Automatic Object Segmentation

Duc Tran
dt2259@nyu.edu

Andrew Weng
aw4108@nyu.edu

Russell Wustenberg
rw2873@nyu.edu

Ye Xu
yx2534@nyu.edu

***Abstract*—Motion analysis is a powerful tool for identifying objects within image sequences and video. While learning-based segmentation methods dominate the space today, previous research shows that a ground-up approach for image segmentation based on motion analysis exists. In this study, we explore and evaluate work done in joint research between the Computer Vision Group at Universität Freiburg and the University of California, Berkeley. We implement their from-principles segmentation pipeline in a naive manner to better understand the methods by which they achieved their results.**

***Index Terms*—segmentation, optical flow, moving image, trajectory analysis, replication study**

I. Introduction

The Computer Vision Group based at Universität Freiburg has published extensively on the topic of motion analysis in image sequences [6]. One of their prominent research streams is object segmentation based on motion cues in video. Video object segmentation is a key task in many computer vision applications.

Today, segmentation is often handled by deep learning agents trained on one or several task-relevant datasets such as [13], [26], and [18]. One predicate for supervised learning is a ground truth in the form of an annotated image or image sequence against which the agent verifies its hypotheses [19]. For many years, these ground truths were prohibitively expensive to acquire. They often involved manual annotation of at least a portion of the image frame or sequence which requires investment of human labour and time.

One cost saving measure available in recent years is to create computer generated datasets, made possible by the recent increase in graphics computing, which allow ground truth encoding on an atomic level [4], [7]. Such an approach, however, is still a specialist exercise and can not be automatically applied to existing video footage. A joint research initiative between the Computer Vision Group and the University of California, Berkeley, sought to develop such a ground-up approach to the segmentation of moving objects in a scene [3], [15], [24].

The Freiburg-Berkeley authors took biological vision as their model. Biological vision does not rely upon a learned visual model for the task of segmentation [23]. Instead, cues are taken from textures, shape and, most notably, motion [16].

In [3], authors Brox and Malik propose an algorithm relying entirely upon information extracted from the image sequence that results in segmentation. Such work is exciting, as it both gives insight into the biological mechanisms behind vision and provides a powerful tool for computer vision research. Furthermore, they claim a number of improvements via their methodologies over well-known techniques in several task domains, such as optical flow estimation and point tracking [3], [15].

The authors do not give this algorithm a unique moniker and, as a product of joint research by a team, there are too many surnames to refer to it in the style of [14] or [8]. For the purposes of this paper, we will refer to the Freiburg-Berkeley algorithm as Long Term Motion Analysis (LTMA), as this sets the approach apart from others. A full implementation of the LTMA method already exists for review as both source code and compiled binaries at [22].

We seek to implement the LTMA algorithm from scratch with the aim of understanding the techniques involved and the principles behind them. One barrier arose that much of the Computer Vision Group research was rapidly subsumed into learning pipelines ([7], [17]) and no articles have been found which unpack the principles of LTMA in a manner similar to [5]. The availability of survey articles such as [5] provide greater access to the concepts that may otherwise be tricky to implement based on more scholarly research.

In this study, we attempt such an 'unpacking' of the LTMA algorithm by attempting a from-principles replication of several of its components. As the algorithm is highly dependent upon previous research from the Computer Vision Group, such as [2], one consideration we had to make was just how many components we sought to replicate. With the thorough but accessible style of informed tutorials in mind, we have primarily focused on some components to the disadvantage of others. One topic which we consciously avoided was variational calculus, as it proved inaccessible in our initial exploration and time was a factor. We hope, by our efforts, to make this body of research more accessible to further research and to popularize a currently underrepresented tool for working with video applications.

II. The LTMA Algorithm

The authors' work was inspired by the theory of the 'Gestalt' as proposed by Max Wertheimer and Kurt Koffka [12]. The German word *die Gestalt*, in this sense, is translated as "the configuration" or "the composition [of parts]." In its use as a psychological theory, 'die *Gestalt*' is the driving philosophy that "the whole of anything is greater than its parts" [Britannica, die Gestalt]. Whereas prior psychological analysis had attempted to atomicize the human experience, *Gestalt*

psychology sought to take in the whole context of a person's experience and seek holistic explanations for behavior.

Likewise, the LTMA method attempts to take in a scene as a whole and, from the composition of its perceived features, construct an overall *Gestalt*. Ultimately, LTMA constructs this *Gestalt* based on the relationship between points analyzed over time and is, categorically, a clustering method. The outcome, so far as we will pursue it in this paper, is a class-based segmentation of tracked points in the sequence which are grouped based on their common motion patterns.

There are a number of steps which define the overall segmentation algorithm. First, each frame is pre-processed, extracting the derivative image with respect to both axes. From these, a threshold is placed on the image so that only prominent features are tracked. Next, these points are tracked throughout the image sequences. The trajectories created during this process are subject to pairwise comparison. The degree of similarity between two trajectories is placed in a constructed affinity matrix. Finally, spectral clustering is performed on this matrix and trajectories are assigned to classes based on their affinities. The outcome is that any trajectories with similar quality of motion are classified as belonging to the same object.

III. Picking Initial Points

A. Background

As the tracking is built on a dense (nearly per-pixel) optical flow estimation, tracking all pixels within an image would be intractable [15]. This is a common issue in point tracking, and many solutions have been proposed for how to select strong points to track within the image. One common trait among popular methods is the use of the structure tensor, also called the second moment matrix [10], [21].

The structure tensor is derived from the gradient of intensity values at a given pixel, $\nabla I_p = (I_x, I_y)$ where I_x is the rate of change with respect to the x-axis, and I_y is the rate of change with respect to the y-axis. The gradient captures the rate of change in intensity value as calculated by the central difference formula. Viewing these as the change in energy across a sampled point in the underlying energy function of the image, the dot product of the gradient with itself captures the *inertia* of the energy distribution at the sampled pixel [1].

Explicitly, for pixel, p, the structure tensor, $S(p)$, is given by

$$S(p) = (\nabla I_p)(\nabla I_p)^T \tag{1}$$

which can be written in the expanded form

$$S(p) = \begin{bmatrix} (I_x(p))^2 & I_x(p)I_y(p) \\ I_x(p)I_y(p) & (I_y(p))^2 \end{bmatrix}. \tag{2}$$

As summarized in [10], three cases arise from the structure tensor which is referred to as the *texture pattern*:

- *Case 1:* there is little change across the pixel so all values are small. This results in a structure tensor where both I_x and I_y are small values, and their products are even smaller.
- *Case 2:* there is a strong response in either the horizontal or vertical direction, but not both. This results in a strong response in either $(I_x)^2$ or $(I_y)^2$ and a clear asymmetry exists across the $2x2$ structure tensor. This indicates a unidirectional response, or an *edge* within the image.
- *Case 3:* there is a strong response in both the horizontal *and* vertical direction. This results in large values across the whole tensor. This strong bi-directional response correlates with a *corner*.

We take the weighted average of the structured tensors within a window around the pixel p and arrive at a robust estimation of the energy change across p. To do this, we define a square window of dimension k, and a weight function, $w(p+k)$, which is a function of the neighborhood pixel $p+k$. The choice of weight function is between an even average over all tensor values in the neighborhood or by weighting the value relative to its distance from the center pixel. This is equivalent to convolving the neighborhood with a filter, K, which is either a box or Gaussian filter.

$$S_\mu(p) = K * \sum_{i=-k+x}^{k+x} \sum_{j=-k+y}^{k+y} S(p(i,j)) \tag{3}$$

In [27], and several prominent sources, the latter method is advocated and is used in our implementation.

From the resulting structure tensor, we calculate its eigenvalues. As in [21] the authors of the LTMA method advocate for thresholding on the equation

$$\min(\lambda_1, \lambda_2) > \lambda \tag{4}$$

where the threshold, λ, is determined relative to the average noise level in the image. This ensures that only points which have sufficient feature qualities are selected as viable points to track.

B. Results

As seen in figure 1, there is a parameter space over which the threshold, λ, can be searched. In the figure, several images were applied between 0.0, which is no thresholding, and 0.9, which results in a nearly black image.

One immediate feature seen in all images is their similarity. Were the gradient alone used, it is clear that many points would still have a response and generate flow values in the next step in the process. As soon as the structure tensor thresholding is applied, the number of detail lines drops dramatically. This method has focused in on prominent lines within the feature space without requiring reconstruction or hysteresis thresholding.

However, while the detail has been reduced, there is also an introduced risk of data loss. In figure 2, we see our chosen optimal lambda, $\lambda = 0.2$, applied to three frames from different image sequences.

While the thresholding manages to capture prominent features in the example with the butler (at bottom), the other two

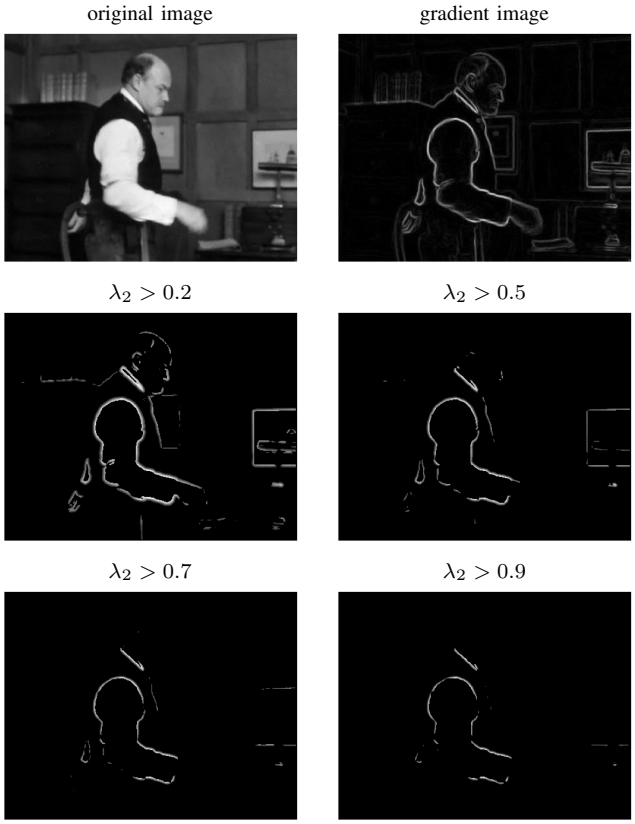


Fig. 1. Results of eigen-thresholding at various levels on a frame taken from *Miss Marple*, drawn from BMS-26 [15].

Fig. 2. Results of eigen-thresholding on several frames taken from *Miss Marple*, drawn from BMS-26 [15].

have flaws at this value of λ. In the top-most image, of a man seated at his desk, we see that the books behind the man maintain prominence in the scene, while his shoulder lacks a defining edge. Even more concerning is in the middle image where the woman in the foreground has vanished entirely from the image, despite being quite prominent in the shot. Meanwhile, background details, like the trim on the house siding, is very strong.

C. Discussion

The results of thresholding by equation 4 cause any pixels with at least one eigenvalue less than the threshold, λ, to be set to zero. A natural question, which is unanswered in the available literature on the LTMA algorithm, is which value to place in the output image for the values which do succeed the thresholding.

One option, and the one in our implementation, is to simply treat the thresholding as a mask and leave the original intensity value at the location. Upon digging into the source code of the original implementation, they opted for an object-oriented approach. For each new image in the sequence, they collect a vector of available points to track based on the thresholding algorithm. This vector is then passed to the tracking function which performs a sparse tracking over only those features in the corner collection. This accumulates into the track object based on piece-wise linearity [25].

We have applied a threshold of $\lambda = 0.2$ to these images.[1]

The authors suggest further subsampling the eigenimage by a factor in the range of [4,16] [15]. This down-sampling was accomplished in our implementation by setting the stride length of the function which checks each region of the image for current tracking to the sub-sampling value. Within the dimensions of a window surrounding the point being checked, we impose the constraint that the overall flow within the region must be greater than the cardinality of the window. Logically, we require at least a response where the average pixel moves at least one pixel in order for the region to warrant tracking.

Two flaws are immediately apparent in this implementation and should be addressed in further iterations of the code. First, the use of a blanket average over a patch in the image is easily subject to error in natural image sequences. Even a strong noise response in the flow calculation could cause an average response to seem significant. Second, the stride method for down-sampling does nothing to manipulate the actual image data and still relies upon a full-resolution image.A coarse-to-fine approach (or vice versa) may be a better approach in the future for generalizing image content rather than sampling in a uniform grid across the image.

[1]The value of all intensities at the thresholding step is taken on normalized images where $I_p \in \{0 \leq p \leq 1\}$.

IV. Point Tracking

A. Background

Once the image has been reduced to only trackable points, their movement between frames can be estimated using optical flow. The published research does not address the practical implementation of these tracks, merely stating that the goal is to construct a history of each tracked point across the sequence.

One common data structure used in medical imaging for point tracking is to construct a 3-dimensional tensor with a depth equal to the number of sequential frames plus one. The first frame, the key frame, is initialized as the origin point for each underlying pixel in the first frame of the sequence. In subsequent frames, the pixel's location is tracked relative to the previous frame. The result is a piecewise-linear chain of vectors which traverse the image plane.

While an attempt was made to implement this method, the technique makes the assumption that pixels only move within the domain of the image frame. In natural image sequences, objects may enter or exit the frame at any time, requiring the termination and initialization of tracks throughout the sequence. Upon checking [25], we were confirmed that an object-oriented approach was the appropriate solution to the problem.

We define a class of Track objects which collect a history of pixel coordinates over their lifespan. These are kept in an array of trajectories that grows over the length of the sequence as new objects enter the scene or static objects begin to move.

In early tests, it was not irregular for upwards of 90,000 tracks to be initialized over the course of a 10 frame sequence. These tracks, and their subsequent manipulations, form the core of the computational expense of the program and so the regulation of their relative explosion is of critical concern with the structure of the code.[2]

The above point about these tracks gives some context to the claim of the authors that tracking points from frame to frame could theoretically be done on a per-pixel basis with a dense enough optical flow estimator. They claim that by regularizing based on the structure tensor thresholding in IV, we will be able to densely estimate flow while avoiding intractability.

Optical flow provides the estimated displacement between frames for each track. For their flow technique, the authors explicitly favour Large Displacement Optical Flow (LDOF), first proposed in [2], as a variational flow estimation technique looking to improve upon known weaknesses in other popular flow trackers [15]. They do state that any suitably dense technique would work [3].

We have chosen the Farnebäck flow tracker, [9], for the reason that it is conveniently available via the OpenCV Python module and was developed with similar aims as the LDOF algorithm. While we will not go into the mathematics in depth, [9] estimates optical flow by approximating a polynomial equation which produces the local neighborhood around a given pixel. This is in contrast to the energy minimiaztion method used in the other two most popular methods, [14] and [11]. A complete introduction to the Farnebäck estimation method can be found at [8].

Much research has been done on the topic of track continuity, specifically concerning occlusion and disocclusion. One strength of the LTMA method is that such track repair is not necessary [3]. This is because if something moves in a certain way before it is occluded, it will move the same way after disocclusion. As the points will ultimately be grouped on these motion affinities, points should be clustered appropriately. We still must take care to detect when occlusion occurs, however, as tracking must cease when the trajectories become unreliable [15].

[2]In truth, we are still experimenting with how to best control their generation and would have found possible methods if more time was available.

V. Occlusion Detection

A. Background

The paper [15] states "tracking has to be stopped as soon as a point gets occluded. This is very important, as otherwise the point trajectory will share the motion of two different objects". The author proposed 2 main steps to decide whether a point should continue to be tracked:

- Compute inverse optical flow.
- Verify inverse and forward flow at each point with a tolerance interval.

As we try to unpack these 2 main steps, we discover that we cannot compute the inverse optical flow by simply reverse frame or flip the sign of the forward flow because of occlusion and disocclusion issue. Also we start in a grid but combine with forward flow, we won't end in a grid structure. The paper [20] introduce a technique for us to compute the inverse flow by using bilinear interpolation.

B. Inverse Optical Flow

1) Compute the forward and the backward (reverse frame) with Farnerback optical flow.
2) Compute the magnitude of the forward flow and the backward flow (reverse frame).
3) At the first frame t, each point will have the forward vector (u, v) that tells where it end up in second frame t+1. However, many points will end up off the grid at subpixel. We find the 4 neighboring points at subpixel position and compute the area from subpixel to the 4 neighboring points (bilinear inpterpolation). Total we have 4 area at every subpixel.
4) We do a condition check at every neighboring point if their areas is greater or equals to 0.25 and the magnitude of the forward flow is greater than the magnitude of the backward flow. We replace the backward flow at that neighboring point to be the flipped sign of the forward flow rather than keeping the backward flow compute by reverse frame.

C. Verification with tolerance interval on magnitude

Once the inverse optical flow is obtained, we proceed to checking the forward and inverse flow of a point whether they

are within a tolerance interval. The [2] provides the details algorithm of the interval:

$$|w + \hat{w}|^2 < 0.01 * (|w|^2 + |\hat{w}|^2) + 0.5 \tag{5}$$

where

$\|\cdot\|$	indicates the Euclidean norm.
w	is the forward flow vector.
$\hat{w}$	is the inverse flow vector.
$\|w + \hat{w}\|^2$	is the difference between forward and inverse flow.

(5) gives the tolerable amount of variance between the forward and backward flow vectors. If difference between forward and backward flow is less than the tolerable amount, we allow the point to be tracked.

It should be noted that the point at frame t + 1 will be at a sub-pixel location. We have to interpolate the point to obtain the inverse flow vector. This won't affect our result because we already compute inverse flow by bilinear interpolation.

This issue arises again when we assign the pixel location at frame $t+1$ to the track history. As the destination is sub-pixel, it is not defined in the research whether to keep this floating point location or to place the track back "on the grid." We have opted for the latter case, as it is otherwise difficult to establish which points are or are not tracked with every iteration of the algorithm. This is a subject of worth exploration in future investigation.

D. Boundary verfication

In [3], they also advocate that tracking stop for points which lie along motion boundaries. Such points have volatile fluctuations which lead to erratic behavior. The define the constraint on volatility by the equation:

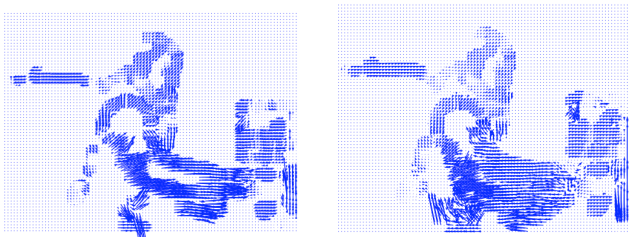
$$|\nabla u|^2 + |\nabla v|^2 > \alpha * |w|^2 + \beta \tag{6}$$

where

∇u and ∇v	are the gradients with respect to the u and v components of the flow field.
$\|\cdot\|^2$	is the squared magnitude of the vector it encloses.
w	is forward flow vector.
α	is a weight parameter.
β	is a biasing parameter.

∇u and ∇v are, at first, tricky terms to conceptualize. Recall that u and v, at a given pixel, represent the amount of displacement relative to time. This makes them equivalent to *velocity* values. The derivative value of velocity is tantamount to *acceleration*. Therefore, the gradient, ∇u, captures the rate of acceleration of the pixel with respect to both the x- and y-axes.

In [3], the parameters α and β are given the values of $\alpha = 0.01$ and $\beta = 0.002$. This imposes a harsh penalty on any pixel which exceeds the "speed limit" of of the tracking algorithm.



Forward flow Inverse flow

Fig. 3. Compare forward and inverse flow by direction of vector from *Miss Marple*, drawn from BMS-26 [15].

Forward flow Inverse flow

Fig. 4. Compare forward and inverse flow by magnitude of vector from *Miss Marple*, drawn from BMS-26 [15].

E. Results

Figures 3 and 4 show the results of forward flow in comparison to inverse flow. We see that both estimations approximate the structure of the moving object and, notably, the inverse is almost identical to the forward flow.

By picking a point and following the forward vector, we will end up at a subpixel location in the next frame $t + 1$. By interpolating on the flow values surrounding this point, we obtain an accurate the inverse vector. After we check the difference in magnitude of forward and inverse flow vectors to ensure that they fall within tolerance equations 5 and 6.

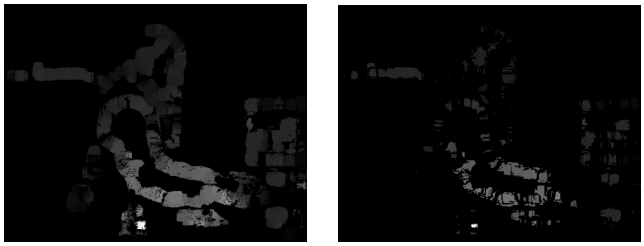
The result of the surviving vectors is shown in figure 5. On the left is the magnitude, displayed as intensity, of all points which will be allowed to track into the next frame. Compare this with the forward flow magnitude in 4. The result has fewer points and the shape along edges of the object is less defined, indicating fewer tracked trajectories. Some dark areas start to appear in between strong magnitude area. With extra checking with 6, we see even fewer permissible tracks along the boundary. This is shown in the figure on the right in figure 5 on the right.

These results are as expected. Few points are allowed to track between frames, but we have ensured that these maintain the structure of the moving object. Points that have become occluded and that lie along volatile boundaries are removed.

VI. Keeping track of trajectories

A. Background

To allow for a single trajectory to be tracked throughout the course of multiple frames, we construct a Track object that represents a trajectory throughout the entirety of the sequence.



Magnitude interval | Magnitude interval with boundary

Fig. 5. Compare final result of set of point with their forward vector with magnitude difference interval tolerance and boundary tolerance. The result is display in form of magnitude of point that allowed to track from *Miss Marple*, drawn from BMS-26 [15].

A Track object is initialized when the described criteria for point-tracking is met; track objects are never destroyed, but may be marked as 'dead' if they fail to meet the point-tracking criteria at any point.

B. The Track Object

```
class Track:
    def __init__(self, y, x, frame):
        self.origin  = (y,x,frame)
        self.history = []
        self.curr_position = self.origin
        self.live  = True
        self.label = -1
        self.trajectory_ddt = []
```

Two essential pieces of information the Track object carries are the history of (y, x) coordinates that a Track has been found at and the smoothed trajectory that it took to reach those coordinates.

`List<point> history`

- A list of $(y, x, frame)$ tuples carrying the (y, x) coordinate through which the Track has passed and the frame at which it did.

`Tuple trajectory_ddt`

- A list of forward $(\frac{dx}{dt}, \frac{dy}{dt}, frame)$ values, averaged over a number of frames. This window is a hyperparameter, and if the number of available frames is smaller than the specified window only the available frames are used.

VII. The Affinity Matrix

A. Background

As stated in II, we generally assume that objects that move together in a similar fashion belong to the same unified group [3][15]. Even more reliable, though, is the idea that elements that are physically part of the same object ought to move together.

However, in the context of object segmentation from video we cannot immediately assume that elements that move in a similar direction belong to the same object. The following problems arise: First, two elements in close proximity may move in concert, causing them to appear as one object to our algorithm.

An example of this would be two distinct cars driving on a highway at perfectly synchronized speed. Another example would be a top-down View of two trains traveling at the same speed on parallel tracks. They will similarly be mistakenly classified as the same object.

As it is presumed that the objects live in an imperfect world, there will be some point at which they deviate from their perfectly aligned courses. We develop an algorithm that identifies this moment of greatest divergence and, upon this divergence, conclude that these are different objects.

B. Algorithm

The Brox-Ochs-Malik method[15], which we have implemented, takes into account both the similarity in movement patterns across the image sequence and spatial proximity of elements. Importantly, the similarity in movement patterns between two tracks is calculated at the point where the movement between the track is *most dissimilar*. At this point, we sample the difference in movement as well as the spatial difference between the two tracks.

Together, these two components define a relation between two tracks which is described as the 'distance' between two time-concurrent trajectories. Distance values range from [0,inf), where a 0 value indicates that the pair of trajectories have no spatio-trajectory distance with one another and a large value indicates extreme dissimilarity.

Pairwise distance is calculated as the maximum distance between the trajectory vectors throughout time, multiplied by the mean spatial Euclidean distance between the two Tracks. Pairwise distance is only calculated for Tracks that exist concurrently in the image sequence.

$$d_t^2(A, B) = d_{sp}(A, B)d_{tr}(A, B) \tag{7}$$

Trajectory distance $d_{tr}(A, B)$ is calculated as the squared distance between two trajectory vectors at the frame *t* of greatest difference.

$$d_{tr}(A, B) = \max_{t \in A \cap B} (u_t^A - u_t^B)^2 + (v_t^A - v_t^B)^2 \tag{8}$$

Spatial distance $d_{tr}(A, B)$ is calculated the Euclidean distance between two points at the time of maximum trajectory distance.

$$d_{sp}(A, B) = \sqrt{(x_t^A - x_t^B)^2 + (y_t^A - y_t^B)^2} \tag{9}$$

Distance $d_t^2(A, B)$ is converted to Affinity in the final step of calculation by application of a standard exponential function and a hyperparameter λ.

$$a(A, B) = \exp(-\lambda d_t^2(A, B)) \tag{10}$$

Affinity is thus a normalized inverse of our calculated 'distance' - it expresses the spatial closeness and propensity for joint movement between two tracks. We then assemble these

affinities into symmetric matrix A - the 'affinity' matrix. The affinity matrix holds the pairwise relationship between a given Track and every other Track in the set of tracked trajectories. Affinity between non-concurrent Tracks is 0, and the diagonal of A is 1; the identity of every Track.

VIII. Segmentation with Spectral Clustering

Once the affinity matrix is obtained, we can proceed with spectral clustering. The result of clustering should be a fully segmented image. Spectral clustering is unique in its effectiveness to cluster points within a complex shape. Spectral clustering uses eigenvalue and eigenvectors as a basis for clustering, unlike clustering based on location or intensity. As discussed in section VII, spectral clustering treats the input affinity matrix as a graph where each entry defines an edge whose weight is the *affinity* between two nodes. On the right are the images that showcase the performance of the spectral clustering in comparison to k-means clustering. The circle data is generated using $sklearn.dataset, make\_circles$ function. As we can clearly see the pure location based k-means failed to tell the difference between two connected groups (components) in the image. In contrast, spectral clustering is based on discrepancy in connection between connected points to segment data. In this case it did successfully segmented the whole image into two cluster groups, one outside and one inside.

There are three major steps to implement this clustering method: constructing the graph Lapacian, computing the eigenvalues and vectors of that graph, and finding small eigenvalues.

A. Generating the Graph Laplacian

The graph Laplacian is generated from two matrices, the adjacency matrix and the degree matrix.

The adjacency matrix is obtained by imposing a threshold on the affinity matrix. If the Gaussian distance between two nodes in affinity matrix is greater than a chosen value, the adjacency value is 0, 1 otherwise. The degree matrix is a diagonal matrix showing the degree of each node. It is generated by taking the sum of each row of the adjacency matrix and placing it in the appropriate position. Below are the example of the adjacency matrix and the degree matrix, generated from a data set of the following coordinates:

$$\begin{Bmatrix} 1,3 & 2,1 & 1,1 \\ 3,1 & 7,8 & 9,8 \\ 8,9 & 8,7 & 13,14 \\ 14,16 & 15,16 & 14,15 \end{Bmatrix}$$

Subtracting the adjacency matrix from the degree matrix gives the graph Laplacian. The adjacency matrix, degree matrix and laplacian that calculated from the sample data we use are shown in the next page.

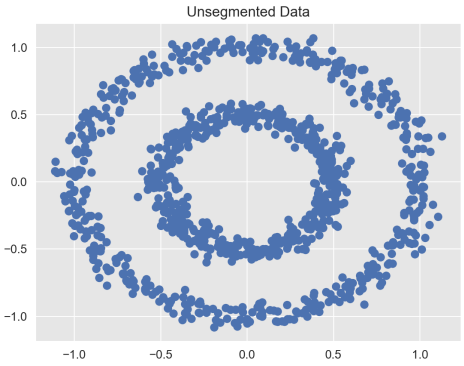


Fig. 6. Unsegmented Image

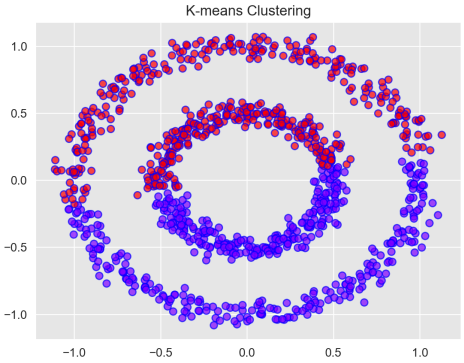


Fig. 7. Image after k-Means

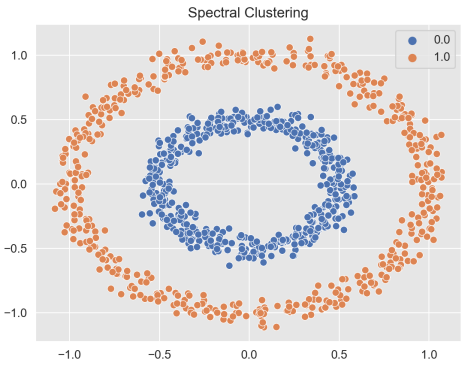


Fig. 8. Image after Spectral Clustering

B. Computing eigenvalues and eigenvectors

Next we generate the eigenvalues and eigenvectors of the graph Laplacian. This can be quickly obtained using built in functions in Python's numpy module.

Spectral clustering is assumes that is is applied to connected coordinates points. An eigenvalue of 0 means a "gap" exists between two data points. The broken connection serves as the basis for clustering. This leads to the unique property of the graph Laplacian that, if the graph(W) has a total of k eigenvalues equal to 0, then that would mean the graph can be

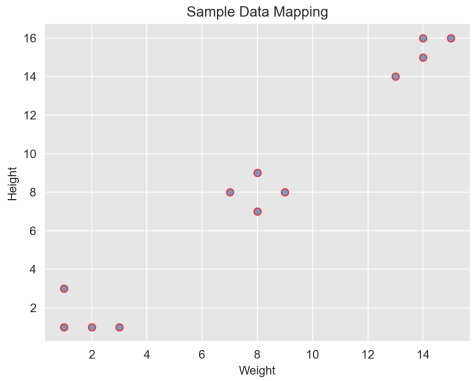


Fig. 9. Sample Data Mapping

```
Adjacency Matrix:
[[1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 1 1 1 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1]]
```

Adjacency Matrix

```
Degree Matrix:
[[4 0 0 0 0 0 0 0 0 0 0 0]
 [0 4 0 0 0 0 0 0 0 0 0 0]
 [0 0 4 0 0 0 0 0 0 0 0 0]
 [0 0 0 4 0 0 0 0 0 0 0 0]
 [0 0 0 0 4 0 0 0 0 0 0 0]
 [0 0 0 0 0 4 0 0 0 0 0 0]
 [0 0 0 0 0 0 4 0 0 0 0 0]
 [0 0 0 0 0 0 0 4 0 0 0 0]
 [0 0 0 0 0 0 0 0 4 0 0 0]
 [0 0 0 0 0 0 0 0 0 4 0 0]
 [0 0 0 0 0 0 0 0 0 0 4 0]
 [0 0 0 0 0 0 0 0 0 0 0 4]]
```

Degree Matrix

divided into k components or clusters. Within each component there will be all data points whose eigenvector belongs to one specific eigenvalue 0.

This principle is the core mechanic of the spectral clustering method. Whenever an eigenvalue is 0, all the eigenvectors connected to it belong to that subgraph, or cluster. If there are k connected components related to that eigenvalue, then all k belong to the same cluster.

So as a summary, the number of 0 eigenvalues means how many components or clusters we have for the graph. Graphs below are the eigenvalues and eigenvectors that we generated from the laplacian matrix we have before. As we can see the whole second column tends to have value of 0, not exact but really close. As this indicates we should have three clusters in the graph. And in our case from the data plot shown before that maps all the data point clearly expresses the same message.

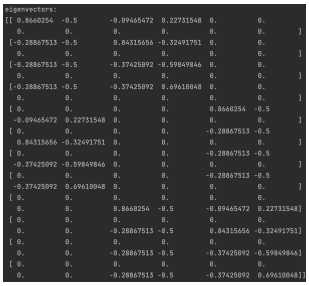
C. Find the small eigenvalues

By sorting the eigenvectors over the image and finding these places where the eigenvalues are 0, we form the final clusters. Within each cluster, all points will be considered as connected within the cluster.

Once done, we can apply k-Means Clustering method to the organized matrix returned from sorted out 0 eigenvalues in this step to obtain the cluster index for each data value

```
Laplacian Matrix:
[[ 3 -1 -1 -1  0  0  0  0  0  0  0  0]
 [-1  3 -1 -1  0  0  0  0  0  0  0  0]
 [-1 -1  3 -1  0  0  0  0  0  0  0  0]
 [-1 -1 -1  3  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  3 -1 -1 -1  0  0  0  0]
 [ 0  0  0  0 -1  3 -1 -1  0  0  0  0]
 [ 0  0  0  0 -1 -1  3 -1  0  0  0  0]
 [ 0  0  0  0 -1 -1 -1  3  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  3 -1 -1 -1]
 [ 0  0  0  0  0  0  0  0 -1  3 -1 -1]
 [ 0  0  0  0  0  0  0  0 -1 -1  3 -1]
 [ 0  0  0  0  0  0  0  0 -1 -1 -1  3]]
```

Graph Laplacian



Eigenvectors of the Lapacian

```
eigenvalues:
[ 4.00000000e+00 -1.11022302e-16  4.00000000e+00  4.00000000e+00
  4.00000000e+00 -1.11022302e-16  4.00000000e+00  4.00000000e+00
  4.00000000e+00 -1.11022302e-16  4.00000000e+00  4.00000000e+00]
```

Eigenvalues of the Laplacian

Fig. 10. Eigendecomposition of the graph Laplacian

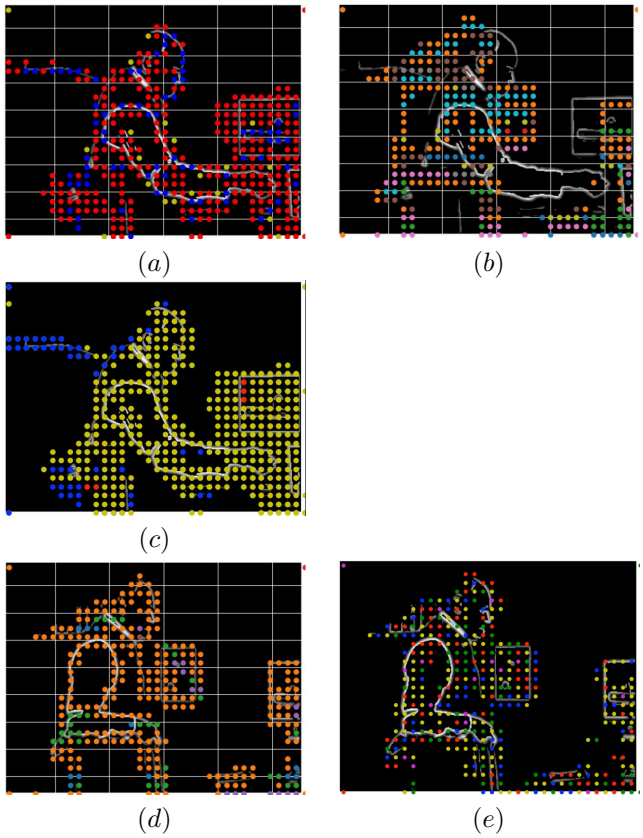


Fig. 11. Attempts at segmentation based on clustering from our implementation.

that we used to generated affinity matrix. Thus complete the spectral clustering steps. After obtaining the "tags", the cluster each data belongs to. We assign them to our Track objects.

IX. Discussion of Results

Figure 11 shows the results of several attempts to find optimal parameters for our incarnation of the LTMA segmentation algorithm. Before too much success is touted on their segmentation, keep in mind that we were clustering based on

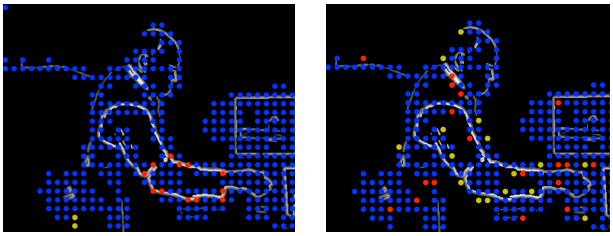


Fig. 12. Two promising results from our implementation.

a severely thresholded image. This, in effect, proved to be half the battle in reducing the number of points. That said, we do see that points are appropriately initialized and, not pictured here, do track appropriately through the sequence.

As noted in previous sections, there were a number of implementation issues which we discovered on the way to this final product which may be contributing to the outcome of the pipeline. That said, there are a number of telltale signs of possible success which give us confidence in our implementation.

One major failing of our implementation is in distinguishing between the man in the image and the picture frame on the wall behind him. As he moves through the space, he crosses from left to right and the photo crosses from right to left. These should have opposite relationships in the eyes of our clustering algorithm. However, no matter the length of time that we allowed them to run, they segmented as similar or same clusters.

The upper left image, (a), shows that the boat within the image frame was segmented differently than the frame around it. This also occurs to a lesser degree in (b). This is interesting as it implies that the algorithm found a significant difference between objects despite their physical placement in the scene.

Implementations (b) and (c) show significant responses of same-cluster points within the torso of the man.[3] This also is shown mildly in (c) where the leading edge of the picture frame and the man's hand are, appropriately, tracked as moving oppositely from the man's general direction. However, the abundance of tracks moving to the right make this a weak claim.

The most promising results of our work are shown in figure 12, where the red dots, which lie upon ideal tracking corners, appropriately segment the man's arm. These clustering attempts were done using only two frames, rather than a multi-frame sequence. This implies that we may have benefited from more time debugging at a low-frame count and then iterating upwards to the full sequence. While there still exist issues in segmenting the picture frame, at least one cluster in each of the images in 12 seems to approximate a segmentation.

Figure 13 shows the results of the LTMA clustering algorithm from the original implementation when applied to the *Miss Marple* sequence seen in figure 2. As can be seen, their approach cleanly segments the butler as one class and the

[3]Due to an indexing error, the image underneath (b) is not the same as the tracks layered on top.

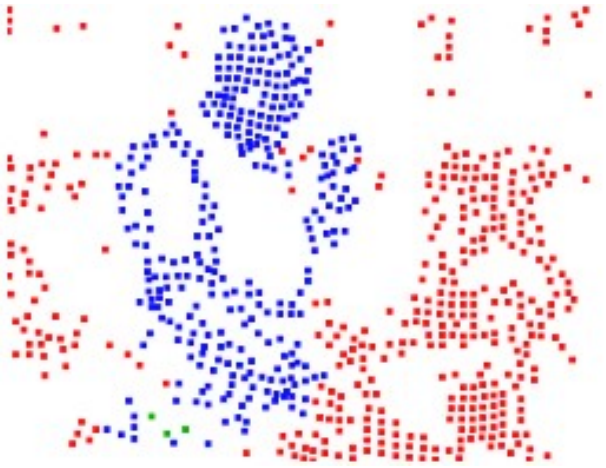


Fig. 13. Results from [15] on the sequence features in figure 1.

background as another. There are a few mis-labeled clusters which exist on the man's hip in green. The phone, which is in the man's hand, is not segmented as an independent object in this incarnation of their results.

In comparison, our results constitute a somewhat successful replication study. While our efforts were valiant in spirit, our final classification is admittedly lackluster. However, we were able to successfully (albeit as independent components) implement eigen-thresholding, architect an Object-Oriented solution to storing trajectory information, calculate affinities between trajectories, and implement spectral clustering.

That being said, we did not get around to implementing a number of discussed features of the LTMA pipeline. Among these components it a regularization technique applied to the trajectory components at the time of $d_t^2(A, B)$ calculation. Our target implementation [15] references the division of the squared difference in trajectories by an estimation of average flow within a neighborhood of the tracked point. We predict that this measure would reduce the 'chaos' in our prediction and would de-noise our classification. We believe that we would have more leeway to adjust hyperparameters if we had more confidence that we were not clustering noise.

X. Conclusion

Ultimately, we found that the LTMA algorithm [15] is an impressive implementation of a non-learning and domain-specific algorithm that deserves further attempts at replication.

References

[1] Josef Bigun and Gosta H. Granlund. "OPTIMAL ORIENTATION DETECTION OF LINEAR SYMMETRY." In: 1987, pp. 433–438. ISBN: 081860777X.

[2] Thomas Brox, Christoph Bregler, and Jitendra Malik. "Large displacement optical flow". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*. Vol. 2009 IEEE. 2009, pp. 41–48. ISBN: 9781424439935. DOI: 10.1109/CVPRW.2009.5206697.

[3] Thomas Brox and Jitendra Malik. "Object segmentation by long term analysis of point trajectories". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6315 LNCS. PART 5. 2010, pp. 282–295. ISBN: 3642155545. DOI: 10.1007/978-3-642-15555-0{_}21.

[4] Daniel J Butler, Jonas Wulff, Garrett B Stanley, et al. "A naturalistic open source movie for optical flow evaluation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7577 LNCS. PART 6. 2012, pp. 611–625. ISBN: 9783642337826. DOI: 10.1007/978-3-642-33783-3{_}44.

[5] Lin Chuan-en. *Introduction to Motion Estimation with Optical Flow*. 2019. URL: https://nanonets.com/blog/optical-flow/.

[6] *Computer Vision Group, Universität Freiburg*. 2017. URL: https://lmb.informatik.uni-freiburg.de/index.php.

[7] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, et al. "FlowNet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2015 Inter. 2015, pp. 2758–2766. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.316.

[8] Gunnar Farnebäck. "Polynomial expansion for orientation and motion estimation". PhD thesis. Linköpinos Universitet, 2002, p. 196. ISBN: 9173734756. URL: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.2686.

[9] Gunnar Farnebäck. "Two-frame motion estimation based on polynomial expansion". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2749 (2003), pp. 363–370. ISSN: 16113349. DOI: 10.1007/3-540-45103-x{_}50. URL: http://www.isy.liu.se/cvl/.

[10] Chris Harris and Mike Stephens. "A Combined Corner and Edge Detector". In: *Alvey Vision Conference*. 1988, pp. 147–152. DOI: 10.5244/c.2.23.

[11] B. K.P. Horn and Brian G Schunck. "Determining optical flow." In: *Computer vision* (1981), pp. 185–203. DOI: 10.7551/mitpress/1413.003.0014.

[12] Kurt Koffka. "Perception: An introduction to the Gestalt-theorie". In: *Psychological Bulletin* 19.10 (1922), pp. 531–585. DOI: 10.1037/h0072422.

[13] Tsung Yi Lin, Michael Maire, Serge Belongie, et al. "Microsoft COCO: Common objects in context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8693 LNCS. PART 5. 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1{_}48.

[14] Bruce D Lucas and Takeo Kanade. "ITERATIVE IMAGE REGISTRATION TECHNIQUE WITH AN APPLICATION TO STEREO VISION." In: vol. 2. 1981, pp. 674–679.

[15] Peter Ochs, Jitendra Malik, and Thomas Brox. "Segmentation of moving objects by long term video analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.3 (2011), pp. 500–513.

[16] Yuri Ostrovsky, Ethan Meyers, Suma Ganesh, et al. "Visual parsing after recovery from blindness". In: *Psychological Science* 20.12 (2009), pp. 1484–1491. ISSN: 09567976. DOI: 10.1111/j.1467-9280.2009.02471.x.

[17] Anestis Papazoglou and Vittorio Ferrari. "Fast object segmentation in unconstrained video". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1777–1784. ISBN: 9781479928392. DOI: 10.1109/ICCV.2013.223. URL: https://doi.org/10.1109/ICCV.2013.223.

[18] Esteban Real, Jonathon Shlens, Stefano Mazzocchi, et al. *YouTube-BoundingBoxes: A large high-precision human-annotated data set for object detection in video*. 2017. URL: https://research.google.com/youtube-bb..

[19] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2021, p. 1115.

[20] Javier Sánchez. *Computing Inverse Optical Flow*. Tech. rep. Centro de Tecnologías de la Imagen, Universidad de Las Palmas de Gran Canaria, 2013, p. 27. URL: https://www.researchgate.net/publication/258547557_Computing_Inverse_Optical_Flow.

[21] Jianbo Shi and Carlo Tomasi. "Good features to track". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. ISBN: 0818658274. DOI: 10.1109/cvpr.1994.323794. URL: http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf.

[22] *Software/Datasets — The Computer Vision Group*. 2017. URL: https://lmb.informatik.uni-freiburg.de/resources/software.php.

[23] Elizabeth S. Spelke. "Principles of Object Perception". In: *Cognitive Science* 14.1 (Jan. 1990), pp. 29–56. DOI: 10.1207/s15516709cog1401{_}3.

[24] Narayanan Sundaram, Thomas Brox, and Kurt Keutzer. "Dense point trajectories by GPU-accelerated large displacement optical flow". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6311 LNCS. PART 1. 2010, pp. 438–451. ISBN: 3642155480. DOI: 10.1007/978-3-642-15549-9{_}32.

[25] Narayanan Sundaram, Thomas Brox, and Kurt Keutzer. */eccv2010_trackingGPU/tracking.cpp*. Freiburg, Germany, 2011. URL: https://lmb.informatik.uni-freiburg.de/people/brox/pub/sundaram_eccv10.pdf.

[26] David Tsai, Matthew Flagg, and James Rehg. *Motion Coherent Tracking with Multi-label MRF Optimazion*. 2010.

[27] Heng Wang, Alexander Kläser, Cordelia Schmid, et al. "Action Recognition by Dense Trajectories". In: *IEEE*

Conference on Computer Vision & Pattern Recognition. Vol. 10. 1109. 2011, pp. 3169–3176. DOI: 10.16182/j.issn1004731x.joss.201709023. URL: https://hal.inria.fr/inria-00583818.