

Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

**Embedded Software for Pedestrian Detection using the
Raspberry Pi**

Author:

Caio Christóvão da Silva Porto

Advisor:

Prof. Carlos José Ribas d'Avila, M. Sc.

Examiner:

Prof. Aloysio de Castro Pinto Pedroza, D. Sc.

Examiner:

Prof. Heraldo Luís Silveira de Almeida, D. Sc.

DEL

March 2014

Dedication

To my parents, siblings and grandparents.

Acknowledgement

To my parents, siblings and grandparents who helped me and guided me through my education.

To Professor Carlos José Ribas d'Avila for helping me achieve my professional objectives and believing in my work.

To professors Aloysio de Castro Pinto Pedroza and Heraldo Luís Silveira de Almeida for accepting to participate as examiners.

To Professor Katia Obraczka for welcoming me in her laboratory and giving me the opportunity to work with her.

To Kevin Abas for working with me in this project and helping I while in UC Santa Cruz.

To my friends Oliver von Behr Kuster, Felippe Kern Noel and Rafael Rodriguez Martinho for the friendship and fun shared with me through these years of undergraduate education.

Abstract of Undergraduate Project presented to POLI/UFRJ as partial fulfillment of the requirements for the degree of Electronics and Computer Engineer.

Embedded Software for Pedestrian Detection Using the Raspberry Pi

Caio Christóvão da Silva Porto

March/2014

Advisor: Carlos José Ribas d'Avila

Course: Electronics and Computer Engineering

Nowadays surveillance cameras are composed of a simple hardware responsible for the capture and upload of images to servers.

This project aims the creation of an embedded software capable of processing and analyzing the captured images within the node itself, and then decides if it should be recorded for later upload.

In this document is described the implementation of an embedded software used for the pedestrians detection.

Keywords: Raspberry Pi, Surveillance, OpenCV, Object Detection.

Summary

Dedication	ii
Acknowledgement	iii
Abstract	iv
Figure List	vii
Chart List	viii
Abbreviation List	ix
Chapter 1 - Introduction	1
1.1 – Chapters Contents	1
1.2 - Theme	1
1.3 - Objectives	1
1.4 - Justification	1
1.5 - Constraints	2
1.6 – Description	2
Chapter 2 – Object Detection Techniques	3
2.1 – Haar-like Features	3
2.2 – Histogram of Oriented Gradients	4
2.3 - Background Subtraction.....	5
2.3.1 – Frame Differencing.....	5
2.3.2 – Mean Filter	6
2.3.3 – Mixture of Gaussians.....	7
Chapter 3: Hardware	8
3.1 – Raspberry Pi	8
3.1.1 – Specifications.....	8
3.2 – Camera Module	9
3.2.1 – Specifications.....	10
3.2.2 – API.....	11

Chapter 4 – Project Detailing.....	12
4.1 – Block Diagram	12
4.2 – Solution	13
Chapter 5 – Starting Configurations	14
5.1 – Camera	14
5.2 – Video Recording	15
Chapter 6 – Detection and Logic	16
6.1 – Pedestrian Detection.....	16
6.2 – Storage and Power Control	19
Chapter 7 – Results	20
Chapter 8 – Conclusion and Future Works.....	22
References.....	23
Apendix 1 - Pedestrian Detection Program	25

Figure List

Figure 1 - Feature types used by Viola and Jones.	3
Figure 2 - Example of the average gradient image after training.	4
Figure 3 - Frame Differencing diagram.	5
Figure 4 - Raspberry Pi board.	8
Figure 5 - Camera module.	10
Figure 6 - Software architecture blocks diagram.	12
Figure 7 - Object moving in four consecutive frames.	20
Figure 8 - Detection errors that can generate false alarms: (a) Object occlusion, (b) reflex, (c) low contrast and (d) shape changes.....	20

Chart List

Table 1 - Results example for a Mean Filter algorithm for different N.....	6
Table 2 - Raspberry Pi specifications.	9
Table 3 - Camera parameters and values.	11
Table 4 - Function <i>set_camera_parameters</i> description.	14
Table 5 - Camera configuration values.	14
Table 6 - Description of the <i>open</i> function	15
Table 7 - Description of the OpenCV used functions.....	17

Abbreviation List

UFRJ – Universidade Federal do Rio de Janeiro

API – Application Programming Interface

PIR – Passive Infrared Sensor

Chapter 1 - Introduction

1.1 – Chapters Contents

In the chapter 1, the proposed problem is studied to be solved in this Project. The chapter 2 regards to the study of techniques for object detection. While in the chapters 3, the study is focused on the description of the hardware and its API.

Starting on chapter 4, the project description and the solutions adopted are shown. In the chapter 5, the implementation of the camera and video configuration is presented. Whereas in chapter 6, the implementation of the pedestrian detection and the video record logic are shown.

The chapter 7 shows the results obtained after the tests and, at last, in the chapter 8, the conclusions and the future works are presented.

1.2 - Theme

The theme of this work is the development of an embedded system, using the Raspberry Pi minicomputer, which will be capable of detect pedestrians using a camera module and make decisions.

1.3 - Objectives

The main objective is to develop an embedded system capable of preprocessing images, captured by the camera module, to detect pedestrians on a scene. Furthermore, the system should be able to characterize the video in order to define if it is relevant to save it and, later, upload to the server.

1.4 - Justification

Surveillance cameras with low energy consumption, up until now, use movement detection sensors to know when it is necessary its activation. Even though, the use of this kind of system in some environments generates large amounts of false alarms, due to animals and plants in most of the cases.

1.5 - Constraints

For this type of systems, we have, as constraint, the utilization of the minicomputer Raspberry Pi and its camera module due to the low energy consumption and the low cost. Furthermore, the system must analyses the images in a fast and optimized way, with the goal of save energy and keep an acceptable video quality. It is important to mention that the camera should be able to be deployed in different types of scenarios with the need of minimum or none calibration.

1.6 – Description

The Project is constituted of a Raspberry Pi and its camera module, assuming the functionality of surveillance camera. It will use a very lightweight Linux distribution, a Raspberry Pi Arch Linux version, as the embedded operating system for compilation and execution of image processing programs.

The main program will be implemented in C++, regarding the constraints of the OpenCV (Open Source Computer Vision Library) and the Raspberry Pi camera module API. The public API for the camera is currently implemented in C and most of the OpenCV functions are only implemented in C++ and Python.

The program will be divided in two main sections: image capturing and image processing. The first is a loop responsible for configuring the camera parameters for better performance of the image processing algorithms and for the continuous image capture. The second is responsible for processing each image captured, looking for pedestrians and recording the images following the defined power and storage save rules.

Chapter 2 – Object Detection Techniques

In this chapter a study is made of the most popular types of object detection methods, showing its advantages and disadvantages. The objective here is to find the best solution that fits the needs and the constraints of the system.

2.1 – Haar-like Features

The Haar-like Features [1] technique was the first to provide object detection with real-time rates. Given an image, a bounding box is applied to the target object, which is called “detection window”. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. As shown in Figure 1, Viola and Jones have four different types of features for better coverage of horizontal and vertical features.

Because such a Haar-like feature is only a weak learner or classifier, a large number of Haar-like features are necessary to describe an object with sufficient accuracy. Therefore, Viola and Jones features are organized in a classifier cascade.

The key advantage of a Haar-like feature over most other features is its calculation speed. Due to the use of *integral images* [2], a Haar-like feature of any size can be calculated in constant time. On the other hand, this type of object detection technique will not be helpful for this project, since a lot of training is required for a good result and this project focus on the flexibility of positioning the cameras in different scenarios with minimal calibration.

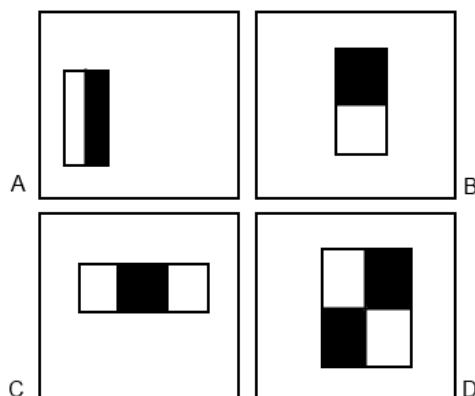


Figure 1 - Feature types used by Viola and Jones.

2.2 – Histogram of Oriented Gradients

The HOG [3] are feature descriptors that represent the intensity gradient or edge orientation in localized portions of an image. These descriptors are sufficient to describe the objects appearance and shape. The implementation is done by dividing the image into small regions, called “cells”, and for each cell is calculated the histogram of gradient directions or edge orientations for the pixels of the cell. The representation of the descriptor is the combination of the histograms, Figure 2.

The biggest advantage of the HOG is that it supports invariance to geometric and photometric transformations: translations or rotations. Such changes make little difference if they are much smaller than the local spatial or orientation size. In addition, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body segmentations to change their appearance and be ignored as long as they maintain a roughly upright position. The HOG thus fits perfectly for human detection.

After testing the HOG algorithm implementation and searching about its performance, it was possible to conclude that it is not suitable for our platform. Using the method for full body detection, it was obtained an approximately 1 frame per minute processing speed. This result is due to the hardware constraint and the complexity of the algorithm that even in modern processor is only able to achieve 14fps using a low-resolution image (640 x 480). Besides the processing speed, this method of object detection requires training as it is based on features description.



Figure 2 - Example of the average gradient image after training.

2.3 - Background Subtraction

The Background Subtraction, also called Foreground Detection, is a technique used for extraction of the foreground image. It is a widely used method for object detection in videos from static cameras. This approach detects moving objects from the difference between the current frame and a reference frame called “background model”.

2.3.1 – Frame Differencing

In the Frame Differencing technique, it is done a subtraction between the current frame ($t + 1$) and the previous frame (t). As we can see, it is assumed that the background is the previous frame.

$$D(t + 1) = |V(x, y, t + 1) - V(x, y, t)|$$

To improve the subtraction and remove image noise, a threshold Th is inserted.

$$|V(x, y, t) - V(x, y, t + 1)| > Th$$

The advantage is that this approach is very quick to adapt to changes in lighting or camera motion. On the other hand, if an object stops it is no long detected and only a small portion of the objects’ pixels is detected, Figure 3.

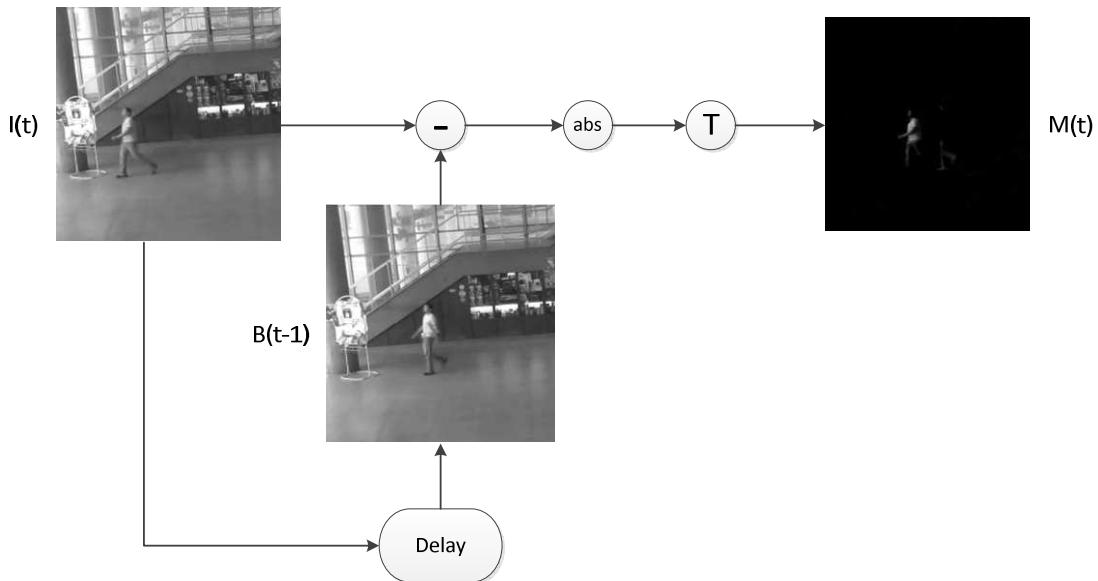


Figure 3 - Frame Differencing diagram.

2.3.2 – Mean Filter

The Mean Filter is very similar to the frame differencing, but this time, to improve the background image it is done an average of N the previous images. We assume that the background is more likely to appear in a scene.

$$B(x, y) = \frac{1}{N} \sum_{i=1}^N V(x, y, t - 1)$$

This average corresponds to averaging the corresponding pixel for all the N images. Then, again, a threshold is applied to reduce the subtraction noise:

$$|V(x, y, t) - B(x, y)| > Th$$

Table 1 - Results example for a Mean Filter algorithm for different N.

N	Background Model	Foreground Mask
1 0		
2 0		
5 0		

2.3.3 – Mixture of Gaussians

The MoG is a type of adaptive background method first implemented by Stauffer and Grimson [5] with many improvements later. It is assumed that every pixel's intensity values in the video can be modeled using a mixture of Gaussians model. At each interaction, Gaussians are evaluated using a simple heuristic to determine which ones are mostly like to correspond to the background. Pixels that do not match with the background Gaussians are classified as foreground. Then, the foreground pixels are grouped using 2D connected component analysis.

This method has a “threshold” for each pixel and these “thresholds” are adapting by time, what can improve the noise reduction considerably when compared to the previous shown methods. Another big advantage is that this method, despite of the mathematical complexity for humans, have a good performance even in low-end hardware, like the Raspberry Pi (up to 3 fps @ 426 x 240).

Chapter 3: Hardware

3.1 – Raspberry Pi

The Raspberry Pi (Figure 4) is a credit-card-sized single-board computer developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. The first reports of arrival was on April of 2012, and currently it has already been shipped more than 2 million units worldwide for all type of consumers.

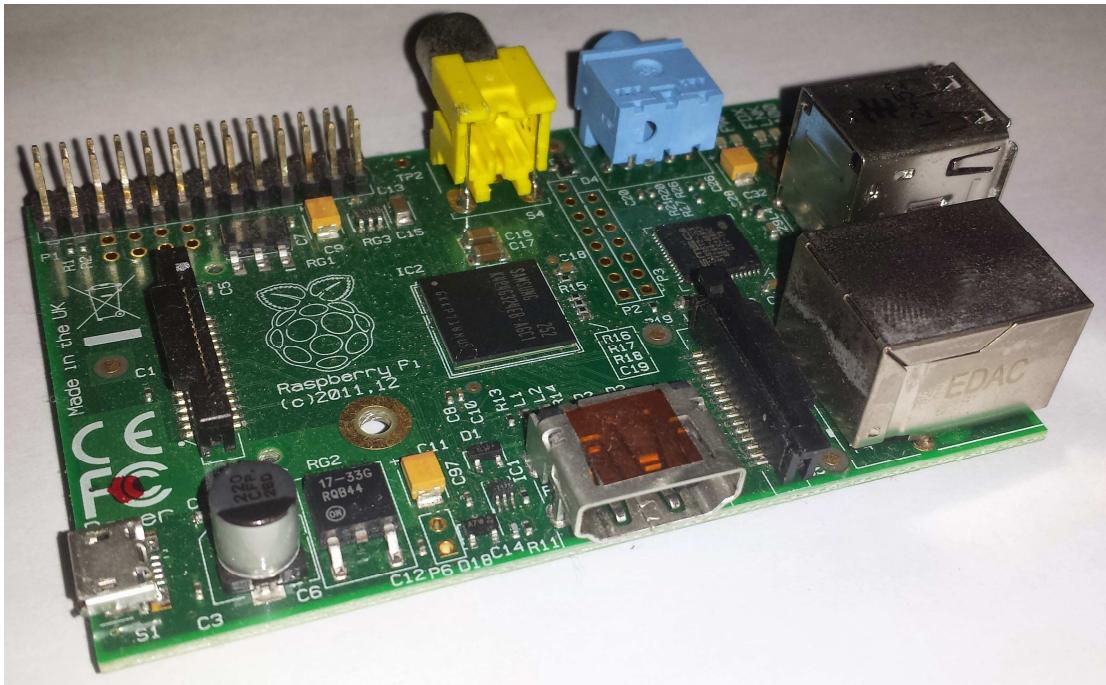


Figure 4 - Raspberry Pi board.

3.1.1 – Specifications

The Raspberry Pi specifications are shown on the Table 2. We can see that its power consumption and processor speed are relatively low if compared to a regular computer. The power consumption and its popularity were my main reasons for picking up this hardware.

Table 2 - Raspberry Pi specifications.

Price	US\$ 35
Soc	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, and single USB port)
GPU	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)
Memory	Broadcom VideoCore IV @ 250 MHz OpenGL ES 2.0 (24 GFLOPS) MPEG-2 and VC-1 (with license), 1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder
USB 2.0 Ports	2
Video input	A CSI input connector for the camera module
Video outputs	Composite RCA (PAL and NTSC), HDMI (rev 1.3 & 1.4), raw LCD Panels via DSI 14 HDMI resolutions from 640×350 to 1920×1200 plus various PAL and NTSC standards.
Audio outputs	3.5 mm jack, HDMI, and, as of revision 2 boards, I ² S audio
Onboard storage	SD / MMC / SDIO card slot
Onboard network	10/100Mbps Ethernet
Low-level peripherals	8 × GPIO, UART, I ² C bus, SPI bus with two chip selects, I ² S audio, +3.3 V, +5 V, ground
Power ratings	700 mA (3.5 W)
Power source	5 volt via MicroUSB or GPIO header
Size	85.60 mm × 53.98 mm
Weight	45g

3.2 – Camera Module

The Raspberry Pi camera module board (Figure 5) was recently launched on May 2013 allowing the use of the Raspberry Pi for real-time video processing purposes. With the module, two new programs were introduced to help the interaction with this new module: Raspistill and Raspivid. The first one is dedicated to static images, while the second one to record videos.

The lack of documentation and developer friendly architecture gave a lot of work to the pioneers. The first step of all the developers was to try to understand the bad documented source code and then create their own modifications.

In this chapter, it will describe and explain the existing library for communication with the camera module board. The results of my research presented here are focused on the use of the camera to capture static images.

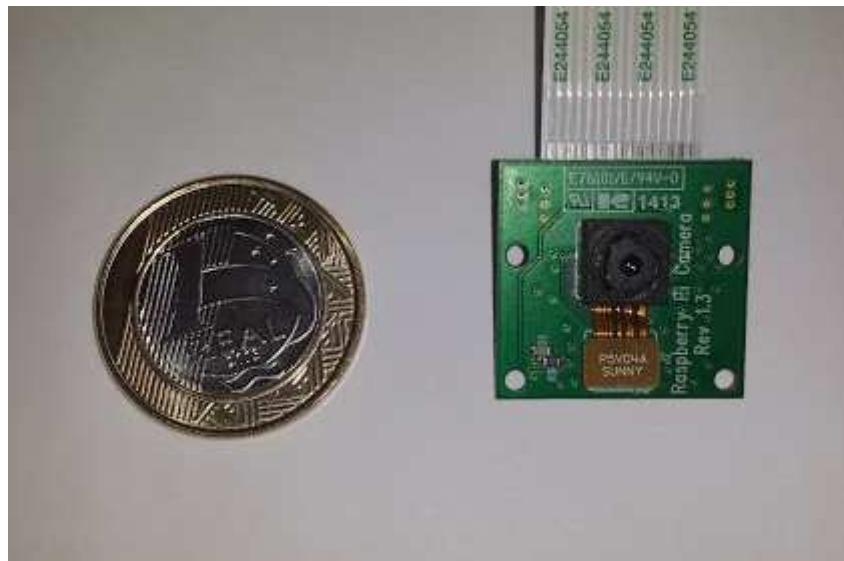


Figure 5 - Camera module.

3.2.1 – Specifications

The Raspberry Pi Camera Board features a 5MP (2592×1944 pixels) Omnivision 5647 sensor [6] in a fixed focus module. The module attaches to Raspberry Pi, by way of a 15 Pin Ribbon Cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI), which was designed especially for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the BCM2835 processor.

- Still Pictures Resolution – 2592 x 1944
- Video Resolution – 1080p @ 30fps, 720p @ 60fps and VGA @ 60/90fps
- Size – 20 x 25 x 9mm
- Weight – 30g
- Price - US\$ 29.00

3.2.2 – API

The two applications released with the camera were based on the mmal API, which runs over OpenMAX. The mmal API provides an easier way to communicate with the hardware than the OpenMAX and it was developed by Broadcom to use specifically with this camera model.

Table 3 - Camera parameters and values.

Parameter	Values
Sharpness	-100 to 100
Contrast	-100 to 100
Brightness	0 to 100
Saturation	-100 to 100
Video Stabilization	0 or 1
Exposure Compensation	-10 to 10
Rotation	0 to 359
Horizontal Flip	0 or 1
Vertical Flip	0 or 1
Exposure Mode	Off, Auto, Night, Backlight, Spotlight, Sports, Snow, Beach, Very Long, Fixed fps, Anti Shake, Fireworks
Exposure Metering Mode	Average, Spot, Backlit, Matrix
Auto White Balance Mode	Off, Auto, Sunlight, Cloudy, Shade, Tungsten, Fluorescent, Incandescent, Flash, Horizon
Image Effect	None, Negative, Solarize, Posterize, Whiteboard, Blackboard, Sketch, Denoise, Emboss, Oil paint, Hatch, Gpen, Pastel, Water Color, Film, Blur, Saturation, Color Swap, Washed Out, Posterise, Color Point, Color Balance, Cartoon
Color Effect – Chrominance	16 to 240

Chapter 4 – Project Detailing

4.1 – Block Diagram

The block diagram on Figure 4 illustrate the software architecture. Observing the figure, we can see that the Camera API module is responsible for configuring the camera parameters and capturing the images.

The Image Processing block is subdivided in three blocks responsible for doing the computer vision processes. The Background Subtraction extracts the moving objects from the image, while the Contour Detection applies a contour to the objects extracted. In the end, the objects are classified for future decisions.

The Decision Algorithm block is responsible for the decisions of recording the video and stopping the program in case of no pedestrian on the scene.

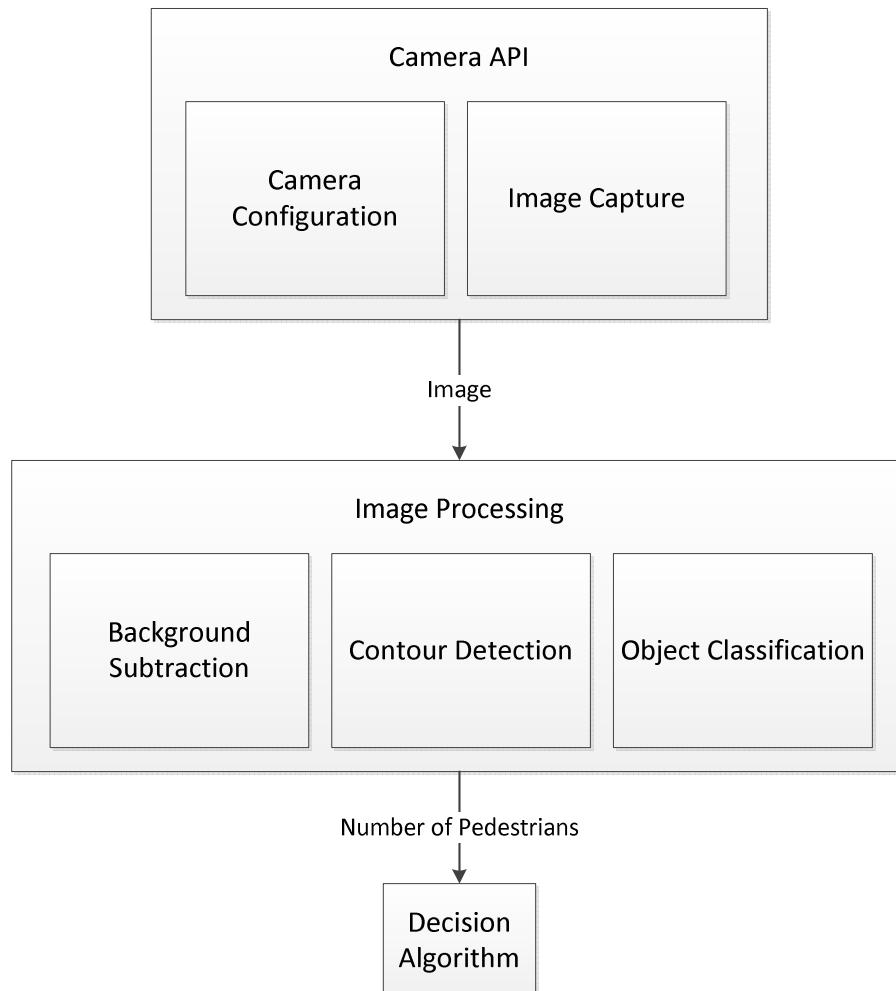


Figure 6 - Software architecture blocks diagram.

4.2 – Solution

The adopted solution is based on the results of the studies made on the previous sections. The system will take pictures in a loop and for each image captured the image processing will be done. The Image Processing will be able to detect the presence of pedestrians, distinguishing from animals or vehicles.

The first version of the camera module released has an IR filter in front of its image sensor. As this surveillance system has as one of the requirements the capability of deploy in different environment, the camera should be sensitive to dark scenes. The solution found was to remove the IR filter [9], allowing the use of IR lights to illuminate the scene during the nighttime.

The study of the object detection techniques using background subtraction, directed to the use of the MoG solution. As explained, it will have better performances in low-end hardware, like Raspberry Pi. After detecting the objects, it is needed to classify them. The simplest and fastest way is to observe the objects' dimensions. After detecting the contour of the object, it is fitted a bounding box around it. The bounding box gives the height and the width of the objects in pixels units, but we can estimate if it is a human by the ratio between height and width.

Considering that the Raspiberry Pi will be working as a part of a bigger solution, it is possible to say that the system contains a PIR sensor to save energy and turn on the Raspberry Pi if a movement is detected. The last step is the storage and power save decisions. When the Raspberry Pi turns on, the program will start looking for humans on the scene. When detected the presence of humans, the program will start recording videos of 5 minutes until no human is on the scene anymore. This way we can save storage by recording only important information. The use of small video files are to make easier for upload to a server, if needed. In case of the program does not detect human for a certain period, it can power off the board to save energy.

Chapter 5 – Starting Configurations

By analyzing the camera API and its configuration options, with performance experiments, it was possible to establish what would be the best configuration for the camera and the recorded video. The configurations shown in this section are considering the constraints for better storage management and hardware performance. For this implementation, it was considered that the configurations are not changed through the program execution, but the exposure, as explained in section 6.1.

5.1 – Camera

The camera is configured with the functions implemented by the Raspberry Pi team to make easier for the developers to interface with the mmal API. There are several functions, one for each parameter; to simplify it was created a function that is able of configuring all parameters at once, as it is shown on Table 4 followed by the arguments, on Table 5, used for the configuration.

It is important to note that the exposure is set to *off* on the configurations in order to avoid abrupt changes in the camera exposure due to light changes. It will be better explained on chapter 6.

Table 4 - Function *set_camera_parameters* description.

int set_camera_parameters(MMAL_COMPONENT_T *camera, const RASPICAM_CAMERA_PARAMETERS params)	
camera	Camera object created for communication with the hardware
params	Arguments list with the configuration values

Table 5 - Camera configuration values.

Parameter	Values
Sharpness	0
Contrast	0
Brightness	50
Saturation	0
Video Stabilization	0
Exposure Compensation	0

Rotation	180
Horizontal Flip	0
Vertical Flip	0
Exposure Mode	Off
Exposure Metering Mode	Auto and Off (section 6.1)
Auto White Balance Mode	Off
Image Effect	None
Color Effect – Chrominance	Disabled

5.2 – Video Recording

The video is recorded by saving the frames captured by the camera with the OpenCV library. Every time a video is recorded, it is generated a random string for the file name and it is added the extension “.avi”. The use of a random string was required, instead of the time, because the Raspberry Pi will have the power interrupted to save power, so it will not keep the current time. On the Table 6, there is a description of the function parameters and the arguments used for creating and setting up the video output file.

Table 6 - Description of the *open* function

<code>bool VideoWriter::open(const string& filename, int fourcc, double fps, Size frameSize, bool isColor=true)</code>	
filename – File Name	<i>Random.avi</i>
fourcc – Four CC code of codec	DIVX
fps – Frames per second	2.5
frameSize – Resolution	1280 x 720
isColor – Color or Black and White	False

Chapter 6 – Detection and Logic

In this chapter, it is explained the main algorithms used for the image processing and the video recording. It is considered the number of frames as the time unity, since every cycle of the loop corresponds to a frame, every frame corresponds to 1/2.5 of a second and each instruction in the loop is executed only once.

6.1 – Pedestrian Detection

Before executing the MoG algorithm, some steps were necessary to generate a better image to reach the expected performance. The camera API does not provide a method to set up a fixed value to the exposure. The exposure is very important when we are working with a camera that must be able to see in bright and dark ambient without interfering in the object detection. Therefore, it was needed a workaround to fix it.

Considering that the camera will only turn on when a pedestrian is detected, we can suppose that it will turn on and off many times. With that in mind, we can set up an exposure for each time it comes up. The camera exposure is set up to *Auto* while it capture the first two frames. After that, the camera will already be configured for the best exposure; therefore, we can turn off the exposure. Setting it to *Off* was observed that it saves the previous exposure configuration preventing the camera from abrupt brightness changes.

The first step of the image processing is the resize of the captured image. The original image, as shown in the camera configuration section, is being capture in 1280 x 720, but processing an image with this size would take longer, reducing the processing speed to 1 fps. In the tests by reducing the image for one third, it is possible to obtain results close to the one with the original image, but this time with a processing speed of 2.5 fps.

The next step is to execute the MoG function implemented in OpenCV library. In this case, it is used the method described by Zivkovic and Heijden [7] and implemented in the class *BackgroundSubtractorMOG2*. The constructor and the operator for the class are shown on the Table 7, as well the other functions used.

Table 7 - Description of the OpenCV used functions.

BackgroundSubtractorMOG2::BackgroundSubtractorMOG2(int history , float varThreshold , bool bShadowDetection =true);	
history	Number of frames used on the background average
varThreshold	Number of Gaussian mixtures
bShadowDetection	Detection of object's shadow

void BackgroundSubtractorMOG2::operator()(InputArray image , OutputArray fgmask , double learningRate =-1);	
image	Input image
fgmask	Output foreground mask image
learningRate	Learning rate of the background update

void erode(InputArray src , OutputArray dst , InputArray kernel , Point anchor =Point(-1,-1), int iterations =1, int borderType =BORDER_CONSTANT, const Scalar& borderValue =morphologyDefaultBorderValue())	
src	Input image
dst	Output image
kernel	Structuring element used for erosion
anchor	Position of the anchor within the element
iterations	Number of times the erosion is applied
borderType	Pixel extrapolation method
borderValue	Border value in case of constant border

void dilate(InputArray src , OutputArray dst , InputArray kernel , Point anchor =Point(-1,-1), int iterations =1, int borderType =BORDER_CONSTANT, const Scalar& borderValue =morphologyDefaultBorderValue())	
src	Input image
dst	Output image
kernel	Structuring element used for erosion
anchor	Position of the anchor within the element
iterations	Number of times the erosion is applied

borderType	Pixel extrapolation method
borderValue	Border value in case of constant border

```
void findContours(InputOutputArray image, OutputArrayOfArrays contours,
OutputArray hierarchy, int mode, int method, Point offset=Point())
```

image	Input image
contours	Output array of contours detected
hierarchy	Output array that shows when a contour is inside another
mode	Contour retrieval mode
method	Contour approximation method
offset	Offset that shifts every contour point

```
double arcLength(InputArray curve, bool closed)
```

curve	Input array of curves
closed	If true, the curve is closed (first and last vertices are connected)

```
Rect boundingRect(InputArray points)
```

points	Point set
--------	-----------

The constructor has a peculiarity, because this implementation of the MoG method permits the detection of shadows, what can be used to remove them from the foreground. The operator function extract the foreground mask and automatically calculate the new background model, simplifying the method.

The foreground mask can be quite noisy, so in order to get rid of the noise Openings are done. Opening is a morphological operation composed of two other morphological transformation: erosion and dilation (functions *erode* and *dilate*). The number of transformation need to be calibrated based on the scenario to obtain better results. The values can be set in runtime through the configuration file.

With the clean image in hands, it is possible to find the contours of the objects by using the function *findContours*. To remove once again the noise, it is calculated

the length of the contours, with the function *arcLength*, and ignore the ones with length below or above calibrated numbers. This calibration is important because the object can assume different sizes on the image, depending on where the camera is deployed, so the ratio between noise and object is also different.

The human detection is made by calculating the object's dimension (height and width). For that, it is applied a bounding box using the function *boundingRect*. This function only needs, as argument, the contour line of the object detected and it will fit a bounding box around it. Knowing the dimensions of the box, they are considered as the dimensions of the object. To differentiate the humans from other types of objects, the dimensions of the human body are used, in other words, the height is usually greater than the width.

6.2 – Storage and Power Control

The storage and power control, despite of its low complexity, is responsible for the main objectives of this project. First, reduce the number of false alarms generated by the simples systems that make use of PIR sensors as the only source of movement detector, and by consequence, reduce the amount of data generated. Second, reduce the power consumption, by shutting down the system when no pedestrian is detected.

In every program cycle, the program analyzes the image looking for pedestrians. This process starts as soon as the program is executed and only stops when it is powered off. For controlling the video recording and the shutdown of the board, there were implemented four rules:

- If a pedestrian is detected, it starts recording the video and the number of frames captured.
- If during the first six frames, half of them does not contain any pedestrians, it stops recording and shuts down the board.
- If no pedestrian is detected for 15 consecutive frames (5 seconds), it stops recording and shuts down the board.
- If the video being recorded reaches 5 minutes long it is closed and a new one is created to continue.

Chapter 7 – Results

The results listed here were obtained in tests made on campus during high-traffic hours to test the algorithm processing speed and performance. In this proof of concept, it is expected to be able of detecting human objects with low influence of the environment in the detection results.



Figure 7 - Object moving in four consecutive frames.

On the Figure 7, we can see the object moving through four consecutive frames. It is possible to note that single objects are well detected by the program. During the tests was observed that sometimes the object is not detected, as it is not using any tracking algorithm, the object gets lost in some frames.

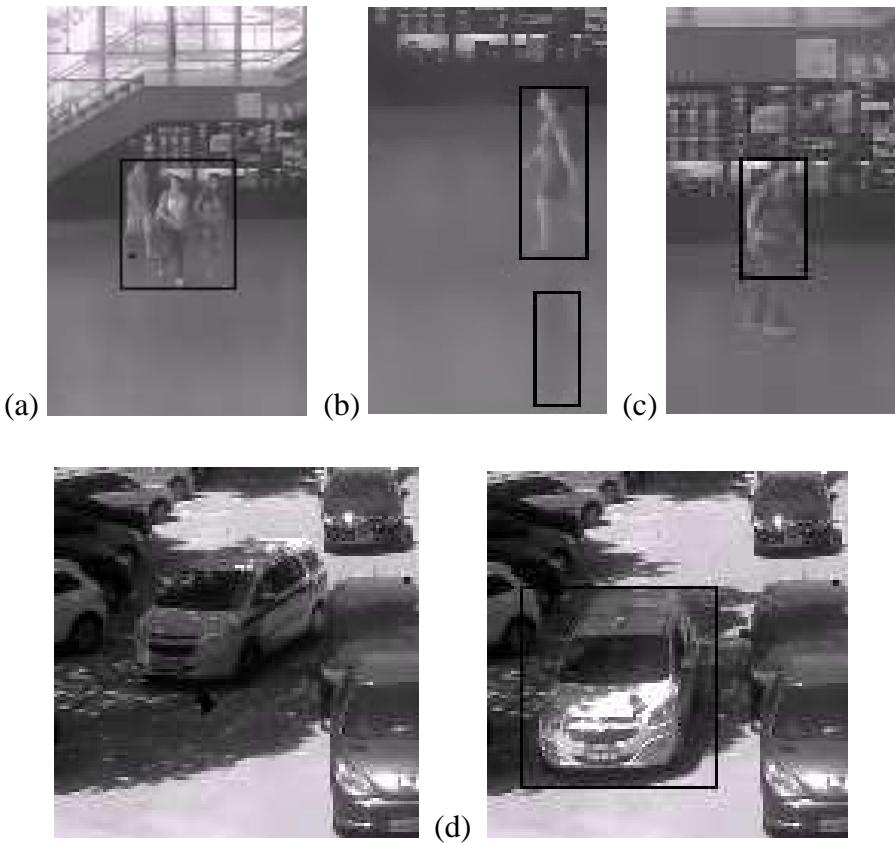


Figure 8 - Detection errors that can generate false alarms: (a) Object occlusion, (b) reflex, (c) low contrast and (d) shape changes.

On the other hand, the detection algorithm failed in a few cases. Pedestrians can be walking in groups (Figure 7a) and crossing by each other or by a static object. In these cases, they will not be detected, or the detection will be wrong. To reduce the error events, a stricter dimension classification can be applied to exclude group of people. Researches are being made to develop occlusion handles capable of detect the objects even when occluded [10, 12].

A false alarm can occurs when the scene contains reflexive surfaces or shadows; the same object is detected multiple times, one for each surface (Figure 7b). A fix for this problem is harder to obtain, because sometimes it is not possible to compare the objects or detect shadows. This problem will not affect our system directly, since we are not focusing on each object and shadows can be detected separated as they have a different contrast from the object.

The low contrast can be a difficulty for the MoG algorithm, so after the foreground extraction, some parts of the object are not detected (Figure 7c) or the low amount of points for the object are missinterpreted as noise. Increasing the resolution or using color images can reduce the errors caused by the contrast, but it would represent a slower image processing.

Moving objects can change their shapes through the time, as the camera sees them. If a person crouch for any reason, it will not be detected, or if a car or a motorcycle are viewed in vertical position relative to the camera (Figure 7d), the dimensions of the object will correspond to a human, as defined by the algorithm. Camera networks are helping to solve this problem, because they are able to track the same object between all the cameras, and with it, it is possible to see the same object by different angles [11].

Chapter 8 – Conclusion and Future Works

The results presented show that the project could solve, in a simple way, two well knowns problems of the surveillance system, the storage data amount and the low energy efficiency. Despite the fact of the human detection algorithm not being capable of achieve perfect accuracy, it could detect most of the moving bodies that crossed by the camera at least once in a period of 1 second, period necessary to continue recording after starting the program. The results are already very significant, considering that the low power and storage consumption were achieved by a system cheaper and smarter than the current commercial systems.

The program needs improvements to speed up the image processing, to be able to achieve better detection results in a higher frame rate. New studies are being made to allow images to be processed on the GPU of the Raspberry Pi, achieving better results with the use of specific libraries. The use of new algorithms for object detection, like the one described by Zhu et al [13], can also be good improvements.

This project was developed imagining it as a part of major system. In the idealized system, this program would be the module responsible for acquiring the video. In future works, it is planed the development of new modules to reduce the energy necessary for surveillance cameras. By adding a solar energy panel, the camera will be capable of harvesting the solar energy and be wireless. With a very low power Wi-Fi module, it will be able to upload the recorded videos to a server. In addition, to control all the modules, in order to keep the performance and control the energy usage through periods of little or no solar energy supplied, there will be a control module.

References

- [1] Viola, Paul, and Michael Jones. "Robust real-time object detection." International Journal of Computer Vision 4 (2001).
- [2] Crow, Franklin C. "Summed-area tables for texture mapping." ACM SIGGRAPH Computer Graphics. Vol. 18. No. 3. ACM, 1984.
- [3] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.
- [4] Birgi Tumeroy. "Background Subtraction". The University of Texas at Austin. http://www.cs.utexas.edu/~grauman/courses/fall2009/slides/lecture9_background.pdf
- [5] Stauffer, Chris, and W. Eric L. Grimson. "Adaptive background mixture models for real-time tracking." Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on. Vol. 2. IEEE, 1999.
- [6] OV5647 - 5-megapixel $\frac{1}{4}$ " Image Sensor with 1.4 μm Omni BSI Technology Offering HD Video. <https://www.dropbox.com/s/yfzkhm4i3iqhffe/ova5647.pdf>
- [7] Z.Zivkovic, F. van der Heijden. "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction", Pattern Recognition Letters, vol. 27, no. 7, pages 773-780, 2006.
- [8] OpenCV – MoG Documentation
http://docs.opencv.org/trunk/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=mog#id12
- [9] IR filter shenanigans. <http://www.raspberrypi.org/archives/4088>
- [10] Johnsen, Swantje, and Ashley Tews. "Real-time object tracking and classification using a static camera." Proceedings of IEEE International Conference on Robotics and Automation, workshop on People Detection and Tracking. 2009.

- [11] Chen, Phoebus, et al. "A low-bandwidth camera sensor platform with applications in smart camera networks." *ACM Transactions on Sensor Networks (TOSN)* 9.2 (2013): 21.
- [12] Raman, Rahul, Pankaj K. Sa, and Banshidhar Majhi. "Occlusion prediction algorithms for multi-camera network." *Distributed Smart Cameras (ICDSC)*, 2012 Sixth International Conference on. IEEE, 2012.
- [13] Zhu, Fang, Chen Zhao, and Jinmei Cheng. "ECLC: Edge character and latency connection enhanced inter-frame difference algorithm for real-time outdoor surveillance." *Distributed Smart Cameras (ICDSC)*, 2012 Sixth International Conference on. IEEE, 2012.

Appendix 1 - Pedestrian Detection Program

File mydemo.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <sstream>
#include <fstream>
#include <string>
#include <time.h>

#include <opencv2/opencv.hpp>
#include <opencv2/core/core_c.h>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

extern "C" {
#include "bcm_host.h"
#include "interface/vcos/vcos.h"

#include "interface/mmal/mmal.h"
#include "interface/mmal/mmal_logging.h"
#include "interface/mmal/mmal_buffer.h"
#include "interface/mmal/util/mmal_util.h"
#include "interface/mmal/util/mmal_util_params.h"
#include "interface/mmal/util/mmal_default_components.h"
#include "interface/mmal/util/mmal_connection.h"

#include "RaspiCamControl.h"
#include "RaspiPreview.h"
#include "RaspiCLI.h"

#include "vgfont.h"
}

using namespace cv;
using namespace std;
/// Camera number to use - we only have one camera, indexed from 0.
#define CAMERA_NUMBER 0

// Standard port setting for the camera component
#define MMAL_CAMERA_PREVIEW_PORT 0
#define MMAL_CAMERA_VIDEO_PORT 1
#define MMAL_CAMERA_CAPTURE_PORT 2

// Video format information
#define VIDEO_FRAME_RATE_NUM 30
```

```

#define VIDEO_FRAME_RATE_DEN 1

/// Video render needs at least 2 buffers.
#define VIDEO_OUTPUT_BUFFERS_NUM 3

/// Open cv numbers of frames processed
#define OPENCV_CONFIG_FRAMES 100

int mmal_status_to_int(MMAL_STATUS_T status);

/** Structure containing all state information for the current run
 */
typedef struct
{
    int timeout;                                /// Time taken before frame is
    grabbed and app then shuts down. Units are milliseconds
    int width;                                  /// Requested width of image
    int height;                                 /// requested height of image
    int bitrate;                               /// Requested bitrate
    int framerate;                            /// Requested frame rate (fps)
    int intraperiod;                           /// Intra-refresh period (key frame
    rate)
    char *filename;                            /// filename of output file
    int verbose;                               /// !0 if want detailed run
    information
    int demoMode;                             /// Run app in demo mode
    int demoInterval;                         /// Interval between camera
    settings changes
    int opencv_width;                          /// Size of the opencv
    image
    int opencv_height;                         ///
    Size of the opencv image
    int maxLength;                            /// Number of frames on each video
    int consecutiveHumans;                   ///
    Minimum number of consecutive humans detection to continue recording
    int consecutiveNoHumans;                 /// Minimum
    number of consecutive frames without humans to stop recording

    RASPIPREVIEW_PARAMETERS preview_parameters;   /// Preview
    setup parameters
    RASPICAM_CAMERA_PARAMETERS camera_parameters; // Camera
    setup parameters

    MMAL_COMPONENT_T *camera_component;          ///
    Pointer to the camera component
    MMAL_CONNECTION_T *preview_connection;       /// Pointer to
    the connection from camera to preview

```

```

        MMAL_POOL_T *video_pool;                                ///
Pointer to the pool of buffers used by video port
} RASPIVID_STATE;

/** Struct used to pass information in video port userdata to callback
*/
typedef struct
{
    string fileName;                                         ///
File name
    VideoWriter fileHandle;                                 ///
File handle to write buffer data to.
    VideoWriter fileHandle2;                               /// File
handle to write buffer data to.
    RASPIVID_STATE *pstate;                                ///
pointer to our state in case required in callback
    int abort;                                           ///
// Set to 1 in callback if an error occurs to attempt to abort the capture
    Mat image;                                         ///
Main image captured
    Mat image2;                                         ///
Image processed in opencCV
    int humanDetected;                                  ///
Number of consecutive frames with human detected
    int noHumanDetected;                                ///
Number of consecutive frames without human
    int framesRecorded;                                 ///
Number of frames recorded
    bool uploading;                                     ///
Control if any file is being uploaded.
    VCOS_SEMAPHORE_T complete_semaphore;
} PORT_USERDATA;

/**
* Assign a default set of parameters to the state passed in
*
* @param state Pointer to state structure to assign defaults to
*/
static void default_status(RASPIVID_STATE *state)
{
    if (!state)
    {
        vcos_assert(0);
        return;
    }

    // Default everything to zero
    memset(state, 0, sizeof(RASPIVID_STATE));

    // Now set anything non-zero

```

```

state->timeout = 5000; // 5s delay before take image
state->width = 1280; // Default to 1080p
state->height = 720;
state->bitrate = 17000000; // This is a decent default bitrate for 1080p
state->framerate = VIDEO_FRAME_RATE_NUM;
state->intraperiod = 0; // Not set
state->verbose = 0;
state->demoMode = 0;
state->demoInterval = 250; // ms

// Setup preview window defaults
raspiview_set_defaults(&state->preview_parameters);

// Set up the camera_parameters to default
raspicamcontrol_set_defaults(&state->camera_parameters);
}

/**
 * buffer header callback function for camera control
 *
 * Callback will dump buffer data to the specific file
 *
 * @param port Pointer to port from which callback originated
 * @param buffer mmal buffer header pointer
 */
static void camera_control_callback(MMAL_PORT_T *port,
MMAL_BUFFER_HEADER_T *buffer)
{
    MMAL_BUFFER_HEADER_T *new_buffer;
    PORT_USERDATA *userdata = (PORT_USERDATA *) port->userdata;
    RASPIVID_STATE *pstate = userdata->pstate;

    int w = userdata->pstate->width;
    int h = userdata->pstate->height;

    Mat py = Mat(Size(w, h), CV_8UC1); // Y component of YUV
    I420 frame

    mmal_buffer_header_mem_lock(buffer);

    memcpy(userdata->image.data, buffer->data, w * h); // read Y

    userdata->framesRecorded++;

    // Check if the video didn't record humans in the first 3 seconds
    if(userdata->framesRecorded > 4 && userdata->framesRecorded < pstate-
>maxLength && userdata->humanDetected < pstate->consecutiveHumans)
    {
        //remove(userdata->fileName.c_str());
    }
}

```

```

        //system("poweroff");
        //cout<<"No humans on start\n";
    }
    // Check if the video is less then the maximun length and didn't have human in
    the pre determinated time
    else if(userdata->framesRecorded < pstate->maxLength && userdata-
>noHumanDetected >= pstate->consecutiveNoHumans)
    {
        //stop recording and power off
        //system("poweroff");
        //cout<<"No humans for a while\n";
    }
    // Check if the video has reached the maximum length and open a new one
    else if(userdata->framesRecorded >= pstate->maxLength)
    {
        //userdata->fileName = to_string(rand()) + ".avi";
        //userdata->fileHandle.open(userdata->fileName,
CV_FOURCC('D','I','V','X'), 3, userdata->image.size(), false);
        //userdata->framesRecorded = 0;
        //cout<<"Long enough to create a new file\n";
    }

mmal_buffer_header_mem_unlock(buffer);

if (vcos_semaphore_trywait(&(userdata->complete_semaphore)) != VCOS_SUCCESS) {
    vcos_semaphore_post(&(userdata->complete_semaphore));
}

mmal_buffer_header_release(buffer);

// and send one back to the port (if still open)
if (port->is_enabled) {
    MMAL_STATUS_T status;

    new_buffer = mmal_queue_get(pstate->video_pool->queue);

    if (new_buffer)
        status = mmal_port_send_buffer(port, new_buffer);

    if (!new_buffer || status != MMAL_SUCCESS)
        printf("Unable to return a buffer to the video port\n");
}
/***
* Create the camera component, set up its ports
*
* @param state Pointer to state control struct

```

```

*
* @return MMAL_SUCCESS if all OK, something else otherwise
*
*/
static MMAL_STATUS_T create_camera_component(RASPIVID_STATE *state)
{
    MMAL_COMPONENT_T *camera = 0;
    MMAL_ES_FORMAT_T *format;
    MMAL_PORT_T *preview_port = NULL, *video_port = NULL, *still_port =
NULL;
    MMAL_STATUS_T status;

    /* Create the component */
    status =
mmal_component_create(MMAL_COMPONENT_DEFAULT_CAMERA,
&camera);

    if (status != MMAL_SUCCESS)
    {
        vcos_log_error("Failed to create camera component");
        goto error;
    }

    if (!camera->output_num)
    {
        status = MMAL_ENOSYS;
        vcos_log_error("Camera doesn't have output ports");
        goto error;
    }

    preview_port = camera->output[MMAL_CAMERA_PREVIEW_PORT];
    video_port = camera->output[MMAL_CAMERA_VIDEO_PORT];
    still_port = camera->output[MMAL_CAMERA_CAPTURE_PORT];

    // set up the camera configuration
    {
        MMAL_PARAMETER_CAMERA_CONFIG_T cam_config =
        {
            { MMAL_PARAMETER_CAMERA_CONFIG,
            sizeof(cam_config) },
            .max_stills_w = state->width,
            .max_stills_h = state->height,
            .stills_yuv422 = 0,
            .one_shot_stills = 0,
            .max_preview_video_w = state->width,
            .max_preview_video_h = state->height,
            .num_preview_video_frames = 3,
            .stills_capture_circular_buffer_height = 0,
            .fast_preview_resume = 0,

```

```

        .use_stc_timestamp =
MMAL_PARAM_TIMESTAMP_MODE_RESET_STC
    };
    mmal_port_parameter_set(camera->control, &cam_config.hdr);
}

// Now set up the port formats
// Set the encode format on the video port
format = video_port->format;
format->encoding = MMAL_ENCODING_JPEG;
format->es->video.width = state->width;
format->es->video.height = state->height;
format->es->video.crop.x = 0;
format->es->video.crop.y = 0;
format->es->video.crop.width = state->width;
format->es->video.crop.height = state->height;
format->es->video.frame_rate.num = state->framerate;
format->es->video.frame_rate.den = VIDEO_FRAME_RATE_DEN;

video_port->buffer_size = video_port->buffer_size_recommended;

if (video_port->buffer_size < video_port->buffer_size_min)
video_port->buffer_size = video_port->buffer_size_min;

video_port->buffer_num = video_port->buffer_num_recommended;

if (video_port->buffer_num < video_port->buffer_num_min)
video_port->buffer_num = video_port->buffer_num_min;

status = mmal_port_format_commit(video_port);

if (status != MMAL_SUCCESS)
{
    vcos_log_error("camera video format couldn't be set");
    goto error;
}

/* Enable component */
status = mmal_component_enable(camera);

if (status != MMAL_SUCCESS)
{
    vcos_log_error("camera component couldn't be enabled");
    goto error;
}

// Create pool from camera still port
if((state->video_pool = mmal_port_pool_create(video_port, video_port-
>buffer_num, video_port->buffer_size)) == NULL)
{

```

```

        vcos_log_error("Error creating video pool\n");
        goto error;
    }

raspicamcontrol_set_all_parameters(camera, &state->camera_parameters);

state->camera_component = camera;

if (state->verbose)
    fprintf(stderr, "Camera component done\n");

return status;

error:

if (camera)
    mmal_component_destroy(camera);

return status;
}

/***
 * Destroy the camera component
 *
 * @param state Pointer to state control struct
 *
 */
static void destroy_camera_component(RASPIVID_STATE *state)
{
    if (state->camera_component)
    {
        mmal_component_destroy(state->camera_component);
        state->camera_component = NULL;
    }
}

/***
 * Connect two specific ports together
 *
 * @param output_port Pointer the output port
 * @param input_port Pointer the input port
 * @param Pointer to a mmal connection pointer, reassigned if function successful
 * @return Returns a MMAL_STATUS_T giving result of operation
 *
 */
static MMAL_STATUS_T connect_ports(MMAL_PORT_T *output_port,
MMAL_PORT_T *input_port, MMAL_CONNECTION_T **connection)
{
    MMAL_STATUS_T status;

```

```

status = mmal_connection_create(connection, output_port, input_port,
MMAL_CONNECTION_FLAG_TUNNELLING |
MMAL_CONNECTION_FLAG_ALLOCATION_ON_INPUT);

if (status == MMAL_SUCCESS)
{
    status = mmal_connection_enable(*connection);
    if (status != MMAL_SUCCESS)
        mmal_connection_destroy(*connection);
}

return status;
}

/***
* Checks if specified port is valid and enabled, then disables it
*
* @param port Pointer the port
*
*/
static void check_disable_port(MMAL_PORT_T *port)
{
    if (port && port->is_enabled)
        mmal_port_disable(port);
}

/***
* Set the camera configuration
*
* @param camera Pointer the camera component
* @param params Parameters to be sent to the camera
*
*/
int set_camera_parameters(MMAL_COMPONENT_T *camera, const
RASPICAM_CAMERA_PARAMETERS params)
{
    int result;

    result = raspicamcontrol_set_saturation(camera, params.saturation);
    result += raspicamcontrol_set_sharpness(camera, params.sharpness);
    result += raspicamcontrol_set_contrast(camera, params.contrast);
    result += raspicamcontrol_set_brightness(camera, params.brightness);
    result += raspicamcontrol_set_ISO(camera, params.ISO);
    result += raspicamcontrol_set_video_stabilisation(camera,
params.videoStabilisation);
    result += raspicamcontrol_set_exposure_compensation(camera,
params.exposureCompensation);
    result += raspicamcontrol_set_exposure_mode(camera,
params.exposureMode);

```

```

        result += rascamcontrol_set_metering_mode(camera,
params.exposureMeterMode);
        result += rascamcontrol_set_awb_mode(camera, params.awbMode);
        result += rascamcontrol_set_imageFX(camera, params.imageEffect);
        result += rascamcontrol_set_colourFX(camera, &params.colourEffects);
        result += rascamcontrol_set_rotation(camera, params.rotation);
        result += rascamcontrol_set_flips(camera, params.hflip, params.vflip);

        return result;
    }

/***
* main
*/
int main(int argc, const char **argv)
{
    // Our main data storage vessel..
    RASPIVID_STATE state;

    MMAL_STATUS_T status = MMAL_SUCCESS;
    MMAL_PORT_T *camera_preview_port = NULL;
    MMAL_PORT_T *camera_video_port = NULL;
    MMAL_PORT_T *camera_still_port = NULL;
    MMAL_PORT_T *preview_input_port = NULL;
    bcm_host_init();

    // Register our application with the logging system
    vcos_log_register("RaspiVid", VCOS_LOG_CATEGORY);

    default_status(&state);

    // OK, we have a nice set of parameters. Now set up our components
    // We have two components. Camera and Preview

    if ((status = create_camera_component(&state)) != MMAL_SUCCESS)
    {
        vcos_log_error("%s: Failed to create camera component", __func__);
    }
    else if ((status = raspipreview_create(&state.preview_parameters)) !=
MMAL_SUCCESS)
    {
        vcos_log_error("%s: Failed to create preview component", __func__);
        destroy_camera_component(&state);
    }
    else
    {
        PORT_USERDATA callback_data;

        srand (time(NULL));

```

```

    //size of the image processed by the OpenCV
    float scale = 3;
    state.opencv_width = 1280 / scale;
    state.opencv_height = 720 / scale;

    float scale_width = 1280 / state.opencv_width;
    float scale_height = 720 / state.opencv_height;

    /* setup opencv */
    callback_data.image = Mat(Size(state.width, state.height), CV_8UC1);
    callback_data.image2 = Mat(Size(state.width, state.height),
CV_8UC1);

    //set up video file
    callback_data.fileName = to_string(rand()) + ".avi";
    VideoWriter record(callback_data.fileName,
CV_FOURCC('D','T','V','X'), 2.5, callback_data.image.size(), false);
    VideoWriter record2("Demo_" +callback_data.fileName,
CV_FOURCC('D','T','V','X'), 2.5, Size(state.opencv_width, state.opencv_height),
false);
    callback_data.fileHandle = record;
    callback_data.fileHandle2 = record2;

    if (state.verbose)
        fprintf(stderr, "Starting component connection stage\n");

    camera_preview_port = state.camera_component-
>output[MMAL_CAMERA_PREVIEW_PORT];
    camera_video_port = state.camera_component-
>output[MMAL_CAMERA_VIDEO_PORT];
    camera_still_port = state.camera_component-
>output[MMAL_CAMERA_CAPTURE_PORT];
    preview_input_port = state.preview_parameters.preview_component-
>input[0];
    RASPICAM_CAMERA_PARAMETERS paramsCamera;

    paramsCamera.sharpness = 0;
    paramsCamera.contrast = 0;
    paramsCamera.brightness = 50;
    paramsCamera.saturation = 0;
    paramsCamera.ISO = 400;
    paramsCamera.videoStabilisation = 0;
    paramsCamera.exposureCompensation = 0;
    paramsCamera.exposureMode =
MMAL_PARAM_EXPOSUREMODE_AUTO;
    paramsCamera.exposureMeterMode =
MMAL_PARAM_EXPOSUREMETERINGMODE_AVERAGE;
    paramsCamera.awbMode = MMAL_PARAM_AWBMODE_OFF;
    paramsCamera.imageEffect = MMAL_PARAM_IMAGEFX_NONE;
    paramsCamera.colourEffects.enable = false;

```

```

paramsCamera.colourEffects.u = 128;
paramsCamera.colourEffects.v = 128;
paramsCamera.rotation = 180;
paramsCamera.hflip = paramsCamera.vflip = 0;

set_camera_parameters(state.camera_component, paramsCamera);

if (status == MMAL_SUCCESS)
{
    // Set up our userdata - this is passed though to the callback
    where we need the information.
    callback_data.pstate = &state;
    callback_data.abort = 0;
    callback_data.humanDetected = 0;
    callback_data.noHumanDetected = 0;
    callback_data.uploading = false;
    callback_data.framesRecorded = 0;

    camera_video_port->userdata = (struct
MMAL_PORT_USERDATA_T *)&callback_data;

    // Enable the video port and tell it its callback function
    status = mmal_port_enable(camera_video_port,
camera_control_callback);

    if (status != MMAL_SUCCESS)
    {
        vcos_log_error("Failed to setup video");
        goto error;
    }

    // Send all the buffers to the camera video port
    {
        int num = mmal_queue_length(state.video_pool-
>queue);
        int q;

        for (q = 0; q < num; q++)
        {
            MMAL_BUFFER_HEADER_T *buffer =
mmal_queue_get(state.video_pool->queue);

            if (!buffer) {
                printf("Unable to get a required buffer %d
from pool queue\n", q);
            }

            if (mmal_port_send_buffer(camera_video_port,
buffer) != MMAL_SUCCESS) {

```

```

        printf("Unable to send a buffer to video
port (%d)\n", q);
    }
}
}

//Start image capture
if (mmal_port_parameter_set_boolean(camera_video_port,
MMAL_PARAMETER_CAPTURE, 1) != MMAL_SUCCESS)
{
    goto error;
}

vcos_semaphore_create(&callback_data.complete_semaphore,
"mmal_opencv_demo-sem", 0);
int opencv_frames = 0;

//Background subtraction variables
BackgroundSubtractorMOG2 bg;
bg.set("detectShadows", true);

//Configuration parameters from the configuration file
int dilate_n = 5;
//Number of Dilates
int erode_n = 1;
//Number of Erodes
state.maxLength = 900; //10
minutes of video approximately (seconds * 3)
state.consecutiveHumans = 1;
state.consecutiveNoHumans = 45; //15 seconds
without human on the scene (seconds * 3)

//Variables of execution
bool firstRead = false;
bool exposure = false;

while (1) {
    if
(vcos_semaphore_wait(&(callback_data.complete_semaphore)) ==
VCOS_SUCCESS) {
        opencv_frames++;

        //Turn off exposure after the initialization
        if(opencv_frames == 2 && exposure == false )
        {
            paramsCamera.exposureMode =
MMAL_PARAM_EXPOSUREMODE_OFF;

            set_camera_parameters(state.camera_component, paramsCamera);
            exposure = true;

```

```

        }

        //Read configuration file
        if(openCV_frames >=
OPENCV_CONFIG_FRAMES || firstRead == false)
{
    string line="";
    ifstream file;

    file.open("config");
    if (!file.is_open())
    {
        cout<<"Error opening
configuration file\n";
    }

    while(!file.eof())
    {
        string parameter;
        string value;

        getline(file, line);
        if(line.length()>=3)
        {
            parameter =
line.substr(0,line.find('='));

            value =
line.substr(line.find('=')+1);

            if(parameter == "dilate_n")
{
                int aux;
                istringstream (
value ) >> aux;
                dilate_n=aux;
            }
            else if(parameter ==
"erode_n")
{
                int aux;
                istringstream (
value ) >> aux;
                erode_n=aux;
            }
            else if(parameter ==
"maxLength")
{
                int aux;
                istringstream (
value ) >> aux;
            }
        }
    }
}

```

```

state.maxLength=aux;
}
else if(parameter ==
"consecutiveHumans")
{
    int aux;
    istringstream (
value ) >> aux;

state.consecutiveHumans=aux;
}
else if(parameter ==
"consecutiveNoHumans")
{
    int aux;
    istringstream (
value ) >> aux;

state.consecutiveNoHumans=aux;
}

}

}

file.close();
if(!firstRead)
firstRead = true;
else
opencv_frames = 0;
}

//Resizes the userdata.image and saves on
userdata.image2 with the size of userdata.image2
Mat aux;
resize(callback_data.image, aux,
Size(state.opencv_width, state.opencv_height), 0, 0, CV_INTER_LINEAR);

//BG Subtract
Mat fore;
bg.operator() (aux,fore);

//Morphological transformations
erode(fore, fore, Mat(), Point(-1,-1), erode_n);
dilate(fore, fore, Mat(), Point(-1,-1), dilate_n);

//Find contours
vector<vector<cv::Point> > contours;

findContours( fore, // binary input image
contours, // vector of vectors of points

```

```

CV_RETR_EXTERNAL, // retrieve only
external contours
contours

//Verify if the contour is a human
int human = 0;
for(int i=0;i < contours.size();i++)
{
    double arc = arcLength(contours[i], true);
    cout<<arc<<"\n";
    if(arc > 100)
    {
        //Apply bounding box
        Rect body_i =
boundingRect(contours[i]);

        //check if the height is bigger than
the width
        if(body_i.height > body_i.width)
        {
            rectangle(aux,
body_i,
Scalar(0,255,255),
2
);
            human++;
        }
    }
}

//Add human detected
if(human > 0)
{
    callback_data.humanDetected++;
    callback_data.noHumanDetected = 0;

    cout<<"Humans
detected:"<<callback_data.humanDetected;
}
//Add human not detect
else
{
    callback_data.humanDetected = 0;
    callback_data.noHumanDetected++;
}
//Demo
callback_data.fileHandle2 << aux;

```

```

        //Generate video in grey scale.
        callback_data.fileHandle <<
callback_data.image;
    }
}
else
{
    //mmal_status_to_int(status);
    vcos_log_error("%s: Failed to connect camera to preview",
__func__);
}
}

error:
printf(stderr, "Close down completed");
}

return 0;
}

```