COMS W4735y: Visual Interfaces To Computers – Assignment 1
Visual Combination Lock
Michelle Tadmor – mdt2125@columbia.edu
February 17, 2015

Matlab based combination lock allows setting and checking hand gestured passcodes from video files.
Check here for a video of the recognizer in action:
https://www.youtube.com/watch?v=lMOfdBqoBj0

Code is appended in the end of this file.

1. Domain engineering step
   Hardware:
   Video capture device: iPhone 6, iOS 8.1.3
   Video Analysis: MacBook pro, OS X Yosemite.

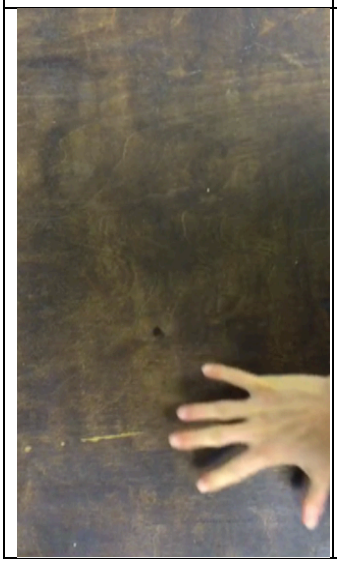
   Physical Domain:
   I tried a few setups, all of which had the camera filming from above in natural room light:
   1. Filming hand on a brown wooden table with texture. Wearing a bracelet
   2. Filming hand on a blue sheet with some texture wearing a sweater with a pattern
   3. Filming hand on a clean black sheet, no texture. Wearing a black sleeve.

   Software:
   Matlab R2014b with Image Processing Toolbox Video format: 30fps, Apple QuickTime Movie (`.mov`) file format
   Videos were transferred to a MacBook pro, OS X by email

   Access & Display:
   Video object is attained by calling the Matlab '`VideoReader`' command. Still images from the video are loaded one after by calling '`readFrame`' on the video object.
   Frame is displayed using '`imshow`' and various Matlab plotting commands such as plot, scatter, and more are used to plot the computed contours and metrics.

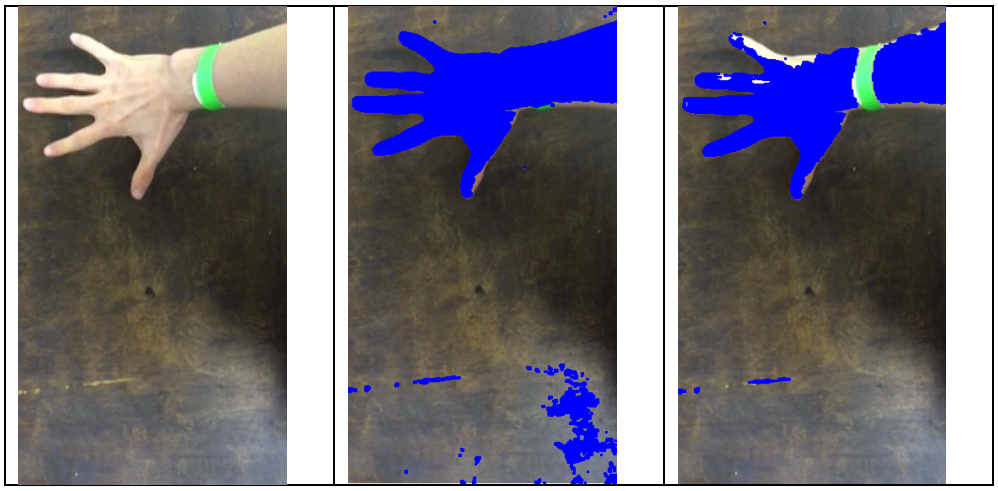
2. Data reduction
   Discretization (isSkin)
   Images are discretized by intensity threshold using Matlab's function '`im2bw`'. Anything below a given threshold is set to zero, anything above is set to 1. The threshold is first guessed using '`graythresh`'

and then iteratively increased until the second largest continuous region in the image is over 80% the size of the largest continuous region (meaning there are two similarly sized large continuous regions). This adjustment breaks incase a predefined maximum intensity threshold is reached.

image	Initial intensity	Increased intensity breaks up the secondary background noise
		

Optional flag allows the user incorporate a skin color quality into the discretization process. Pixels are further 'turned off' (set to zero) if the red rgb24 value is lower than a certain buffer above the green and blue values (buffer = 10 points higher). Originally, I tried to set a general range for skin RGB values but I found that to condition the relationship between the values was more robust than absolute values due to different brightness and contrasts across different videos.

Image	Intensity filter alone	Skin filter added bracelet is recovered

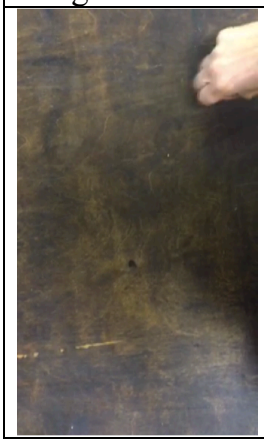
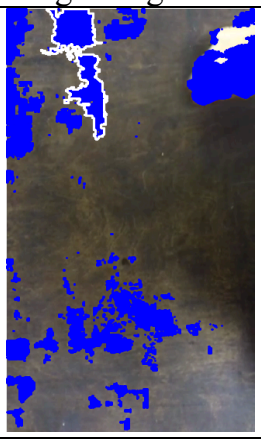
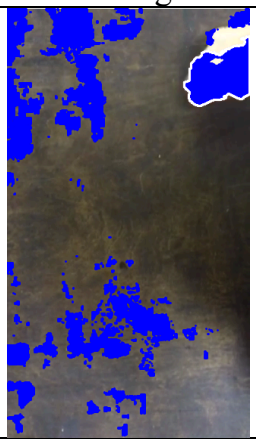


Segmentation (isHand)

The hand is selected as the largest continuous region in the discretized image. The collection of all continuous regions is extracted using the Matlab command '`bwboundaries`' and sorted by size.

To combat rare, large false positive skin areas that actually belong to the background or arm from being selected as 'hand', I employ another heuristic. The second largest region is selected instead of the largest if it satisfies the following:

1. Big enough (at least half the size of the largest)
2. Located in a more reasonable position (the center of mass calculated for both and if the second largest is within ten percent of the image size distance of the median of the last ten recorded hand positions)

Image	Largest region	Second largest
		

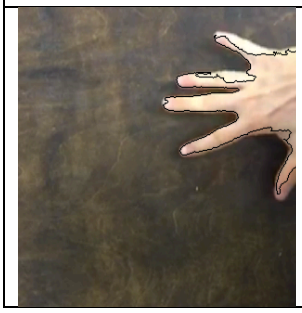
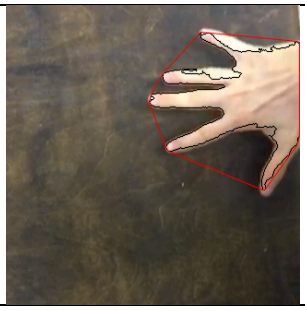
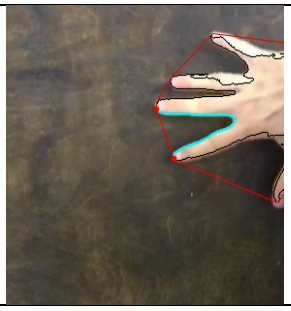
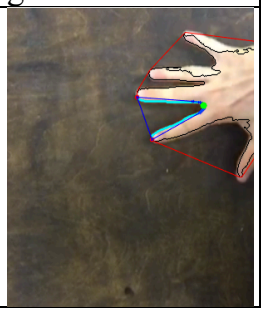
The Where:

The position (center of mass) of the hand is calculated by taking the mean of the pixel positions for the continuous region resolved as described above.

The What:

To determine whether the hand is in a fist, a peace sign, or splayed out, I use convex hulls and "convexity defects" to find and count the finger folds. I placed "convexity defects" here in quote since I am not using a known algorithm to retrieve them. I made up my own approach to find the most extreme points in the folds of the fingers as so:

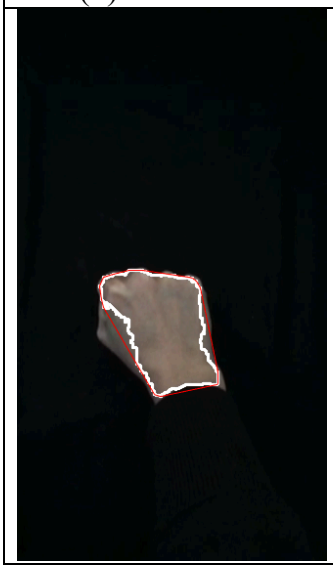
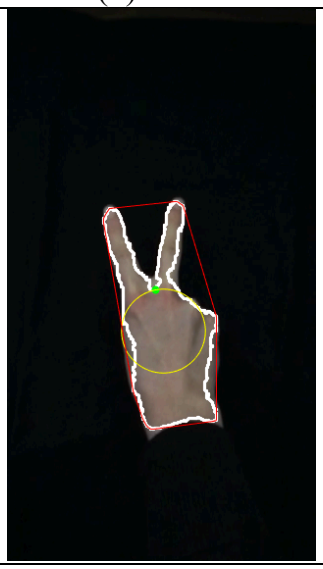
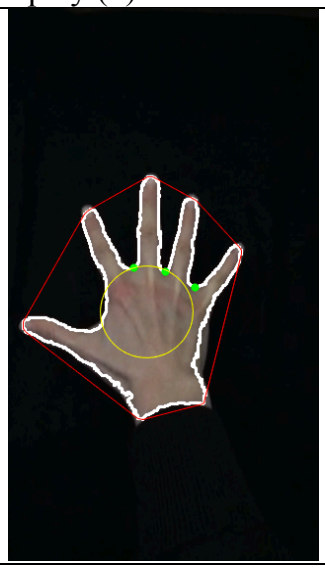
Psuedo Algorithm for finding finger folds	
1	For each two consecutive points in the convex hull, 'i' and 'j'
2	If the hand contour has more than 30 pixels between 'i'the two points
3	Compute the convex hull of the contour fold from 'i' to 'j'.
4	Find point 'p' (not equal to 'j') on the new convex hull that is the furthest away from 'i'.
5	If 'p' is further away from 'i' than 'i' is from 'j' then set 'p' to be a finger fold.

Image with hand contour	Convex hull	Contour of finger fold	Finger fold's convex hull and extreme point highlighted in green
			

If I find a single fold, the hand is in 'peace sign', 2 or more such folds the hand is splayed, otherwise closed fist. In the end

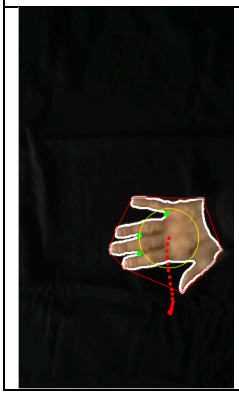
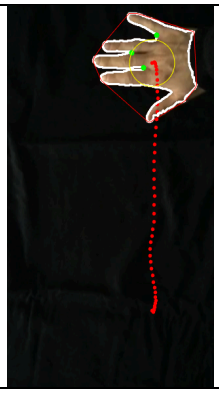
the splayed vector can have values either 0, 1, or 2 for fist, peace, or splayed respectively.

Here are the three hand recovered positions from video:

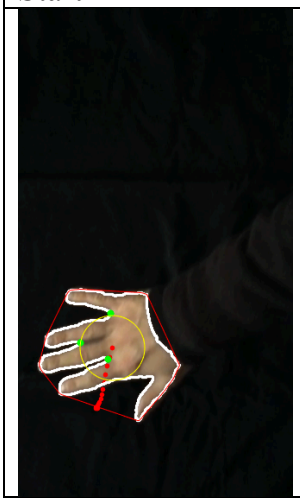
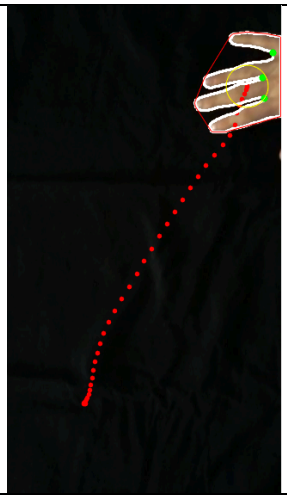
Fist (0)	Peace (1)	Splay (2)
		

3. Parsing and performance step.
   Grammer and vocabulary:
   Hand can gesture across the frame in 1 of 8 directions:
   1. Up
   2. Down
   3. Right

   Example:

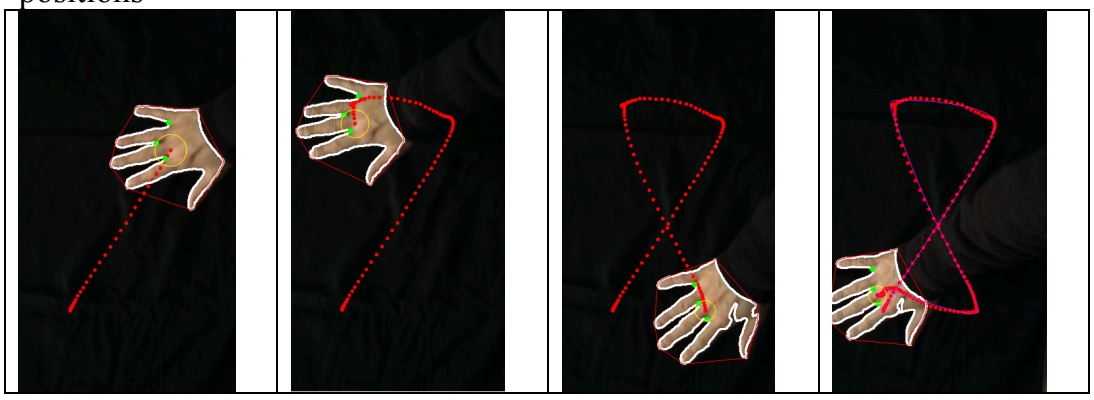
Start	End
	

4. Left
5. Diagonally up right
   Example:

Start	End
	

6. Diagonally up left
7. Diagonally down right
8. Diagonally down left.

Pattern example: Gesturing the figure eight. Red dots along the recorded positions

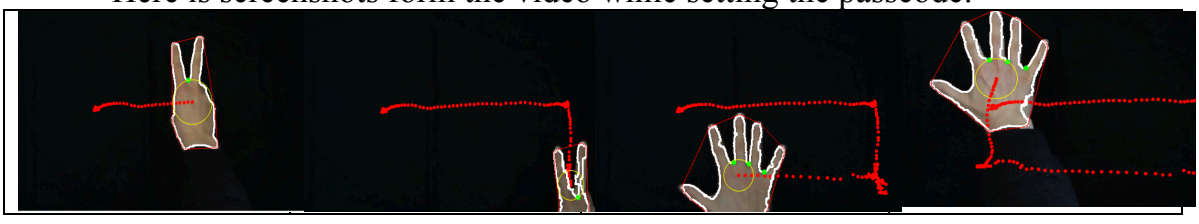


Hand can be flayed open, peace sign, or fist closed.
These options give 8X3 (24) characters to create a password with.
There must be a slight pause or even just a slowing down at turns for a new gesture to be recognized.

Swipe Direction Recognition and Movement Cleaning:

I used only relative positions to learn the directions rather than dividing the screen to quadrants. This gives the user more free movements and less attachment to specific locations on camera. The center of mass of each frame is subtracted from the center of mass computed in frame after resulting in a vector of directions.
The directions are smoothed using a Savitsky-Golay smoothing filter (using Matlab function '`smooth`').
The directions are then discretized by setting to zero a direction below a minimum magnitude(I specified 2 pixels per frame as minimum) and then setting then setting the direction vector to the sign of the vector resulting in the desired format of [<-1\0\1>,<-1\0\1>] for each frame.
Removal of sporadic outlier directions:
A new direction begins only if all five frames immediately after agree in the new direction and otherwise the frame direction is removed.
Direction stretches (repeating directions) are consolidated to a single direction for final passcode. The hand position (fist, peace, or splay) is selected by which was recorded the most times during a repetitive directional stretch.
Finally, between each direction change a consolidated direction [0 0] is expected and removed from final passcode. If the hand doesn't slow down enough to record the [0 0] direction, then one of the directions will be deleted.

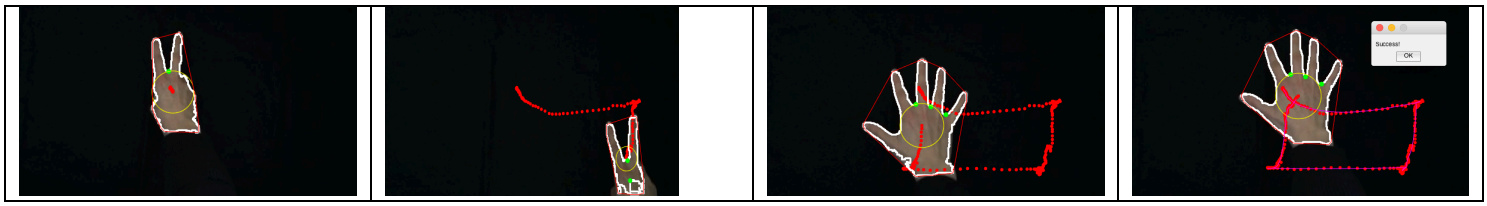
Performace:
Example use case:
Two code files for setting passcode are supplied
setPasscode.m and checkPassword.m
These files allow the user to select a video file from the hard drive and to save or check against the latest saved passcode.
Here is screenshots form the video while setting the passcode:



The passcode is a square:
right (peace), down (peace), left(splay), up (splay)
it is save as a matrix where the first two columns save the direction and the last one the hand layout:

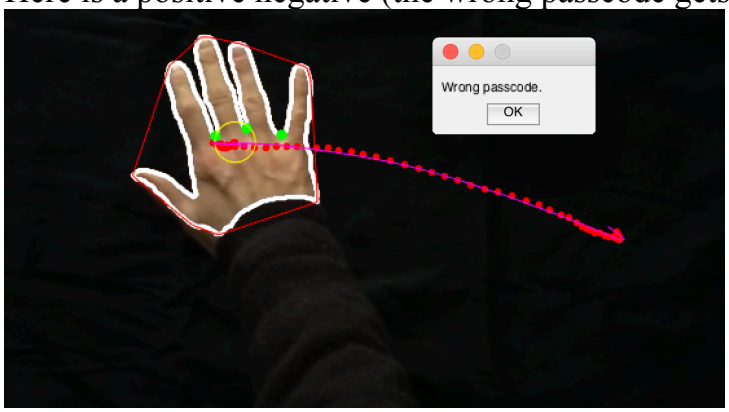
0	1	1
1	0	1
0	-1	2
-1	0	2

Here are screenshot while checking for the passcode against a different video (last image shows a dialog that reads “success!”
Note that in this video the hand starts roughly at the center to prove the strength of the relative vs absolute positions.

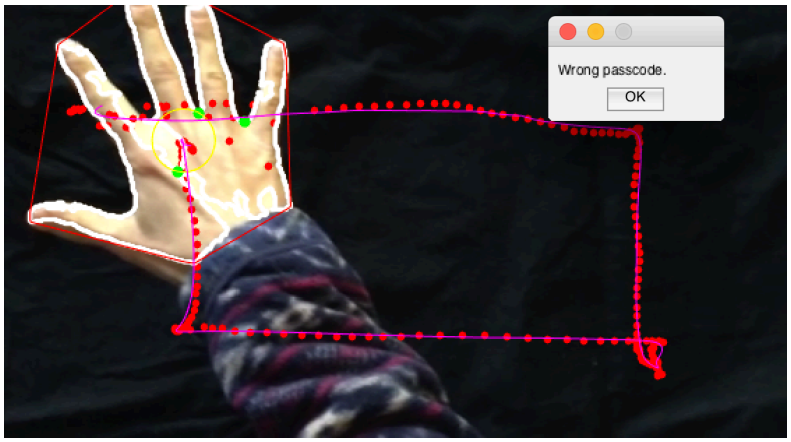


Here it is a third time against a different background:

Here is a positive negative (the wrong passcode gets rejected)



Here is a false positive (the correct passcode gets rejected) the image was misinterpreted because the correct hand layout is misidentified. This is happening because the faulty segmentation leads to many ‘convexity defects’



Matlab Code:

setPasscode.m

```
% Set a passcode either directly by supplying the passcode matrix
otherwise
% a file choose will open for you.
function passcode=setPasscode(passcode)
    % If left unspecified, ask user to supply a movie clip in '.mov'
format
    if ~exist('passcode', 'var')
        [filename, pathname, ~] = uigetfile('*.mov', 'Load New Password
Video');
        if isequal(filename,0) || isequal(pathname,0)
            passcode = [];
            return;
        end
        filename = [pathname filename];

        % == interpret video of gesture to passcode ==
        passcode=interpret_gesture(filename);
    end

    [curr_path, ~, ~] = fileparts(mfilename('fullpath'));
    passcodefilename = [curr_path filesep 'currpasscode.mat'];
    save(passcodefilename, 'passcode');
end
```

checkPasscode.m

```
% Check a passcode by supplying the passcode video file or otherwise
% a file chooser will open for you.
function passcode=checkPasscode(filename)
    passcode = [];
```

```
    % If left unspecified, ask user to supply a movie clip in '.mov'
format
    if ~exist('filename', 'var')
        [filename, pathname, ~] = uigetfile('*.mov', 'Load New Password
Video');
        if isequal(filename,0) || isequal(pathname,0)
            return;
        end
        filename = [pathname filename];


        % == interpret video of gesture to passcode ==
        passcode=interpret_gesture(filename);
    end

    [curr_path, ~, ~] = fileparts(mfilename('fullpath'));
    passcodefilename = [curr_path filesep 'currpasscode.mat'];
    file=load(passcodefilename);
    real_passcode = file.passcode;

    if any(size(passcode) ~= size(real_passcode)) ||
(any(any(real_passcode~=passcode)))
        msgbox('Wrong passcode.');
        passcode
        real_passcode
    else
        msgbox('Success!');
    end
end
```

interpret_gesture.m

```
function passcode=interpret_gesture(filename)

% params
verbos = true;
pink_skin = true;

% clean workspace
clc;
close all;

% init variables
pos = [];
splayed = [];
passcode = [];

% If left unspecified, ask user to supply a movie clip in '.mov' format
if ~exist('filename', 'var')
    [filename, pathname, ~] = uigetfile('*.mov', 'Load Password
Video');
    if isequal(filename,0) || isequal(pathname,0)
        return;
    else
```

```
        filename = [pathname filename];
    end
end

% open video
vidObj = VideoReader(filename);

% while video still has frames
while hasFrame(vidObj)
    % read new frame
    vidFrame = readFrame(vidObj);
    imshow(vidFrame);
    hold on;

    % Segment image
    [thresh, em] = graythresh(vidFrame);

    % convert to pos\neg image
    binFrame = rgb2bin(vidFrame, thresh, pink_skin);
    [contours, regions] = segment_image(binFrame);

    % locate hand in the many continous regions
    if (length(contours)>1)

        % check if there is more than one large region - go tougher on
thresh
        while length(contours{1})*.9 < length(contours{2})
            thresh = thresh*1.05; % increase by five percent
            binFrame = rgb2bin(vidFrame, thresh, pink_skin);
            [contours, regions] = segment_image(binFrame);
            if (thresh>.9)
                break;
            end
            if verbos
                fprintf('\nwarning: increased thresh to %g', thresh);
            end
        end

        % check for hand in the two largest continous regions
        handidx = 1;
        reg1 = getInds(regions==1);
        reg2 = getInds(regions==2);
        if length(reg1)/2 < length(reg2) && ...
            ~isCloseToRecent(reg1, binFrame, pos) &&
isCloseToRecent(reg2, binFrame, pos)
            if verbos
                fprintf('\nwarning: selected second largest region');
            end
            handidx = 2;
        end

        contHand = contours{handidx};
        binHand  = (regions==handidx);
    else
        contHand = contours{1};
        binHand  = (regions==1);
    end
```

```
    % Where: center of mass
    [i,j,~] = find(binHand);
    pos(end+1, :) = round(mean([i, j]));

    % what: use convex hull
    ch = convhull(contHand);

    % search for convexity defects
    defects = [];
    for p=3:length(ch)
        if abs(ch(p-1) - ch(p)) < 40;
            continue; % must have at some pixels inbetween to be
considered.
        end
        d = norm(contHand(ch(p-1), :)-contHand(ch(p), :));
        from=min(ch(p-1), ch(p));
        to=max(ch(p-1), ch(p));
        nuk = contHand(from:to, :);
        nukch = convhull(nuk);
        [nn, dists] = knnsearch(nuk(1, :), nuk(nukch(1:(end-2)), :));
        if any(dists >= d*1.1)
            [~,maxdistsid] = max(dists);
            defects(end+1,:) = nuk(nukch(maxdistsid), :);

        end
    end

    % estimate a circle around palm
    if ~isempty(defects)
        [~, r] = knnsearch(pos(end, :),defects);
        palm_r = median(r);
    end

    % is splayed or fist
    splayed(end+1) = size(defects, 1);

    % plot the frame with identified objects and metrics
    if (verbos)
%         frameInds = getInds(binFrame);
%         scatter(frameInds(:,2), frameInds(:,1), 100, '.b');
        scatter(pos(:,2), pos(:,1), 400, '.r');
        scatter(contHand(:,2),contHand(:,1), 100, '.w');
        if ~isempty(defects)
            scatter(defects(:, 2), defects(:, 1), 800,'.g');
            plot_circle(pos(end, :),palm_r, '-y');
        end
        plot(contHand(ch,2), contHand(ch,1), '-r');
        title(['em ' num2str(em)]);
    end

    pause(1/(vidObj.FrameRate*100));
    hold off;
end
hold on;
```

```
% == analyse gesture sequence ==

% examine the vectors of directions

% filter for reducing noise
spos1 = smooth(pos(:,1), 35,'sgolay');
spos2 = smooth(pos(:,2), 35,'sgolay');
spos = [spos1 spos2];

% grab the differences vector for direction
sdirs = diff(spos);

% expect at least consistent clear (magnitute) movements in a direction
magnitute = 2;
win = 5;
bindirs = sdirs;
bindirs(abs(sdirs)<magnitute) = 0;
bindirs = sign(bindirs);

% remove sporadic directional movements
rem = []; %list of indices to remove
curr_dir = [8,8];
for diri=1:(size(bindirs,1)-win)
    dir = bindirs(diri,:);
    if ~ismember(dir, curr_dir, 'rows')
        if (win)==sum(ismember(bindirs((1:win)+diri,:),dir, 'rows'))
            curr_dir = dir;
        else
            rem(end+1) = diri;
        end
    end
end
bindirs(rem, :) = [];
splayed(rem) = [];

% remove consecutive duplicates to get phrase
detected_directions_inds = find(~ismember(diff(bindirs),[0 0],
'rows'));
detected_directions = bindirs(detected_directions_inds, :);
tmp_splayed = [];
inds = [1; detected_directions_inds(:)];
for diri=2:numel(inds)
    % vector of splayed on stretch
    splayed_stretch = splayed(inds(diri-1):inds(diri));

    % save the number of defects most common for the stretch
    table = tabulate(splayed_stretch);
    [~, sind] = sort(table(:,2), 'descend');
    tmp_splayed(diri-1) = table(sind(1),1);
end
splayed = tmp_splayed;

% decide between consecutive non zero directions and short stretched
rem = [];
stretches = diff([1; detected_directions_inds(:)]);
for diri=2:numel(stretches)
    % if it is no a zero direction
```

```
    if ~ismember([0 0], bindirs(detected_directions_inds(diri), :),
'rows')...
        && ~ismember([0 0], bindirs(detected_directions_inds(diri-1),
:), 'rows')
        if stretches(diri) > stretches(diri-1)
            if stretches(diri-1) < 20
                rem(end+1) = diri-1;
            end
        else
            if stretches(diri) < 20
                rem(end+1) = diri;
            end
        end
    end
    if stretches(diri) < 15
        rem(end+1) = diri;
    end
end
f_directions = detected_directions;
f_directions(rem,:) = [];
splayed(rem)=[];

% remove duplicates again because previous step may create dups
keep = [true;~ismember(diff(f_directions),[0 0], 'rows')];

splayed = splayed(keep);
f_directions = f_directions(keep,:);

% remove rows of zeros
keep = ~ismember(f_directions,[0 0], 'rows');
f_directions = f_directions(keep,:);
splayed = splayed(keep);
splayed(splayed>2) = 2;

passcode = [f_directions splayed(:)];

plot(spos2, spos1, '-m');

end

% check if contour is the same object we've been tracking lately
function hand=isCloseToRecent(contour, img, past_positions)
% current logic: check if the new center of mass is within ten percent
of
% the median of the last ten positions.
if size(past_positions,1)<11
    hand=true;
    return;
end

hand = norm(mean(contour)-median(past_positions(end-10:end, :))) <
length(img)/20;
end

% segments an image to continous regions. regions\contours are returned
% sorted in descending length of perimeter.
```

```
function [contours, regions]=segment_image(binFrame)

    % continous regions - contours
    [contours,orig_regions] = bwboundaries(binFrame,'noholes');

    % sort by size
    table = tabulate(orig_regions(orig_regions~=0));

    [sorted, idx] = sort(table(:,2), 'descend');
    contours = contours(idx);

    % sort region numbers
    regions= orig_regions;
    for i=1:numel(idx)
        regions(orig_regions==idx(i)) = i;
    end
end

% converts an rgb image to a binary matrix where 1 identifies pixels of
a
% hand and zeros are background
function binFrame = rgb2bin(img, thresh, pink_skin)
    binFrame = im2bw(img, thresh);

    if (pink_skin)
        % Detect Skin
        Cr = img(:,:,1);
        Cg = img(:,:,2);
        Cb = img(:,:,3);
        binFrame = binFrame & Cr>(Cg+10) & Cr>(Cb+10) & Cb<=230;
    end
end

% plots a circle with a given center and radius on top of image
function plot_circle(center, radius, style)
    n = 100;
    t=linspace(0,2*pi,n);
    r=ones(1,n)*radius;
    [x,y] = pol2cart(t,r);
    x=x+center(1);
    y=y+center(2);
    plot(y,x,style);
end

% returns a two-column matrix with indices corresponding to the indices
% where the given matrix is positive
function inds=getInds(mat)
    [i,j,~] = find(mat);
    inds = [i,j];
end
```