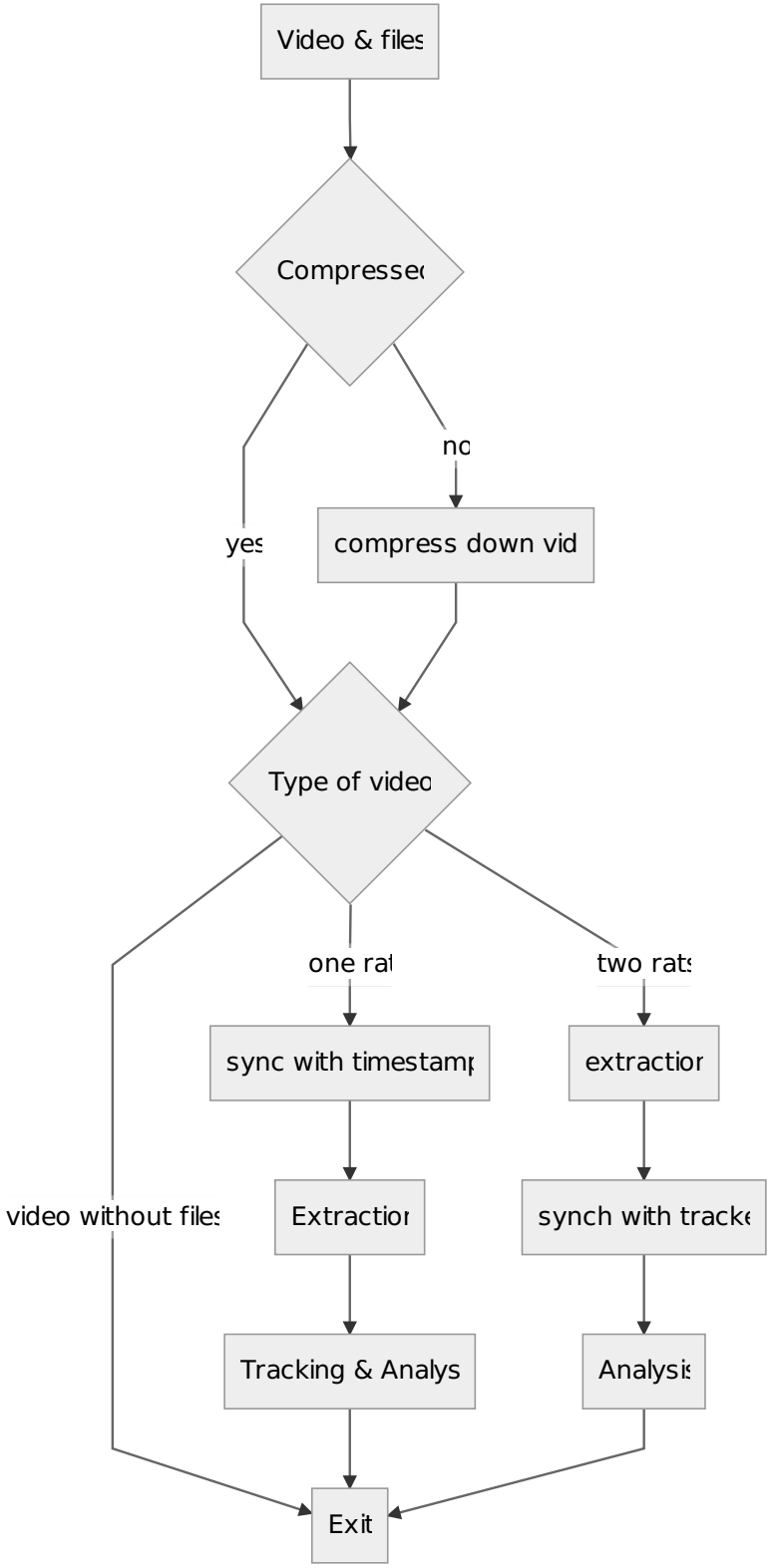
Pipeline Documentation



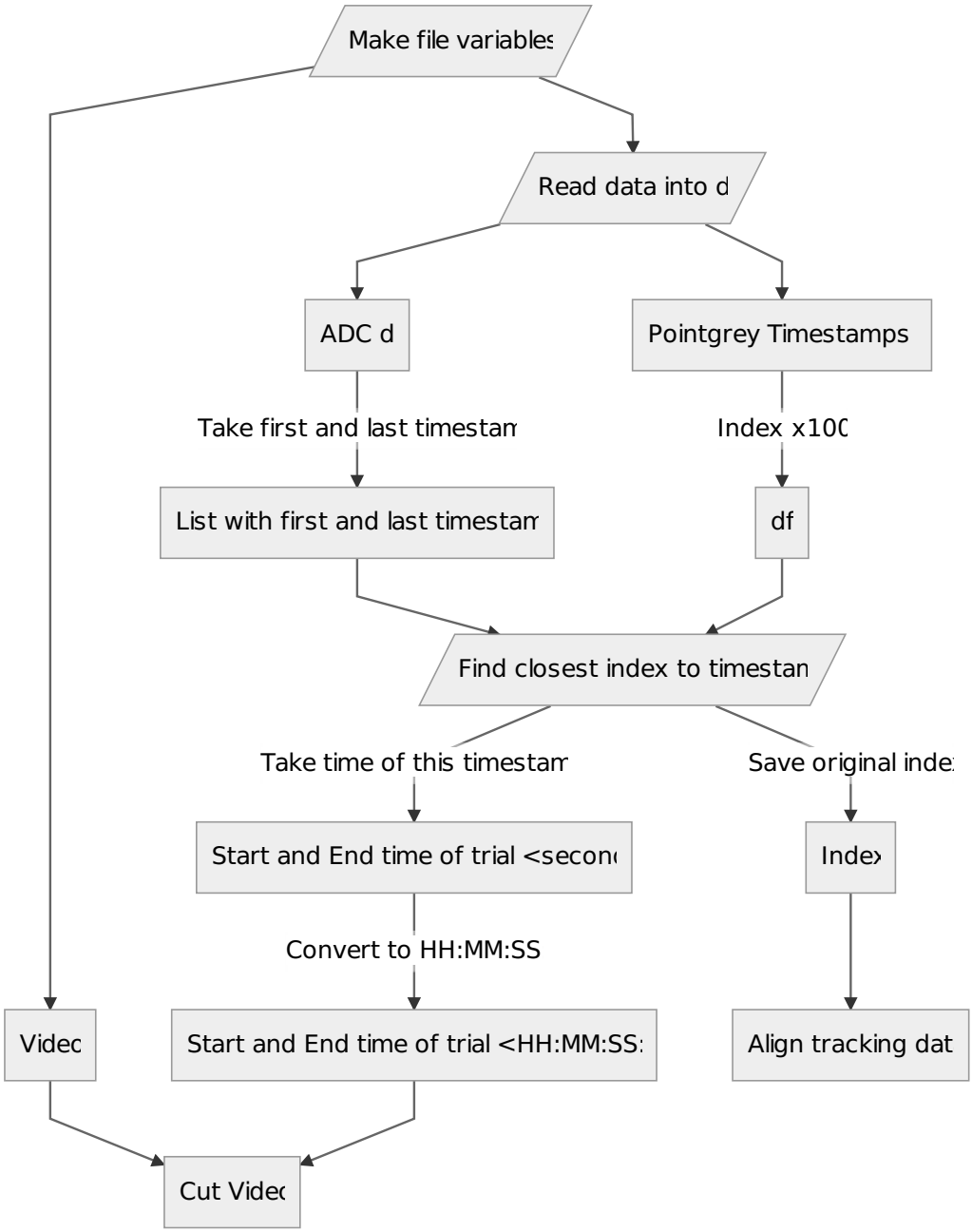
Compression

```
import os

def  find_downsampled():
 all = []
 for root, dirs, files in os.walk("/content/drive/My Drive/Stage/"):
  for  file  in files:
   if  file.endswith(".avi"):
    if  'downsampled'  in  file:
     all.append(os.path.join(root,  file))
 return all

def  find_finished_total():
 total = 0
 finished = 0
 for root, dirs, files in os.walk("/content/drive/My Drive/Stage/"):
  for  file  in files:
   if  file.endswith(".avi"):
    if  'downsampled'  not  in  file:
     total += 1
     file_z = file.replace('.',  'downsampled.')
     if os.path.join(root, file_z)  in all:
      finished += 1
 return finished, total

def  downsample_stuff(all, finished, total):
for root, dirs, files in os.walk("/content/drive/My Drive/Stage/"):
 for  file  in files:
  if  file.endswith(".avi"):
   old_file = os.path.join(root,  file)
   already_in = False
   if  'downsampled'  not  in  file:
    file_z = file.replace('.',  'downsampled.')
    if os.path.join(root, file_z)  in all:
     already_in = True
    if already_in is  False:
     old_file = os.path.join(root,  file)
     file_l = file.split('.')
     file_l[1] = 'downsampled'
     file_l.append('.avi')
     file = ''.join(file_l)
     new_file = os.path.join(root,  file)
     !ffmpeg -i "$old_file" -filter:v scale=-1:256 -c:a copy "$new_file"
     finished += 1
     print(total,  '/', finished)
     os.remove(old_file)

    if already_in is  True:
```

```
     os.remove(old_file)



all = find_downsampled()
finished, total = find_finished_total()
downsample_stuff(all, finished,total)
print(total,  '/', finished)
print(total,  '/',  len(all))

# everything downsampled is "done" so -- all = done
# everything not downsampled is to do -- so create list with that
# everything everything is all avi videos -- so create list with that
# total = all non downsampled videos
# all = all downsampled videos
# finished = all already downsampled videos
```

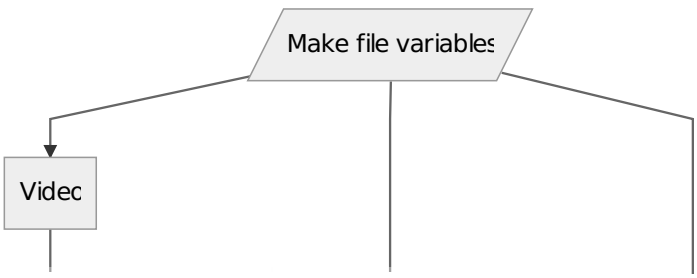
Synchronization

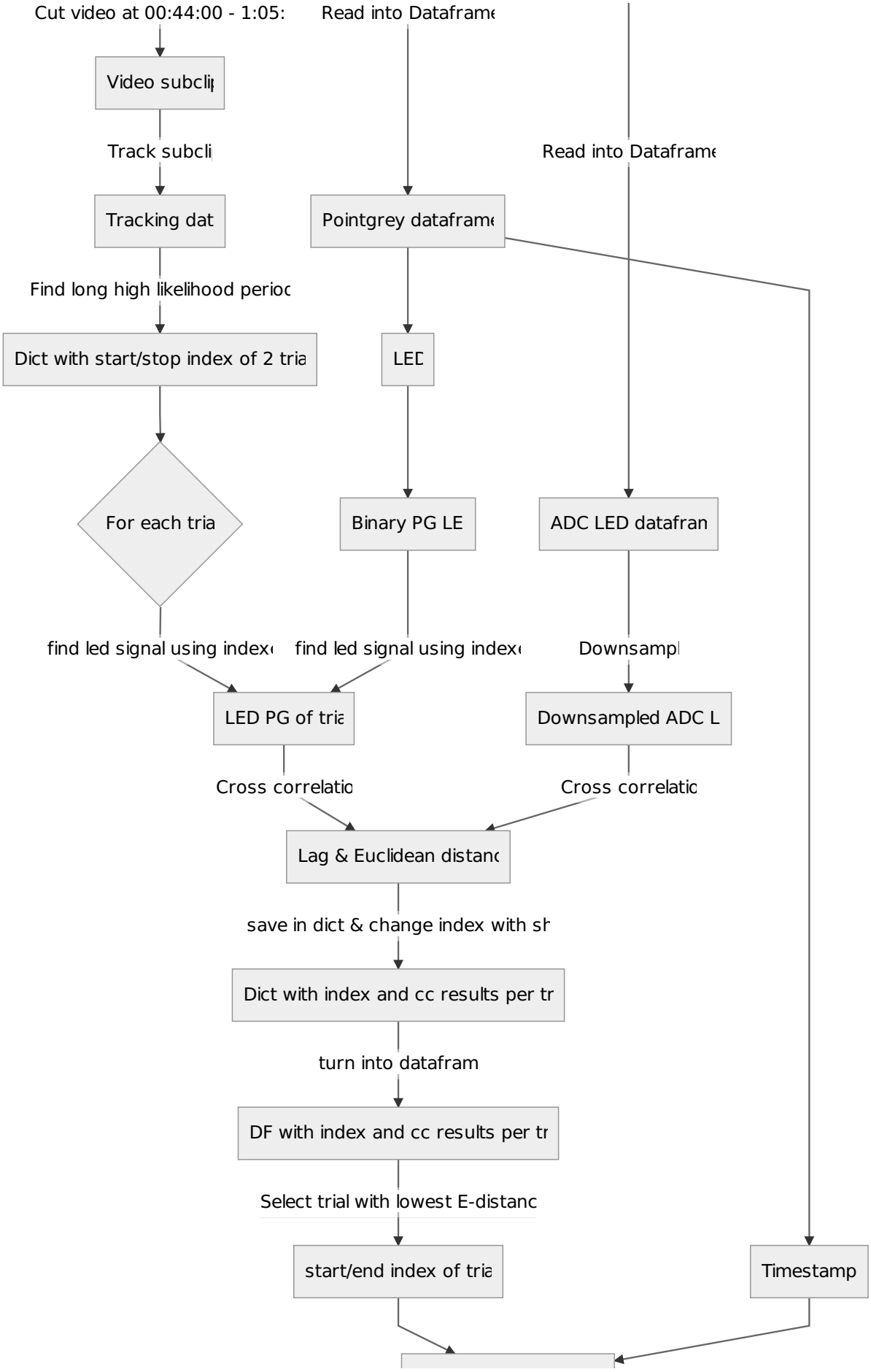
synchronization with timestamps

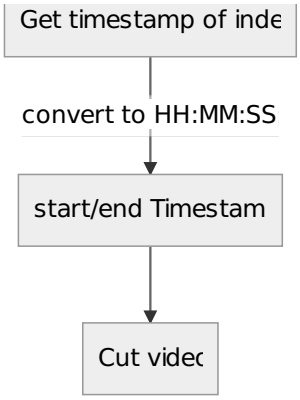


synchronization with tracker

An example for trial 1:




Cut video at 00:44:00 - 1:05:
Read into Datafram
Video subcli
Track subcli
Read into Datafram
Tracking dat
Pointgrey datafram
Find long high likelihood perioc
Dict with start/stop index of 2 tria
LED
For each tria
Binary PG LE
ADC LED datafran
find led signal using index
find led signal using index
Downsampl
LED PG of tria
Downsampled ADC L
Cross correlatio
Cross correlatio
Lag & Euclidean distanc
save in dict & change index with sh
Dict with index and cc results per tr
turn into datafram
DF with index and cc results per tr
Select trial with lowest E-distanc
start/end index of tria
Timestamp



Code for functions in flowcharts:

Make file variables

```
ADC_file = ADC_loc+'/100_ADC'+ADC+'_0.continuous'

Pointgrey_files = make_pg_files(date_serial)
FRAME_file = Pointgrey_files[0]
TIME_file = Pointgrey_files[1]
LED_file = Pointgrey_files[2]
VIDEO_file = Pointgrey_files[3]
```

Read into dataframe

```
# load in data as dataframes

# loading of ADC using openEphys.py
data = load(file_path+ADC_file)
DF_ADC_samples, DF_ADC_timestamps = load_data_into_dataframe(data)

# Reading in of the Pointgrey files
DF_p_frames = pd.read_csv(file_path+FRAME_file, header=None)
DF_p_timestamps = pd.read_csv(file_path+TIME_file, header=None)
DF_p_led = pd.read_csv(file_path+LED_file, header=None)

# conversion of binary timestamps to seconds
DF_p_timestamps = convert_and_uncycle_timestamps(DF_p_timestamps)
DF_p_timestamps = DF_p_timestamps.apply(lambda x : x - DF_p_timestamps[0][0])
```

Downsample (function)

```
def  downsample_ADC_led(DF_ADC_samples):
 DF_ADC_samples.rename(columns={0  :  'value'}, inplace=True)
 DF_ADC_samples['state'] = 0
 DF_ADC_samples.loc[DF_ADC_samples.value.between(0,1),  'state'] = 0
 DF_ADC_samples.loc[DF_ADC_samples.value.between(1,5),  'state'] = 1

 group_size = 1000
 lst = [DF_ADC_samples.iloc[i:i+group_size]  for i in  range(0,len(DF_ADC_samples)-group_size+
 downsampled = []
 for df in lst:
  m = df["state"].iloc[-1]
  downsampled.append(m)
 downsampled_DF = pd.DataFrame(downsampled)

return downsampled_DF
```

Index x 1000 & taking first and last timestamp

```
# find the starting timestamp and stopping timestamp in the continuous files
s = DF_ADC_timestamps[0][0]
st = DF_ADC_timestamps.iloc[-1][0]

# multiply this number by 1000 to make it comparable to
# the continuous timestamps
# (continuous file timestamps are sample numbers)
DF_p_timestamps['index1'] = DF_p_timestamps.index*1000
```

Find Closest value to index

```
c_s = DF_p_timestamps.iloc[(DF_p_timestamps['index1']-s).abs().argsort()[:1]].index.tolist()
c_st = DF_p_timestamps.iloc[(DF_p_timestamps['index1']-st).abs().argsort()[:1]].index.tolist()
```

Conversion s to HH:MM:SS:ff

```
start = pd.to_datetime(DF_p_timestamps[0].iloc[c_s[0]+phase], unit='s').strftime('%H:%M:%S.%f'
stop = pd.to_datetime(DF_p_timestamps[0].iloc[c_st[0]+phase], unit='s').strftime('%H:%M:%S.%f'
```

Cutting video for subclip

```
b = DF_p_timestamps.loc[DF_p_timestamps[0] > 2650]
e = b.loc[b[0] > 3900]

begin= b.iloc[1][0]
end= e.iloc[1][0]

start= pd.to_datetime(begin, unit='s').strftime('%H:%M:%S.%f')
stop= pd.to_datetime(end, unit='s').strftime('%H:%M:%S.%f')
cut(video_file_path=file_path+VIDEO_file, start_time= start, end_time=stop)
```

Cut (function)

```
def  cut(video_file_path, start_time, end_time):
 """
 cuts the video using ffmpeg using the start and end time and the video
 Keyword arguments:
 video_file_path -- full path to the video
 start_time -- string of time in HH:MM:SS:ff format
 end_time -- string of time in HH:MM:SS:ff format
 """
 output_file_path = re.search("^[\/].+\/", video_file_path)
 output_file_path_raw = output_file_path.group(0)
 delsplit = re.search("\/(?:.(?!\/))+$", video_file_path)
 filename = re.sub("^[\/]",  "", delsplit.group(0))
 filename_raw = re.sub(".{4}$",  "", filename)
 file_extension = re.search(".{3}$", filename)
 file_extension_raw = file_extension.group(0)

 os.environ['inputFile'] = video_file_path
 os.environ['outputPath'] = output_file_path_raw
 os.environ['startTime'] = start_time
 os.environ['endTime'] = end_time
 os.environ['fileName'] = filename_raw
 os.environ['fileExtension'] = file_extension_raw

 !ffmpeg -hide_banner -i "$inputFile" -ss "$startTime" -to "$endTime" -c copy "$outputPath"/"$
```

Tracking of subclip

```
ProjectFolderName = 'Stage/downsampled_tracker-sanne-2021-09-16'
VideoType = 'avi'
path_config_file = '/content/drive/My Drive/'+ProjectFolderName+'/config.yaml'
t_video = file_path+VIDEO_file
t_video = t_video.replace('.avi',  '-TRIM.avi')
deeplabcut.analyze_videos(path_config_file, t_video, save_as_csv=True)
```

Finding periods of high likelihood

```
DLCscorer='DLC_resnet50_downsampled_trackerSep16shuffle1_322500'
a_file = t_video.replace('.avi', DLCscorer+'.h5')
#loading output of DLC
Dataframe = pd.read_hdf(a_file)
df = Dataframe['DLC_resnet50_downsampled_trackerSep16shuffle1_322500']['Head']['likelihood'].t

trials = {}
# Find all values that have a small difference between -0.25 and 0.25 as well as a likelihood
# then groupby to only find this pattern consecutivly and count it up
x = df[(df['likelihood'].diff().between(-0.25,0.25))&(df['likelihood'].between(0.99,1.0))].gro

# for all the newly made groups with the previous criteria only keep the longest ones
# as these should correlate with the trials
for k, v in x:
 if  len(v) > 1000:
  trials[k] = [v.index[1], v.index[-1]]
```

Correcting Index & saving

```
# k = group number, v = list with start and stop
# y = index value
# b = cut index value

num = 0
trial_dic = {}

for k, v in trials.items():
 num = num+1
 trial_dic[num] = []
 for y in v:
  trial_dic[num].append(y+b_index)
```

Binarizing LED

```
DF_p_led.rename(columns={0:  'value'}, inplace=True)
DF_p_led['state'] = 0
DF_p_led.loc[DF_p_led['value'] <= 500,  'state'] = 0
DF_p_led.loc[DF_p_led['value'] > 500,  'state'] = 1
```

For loop Cross correlation

```
for k in trial_dic.keys():
 whole = DF_p_led.state.iloc[trial_dic[k][0]: trial_dic[k][1]].to_numpy()
 trial = DF_ADC_Led[0].to_numpy()
 shift,e_distance = compare_and_plot_signals_with_alignment(whole, trial, bshift_method = 'all
 print(len(whole)-len(trial))
 trial_dic[k][0] = trial_dic[k][0] + shift
 trial_dic[k][1] = trial_dic[k][1] + shift + abs(len(whole)-len(trial))
 trial_dic[k].append(shift)
 trial_dic[k].append(e_distance)
 # value connected to key now looks like : [start index, end index, shift, elucedian distance]
```

Correction with cross correlation results

```
df = pd.DataFrame.from_dict(trial_dic, orient='index',
columns=['begin','end','shift',  'elucidean distance'])
correct_trial_index = df['elucidean distance'].idxmin()
begin_shift = DF_p_timestamps.iloc[trial_dic[correct_trial_index][0]].values
end_shift = DF_p_timestamps.iloc[trial_dic[correct_trial_index][1]].values
start= pd.to_datetime(begin_shift[0], unit='s').strftime('%H:%M:%S.%f')
stop= pd.to_datetime(end_shift[0], unit='s').strftime('%H:%M:%S.%f')

cut(video_file_path=file_path+VIDEO_file, start_time= start, end_time=stop)
```

Extraction

cuts the video using ffmpeg using the start and end time and the video
Keyword arguments:
video_file_path – full path to the video
start_time – string of time in HH:MM:SS:ff format
end_time – string of time in HH:MM:SS:ff format
trial – string that looks like “trial#”

```
import os, re

def  cut(video_file_path, start_time, end_time,trial):
 output_file_path = re.search("^[\/].+\/", video_file_path)
 output_file_path_raw = output_file_path.group(0)
 delsplit = re.search("\/(?:.(?!\/))+$", video_file_path)
 filename = re.sub("^[\/]",  "", delsplit.group(0))
 filename_raw = re.sub(".{4}$",  "", filename)
 file_extension = re.search(".{3}$", filename)
 file_extension_raw = file_extension.group(0)
 # turn into os environment variables (so we can use them in the command like $variable)
 os.environ['inputFile'] = video_file_path
 os.environ['outputPath'] = output_file_path_raw
 os.environ['startTime'] = start_time
 os.environ['endTime'] = end_time
 os.environ['fileName'] = filename_raw
 os.environ['fileExtension'] = file_extension_raw
 os.environ['trial'] = trial
 !ffmpeg -hide_banner -i "$inputFile" -ss "$startTime" -to "$endTime" -c copy "$outputPath"/"$
```

Tracking & Analysis

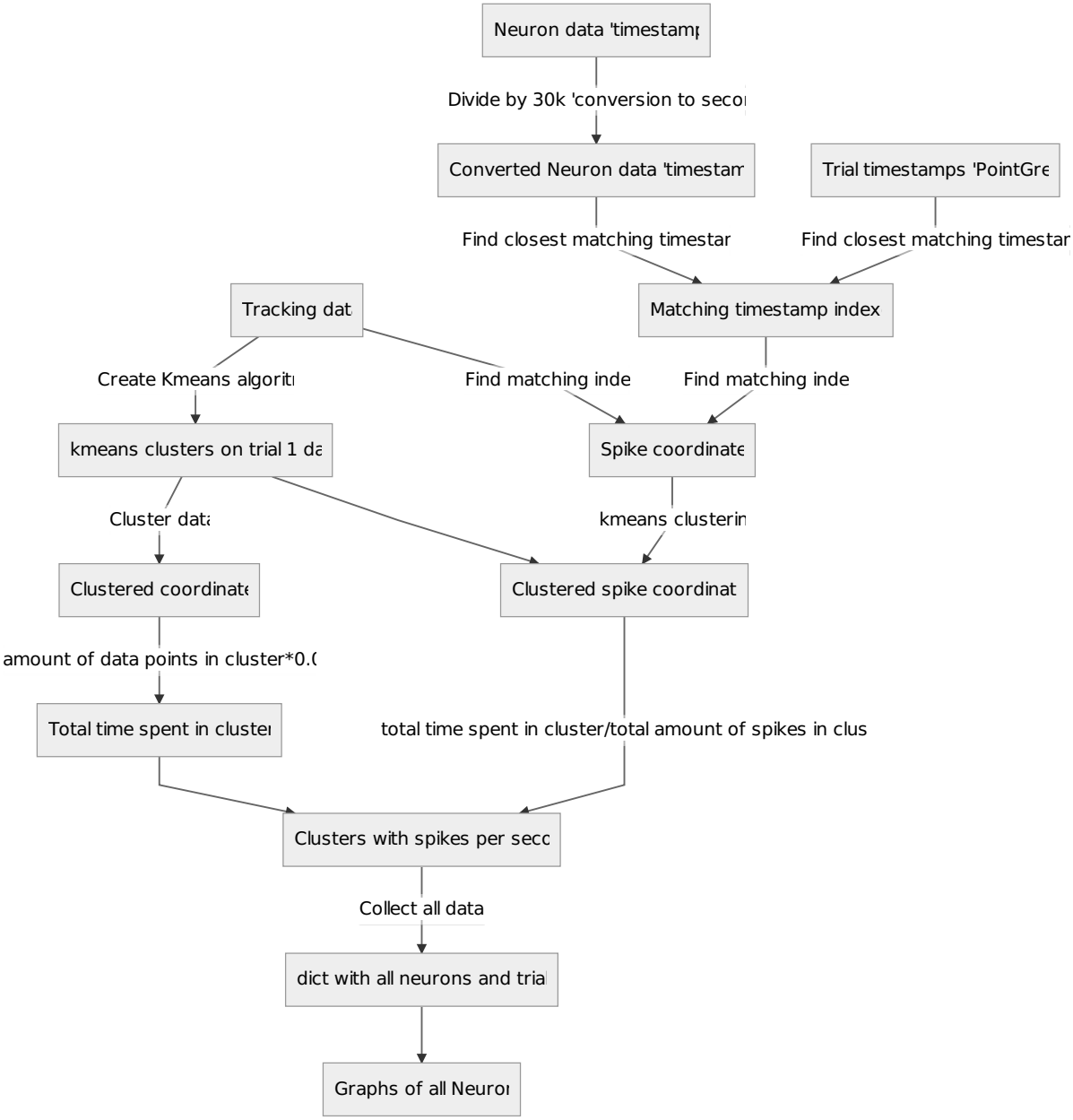
How to setup and use a DLC project:

1. Install Deeplabcut (Documentation on this here)
    - It's possible you'll run into this error `failed build due to wxPython` An explanation on how to solve this can be found here
    - There is also a possibility that you will need to download a wheel to get it working on linux/ubuntu (information on this is mentioned in the deeplabcut documentation.
2. Start the env `conda activate DEEPLABCUT`
3. Click on create project
    - Add the videos you want to label
    - No need to change settings
4. Now open the config file created
    - Fill in the bodyparts you want to track under 'bodyparts:'
    - Fill in the skeleton connections that would be usefull to you under 'skeleton:' `- -` in front of the body part 1 and then `-` in front of the body part you want to connect to body part 1
5. Now you can extract frames from the videos
    - You can extract them manually (which is best but takes time) if you do so make sure to add a diverse selection, many similair frames will not improve the tracker.
    - You can also simply extract them normally (there is settings in the config file for how many frames per video will be extracted `numframes2pick:`)
6. Now you can label these frames in the label frame tab
    - with left click you can remove a point you put

7. Once you have labeled everything we can move the project into colab
    - Copy paste the entire directory onto your google drive
8. Create a training dataset
9. Start training
    - The more itterations the better we should aim for around 200k+ to 500k
    - Basically go on till the loss stabelizes for a long time(this is given every itteration)
    - There is files called `pose_cfg.yaml` inside the `dlc_models` directory inside these there are loads of parameters you can change around. (none of this is necessary but some might give better results)
10. Evaluate model
    - You'll get a pixel error, the lower the better we hope to see a number like 2 to 8 in there. the lower the better
11. If its good enough then you can start using the model for analysis and make labeled videos
    - however if the result isnt good enough then either add more images or continue training for more itterations
        - click here for a forum post on how to retrain your model and start from a snapshot(previous itteration of your model)
12. Analyze videos and create plots
    - in these plots you can see for instance the likelihood, and the trajectory. If this is not what you expected (very low likelihood) you can take this video and put it in for relabeling in the GUI, this will then be added to the training set after you've relabeled. Then you can train the model again to adapt to this video.

Analysis

Neuron analysis

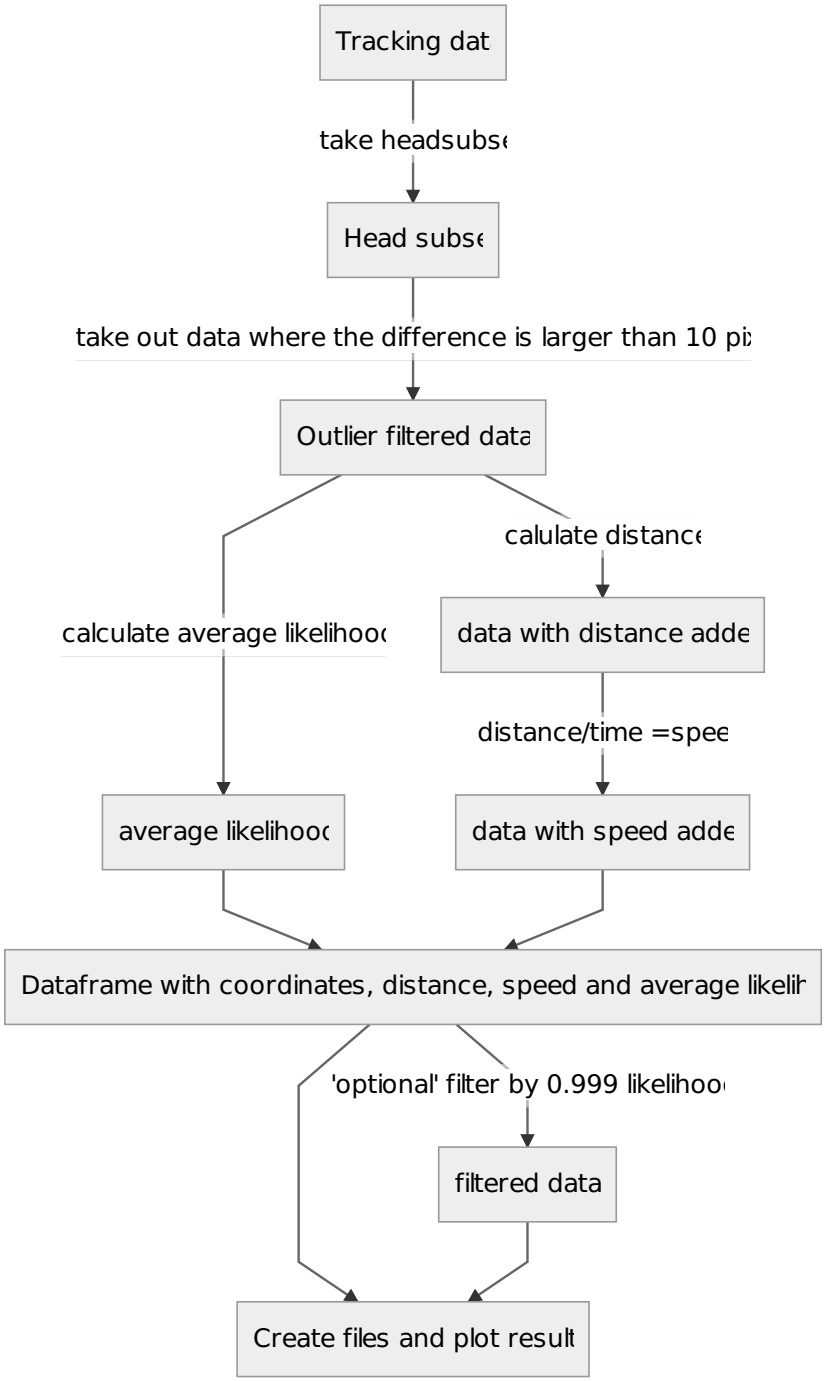



```
def  Neuron_Analysis(ADC_loc,kmeans, path,df_xy,n_clusters, Trial1_times):

 mat = 'R3SD3_{}'.format(ADC_loc)
 mat_fname = path+ADC_loc+'/R3SD3_{}.mat'.format(ADC_loc)
 pred = kmeans.predict(df_xy[['x',  'y']])
 frame = pd.DataFrame(df_xy)
```

```
# add what coordinate set belongs to what cluster
frame['cluster'] = pred

# find how many data points are in each cluster
total_frame = frame['cluster'].value_counts()

# calculate time by taking that 1 point is 1 frame and 1 frame = 0.033s
total_frame = total_frame.apply(lambda x: x * 0.033)

# centroids in necessary or just wanna check
centroids = kmeans.cluster_centers_

# show a scatter plot of the clusters and trajectory with centroids
plot = plt.scatter(frame['x'], frame['y'],
     c=frame['cluster'],
     s=50,
     alpha=0.5,
     cmap='Set2')
plot = gaussian_filter(plot, sigma=50)
plt.title('Kmeans clusters on trajectory of {}'.format(ADC_loc))
plt.gca().invert_yaxis()
plt.colorbar()
plt.scatter(centroids[:,  0], centroids[:,  1], c='red', s=50)
plt.show()

# setting up the mat file
mat_contents = sio.loadmat(mat_fname)

# receive the subset of the trial timestamp data
print(mat_fname)

# make original index into column incase its needed
Trial1_times['full_index'] = Trial1_times.index

# reset the index so it starts from 0
Trial1_times = Trial1_times.reset_index(drop=True)
neur = 0
neur_dic = {}

#fig, axarr = plt.subplots(4, 5, sharex=True, sharey=True)
for ar in mat_contents[mat][0]:
 neur = neur + 1
 neuron1 = ar

if  len(neuron1) > 1:
 neuron1_conv = [number / 30000  for number in neuron1]
 neuron1_conv_c = []

 # connect them into one array
 for l in neuron1_conv:
```

```
   neuron1_conv_c.extend(l)

  # getting the indexes of the spikes accorind to the trial video
  spike_indexes = []
  for n in neuron1_conv_c:
   x = Trial1_times.iloc[(
   Trial1_times[0]-n).abs().argsort()[:1]].index.tolist()
   if x[0] != 0:
   spike_indexes.append(x[0])

  # finding the corresponding x and y coordinates
  x_list = []
  y_list = []

  for n in spike_indexes:
   #x = df_xy['x'].iloc[n]
   x_list.append(df_xy['x'].iloc[n])
   y_list.append(df_xy['y'].iloc[n])

  # combine the lists
  data_tuples = list(zip(x_list, y_list))
  df_xy_neurons = pd.DataFrame(data_tuples, columns=['x',  'y'])


  if  len(df_xy_neurons) > 0:
   # predict in which cluster each coordinate set belongs
   pred = kmeans.predict(df_xy_neurons)
   frame = pd.DataFrame(df_xy_neurons)

   # add the cluster data to each coordinate set
   frame['clusters'] = pred
   neuron_frame = frame['clusters'].value_counts()
   neuron_frame = pd.concat([neuron_frame, total_frame], axis=1)

   # calculate the spikes per second by:
   # amount of spikes in cluster /time spent in cluster
   neuron_frame['Result'] = neuron_frame['clusters']/neuron_frame['cluster']

   # value is the value which will be assigned to each cluster based on the
   # last calculation
   frame['value'] = np.NaN
   # each cluster gets its spikes per second value which will decide the
   # color in the graph
   for i in  range(n_clusters):
    frame.loc[frame.clusters == i,  'value'] = neuron_frame.iloc[i]['Result']
    neur_dic[neur] = frame

return neur_dic, total_frame
```

Speed




```
import pandas as pd
from pathlib import Path
import numpy as np
import os
import matplotlib.pyplot as plt

def  calculateDistance(x1,y1,x2,y2):
 dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
 return dist

def  calculateSpeed(video, scorer, likelihood_filter):
 %matplotlib inline
```

```
%matplotlib inline

file = video.replace('.avi', scorer+'.h5')
#loading output of DLC
Dataframe = pd.read_hdf(file)

#Getting data of the head
head_x = Dataframe['DLC_resnet50_downsampled_trackerSep16shuffle1_322500']['Head']['x']
head_y = Dataframe['DLC_resnet50_downsampled_trackerSep16shuffle1_322500']['Head']['y']
head_l = Dataframe['DLC_resnet50_downsampled_trackerSep16shuffle1_322500']['Head']['likelihoo

speed_list = []

head_x[(head_x.diff() > 10) | (head_x.diff() < -10)] = np.nan
head_x = head_x.fillna(method='ffill')
head_y[(head_y.diff() > 10) | (head_y.diff() < -10)] = np.nan
head_y = head_y.fillna(method='ffill')

#calculate speed by calculating distance of rat between frames
#then using the distance to calculate the speed
for i in  range(len(Dataframe)-1):
 x1 = head_x[i]
 x2 = head_x[i+1]
 y1 = head_y[i]
 y2 = head_y[i+1]
 l1 = head_l[i]
 l2 = head_l[i+1]
 speed = calculateDistance(x1,y1,x2,y2)/0.033  #*0.2645833333)/10 (in case of conversion to c
 av_l = (l1+l2)/2

 #saving the data in the format
 #xy1 = x coordinate frame 1 -- y coordinate frame 1
 #xy2 = x coordinate frame 2 -- y coordinate frame 2
 #likelihood = average of likelihood between these frames
 #speed = speed calculated with distance xy1 and xy2 and time of 0.033 sec
 speed_list.append(['{x1}-{y1}'.format(x1=x1,y1=y1),'{x1}-{y1}'.format(x1=x2,y1=y2), av_l, sp
df = pd.DataFrame(speed_list, columns=['xy1','xy2','likelihood','speed'])

#filter to only keep above 0.99 likelihood
df.speed[df.likelihood < 0.999] = 0

#Write to files
df.to_csv('xy_likelihood_speed.csv')
df.speed.to_csv('speed.txt', index=False)
ax = df.speed.plot(title='Speed of rat during trial')
ax.set(xlabel='frame numbers', ylabel='speed in (cm/s)')

#Plot graph
plt.show()

return df
```