Seminar Project (5 ECTS)
for

Adaptive Frame Rate for Egocentric Vision

by Maria Monzon

Date: July 15, 2019
Supervisors: Giovanni Schiboni, Prof. Dr. Oliver Amft

Contents

1 Introduction

The increasing development new media and data acquisition techniques have lead to new innovative video recording set ups. A clear example of that is the egocentric video, where a camera on head or on the chest approximates the visual field of the camera wearer.

This new camera setting offers a valuable perspective to understand user's activities and their context. In our case, the wearable head-mounted egocentric camera set up pursued a dietary event spotting in a free living condition. [1]

A main feature of free-living data is a long recording duration. However, this kind of camera often acquire irrelevant data for the analysis task.

In this work we introduce an adaptive sampling strategy to dynamically tune the sampling rate of a wearable egocentric camera. The adaptive sampling rate is based on a context measure. It is a motion measure, indicative if the recorded activity might be of our interest.

The clear advantages of an adaptive frame rate is saving energy as the camera will not be acquiring data full time. This would help to overcome the limitations of the video recording, in terms of power consumption, while maintaining acceptable performance. Furthermore, less data acquisition will also mean an energy saving in processing.

This new set up will enable longer video recordings, improve the autonomy of the egocentric video frameworks and the the need of small batteries.

2 Related work

In literature we can find similar works that aim to reduce the energy consumption. *Salim et al.* [2] proposed a method to combine frame rate and similarity detection for video sensors. A context measure based on the similarity between consecutive frames was employed, composed by a measure of colour intensity and comparison between edges. The number of frames employed in the sensor network were reduced without losing any important information.

A similar study focused on egocentric video was carried out by *Possas et al.* [3]. A reinforcement learning model-free method to learn energy-aware policies was introduced. Their work maximized the use of low-energy cost predictors while keeping competitive accuracy levels. The context measure used was a motion predictor and a vision predictor, based on the type of activity detected.

For the simulation of energy consumption examples of energy model for image sensors can be found. *LiKamWa et al.*[4] introduced an energy characteristics of CMOS image sensors in the context of computer vision applications. The authors described an experimental power characterization of five CMOS image sensors energy in order to evaluate the energy per pixel under various image quality requirements and energy-proportional mechanisms.

We based our investigation on results presented by LiKamWa et al for the simulation of the sensing components in order to evaluate the effectiveness of adaptive frame rate strategy.

3 Methodology Summary

The egocentric video with adaptive sampling framework is divided in two main blocks: adaptive sampling strategy and machine learning pipeline, as showed in the block diagram of Figure 1.

The adaptive sampling strategy starts processing the CMOS acquired frames from which a motion based context measure is extracted. From that context measure a mapping into a linear response model is done. Finally, the output of the response model indicates the compression factor for the next instant. Once the frame is sampled, it is forward feeded to the onvolutional neural network (CNN) as a knowlwdge extractor. The output will be the input of the heuristic decision to determine if a dietary event is present.

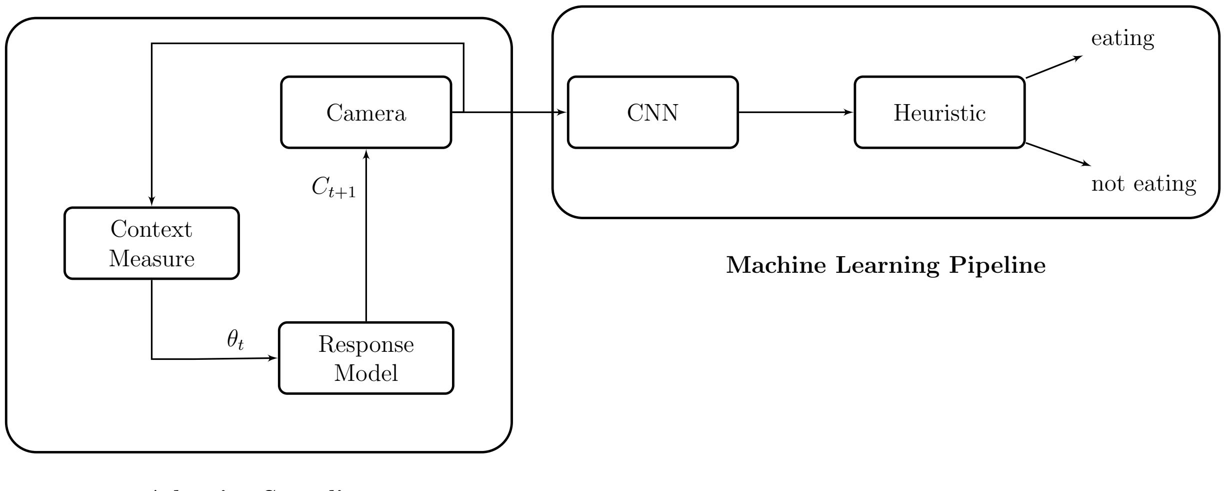

Figure 1: Block diagram of our framework

We implemented a computational simulation to evaluate the efficiency of our sampling strategy in terms of energy consumption. Our energy model considers the CMOS sensor, the response model and CNN processing by the microcontroller. For the energy consumption model an analytic breakdown of the code was done. For each algorithm a counting of floating-point operations (FLOP) was carried out.

4 Methods

4.1 Dataset

The original dataset is taken from the work of *Schiboni et al.* [1]. The video recordings consist of normal daily routine activities such as attending academic classes, having lunch, displacement in public transport...The average recording time per day is around 8 hours, that were downsampled to only one image per second. The total dataset is composed 14373 annotated frames.

Day	Total Frames	Bottle	Can	Dish	Mug	Glass	Annotated Frames
Day 1	31980	580	-	2973	631	497	2791
Day 2	22570	220	4	1290	2	1	1461
Day 3	32814	947	417	4034	618	1836	4574
Day 4	24953	446	108	1790	3	9	2312
Day 5	26895	81	7	2137	1069	1104	3234
Total	139212	2242	536	12224	2323	3447	14373

Table 1: Dataset Annotations Summary

This dataset was originally to train neural networks, but a further dataset containing the spotting of dietary events was derived. In this activity dataset, 5 days of video were labelled with activity interval such as fluid drink, food intake, but also food preparation and cooking, cleaning dishes...

4.2 Adaptive Sampling

Adaptive sampling is a technique designed such that the sampling is modified in real time based on a feature learned from previous acquired data. In our work, the adaptive sampling strategy is composed of two steps: context measure (section 4.2.1) and response model (section 4.2.2).

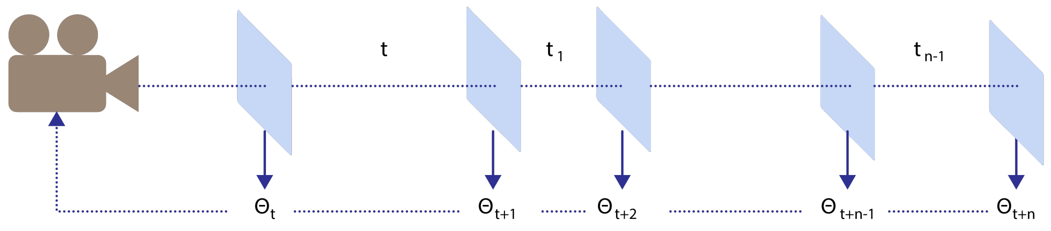

Figure 2: Adaptive Sampling overview

4.2.1 Context measure

Our aim for the seminar is to extract a context measure for tuning the adaptive frame rate. A context measure is any quantifiable information that can be used to characterize the situation of an action or an environment of an entity. In our work we implement a motion based context measure, following a common pipeline in computer vision.

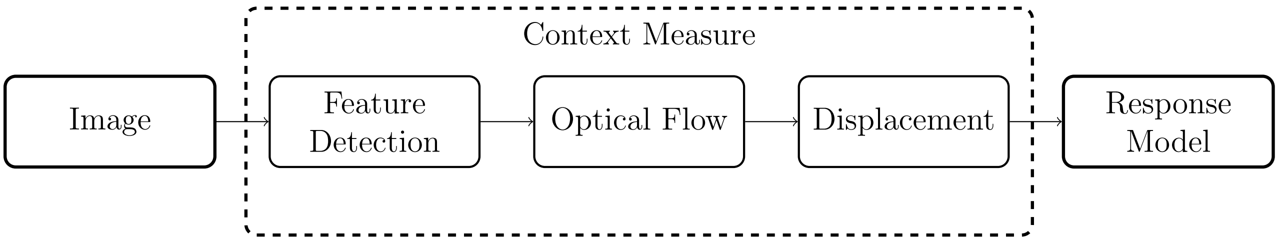

Figure 3: Adaptive sampling block diagram. First features are detected for every image. Then the optical flow is calculated for those features between two frames. Finally the motion measure is derived from the optical flow, that will be the input feed forwarded to the response model.

4.2.1.1 Feature detection

Features, in image processing, are specific structures such as points, edges, corners or objects. Feature detection and extraction derive values (or features) from a given image, descriptive for the object detection, facilitating the further learning steps.

Shi Tomasi Corner Detection A corner is an image point which is composed by two different dominant edge directions in the near proximity of the point. *Shi and Tomasi* [5] follows an identical approach to Harris Detector[6], but they made a small modification in the score function of the corners.

Both detectors discern points based on the local intensity variation, quantifying local changes within the image and small shifted neighbor patches. A small region around the corner should have a large intensity change when correlated with windows shifted in any direction.

Our implementation in python is done using the Open CV library. Shi-Tomasi method algorithm finds the N strongest corners in the image, in gray scale. It take as an input the following parameters:

- $maxCorners$: maximum number of corners wanted to be found
- $blockSize$: It is the size of neighbourhood considered for corner detection

- *thr*: minimum quality level of corner below which everyone is rejected (between 0-1)
- *minDist*: minimum euclidean distance between corners detected

The idea and main steps form the Shi Tomasi corner detection can be found in the next algorithm. Further details can be found in section 4.4.2.1

Algorithm 1 ShiTomasiCornerDetection

```
    Input: Gray scale Image $I$
    Output The 2D coordinates of a corner pixels
 1: procedure ShiTomasiCornerDetection($I, maxCorners, thr, minDist, blockSize$)
 2:     $I_x, I_y \leftarrow$ SpatialImageDerivatives($I$)
 3:     $I_G \leftarrow$ AutocorrelationMatrix($I_x, I_y$)
 4:     for $p \in I$ do
 5:         $M \leftarrow$ SecondMomentMatrix($I_G, blockSize$)
 6:         $\lambda_{min} \leftarrow$ MinEigenvalue($M$)
 7:         if $\lambda_{min} \geq thr$ then
 8:             $corners(p) \leftarrow \lambda_{min}$
 9:         else
10:             $corners(p) \leftarrow 0$
11:         $corners \leftarrow$ selectCorners($corners$)
12:     return $corners$
```

4.2.1.2 Lukas Kanade Optical Flow Estimation

The optical flow is the apparent motion of the brightness pattern in an image caused by the projection of real motion on the focal plane [7].

The *Lucas Kanade* method assumes that the displacement of the image content between two consecutive instants within a neighbourhood window (adjacent frames) is constant. The algorithm only uses the gray value at each pixel, the level of intensity of the image. Lucas-Kanade method takes a $k \times k$ patch around the point and assumes that the flow is constant in a local neighbourhood of the pixel i.e. the level within a local image window. So all the $k \times k$ points have the same motion. So now our problem turns into solving two unknown variables with $k \times k$ equations. The final solution of the Lucas Kanade optical flow is a 2 equations system,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {I_{x_i}}^2 & \sum_i I_{x_i} I_{y_i} \\ \sum_i I_{x_i} I_{y_i} & \sum_i {I_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_{x_i} I_{t_i} \\ -\sum_i I_{y_i} I_{t_i} \end{bmatrix} \tag{1}$$

where I_x, I_y state for image derivatives in horizontal and vertical directions correspondingly and I_t the time derivative. The summatory extends along all the values of the neighbourhood patch. The pseudo-code of the Lucas Kanade implementation can be seen in more details in the Algorithm 2

Algorithm 2 Lucas Kanade Optical Flow

```
    Input: 2D $I, J$ images and pixel with the width of a window
    Output $\boldsymbol{u}$ The 2D displacement of the pixel represented by W
 1: procedure LucasKanade(I,J,W)
 2:     $\boldsymbol{u} \leftarrow 0$
 3:     $iter \leftarrow 0$
 4:     $G_x, G_y \leftarrow$ Gradient($I$)
 5:     while $|\boldsymbol{u}_\Delta| < \epsilon$ or $iter \geq MaxIter$ do        ▷ Until convergence
 6:         $Z \leftarrow$ Compute2DGradients($G_x, G_y, W$).
 7:         $e \leftarrow$ ComputeErrorVector($I, J, G_x, G_y, W, \boldsymbol{u}$)
 8:         $\boldsymbol{u}_\Delta \leftarrow$ SolveGRadientIncrement($Z, e$)
 9:         $\boldsymbol{u} \leftarrow \boldsymbol{u} + \boldsymbol{u}_\Delta$
10:         $iter \leftarrow iter + 1$
11:     return $\boldsymbol{u}$
```

Our implementation in python follows the further approach of the pyramidal implementation from *Bouguet* [8]. An iterative implementation provides sufficient local tracking accuracy. The main idea to apply Iterative Lukas Kanade Algorithm is:

1. Estimate displacement at each pixel by solving Lucas Kanade optical flow equations
2. Warp I_t towards I_{t+1} using the computed flow
3. Repeat until convergence or a fixed number of iterations

The iterative algorithm can be schemitized in the following diagram. (Figure 4) but further details can be found in the analytic breakdown of the code in Section 4.4.2.2

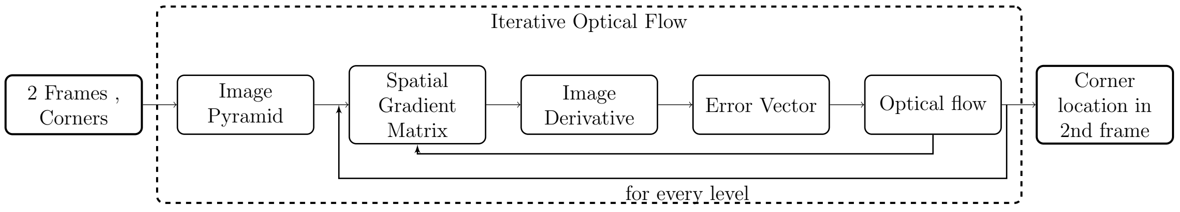

Figure 4: Pyramidal Lucas Kanade optical flow Detector

4.2.1.3 Derive Displacement

Th output of the optical flow are the keypoint descriptors coordinates in the second frame, i.e. corners. The final step is to the extract the context measure θ. The mean of the euclidean distance between all the corners in both frames was computed. Our context measure is a motion based indicative of the total displacement between 2 sampled frames.

$$\theta_t = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(u_{x_i} - v_{x_i})^2 + (u_{y_i} - v_{y_i})^2} \tag{2}$$

4.2.2 Response Model

The feed-forwarded response model takes as input the context measure, the derived mean displacement from the optical flow between frames. A linear mapping function converts the input to compression a factor according to Equation. 3

$$C_{t+1} = C_{min} + \left[\frac{C_{max} - C_{min}}{\theta_{max} - \theta_{min}} \cdot \theta_t - \theta_{min} \right] \tag{3}$$

where θ_t represents the context measure, θ_{min} the minimum and θ_{max} maximum context measure values and C_{max}, C_{min} the limits of the compression output factor.

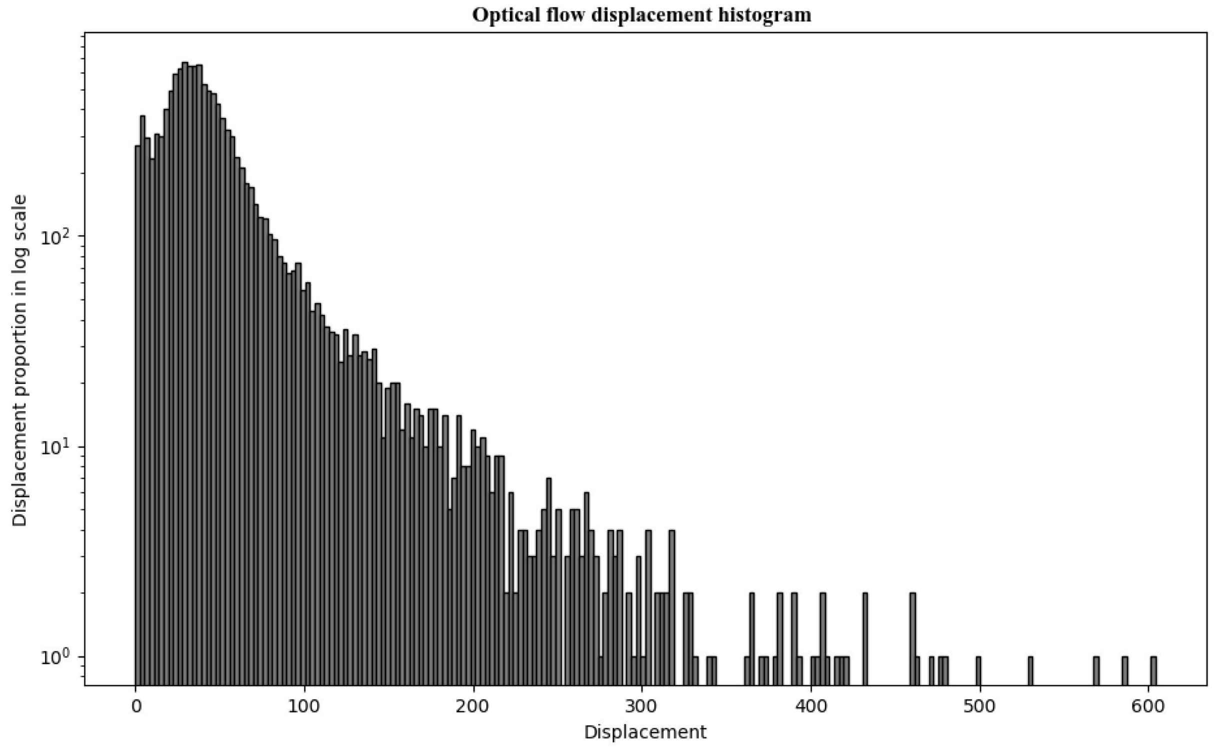

Figure 5: Example of histogram where the displacement context measure for day 2 is shown. The y-axis is plotted in logarithm scale.

We fixed these parameter by plotting various histograms of the context measure.The parameter θ_{min} was fixed at a value of 5 whereas θ_{max} is varying to obtain different sampling reduction. Analogously, a maximum compression factor was set to 1 and the minimum to 0.1. These sampling strategy system parameters can be modified.

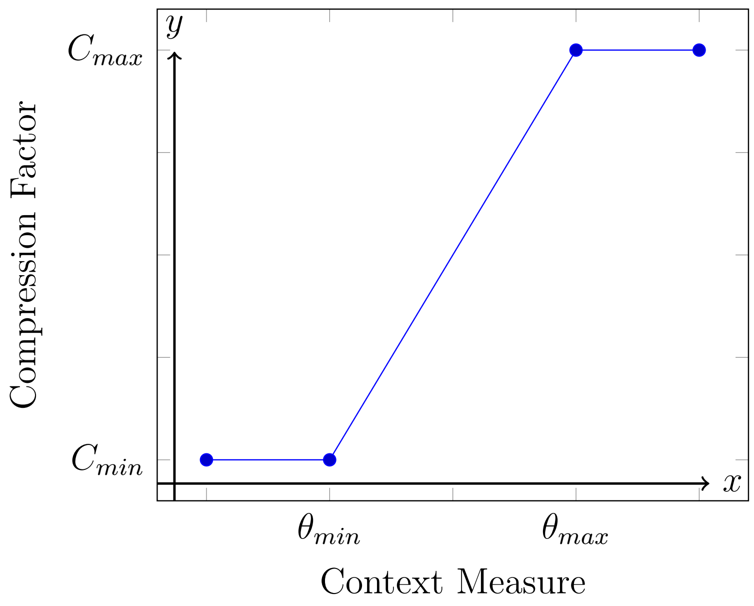

Figure 6: Linear response model for the context measure and compression factor of the adaptive sampling framework.

The compression factor is set as a threshold for a further random sampling strategy. Random sampling is a kind of non-uniform sampling where a sample is considered based on a stochastic decision. The compression factor can be defined as the portion of sampled frames between the total within batch.

$$C = \frac{Sampled\ frames}{Total\ frames} \tag{4}$$

Therefore a compression factor of 0.1 will mean that there is a probability that 1 out of 10 frames will be not sampled. It represents the probability of a skipping a frame in our random adaptive sampling strategy.

4.3 Machine Learning Pipeline

4.3.1 Convolutional Neural Network (CNN)

In order to detect dietary events, in our work we employ the deep convolution network presented in [1]. Specifically, a Darknet framework was implemented but following a YOLO9000 [9] structure to enable the object detection in real time. Both classification and detection from this work is done on using that pretrained network.

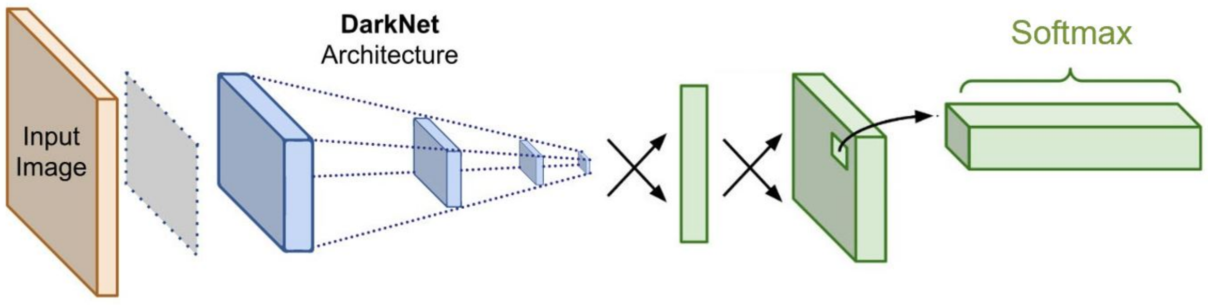

Figure 7: CNN Arquitecture representation

This Deep Neural Network was trained using the so-called transfer learning technique. A neural network model is first trained on similar context and then it is exploited to improve generalization of another deep learning network. Classification uses only the first 16 layers of the total proposed in Darknet 19.

4.3.2 Heuristics

The output of the CNN are the labels indicating if a frame contains a dietary object, such as bottle, can, dish, mugs or glass. Individual frames therefore were associated with a binary value:

$$F = \begin{cases} 1, & \text{if object present} \\ 0, & \text{otherwise} \end{cases}$$

An intuitive heuristic was adopted determine a true positive. If three or more consecutive frames had a diatary object $F = 1$, then it was considered to spot a dietary event .

4.4 Energy Model

In order to determine the energy saving score of our approach, it is necessary to determine the energy consumption of our set up. Unfortunately, it is unfeasible to perform actual energy measures without interfere in the energy consumption process. Therefore, to validate the energy saving, the final point of our work was to derive and simulate the energy consumption models for the microprocessor and the camera.

$$E_{total} = E_{CMOS} + E_{microcontroller} \tag{5}$$

The total energy will be the sum of the both, sensing component and processor of our set up. Our energy will consider the energy of CMOS sensor for the video recording. The microcontroller energy consumption corresponds to the processing of the response model and the computations for the machine learning pipeline.

4.4.1 CMOS Sensor Energy Model

The energy consumption model for the CMOS sensor is calculated based on the actual work of *Likamwa et all.*[4] In their work they revealed that the energy consumption of the sensor can be reduced by putting components in standby after exposure time. According to this aggressive standby mode, the corresponding to the energy per frame is given by the following expression:

$$E_{frame}[\text{Wh}] \approx P_{idle}[\text{W}]T_{exp}[\text{h}] + P_{active}[\text{W}]T_{active}[\text{h}] \tag{6}$$

where P_{idle} is the power consumption in idle state, P_{active} power consumption in active state, T_{exp} corresponds to the exposure time of the camera and T_{active} time in active state.

The power consumption values are extracted from [4]. The power consumption values for a CMOS with similar characteristics to our set up are:

$$P_{active} = 225.1 \text{ mW} \qquad P_{idle} = 218.6 \text{ mW}$$

The time in active state can be approximated as the relation between number of pixels in a frame and the clock frequency. Our camera records in a resolution of 640x640.

$$T_{active} = \frac{N}{f} = \frac{640 \cdot 640\ [pixel]}{24\ [MHz]} = \frac{409600}{24000000} = 0.017066\ s = 4.74 \cdot 10^{-6} \text{ h} \tag{7}$$

The exposure time value is fixed at 50 ms. It is the maximum reasonable value for outdoor conditions, according to *Likamwa et al.*

$$T_{exp} = 50.00\ ms = 13.88 \cdot 10^{-6} \text{ h}$$

When the CMOS sensor completes the acquisition of a frame it enters aggressive standby mode. The energy saving comes due to the fact the frame rate is adapted extending the standby time, where it consumes minimal power. Therefore, the energy per frame remains constant even if the frame rate changes.

We will like to remark that our frame rate is 1 FPS, thus the time of standby is defined as the remaining time until 1 second (frame time). The summary of the energy consumption simulation model can be found in Table 2

Parameters	Value
P_{active}	225.1 mW
P_{idle}	218.6 mW
T_{active}	0.017 s
T_{exp}	0.0500 s
$T_{standby}$	0.9323 s
E_{frame}	4.1 µWh

Table 2: CMOS energy model characterization summary

4.4.2 Microcontroller Energy Model

The microcontroller also needs an energy estimation that is proportional to the execution time. The energy model can be derived directly from the execution time for both, the response model and the CNN. The energy of the microcontroller can be estimated as

$$E_{frame}[\mathrm{Wh}] = P_{active}[\mathrm{W}] \cdot t_{exec}[\mathrm{h}] \tag{8}$$

The micro controller used in our work is Arm Cortex A7 from a raspberry Pi 2 (ARM Cortex A72 Quad Core). The details about the microcontroller can be found in Table 3

Microcontroller	$P_{average}$	P_{idle}	GFLOPS/W	GFLOPS	Frequency
ARM Cortex A7	3.4 W	1.8 W	0.432	1.47	1.5 GHz

Table 3: ARM Cortex A7 technical especifications

The execution time can be estimated as the number of floating point operator executed divided by the number of cycles the mincrocontroler needs. FLOPS stands for floating point operations per second a processor can perform per second i.e. a computer speed measure. The power consumption can be derived from the measure GFLOPS per Watt (GFLOPS/W) i.e. the number of 10^9 floating point operations per second that can be executed with a Watt of electrical power.

$$P_{active} = \frac{GFLOPS}{GFLOPS/W} \approx 3.4W \tag{9}$$

For the estimation of the execution time we need consequently the number of FLOP. FLOP is single floating operations done by the microcontroller during the runtime of the simulation. That is estimated by providing for each function the number of arithmetical operations.

$$E_{microcontroller} = E_{context} + E_{CNN} \tag{10}$$

Each algorithm has been decomposed in sub-algorithms blocks. The total number of estimations has been derived. In the next subsection the analytical break down of the code is detailed. For the purposes of FLOPS measurements, usually only additions and multiplications are included. Things like divisions, square roots... have been also taken into as single operation. The comparisons are depreciated as they are too trivial.

4.4.2.1 Analytic Break Down of Shi Tomasi Corner Detector

The algorithm of Shi Tomashi for corner detection can be decomposed in this four main steps: calculate image derivatives with a gradient filter, calculate the weighted auto-correlation matrix from the. For each corner calculate the minimum eigenvalue and finally compute a cornerness score and select the strongest ones.

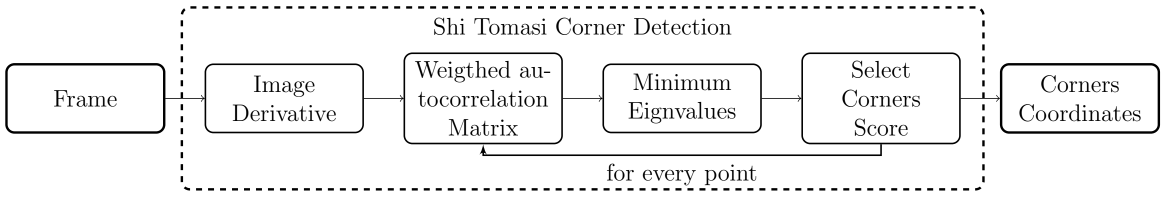

Figure 8: Shi Tomasi corner detector

The first step of the algorithm is to perform the spatial image derivative of the frame. In our implementation the derivate are calculated by the convolving it with the Sobel filter. The output of the algorithm is the spatial horizontal and vertical derivates of the image.

Algorithm 3 SpatialImageDerivatives

```
    Input: Gray scale Image $I$
    Output Spatial image derivatives with respect to x and y
1: procedure PartialDerivative($I, maxCorners, thr, minDistance, blockSize$)
2:     $kernel_x \leftarrow$ SobelFilterGx($m$)
3:     $kernel_y \leftarrow$ SobelFilterGy($m$)
4:     for $p_x, p_y \in I$ do
5:         $I_x[p_x, p_y] \leftarrow$ 2DConvolution($I, kernel_x$)
6:         $I_y[p_x, p_y] \leftarrow$ 2DConvolution($I, kernel_y$)
7:     return $I_x, I_y$
```

The mathematical formulation of 2-D convolution is given by

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$I_x\left[i, j\right] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} k_x\left[m, n\right] \cdot I\left[i-m, j-n\right] \tag{11}$$

The total FLOP operation are summarized on the following table where M corresponds to the x-size and N to the y-size of the Sobel filter and P_x, P_y to the pixels in the image correspondingly.

operation	add	mult	div	sqrt	comparisons
2D Convolution	$M \cdot N$	$M \cdot N$	0	0	0
Image derivatives	$2(P_xP_y)(M \cdot N)$	$2(P_xP_y)(M \cdot N)$	0	0	0

Table 4: Floating point operations for spatial image derivatives

Once we have the image derivates, the weighted autocorrelation matrix is calculated. This step is again subdivided in 2function in our analysis, the simple computation of the autocorrelation matrix and then the weighted sum. The autocorrelation matrix is just composed of the element wise multiplications of the image derivatives as follows:

$$I_G = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} = \begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix} \tag{12}$$

where I_x corresponds to the image x derivative and I_y the y-direction image derivative. The pseudo algorithm pf this simple function can be seen below.

Algorithm 4 AutocorrelationMatrix

```
    **Input**:Spatial image derivatives $I_x, I_y$
    **Output** Autocorrelation Matrix of the image $I_G$
1: **function** AUTOCORRELATIONMATRIX($I_x, I_y$)
2:     **for** $p_x, p_y \in I$ **do**
3:         $I_{xx}[p_x, p_y] \leftarrow I_x[p_x, p_y] \cdot I_x[p_x, p_y]$
4:         $I_{yy}[p_x, p_y] \leftarrow I_y[p_x, p_y] \cdot I_y[p_x, p_y]$
5:         $I_{xy}[p_x, p_y] \leftarrow I_x[p_x, p_y] \cdot I_y[p_x, p_y]$
6:     $I_G = [[I_{xx}, I_{xy}], [I_{xx}, I_{yy}]]$
7:     **return** $I_G$
```

operation	add	mult	div	sqrt	comparisons
Tensor computation	0	$P_x \cdot P_y$	0	0	0
Autocorrelation matrix	0	$3(P_x P_y)$	0	0	0

Table 5: Floating point operations for autocorrelation matrix

For each point p_x, p_y the weighted autocorrelation matrix is calculated, just averaged in neighbourhood of a point.The window w is a binary window. The average window or block size is specified as an input parameter.

$$M = \sum_{x=-w_x}^{w_x} \sum_{x=-w_y}^{w_y} w(x,\ y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{13}$$

Algorithm 5 Second Moment Matrix

```
    Input:Autocorrelation matrix of the image $I_G$ and window size
    Output Summed Second Moment Matrix $M$
 1: function SummedMatrix($I_G, blockSize$)
 2:     $M = [[0,0],[0,0]]$
 3:     $w \leftarrow blockSize/2$
 4:     for $i = -w$ to $w$ do
 5:         for $j = -w$ to $w$ do
 6:             $M_{11} \leftarrow M_{11} + I_{xx}[p_x + i, p_y + j]$
 7:             $M_{12} \leftarrow M_{12} + I_{xy}[p_x + i, p_y + j]$
 8:             $M_{22} \leftarrow M_{22} + I_{yy}[p_x + i, p_y + j]$
 9:     $M_{21} \leftarrow M_{12}$
10:     return $M$
```

The result is the second moment 2x2 matrix ***M*** which summarizes the predominant directions of the gradient in a specified neighbourhood. Note that the inverse diagonal elements are equal. The total operations of both blocks can be seen in the following table:

operation	add	mult	div	sqrt	comparisons
Weighted matrix	$3 \cdot 2(w_x \cdot w_y)$	0	0	0	0
Total per image	$3 \cdot 2(w_x \cdot w_y)(P_x P_y)$	0	0	0	0

Table 6: Floating point operations for autocorrelation weighted matrix

Shi and Tomasi is based on the minimum eigenvalue of the autocorrelation matrix. The score function for the corners is defined as

$$R = min(\lambda_1, \lambda_2) \tag{14}$$

where λ refers to the eigenvalues of the matrix $\boldsymbol{M}$. The autocorrelation matrix of an image is symmetric and positive semidefinite, yielding two real non-negative eigenvalues. Taking advantage of this condition, the eigenvalues can be derived directly from $\boldsymbol{M}$.

$$\lambda_{min} = \frac{M_{11} + M_{22} + \sqrt{(M_{11} - M_{22})^2 + 4M_{12}^2}}{2} \tag{15}$$

Algorithm 6 Select Minimum Eigenvalue

```
    Input:Second Moment Matrix $M$
    Output Minimum eigenvalue $\lambda_{min}$ of the Summed autocorrelation Matrix
 1: function MinEigenvalue($M$)
 2:     $\lambda_{min} \leftarrow 0.5 \cdot (M_{11} + M_{21} + sqrt((M_{11} - M_{21})^2 + 4M_{12}^2)$
 3:     return $\lambda_{min}$
```

operation	add	mult	div	sqrt	comparisons
Minimum eigenvalue	4	2	1	1	0
Total Image	$4(P_xP_y)$	$2(P_xP_y)$	P_xP_y	P_xP_y	0

Table 7: Floating point operations of the minimum eigenvalue computation

The final steps is to select the corners according to a specified quality level and the non-maxima suppression criteria. All corners below a quality level are rejected. The remaining corners based on quality are sorted in the descending order. In our analysis we named this function block as sort descending. Once we have all the sorted corners all all the nearby corners in the range of minimum distance are thrown away. The euclidean distance is computed .

$$D_e = \sqrt{(corner_{x_1} - corner_{x_2})^2 + (corner_{y_1} - corner_{y_2})^2} \tag{16}$$

The corners at closer than the minimum distance are thrown away. The final output of the Shi Tomasi corner detection algorithm are the $MaxNumber$ strongest corners. In our analysis we named this algorithm block as select corners, detailed below:

Algorithm 7 Select Corners

```
    Input: corners, distance $d$
    Output Selected corners
1: function SelectCorners($corners, d, N$)
2:     $cornersSorteed \leftarrow$ SortDescending($corners$)
3:     for $c \in cornersSorted$ do
4:         $i \leftarrow 0$
5:         for $corner \in cornersSorted$ do
6:             if EuclideanDistance($c, corner$) $> d$ then
7:                 $selCorners[i] \leftarrow corner$
8:                 $i \leftarrow i + 1$
9:     Return $selCorners[0 : N]$
```

operation	add	mult	div	sqrt	comparisons
Sort descending	0	0	0	0	n
Euclidean distance	3	2	0	1	0
Select Corners	$3C^2$	$2C^2$	0	C^2	$C \cdot C^2$

Table 8: Floating point operations of selection of strongest corners

The final floating point operations per frame for the Shi Tomasi corner detector are summarized in the table below.

operation	add	mult	div	sqrt
Image derivatives	$2(P_xP_y)(MN)$	$2(P_xP_y)(MN)$	0	0
Weigthed autocorrelation	$3\cdot 2(w_x\cdot w_y)(P_xP_y)$	$3(P_xP_y)$	0	0
Minimum eigenvalue	$4(P_xP_y)$	$2(P_xP_y)$	P_xP_y	P_xP_y
Select corners	$3C^2$	$2C^2$	0	C^2
Shi Tomasi detector	$2(P_xP_y)(MN+3w_xw_y)$ $+4P_xP_y+3C^2$	$2(P_xP_y)(MN)$ $5(P_xP_y)+2C^2$	P_xP_y	$C^2+P_xP_y$

Table 9: Floating point operations per frame for Shi Tomasi corner detection

4.4.2.2 Analytic break down of Lucas Kanade optical flow

The Lucas Kanade pyramidal algorithm can be further decomposed in this four main subblocks:

Algorithm 8 pyramidal tracking Lucas Kanade Optical Flow

```
    Input: 2D $I, J$ images and $\boldsymbol{u}$ a point on image I
    Output $\boldsymbol{u}$ The 2D displacement of the pixel represented by W
1:  function PyramidalLucasKanade(I,J,$\boldsymbol{u}$)
2:      $\{I^L\}_{L=0,\dots,L_m}$ , $\{J^L\}_{L=0,\dots,L_m}$ ←ImagePyramidrepresentations($I, J$)
3:      $\boldsymbol{g}^{L_m} \leftarrow [0,0]$                    ▷ Initialization of pyramidal guess
4:      for $L = L_m$ to $L_0$ do
5:          $\boldsymbol{u^L} \leftarrow \boldsymbol{u}/2^L$
6:          ${I_x}^L \leftarrow$ ImageDerivative($I^L$)
7:          ${I_y}^L \leftarrow$ ImageDerivative($I^L$)
8:          $G \leftarrow$ SpatialGradientMatrix(${I_x}^L, {I_y}^L$)
9:          $\boldsymbol{v^{(0)}} \leftarrow [0,0]$                ▷ Initialization of iterative L-K
10:         while $|\boldsymbol{\eta}_\Delta| < \epsilon$ or $k \le K$ do
11:             $dI_k \leftarrow I^L - J^L(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1})$
12:             $\boldsymbol{b_k} \leftarrow$ ComputeErrorVector($dI_k, I_x, I_y$)
13:             $\boldsymbol{\eta}_\Delta^{(k)} \leftarrow G^{-1}\cdot \boldsymbol{b_k}$        ▷ Lukas Kanade Optical Flow
14:             $\boldsymbol{v}^{(k)} \leftarrow \boldsymbol{v}^{(k-1)} + \boldsymbol{\eta}_\Delta^{(k)}$        ▷ Guess for next iteration
15:             $k \leftarrow k+1$
16:         $\boldsymbol{d^L} \leftarrow \boldsymbol{v^{(K)}}$                ▷ Final optical flow at level L
17:         $\boldsymbol{g^{L-1}} \leftarrow 2(\boldsymbol{g^L} + \boldsymbol{d^L})$   ▷ Update the guess for next level $[g_x^{L-1}, g_y^{L-1}]$
18:     $\boldsymbol{d} \leftarrow \boldsymbol{d}^{L_0} + \boldsymbol{g}^{L_0}$
19:     $\boldsymbol{v} \leftarrow \boldsymbol{u} + \boldsymbol{d}$
20:     return $\boldsymbol{v}$              ▷ corresponding point u is at location v on image J
```

In the iterative algorithm, is computed executing the Lucas Kanade optical flow in different levels of an image pyramid. An image pyramid is a collection of images that are successively downsampled (or upsampled by interpolation) the specified number of levels. The details of the algorithm implement follows the approach of *Bouguet* [8]

Algorithm 9 Image pyramid representation

```
    Input:Image $I$ and $L_m$ is the height of the pyramid
    Output Summed Second Moment Matrix $M$
 1: function ImagePyramid($I, L_m$)
 2:     for $L = 0$ to $m$ do
 3:         for $x = 0$ to $n_x^{L-1} - 1$ do
 4:             for $y = 0$ to $n_x^{L-1} - 1$ do
 5:                 $I^L[x,y] \leftarrow \frac{1}{4} \cdot I^{L-1}[2x,2y]$
 6:                 $I^L[x,y] \leftarrow \frac{1}{8}\left(I^{L-1}[2x-1,2y]+I^{L-1}[2x+1,2y]+I^{L-1}[2x,2y-1]+I^{L-1}[2x,2y+1]\right)$
 7:                 $I^L[x,y] + \leftarrow \frac{1}{16}\left(I^{L-1}[2x-1,2y-1]+I^{L-1}[2x+1,2y+1]+I^{L-1}[2x-1,2y-1]+I^{L-1}[2x+1,2y+1]\right)$
 8:         $n_x^L \leq \frac{n_x^{L-1}+1}{2}$
 9:         $n_y^L \leq \frac{n_y^{L-1}+1}{2}$
10:     return $I_L$
```

In our implementation the zero level image is the original frame $I^0 = I$. The image width and height at that level are defined as $n_x^0 = P_x$ and $n_y^0 = P_y$. The pyramid representation is then built in downsampling the image recursively. The floating point operations are summarized below, being m the total number of levels of the Pyramid.

operation	add	mult	div	sqrt	comparisons
Image calculation	8	0	3	0	0
Index calculations	1	0	1	0	0
Image pyramid	$8m \cdot \frac{n^m}{2m} + 2m$	0	$3m \cdot \frac{n^m}{2m} + 2m$	0	0

Table 10: FLOP Image pyramid computation

where m are the levels of the pyramid and n the pixels

The spatial gradient matrix equation is defined analogously to the Shi Tomasi corner detector weighted image over the Image derivates

$$\boldsymbol{G} = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} I_x^2[x,y] & I_x[x,y]I_y[x,y] \\ I_x[x,y]I_y[x,y] & I_y^2[x,y] \end{bmatrix} \tag{17}$$

The image derivatives I_x and I_y may be computed directly from the first image in the neighbourhood of the the $(2x+1)(2y+1)$ of the input corner.

The next step and the first one from the iterative algorithm is to compute the image difference between the current frame and the displaced next frame.

$$\delta I_k[x,y] = I^L[x,y] - J^L(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1}) \tag{18}$$

operation	add	mult	div	sqrt	comparisons
Image derivatives	$3(P_xP_y)(MN)$	$2(P_xP_Y)(MN)$	0	0	0
Spatial gradient	$3\cdot(2w_x+1)(2w_y+1)$	$3\cdot(2w_x+1)(2w_y+1)$	0	0	0

Table 11: FLOP spatial gradient matrix

where g^L is the guess at the lowest level of the pyramid and $\boldsymbol{v}$ the new iterative displacement guess derived from residual optical flow. This operations just is composed by a subtractions of the transposed image so the operations are proportional to the pixels of the images.

operation	add	mult	div	sqrt	comparisons
Image difference	$n_x^L n_y^L$	0	0	0	0

Table 12: FLOP Spatial gradient matrix

where n^L just correspond to the number of pixels in the neighbourhood of the corner so $n_x \equiv 2w_x + 1$. The next step is to find the so-called image mismatch error $\boldsymbol{b}_k$.

$$\boldsymbol{b}_k = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} \delta I_k[x,y] I_x[x,y] \\ \delta I_k[x,y] I_y[x,y] \end{bmatrix} \tag{19}$$

This error vector captures the residual difference between the image patches after translation. $\boldsymbol{b}_k$ is defined as the weighted sum between the image difference and the image derivates corrispondingly compute with the following pseudocode

Algorithm 10 Computation of Mismatch Image vector

```
    Input:Image difference $\delta I_k$, Spatial image derivatives $I_x, I_y$ and window sizes
    OutputError mismatch vector $b$
1: function ComputeErrorVector($dI_k, I_x, I_y, w_x, w_y$)
2:     $b_x \leftarrow 0$
3:     $b_y \leftarrow 0$
4:     for $i = -w_x$ to $w_x$ do
5:         for $j = -w_y$ to $w_y$ do
6:             $b_x \leftarrow b_x + \delta I_k[p_x+i, p_y+j] \cdot I_x[p_x+i, p_y+j]$
7:             $b_y \leftarrow b_y + \delta I_k[p_x+i, p_y+j] \cdot I_y[p_x+i, p_y+j]$
8:     return $\boldsymbol{b} = [b_x, b_y]$
```

The total amount of floating point operations are summarized in Table 13

operation	add	mult	div	sqrt	comparisons
Error vector	$2(2w_x+1)(2w_y+1)$	$2(2w_x+1)(2w_y+1)$	0	0	0

Table 13: FLOP error vector

Finally the next step is to approximate the solution to the optical flow equation which has the form:

$$\boldsymbol{\eta}_{\Delta}^{(k)} \leftarrow G^{-1} \cdot \boldsymbol{b_k} \tag{20}$$

For solving this equation as **G** is a full-rank 2×2 matrix,then we can compute the inverse directly ($|G| = g_{11}g_{22} - g_{12}g_{21} \neq 0$)

$$\boldsymbol{G} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} \quad \rightarrow \quad \boldsymbol{G}^{-1} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}^{-1} = \frac{1}{|\boldsymbol{G}|} \begin{bmatrix} g_{22} & -g_{12} \\ -g_{21} & g_{11} \end{bmatrix}$$

operation	add	mult	div	sqrt	comparisons
Matrix Inverse	2	2	4	0	0
Vector Multiplication	2	4	0	0	0

Table 14: FLOP optical flow equation solving

The rest of the steps of the algorithm are just the updating of parameters that can be all englobed as a vector addition where for each update the floating point operation are indicated as vector operations in the summary Table15.

operation	add	mult
Image pyramid	$9 \cdot \frac{P_x P_y^m}{2} + 2m$	$3 \cdot \frac{P_x P_y^m}{2} + 2m$
Image derivative	$(2w_x + 1)(2w_y + 1)$	0
Spatial gradient	$3 \cdot (2w_x + 1)(2w_y + 1)$	$3 \cdot (2w_x + 1)(2w_y + 1)$
Image difference	$(2w_x + 1)(2w_y + 1)$	0
Error vector	$2 \cdot (2w_x + 1)(2w_y + 1)$	$2 \cdot (2w_x + 1)(2w_y + 1)$
Optical flow	$4CmK_m$	$6CmK_m$
vectors operations	$2CmK_m + Cm + 2$	0
Pyramidal LK	$9 \cdot \frac{P_x P_y^m}{2} + 2m + 6CmK_m + Cm + 2+$ $(2w_x + 1)(2w_y + 1)mC(4 + 3K_m)$	$3 \cdot \frac{P_x P_y^m}{2} + 2m + 6CmK_m$ $(2w_x + 1)(2w_y + 1)mc(3 + 2K_m)$

operation	div	sqrt	comp
Image pyramid	0	0	0
Image derivative	$(2w_x + 1)(2w_y + 1)$	0	0
Spatial gradient	0	0	0
Image difference	0	0	0
Error vector	0	0	0
Optical flow	$4CmK_m$	0	0
Vectors operations	cm	0	0
Pyramidal LK	$c \cdot m(2w_x + 1)(2w_y + 1)$ $+4C \cdot m \cdot K_m + c \cdot m$	0	0

Table 15: FLOP Lucas Kanade optical flow

where $P_x \cdot P_y$ stands for the total number of pixel of the image , w_x, w_y indicate the window width and height, m indicates the level of the pyramid size and C indicate the number of corners returned by Shi Tomasi algorithm and K_m indicate the number of iterations of the iterative optical flow.

4.4.2.3 Analytic Break Down of CNN

The Darknet-19 structure in YOLO9000 has originally 26 layers and consists of convolutional and maxpool-layer [9]. However in our work, in classification uses only the first 16 layers. The FLOP for the convolutional layers are calculated according to the Eq 21.

$$GFLOP = (2.0 \cdot filters \cdot size \cdot size \cdot channels \cdot output_{height} \cdot output_{width})/10^9 \tag{21}$$

Type	Filters	Size/ Stride	Output	FLOP
Convolutional	32	3 x 3	224 x 224	28901376
Maxpool		2 x 2/2	112 x 112	54656
Convolutional	64	3 x 3	112 x 112	14450688
Maxpool		2 x 2/2	56 x 56	12544
Convolutional	128	3 x 3	56 x 56	7225344
Convolutional	64	1 x 1	56 x 56	401408
Convolutional	128	3 x 3	56 x 56	7225344
Maxpool		2 x 2/2	28 x 28	3136
Convolutional	256	3 x 3	28 x 28	3612672
Convolutional	128	1 x 1	28 x 28	200704
Convolutional	256	3 x 3	28 x 28	3612672
Maxpool		2 x 2/2	14 x 14	784
Convolutional	512	3 x 3	14 x 14	1806336
Convolutional	256	1 x 1	14 x 14	1000352
Convolutional	512	3 x 3	14 x 14	1806336
Convolutional	256	1 x 1	14 x 14	1000352
Softmax	-	-	-	-
Total				71314704

Table 16: Darknet CNN FLOPS operation

The total number of Gigaflops performed by the CNN per frame is 0.0713 GFLOPS.

$$t_{exec} = \frac{0.0713 \quad \text{[GFLOP]}}{1.47 \ \text{[GFLOPS]}} = 0.0485s$$

Therefore the total energy consumed per frame by the computation of the CNN is the multiplication of the execution time by the power:

$$E_{CNN}\ [\text{Wh}] = P_{active}\ [\text{W}] \cdot t_{exec}\ [\text{h}] = 3.4\ [\text{W}] \cdot 13.45 \cdot 10^{-6}\ [\text{h}] \approx 45.8 \cdot 10^{-6}\ \text{Wh} \approx 45.8\ \mu\text{Wh}$$

5 Evaluation Methods

In order to validate the energy saving potential of the adaptive sampling, we evaluated the estimation approach with the labels predicted by the CNN and the ground truth manual annotated labels.

The evaluation performance metrics was obtained using an overlapping window w, in order to be adapted the spotting event characteristics.

To score the performance a similarity measure that quantifies the overlap between two windows was used. The first window w is the window region detected by the object detector labels and second is defined the by the ground truth labels.

A true positive (TP) is defined as an overlapping between the two windows is above a threshold of 0.5. Analogously, the False Negative (FN) was the number of retrieved objects where the ground truth windows region was overlapping intersection was not bigger than the threshold i.e. the objects that a ideal object detector would have classified as positive samples.

The detection results were evaluated using the recall(R), precision(P) and F1 score (F1) metrics. Precision it is commonly defined as the fraction of correct classified samples (TP) among all positive samples.

$$P = \frac{TP}{TP + FP} \tag{22}$$

Recall is the fraction of the correct classified samples over the samples classified as positive samples.

$$R = \frac{TP}{TP + FN} \tag{23}$$

A further weighted average derived form recall and precision was used, F1 score or F-measure (F1). This measure penalizes imbalanced precision and recall scores.

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{24}$$

The scores were tested in a 10-folds and after averaged. Firstly, average precision and average recall was computed to summarize the precision for each day.

$$AP = \frac{1}{K}\sum_{k=1}^{K} P_k \qquad AR = \frac{1}{K}\sum_{k=1}^{K} R_k \tag{25}$$

Eventually, $\overline{AP}$ was computed as weighted average of the APs from all days. The final mean measures were taken averaged the 5 day records of our dataset. The analogous procedure was followed for average recall. Finally, the F1 score was also derived.

6 Results

In this section, the results of the spotting event performance and energy consumption reduction due to the adaptive sampling strategy are shown.

6.1 Spotting Performance

The results of the spotting event performance are presented in the following figures. The average measure metrics within the 5 days is represented by the black curve. In these plots the energy consumption measure was used to estimate the energy consumption percentage in degradation of the performance

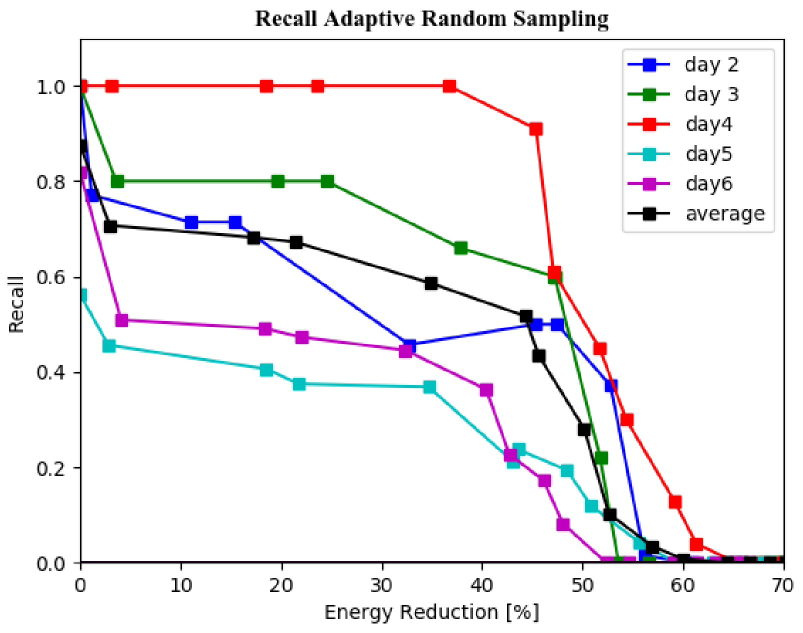

Figure 9: Recall and energy saving trade-off curve

The results show that the higher the compression factor, the lower the precision is. The performance decay it is due to the compression factor makes implies less frames are sampled and therefore detected.

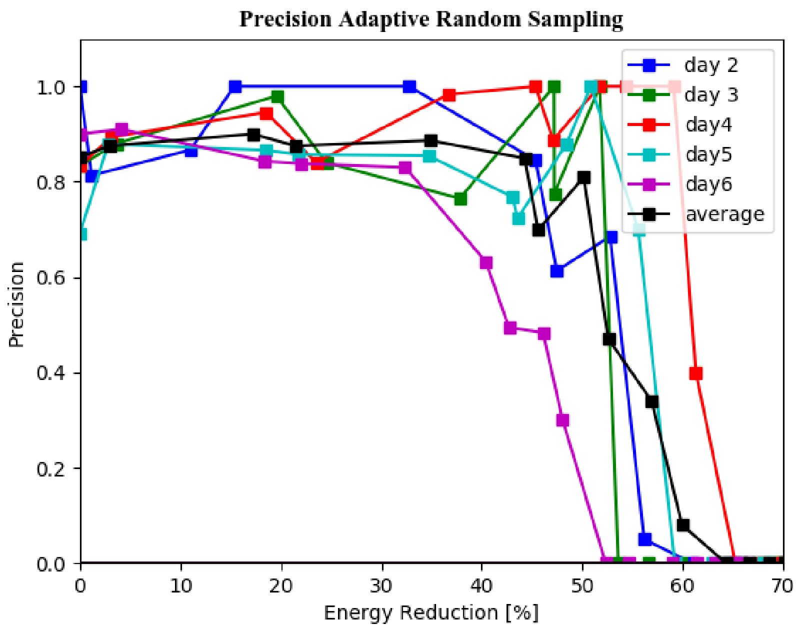

Figure 10: Precision and energy saving trade-off curve

The performance based on the energy consumption trade off can be further seen in the curve that ı

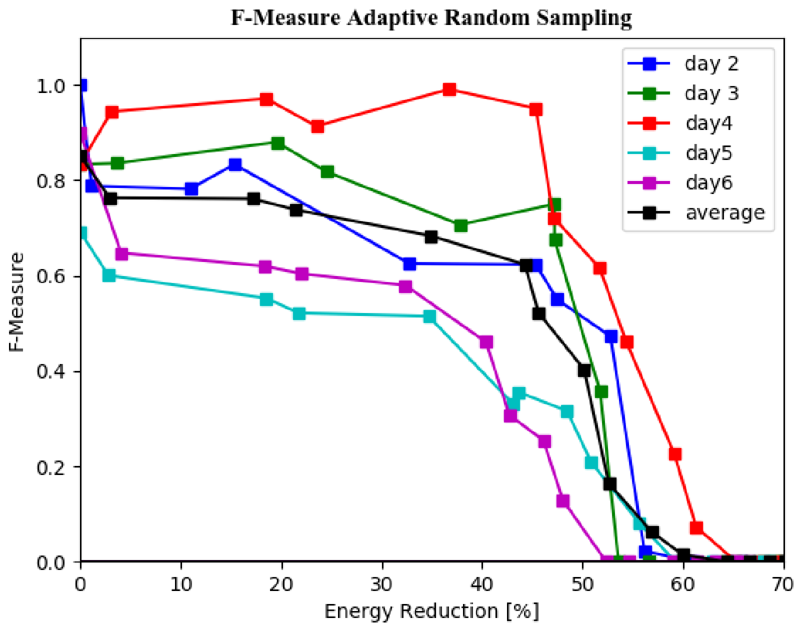

Figure 11: F1 score and energy saving trade-off curve

6.2 Energy Consumption

The energy consumption was also calculated in Wh for every day and every parameter. The mean energy consumption can be found that the consumption is still is suitable for an egocentric video portable set up.

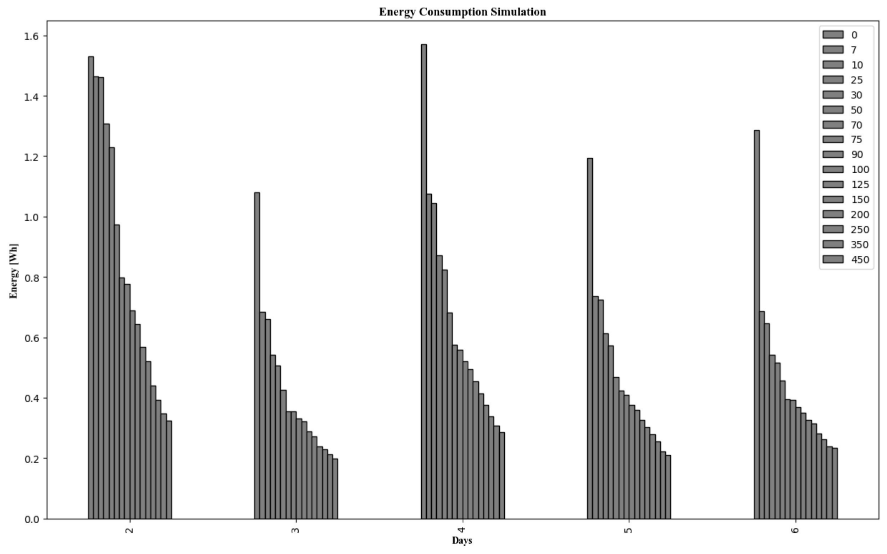

Figure 12: Energy consumption simulation of the framework for different parameters of the linear response model

7 Discussion

Detection performance results of the proposed dietary spotting event in free living conditions are an average of 44% of energy reduction precision, $P = 0.59$, recall $R = 0.85$, F1-score $F1 = 0.62$. The F1 score gets a degradation of a 27 % with regards to the full sampling rate F1 score which enable an energy saving up to 45% of the energy consumption.

The curve results sometimes present a higher metric value than lower compression factors. This is due to the characterization of the performance metrics with an overlapping window. The difference performance within days it is due to the variability in activities. Every day recording contains video content of distinct lighting conditions and variable number of objects and events duration. These all factors explain the variability in the precision and recall curves.

Secondly, we derived an energy model simulation for the egocentric video activity detection pipeline incorporating a context measure. From this result can be seen that it is feasible to implement the proposed architecture in a wearable device computing device. The context measure overhead is compensated by the energy reduction and performance trade-off.

Future work can expand this idea in several interesting directions, such as trying to reduce the image resolution in order to reduce the computational overload. Further research is required to evaluate other hardware configurations and optimize the hardware implementation of the context measure. In addition, further characterization of variables that may affect the energy consumption could be studied.

We must recall that we can achieve a 60 % of the Sampling reduction still maintaining the F1 score in the order of 0.6. Also there is high variance in each day and that is due to the stochasticity of the behavioral pasterns of the camera wearer.

8 Conclusion

In this work, A new adaptive sampling strategy based on a motion context measure for egocentric video was presented. The new method enables to save energy keeping a promising performance rate.

An energy model to estimate the amount of power consumption of the video activity recgonition elements was derived. We validate aour adaptive sampling strategy in free-living egocentric video by simulating the the camera and microcontroler.

Our approach enables energy saving and therefore the longer run time or reduce battery size in egocentric video set up. The sampling strategy could be extended to other spotting events applications further than the dietary activity recognition. The obtained results show that our adaptive sampling strategy is a great trade off between energy detecting approach and performance.

References

[1] G. Schiboni and O. Amft, “A Privacy-Preserving Wearable Camera Setup for Dietary Event Spotting in Free-Living,” in *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom) Workshops*, pp. 872–877, 2018.

[2] C. Salim, A. Makhoul, R. Darazi, and R. Couturier, “Combining frame rate adaptation and similarity detection for video sensor nodes in wireless multimedia sensor networks,” 09 2016.

[3] R. Possas, S. P. Caceres, and F. Ramos, “Egocentric Activity Recognition on a Budget,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 5967–5976, 2018.

[4] R. Likamwa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl

[5] J. Shi and C. Tomasi, “Good features to track,” pp. 593–600, 1994.

[6] C. Harris and M. Stephens, “A combined corner and edge detector,” in *In Proc. of Fourth Alvey Vision Conference*, pp. 147–151, 1988.

[7] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision (ijcai),” vol. 81, 04 1981.

[8] J.-y. Bouguet, V. Tarasenko, B. D. Lucas, and T. Kanade, “Pyramidal implementation of the lucas kanade feature tracker,” *Imaging*, vol. 130, no. x, pp. 1–9, 1981.

[9] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger,” *CoRR*, vol. abs/1612.08242, 2016.

List of Figures

List of Tables