Heuristic-based Approach to detect Global Touch Gestures performed on Touch Devices

Hrishikesh Pardeshi
Adobe systems Pvt. Ltd
No. 5, Salarpuria Infinity,
Bangalore -29, India

Chandranath Bhattacharyya
Adobe systems Pvt. Ltd
No. 5, Salarpuria Infinity,
Bangalore -29, India

ABSTRACT

Global gestures on touch devices need to be detected so that this information can later be used to create checkpoints in a screen recording of the touch device. Currently, there is no uniform and legal solution to do so on devices like the iPad.

We propose a system with two cameras (RGB cameras) which will be able to detect global gestures performed on any touch device (or in fact, any surface like a book). The system will be able to track all touch gestures performed on a surface and either associate live actions with it or store the metadata for later use.

This paper primarily focuses on the heuristics applied to be able to make the system robust. It also aims to counter problems arising out of motion blur, lighting variations etc.

Keywords

Touch gestures, Touch devices, heuristics, and global gesture recognition.

1. INTRODUCTION

Touch Devices like the iOS and Android Tablets, phones etc. do not permit global gesture recognition. A particular application on these devices is unable to recognize gestures performed for other applications. For example, your application won't be able to record gestures performed while playing different games.

Global gestures on touch devices need to be detected so that this information can later be used to create checkpoints in a screen recording of the touch device. This facilitates creation of training sessions. For example, in order to teach a user how to browse the photo gallery on a smartphone, the author can create a screen recording of a real-time usage of the photo gallery. Then, if the gestures performed by the author are detected, these can be used to create checkpoints for the user and thereby, ask the user to perform the specific gesture at that specific location and point in time to be able to proceed.

In this paper, we propose a system with two cameras (RGB cameras) which will be able to detect global gestures performed on any touch device (or in fact, any surface like a book).

The system will detect the following:

1. Type of gesture – single tap, multiple taps, long press, left/right/down/top swipe, pinch, zoom.
2. Location of gesture – start and end.
3. Duration of gesture.

1. SETUP

1.1 Setup specifications

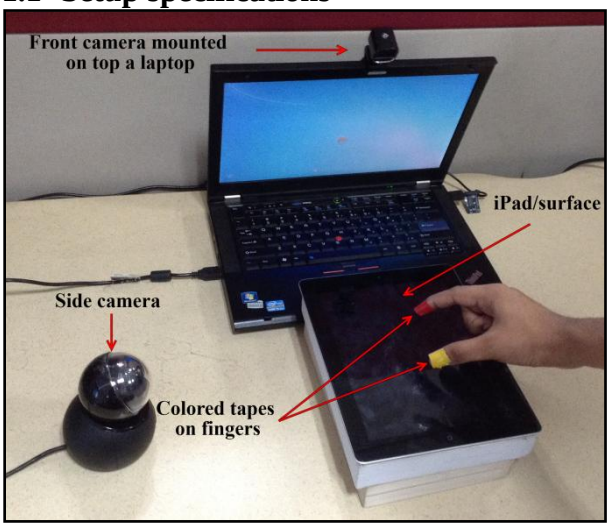

Figure 1. Image of the setup

1. The front camera has a projected 2D view of the entire surface and is mainly responsible for location handling.
2. The side camera can only observe the side view of the surface and will primarily be used to determine if a touch happens or not.
3. Figure 1 shows the fingertips of the user covered with colored tapes. Colored tapes are used to facilitate finger tracking.

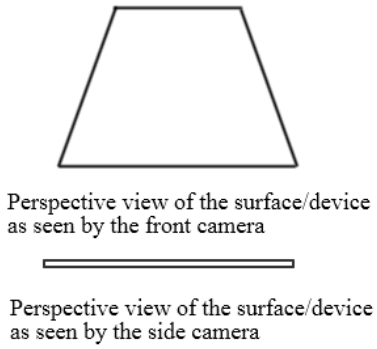

Figure 2. Perspective view of the front camera and side camera

With this setup, we will have two camera feeds/video streams. Each of the video streams is independently processed and both the streams are synchronized on global time.

1.2 Setup rationale

The setup of the cameras is an important part since no matter how you place the cameras; each camera will have its own perspective view of the 3D world.

1. A touch will indicate the start of a gesture. Hence, we need a camera to exclusively determine the touch event and thereby, details about the start and end of a gesture. The side camera does this in the setup.
2. We need a front camera having an entire view of the surface. This allows the front camera to define surface boundary and discard any random movement outside the surface boundary. The front camera is primarily responsible for determining the location of touch.

   The front camera could have been placed again as a side camera on the side perpendicular to the first camera. However, a camera placed on the side will have a perspective view (figure 3) and hence, we won't be able to find the correct location of touch.
3. The fingers need to be tracked throughout the recording. The user will have colored tapes attached to his/her fingers. This enables color detection methods to do finger tracking.

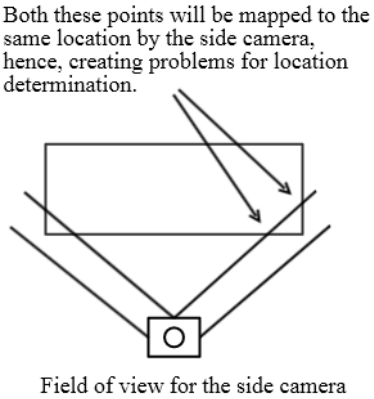

Figure 3. Perspective view of the side camera

2. PROBLEM STATEMENT

With the setup mentioned above, all global touch gesture algorithms from touch devices can be ported to this setup. Touch gestures are performed with the fingers and hence, it is necessary to track the fingers. As mentioned in the setup, we use colored tapes on the tip of fingers, which will be tracked as colored blobs. In our particular implementation, we aimed to detect single and two finger gestures. Hence, the index finger with color A and thumb with color B will be tracked.

When the cameras perform gesture detection, we run into problems of motion blur, lighting variations etc. These problems cause the blob tracking algorithm to loose blobs in certain frames. In order to tackle these problems and be able to accurately detect the gestures performed, we propose the following heuristics on top of the standard touch gesture algorithms.

3. RELATED WORK

Global gestures are generally detected by **hooking** your process to the operating system process and receiving all global gestures before the operating system does. On touch devices like smartphones and tablets, hook APIs (application programming interfaces) are not provided.

Certain applications on Android devices have found a workaround; however, the hacks are limited to only android devices [4].

With the setup mentioned in this paper, global gestures can be detected irrespective of the device being a smartphone or a tablet, iOS compatible or android compatible device etc. A device like Leap motion [1], which uses infrared cameras for depth awareness, can be used for tracking the fingers and detecting touch gestures. However, the use of two web/usb cameras is a cheaper and commonplace solution. Microsoft Kinect [2] also uses infrared cameras; however, the official documentation states that it is not intended and feasible for finger tracking.

4. HEURISTICS

4.1 General Touch Heuristics

1. While a gesture is performed, the blob might not be visible in a particular frame (frame 2, figure 4). The question is what needs to be done for frames in which no blob is detected. We have two options - End the gesture or ignore the frame.

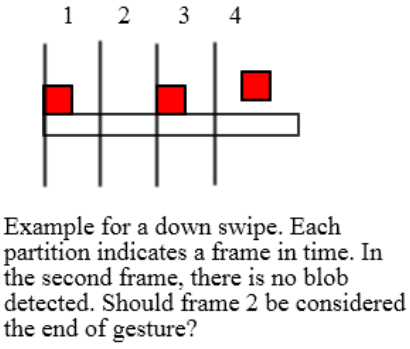

Figure 4. Blob loss during a swipe

If the camera is unable to detect any blob in a frame amidst a gesture, we do not end the gesture and simply, ignore that particular frame and do nothing. A gesture which has been started off by the touch on

a surface will end only when a blob is seen in the frame and is not touching the surface.

To summarize, frames in which no blob is detected are all discarded and do not affect the gesture detection logic in any way.

2. The blob might be shifted upwards (frame 3, figure 5) since the lower portion of the finger might get blurred during the swipe movement. This results in 2 gestures being detected when actually only one was performed.

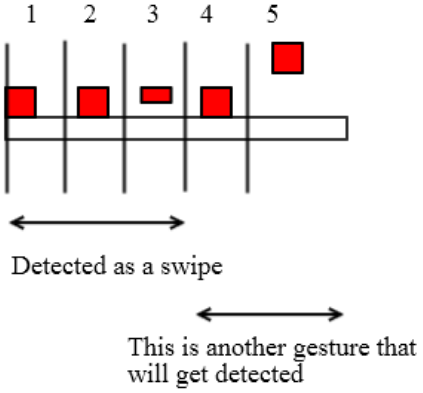

Figure 5. A specific case of down swipe

In order to tackle this scenario, we skip/avoid gesture detection logic for a particular time (e.g. 500ms/1s) after a gesture other than tap/long press has been detected (this includes swipes, pinch/zoom etc.).

In figure 5, after processing frame 3, the gesture will be marked as a down swipe. This heuristic will now introduce a skip time. Hence, frames 4 and 5 will not be processed for gesture detection and thereby, the false positive of tap will be avoided.

3. Effects like motion blur or lighting variations will create problems in blob detection. The blob size will vary throughout the recording and will generally be a fraction of the ideal blob size (frame 2, figure 6a).

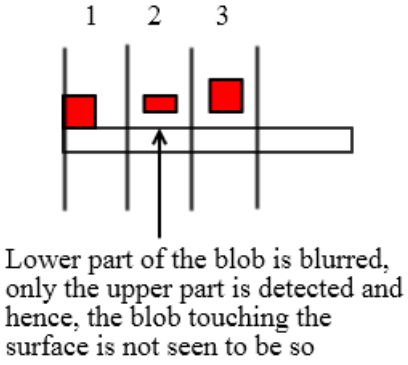

Figure 6a. Blob size reduces during a gesture

In order to tackle this, we apply the following heuristic:

While processing the first n frames of the camera feed, an ideal blob size is fixed by taking the maximum blob size/area observed among the n frames.

Ideal blob size (area) = max (blob size during first n frames)

If for any frame, the blob size falls below threshold (e.g. 75% of ideal blob size), the width and height of the blob is increased so that the modified blob size equals the ideal blob size.

After applying the heuristic to figure 6a,

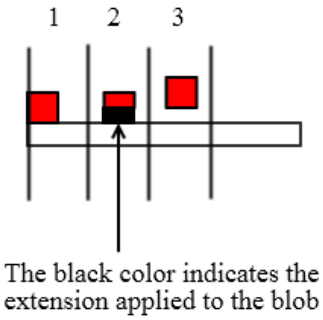

Figure 6b. Heuristic applied based on blob size

Note here that, the extension is always applied downwards. Hence, the blob can actually be seen lying just below the surface. This however will still be detected as a touch since the touch threshold that is applied is for the **absolute** difference. This means a touch (hypothetically) done from below the surface will also be deemed valid.

Also, this heuristic is applied only when the blob does not touch the surface. If a touch is already detected and the corresponding blob size is below the threshold, this heuristic is not applied as in the following case

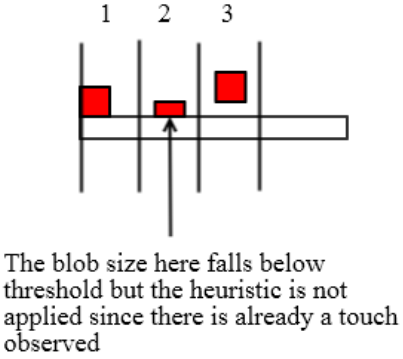

Figure 6c. Negative example for applying the heuristic

4.2 Front Camera Heuristics

1. Only the gestures performed on the device are to be considered. So, we need to determine if the gesture is performed within the bounds of the device.

The front camera has a complete (perspective) view of the device /surface. Hence, the front camera is used to check if the blob lies inside the device for every frame. When the side camera stream is being processed, we keep on checking if the blob was inside the surface bounds for the corresponding frame in the front camera stream. Only those frames for which the blob lies inside the device are considered for gesture detection by the side camera.

The default behavior for this heuristic is to return true, i.e. if the front camera is unable to detect any blob in a particular frame, it returns true by default.

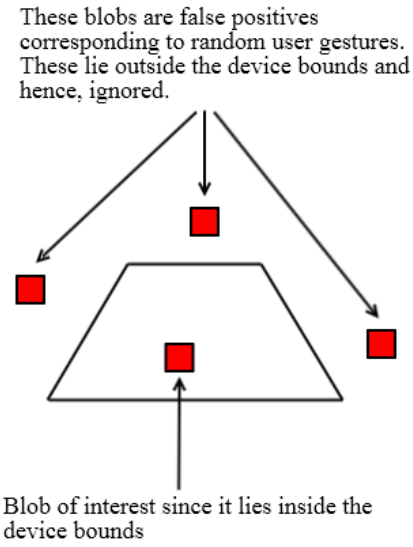

Figure 7. Blobs corresponding to random user gestures

2. The system with two cameras requires that there be a synchronization mechanism for both the cameras. Whenever the side camera sees a touch happening, the corresponding blob position in the front camera is noted. For this purpose, the front camera frame time closest to the side camera frame time is chosen.

   The synchronization mechanism or motion blur might lead to a loss of blob in the frame of interest of the front camera. In a rapid swipe movement, this might actually result in the left/right swipe being not detected at all. As an example, the side camera observes touch events in frames x, x+1, x+2 and x+3. It checks for blob position in corresponding frames from front camera. If there was no blob seen in frame x and x+1, it will hamper detection of left/right swipe.

   To avoid this scenario, if the front camera is not able to see a blob in the current frame, we check for blob in frames with offset -1 (previous), +1 (next), -2 (previous to previous), +2 (next to next).

3. The side camera view can be restricted to a white/neutral background. However, for the front camera, the background is difficult to be avoided. This increases the chance for losing the colored blob to a portion in the background. For example, if the user is wearing a shirt which is of the same color as the blob, the blob detection algorithm may go ahead and find the blob in the user's shirt.

   To avoid this, if the blob is displaced by a huge distance (threshold which is higher than swipe threshold) between consecutive frames, it cannot be part of a gesture/movement. It must be a false blob detected in the background. Hence, such a movement is prohibited (figure 8). In this case, the position of blob from the previous frame itself is marked as the current position of the blob.

   However, if such a movement occurs from a blob outside the device to a blob now lying inside the device, it is accepted.

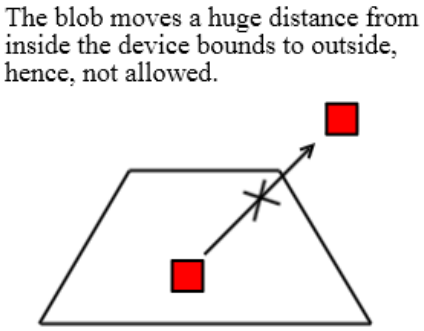

Figure 8. Random blobs in the background getting detected

4.3 Tap Heuristics

1. Consider the scenario in figure 9. Suppose that the user had actually performed a swipe but during processing, this event does not satisfy swipe gesture logic. Hence, it will be falsely detected as a tap/long press.

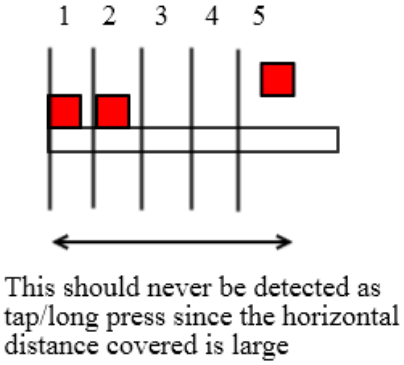

Figure 9. False positive of tap/long press

In order to avoid the false positive, an additional distance threshold is put in for tap/long press. We assert that during a tap/long press, the finger won't make such a large displacement in the horizontal direction.

2. Consider the case in figure 10:
   a. User moves his/her hand outside the device boundary.
   b. As part of this motion, his/her finger moves from below the device to above the device (as seen by side camera).
   c. The front camera is unable to track the blob.
   d. To the side camera, this seems like a tap (which it isn't)

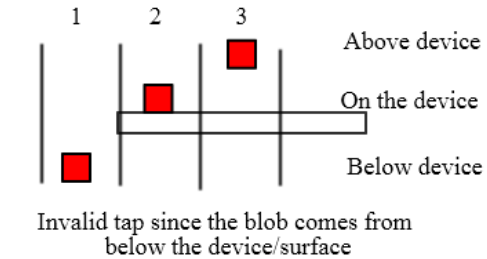

Figure 10: Not a tap

To handle this scenario, we apply the following secondary heuristic for taps. This states that there are 3 states for the finger w.r.t device – a) below device, b) on device and c) above device. For every frame in which the blob is detected, the position of the blob w.r.t device is marked as one of the three. If for any frame the blob is not seen, the position is not modified.

For a tap, the correct sequence for finger movement is: a) Above device b) On the device c) Above device (Figure 10).

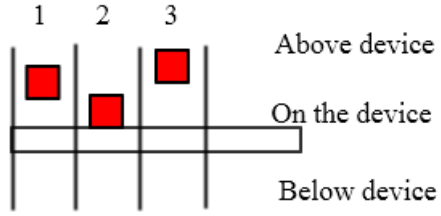

Figure 11. Sequence for a valid tap

In frame 2 of figure 10, the finger is seen lying on the surface of device. Using this heuristic, this won't be regarded as a tap (or any gesture) since the finger was below the device before the touch happened.

4.4 Swipe Heuristics

1. A swipe is identified by the distance it travels on the touch device.

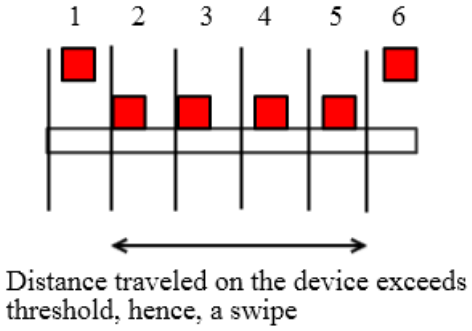

Figure 12a. An ideal down swipe

However, in practical scenarios, due to blob loss we have a situation like:

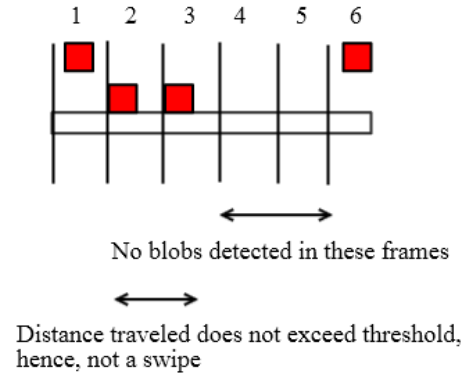

Figure 12b. A practical scenario of down swipe

To tackle this, the heuristic applied is to take the position of blob from the frame which marks the end of the gesture i.e. the first blob detected after the finger has been picked up from the surface.

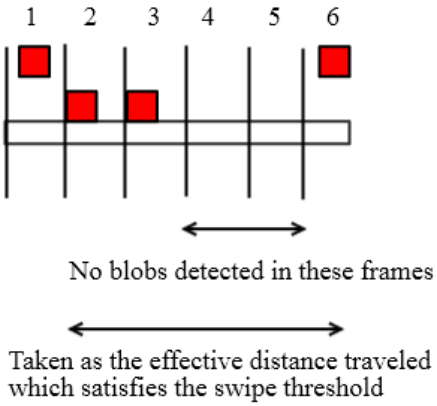

Figure 12c. Heuristic applied to down swipe

2. In case of a 2 camera system, each camera has its own perspective view. Hence, for e.g. when a top/down swipe is seen by the side camera as vertical distance travelled, there is also a small horizontal movement seen in the front camera. The side camera sees the movement as a top/down swipe and the front camera sees the movement as a left/right swipe, and, hence a clash.

   In order to tackle this, we first let both the cameras calculate the distance travelled in horizontal/vertical directions. Now, the direction in which the distance travelled is greater is marked as the winning candidate.

4.5 Pinch/Zoom Heuristics

1. Single finger and multi-finger gestures have mutually exclusive logic. By this we mean that if the user touches the device with two fingers, he/she is intending to perform a multi-finger gesture and not a single finger gesture.

   Hence, when both the fingers touch the device, skip the gesture detection logic for single finger gestures. This ensures that single finger gestures and two/multi finger gestures have mutually exclusive detection logic.

2. The pinch/zoom gesture is detected as follows:
   a. Both fingers touching the surface indicate the start of a pinch/zoom gesture.
   b. For every consequent frame, the current distance between both fingers is compared to the previous distance between the fingers.
   c. During the gesture event, this distance should ideally decrease/increase for every consecutive frame.
   d. If (c) continues over a specified number of frames, a pinch/zoom is successfully detected.

   The heuristics applied are as follows:

   a. A zero threshold of n (=3, say) pixels is adopted. This means that for a frame x+1, if the blobs from frame x have moved <= zero threshold pixels in a direction opposite to the

intended direction, the gesture still continues and no action is taken.
For example, if the distance between the fingers is decreasing frame by frame (indicating a pinch), and for a particular frame this distance increases rather than decreasing, it indicates end of gesture. However, if this change is less than the zero threshold, it indicates a minor movement which is ignored.

b. In case of (a), the previous distance update is skipped i.e. if heuristic (a) gets applied for frame n, then while examining frame n+1, the previous distance would refer to distance from frame n-1 and not frame n.

4.6 Home Button Heuristic

1. A device like the iPad has a home button which lies outside the screen boundary. If the user does a home button press event, it won't be detected since the front camera would return that the finger does not lie inside the surface and hence, is a false tap.
To counter this, a separate module determines if at the current moment in time, the finger is near the home button. This is achieved by applying offset downwards from the lower edge of the device, where the home button resides.
Thus, if the finger is near the home button during a touch event seen by the side camera, it is concluded to be a home button press.

2. The gesture detection algorithm requires that the device /surface boundary be known. This boundary can be explicitly specified by the user or can be detected by the algorithm. We use color detection to detect the surface boundary. We ask the user to open up a white screen on the device during the process of boundary detection. We then find all white contours in the given frame. The largest contour corresponds to the surface bounds. We form a quadrilateral from the contour and this quadrilateral is the boundary of the surface.
The device bounds are essential to determine the location of touch on the surface. The touch location is calculated as shown in figure 13

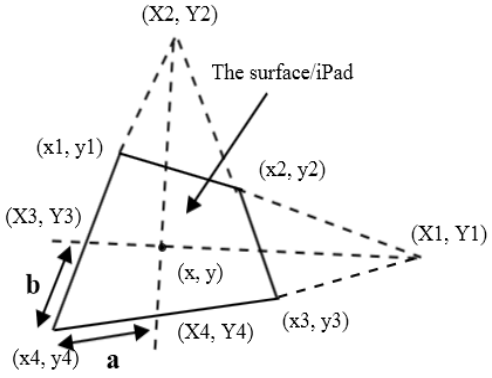

Figure 13. Determining the touch location

In figure 13, (x, y) refers to the co-ordinates of the point of touch in the frame. Using line extension and intersection, we find points (X1, Y1), (X2, Y2), (X3, Y3) and (X4, Y4) in that order. The final aim is to calculate the intercepts a and b, which correspond to co-ordinates within the device bounds.

5. RESULTS

We made use of OpenCV [5] for processing video streams, color-tracking etc.

We used the following thresholds to account for practical usage scenario (The thresholds mentioned in terms of pixel values are w.r.t a video resolution of 640 X 480)

1. If the distance traveled is greater than **15 pixels**, the gesture is a swipe.

2. If the finger lies within **7 pixels** from the device/ surface, it is considered a touch on the surface.

3. The gesture detection logic is skipped for **500 milliseconds** after a swipe or pinch/zoom is detected.

4. All blobs having an area less than **20 pixel^2** are ignored.

5. A tap will be recognized as a long press if the touch on the surface remains for more than **1 second**.

6. Two taps are detected as a double tap if they occur within **250 milliseconds** of each other. Otherwise, they are detected as two separate taps. This logic extends for n taps.

7. For a successful pinch/zoom, the distance traveled on the surface should be greater than **20 pixels**.

The thresholds mentioned above are the minimum/maximum thresholds with which the system works correctly. These thresholds have been obtained after repetitive experiments.

The heuristics were developed considering specific use-cases that come up since every user has a peculiar way of performing touch gestures.

The setup was tested with 7 users with applications including Photo gallery, temple run etc. We faced more challenges while capturing gestures for games like temple run etc. since the user is not consciously performing any gesture.

The following table shows the data for specific scenarios of 3 users

User	Gestures performed	Gestures detected correctly	Accuracy
CB	36	35	97.22%
VS	24	24	100%
KF	58	56	96.55%

Table 1. Results

The following table shows the results for an experiment conducted with the system without using heuristics (row 1) and the system with the heuristics (row 2)

System	Gestures Performed	Gestures detected correctly	Gestures not detected or incorrectly detected
Without heuristics	22	9	15
With heuristics	22	20	2

Table 2: Results with and without heuristics

6. FUTURE WORK

The setup mentioned in this paper was devised with an aim to recognize global touch gestures performed on touch devices and store the metadata for later use. However, any surface can be used as a track pad to perform live actions on a machine associated with the gestures detected. This would allow porting all touch gestures like swipe, tapping etc. to non-touch devices.

Associating live actions with gesture detection would require the system to be fast. To improve performance, some set of heuristics might need to be avoided in certain conditions and can be an area of investigation.

7. REFERENCES

[1] Leap motion device: https://www.leapmotion.com/

[2] Microsoft Kinect device: http://www.microsoft.com/en-us/kinectforwindows/

[3] D. Exner, E. Bruns, D. Kurz, A. Grundhofer, and O. Bimber, "Fast and robust CAMShift tracking", IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp.9-16, 2010.

[4] The SwipePad application for android devices is able to recognize global gestures https://play.google.com/store/apps/details?id=mobi.conduction.swipepad.android&hl=en

[5] OpenCV, an open-source library for computer vision http://opencv.org/