In [1]:
```
#import modules
from __future__ import division, unicode_literals, print_function  # for c
ompatibility with Python 2 and 3

import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import pims
import trackpy as tp
import os
from datetime import datetime
from dateutil import parser
import re

#MatPlotLib
import matplotlib as mpl
import matplotlib.pyplot as plt

#Scikit-image
from skimage.transform import rotate
from skimage.util import crop
from skimage import exposure
from skimage import data, img_as_float, img_as_uint

#magic commands
%matplotlib inline
%matplotlib notebook

#tweak styles
mpl.rc('figure', figsize = (10,6))
mpl.rc('image', cmap = 'gray')

#get some DeprecationWarnings in sklearn module. Ignore these warnings for
 now.
import warnings
warnings.filterwarnings('ignore')
```

SUPPORT FUNCTIONS

In [2]:
```
def adapthist(im):
    """
    Enhance contrast by adaptive histogram equalization
    """

    im_adapteq = exposure.equalize_adapthist(im, clip_limit=0.01)

    return im_adapteq

def showim2(im1, im2):
    """
    Show two images side-by-side
    """
    #Show images
```

```
    f, ax = plt.subplots(1, 2, sharey=True)
    plt.axes(ax[0])
    plt.imshow(im1)
    plt.axes(ax[1])
    plt.imshow(im2)

def showparticle(trajectory, particle, framelist):
    """
    Show specific particle for frames specified in framelist
    """
    #select rows for corresponding particle and sort by frame
    select_traj = trajectory[trajectory['particle']==4].sort_values(['fram
e'], ascending=True)

    #loop by frames
    fig = -1
    for f in framelist:
        data = select_traj[select_traj['frame'] == f]
        fig = plt.figure()
        tp.annotate(data, image, plot_style={'markersize':5, 'markeredgewi
dth':1})
        fig.suptitle("Frame:" + str(f))


def timestamp(frameinterval):
    """
    return complete time stamp from MetaMorph tiff file
    """

    frameseq = np.arange(frameinterval[0],frameinterval[1]+1,1)
    #new pandas data frame to store time information
    df_ = pd.DataFrame(columns=['year','month','day','hour','minute','seco
nd','millisecond'])

    for f in frameseq:
        #extract meta data
        metadata = str(frames[f].metadata['ImageDescription'])

        #find the correct time stamp string between two consistent fields
in metadata
        sub1 = '<prop id="acquisition-time-local" type="time"'
        sub2 = '<prop id="modification-time-local" type="time"'
        ix1 = metadata.find(sub1)
        ix2 = metadata.find(sub2)
        metadata2 = metadata[ix1:ix2]

        #remove parenthesis from string
        metadata3 = re.sub('\"','', metadata2)
        #search for the exact time string
        match = re.search(r'\d{8} \d{2}:\d{2}:\d{2}.\w+', metadata3)
        #now, strip the time and date values from the time string
        TO = datetime.strptime(match.group(), '%Y%m%d %H:%M:%S.%f')
        #add data to data frame
        df_.loc[f] = [TO.year,TO.month,TO.day,TO.hour,TO.minute,TO.second,
TO.microsecond/1000]

    return(df_)
```

```
def partparms(trajectory):
    """
    IN: trajectories OUT: particle_data = ['particle','r0_um','r1_um','dis
t_um','dt_s','vel_umps']
    """
    #create empty output data frame
    dfcols = ['particle','r0_um','r1_um','dist_um','dt_s','vel_umps']
    particle_data = pd.DataFrame(columns=dfcols)
    #store particle numbers
    partnumbers = np.sort(trajectory['particle'].unique())

    for p in partnumbers:

        #filter single particle trajectory and sort by frame number
        trajectory_p = pd.DataFrame(trajectory[trajectory['particle'] == p
].sort_values(['frame'], ascending=True))
        #create empty data frame for trajectory p
        df_ = pd.DataFrame(columns=dfcols)

        for line in range(1,len(trajectory_p)):
            dframe = trajectory_p.iloc[line,-2] - trajectory_p.iloc[line-1
,-2]
            dt = dframe * scaninterval / 1000 #[s]
            r0 = np.array([trajectory_p.iloc[line-1,0] * pixel_calibration
_x, trajectory_p.iloc[line-1,1] * pixel_calibration_y])
            r1 = np.array([trajectory_p.iloc[line,0] * pixel_calibration_x
, trajectory_p.iloc[line,1] * pixel_calibration_y])
            dist = np.linalg.norm(r1-r0) #[um]
            vel = dist/dt #[um/s]
            #record entries only if dframe = 1. A jump in frame number ind
icates that the particle was lost
            if dframe == 1:
                df_.loc[len(df_)] = [p,r0,r1,dist,dt,vel]

        #add the data to to output array
        particle_data = particle_data.append(df_, ignore_index = True)

    return(particle_data)

def stat_partparms(trajectory, particle_data):
    """
    OUT: mean_particle_data = ['particle','r0_um','r1_um','total_dist_um',
'diag_um','dt_s','vel_umps']
    """
    #prepare output data frame
    dfcols = ['particle','total_dist_um','diag_um','total_time_s','mean_ve
l_umps']
    stat_particle_data = pd.DataFrame(columns=dfcols)

    #loop over all particles
    partnumbers = np.sort(particle_data['particle'].unique())
    for p in partnumbers:

        #filter measurements and trajectory for particle p
        particle_data_p = particle_data[particle_data['particle'] == p]
        trajectory_p = trajectory[trajectory['particle'] == p]
```

```
        #create empty data frame for trajectory p
        total_dist_um = particle_data_p['dist_um'].sum()
        total_time_s = particle_data_p['dt_s'].sum()
        mean_vel_umps = particle_data_p['vel_umps'].mean()
        #calculate the (approximate) trajectory diagonal size for particle
 p
        diag_um = tp.diagonal_size(trajectory_p) * np.mean([pixel_calibrat
ion_x, pixel_calibration_y])
        #add results to the output data frame
        stat_particle_data.loc[len(stat_particle_data)] = [p, total_dist_u
m, diag_um, total_time_s, mean_vel_umps]

    return(stat_particle_data)

def filter_particle_data(particle_data, window):
    """
    OUT: sliding average filter = ['particle','r0_um','r1_um','dist_um','d
t_s','vel_umps']
    """

    #this filtering can be done already in the partparms function!!!
    #create empty output data frame
    dfcols = ['particle','r0_um','r2_um','dist_um','dt_s','vel_umps','wind
ow']
    filter_particle_data = pd.DataFrame(columns=dfcols)
    #store particle numbers
    partnumbers = np.sort(particle_data['particle'].unique())

    for p in partnumbers:

        #filter single particle trajectory and sort by frame number
        particle_data_p = particle_data[particle_data['particle'] == p]

        #loop for averaging
        line = np.ceil((window-1)/2)
        step = np.floor(window/2)

        while (line + step +1 < particle_data_p.shape[0]):
            dist = particle_data_p['dist_um'].iloc[line-step:line+step+1].
sum()
            dt = particle_data_p['dt_s'].iloc[line-step:line+step+1].sum()
            vel = dist / dt
            r0 = particle_data_p['r0_um'].iloc[line-step]
            r2 = particle_data_p['r1_um'].iloc[line+step+1]
            #add results to the output data frame
            filter_particle_data.loc[len(filter_particle_data)] = [p,r0,r2
,dist,dt,vel,window]
            line += 1

    return(filter_particle_data)
```

TRAJECTORY FILTER FUNCTIONS

In [3]:
```
def filtered_trajectory_dist(trajectory, stat_particle_data, dist):
    #filter trajectories by total distance within [min,max] interval
    """
```

```
    Filter trajectories by total_dist_um [min,max]. use max = 0 for ignori
ng upper limit
    """
    #find those particles that have a trajectory within the dist[min,max]
interval
    if dist[1] == 0:
        filtered_stat = stat_particle_data[(stat_particle_data['total_dist
_um'] >= dist[0])]
    else:
        filtered_stat = stat_particle_data[(stat_particle_data['total_dist
_um'] >= dist[0]) &
                                           (stat_particle_data['total_dist
_um'] <= dist[1])]

    #figure out the particle numbers
    partnumbers = np.sort(filtered_stat['particle'].unique())

    #select those numbers in the trajectory frame
    #create empty data frame
    filtered_trajectory = pd.DataFrame(columns=trajectory.columns.values)

    for p in partnumbers:
        trajectory_p = trajectory[trajectory['particle'] == p]
        #add to output data frame
        filtered_trajectory = filtered_trajectory.append(trajectory_p, ign
ore_index = True)

    return(filtered_trajectory)


def filtered_trajectory_diag(trajectory, stat_particle_data, diag):
    #filter trajectories by total distance within [min,max] interval
    """
    Filter by length of diagonal of rectangular box containing the traj [m
in,max]. max = 0 for no upper limit
    """
    #find those particles that have a trajectory within the dist[min,max]
interval
    if diag[1] == 0:
        filtered_stat = stat_particle_data[(stat_particle_data['diag_um']
>= diag[0])]
    else:
        filtered_stat = stat_particle_data[(stat_particle_data['diag_um']
>= diag[0]) &
                                           (stat_particle_data['diag_um']
<= diag[1])]

    #figure out the particle numbers
    partnumbers = np.sort(filtered_stat['particle'].unique())

    #select those numbers in the trajectory frame
    #create empty data frame
    filtered_trajectory = pd.DataFrame(columns=trajectory.columns.values)

    for p in partnumbers:
        trajectory_p = trajectory[trajectory['particle'] == p]
        #add to output data frame
```

```
        filtered_trajectory = filtered_trajectory.append(trajectory_p, ign
ore_index = True)

    return(filtered_trajectory)
```

Data file and path information

In [4]:
```
filedir = os.getcwd()
resultsdir = os.getcwd()
filename = 'H160122-X20-TAILMID-FL80FPS.tif'
file = os.path.join(filedir, filename)
#frames = pims.TiffStack(file)
frames = pims.TiffStack(file, process_func=adapthist)
#metadata
metadata = frames[0].metadata['ImageDescription']
```

In [5]:
```
print(metadata)
```


```
<MetaData>
<prop id="Description" type="string" value="Acquired from AndorSdk3 Camera
&#13;&#10;Exposure: 5 msec&#13;&#10;Binning: 1 X 1&#13;&#10;Region: 256 x
216, offset at (0, 0)&#13;&#10;Digitizer: 200 MHz - lowest noise&#13;&#10;
Gain: 11-bit (high well capacity)&#13;&#10;Cooler On: 1&#13;&#10;Camera St
ate: Non-Overlapped&#13;&#10;"/>
<prop id="MetaDataVersion" type="float" value="1"/>
<prop id="ApplicationName" type="string" value="MetaMorph"/>
<prop id="ApplicationVersion" type="string" value="7.8.0.0"/>
<PlaneInfo>
<prop id="plane-type" type="string" value="plane"/>
<prop id="pixel-size-x" type="int" value="256"/>
<prop id="pixel-size-y" type="int" value="216"/>
<prop id="bits-per-pixel" type="int" value="16"/>
<prop id="autoscale-state" type="bool" value="off"/>
<prop id="autoscale-min-percent" type="float" value="0"/>
<prop id="autoscale-max-percent" type="float" value="0"/>
<prop id="scale-min" type="int" value="100"/>
<prop id="scale-max" type="int" value="315"/>
<prop id="spatial-calibration-state" type="bool" value="on"/>
<prop id="spatial-calibration-x" type="float" value="0.323624"/>
<prop id="spatial-calibration-y" type="float" value="0.323624"/>
<prop id="spatial-calibration-units" type="string" value="um"/>
<prop id="image-name" type="string" value="H160122-X20-TAILEND-FL80FPS"/>
<prop id="threshold-state" type="string" value="ThresholdOff"/>
<prop id="threshold-low" type="int" value="0"/>
<prop id="threshold-high" type="int" value="65535"/>
<prop id="threshold-color" type="colorref" value="4080ff"/>
<prop id="zoom-percent" type="int" value="209"/>
<prop id="gamma" type="float" value="1"/>
<prop id="look-up-table-type" type="string" value="by-wavelength"/>
<prop id="look-up-table-name" type="string" value="Set By Wavelength"/>
<prop id="photonegative-mode" type="bool" value="off"/>
<prop id="gray-calibration-curve-fit-algorithm" type="int" value="4"/>
<prop id="gray-calibration-values" type="float-array" value=""/>
<prop id="gray-calibration-min" type="float" value="-1"/>
<prop id="gray-calibration-max" type="float" value="-1"/>
<prop id="gray-calibration-units" type="string" value=""/>
```

```
<prop id="plane-guid" type="guid" value="{FB447B9A-771D-40F5-B5E9-B5B4FEBD
38C4}"/>
<prop id="acquisition-time-local" type="time" value="20160122 14:22:38.646
"/>
<prop id="modification-time-local" type="time" value="20160129 14:46:02.23
7"/>
<prop id="stage-position-x" type="float" value="17599"/>
<prop id="stage-position-y" type="float" value="-1918.2"/>
<prop id="z-position" type="float" value="0"/>
<prop id="wavelength" type="float" value="0"/>
<prop id="camera-binning-x" type="int" value="1"/>
<prop id="camera-binning-y" type="int" value="1"/>
<prop id="camera-chip-offset-x" type="float" value="0"/>
<prop id="camera-chip-offset-y" type="float" value="0"/>
<prop id="_IllumSetting_" type="string" value="GFP ON EXT"/>
<prop id="_MagSetting_" type="string" value="20x"/>
<custom-prop id="Emulation for Shutter button" type="string" value="Closed
"/>
<custom-prop id="Lumencor Blue Intensity" type="float" value="255"/>
<custom-prop id="Lumencor Blue Shutter" type="string" value="Closed"/>
<custom-prop id="Lumencor Cyan Intensity" type="float" value="0"/>
<custom-prop id="Lumencor Cyan Shutter" type="string" value="Closed"/>
<custom-prop id="Lumencor Green Intensity" type="float" value="255"/>
<custom-prop id="Lumencor Green Shutter" type="string" value="Closed"/>
<custom-prop id="Lumencor Red Intensity" type="float" value="255"/>
<custom-prop id="Lumencor Red Shutter" type="string" value="Closed"/>
<custom-prop id="Lumencor Teal Intensity" type="float" value="255"/>
<custom-prop id="Lumencor Teal Shutter" type="string" value="Closed"/>
<custom-prop id="Lumencor UV Intensity" type="float" value="255"/>
<custom-prop id="Lumencor UV Shutter" type="string" value="Closed"/>
<custom-prop id="Ti Filter Block 1" type="string" value="-----"/>
<custom-prop id="Ti Filter Block 2" type="string" value="GFP"/>
<custom-prop id="Ti Objective" type="string" value="Plan Apo 20x / 0.75"/>
<custom-prop id="Ti Optical Path" type="string" value="Left Port"/>
</PlaneInfo>
<SetInfo>
<prop id="number-of-planes" type="int" value="500"/>
</SetInfo>
</MetaData>
```


Calibrations

In [6]:

```
#Microscope calibrations
print('Pixel calibration for Andor Neo sCMOS 2560 x 2160 camera')
print('Image file: ', filename)
objective_magnification = 20
sideport_magnification = 0.38
chip_size_x = 16.6 #full frame chip size X [mm]
chip_size_y = 14.0 #full frame chip size Y [mm]
chip_pixels_x = 2560 #full frame chip size X [pixels]
chip_pixels_y = 2160 #full frame chip size Y [pixels]
field_ratio_x = frames[0].shape[1] / chip_pixels_x
field_ratio_y = frames[0].shape[0] / chip_pixels_y
field_size_x = chip_size_x * field_ratio_x / objective_magnification / sid
eport_magnification
```

```
field_size_y = chip_size_y * field_ratio_y / objective_magnification / sid
eport_magnification
pixel_calibration_x = chip_size_x * 1000 / chip_pixels_x / objective_magni
fication / sideport_magnification
pixel_calibration_y = chip_size_y * 1000 / chip_pixels_y / objective_magni
fication / sideport_magnification
pixel_calibration = np.mean([pixel_calibration_x, pixel_calibration_y]) #m
ean value to convert pixels into microns
print('pixel_calibration_x = ', '%.3f' % pixel_calibration_x, '[um/pixel]'
)
print('pixel_calibration_y = ', '%.3f' % pixel_calibration_y, '[um/pixel]'
)
print('field_size_x = ', '%.2f' % field_size_x, '[mm]')
print('field_size_y = ', '%.2f' % field_size_y, '[mm]')
#date and frame rate
stamp = timestamp([0,10])
frame_year = int(stamp['year'][0])
frame_month = int(stamp['month'][0])
frame_day = int(stamp['day'][0])
time =stamp['minute'] * 60 * 1000 + stamp['second'] * 1000 + stamp['millis
econd']
timearray = np.array(time)
dtlist = [(timearray[i]-timearray[i-1]) for i in np.arange(1,len(timearray
),1)]
scaninterval = np.mean(dtlist)
scanrate = 1/scaninterval * 1000
print('mean scan interval = ', scaninterval, '[ms]')
print('mean frame rate = ', scanrate, '[fps]')
```

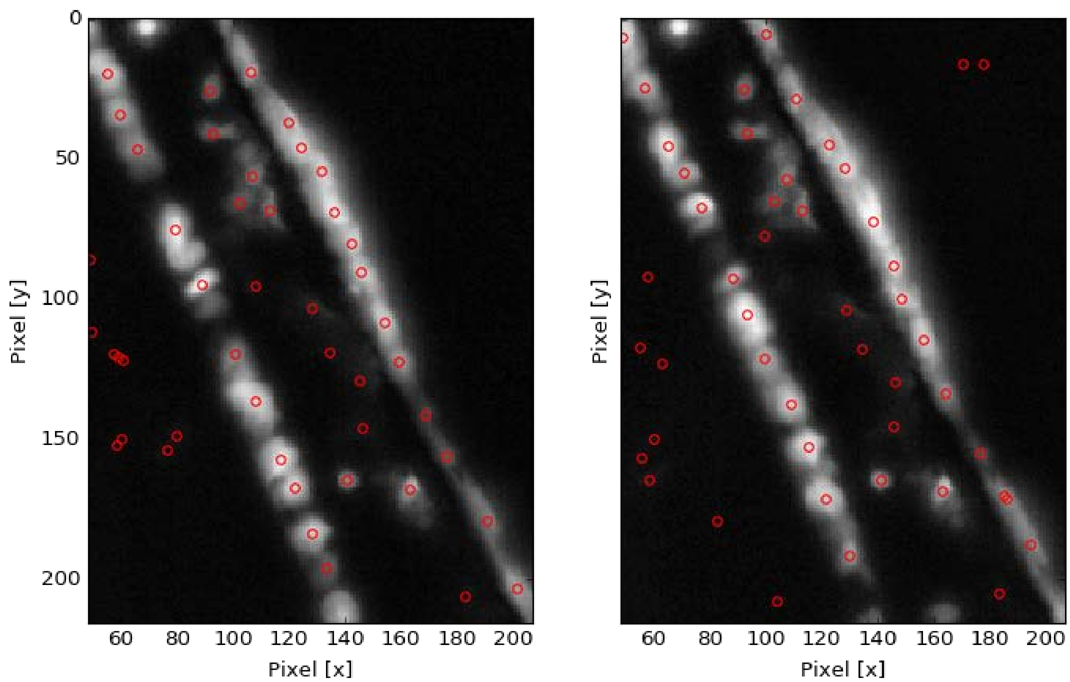
```
Pixel calibration for Andor Neo sCMOS 2560 x 2160 camera
Image file:  H160122-X20-TAILMID-FL80FPS.tif
pixel_calibration_x =  0.853 [um/pixel]
pixel_calibration_y =  0.853 [um/pixel]
field_size_x =  0.22 [mm]
field_size_y =  0.18 [mm]
mean scan interval =  8.3 [ms]
mean frame rate =  120.481927711 [fps]
```

LOCATE FEATURES

In [7]:

```
#Locate features
fnumber = 50 #current frame to look at
diam = 11 #odd integer with features extent in pixels
minm = 0.03 #minimum integrated brightness
sep = 1 #minimum separation between features
perc = 20 #features must have a peak brighter than pixels in this percenti
le
topnum = 500 #return only the N brightest features above minmass

feats1 = tp.locate(frames[fnumber], invert=False, diameter=diam, minmass=m
inm, separation=sep, percentile=perc,
                  topn=topnum)
feats2 = tp.locate(frames[fnumber+10], invert=False, diameter=diam, minmas
s=minm, separation=sep, percentile=perc,
                  topn=topnum)
```

```
f, ax = plt.subplots(1, 2, sharey=True)
plt.axes(ax[0])
tp.annotate(feats1, frames[fnumber], plot_style={'markersize':5, 'markered
gewidth':1})

plt.axes(ax[1])
tp.annotate(feats2, frames[fnumber+10], plot_style={'markersize':5, 'marke
redgewidth':1})

#set axis properties
[ax[i].set_xlabel('Pixel [x]') for i in range(2)]
[ax[i].set_ylabel('Pixel [y]') for i in range(2)]
[ax[i].legend_.remove() for i in range(2)]
```



Out[7]: [None, None]

FIND FEATURES IN ALL FRAMES

In [8]:
```
feats = tp.batch(frames, diameter=diam, minmass=minm, separation=sep, inve
rt=False, percentile=perc,
                 topn=topnum)
print('Total number of features: ',feats.shape)
```

```
Frame 499: 79 features
Total number of features:  (32740, 9)
```

LINK FEATURES INTO PARTICLE TRAJECTORIES

In [9]:
```
searchrange = 5 #Maximum distance features can move between frames
memory = 3 #Maximum number of frames during which a feature can vanish, th
an reappear and consider the same particle
trajs = tp.link_df(feats, 5, memory=3)
```

```
Frame 499: 79 trajectories present
```

FILTER TRAJECTORIES

In [10]:
```
#get rid of spurious trajectories (minimum number of frames)
trajsF1 = tp.filter_stubs(trajs, threshold = 5)
# Compare the number of particles in the unfiltered and filtered data.
print('Before:', trajs['particle'].nunique())
print('After:', trajsF1['particle'].nunique())
```

```
Before: 6293
After: 1445
```

In [11]:
```
#calculate difference vectors (r0,r1), distance travelled, time passed and
 velocity for each frame and all particles
particle_data = partparms(trajsF1)
```

In [12]:
```
particle_data = filter_particle_data(particle_data, window = 5)
```

In [13]:
```
particle_data.head()
```

Out[13]:

	particle	r0_um	r2_um	dist_um	dt_s	vel_umps	window
0	1	[87.744889199, 9.35108167178]	[90.8228623841, 15.4143409753]	5.728388	0.0415	138.033436	5
1	1	[88.3202999771, 10.3292140127]	[91.1492230196, 16.2020732551]	5.693591	0.0415	137.194959	5
2	1	[88.5993442413, 11.1436292752]	[91.3290846138, 16.3892093361]	5.685360	0.0415	136.996619	5
3	1	[89.2136827137, 12.1165659467]	[92.005710967, 17.9164843115]	4.794258	0.0415	115.524279	5
4	1	[89.4893652769, 12.9534382354]	[92.5325093337, 18.7968146789]	5.583594	0.0415	134.544423	5

In [14]:
```
#calculate particle statistics (total_dist_um, total_time_s, mean_vel_umps
)
stat_particle_data = stat_partparms(trajsF1, particle_data)
q3_diag = stat_particle_data['diag_um'].quantile(q=0.75)
q1_diag = stat_particle_data['diag_um'].quantile(q=0.25)
median_diag = stat_particle_data['diag_um'].quantile(q=0.50)
```

In [15]:
```
#filter the trajectories by diag_um interval
filtered_trajectory = filtered_trajectory_diag(trajsF1, stat_particle_data
, [q1_diag, 0])
#filter the trajectories by diag_um interval
#filter trajectories by diagonal trajectory size
print('Original number of trajectories: ', len(trajsF1['particle'].unique(
)))
print('Filtered number of trajectories: ', len(filtered_trajectory['partic
le'].unique()))
```

```
Original number of trajectories:  1445
Filtered number of trajectories:  633
```

In [16]:
```
#calculate new statistics of filtered trajectories
filtered_particle_data = partparms(filtered_trajectory)
filtered_stat_particle_data = stat_partparms(filtered_trajectory, filtered
_particle_data)
```

In [17]:
```
#velocity statistics
q3_vel = filtered_stat_particle_data['mean_vel_umps'].quantile(q=0.75)
q1_vel = filtered_stat_particle_data['mean_vel_umps'].quantile(q=0.25)
q1_diag = filtered_stat_particle_data['diag_um'].quantile(q=0.25)
q3_diag = filtered_stat_particle_data['diag_um'].quantile(q=0.75)
median_vel = filtered_stat_particle_data['mean_vel_umps'].quantile(q=0.50)
max_vel = filtered_stat_particle_data['mean_vel_umps'].max()
mean_vel = filtered_stat_particle_data['mean_vel_umps'].mean()
print('Mean velocity:', mean_vel, ' um/s')
print('Maximum velocity', max_vel, ' um/s')
print('Median velocity: ', median_vel, ' um/s')
print('[q1,q3] quantiles: [', q1_vel,',', q3_vel,'] um/s')
print('[q1,q3] quantiles: [', q1_diag,',', q3_diag,'] um')
```

```
Mean velocity: 181.991984636  um/s
Maximum velocity 414.055713521  um/s
Median velocity:  169.279015259  um/s
[q1,q3] quantiles: [ 108.139896182 , 257.404875404 ] um/s
[q1,q3] quantiles: [ 8.80129576249 , 27.606805088 ] um
```

In [18]:
```
filtered_stat_particle_data.head()
```

Out[18]:

	particle	total_dist_um	diag_um	total_time_s	mean_vel_umps
0	1	13.622958	13.544328	0.1162	117.237161
1	2	17.981607	17.852088	0.0747	240.717626
2	5	26.484783	26.307487	0.1328	199.433605
3	10	48.357723	43.282175	0.2075	233.049269
4	20	31.339962	31.289918	0.1079	290.453768

filtered_trajectory

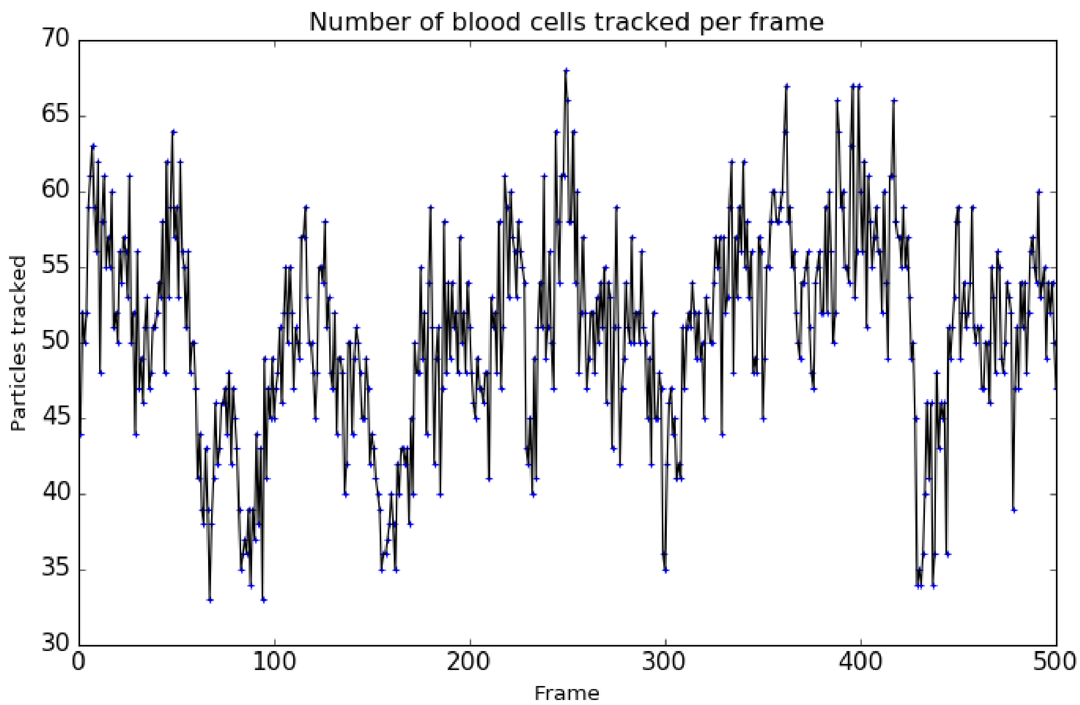
filtered_particle_data filtered_stat_particle_data

In [ ]:
```
stat_particle_data.head()
```

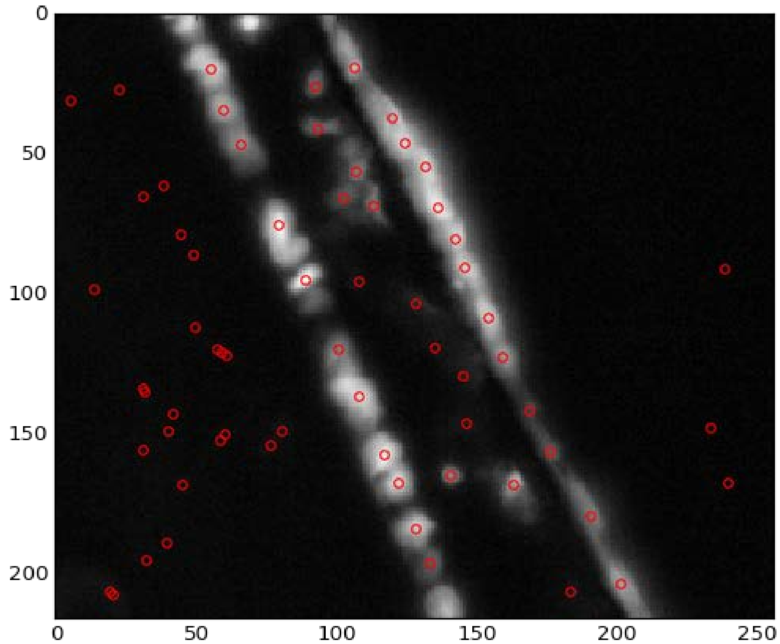
STATISTICS GRAPHS

In [19]:
```
#Number of tracked blood cells per frame
fnumbers = trajsF1['frame'].unique()
PartPerFrame = [trajsF1[trajsF1['frame']==fnumbers[i]].shape[0] for i in f
numbers]
fx = np.arange(1,len(PartPerFrame)+1,1)
plt.figure()
```

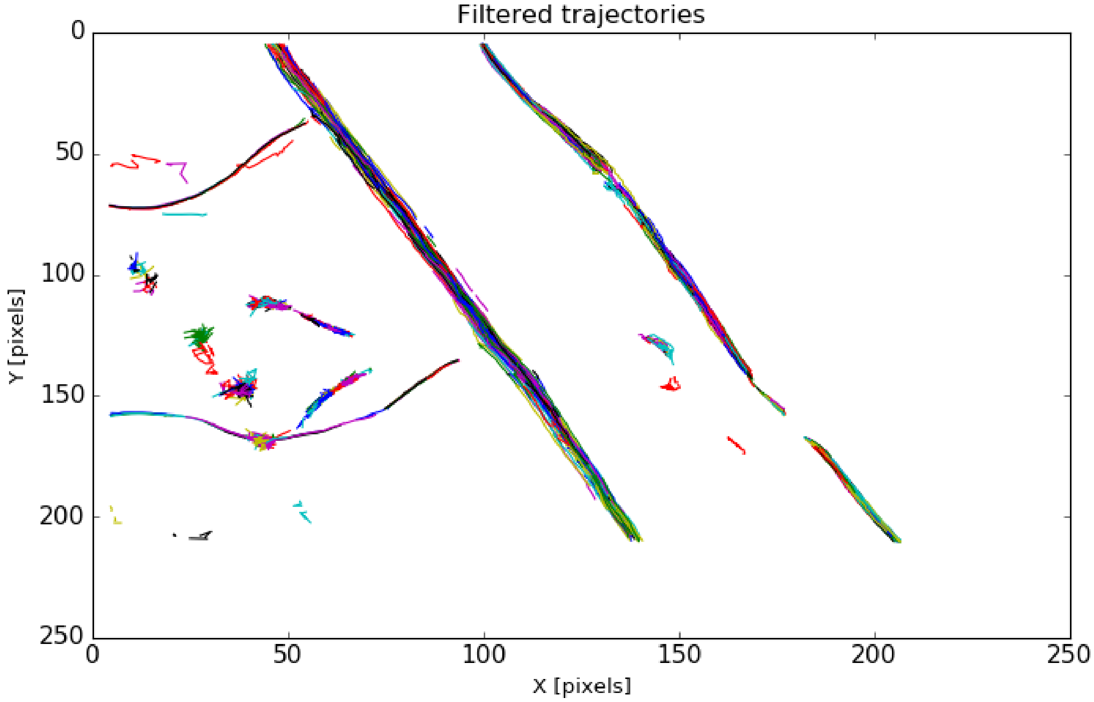
```
ax0=plt.axes()
ax0.scatter(x=fx, y=PartPerFrame, s=20, marker='+', linewidths=1)
PartLine = mpl.lines.Line2D(xdata=fx,ydata=PartPerFrame, linewidth=1, colo
r='black')
ax0.add_line(PartLine)
ax0.axis(xmin=0, xmax=len(frames))
ax0.set_xlabel('Frame')
ax0.set_ylabel('Particles tracked')
ax0.set_title('Number of blood cells tracked per frame')
for tick in ax0.xaxis.get_major_ticks():
        tick.label.set_fontsize(14)
for tick in ax0.yaxis.get_major_ticks():
        tick.label.set_fontsize(14)
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'ClsPerFrame.png'
plt.savefig(f, pad_inches = 0)
```



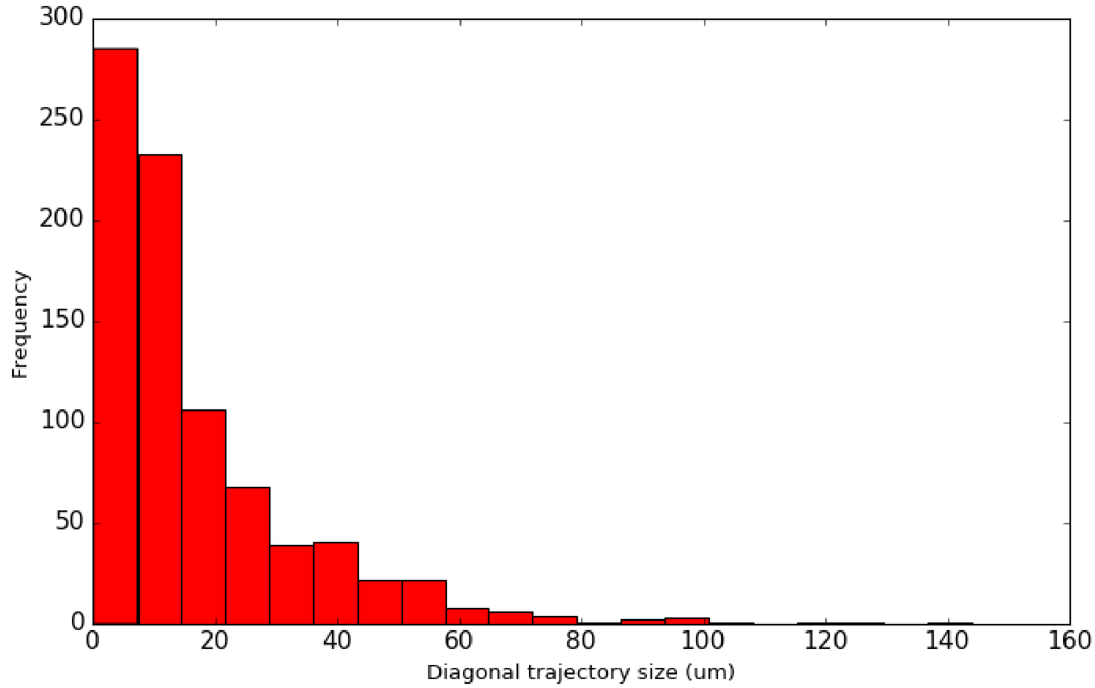
In [20]:
```
#Show image with marked blood cells
plt.figure()
ax0 = plt.axes()
tp.annotate(feats1, frames[fnumber], plot_style={'markersize':5, 'markered
gewidth':1})
ax0.legend_.remove()
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'FrameWithCircles.png'
plt.savefig(f, pad_inches = 0)
```



In [21]:
```
#plot trajectories
#show only the long trajectories
plt.figure()
ax0=plt.axes()
plt.axes(ax0)
tp.plot_traj(filtered_trajectory)
#tp.plot_traj(filtered_trajectory)
ax0.legend_.remove()
#ax0.legend(bbox_to_anchor=(1.1, 1.05))
plt.gca().invert_yaxis()
ax0.set_xlabel('X [pixels]')
ax0.set_ylabel('Y [pixels]')
ax0.invert_yaxis()
ax0.set_title('Filtered trajectories')
for tick in ax0.xaxis.get_major_ticks():
        tick.label.set_fontsize(14)
for tick in ax0.yaxis.get_major_ticks():
        tick.label.set_fontsize(14)
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'Trajects_Filtered.png'
plt.savefig(f, pad_inches = 0)
```



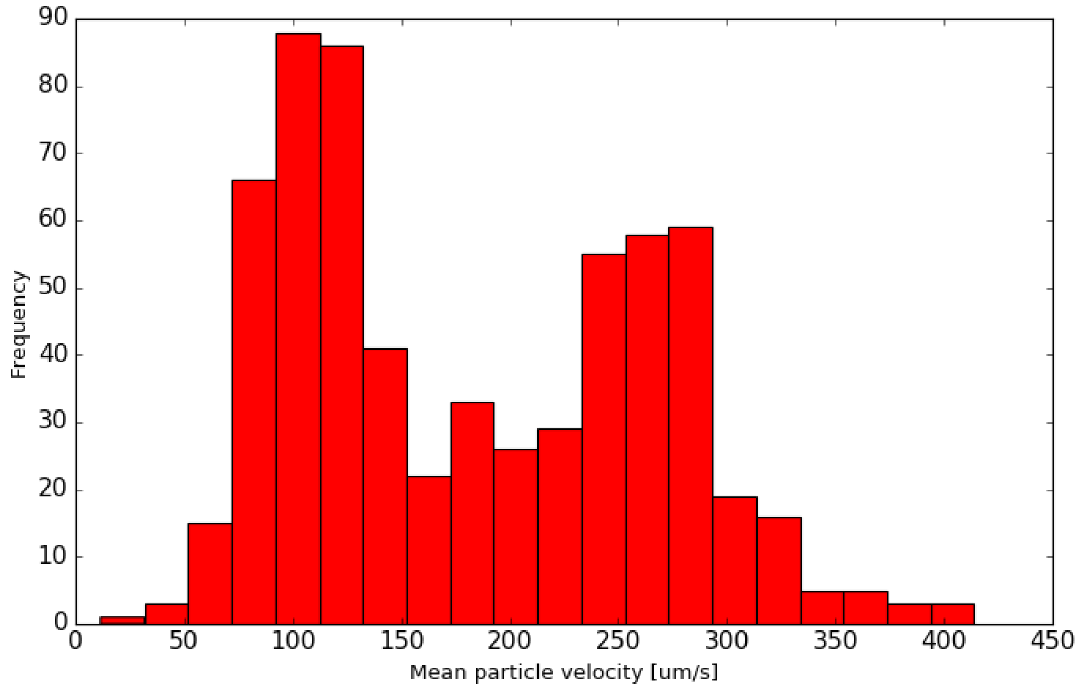
In [22]:
```
#hist: Diagonal trajectory size
plt.figure()
ax1=plt.axes()
ax1.hist(stat_particle_data['diag_um'], histtype='bar', color='red', bins=
20)
#plot((median, median), (0, 25), 'k-')
ax1.set_xlabel('Diagonal trajectory size (um)')
ax1.set_ylabel('Frequency')

for tick in ax1.xaxis.get_major_ticks():
        tick.label.set_fontsize(14)
for tick in ax1.yaxis.get_major_ticks():
        tick.label.set_fontsize(14)
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'Traj_Diag_Size.png'
plt.savefig(f, pad_inches = 0)
```



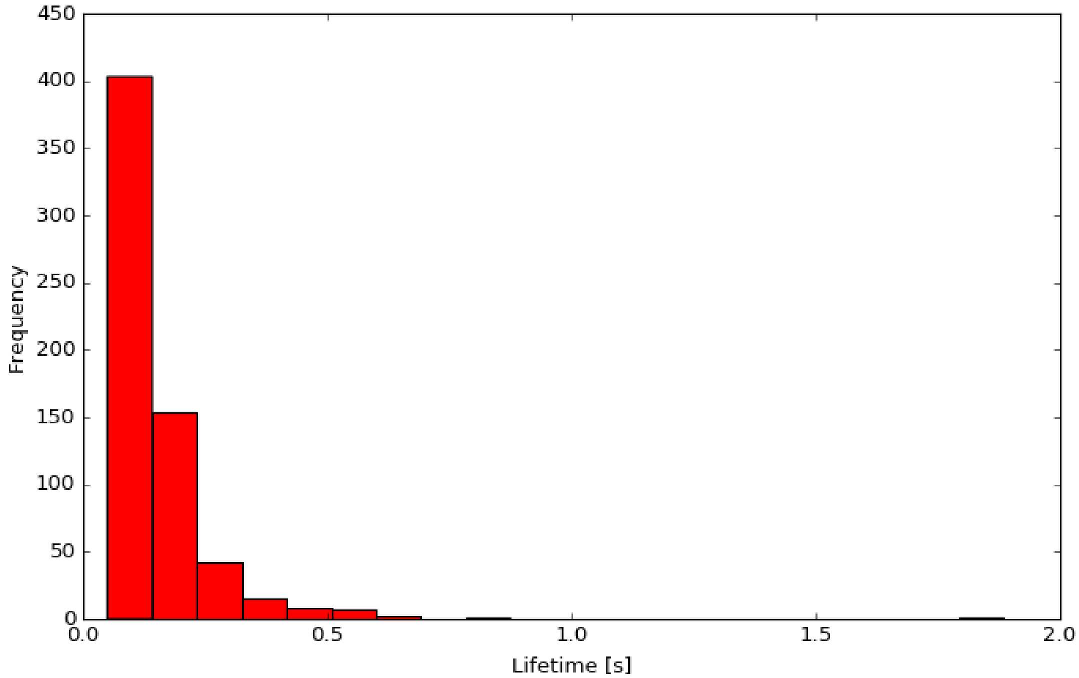
In [ ]:

In [23]:

```
#hist: Velocity profile
plt.figure()
ax1=plt.axes()
ax1.hist(filtered_stat_particle_data['mean_vel_umps'], histtype='bar', col
or='red', bins=20)
ax1.set_xlabel('Mean particle velocity [um/s]')
ax1.set_ylabel('Frequency')
for tick in ax1.xaxis.get_major_ticks():
        tick.label.set_fontsize(14)
for tick in ax1.yaxis.get_major_ticks():
        tick.label.set_fontsize(14)
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'Part_Velocity_Filtered.png'
plt.savefig(f, pad_inches = 0)
```



In [24]:
```
#hist: Total livetime
plt.figure()
ax1=plt.axes()
ax1.hist(filtered_stat_particle_data['total_time_s'], histtype='bar', colo
r='red', bins=20)
ax1.set_xlabel('Lifetime [s]')
ax1.set_ylabel('Frequency')
for tick in ax0.xaxis.get_major_ticks():
        tick.label.set_fontsize(14)
for tick in ax0.yaxis.get_major_ticks():
        tick.label.set_fontsize(14)
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'Lifetime.png'
plt.savefig(f, pad_inches = 0)
```



In [26]:

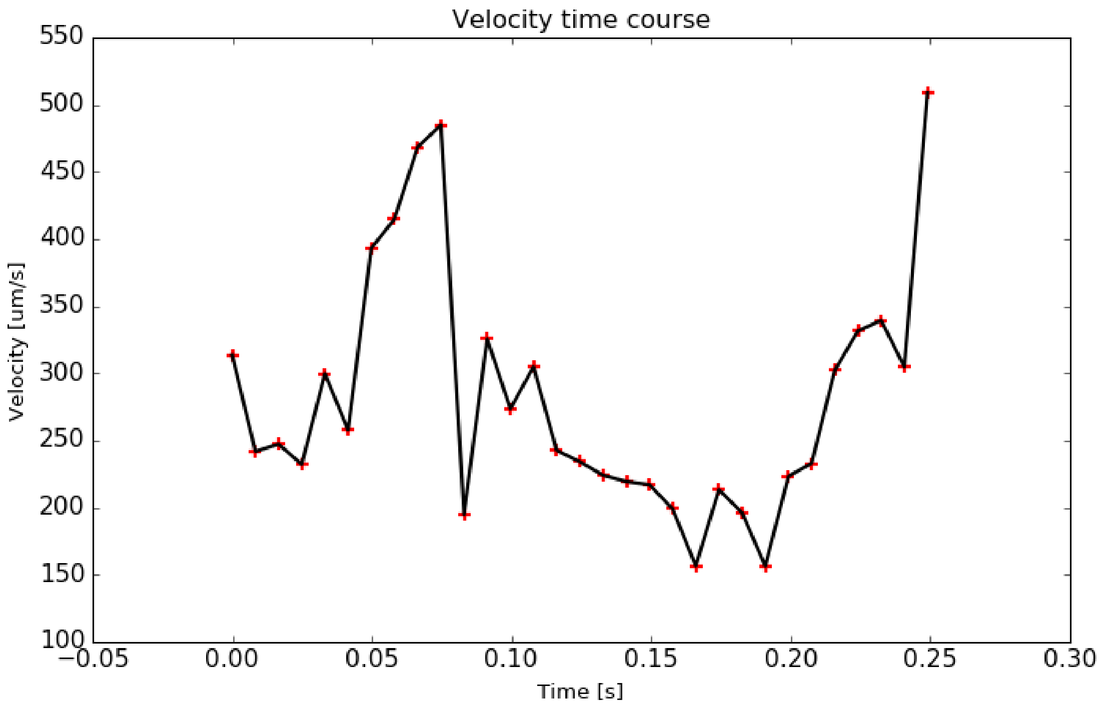
```
#plot the time course of the largest trajectories
#filtered_trajectory
#filtered_particle_data
#filtered_stat_particle_data
#figure out the particle numbers

q_diag = stat_particle_data['diag_um'].quantile(q=0.9)
filtered_trajectory_q = filtered_trajectory_diag(filtered_trajectory, filt
ered_stat_particle_data, [q_diag, 0])
#re-calculate statistics (to be avoided in the future)
filtered_particle_data_q = partparms(filtered_trajectory_q)
filtered_stat_particle_data_q = stat_partparms(filtered_trajectory_q, filt
ered_particle_data_q)
#filter the trajectories by diag_um interval
#filter trajectories by diagonal trajectory size
print('Original number of trajectories: ', len(trajsF1['particle'].unique(
)))
print('Filtered number of trajectories: ', len(filtered_trajectory_q['part
icle'].unique()))
```

```
Original number of trajectories:  1445
Filtered number of trajectories:  85
```

In [27]:

```
#prepare on particle trajectory
partnumbers = np.sort(filtered_stat_particle_data_q['particle'].unique())
p = 74
#for p in range(len(partnumbers)):

#New data frame
dfcols = ['particle','time_s','dist_um','vel_umps']
timeseries_p = pd.DataFrame(columns = dfcols)

#select particle data
```

```
q9_p = filtered_particle_data_q[filtered_particle_data_q['particle'] == pa
rtnumbers[p]]

#create data lists
t = list([0])
d = list([0])
v = list([q9_p['vel_umps'].iloc[0]])
part = list([partnumbers[p]])
for k in np.arange(1, q9_p.shape[0], 1):
    t2 = t[k-1] + q9_p['dt_s'].iloc[k]
    t.append(t2)
    d2 = d[k-1] + q9_p['dist_um'].iloc[k]
    d.append(d2)
    v2 = q9_p['vel_umps'].iloc[k]
    v.append(v2)
    part.append(partnumbers[p])

timeseries_p['particle'] = part
timeseries_p['time_s'] = t
timeseries_p['dist_um'] = d
timeseries_p['vel_umps'] = v

#plot time series
plt.figure()
ax0=plt.axes()
plt.axes(ax0)
#plot velocity versus time for particle p
t = timeseries_p['time_s']
v = timeseries_p['vel_umps']
ax0.scatter(x=t, y=v, s=60, marker='+', linewidths=2, color = 'red')
vline = mpl.lines.Line2D(xdata=t,ydata=v, linewidth=2, color = 'black')
ax0.add_line(vline)
#ax0.legend(bbox_to_anchor=(1.1, 1.05))
ax0.set_xlabel('Time [s]')
ax0.set_ylabel('Velocity [um/s]')
ax0.set_title('Velocity time course')
for tick in ax0.xaxis.get_major_ticks():
        tick.label.set_fontsize(14)
for tick in ax0.yaxis.get_major_ticks():
        tick.label.set_fontsize(14)
#save image
basename = filename[:-4] +'_'
f = resultsdir + '\\' + basename + 'Velcity_Time.png'
plt.savefig(f, pad_inches = 0)
```


Velocity time course
Velocity [um/s]
Time [s]
550
500
450
400
350
300
250
200
150
100
−0.05
0.00
0.05
0.10
0.15
0.20
0.25
0.30

In [ ]:
```
1/0.3 * 60
```

In [ ]:
```
```