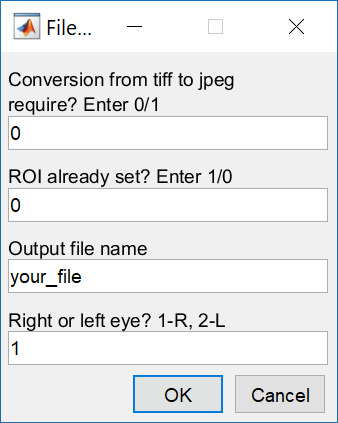
Documentation

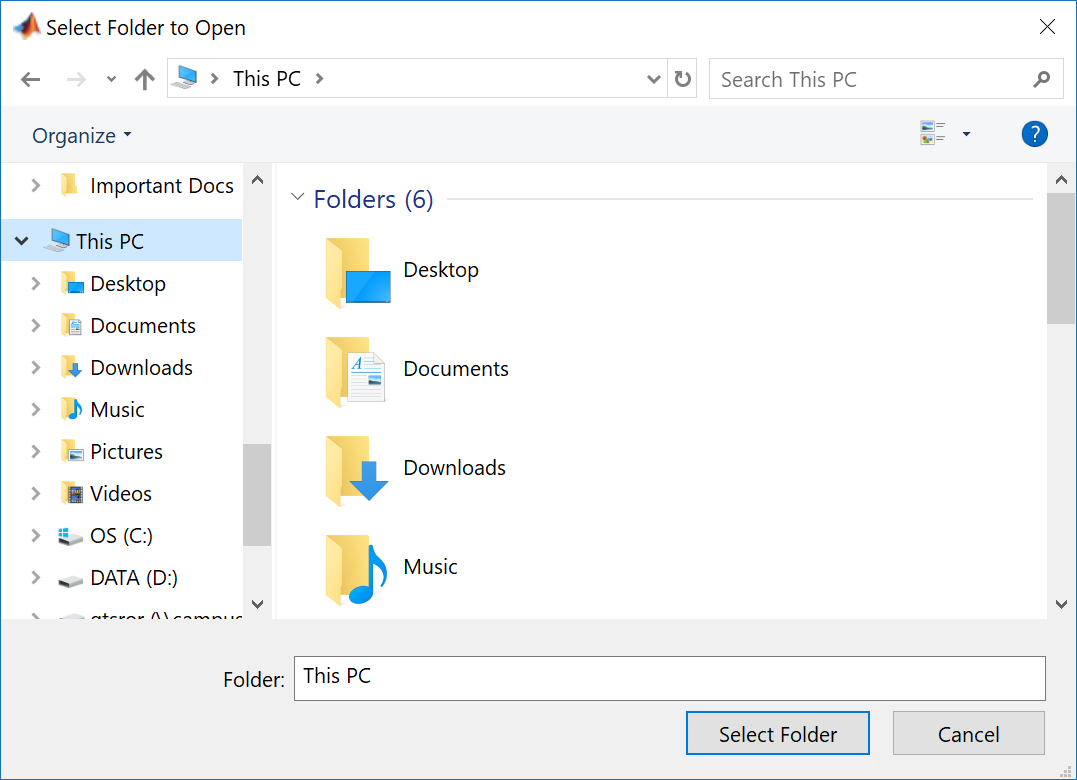
# Phase 1 – Load a video (recorded and saved as a stack of .tiff files)

This phase is required since MATLAB cannot handle raw .tiff files and they must be converted to jpeg for loading into the workspace.

1. **User input** is required to determine whether conversion is required in the following dialog:



* *For the first field - 0 – conversion not required; 1 – conversion required*
* *For the second field – 0 – ROI was not set before; 1 – ROI was set beforehand*
* *Third field – user to input the desired file name. Should describe the video analyzed*
* *Fourth field – user to input 1 or 2 to describe the eye analyzed. 1 – right eye; 2 – left eye.*
* *Fifth field – indicating if an output video is required. 1 – yes; 0 - no*
  1. In case **conversion is required**, function tiff2jpg is called.
* User will have to select the directory containing the tiff files.
* User must make sure a new empty folder with the same name, but with suffix “*\_jpegs”*, exists in the parent directory. For example, if the folder containing the tiffs is named “video1\_tiff”, a folder named “video1\_jpegs” should be in the parent directory of the tiff folder.
* A popup box will require the user to select the tiff files directory:



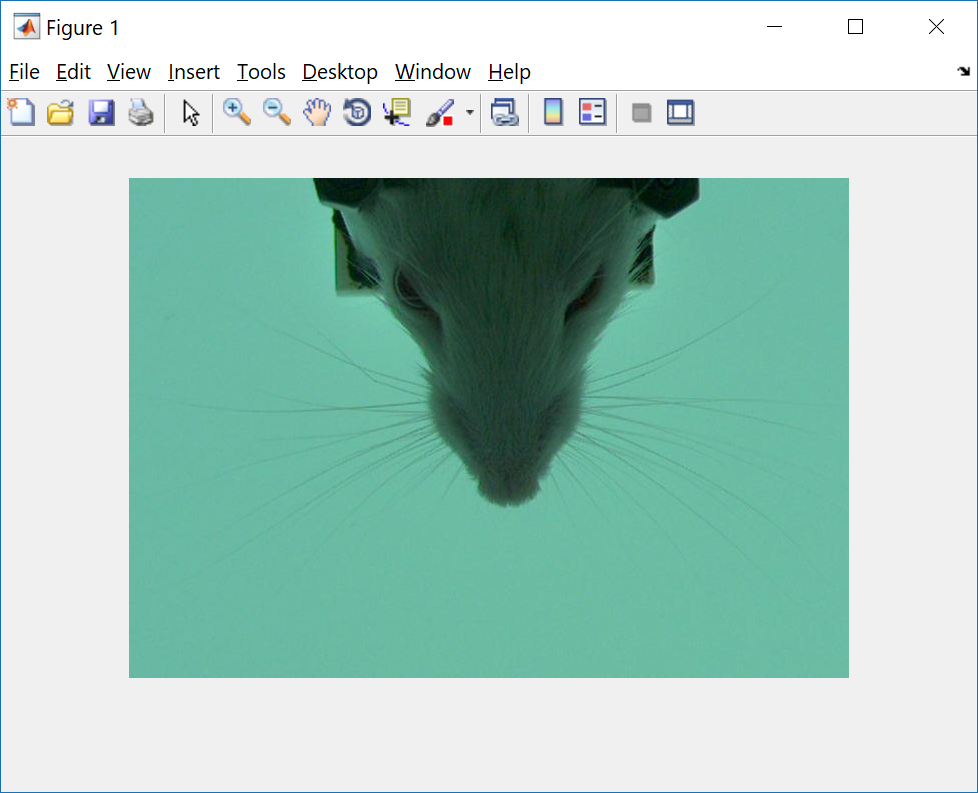
* All jpeg files will be saved in the “\_jpegs” folder.
* The conversion process uses an external function nestedSortStruct (created by Jack Hughey, 2010, downloaded from [MathWorks File Exchange](https://www.mathworks.com/matlabcentral/fileexchange/28573-nestedsortstruct)).
* An output .mat file containing the sorted indices of the new files will be added to the \_jpegs folder.  
  1. Conversion **will not be required** in case all tiff files have been converted to jpeg in the past.
* In this case, the user will be prompted to pick the \_jpegs directory containing all files in .jpeg format.
* The folder must contain indexing .mat file named sortedStruct.mat. This file will be loaded to the workspace to organize frame input later on.

# Phase 2 – processing of first frame

During this phase, the first frame of the video will be loaded using the indexing matrix from previous phase, as well as the user input regarding ROI.

2.1. In case no ROI was defined before (user input = 0):

* The user will define the ROI in a dialog box:



* The user will then define the initial reference contour, by drawing a closed polygon around the eye in the provided interface. This interface is launched by eye\_edge.
* After definition of eye contour, all parameters needed are extracted (including area, angle, coordinates etc.). This is saved to a .mat file to allow quick access to the same video later on.

2.2. In case an ROI has been defined already, the user will be prompted to select an input file. Here, the user will select an *\_init.mat* file that was generated in the past.

# Phase 3 – Active contour tracking

In stage 3, tracking begins automatically.

The code has one of two possible setups – one that enables video output showing the traced eye contour and ellipse (but takes exponentially longer to run), and another that does not. Both setups will produce the output signal in a mat file.

One .jpeg file will be loaded at a time to reduce memory usage.

While videos were recorded at 500fps, blink tracking can be properly done with 250fps. Therefore, we only sample every other frame during this process. In case recording frame rate changes, this **must be modified** in the main file (eyelid\_tracking) line **234** and **336**.

The function countour\_track is called, to initiate edge tracking. This function is the main module of the algorithm.

Within countour\_track, several other functions are called. All functions are documented and explained in MATLAB’s documentation pages (including relevant\_eye, extract\_ellipse, plot\_ellipse)

Upon finishing the tracking module, several flags (described in the code clearly) will mark a case where a full blink is detected, and initiate a reverse run of the algorithm for a certain amount of frames (described in the thesis fully).

Important to note here, the default number of frames to skip was set at **140 frames**, when sampling rate was around **500fps**. In case camera / recording settings are changed, this **must be altered as well**!

# Phase 4 – Output post processing

During this phase, NaNs are interpolated to create a smooth output signal. Keep in mind, no more than a single NaN is allowed in a row, making interpolation unlikely to change the output signal significantly.

This phase uses an external, open source function naninterp (created by E. Rodriguez, 2005, downloaded from [MathWorks File Exchange](https://www.mathworks.com/matlabcentral/fileexchange/8225-naninterp)).

The following outputs will then be saved into a .mat file:

* Minor axis after interpolation
* Minor axis before interpolation
* Eye area after interpolation
* Eye area before interpolation
* Center coordinates signal for each frame