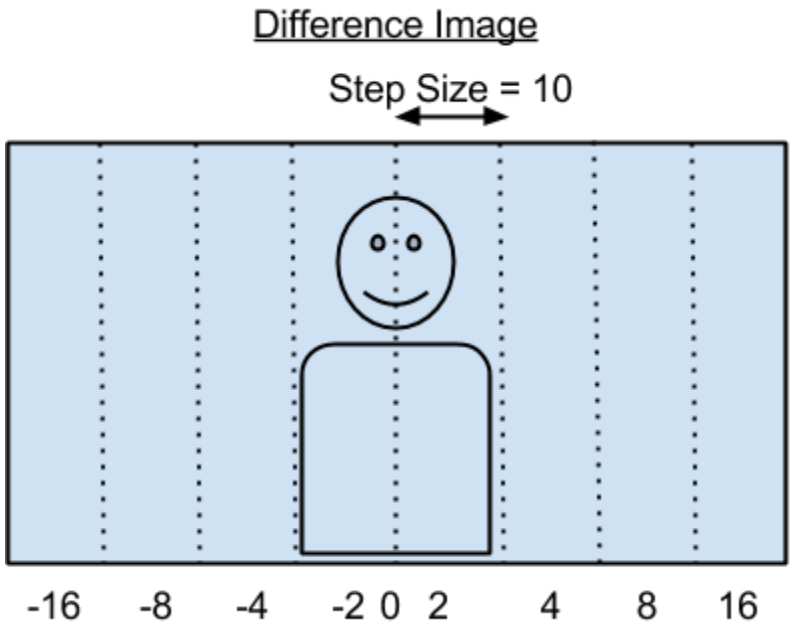
Hi Artem

I have attached an implementation in MATLAB of the assignment you gave me last Monday. You mentioned I avoid using Optical Flow as it was computationally expensive. You also stated I come up with an original approach. Based on these inputs I avoided using any well known Computer Vision algorithm. I came up with an algorithm that is a simple Frame Differencing technique. A brief overview of the approach.

Algorithm:

1. Capture an image frame from the video stream. Call this Frame 1. Convert this image to grayscale.
2. Capture the next frame from the video stream. Call this Frame 2. Convert this image to grayscale.
3. Find the absolute difference between Frame 1 and Frame 2 and store the difference as AD1.
4. Smooth the difference image AD1 with a 2D Gaussian kernel to remove noise, lighting variations and insignificant intensity variations.
5. Morphologically open the filtered grayscale difference image. The open operation assigns small regions of detected intensity differences (small regions of foreground embedded in the background) as background pixels. This helps eliminate insignificant motion artifacts.
6. Morphologically close the image obtained in step 5. The close operation assigns small holes (small regions of background embedded in the foreground) as foreground pixels. This helps recognize small regions which have moved across frames but detected as zero due to similar intensity across frames.
7. Threshold the resultant image obtained in step 6 such that only pixels that have a value greater than the threshold are selected. Let the thresholded difference image be denoted by AD_Th1. In my algorithm I have picked the pixel difference threshold as 40.
8. Compute a weight matrix Y_WT which has the same dimensions as the frame difference. Group every 10 columns of pixels. Assign notional weights to these groups of pixel columns in the manner shown in the diagram below.



For example, let pixels that are in the center column will have a weight of 0 (I have assigned a weight of one to the centre column). The next 10 columns to the right of the central column will have a weight of 2^1. The next 10 columns to the right will have 2^2. And so on. Similarly the 10 columns to the left of the central column will have weight of -2^1. The next 10 will have weight as -2^2 and so on.

9. For pixels that show up in the thresholded difference image AD_Th1, calculate the dot product of all those pixels and their associated column weight matrix Y_WT. Sum up this dot product and normalize the sum by total number of pixels above the threshold in image AD_Th1. Let this value be A1.
10. To recap AD1 is the difference image of Frame 1 and Frame 2. A1 is the weight of the difference image AD1.
11. Similarly capture a third image called Frame 3. Compute AD2 as the absolute difference between Frame 2 and Frame 3. Compute the weight of AD2 as A2.
12. Capture a fourth image called Frame 4. Compute AD3 as the difference between Frame 3 and Frame 4. Compute the weight of AD3 as A3.
13. Store Frame 4 in a variable called PrevFrame.
14. If (A1 < A2 < A3) then the direction of motion is towards the RIGHT.
15. If (A1 > A2 > A3) then the direction of motion is towards the LEFT.
16. If
    a. No direction is detected obtain the next image frame. Call this frame as NewFrame. Store A2 in A1, A3 in A2, compute A3 as the difference between PrevFrame and NewFrame. Assign NewFrame to PrevFrame. Go to Step 14. (The basic idea in the algorithm is to compute A1, A2 and A3 weights for 4 consecutive frames and detect motion by checking for the values of A1, A2 and A3. As I am using consecutive frames, I implemented a circular buffer to store A1, A2 and A3. On acquiring the next frame, I store A2 in A1, A3 in A2, and compute

only A3 instead of calculating all the three weights (A1, A2 and A3) for 4 consecutive frames when a new frame is acquired.

b. A direction is detected, clear out A1, A2 and A3 and PrevFrame. Add a delay of ~1 secs to avoid detecting false motion direction changes. (The delay has been added by skipping processing of a fixed number of video image frames after the motion detection event). Restart processing from Step 1.

I have coded up the above algorithm in MATLAB. Please see the README file for instructions on how to run the MATLAB program. To run the program a computer with a Webcam and MATLAB installed on it is needed.

Testing:

1. Ran tests for the three different scenarios that you provided in the assignment video link.
    a. Scenario 1: Tilt the head to indicate direction of motion
    b. Scenario 2: Turn the head to indication direction of motion
    c. Scenario 3: Swipe hand to indicate direction of motion

Caveats:

1. There are false positives being generated by the program. Sorry about that. I realize you have asked for the algorithm to be robust. I have tried to tune the algorithm as much as possible to avoid false positives.
    a. When detecting hand gestures if the arm is also visible along with the palm it can cause incorrect motion direction detection.
    b. When turning the head one needs to turn it with a bit of a swivel. This is because the step size for the column grouping is set to 10. If the column grouping is set to 5 or something smaller it helps detect the swivel motion of the head better. But this increases false positives if the person is shifting around a bit when stationary. The value of 10 was chosen as the step size based on some of the testing I did for the three scenarios.
    c. When tilting the head one needs to tilt well. Or the motion does not get detected. Again this depends on the value of the step size of 10.

Optimization:

1. Least Recently Used (LRU) approach among the three difference image weights A1, A2 and A3. In my algorithm I maintain a circular buffer of size 3 for the difference image weights. When I don't detect any motion I get rid of A1 by assigning A2 to A1, A3 to A2 and compute a new weight value of A3 using the last stored image and a newly obtained image. By storing only the difference weights as opposed to the last four recently used images memory consumption and compute cycles are lesser.
2. Also by always using the three most recently computed difference image weights the algorithm remains adaptive as it continually compares newly acquired images with recent images it has already processed.

3. By clearing out A1, A2 and A3 and by injecting a delay of about 1 sec when a motion direction is detected I found that the number of false positives for direction is reduced. This is to avoid incorrect/unnecessary motion direction detection when the head swings from the center to the right and then comes back from the right to the center. Without the delay and buffer clear the swing from center to right was detected correctly. But immediately a left motion was also detected for the motion of the head from the right to the center. To eliminate false positives like this the delay was added.

Alternative algorithm approaches:

1. To reduce false positives I tried with an approach where the condition was: (A1 >= 0 && A2 >= 0 && A3 > 0 && (A1 < A2 < A3)) OR (A1 <= 0 && A2 < 0 && A3 < 0 && (A3 < A2 < A1)) to detect right OR left directions. While this reduced false positives it also caused some valid direction detection cases in the scenario of hand swipe to be dropped. For this reason I have kept the condition check simple for now.
2. Another approach was to detect when the direction of motion changed. This would be true for scenarios where the head was being tilted/turned. If (A1>=0 && A2>=0 && A3>0 && A1 < A2 && A2 > A3) OR (A1<=0 && A2<0 && A3<0 && A1 > A2 && A2 < A3) then these would detect the change in direction when the head would move from the center to the right extreme and back OR from the center to the left extreme and back,respectively. The obvious problem with this approach was that it would not handle the hand swipe scenario robustly. For this reason this algorithm was not pursued. However, the advantage with this method is that once the motion is detected, a small delay (half of the delay in the current technique) is sufficient to prevent false positives.
3. I designed the weight matrix Y_WT for the approach mentioned above in item 2. This design of Y_WT works well for the current approach also. One could also have increasing positive weights for the different pixel column groups as opposed to having negative and positive weights.
4. The initial approach when I started work on this assignment was to use Optical Flow. But as you had stated it was computationally expensive the use of Optical Flow as an approach was dropped.
5. Another approach I thought of was to perform Canny Edge Detection of the face/hand on screen. Once a contour is created I compute the area of the contour and try to find the center of the contour. By computing the center of the contour between successive images I could try to find out the direction in which the center was moving. However I did not have time to try out this approach. You mentioned to come up with an original algorithm. Further the Canny Edge Detection algorithm is also computationally expensive.

Details about the MATLAB files:

The MATLAB code has been written using MATLAB R2011a. There are 7 MATLAB source files in the attachment. The files are:

1. main.m: Run this file to start the GUI
2. Approach_used_currently.m :
   This file uses the current method for detect motion.
3. Approach_1_in_alternative_algorithm_approaches.m:
   This file uses the method described in #1 in the "Alternative algorithm approaches" section. To try out this algorithm in main.m one can change the `UpdatePreviewWindowFcn` to "Approach_1_in_alternative_algorithm_approaches". The update preview function is set using the following command:
   `setappdata(hImage,'UpdatePreviewWindowFcn',@name_of_function);`
   Please replace "@name_of_function" by
   "@Approach_1_in_alternative_algorithm_approaches"
4. Approach_2_in_alternative_algorithm_approaches.m
   This file uses the method described in #2 in the "Alternative algorithm approaches" section. To try out this algorithm in main.m one can change the `UpdatePreviewWindowFcn` to "Approach_2_in_alternative_algorithm_approaches". The update preview function is set using the following command:
   `setappdata(hImage,'UpdatePreviewWindowFcn',@name_of_function);`
   Please replace "@name_of_function" by
   "@Approach_2_in_alternative_algorithm_approaches"
5. get_parameters.m: loads parameters and weight matrix y_wt while computing the weights A1, A2 or A3.
6. set_tunable_parameters.m: Change the tunable parameters by resetting their values in the function set_tunable_parameters.m before running the main.m program. This function is called in main.m to precompute the weight matrix Y_WT. The parameter n_disp_frames is not present in set_tunable_parameters.m. To change n_disp_frames, please find the parameter in the following functions depending on the motion detection approach you would like to use:
   a. Approach_used_currently.m
   b. Approach_1_in_alternative_algorithm_approaches.m
   c. Approach_2_in_alternative_algorithm_approaches.m
7. Find_a.m: computes the weight (A1, A2 or A3) for a pair of adjacent frames.

Tunable parameters in the code:

1. ***steps*** decrease this value to detect small tilts in head. This may lead to false positives in cases when the subject is casually moving his/her head.
2. ***step_wt*** increase this value to give more weightage to motion away from the frame centre. Useful in approach 2 suggested in alternative algorithm approaches.
3. ***sigma*** standard deviation of the Gaussian filter used to smooth the frame difference. Increase sigma if video quality of webcam is very noisy.

4. ***radius_str_ele*** radius of the structuring element used for morphological opening and closing. Increase the value to prevent motion artifacts from being detected in frame difference.
5. ***absolute_diff_threshold*** denotes the threshold above which pixel differences signify motion across frames
6. ***n_disp_frames*** Denotes the number of frames which have to be played back to the GUI without processing them. Increase this value to increase the delay when motion is detected to reduce false positives. Increasing this value also results in the detected direction of motion to persist for a longer time on the GUI.

   I have also enabled the detected direction to appear on the command window.

Running the program:
To run the MATLAB program, please have the following:

1. A computer/laptop with a WebCam
2. MATLAB installed on it. It should be able to run the MATLAB code that I have written using the 2011a (7.1.2.0) version of MATLAB.

Run Time:
On my system, it took 0.09 seconds on an average to detect motion.

Please let me know if you have any questions.

It was challenging and fun to work on this assignment. I am grateful for the opportunity.

Thank you

Harini Kishan