Efficient face tracking and recognition

Computer Vision For Faces - Final Project Report

David “Dave” Snowdon, 11th August 2018

1. Introduction

I completed the taught portion of the “Computer Vision For Faces” course in October 2017. However, I got side-tracked, and it’s only in summer 2018 that I decided to write-up the current state of the project. It is not as complete as I would have wished but I don’t want to delay further while I resolve issues that were “stretch goals” for the original proposal.

Although I got the code working on the Raspberry Pi 3 I had hoped to get it working for the Raspberry Pi Zero as the low-power and tiny form factor make it a very attractive platform for applications such as a home security system. However the code compiled for the Raspberry Pi 3 did not run on the Pi Zero since the Pi 3 has a newer ARM processor which implements instructions not present on the Pi Zero. I have attempted to cross-compile OpenCV for the Pi Zero but have run into problems[1] that I have not yet managed to solve despite following a process that appears to have worked for earlier versions of Raspbian and OpenCV[2].

Since the aim of the project was to write an efficient face tracking system, I decided that it might be helpful to collect micro-benchmarks for the various operations required of the system as well as overall benchmarks of system performance.

The source code for the project (including the micro-benchmark results) is available on Github at https://github.com/davesnowdon/face-manager Apart from a script to run and process micro-benchmark results the entire project is written in C++11.

The following sections present the original project proposal, the micro-benchmarking results, the structure of the code, measurements of system performance and finally the next steps and conclusions.

2. Original Project Proposal

Systems such as social robots need to be able to interact with people. Part of this involves knowing when the robot is being observed and recognising the people interacting with the robot. A naive implementation would involve multiple stages:

- capturing video frames in real-time
- detecting faces in the video frames

1 https://stackoverflow.com/questions/51744641/link-fails-when-cross-compiling-opencv-3-3-1-for-raspberry-pi-zero-even-though-l

2 Original cross-compilation notes https://github.com/HesselM/rpicross_notes when I get OpenCV 3.3.1 compiled for Raspbian stretch on the Pi Zero I will post updated notes here https://github.com/davesnowdon/rpicross_notes

- recognising the faces if any detected in the previous step

However running the face detection and face recognition on every frame would be highly inefficient and expensive both in terms of compute resources but also network and money if the face recognition is performed by an API hosted in the cloud. We therefore need additional logic to avoid the expense of the detect faces and recognise faces steps unless they are actually needed.

In order to avoid calling the face recognition code for every frame we will track faces as they move in the camera view and so perform the recognition step only on new faces.

The desired flow therefore looks like:

- capture video frame
- detect whether there have been significant changes (possibly with assistance from video hardware)
- if image change greater than threshold run face detection
- match detect faces with locations of faces in previous frame
- determine if any of the faces are new (have just appeared in camera view)
- perform face recognition on new faces (using third-party API or, stretch goal, on the device itself)

The initial implementation will run on a desktop computer, but we will attempt to run at least everything except the face recognition on a Raspberry Pi.

3. System design and implementation

Since efficiency is essential and I wanted the software to work on relatively low-power devices such as the Raspberry Pi 3 and Raspberry Pi Zero I decided to write the software entirely in C++11. There are five main parts to the current implementation, which are not well separated in the Github repository:

1. The face manager itself: is implemented in the Manager class is responsible for accepting each video frame and updating the current state (set of known persons and location of each currently visible person). The face detection, extraction and face descriptor computation is abstracted into a separate class, FaceDetector.

2. Motion detection: Several algorithms were implemented since it was not clear which is the most efficient approach. The base class MotionDetector provides a uniform interface so that other code does not need to be aware which implementation is in use.

3. Various utility functions. One issue I encountered was that it was not always clear where things were going wrong and without seeing intermediate images it was hard to debug. I, therefore, built a simple logging system that allowed me to log both images and text so I could see whether frames were processed in a way that looked reasonable.

4. Benchmarks: These consist of the micro-benchmark suite (micro-benchmark.cpp) and an executable to run the complete system against a pre-recorded video file. All benchmarks used pre-recorded video for repeatability.

5. Executables: These demonstrate how the Manager class can be used.

3.1 Design

The flowchart below shows the high-level operation of the system. Black rectangles indicate code that is specific to the application. Green rectangles are OpenCV functions and blue represents code provided by the face manager library.

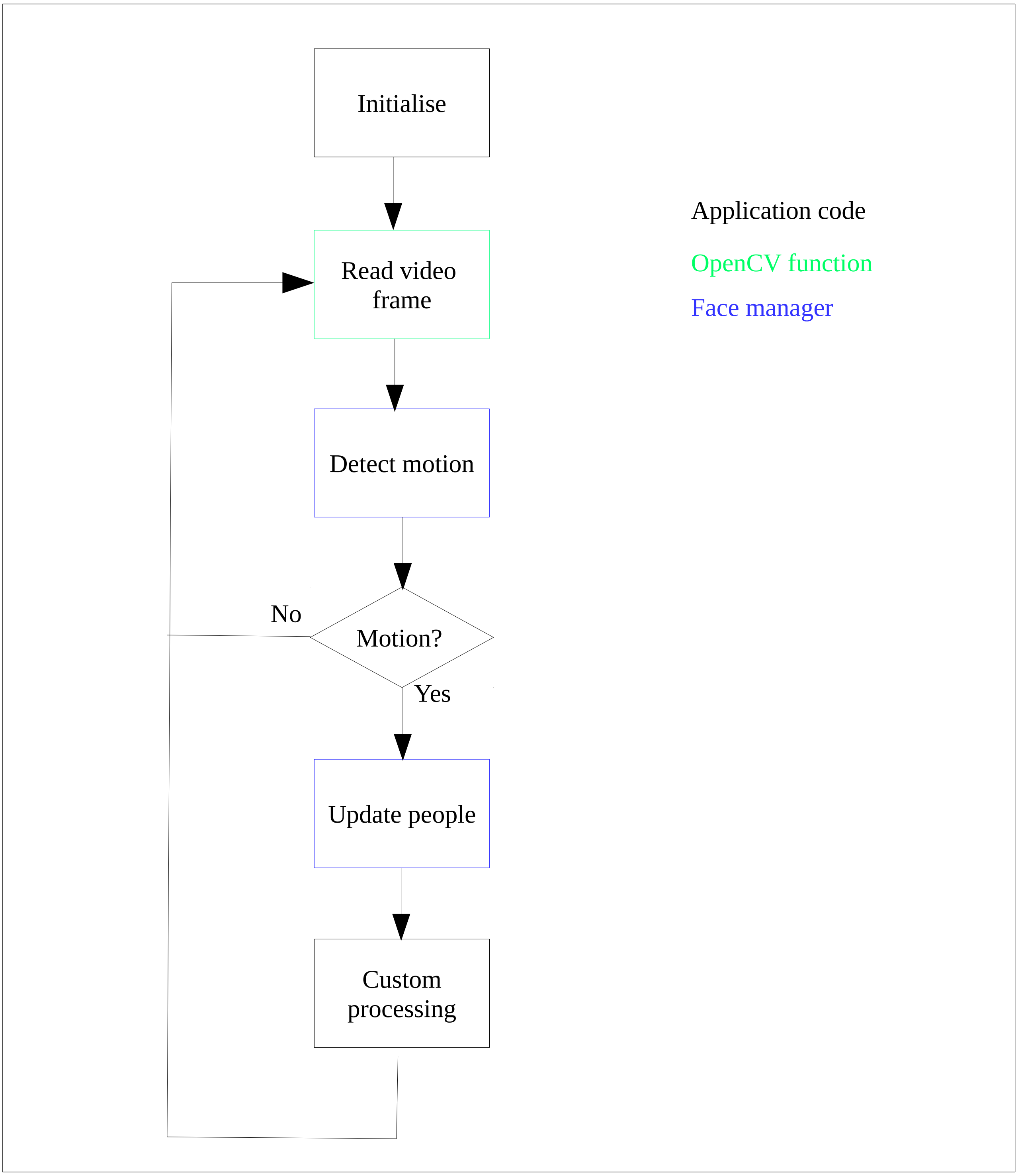


Figure 1: Overall flow

The Manager class tracks known people currently visible or not. Each person is identified in three different ways:

1. local ID: simple incrementing integer assigned automatically when a new face is detected. These are not persisted and mostly used by the Manager class to correlate people with the trackers.
2. External ID: This is optional and provided by the application as way to identify the person to the application. This is represented as a std::string and could be a database ID or the name of the person.
3. Face descriptor: This is computed automatically when a face is a detected and is used to determine whether two faces are from the same person. This is a 128D vector computed by a pre-trained deep neural network.

Additionally the following information is also associated with each person (and stored in instances of the Person class together with the identifiers mentioned above):

- Bounding box – location of the person's face in the most recent frame if the person is currently visible.
- Face image – an image of the person's face as used to generate the face descriptor. This may be updated if the manager determines a better quality image is available (for example if the face was blurred when it was first visible and a future image is sharper).
- Face blur – how blurred the face image is. This is not currently computed but the idea is in future to attempt to determine the image quality when a face is detected and automatically replace the stored face image if a better quality image becomes available in a later frame.

The Manager class uses the following data structures to store the current state of the world:

- A map of local ID (int) to person object (managed by a `shared_ptr`). This contains all people known to the system whether they are currently visible or not. Each time the Manager detects a new face or is told about a new person by the application via its API a person object is created and stored in here.
- A map of local ID (int) to `dlib::correlation_tracker` (managed by a `unique_ptr`) this stores the trackers used to track the movement of faces.

The Manager class has the following configurable parameters (not all of these are currently exposed via its API):

- descriptor threshold: maximum difference between two face descriptors that can still be considered to be the same person (default 0.6)
- bounding box threshold: Minimum Intersection over Union (IoU) value to treat two bounding boxes as the same (default 0.5)
- min tracker confidence: The lowest confidence value we'll accept on a tracker update before considering that the tracker has lost the tracked object (default 7)

- Margin around detected face to use when instantiating a tracker to track the face (default 10 horizontal pixels, 20 vertical pixels).
- Number of frames between each run of the face detector (current default is 5 but this will probably need to be increased on the Raspberry Pi).
- Use jitter: Whether to create face descriptors by creating creating multiple versions of the face image by moving zooming, translating and rotating the source image, computing a face descriptor for each copy and then averaging them. David King reports that this may increase face detection accuracy by a small amount but is very expensive to since it computes 100 face descriptors and so this value is false by default.

3.2 Implementation

The bulk of the work done by the Manager occurs in the newFrame() method and is shown in Figure 2.

When passed a new frame the Manager updates all the existing trackers. It then removes any that report a low confidence score.

If enough frames have passed since face detection was last run the Manager runs the face detector on the frame. If any faces have been detected the Manager uses the bounding boxes generated by the trackers and face detector to determine which detected faces match which trackers.

Any detected faces that don’t sufficiently overlap with an existing tracker are considered to be new faces. The face landmarks are computed and the face image is extracted and used to compute a face descriptor. The face descriptor is then used to determine whether this person is already known to the system. If the person is not known then a new Person object is created and added to the map of known people. In either case a new tracker is created and associated with the local ID of the person.

Finally any trackers that did not match a detected face are removed.

3.3 Executables

The library contains the following executables:

- Manager demo – takes a video file and some people definitions (name and image) and produces an annotated video tracking faces as they appear.
- Manager benchmark – computes frame rate measurements for a specified video and different motion detection algorithms.
- Micro benchmarks – runs the micro-benchmarking suite. The run-benchmarks.sh script is used to run this and extract the results averaged over five runs as a CSV file.
- Security camera (PLANNED) – similar to manager demo but creates video files for each time when at least one person is visible and ignores all other frames. The idea is that this could be used as the basis of a security monitoring system with alerts being produced each time a new video file is created.

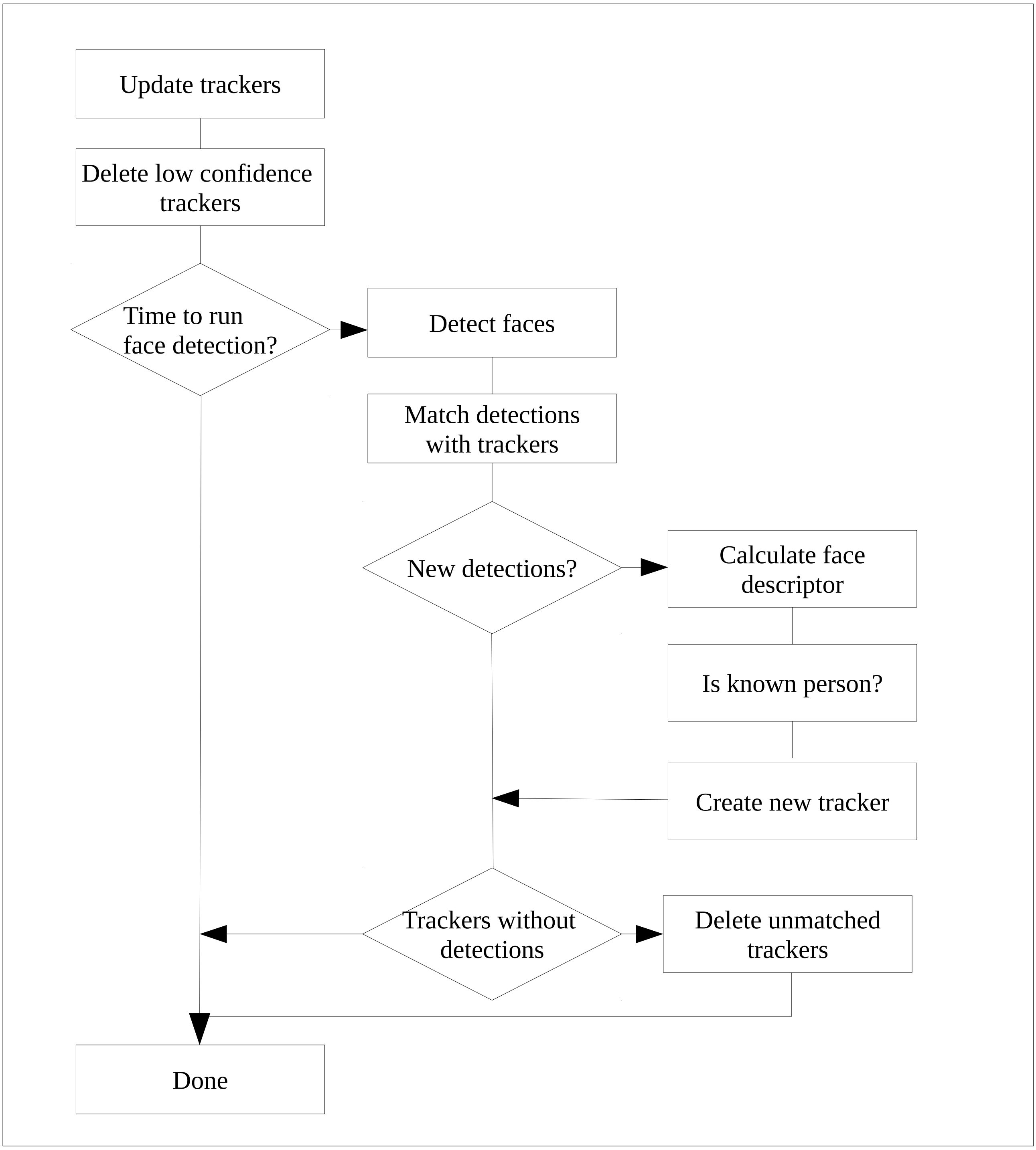


Figure 2: The work done by the Manager on each frame passed to it

4. Micro Benchmarks

As I implemented the face manager code I identified the operations I was using from OpenCV and dlib and made a list of them. I wrote a simple program to run the operations repeatedly and record the average (mean) time taken for the operation. To determine how much the performance depended on the size of the input each operation was run on four different image sizes for each platform: 500x375, 1296x972, 2016x1512 and 4032x3024. The sole exception was the face recognition operation which operates on a fixed size 150x150 face image. In the desktop case, each operation was run 10,000 times since the Raspberry Pi3 was significantly slower than the desktop I used 100 iterations. Since the benchmark harness required calling the routine as a function, I also benchmarked the time required to invoke an empty function and used that to adjust the overall time for the other operations. As much as possible any setup required to was done once at initialisation time to reduce overhead. Finally, I wrote a bash script to run the benchmark suite five times and then format the results as a CSV file I could then import into a spreadsheet for further processing.

The following section shows the results for the different categories of operation. The figures in black indicate the average time for the operation and the figures in red show the number of operations performed per second.

The results should be taken with a large grain of salt and there are several things that should be improved before they are taken too seriously:

- Running at the large resolutions caused swapping on the Pi3 so the results are unlikely to be pure CPU
- The benchmarks were not all run at the same time so there may be some slight variation in performance.
- The tracker benchmarks were run repeatedly on the same patch - it would be more realistic to run them against different patches and over several frames.

Once again, please don't take the exact values too seriously. They are just a guide to the cost of the underlying operations used by the face manager.

4.1 Basic OpenCV operations

	Desktop				**RaspberryPi3**			
	X86_64				**armv7l**			
Operation	**500x375**	**1296x972**	**2016x1512**	**4032x3024**	**500x375**	**1296x972**	**2016x1512**	**4032x3024**
Basic operations								
Accumulate	1.483515E-05	0.0002302816	0.0007181482	0.0040939166	0.0015603909	0.0106201709	0.0257108107	0.1029791907
	67407.478813	4342.508796	1392.4701862	244.26486668	640.86506514	94.160443531	38.894145041	9.7106997377
Bitwise and	4.158377E-06	8.808165E-05	0.0002117032	0.0011732506	0.0001656875	0.0011746989	0.0028587227	0.0115501107
	240478.42972	11353.102604	4723.5934874	852.33280409	6035.4592137	851.28199315	349.80657772	86.579256861
Blur image	0.0027752146	0.0188356386	0.0453431786	0.1881965986	0.1014433909	0.5468717909	0.1304977907	12.420139991
	360.33249087	53.090846543	22.05403393	5.313592314	9.8577146463	1.8285821589	7.66296498	0.0805143904
Dilate	8.151933E-05	0.0004023862	0.0009186174	0.0045300846	0.0081188289	0.0540822909	0.1018567907	0.6189929907
	12267.029314	2485.1745105	1088.592451	220.74642783	123.17047389	18.490340994	9.8177057538	1.6155271789
Erode	4.052915E-05	0.0002169926	0.0005032284	0.0026569006	0.0037346529	0.0248029709	0.0597781307	0.2838349907
	24673.599661	4608.451467	1987.1691259	376.37839685	267.76250253	40.317750846	16.728525775	3.5231737904
Find contours	0.0002460424	0.0012471406	0.0029341866	0.0095003746	0.0030030509	0.0204498909	0.0458415507	0.1647037907
	4064.339649	801.83419307	340.80995042	105.25900703	332.99469156	48.90001644	21.814270787	6.0715056759
Frame difference	2.005607E-05	0.0002629376	0.000797119	0.0044529626	0.0023589329	0.0158536509	0.0382899307	0.1827213907
	49860.21907	3803.1833	1254.5177846	224.56959174	423.9204983	63.076953558	26.116526775	5.4728129872
Norm2	9.174919E-06	0.000082662	0.000211768	0.0013286786	0.0006785913	0.0049776349	0.0120139907	0.0473037107
	108992.78641	12097.458707	4722.1480905	752.62744268	1473.6411095	200.89862455	83.236288924	21.139990615
Sum	1.919281E-06	2.476923E-05	0.0001053734	0.0004576396	0.0003173099	0.0021724109	0.0052619187	0.0210067307
	521028.41506	40372.673496	9490.0583284	2185.1254411	3151.4934691	460.31807856	190.04474591	47.603790179
Threshold	2.540725E-05	3.527415E-05	0.000049274	0.0006007734	0.0003083645	0.0017215829	0.0034342607	0.0157652307
	39358.845788	28349.372698	20294.682722	1664.5210138	3242.9157046	580.86079636	291.18348634	63.430724219

4.2 Convert to greyscale and resize

The greyscale & resize benchmark was an attempt to see whether the order of operation was important when resizing the images for the motion detection code. As is shown below it is generally faster to convert an image to greyscale first and then resize.

	Desktop X86_64				RaspberryPi3 armv7l			
Operation	**500x375**	**1296x972**	**2016x1512**	**4032x3024**	**500x375**	**1296x972**	**2016x1512**	**4032x3024**
Resize								
Resize image		0.0021630766	0.0039327046	0.0121784186		0.0316321509	0.0620068107	0.1983659907
		462.3044725	254.27793186	82.11246717		31.613405106	16.127260682	5.0411867303
Grayscale then resize		0.0017442246	0.0045428926	0.0105414786		0.0248625509	0.0529247107	0.1861065907
		573.32065108	220.12406662	94.863352195		40.221134388	18.894765545	5.3732648387
Resize then grayscale		0.0022651486	0.0040529166	0.0125967786		0.0351606509	0.0705794507	0.2090099907
		441.47213439	246.735892	79.385375366		28.440884202	14.168429908	4.7844602869

4.3 Image conversions

Since the code uses both OpenCV and dlib, there are times when we need to convert between OpenCV and dlib images. The mean squared error motion detection code also converts images from integer to floating point format and so that operation is also benchmarked here.

	Desktop X86_64				RaspberryPi3 armv7l			
Operation	**500x375**	**1296x972**	**2016x1512**	**4032x3024**	**500x375**	**1296x972**	**2016x1512**	**4032x3024**
Conversion								
Convert CV image to dlib	4.821288E-08	4.863304E-08	4.866166E-08	4.867874E-08	4.924948E-07	5.038178E-07	4.98224E-07	5.536056E-07
	20741345.466	20562152.808	20550059.328	20542848.891	2030478.2913	1984844.5212	2007129.3234	1806340.1093
Convert CV image to float	1.456639E-05	0.000227629	0.0005908448	0.0024519666	0.0007146827	0.0048583129	0.0117962307	0.0475528507
	68651.193632	4393.1127935	1692.4917486	407.83589281	1399.2223884	205.83277069	84.772841977	21.029233485

4.4 Tracking

To track faces without performing face detection and face recognition on every frame, we use a tracker to follow faces after they have been detected. The dlib correction tracker outperforms the OpenCV KCF and MEDIANFLOW trackers by a significant amount. The KCF tracker performs particularly badly at 4032x3024 on the Raspberry Pi 3 taking 1294 seconds for a single update. I suspect this is due to swapping since the the benchmark made use of a considerable amount of swap space while running on the Pi3. The MEDIANFLOW tracker however was not affected to anything like the same degree suggesting hat the KCF tracker has particularly high memory requirements or scales very poorly with image size.

	Desktop				**RaspberryPi3**			
	X86_64				**armv7l**			
Operation	**500x375**	**1296x972**	**2016x1512**	**4032x3024**	**500x375**	**1296x972**	**2016x1512**	**4032x3024**
Trackers								
dlib correlation tracker update	0.0071559886	0.0093518126	0.0123145186	0.0210228786	0.1011279909	0.1177681909	0.1370711907	0.2926627907
	139.74309517	106.93114155	81.204960585	47.567225096	9.8884590838	8.491257211	7.2954790499	3.4169017443
OpenCV KCF tracker update	0.0158860986	0.1221185986	0.2930873986	1.1080659986	0.2229077909	3.1257899909	9.5069719907	1294.94
	62.948117303	8.188760854	3.4119515362	0.9024733195	4.4861599322	0.3199191254	0.1051859626	0.0007722366
OpenCV MEDIANFLOW tracker update	0.0105750986	0.0104390786	0.0178010786	0.0889388586	0.1307757909	0.1328615909	0.3044901907	2.9318399907
	94.561765811	95.793894799	56.17637115	11.243679258	7.6466752241	7.5266297311	3.2841780477	0.3410827341

4.5 Face detection

To achieve the best frame rate possible, we don't perform face detection on every frame, but the cost can't be so prohibitive that we cannot afford to run it at all. The dlib face detection worked reasonably well on the desktop but performed poorly on the Raspberry Pi, so I compared it with the OpenCV HAAR based face detection.

The OpenCV HAAR based face detection certainly performs better on the Raspberry Pi, however after talking to Davis King it seems I will not be able to take advantage of that without re-training the landmark detector. This is because the landmark detector is trained using the bounding boxes generated by a specific face detector and therefore won’t work as well with a different face detector as the offsets will likely be different. Also, according to

Davis King, the OpenCV face detector generates more false positives. Using the OpenCV face detector, and assembling a dataset and re-training the dlib face landmark detector does not seem like a practical approach.

Instead, given that the cost of face detection increases with image size, a practical approach (also suggested by Davis King) would be to perform face detection on a smaller image and then scale the bounding box in order to perform face landmark detection and extract the face image for face recognition.

Operation	Desktop X86_64 500x375	1296x972	2016x1512	4032x3024	RaspberryPi3 armv7l 500x375	1296x972	2016x1512	4032x3024
Face detection								
dlib	0.0285216986	0.1812475986	0.4269881986	1.6902359986	1.7128259909	12.014679991	29.428219991	119.13259999
	35.061025397	5.5173144779	2.3419851022	0.5916333582	0.5838304681	0.0832315135	0.0339809883	0.008394008
OpenCV HAAR	0.0109435586	0.0624733186	0.1217815986	0.3696911986	0.1708283909	0.8649631909	3.1533039907	5.1528799907
	91.377954273	16.006833348	8.2114211937	2.7049602579	5.8538278964	1.1561185615	0.3171276867	0.1940662313

4.6 Face landmarks

When a face is detected, we need to find the face landmarks so that we can accurately scale and crop out the region of the image containing the face. Since the landmark detection is only required to extract the face I used the new(ish) dlib 5-point landmark detection rather than the 68-point landmark detection since that keeps the memory requirements as low as possible. Once the face landmarks have been located, we use them to extract a 150x150 face "chip" that we can use for face recognition.

Operation	Desktop X86_64 500x375	1296x972	2016x1512	4032x3024	RaspberryPi3 armv7l 500x375	1296x972	2016x1512	4032x3024
Face landmarks								
Face landmarks	0.000795521	0.000844107	0.0008748834	0.0009140644	0.0105200109	0.0112659309	0.0113859707	0.0118101507
	1257.0377946	1184.6838914	1143.0094174	1094.0147832	95.056935952	88.7631933	87.827382262	84.672924711
Extract face chip	0.00049825	0.0020840526	0.0041887346	0.0164192586	0.0038506209	0.0154132909	0.0311193307	0.1316607907
	2007.0244688	479.83433145	238.73558204	60.904089672	259.69837913	64.879071451	32.134367221	7.5952756685

4.7 Get face descriptor (face recognition)

We perform face recognition by computing 128-dimensional descriptors for each face image using a pre-trained deep neural network provided by the dlib models repository. By comparing the distance between two face descriptors, we can determine whether they are likely to represent the same person. This operation is always performed on 150x150 pixel face images this is the only benchmark which is not run at multiple image sizes.

	Desktop x86_64 (with CUDA)	**Raspberry Pi 3**
Time per operation	0.0022692426	0.5631711909
Operations / second	441	1.8

5. System benchmarks and results

There are two aspects of the complete system to evaluate:

1. The different algorithms for motion detection and their performance
2. The performance of the face manager code compared to a naive implementation which always performs the face detection and recognition steps.

An example of a naive implementation is the facerec_from_webcam_faster.py example from Adam Geitgey's face recognition library[3] which reduces the overhead of the face detection and recognition code by running it every other frame. The main loop for this code is shown in Figure 3.

The following sections describe the test data used for evaluation and the results of running the system benchmarks.

5.1 Video test data

This section describes the video files used to evaluate the system and the method by which they were captured. Most files were captured using a Raspberry Pi Zero with an attached Raspberry Pi camera in order to capture video closely matching what the system would be processing if running on one of the intended platforms. The videos were captured at a resolution of 1296x972 and converted to MP4 format using the following commands:

```
raspivid -w 1296 -h 972 -o <output filename>.h264 -t 10000
```

```
MP4Box -add nomotion1.h264 nomotion1.mp4
```

The multi-face tracking video was captured using an Olympus Air A01 lens camera at a resolution of 1920x1080

Since we wanted to determine the affect of the motion detection code whether or not faces were detected there are two sets of videos one with visible faces and one without.

3 https://github.com/ageitgey/face_recognition

```
while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Resize frame of video to 1/4 size for faster face recognition
    # processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (which OpenCV uses) to RGB
    # color (which face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current
        # frame of video
        face_locations =
            face_recognition.face_locations(rgb_small_frame)
        face_encodings =
            face_recognition.face_encodings(rgb_small_frame,
                                            face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches =
                face_recognition.compare_faces(known_face_encodings,
                                               face_encoding)
            name = "Unknown"

            # If a match was found in known_face_encodings,
            # just use the first one.
            if True in matches:
                first_match_index = matches.index(True)
                name = known_face_names[first_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame
```

Figure 3: Main loop from Adam Geitgey's facerec_from_webcam_faster.py[4] example

4 https://github.com/ageitgey/face_recognition/blob/master/examples/facerec_from_webcam_faster.py

Video file	Frame size	Description
emptyroom1	1296x972	Empty office room
emptyroom2	1296x972	View of stairs with no movement
lightingchange1	1296x972	Empty office room with no movement, but lighting changes
lightingchange2	1296x972	View of stairs with no movement, but lighting changes
movinghuman4	1296x972	Person walks upstairs with back to camera
street1	1296x972	Street view. Person walks past and off screen
street2	1296x972	Street view. Care drives slowly past
street5	1296x972	Street view. Birds fly past in distance

Table 1: Video files without visible faces

Video file	Frame size	Description
movinghuman1	1296x972	Person walks into frame and sits down
movinghuman2	1296x972	Person walks slowly downstairs towards camera
multi-face	1920x1080	4 different people walk past the camera with multiple people in view at several points
street3	1296x972	Person walks past and looks at camera

Table 2: Video files with visible faces

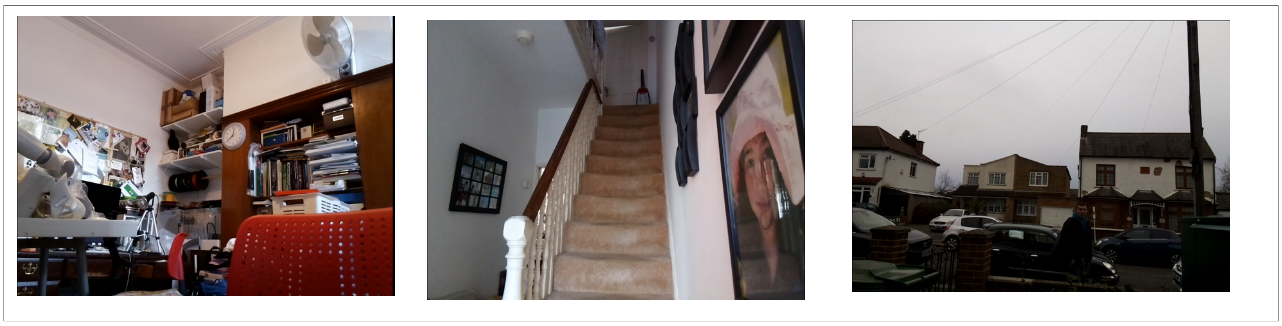
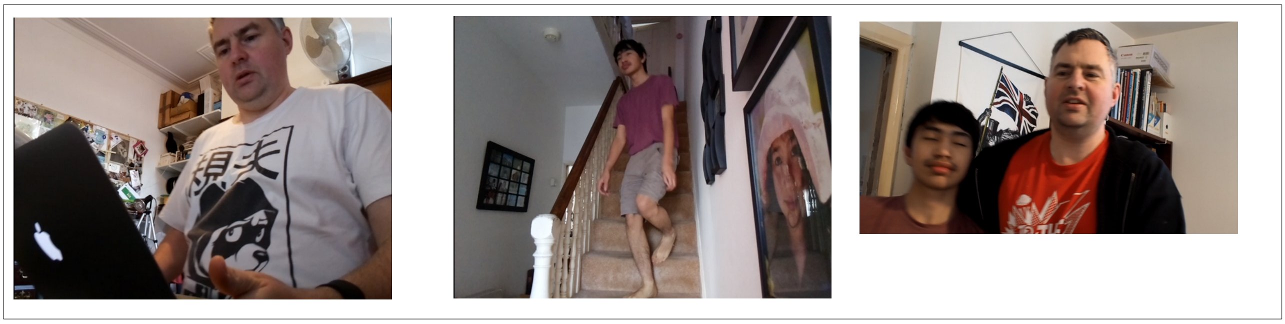


Figure 4: Locations used for videos office (room1), stairs (room2) and street.

Figure 5: Examples with faces: movinghuman1, movinghuman2 & multi-face



5.2 Evaluation methodology

The aim of the face manager is to track people in front of a camera whilst doing the minimum amount of computation. The reasoning behind this is that tracking people is part of a larger application and we therefore want to leave as much computation as possible for the main part of the application to run. Since each video has different faces appearing at different sizes and durations it is necessary to measure performance with multiple videos both with and without faces to track. We calculate the work done by the face manager in two ways:

- The mean frame-rate when processing a video – the more efficient the face manager is at processing a video the higher the frame-rate should be. The never detect motion detector gives us the absolute maximum possible frame-rate: zero work done by the face manager. The other motion detector scenarios allow us to see which motion detectors give best performance.
- Counts of the number of face detection and face recognition operations. The higher the count of operations the more work the face detector is doing.

We make the assumption that we detect and track all faces that can be detected. The objective of this exercise is *not* to evaluate the performance of OpenCV and dlib, just use them in the most efficient way possible.

The final performance metric to consider is how many motion frames do the motion detection algorithms detect. Ideally in the videos with nothing happening, such as the empty room and lighting change scenarios no motion should be detected whereas motion should be detected in the videos which contain moving people.

The manager-benchmark executable takes a video file as input and processes it multiple times to test the following combinations of a motion detector and face manager:

- Naive (no manager): the face detector is run on all frames with motion detected and any faces found are extracted and have the face descriptor computed. The code does not lookup the faces against known faces.
- Manager, detect 5 - the manager is used and run faces detection every 5 frames with motion and uses the dlib correlation tracker to track faces between invocations of the face detector
- Manager, detect 10 - uses the manager but only runs face detection every 10 frames with motion

For each of the above scenarios, we test with each of the following motion detectors:

- Always - singles every frame as containing motion. Used so we can compare manager performance with naive implementations
- Never - never indicates a frame contains motion. Used so we can determine the overhead of just reading the frames from file without any additional work. We only report the results for the naive implementation since no work is done by the manager anyway.
- Every other - reports motion every other frame. Used so we can compare manager with the example from Adam Geitgey's library.
- Every 10 - reports motion every 10 frames. Used to see whether a dumb approach compares with the computation required to detect motion.

- Contours - uses a C++ implementation of the algorithm presented on the PyImageSearch blog: https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/
- MSE - use mean squared error. C++ Implementation of algorithm described on PyImageSearch blog: https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/
- MSE with blur - blur image first before computing the mean squared error
- Frame difference - algorithm derived from this blog post http://www.steinm.com/blog/motion-detection-webcam-python-opencv-differential-images/
- Frame difference with blur - like frame difference but the image is blurred before the images are compared

Each file is therefore processed 25 times to evaluate all the above combinations of motion detector and manager (not 27 because we ignore the "Never" motion detector for the face manager trials).

The run-manager-benchmark.sh script automates running manager-benchmark for all the videos and collates the results into a CSV file.

5.3 Results

We separate the results according to motion detection performance and overall system performance. For the motion detection algorithms we are interested in the performance (how much of a speed penalty they impose) and how sensitive they are to motion. In the case of the complete system we are interested in how much it improves over the naive approach.

5.3.1 Motion Detection

The two relevant metrics for motion detection are how fast does the algorithm run and how sensitive it is (how much motion does it detect). For each of the thirteen test videos I measured the average frame rate to just read the file data (the “never” motion detector) and subtracted that from the times from the runs with one of the five motion detection algorithms under test. In order to get a single number summary of how sensitive the motion detector is I computed the average number of motion frames for each algorithm – this number is probably not particularly meaningful but it does make it possible to compare the algorithms. The full set of performance metrics for each of the algorithms can be found in the file manager-benchark-results.ods in the “motion detection” tab. Table 3, below gives the average per frame times for each algorithm and the computed “sensitivity”

Method	**Time**	**Sensitivity**
CONTOURS	0.0137	82
MSE	**0.0107**	**88**
MSE_WITH_BLUR	0.0121	72
DIFF	0.0107	73
DIFF_WITH_BLUR	0.0132	61
Min	**0.0107**	**61**
Max	**0.0137**	**88**

Table 3: Motion detection performance summary

From Table 3 we can see that the MSE (mean squared error) algorithm is both the fastest and the most sensitive. However, just reporting the most motion frames does not indicate that the algorithm is the most effective. We don’t want to get spurious detections when there is no motion. We don’t have a ground truth motion value for each video as I felt that manually attempting to compute the number of motion frames would be difficult for a human and error prone. However, there are videos that contain no motion (for example the emptyroom and lightingchange videos) and we can use those to determine if an algorithm is likely to report spurious motion. MSE and MSE-with-blur were the only algorithms to report motion for lightingchange1 (8 motion frames). MSE was the only algorithm to report motion for street1 (2 frames). The versions of the algorithms with blur were both slower and less sensitive than the versions without blur, which is to be expected.

The next most sensitive algorithm, DIFF, was only very slightly less performant than MSE (by 0.000039287 seconds/frame) and so might be a good choice also if more resistance to false positives is desired.

5.3.2 Face Manager

When comparing the face manager with the naive approach we are only interested in the cases where there was actual work to do. Frames with no detected motion are never considered for further processing and so do not exercise the face manager part of the system. For this reason when comparing the benchmark results here we consider only the naive, manager 5 and manager 10 processing types and strip out all combinations where no motion was detected at all when processing the video file. We therefore have 66 datapoints for each of the naive, manager 5 and manager 10 scenarios. Given that none of the videos have particularly fast moving faces we make the simplifying assumption that all cases tracked all faces that could be tracked using OpenCV and dlib.

For each scenario we considered two pieces of information:

- The mean framerate when processing the video (without any use of threads)
- The amount of work done. We calculated this by counting the number of face detect, face extract and face descriptor operations and multiplying them by the time required (as calculated by the micro benchmarks). Since every face descriptor operation was proceeded by a face extract we only show the face descriptor and face detect counts in the results summary (manager-benchmark-results.ods)

In order to have a simple point of comparison we then took the mean values for frames per second and work done and calculated the mean across all 66 scenarios. We then used the naive values as a baseline and compared with the manager scenarios.

Method	Mean FPS	Speed increase	Mean Work	Work fraction
Naive	24		32.73	
Manager 5	46	2.97	7.41	0.36
Manager 10	55	4.02	4.85	0.29

Table 4: Face manager performance compared with naive approach

We can see from table 4 that the face manager offers a significant speed advantage compared to the naive approach. The manager 4 scenario gives, on average, almost 3 times faster frame rate which rises to 4 times faster for the manager 10 scenario. This correlates with manager 5 doing slightly more than 1/3 the work of the naive approach and manager 10 doing slightly more than ¼ of the work.

Looking at the results in detail however, shows that the manager was not always faster than the naive approach. Every scenario using the "every 10" motion detector[5] indicated that the naive approach did almost exactly the same amount of work as the face manager scenarios and led to a the naive approach giving a slightly faster frame rate (by a factor of between 0.01 to 0.03). The reason for this surprising lack of performance turns out to be a bug in the face manager code (tracked by GitHub issue #10[6]). In the Manager::newFrame() method the code determines when to perform face detection instead of just tracking using the absolute frame number. This means that in the case where several frames have elapsed without significant motion the manager could end up performing face detection on every invocation of newFrame() since not much can have changed since the last invocation (otherwise the motion detector would have fired earlier) we are wasting computation and potentially performing slightly worse than a naive algorithm which always performs face detection. We need to base the decision on whether to do face detection on the number of frames newFrame() has processed not the absolute number of frames. We may still want to force a detection periodically if a very large number of frames have elapsed. This shows that once the bug is fixed we should be able to get even better performance from the face manager.

6. Follow up work

The following work was not complete at the time of writing but is planned:

- Fix the bug in which the face manager uses the absolute frame number to determine whether to run face detection (GitHub issue #10).
- Compile OpenCV 3.3.1 (or later), dlib and this project for the Raspberry Pi Zero and tune to find settings that give acceptable performance.
- Update the FaceDetector class to use OpenCV face tracking when compiled for the Raspberry Pi.
- Implement the security camera executable and tune for the Raspberry Pi Zero. This will use multiple threads to enable maximum performance by allowing computation to take place in parallel with reading new frames from the camera and writing frames to disk (when required).
- Use as the basis of interactive applications with robots (the original motivation for this project) and update the Manager API as required to enable this.
- Implement the ability to run face detection on lower resolution images whilst extracting face landmarks and face images from the original frame in order to get the best balance of frame-rate with high resolution face images (for more reliable face recognition). This will also entail further tuning to find the lowest resolution images in which faces can be detected at a reasonable distance[7].
- It's possible that the Raspberry Pi Zero might not be capable of running the complete face manager "stack" in this case we might want to implement motion detection on the Raspberry Pi and send motion frames to another device for face detection and recognition. This

5 This "detector" always signalled motion every 10 frames.

6 https://github.com/davesnowdon/face-manager/issues/10

7 The dlib face detector does not work well with small face images so running face detection on smaller images trades of frame-rate for reliability of face detection.

approach is used by Jeff Bass's "yin-yang ranch"[8] which uses ZeroMQ[9] as the message transport via a python wrapper[10].

7. Conclusion

This has been an interesting project and allowed me to refresh my C++ knowledge and put into practice several of the concepts I learned from the "Computer Vision for Faces" course. I did not originally expect to be able to do face recognition on a platform such as the Raspberry Pi 3 without using a cloud-hosted API.

A major source of frustration has been trying to compile OpenCV 3.3.1 for the Raspberry Pi Zero. If it were not for that I would have completed this report considerably earlier as I was originally hoping to include benchmarks for the Pi Zero with this report. Once I manage to get the code working on the Raspberry Pi I will publish the complete set of benchmark results for all three platforms (x86 + CUDA, Raspberry Pi 3 and Raspberry Pi Zero) in the Github repo at: https://github.com/davesnowdon/face-manager

Finally, I would like to thank Satya Mallick and the team at LearnOpenCV for producing the course and their support while I followed it. I've also appreciated being a part of the student community for the course on https://forum.learnopencv.com/ I would also like to thank Davis King, the author of dlib (http://dlib.net/) for making the library available and his many informative blog posts. Adrian Rosebrock of PiImageSearch (https://www.pyimagesearch.com) also deserves thanks for his help, many highly informative blog posts and for the PyImageSearch gurus course.

Appendix 1 – Detailed motion detector results

The following tables are taken from the motion detector tab in manager-benchmarks-results.ods all values are a result of running on a desktop x86_64 machine with an Nvidia GTX 1080 graphics card. Here the intent is to compare different algorithms so only a single platform was used to compute the benchmarks.

Baseline read performance

Filename	# frames	FPS	Seconds / frame
emptyroom1.mp4	284	519.091	0.0019264445
emptyroom2.mp4	284	520.254	0.001922138
lightingchange1.mp4	284	518.064	0.0019302634
lightingchange2.mp4	284	519.065	0.001926541
movinghuman1.mp4	285	525.972	0.0019012419
movinghuman2.mp4	285	526.65	0.0018987943
movinghuman4.mp4	285	531.885	0.0018801057
stillhuman1.mp4	284	502.96	0.0019882297
street1.mp4	288	473.018	0.0021140845
street2.mp4	288	510.291	0.0019596662
street3.mp4	288	481.156	0.002078328
street5.mp4	288	488.032	0.002049046
multi-face.mov	910	318.242	0.0031422628

8 https://github.com/jeffbass/yin-yang-ranch
9 http://zeromq.org/
10 https://github.com/jeffbass/imagezmq

Table a1: Baseline video file read performance

CONTOURS	4			
Filename	**FPS**	**Seconds / frame**	**Adjusted**	**# motion frames**
emptyroom1.mp4	66.6825	0.0149964383	0.0130699938	0
emptyroom2.mp4	66.2573	0.0150926766	0.0131705385	0
lightingchange1.mp4	66.5152	0.0150341576	0.0131038942	0
lightingchange2.mp4	66.1832	0.0151095746	0.0131830336	0
movinghuman1.mp4	65.6214	0.0152389312	0.0133376893	222
movinghuman2.mp4	66.2504	0.0150942485	0.0131954542	45
movinghuman4.mp4	66.1273	0.0151223474	0.0132422417	150
stillhuman1.mp4	66.6454	0.0150047865	0.0130165568	0
street1.mp4	65.9374	0.0151658998	0.0130518153	0
street2.mp4	66.0963	0.0151294399	0.0131697738	0
street3.mp4	65.9746	0.0151573484	0.0130790204	56
street5.mp4	65.8772	0.0151797587	0.0131307127	0
multi-face.mov	43.6015	0.0229349908	0.019792728	589
Mean			**0.0136571887**	**82**

Table a2: Contours motion detection algorithm

MSE	5			
Filename	**FPS**	**Seconds / frame**	**Adjusted**	**# motion frames**
emptyroom1.mp4	82.876	0.0120662194	0.0101397749	0
emptyroom2.mp4	82.7591	0.0120832634	0.0101611253	0
lightingchange1.mp4	82.8733	0.0120666125	0.0101363491	8
lightingchange2.mp4	82.7209	0.0120888433	0.0101623023	0
movinghuman1.mp4	82.9893	0.0120497462	0.0101485043	235
movinghuman2.mp4	82.3969	0.0121363789	0.0102375847	72
movinghuman4.mp4	82.6137	0.0121045299	0.0102244242	184
stillhuman1.mp4	82.6513	0.0120990232	0.0101107936	0
street1.mp4	81.854	0.0122168739	0.0101027895	2
street2.mp4	82.2763	0.0121541683	0.0101945022	0
street3.mp4	81.809	0.012223594	0.010145266	59
street5.mp4	82.0203	0.0121921037	0.0101430577	0
multi-face.mov	49.6385	0.0201456531	0.0170033903	589
Mean			**0.0106853742**	**88**

Table a3: Mean Squared Error motion detection algorithm

MSE_WITH_BLUR	6			
Filename	**FPS**	**Seconds / frame**	**Adjusted**	**# motion frames**
emptyroom1.mp4	75.1415	0.0133082251	0.0113817807	0
emptyroom2.mp4	74.2908	0.0134606169	0.0115384789	0
lightingchange1.mp4	74.4475	0.0134322845	0.0115020211	8
lightingchange2.mp4	74.0774	0.0134993939	0.0115728529	0
movinghuman1.mp4	75.2755	0.0132845348	0.0113832929	202
movinghuman2.mp4	73.9004	0.0135317265	0.0116329322	35
movinghuman4.mp4	73.6774	0.0135726831	0.0116925774	96
stillhuman1.mp4	74.1724	0.0134821039	0.0114938742	0
street1.mp4	73.558	0.0135947144	0.0114806299	0
street2.mp4	73.4495	0.0136147966	0.0116551304	0
street3.mp4	73.6166	0.0135838928	0.0115055647	45
street5.mp4	73.3761	0.0136284158	0.0115793698	0
multi-face.mov	46.3229	0.0215875949	0.0184453321	550
Mean			**0.012066449**	**72**

Table a4: Mean Squared Error with blur motion detection algorithm

DIFF	7			
Filename	**FPS**	**Seconds / frame**	**Adjusted**	**# motion frames**
emptyroom1.mp4	82.0201	0.0121921334	0.0102656889	0
emptyroom2.mp4	82.9243	0.0120591913	0.0101370533	0
lightingchange1.mp4	82.3637	0.012141271	0.0102110075	0
lightingchange2.mp4	82.3559	0.0121424209	0.0102158799	0
movinghuman1.mp4	82.3775	0.012139237	0.0102379952	199
movinghuman2.mp4	82.553	0.0121134302	0.0102146359	49
movinghuman4.mp4	82.2458	0.0121586756	0.0102785699	96
stillhuman1.mp4	82.6922	0.012093039	0.0101048093	0
street1.mp4	81.938	0.0122043496	0.0100902652	0
street2.mp4	81.7191	0.0122370413	0.0102773751	0
street3.mp4	81.9127	0.0122081191	0.0101297911	58
street5.mp4	81.9261	0.0122061223	0.0101570764	0
multi-face.mov	49.4005	0.0202427101	0.0171004473	543
Mean			**0.0107246612**	**73**

Table a5: Pixel differencing motion detection algorithm

DIFF_WITH_BLUR	8			
Filename	**FPS**	**Seconds / frame**	**Adjusted**	**# motion frames**
emptyroom1.mp4	68.3918	0.0146216359	0.0126951914	0
emptyroom2.mp4	68.3468	0.0146312629	0.0127091249	0
lightingchange1.mp4	67.8168	0.0147456088	0.0128153453	0
lightingchange2.mp4	68.2236	0.0146576844	0.0127311434	0
movinghuman1.mp4	67.3515	0.0148474793	0.0129462374	151
movinghuman2.mp4	68.8898	0.014515937	0.0126171428	20
movinghuman4.mp4	67.7663	0.0147565973	0.0128764916	75
stillhuman1.mp4	67.7734	0.0147550514	0.0127668217	0
street1.mp4	67.5614	0.0148013511	0.0126872666	0
street2.mp4	68.6902	0.0145581175	0.0125984513	0
street3.mp4	67.5651	0.0148005405	0.0127222125	55
street5.mp4	67.9858	0.0147089539	0.012659908	0
multi-face.mov	44.5088	0.0224674671	0.0193252043	490
Mean			**0.0132423493**	**61**

Table a6: Pixel differencing with blur motion detection algorithm

Appendix 2 – Detailed face manager results

This section presents the detailed values used to compute table 4 in section 5.3.2 and can also be found in the “face manager” tab of manager-benchark-results.ods

Note the mean values shown under each table relate to the full set of results (both videos with and without faces).

File	Motion	#motion frames	FPS	Face detect	Face descriptor	Work
emptyroom1.mp4	ALWAYS	284	7.01531	284	0	51.474318002
emptyroom1.mp4	EVERY2	142	13.8634	142	0	25.737159001
emptyroom1.mp4	EVERY10	28	59.6348	28	0	5.0749327608
emptyroom2.mp4	ALWAYS	284	7.13093	284	0	51.474318002
emptyroom2.mp4	EVERY2	142	14.0164	142	0	25.737159001
emptyroom2.mp4	EVERY10	28	60.389	28	0	5.0749327608
lightingchange1.mp4	ALWAYS	284	7.09866	284	0	51.474318002
lightingchange1.mp4	EVERY2	142	14.0027	142	0	25.737159001
lightingchange1.mp4	EVERY10	28	59.9115	28	0	5.0749327608
lightingchange1.mp4	MSE	8	61.5062	8	0	1.4499807888
lightingchange1.mp4	MSE_WITH_BLUR	8	56.9465	8	0	1.4499807888
lightingchange2.mp4	ALWAYS	284	7.12506	284	0	51.474318002
lightingchange2.mp4	EVERY2	142	14.0136	142	0	25.737159001
lightingchange2.mp4	EVERY10	28	60.2868	28	0	5.0749327608
movinghuman4.mp4	ALWAYS	285	7.10193	285	0	51.655565601
movinghuman4.mp4	EVERY2	142	14.0395	142	0	25.737159001
movinghuman4.mp4	EVERY10	28	60.2857	28	0	5.0749327608
movinghuman4.mp4	CONTOURS	150	11.4587	150	0	27.18713979
movinghuman4.mp4	MSE	184	9.74712	184	0	33.349558142
movinghuman4.mp4	MSE_WITH_BLUR	96	16.4844	96	0	17.399769466
movinghuman4.mp4	DIFF	96	16.7721	96	0	17.399769466
movinghuman4.mp4	DIFF_WITH_BLUR	75	19.51	75	0	13.593569895
stillhuman1.mp4	ALWAYS	284	7.06372	284	0	51.474318002
stillhuman1.mp4	EVERY2	142	13.9686	142	0	25.737159001
stillhuman1.mp4	EVERY10	28	60.6459	28	0	5.0749327608
street1.mp4	ALWAYS	288	7.07641	288	0	52.199308397
street1.mp4	EVERY2	144	14	144	0	26.099654198
street1.mp4	EVERY10	28	64.64	28	0	5.0749327608
street2.mp4	ALWAYS	288	7.10161	288	0	52.199308397
street2.mp4	EVERY2	144	14.007	144	0	26.099654198
street2.mp4	EVERY10	28	64.5208	28	0	5.0749327608
street5.mp4	ALWAYS	288	7.08467	288	0	52.199308397
street5.mp4	EVERY2	144	13.9732	144	0	26.099654198
street5.mp4	EVERY10	28	64.692	28	0	5.0749327608
Mean			**24.011160606**			**32.726519866**

Table a7: Naive algorithm processing videos without faces

File	Motion	FPS	Faster?	Face detect	Face descriptor	Work	Less work?
emptyroom1.mp4	ALWAYS	33.6688	4.7993317473	56	0	10.149865522	0.1971830986
emptyroom1.mp4	EVERY2	59.4284	4.2867117734	28	0	5.0749327608	0.1971830986
emptyroom1.mp4	EVERY10	59.8089	1.0029194363	28	0	5.0749327608	1
emptyroom2.mp4	ALWAYS	33.5338	4.7025843754	56	0	10.149865522	0.1971830986
emptyroom2.mp4	EVERY2	59.4769	4.2433791844	28	0	5.0749327608	0.1971830986
emptyroom2.mp4	EVERY10	59.5252	0.9856960705	28	0	5.0749327608	1
lightingchange1.mp4	ALWAYS	33.5617	4.7278923064	56	0	10.149865522	0.1971830986
lightingchange1.mp4	EVERY2	58.9502	4.2099166589	28	0	5.0749327608	0.1971830986
lightingchange1.mp4	EVERY10	59.3464	0.9905677541	28	0	5.0749327608	1
lightingchange1.mp4	MSE	74.6332	1.2134256384	2	0	0.3624951972	0.25
lightingchange1.mp4	MSE_WITH_BLUR	67.9054	1.1924420289	2	0	0.3624951972	0.25
lightingchange2.mp4	ALWAYS	32.2446	4.5255197851	56	0	10.149865522	0.1971830986
lightingchange2.mp4	EVERY2	58.1612	4.15033967	28	0	5.0749327608	0.1971830986
lightingchange2.mp4	EVERY10	58.0536	0.9629570652	28	0	5.0749327608	1
movinghuman4.mp4	ALWAYS	33.0111	4.6481871829	57	0	10.33111312	0.2
movinghuman4.mp4	EVERY2	59.4308	4.2331137149	28	0	5.0749327608	0.1971830986
movinghuman4.mp4	EVERY10	59.8475	0.9927312779	28	0	5.0749327608	1
movinghuman4.mp4	CONTOURS	32.7524	2.8582998071	31	0	5.6186755566	0.2066666667
movinghuman4.mp4	MSE	31.2833	3.2094916242	39	0	7.0686563454	0.2119565217
movinghuman4.mp4	MSE_WITH_BLUR	41.0282	2.4889107277	21	0	3.8061995706	0.21875
movinghuman4.mp4	DIFF	45.3559	2.7042469339	19	0	3.4437043734	0.1979166667
movinghuman4.mp4	DIFF_WITH_BLUR	45.8376	2.3494413121	14	0	2.5374663804	0.1866666667
stillhuman1.mp4	ALWAYS	33.5466	4.7491406794	56	0	10.149865522	0.1971830986
stillhuman1.mp4	EVERY2	60.3082	4.3174119096	28	0	5.0749327608	0.1971830986
stillhuman1.mp4	EVERY10	59.5892	0.9825759037	28	0	5.0749327608	1
street1.mp4	ALWAYS	33.5214	4.7370630023	57	0	10.33111312	0.1979166667
street1.mp4	EVERY2	63.8759	4.5625642857	28	0	5.0749327608	0.1944444444
street1.mp4	EVERY10	63.9969	0.990051052	28	0	5.0749327608	1
street2.mp4	ALWAYS	33.6046	4.7319692295	57	0	10.33111312	0.1979166667
street2.mp4	EVERY2	63.1361	4.5074676947	28	0	5.0749327608	0.1944444444
street2.mp4	EVERY10	63.5735	0.985317913	28	0	5.0749327608	1
street5.mp4	ALWAYS	33.5898	4.7411947204	57	0	10.33111312	0.1979166667
street5.mp4	EVERY2	64.1369	4.5899937022	28	0	5.0749327608	0.1944444444
street5.mp4	EVERY10	63.9294	0.9882118345	28	0	5.0749327608	1
Mean		**46.276327273**	**2.9745537079**			**7.4092422711**	**0.3559837194**

Table a8: Face manager processing videos without faces with detect set to every 5 frames

File	Motion	FPS	Faster?	Face detect	Face descriptor	Work	Less work?
emptyroom1.mp4	ALWAYS	59.7824	8.5217046716	28	0	5.0749327608	0.0985915493
emptyroom1.mp4	EVERY2	59.5855	4.2980437699	28	0	5.0749327608	0.1971830986
emptyroom1.mp4	EVERY10	59.5474	0.9985344128	28	0	5.0749327608	1
emptyroom2.mp4	ALWAYS	59.1282	8.291793637	28	0	5.0749327608	0.0985915493
emptyroom2.mp4	EVERY2	59.1246	4.2182443423	28	0	5.0749327608	0.1971830986
emptyroom2.mp4	EVERY10	59.5818	0.9866333273	28	0	5.0749327608	1
lightingchange1.mp4	ALWAYS	59.5442	8.3880901466	28	0	5.0749327608	0.0985915493
lightingchange1.mp4	EVERY2	59.0009	4.2135373892	28	0	5.0749327608	0.1971830986
lightingchange1.mp4	EVERY10	59.4045	0.9915375178	28	0	5.0749327608	1
lightingchange1.mp4	MSE	77.4154	1.2586601026	1	0	0.1812475986	0.125
lightingchange1.mp4	MSE_WITH_BLUR	69.9188	1.2277980209	1	0	0.1812475986	0.125
lightingchange2.mp4	ALWAYS	57.717	8.1005633637	28	0	5.0749327608	0.0985915493
lightingchange2.mp4	EVERY2	57.4893	4.1023933893	28	0	5.0749327608	0.1971830986
lightingchange2.mp4	EVERY10	57.5428	0.954484232	28	0	5.0749327608	1
movinghuman4.mp4	ALWAYS	59.8534	8.4277654102	28	0	5.0749327608	0.098245614
movinghuman4.mp4	EVERY2	59.8823	4.2652729798	28	0	5.0749327608	0.1971830986
movinghuman4.mp4	EVERY10	60.0055	0.9953521316	28	0	5.0749327608	1
movinghuman4.mp4	CONTOURS	44.1352	3.8516760191	15	0	2.718713979	0.1
movinghuman4.mp4	MSE	45.5525	4.6734317419	19	0	3.4437043734	0.1032608696
movinghuman4.mp4	MSE_WITH_BLUR	56.2236	3.4107155856	8	0	1.4499807888	0.0833333333
movinghuman4.mp4	DIFF	59.284	3.5346796167	9	0	1.6312283874	0.09375
movinghuman4.mp4	DIFF_WITH_BLUR	54.65	2.8011276269	7	0	1.2687331902	0.0933333333
stillhuman1.mp4	ALWAYS	59.9426	8.4859818906	28	0	5.0749327608	0.0985915493
stillhuman1.mp4	EVERY2	60.1336	4.3049124465	28	0	5.0749327608	0.1971830986
stillhuman1.mp4	EVERY10	60.2681	0.993770395	28	0	5.0749327608	1
street1.mp4	ALWAYS	64.0772	9.0550434472	28	0	5.0749327608	0.0972222222
street1.mp4	EVERY2	63.9554	4.5682428571	28	0	5.0749327608	0.1944444444
street1.mp4	EVERY10	64.1382	0.992237005	28	0	5.0749327608	1
street2.mp4	ALWAYS	64.3479	9.0610298228	28	0	5.0749327608	0.0972222222
street2.mp4	EVERY2	64.2729	4.588627115	28	0	5.0749327608	0.1944444444
street2.mp4	EVERY10	64.2533	0.9958540502	28	0	5.0749327608	1
street5.mp4	ALWAYS	64.2526	9.0692438745	28	0	5.0749327608	0.0972222222
street5.mp4	EVERY2	64.1886	4.5936936421	28	0	5.0749327608	0.1944444444
street5.mp4	EVERY10	63.7245	0.9850445186	28	0	5.0749327608	1
Mean		**55.030321212**	**4.0186200168**			**4.8531726263**	**0.2944588687**

Table a9: Face manager processing videos without faces with detect set to every 10 frames

File	Motion	#motion frames	FPS	Face detect	Face descriptor	Work
movinghuman1.mp4	ALWAYS	285	6.99147	285	131	52.336425289
movinghuman1.mp4	EVERY2	142	13.8212	142	65	26.074990144
movinghuman1.mp4	EVERY10	28	59.4694	28	13	5.1424989894
movinghuman1.mp4	CONTOURS	222	8.10999	222	93	40.720325294
movinghuman1.mp4	MSE	235	7.77391	235	108	43.154505109
movinghuman1.mp4	MSE_WITH_BLUR	202	8.87038	202	76	37.007017484
movinghuman1.mp4	DIFF	199	9.00798	199	73	36.447682482
movinghuman1.mp4	DIFF_WITH_BLUR	151	11.3038	151	42	27.586678281
movinghuman2.mp4	ALWAYS	285	7.07267	285	66	51.998594146
movinghuman2.mp4	EVERY2	142	13.98	142	29	25.887883665
movinghuman2.mp4	EVERY10	28	60.0822	28	5	5.1009197718
movinghuman2.mp4	CONTOURS	45	26.6799	45	38	8.3536432206
movinghuman2.mp4	MSE	72	20.7333	72	55	13.33568422
movinghuman2.mp4	MSE_WITH_BLUR	35	32.0895	35	29	6.4943906148
movinghuman2.mp4	DIFF	49	27.1425	49	36	9.0682388106
movinghuman2.mp4	DIFF_WITH_BLUR	20	40.3142	20	17	3.7133078094
street3.mp4	ALWAYS	288	7.07614	288	0	52.199308397
street3.mp4	EVERY2	144	13.9485	144	0	26.099654198
street3.mp4	EVERY10	28	64.7508	28	0	5.0749327608
street3.mp4	CONTOURS	56	23.9778	56	0	10.149865522
street3.mp4	MSE	59	24.5795	59	0	10.693608317
street3.mp4	MSE_WITH_BLUR	45	28.3909	45	0	8.156141937
street3.mp4	DIFF	58	24.6702	58	0	10.512360719
street3.mp4	DIFF_WITH_BLUR	55	24.3094	55	0	9.968617923
multi-face.mov	ALWAYS	910	4.3304	910	462	167.33651454
multi-face.mov	EVERY2	455	8.55762	455	231	83.668257271
multi-face.mov	EVERY10	91	38.269	91	46	16.732611974
multi-face.mov	CONTOURS	589	5.88014	589	434	109.01050813
multi-face.mov	MSE	589	5.92514	589	457	109.13004838
multi-face.mov	MSE_WITH_BLUR	550	6.27116	550	410	101.81711413
multi-face.mov	DIFF	543	6.35564	543	400	100.49640692
multi-face.mov	DIFF_WITH_BLUR	490	6.88764	490	350	90.630414084
Mean			**24.011160606**			**32.726519866**

Table a10: Naive algorithm processing videos with faces

File	Motion	FPS	Faster?	Face detect	Face descriptor	Work	Less work?
movinghuman1.mp4	ALWAYS	29.8817	4.2740224874	57	7	10.367494936	0.198093295
movinghuman1.mp4	EVERY2	55.277	3.9994356496	28	3	5.0905249674	0.1952263429
movinghuman1.mp4	EVERY10	58.2288	0.9791388512	28	5	5.1009197718	0.9919145891
movinghuman1.mp4	CONTOURS	26.0572	3.2129756017	44	3	7.990486545	0.1962284556
movinghuman1.mp4	MSE	25.9112	3.3330975018	47	3	8.5342293408	0.1977598705
movinghuman1.mp4	MSE_WITH_BLUR	28.0949	3.1672713007	41	3	7.4467437492	0.2012251799
movinghuman1.mp4	DIFF	29.3357	3.2566346728	40	4	7.2706935528	0.1994830139
movinghuman1.mp4	DIFF_WITH_BLUR	32.6138	2.8852067446	31	3	5.6342677632	0.2042387164
movinghuman2.mp4	ALWAYS	31.613	4.4697405647	57	6	10.362297533	0.1992803402
movinghuman2.mp4	EVERY2	57.6394	4.1229899857	28	4	5.0957223696	0.1968381207
movinghuman2.mp4	EVERY10	58.7396	0.9776539474	28	5	5.1009197718	1
movinghuman2.mp4	CONTOURS	50.8338	1.905321984	8	2	1.4603755932	0.1748190047
movinghuman2.mp4	MSE	48.7417	2.3508896317	14	4	2.5582559892	0.1918353754
movinghuman2.mp4	MSE_WITH_BLUR	58.5219	1.8237086898	6	2	1.097880396	0.169050564
movinghuman2.mp4	DIFF	57.0113	2.1004439532	9	1	1.6364257896	0.1804568477
movinghuman2.mp4	DIFF_WITH_BLUR	58.2604	1.4451582817	4	1	0.7301877966	0.1966407942
street3.mp4	ALWAYS	33.6783	4.7594168572	57	0	10.33111312	0.1979166667
street3.mp4	EVERY2	64.2875	4.6089185217	28	0	5.0749327608	0.1944444444
street3.mp4	EVERY10	64.2085	0.9916248139	28	0	5.0749327608	1
street3.mp4	CONTOURS	49.9261	2.0821801833	10	0	1.812475986	0.1785714286
street3.mp4	MSE	54.5246	2.2182957343	12	0	2.1749711832	0.2033898305
street3.mp4	MSE_WITH_BLUR	56.1866	1.979035536	8	0	1.4499807888	0.1777777778
street3.mp4	DIFF	54.1827	2.1962813435	12	0	2.1749711832	0.2068965517
street3.mp4	DIFF_WITH_BLUR	49.3086	2.0283758546	11	0	1.9937235846	0.2
multi-face.mov	ALWAYS	19.0121	4.3903796416	182	21	33.096208391	0.1977823458
multi-face.mov	EVERY2	35.7845	4.1815948827	91	17	16.58188731	0.1981861204
multi-face.mov	EVERY10	37.3048	0.9748046722	91	36	16.680637952	0.9968938488
multi-face.mov	CONTOURS	17.8884	3.0421724653	120	17	21.838067669	0.2003299319
multi-face.mov	MSE	18.5507	3.1308458534	118	18	21.480769874	0.1968364368
multi-face.mov	MSE_WITH_BLUR	18.9573	3.0229335562	111	17	20.206839282	0.1984621098
multi-face.mov	DIFF	19.6037	3.0844572694	108	14	19.64750428	0.1955045447
multi-face.mov	DIFF_WITH_BLUR	20.4182	2.9644696877	95	14	17.291285498	0.1907889937
Mean		**46.276327273**	**2.9745537079**			**7.4092422711**	**0.3559837194**

Table a11: Face manager processing videos with faces with detect set to every 5 frames

File	Motion	FPS	Faster?	Face detect	Face descriptor	Work	Less work?
movinghuman1.mp4	ALWAYS	50.6233	7.2407233386	28	7	5.1113145762	0.0976626613
movinghuman1.mp4	EVERY2	55.3132	4.0020548143	28	3	5.0905249674	0.1952263429
movinghuman1.mp4	EVERY10	57.9501	0.9744524075	28	5	5.1009197718	0.9919145891
movinghuman1.mp4	CONTOURS	36.9667	4.5581683825	21	3	3.8217917772	0.0938546475
movinghuman1.mp4	MSE	37.3778	4.8081081464	24	3	4.365534573	0.1011605755
movinghuman1.mp4	MSE_WITH_BLUR	39.6567	4.4706878398	20	3	3.6405441786	0.098374428
movinghuman1.mp4	DIFF	41.9493	4.6569042116	20	4	3.6457415808	0.1000267049
movinghuman1.mp4	DIFF_WITH_BLUR	43.1442	3.8167872751	16	2	2.910356382	0.1054986161
movinghuman2.mp4	ALWAYS	55.8973	7.9032812219	28	6	5.106117174	0.0981972159
movinghuman2.mp4	EVERY2	58.0087	4.1494062947	28	4	5.0957223696	0.1968381207
movinghuman2.mp4	EVERY10	58.9935	0.9818798246	28	5	5.1009197718	1
movinghuman2.mp4	CONTOURS	56.8592	2.1311624107	4	2	0.7353851988	0.0880316743
movinghuman2.mp4	MSE	57.6114	2.7786893548	8	4	1.4707703976	0.1102883342
movinghuman2.mp4	MSE_WITH_BLUR	64.4049	2.0070396859	3	1	0.548940198	0.084525282
movinghuman2.mp4	DIFF	63.0008	2.3211126462	6	1	1.0926829938	0.1204956129
movinghuman2.mp4	DIFF_WITH_BLUR	63.9035	1.5851362547	1	1	0.1864450008	0.0502099504
street3.mp4	ALWAYS	64.2737	9.0831583321	28	0	5.0749327608	0.0972222222
street3.mp4	EVERY2	64.3485	4.6132917518	28	0	5.0749327608	0.1944444444
street3.mp4	EVERY10	64.305	0.993115143	28	0	5.0749327608	1
street3.mp4	CONTOURS	58.4047	2.4357822653	4	0	0.7249903944	0.0714285714
street3.mp4	MSE	65.4381	2.6623039525	6	0	1.0874855916	0.1016949153
street3.mp4	MSE_WITH_BLUR	63.1093	2.2228707086	4	0	0.7249903944	0.0888888889
street3.mp4	DIFF	64.3976	2.6103396	6	0	1.0874855916	0.1034482759
street3.mp4	DIFF_WITH_BLUR	57.7589	2.3759903576	5	0	0.906237993	0.0909090909
multi-face.mov	ALWAYS	33.4158	7.7165619804	91	15	16.571492506	0.099030941
multi-face.mov	EVERY2	35.8086	4.1844110863	91	19	16.592282114	0.1983103587
multi-face.mov	EVERY10	37.2613	0.9736679819	91	36	16.680637952	0.9968938488
multi-face.mov	CONTOURS	24.5247	4.1707680429	59	12	10.755977144	0.0986691772
multi-face.mov	MSE	25.5552	4.3130120132	58	12	10.574729545	0.0969002553
multi-face.mov	MSE_WITH_BLUR	25.9255	4.1340836464	55	12	10.030986749	0.098519653
multi-face.mov	DIFF	26.6735	4.1968236086	54	11	9.8445417486	0.0979591415
multi-face.mov	DIFF_WITH_BLUR	27.216	3.9514260327	47	12	8.5810059606	0.0946813059
Mean		**55.030321212**	**4.0186200168**			**4.8531726263**	**0.2944588687**

Table a11: Face manager processing videos with faces with detect set to every 10 frames