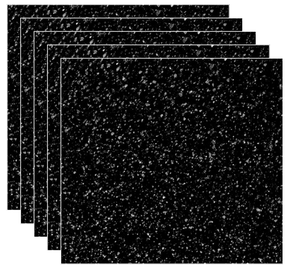
Particle Tracking

Particle Pals

Images/video



BackgroundImage

PredictiveTracker

Particle tracks

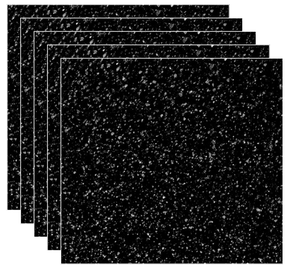
ParticleFinder

FindParticles

FindRegions

PredictiveTracker

Images/video



BackgroundImage

PredictiveTracker

Particle tracks

ParticleFinder

FindParticles

FindRegions

Input Parameters

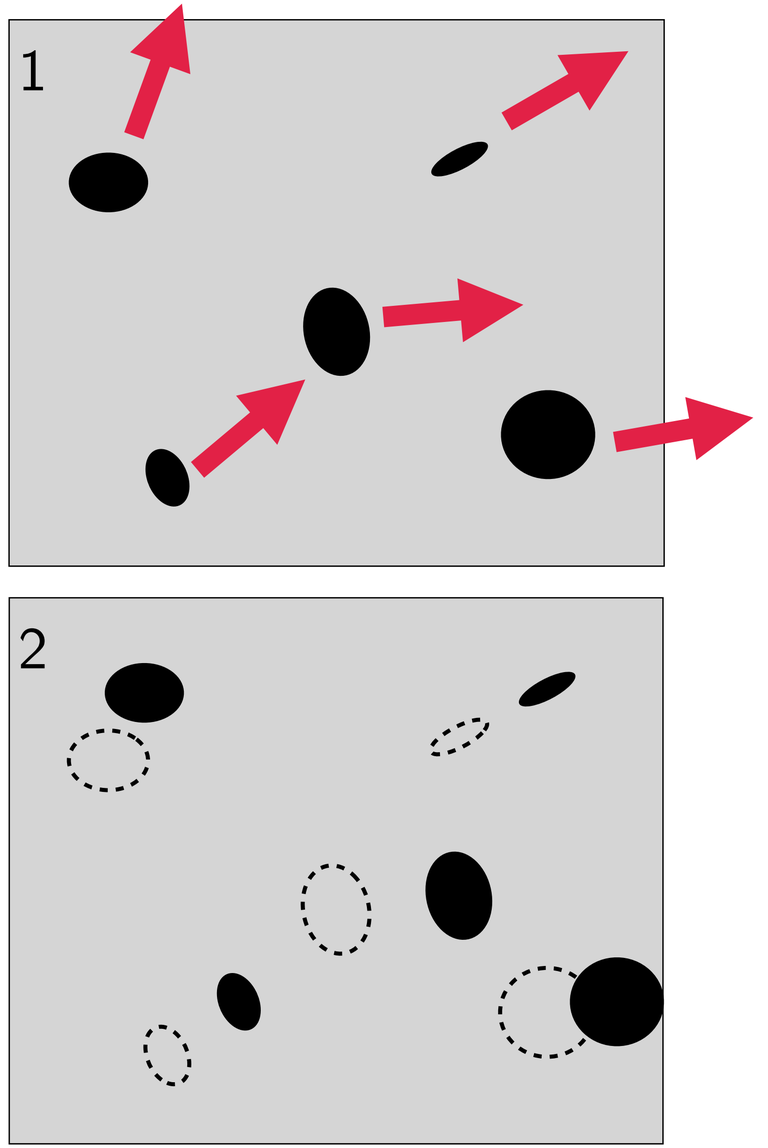
- **`inputnames`:** Specifies the input movie file or image stack.
- **`threshold`:** Sets the brightness threshold for particle identification.
- **`max_disp`:** Maximum displacement allowed for particle tracking.
- **`bground_name`:** Name of the file containing the background image.
- **`minarea`:** Minimum area for particle identification (default is 1 pixel).
- **`invert`:** Determines whether to track bright (0), dark (1), or any contrast (-1) particles.
- **`noisy`:** Controls the plotting and visualization options.

Output

- **`vtracks`:** Structure array containing tracked particle information (length, coordinates, times, velocities).
- **`ntracks`:** Total number of identified tracks.
- **`meanlength`:** Mean length of particle tracks.
- **`rmslength`:** Root mean square length of particle tracks.

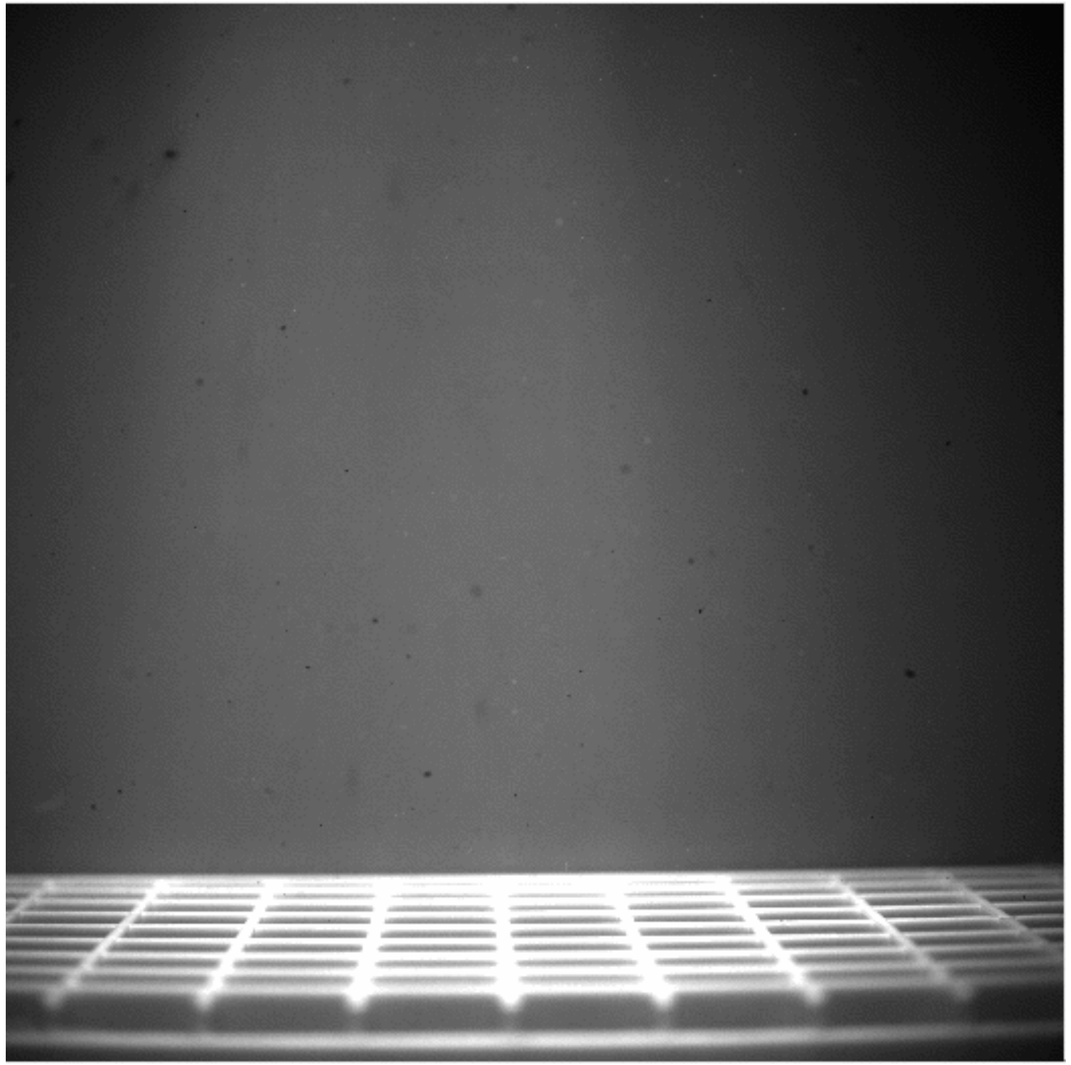
Algorithm steps

- Particle identification is done using the `ParticleFinder` function.
- Particle tracks are initialized based on the identified particles in the first frame.
- Tracks are extended by predicting the next position using kinematic information.
- Tracks are matched with particles in the subsequent frames.
- New tracks are initiated for unmatched particles.
- Tracks are pruned based on a minimum length criterion.
- Velocities are computed for the final tracks.



Visualization

- If the `noisy` parameter is set, the function can plot and save frames with overlaid particle tracks.
- The function supports various movie formats, including image stacks, uncompressed AVI, TIFF, GIF, etc.



PredictiveTracker structure

1. Inputs

Previous slide

2. Set Defaults

Sets default values for some parameters and defines additional constants.

3. Parse Inputs

Checks if the required input parameters are provided; if not, it raises an error. It then assigns default values to any missing optional parameters.

4. Particle Finding

The function uses the `ParticleFinder` function to identify particles in each frame of the input movie.

- It extracts particle positions (`x` and `y`),
- times (`t`),
- and orientation angles (`ang`).

5. Set Up Track Structure

The code initializes a structure array called tracks to store information about particle tracks. The structure includes fields such as

- Particle ID
- frames it appears
- length (`len`),
- x and y coordinates (`X` and `Y`),
- times (`T`),
- and orientation (`Theta` for particles with `minarea` not equal to 1).

6. Loop Over Frames

The function iterates over each frame of the movie and performs the following steps:

Match Tracks with Kinematic Predictions:

- For each active track, it predicts the next position based on kinematic information.
- It calculates the cost (distance) between the predicted positions and particles in the next frame.
- Matches are established, and new tracks are started for unmatched particles.

Keep Track of Active Tracks:

- The function keeps track of active tracks and updates information about the number of particles found, active tracks, and new tracks started.

7. Prune Tracks

The function removes tracks that are too short, based on a minimum length criterion.

8. Compute Velocities

Velocities are computed for each track using a convolution operation with a differentiation kernel.

9. Plotting and Visualization

If the `noisy` parameter is set, the function plots particle tracks over the movie frames. It can save frames as images if specified.

10. Output

The function returns the final tracked particle information, including

- the structure array `vtracks`,
- the total number of tracks (`ntracks`),
- the mean track length (`meanlength`),
- and the root mean square track length (`rmslength`).

Dependencies

The `PredictiveTracker` function relies on the `ParticleFinder` function and, in some cases, the `read_uncompressed_avi` function. Here's a brief overview of these dependencies:

1. **`ParticleFinder` Function:**

   Purpose: This function is used to identify particles in each frame of the input movie.

   Usage: The ParticleFinder function takes various parameters, including the input movie file, threshold for particle identification, frame range, background image, minimum area, invert option, and a flag for visualization.

   Output: The function returns particle positions (`x` and `y`), times (`t`), and orientation angles (`ang`).

   Dependency: The `PredictiveTracker` function calls `ParticleFinder` to perform particle identification.

2. **`read_uncompressed_avi` Function:**

   Purpose: This function is required for reading uncompressed AVI files.

   Usage: It reads the uncompressed AVI file and, if specified, returns movie frames and additional information.

   Dependency: The PredictiveTracker function uses `read_uncompressed_avi` to read frames from AVI files.

   Availability: The code mentions that `read_uncompressed_avi.m` can be downloaded from a specified URL: http://leviathan.eng.yale.edu/software.

Steps in each loop iteration

1. Velocity Estimation:

- The positions of currently active tracks are obtained (`now` and `prior`).

- Velocities are estimated based on the difference between current and prior positions.

2. Future Position Prediction:

- The next position is estimated using kinematics, adding the velocity to the current position.

the code assumes a constant velocity model for particle motion, and it predicts the future position by extrapolating from the current and previous positions using a simple linear kinematic model. This is a common approach in particle tracking when there is no information about forces acting on the particles, and their motion is assumed to be approximately linear between consecutive frames.

3. Matching Tracks with Particles:

- For each active track, the function compares the estimated positions with particles in the next frame (`fr1`).

- Costs (distances) are calculated, and the best match is determined based on the minimum cost.

- Tracks that exceed the maximum displacement (`max_disp`) are excluded from consideration.

- If multiple tracks compete for the same particle, the one with the lower cost is selected.

4. Updating Track Information:

- The function updates the active tracks with the matched particles.

- New tracks are initiated for particles in the current frame that found no match.

- The active track list is updated.

5. Displaying Progress Information:

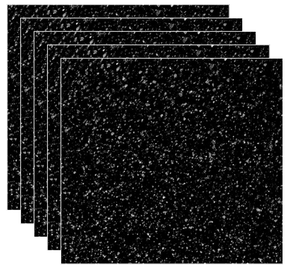
- The function prints information about the processing of the current frame, including the number of particles found, active tracks, new tracks started, tracks that found no match, and the total number of tracks.

6. Iterating to the Next Frame:

- The loop continues to the next frame.

ParticleFinder

Images/video



BackgroundImage

PredictiveTracker

Particle tracks

ParticleFinder

FindParticles

FindRegions

Input Parameters

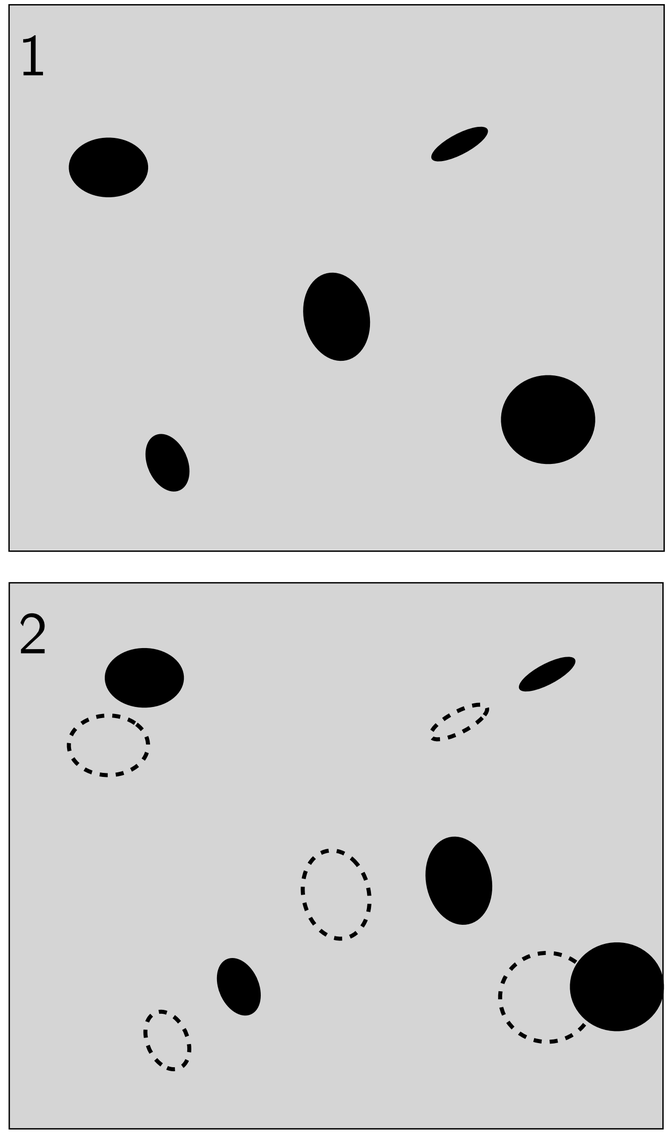
- **`inputnames`:** Specifies the source of the movie (image files, image stack, or AVI file).
- **`threshold`:** Minimum brightness difference to identify a particle.
- **`framerange`:** Range of frames to consider.
- **`outputname`:** Optional name for a binary file to save particle positions.
- **`bground_name`:** File name for the background image.
- **`arealim`:** Particle size limits (either a single value for minimum size or a two-element vector [min, max]).
- **`invert`:** Controls whether to seek bright or dark particles (0 for bright, 1 for dark, -1 for any contrast).
- **`noisy`:** If non-zero, it triggers plotting and optional saving of frames.

Output

- **`x`:** x-coordinates of identified particles.
- **`y`:** y-coordinates of identified particles.
- **`t`:** Time or frame numbers of identified particles.
- **`ang`:** Orientations of particles (angle).

Algorithm steps

1. Determines the type of movie (AVI, image stack, or individual images).
2. Reads the background image.
3. Loops through frames, identifies particles, and stores their positions and orientations.
4. Optionally saves particle positions to a binary file.
5. Optionally plots particle locations and saves frames.



ParticleFinder structure

1. Inputs

Previous slide

2. Set Defaults

Sets default values for optional input parameters (`framerange`, `bground_name`, `arealim`, `invert`, `noisy`). If these parameters are not provided, defaults are used. The function also handles cases where `framerange` is provided as a single value, converting it into a two-element vector.

3. Movie Type Detection and Information Retrieval

Determines the type of movie based on the file extension of the provided `inputnames`. The code checks if it's an AVI file, an image stack, or individual images.

4. Pre-computation and Initialization

Pre-computes logarithms for locating particle centers. Sets up default values and initializes variables like `Nf` (total frame count).

5. Background image reading

Reads the background image from the specified file (`bground_name`). If the file is not found, it creates a blank background.

6. Output file initialization

If an output file name (`outputname`) is provided, initializes and opens the file for writing, writing the total frame count as the header.

7. Main loop through frames

Iterates through each frame of the movie. Reads the frame based on the movie type and performs particle identification.

8. Particle identification and data storage

If particles are identified in the current frame, stores their positions, times, and orientations in arrays (`x`, `y`, `t`, `ang`).

9. Plotting

If the noisy parameter is non-zero, it triggers the plotting and optional saving of frames.

10. Cleanup and output

Cleans up the unused portions of the arrays (x, y, t, ang). Closes the output file if it was opened.

Dependencies

ParticleFinder uses two helper functions, FindParticles and FindRegions. This code is designed to identify and track particles in a series of images, an image stack, or an uncompressed AVI file.

1. **FindParticles Function:**
   - Identifies small particles brighter than their neighbors.
   - Performs sub-pixel localization using Gaussian fits.
   - Returns particle positions in the pos array.
2. **FindRegions Function:**
   - Identifies regions brighter than a threshold with specified area limits.
   - Computes centroids and orientations of the identified regions.
   - Returns region centroids in the pos array and orientations in the ang array.

Steps in each loop iteration

1. Read Frame:

- The loop iterates over each frame (`ii`) in the specified frame range.

- The variable `tt` represents the current time (frame number).

- Depending on the type of movie (`movtype`), the function reads the frame (`im`) from the movie file using the appropriate method (AVI, image stack, or individual images).

2. Image Processing:

- If the frame is in color (3D array), it is converted to grayscale by taking the mean along the third dimension.

- Background subtraction is performed based on the specified `invert` parameter. If `invert == 1`, it seeks dark particles on a light background; if `invert == 0`, it seeks light particles on a dark background; if `invert == -1`, it seeks any contrast.

3. Particle Identification:

- Depending on the `arealim` parameter, the function either calls `FindParticles` or `FindRegions` to identify particles or regions in the frame.

    - If `arealim == 1`, it identifies particles using a sub-pixel accuracy method.

    - If `arealim` is a vector, it identifies regions with areas within the specified limits and estimates their orientations.

4. Store Particle Information:

- If particles or regions are identified (`N > 0`), the function stores their positions, times, and orientations in pre-allocated arrays (`x`, `y`, `t`, `ang`).

- If it's the first frame (`ii == 1`), the arrays are pre-allocated.

`5. Update Memory Location:`

- The variable `memloc` keeps track of the memory location in the arrays where the particle information is stored.

- It is updated based on the number of particles (`N`) identified in the current frame.

6. Display Progress:

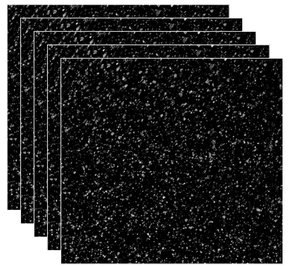
- The function prints information about the number of particles found in the current frame and the total progress.

7. Write to Output File (if applicable):

- If an output file is specified (`writefile == true`), the function writes the number of particles and their positions to the file.

After the loop completes, the function performs cleanup steps, such as removing unused portions of the arrays and closing the output file. If specified, it may also plot frames and save them to disk based on the noisy parameter. The function concludes by displaying a "Done" message.

Images/video



BackgroundImage

PredictiveTracker

Particle tracks

ParticleFinder

FindParticles

FindRegions

FindParticles function

Input Parameters:

`im`: Input image.

`threshold`: Minimum brightness difference to identify a particle.

`logs`: Logarithms of relevant image intensities for faster computation.

Output:

`pos`: Particle positions (x, y coordinates).

Functionality:

1. Local Maxima Identification:

    - The function identifies local maxima in the input image (`im`) that are above the specified threshold.

    - It checks if the identified maxima are brighter than their four nearest neighbors.

2. Sub-Pixel Localization:

    - Sub-pixel localization is achieved by applying a Gaussian fit in each spatial direction.

    - For each identified maximum, the function computes the `x` and `y` coordinates with sub-pixel accuracy.

3. Handling Unreliable Maxima:

    - Removes maxima near the image boundaries to avoid unreliable localization.

4. Logarithmic Intensity Lookup:

- Utilizes pre-computed logarithms (`logs`) of relevant image intensities for faster computation.

5. Output:

    - Returns the particle positions as a two-column array (`pos`), where the first column represents x-coordinates, and the second column represents y-coordinates.

FindRegions function

Input Parameters:

`im`: Input image.

`threshold`: Minimum brightness difference to identify a region.

`arealim`: Particle size limits (either a single value for minimum size or a two-element vector [min, max]).

Output:

`pos`: Region centroids (x, y coordinates).

`ang`: Orientations of regions in radians.

Functionality:

1. Region Identification:

    - The function identifies regions in the input image (`im`) that are brighter than the specified threshold.

2. Region Properties:

    - Utilizes the `regionprops` function to compute properties of the identified regions, including the weighted centroid, area, pixel values, and pixel list.

3. Filtering Regions:

    - Removes regions on the image boundaries and those with areas outside the specified limits (`arealim`).

4. Orientations Calculation:

    - Computes the orientation of each region using the second-order moments of the region's pixel values.

    - The orientation is returned in radians.

5. Output:

    - Returns the region centroids in the `pos` array and orientations in the `ang` array.