Activity Recognition from Single-Pixel Cameras

Siddhant Sharma, Neladri Bose

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

December 11, 2015

Technical report No. ECE-2015-07

Summary

With activity recognition being a hot topic in the field of video processing, extensive research has been carried out to recognize indoor activities using very low temporal and spatial resolution cameras. Reliable recognition using low data would facilitate a new platform for smart rooms that can record and track user activities without compromising their privacy. In this project we aim to recognize four activities inside a room using a set of six single-pixel cameras and to build a prototype for a smart-room environment. We use a single-pixel camera testbed at the Smart Lighting Undergraduate Research Project (**SLURP**) Lab along with MATLAB tools to develop an effective activity detection and activity recognition algorithm for the system. Using real subjects to record activities in the room, we train the software to learn individual camera patterns for each activity and use these patterns to recognize actions in a real-time environment. We found that for a given set of constraints, the system could recognize the trained actions with a reasonable overall efficiency of 70.625% and a maximum of 96.25% for one of the activities. Since our aim was to provide a system prototype, the conclusive results were based on simple signal processing techniques and could serve as a basis for further improvements.

This project was completed within EC720 graduate course entitled "Digital Video Processing" at Boston University in the fall of 2015.

Table of Contents

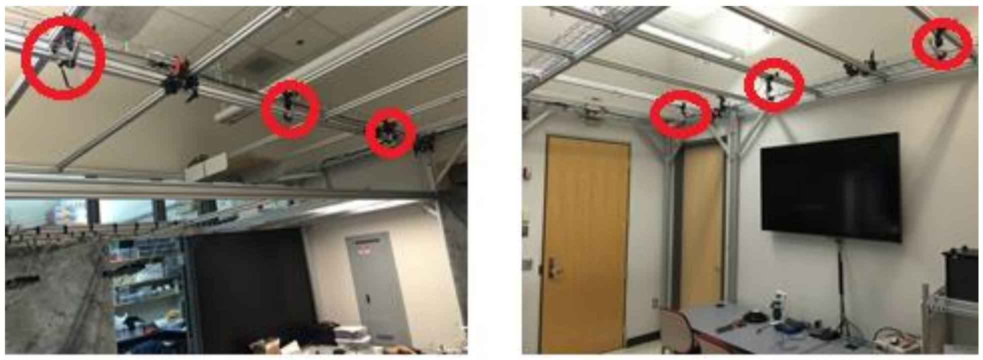
List of Figures

List of Tables

1 Introduction

Single-pixel cameras are poised to revolutionize activity recognition in the near future. As a single pixel does not deliver any detailed information about a scene, we can use it to our advantage to create motion tracking algorithms that preserve a user's privacy. These motion detection and activity recognition methods can be of valuable use in various applications like motion tracking, smart lighting for smart rooms, home automation, indoor position tracking, detecting unused parking spots, etc. In this project, we develop a robust s*ingle-user activity recognition algorithm*, using a set of single-pixel cameras in an experimental testbed. The testbed consists of six RGB **TCS3414** sensors that are mounted and positioned in a room as shown in Figs. 1-2. Since color information in the context of motion is redundant, we only use the luminance values of pixels, rounded off to the nearest integer as a true pixel value of the camera. The RGB to grayscale conversion is performed according to equation (1).

The network connection in the room is such that the sensors are multiplexed to transmit pixel values recorded over time to a Raspberry Pi which then transfers the data to a desktop computer in the room through a secure network connection. Using "Putty" to connect to Raspberry Pi, we obtain RGB pixel values on the desktop computer, where we interpret these values using the Robot Raconteur toolbox in MATLAB.



(a) *(b)*

Figure 1: Picture of the testbed setup where red circles identify the location of the single-pixel RGB sensors in the room

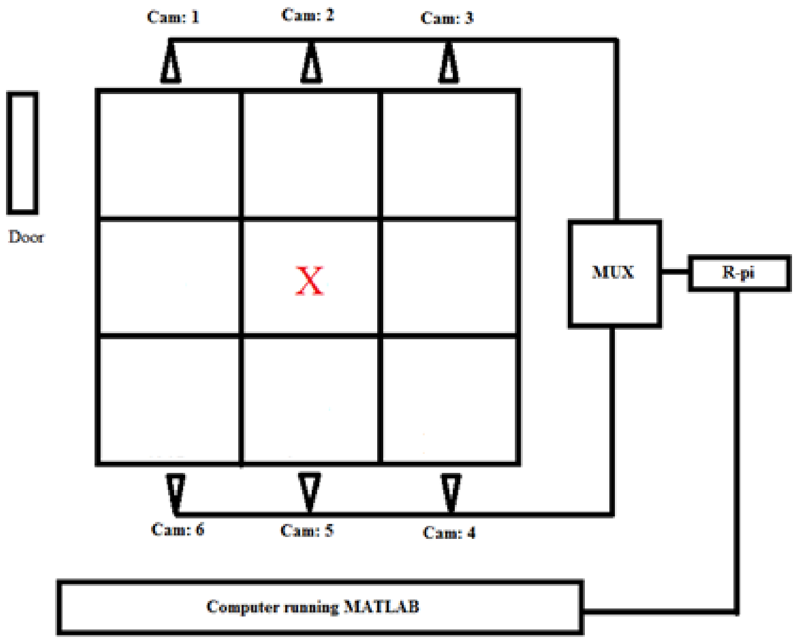


Figure 2: Block diagram of the room with 'X' marking the location of the user performing the gesture

The luminance is computed as follows:

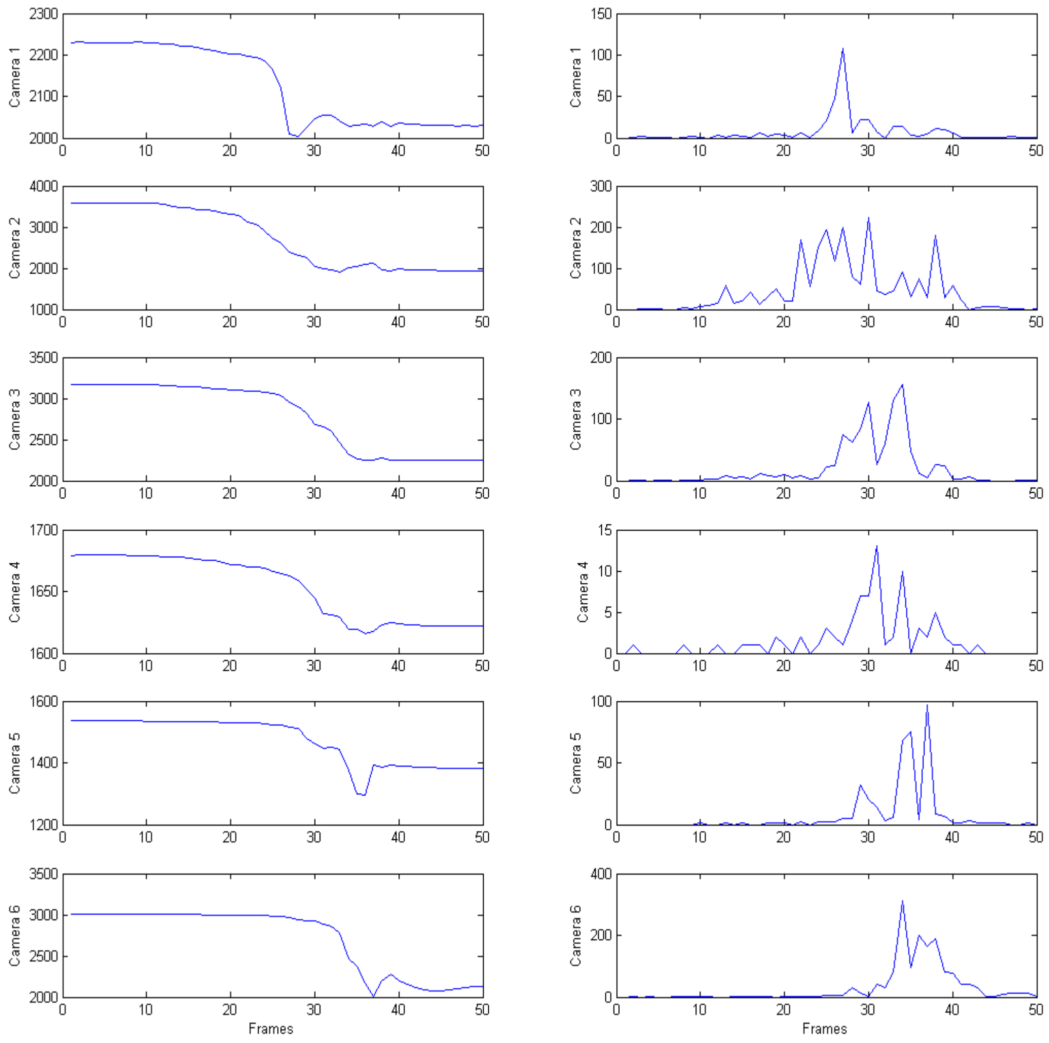
$$p[k] = p_R[k] * 0.2989 + p_G[k] * 0.587 + p_B\ [k] * 0.114 \qquad (1)$$

where $p[k]$ is the luminance value of frame k and $p_R[k], p_G[k], p_B[k]$ are pixels values of frame k corresponding to red, green and blue primary colors, respectively.

For each **TCS3414** color sensor, the luminance equivalent to a single RGB pixel from the sensor can be said to contain the majority of brightness information in the room. Since these luminance values are highly sensitive and range from 0 to 65000, we allocate 16 bits per pixel for storage. Also, from the conclusive results on temporal resolution and impact of camera count as mentioned in earlier work [1], we can assume that a well-positioned setup of 6 cameras with a temporal frequency of 6 frames per second would yield a reasonable Correct Classification Rate (CCR) when recognizing actions [1]. Thus, a frame containing six pixels would need a storage space of 96 bits. In comparison with low resolution cameras e.g., VGA with a resolution of 640 x 480, we achieve a compression ratio of 25600:1 per frame. Thus, using single-pixel cameras to process activities can help reduce the data storage, transmission bandwidth needed as well as the computational complexity.

2 Activity Detection

A principal drawback of detecting motion in a 1-D single-pixel camera sequence in comparison to high-resolution camera videos, is that the ground truth of motion cannot be visualized by looking at the activity data samples. With problems in single-pixel sensors, like signal fluctuations in changing lighting conditions and sensor noise, recognizing the onset and offset of an activity is a difficult task. To explain this aspect, an instance of an activity is plotted in Fig. 3a that indicates the pixel values captured by the six cameras for a time duration of 50 frames (approximately 8 seconds) when a random user enters the room and stands at position X, as indicated in Fig. 2.

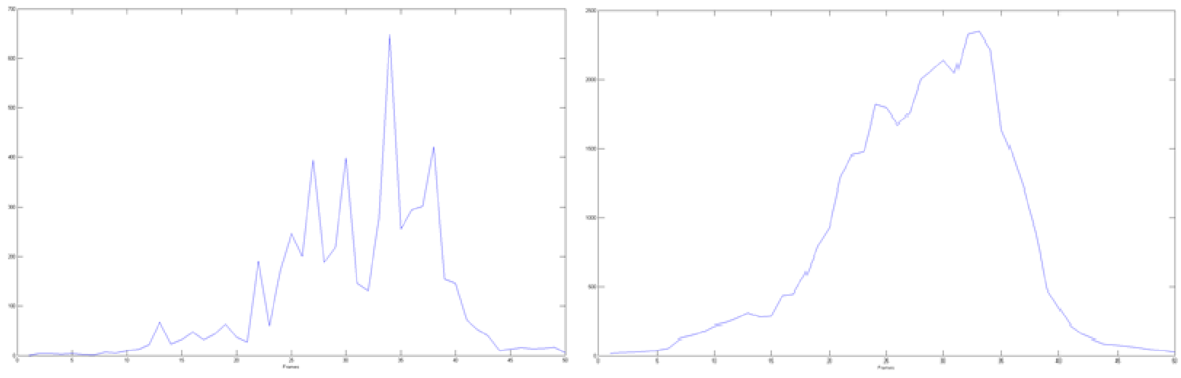


(a) *(b)*

Fig. 3 (a) Values p[k] captured by six cameras over 50 frames (k=1-50); (b) Absolute value of frame difference of the six cameras as a function of frame number k.

From the six sequences in Fig. 3a it is hard to pinpoint the exact frame that corresponds to the beginning and the end of motion. The fall in pixel values is only an indication that the user was occluding a bright room with a dark textured clothing. Thus, instead of matching a decision as to motion based on true pixel values, we check for a feature that tells us more about motion than the nature of lighting. Since temporal change is a measure of motion, analyzing the temporal derivative of the frames of the activity would give a better response. Fig. 3b is a plot of the absolute value of the temporal derivative, or the absolute value of frame differences for each camera. In comparison to Fig. 3a, Fig. 3b gives a better understanding of motion onset and offset as detected by each camera.

However, the frame corresponding to a significant rise in gradient values is different for every camera as the cameras differ in position and field of view. To reach a conclusive result, in Fig. 4a we plot the sum of absolute values of frame differences over all six cameras. Although fairly conclusive results about the beginning and end user activity can be drawn from Fig. 4a, the sharp changes in gradients over the course of the activity indicate that evaluating a decision as to motion over a single frame difference would be more susceptible to camera noise and is thus more likely to cause false detections due to measurement fluctuations. Thus, to obtain a smoother activity flow in Fig. 4b we plot α_k which is the summation of the sum of frame differences for each camera over 6 frames calculated using a sliding window approach as described in equation (2):



(a) *(b)*

Fig. 4 (a) Sum of the absolute values of frame differences over all six cameras for frame k; (b) Plot of α_k vs frame k

$$\alpha_k = \sum_{cam=1}^{6} \sum_{t=k}^{k+5} | p_{cam}[t+1] - p_{cam}[t] | \qquad (2)$$

where $p_{cam}[t]$ is the luminance value of frame t for a given camera. Thus α_k from equation (2) can be used as a measure for determining motion in the room. Since a noisy camera would introduce some fluctuation in pixel values even for static condition, a threshold **ά1** must be calculated to eliminate the interference of noise for decisions at the Start and End of activity. **ά1** can be calculated by estimating the static background condition for the first 6 frames. The value(s) of threshold **ά1** based on range of α_k for the first 6 frames are given in Table 1.

To further improve efficiency of the system, we introduce a threshold **ά2** that is helpful in categorizing residual motion. Residual motion may be defined as twitchy or shaky involuntary movement that is not intended to trigger an activity. The threshold for residual motion **ά2** is given by **ά2 = ά1** + β, where β is a measure of residual tolerance

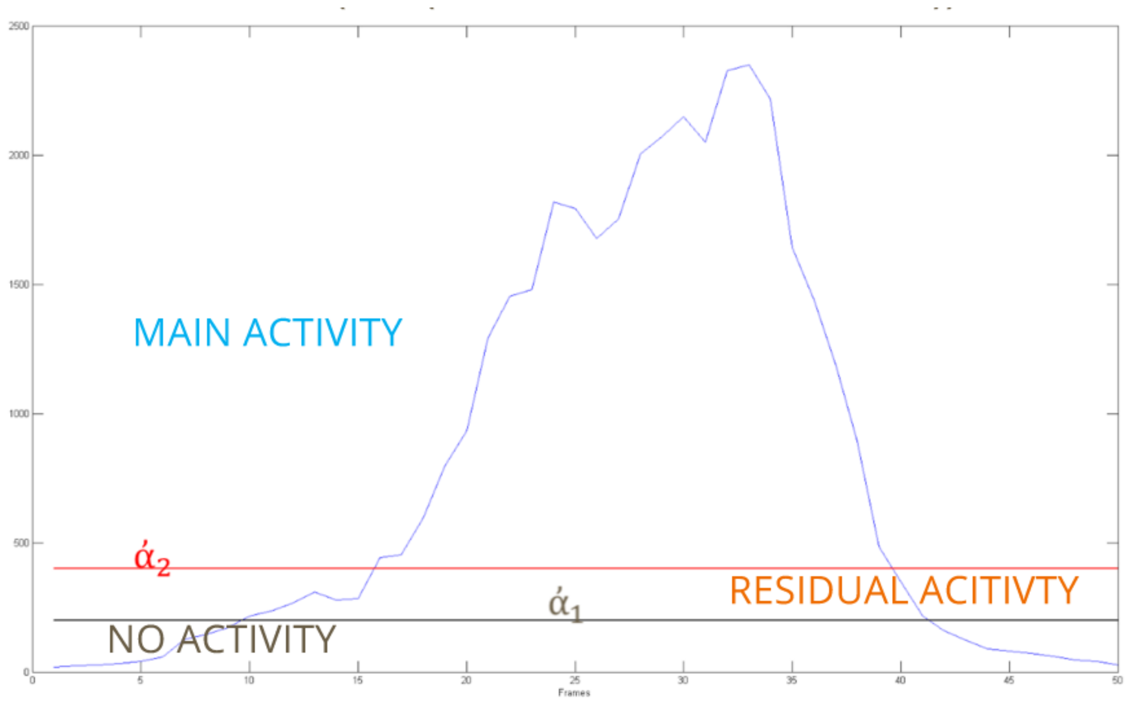
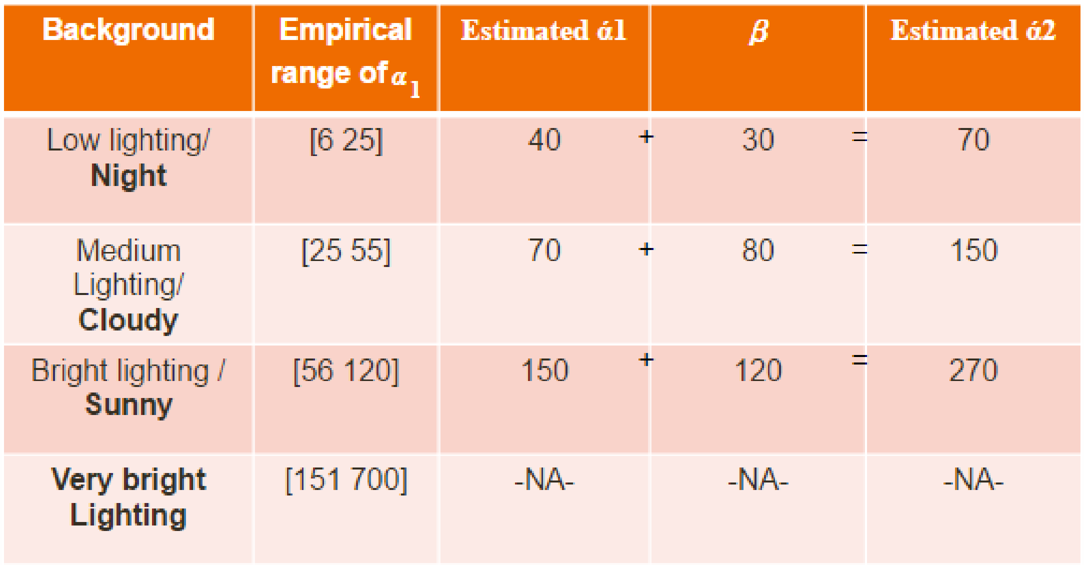


*Fig. 5: Pictorial representation of the effect of thresholds **ά1** and **ά2** on an activity instance from Fig. 3*

The interpretation of three possible states is as follows:

$$\alpha_k \geq \acute{\alpha}_2 \rightarrow Activity\ to\ be\ Recognized$$
$$\acute{\alpha}_1 \leq \alpha_k \leq \acute{\alpha}_2 \rightarrow Residual/Jittery\ Actions$$
$$\alpha_k \leq \acute{\alpha}_1 \rightarrow No\ Motion$$

The camera sensor TCS3414 was found to show a nonlinear behavior for fluctuations in different lighting conditions. The noise was more prominent in bright light than dim light. The values of **ά1**, **ά2** estimated over a range of α_1 (k=1 for the first 6 frames) to overcome the effects of the nonlinearity in cameras for different lighting conditions is given in Table 1. Since the fluctuations were out of reasonable bounds in very bright light, we advise to keep minimum lighting condition during activity detection.



Background	Empirical range of α_1	Estimated ά1	β	Estimated ά2
Low lighting/ **Night**	[6 25]	40 +	30 =	70
Medium Lighting/ **Cloudy**	[25 55]	70 +	80 =	150
Bright lighting / **Sunny**	[56 120]	150 +	120 =	270
Very bright Lighting	[151 700]	-NA-	-NA-	-NA-

*Table 1: Estimated values of threshold **ά1** and **ά2** for different lighting conditions*

Note: The range of values of α_1 in the above background conditions was set by using the minimum and maximum values of 10 observations of α_1 during different times of the day. **ά1** was estimated by adding κ to the maximum value of α_1. κ =15 for low and medium lighting and κ =30 for bright lighting. β was estimated by selecting the least integer value added to **ά1**, that triggered an activity. A more detailed approach can be made to set standard values for thresholds.

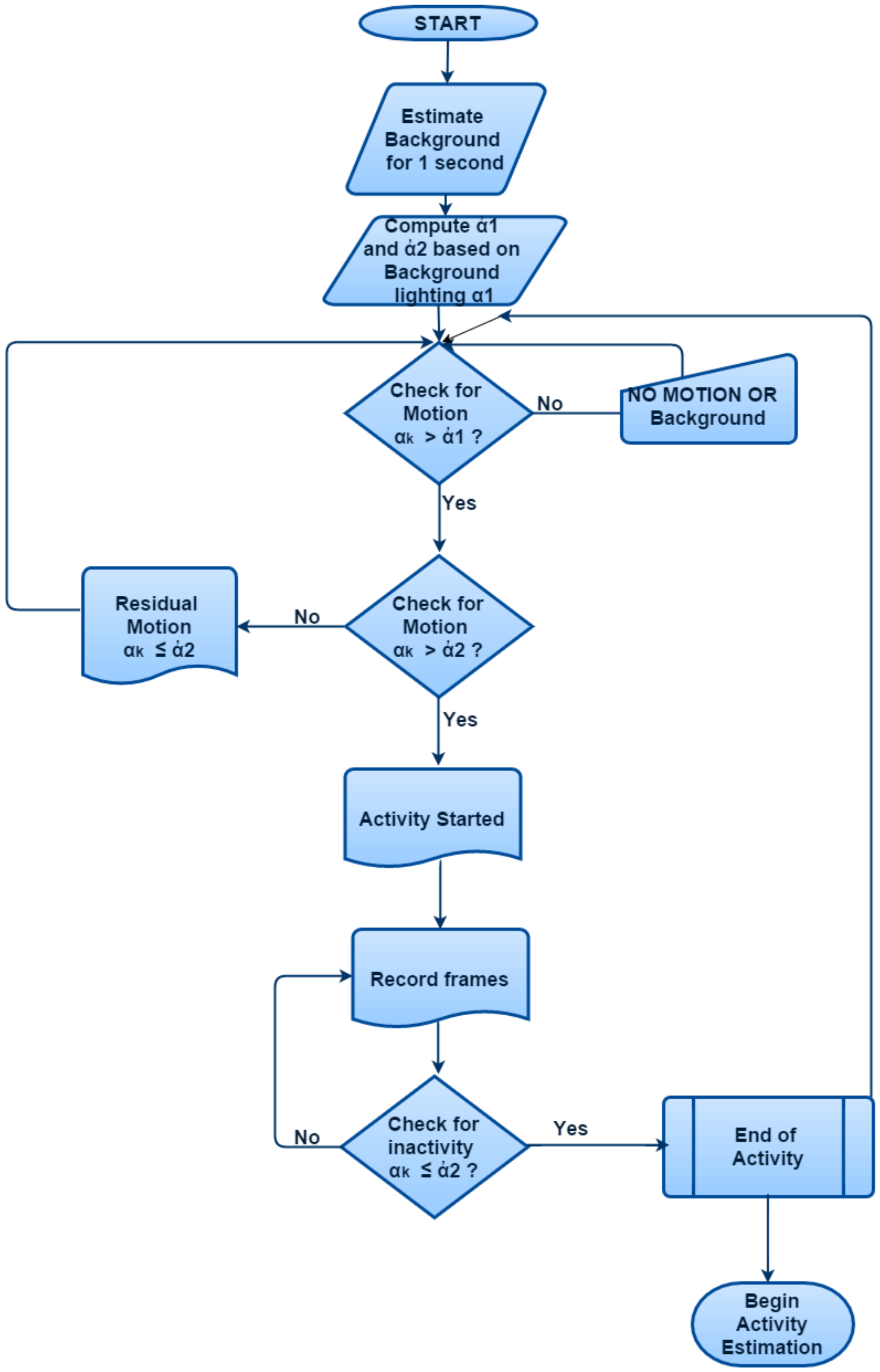

START
Estimate Background for 1 second
Compute ά1 and ά2 based on Background lighting α1
Check for Motion αk > ά1 ?
No
NO MOTION OR Background
Yes
Check for Motion αk > ά2 ?
No
Residual Motion αk ≤ ά2
Yes
Activity Started
Record frames
Check for inactivity αk ≤ ά2 ?
No
Yes
End of Activity
Begin Activity Estimation

Fig. 6 Flowchart for activity detection algorithm. For synchronizing activity detection at runtime, a single-tone beep sound is played every time the system detects the start and end of an activity.

3 Activity Recognition

The second section of our project deals with recognition of the detected activity. This is done by obtaining the closest match of the activity sequence to a set of training samples. A decision based on the combined decisions of the six cameras elicits a response to that particular gesture. To simplify calculation we trained our program to respond to four specific gestures: raising two hands in unison, sitting down on a chair, getting up from a chair, and writing (a motion similar to writing 3 capital A's). These gestures are also shown in Table 3, performed at predefined location X, from Fig. 3 facing camera 2. The activity recognition algorithm can be divided into four stages that are described next.

3.1 Interpolation

In order to get a decision for every camera, we individually analyze the 1-D time dependent sequence of true pixels for every camera. As the time duration of each activity may vary, we need to process the data over a standard frame length "L". For this purpose, we interpolate the detected true pixel activity sequence for every camera, independently to a frame length L using cubic spline interpolation as follows.

$$act_{cam}[i] = Cubic\ Spline\ Interpolator\{p_{cam}[k]\} \tag{3}$$

The range of *i* varies from 1 to L. An arbitrary value of L = 40 frames was used.

3.2 Mean-Variance Equalization

To remove the effects of global luminance and variation in the texture of clothing between users, we match the Mean-Variance Equalized (MVE) version of the interpolated activity sequence $act_{cam}[i]$ for each camera with the corresponding MVE and interpolated data from the training set. Equation (4) mathematically expresses the MVE process. Figs. 7-10 display the true-pixel value sequences as well as the interpolated and MVE processed sequences for activities 1, 2, 3, 4 for all six cameras. By performing MVE, we not only retain the pattern of the true pixel data, but also reduce the dynamic range of the sequences.

$$\widehat{act}_{cam}[i] = \frac{act_{cam}[i] - \mu_{cam}}{\sigma_{cam}^2} \tag{4}$$

where μ_{cam} is the mean and σ^2_{cam} is the variance of act_{cam}.

3.3 Nearest-Neighbor Classification

The next stage attempts to recognize an action for every camera by performing Nearest-Neighbor (NN) classification over the training set. The nearest neighbor *j* is the index of the training activity sequence that minimizes the distance metric $dist_{cam}[j]$ for a given camera. Equation (5) gives the formula for computing the distance:

$$dist_{cam}[j] = \sum_{i=1}^{40} |\widehat{act}^{j}_{cam}[i] - \widehat{act}_{cam}[i]\,| \tag{5}$$

where $\widehat{act}^{j}_{cam}$is the *i*th sample of the *j*th training sequence after interpolation and MVE, and the nearest neighbor is found as follows:

$$decision_{cam} = CLASS\left\{\underset{j}{argmin}\ \ dist_{cam}[j]\right\} \tag{6}$$

Our training data set consists of 8 users performing 10 trials of each of the 4 gestures, i.e., 80 activity sequences per gesture, creating a set of 320 sequences for all four activities.

Activity name	**Activity/Class number**	**Range of j**
Raising two hands in unison	Class 1	[1 80]
Sitting on a chair	Class 2	[81 160]
Getting up from a chair	Class 3	[161 240]
Writing	Class 4	[241 320]

Table 2: Details of the names of activities and their class numbers along with the range of j falling under a class

3.4 Fusion of Decisions

The last stage in the recognition process is the fusion of decisions from different cameras. We compute the final decision from a set of 6 decisions (1 decision per camera) by obtaining a unique majority. A unique majority is obtained when at least 3 of the 6 camera decisions favor an activity and the remaining decisions do not unanimously favor another activity.

If the system is able to obtain a unique majority, it returns the value of the class corresponding to the recognized gesture and informs the user by a voice feedback. For example, the voice feedback associated with activity 1 is “You are raising your hands”. If the system fails to obtain a decision by unique majority, the audio file “Activity not recognized” is played back. For example, three cameras supporting activity 2 and the other three supporting activity 4 does not form a unique majority and the system will fail to recognize the activity.

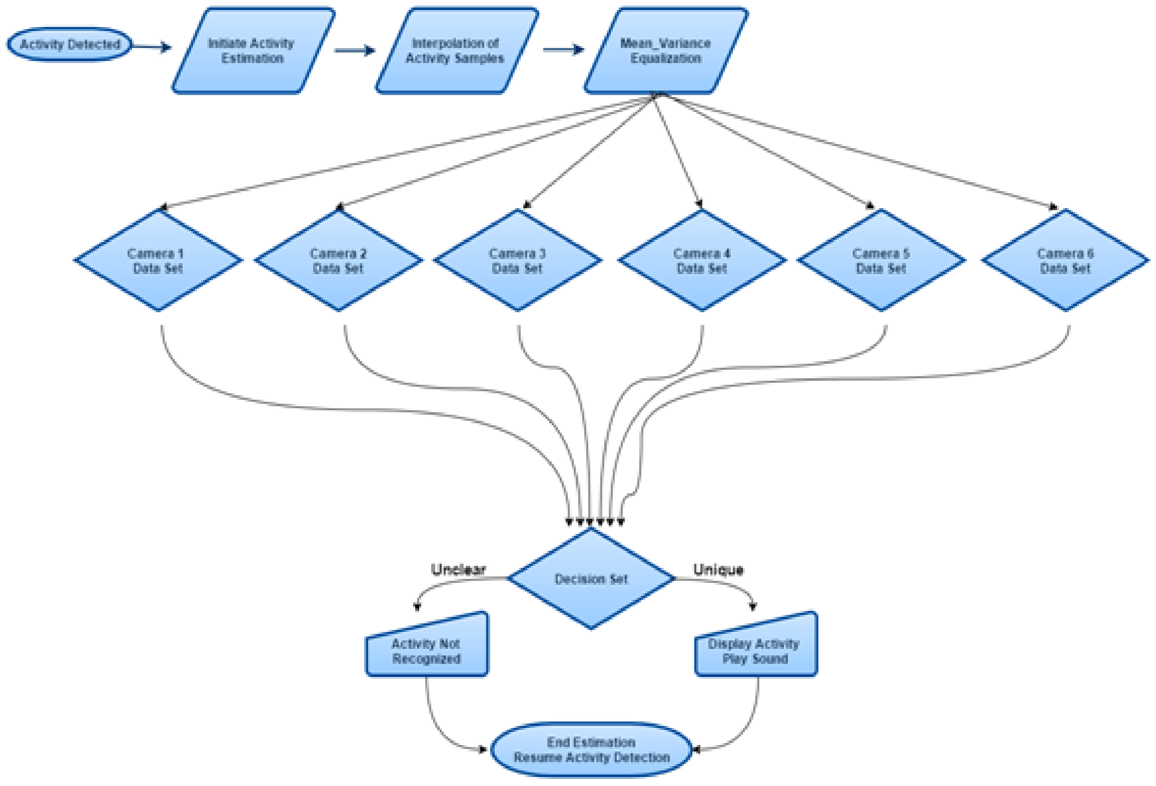
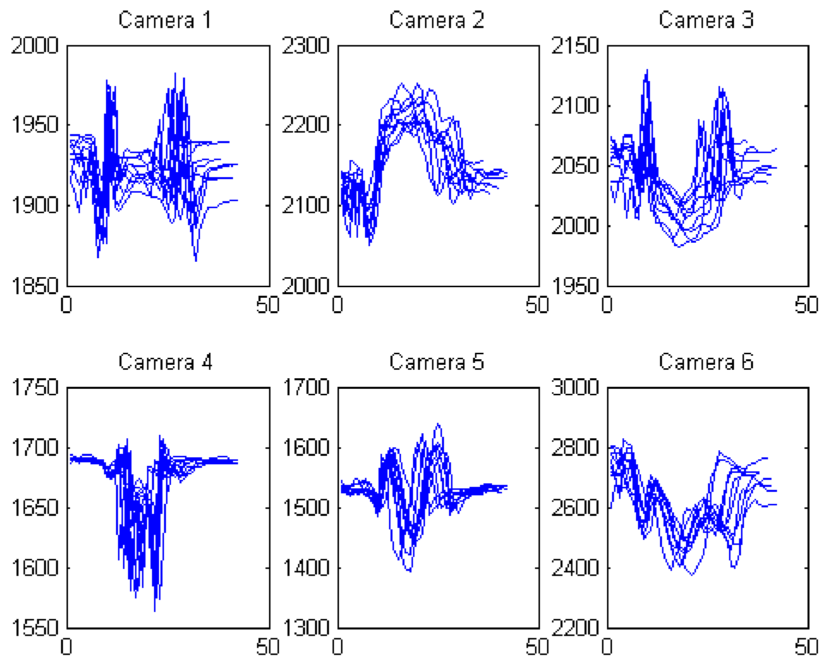


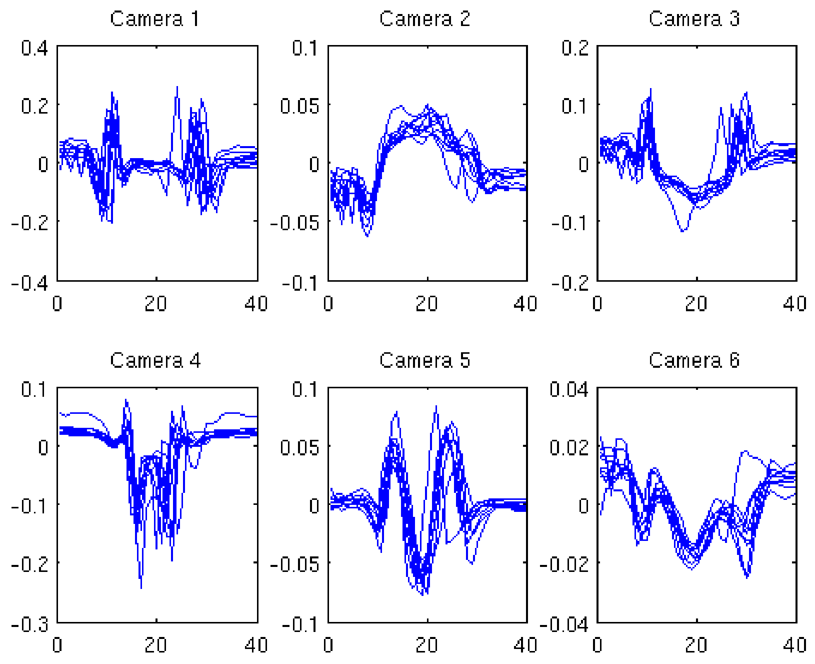
Fig. 7 Flowchart of our activity recognition algorithm

4 Experimental results

Plots for activities 1, 2, 3, and 4 are as shown in Figures 8, 9, 10, and 11 respectively.

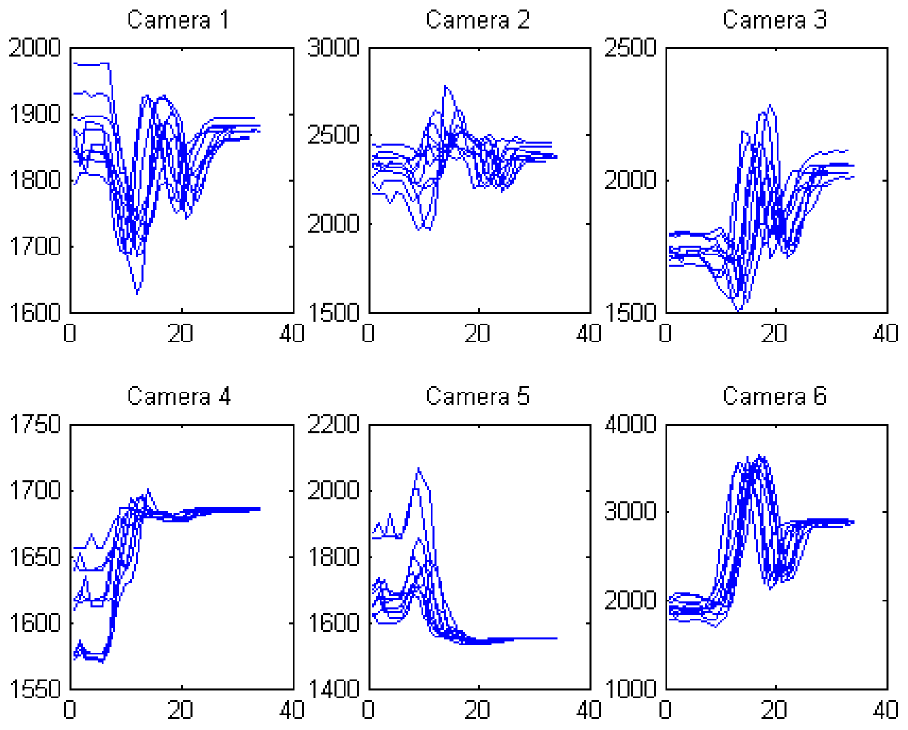


(a): Original Signal

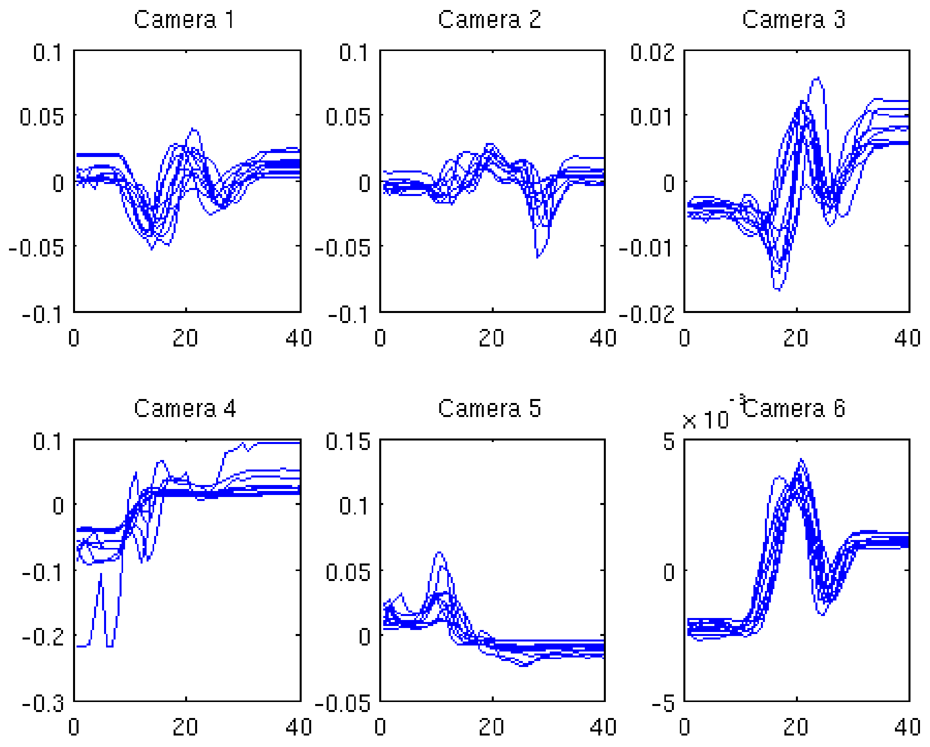


(b): Signal after interpolation and mean-variance equalization

Fig. 8: Ten signals captured when recording activity 1

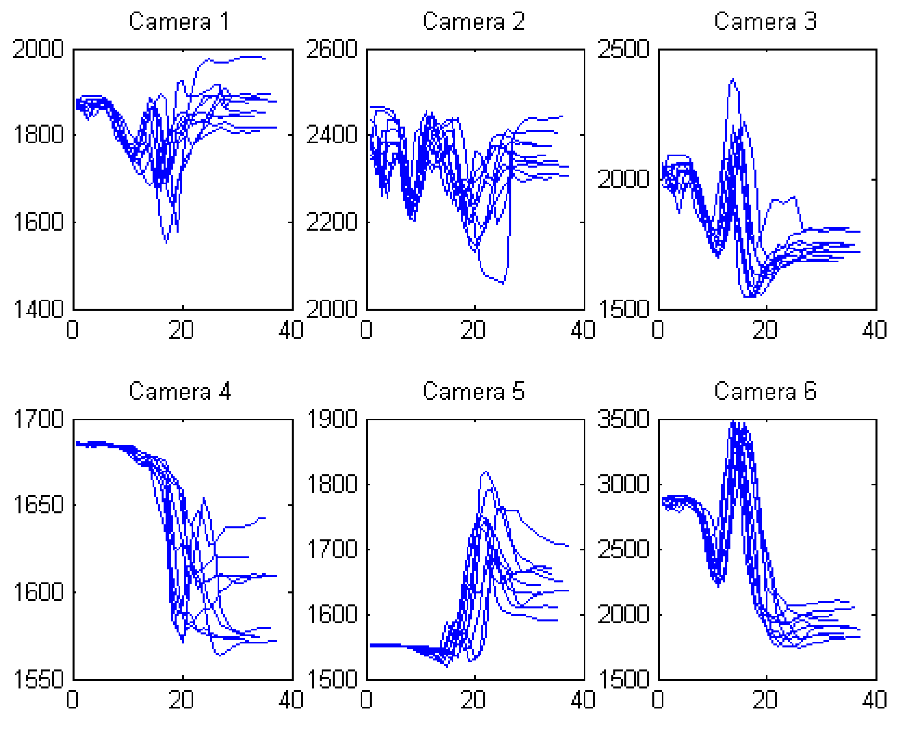


(a): Original Signal

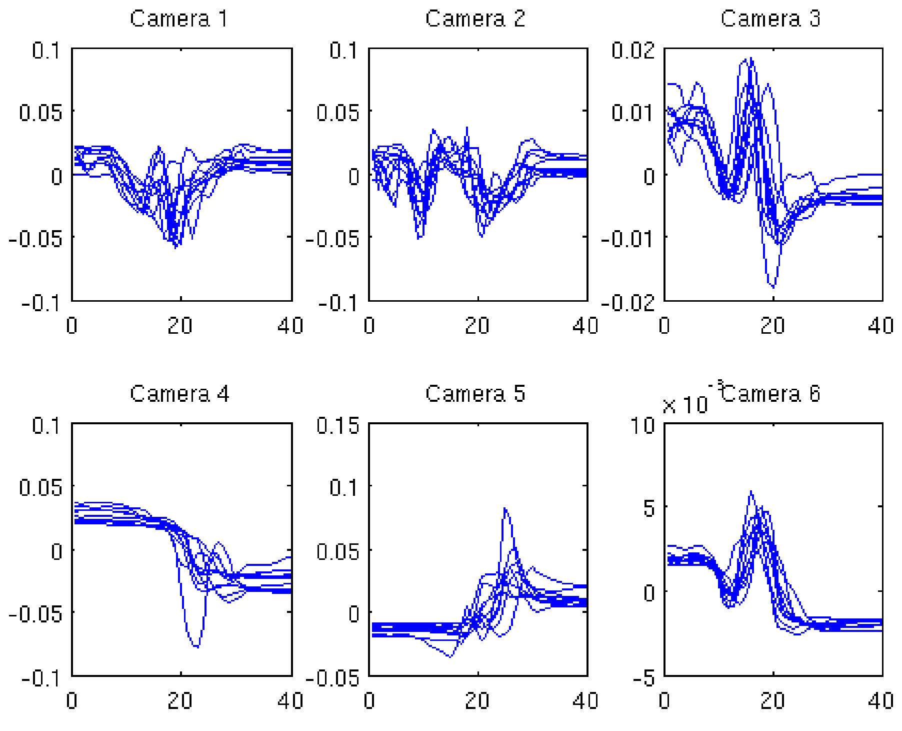


(b): Signal after interpolation and mean-variance equalization

Fig. 9: Ten signals captured when recording activity 2

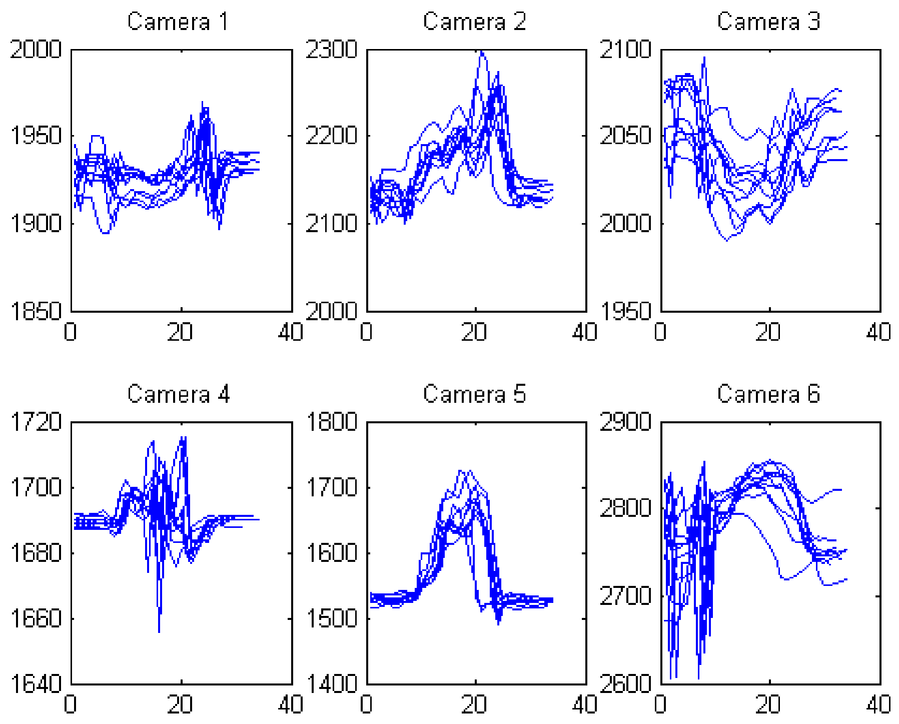


(a): Original Signal

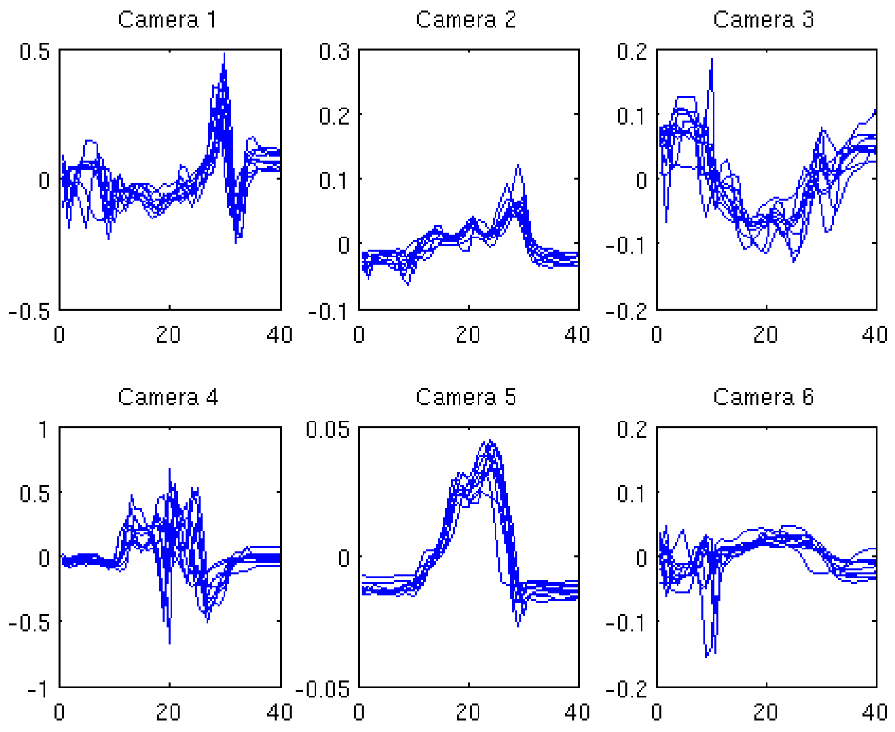


(b): Signal after interpolation and mean-variance equalization

Fig. 10: Ten signals captured when recording activity 3



(a): Original Signal



(b): Signal after interpolation and mean-variance equalization

Fig. 11: Ten signals captured when recording activity 4

4.1 Discussion of Results

a) Evaluation of Activity Detection Algorithm

Since establishing ground truth for the exact frame indicating onset and offset was a tedious task, we did not include statistics for the activity detection algorithm. However, during a live demonstration of the project, the system delay for the activity start and activity end was fair and user-acceptable. This delay can be further reduced by optimizing calculations of motion detection and setting adaptive measures for threshold values **ά1** and **ά2**. Adding **ά3** could further increase the robustness of the system.

b) Evaluation of Activity Recognition Algorithm

In order to derive fair and conclusive results for activity-recognition statistics, we performed leave-one-out cross-validation (LOOCV) technique on samples from the dataset used for training. All activity sequences from all users were tested for the predefined ground truth once, based on training data from other users and the remaining 9 sequences of the same user. The results of leave-one-out cross-validation are shown in Table 3.

Activity	**Success/CCR**	**No recognition**	**False Recognition**
Writing	51.25%	8.75%	40%
Sitting down	96.25%	0%	3.75%
Getting up	87.5%	2.5%	10%
Hand Raise	47.5%	13.75%	38.75%
Total	70.625%	6.25%	23.125%

Table 3: Statistics of activity recognition using leave-one-out cross-validation test

Considering that the system was trained on only a small number of users, the statistics of Correct Classification Rate derived from the leave-one-out cross-validation test were quite satisfactory. An overall success rate of 70.625% indicates that the system is capable of recognizing activities for a small dataset, and thus has scope for improvement. A high false recognition rate could indicate that we could improve our decision metric by fusing the decision metric and reaching a conclusive result instead of a fusion of decisions on every camera. This factor is open to further research.

To further improve the success rate, we could use more than one feature to obtain the distance metric and average a weighted sum of the distances to reach a decision on every camera. A good addition to this feature would be the interpolated and mean-variance equalized values of the temporal gradients of the activity. Another feature could be the third and fourth order moments of the true pixel values of the activity. An optimization and pattern recognition approach could lead to an equation for the fusion of the distance measures.

c) Evaluation of the Choice of Activities

Looking at patterns in Figs. 8-11 it can be concluded that activity 2 and activity 3 that correspond to sitting on a chair and standing up from the sitting position, respectively, form a set of orthogonal activities**.** Thus, it is safe to assume that a higher percentage on their success rate is due to their orthogonal characteristic. However, there are many possible variations for performing activity 1 and activity 4, and hence their low success rate is justified. For purposes of activity recognition irrespective of the user, activity 2 and activity 3 would prove to be a better set. Activity 1 and 4 would be of higher importance in an application of user authentication or identification. For example, since no two users have an identical signature, an action where the user signs his name in the air could serve as a good activity to identify the user. Despite the plausibility of the idea, a large amount of training would be required at the user's end to ensure a sufficiently-high correct classification rate.

4.3 System constraints and proposed improvements

Following is a list of constraints that we placed

1. Lighting conditions must not be too bright and must not change after the program is running.

   Improvement: Tuning the system to bright light and setting adaptive measure on **ά1** *and* **ά2** *in static conditions could solve the problem.*

2. Activity must be restricted to the user at location X

Improvement: Training the system over a single activity performed at different locations in the room and using numerical weights on each camera as a feature to recognize user location.

3. User must face camera 2 while performing an activity
   Improvement: Training the system over the same activity performed at different rotation angles at a specific position in the room and finding the optimum feature to maximize correlation between data patterns obtained at different angles.

4. Activity must be continuous, i.e. the user must not pause during an activity.
   Improvement: System can be designed to merge two or more activities in case of no match or low confidence match.

5 Conclusions and Possible Improvements

We designed and implemented an algorithm for detecting progressive and residual motion in real time, with reasonable delay under different lighting conditions using six single pixel cameras. To recognize an activity, we trained the system for four activities and used Nearest-Neighbor classification on the interpolated and mean-variance equalized values of true pixels at run time. By taking a majority vote on the decision at every camera we were able to adequately recognize each of the activities with an overall correct classification rate of 70.625% and a maximum of 96.25% for sitting down. These results show that single-pixel cameras can provide a viable solution for accurate gesture recognition, while maintaining user privacy. Thus, this experiment provides a foundation for further improvements on the method of activity recognition using single-pixel cameras that can exploit higher degrees of freedom through a more robust algorithm.

Appendix

Below is the Matlab source code developed for this project

Activity_tracker.m

```
%% Connection setup
%EC720
%Code developed by Siddhant Sharma & Neladri Bose
%This is the activity detection algorithm
%It estimates background for a couny of 12 and returns the lighting
%condition based on the average of the sum of gradients for 12 frames
%After 12 frames a user can enter the room and the system detects activity
%start and stop by a beep sound. The activity frames are recorded in ACT{}
%This can be used to get training data; comment line <>
% As a standard measure, Store the training frames as front sit stand and
% write (or choose your unique <activityname>)
% save file <username>_<activityname> i.e eg sam_front sam_write sam_sit
% Once you get all name_activity files , run Decision_CCR_metric




clc;clear all;
urls={'tcp://192.168.1.202:2335/{0}/ColorSensor'};
urllength=length(urls);
global connections
connections=cell(urllength,1);
for i=1:urllength
connections{i}=RobotRaconteur.Connect(urls{i});
end
disp('The connection is made');
%% Background Estimation
disp('Estimating Background till count of 12');
gradient_back=zeros(6,1);
for back=1:12; %taking the first 12 frames for background
for j=1:length(connections)

    readings=double(connections{j}.ReadSensors()); % Reads the data from the sensors, %produces
an 8(muxboards)x8(outputs)x6(reading values) array of values

end
back %counts upto 12 frames during runtime

   %6 sensors, R,G,B values for each
RGBs(1:3,1:3)=readings(1,1:3,1:3);   %cameras 1,2,3
RGBs(4,1:3)=readings(1,5,1:3); %camera #4 is made camera #5
RGBs(5,1:3)=readings(1,4,1:3); %camera #5 is made camera #4
RGBs(6,1:3)=readings(1,6,1:3);   %camera 6
   %converts to grayscale values
g_val(1:6) = 0.299*RGBs(1:6,1) + 0.578*RGBs(1:6,2) + 0.114*RGBs(1:6,3);
g_val = g_val';                %transposes matrix


   % stores values as integers
background_pixels(1:6,back) = double(int16(g_val(1:6)));
```

```
    %gradiant calculation for every frame starting frame 2
    if (back >= 2)
        gradient_back = background_pixels(:,back)-background_pixels(:,back-1);
    end

        %sequentially storing gradiants in an array
        background(:,back)=background_pixels(1:6,back);
        %summing gradiants on all cameras
        g(back)=sum(abs(gradient_back));
        %summing the sum of gradiants on all frames
        G=sum(g);


        %last frame on test
    if back==12
        background_sum=floor(G/2); %average of sum of gradiant for 6 frames(1 second)
     %setting 3 threshold values for activity detection based on background
        if background_sum<23          %night
            disp('Low Lighting Condition');
            thresh3= background_sum+18;
            thresh2=thresh3+30;
            thresh1=thresh2+50;
        elseif background_sum<55     %normal cloudy
             disp('Medium Lighting Condition');
            thresh3= background_sum+60;
            thresh2=thresh3+80;
            thresh1=thresh2+40;
        elseif background_sum<150   %normal sunny 66 82   71   69 32 84 52 53 64 25 104 23 64 68
113 73 98 72 51 50 26 28 44 25 36 54 67 119 106
             disp('High Lighting Condition');
            thresh3= background_sum+100;
            thresh2=thresh3+120;
            thresh1=thresh2+80;
%           elseif   background_sum<800                         %bright/sunny 213 136 672 592 569 340
404 794 162 307 328 440 417 437
%               disp('Very high lighting condition');
%               thresh3= background_sum+250;
%               thresh2=thresh3+300;
%               thresh1=thresh2+250;
        else
            disp('Ending program because its too noisy');
        end
    end
end
    clearvars g_val RGBs back G j; %clearing useless variables
%% Runtime
%Variables being initialized
i=1; %this acts as a switch for detecting each activity (i=0 at the end of an activity)
k=0; %stores the number of activities performed since the program was run
fps=6; % it should be 6.5 fps
cams=6;
ACT{1,10}={}; %stores activities in a cell

while (i==1)
i=0; k=k+1;
count = 1; %stores the frame count
window = 1; %keeps oscillating from 1 to 6, to get the gradiants over 1 sec
pixel_data= zeros(cams,fps); %stores camera pixels info
```

```
gradient_1=zeros(6,1); %gradiant per frame
 temp=0; %this increments after activity starts
 len=6;   %this increments after activity progresses
g=zeros(1,len);
buffer=zeros(cams,len); %stores the pixel values in a buffer for
a_rolloff=0; %this switch gets active when activity progression falls thresh2
%(but greater than thresh3)
%roll_off_buffer=zeros(6,3); %stores frames after apple=1 limit=3 frames
while(i==0)
for j=1:length(connections)

      readings=double(connections{j}.ReadSensors());
% Reads the data from the sensors,
%produces an 8(muxboards)x8(outputs)x6(reading values) array of values

end
RGBs(1:3,1:3)=readings(1,1:3,1:3);   %6 sensors, R,G,B values for each
RGBs(4,1:3)=readings(1,5,1:3);
RGBs(5,1:3)=readings(1,4,1:3);
RGBs(6,1:3)=readings(1,6,1:3);
%converts to grayscale values
g_val(1:6) = 0.299*RGBs(1:6,1) + 0.578*RGBs(1:6,2) + 0.114*RGBs(1:6,3);
g_val = g_val'; %transposes matrix
     % stores values as integers
pixel_data(1:cams,count) = double(int16(g_val(1:cams)));
     %gradiant calculation

     if (count >= 2)
          gradient_1 = pixel_data(:,count)-pixel_data(:,count-1);
     end
     if (window<=6)
          buffer(:,window)=pixel_data(1:cams,count);
          g(window)=sum(abs(gradient_1));
          G=sum(g);
     end

     if (G>=thresh1 && temp==0)   % threshold to start activity
          disp('Activity started')
          beep;
          temp=count; %temp switch activited
          activity(:,1:6)=buffer;
          %start of activity first 6 pixels stored from buffer to activity
     end

     if (G>=thresh2 && G < thresh1 && temp==0)
          %act is between(tresh2,thresh1)
          disp('Activity about to be started')
     end

     if (G>=thresh3 && G < thresh2 && temp==0)
          %act is between (tresh3,thresh2)
          disp('Residual motion') %frames not stored
     end

     if (G<thresh3 && temp==0)
          %act is less than tresh3 and activity is not progressing
          %i=i+1;
          disp(' No motion :Background') %frames not stored
     end
```

```
    if (G<thresh3 && temp~=0)   %very unusual case :
  %act is less than tresh3 and activity is progressing
        %i=i+1;
        disp('static during activity') %frames not stored
    end

    if (G>=thresh2 && a_rolloff==0 && temp~=0)
        %act is greater than tresh2
        len=len+1; %len incremented to store more frames in activity
        activity(:,len)=pixel_data(:,count);   %frames stored
        disp('Activity in progress')

    elseif (G>thresh3 && G<thresh2 && temp~=0)
%jitter during rolloff (do not get back to activity switch enabled)
        %roll_off_buffer(1:6,apple+1)=pixel_data(:,count);
        a_rolloff=a_rolloff+1;
        disp('You are ending your activity')
    end

    if (G>=thresh2 && a_rolloff~=0 && temp~=0) %very unusual case :
%when act falls below thresh2 and further jumps to greater than thresh2
        disp('You should not be moving now')
    end
    %very unusual case when rolloff does not exceed 2

    if ((G<=thresh3 && temp~=0) || (a_rolloff>=3 && temp~=0))
    %conditions of ending activity can be modified here
    %Acitvity must end after 3 frames of being below thresh2
        disp('Activity Ended')
        beep;
        %roll_off=(count-temp)-(len+1); %gives rolloff time
        ACT{1,k}=activity;
        activity_estimation_final(activity) %calling the estimation block
        clearvars activity;
        %call KNN function
        i=i+1;
    end

    count = count+1; % update count of frame
window=window+1; %update gradiant window
    if window==7
        window=1; %let window oscillate from 1 to 6
    end
end
end
```

Activity_estimation.m

```
%EC720
%Code developed by Siddhant Sharma & Neladri Bose
function   activity_estimation(activity)
%This function is used when one user input is given as samples

%interp and MVE
load('siddhant.mat')
[ANR,fs]=audioread('ANR.mp3');
[sit,fs]=audioread('sit.mp3');
[stand,fs]=audioread('stand.mp3');
[front,fs]=audioread('front.mp3');
[write,fs]=audioread('write.mp3');
activity_interp=zeros(6,40);
activity__interp_MVE=zeros(6,40);
for i=1:6
     activity_interp(i,1:40)=interp1(1:size(activity(i,:),2),activity(i,:),
linspace(1,size(activity(i,:),2),40),'pchip');
     activity__interp_MVE(i,1:40)=(activity_interp(i,:) - mean(activity_interp(i,:))) ./ var(activity_interp(i,:));
end

      group = [ones(10,1); 2*ones(10,1); 3*ones(10,1); 4*ones(10,1)];
      class=zeros(6,1);
      class(1)=knnclassify(activity__interp_MVE(1,:),TRAINING_MVE_cam1,group,1,'cityblock');
      class(2)=knnclassify(activity__interp_MVE(2,:),TRAINING_MVE_cam2,group,1,'cityblock');
      class(3)=knnclassify(activity__interp_MVE(3,:),TRAINING_MVE_cam3,group,1,'cityblock');
      class(4)=knnclassify(activity__interp_MVE(4,:),TRAINING_MVE_cam4,group,1,'cityblock');
      class(5)=knnclassify(activity__interp_MVE(5,:),TRAINING_MVE_cam5,group,1,'cityblock');
      class(6)=knnclassify(activity__interp_MVE(6,:),TRAINING_MVE_cam6,group,1,'cityblock');
      c(1)=size(find(class==1),1);
      c(2)=size(find(class==2),1);
      c(3)=size(find(class==3),1);
      c(4)=size(find(class==4),1);
      %camera_support=max(c);
      which_act=find(c==max(c));
      if(size(which_act,2)>1)
           sound(ANR,fs);
      elseif which_act==1
           sound(front,fs);
      elseif which_act==2
           sound(sit,fs);
      elseif which_act==3
           sound(stand,fs);
      else
           sound(write,fs);
      end
end
```

References

[1] Dai, Ji; Wu, Jonathan; Saghafi, Behrouz; Konrad, Janusz; Ishwar, Prakash, "Towards Privacy-Preserving Activity Recognition Using Extremely Low Temporal and Spatial Resolution Cameras," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Workshop on Analysis and Modeling of Faces and Gestures (AMFG),* pp. 68-76, June 2015.

[2] Dai, Ji; Saghafi, Behrouz; Wu, Jonathan; Konrad, Janusz; Ishwar, Prakash, "Towards Privacy-Preserving Recognition of Human Activities.", in *Proc. IEEE Int. Conf. Image Processing*, pp. 4238-4242, Sept. 2015.