03a_CAMSHIFT

November 13, 2019

1 Assigment 3: CAMSHIFT

1.1 Paper

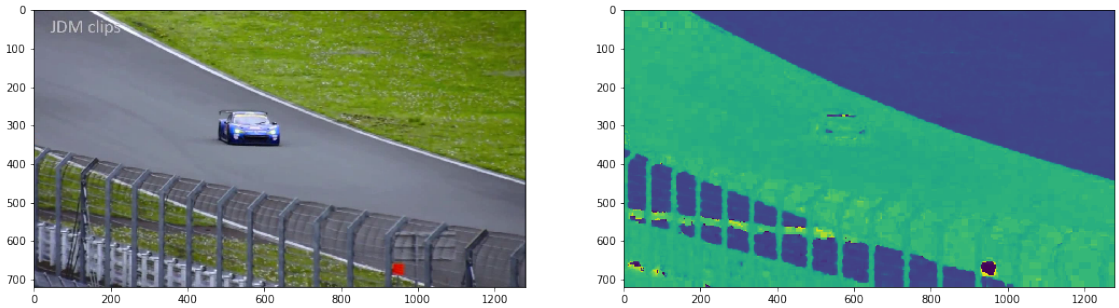
Read the paper "Bradski_etal_1998_camshift.pdf" in KVV (under "Resources/papers").

1.2 Calculate histogram

- Implement a function that creates a color histogram. Pass either an image and ROI, or the image underlying the ROI.
- For this purpose, a second (or third) parameter can be passed to specify the number of bins.
- Load the image "images/racecar.png" and convert the image to the HSV color space. Plot the Hue channel. (**RESULT**)

In [16]:
```
# dieser Code wurde als Musterlösung von Tobias Schülke zur Verfügung gestellt und vo

%matplotlib inline
from skimage import io, color
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.patches as patches
import os
import warnings; warnings.simplefilter('ignore')


IMAGES_PER_ROW = 4

MIN_SATURATION_CAR = 0.2
MIN_VALUE_CAR = 0.5
MIN_SATURATION_TACO = 0.8
MIN_VALUE_TACO = 0.2

ROI_FRAME_MARGIN_CAR = 60
ROI_FRAME_MARGIN_TACO = 20


def create_color_histogram(img, bins=360, normalize=False):
    hues = color.rgb2hsv(img)[:, :, 0]
```

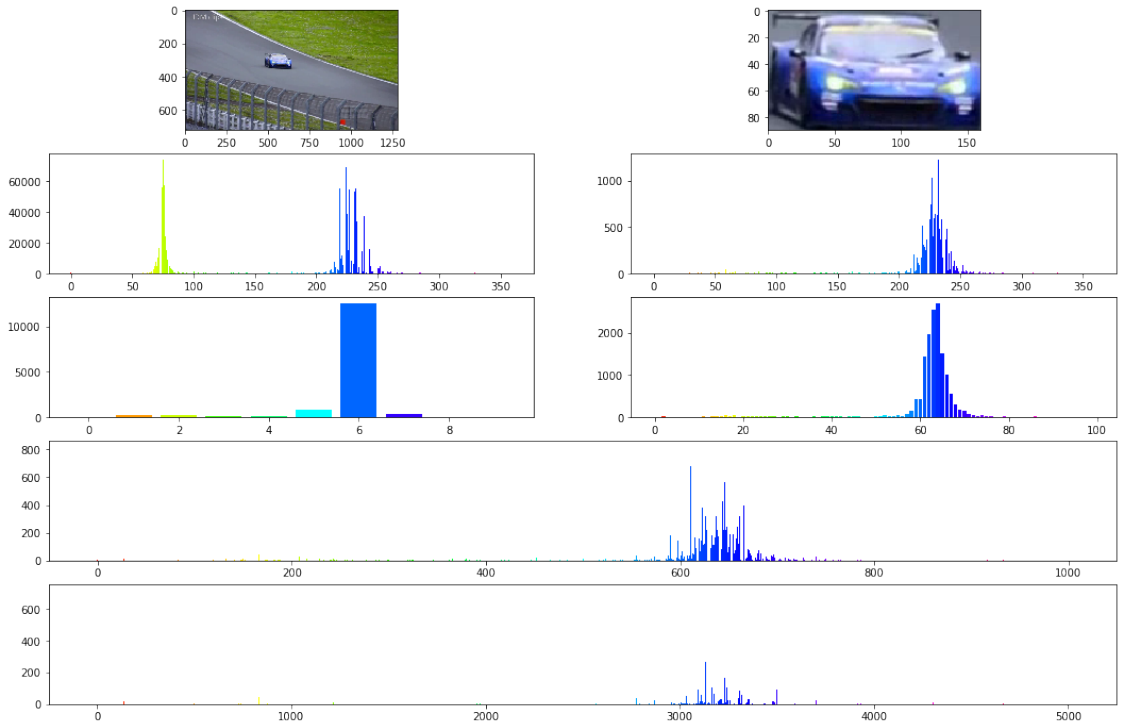
```
    hist, bin_edges = np.histogram(hues, bins=bins, range=(0, 1))
    if normalize:
        hist = hist / np.max(hist)
    return hist, bin_edges

def plot_histogram(ax, hist, bins=360):
    X = range(0, bins)
    hsv_colors = np.array([[h/bins, 1, 1] for h in X])
    rgb_colors = (
        color.hsv2rgb(
            # reshape to make 'hsv2rgb' work because it expects an image
            hsv_colors.reshape((bins, 1, 3))
        )
        .reshape((bins, 3))
    )
    return ax.bar(X, hist, color=rgb_colors)

def create_and_plot_histogram(ax, img, bins=360):
    hist, bin_edges = create_color_histogram(img, bins=bins)
    return hist, bin_edges, plot_histogram(ax, hist, bins=bins)


image = io.imread('images/racecar.png')

fig = plt.figure(figsize=(18, 12))
ax1 = plt.subplot(2, 2, 1)
ax2 = plt.subplot(2, 2, 2)

ax1.imshow(image)
ax2.imshow(color.rgb2hsv(image)[:, :, 0])

None
```



- display the histogram for the Hue channel for the entire image and for the RIO (x,y) = (480, 260) to (640, 350). Vary the number of bins for testing purposes (**RESULT**).

In [17]:
```
image_car = image[260:350, 480:640]

fig = plt.figure(figsize=(18, 12))
grid = fig.add_gridspec(5, 2)
ax1 = fig.add_subplot(grid[0, 0])
ax2 = fig.add_subplot(grid[0, 1])
ax3 = fig.add_subplot(grid[1, 0])
ax4 = fig.add_subplot(grid[1, 1])
ax5 = fig.add_subplot(grid[2, 0])
ax6 = fig.add_subplot(grid[2, 1])
ax7 = fig.add_subplot(grid[3, :])
ax8 = fig.add_subplot(grid[4, :])

ax1.imshow(image)
ax2.imshow(image_car)
create_and_plot_histogram(ax3, image)
create_and_plot_histogram(ax4, image_car)
create_and_plot_histogram(ax5, image_car, bins=10)
create_and_plot_histogram(ax6, image_car, bins=100)
create_and_plot_histogram(ax7, image_car, bins=1000)
create_and_plot_histogram(ax8, image_car, bins=5000)
```

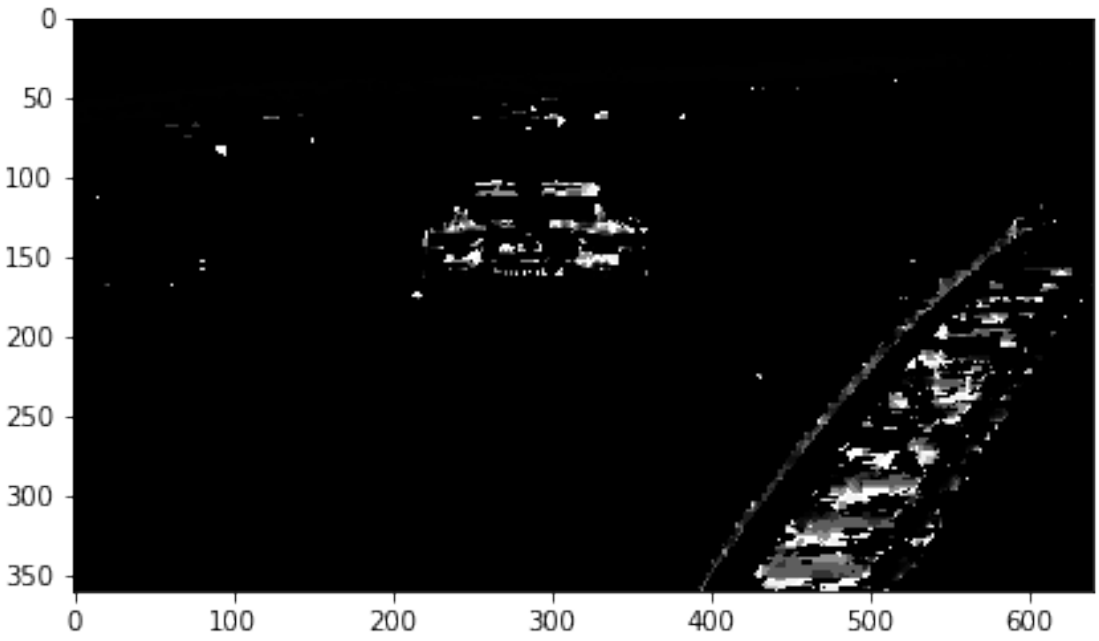
```
None
```



1.3 probability distribution

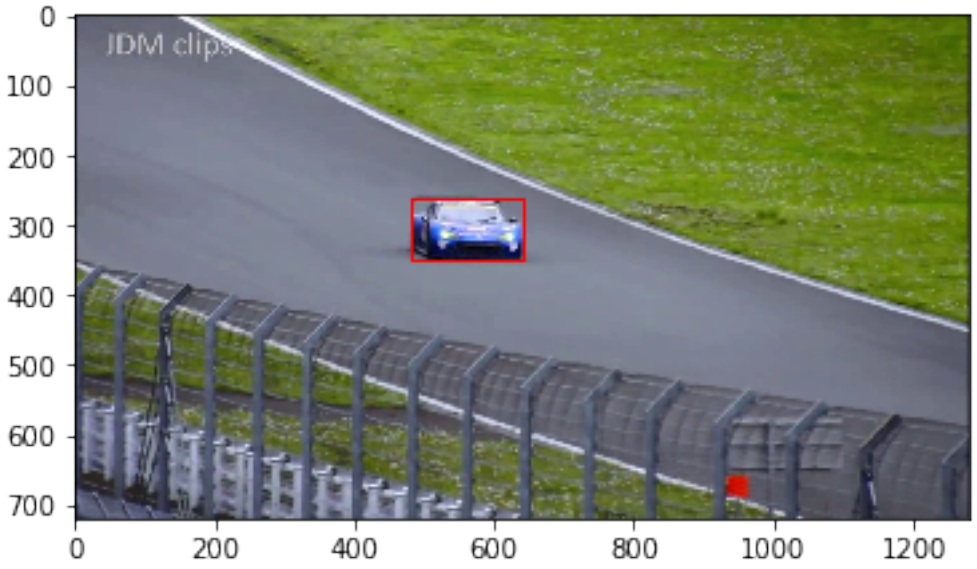
- implement the method outlined in the CAMSHIFT paper to create a probability distribution for a given object's hue histogram
- create the histogram of the car from the image "racecar.png" and apply the new function to the last frame of the video (images/racecar/151.jpg) (**RESULT**)

```
In [117]: # Tip: in der Nacht sind alle Katzen grau ;)
          def create_prob_distribution(img, hist, bin_edges, min_saturation, min_value):
              img_hsv = color.rgb2hsv(img)
              # We want grayscale.
              prob_dist = np.zeros((img.shape[0], img.shape[1]))
          #     print('hsv', img_hsv.shape)
              # Ignore irrelevant pixels (see page 3 in the paper).
              indices = np.logical_and(img_hsv[:, :, 1] >= min_saturation, img_hsv[:, :, 2] >=
          #     print('indices', indices.shape)
              hues = img_hsv[indices, 0]
          #     print('hues', hues.shape)
              binned_hues = np.digitize(hues, bin_edges)
          #     print('>', binned_hues.shape)
              max_index = len(bin_edges) - 1
              binned_hues[np.where(binned_hues == max_index)] -= 1
              prob_dist[indices] = hist[binned_hues]
              return prob_dist


          last_frame = io.imread('images/racecar/151.jpeg')
          histogram_car, bin_edges = create_color_histogram(image_car, bins=100, normalize=Tru
          prob_dist = create_prob_distribution(last_frame, histogram_car, bin_edges, MIN_SATURA
          io.imshow(prob_dist)

          None
```



1.3.1 show ROI in image

```
In [19]: def draw_roi(image, x, y, width, height, out=plt):
             ax = plt.gca()
             ax.add_patch(
                 patches.Rectangle(
                     (x, y),
                     width,
                     height,
                     fill=False,
                     edgecolor="red"
                 )
             )

             out.imshow(image)

         draw_roi(image, 480, 260, 160, 90)
```



1.4 Exercise 3.2 - Mean Shift

- Implement the Mean Shift method for a ROI as described in the lecture. Test the algorithm on the image sequences "images/racecar/*.jpg" or "images/taco/*.jpg". Select the appropriate tracking window (to reduce the file size I have reduced the images by a factor of 2, i.e. the ROI from above must be adjusted accordingly).
- Draw the trajectory of the objects as returned by Mean Shift. (**RESULT**)

```
In [217]: from math import ceil

          from numpy.linalg import norm
          from skimage.draw import circle


          def get_center(rect):
              return np.mean(rect.T, axis=1).round()


          def get_diag_len(rect):
              top_left, bottom_right = rect
              return np.sqrt(np.square(top_left[0] - bottom_right[0]) + np.square(top_left[1] -


          def get_roi(img, rect):
              top_left, bottom_right = rect
              return img[top_left[0]:bottom_right[0], top_left[1]:bottom_right[1]]
```

```
def get_shifted_next_window(window, moments, v, roi):
    return window + v.round().astype(np.int)


def mean_shift(initial_window, frames, create_prop_dist,
               *,
               frame_indices_to_draw=None,
               keyframes=None,
               ellipsis_indices_to_draw = None,
               shift_threshold=4,
               max_iterations=5,
               get_moments=get_moments,
               get_next_window=get_shifted_next_window,
               draw_ellipsis=None,):
    first_frame, frames = frames[0], frames[1:]

    if frame_indices_to_draw is None:
        frame_indices_to_draw = []
    if keyframes is None:
        keyframes = []
    if ellipsis_indices_to_draw is None:
        ellipsis_indices_to_draw = []

    window = initial_window
    centroids = [get_center(window)]

    NCOLS = 3
    fig, axes = plt.subplots(
        nrows=ceil((len(frame_indices_to_draw) + len(keyframes)) / NCOLS),
        ncols=NCOLS,
    )
    fig.set_figheight(18)
    fig.set_figwidth(18)
    plt.subplots_adjust(wspace = .01)
    fig.tight_layout()
    axes = np.asarray(axes).ravel()

    i = -1
    plot_index = 0
    for frame in frames:
        i += 1
        prev_centroid = centroids[-1]
        temp_centroids = [prev_centroid]

        num_iterations = -1
        while True:
            num_iterations += 1
```

```
            roi = get_roi(frame, window)
            hist, bin_edges = create_color_histogram(roi, bins=100, normalize=True)
            prop_dist = create_prop_dist(roi, hist, bin_edges)

            moments = get_moments(prop_dist)
            if moments[0] == 0:
                print('M_00 == 0, i =', i)
                break

            centroid = window[0] + get_centroid(moments)
            temp_centroids.append(centroid)

            v = centroid - prev_centroid
            prev_centroid = centroid
#             window += v.round().astype(np.int)
            window = get_next_window(window, moments, v, roi)
            if norm(v) <= shift_threshold or num_iterations > max_iterations:
                break

        centroids.append(centroid)
        if i in frame_indices_to_draw:
            axes[plot_index].imshow(draw_centroids(frame, temp_centroids))
            axes[plot_index].set_title(f'shifting mean {i}')
            plot_index += 1
        if i in keyframes:
            j = keyframes.index(i)
            start = keyframes[j - 1] if j > 0 else 0
            c = centroids[start:keyframes[j]]
            axes[plot_index].imshow(draw_centroids(frame, c))
            axes[plot_index].set_title(f'frames {start} - {keyframes[j]}')
            plot_index += 1
        if i in ellipsis_indices_to_draw and draw_ellipsis:
            axes[plot_index].imshow(draw_ellipsis(frame, centroid, moments))
            axes[plot_index].set_title(f'roll {i}')
            plot_index += 1

    return centroids


def get_moments(prop_dist):
    """Returns 0th and 1st image moments."""
    M_00 = np.sum(prop_dist)
    # Because we only have positive values (propabilities)
    # we know that all values must be zero, thus:
    # M_00 == x y I(x,y) == 0  =>  x y x*I(x,y) == M_10 == 0
    if M_00 > 0:
        num_rows, num_cols = prop_dist.shape
        x_indices = np.arange(0, num_cols)
```

```
            y_indices = np.arange(0, num_rows)
            M_10 = np.sum(prop_dist[:] * x_indices)
            M_01 = np.sum(prop_dist.T[:] * y_indices)
            M_11 = np.sum((prop_dist[:] * x_indices).T[:] * y_indices)
        else:
            M_10 = M_01 = M_11 = 0
        return (M_00, (M_10, M_01, M_11))


    def get_centroid(moments, row_first=True):
        M_00, (M_10, M_01, M_11) = moments[0], moments[1]
        centroid = np.array([M_01, M_10])
        if M_00 > 0:
            centroid = centroid / M_00
        if row_first:
            return centroid
        return np.flip(centroid)


    def draw_centroids(img, centroids):
        img = np.copy(img)
        for centroid in centroids:
            rr, cc = circle(*centroid, radius=5, shape=img.shape)
            img[rr, cc] = [255, 0, 0]
        return img


    frames = io.imread_collection('images/racecar/*')
    initial_window = np.array([[130, 237], [175, 316]])

    centroids = mean_shift(
        initial_window,
        frames,
        lambda img, hist, bin_edges:
            create_prob_distribution(img, hist, bin_edges, MIN_SATURATION_CAR, MIN_VALUE
        frame_indices_to_draw=[2, 30],
        keyframes=[20, 60, 64, 71, 89, 111, 150],
    )

    None
```

```
M_00 == 0, i = 89
M_00 == 0, i = 90
M_00 == 0, i = 92
```

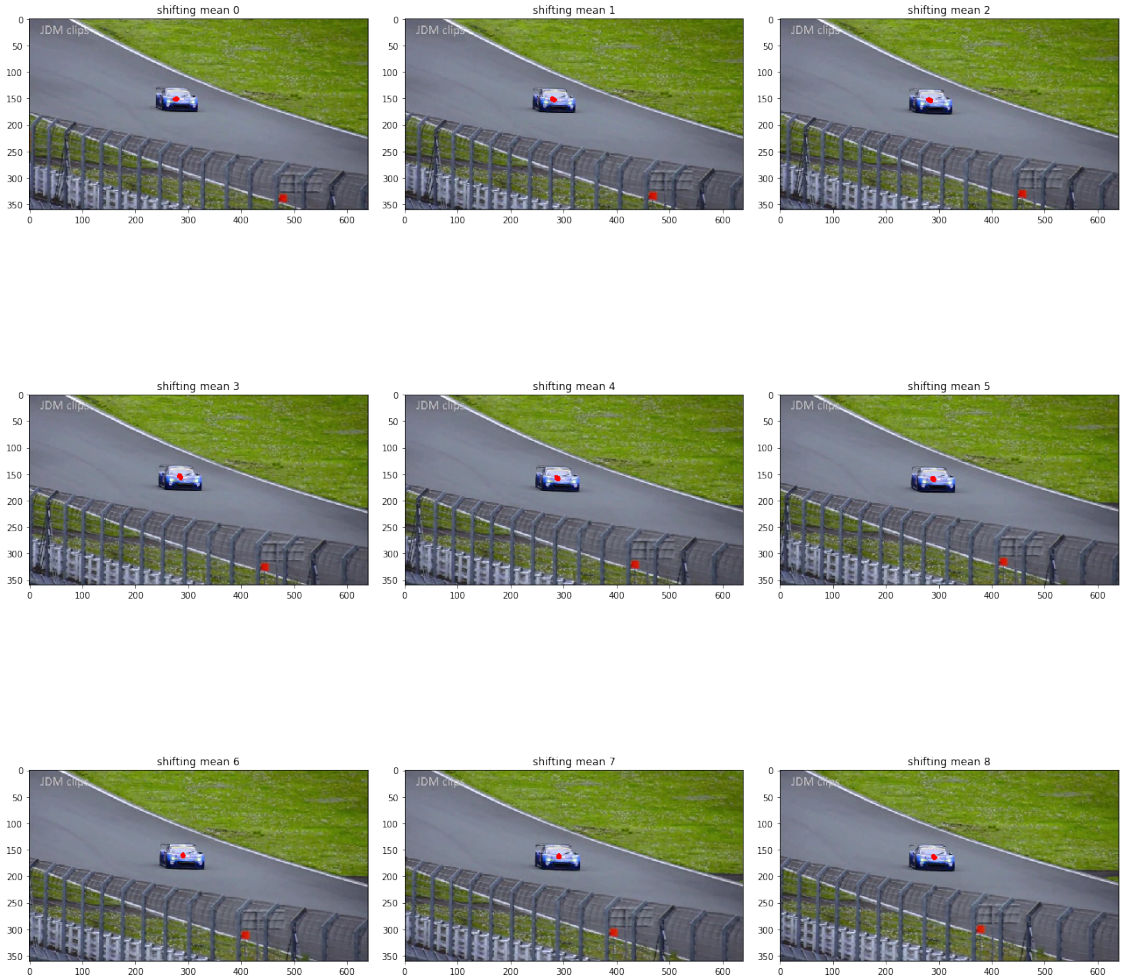
1.5 Exercise 3.3 - CAMSHIFT

- extend your algorithm by adjusting the size of the ROI and finding the object's orientation
- execute the algorithm again on one of the image sequences and draw an ellipse on the image, which represents the found parameters (**RESULT**)

```
In [203]: def get_scaled_next_window(window, moments, v, roi):
              """
              For 2D color probability distributions where the maximum pixel value is 255,
              we set window size s to
              s = 2*sqrt(M_00/256)
              """
              shifted_window = get_shifted_next_window(window, moments, v, roi)
              M_00 = moments[0]
              max_hue = np.max(roi[:, :, 0])
              scaling_factor = 2 * np.sqrt(M_00 / max_hue)
              scaled_window = shifted_window
```

```
            return scaled_window


        frames = io.imread_collection('images/racecar/*')
        initial_window = np.array([[130, 237], [175, 316]])

        centroids = mean_shift(
            initial_window,
            frames,
            lambda img, hist, bin_edges:
                create_prob_distribution(img, hist, bin_edges, MIN_SATURATION_CAR, MIN_VALUE
            frame_indices_to_draw=[*range(0, 9)],
            keyframes=[],
            get_next_window=get_scaled_next_window,
        )

        None
```

```
M_00 == 0, i = 89
M_00 == 0, i = 90
M_00 == 0, i = 92
```



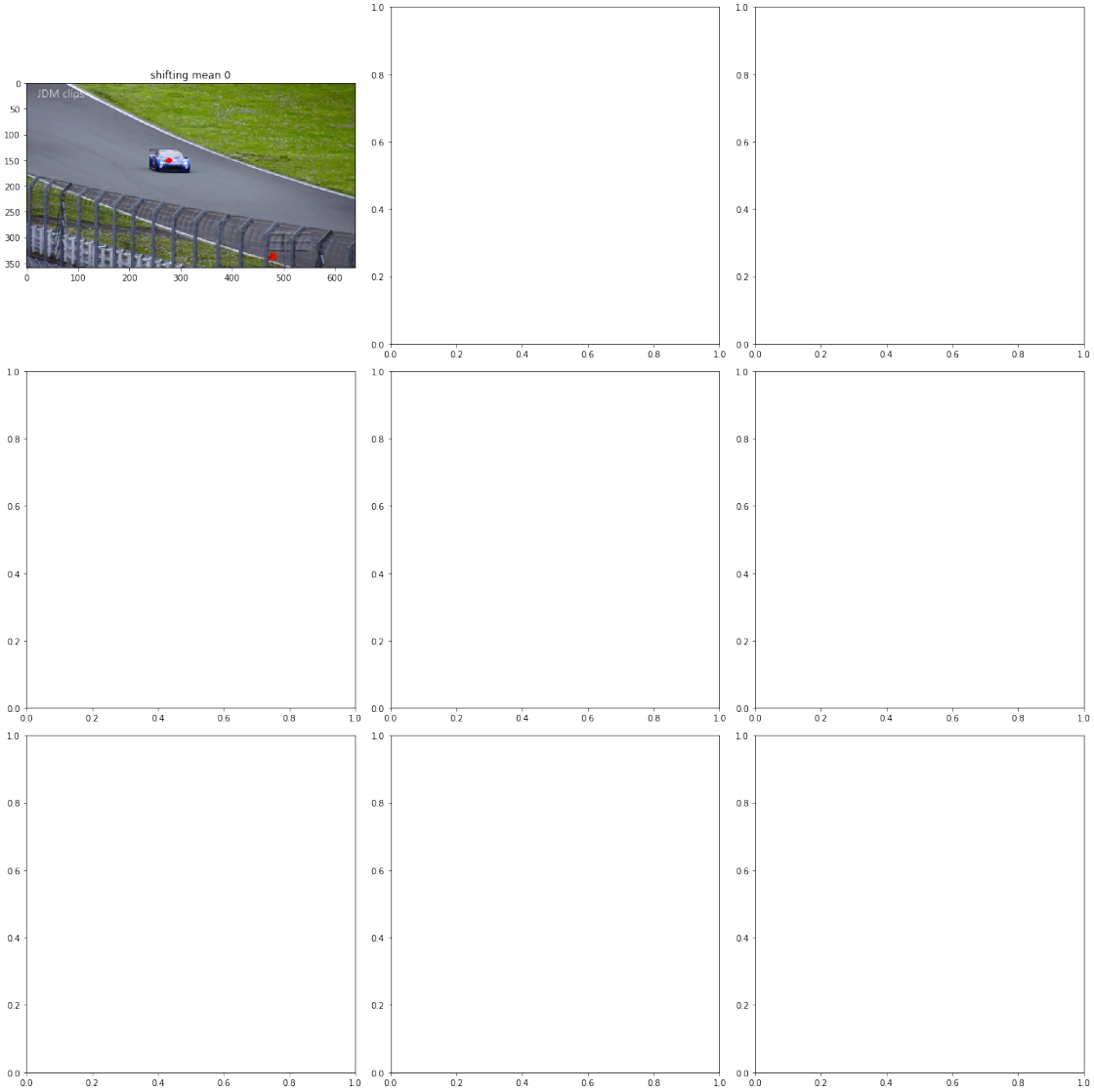
In [221]:
```
from skimage.draw import ellipse_perimeter
```

```
    def get_moments2(prop_dist):
        """Returns 0th, 1st and 2nd order image moments."""
        # (M_00, (M_10, M_01))
        lower_order_moments = M_00, M_1 = get_moments(prop_dist)

        if M_00 > 0:
            num_rows, num_cols = prop_dist.shape
            x_indices = np.arange(0, num_cols)
            y_indices = np.arange(0, num_rows)
            M_20 = np.sum(prop_dist[:] * np.square(x_indices))
            M_02 = np.sum(prop_dist.T[:] * np.square(y_indices))
        else:
            M_20 = M_02 = 0
        return (*lower_order_moments, (M_20, M_02))
```

```
def get_roll_params(centroid, moments):
    x_c, y_c = centroid
    M_00, (M_10, M_01, M_11), (M_20, M_02) = moments

    if M_00 > 0:
        a = M_20/M_00 - x_c**2
        b = 2 * (M_11/M_00 - x_c*y_c)
        c = M_02/M_00 - y_c**2
    else:
        a = -x_c**2
        b = 2 * (-x_c*y_c)
        c = -y_c**2
    p = a + c
    q = np.sqrt(b**2 + (a - c)**2)

    angle = 0.5 * np.arctan(b / (a - c))

    l = np.sqrt(0.5 * (p + q))
    w = np.sqrt(0.5 * (p - q))
    return angle, l, w


def draw_ellipsis(img, centroid, moments):
    img = np.copy(img)
    orientation, l, w = get_roll_params(centroid, moments)
    rr, cc = ellipse_perimeter(*centroid, int(l), int(w), orientation, shape=img.sha
    img[rr, cc] = [255, 0, 0]
    return img



frames = io.imread_collection('images/racecar/*')
initial_window = np.array([[130, 237], [175, 316]])

centroids = mean_shift(
    initial_window,
    frames,
    lambda img, hist, bin_edges:
        create_prob_distribution(img, hist, bin_edges, MIN_SATURATION_CAR, MIN_VALUE_
    frame_indices_to_draw=[*range(0, 9)],
    keyframes=[],
    ellipsis_indices_to_draw=[*range(0, 9)],
    get_moments=get_moments2,
    get_next_window=get_scaled_next_window,
    draw_ellipsis=draw_ellipsis,
)
```

```
None
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-221-98baf73d479f> in <module>()
     62     get_moments=get_moments2,
     63     get_next_window=get_scaled_next_window,
---> 64     draw_ellipsis=draw_ellipsis,
     65 )
     66


<ipython-input-217-8f8fcd6819e4> in mean_shift(initial_window, frames, create_prop_dist
     99             plot_index += 1
    100         if i in ellipsis_indices_to_draw and draw_ellipsis:
--> 101             axes[plot_index].imshow(draw_ellipsis(frame, centroid, moments))
    102             axes[plot_index].set_title(f'roll {i}')
    103             plot_index += 1


<ipython-input-221-98baf73d479f> in draw_ellipsis(img, centroid, moments)
     43     img = np.copy(img)
     44     orientation, l, w = get_roll_params(centroid, moments)
---> 45     rr, cc = ellipse_perimeter(*centroid, int(l), int(w), orientation, shape=img.sh
     46     img[rr, cc] = [255, 0, 0]
     47     return img


ValueError: cannot convert float NaN to integer
```


shifting mean 0
JDM clips