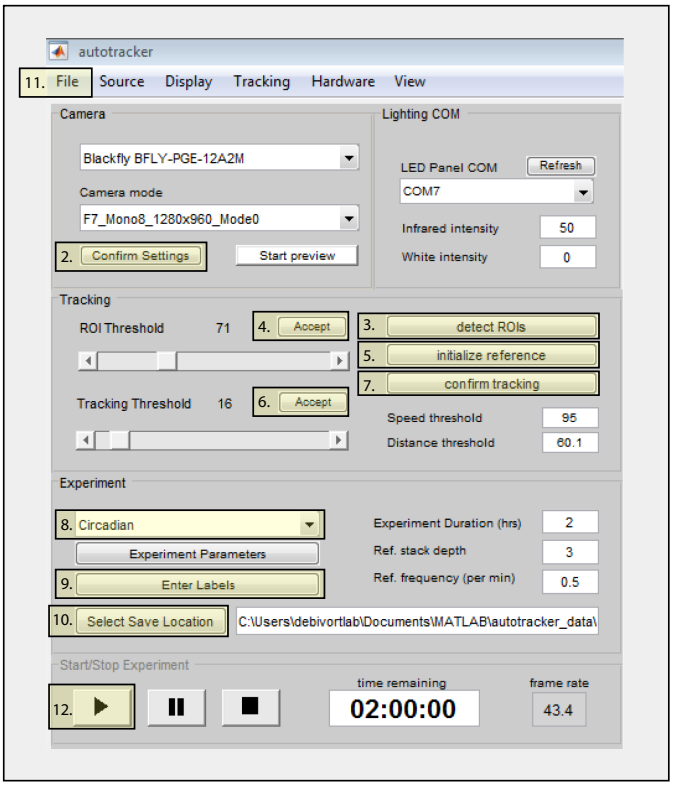
Autotracker - User Manual

May 9, 2018

Contents

1 Introduction

1.1 Quickstart Guide



QUICK START GUIDE

1. Type and run *autotracker* in the MATLAB command window
2. Select mode and confirm camera
3. Run ROI detection
4. Adjust and accept ROIs
5. Initialize background referencing
6. Adjust tracking threshold and accept reference image
7. Confirm tracking and sample tracking noise
8. Select protocol and adjust experiment parameters (if applicable)
9. Enter labels
10. Select directory to save output
11. Save parameter configuration for future use
12. Begin tracking

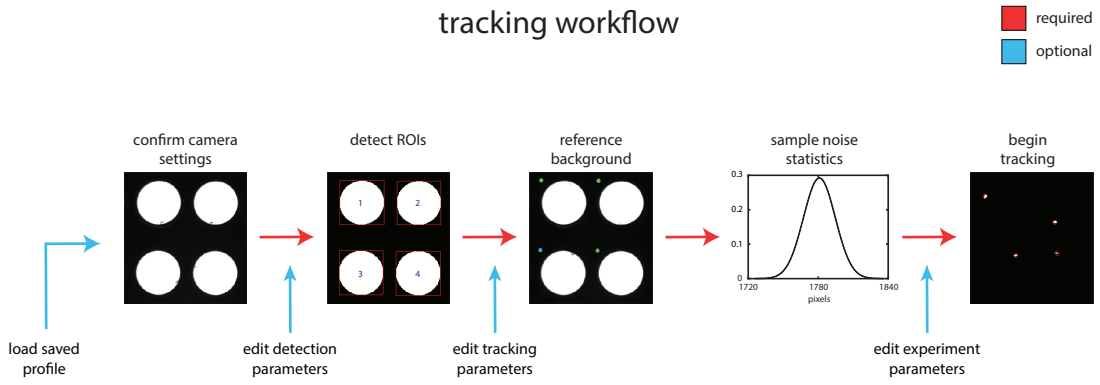


figure 1.1 - Every tracking session follows a standard workflow with multiple stages for optional customization. The time needed to setup a new session can be substantially reduced by loading saved parameters.

1.2 Overview

Autotracker is a MATLAB-based software package for real-time, high-throughput tracking of individuals. Rapid tracking and identity sorting in real-time means that Autotracker is particularly useful for applications requiring closed-loop control of individual stimulus delivery and massively high-throughput behavioral screens where video storage and processing would otherwise be rate-limiting. Many existing tracking platforms such as Ctrax, idTracker, and FlyTracker are designed to track and maintain individual identities of mixed groups individuals. Tracking identity in mixed groups requires resolving identities through collisions where bodies are overlapping. This approach offers the ability to study social behaviors, but because collision resolution is computationally expensive, software packages focused on solving this perform acquisition and tracking separately. Assigning centroid identity by associating individual with a predefined region of interest (ROI) allows Autotracker to track individuals in real-time and avoid recording image data altogether.

Autotracker offers additional utility through extensive GUI-based customization of hardware settings, advanced tracking parameters, ROI-detection modes, and saved preferences. In addition to centroid data, Autotracker can output timestamps, body orientation, body size, speed, heaving direction, angular velocity, and other custom defined metrics. Support for cameras, external displays, and COM devices makes integrating hardware into experimental paradigms simple. Autotracker facilitates experimental paradigms that require closed loop control of individual stimuli based on individual behavior. Furthermore, a small data footprint and robustness to tracking noise makes Autotracker ideal for recording behavior over long time scales.

1.3 Target audience

The target audience of Autotracker is any user wanting to use a single, integrated platform for recording body position and orientation based behavioral metrics over time. Many features of behavior can be calculated or inferred from these simple pieces of information. To extract these measurements, Autotracker uses thresholded images to track binary blobs. This means that the software is optimized for tracking many individuals at very low-resolution and not for high-resolution tracking of postural subfeatures (eg. tracking positions or orientations of individual body parts). Relying in ROIs to track individual indentity also means that Autotracker is built for high-throughput experiments where individuals are physically separated into isolated arenas (eg. bottomless 96 well plates). Autotracker is not designed for collision resolution and therefore does not track the identities of multiple individuals sharing the same arena.

1.4 Experimental workflow

Every live experiment conducted in Autotracker follows the same fundamental workflow to ensure that the software assign regions of interest, separates the interesting parts of the image from background, and filters out noise in the image that would otherwise cause errors in centroid estimation. Before recording data, the user must:

1. Select and confirm settings for the camera
2. Automatically or manually set ROIs
3. Initialize a background reference image
4. Collect a sample of clean imaging
5. Set experiment parameters

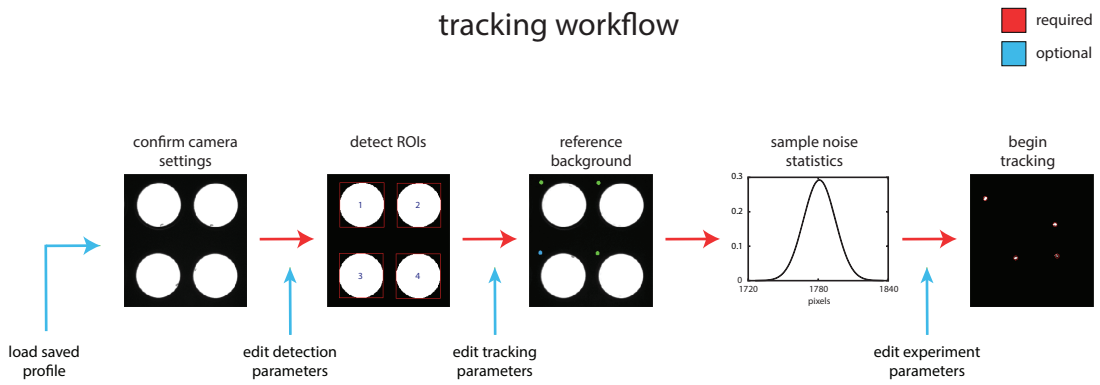


figure 1.3 - Every tracking session follows a standard workflow with multiple stages for optional customization. The time needed to setup a new session can be substantially reduced by loading saved parameters.

Select and confirm settings for the camera

Autotracker will automatically detect any available camera with an associated MATLAB imaq adaptor installed and auto-populate the camera panel drop down with detected cameras. If the camera is not automatically detected, see Camera detection. Select the appropriate camera and the camera mode and click "Confirm Settings". After initializing the camera, an initial preview image will be visible in the display window and the downstream ROI detection controls will be enabled. The camera preview button can be used to start and stop a streaming preview from the camera at any time the camera is not actively in use.

Automatically or manually select ROIs

Before individual identities can be tracked frame to frame, regions of interest must be set. Autotracker sets ROIs through either *automatic* or *grid* modes. ROI detection mode is set to automatic by default. Automatic ROI detection works on two fundamental assumptions. The first is that your ROIs will be bright regions separated by dark boundaries. The second is that ROIs will all

generally be about the same size. To run automatic ROI detection, select ***detect ROIs*** and adjust the ROI threshold slider bar to find a threshold that cleanly separates ROIs from the background. Bounding boxes and ROI numbers will be displayed over any detected ROIs in real time. Select ***Accept*** once the ROIs are properly detected to confirm the ROI positions. See ROI detection for more information on automatic and grid based ROI detection modes.

Initialize a background reference image

To track changes in the image from frame to frame, Autotracker calculates a difference image between each frame and a background reference image. The background reference is generated by computing a median image from snapshots of each ROI when the individual is in different locations. This effectively generates an image of the ROIs without individuals. In order for this strategy to work, Autotracker must be able to track individuals in each ROI to ensure that the individual is in distinct locations each time a new image is taken. To collect a reference image, select "initialize reference" and adjust the tracking threshold slider until individuals are detected. Colored indicators next to each ROI will progress from purple to green as sample images of each ROI are collected. The "centroid numbers" option in the View menu bar can be selected at any time to visualize the assignment of centroid identity. See Background referencing for more information on how to optimize a starting reference image.

1.5 Getting the most out of Autotracker

Fast, accurate tracking can be highly dependent on optimization of parameters, camera configuration, and illumination. Autotracker attempts to make the task from setup to data collection as streamlined as possible by making some of these decisions for you. Since every experiment and tracking setup is a bit different, getting optimal tracking performance out of Autotracker may require adjusting both software and hardware parameters. Autotracker has many integrated features to help the user make these decisions and then save the preferences to a user profile to ensure that the setup process is only necessary once. Here is a helpful but non-comprehensive list of things that will help you get the most out of Autotracker:

high priority

- Behavioral arenas that facilitate ROI detection
- Bright, diffuse backlighting
- Proper camera configuration
- Tracking parameters customized to your ROIs and tracked objects
- Adjust acquisition rate to avoid "over-tracking"
- Save your settings to a profile

recommended

- Camera calibration to reduce lens fisheye
- Use display options to visualize and validate tracking features in real time
- Calibrate Autotracker to measure distances in mm

optional

- Display visual stimuli with PsychToolbox
- Use COM objects to control peripheral hardware

2 Hardware Setup

2.1 Tracking box

Many of the challenges in achieving robust tracking over long time scales that would be very difficult to solve by adjusting parameters in the tracking can be entirely avoided by constructing a dedicated space to for your tracking setup. A tracking box has the combined benefit isolating both the camera and the tracked individuals from external perturbations. Use the sample schematic below as a reference for good tracking conditions before attempting to optimize other hardware or software parameters (fig 2.1). The only essential features of a tracking box are opaque walls, a camera mount, and a diffuse illumination source. The difference image calculation used by Autotracker is very sensitive to any changes within the field of view of the camera. This is good because it means that the software is sensitive to very small movements of small objects, but it also means that the only movement visible to the camera should be the tracked objects. Autotracker can automatically detect and adjust for spikes of noise in the image, but walls will drastically reduce the time Autotracker spends attempting to correct noisy imaging and will increase the time spent tracking.

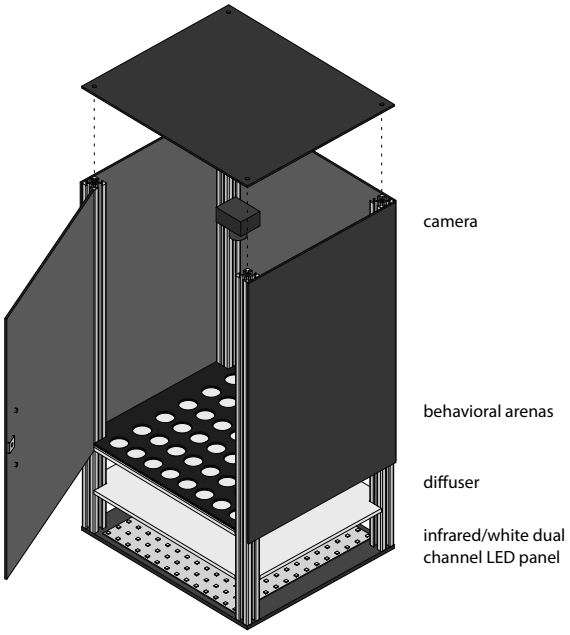


figure 2.1 - Sample tracking platform constructed from aluminum rails and acrylic plastic. Backlit behavioral arenas and opaque walls enhance ROI contrast and reduce imaging noise.

In addition to walls, use clear sanded plastic or diffuser film either between the illumination source and behavioral arenas or on the floor of the arenas themselves to diffuse the light source. Having a sanded or matte finish on all surfaces inside of the tracking box can also help reduce reflections and glares inside the box.

2.2 Individual behavioral arenas

Autotracker ROI setting can operate in two different modes: *automatic* and *grid* modes. Choosing approriate individual behavioral arenas will be dependent on the ROI detection mode used. Automatic detection mode works by finding bright regions of the image that are all roughly the same size. This means that ideal

behavioral arenas used for automatic ROI detection are backlit, transparent areas separated by opaque areas in between. Sample arena construction for automatic detection is shown below in fig. 5.

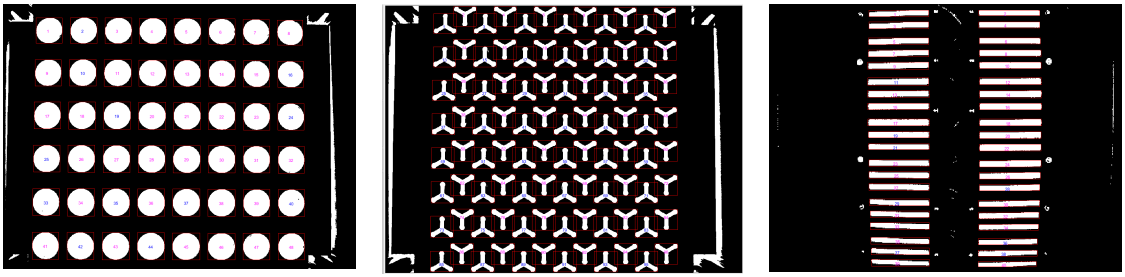


figure 2.2.1 - Sample images of automatic ROI detection.

Grid detection mode works only on the assumption that ROIs will be arranged in regularly spaced rows and columns. The user is prompted to draw and adjust the position of one or more grids of any arbitrary dimensions to specify ROIs. This means that behavioral arenas can be anything with stereotyped dimensions. Wellplates of any dimensions make ideal behavioral arenas for grid mode detection. Grid mode also works well with linear arrays of tunnels. Sample arena constructions for grid detection mode are shown below in fig. 6.

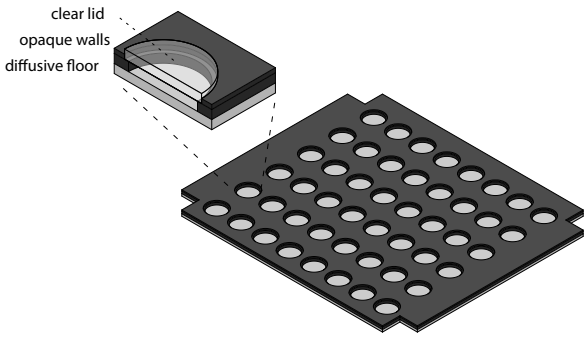


figure 2.2.2 - Sample tracking arena constructed from layered, laser cut acrylic. Semi-transparent floors and opaque walls create high-contrast, easily detectable ROIs.

2.3 Illumination

Quality illumination is a prerequisite for high-quality tracking. An ideal light source will be evenly diffuse and sufficiently bright. Because Autotracker uses a single tracking threshold for the entire field of view of the camera, even illumination is the top priority for an illumination source. Autotracker can correct for uneven brightness, but having an even illumination source will ensure that the quality of tracking is equally good in all parts of the cameras field of view. Illumination sources brightness is only important to the point that it is bright enough. Any illumination source bright enough to saturate the camera when the aperture is fully open is sufficiently bright for tracking.

Using an infrared light source and longpass filter on the camera has the dual benefit of reducing sensitivity of the tracking to perturbation by most external light sources (not sunlight) and allowing for independent control of visible light sources to deliver stimuli not visible to the camera. An affordable example of two color channel LED light panels configured with white and infrared LEDs can be found here.

2.4 Camera configuration

Autotracker has built-in tools to automatically detect and configure any camera visible to MATLAB. The appropriate configuration for your camera will largely depend on your particular experiment. Camera customization features are accessible under the "Hardware" menu bar. But before configuring the camera in Autotracker, ensure that the camera and camera lens are properly setup for tracking. The following are good guidelines for configuring your camera:

- Aperture opened enough that ROIs are nearly saturation
- Lens maximally zoomed to reduce lens fisheye
- Lens focused on ROIs
- Camera rigidly fixed to a camera mount

2.4.1 camera detection

Camera support in Autotracker is built on MATLAB's Image Acquisition Toolbox. Before a camera can be detected by Autotracker, the associated the associated MATLAB camera adaptors must first be installed. The appropriate adaptor to install will depend on the camera manufucturer. See MATLAB's tutorial for complete instructions on installing MATLAB camera support packages.

Once the camera adaptors are installed, a list of available cameras and camera modes should auto-populate upon launching Autotracker. The list available cameras can be refreshed under the hardware menubar.

2.4.2 camera modes

Autotracker will automatically detect available camera modes for operating at different pixel resolutions and color bit depths. The tracking is designed to work with 8-bit color depth because most tracking operations are performed on binary images. It is also strongly recommended that monochromatic cameras are used. By default, Autotracker only tracks the green color channel of RGB images.

2.4.3 camera settings

Camera properties such as exposure time, shutter speed, gain, and frame rate can be adjusted through *camera settings* under the *Hardware* menu bar. Because the configurable settings are specific to each device, the properties that can be adjusted here will depend on your camera. Many cameras have modes to automatically adjust the camera exposure time, shutter speed, frame rate, gain or focus in real time. These modes are also often enabled by default, which may be good for many applications but is highly problematic for tracking. Constantly fluctuating images will introduce a lot of noise into the tracking due to the sensitivity of difference image to even minor changes from frame to frames.

To adjust camera settings, first initiate *preview camera* and leave it running to get feedback on the settings as they change. Select ***Hardware*** > ***Camera*** > ***camera settings*** to open the camera settings window. Keep in mind that the available settings and names of each property will be dependent on the camera. From the settings menu:

- Ensure that any automatically adjust fields are set to "off" or "manual"
- Adjust exposure and shutter speeds until the preview is just below saturation
- Set max frame rate (target acquisition rate can be set lower in tracking parameters)
- Close window to save settings

3 Starting a New Experiment

Before running any experiment, Autotracker must go through a simple setup that follows the workflow shown in fig. 1. Autotracker's user interface enforces this workflow by progressively enabling downstream controls as each sequential step is completed. To access the next step in the workflow, the user must complete the preceeding step. Keep in mind that loading a custom profile will allow many of these steps to be skipped and substantially reduce the time needed to setup a new experiment.

3.1 Confirm camera and camera mode

The only control panels enabled by default are camera and illumination (not covered here). The user must minimally select a camera, image resolution, and data output format from the drop down menus. Once selected, *confirm settings* to initialize the camera. At this point, it is also a good to preview the camera and adjust any other camera settings if necessary. Because the tracking and ROI thresholds selected in the downstream steps can be sensitive to changes in exposure, shutter speed, and gain, it is strongly recommended that these settings are manually configured before continuing.

3.2 ROI detection

A region of interest (ROI), is a static location in the camera's field of view that tells Autotracker where to expect tracked objects. The software can handle ROIs with little or no movement by ignoring any traces in those ROIs, but any movement outside of an ROI will never be recorded in the first place. It is therefore important ROIs are set properly before the tracking begins. ROI detection in Autotracker comes in two flavors: *auto* and *grid* detection modes. As the name suggests, *auto* mode is essentially instantaneous and easy, but requires your ROIs and imaging setup to meet particular conditions. This mode is enabled by default. On the other hand, *grid* mode tolerates greater flexibility in ROI and imaging conditions but requires the user to draw and position an ROI grid over the image. ROI detection modes can be switched under ***Tracking*** > ***tracking parameters*** > ***ROI detection***.

auto mode

Automatic ROI detection works by finding a threshold value that separates bright regions of the image from a dark background. Before applying the threshold, Autotracker first attempts to correct for in vignetting or global unevenness in the illumination. Images commonly tend to be brighter in the center and dimmer at the edges, correcting this unevenness makes it easier to find a single threshold value that will cleanly separate all ROIs in the field of view from the background.

To run automatic ROI detection, select ***detect ROIs*** and adjust the ROI threshold slider until the ROIs are cleanly separated from the background. Displayed ROI numbers and bounding boxes show the automatically assigned identity of each ROI in the image and its boundaries. A vertical orientation indicated by font color will also be assigned to each ROI. This is useful for ROIs with a clear vertical assymetry such as the ones shown in **fig 3.2.1**. Orientation can largely be ignored for ROIs without assymetry.

The following steps may help in cases where auto mode detection either fails to pick up or inaccurately identifies one or more ROIs:

- increase the threshold if it incorrectly assigns identities to non-ROIs
- decrease the threshold if fails to detect real ROIs
- ensure nothing dark or opaque bisects any ROIs
- use manual vignette correction if uneven illumination causes ROI dropping at the edges of the camera field of view

- manually edit individual ROIs if needed

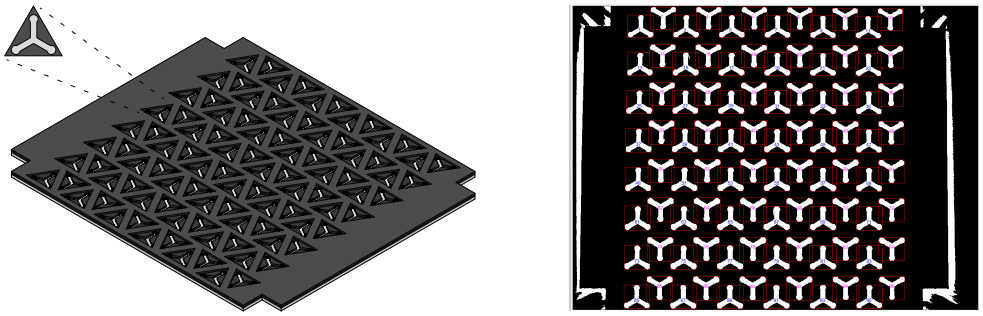


figure 3.2.1 - Sample schematic (left) and ROI detection (right) of an arena optimized for automatic detection. Non-grid structure of arenas makes it unsuitable for grid detection mode. Automatic ROI detection records vertical asymmetry of ROIs. In a Y-shaped arena, recording the orientation makes it easy to infer endpoints of the maze arms.

grid mode

Grid mode ROI detection avoids the need for using imaging tricks to assign ROIs but requires a little user input to make assignments. This mode also makes the assumption that boundaries between ROIs can be drawn in a regular grid-like structure.

After starting grid-based detection by selecting *detect ROIs*, the user interface controls will be temporarily disabled as it waits for the user to drag and drop a new grid into the imaging window. A grid settings user panel will appear with options to add and remove grids as well as change the dimensions of each grid. Once a grid is placed in the imaging window, customize the grid by repositioning corners or editing the number of rows and columns. The ROI bounds displayed in the imaging window show the enclosed space in which a centroid will be assigned to a particular ROI. The bounds should be positioned such that they fall in the space arenas. New grids can be added or removed at any time by selecting the + and - controls. Multiple grids allows several trays to be imaged simultaneously by the same camera. This works particularly well when imaging multiple well plates at the same time. Once finished editing grids, select *Accept* to save the ROI positions.

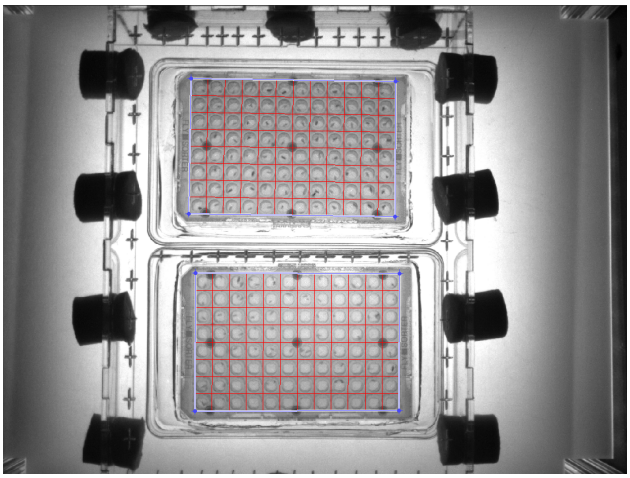


figure 3.2.2 - Grid detection mode captures regularly arrayed and low-contrast ROIs.

3.3 Background referencing

Once ROI locations are set, Autotracker creates a background reference image to keep track of the frame to frame differences between the current frame and the appearance of the arenas without any tracked individuals. Autotracker uses periodic sampling of the background throughout tracking to constantly update the background appearance, but a reference must first be initialized before tracking can begin. Because the arenas are not actually empty when this reference is created, Autotracker takes snapshots of each ROI separately any time a tracked individual has moved far enough away from any position where a previous snapshot was taken. A median image is computed for each ROI separately and then combined into a single master reference for entire field of view. Initializing a relatively accurate reference *very important* because it allows for accurate noise profiling, which is essential for robust tracking.

Select ***initialize reference*** to begin referencing. The imaging window will display a thresholded difference image between the reference and the current frame. Because the reference is first initialized to a sample image at the start of referencing, the image should be largely blank at first but begin to populate with individuals as they begin to move. New samples of each ROI will be progressively collected as individuals move around in each ROI. Circular indicators to the upper left of each ROI will progress from purple to green as more references are taken for each ROI, with green indicating referencing complete for a given ROI. Adjust the tracking threshold until pixel noise is largely absent from the thresholded image. Select *Accept* on the tracking threshold slider to set the reference image. *Note:* Rougly half or more of the ROIs at green should be sufficient to proceed to sampling noise statitics.

3.4 Noise profiling

Difference imaging is extremely sensitive to even minor changes between the background reference and the current frame. For that reason, minor pertubations to the imaging such ambient vibrations over long time scales or acute pertubations such as bumping tracking box can drastically increase the noise in tracking. Autotracker samples the distribution of above threshold pixels during a sample, clean tracking period of one hundred frames so that it can constantly monitor the quality of and counter any deterioration in the difference image. For this approach to work, the distribution collected during this period must be a relatively accurate representation of what it should look during the rest of the experiment.

Select ***confirm tracking*** to begin sampling. The tracking threshold can be adjusted up or down to bias sampling to be either more or less sensitive to noise during tracking. Noise sampling tips:

- Try to get at least half of all ROIs completely referenced during the previous step
- Slightly lower the tracking threshold if too few individuals are showing above the difference image
- Obtain more accurate noise profiling or further lower the tracking threshold during sampling if noise threshold reference reset is constantly triggered during tracking

3.5 Experiment settings

With the tracking setup complete, all that remains is to select the experiment to be run and, if necessary adjust a few settings before beginning tracking.

3.5.1 select experiment

The experiment selection determines which experiment and data processing protocols to execute once tracking begins. If you only want to record centroid coordinates and time stamps, select *Basic Tracking* from the dropdown menu and move to the next step. Autotracker was designed not only to record basic

tracking data, but to run a suite of experimental protocols. Those protocols differ in the raw data they record as well as their hardware control schema and behavioral metrics recorded. Users can define their own custom experiments or select from a list of pre-defined protocols. More detailed explanations of the pre-defined experiments in Autotracker see the original Autotracker publication.

3.5.2 parameters

If necessary, adjust the duration of tracking in hours, the number of reference samples per minute, and the number of reference samples to collect per ROI. Depending on the selected experiment, additional customizable may be available under *experiment parameters*. Keep in mind not all protocols, including *Basic Tracking* will have any additional parameters to adjust.

3.5.3 labels

Select *labels* to attach meta data to any particular range of ROIs. By default, Autotracker has fields for recording genetic strain, sex, treatment condition, ID numbers, Day of testing, tracking box, tracking arena/-plate, and any additional comments. To append labels, enter a range or ROIs the label applies to, fill in the labels, and select *Accept*. Distinct categories can be entered in subsequent rows. *Note:* Autotracker will auto-generate a file label from the first row of entries.

3.5.4 save path

Browse to a parent directory for the autotracker output. Autotracker will autogenerate a new directory within the chosen save path with the time stamp and label information for the experiment where it will save all raw and processed data.

3.6 Save your settings - *recommended*

Before starting the experiment, it is *strongly* recommended to save a profile for the current configuration of *Autotracker*. By loading a saved configuration profile in the future, you will avoid having to do any parameter configuration. Only information such as the ROI positions and reference image that is unique to each instance of tracking will not be saved. Saving a profile not only substantially simplifies the setup process, but also ensures consistency that will make it easier to compare recordings across sessions.

To save a new profile, select *File* > *save a new preset*. A profile can be loaded at any time, but because presets contain information about camera configuration, it will be necessary to re-initialize the camera anytime a new preset is loaded.

4 Running Analysis

Autotracker will execute an analyze file for the accompanying run file upon finishing tracking. Processing times can vary greatly depending on your computer and the size of the raw data files. Once analysis is complete, the user will be given the option to view raw centroid data. At this point, the *expmt* master can be loaded into the MATLAB workspace by copying and executing the command printed to the command window or by browsing to the save directory and manually loading the file.

Users may wish to try processing the data multiple different ways or set optional processing flags or parameters. For this reason, Autotracker has functions for re-processing data files, repairing broken references to raw data files, and setting optional processing flags.

The simplest way to get started is the ***analyze_multiFile*** function. This function allows the user to browse and select a parent directory containing all files to be reprocessed. Once a parent directory is selected, the function will search recursively through all directories underneath for any .mat files. An optional keyword can be provided to restrict the query to file names containing the keyword argument. The *expmt.Name* property is used to query and execute to the accompanying analyze file for each file sequentially. This means that *expmt* files of many different types can be processed together in batches. The following name-value pairs can be set to customize the analysis:

Name	Description	Values	Default	Data Types
Keyword	Restricts file search to .mat files containing keyword	*any string*	none	string
Save	Toggles figure and file saving	true | false	true	binary
Raw	Sets raw data files to be generated from centroid trace features	'Speed' | 'Direction' | 'Theta' | 'Radius'	none	string, cell array of strings
Bootstrap	Toggles parsing speed data into discreet bouts for bootstrap re-sampling	true | false	false	binary
Regress	Toggles correction of speed data by regressing out lens distortion	true | false	false	binary
Handedness	Toggles extrapolation of handedness metrics from centroid traces	true | false	false	binary
Slide	Toggles calculating sliding average speed in Circadian experiments	true | false	false	binary
AreaThresh	Toggles calculating individual area thresh for parsing floor/ceiling bouts in Circadian experiments	true | false	false	binary

table 4.1 - Processing name-value pairs for analyze_multiFile

5 Data Output

Data is output from Autotracker in three phases. Once the play button is pushed, Autotracker generates and saves a .mat file with all meta data during initialization of the experiment. During tracking, Autotracker writes raw data fields to the hard drive. Once tracking is finished, a round of protocol-specific, automatic data processing is performed to calculate metrics for higher level features of the raw data.

5.1 Raw data

Raw data is saved frame by frame to binary data files located in the subdirectory under the user-defined save path. Each tracked field is saved independently to its own file. The data can be read directly into MATLAB or accessed indirectly through memmaps in the data master struct. The simplest way to access the data quickly is through the automatically generated memory maps. However, the binary data files can be accessed via MATLAB's *fread* function. If attempting to access the raw data directly, keep in mind that the data must be read in the correct precision. In addition to the ability to record user-defined raw data, Autotracker can record the following pre-defined raw data fields: *centroid*, *inter-frame interval (ifi)*, *area*, *orientation*, and *speed*.

5.2 Master data struct (expmt)

A single data struct, *expmt*, allows the user to access all raw and processed data output. The master struct is saved to a .mat file in an auto-generated, time-stamped directory under the save directory. The master data struct can be easily loaded into MATLAB and is a convenient way to browse and manipulate all the data. See MATLAB's tutorial on structs if you are unfamiliar with working with MATLAB structs. See table 5.2 for a complete details of the contents of the *expmt* struct.

5.2.1 experiment meta data

All meta data from the experiment is saved to the master struct during initialization of tracking. By default, Autotracker records the following meta data:

- time, date, and duration
- ROI position and dimensions
- label meta data
- camera and other hardware settings
- referencing and tracking parameters
- imaging noise statistics
- raw data file paths, format, and dimensions
- protocol specific parameters

5.2.2 Memory mapped raw data

Raw data files can be accessed through the master data struct through the use of memory maps. Although Autotracker has a much lighter data footprint than raw video data, raw data files can still be cumbersome, or impossible to hold in memory. Because Autotracker can efficiently track and record activity from thousands of individuals over very long timescales, raw data files can easily exceed several gigabytes in size. For this reason, raw data is dynamically read from the hard drive to avoid *out of memory* errors. Mapped raw data can be dynamically accessed under *expmt.(raw field name).map.Data.raw* like a normal MATLAB array. For example, centroid data can be accessed under ***expmt.Centroid.map.Data.raw*** and is formated as $MxNxP$ matrix where M = number of ROIs, N = 2 (x,y), and P = number of frames. If we wanted to assign the centroid coordinates for ROI #10 for the entire length of the experiment to temporary variables, we could call:

x = expmt.Centroid.map.Data.raw(10, 1, :);
y = expmt.Centroid.map.Data.raw(10, 2, :);

5.2.3 auto-processed centroid features

When tracking is finished, Autotracker runs a protocol-specific, data-processing script to generate memory maps to the raw data and calculate higher level behavioral features. All tracking protocols in Autotracker minimally record centroid position and inter-frame interval. As examples of higher level features, Autotracker uses post-processing to generate measures of: *individual activity*, *locomotor handedness*, and *stimulus evoked behaviors*.

5.3 Figures

Depending on the selected tracking protocol, Autotracker may output figures during post-processing. All figures are saved to an auto-generated *figures* directory under the user specified save location. The figure directory is saved to *expmt.figdir* in master struct.

5.3.1 Raw Trace Browser

Users will be prompted to browse raw trace data upon tracking completion. Select ***browse traces*** to open a simple GUI for plotting centroid data. Traces from any *expmt* master struct can be browsed at anytime by loading the .mat file and running ***plotTraces(expmt)*** from the MATLAB command line.

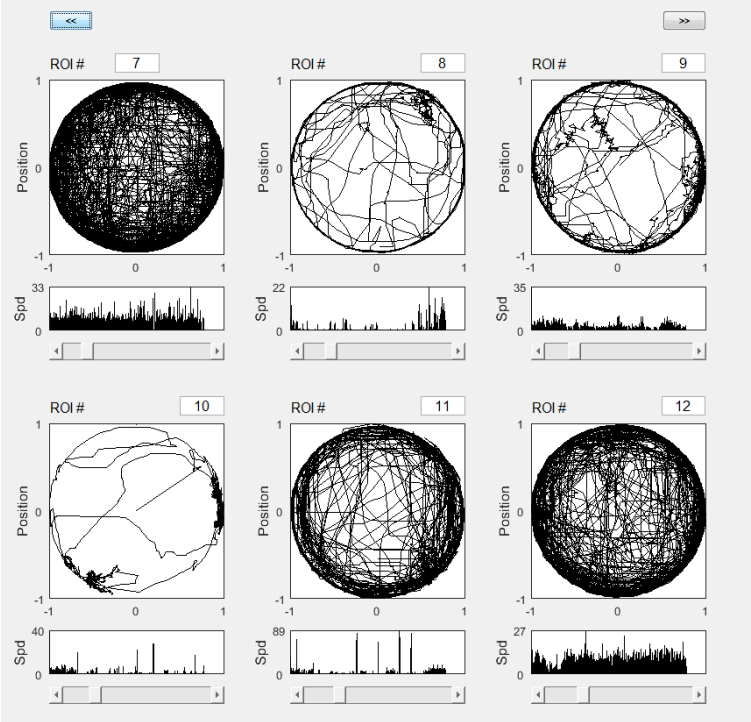


figure 4.3.1 - Raw trace browser utility can be used to view raw centroid traces. Plot traces for an *expmt* master struct by running *plotTraces(expmt)*

expmt struct fields

Raw Data

Field	Description	Subfields
Centroid	struct containing raw centroid data memory map (*numROIs x 2 x numFrames*) as well as raw centroid data file path, data precision, file ID, and dimensions for each frame	map, path, precision, fID, dim
Time	struct containing raw time data memory map formatted as interframe intervals (rather than timestamps) as well as raw time data file path, data precision, file ID, and dimensions for each frame	map, path, precision, fID, dim
<raw_output_field>	structs containing raw data memory map for any other specified raw data fields in the above format	map, path, precision, fID, dim
fields	cell array of strings listing raw output field names (eg. ['Centroid'; 'Time'; 'Area'])	none

Meta Data

Field	Description	Subfields
parameters	struct containing autotracker parameter values	duration, ref_depth, ref_freq, ROI_thresh, track_thresh, speed_thresh, distance_thresh, vignette_sigma, vignette_weight, area_min, area_max, target_rate, mm_per_pix, units, ROI_mode, sort_mode, ROI_tol, edit_rois, dilate_sz
ROI	struct containing information about ROI number, position, orientation, size, shape, imaging mask, ROI definition mode,	n, centers, corners, orientation, im, mask, pixIdx, mode, (**optional**: shape, vec, row, col, grid, tform, cam_dist)
nTracks	number of objects tracked (ie. number of ROIs)	none
nFrames	length of the tracking in frames	none
Name	name of the experiment tracking protocol (eg. Basic Tracking, Y-maze, Circadian)	none
source	image data source (ie. camera/video)	none
ref	struct containing background reference image, image stack, number, timer, inter-reference distance threshold, tracked object positions for each image in the reference stack, and reference update trigger	im, stack, ct, t, thresh, cen, update
noise	struct containing noise sample meta data including raw distribution, mean, standard deviation of number of above threshold pixels during sampling period as well as distributions, means, and standard deviations of above threhsold pixels for each ROI	dist, mean, std, roi_dist, roi_mean, roi_std
vignette	struct containing vignette correction image and mode	im, mode
labels	cell array containing the values from label UI table	none
labels_table	ROI-wise formatted labels containing label info for each individual ROI	Strain, Sex, Treatment, ID, Day, Tray, Box, Comments
drop_ct	raw number or fraction of total frame number excluded from tracking due to excess noise	none

File Pathing Info

Field	Description	Subfields
fdir	directory of the *expmt* struct .mat file	none
rawdir	directory of the the associated raw data files	none
fLabel	auto-generated label string for creating unique and identifiable directories and file names (eg. 'date_trackingname_strain_ID#')	none
fpath	original user-selected save directory	none
figdir	directory for auto-generated figures	none

Hardware Info

Field	Description	Subfields
camInfo	struct containing camera video object as well as adaptor and device info	vid, Device Info, DeviceIDs, activeID, AdaptorName
video	struct containing video file object, directory, file names, file number, duration, and frame number	vid, fdir, fnames, nVids, total_duration, nFrames
light	struct containing illumination panel PWM values	white, infrared
COM	illumination panel controller COM object	port, name, status, byteorder
AUX_COM	cell containing list of available auxillary COM ports	none
projector	struct containing scattered interpolants used to convert camera pixel coordinates to projector pixel coordinates	Fx, Fy
scrProp	struct containing screen properties for displaying Psychtoolbox stimuli	window, screenNumber, windowRect, xCenter, yCenter, vbl, waitframes

table 4.1 - Processing name-value pairs for analyze_multiFile

6 Advanced Tracking Features

Autotracker understands that the setup and goals of tracking can vary greatly from instance to instance. For that reason, many tools have been included to offer flexibility and customization in how ROIs and centroids are tracked, sorted, and measured.

6.1 Tracking parameters

Many of tracking parameters are used to track, identify, or sort ROIs and tracked objects. These parameters can be adjusted under *Tracking* > *tracking parameters*.

- **speed threshold** - maximum allowed distance traveled per second. Each update in an object's position is time-stamped. Every frame, Autotracker starts with unassigned blobs in a difference image. To assemble blob positions into traces over time, tracked objects are assigned to an ROI. The speed threshold excludes centroids in any given frame that have moved too far from the last recorded centroid for the paired ROI. This parameter helps prevent the centroid from jumping to pixel. *Note:* toggling the speed threshold radio button after ROI detection and referencing will display a measured rolling average speed and standard deviation for each ROI to help with parameter setting.

- **distance threshold** - maximum allowed distance to the center of an ROI. This parameter is used to identify which ROI to assign blob centroids. This parameter is only used when *distance* centroid sort mode is selected. *Note:* toggling the distance threshold radio button will display a circle showing the range of inclusion for each ROI.

- **area thresholds** - these parameters set lower and upper bounds that blobs in the threshold image will be tracked. Objects in the image outside of this range are excluded from tracking in the current frame. *Note:* toggling the area threshold radio button after ROI detection and referencing will display a measured rolling average area and standard deviation for each ROI to help with parameter setting. This will also display concentric circles of area equal to the specified bounds.

- **vignette gaussian weight** - sets the multiplicative weight of a gaussian mask applied to the entire image during ROI detection. Detecting all ROIs in an image of varying luminance can be challenging. Once Autotracker knows which parts of the image are foreground and which are background, a vignette filter can easily be calculated. Before foreground and background are distinguished, some assumptions can help smooth the luminance profile of the image. Many camera lenses and luminance sources create images that are slightly brighter in the center and dimmer at the edges. By default, Autotracker assumes a gaussian profile of global luminance in the image and applies a filter to the image to smooth the global luminance. A lower weight will reduce the smoothing applied to the image. *Note:* this parameter is only used during automatic ROI detection and does not apply to *grid mode* detection or object tracking.

- **vignette gaussian sigma** - sets the standard deviation of the above vignette correction gaussian as a fraction of the image height. *Note:* this parameter is only used during automatic ROI detection and does not apply to *grid mode* detection or object tracking.

- **target acquisition rate** - sets the maximum acquisition rate of tracking in hertz. Adjusting this parameter only ensures that tracking will not exceed this rate. The maximum achievable rate will depend on computer hardware, image resolution, number of objects tracked and centroid sorting mode. For more information on optimizing acquisition rate, see improving tracking performance.

- **ROI clustering tolerance** - sets the maximum number of standard deviations in vertical distance for adjacent ROIs to be assigned to the same ROI. To automatically assign numbers to ROIs when using *auto* ROI detection mode, Autotracker first sorts ROIs by their vertical center of mass and measures the vertical distance between adjacent ROIs. Any pair of ROIs that are within the maximum number of standard deviations of one another are said to belong to the same row. Decreasing this value will make vertical clustering of ROIs more stringent. Increasing the value will cluster ROIs that are more vertically separated into the same row of ROIs. *Note:* this parameter is only used during automatic ROI detection and does not apply to *grid mode* detection or object tracking.

- **dilation size** - sets the number of pixels to dilate and erode the threshold image. Dilation and erosion helps fill in gaps between above threshold pixels that, ideally, should belong to the same blob. This is particular helpful in under-illuminated images where tracked individuals are split into multiple nearby blobs. Increasing this value will result in slower tracking speed but will help stitch nearby pixel islands into the same blob. *Note:* set this value to zero to disable dilation and erosion.

- **sort mode** - sets mode by which centroids are assigned to ROIs. *Distance* and *bounds* sorting modes requires centroids to be within the specified distance of the center or fall within the bounds of an ROI, respectively. Centroids not meeting the requirement are not eligible for assignment to that ROI. *Distance* mode is most useful with round shaped ROIs. *Bounds* mode is most useful with elongated ROIs that have an asymmetric width and height. *Note:* bounds sorting mode is automatically employed when grid ROI detection mode is used.

- **ROI detection mode** - sets the mode by which ROIs in the camera's field of view are defined. See ROI detection for more details on these tracking modes.

6.2 Converting pixel units to mm

By default Autotracker measures distances in pixel units. A tool is included to allow calculating a pixel/millimeter conversion factor by drawing a line over a known distance in the image. To begin, measure the length in mm of any object that will go into the camera's field of view (eg. length of a 96 well plate). Place the object in under the camera and select *Tracking* > *distance scale* to open the conversion tool. Enter the length of the target object in mm in the space provided and select *draw line*. Click and drag in the camera preview window to drag and drop a line along the length of the target object to automatically calculate and store a pixel/mm conversion factor. If necessary, reposition the line end points and select *update* to calculate a new conversion factor. Close the window to save and exit. For accurate measurement, ensure the following:

- behavioral platform is perpendicular to the camera
- conversion factor is reacquired if the camera adjusted
- camera is calibrated to correct for fisheye lens distortion

6.3 Camera calibration

Camera parameter objects output by MATLAB's camera calibrator app can be used to correct fisheye lens distortion in images. To use camera calibration in Autotracker, follow the instructions in MATLAB's tutorial to calculate and export camera parameters to correct images. Save the exported object ~*autotracker/hardware/camera_calibra*
in a .mat file. Create the *camera_calibration* directory if necessary. Autotracker will automatically look for camera parameter objects in the specified directory upon initialization. Once the camera parameter object is exported and saved, toggle calibration under *Hardware* > *camera* > *use calibration*.

6.4 Vignette correction

Variation in baseline luminance across images can make it difficult to cleanly separate out tracked objects and ROIs with a single threshold value. Autotracker uses vignette correction to increase the dynamic range over which objects can be separated from the background. To achieve this, the software employs two different strategies to smooth the baseline illumination of all ROIs in the image.

- **Gaussian correction** is utilized in *auto ROI detection* mode prior to initial detection of ROIs. Uneven illumination occurs most commonly as vignetting (ie. light fall-off) at the edges of image (fig 5.4). Autotracker uses the assumption of a gaussian profile of vignetting centered on the image to smooth out the illumination. The width and weighting of the gaussian can be adjusted under *tracking parameters*. Figure 5.4 illustrates how correcting vignetting can allow all ROIs in the image to be cleanly separated from the background.

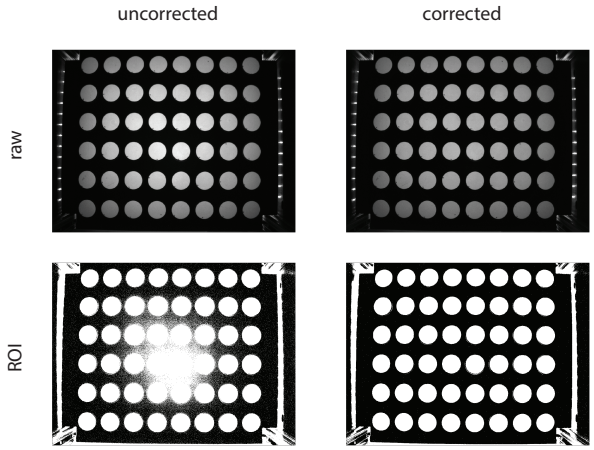


figure 5.4 - vignette correction can improve automatic ROI detection

- **ROI-based correction** can only be employed after ROI detection because it relies on using ROI definitions to pick out dim ROIs in the image. Using this information, Autotracker generates a subtraction matrix from the reference image. This matrix reduces all bright areas of image to the brightness of the dim ROI. This has the advantage of not assuming any particular shape to the variation in luminance across the image and often results in more even illumination than gaussian correction. A sample dim ROI can be picked manually by selecting ***Tracking*** > ***vignetting***. Click and drag in the camera preview window to select a region of the image at the target brightness.

6.5 Video recording

Raw video data can be streamed to disk simultaneous with tracking. Uncompressed video files are useful for software packages that train behavioral classifiers on the raw pixel data such as the Janelia Automatic Animal Behavior Annotator (JAABA). Videos can additionally be used for manual annotation behaviors or fed back into Autotracker or other tracking programs for independent validation of traces.

Toggle video recording by selecting ***Tracking*** > ***video*** > ***record***. By default, Autotracker saves uncompressed video data. Because raw video files can be very large, compression can be toggled by selecting *Tracking* > *video* > *compress*. It is strongly recommended that uncompressed videos are saved for any application using the raw pixel data as compression will result in the loss of information.

6.6 Tracking performance

Low frame rates are often sufficient to capture behavior and can help reduce files sizes and data processing. But closed-loop stimulus delivery may demand acquisition rates that far exceed these rates. Autotracker's real-time tracking excels at applications requiring tight closed-loop feedback. By optimizing your setup, Autotracker can easily achieve acquisition rates greater than 60Hz. Each of the following can limit tracking performance:

- background pixel noise
- *grid* ROI detection mode
- camera frame rate
- low target rate set under ***Tracking*** > ***tracking parameters***
- unoptimized minimum and maximum area thresholds
- display update mode (disabled by selecting ***Display*** > ***none***)
- blob dilation/erosion (disabled by setting dilation size = 0)

7 Creating a Custom Experiment

Every tracking protocol (eg.*Basic Tracking*, *Circadian* or *Y-maze*) follows the same simple workflow consisting of a run file and an analysis file. An example of a custom experiment is shown below by building off of the tracking script for the ***Basic Tracking*** protocol.

7.1 Run file template

Use the *Basic Tracking* run file located in <*autotracker_directory*>*/experiments/Basic Tracking/run_basictracking.m* as a template for creating a custom run file. Each run file will consist of initialization, tracking, and clean-up routines.

Initialization | Tracking Loop | Clean-up

```
% update notifications log
gui_notify(['executing ' mfilename '.m'],gui_handles.disp_note);

% clear memory
clearvars -except gui_handles expmt trackDat

% get image handle
imh = findobj(gui_handles.axes_handle,'-depth',3,'Type','image');

% properties of the tracked objects to be recorded
trackDat.fields={'Centroid';'Time';'Stimulus'};

% initialize labels, files, and cam/video
[trackDat,expmt] = autoInitialize(trackDat,expmt,gui_handles);

% lastFrame = false until last frame of the last video file is reached
trackDat.lastFrame = false;

% ------------------------------- %
% insert custom intialization here %
% ------------------------------- %

trackDat.Stimulus = false(expmt.ROI.n,1);
trackDat.CenterDistance = NaN(expmt.ROI.n,1);
```

figure 6.1.1 - The basic initialization template has been modified to include initialization of our custom fields (highlighted regions). The custom raw data field, *Stimulus* has been added to the list of tracked fields so that the data will be written to the hard drive each frame. A second field has been initialized to store the distance of each object to the center of its ROI for the current frame.

Initialization

The initialization routine ensures that the camera, data files, and tracking variables are setup for tracking. The setup can simply be copied for most experiments with two notable exceptions:

- Raw data fields are defined in ***trackDat.fields*** and will automatically generate and save to raw data files in each frame. On each iteration of the tracking loop, the Autotracker will to write any data in ***trackDat.***<***field_name***> to the paired raw data file. This means that *trackDat* must have an associated field for each entry to *trackDat.fields*. For custom data fields, the field must be defined and updated in the tracking routine. The minimum tracked fields (ie. Centroid and Time) are defined in the example below (fig. 6.1.1). For this example, let's assume we want to trigger a stimulus delivery when tracked objects are close to the center of the ROI and that we want to save a raw data file that reads *true* on frames where a stimulus was triggered and *false* on any other frame. For this purpose, a custom data field *Stimulus* has been added to the list of tracked fields.
- Any custom parameters and values should be defined at the end of the initialization routine. This can be initialization of a custom data field described above or any temporary values that are not to be saved to a data file. Here we define our custom raw data field, *trackDat.Stimulus*, and initialize it to *false*. It has been defined as number of ROIs in length since each tracked individual will receive stimuli independently. It is worth noting that Autotracker requires raw data fields to have the same dimensions and be of the same data type on each frame. We also define a field, *trackDat.CenterDistance* that will not be saved since it is unlisted in *trackDat.fields*.

Tracking Loop

The main tracking loop consists of six core sub-routines essential to any experiment in Autotracker:

1. update time-keeping variables
2. query the next frame of data from camera or video file
3. track objects in the current frame and match identities to previous frames
4. write each field of raw data in *trackDat.fields* to the hard drive
5. check to see if the background reference needs to be updated
6. update the display if necessary

Most custom experiments can be accomplished without altering any of the core sub-routines. In most cases, a single sub-routine between steps 3-4 to define in custom data fields and implement hardware control. Between steps 3-4, objects have already been tracked and have had their identities matched to traces from previous frames, but the data has not yet been written to the hard drive(fig. 6.1.2). This makes it simple to write a single function to filter existing fields or define custom fields for raw data saving. In this example, a function (*customStimulusSubRoutine*) has been included to find tracked objects within some threshold distance to the center of its ROI (fig. 6.1.3). The result is stored in the raw data field *trackDat.Stimulus* and is used to filter which ROIs receive a stimulus.

Initialization | **Tracking Loop** | Clean-up

```
% run experimental loop until duration is exceeded or last frame
% of the last video file is reached
while ~trackDat.lastFrame

    % update time stamps and frame rate
    [trackDat] = autoTime(trackDat, expmt, gui_handles);

    % query next frame and optionally correct lens distortion
    [trackDat,expmt] = autoFrame(trackDat,expmt,gui_handles);

    % track, sort to ROIs, evaluate noise level, output optional fields
    trackDat = autoTrack(trackDat,expmt,gui_handles);

    % ---------------------- %
    % insert custom routines %
    % ---------------------- %
    trackDat = customStimulusSubRoutine(trackDat,expmt);

    % output data tracked fields to binary files
    [trackDat,expmt] = autoWriteData(trackDat, expmt, gui_handles);

    % update reference or reset if noise thresh is exceeded
    [trackDat, expmt] = autoReference(trackDat, expmt, gui_handles);

    % update the display data
    trackDat = autoDisplay(trackDat, expmt, imh, gui_handles);

end
```

figure 6.1.2 - Custom routines (highlighted) can be called between steps 3-4 to define custom data fields and implement hardware control.

Custom Routine

```
% Example custom tracking sub-routine
function trackDat = customStimulusSubRoutine(trackDat,expmt)

    % calculate distance of each object to center of ROI
    dx = diff(trackDat.Centroid(:,1) - expmt.ROI.centers(:,1));
    dy = diff(trackDat.Centroid(:,2) - expmt.ROI.centers(:,2));
    trackDat.CenterDistance = sqrt(dx.^2 + dy.^2);

    % find animals close enough to center for stimulus delivery
    % store result in trackDat.Stimulus for writing to hard drive
    thresh = expmt.parameters.distance_thresh;
    trackDat.Stimulus = trackDat.CenterDistance < thresh;

    % target the ROIs specified by trackDat.Stimulus
    deliverStimulus(trackDat.Stimulus,expmt);

end
```

figure 6.1.3 - Example custom routine where our raw data field is assigned prior to writing data to the hard drive. Distance to center is calculated and compared to a threshold value (defined outside of the run file our during initialization). The data is then passed to a hypothetical function that delivers stimuli to the ROIs specified by *trackDat.Stimulus*.

Clean-up

The clean-up routine will be largely the same for all tracking experiments and consists primarily of the *autoFinish* sub-routine. The *autoFinish* sub-routine executes by default unless flagged by the user by selecting ***Stop*** > ***Delete***, which removes all data files and directories associated with the experiment. During clean-up, the expmt master struct is updated and re-saved before analysis, raw data files are closed, temporary display objects are removed, the camera is stopped, and outputs to the run file are assigned. Insert code for shutting down external hardware or cleaning up the UI here.

Initialization | Tracking Loop | Clean-up

```
% wrap-up if finish flag is set
if expmt.Finish

    % % auto process data and save master struct
    expmt = autoFinish(trackDat, expmt, gui_handles);

end

for i=1:nargout
    switch i
        case 1, varargout(i) = {expmt};
        case 2, varargout(i) = {trackDat};
    end
end
```

figure 6.1.4 - The finish sub-routine backups the *expmt* .mat file, cleans the UI, shuts down open hardware sessions, and closes raw data files.

7.2 Analyze file template

The analysis template consists of two core sub-routines useful for interfacing with Autotracker data files: *autoDataProcess* and *autoFinishAnalysis*.

autoDataProcess

- parses name-value pairs for data processing options
- initializes the processing *options* struct
- initializes memmap objects for accessing the raw data files
- calculates and saves trace features (*optional*)
  - speed
  - heading angle
  - four quadrant inverse tangent
  - distance to ROI center
- parses traces into discreet movement bouts (*optional*)
- corrects speed data for fisheye lens distortion (*optional*)

autoFinishAnalysis

- re-saves *expmt* struct following analysis
- closes open raw data files
- resets the *expmt* struct for next experiment

Analyze File

```
function expmt = analyze_basictracking(expmt,varargin)

% This function provides a sample analysis function to run after the
% run_basictracking.m. It takes the experimental master data
% struct (expmt) as an input, processes the data to extract features
% and store them to file.

% Parse inputs, initialize raw data memmaps, format in master struct,
% process centroid data
[expmt,options] = autoDataProcess(expmt,varargin{:});

% --------------------------- %
% insert custom analysis here %
% --------------------------- %

clearvars -except expmt options

% Update and save expmt master struct, close open raw data files,
% and reset the expmt master struct for new experiment
autoFinishAnalysis(expmt,options)
```

figure 6.1.4 - The finish sub-routine backups the *expmt* .mat file, cleans the UI, shuts down open hardware sessions, and closes raw data files.

8 Visual Stimulus Delivery with Psychtoolbox

Autotracker supports closed-loop targeting of visual stimuli to individual ROIs using an external display or projector. Visual stimuli can be crafted and displayed with Psychtoolbox, a freely-available, MATLAB based software package for psychology and neuroscience research. Psychtoolbox is not included with Autotracker and must be downloaded and installed separately. Psychtoolbox is an incredibly powerful and expansive package developed by Mario Kleiner. See one of the many online resources to learn how to generate and display stimuli.

8.1 Projector box

Mounting an overhead projector to the behavioral box allows targeting of visual stimuli to individual ROIs. The camera can be placed slightly off-center so the behavioral platform is visible to both the camera and projector. Equipping a long-pass filter to the camera allows tracking done in infrared without interference from the projector (fig. 8.1).

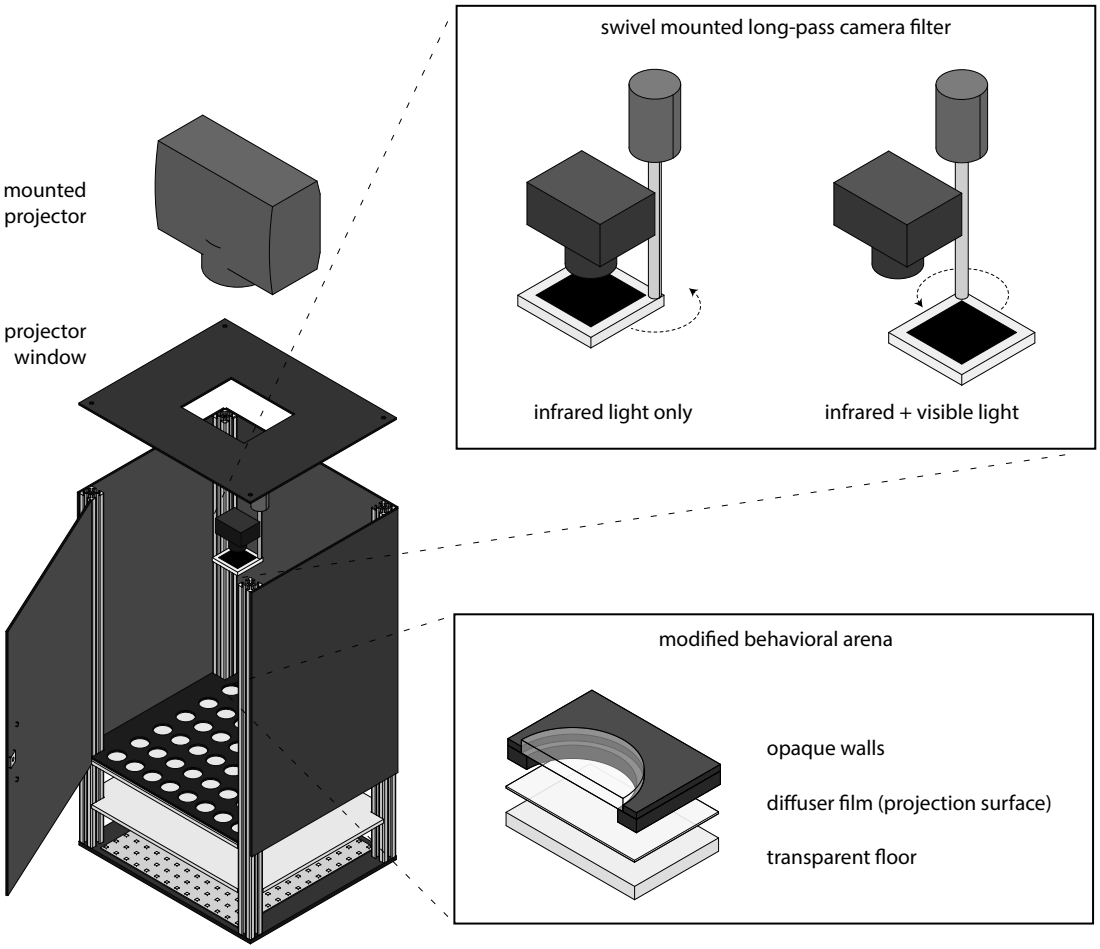


figure 8.1 - Sample schematic of projector behavioral box. An external long-pass filter can be rotated in and out of the imaging path allowing the camera to switch between imaging only infrared light and imaging both infrared and visible light. The overhead projector targets individual ROIs underneath by projecting visual stimuli on a diffuser film on the floor of the behavioral arenas.

In order to accurately target objects within the camera's field of view with the projector, a mapping must be made between the pixel coordinates of both the camera and projector. Autotracker creates this mapping by tracking a spot that is rastered through the projector's display. For this to work, the projection must be visible to the camera during registration but invisible to the camera during tracking. Mounting a long-pass filter on a swivel makes it easy to switch between the two without disturbing the camera.

8.2 Registering the projector to the camera

Any coordinate mapping between the camera and the projector will only accurate for a particular configuration and positioning of both. If either is moved during or after registration, the process must be started over again to have an accurate mapping. Having a high contrast, non-reflective projection surface will greatly improve the ease of tracking the projector with the camera. A flat white sheet of paper or matte white painted surface works well. Do the following to register the projector in Autotracker:

1. Ensure that Psychtoolbox is installed by executing ***PsychtoolboxVersion*** in the MATLAB command window (Autotracker support built on version 3)
2. Remove the infrared filter from the imaging path and start the camera preview
3. Place a high-contrast, white surface on the behavioral platform
4. Ensure no sources of illumination but the projector will be visible to the camera
5. Drag a window into the projector display (anything with high-contrast black and white text) and ensure the projector is in focus
6. Adjust the camera aperture or exposure (under camera settings) and focus until the projector display is clearly visible in the camera preview
7. Select ***Hardware*** > ***projector*** > ***register projector***
8. Select and confirm the projector screen and registration parameters when prompted to begin registration
    - *Grid Step Size* sets how finely the projector rasters its field of view
    - *Spot Radius* sets the size of the rastered dot (the apparent size should be between 1-5% the area of the camera)
9. Once registration is finished, the output mapping will be saved to file under: ***~autotracker/hardware/projector_fit/***

8.3 Using registration output in experiments

Once registration is complete, no additional configuration of the projector or registration mapping is required to run the prepackaged ***Optomotor***, ***Slow Phototaxis***, and ***Temporal Phototaxis*** experiments. To use the mapping in custom experiments, run the following command:

expmt = initialize_projector(expmt, bg_color)

This initializes a Psychtoolbox window to the screen set under registration parameters with the background color specified by the RGB triplet, *bg_color*. Additionally, projector initialization stores the registration mapping and screen properties to the *expmt* struct. To convert from camera coordinates to projector coordinates, use the 2D scattered interpolants stored under *expmt.projector*:

Name	Description
screenNumber	Display number of the screen set under *Hardware > projector > set registration parameters*
window	Reference to the open Psychtoolbox window object
windowRect	Pixel dimensions of the screen in the format [0, 0, width, height]
xCenter	x-coordinate of the screen center
yCenter	y-coordinate of the screen center
black/white	screen color index of black and white
vbl	high-precision vertical retrace time stamp for when last stimulus was flipped to the screen
ifi	inter-frame interval of the screen (inverse of refresh rate)
waitframes	number of frames to wait before flipping stimuli to screen

Table 8.3 - Field names and descriptions of the screen properties output to *expmt.scrProp* following projector initialization. The properties stored here be used to accurately time and display stimuli to the Pyschtoolbox window opened during initialization.

projector_x = expmt.projector.Fx(cam_x, cam_y);
projector_y = expmt.projector.Fy(cam_x, cam_y);

Detailed use of Autotracker and Psychtoolbox to deliver stimuli under closed-loop control is beyond the scope of this manual, but sample implementations are available in ***run_optomotor.m*** and ***run_slowphototaxis.m***.

9 Tracking with Video Input

Autotracker accepts video files as an input source for tracking. Select *Source* > *video file* to switch inputs. Many programs will output raw video data in *avi2* format, which has a 2GB file size limit. When the file size limit is reached, subsequent files are automatically generated. To allow sequential tracking across multiple files, Autotracker will automatically track all videos under the target directory. The software makes the assumption that all files under the same directory belong to the same experiment and will only generate a single .mat output file. The software can take *.avi*, *.mp4*, *.m4v*, *.mov*, *.wmv* and *.mpg* file formats as input. Because video files are not tracked in real time, the time-stamps output from video file tracking are inferred from the frame rate of the file and may not be accurate.