Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Lecturas en Ciencias de la Computación
ISSN 1316-6239

Algoritmos de Visión por Computador para un SBC

Francisco Moreno y Esmitt Ramírez

RT 2017-01

Centro de Computación Gráfica
Caracas, marzo 2017

Algoritmos de Visión por Computador para un SBC

Francisco Moreno
Centro de Computación Gráfica, Facultad de Ciencias, Universidad Central de Venezuela
Caracas, Venezuela. 1010-A
francisco.moreno@ciens.ucv.ve

Esmitt Ramírez
Centro de Computación Gráfica, Facultad de Ciencias, Universidad Central de Venezuela
Caracas, Venezuela. 1010-A
esmitt.ramirez@ciens.ucv.ve

***Resumen*—El reconocimiento de objetos, la detección de eventos, la reconstrucción de escenas y la restauración de escenas, son parte de las aplicaciones donde se emplea la Visión por Computador. Se conoce como Visión Artificial o Visión por Computador al conjunto de técnicas de adquisición, procesamiento, análisis y comprensión de las imágenes del mundo real. Su utilización a nivel mundial es extensa, desde sistemas de seguridad y validación, hasta los sistemas computacionales base de los autos autónomos. Esto ha traído como consecuencia que se empleen diversos tipos de hardware, que permitan su utilización en ambientes reales. Un ejemplo notable de este tipo de hardware, es el Computador de Placa Reducida (*SBC - Single-Board Computer*), el cual está diseñado como un microprocesador con la memoria *RAM*, las entradas/salida en una sola placa base de tamaño pequeño (i.e. menor al tamaño de un teléfono celular). En este trabajo, se profundiza en la configuración y utilización del *Raspberry PI* como un *SBC* para ejecutar algoritmos de Visión por Computador. Los algoritmos de Visión por Computador se ejecutan empleando la biblioteca *OpenCV*, que permite reconocer, detectar, reconstruir y restaurar imágenes y vídeos Así, se muestra un proceso completo de instalación y ejecución de *OpenCV* en la *Raspberry* tal que simplifique su utilización. *Python* fue el lenguaje de programación seleccionado para nuestras pruebas, donde mostramos el impacto del rendimiento de algoritmos de visión por computador en la *Raspberry Pi*.**

***Palabras Claves*—Visión por Computador, OpenCV, Raspberry Pi, Python, Imágenes**

I. Introducción

El área de Visión por Computador contempla el conjunto de técnicas que son empleadas para principalmente comprender las imágenes capturadas, por algún dispositivo de adquisición. Para comprender las imágenes (o vídeo), se requiere además de un conjunto de herramientas del campo de la estadística, geometría, física, entre otras disciplinas. Existe una relación de cooperación entre la Visión por Computador, la Visión de Máquina y el Procesamiento de Imágenes o Señales. Actualmente, existen diversas investigaciones multidisciplinarias en diversos centros de investigación del mundo, donde la Visión por Computador es considera un pilar fundamental para las nuevas tecnologías.

Como parte de estas investigaciones, se busca encontrar el hardware acorde a la captura de las imágenes o vídeo para su procesamiento. Así, la adquisición de estos datos se obtiene por diversos medios: desde una cámara fotográfica, hasta un tomógrafo computarizado. Adicional al dispositivo de captura se requiere de un computador que procese dichos algoritmos o un ente que procese la información para obtener la salida deseada. Es común, tener el dispositivo de captura junto con el dispositivo de procesamiento (e.g. tablet con cámara, teléfono con cámara, laptop). Sin embargo, en muchas oportunidades resulta práctico tener un dispositivo que se ajuste muy bien al ambiente real de procesamiento.

En aplicaciones del mundo real, no resulta sencillo aspectos como la alimentación de energía del dispositivo de captura/procesamiento, el espacio que éste ocupa, o el aspecto visual que pueda tener. Así, resulta ideal contar con un dispositivo de tamaño "pequeño" (de acuerdo a las dimensiones del problema a tratar), con bajo consumo de energía, y que sea portable para diversos ámbitos de trabajo. Un ejemplo notable de este dispositivo es el Raspberry PI, el cual es un computador de placa reducida (SBC por sus siglas en inglés) de bajo costo, el cual puede ser utilizado para sistemas de seguridad, ambientes industriales, investigación, y sistemas autónomos.

En este artículo, presentamos una serie de algoritmos de Visión por Computador empleando OpenCV que son ejecutados en el Raspberry PI como dispositivo de captura y de procesamiento, con el objeto de demostrar su uso efectivo. Así, en este documento la información se presenta como sigue: en el capítulo II se presenta los conceptos asociados al *SBC*, particularmente se indaga en el *Raspberry Pi*. El capítulo III muestra una breve introducción al concepto y técnicas de Visión por Computador. Luego, el capítulo IV muestra la configuración necesaria de la biblioteca *OpenCV* para ser empleada en el *Raspberry Pi*. En el capítulo V se muestra la implementación de algunos algoritmos para demostrar las capacidades del *Raspberry Pi*, el capítulo VI contiene un conjunto de pruebas realizadas con las técnicas implementas, así como observaciones en base a los resultados obtenidos. Por último, se presentan un grupo de conclusiones y consideraciones en el capítulo VII.

II. Computador de Placa Reducida - SBC

Una computadora de placa reducida o *SBC* (*Single-Board Computer*), por sus siglas en inglés, es un computador completo, construido en una única placa de circuitería la cual contiene microprocesadores, memoria, dispositivos de entrada/salida y cualquier otro componente requerido para

ser un computador completamente funcional. A diferencia de un computador personal convencional, generalmente una *SBC* no depende o posee puertos de expansión para agregar funcionalidades periféricas o permitir la expansión de las capacidades del hardware [1].

El primer hardware con las características de un *SBC*, fue presentado en 1976 bajo el nombre "dyna-micro" el cual estaba basado en el *Intel C8080A*. Este incluía la primera *EPROM* desarrollada por *Intel*, la "dyna-micro" fue renombrada y popularizada por *E&L Instruments of Derby* en 1976 bajo el nombre de "MMD-1"(ver la Figura 1). Desde entonces las *SBC* han evolucionado a medida que la tecnología de circuitos integrados y microprocesadores evoluciona, incrementando su poder de cómputo y disminuyendo considerablemente su tamaño.

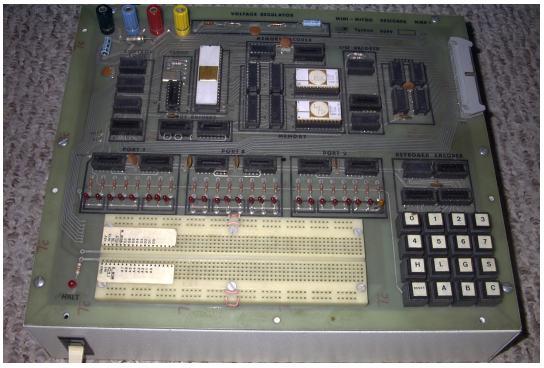

Figura 1. Mini-Micro Designer 1 (MMD-1) Creada En 1976.

Las *SBC* son comúnmente utilizadas en demostraciones o prototipos, como herramienta con fines educacionales, o como un sistema embebido, funcionando como unidades que proveen control y sirven de interfaz en sistemas más complejos como cadenas de producción robótica. Estas resultan preferidas versus un computador multi tarjetas, comparable en cuanto a capacidad de cómputo, radica en el hecho que las *SBC* son normalmente más compactas, livianas, robustas, confiables y de bajo consumo eléctrico, que sus contrapartes.

Con el paso del tiempo el mercado de *SBC* ha crecido considerablemente, permitiendo la aparición de una gran variedad de compañías y dispositivos, en los cuales varían las capacidades de hardware (i.e. principalmente el procesador y cantidad de memoria *RAM*) y su costo. En el este artículo, la experimentación se efectúa con un hardware *Raspberry Pi modelo 3 B* la cual fue diseñada por *Raspberry Pi Foundation* [2]. La Figura 2 muestra un ejemplo del aspecto visual del dispositivo de 12.19 cm de largo $\times$ 7.62 cm de ancho $\times$ 3.30 cm de altura. Las características de este modelo, que reemplazó a la versión 2 en febrero del 2016, son las siguientes:

- CPU Quad-Core ARMv8 a 1.2GHz 64-bit.
- Chip de procesamiento gráfico 3D VideoCore IV.

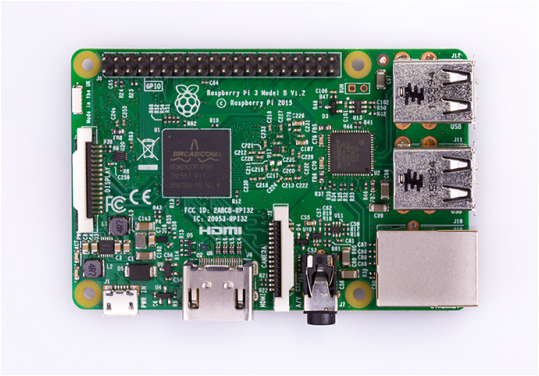

Figura 2. Imagen del Raspberry Pi 3 Versión B [2].

- RAM de 1GB.
- 40 GPIO pins (Puerto de propósito general de entrada y salida).
- Puerto Ethernet de 100 Mb.
- 802.11n Wireless LAN.
- Bluetooth versión 4.1 con Bluetooth Low Energy (BLE).
- 1 puerto HDMI, 4 puertos USB.
- 1 puerto combinado de audio 3.5mm y vídeo.
- 1 interfaz para cámara (CSI).
- 1 interfaz para despliegue (DSI).
- 1 ranura para MicroSD.

Por otra parte, el módulo de cámara utilizado (Raspberry Pi Camera) se puede apreciar en la Figura 3 y posee las siguientes características [3]:

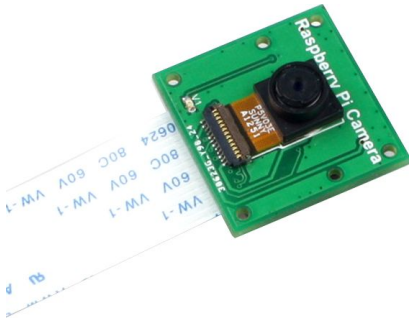

Figura 3. OV5647 Módulo de Cámara para el Raspberry Pi[3]

- Sensor de 5 mega píxeles con el sensor Om5 Omnivision OV5647 en un lente focal fijo.
- Filtro infrarrojo integrado.
- Ángulo de visión: 54° $\times$ 41°.
- Campo de visión: 2.0 m $\times$ 1.33 m a 2 m.
- Enfoque fijo: 1 m al infinito.
- Resolución máxima de captura de imágenes estáticas: 2592 px. $\times$ 1944 px.
- Resolución máxima de vídeo: 1080p.
- Máxima captura por segundo(FPS) : 30 fps.
- Tamaño: 25 mm $\times$ 24 mm.

- Cable de cinta plana de 15 cm a puerto serial de cámara de 15 pines.

Conocer las características del hardware resulta importante debido a que existen diversos SBC en el mercado. Muchos ofrecen diversas características enfocándose en aspectos esenciales como número de instrucciones que pueden procesar por instante de tiempo, capacidad de almacenamiento, conectividad con otros periféricos, entre otros. Uno de los aspectos que destaca al Raspberry sobre otros hardware es su bajo costo (e.g. aproximadamente $35), así como su incursión inicial en las escuelas públicas de Reino Unido [4]. Para finales del año 2016, Raspberry Pi Foundation había comercializado alrededor de 10 millones de Raspberry Pi [5]. Debido a las características mencionadas anteriormente, desde su aparición éste ha sido empleado para diversas investigaciones a nivel mundial [6], [7], [8], [9].

III. Visión por Computador

Tal como se menciona en [10], la visión por computador es la capacidad del computador para ver el mundo que la rodea, más precisamente para deducir propiedades y estructuras de su entorno a partir de un conjunto de imágenes.

La visión por computador procura emular en una máquina la forma como el ser humano percibe el mundo mediante la visión, proveyendo al computador la capacidad de identificar y/o reconocer patrones en una imagen. Así, le permite inferir algún tipo de conocimiento útil para la aplicación de interés. Por ende, un sistema de visión por computador puede contar con el siguiente conjunto de etapas:

A Adquisición de la imagen.
B Pre-procesamiento de la imagen.
C Extracción de características.
D Identificación y/o generación de conocimiento.
E Toma de acciones en base al conocimiento adquirido.

III-A. Adquisición de la imagen

La adquisición de la imagen requiere de dos componentes claves: un sensor que capture algún espectro electromagnético que sea capaz de generar una corriente eléctrica al ser estimulado; y un proceso de digitalización capaz de discretizar la señal emitida por el sensor. La generación de la imagen conlleva un muestreo continuo de la imagen en un conjunto discreto de posiciones y luego un proceso de cuantización de la muestra adquirida [10].

Los dispositivos de adquisición utilizados en visión por computador pueden variar desde un cámara digital (i.e. detección de intensidades lumínicas), sensores de profundidad, ultrasonido, láser, dispositivos de resonancias magnéticas, entre otros.

III-B. Pre-procesamiento de la imagen

Las imágenes a procesar pueden contener ruido y/o imperfecciones producto del proceso de adquisición en sí. Entonces, el pre-procesamiento conlleva aplicar un conjunto de operaciones con la intención de corregir o suavizar imperfecciones y mejorar la calidad de la imagen. Algunas de las tareas que se aplican en esta etapa son:

- Técnicas de remoción de ruido como filtros de suavizado.
- Cambio de la dimensión de la imagen para mejorar o acelerar los resultados de etapas posteriores.
- Incremento de contraste para resaltar elementos en la imagen.
- Cambios del espacio de color (e.g. RGB, HSL, CieLab).

III-C. Extracción de características

Esta etapa consiste en la extracción de un conjunto de características que faciliten la identificación de regiones o zonas de interés en la imagen procesada. Dichas características pueden ser generadas por técnicas sencillas como detección de bordes, líneas y contornos, segmentación de la imagen basada en color o intensidad (binarización), segmentación por análisis del histograma o técnicas más complejas como segmentación de la imagen por textura, por patrones o detectores de forma complejos como los histogramas de gradientes orientados (para más detalle ver [11]).

El objetivo de esta etapa es agrupar la información de un conjunto de píxeles en una estructura que actué como un descriptor complejo. Dicho descriptor puede basarse en propiedades locales como el valor de píxel con respecto a sus vecinos, la ubicación del píxel en la imagen con respecto a su entorno, o su contribución global en la imagen.

Es importante destacar que un sistema de visión por computador puede concluir en esta etapa dependiendo de la técnica que implemente.

III-D. Identificación y/o generación de conocimiento

Tomando como insumo las características extraídas en la fase previa, es posible emplear a emplearlas como mecanismos de información para identificar o detectar elementos complejos dentro de la imagen. Esto con el objetivo de que puedan servir para generar algún tipo de conocimiento y tomar alguna acción.

En esta etapa la visión por computador se convierte en un área multidisciplinaria, donde se combinan los resultados de la extracción de características (procesamiento de imágenes) con técnicas de aprendizaje automático (inteligencia artificial). Generalmente, las técnicas de aprendizaje automático se emplean para detectar o identificar objetos, dado un conjunto de características, o para aprender que dichas características describen un objeto o situación en particular.

Nuevamente, esta etapa puede ser el punto de culminación del sistema de visión por computador desarrollado.

III-E. Toma de acciones en base al conocimiento adquirido

Esta etapa puede ser tan simple como enviar una alerta o informar de la detección de algún elemento en la imagen, hasta unas más complejas como mover un brazo robótico a un área específica en base a la imagen percibida, retroalimentar una base de conocimientos para la toma de futuras decisiones, o para ajustar el proceso de extracción de características en técnicas en las que se aprenden las características a extraer para un contexto dado, como las redes convolutivas [12][13].

Tareas Comunes en Visión por Computador

Algunas de las tareas típicas de la visión por computador se pueden agrupar en las siguientes categorías:

- Reconocimiento.
- Análisis de Movimiento.
- Reconstrucción de Escenas.
- Restauración de Imágenes.

Particularmente, en el área de reconocimiento se desarrollan aplicaciones como:

- Detección de objetos.
- Identificación de rostros.
- Reconocimiento de caracteres.
- Reconocimiento de escritura y/o firmas.
- Buscadores de imágenes.
- Reconocimiento de gestos.
- Estimación de poses.

El análisis de movimiento consiste en procesar un conjunto de imágenes distribuidas en el tiempo para detectar velocidad (cambios) en puntos de la imagen. Se pueden realizar técnicas de vigilancia para detectar cambios en un vídeo que pueden estar asociados con movimiento, seguimiento de objetos y determinar velocidad en zonas de una imagen por la variación de colores.

Por su parte, la reconstrucción de escenas comprende reconstruir un objeto 3D a partir de una secuencia de imágenes que contienen distintos ángulos del objeto a reconstruir. Por otra parte, la restauración de imágenes comprende todas las técnicas requeridas para eliminar ruido o daños sobre una imagen, empleando información presente en la imagen. Un ejemplo notable es la técnica conocida como *Inpainting* [14].

IV. Configuración

En esta sección se describe el proceso técnico para configurar el *Raspberry Pi* con las herramientas mínimas necesarias para desarrollar las aplicaciones de visión por computador descritas en el capítulo V. Partiendo de la instalación del sistema operativo *Raspbian* en una tarjeta *MicroSD*, hasta la instalación de *OpenCV* y posterior configuración para ser utilizado en el lenguaje de programación *Python*.

IV-A. Instalación de Raspbian

Raspbian es un sistema operativo basado en una distribución Linux Debian, que se encuentra optimizada para el Raspberry Pi. Para su instalación. es necesario descargar su versión más reciente de la página oficial de *Raspberry*[1]. Para Marzo 2017, la ultima versión es *Raspbian Jessie* en su opción *Lite* o con el sistema de escritorio *Pixel* (para el resto de la instalación es indiferente la opción seleccionada).

Es importante conocer desde que sistema operativo inicial se realizará la instalación. Si la instalación será realizada en el SO *Windows*, los pasos a seguir son:

- Extraer la imagen de *Raspbian* del archivo .zip.
- Descargar y ejecutar en modo administrador la aplicación *Win32DiskImager*[2].
- Seleccionar la imagen de *Raspbian* previamente extraída y la unidad SD.
- Presionar la opción *Write* y esperar a que la instalación concluya.

Por el contrario, si la instalación es realizada en una distribución de *Linux* el proceso consiste en aplicar los siguientes pasos descritos con mayor detalle en [15]:

- Verificar los dispositivos montados (i.e. los dispositivos activos) con el comando:

```
$ df –h
```

- Conectar la *MicroSD* al computador y verificar nuevamente los dispositivos montados.
- Desmontar la *MicroSD* sustituyendo *nombre_de_la_unidad* por el nombre del dispositivo en el siguiente comando:

```
$ umount /dev/nombre_de_la_unidad
```

- Copiar la imagen de *Raspbian* a la *MicroSD* con el siguiente comando:

```
$ dd bs=4M if=ruta_a_la_imagen_de_raspbian truncate↩
    ––reference ruta_a_la_imagen_de_raspbian of=/↩
   dev/nombre_de_la_unidad status=progress
```

- Ejecutar el siguiente comando para asegurar que es seguro realizar la extracción de la *MicroSD* y posteriormente extraerla:

```
$ sync
```

Para este punto, la distribución se Raspbian se encuentra en la tarjeta *microSD* la cual puede ser colocada en el Raspberry Pi, y poder observar su respectiva instalación. El siguiente paso consiste en instalar *OpenCV 3* como una biblioteca para la manipulación de imágenes y vídeo.

[1]https://www.raspberrypi.org/downloads/raspbian/
[2]https://sourceforge.net/projects/win32diskimager/

IV-B. Instalando OpenCV

Se asume que el *Raspberry Pi* tiene previamente instalado el sistema operativo *Raspbian Jessie*, y se describen a continuación los pasos a realizar para la instalación de *OpenCV*.

Expandir el sistema de archivos: Es buena práctica asegurarse que el sistema operativo ocupe en su totalidad todo el espacio que se encuentre disponible en la *micro-SD*. Para ello se ejecuta en una terminal el comando:

```
$ sudo raspi-config
```

Este comando despliega un menú en el cual se debe seleccionar la opción "*1.Expand File Sytem*", posteriormente se selecciona la opción "*1<Finish>*"y se reinicia el *Raspberry* para que los cambios surtan efecto con el comando:

```
$ sudo reboot
```

Es posible que durante la instalación se requiera de más espacio del disponible si la SD es de baja capacidad (i.e. 8 GB o menos), por tanto es recomendable liberar un poco de espacio extra eliminando el *Wolfram engine* preinstalado en *Raspbian*, para ello se ejecuta el comando:

```
$ sudo apt-get purge wolfram-engine
```

Instalar dependencias: Primero se deben actualizar los paquetes existentes en el sistema ejecutando los siguientes comandos:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Luego, se debe actualizar el *Firmware* del *Raspberry Pi* y reiniciar el *Raspberry* para que surtan efecto los cambios:

```
$ sudo rpi-update
$ sudo reboot
```

Posteriormente, se instalan un conjunto de herramientas de desarrollo y *CMake* para compilar *OpenCV*:

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

Se instalan los paquetes necesarios para cargar distintos formatos de imagen:

```
$sudo apt-get install libjpeg-dev libtiff5-dev libjasper-↩
     dev libpng12-dev
```

De igual manera, se instalan los paquetes necesarios para poder manipular los vídeos y realizar la captura de vídeo directo:

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

Para poder hacer uso del sub-módulo *highgui* de *OpenCV* requerido para el despliegue de imágenes e interfaces básicas, es necesario instalar el paquete de desarrollo de *GTK*:

```
$ sudo apt-get install libgtk2.0-dev
```

Por último, se debe instalar las cabeceras de *Python* para realizar la compilación de *OpenCV* asociada con *Python* e instalar un conjunto de dependencias con el fin de optimizar cálculos que realiza *OpenCV* internamente:

```
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python3-dev
```

Descargar el código fuente de OpenCV: Los siguientes comandos tienen como propósito descargar la versión *3.1.0* de *OpenCV* y extraer su contenido mismo en el directorio *home* del sistema:

```
$ cd ~
$ wget -O opencv.zip https://github.com/Itseez/opencv/↩
    archive/3.1.0.zip
$ unzip opencv.zip
```

De igual manera, se debe decargar un conjunto de extensiones de *OpenCV* para tener acceso a mayor cantidad de funcionalidades provistas por la comunidad:

```
$ wget -O opencv_contrib.zip https://github.com/Itseez/↩
    opencv_contrib/archive/3.1.0.zip
$ unzip opencv_contrib.zip
```

Configurar Python 3: Previo a instalar *OpenCV* se instalará *pip* el cual es un manejador de paquetes de *Python*, empleado para instalar distintos paquetes que se requieren para el desarrollo en *Python*:

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
```

Una vez instalado *pip*, se instala la dependencia *NumPy*, la cual es necesaria para ciertos cálculos numéricos:

```
$ pip install numpy
```

IV-B1. Compilando e instalando OpenCV: Inicialmente se realiza el proceso de configuración inicial:

```
$ cd ~/opencv-3.1.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D INSTALL_PYTHON_EXAMPLES=ON \
  -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.1.0/↩
      modules \
  -D BUILD_EXAMPLES=ON ..
```

Es importante verificar que el resultado del *CMAKE* contenga en su sección de *Python 3* los valores de *Interpreter* y de *numpy*. Estos deben coincidir con los directorios de instalación de *Python 3* y de *NumPy* respectivamente, una vez verificado que todo este bien se ejecuta la compilación con el comando:

```
$ make -j4
```

Si la compilación culminó sin erroresm se ejecutan los siguientes comandos para realizar la instalación de *OpenCV*:

```
$ sudo make install
$ sudo ldconfig
```

Probando la instalación de OpenCV: A fin de verificar que la instalación de *OpenCV* fue correctamente realizada, se pueden ejecutar los siguientes comandos en consola, los cuales debería arrojar como resultado la versión de *OpenCV* instalada en el sistema:

```
$ python
>>> import cv2
>>> cv2.__version__
```

Posterior a esto es recomendable liberar el espacio extra requerido durante la instalación eliminando los directorios utilizados para compilar *OpenCV*:

```
$ cd ~
$ rm -rf opencv-3.1.0 opencv_contrib-3.1.0
```

Una vez verificado la instalación correcta de *OpenCV*, es posible escribir algoritmos de visión por computador para ser ejecutados en la Raspberry Pi.

V. Visión por Computador en el Raspberry Pi

En esta sección se detallan los algoritmos de visión por computador implementados y probados en el *Raspberry Pi*. Todos los algoritmos implementados poseen la misma estructura, compuesta de un conjunto de código encargado de la configuración de parámetros de entrada, configuración de la cámara del *Raspberry Pi*, y la invocación a una función denominada $processFrame$ la cual es la implementa la técnica de visión por computador a desarrollar.

En el algoritmo 1, se puede observar un ejemplo de la estructura básica del bloque de configuración.

```
1  # Bloque de argumentos que recibe el sistema
2  ap = argparse.ArgumentParser()
3  ap.add_argument("-c", "--capturescreenshots", help="Bool que indica si se desea capturar screenshots", type=bool, default=False)
4  ap.add_argument("-s", "--numberscreenshots", help="Numero de screenshots que deben capturarse", type=int, default=5)
5  ap.add_argument("-w", "--warmupscreenshots", help="Numero de segundos que se espera antes de capturar el primer screenshot", type=int, default=10)
6  ap.add_argument("-i", "--intervalscreenshots", help="Intervalo en segundos en el cual se toman los screenshots", type=int, default=1)
7  args = vars(ap.parse_args())
8
9  # Configuración de la cámara
10 camera = PiCamera()
11 camera.resolution = (640, 480)
12 camera.framerate = 25
13 rawCapture = PiRGBArray(camera, size = (640, 480))
14 time.sleep(1)
15
16 # Se inicia la captura de vídeo por la cámara
17 for f in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
18   frame = f.array
19   # Se procesa la imagen capturada
20   (frame) = processFrame(frame)
21
22   # Se visualiza la imagen final con la deteccion de caras
23   cv2.imshow("Video actual", frame)
24
25   # Se permite la captura del siguiente frame
26   rawCapture.truncate(0)
27
28   # Se captura una tecla del teclado
29   key = cv2.waitKey(1) & 0xFF
30
31   # En caso de que se presionase la tecla "q", se interrumpe la captura de vídeo
32   if key == ord("q"):
33     break
```

Algoritmo 1. Algoritmo Base

Las líneas 2-7 corresponden con la especificación de los argumentos de entrada de la aplicación. En las líneas 10-14 se configura la cámara del *Raspberry Pi*, se crea el objeto correspondiente a la cámara, se inicializa la resolución de captura, los frames por segundo de captura y se espera un tiempo para permitir a la cámara encenderse para prepararse para la captura.

Continuando, la línea 17 inicia el proceso de captura continua de imágenes por la cámara (vídeo), la línea 18 realiza la captura de un frame, la línea 20 invoca a la función que procesa el frame e implementa la técnica de visión por computador. La línea 23 despliega el frame procesado por la técnica de visión por computador, dependiendo de la técnica aplicada puede que se despliegue más de una imagen para demostrar todas las etapas de procesamiento de un frame. La línea 26, permite a la cámara realizar una nueva captura de imagen y las tres líneas restantes detienen el proceso de captura si se presiona la tecla “q”en el teclado.

A continuación, se detalla cada una de las técnicas desarrolladas que corresponde con el proceso implementado dentro de la función $processFrame$.

V-A. *Chroma Key*

El algoritmo implementado selecciona el color dominante en la imagen analizada y lo sustituye por los colores en una imagen dada. Se asume que en la imagen suministrada el fondo de la misma estará compuesto por un color uniforme, de manera que el efecto generado es sustituir el fondo de la imagen original por un fondo preestablecido.

Figura 4. Resultados del Algoritmo Chroma Key, (a) Imagen original Capturada, (b) Máscara Generada para Descartar los Colores Dominantes, (c) Composición luego de Reemplazar el Nuevo Fondo en la Imagen Original.

El algoritmo 2 implementa la función $processFrame$ para aplicar el proceso de sustitución del fondo, la misma recibe la imagen a procesar ($frame$) y el fondo con el cual realizará la sustitución ($background$). Inicialmente se transforma la imagen al espacio de colores *HSV*, en este esquema de color, el croma o *Hue* es un canal especifico y el de interés para este caso, el brillo y la saturación son canales aparte y de menor importancia (ver líneas 2-5). Utilizando el croma (*hue*) se crea un histograma para identificar el color dominante en la imagen que debería corresponder con el fondo (línea 7), posteriormente se ordena el histograma y se toma el valor que más se repita en el mismo, como el color dominante (líneas 9-11).

Luego, se selecciona un pequeño rango de colores alrededor del color dominante (líneas 12-18). Con el rango de cromas seleccionado se crea un rango de colores en el modelo HSV donde se varia la saturación desde un valor mínimo, para obtener los colores más grises y opacos de los cromas seleccionados hasta obtener los colores más brillantes y puros del rango de cromas (líneas 21-22). El rango seleccionado se utiliza para crear una máscara sobre la imagen en el esquema de colores *HSV* (línea 25), con dicha máscara se seleccionan los píxeles con colores dominantes en la imagen original y se sustituyen con el fondo provisto (línea 27), posteriormente la máscara se invierte y se pueden seleccionar los píxeles que no tienen colores dominantes (líneas 29-31).

Las dos imágenes resultantes del uso de las máscaras son mezcladas para generar la imagen final (línea 33). Finalmente, la función retorna la composición final y la máscara utilizada para seleccionar los píxeles con colores no dominantes (ver Figura 4).

```
1  def processFrame(frame, background):
2    # Se transforma la imagen a el espacio de color HSV
3    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
4    # Se toma solo el canal del Hue o Color
5    (h, _, _) = cv2.split(hsv)
6    # Se calcula el histograma para el Hue que tiene 180↩
         valores
7    (hueHist, _) = np.histogram(np.ravel(h) ↩
         ,180,[0,180])
8    # Se ordena los valores del histograma
9    sortedHistIndex = np.argsort(hueHist)
10   # Se toma el color dominante
11   hightValue = sortedHistIndex[179]
12   # Se toma un pequeño rango de colores alrededor del ↩
         color dominante
13   minValue = hightValue - 5
14   maxValue = hightValue + 5
15   if minValue < 0:
16     minValue = 0
17   if maxValue > 179:
18     maxValue = 179
19
20   # rango de colores a enmascarar
21   lowerValue = np.array([minValue,50,50])
22   upperValue = np.array([maxValue,255,255])
23
24   # Se crea la máscara
25   mask = cv2.inRange(hsv, lowerValue, upperValue)
26   # Se selecciona los pixeles que corresponden al ↩
         fondo
27   maskBackground = cv2.bitwise_and(background, ↩
         background, mask=mask)
28   # Se invierte la mascara
29   mask = cv2.bitwise_not(mask)
30   # Se seleccionan los pixeles que no corresponden al ↩
         fondo
31   maskObject = cv2.bitwise_and(frame, frame, mask=mask↩
         )
32   # Se mezclan las dos imagenes generadas
33   res = cv2.add(maskBackground, maskObject)
34
35   return (mask, res)
```

Algoritmo 2. Chroma Key

V-B. Detección de Movimiento

Para detectar movimiento en una escena es necesario conocer el fondo o zonas estáticas de la escena observada. Posterior a identificar el fondo, un movimiento corresponderá con alguna variación de color o intensidad en la imagen que fue preestablecida como fondo. En el algoritmo 3 se muestra una implementación simple de detección de movimiento, la función $processFrame$ recibe como parámetros la imagen sobre la cual se detectará movimiento ($frame$) y el fondo de la escena ($backGround$), y genera como resultado una imagen con las zonas en las que ocurrió movimiento enmarcadas.

Figura 5. Resultados del Algoritmo de Detección de Movimiento, (a) Imagen Original con las Zonas donde hubo Movimiento Remarcadas, (b) Diferencia entre el Fondo (zonas estáticas) y las Zonas con Movimiento, (c) Máscara Generada en Base a la Diferencia entre Zonas Estáticas y en Movimiento.

Inicialmente, se toma la imagen sobre la cual se detectará movimiento y se aplica un pre procesamiento que consiste en reducir su dimensión para disminuir el cómputo. Así, se transforma a escala de grises y se aplica un filtro *gaussiano* para disminuir los cambios bruscos que puedan detectarse como movimiento y que correspondan con ruido generado por el dispositivo de captura (ver línea 1-8 del algoritmo 3). Si la imagen a procesar es la primera, la misma se asumirá como el fondo de la escena a grabar (línea 11-12).

Posteriormente se acumula parte de la imagen tomada, al fondo que se tiene calculado, esto tiene como objetivo permitir que el fondo se adapte a los cambios de iluminación que puedan surgir en la escena producto del paso del tiempo o adaptarse a cambios como aparición o remoción de objetos en el fondo (línea 15). Con el fondo actualizado se calcula la diferencia que exista entre el fondo y la imagen a la cual se detectará el movimiento. Entonces, se genera una máscara con los cambios detectados y la misma se dilata para rellenar posibles agujeros (líneas 17-20). Sobre la máscara se calcula los contornos de la misma (línea 23), con la intención de segmentar las zonas de la imagen que estén separadas (distintas zonas donde hubo movimiento y/o variación). Luego, se evalúa cada contorno generado y si su área es lo suficientemente significativa se calcula su caja recubridora o *bounding box* (líneas 26-31).

Como resultado de la segmentación pueden existir zonas donde se detecte movimiento donde se encuentren englobadas por zonas mas grandes (máximos locales), para eliminar dichas zonas se aplica un proceso de supresión de máximos (líneas 34-35). Las zonas que sobreviven al proceso de supresión de máximos, son dibujadas en la imagen final (líneas 38-39). Finalmente, la función retorna la imagen con la detección enmarcada ($frame$), la diferencia calculada entre el fondo y la imagen sobre la que se calcula movimiento ($frameDelta$), la máscara generada con dichas diferencias ($thresh$) y el fondo actualizado ($backGround$) (como se muestra en la Figura 5).

La técnica descrita para diferenciar el fondo de los objetos en movimiento es sencilla de implementar y muy propensa a errores, existen técnicas mucho más complejas implementadas en *OpenCV*, basándose en trabajos como [16][17]. Sin embargo, éstas requieren de mayor poder de cómputo que comprometen el rendimiento general de la aplicación.

```
1  def processFrame(frame, backGround):
2    # Se redimensiona el frame para disminuir el procesamiento
3    frame = imutils.resize(frame, width=min(400, frame.shape[1]))
4
5    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
6
7    # Se reduce ruido
8    gray = cv2.GaussianBlur(gray, (21, 21), 0)
9
10   # Se define la primera imagen como fondo
11   if backGround is None:
12     backGround = gray.copy().astype("float")
13
14   # Se acumula la nueva imagen en el fondo
15   cv2.accumulateWeighted(gray, backGround, 0.05)
16   # Se calcula la diferencia entre el fondo y la nueva imagen capturada
17   frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(backGround))
18   # Se crea una máscara con las diferencias y se rellenan los huecos
19   thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]
20   thresh = cv2.dilate(thresh, None, iterations=3)
21
22   # Se calculan los contornos en la máscara
23   (_, cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
24
25   # Se calculan los bounding box de las siluetas si son lo suficientemente grandes
26   rects = []
27   for c in cnts:
28     if cv2.contourArea(c) < args["min_area"]:
29       continue
30
31     rects.append(cv2.boundingRect(c))
32
33   # Se eliminan los maximos locales o falsas detecciones
34   rects = np.array([[x, y, x + w, y + h] for(x, y, w, h) in rects])
35   pick = non_max_suppression(rects, probs=None, overlapThresh=0.65)
```

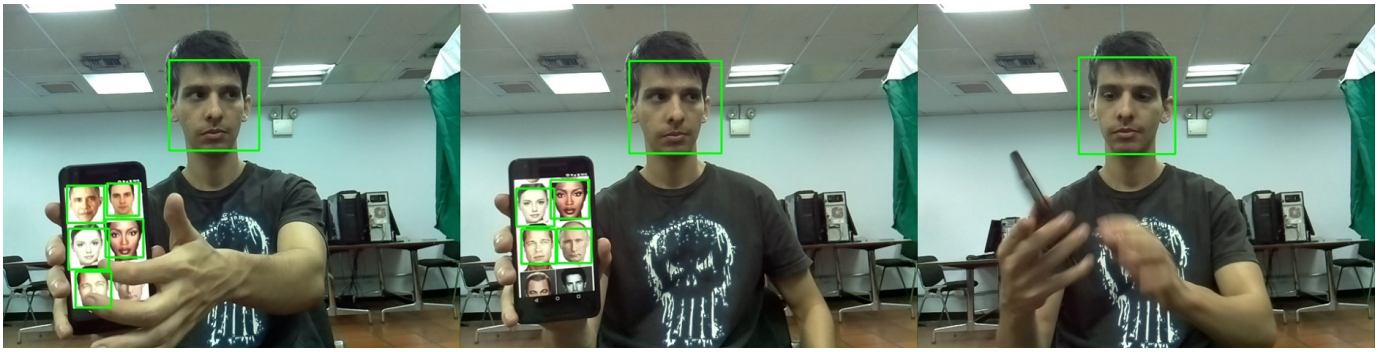

Figura 6. Muestra de los Resultados del Algoritmo de Detección de Caras.

```
36
37    # Se enmarcan las zonas con movimiento
38    for (xMin, yMin, xMax, yMax) in pick:
39      cv2.rectangle(frame, (xMin, yMin), (xMax, yMax), ↩
            (0, 0, 255), 2)
40
41    return (frame, frameDelta, thresh, backGround)
```

Algoritmo 3. Detección de movimiento con fondo adaptativo

V-C. *Detección de Caras*

La idea de este algoritmo es detectar todas las caras que se encuentren presentes en una imagen con la restricción de que las caras a detectar deben estar en posición frontal (viendo a la cámara y lo menos inclinadas posibles). Para esto se hace uso del clasificador *Haar-Cascade* propuesto inicialmente en [18] y extendido por [19], cuya función es identificar si una imagen corresponde con una cara o no, mediante la extracción de un conjunto de características.

Las características extraídas corresponden a aplicar un grupo de máscaras de convolución diseñadas para operaciones como identificar líneas horizontales, verticales y diagonales. El clasificador es entrenado con un conjunto de imágenes separadas en un grupo con caras y uno sin caras, se aplican los extractores de características, y aquellas características que sirvan para diferenciar una imagen con una cara de una sin caras, son almacenadas, los parámetros de dichas máscaras son mejorados gradualmente con la intención de mejorar la detección.

El problema con este proceso es que para una imagen de 24×24 se puede obtener hasta 6000 características para identificar si es o no un rostro, debido a esto se aplica el proceso de cascada. Este proceso consiste en aplicar clasificadores más sencillos y menos precisos primero para descartar prontamente zonas donde no hay caras, y a medida que se baja en nivel de la cascada, la cantidad de características a evaluar son mayores y la precisión del clasificador es mejor.

OpenCV cuenta con una implementación del clasificador *Haar-Cascade* para la identificación de caras ya implementado y entrenado, por ende para utilizarlo solo es necesario utilizar el archivo XML que cuenta con la configuración del clasificador ya entrenado como se puede observa en el algoritmo 4.

```
1  faceCascade = cv2.CascadeClassifier("↩
       haarcascade_frontalface_default.xml")
```

Algoritmo 4. Carga del clasificador Harr-Cascade para detección de caras frontales

En el algoritmo 5, la función $processFrame$ recibe la imagen donde se detectarán posibles caras ($frame$) y el clasificador de *Haar-Cascade* previamente configurado. Primero se transforma la imagen en escala de grises (línea 3) ya que el clasificador trabaja con imágenes en escala de grises. Luego, se aplica el clasificador a toda la imagen empleando una ventana deslizante sobre ésta, aplicando el procesamiento en distintas resoluciones de la imagen original debido a que dependiendo de la cercanía de la cara a la cámara la misma puede ocupar más o menos píxeles que los evaluados por la ventana (línea 5). Por último, se enmarcan en la imagen original las zonas donde se detectaron las posibles caras (ver Figura 6).

```
1  def processFrame(frame, detector):
2    # Se transforma la imagen capturada a escala de ↩
         grises
3    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
4    # Se detectan las caras en la imagen aplicando una ↩
         ventana deslizante en multiples resoluciones
5    faces = detector.detectMultiScale(gray, scaleFactor↩
         =1.1, minNeighbors=5, minSize=(30, 30))
6    # Se enmarca en un recuadro las areas en las que se ↩
         detecta una cara
7    for (x, y, w, h) in faces:
8      cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, ↩
           0), 2)
9    return frame
```

Algoritmo 5. Detección de caras frontales con Harr-Cascade

VI. Experimentación

A fin de verificar la efectividad del *Raspberry Pi* como herramienta para el desarrollo de sistemas de visión por computador, se realizaron un conjunto de pruebas de los algoritmos explicados en la capítulo V. Sus resultados se encuentra expresados en la Tabla I. Nótese, que todas las pruebas fueron realizadas en el *Raspberry Pi 3 B* utilizando como dispositivo de captura el módulo de cámara, la cual fue configurada para que realizara una captura continua a una

Tabla I
RESULTADOS DE LA EVALUACIÓN DE LOS DISTINTOS ALGORITMOS IMPLEMENTADOS EN EL *Raspberry Pi*

Algoritmo	**Captura secuencial**		**Captura paralela**	
	Sin despliegue (FPS)	**Con despliegue (FPS)**	**Sin despliegue (FPS)**	**Con despliegue (FPS)**
Solo captura desde la cámara	30.28	15.16	201.22	50.76
Chroma Key	7.67	6.81	12.86	11.83
Detección de Movimiento	7.68	7.60	15.73	13.79
Detección de Caras	5.31	4.97	7.66	7.16

resolución de 320 $\times$ 240 píxeles y a una velocidad de captura de 32 frames por segundo. Las pruebas fueron divididas en pruebas sin despliegue de la imagen procesada y con el despliegue de la imagen procesada con la intención de poder ver el efecto generado por dibujar la imagen a procesar.

La forma como se realiza la captura de la imagen y su posterior procesamiento, fue tratado de dos formas distintas. En un caso se realizó un procesamiento secuencial donde se solicita el siguiente frame a la imagen mediante una instrucción bloqueante, posterior a su adquisición se procesa el frame y se solicita el siguiente frame. Para el segundo caso, se trata la captura de la imagen en un hilo separado al de procesamiento del frame. De esta forma, se puede procesar más frames por segundo al eliminar el cuello de botella presente por la instrucción de entrada y salida correspondiente a solicitar el siguiente frame a la cámara.

Analizando los resultados presentes en la Tabla I, se puede observar que la primera prueba realizada no posee ningún procesamiento del frame capturado, esta prueba fue diseñada para tener un punto de referencia y conocer la velocidad máxima de procesamiento del *Raspberry Pi* y de su módulo de cámara.

Se puede observar que sin desplegar la imagen adquirida, los frames por segundo son cercanos a la velocidad de captura de la configuración inicial de la cámara, mostrando que el despliegue de una imagen deteriora el rendimiento en aproximadamente 50 %. Por otra parte, los resultados de realizar el proceso de captura en paralelo presentan una significativa mejora en la cantidad de frames por segundo procesados.

El comportamiento observado en las pruebas de los algoritmos restantes es el esperado, donde la técnica que requiere de mayor cómputo (Detección de Caras) posee el rendimiento más bajo de las 3 técnicas implementadas. A lo largo de todas las pruebas, se puede observar que el hecho de desplegar la imagen procesada conlleva una pérdida de frames por segundo que puede llegar a ser significativa como lo evidencia la prueba uno, donde el rendimiento fue un 50 % menor para el caso de la captura secuencial. Igualmente, se puede deducir que realizar la captura de las imágenes de manera paralela incrementa sustancialmente el rendimiento general de la imagen teniendo una mejora en la tasa de frames por segundo que oscila entre el 664 % y el 144 %. Para mayor información sobre la captura en paralelo referirse a [20].

VII. DISCUSIÓN

En esta investigación, se desarrollaron un conjunto de técnicas que competen al área de visión por computador en un *Raspberry Pi 3* con la intención de evaluar sus capacidades como herramienta en dicha área de trabajo. Como se observó en las pruebas expuestas en el capítulo VI, los resultados obtenidos no son adecuados para aplicaciones en tiempo real, sin embargo los resultados obtenidos para las técnicas que requieren menor costo computacional son aceptables.

Por ende se considera que el *Raspberry Pi 3* puede ser empleado en las etapas prematuras de los sistemas de visión por computador: la etapa de adquisición y pre procesamiento de la imagen. Mientras que las etapas siguientes, que son más complejas e intensivas computacionalmente, pueden ser delegadas a hardware de mayor capacidad de cómputo mediante una arquitectura cliente servidor.

Igualmente, se considera importante realizar el mismo estudio utilizando otros lenguajes de programación como C o C++, que permitan tener un mejor control sobre el hardware. Así, se procura explotar al máximo las características del hardware y posiblemente alcanzar mejor rendimiento de la aplicación en general.

REFERENCIAS

[1] C. Ortmeyer, "Then and now, a brief history of single board computers," *Electronic design uncovered*, vol. 6, 2014.

[2] Raspberry Pi Foundation, "Raspberry PI 3 Model B," [Accedido: 26-03-2017]. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[3] Lee Jackson, "New Low Cost OV5647 Mini Camera Module for Raspberry Pi Now Available." [Online]. Available: http://www.arducam.com/lowcost-raspberry-pi-mini-camera-module/

[4] J. Moorhead, "Raspberry Pi device will 'reboot computing in schools'," 2012. [Online]. Available: https://goo.gl/tRt5t9

[5] Raspberry Pi Foundation, "Ten Millionth Raspberry Pi, and a New Kit," 2016. [Online]. Available: https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/

[6] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures," in *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, July 2013, pp. 108–112.

[7] S. Ferdoush and X. Li, "Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications," *Procedia Computer Science*, vol. 34, pp. 103 – 110, 2014.

[8] V. Vujović and M. Maksimović, "Raspberry pi as a sensor web node for home automation," *Computers Electrical Engineering*, vol. 44, pp. 153 – 171, 2015.

[9] X. Q. Li, X. Ding, Y. Zhang, Z. P. Sun, and H. W. Zhao, "Iot family robot based on raspberry pi," in *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, June 2016, pp. 622–625.
[10] J. M. d. l. C. Gonzalo Pajares, *Visión por Computador*. Ra-Ma, 2001.
[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *In CVPR*, 2005, pp. 886–893.
[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.
[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, p. 2012.
[14] M. Bertalmio and G. Sapiro, "Image inpainting," 2000, pp. 417–424.
[15] Raspberry Pi Foundation, "Installing Operating System Images on Linux," [Accedido: 26-03-2017]. [Online]. Available: https://www.raspberrypi.org/documentation/installation/installing-images/linux.md
[16] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," 2004.
[17] P. Kaewtrakulpong and R. Bowden, "An improved adaptive background mixture model for realtime tracking with shadow detection," 2001.
[18] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," 2001.
[19] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid objection detection," *IEEE ICIP*, p. 2002.
[20] Adrian Rosebrock, "Increasing Raspberry Pi FPS with Python and OpenCV." [Online]. Available: http://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/