Universidad Zaragoza
1542

Trabajo Fin de Grado

Control automático de semáforos basado en procesado digital de imágenes sobre Raspberry Pi

Automatic control of traffic lights based on digital image processing on Raspberry Pi

Autor/es

Zhuolin JIN

Director/es

Francisco José TORCAL-MILLA
Ana LOPEZ-TORRES

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Año 2019

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe entregarse en la Secretaría de la EINA, dentro del plazo de depósito del TFG/TFM para su evaluación)

D./Dª. Zhuolin Jin, con NIE X5759224R, en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster)

(Título del Trabajo)

Control automático de semáforos basados en procesado digital de imágenes sobre Raspberry Pi

Automatic control of traffic lights based on digital image processing on Raspberry Pi

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 20 de Septiembre de 2019

Fdo:

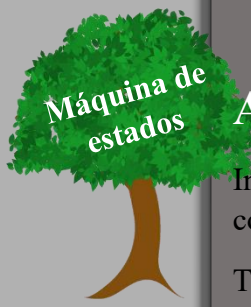
Control automático de semáforos basado en procesado digital de imágenes sobre Raspberry Pi

Resumen

En este trabajo se muestra el desarrollo de un dispositivo compacto y barato para el control activo de un semáforo utilizando como placa de control la Raspberry Pi.

El dispositivo se basa en algoritmos de visión por computador para determinar cuántas personas hay en cada momento esperando para cruzar y el tiempo que llevan esperando.

Posteriormente utiliza una máquina de estados para determinar si debe abrirse o no el semáforo y cuánto tiempo debe permanecer abierto. En este aspecto, también es sensible a la velocidad de avance de los peatones, manteniendo el semáforo abierto más tiempo si es necesario, dentro de un cierto margen máximo.



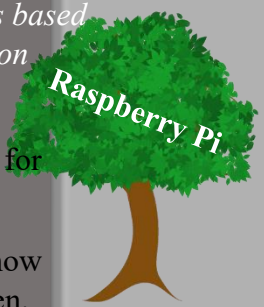
Automatic control of traffic lights based on digital image processing on Raspberry Pi

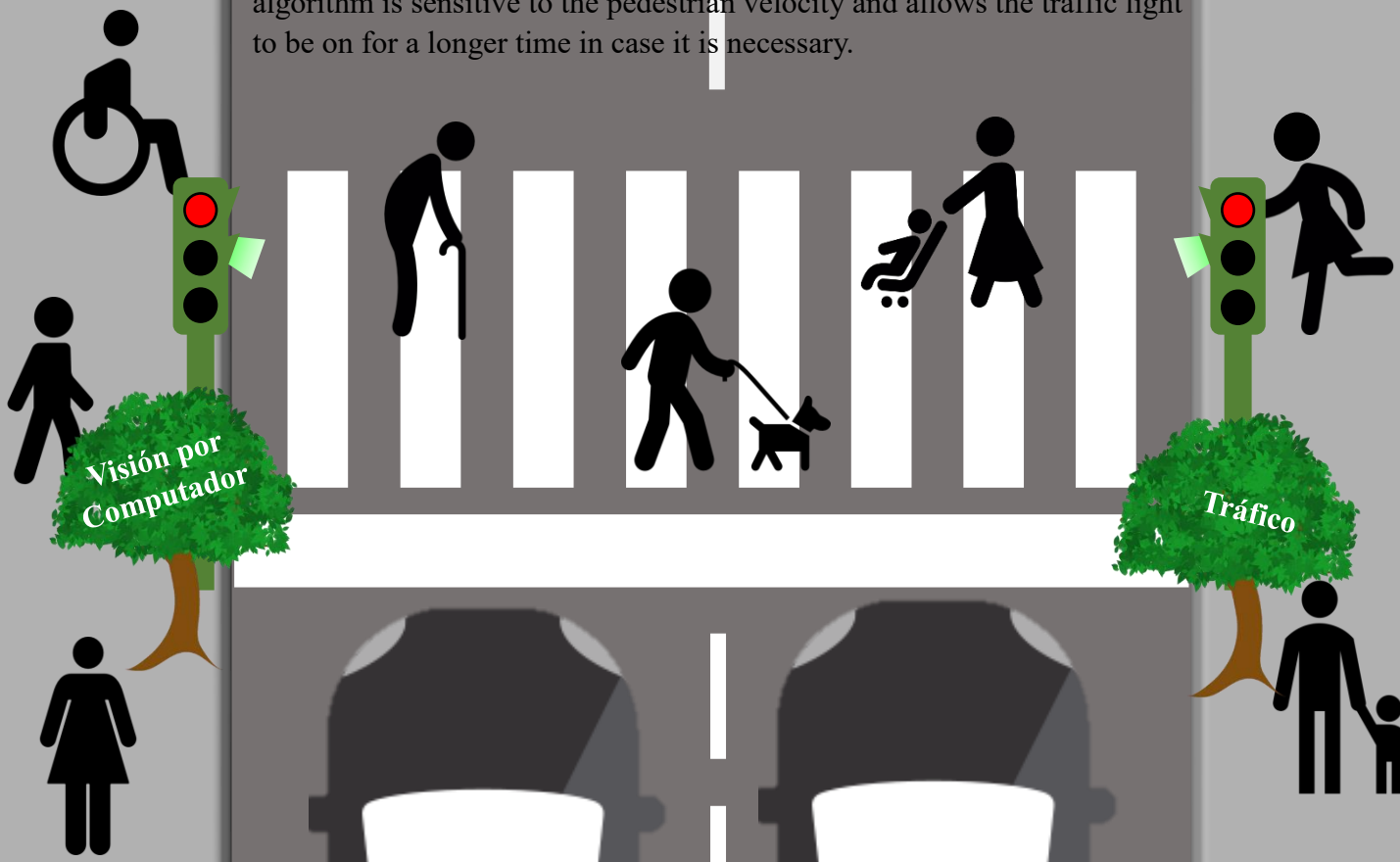
Abstract

In this work, we show the development of a low cost portable device for controlling a traffic light using a Raspberry Pi board.

The algorithm is based on computer vision and is able to determine how many people are waiting to cross the street and how long they have been.

Then it uses a state machine to determine whether the traffic light should be on or off and how long it should remain on. Related to this aspect, the algorithm is sensitive to the pedestrian velocity and allows the traffic light to be on for a longer time in case it is necessary.

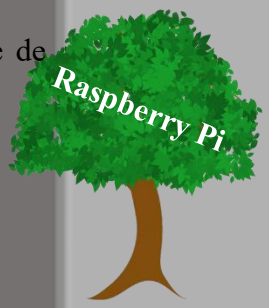


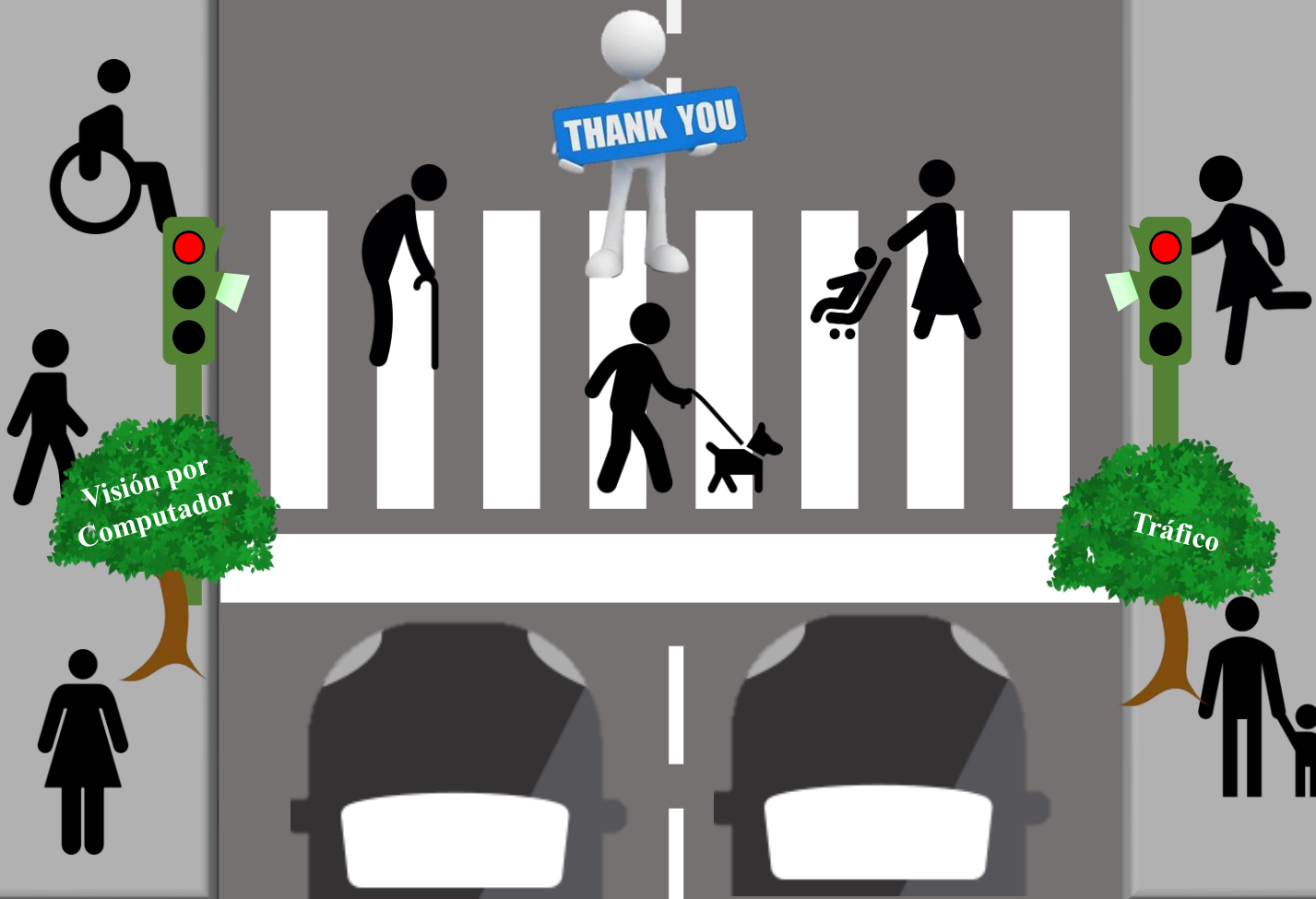


Agradecimientos

Aprovecho este espacio para dar las gracias a

- Cátedra Mobility City de la Universidad de Zaragoza, por sus apoyos y apuestas en el campo de la movilidad urbana sostenible.
- Los profesores, Francisco José Torcal Milla y Ana López Torres, por aceptarme para realizar este proyecto, y sobre todo por sus apoyos y guías.
- Los amigos y compañeros, por sus participaciones en la fase de prueba del sistema.

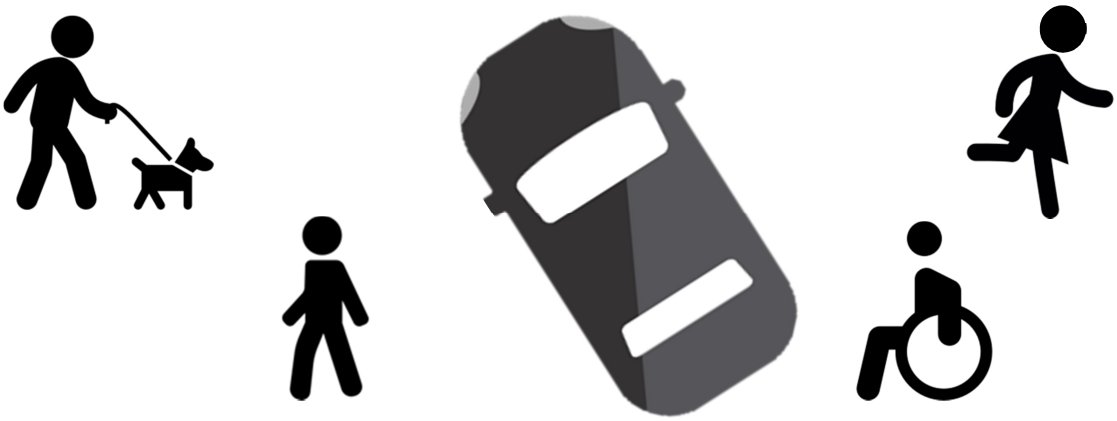




1. Contenido

Figuras:

Tablas:



1. Introducción

El control y monitorización de la movilidad tanto urbana como interurbana es de vital importancia en la sociedad actual. El uso óptimo de las señales activas de tráfico, como es la red semafórica, contribuye a que la circulación sea más ágil, pero hay ocasiones en las cuales una mera programación pasiva de la red semafórica no atiende a las circunstancias del tráfico en tiempo real. Un ejemplo claro podría ser un semáforo de peatones que se pone en verde cuando no hay ningún peatón para cruzar, entorpeciendo y retrasando el tráfico rodado sin necesidad, ilustrado en la figura 1. En este trabajo se va a tratar de dar solución a este problema planteado a pequeña escala.

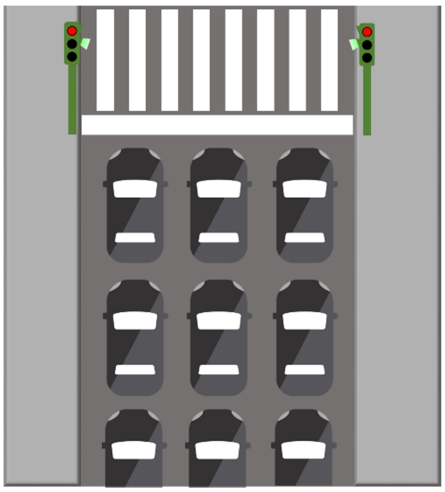


Figura 1 Ejemplo tráfico mal gestionado

El objetivo del proyecto es generar un dispositivo **sencillo**, de **bajo coste** y funcionamiento **a tiempo real** que reduzca el tiempo inútil de espera tanto para peatones como para vehículos. Para ello, se ha ideado la realización de un control automático de regulación de apertura y cierre de semáforos en función del **tiempo de espera** y el número de **peatones**, basado en visión por computador, que preserve además el anonimato de las personas cuyas imágenes sean registradas. Dicho control se implementará sobre un ordenador monoplaca, Raspberry PI 3 B+ y el algoritmo estará basado en librerías de acceso libre, como son OpenCV, ejecutadas en Python.

1.1 Metodología

Como primer paso, se estudiará cómo se ha resuelto el problema de gestión de la temporalización de los semáforos con anterioridad. Igualmente, dentro de la parte de análisis del problema en cuestión, se revisará el estado del arte de los algoritmos de detección de personas aplicados a problemas que pueden ser diferentes al tratado en este trabajo fin de grado.

A continuación, el trabajo presenta dos partes diferenciadas, una de hardware (manejo y puesta en marcha de la placa Raspberry Pi y los elementos periféricos) y otra de software (creación de algoritmos basados en OpenCV sobre Python). Se escogerá un escenario de prueba y se realizará un software que sea capaz de discernir el número de personas que hay en la escena, en el que se grabarán situaciones asociadas al paso de personas por la proximidad de un semáforo, vista desde arriba, y además, lo más independientemente posible de las condiciones meteorológicas y de luminosidad.

Se va a implementar un prototipo en el que, además de grabar imágenes como fuente de información para el algoritmo, se va a simular la conexión a un semáforo cuyos estados se van a controlar con la salida del algoritmo. Para ello se debe diseñar el protocolo de control que combine diferentes variables de entrada (número de personas presentes, tiempo de espera, tiempo desde el último cambio de estado del semáforo…) que determine cuándo el semáforo va a estar en rojo y cuando en verde. Se validará el prototipo con imágenes y vídeos de peatones de acuerdo con participar en el proyecto.

Además, se compararán los resultados obtenidos con otros algoritmos de detección de personas en términos de rapidez, complejidad computacional y rendimiento.

El trabajo se va a estructurar en varias fases, que se esquematizan en la figura 2:

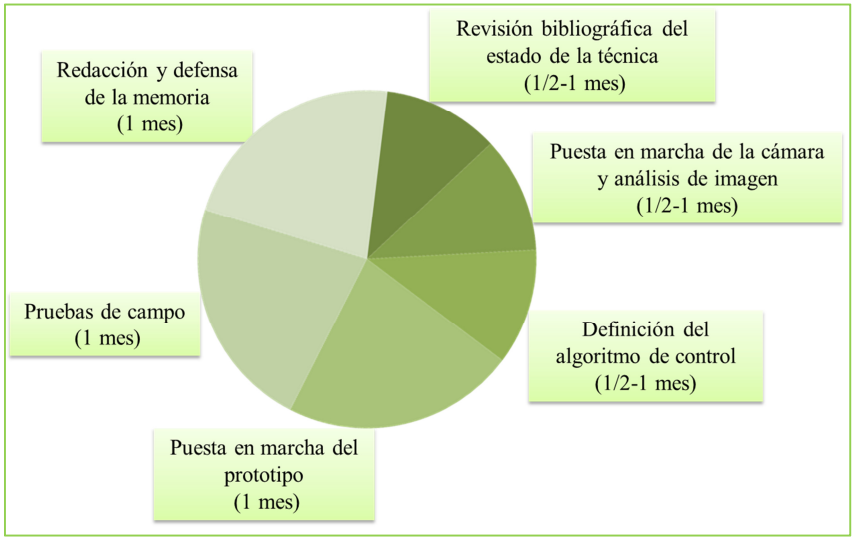


Figura 2 Fases del proyecto

1.2 Resumen de la memoria

La estructura del resto de capítulos de esta memoria, es la siguiente:

- **Estado de arte**: información relacionada con el trabajo.
- **Hardware**: descripción de los componentes hardware usados y su montaje.
- **Software**: descripción de los componentes software usados y el algoritmo implementado.
- **Algoritmo de visión**: presentación de las diferentes técnicas de detección de personas y explicación del algoritmo de visión implementado.
- **Máquina de estados**: explicación del funcionamiento general del semáforo.
- **Resultados y conclusiones**: análisis de los resultados obtenidos.

2. Estado de arte

Las personas cada día se mueven más, bien sea en vehículos particulares, transporte público o a pie. Es por esto que el volumen de tráfico y su complejidad han aumentado de forma desorbitada en las últimas décadas. Para atender a esta complejidad ha aparecido una creciente demanda de sistemas de control activo y adaptativo del tráfico tanto rodado como pedestre, [1]. La motivación principal de este tipo de sistemas es dinamizar el tráfico de forma que no se produzcan atascos o que, de producirse, afecten en menor medida a los usuarios.

La palabra semáforo proviene del griego σῆμα (sema) y φόρος (foros), significa señal y portador, por lo que un semáforo es lo que "lleva las señales". En la figura 3 se representa los hitos más relevantes en la historia del semáforo, [2]:

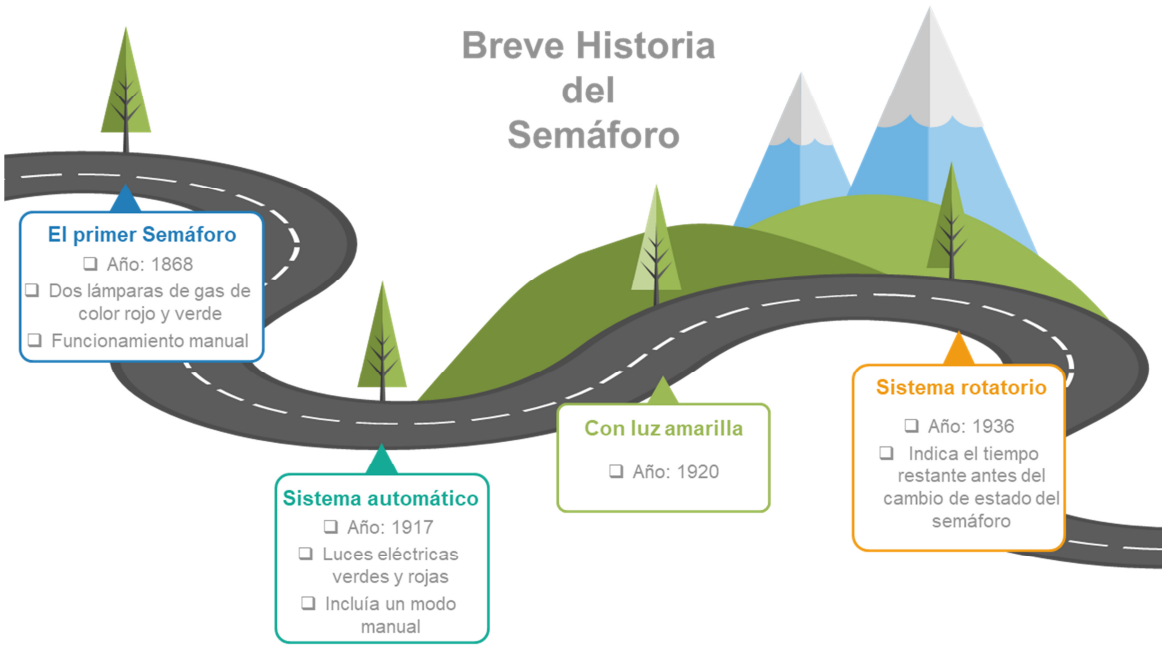


Figura 3 Línea de tiempo histórica del semáforo

La gestión más simple del semáforo es de regulación estática, en el que el ciclo del apagado y del encendido de las luces es fijo. Una mejora de esta gestión, es la regulación en función del instante temporal. Según el día y la hora, el semáforo da más preferencia al conductor o al peatón. También existe semáforos que incluyen un botón para que peatón pueda pedir manualmente el paso, siendo esta una de las alternativas que soluciona el problema planteado en la introducción. La gran desventaja de este tipo de semáforo, es su requerimiento de la intervención humana, que puede causar incomodidades al usuario. Por ejemplo, cuando el peatón tiene las manos ocupadas o cuando no se da cuenta de la necesidad de pulsar el botón.

Actualmente, el concepto de tráfico inteligente, ya no es algo desconocido, sino una meta/tendencia en todas las ciudades. Los semáforos inteligentes aún no están extendidos, pero ya hay ciudades que los tienen, y están en la fase de prueba. Son semáforos que combinan la visión artificial, el análisis de datos y la comunicación. El dispositivo resultante de este proyecto, es pues, un semáforo inteligente a pequeña escala. Un ejemplo de noticia sobre semáforo inteligente:

https://elpais.com/tecnologia/2019/05/30/actualidad/1559210973_315830.html

Está demostrado que con un sistema de control de semáforos pasivo el tiempo perdido tanto por conductores como por peatones en un trayecto promedio es elevado. Este hecho mejora si aplicamos al control semafórico algoritmos adaptativos como lógica fuzzy, algoritmos evolutivos [3], algoritmos de reforzamiento [4], etc. Gran parte de este tipo de algoritmos se basan en primera instancia en la captura de imágenes del tráfico y su análisis y procesado en tiempo real, [5]. Es en este aspecto donde tienen cabida los algoritmos de visión por computador e inteligencia artificial tan ampliamente usados en muchos otros ámbitos científicos y tecnológicos, [6].

La visión artificial es un subcampo de la inteligencia artificial. De una forma simple y sintetizada, consiste en capturar imágenes y procesar el contenido que hay en ellas para obtener información. En la figura 4, se lleva a cabo un breve recorrido por la historia de la visión por computador, [7]:

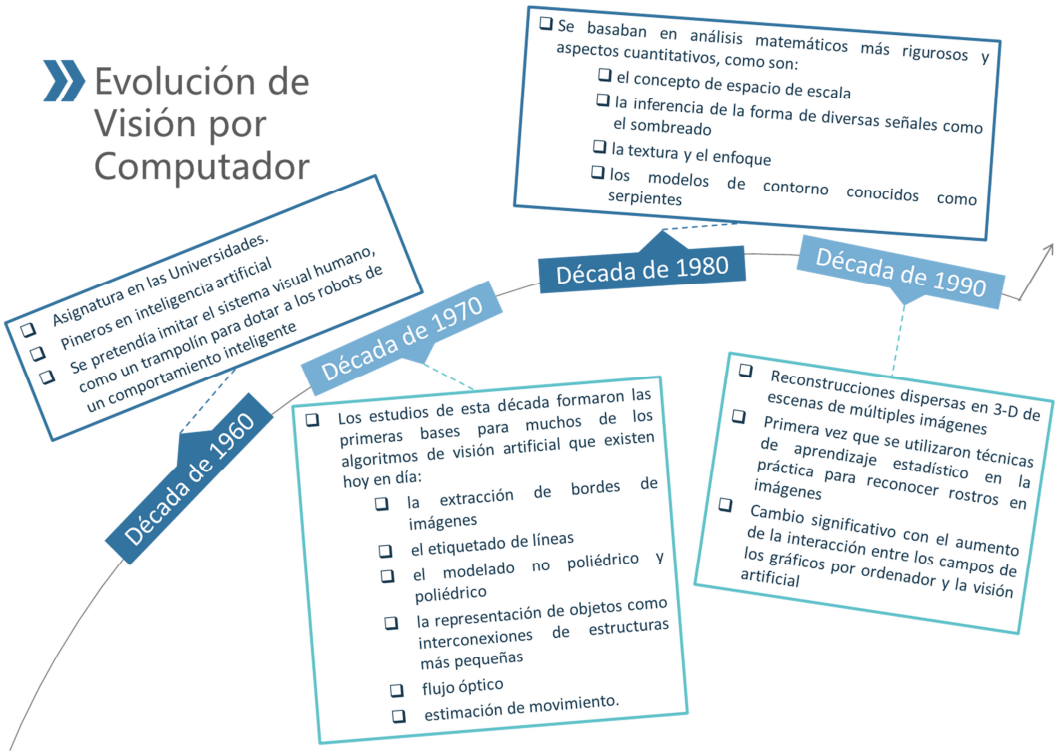


Figura 4 Línea de tiempo de la evolución de Visión por Computador

Por otro lado, la miniaturización y portabilidad de dispositivos es un hecho a tener en cuenta en cualquier diseño. En esta línea han aparecido en los últimos años miniordenadores y placas procesadoras con altas capacidades de cálculo como Arduino, BeagleBone, Minnowboard, Raspberry Pi, etc. En particular, Raspberry Pi ha sido utilizada como placa de control en numerosas aplicaciones tan dispares como la domótica, servidores web, detección multisensor, etc, [8].

3. Hardware

En este capítulo se detalla los componentes hardware del proyecto. Empezando por el listado de componentes y terminando con la interconexión entre ellos.

3.1 Listado

Hardware	Precio por unidad(euros)	Unidad	Especificación	
Ordenador monoplaca	50	1	Raspberry Pi Versión 3B+ Figura 5	Principales
Cámara	26	1	Raspberry Pi Picamera Module V2 8MP Figura 6	
Semáforo	10	1	Figura 7	
Monitor	100	1	Conexión preferiblemente HDMI	Auxiliares
Teclado	10	1	Conexión USB	
Ratón	10	1	Conexión USB	
Resistencia	0.5	5	Mayor de 65Ω	
Cables		10	Latiguillos	
LED		5	2 Led verde, 2 Led rojo, 1 Led amarillo	

Tabla 1 Elementos de Hardware

El cerebro del dispositivo es la placa procesadora Raspberry Pi 3 model B+. Para la toma de imágenes se ha conectado a la placa una cámara Picamera, desarrollada exclusivamente para Raspberry Pi y optimizada con tal fin. Por otro lado, se han configurado los pines I/O necesarios como salidas analógicas para el control de las respectivas luces del semáforo, simuladas con el uso de LEDs. El resto de elementos auxiliares sirven de apoyo a la hora de construir el prototipo.

Figura 5 Placa procesadora Raspberry Pi 3 model B+

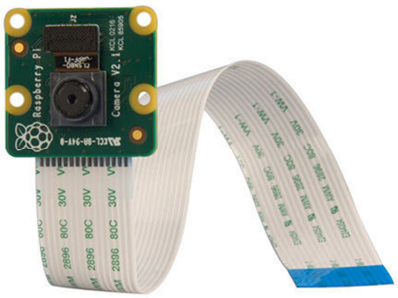


Figura 6 Placa cámara Picamera

Figura 7 Semáforo

3.2 Conexión

A continuación se explicará las conexiones hardware necesarias para construir el dispositivo. En la Figura 8 se muestra un dibujo simple del aparato mostrando la placa procesadora Raspberry Pi, la cámara Picamera y un semáforo.

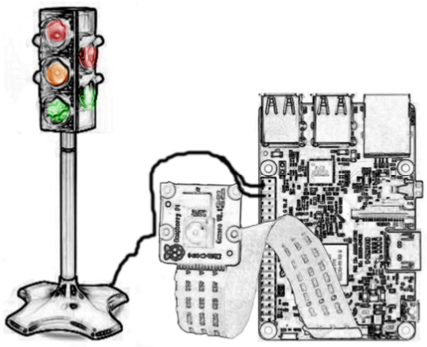


Figura 8 Esquema del dispositivo

❖ Elemento auxiliar – Placa

La alimentación de la Raspberry Pi se realiza mediante un cable micro USB. Con los cuatro puertos USB que tienen la placa, se puede conectar el teclado y el ratón. Por otro lado, la salida HDMI es para la conexión del monitor. En la figura 9, se pueden localizar los distintos puertos:

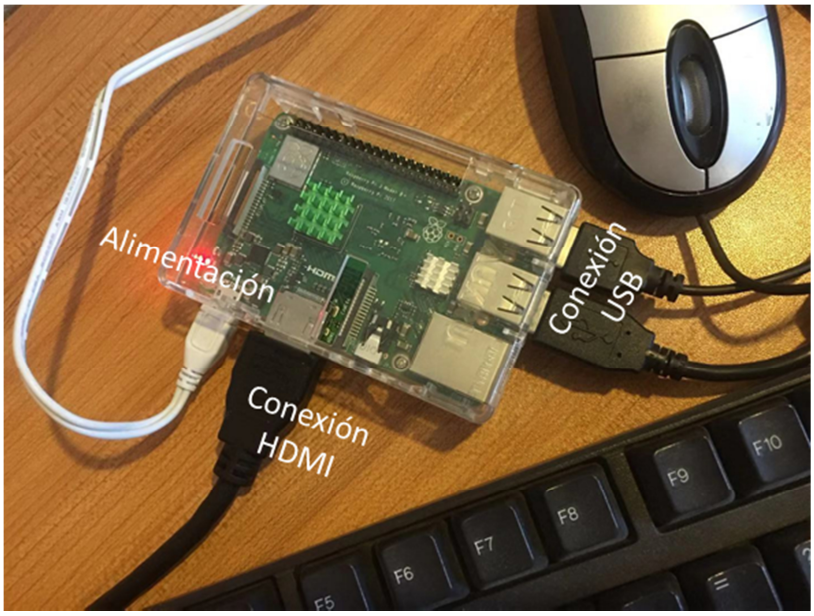


Figura 9 Conexión con los elementos auxiliares

❖ Cámara – Placa

El módulo de cámara PiCamera es un complemento diseñado a medida para Raspberry Pi. Se conecta a Raspberry Pi a través de un zócalo en la superficie superior de la placa, tal y como se demuestra en la figura 10:

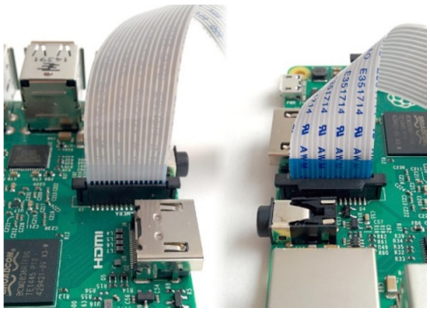


Figura 10 Conexión cámara-procesador

❖ Led - Placa

Los pines de salida de la placa de Raspberry Pi ofrece 3.3V cada uno, la tensión de un led son de 2V con una corriente máxima admisible de 20mA. Por lo que la tensión de caída mínima en la resistencia será de 1.3V (3.3V-2V). Para no superar la intensidad máxima del led, la resistencia debe ser mayor de 65Ω (1.3V/20mA). En la figura 11, se ilustra la numeración de los pines por el número del pin del enchufe y un ejemplo de conexión para un led [9]:

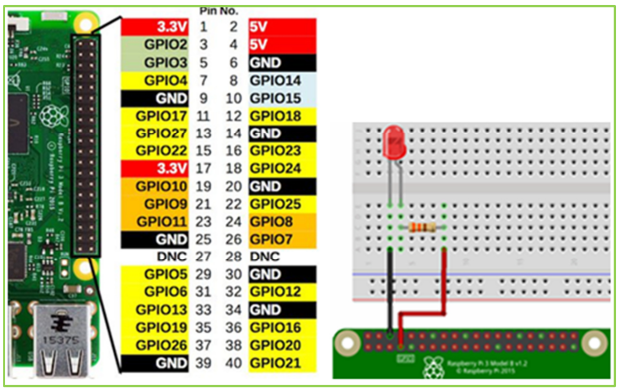


Figura 11 Numeración de los pines de salida del Raspberry Pi y conexión para un led

Se ha decidido utilizar resistencias de 1kΩ, las salidas 16 y 18 para las luces del peatón, y las salidas 11, 13 y 15 para los coches. En la figura 12 se muestra el resultante:

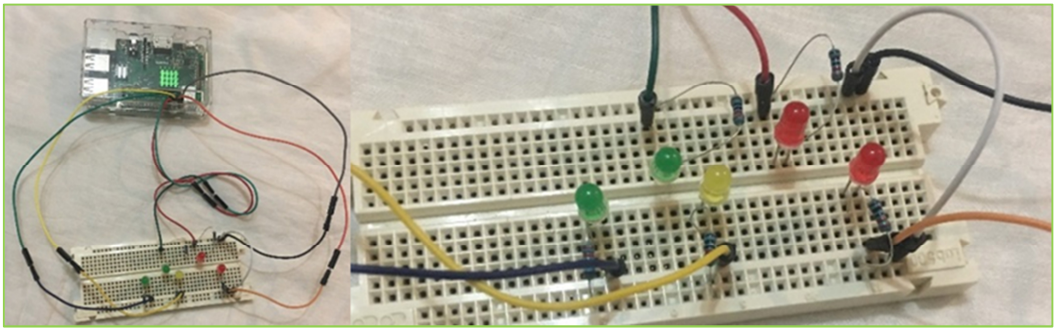


Figura 12 Conexión Led-Resistencia-Placa

4. Software

A continuación se definen los componentes software del proyecto. Comenzando por el listado de componentes, continuando con el algoritmo.

4.1 Listado

	Detalle
Sistema operativo	Raspbian GNU 9.4
Lenguaje de programación	Python 2.7
Entorno de Desarrollo Integrado	Thonny Python IDE
Cámara	Librería Picamera
Manipulación de imagen	Librería OpenCV 3.4
Led	Librería GPIO
Tiempo	Librería time

Tabla 2 Listado Software

En la página oficial del Raspberry Pi hay abundantes informaciones y tutoriales, entre los que se encuentra también los pasos a seguir para instalar el sistema operativo Raspbian:

https://www.raspberrypi.org/downloads/raspbian/

El entorno de desarrollo integrado Thonny Python IDE [10] viene instalado por defecto en el sistema operativo. Es un entorno sencillo y fácil de usar. Se abre desde la pestaña de Menu>Programming, tal y como se demuestra en la figura 13:

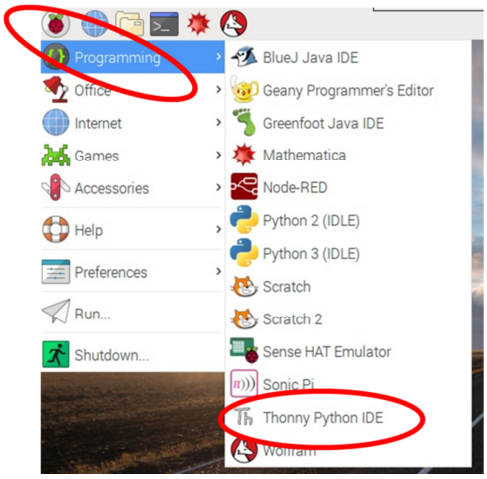


Figura 13 entorno de desarrollo integrado Thonny Python IDE

El algoritmo generado para este trabajo está basado en las librerías de procesado digital de imagen OpenCV funcionando sobre el lenguaje de programación Python. Se trata por tanto de un software completamente libre con las ventajas que ello conlleva a la hora de la aplicación final y puesta en marcha del dispositivo.

Existen muchas formas de descargar e instalar las librerías, en este proyecto se ha optado instalarlos mediante Miniconda.

Anaconda es una distribución gratuita y de código abierto de los lenguajes de programación Python y R orientados a programación, que tienen como objetivo simplificar la administración e implementación de paquetes, [11]. En la figura 14 se exponen sus contenidos:

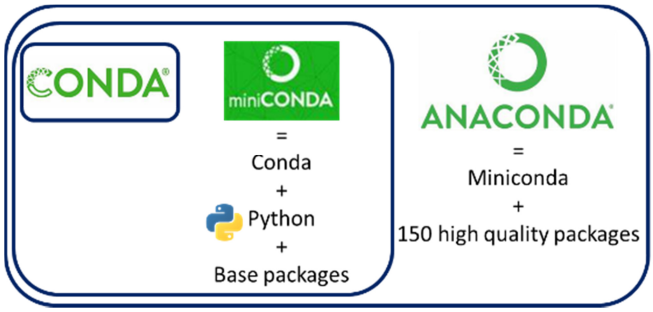


Figura 14 Conda-Miniconda-Anaconda

Miniconda es una versión simplificada de Anaconda, que proporciona el intérprete de Python, junto con una herramienta de línea de comandos llamada conda que funciona como un gestor de paquetes multiplataforma orientado a paquetes de Python. Una vez obtenido miniconda, es muy fácil instalar cualquier librería. La instalación de miniconda, Python y OpenCV tarda unos 30 minutos en total. En el siguiente enlace se especifican los pasos a seguir:

https://www.hackster.io/lbtinglb/install-opencv-through-miniconda-in-second-on-rasberry-pi-3-4eec27?source=post_page-----34c9740d78e4----------

Los archivos generados y sus distribuciones están reflejados en la figura 15, y sus contenidos se detallan en los próximos apartados.

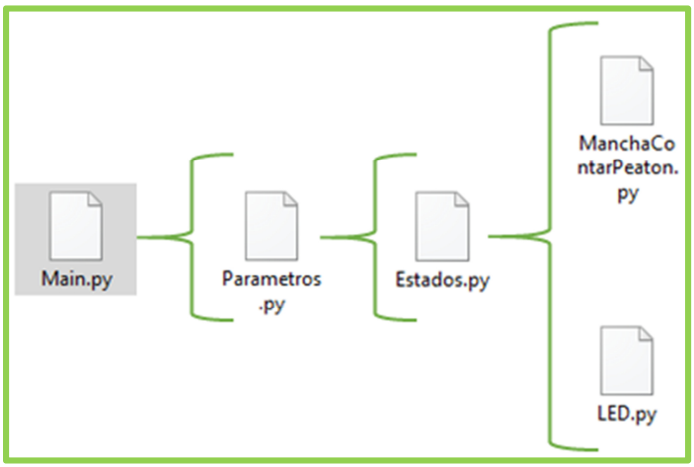


Figura 15 Distribución de archivos

Main.py: es el programa principal.

Parametros.py: se encuentra todos los parámetros regulables.

Estados.py: clase de control del semáforo.

LED.py: clase de control de los leds.

ManchaContarPeaton.py: clase de la técnica de visión para recuento de peatones.

4.2 Algoritmo:

El funcionamiento del programa se resume en la figura 16:

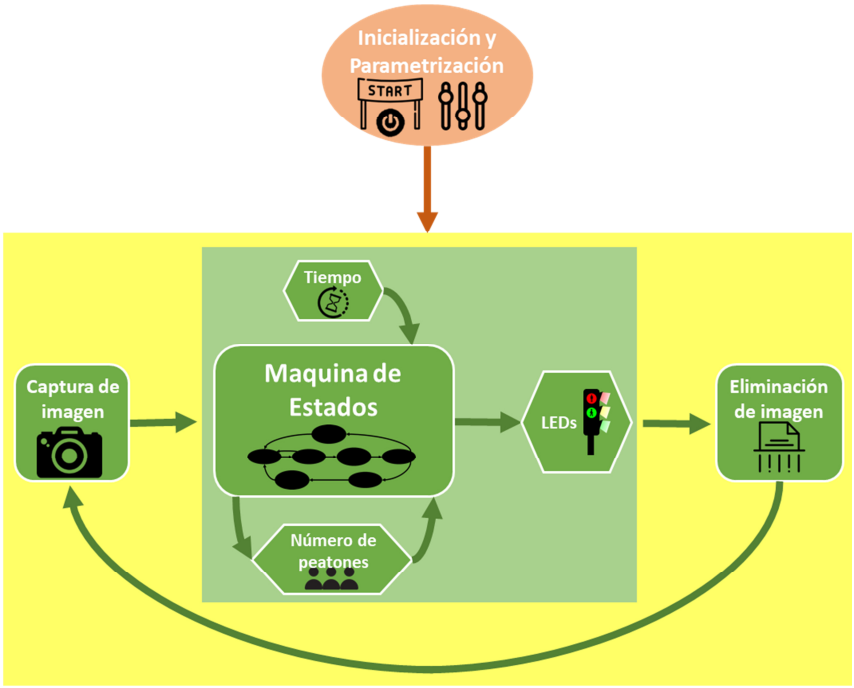


Figura 16 Diagrama de flujo del programa

Zona Roja: se ejecutara al principio del programa y solamente una vez. Consiste en inicializar y parametrizar los variables y los parámetros del sistema. Según la posición de la cámara es necesario regularlo manualmente aunque se podría automatizar.

Zona Amarilla: es el proceso cíclico que se ejecuta de forma infinita. Comienza haciendo una captura de imagen sobre el escenario, luego lo procesa la máquina de estados y termina eliminando la imagen capturada.

Zona Verde: es la zona de procesamiento de imagen y control de semáforo. El diagrama simplificado de la máquina de estados se visualiza en la figura 17:

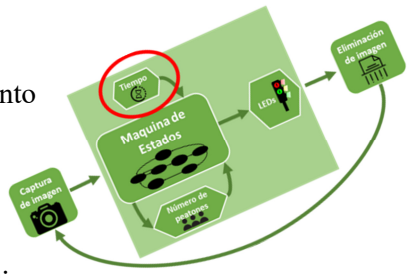
Figura 17 Maquina de estados

En cada uno de los estados se realiza el análisis correspondiente del fotograma para conocer el número de peatones, y se define también el apagado o el encendido de los LEDs. La transición de un estado a otro se realiza en función del número de peatones y del tiempo transcurrido.

En los sucesivos apartados se detallarán los contenidos de cada uno de los bloques del proceso ciclico (zona amarilla).

❖ Tiempo

La librería de control de tiempo se llama "time". Viene junto con miniconda como librería básica. Para usarla, sola hace falta importarla:

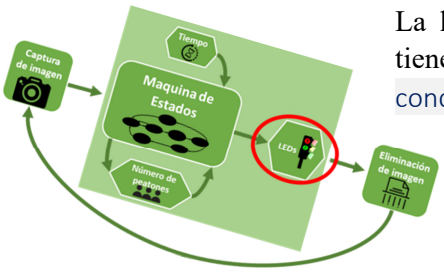



```
import time
```

Alguno de los métodos más comunes de la librería "time" son:

```
tActual=time.time() #Devuelve el tiempo actual.
time.sleep(t) #Suspende la ejecución durante t segundos.
```

❖ Led



La librería de salida de los pines de la placa procesadora, tiene el nombre de GPIO. Se instala ejecutando el comando: conda install -c poppy-project rpi.gpio en el terminal.

GPIO = general-purpose input/output (entrada/salida de uso general)

El código de la clase LED creada para este trabajo se encuentra en el anexo IV, parte de código es la siguiente:

```
import RPi.GPIO as GPIO
class LED:
    def __init__(self):
        self
        GPIO.setmode(GPIO.BOARD)
        GPIO.setwarnings(False)
        GPIO.setup(16,GPIO.OUT)#PeatonRojo
        GPIO.setup(18,GPIO.OUT)#PeatonVerde
        GPIO.setup(11,GPIO.OUT)#CocheRojo
        GPIO.setup(13,GPIO.OUT)#CocheAmarillo
        GPIO.setup(15,GPIO.OUT)#CocheVerde
        GPIO.output(16,GPIO.LOW)#PeatonRojo
        GPIO.output(18,GPIO.LOW)#PeatonVerde
        GPIO.output(11,GPIO.LOW)#CocheRojo
        GPIO.output(13,GPIO.LOW)#CocheAmarillo
        GPIO.output(15,GPIO.LOW)#CocheVerde
    def PeatonRojo(self):
        GPIO.output(16,GPIO.HIGH)#PRojo
        GPIO.output(18,GPIO.LOW)#PVerde
    def LimpiarLED(self):
        GPIO.cleanup()
```

```
import RPi.GPIO as GPIO
```

Importar la librería de salida de los pines y llamarle simplificando con el nombre de GPIO.

```
GPIO.setmode(GPIO.BOARD)
```

La opción BOARD especifica que se está refiriendo a los pines por el número del pin del enchufe.

```
GPIO.setwarnings(False)
```

Deshabilitar las advertencias.

```
GPIO.setup(16,GPIO.OUT)#PeatonRojo
```

Configurar el pin 16 como salida.

```
GPIO.output(16,GPIO.LOW)#PeatonRojo
```

Establecer la salida del pin 16 a bajo, a 0V.

```
GPIO.output(16,GPIO.HIGH)#PRojo
```

Establecer la salida del pin 16 a alto, a 3V.

```
GPIO.cleanup()
```

Limpiar todos los puertos que se ha utilizado.

❖ Cámara

Una vez conectada físicamente la cámara, debe habilitarse en el procesador. Para ello, se debe abrir un terminal y ejecutar el comando: $ `sudo raspi-config`

Aparecerá la siguiente ventana de la figura 18, se debe elegir la opción 5, activar la cámara, finalizar y reiniciar el equipo.

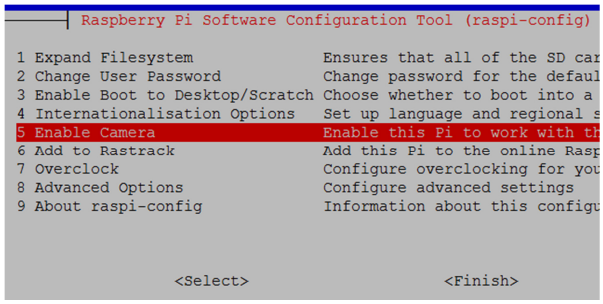
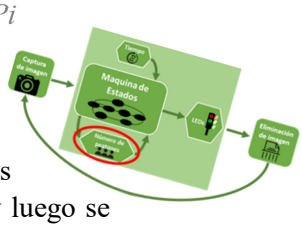


Figura 18 Ventana habilitar cámara

Con la cámara habilitada, se procede a instalar la librería PiCamera ejecutando el comando: `conda install -c gaiar picamera`

En el Anexo VI, está un programa sencillo de captura, muestra y eliminación de la imagen con la cámara

Mucha más información sobre la librería PiCamera: https://picamera.readthedocs.io/en/release-1.10/quickstart.html



5. Algoritmo de visión

En el presente capitulo se van a introducir en primer lugar los conceptos teóricos que a los que se han recurrido durante el desarrollo del algoritmo y luego se profundiza sobre el código implementado.

5.1 Técnicas de visión

❖ Operaciones básicas

Escala gris

Consiste en transformar una imagen de colores a blanco y negro. Hay muchas formas de conseguirlo, la más simple es usar la función de leer imagen y leerlo en escala gris. A continuación se muestra el código y su efecto, figura 19:

```
imgGris=cv2.imread("imgOriginal.jpg",0)
#El cero indicar leer en escala gris
```

Figura 19 Efecto escala gris

Recorte

Consiste en escoger la zona de interés de la imagen. El código es muy simple, solo hace falta indicar dos puntos, inicial y final. En la figura 20 se demuestra su efecto:

```
imgRecortado=imgOriginal[400:800,1:500]
#del X inicial 400 a X final 800
#del Y inicial 1 a Y final 500
```

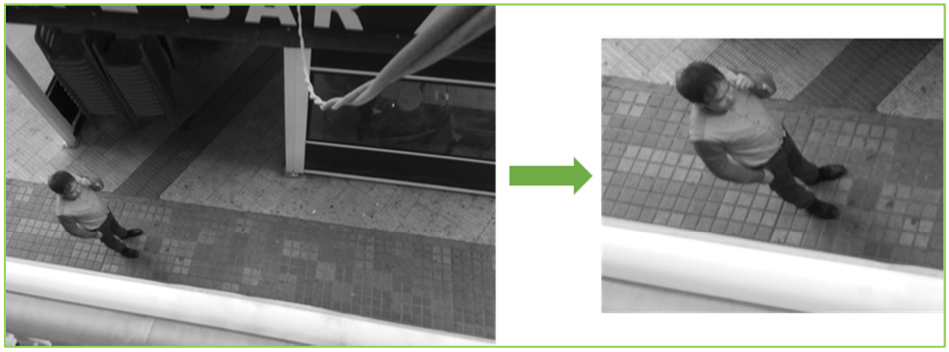


Figura 20 Efecto recorte

Rotar

Consiste en rotar la imagen. Para ello es mejor definir un método:

```
def Rotar(img,angulo):
    rows,cols = img.shape
    M=cv2.getRotationMatrix2D((cols/2,rows/2),angulo,1)
    imgRotada=cv2.warpAffine(img,M,(cols,rows))
    return imgRotada
imgRotado=Rotar(imgOriginal,10)
```

En la figura 21, se muestra la imagen rotada:

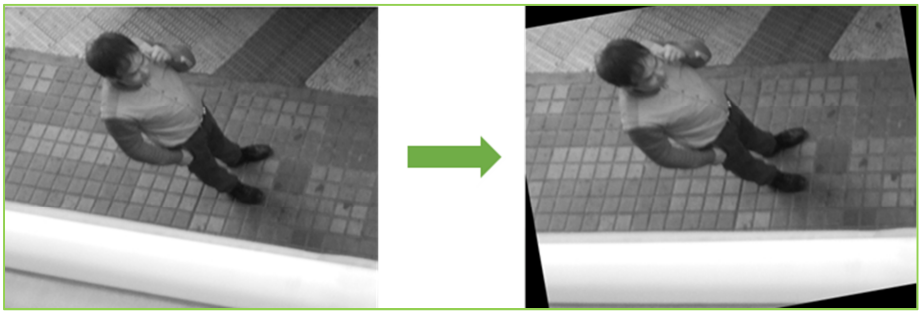


Figura 21 Efecto rotar

❖ Sustracción de fondo

La substracción de fondo, en inglés *background subtraction*, consiste en identificar todos los píxeles de una imagen bien como fondo, o como primer plano, [12], [13] y [14]. En el primer plano se engloban todos los objetos relevantes, mientras que en el fondo se encuentran todos aquellos objetos sin interés. En la figura 22 se ilustra dicha idea:

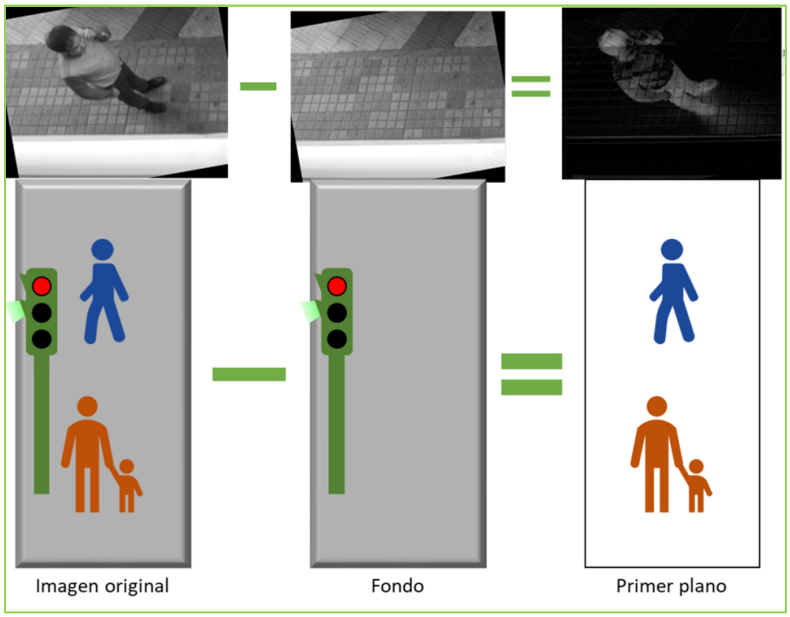


Figura 22 Sustracción de fondo

Normalmente tras una sustracción de fondo, se realiza una binarización de la imagen de primer plano, para eliminar ruidos y sobretodo destacar elementos de interés. Consiste en establecer un umbral, por el cual, todos aquellos que estén por encima del umbral serán píxeles blancos y los que estén por debajo serán negros, tal y como se ilustra en la figura 23:

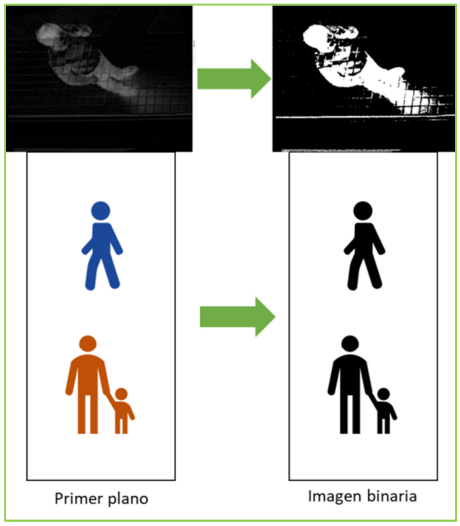


Figura 23 Binarización de imagen

Los métodos de sustracción de fondo más usados son:

Sustracción estática:

Consiste en hacer una resta directa entre la imagen de fondo y la imagen de análisis. Es un método sencillo y rápido, pero es muy sensible a los cambios de la escena, iluminación, sombra,… Se recomienda su uso para entornos controlados.

El código del método es la siguiente:

```
    def SustraccionFondoEstatico(self,frame,fondo):
        FrameGau = cv2.GaussianBlur(frame, (self.Gau, self.Gau), 0)
        FondoGau = cv2.GaussianBlur(fondo, (self.Gau, self.Gau), 0)
        resta = cv2.absdiff(FondoGau, FrameGau)
        imgBinaria = cv2.threshold(resta, self.LimiteBi, 255, cv2.THRESH_
BINARY)[1]
        return imgBinaria
```

Con la función de desenfoque gaussiano, cv2.GaussianBlur, se consigue minimizar el ruido de la imagen y los detalles:

```
        FrameGau = cv2.GaussianBlur(frame, (self.Gau, self.Gau), 0)
        FondoGau = cv2.GaussianBlur(fondo, (self.Gau, self.Gau), 0)
```

Se calcula la diferencia absoluta entre las dos imagines y luego se binariza la imagen con la función cv2.threshold:

```
        resta = cv2.absdiff(FondoGau, FrameGau)
        imgBinaria = cv2.threshold(resta, self.LimiteBi, 255, cv2.THRESH_
BINARY)[1]
```

Sustracción dinámica:

Se trata de métodos basados en la memoria, tienen en cuenta los fotogramas anteriores, y van actualizando la imagen de fondo. Se usan normalmente para detectar objetos en movimiento. A diferencia del estático son mucho menos sensibles a los cambios del entorno y se obtienen mejores resultados a cambio de tener más memoria y aumentar el tiempo de cómputo.

El más famoso es el BackgroundSubtractorMOG2, que está basado en el modelo Gaussian Mixture. Se trata de un modelo compuesto por la suma de varias distribuciones Gaussianas, y posee los siguientes parámetros principales:

- history: número de los últimos fotogramas que afectan al modelo de fondo.
- varThreshold: umbral de varianza para decidir si una muestra está bien descrita por el modelo de fondo o no.
- detectShadows: Si el valor es verdadero, el algoritmo detecta sombras y las marca.

En el archivo *ManchaContarPeaton.py* se encuentra el código de la creación y uso del método:

La creación del método:

```
    def MetodoDinamico(self,metodoDinamico):
        self.metodoDinamico=metodoDinamico
```

La utilización del método:

```
    def AplicarMetodoDinamico(self,frame):
        imgBinaria = self.metodoDinamico.apply(frame,learningRate=-1)
        return imgBinaria
```

En el archivo *Parametro.py* es donde se define el método a usar y sus parámetros:

```
    #Metodo fondo dinamico:
MOG2 = cv2.createBackgroundSubtractorMOG2(history=70, varThreshold=16, de
tectShadows=False)
Analisis.MetodoDinamico(MOG2)
```

En la siguiente enlace, muchas más información sobre la clase BackgroundSubtractorMOG2

https://docs.opencv.org/ref/2.4/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html

❖ Operaciones morfológicas

La imagen de salida del proceso de substracción de fondo suele ser una imagen binaria con defectos y/o ruidos, que necesita mejoras. Las operaciones más comunes sobre imágenes binarias reciben el nombre de operaciones morfológicas, entre ellas:

- Erosión: Proceso de eliminar píxeles del contorno de un objeto. Se produce un decrecimiento de su área.
- Dilatación: Proceso de añadir píxeles al contorno de un objeto. Se produce un incremento de su área.
- Apertura: Erosión seguida de una dilatación.
- Cierre: Dilatación seguida de una erosión.

Se crea en primer lugar el elemento morfológico y luego se llama a las funciones:

```
kernelEro = np.ones((5,5),np.uint8)
kernelDila = np.ones((25,25),np.uint8)

Ero = cv2.erode(imgBinaria,kernelEro,iterations=1)
Dila = cv2.dilate(imgBinaria, kernelDila, iterations=1)
EroDila = cv2.dilate(Ero, kernelDila, iterations=1)
```

En la figura 24, se ilustra las salidas después de aplicar las operaciones.

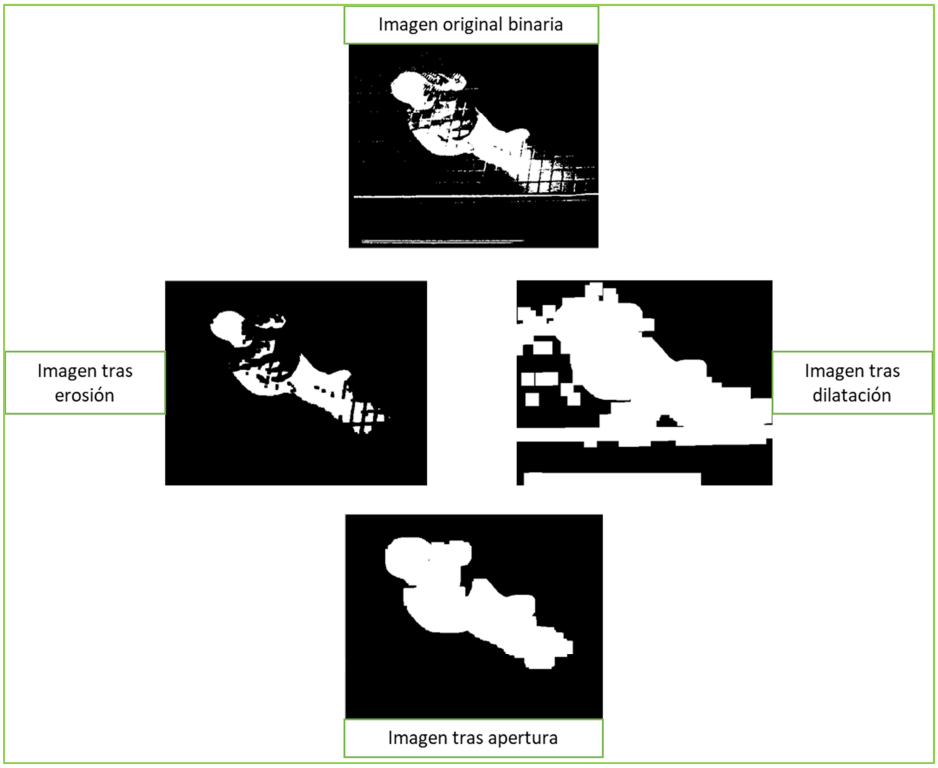


Figura 24 Efecto operación morfológica

❖ Buscar contornos

Con la función cv2.findContours se puede buscar en una imagen los contornos de los objetos, y es una herramienta muy útil para el análisis de forma, área y reconocimiento.

Para una mejor precisión, se recomienda utilizar imágenes binarias, pues en OpenCV encontrar contornos es como encontrar objetos blancos sobre el fondo negro.

La documentación que ofrece OpenCV sobre esta función está en el siguiente enlace:

https://docs.opencv.org/3.0.0/d4/d73/tutorial_py_contours_begin.html

❖ Histograma de gradientes orientados y Support vector machines (HOG y SVM)

En matemáticas, el 'gradiente' es una generalización multivariable de la derivada. Mientras que una derivada se puede definir solo en funciones de una sola variable, para funciones de varias variables, el gradiente toma su lugar, [15].

El gradiente puede proporcionar información de los cambios de intensidad en una imagen. Como cualquier vector, el gradiente se define por su magnitud y su orientación. La magnitud de cada pixel da información sobre la forma de un objeto, y la orientación aporta la forma del contorno. En la figura 25, se puede ver como es la gradiente de una imagen:

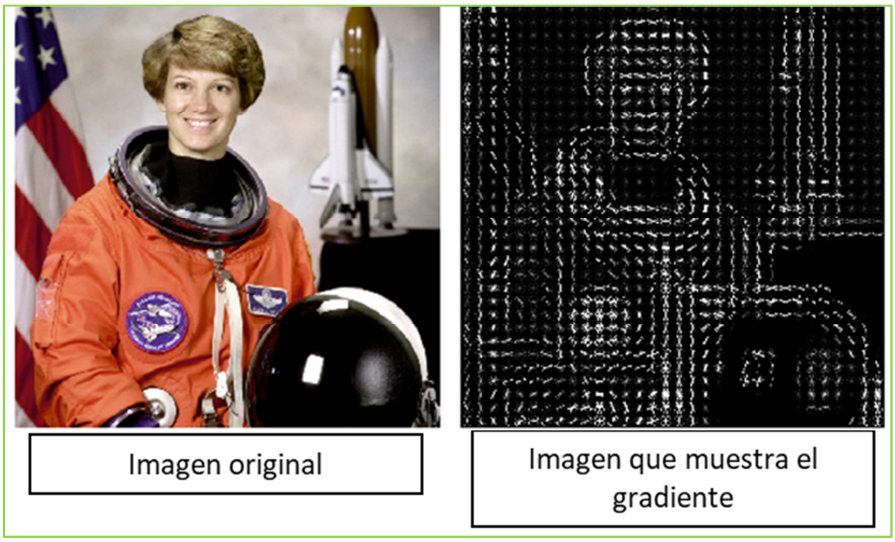


Figura 25 Imagen original VS Imagen gradiente

La información local del gradiente para cada uno de los pixeles es muy sensible a pequeñas variaciones, por lo que es necesario convertir dicha información en una representación global de toda la imagen, en forma de vector de características. Este descriptor global se conoce con el nombre de HOG (**histograma** de **gradientes orientados**).

SVM (en inglés *support vector machines*) es un método de **clasificación** de aprendizaje **supervisado** conocido como máquinas de vectores de soporte. Tomando los datos de entrada como conjuntos de vectores en un espacio n-dimensional, el algoritmo de SVM construye un hiperplano de separación en ese espacio.

En el siguiente enlace se ofrece un curso online sobre detección de objetos, y en la semana 4 del curso, se encuentra detallada toda la teoría de detector de peatones basado en HOG y SVM:

https://www.coursera.org/lecture/deteccion-objetos/l4-1-detector-de-peatones-basado-en-hog-svm-AMcDl

En el anexo VII se encuentra un ejemplo práctico de uso de dicha técnica y en la figura 26 el resultado:

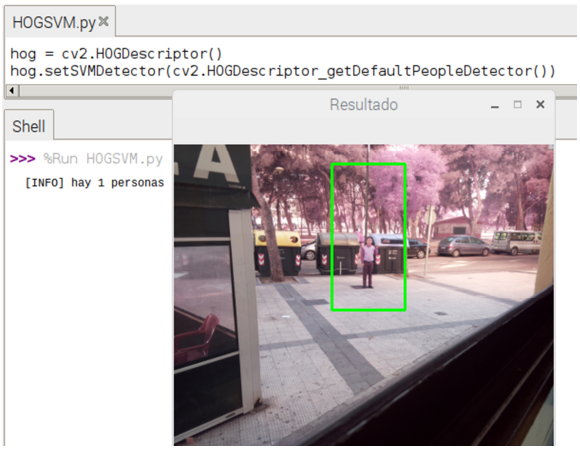


Figura 26 Resultado HOG y SVM

❖ Seguimiento

Asignando una identificación única a cada objeto detectado, se podría hacer un seguimiento del objeto y mejorar el recuento de personas. En el siguiente enlace se encuentra un tutorial de seguimiento y conteo de peatones:

https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

En el anexo VIII está parte del código del enlace anterior, con su técnica de seguimiento, pero con la detección de peatones mediante sustracción de fondo. Los resultados se muestran en la figura 27:

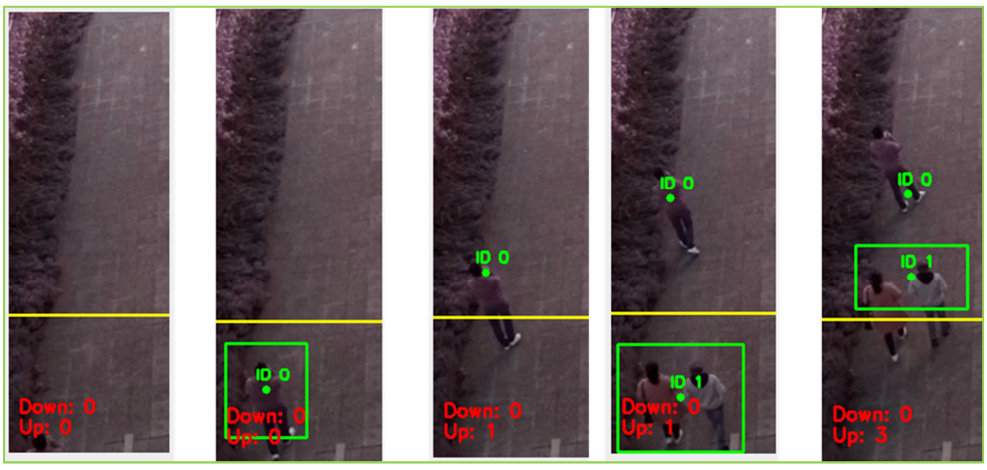


Figura 27 Resultado seguimiento

5.2 Algoritmo de detección de peatón mediante la técnica de sustracción de fondo

Se ha hecho pruebas con varios algoritmos para conseguir la detección del peatón. El implementado en este proyecto está basado en la sustracción de fondo, la principal razón de utilizar esta técnica es por su rapidez y su simplicidad. El algoritmo de visión artificial consiste en los siguientes cuatro pasos:

1) Se hace un filtro y un tratamiento sobre la imagen de entrada, escogiendo la zona de interés y adaptándola para el posterior procesado. Son operaciones básicas de imagen, rotación y recorte, tal y como se muestra en la figura 28:

Figura 28 Paso de imagen original a filtrada

2) Se substrae el fondo, consiguiendo una imagen binaria, ilustrado en la figura 29. Para conseguirlo se hace una combinación entre un método dinámico y un método estático. El dinámico para actualizar el fondo y detectar peatones en movimiento, y el estático para la zona de espera, cuando el peatón se queda quieto.

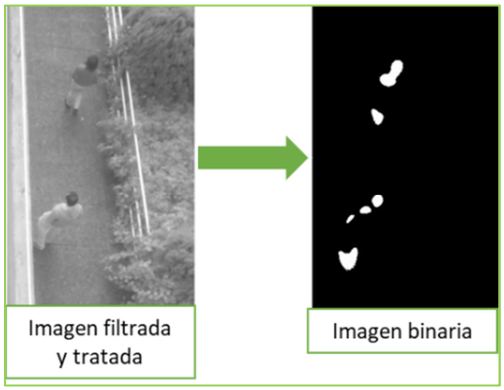


Figura 29 Paso imagen filtrada a binaria

3) Se manipula la imagen binaria con las operaciones morfológicas para obtener una imagen de manchas, donde las manchas son personas. En la figura 30 se puede observar el paso:

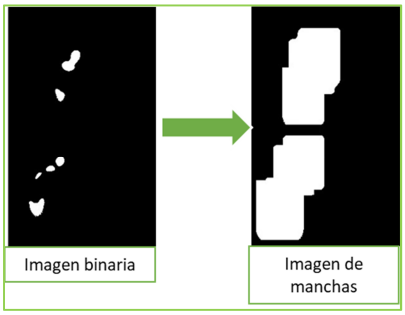


Figura 30 Paso de imagen binaria a de manchas

4) Se integra el área de las manchas, deduciendo así el número de peatones. Consiste en sumar todas las áreas de manchas que fueran posibles peatones, y se divide dicha área total por el área aproximada que ocuparía una persona

El código completo de la clase ManchaContarPeaton.py se encuentra en el anexo V, los dos métodos principales son:

```
    def ConteoPeatonDinamico(self,frame):
        #Zono de Interes
        imgInteres=self.FiltroZonaInteres(frame)
        #Sustraccion de Fondo
        imgBinaria = self.AplicarMetodoDinamico(imgInteres)
        #Manipulacion
        imgMancha = self.ManipulacionImagenBinaria(imgBinaria,self.Elemen
toEro,self.ElementoDila,self.iteEroD,self.iteDilaD)
        #Conteo
        Npeaton = self.BuscaManchaContorno(imgMancha,self.AreaMinD,self.A
reaEstandar)
        return Npeaton

    def ConteoPeatonEstatico(self,frame,fondo):
        #Zono de Interes
        imgInteres=self.FiltroZonaInteres(frame)
        #Sustraccion de Fondo
        imgBinaria = self.SustraccionFondoEstatico(imgInteres,fondo)
        #Manipulacion
        imgMancha = self.ManipulacionImagenBinaria(imgBinaria,self.Elemen
toEro,self.ElementoDila,self.iteEro,self.iteDila)
        #Conteo
        Npeaton = self.BuscaManchaContorno(imgMancha,self.AreaMin,self.Ar
eaEstandar)
        return Npeaton
```

El origen de los errores importantes se produce en la fase de substracción de fondo, cuando la diferencia entre el primer plano y el fondo es muy grande (cambio brusco) o muy pequeño (peatón camuflado). Con las operaciones morfológicas de erosión y dilatación, se pretende eliminar los ruidos de la imagen binaria y aumentar los tamaños de las personas respectivamente. Pero si el fotograma binario de entrada llevaba errores significativos, con estas operaciones dichos errores se incrementarán.

El tiempo de cómputo de cada paso es:

- Paso Cero: Operaciones de la cámara, captura y eliminación de imagen: 0.2segundos
- Paso Uno: De original a filtrada, 0.1segundos
- Paso Dos: De filtrada a binaria, 0.04segundos
- Paso Tres: De binaria a de manchas, 0.02segundos
- Paso Cuatro: Contar manchas y calcular número de peatones, 0.01segundos

El tiempo total en el que dedica el algoritmo al análisis de la imagen, desde su entrada hasta su eliminación es, como máximo unos 0,4segundos en los peores casos.

En las siguientes figuras se muestran los resultados de detección de ninguno, uno y dos peatones:

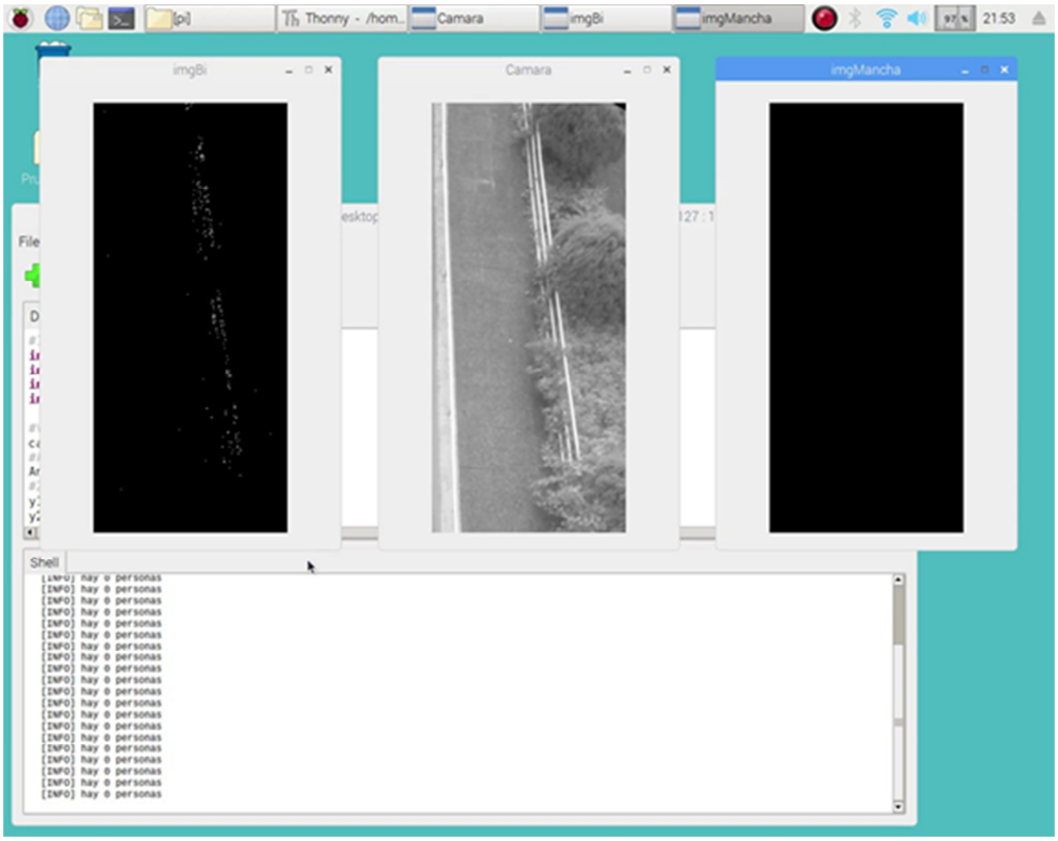


Figura 31 Resultado sin peatón

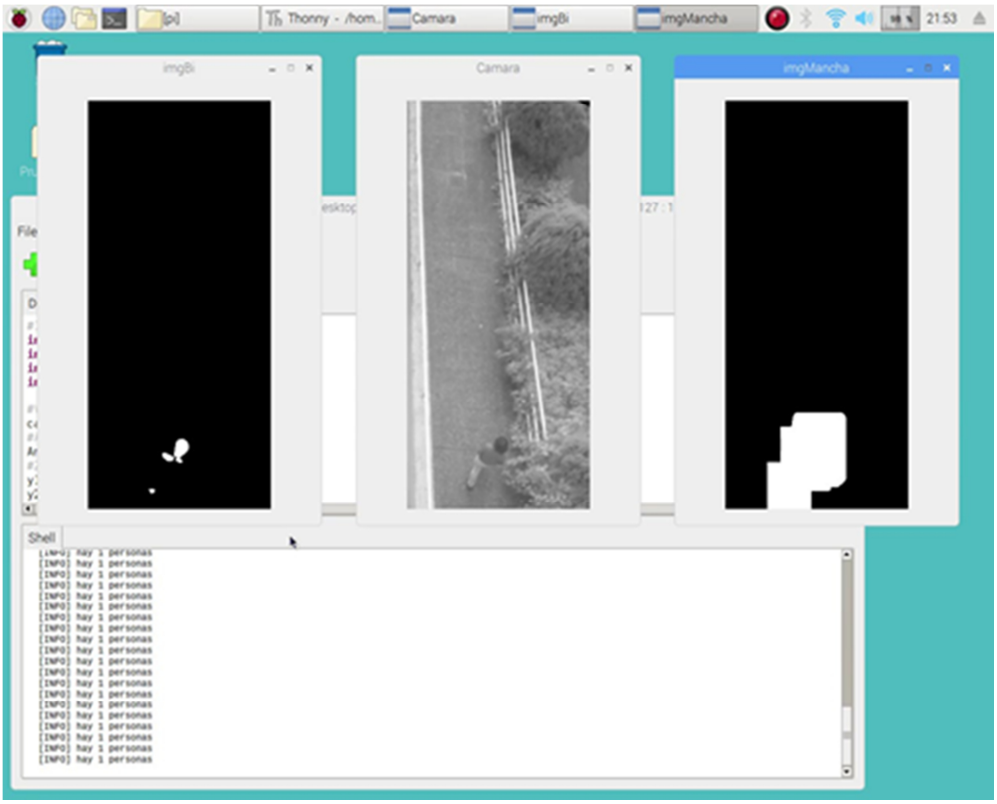


Figura 32 Resultado un peatón

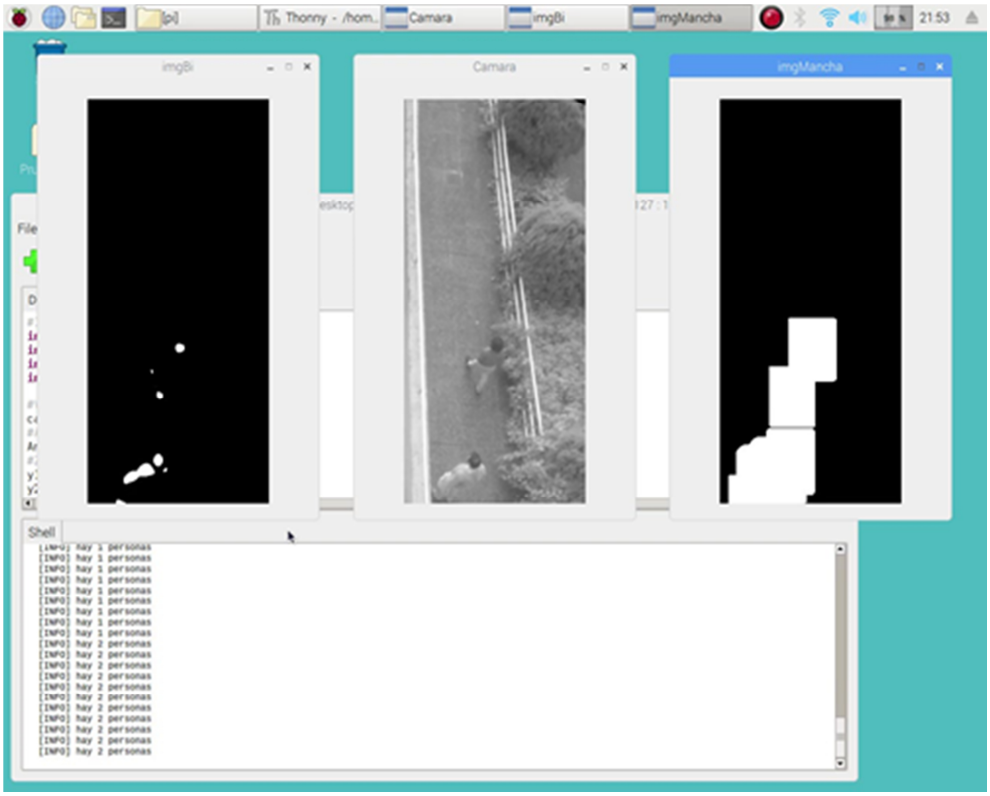


Figura 33 Resultado dos peatones

6. Máquina de Estados

En esta sección se explicara el funcionamiento del semáforo, desglosando cada estado del diagrama de flujo de la figura 34:

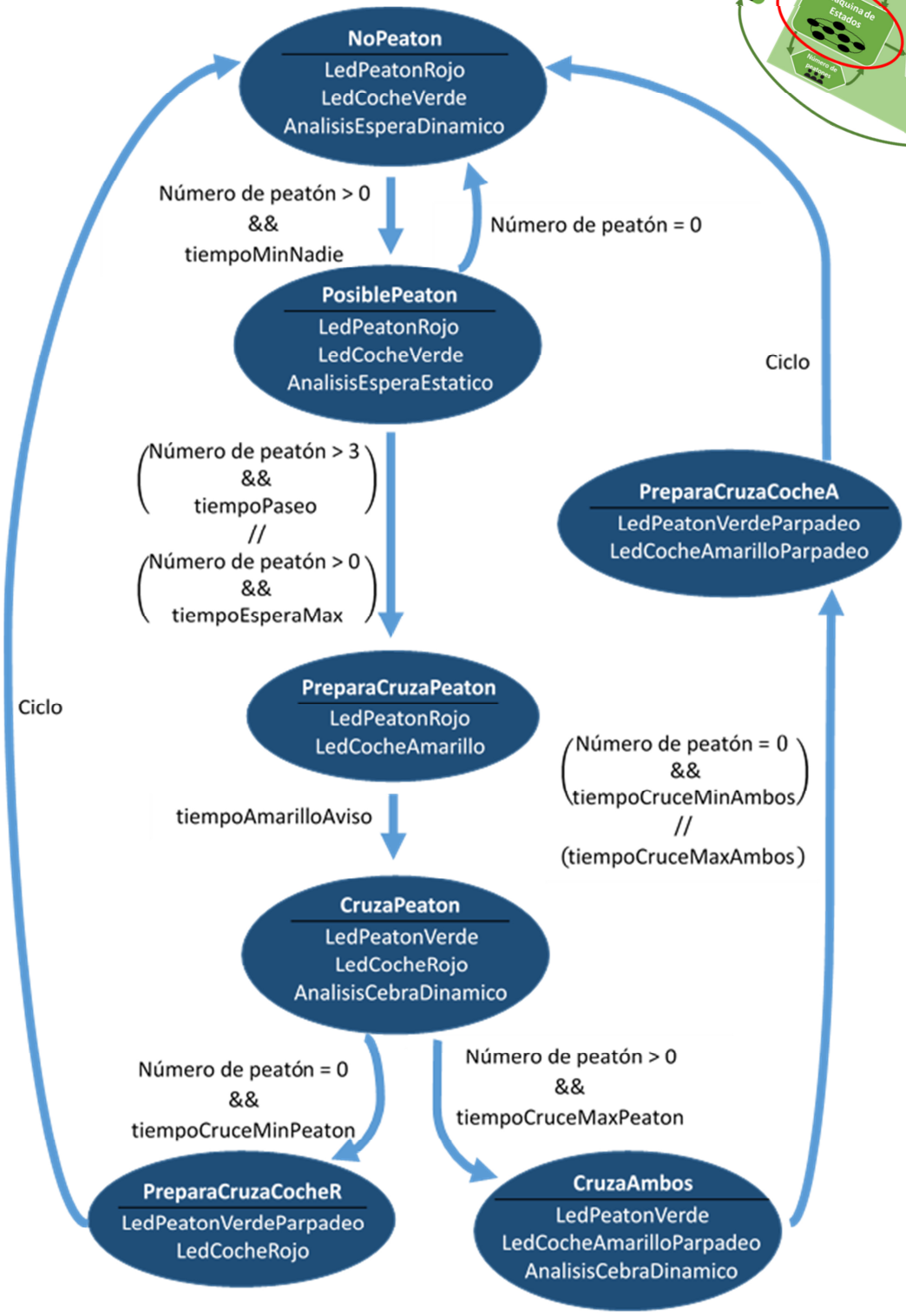


Figura 34 Máquina de estados

En las sucesivas tablas se detalla cada estado de la maquina de estados junto con una ilustración:

Estado Actual:	**NoPeaton:** Ausencia de peatón.			
Salida:	Luces del coche:	Rojo: Apagado	Amarillo: Apagado	Verde: Encendido
	Luces del peatón:	Rojo: Encendido		Verde: Apagado
Análisis de imagen:	Análisis dinámico en la zona de espera			
Condición de pasar a otro estado:	Si detecta que el número de peatón es mayor de cero y ya ha transcurrido el tiempo de transición, pasa al estado "PosiblePeaton", con el último fondo actualizado.			
Parámetros:	➢ Self.tiempoMinNadie : es el tiempo de transición. El tiempo mínimo necesario que debe estar dentro de este estado para dar valido el resultado de número de peatones. Ante un cambio brusco, el sistema de detección de peatón puede dar falsos resultados.			

```
    def NoPeaton(self,frame,t):
        estado='NoPeaton'
        fondo=None
        self.Semaforo.PeatonRojo()
        self.Semaforo.CocheVerde()
        NPeaton=self.Analisis.ConteoPeatonDinamico(frame)
        tActual=time.time()
        if NPeaton>0 and tActual-t>self.tiempoMinNadie:
            estado='PosiblePeaton'
            fondo=self.Analisis.FondoDinamico()
        return estado,fondo,tActual
```

Tabla 3 Estado NoPeaton

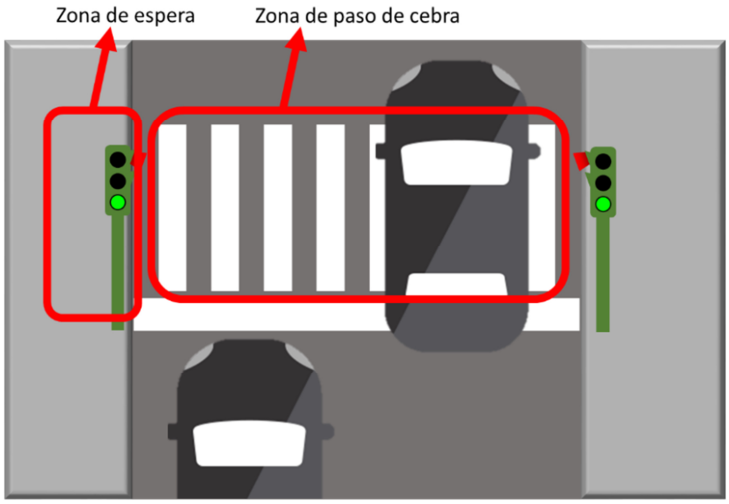


Figura 35 Ausencia de peatón, paso para los coches

Estado Actual:	**PosiblePeaton:** Posible peatón con intención de cruzar.			
Salida:	Luces del coche:	Rojo: Apagado	Amarillo: Apagado	Verde: Encendido
	Luces del peatón:	Rojo: Encendido		Verde: Apagado
Análisis de imagen:	Análisis estático en la zona de espera			
Condición de pasar a otro estado:	Si no detecta presencia de peatón, pasa al estado "NoPeaton" Si hay más de tres peatones, y el tiempo transcurrido es mayor del tiempo de paseo, pasa al estado "PreparaCruzarPeaton" Si hay peatón, y el tiempo transcurrido es mayor del tiempo máximo de espera, pasa al estado "PreparaCruzarPeaton"			
Parámetros:	➢ self.tiempoPaseo: sirve para detectar aquellos peatones que no tienen intención de cruzar ➢ self.tiempoEsperaMax: tiempo máximo que un peatón está esperando para cruzar.			

```
    def PosiblePeaton(self,frame,fondo,t):
        estado='PosiblePeaton'
        self.Semaforo.PeatonRojo()
        self.Semaforo.CocheVerde()
        NPeaton=self.Analisis.ConteoPeatonEstatico(frame,fondo)
        tActual=time.time()
        if NPeaton==0:
            estado='NoPeaton'
        elif (tActual-t>self.tiempoPaseo and NPeaton>3) or tActual-
t>self.tiempoEsperaMax:
            estado='PreparaCruzaPeaton'
        return estado, tActual
```

Tabla 4 Estado PosiblePeaton

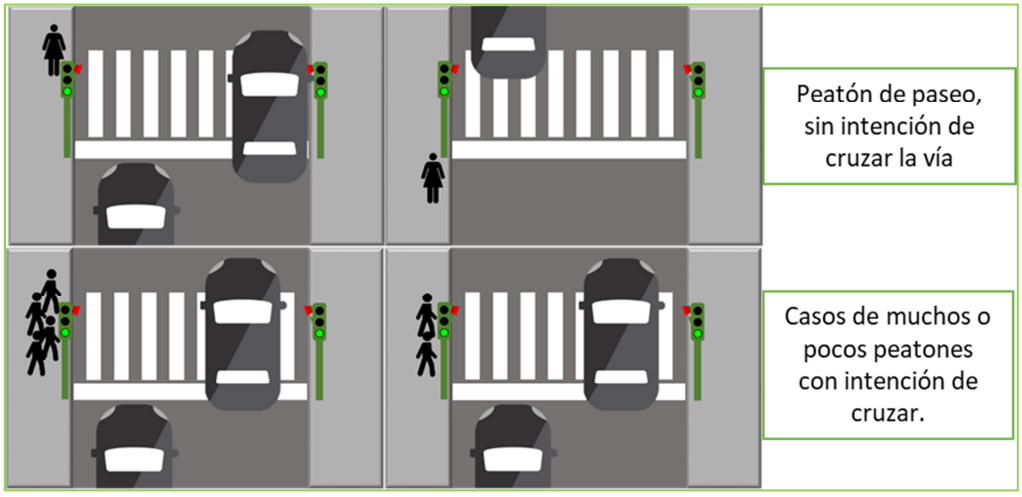


Figura 36 Posibles casos de abrir pasó al peatón

Estado Actual:	**PreparaCruzaPeaton:** Se prepara para dar el paso al peatón y avisar al conductor de frenar.			
Salida:	Luces del coche:	Rojo: Apagado	Amarillo: Encendido	Verde: Apagado
	Luces del peatón:	Rojo: Encendido		Verde: Apagado
Análisis de imagen:	Sin análisis			
Condición de pasar a otro estado:	Transcurrido el tiempo de aviso al conductor, pasa al estado "CruzaPeaton"			
Parámetros:	➢ self.tiempoAmarillo: tiempo que esta la luz amarilla del coche encendida			

```
    def PreparaCruzaPeaton(self,t):
        estado='PreparaCruzaPeaton'
        self.Semaforo.PeatonRojo()
        self.Semaforo.CocheAmarillo()
        tActual=time.time()
        if tActual-t>self.tiempoAmarillo:
            estado='CruzaPeaton'
        return estado,tActual
```

Tabla 5 Estado PreparaCruzaPeaton

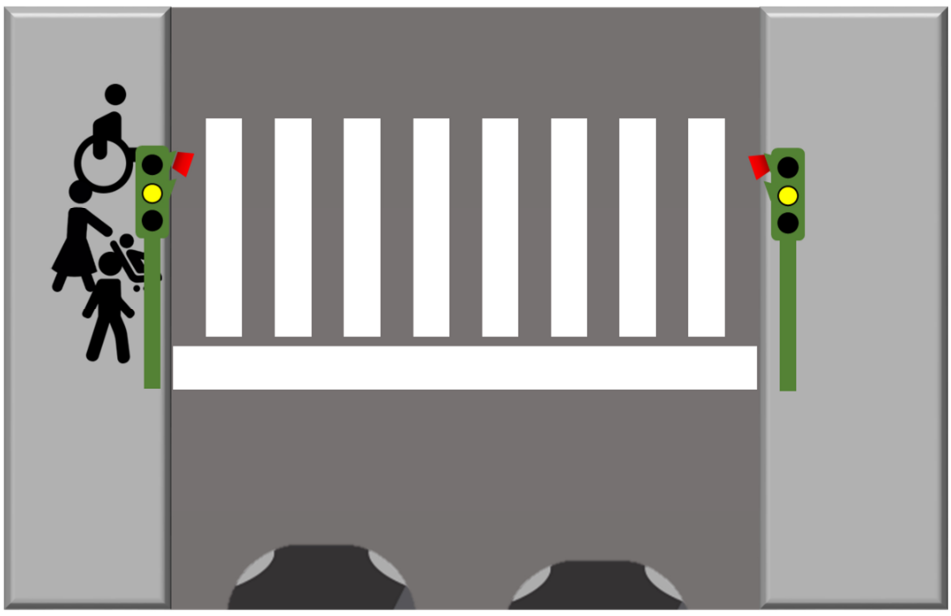


Figura 37 Ejemplo estado PreparaCruzaPeaton

Estado Actual:	**CruzaPeaton:** Coche parado, peatón cruzando.			
Salida:	Luces del coche:	Rojo: Encendido	Amarillo: Apagado	Verde: Apagado
	Luces del peatón:	Rojo: Apagado		Verde: Encendido
Análisis de imagen:	Análisis dinámico en la zona de paso de cebra			
Condición de pasar a otro estado:	Si ya ha pasado el tiempo mínimo de cruce de peatón, y no hay peatón, pasa al estado "PreparaCruzaCocheR" Cuando ya ha pasado el tiempo máximo de cruce, y aún hay peatones en cruce, se pasa al estado "CruzaAmbos"			
Parámetros:	➢ self.tiempoCruceMinPeaton: tiempo mínimo que necesita un peatón para cruzar ➢ self.tiempoCruceMaxPeaton: tiempo máximo que los coches estén parados por la presencia del peatón.			

```
def CruzaPeaton(self,frame,t):
    estado='CruzaPeaton'
    self.Semaforo.PeatonVerde()
    self.Semaforo.CocheRojo()
    NPeaton=self.AnalisisCebra.ConteoPeatonDinamico(frame)
    tActual=time.time()
    if NPeaton==0 and tActual-t>self.tiempoCruceMinPeaton:
        estado='PreparaCruzaCocheR'
    elif tActual-t>self.tiempoCruceMaxPeaton:
        estado='CruzaAmbos'
    return estado,tActual
```

Tabla 6 Estado CruzaPeaton

Figura 38 Ejemplo estado CruzaPeaton

Estado Actual:	**CruzaAmbos:** Peatón cruzando libremente, conductor cruzando con cuidado			
Salida:	Luces del coche:	Rojo: Apagado	Amarillo: Parpadeando	Verde: Apagado
	Luces del peatón:	Rojo: Apagado		Verde: Encendido
Análisis de imagen:	Análisis dinámico en la zona de paso de cebra			
Condición de pasar a otro estado:	Si ha pasado el tiempo máximo de cruce ambos, pasa al estado "PreparaCruzaCocheA" Si ya no hay peatones cruzando y ha pasado el tiempo mínimo de este estado, pasa al estado "PreparaCruzaCocheA"			
Parámetros:	➢ controlAmarillo: variable creado para controlar cuando tiene que encender la led del amarillo. ➢ self.ParpadeoAmarillo: controla el tiempo que el amarillo este encendido o apagado ➢ self. tiempoCruceMaxAmbos: tiempo máximo en este estado ➢ self.tiempoCruceMinAmbos: tiempo mínimo en este estado			

```
    def CruzaAmbos(self,frame,t,controlAmarillo):
        estado='CruzaAmbos'
        if controlAmarillo<(self.ParpadeoAmarillo/2):
            self.Semaforo.CocheAmarillo()
            controlAmarillo=controlAmarillo+1
        elif controlAmarillo<self.ParpadeoAmarillo:
            self.Semaforo.CocheNada()
            controlAmarillo=controlAmarillo+1
        if controlAmarillo==self.ParpadeoAmarillo: controlAmarillo=0
        self.Semaforo.PeatonVerde()
        NPeaton=self.AnalisisCebra.ConteoPeatonDinamico(frame)
        tActual=time.time()
        if tActual-t>self.tiempoCruceMaxAmbos or (NPeaton==0 and tActual-
t>self.tiempoCruceMinAmbos):
            estado='PreparaCruzaCocheA'
        return estado,controlAmarillo
```

Tabla 7 Estado CruzaAmbos

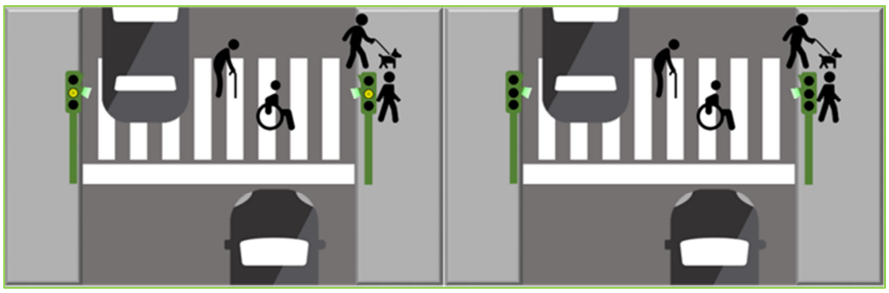


Figura 39 Ejemplo estado CruzaAmbos

Estado Actual:	**PreparaCruzaCocheR:** Prepara el cruce para los conductores, avisando a los peatones del cambio de estado, con el semáforo de coche en rojo
Salida:	Luces del coche: Rojo: Encendido; Amarillo: Apagado; Verde: Apagado Luces del peatón: Rojo: Apagado; Verde: Parpadeando
Análisis de imagen:	Sin análisis
Condición de pasar a otro estado:	Cuando termine el tiempo de aviso, pasa al estado "NoPeaton"
Parámetros:	➢ self. tiempoCiclo: tiempo de ciclo ➢ self. Nciclo: número de ciclo, número de parpadeo Cada mitad de ciclo la luz verde del peatón esta encendido, y la otra mitad apagado.

```
    def PreparaCruzaCocheR(self):
        self.Semaforo.CocheRojo()
        ciclo=0
        while ciclo<self.Nciclo:
            self.Semaforo.PeatonVerde()
            time.sleep(self.tiempoCiclo/2)
            self.Semaforo.PeatonNada()
            time.sleep(self.tiempoCiclo/2)
            ciclo=ciclo+1
        estado='NoPeaton'
        return estado
```

Tabla 8 Estado PreparaCruzaCocheR

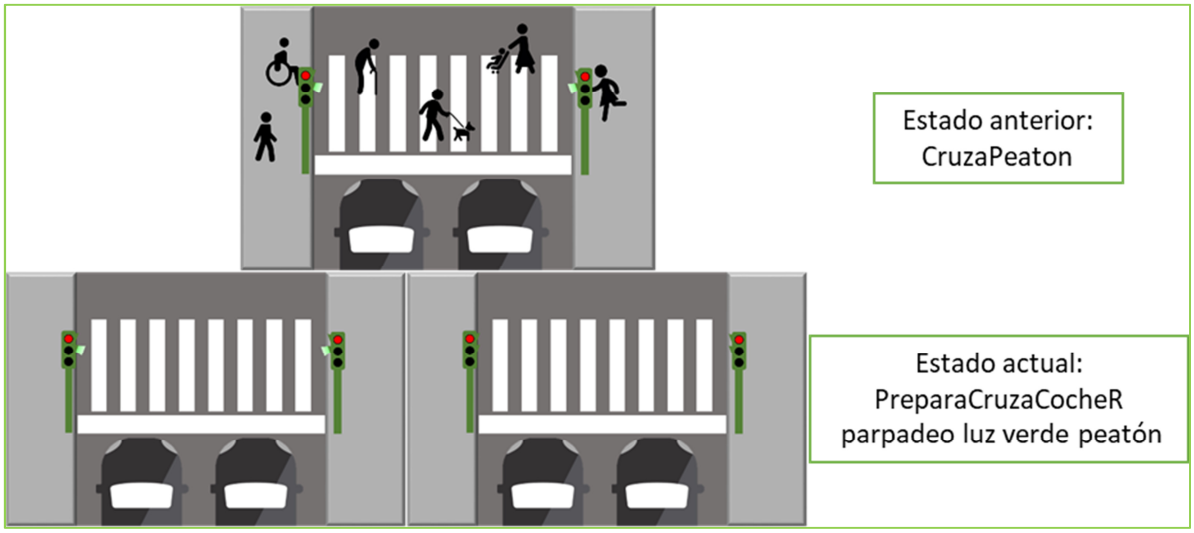


Figura 40 Ejemplo estado PreparaCruzaCocheR

Estado Actual:	**PreparaCruzaCocheA:** Prepara el cruce para los conductores, avisando a los peatones del cambio de estado, con el semáforo de coche en amarillo parpadeando.
Salida:	Luces del coche: Rojo: Apagado; Amarillo: Parpadeando; Verde: Apagado Luces del peatón: Rojo: Apagado; Verde: Parpadeando
Análisis de imagen:	Sin análisis
Condición de pasar a otro estado:	Cuando termine el tiempo de aviso, pasa al estado "NoPeaton"
Parámetros:	➢ self. tiempoCiclo: tiempo de ciclo ➢ self. Nciclo: número de ciclo, número de parpadeo ➢ Cada mitad de ciclo la luz verde del peatón y el amarillo del coche están encendidos, y la otra mitad apagados.

```
def PreparaCruzaCocheA(self):
    ciclo=0
    while ciclo<self.Nciclo:
        self.Semaforo.PeatonVerde()
        self.Semaforo.CocheAmarillo()
        time.sleep(self.tiempoCiclo/2)
        self.Semaforo.PeatonNada()
        self.Semaforo.CocheNada()
        time.sleep(self.tiempoCiclo/2)
        ciclo=ciclo+1
    estado='NoPeaton'
    return estado
```

Tabla 9 Estado PreparaCruzaCocheA

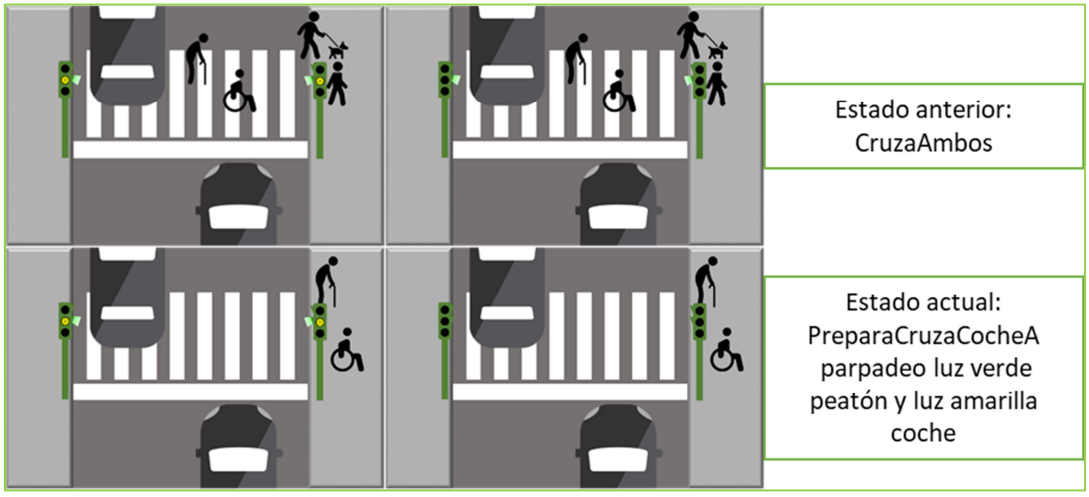


Figura 41 Ejemplo estado PreparaCruzaCocheA

7. Resultados y Conclusiones

En la figura 42 se muestra un resultado de dispositivo final:

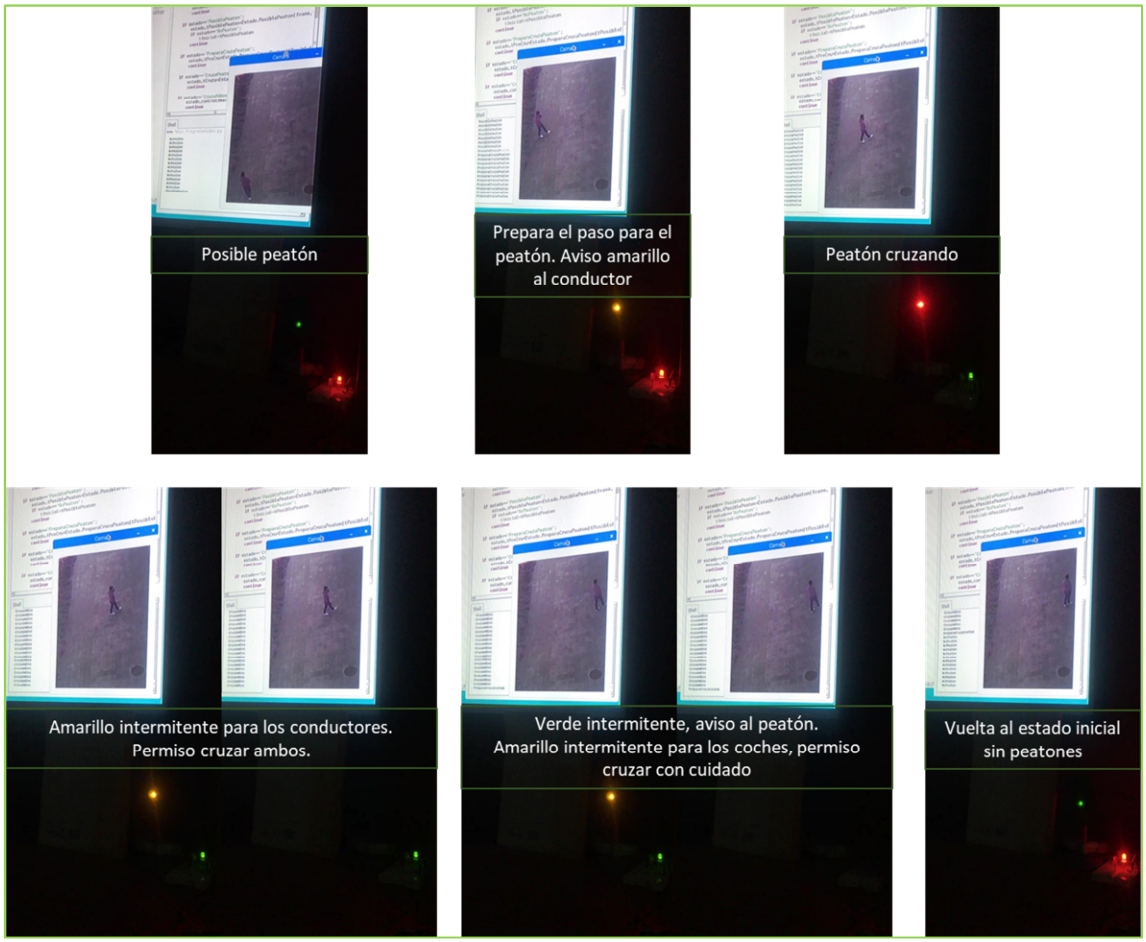


Figura 42 Resultado semáforo

El proyecto resultante es un sistema **económico** de control de semáforo a **tiempo real** con capacidad de visión artificial. Tanto el hardware como el software son de libre licencia y de bajo coste, alcanzable para todo el usuario.

A diferencia de los otros métodos de detección de personas, la técnica escogida tiene la gran ventaja de ser **simple**, sencilla, y sobre todo, **rápida**. Los tiempos de cómputos obtenidos en el procesamiento de imagen con el método de sustracción de fondo es aproximadamente 0.4segundos por imagen, mientras que con la pareja HOG y SVM ha sido de 1segundo/imagen.

La desventaja de este algoritmo del recuento de peatones es la **precisión**. No es exacto, sino que puede tener un error de una a dos personas en condiciones normales o errores más grandes si se produce cambios bruscos entre el último fotograma y el actual. Para perfeccionarlo se ha probado en hacer el conteo de peatones añadiendo un proceso de seguimiento, sí que se ha mejorado la precisión, pero a costa de aumentar el tiempo de computo al triple.

Sin embargo, a pesar de los posibles fallos que puede dar el algoritmo de visión, no afecta al funcionamiento general del sistema. Con la presencia del bloque de tiempo, que controla los tiempos máximos y mínimos en cada estado, se garantiza los movimientos entre los estados, y así, la actividad cíclica del semáforo.

Además, el sistema no necesita saber el número de peatones exactos, solo le interesa conocer si hay peatón o no, y si hay, muchos o pocos, ilustrado en la figura 43. Por lo que los errores pequeños en el recuento son despreciables.

Figura 43 Diferentes casos de número de peatones

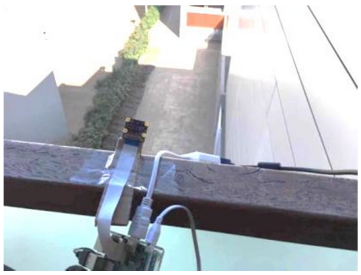


Figura 44 Grabación de video e imagen real

Con la pareja HOG y SVM se obtienen mejores resultados en la precisión, a costa de perder el anonimato de las personas, ya que necesita ver a los peatones de frente. En cambio, con el método implementado sí que se consigue el anonimato, acepta la captura de imagen desde cualquier perspectiva, y como no, desde arriba. Para simular la situación real, se ha grabado los videos e imágenes desde la segunda planta de un piso, en la figura 44 se muestra la colocación de la cámara y la placa procesadora:

Como líneas futuras se plantean:

- Implementación sobre un semáforo real. Utilizando las salidas del Raspberry Pi como señal de control para el semáforo real.
- Detección de vehículo. Aplicar los algoritmos de visión para detectar tanto vehículos como peatones.
- Información. Recopilar informaciones sobre número de peatones y vehículos, del flujo tráfico, …
- Comunicación entre varios semáforos, entre los usuarios.

El resultado de este proyecto es solamente un punto de inicio, un primer paso hacia el trafico inteligente y sostenible.

Para finalizar, se ha dado difusión parcial de este trabajo en la XI Reunión española de optoelectrónica, celebrada en Zaragoza entre los días 3 y 5 de julio de 2019, donde se presentó una comunicación en formato póster titulada “Control activo de un semáforo basado en algoritmos de visión por computador con Raspberry Pi”. En el Anexo IX se muestra una copia de la comunicación.

8. Bibliografía

Estado de arte:

[1] B. DE SCHUTTER, B. DE MOOR, *"Optimal Traffic Light Control for a Single Intersectionm"*, European Journal of Control, 4, 3, 260-276, 1998.

[2] https://es.wikipedia.org/wiki/Sem%C3%A1foro

[3] Henk TAALE, Thomas BÄCK, Mike PREUSS, A.E. EIBEN, J.M. DE GRAFF, C.A. SCHIPPERS, *"Optimizing traffic light controllers by means of evolutionary algorithms"*, Proceedings of the EUFIT'98 Conference, 1730-1734, 1998.

[4] K.J. PRABUCHANDRAN, A.N. HEMANTH KUMAR, Bhatnagar SHALABH, *"Multi-agent reinforcement learning for traffic signal control"*, International IEEE Conference on Intelligent Transportation Systems (ITSC), 2529-2534, 2014.

[5] Michael BOMMES, Adrian FAZEKAS, Tobias VOLKENHOFF, Markus OESER, *"Video Based Intelligent Transportation Systems – State of the Art and Future Development"*, Transportation Research Procedia, 14, 4495-4504, 2016.

[6] C.L. WAN, K.W. DICKINSON, "*Road Traffic Monitoring Using Image Processing—A Survey of Systems, Techniques and Applications*", IFAC Proccedings Volumens, 23, 2, 27-34, 1990.

[7] https://en.wikipedia.org/wiki/Computer_vision

[8] C.L. WAN, K.W. DICKINSON, "*Road Traffic Monitoring Using Image Processing—A Survey of Systems, Techniques and Applications*", IFAC Proccedings Volumens, 23, 2, 27-34, 1990

Hardware:

[9] https://www.programoergosum.com/cursos-online/raspberry-pi/238-control-de-gpio-con-python-en-raspberry-pi/intermitente

Software:

[10] https://thonny.org/

[11] https://en.wikipedia.org/wiki/Anaconda

[12] https://go-bees.readthedocs.io/es/develop/documentacion/3_ConceptosTeoricos.html

[13] Zoran Zivkovic, "*Improved Adaptive Gaussian Mixture Model for Background Subtraction*", In Proc. ICPR, 2004

[14] Patrick VANNOORENBERGHE, Cina MOTAMED, Jean-Marc BLOSSEVILLE, and Jack-Gérard POSTAIRE Automatic *"Pedestrian Recognition Using Real-Time Motion Analysis"*

[15] https://es.wikipedia.org/wiki/Gradiente

Universidad Zaragoza
1542

Anexo

Control automático de semáforos basado en procesado digital de imágenes sobre Raspberry Pi

Automatic control of traffic lights based on digital image processing on Raspberry Pi

Autor/es

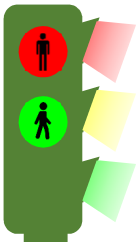
Zhuolin JIN

Director/es

Francisco José TORCAL-MILLA
Ana LOPEZ-TORRES

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

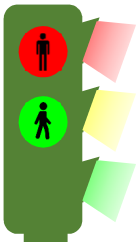
Año 2019



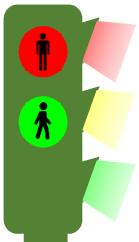
Anexo I
Código Programa Main.py

Autor:
Zhuolin JIN

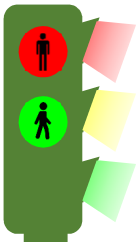
Directores:
Francisco José TORCAL-MILLA
Ana LOPEZ-TORRES



```
from Parametros import *
from picamera.array import PiRGBArray
from picamera import PiCamera


#Crear e inicializar Camara:
Camara=PiCamera()
rawCapture = PiRGBArray(Camara)
Camara.resolution = (640, 480)
rawCapture = PiRGBArray(Camara, size=(640, 480))
time.sleep(0.1)


estado='NoPeaton'
controlAmarillo=0
tInicial=time.time()

for image in Camara.capture_continuous(rawCapture, format="bgr",use_video
_port=True):
    frame = image.array

    #Mostrar y Salir con 's'
    print(estado)
    frameAnalizado=frame[y1:y2,x1:x2]
    cv2.imshow('Camara',frameAnalizado)
    k = cv2.waitKey(30) & 0xff
    if k == ord("s"):
        break

    #Maquina de Estados:
    if estado=='NoPeaton':
        estado,fondo,tNoPeaton=Estado.NoPeaton(frame,tInicial)
        continue

    if estado=='PosiblePeaton':
        estado,tPosiblePeaton=Estado.PosiblePeaton(frame,fondo,tNoPeaton)
        if estado=='NoPeaton':
            tInicial=tPosiblePeaton
        continue

    if estado=='PreparaCruzaPeaton':
        estado,tPreCru=Estado.PreparaCruzaPeaton(tPosiblePeaton)
        continue
```

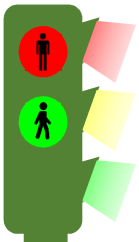
```
    if estado=='CruzaPeaton':
        estado,tCruza=Estado.CruzaPeaton(frame,tPreCru)
        continue

    if estado=='CruzaAmbos':
        estado,controlAmarillo=Estado.CruzaAmbos(frame,tCruza,controlAmar
illo)
        continue

    if estado=='PreparaCruzaCocheA':
        estado=Estado.PreparaCruzaCocheA()
        tInicial=time.time()
        continue

    if estado=='PreparaCruzaCocheR':
        estado=Estado.PreparaCruzaCocheR()
        tInicial=time.time()
        continue

        #Borrar la secuencia, para el siguiente frame
    rawCapture.truncate(0)


# Liberar cámara, cerrar ventanas, limpiarLED
Camara.stop_preview()
cv2.destroyAllWindows()
LED.LimpiarLED()
```

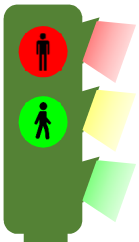
Anexo II
Código Parametros.py

```
#################IMPORTAR################
import cv2
import time
import numpy as np
from ManchaContarPeaton import ContarPeatonMancha
from LED import LED
from Estados import EstadosSemaforo
#################IMPORTAR################


#################LED################
LED=LED()
#################LED################


#################Analisis Imagen Zona Espera################
    #Crear objeto
Analisis=ContarPeatonMancha()
    #Angulo de Rotar y Zona de Recorte:
AnguloRota=0
y1=250
y2=677
x1=780
x2=920
Analisis.ZonaInteres(AnguloRota,y1,y2,x1,x2)
    #Elementos de las opMorfologicas
ElementoEro = np.ones((5,5),np.uint8)
ElementoDila = np.ones((20,15),np.uint8)
iteEroD=1
iteDilaD=1
iteEro=1
iteDila=1
Analisis.ParametroMorfologico(ElementoEro,ElementoDila,iteEroD,iteDilaD,i
teEro,iteDila)
    #Areas de las manchas:
AreaMinD=200
AreaMin=500
AreaEstandar=2700
Analisis.AreaMancha(AreaMinD,AreaMin,AreaEstandar)
    #Metodo fondo dinamico:
MOG2 = cv2.createBackgroundSubtractorMOG2(history=70, varThreshold=16, de
tectShadows=False)
Analisis.MetodoDinamico(MOG2)
```

```
    #ParametroSustEstatico
Gau=13
LimiteBi=27
Analisis.ParametroSustEstatico(Gau,LimiteBi)
################Analisis Imagen Zona Espera################


################Analisis Imagen Zona Paso de Cebra################
    #Crea objeto
AnalisisCebra=ContarPeatonMancha()
    #Angulo de Rotar y Zona de Recorte:
AnguloRotaCebra=0
y1Cebra=250
y2Cebra=677
x1Cebra=920
x2Cebra=1100
AnalisisCebra.ZonaInteres(AnguloRotaCebra,y1Cebra,y2Cebra,x1Cebra,x2Cebra)
    #Elementos de las opMorfologicas
ElementoEro = np.ones((5,5),np.uint8)
ElementoDila = np.ones((20,15),np.uint8)
iteEroD=1
iteDilaD=3
iteEro=1
iteDila=1
AnalisisCebra.ParametroMorfologico(ElementoEro,ElementoDila,iteEroD,iteDi
laD,iteEro,iteDila)
    #Areas de las manchas:
AreaMinD=200
AreaMin=500
AreaEstandar=2700
AnalisisCebra.AreaMancha(AreaMinD,AreaMin,AreaEstandar)
    #Metodo fondo dinamico:
MOG2 = cv2.createBackgroundSubtractorMOG2(history=70, varThreshold=16, de
tectShadows=False)
AnalisisCebra.MetodoDinamico(MOG2)
################Analisis Imagen Zona Paso de Cebra################
```

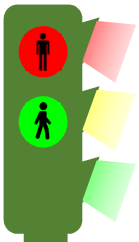


```
################Semaforo################
    #Crea objeto
Estado=EstadosSemaforo(LED,Analisis,AnalisisCebra)
    #tiempo posible
tiempoMinNadie=1
tiempoPaseo=20
tiempoEsperaMax=30
tiempoAvisoAmarillo=3
Estado.tPosible(tiempoMinNadie,tiempoPaseo,tiempoEsperaMax,tiempoAvisoAma
rillo)
    #tiempo cruce peaton
tiempoCruceMinPeaton=9
tiempoCruceMaxPeaton=11
Estado.tCrucePeaton(tiempoCruceMinPeaton,tiempoCruceMaxPeaton)
    #tiempo cruce ambos
ParpadeoAmarillo=8
tiempoCruceMinAmbos=5
tiempoCruceMaxAmbos=25
Estado.tCruceAmbos(ParpadeoAmarillo,tiempoCruceMinAmbos,tiempoCruceMaxAmb
os)
    #parpadeo aviso peaton
Nciclo=10
tiempoCiclo=0.8
Estado.tParpadeo(Nciclo,tiempoCiclo)
################Semaforo################
```

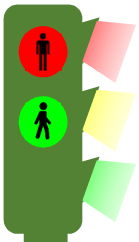



Anexo III
Código Estados.py

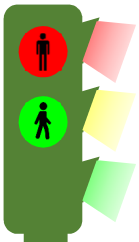
```
import cv2
import time
import numpy as np
from ManchaContarPeaton import ContarPeatonMancha
from LED import LED

class EstadosSemaforo:
    def __init__(self,LED,Analisis,AnalisisCebra):
        self
        self.Semaforo=LED
        self.Analisis=Analisis
        self.AnalisisCebra=AnalisisCebra

    def tPosible(self,tiempoMinNadie,tiempoPaseo,tiempoEsperaMax,tiempoAm
arillo):
        self.tiempoMinNadie=tiempoMinNadie
        self.tiempoPaseo=tiempoPaseo
        self.tiempoEsperaMax=tiempoEsperaMax
        self.tiempoAmarillo=tiempoAmarillo

    def tCrucePeaton(self,tiempoCruceMinPeaton,tiempoCruceMaxPeaton):
        self.tiempoCruceMinPeaton=tiempoCruceMinPeaton
        self.tiempoCruceMaxPeaton=tiempoCruceMaxPeaton

    def tCruceAmbos(self,ParpadeoAmarillo,tiempoCruceMinAmbos,tiempoCruce
MaxAmbos):
        self.ParpadeoAmarillo=ParpadeoAmarillo
        self.tiempoCruceMinAmbos=tiempoCruceMinAmbos
        self.tiempoCruceMaxAmbos=tiempoCruceMaxAmbos
    def tParpadeo(self,Nciclo,tiempoCiclo):
        self.Nciclo=Nciclo
        self.tiempoCiclo=tiempoCiclo

    def NoPeaton(self,frame,t):
        estado='NoPeaton'
        fondo=None
        self.Semaforo.PeatonRojo()
        self.Semaforo.CocheVerde()
        NPeaton=self.Analisis.ConteoPeatonDinamico(frame)
        tActual=time.time()
        if NPeaton>0 and tActual-t>self.tiempoMinNadie:
            estado='PosiblePeaton'
            fondo=self.Analisis.FondoDinamico()
        return estado,fondo,tActual
```



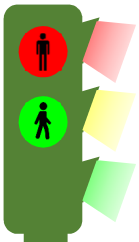
```
    def PosiblePeaton(self,frame,fondo,t):
        estado='PosiblePeaton'
        self.Semaforo.PeatonRojo()
        self.Semaforo.CocheVerde()
        NPeaton=self.Analisis.ConteoPeatonEstatico(frame,fondo)
        tActual=time.time()
        if NPeaton==0:
            estado='NoPeaton'
        elif (tActual-t>self.tiempoPaseo and NPeaton>3) or tActual-
t>self.tiempoEsperaMax:
            estado='PreparaCruzaPeaton'
        return estado, tActual



    def PreparaCruzaPeaton(self,t):
        estado='PreparaCruzaPeaton'
        self.Semaforo.PeatonRojo()
        self.Semaforo.CocheAmarillo()
        tActual=time.time()
        if tActual-t>self.tiempoAmarillo:
            estado='CruzaPeaton'
        return estado,tActual



    def CruzaPeaton(self,frame,t):
        estado='CruzaPeaton'
        self.Semaforo.PeatonVerde()
        self.Semaforo.CocheRojo()
        NPeaton=self.AnalisisCebra.ConteoPeatonDinamico(frame)
        tActual=time.time()
        if NPeaton==0 and tActual-t>self.tiempoCruceMinPeaton:
            estado='PreparaCruzaCocheR'
        elif tActual-t>self.tiempoCruceMaxPeaton:
            estado='CruzaAmbos'
        return estado,tActual
```

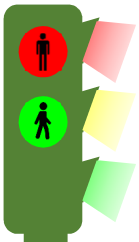
```
    def CruzaAmbos(self,frame,t,controlAmarillo):
        estado='CruzaAmbos'
        if controlAmarillo<(self.ParpadeoAmarillo/2):
            self.Semaforo.CocheAmarillo()
            controlAmarillo=controlAmarillo+1
        elif controlAmarillo<self.ParpadeoAmarillo:
            self.Semaforo.CocheNada()
            controlAmarillo=controlAmarillo+1
        if controlAmarillo==self.ParpadeoAmarillo: controlAmarillo=0
        self.Semaforo.PeatonVerde()
        NPeaton=self.AnalisisCebra.ConteoPeatonDinamico(frame)
        tActual=time.time()
        if tActual-t>self.tiempoCruceMaxAmbos or (NPeaton==0 and tActual-
t>self.tiempoCruceMinAmbos):
            estado='PreparaCruzaCocheA'
        return estado,controlAmarillo


    def PreparaCruzaCocheR(self):
        self.Semaforo.CocheRojo()
        ciclo=0
        while ciclo<self.Nciclo:
            self.Semaforo.PeatonVerde()
            time.sleep(self.tiempoCiclo/2)
            self.Semaforo.PeatonNada()
            time.sleep(self.tiempoCiclo/2)
            ciclo=ciclo+1
        estado='NoPeaton'
        return estado


    def PreparaCruzaCocheA(self):
        ciclo=0
        while ciclo<self.Nciclo:
            self.Semaforo.PeatonVerde()
            self.Semaforo.CocheAmarillo()
            time.sleep(self.tiempoCiclo/2)
            self.Semaforo.PeatonNada()
            self.Semaforo.CocheNada()
            time.sleep(self.tiempoCiclo/2)
            ciclo=ciclo+1
        estado='NoPeaton'
        return estado
```

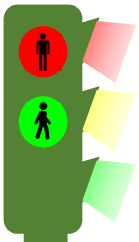
Anexo IV
Código LED.py



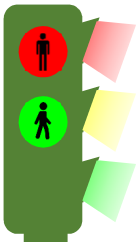
```
import RPi.GPIO as GPIO
import time


class LED:
    def __init__(self):
        self
        GPIO.setmode(GPIO.BOARD)
        GPIO.setwarnings(False)
        GPIO.setup(16,GPIO.OUT)#PeatonRojo
        GPIO.setup(18,GPIO.OUT)#PeatonVerde
        GPIO.setup(11,GPIO.OUT)#CocheRojo
        GPIO.setup(13,GPIO.OUT)#CocheAmarillo
        GPIO.setup(15,GPIO.OUT)#CocheVerde
        GPIO.output(16,GPIO.LOW)#PeatonRojo
        GPIO.output(18,GPIO.LOW)#PeatonVerde
        GPIO.output(11,GPIO.LOW)#CocheRojo
        GPIO.output(13,GPIO.LOW)#CocheAmarillo
        GPIO.output(15,GPIO.LOW)#CocheVerde



    def PeatonRojo(self):
        GPIO.output(16,GPIO.HIGH)#PRojo
        GPIO.output(18,GPIO.LOW)#PVerde
    def PeatonVerde(self):
        GPIO.output(16,GPIO.LOW)#PRojo
        GPIO.output(18,GPIO.HIGH)#PVerde
    def PeatonNada(self):
        GPIO.output(16,GPIO.LOW)#PRojo
        GPIO.output(18,GPIO.LOW)#PVerde


    def CocheRojo(self):
        GPIO.output(11,GPIO.HIGH)#CRojo
        GPIO.output(13,GPIO.LOW)#CAmarillo
        GPIO.output(15,GPIO.LOW)#CVerde
    def CocheAmarillo(self):
        GPIO.output(11,GPIO.LOW)#CRojo
        GPIO.output(13,GPIO.HIGH)#CAmarillo
        GPIO.output(15,GPIO.LOW)#CVerde
    def CocheNada(self):
        GPIO.output(11,GPIO.LOW)#CRojo
        GPIO.output(13,GPIO.LOW)#CAmarillo
        GPIO.output(15,GPIO.LOW)#CVerde
```

```
    def CocheVerde(self):
        GPIO.output(11,GPIO.LOW)#CRojo
        GPIO.output(13,GPIO.LOW)#CAmarillo
        GPIO.output(15,GPIO.HIGH)#CVerde


    def LimpiarLED(self):
        GPIO.cleanup()
```

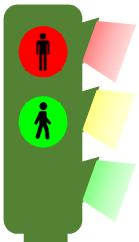



Anexo V
Código
ManchaContarPeaton.py

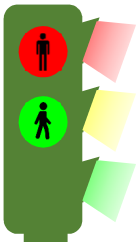


```
#Importación de librerías
import cv2
import math


class ContarPeatonMancha:
    def __init__(self):
        self

    def ZonaInteres(self,angulo,y1,y2,x1,x2):
        self.angulo=angulo
        self.y1=y1
        self.y2=y2
        self.x1=x1
        self.x2=x2

    def MetodoDinamico(self,metodoDinamico):
        self.metodoDinamico=metodoDinamico

    def ParametroMorfologico(self,ElementoEro,ElementoDila,iteEroD,iteDil
aD,iteEro,iteDila):
        self.ElementoEro=ElementoEro
        self.ElementoDila=ElementoDila
        self.iteEroD=iteEroD
        self.iteDilaD=iteDilaD
        self.iteEro=iteEro
        self.iteDila=iteDila

    def AreaMancha(self,AreaMinD,AreaMin,AreaEstandar):
        self.AreaMinD=AreaMinD
        self.AreaMin=AreaMin
        self.AreaEstandar=AreaEstandar

    def ParametroSustEstatico(self,Gau,LimiteBi):
        self.Gau=Gau
        self.LimiteBi=LimiteBi

    def FiltroZonaInteres(self,img):
        img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        rows,cols= img.shape
        M=cv2.getRotationMatrix2D((cols/2,rows/2),self.angulo,1)
        dst=cv2.warpAffine(img,M,(cols,rows))
        dstRe=dst[self.y1:self.y2,self.x1:self.x2]
        return dstRe
```


```
    def ManipulacionImagenBinaria(self,imgBinaria,ElemEro,ElemDila,itEro,
itDila):
        imgEro = cv2.erode(imgBinaria,ElemEro,iterations=itEro)
        imgDila = cv2.dilate(imgEro,ElemDila,iterations=itDila)
        imgMancha = imgDila.copy()
        return imgMancha



    def BuscaManchaContorno(self,imgMancha,AreaMinimaMancha,AreaEstandar):
        im, contornos, hierarchy = cv2.findContours(imgMancha,cv2.RETR_TR
EE,cv2.CHAIN_APPROX_SIMPLE)
        Npeaton=0
        AreaManchaTotal=0
        for c in contornos:
            AreaMancha=cv2.contourArea(c)
            if AreaMancha < AreaMinimaMancha: # Eliminamos los contornos
más pequeños
                continue
            AreaManchaTotal=AreaManchaTotal+AreaMancha
        Npeaton = math.ceil(AreaManchaTotal/AreaEstandar)
        return Npeaton



    def SustraccionFondoEstatico(self,frame,fondo):
        FrameGau = cv2.GaussianBlur(frame, (self.Gau, self.Gau), 0)
        FondoGau = cv2.GaussianBlur(fondo, (self.Gau, self.Gau), 0)
        resta = cv2.absdiff(FondoGau, FrameGau)
        imgBinaria = cv2.threshold(resta, self.LimiteBi, 255, cv2.THRESH_
BINARY)[1]
        return imgBinaria



    def AplicarMetodoDinamico(self,frame):
        imgBinaria = self.metodoDinamico.apply(frame,learningRate=-1)
        return imgBinaria



    def FondoDinamico(self):
        fondo=self.metodoDinamico.getBackgroundImage()
        return fondo
```



```
    def ConteoPeatonDinamico(self,frame):
        #Zono de Interes
        imgInteres=self.FiltroZonaInteres(frame)
        #Sustraccion de Fondo
        imgBinaria = self.AplicarMetodoDinamico(imgInteres)
        #Manipulacion
        imgMancha = self.ManipulacionImagenBinaria(imgBinaria,self.Elemen
toEro,self.ElementoDila,self.iteEroD,self.iteDilaD)
        #Conteo
        Npeaton = self.BuscaManchaContorno(imgMancha,self.AreaMinD,self.A
reaEstandar)
        return Npeaton


    def ConteoPeatonEstatico(self,frame,fondo):
        #Zono de Interes
        imgInteres=self.FiltroZonaInteres(frame)
        #Sustraccion de Fondo
        imgBinaria = self.SustraccionFondoEstatico(imgInteres,fondo)
        #Manipulacion
        imgMancha = self.ManipulacionImagenBinaria(imgBinaria,self.Elemen
toEro,self.ElementoDila,self.iteEro,self.iteDila)
        #Conteo
        Npeaton = self.BuscaManchaContorno(imgMancha,self.AreaMin,self.Ar
eaEstandar)
        return Npeaton
```

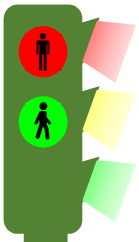



Anexo VI
Código Programa Cámara

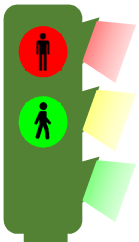
Autor:
Zhuolin JIN

Directores:
Francisco José TORCAL-MILLA
Ana LOPEZ-TORRES

```
import cv2
import time
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera


#Crear e inicializar Camara:
Camara=PiCamera()
rawCapture = PiRGBArray(Camara)
Camara.resolution = (640, 480)
rawCapture = PiRGBArray(Camara, size=(640, 480))
time.sleep(0.1)

t1Max=0
t=0
for frame in Camara.capture_continuous(rawCapture, format="bgr",use_video
_port=True):
    image = frame.array
    t1=time.time()-t
    if t1>t1Max and t!=0:
        t1Max=t1

    cv2.imshow('Camara',image)
    #Salir con 's'
    k = cv2.waitKey(30) & 0xff
    if k == ord("s"):
            break

    #Borrar la secuencia, para el siguiente frame
    t=time.time()
    rawCapture.truncate(0)

# Liberar cámara y cerrar ventanas
Camara.stop_preview()
cv2.destroyAllWindows()
print("LeerImagen cuesta: %.5f seg" %(t1Max))
```

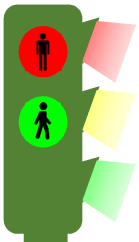


Anexo VII
Código HOG y SVM

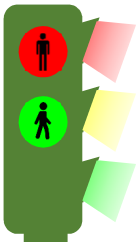


```
import cv2
from imutils.object_detection import non_max_suppression
import numpy as np

class ContarPeatonHOG:
    def __init__(self):
        self.Descriptor=cv2.HOGDescriptor()
        self.Descriptor.setSVMDetector(cv2.HOGDescriptor_getDefaultPeople
Detector())

    def ParametroContador(self,VentanaX,VentanaY,RellenoX,RellenoY,Escala
,UmbralNMS):
        self.VentanaX=VentanaX
        self.VentanaY=VentanaY
        self.RellenoX=RellenoX
        self.RellenoY=RellenoY
        self.Escala=Escala
        self.UmbralNMS=UmbralNMS

    def ConteoPeatonHOG(self,image):
        (rects, weights) = self.Descriptor.detectMultiScale(image, winStr
ide=(self.VentanaX, self.VentanaY),    padding=(self.RellenoX, self.Relle
noY), scale=self.Escala)
        #Non-maxima suppression:
        rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
        pick = non_max_suppression(rects, probs=None, overlapThresh=self.
UmbralNMS)
        return len(pick)
```

```
import cv2
import time
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera
from imutils.object_detection import non_max_suppression
from HOGContarPeaton import ContarPeatonHOG

#Crear e inicializar Camara:
Camara=PiCamera()
rawCapture = PiRGBArray(Camara)
Camara.resolution = (640, 480)
rawCapture = PiRGBArray(Camara, size=(640, 480))
time.sleep(0.1)
```

```
#Crear e parametrizar HOG:
Analisis=ContarPeatonHOG()
VentanaX=4
VentanaY=4
RellenoX=8
RellenoY=8
Escala=1.05
UmbralNMS=0.65
Analisis.ParametroContador(VentanaX,VentanaY,RellenoX,RellenoY,Escala,Umb
ralNMS)

t1Max=0
for frame in Camara.capture_continuous(rawCapture, format="bgr",use_video
_port=True):
    image = frame.array

    #Detectar Peaton:
    t=time.time()
    Npeaton=Analisis.ConteoPeatonHOG(image)
    t1=time.time()-t
    if t1>t1Max:
        t1Max=t1

    cv2.imshow('Camara',image)
    print("[INFO] hay {} personas".format(Npeaton))
    #Borrar la secuencia, para el siguiente frame
    rawCapture.truncate(0)

    #Salir con 's'
    k = cv2.waitKey(30) & 0xff
    if k == ord("s"):
            break

# Liberar cámara y cerrar ventanas
Camara.stop_preview()
cv2.destroyAllWindows()
print("ContadorHOG cuesta: %.5f seg" %(t1Max))
```

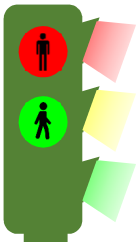



Anexo VIII
Código Seguimiento

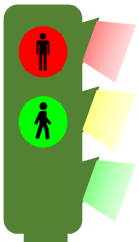
Fuente:

https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

```
from pyimagesearch.centroidtracker import CentroidTracker
from pyimagesearch.trackableobject import TrackableObject
import numpy as np
import time
import cv2
import math
#Video Entrada
cap = cv2.VideoCapture('02.h264')

y1=240
y2=677
x1=450
x2=610

##Inicilizar tamano video
W = None
H = None
L = 300
#Seguimiento
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = {}

#Numero de subida y bajada
totalDown = 0
totalUp = 0

#Elementos de las opMorfologicas
ElementoEro = np.ones((5,5),np.uint8)
ElementoDila = np.ones((15,15),np.uint8)
#Crear metodo eliminaFondo
EliminaFondo = cv2.createBackgroundSubtractorMOG2(history=70, varThreshol
d=16, detectShadows=True)
#Funcion manipulacion
def ManipulacionImagenBinaria(imgBinaria,ElemEro,ElemDila,itEro,itDila):
    imgEro = cv2.erode(imgBinaria,ElemEro,iterations=itEro)
    imgDila = cv2.dilate(imgEro,ElemDila,iterations=itDila)
    imgMancha = imgDila.copy()
    return imgMancha

tMax=0
while True:
```

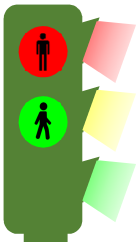


```
    tInicio=time.time()
    # Leer el siguiente frame
    Video, frame = cap.read()
    # Si ha llegado al final del vídeo, salir
    if not Video:
        break
    frame=frame[y1:y2,x1:x2]
    #GuardaManchas
    rects=[]
    #Tamano frame
    if W is None or H is None:
        (H, W) = frame.shape[:2]
    #Dibuja Linea
    cv2.line(frame, (0, L), (W, L), (0, 255, 255), 2)

    #Sustraccion de Fondo
    imgBinariaSombra = EliminaFondo.apply(frame,learningRate=-1)
    imgBinaria = cv2.threshold(imgBinariaSombra, 128, 255, cv2.THRESH_BIN
ARY)[1]
    #Manipulacion
    imgMancha = ManipulacionImagenBinaria(imgBinaria,ElementoEro,Elemento
Dila,1,3)
    #BuscaMancha
    im, contornos, hierarchy = cv2.findContours(imgMancha,cv2.RETR_TREE,c
v2.CHAIN_APPROX_SIMPLE)
    for cm in contornos:
        AreaMancha=cv2.contourArea(cm)
        if AreaMancha < 3000: # Eliminamos los contornos más pequeños
            continue
        # DibujaRectangulo
        (startX, startY, w, h) = cv2.boundingRect(cm)
        endX=startX+w
        endY=startY+h
        peso=math.ceil(AreaMancha/4000)
        cuadro=(startX,startY,endX,endY,peso)
        rects.append(cuadro)
        cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 255, 0),
 2)

    objects=ct.update(rects)
    # loop over the tracked objects
    for (objectID, centroid) in objects.items():
            # check to see if a trackable object exists for the current
            # object ID
            to = trackableObjects.get(objectID, None)
```

```
            # if there is no existing trackable object, create one
            if to is None:
                    to = TrackableObject(objectID, centroid)

            # otherwise, there is a trackable object so we can utilize it
            # to determine direction
            else:
                    # the difference between the y-
coordinate of the *current*
                    # centroid and the mean of *previous* centroids will
tell
                    # us in which direction the object is moving (negativ
e for
                    # 'up' and positive for 'down')
                    y = [c[1] for c in to.centroids]
                    direction = centroid[1] - np.mean(y)
                    to.centroids.append(centroid)

                    # check to see if the object has been counted or not
                    if not to.counted:
                            # if the direction is negative (indicating th
e object
                            # is moving up) AND the centroid is above the
 center
                            # line, count the object
                            if direction < -20 and centroid[1] < L:
                                totalUp += centroid[2]
                                to.counted = True


                            # if the direction is positive (indicating th
e object
                            # is moving down) AND the centroid is below t
he
                            # center line, count the object
                            elif direction >20 and centroid[1] > L:
                                totalDown += centroid[2]
                                to.counted = True

            # store the trackable object in our dictionary
            trackableObjects[objectID] = to

            # draw both the ID of the object and the centroid of the
            # object on the output frame
            text = "ID {}".format(objectID)
```


```
            cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0),
 -1)

    # construct a tuple of information we will be displaying on the
    # frame
    info = [
            ("Up", totalUp),
            ("Down", totalDown),
    ]

    # loop over the info tuples and draw them on our frame
    for (i, (k, v)) in enumerate(info):
            text = "{}: {}".format(k, v)
            cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)


    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("s"):
            break

    tCiclo=time.time()-tInicio
    if tCiclo>tMax:
        tMax=tCiclo


# close any open windows
cv2.destroyAllWindows()
print("tiempo ciclo es  %.3f seg" %(tMax))
```



Anexo IX

Control activo de un semáforo basado en algoritmos de visión por computador con Raspberry Pi

Autor:
Zhuolin JIN

Directores:
Francisco José TORCAL-MILLA
Ana LOPEZ-TORRES

Control activo de un semáforo basado en algoritmos de visión por computador con Raspberry Pi

Computer vision-based active control of a traffic light with Raspberry Pi

Zhuolin JIN, Francisco José TORCAL-MILLA [1, 3, 4], Ana LOPEZ-TORRES [2, 3], José Antonio GOMEZ-PEDRERO [4, 5]

1. Departamento de Física Aplicada, Universidad de Zaragoza, C/ Pedro Cerbuna 12, 50009, Zaragoza (España).
2. Departamento de Ingeniería Electrónica y Comunicaciones, Escuela Universitaria Politecnica de Teruel, Calle Atarazana 2, 44003 Teruel (España)
3. Instituto de Investigación en Ingeniería de Aragón (i3a), Grupo de Tecnologías Ópticas Láser, Universidad de Zaragoza, Zaragoza (España)
4. Applied Optics Complutense Group, Universidad Complutense de Madrid, Madrid (España)
5. Departamento de Óptica, Facultad de Óptica y Optometría, Universidad Complutense de Madrid, C/ Arcos de Jalón 118, 28037, Madrid (España).

Persona de contacto: Francisco José Torcal-Milla (fjtorcal@unizar.es)

RESUMEN:

El surgimiento en los últimos años de miniordenadores de bajo coste con capacidad de cálculo significativa ha propiciado el desarrollo de dispositivos baratos y portátiles con multitud de aplicaciones tecnológicas y científicas. En este trabajo se muestra el desarrollo de un dispositivo compacto y barato para el control activo de un semáforo utilizando como placa de control la Raspberry Pi. El dispositivo se basa en algoritmos de visión por computador para determinar cuántas personas hay en cada momento esperando para cruzar y el tiempo que llevan esperando. Posteriormente, utiliza una máquina de estados para determinar si debe abrirse o no el semáforo y cuánto tiempo debe permanecer abierto. En este aspecto, también es sensible a la velocidad de avance de los peatones, manteniendo el semáforo abierto más tiempo si es necesario, dentro de un cierto margen máximo.

Palabras clave: visión por computador, máquina de estados, Raspberry Pi, control activo, semáforo, tráfico.

ABSTRACT:

The emergence of low cost minicomputers with acceptable computing velocity has led to the development of cheap and portable devices with multitude of applications. In this work, we show the development of a low cost portable device for controlling a traffic light using a Raspberry Pi board. The algorithm is based on computer vision and is able to determine how many people are waiting to cross the street and how long they have been. Then it uses a state machine to determine whether the traffic light should be on or off and how long it should remain on. Related to this aspect, the algorithm is sensitive to the pedestrian velocity and allows the traffic light to be on for a longer time in case it is necessary.

Key words: Computer vision, state machine, Raspberry Pi, active control, traffic light, traffic.

1.- Introducción

Las personas cada día nos movemos más, bien sea en vehículos particulares, transporte público o a pie. Es por esto que el volumen de tráfico y su complejidad han aumentado de forma desorbitada en las últimas décadas. Para atender a esta complejidad ha aparecido una creciente demanda de sistemas de control activo y adaptativo del tráfico tanto rodado como pedestre, [1]. La motivación principal de este tipo de sistemas es dinamizar el tráfico de forma que no se produzcan atascos o que, de producirse, afecten en menor medida a los usuarios de la vía pública. Está demostrado que con un sistema de control de semáforos pasivo el tiempo perdido tanto por conductores como por peatones en un trayecto promedio es elevado. Este hecho mejora si aplicamos al control semafórico algoritmos adaptativos como lógica fuzzy, algoritmos evolutivos [2], algoritmos de reforzamiento [3], etc. Gran parte de este tipo de algoritmos se basan en primera instancia en la captura de imágenes del tráfico y su análisis y procesado en tiempo real, [4]. Es en este aspecto donde tienen cabida los algoritmos de visión por computador e inteligencia artificial tan ampliamente utilizados en muchos otros ámbitos científicos y tecnológicos, [5].

Si nos centramos en la detección de peatones dentro del contexto de la gestión del tráfico, esta se ha centrado últimamente en el perfeccionamiento de los vehículos autónomos [6] donde se han desarrollado complejos algoritmos basados en *machine learning* [7, 8]. En nuestro caso, en el que se busca el funcionamiento en tiempo real del algoritmo, hemos optado por aplicar técnicas de eliminación de *background* [9] que permiten obtener resultados satisfactorios con un número reducido de operaciones.

Por otro lado, la miniaturización y portabilidad de dispositivos es un hecho a tener en cuenta en cualquier diseño. En esta línea han aparecido en los últimos años miniordenadores y placas procesadoras con altas capacidades de cálculo como Arduino, BeagleBone, Minnowboard, Raspberry Pi, etc. En particular, Raspberry Pi ha sido utilizada como placa de control en numerosas aplicaciones tan dispares como la domótica, servidores web, detección multisensor, etc [10]. Este hecho nos llevó a elegirla como placa de control para la realización de este trabajo.

A continuación, en el apartado 2 se va a describir el hardware empleado y sus conexiones. En el punto 3 se explica el algoritmo de funcionamiento, con especial interés en la obtención de información a partir del análisis de las imágenes. Posteriormente, se analizan los resultados obtenidos a partir de las pruebas diseñadas para evaluar el funcionamiento del sistema. Para terminar, se presentan las principales conclusiones de este trabajo.

2.- Diseño del dispositivo

Tal y como se ha mencionado en la sección anterior, el dispositivo está basado en la placa procesadora Raspberry Pi. En particular, se ha utilizado el último modelo disponible en el mercado, Raspberry Pi 3 modelo B+, [10]. Por otro lado, para la toma de imágenes se ha conectado a la placa una cámara Picamera, desarrollada específicamente para Raspberry Pi y optimizada con tal fin. En cuanto al control semafórico, se han configurado los pines GPIO de la placa necesarios como salidas analógicas para el control de las respectivas luces del semáforo, simuladas con el uso de LEDs. En la Figura 1 se muestra un esquema del dispositivo.

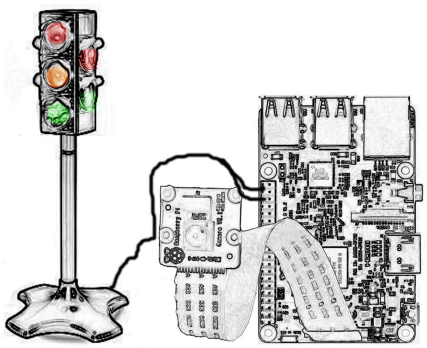


Fig.1: Esquema del dispositivo mostrando la placa procesadora Raspberry Pi, la cámara Picamera y un semáforo.

3.- Algoritmo utilizado

Se ha elegido Python como lenguaje de programación para la realización de este trabajo

[11] ya que se trata de un lenguaje nativo en Raspberry Pi, de uso libre y que encaja a la perfección con las necesidades y requisitos del sistema.

El algoritmo desarrollado consta de tres módulos: la captura de las imágenes, el procesado de las mismas y el control de las luces del semáforo. Para la captura de las imágenes se ha utilizado la biblioteca RaspiCam nativa de Raspberry Pi, ya que se ha optado por usar la PiCamera [12]. Por otro lado, para el tratamiento digital y procesado de las imágenes se ha empleado la librería OpenCV. Se trata de una librería de uso libre ampliamente utilizada en el ámbito científico. Por último, la propia Raspberry Pi ofrece control de los puertos GPIO mediante una clase propia. Se trata por tanto de un software completamente libre con las ventajas que ello conlleva a la hora de la aplicación final y puesta en marcha del dispositivo.

3.1.- Funcionamiento del algoritmo:

Una vez inicializado el algoritmo, este realiza un bucle infinito comenzado por hacer una captura de la imagen sobre el escenario. A continuación procesa la imagen y la información obtenida se usa como entrada para la máquina de estados. Finalmente el sistema actúa sobre las luces del semáforo y elimina la imagen capturada para dar entrada a una nueva imagen, Figura 2.

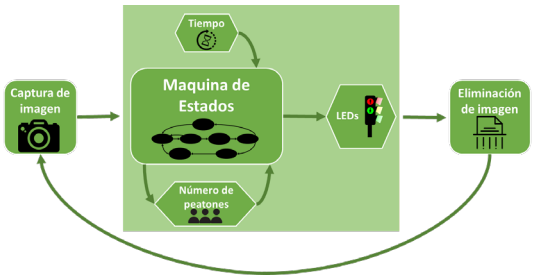


Fig.2: Bucle infinito del algoritmo.

En cada estado de la máquina de estados se realiza el análisis del fotograma para así conocer el número de peatones y definir el apagado o encendido de los LEDs. La transición de un estado a otro se realiza en función del número de peatones presentes en la escena y del tiempo de espera transcurrido. Un diagrama simplificado de la máquina de estados se muestra en la Figura 3.

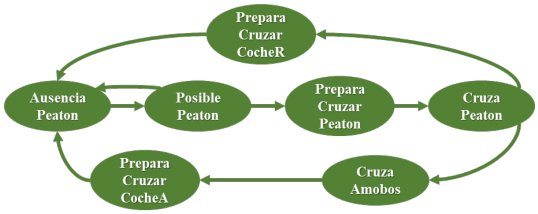


Fig.3: Maquina de estados.

3.2 - Análisis de la imagen.

El algoritmo de visión artificial es el encargado de procesar las imágenes, evaluar en qué estado se encuentra el sistema y así determinar las acciones a tomar. Este se divide en cuatro pasos. En primer lugar, la imagen de entrada es preprocesada, escogiendo la zona de interés y adaptándola para el posterior procesado. En segundo lugar, se substrae el fondo y se binariza la imagen resultante. Luego se manipula la imagen binaria mediante operaciones morfológicas para obtener una imagen de manchas, identificando las manchas como personas y, por último, se evalúan las áreas de las manchas deduciendo así el número de peatones al que equivale cada una. Para este último paso es necesaria una calibración previa del sistema ya que el tamaño de mancha equivalente a una persona dependerá de la altura a la que este colocada la cámara, entre otros factores.

La parte fundamental de procesado de la imagen es la sustracción del fondo, en inglés *background subtraction*. Consiste en clasificar todos los píxeles de un determinado fotograma bien como fondo, o como primer plano. En primer plano se engloban todos los objetos relevantes, mientras que en el fondo se encuentran todos aquellos objetos sin interés. Para ello se obtiene la densidad de probabilidad que caracteriza los píxeles de fondo y se analiza si los valores de cada pixel quedan bien descritos por ella. La función utilizada para la caracterización del *background* estima esta densidad de probabilidad mediante una mezcla de gaussianas cuyo número y peso se adaptan dinámicamente a las características de la imagen [9].

Este análisis dinámico del fondo permite adaptar el análisis a variaciones de la escena no asociadas a la aparición de elementos de interés, como son los cambios de iluminación (que son comunes en un entorno de trabajo como el que nos ocupa, en el que la iluminación ambiente no está controlada). La actualización del *background* se produce a partir de un número de fotogramas y se va adaptando a los valores de intensidad que no se modifican. Es por ello que, si aparece un peatón y se detiene un tiempo suficientemente largo, acabe siendo identificado como fondo de la imagen. Para evitarlo, en este trabajo se ha combinado este método dinámico con un método estático. El dinámico para actualizar el fondo constantemente y detectar peatones en movimiento, y el estático para la zona de espera, cuando el peatón se queda quieto. En todo caso, el fondo dinámico se sigue calculando y almacenando para utilizarlo cuando los peatones salen de la escena.

La clase de OpenCV escogida para llevar a cabo la sustracción del fondo en el método dinámico es *BackgroundSubtractorMOG2* [13], basado en el modelo *Gaussian Mixture* desarrollado en [9] y mejorado con la capacidad de detección de sombras marcadas [14], al tratarse de elementos que pueden inducir a error en la clasificación de un pixel como elemento de interés. Su funcionamiento depende del número de fotogramas utilizado para la estimación dinámica de las propiedades estadísticas del fondo y de un umbral que determina qué gaussianas forman parte de la mezcla en función de la distancia de *Mahalanobis* entre ellas.

Resumiendo, el análisis comienza con una caracterización dinámica del fondo de la escena, hasta que aparece un peatón en la zona de espera, momento en el que el análisis pasa a ser estático. Este consiste en pasar el ultimo fondo actualizado obtenido del método de sustracción de fondo dinámico y el fotograma actual por un filtro de desenfoque gaussiano. Seguidamente se hace una resta directa entre ambos que se binariza, lo que permite identificar los píxeles asociados a la aparición de los peatones. Para evitar errores, se eliminan los ruidos de la imagen binaria utilizando operaciones morfológicas de erosión y dilatación con las que también se incrementan los tamaños de los elementos detectados. Esto debe tenerse en cuenta a la hora de obtener el resultado final del conteo de personas que consiste en sumar todas las áreas de manchas consideradas posibles peatones y dividir el resultado por el área aproximada de una persona, que se obtiene tras el proceso de calibración.

4.- Resultados y discusión

Para calibrar y evaluar el funcionamiento del sistema se han capturado vídeos en un entorno real en los que un grupo de personas realizaba acciones similares a las de esperar para cruzar un paso regulado con semáforo. Se han grabado diferentes situaciones en las que se modificaba el número de personas, la distancia entre ellas o la velocidad de movimiento. También se ha considerado tanto un número creciente de peatones esperando como la posibilidad de que los viandantes entraran y posteriormente abandonaran la escena sin esperar al semáforo. En las figuras 4-6 se visualizan las diferentes fases del proceso.

El sistema presenta una precisión de una/dos personas lo que garantiza la regulación racional del paso de coches y peatones. Los errores más importantes se producen principalmente en la substracción de fondo, cuando la diferencia entre el primer plano y el fondo es muy grande (cambio brusco) o muy pequeño (peatón camuflado). Se comprueba que los errores más críticos se producen en el proceso de binarización, ya que pueden ser incrementados en las fases posteriores.

La ventaja de este algoritmo, en comparación con otros que permiten una mayor precisión, es la sencillez, que permite su ejecución en tiempo real en el dispositivo utilizado. El tiempo total dedicado al análisis de la imagen, desde su entrada hasta su eliminación, es, como máximo, de unos 0,4 segundos en los peores casos, dato muy interesante a tener en cuenta en potenciales aplicaciones.

Fig.4: Imagen original.

Fig.5: Imagen filtrada y girada.

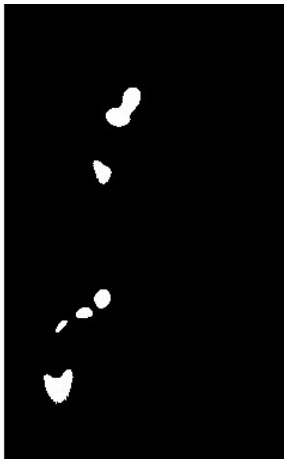


Fig.6: Imagen binaria una vez restado el fondo.

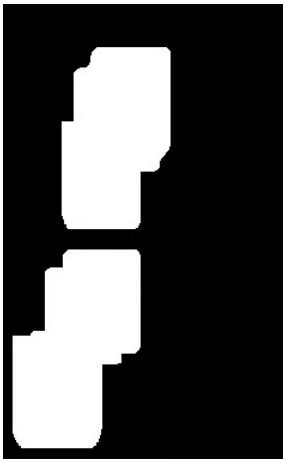


Fig.7: Imagen de manchas tras aplicar las operaciones morfológicas.

5.- Conclusiones

El resultado es un sistema económico de control de semáforos en tiempo real con capacidad de visión artificial. Tanto el hardware como el software son de libre licencia y de bajo coste. Este hecho, haría necesario dotar de una protección robusta a todos sus componentes si se tuviera que poner en marcha en exteriores. A diferencia de otros métodos de detección de personas, la técnica escogida tiene la gran ventaja de ser simple, sencilla, y sobre todo, rápida. Por el contrario, la principal desventaja de este algoritmo es la precisión en el recuento de peatones. No es exacto sino que puede tener un error de una a dos personas en condiciones normales o errores grandes si se produce cambios bruscos entre el último fotograma y el actual, como cambios climatológicos o de iluminación drásticos. Sin embargo, a pesar de los posibles fallos que puede dar el algoritmo de visión, no afecta al funcionamiento general del sistema. Con la presencia del bloque de tiempo, que controla los tiempos máximos y mínimos en cada estado, se garantizan los movimientos entre estados de la máquina de estados y así, la actividad cíclica del semáforo. Además, se protege la privacidad u anonimato de los peatones registrados, ya que para el funcionamiento del algoritmo no se necesita una identificación precisa de los sujetos.

Agradecimientos.

Este trabajo ha contado con la financiación de la Cátedra Mobility City de la Universidad de Zaragoza, convocatoria 2019 de ayudas al estudio para el desarrollo de Trabajos Fin de Grado y Trabajos Fin de Master en el contexto de las tecnologías implicadas en el campo del vehículo conectado y autónomo y la movilidad sostenible. También ha sido financiado en parte por el Gobierno de Aragón-Fondo Social Europeo (Grupo de Tecnología Óptica Láser— E44_17R).

Referencias

[1] B. DE SCHUTTER, B. DE MOOR, *"Optimal Traffic Light Control for a Single Intersectionm"*, European Journal of Control, 4, 3, 260-276, 1998.

[2] Henk TAALE, Thomas BÄCK, Mike PREUSS, A.E. EIBEN, J.M. DE GRAFF, C.A. SCHIPPERS, *"Optimizing traffic light controllers by means of evolutionary algorithms"*, Proceedings of the EUFIT'98 Conference, 1730-1734, 1998.

[3] K.J. PRABUCHANDRAN, A.N. HEMANTH KUMAR, Bhatnagar SHALABH, *"Multi-agent reinforcement learning for traffic signal control"*, International IEEE Conference on Intelligent Transportation Systems (ITSC), 2529-2534, 2014.

[4] Michael BOMMES, Adrian FAZEKAS, Tobias VOLKENHOFF, Markus OESER, *"Video Based Intelligent Transportation Systems – State of the Art and Future Development"*, Transportation Research Procedia, 14, 4495-4504, 2016.

[5] C.L. WAN, K.W. DICKINSON, "Road Traffic Monitoring Using Image Processing—A Survey of Systems, Techniques and Applications", IFAC Proccedings Volumens, 23, 2, 27-34, 1990.

[6] T TOPRAK, B BELENLIOGLU, S DOGAN, B AYDIN and M A SELVER "*On Diversity and Complementarity of Pedestrian Detection Models*" Journal of Physics: Conference Series, Volume 1141, conference 1 (2018)

doi:10.1088/1742-6596/1141/1/012152

[7] Y. TIAN, P. LUO, X. WANG, X. TANG "*Pedestrian Detection aided by Deep Learning Semantic Tasks*" 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) DOI: 10.1109/CVPR.2015.7299143

[8] M. BILAL, M. S. HANIF "*High Performance Real-Time Pedestrian Detection Using Light Weight*" J Sign Process Syst (2019) 91: 117. https://doi.org/10.1007/s11265-018-1374-7

[9] Z. ZIVKOVIC. "*Improved Adaptive Gaussian Mixture Model for Background Subtraction*" 17th International Conference on Pattern Recognition, 2004. ICPR 2004. DOI: 10.1109/ICPR.2004.1333992

[10]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[11] https://opencv-python-tutroals.readthedocs.io/en/latest/

[12]https://picamera.readthedocs.io/en/release-1.10/quickstart.html

[13]https://docs.opencv.org/ref/2.4/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html

[14] Z. ZIVKOVIC and F. van der HEIJDEN. "*Efficient adaptive density estimation per image pixel for the task of background subtraction*" Pattern Recognition Letters (2006) 27: 7, pp 773-780