Crowd Scene Tracking

Michael Feist
mdfeist@ualberta.ca

Tamara Bain
tbain@ualberta.ca

Maciej Ogrocki
ogrocki@ualberta.ca

Benjamin Lavin
blavin@ualberta.ca

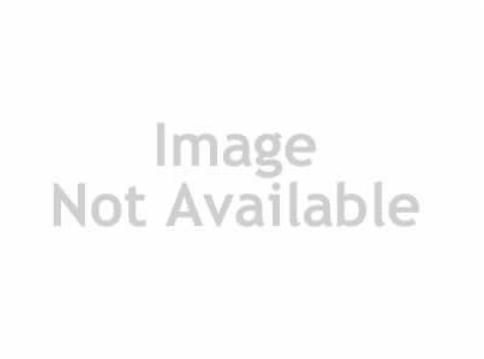

Fig. 1. Results of the density algorithm. Green for areas with low density, yellow for areas of medium density, and red for areas of high density.

A. Density

One crowd parameter that was useful in other areas of the project was crowd density. In order to calculate the density of a crowd we need to first make some assumptions. The first is that the people in the crowd are all moving roughly the same speed. We also assume that the size of a person stays roughly the same as they move through the scene. Finally we assume that the more an area in the scene is changing the more people are in that area.

With these assumptions we are able to calculate the density of a crowd by calculating the temporal difference. To calculate the temporal difference we simply take the current frame and subtract it from the previous frame. We then take the difference and create a threshold image. We found that a change in magnitude greater than 20 was significant enough to warrant valid change. Usually you dont want to include all changes in your threshold image since then the threshold would include noise from the camera. Next we divide the scene into smaller blocks. For scenes like the ones shown in Fig. 1 we found block sizes of 8 by 8 worked well. Next we iterate over the blocks and calculate the percentage of change. We then add the percent of change times some growth rate to the density map. Also it is important to decrease the density of the blocks over time.

$I_t \leftarrow |Im[x, y, t + 1] + Im[x, y, t]|$
$I_{threshold} \leftarrow I_t > threshold$
$blocks_x = Im_{width}/block_{size}$
$blocks_y = Im_{height}/block_{size}$
for *i := 0; i* $< blocks_x$*; i++* **do**
  for *j := 0; j* $< blocks_y$*; j++* **do**
    $density[i, j] \leftarrow I_{density}[i, j] - decay\ rate$
    if $density[i, j] < 0$ **then**
      $density[i, j] \leftarrow 0$
    end
    $x1 \leftarrow i(block_{size})$
    $x2 \leftarrow i(block_{size}) + block_{size}$
    $y1 \leftarrow j(block_{size})$
    $y2 \leftarrow j(block_{size}) + block_{size}$
    $block \leftarrow I_{threshold}[x1 : x2, y1 : y2]$
    $density[i, j] \leftarrow$
    $density[i, j] + \alpha \sum_x^N \sum_y^N block[x, y]$
    if $density[i, j] > 1$ **then**
      $density[i, j] \leftarrow 1$
    end
  end
end

Algorithm 1: Density Calculation

B. Optic Flow

Optic Flow was used in other papers to calculate the flow of the crowd [1][2][5]. Most of these papers used the calculated flow information in a simulator. However, since we did not have a simulator the flow information of the crowd was not as useful to us. That being said we still were easily able to calculate the flow of the crowd using OpenCVs built-in Optic Flow function.

```
flow =
    cv2.calcOpticalFlowFarneback(prev_gray,
    gray, None, 0.5, 3, 20, 3, 5, 1.2, 0)
```

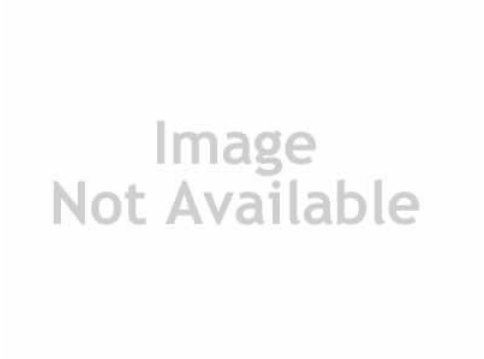

Fig. 2. Results of the Optic Flow algorithm. Small lines point in the direction in which the pixels are found to be moving. In this image you can see that the hand is moving very slightly from left to right.

There are a few issues with using Optic Flow to calculate the flow of the crowd. The first is that when two object are moving against each other they will cancel out the flow of both objects. Secondarily flow is not calculated equally over all objects in the crowd. Larger objects will have a greater in pack on the flow of the crowd as they will take up more space and appear as multiple pedestrians. As such videos that contain large non-pedestrian objects such as cars will be less accurate.

C. Tracking With Lucas-Kanade

In order to track people in a crowd we used Lucas-Kanade to track feature points. To start the algorithm we first search through the first frame in the video feed to find feature points. These feature points are areas in the image where corners are found. More specifically we used Shi-Tomasi corner detection. After we have our points we create a new track for each point.

We then start the main loop of the tracking algorithm. The loop starts by grabbing the next frame in the video feed and then trying to find the new locations of the feature points by using Optic Flow.

```
# The variable p0 is the original location of
    the points and p1 is the new location
p1, st, err =
    cv2.calcOpticalFlowPyrLK(prevgray, gray,
    p0, None, **lk_params)
```

Since people can move in and out of the frame we added a section of code that would add new feature points to the scene. This works by first calculating areas of change between the current frame and the previous frame. Next we search for new feature points in these areas of change. Finally we add the new feature point to our tracking points if that new feature point does not have a similar corresponding tracking point. These are points in relatively the same location.

We also try to remove points from the list of feature points, p1, that are no longer needed. The points that are removed are points that no longer move or stand still for an extended period of time. The main issue with removing points that remain still for an extended period of time is that we can easily remove points that belong to a person that has briefly stopped moving and will continue moving shortly. So for this algorithm to work we need to assume that people are not stopping.

TABLE I
A SIMPLE EXAMPLE OF HOW TRACK_INDEX IS UPDATED

Array	Values
st	$[1, 1, 1, 0, 1, 1]$
$tracks\_index\ before\ update$	$[0, 1, 2, 3, 4, 5]$
$tracks\_index\ after\ update$	$[0, 1, 2, -1, 3, 4]$

TABLE II
A SIMPLE EXAMPLE OF HOW ELEMENTS ARE ADDED TO TRACK_INDEX

Array	Values
$tracks\_index\ before\ update$	$[0, 1, 2, -1, 3, 4]$
$tracks\_index\ after\ update$	$[0, 1, 2, -1, 3, 4, 5]$

After we have our updated list of feature points, p1, we need to add the points to their corresponding track. This posed some initial difficulties since the size of p1 is constantly changing and the indexed points dont always correspond to the same indexed track. What we mean by this is that the point in p1 at index i does not always belong to the track at index i. To keep track of all the changing information we created four arrays.

1) p1: An array of all the active points. These are the points currently being tracked by Lucas-Kanade.
2) st: An array of which points in p1 are no longer active and were removed from p1 during this last iteration. It is a bit array where 1 means that the point is still active and 0 means that the point was deleted.
3) tracks: A 2D array storing the path that each point tracks. It is an m x n array where m is the number of frames and n is the number of points.
4) tracks_index: Since p1 and tracks dont match in size or indexing, we need tracks_index to keep a map between p1 and tracks.

The tracks_index array is updated by first initializing an array with values 0 to n-1, where n is the number of points in p1. Each iteration we obtain a list of the points which were deleted from p1, st. We then iterate over st and, if a point is removed, we set the value in tracks_index corresponding to the index in st to -1, and subtract one from all the remaining values in tracks_index.

After this step, if p1 has more points than the max value in tracks_index, it means that an additional point was added. In this case we append the max value plus one to the end of tracks_index.

Finally, we go through p1 and find the corresponding index in tracks_index. We then add the point in p1 to the correct path in tracks, given the found index in tracks_index.

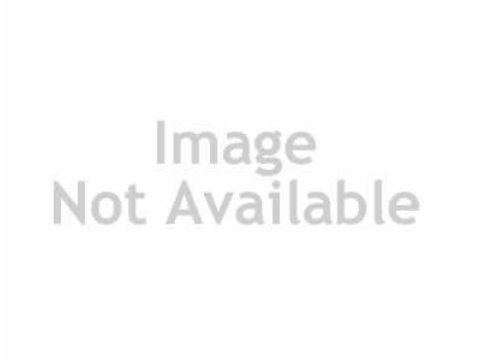

Fig. 3. Results of our trained Haar Cascade that is detecting people.

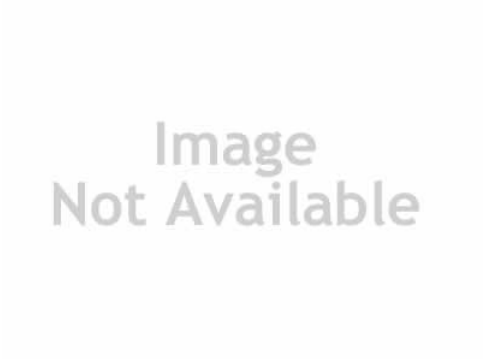

Fig. 4. Results of the tracking algorithm that utilizes the Object Detection.

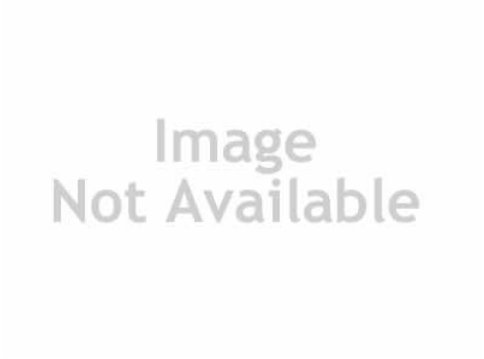

Fig. 5. Results of the Unity crowd playback data outputted from the tracking algorithms.

D. Object Detection

E. Tracking with Object Detection

F. Output

G. Unity

VI. Results

VII. Discussion

References

[1] N. Courty and T. Corpetti, 'Crowd motion capture', *Computer Animation and Virtual Worlds,* vol. 18, no. 4-5, pp. 361-370, 2007.

[2] B. Boghossian and S. Velastin, 'Image Processing System for Pedestrian Monitoring Using Neural Classification of Normal Motion Patterns', *Measurement and Control,* vol. 32, no. 9, pp. 261-264, 1999.

[3] K. Lee, M. Choi, Q. Hong and J. Lee, 'Group Behavior from Video: A Data-Driven Approach to Crowd Simulation', *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2007),* pp. 109-118, 2007.

[4] R. McDonnell, M. Larkin, S. Dobbyn, S. Collins and C. O'Sullivan, 'Clone attack! Perception of crowd variety', *ACM Trans. Graph.,* vol. 27, no. 3, p. 1, 2008.

[5] R. Mehran, A. Oyama and M. Shah, 'Abnormal Crowd Behavior Detection using Social Force Model', *Computer Vision and Pattern Recognition,* vol. 2009, pp. 935-942, 2009.

[6] M. Rodriguez, I. Laptev, J. Sivic, and J. Audibert 'Density-aware person detection and tracking in crowds', *Computer Vision (ICCV),* 2011 IEEE International Conference on, pp. 2423-2430. IEEE, 2011.

[7] D. Zhang, Y. Lu, L. Hu and H. Peng, 'Multi-human Tracking in Crowds Based on Head Detection and Energy Optimization', *Information Technology J.,* vol. 12, no. 8, pp. 1579-1585, 2013.

[8] I. Ali and M. Dailey, 'Multiple human tracking in high-density crowds', *Image and Vision Computing,* vol. 30, no. 12, pp. 966-977, 2012.

[9] F. Zhao and J. Li, 'Pedestrian Motion Tracking and Crowd Abnormal Behavior Detection Based on Intelligent Video Surveillance', *Journal of Networks,* vol. 9, no. 10, 2014.

[10] I. Ali and M. Dailey, 'Head Plane Estimation Improves Accuracy of Pedestrian Tracking in Dense Crowds', *Control Automation Robotics and Vision (ICARCV),* 2010 11th International Conference on. IEEE, 2010.

[11] D. Forsyth, 'Object Detection with Discriminatively Trained Part-Based Models', *Computer,* vol. 47, no. 2, pp. 6-7, 2014.

[12] S. Saxena, F. Brmond, M. Thonnat and R. Ma, 'Crowd Behavior Recognition for Video Surveillance', *Advanced Concepts For Intelligent Vision Systems,* vol. 9783540884576, p. 970, 2008.

[13] R. Eshel and Y. Moses, 'Tracking in a Dense Crowd Using Multiple Cameras', *Int J Comput Vis,* vol. 88, no. 1, pp. 129-143, 2009.

[14] A.B. Chan, Z.-S.J. Liang and N. Vasconcelos, 'Privacy preserving crowd monitoring: Counting people without people models or tracking', *Computer Vision and Pattern Recognition,* 2008. CVPR 2008. IEEE Conference on , pp.1,7, 23-28 June 2008

[15] V.K. Singh, Bo Wu and R. Nevatia, 'Pedestrian Tracking by Associating Tracklets using Detection Residuals,' *Motion and video Computing,* 2008. WMVC 2008. IEEE Workshop on, pp.1,8, 8-9 Jan. 2008

[16] M. Liem and D. Gavrila, 'Joint multi-person detection and tracking from overlapping cameras', *Computer Vision and Image Understanding,* vol. 128, pp. 36-50, 2014.

[17] S. Ali and M. Shah, 'Floor Fields for Tracking in High Density Crowd Scenes', *Computer Vision ECCV 2008,* vol. 5303, pp. 1-14, 2008.

[18] S. Pellegrini, A. Ess, K. Schindler and L. van Gool, 'Youll Never Walk Alone: Modeling Social Behavior for Multi-target Tracking', *Computer Vision,* 2009 IEEE 12th International Conference on, pp. 261 - 268, 2009.

[19] A. Yilmaz, O. Javed and M. Shah, 'Object tracking', *CSUR.,* vol. 38, no. 4, pp. 1-45, 2006.

[20] M. Thida, Y. Leng Yong, P. Climent-Prez, H. Eng and P. Remagnino, 'A Literature Review on Video Analytics of Crowded Scenes', *Intelligent Multimedia Surveillance,* pp. 17-36, 2013.

[21] R. Hartley and A. Zisserman, Multiple View Geometry, *Cambridge University Publishers,* 2nd ed. 2004

[22] G. Shu, A. Dehghan, O. Oreifej, E. Hand and M. Shah, Part-based Multiple-Person Tracking with Partial Occlusion Handling, *Computer Vision and Pattern Recognition,* 2012 IEEE Conference on (pp. 1815-1821), 2012.

[23] P. Viola and M. Jones, 'Rapid Object Detection using a Boosted Cascade of Simple Features Computer Vision and Pattern Recognition Proceedings', *2001 IEEE Computer Society Conference*, vol. 1: I-511 I-518, 2001.