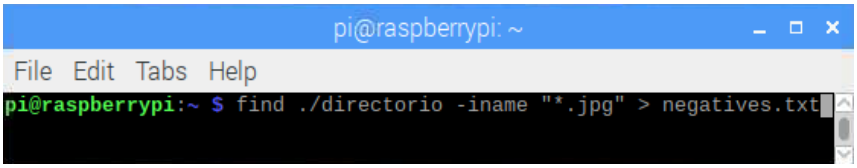

Figura 60: Comando find negatives.

Fuente: Cea, 2017.

A continuación, se realiza la creación de los ejemplos, que son la combinación de las muestras negativas con las positivas. Para esto se utilizan los archivos positives.txt y negatives.txt creados en el paso anterior, y se ingresa en la consola el comando de la figura 61.

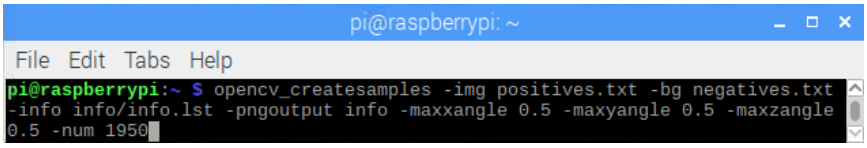

Figura 61: CreateSamples.

Fuente: Cea, 2017.

Con la ejecución de este comando se obtienen imágenes que tienen como base la muestra negativa, y a su vez una muestra positiva en un sector aleatorio de la imagen, guardando estas imágenes en la carpeta info, definido por el parámetro –pngoutput. El archivo info.lst contiene una lista de las combinaciones realizadas por el módulo createsamples, siguiendo el siguiente patrón “0001_0014_0045_0028_0028.jpg 1 14 45 28 28”, el primer elemento es la imagen generada seguido por la extensión jpg, el número que lo sigue son la cantidad de elementos positivos que se encuentran en la imagen, los dos siguientes son las coordenadas donde se ubica, y los últimos dos son el ancho y alto de la muestra positiva. El parámetro –img puede ser una imagen única, en caso de querer entrenar la red para reconocer un objeto único que no variará. Los parámetros –maxxangle, -maxyangle y –maxzangle son las transformaciones que se realizan a las muestras positivas, pudiendo estar hasta un 0.5 de inclinación en el espacio tridimensional, esto facilita la identificación del objeto desde diferentes perspectivas, evitando que tenga que estar de la misma forma siempre; y el último parámetro –num, es el número de muestras con las que se cuenta, se

recomienda no incluir el total exacto de muestras, para generar un margen de error en caso de que el algoritmo de creación de ejemplos no pueda localizar la muestra correctamente.

Luego generaremos un archivo con extensión vec, que contiene toda la información del documento lst, para cada una de las muestras positivas, y el que podemos crear con el siguiente comando de la figura 62.

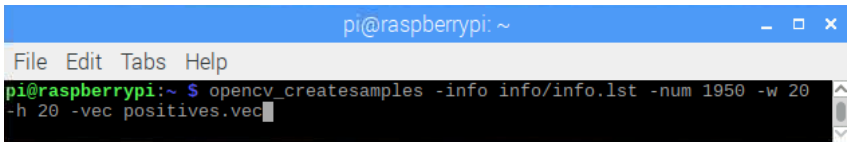

Figura 62: Crear vector.

Fuente: Cea, 2017.

Donde –w y –h son las medidas con las que se quiere redimensionar las muestras positivas, con esto se logra que el objeto se pueda detectar desde una distancia mayor.

El siguiente es paso final del entrenamiento, ya que se cuenta con todo lo necesario para su realización. Se ejecuta el comando de la figura 63 desde el terminal.

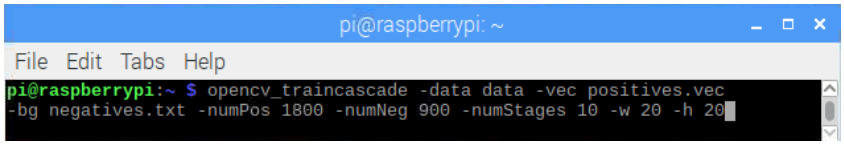

Figura 63: TrainCascade.

Fuente: Cea, 2017.

Donde traincascade es el método proporcionado por OpenCV para entrenar la red neuronal, data es la carpeta en donde se guarda nuestra red entrenada, y nuevamente se reduce el número de elementos que le entregamos a –numPos, que es el valor de muestras positivas, a pesar de originalmente tener 1000 muestras, se puede ampliar dado las tranformaciones realizadas en ellas, que con esto se evita que el algoritmo de entrenamiento utilice todas las muestras disponibles en cada iteración, haciendo que las muestras positivas no se repitan y la red no solo memorice como es el objeto, si no que aprenda a discriminar si corresponde a el o no; esta situación se puede explicar de la siguiente forma, para obtener el color anaranjado, necesitamos mezclar rojo y amarillo, si la red memoriza solo responde que la combinación de rojo y amarillo forman el anaranjado; en cambio, si la red aprende a

discriminar, conoce que al mezclar amarillo y rojo también se puede obtener anaranjado. Por otra parte, se disminuye el número de muestras negativas, ya que al igual que con las muestras positivas, se evita la repetición de las muestras negativas entre cada iteración; se define que el entrenamiento se realice en 10 iteraciones, ya que en la documentación de OpenCV (OpenCVTeam, OpenCV Library, 2017), se recomienda realizar el entrenamiento en esta cantidad de iteraciones; cada una de estas lo que realiza es identificar en que sector en específico se ubica y como se ve el elemento dentro de la imagen tomando en cuenta su color, forma específica, orientación y la luminosidad que pueda tener el objeto.

Ejecutar el entrenamiento de la red neuronal en la raspberry, tomó aproximadamente nueve horas para su realización, pero se pueden efectuar en otro computador que posea un procesador con mejor desempeño de procesamiento, realizar el entrenamiento en otro equipo, no excluye la utilización de la red neuronal en OpenCV instalado en la Raspberry Pi.

Es importante mencionar que, durante el transcurso del entrenamiento, la Raspberry utiliza el 100% de su capacidad de procesamiento, por tanto, no es posible realizar otras acciones con ella.

4.4.4. Detección signo pare en tiempo real.

Para la detección del signo pare, utilizamos como base la captura continua de imagen, y en base a ello añadimos diversas funcionalidades presentes en la librería OpenCV.

```
1  from picamera.array import PiRGBArray
2  from picamera import PiCamera
3  import time
4  from time import sleep
5  import math
6  import cv2
7  import numpy as np
8
9  #Asignacion red neural a variable
10 stop_cascade = cv2.CascadeClassifier('stop_sign.xml')
11
12 #Configuracion de camara
13 camera = PiCamera()
14 camera.resolution = (320,240)
15 camera.framerate = 15
16 rawCapture = PiRGBArray(camera, size=(320,240))
17
18 for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port= True):
19     #frame
20     image = frame.array
21
22     #transformacion imagen a escala de grises
23     gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
24
25     #deteccion de signo pare
26     stop = stop_cascade.detectMultiScale(gray_image,1.3,5)
27     for (x,y,w,h) in stop:
28         cv2.rectangle(image, (x+5,y+5),(x+w-5, y+h-5), (255,0,0),2)
29         cv2.putText(image, 'STOP', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
30
31     #Mostrar imagen
32     cv2.imshow("Frame", image)
33
34
35     key = cv2.waitKey(1) & 0xFF
36
37     rawCapture.truncate(0)
38
39     if key == ord("q"):
40         break
```

Figura 64: Código detección signo pare en tiempo real.

Fuente: Cea, 2017.

El código que se muestra en la figura 64, tiene la siguiente estructura; en principio cuenta con la sección de importe de librerías entre las líneas 1 a la 7, luego se asigna la red neural a un objeto en la línea 10, que nos permite realizar la detección del signo pare. Luego cuenta con la configuración de la cámara y la captura de los frames en las líneas 13 a 18. Se define un ciclo que se repete continuamente capturando la información mediante la cámara, y dentro de el, lo primero que se realiza es asignar a *image* el frame actual, para luego efectuar la transformación de cada uno de ellos a escala de grises, ya que así se realiza una detección más rápida del objeto a identificar; para luego en base a la variable *stop_cascade* definida para la red neuronal, determinar si en alguna coordenada del frame se encuentra el objeto a identificar, ayudados por el método *detectMultiScale()*, que realiza el análisis de la imagen en

busqueda del elemento; este devuelve la coordenada en caso de encontrar una coincidencia, así tambien el ancho y alto del objeto, con el cual se dibuja un rectángulo en su exterior, y lo marca con la etiqueta STOP gracias al método *putText()*, en las líneas 30 y 31 respectivamente.

Luego se muestra la imagen con el rectángulo y el texto mediante una ventana, en este caso llamada Frame, y en la figura 65 se aprecia el resultado de ejecutar este código.

Hay que tener presente que en condiciones de poca luminosidad el reconocimiento de las imágenes es mucho más complejo, por tanto, para estos ejemplos se utiliza una iluminación alta para simplificar el reconocimiento.

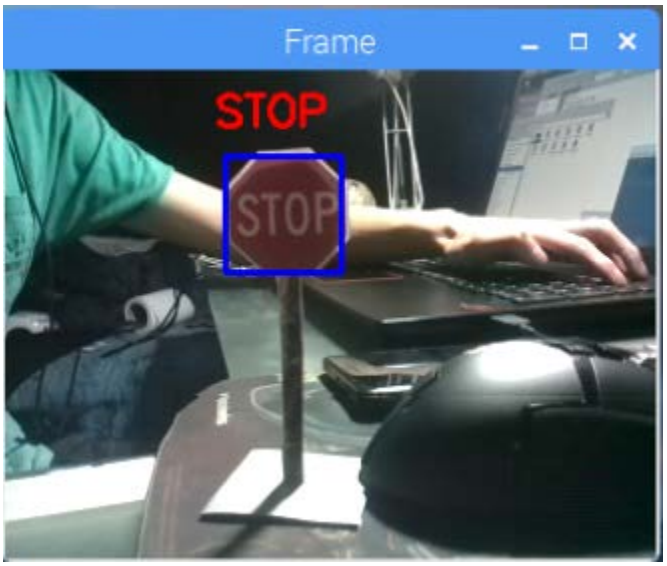

Figura 65: Detección signo pare.

Fuente: Cea, 2017.

4.4.5. Detección de las líneas de la calle en tiempo real.

En esta sección hablaremos del proceso más complejo dentro de los prototipos que se realizaron, ya que conlleva una mezcla de varias funciones presentes en OpenCV.

```
1  from picamera.array import PiRGBArray
2  from picamera import PiCamera
3  import time
4  from time import sleep
5  import math
6  import cv2
7  import numpy as np
8
9  #Configuracion de camara
10 camera = PiCamera()
11 camera.resolution = (320,240)
12 camera.framerate = 15
13 rawCapture = PiRGBArray(camera, size=(320,240))
14 sigma = 0.1
15 time.sleep(0.1)
16
17 def draw_lines(img, lines, color=[255, 0, 0], thickness=10):
18
19     imshape = img.shape
20     left_x1 = []
21     left_x2 = []
22     right_x1 = []
23     right_x2 = []
24     y_min = img.shape[0]
25     y_max = int(img.shape[0]*0.611)
26     for line in lines:
27         for x1,y1,x2,y2 in line:
28             if ((y2-y1)/(x2-x1)) < 0:
29                 mc = np.polyfit([x1, x2], [y1, y2], 1)
30                 left_x1.append(np.int(np.float((y_min - mc[1]))/np.float(mc[0])))
31                 left_x2.append(np.int(np.float((y_max - mc[1]))/np.float(mc[0])))
32             elif ((y2-y1)/(x2-x1)) > 0:
33                 mc = np.polyfit([x1, x2], [y1, y2], 1)
34                 right_x1.append(np.int(np.float((y_min - mc[1]))/np.float(mc[0])))
35                 right_x2.append(np.int(np.float((y_max - mc[1]))/np.float(mc[0])))
36     l_avg_x1 = np.int(np.nanmean(left_x1))
37     l_avg_x2 = np.int(np.nanmean(left_x2))
38     r_avg_x1 = np.int(np.nanmean(right_x1))
39     r_avg_x2 = np.int(np.nanmean(right_x2))
40     cv2.line(img, (l_avg_x1, y_min), (l_avg_x2, y_max), color, thickness)
41     cv2.line(img, (r_avg_x1, y_min), (r_avg_x2, y_max), color, thickness)
42
```

Figura 66: Detección de líneas, configuración.

Fuente: Cea, 2017.

Como se puede apreciar en la figura 66 se comienza añadiendo los import necesarios entre las líneas 1 y 7 del código, en este caso se utiliza la cámara, la librería math para cómputos y operaciones matemáticas, Numpy para transformaciones numéricas, y cv2 para análisis de imágenes.

Luego se detalla la configuración de los parámetros de la cámara, para lo cual en la línea 10 se asigna a *camera* la referencia a Picamera. Se establece su resolución a 320x240, capturando 15 frames por segundo en las líneas 11 y 12 respectivamente; esto se hace para no sobrecargar el procesador de la Raspberry y dado la velocidad con la que se mueve el vehiculo, no es necesaria una captura de frames mayor, en la línea 13 guardamos la información completa de la imagen en la variable *rawCapture*.

Lo siguiente que se realiza es definir un método que se especifica entre las líneas 17 a la 41, este método se encarga de dibujar las líneas encontradas en la imagen que le son pasadas por parámetros, de un color y transparencia definidos.

Este método en términos simples realiza el análisis de una imagen que, dado el lugar donde se realiza la ejecución, solo cuenta con una parcialidad de la imagen definida en una región de interés, conteniendo los límites de la calle dados por las líneas presentes en ella.

Se analiza esta región de la imagen en el cilo definido en la línea 26, donde para cada una de las líneas de la calle encontradas en la región de interés, se obtendrán sus extremos, se calcula si corresponde a la línea izquierda o derecha de la calle con la sentencia if de la línea 28 y se guardan en las variables definidas en las líneas 20 a la 23, el cálculo realizado en la línea 30, corresponde a la obtención de un valor sin decimales, dado los puntos encontrados en la imagen analizada, tomando como referencia la línea mas próxima entre los puntos, calculado en la línea 29 y 33 gracias al método polyfit presente en la librería NumPy; este mismo procedimiento se realiza en las líneas 31, 34 y 35 , estos datos serán utilizados para crear dos líneas con cv2.line() en las líneas 40 y 41 del código, que tienen por parámetros la imagen y las coordenadas, para dibujar en una nueva capa sobre la imagen original ambas líneas.

Hay que destacar que cv2.line(), no modifica la imagen que utiliza como base, por tanto, su valor sigue siendo el mismo que desde el inicio.

Luego de definir el método *draw_lines*, empezamos con el análisis frame a frame, en donde se ejecutan las siguientes instrucciones indicadas en la figura 67.

```
40      cv2.line(img, (l_avg_x1, y_min), (l_avg_x2, y_max), color, thickness)
41      cv2.line(img, (r_avg_x1, y_min), (r_avg_x2, y_max), color, thickness)
42
43  for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port= True):
44      #frame
45      image = frame.array
46
47      #fraccion del frame
48      part_image = image
49
50      #definicion region de interes
51      points = np.array([[(0,240),(320,240),(226,100),(94,100)]], dtype=np.int32)
52
53      #transformacion imagen a escala de grises
54      gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
55
56      #tratamiento imagen
57      mask = np.zeros_like(gray_image)
58      ignore_mask_color = 255
59      cv2.fillPoly(mask, points, ignore_mask_color)
60      masked_image = cv2.bitwise_and(gray_image, mask)
61      v = np.median(gray_image)
62      lower = int(max(0, (1.0 - sigma) * v))
63      upper = int(min(255, (1.0 + sigma) * v))
64      edged = cv2.Canny(masked_image, lower, upper)
65      rho = 2
66      theta = np.pi/180
67      threshold = 65
68      min_line_length = 40
69      max_line_gap = 100
70      line_image = np.copy(image)
71      lines = cv2.HoughLinesP(edged, rho, theta, threshold,np.array([]), min_line_length, max_line_gap)
72      line_img = np.zeros((image.shape[0]), dtype = np.uint8)
73      draw_lines(line_image, lines)
74      weighted = cv2.addWeighted(image, 0.8, line_image, 1.,0.)
75
76      #muestra frame completo
77      cv2.imshow("Frame", weighted)
78
79
80      key = cv2.waitKey(1) & 0xFF
81
82      rawCapture.truncate(0)
83
84      if key == ord("q"):
85          break
```

Figura 67: Detección de líneas, ejecución.

Fuente: Cea, 2017.

En la línea 45 de la figura 67, se obtiene el frame que se utiliza para analizar, y este se duplica en la variable *part_image* definido en la línea 48, con esto, se mantiene un respaldo de la imagen sin modificar para no alterar el contenido original, luego en la línea 51 se define el vector *points*, que contiene cual es la región que interesa analizar en profundidad; se transforma a escala de grises la imagen guardada en la línea 54, ya que esto facilita el análisis que se desea realizar.

Lo siguiente es realizar un tratamiento especial a la imagen, esto se lleva a cabo en las líneas 57 a la 74, lo primero que se hace en la línea 57 es transformar la imagen en escala de grises a una matriz de ceros y unos, donde todo lo de color blanco será 1, y los demás colores 0 definidos por la variable *ignore_mask_color* en la línea 58; luego se guarda lo necesario de la imagen transformada en matriz en la línea 59, dado por los puntos definidos anteriormente, detallando que ignore el blanco; y se define esto como una máscara o capa.

El paso que sigue es asociar ambas imágenes, esto lo se realiza en la línea 60, y para esto utilizamos *cv2.bitwise_and()*, que toma la imagen en escala de grises y la matriz realizada en la línea 59 del código.

Se analiza la imagen buscando cual es el punto medio entre la tonalidad más oscura y la más clara, para esto se utiliza el método *np.median()* definido en la línea 61, a la cual se le pasa por parámetro la imagen, en este caso, la imagen transformada a escala de grises. Así también se calcula el umbral menor y mayor de la imagen, efectuado en las líneas 62 y 63 respectivamente; esto lo se utiliza para destacar los bordes de las figuras presentes, mediante el método *cv2.Canny()* utilizado en la línea 64, donde la variable *edged* contendrá esta información.

Se definen las variables *rho*, *theta*, *threshold*, *min_line_lenght* y *max_line_gap* en las líneas 65 a la 69, que se han definido con los valores recomendados por los autores de OpenCV (OpenCVTeam, OpenCV Library, 2017), estos aportan a la identificación de las líneas continuas, tanto como las líneas discontinuas presentes en la calle, *rho* es una constante, *theta* es el ángulo posible de las líneas presentes en la calle, *threshold* funciona como el límite del ancho de estas líneas, *min_line_lenght* corresponde al tamaño mínimo identificable para la línea discontinua de la calle y se relaciona con *max_line_gap* que determina la separación máxima entre los segmentos de la línea.

Luego se realiza la transformación de imagen *HoughLinesP()* en la línea 71, que utiliza las variables definidas anteriormente para predecir qué dirección tendrán las siguientes líneas que aparecerán en la imagen. El valor retornado se almacena en el vector definido en la línea 71, que contiene las coordenadas de la predicción de las líneas, que se utilizan para dibujar mediante la línea 73, sobre la copia del frame actual definido en la línea 70; esta luego se fusiona con el frame actual en la línea 74. Lo siguiente es mostrar la información obtenida mediante el metodo *imshow* presente en la línea 77, y el resultado se aprecia en las figuras 68 y 69.

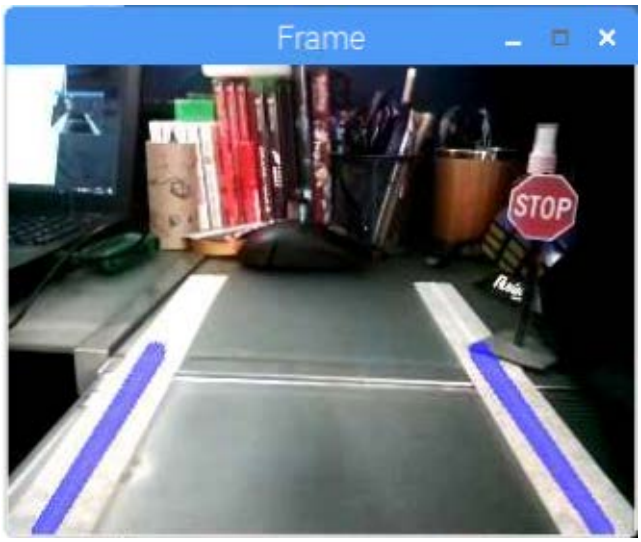

Figura 68: Ambiente simulado.

Fuente: Cea, 2017.

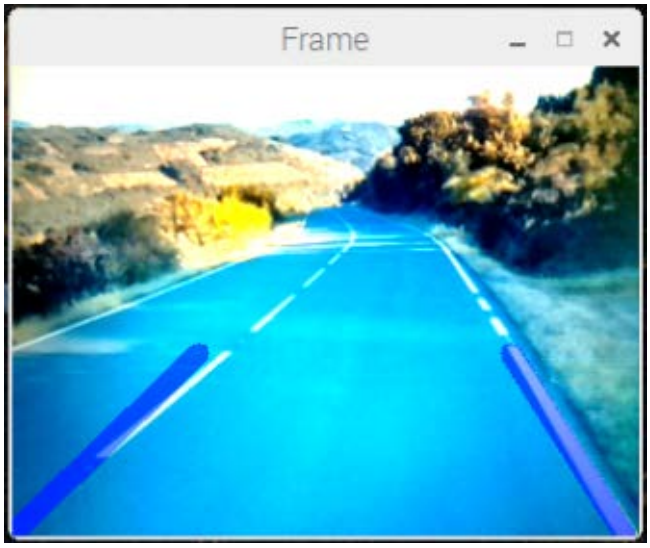

Figura 69: Ambiente real.

Fuente: Cea, 2017.

La figura 68 se efectúa en una recreación de una pequeña calle, y en la figura 69 se realiza utilizando un video de un vehiculo real avanzando por la calle. Esto muestra que, a pesar de ser un prototipo a baja escala, puede ser aplicado tanto en ambientes controlados, tales como competencias donde el robot tenga que mantenerse dentro de un circuito, y a su vez en situaciones reales como lo es un vehiculo motorizado dirigido por las calles.

4.5. Dificultades

Las dificultades encontradas durante la realización del prototipo fueron escasas, ya que la mayoría de los métodos a utilizar presentes en OpenCV, cuentan con documentación oficial sobre su funcionamiento, parámetros requeridos, datos retornados; y a su vez, cuenta con una comunidad que la utiliza de forma activa, y se pueden encontrar ejemplos simples sobre algunas funcionalidades.

La principal dificultad fue intentar realizar un prototipo que utilice las herramientas aportadas por OpenCV de forma conjunta, el cual no fue posible dado la capacidad de computo que puede soportar la Raspberry Pi. Es importante mencionar que se intenta realizar, pero el procesamiento es tan lento, que no se podía trabajar con más de una imagen por segundo, y no permite que el agente robótico desarrolle un buen desempeño.

Python representó una dificultad al inicio del desarrollo del proyecto, ya que dicho lenguaje actualmente no forma parte del plan de estudios, ni como parte del plan común, ni de los electivos, por lo que se presenta como un desafío aprender dicho lenguaje y poder aplicarlo para su desarrollo.

Entrenar una red neuronal no era parte de la primera propuesta, pero dado que se decide realizar el reconocimiento de la señal PARE, se interioriza su utilización. Se investiga su funcionamiento y entrenamiento; esto fue una de las partes que conllevó más tiempo para su implementación, ya que el entrenamiento de esta red neuronal, en un principio no funcionó dado la falta de muestras, tanto negativas como positivas, esto acasionó que se tuvieran que recolectar más muestras, para luego volver a intentar llevar a cabo el entrenamiento, este proceso se realizó tres veces, en principio con 100 muestras, luego con 500, para luego terminar con 1000 muestras; cuando se pudo realizar el entrenamiento completo tardó aproximadamente 9 horas utilizando la Raspberry Pi 3.

4.6. Trabajos futuros

1. Como trabajo futuro, se puede finalizar con el desarrollo del prototipo, cambiando el enfoque que se le dio al proyecto. Empezando por un cambio en el sistema operativo, para dar paso a RoS[22], que optimiza mucho más los procesos al ser un sistema operativo diseñado para robótica.
2. También cambiar la arquitectura de hardware del prototipo, utilizando la Raspberry exclusivamente como un recolector de datos. Para esto, se puede un computador que esté conectado a Raspberry mediante una red inalámbrica, efectuando las tareas pesadas de procesamiento en una CPU[23] que posee una capacidad de computo mucho más elevada.
3. Añadir al módulo de detección de las líneas de la calle, que tan distante se encuentra en ese momento alejado del centro, para así poder mantener al robot o vehiculo en una ubicación óptima.

[22] Robot Operating System.
[23] Por su sigla en inglés, Central Processing Unit / Unidad central de procesamiento.

5. Conclusiones.

Al investigar, y adentrarse en el entorno de la visión artificial, se demuestra como esta área de la informática ha ido creciendo con el paso del tiempo, comenzando a ver prototipos completos y funcionales de vehículos dirigidos por computador, que recogen información del ambiente y la procesan para tomar desiciones; a su vez, OpenCV ha sido un aporte añadiendo más características importantes y significativas con las actualizaciones de sus versiones, por ejemplo la optimización del entrenamiento de la red neuronal, la detección más eficiente de figuras geométricas y la compatibilidad con más lenguajes de programación.

En la actualidad, se están desarrollando muchos proyectos apoyados por esta tecnología, que nos permite trabajar en diversas plataformas, e incluir diversos periféricos como dispositivos móviles, cámaras de todo tipo, desde la clásica webcam usb, hasta el modelo especial hecho para Raspberry, por tanto, la accesibilidad para utilizar OpenCV y la documentación presente en internet nos ayuda de manera valiosa a animarnos a desarrollar proyectos apoyados por esta tecnología.

Así mismo, es importante mencionar que la Universidad del Bío-Bío mediante la carrera de Ingeniería Civil en Informática cuenta con las herramientas necesarias para desarrollar soluciones apoyadas por esta tecnología. También hay muchas teconologías y librerías que aún se desconocen o no se han utilizado, tales como tensorflow y keras, que son librerías para el entrenamiento e implementación de redes neuronales para inteligencia artificial; plataformas de desarrollo, por ejemplo Intel Joule, plataforma desarrollada por intel que cuenta con especificaciones tecnicas de hardware superiores a las de Raspberry Pi 3, contando con más memoría ram, mejor procesador que se traduce en mayor capacidad de computo; también se encuentra UpCore que es similar a Raspberry Pi en tamaño y especificaciones tecnicas, pero a un precio menor; y como consecuencia de estas diversas tecnologías se abren nuevas ventanas hacia el desarrollo, no tan solo de robot diseñados para seguir una línea, o detectar colores; si no que otros con funcionalidades mas complejas tales como reconocimiento de voz para realizar tareas, análisis de crecimiento de frutas en una plantación apoyadas por la visión artificial, etc; con plataformas que son bastante comunes en el sector de la informática, pero que no son utilizados con frecuencia por nosotros.

Con la utilización de la plataforma Raspberry Pi 3 para el desarrollo de esta memoria, se rescata que es una buena plataforma para realizar proyectos, dado la versatilidad que

proporciona contar con un sistema operativo desde donde poder realizar las tareas necesarias, contando con compatibilidad con lenguajes tales como Java, C, C++, Python; librerías para efectuar calculos como Numpy, para utilizar redes neronales y entrenarlas, como ejemplo OpenCV y sus módulos de entrenamiento, Tensorflow y Keras que permiten entrenar y utilizar estas redes; y la mayoría se software disponible para las distribuciones de Linux, que pueden ser instaladas sin mayor complejidad en el sistema. A su vez la compatibilidad con diversos sistemas operativos basados en Linux como Raspbian, Ubuntu Mate, Snappy Ubuntu Core, Pinet, entre otros; añadiendo aun más versatilidad para su utilización. A pesar de sus puntos positivos, también cuenta con puntos negativos, y el que más influyó en esta memoria fueron sus limitadas especificaciones tecnicas a nivel de capacidad de procesamiento, ya que con varias tareas funcionando en paralelo, se hacían notar variadas fluctuaciones en el rendimiento, tornandolo en momentos muy lento hasta el extremo de congelar el sistema operativo.

Las redes neuronales a pesar de su complejidad teórica, terminan siendo un fuerte aliado en la implementación de clasificadores, y que en conjunto con el lenguaje de programación Python utilizado en el desarrollo, facilitaron el desarrollo de los prototipos.

Aprender las nociones básicas de Python no supuso una inversión grande de tiempo, ya que es un lenguaje que cuenta con una sintaxis clara evitando el uso de símbolos, por ejemplo los operadores lógicos ! , || , &&, se escriben directamente como not, or y and respectivamente. Haciendo que los ejemplos encontrados en la documentación oficial de OpenCV escritos en Python fuesen comprendidos al momento de leer. Dada la particularidad de que OpenCV es compatible con Python y el alto volumen de documentación y ejemplos codificados, tanto oficial como de su comunidad, es altamente recomendado interiorizarce en el para desarrollar soluciones apoyadas con robótica.

En general el desarrollo del proyecto no fue complejo, dado la cantidad y calidad de la documentación oficial de OpenCV, pero si se necesitó dedicar una gran cantidad de tiempo para analizar las caracteristicas presentes en la librería, y poder interiorizar los temas necesarios para el correcto funcionamiento de los prototipos, especificamente el área de redes neuronales: también existen limitaciones sobre el análisis de imágenes que se pueden implementar utilizando OpenCV, ya que para realizar funcionalidades más complejas como análisis de patrones, figuras geométricas, análisis de caracteres, etc; necesitamos

complementar a OpenCV con librerías como Numpy, Tensorflow, Math; facilitando los calculos necesarios para su detección.

6. Bibliografía

1. OpenCVTeam. (Octubre de 2017). *Calibración de cámara*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_table_of_contents_calib3d/py_table_of_contents_calib3d.html#py-table-of-content-calib
2. OpenCVTeam. (Octubre de 2017). *Detección de objetos*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_objdetect/py_table_of_contents_objdetect/py_table_of_contents_objdetect.html#py-table-of-content-objdetection
3. OpenCVTeam. (Octubre de 2017). *Funciones principales*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_core/py_table_of_contents_core/py_table_of_contents_core.html#py-table-of-content-core
4. OpenCVTeam. (Octubre de 2017). *Machine learning*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_table_of_contents_ml/py_table_of_contents_ml.html#py-table-of-content-ml
5. OpenCVTeam. (Octubre de 2017). *OpenCV Library*. Obtenido de Open Source Computer Vision Library: https://opencv.org/
6. OpenCVTeam. (Octubre de 2017). *Procesamiento de imagenes*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html#py-table-of-content-imgproc
7. OpenCVTeam. (Octubre de 2017). *Video análisis*. Obtenido de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_video/py_table_of_contents_video/py_table_of_contents_video.html#py-table-of-content-video

8. Press, O. U. (2017). *Oxford Dictionaries*. Obtenido de Oxford Dictionaries: https://es.oxforddictionaries.com/definicion/robot

9. Ripley, B. D. (2008). *Pattern Recognition and Neural Networks.* Cambridge University Press.

10. Sampieri, R. H. (1998). *Metodología de la investigación (Vol. 1).* México: McGraw-Hill.

11. Somerville, I. (2011). *Ingeniería de Software.* México: Pearson.

12. Stanford, U. (2017). *http://www.ros.org*. Obtenido de Robot Operating System.

13. Team, A. (2017). *www.arduino.cc*. Obtenido de Arduino.