Raspberry Pi: a Smart Video Monitoring Platform

David Emanuel Ribeiro Gaspar
Instituto Superior Técnico
November 2014

I. INTRODUCTION

A. Motivation

In about fifteen years, the computing world has changed its focus. While back then, domestic CPUs started to hit 3GHz and above frequencies [1], more recently, parallel computing started to take over as multi-core processors started becoming a priority [2] - it was obvious that bigger benefits could be obtained not by doing things sequentially faster, but by doing several at a time.

More recently, another paradigm change has been taking effect. Mobile processors have seen a gigantic boost in market demand with low-power with ARM processors taking the spotlight from full-fledged x86 units [3]. Apart from the high-end, high-price flagship models, a trail of low-cost alternatives has been left behind which led to a huge increment in popularity for single-board computers. These are very small, low-power computers that offer good performance for a relatively low cost.

Combining a single-board computer and a web-cam, a small, low-cost video processing station can be built and perform a series of functions that can be divided into two groups:

- **Event Triggering**: Software that analyses each frame looking for a particular pattern. Upon detection of that pattern, a series of actions is performed.
- **Over-time collection of data**: Software that analyses every frame and produces an output in the end, like a heat-map: an image calculated from the video feed that indicates, with cool and warm colours, the zones with, respectively, the least and most movement.

The system will be developed with three use cases:

- **Event Triggering by Movement Detection** When overviewing a static scene and movement is detected, an event should be triggered, with the local logging of the timestamp and snapshot.
- **Activity Mapping** Overlooks a scene for a period of time and, at the end, produces a heat map representative of the areas of the scene where the most movement occurred. Two plug-ins will be developed under this premise: one that frequently produces a heat map indicative of the whole activity level since the last map was produced, and another that produces a map at every frame, representative of the activity in the feed for the latest few seconds or minutes.

The system will have the following objectives:

- **Low-Cost** The system’s total cost should be as low as possible.
- **Flexibility** The system should be as flexible and have as little ties to the hardware as possible.
- **Automaticity** The system, both the platform and plug-ins, should fulfill its premise with as little human intervention as possible.

Fig. 1: A Raspberry Pi board.

Peripherals like web-cams have also seen an enormous progress in recent times, providing nowadays excellent video quality and low-cost.

Fig. 2: An example of a heat map.
- **Size** The system should be as small as possible.
- **Versatility** The system should be able to work with good results under various conditions.

The following requisites must be met:
- The system should run on a Raspberry Pi board and, as such, inherits its basic working requirements - a nearby power outlet or battery and an SD memory card containing an appropriate Linux distribution. In order to power both the board and the web-cam, a powered Universal Serial Bus (USB) hub is necessary.
- The web-cam device driver must be an USB Video Device Class (UVC) and compatible with the Video4Linux2 (V4L2) API.
- The camera should be in a fixed position and the background of the image should be as static as possible. The most static the background is, the best the results will be. Also, the area under surveillance should be relatively well lit.
- For triggering events that need network connectivity, it should be provided, whether wired or wireless. None of the implemented plug-ins presented in this document make use of network connectivity, but it must be provided for further flexibility in plug-in development.

II. EXISTING SOLUTIONS

Some systems already include some of the functionalities the system herein described will implement.

A. Security Video-Surveillance

Most of these systems are typically composed of three components: a set of cameras, a central unit with Digital Video Recorder (DVR) capabilities and an interface for viewing and managing footage. The cameras are usually fixed and connected to a wall power outlet. The central unit stores the video taken from each camera in its hard-disk and keeps it for a set period of time.

These systems have a few limitations, the first being lack of practicality: installation or relocation of cameras is not an easy process. The second limitation is image quality, as these systems are very low cost and imaging is often a sacrificed aspect. The last drawback is lack of flexibility - the system is limited to video surveillance tasks. The Swann 3425 Series is an example of such systems.

Logitech Alert takes a different approach by using higher-quality cameras and providing each camera with its own storage. The software solution includes both a mobile application with remote notifications and a web interface for live feed viewing. The cameras use the same connection for data and power transfer.

This alternative also has its shortcomings. It was designed for security uses, not being adaptable for other use cases. It is also not cheap - a base system is, at the time of this writing, priced at $322.98.

B. Sports Data Gathering

Recently, new systems like those of Opta Sports have been developed that allow for the gathering and analysis of a range of data from a sports event. By pointing one or more cameras at a pitch and then making use of image recognition algorithms, data can be gathered and individual heat maps can be generated for each player, and a general one for the whole team.

Such examples may need to employ moderately complex algorithms. In order to build a player’s heat map, the system needs to be able to identify the area of the video pertaining to that player and follow it around the pitch. A potentially more complicated issue is how to know which area of the video feed corresponds to each player. Facial recognition algorithms need to be applied or, alternatively, the previous identification of each player, in which case the manual work of a human is necessary.

These systems employ proprietary architecture and implementation details and are, thus, very difficult to obtain information about and document. The description made here is, then, but a speculation on the implementation such a system can use.

C. Wilderness Cameras

A different application of a camera when connected to an embedded device are wilderness cameras. These are based on a weather-proof case enclosing a camera, a motion sensor, a memory card for local storage, a battery and an embedded system. When installed outdoors, the motion sensor looks for movement and, upon detection, photos or videos are taken. The purpose of this behaviour is to capture footage of wildlife.

![Fig. 3](Swann_DVR8-3425.jpg) Swann DVR8-3425 - an example of a video surveillance system with a set of cameras and a central unit.

![Fig. 4](camera.jpg) An example of a motion-triggered wilderness camera.
These systems are architecturally very simple and limited in their use, but they are also very well tailored for it. The presence of a dedicated motion sensor is one such indication, allowing for very good battery usage. However, they are also lacking in flexibility.

D. Human Monitoring Systems

Some systems are available whose goal is to collect data about the usage of public spaces, like stores or supermarkets. A set of cameras is installed and then run continuously while the space is open. Heat maps can be calculated to assess the level of each area’s movement.

While not a lot of information is available on these systems apart from the offered functionality, it is expectable that their basic architecture is in some points similar to the video surveillance systems: a series of cameras connected to a central device.

Cost and lack of flexibility are concerns for these systems as well. RetailNext[9] and Prism SkyLabs[10] are two examples.

E. Community Approaches

Users and hobbyists have built custom video applications with the processing board. Timelapse videos[11], security systems[12] and motion-triggered cameras[13] are a few examples. These systems have very limited functionality, are usually difficult to configure and depend on their creator’s motivation. Many have completely halted development.

III. RELATED TECHNOLOGY

In this section, the available options for the hardware layer are listed. Software-wise, the functioning of the Frame-Differencing algorithm is explained.

A. Hardware Platforms & Peripherals

1) Processing Platform: In order to build, test and run a complete software package, a hardware platform must be chosen. The two most obvious options are conventional computers and single-board computers.

Conventional computers often take one of two forms - factors: desktops and laptops. Desktops are composed of a central tower and a series of peripherals attached to it. Laptops are much smaller than desktops and already include a keyboard, screen, touchpad, battery and speakers.

A specific type of laptops are netbooks - computers that focus on the core functionalities of a laptop. These computers manage to have even smaller sizes while keeping acceptable levels of performance.

The other option are single-board computers, small electronics boards that share a common basic architecture with a conventional computer. USB and HDMI ports are common in these devices, so usual peripherals like a keyboard, mouse and screen can be connected. They are commonly run by a system-on-chip containing an ARM processor.

There are several models of single-board computers in the market, the most popular being the Raspberry Pi. An alternative is the Intel Galileo[14]. For $70 dollars it offers a 400MHz 32-bit x86 Intel processor, a 10/100 Mbps ethernet connection, two USB 2.0 ports and 256MB RAM.

A third option is the BeagleBone Black[15], which includes a 1GHz ARM Cortex-A8 processor and 512MB of RAM for about 50 euros.

B. Detection Algorithm

Each one of the three modules listed in the Introduction needs its own processing algorithm - the definition of internal structures and how to update them according to the frames coming from the video feed, and how to calculate the final result. All modules share a common core - the Frame Differencing algorithm[16]–[18].

It works by saving one frame - the Reference Frame - in memory and by comparing the new ones to it. Since a frame is basically a matrix of values (usually each pixel being represented by an integer in the range [0; 255]), to compare the two frames, each pixel \( R(x, y) \) in the reference frame is compared to the corresponding pixel \( C(x, y) \) in the new frame by calculating the absolute value of the difference between them two, thus calculating a difference frame. This frame contains only non-negative values and indicates how much change in value there was in each pixel, and can be used to infer whether there was an intrusion, how big it was and in what zones of the video frame it happened.

Its simplicity means that in its simplest form, the Frame Differencing algorithm will provide inadequate results in all but the most trivial applications. Intrusion detection in a room with networking equipment will constantly trigger false positives due to the constant blinking lights. If the camera is overseeing an outdoors scene, changes in lighting and shadow position will also trigger false positives. When used in dark places, the camera will increase the sensor sensibility to make for a clearer image - noise becomes a constant throughout the image, meaning changes in a lot of the frame’s pixels. Several aspects of the algorithm can, fortunately, be adapted and parameterized.

The first and most intuitive is to set a minimum threshold on the number of pixels with difference. By setting this parameter, problems like false positives caused by small changes are minimized, as these cases are usually confined to a small fraction of the frame.

The second parameter is the minimum change in value a pixel has to have in order to be counted towards the previous parameter. By imposing a minimum limit like, for example, 15% in change, problems like the variation of most pixels in images with high sensor sensitivity can be solved, since most of these pixels vary in value but not by a long margin. The usage of this parameter, by itself, is not effective, as a single pixel with a big change is enough to trigger a false positive. Usage of this parameter in combination with the previous one is safer.

If the first parameter is met (presuming the second has too), the frame is set as a detection frame, meaning a frame including an intrusion. A last parameter can be set: the minimum number of consecutive detection frames required...
for a warning to be triggered. By setting a minimum period of time where detection frames have to be consecutively detected, short but swift changes can be ignored.

The solution to gradual lighting changes is to adopt a sliding reference frame mechanism - the reference frame is not static and can be periodically refreshed during execution. Through this mechanism, new frames are compared to one that is chronologically closer to it and such problems can be solved. The parameter here is how often the reference frame should be refreshed, or if not at all.

An important point regarding these parameters (or thresholds) is that there is no one size fits all set of values that is appropriate to all applications. Some manual intervention and fine-tuning can be necessary.

Another way in which Frame Difference’s simplicity is a plus is that it makes it fast. The whole algorithm relies on very simple mathematical operations (subtraction and absolute value) and memory pointer manipulation. Also, most of the computational effort of the algorithm is the calculation of the difference frame - a difference between two matrices. This is a series of subtractions where the values are not dependent on each other, so it can be safely divided into several working threads. By off-loading this effort to the Graphics Processing Unit (GPU), very high speed-ups can be obtained.

Hence, the Frame Differencing algorithm is very versatile, being usable in both simple detection and over-time data collection processing modules. At their most basic level, all three developed Processing Modules use the Frame Differencing algorithm.

C. Discussion

1) Hardware: The choice for the processing platform on which to develop and run the software came down to three categories: desktop computers, laptops and single-board computers.

In their advantage, desktops have their superior performance and upgrading options. Desktop CPU’s usually run at higher voltages than their equivalent laptop model and components are usually cheaper, performance-wise. Desktop motherboards have wide expansion options, making it easy to increase the amount of system memory, system storage or graphics capabilities. But these systems are big and heavy, compromising their use in a mobile-focused solution.

Laptops appear as an apparently viable option, as they are much smaller, have a few hours of autonomy and provide very good performance, but they are not made to be used outside.

Single-boards manage to have acceptable performance, good connectivity and have a wide range of accessories. They are also much cheaper than any available computer. The choice the Raspberry Pi, given its wide availability and easy to find support.

2) Software: The Frame Differencing algorithm, despite its simplicity, is the best suited for most event-triggering applications. The rapid subtraction between frames allows for fast checking of movement and, provided it uses a well-tuned algorithm, can return very good results. It is expected, however, that some manual tuning may be necessary depending on the observed scenario.

IV. PROPOSED ARCHITECTURE

A. Hardware

This section focuses on the two components of the system’s hardware layer: the processing board and its resources and its connected peripherals.

![Hardware schematic of the system.](image)

1) Processing Board & Resources: The processing board is the central piece powering the system. It runs all of the implemented software and algorithms. Having such an architecture allows the software it runs to have access to its computational resource, the first being computational capabilities. The processing board will usually have a Central Processing Unit (CPU) - to allow the usual mathematical computations - and system memory, more commonly known as Random-Access Memory (RAM), which is where the board’s operating system and programs are loaded. Each program has access to a fraction of that memory. Network and USB interfaces are present[4] to allow for remote communication and peripheral connection.

2) Peripherals: In the specific case of the Raspberry Pi, four types of peripheral connections are provided: multimedia interfaces (HDMI, composite and 3.5mm jack interfaces) and USB ports to allow the inclusion of cameras USB hubs and wireless network connections. GPIO and MIPI interfaces are present but will not be used.

B. Software

The software layer of the system is composed by three main modules: the Capture Module, the Processing Module and a small Utilities Module. The capture and processing modules each represent a specific set of behaviors and functions in the system that any implemented instance must provide. Their cooperation and correct coordination are vital to the system, so they both use a common Communication Protocol.
1) Capture Module: The capture module works between the processing module and the source of the video feed, acting as an abstraction layer. It connects to the source of the video feed - be it a camera, a file or a network stream -, obtains the data and exposes a set of functionality to the processing module for it to obtain the images organized in frames. Furthermore, the capture module implements a series of pre-processing algorithms to apply to the video feed before the frames are passed. Examples that have been included are supported in the communication protocol are frame downsampling - reduction of an image's resolution -, histogram equalization - to improve contrast - and colour to monochrome conversion - loss of colour information.

Two capture modules were implemented for this project: one that fetches the data from a USB web-cam and one that obtains it from a local raw YCbCr file. The transition between the two should be nearly transparent to a Processing Module.

Capture from USB Web-cam: captures real world footage from a USB camera. To perform the connection and communication with the device, it uses V4L2.

Two main tasks are performed: querying of device capabilities and image data obtainal. With this information, the module receives the Processing Module’s request for video configuration (a process further explained in the Communication Protocol section) and returns the set that most closely follows this set. After the parameters have been agreed upon, this module queries the device for image data and returns it to the processing module. V4L2 offers mechanisms to support both of these tasks.

Capture from Local File: throughout development, it became apparent that it would be useful to sometimes save video sequences for later use. This version of the capture module was developed to allow this task.

2) Processing Module: The Processing Module is the central part of the software component of the system that it is where the use case is implemented. The processing module communicates with the capture module and obtains the video frames. By reading these frames, updating internal structures, performing algorithms over them and using the processing board’s resources. Being a common program, the Processing Module has access to all of the board’s resources. Being a common program, the Processing Module has access to all of the board’s resources. Being a common program, the Processing Module has access to all of the board’s resources.

Video Capture: captures the frames to a local file and is the simplest implementation of all four. Upon arrival of a frame, it goes through an optional processing stage. The resulting frame is immediately output to the file and the system is ready for the next frame.

Simple Movement Detection: watches over a video feed and uses the Frame Differencing algorithm to detect foreign movement. Upon detection, a pre-defined event is executed.

Starts by updating the reference frame - the frame newer ones are compared to. The update mechanism is run at the start of execution and when a sliding reference frame mechanism is used and the time threshold for reference frame renewal has been surpassed. Subsequent frames that do not require a reference frame renewal are compared to the existing reference frame and a difference frame is calculated. This difference frame is fed to the Movement Detection stage, where the intelligence behind intrusion detection - discussed in the Frame Differencing section is encapsulated. If an intrusion is detected, a series of snapshots is saved.

Activity Mapping: watches the video feed for a period of time and builds a heat map - a scale of colours ranging from cool to warm tones indicating the zones with the least and most movement.

The output consists of the heat-map overlaid on a screenshot from the video feed, for easier identification of the zones in the feed the thermal map is indicating. An option is available to break the execution each X seconds, so that a series of heat maps are generated instead of a single one. This algorithm works with an adaptation of the Frame Differencing algorithm.

This module works, to a certain point, similarly to the Simple Movement Detection module, but a new structure is kept and updated: the accumulation map. This is a matrix the same resolution as the video feed’s frames, but capable of storing a much bigger number than the pixel’s maximum 255 value in each of its cells. It keeps track of how much change in value the pixel corresponding to each cell saw throughout execution. At the end of execution, the values within this structure are then translated to colour values, originating the heat map.

Activity Mapping with Gradual Updating: the Activity Mapping with Gradual Updating is similar to the Activity Mapping, but with a different approach - it is more dynamic and takes a different approach to the updating mechanism in that a parameter X is set and each frame is compared to each of the X frames that came before it. The parameter X defines the number sequentially older frames that the newest one will be compared to.

The end result of this module is the generation of the heat map of the movement registered by the system during the last X frames.

V. Implementation

The implementation of the project had two main points of focus: the Raspberry Pi’s very limited computing capabilities and the project’s objective of being usable in the real world. Efficiency was a constant point of focus during development.

VI. General Structure

The first choice was which programming language to use in the system’s implementation. It was important to choose
one that helped meet the project’s goals - a fast, efficient, widely available language was necessary as good memory management is crucial to ensure efficiency and speed standards were met. These goals narrow the array of choices to two languages - C and C++ - both known for the speed of their executables, stability and good memory management and the most popular Raspberry Pi Linux distribution - Raspbian - includes compilers for both (gcc and g++). C was the chosen language given the author’s greater previous experience with it and the fact that it is easier to implement a C compiler than a C++ one, making the system more easily portable to other platforms.

Three files implement the project’s basic structure: capturemodule.c (Capture Module), processingmodule.c (Processing Module) and utils.c (Utilities Module). The Capture and Processing modules are swappable - any file that correctly implements the communication protocol can be used, and herein lies the swappable modules aspect of the system, that allows it to have a big flexibility on video feed source and frame processing use case.

- **capturemodule.c** An abstraction layer between the Processing Module and the source of the data, implementing the functions that allow it to obtain this video feed, organize its frames and perform the necessary pre-processing and conversion, according to the agreed parameters with the Processing Module. The pre-processing stage refers to a series of algorithms that can be applied on the obtained frame, prior to its return to the Processing Module. The developed version includes two algorithms - the Processing Module decides which ones to use.

- **Frame Downsampling**
  Reduces an image’s resolution. Depend on the downsample factor - a positive integer that defines by how much both the image’s width and height are to be divided. Downscaled resolution is given by:

\[
\text{downscaledResolution} = \frac{\text{nativeHeight}}{\text{downscaleFactor}} \times \frac{\text{nativeWidth}}{\text{downscaleFactor}}
\]

This algorithm works by dividing the original frame into `downscaledResolution` squares, each `downscaleFactor * downscaleFactor` pixels big. Each square in the original frame corresponds to a pixel in the equivalent position in the downscaled frame - this pixel will contain the equivalent to the average pixel value in the original frame’s square. All the pixel values in the original original image’s square are summed and then divided by `downscaleFactor^2`.

A `getFullResolutionFrame()` method is available to provide a frame in native resolution.

- **Histogram Equalization**
  Increase the image’s contrast, making intrusion detection easier. Essentially recalculates the image’s pixel’s values to that they are better distributed throughout the image’s histogram - a graphical representation of the distribution of the image’s pixels throughout their range values. The x-axis in a monochrome picture with a byte per pixel ranges from 0 to 255 and indicates how many of the pixel’s values have that value. The image that originated the above histogram had a majority of grey pixels and very few near white or black. The first step in this process is the calculation of of cumulative distribution function, given as

\[
cdf_x = \sum_{j=0}^{p_x(j)}
\]

with \( p_x(j) \) being the image’s histogram for pixel value \( i \) normalized to [0;1]. The function to calculate a pixel’s current value to its new one is given by

\[
cdf_y(y') = cdf_y(T(k)) = cdf_x(k) \times 255
\]

The Capture Module exposes two functions to obtain the current frame from the video feed: `getFrame()` and `getFullResolutionFrame()`. `getFrame()` gets a frame from the video feed with all previously selected pre-processing applied, as well as downscaling with previously agreed parameters. `getFullResolutionFrame()` works similarly but ignoring the downscaling factor. Neither of these functions has a return value; all frame content is written to the same memory location, previously known by the Processing Module.

- **procmodule.c** The code that implements the system’s use case. Uses the aforementioned Capture Module functions to obtain frames and processes them. A number of specific processing module examples were implemented; those are further analysed later in this chapter.

- **utils.c** A series of utility functions. The frame downsampling algorithm is implemented here, as well as a function to convert from the HSV to YCbCr colour space.

A. Capture from USB Device

The most important and frequently used capture module is its implementation that connects to a USB camera and fetches frames from a real world scene. To connect to the Logitech webcam, the V4L2 API is used, parameterized with the execution parameter struct’s devName string.

The API supports several methods of reading the data off the device (read and write methods, asynchronous I/O methods and streaming I/O), but the one that is most frequently used is the Streaming I/O method. The module uses these functionalities to determine the camera’s supported data reading methods, image resolutions and image formats and, with this data, calculating the appropriate response to the processing module’s request for image resolution and format.

B. Capture from Sequence File

A module that, through the use of standard stdio.h functions like fopen() and fgetc(), reads a sequence file off the local file system and implements the capture module’s usual
API layer over it, organizing the data in frames. The set of offered functions and functionalities is exactly the same as the other capture module, making the transition nearly transparent from the processing module’s point of view - it should only be noticeable since the frame rate retrieval might be much higher and sequenceName is used to index the file instead of devName.

However, only uncompressed YUV files are supported - plans for the future include the implementation of a video decompressing algorithm in a way that is also fully transparent to the processing module.

C. Video Storage

Through the use of the fwrite() function, this module fetches frames from the capture module (usually the one that connects to a USB camera) and saves them to a file in the local file system. The name of the file where the video will be saved is parameterizable.

Plans for the future for this module include the usage of a video compressing algorithm.

D. Movement Detection

Detects movement of any kind in the video and takes a snapshot. Uses the Frame Differencing algorithm.

Starts by allocating memory to hold a downscaled frame - the reference frame other frames will be compared to - , the fetching of a frame and the copy of its contents into the allocated memory. An optional renewReferenceFrameSeconds sets a number of seconds after which a new frame must be fetched and set as the new reference frame. Known as reference frame sliding mechanism, this method allows the module to adapt to slowly changing conditions while minimizing false positives.

The main cycle is based on the successive fetching of frames and their comparison to the reference frame. If a series of parameters is met, an intrusion is detected and a snapshot is saved. The comparison between frames is performed by comparing pixels in the same position with the standard math.h abs() function, that calculates the absolute value of the difference between the two values. If this difference equals or surpasses the pixelValueDifferenceThreshold parameter, it is categorized as a difference pixel - a pixel that saw a substantial enough variation in value to be categorized as a pixel with a possible intrusion.

The comparison continues throughout the frame and the number of difference pixels is accounted for. A second parameter, framePixelsDifferenceThreshold, sets a minimum percentage of the frame’s pixels that must be categorized as difference pixels for the whole frame to be recognized as a difference frame, ie, a frame with an intrusion. A third parameter - secondsWithIntrusion sets the minimum number of seconds filled with consecutive difference frames for a definite intrusion to be recognized and an event to be triggered - the saving of a snapshot.

A last parameter called secondsBetweenSnapshots allows the limiting of the taking of snapshots to one every X seconds, so as not to cause the saving of an excessive number of images.

E. Activity Mapping

Watches over a video feed and, in the end of execution, builds a heat map indicative of the areas with the most movement in the feed.

The most important structure in the Activity Mapping module is the accumulation map - a matrix, the same resolution as a downsampled frame, that keeps track of how much change each pixel saw throughout execution. Since one of the goals of the system is its ability to run for long periods of time without human intervention, the accumulation map is a matrix of unsigned long integers.

The algorithm starts by obtaining two frames: one in native resolution, to be later used in the finished thermal image, and one in downsampled resolution, to be used as a reference frame. Frames are then obtained in the usual fashion and compared to the reference frame, pixel by pixel, by calculating the absolute value of the difference between the two pixels and adding it to the corresponding position in the accumulation map. This module also implements a pixelValueDifferenceThreshold so that only pixel value differences above this level will be logged. In this case, the value added to the accumulation map is the difference minus the threshold.

The cycle continues until execution is manually interrupted or a renewReferenceFrameSeconds parameter is surpassed. In this case, the current accumulation map is transformed into a heat map, output to a file and execution restarts until X seconds have passed again. The most important task of the thermal map generation procedure is to take the accumulation map and map the values it contains to a range of colours, with a direct correlation between value and colour warmth. This was implemented by using the Hue-Saturation-Value colour space, which characterizes colours by a set of three parameters: the Hue, which defines the colour’s wave length, the Saturation and the Value, the colour’s lightness. The Hue of a colour is defined as a value between $0^\circ$ and $360^\circ$.

![Fig. 7: HSV colours according to H and V values. S is set to 1.0 in this figure.](image)

The cooler colours are present at $H=180^\circ$ and the warmer ones at $H=0^\circ$. Both S and V are decimal values between 0.0 and 1.0 - for this project, both are set at 1.0.

The next step is to build the final image, which will be the result of the juxtaposition of the thermal imaging over the native resolution snapshot that was taken in the first step of the execution. This image contains only Y values and is iterated through, with its values being the final thermal map’s Y values, multiplied by a accumMapWeight float.
parameter before being output to the final file, so that greater emphasis is given to the thermal colours. The last step is the inclusion of the thermal colours in the final image. If the frame downscale factor is bigger than 1, the accumulation map’s resolution is smaller than that of the original, native-resolution snapshot. If this is the case, the upscaling algorithm must be performed in order to match the colors derived from the accumulation map to the full-resolution snapshot. This is based on a mathematical operation where the pixels in the original, downscaled-size image $D$ are mapped to the upscaled image $U$ through the equation

$$U(X,Y) = D(Floor(\frac{U_x}{\text{downscale factor}}), Floor(\frac{U_y}{\text{downscale factor}}))$$

After this algorithm is run, the calculated colours are output to the resulting file and the execution is complete. In the case of a 32-bit system (as is the case with the Raspberry Pi), an unsigned long integer is 4 bytes long. Considering a typical case where a native frame is 640 by 480 pixels wide and the downscale factor is 4, a downscaled frame has a resolution of 160 by 120 pixels, so this structure takes up 75KB of memory, which can be demanding in very-low-capacity systems. However, it is vital that the system is able to run unattended for long periods of time. If we assume that a pixel’s value is at most 255, a minimum difference threshold of 30 and the system is running at a pace of 10 frames per second, the fastest a position in the accumulation map can overflow is over 22 days - a long period of time that is very unlikely to be achieved.

### F. Activity Mapping with Dynamic Reference Updating

Is an evolution of the regular Activity Mapping module in that the heat map is calculated after each difference frame and represents the movement registered during the last `queueLength` frames. There is no native resolution frame in this algorithm.

At the start of execution, two circular lists of `queueLength` elements are built. The first contains frames and the second contains difference frames and each contains two pointers: one to the next element in the list, and another to data the same size as a downscaled frame. An accumulation map is also kept. When the first frame arrives, the whole frame list is filled with it.

Every time a frame is obtained, the frame list is updated by discarding the oldest frame in it and saving the new one in the first position. A new difference frame is calculated between the newest (and just obtained) frame and the oldest one and saved in the difference frame list and its oldest structure is thrown out. Lastly, the newest difference frame is added to the accumulation map and the oldest is subtracted from it. The current accumulation map is mapped to a heat map and output to a file, at which point the cycle restarts.

The effect is a file with a series of heat maps, each representative of the movement during the last `queueLength` frames.

### VII. Experimental Results

In order to assess the quality of the system’s results, each of the three main processing modules - Movement Detection, Activity Mapping and Activity Mapping with Dynamic Reference Updating - was tested to evaluate its raw speed and application in a real world scenario. All the tests were performed on the Raspberry Pi.

To test speed performance, a specific metric was used: dropped frame percentage. All tests were performed with a video feed obtained in real time from the USB camera with no movement - frame drops are not logged during event execution or heat map exporting, so this was the best solution to ensure all module were run under the same circumstances.

All three modules were tested on real world video sequences.

Both Activity Mapping Modules were tested by analysing two previously saved video sequences. The first shows a car park with maneuvering cars and passing people:

![Car park sequence with a highlighted moving car.](attachment:image1)

The second shows a roundabout with passing cars and people on the sidewalk:

![Roundabout sequence.](attachment:image2)
A. Movement Detection

The Movement Detection module’s speed performance was tested by varying three of its parameters: the Downscale Factor was set to 1, 2 and 4 (making the program analyse frames at, respectively, 640*480, 320*240 and 160*120), the Histogram Equalization algorithm was tested on both on and off settings and the frame analysis rate was set at every value in the [2;20] range.

Regardless of the applied downscale factor, at about 8 frames per second, at least 10% of the acquired frames will not be processed, so under these conditions, 7 frames per second is the fastest the system shows acceptable performance and even then, when no downscaling is performed, about 15% of frames are lost.

Regarding the downscale factor, until about 14 frames per second, the downscaling algorithm appears to have a positive effect on system performance, with lower drop rates for higher downscale factors - despite the frame downscaling algorithm being somewhat computationally expensive. This seems to indicate that the bottleneck in frame acquiring speed are the memory access times.

Next, the effect of the Histogram Equalization algorithm was assessed. This time, the results are much more uniform, with the frame histogram equalization having a negative effect on performance - with the algorithm turned on, at 5 frames per second, drop rates have nearly reached 50%, with similar rates with the algorithm turned off not reaching such levels until frames are being acquired twice as fast.

This conclusion makes sense - this algorithm provides no benefit in performance, merely adding more work to each frame.

The Movement Detection Module was also tested by pointing the camera at a hallway and having a person cross the scene at random points in time and, after execution, analysing the taken screenshots. The system took screenshots at the correct times; an example of a result is visible below:

Fig. 10: Screenshot taken by the Movement Detection module.

B. Activity Mapping

The Activity Mapping module was tested under similar circumstances as the Activity Mapping module, except that the Histogram Equalization algorithm was turned off for the whole duration of the tests.

Results are harder to interpret, with the maximum acceptable frame acquiral rate being 6, which makes sense as this algorithm is more complex than the one used by the Movement Detection module. The change in frame downscale factor does not seem to produce constant results across the frame acquiral rates.

The Activity Mapping Module was tested on the aforementioned sequences, correctly outputting the heat maps. Two screenshots are visible below:

Fig. 11: Heat map representative of ten seconds of movement in the car park sequence.

Fig. 12: Heat map representative of ten seconds of movement in the roundabout sequence.

C. Activity Mapping with Gradual Updating

The parameters used to test the Activity Mapping with Gradual Updating were the same as the previous module.

Despite this module being slightly more complex than the standard Activity Mapping module, with a lot more of memory copying operations, shown performance is similar, with maximum acceptable rate at 5 frames per second.
VIII. FUTURE WORK

The exploration of more complex algorithms and solutions for implementation by the system is an interesting path, but with caution as with the mentioned algorithms the Raspberry Pi seems to already be reaching its limits. The usage of video compression algorithms to greatly reduce the space taken by the captured sequences is a feature to implement in future versions.

Another possibility is the exploration of human detection procedures like the usage of Histograms of Oriented Gradients for human shape recognition in images, which could allow for even more interesting applications.

IX. CONCLUSIONS

The system can be deployed, used and useful in the real world - it has a low cost while keeping good performance levels and results. The software modules system allows for a very good flexibility in performed functions.

REFERENCES