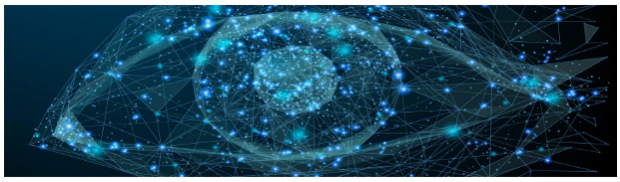
Computer Vision
16720-B Fall 2021



16720 (B) Object Tracking in Videos - Assignment 6

Instructor: Kris TAs: Arka, Rohan, Rawal, Sheng-Yu, Jinkun

Instructions

This section should include the visualizations and answers to specifically highlighted questions from Q1 to Q3. This section will need to be uploaded to gradescope as a pdf and manually graded (this is a separate submission from the coding notebooks)

1. Students are encouraged to work in groups but each student must submit their own work. Include the names of your collaborators in your write up. Code should Not be shared or copied. Please properly give credits to others by LISTING EVERY COLLABORATOR in the writeup including any code segments that you discussed, Please DO NOT use external code unless permitted. Plagiarism is prohibited and may lead to failure of this course.

2. **Start early!** This homework will take a long time to complete.

3. **Questions:** If you have any question, please look at Piazza first and the FAQ page for this homework.

4. All the theory question and manually graded questions should be included in a single writeup (this notebook exported as pdf or a standalone pdf file) and submitted to gradescope: pdf assignment.

5. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge issue when you put everything together. We provide some simple checks in the notebook cells, but make sure you verify them on more complicated samples before moving forward.

6. **Do not import external functions/packages other than the ones already imported in the files:** The current imported functions and packages are enough for you to complete this assignment. If you need to import other functions, please remember to comment them out after submission. Our autograder will crash if you import a new function that the gradescope server does not expect.

7. Assignments that do not follow this submission rule will be **penalized up to 10% of the total score**.

Preliminaries

In this section, we will go through some of the basics of the Lucas-Kanade tracker and the Matthews-Baker tracker. The following table contains a summary of the variables used in the rest of the assignment.

Symbol	Vector/Matrix Size	Description
u	1×1	Image horizontal coordinate
v	1×1	Image vertical coordinate
$\mathbf{x}$	2×1 or 1×1	pixel coordinates: (u, v) or unrolled
$\mathbf{I}$	$m \times 1$	Image unrolled into a vector (m pixels)
$\mathbf{T}$	$m \times 1$	Template unrolled into a vector (m pixels)
$\mathbf{W(p)}$	3×3	Affine warp matrix
$\mathbf{p}$	6×1	parameters of affine warp
$\frac{\partial \mathbf{I}}{\partial u}$	$m \times 1$	partial derivative of image wrt u
$\frac{\partial \mathbf{I}}{\partial v}$	$m \times 1$	partial derivative of image wrt v
$\frac{\partial \mathbf{T}}{\partial u}$	$m \times 1$	partial derivative of template wrt u
$\frac{\partial \mathbf{T}}{\partial v}$	$m \times 1$	partial derivative of template wrt v
$\nabla \mathbf{I}$	$m \times 2$	image gradient $\nabla \mathbf{I}(\mathbf{x}) = \left[\frac{\partial \mathbf{I}(\mathbf{x})}{\partial u} \frac{\partial \mathbf{I}(\mathbf{x})}{\partial v}\right]$
$\nabla \mathbf{T}$	$m \times 2$	image gradient $\nabla \mathbf{T}(\mathbf{x}) = \left[\frac{\partial \mathbf{T}(\mathbf{x})}{\partial u} \frac{\partial \mathbf{T}(\mathbf{x})}{\partial v}\right]$
$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$	2×6	Derivative of affine warp wrt its parameters
$\mathbf{J}$	$m \times 6$	$\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ or $\nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
$\mathbf{H}$	6×6	$\mathbf{J}^T \mathbf{J}$

Template

A template describes the object of interest (eg. a car, football) which we wish to track in a video. Traditionally, the tracking algorithm is initialized with a template, which is represented by a bounding box around the object to be tracked in the first frame of the video. For each of the subsequent frames in the video, the tracker will update its estimate of the object in the image. The tracker achieves this by updating its affine warp.

Warps

What is a warp? An image transformation or warp $\mathbf{W}$ is a function that acts on pixel coordinates $\mathbf{x} = \begin{bmatrix} u & v \end{bmatrix}^T$ and maps pixel values from one place to another in an image $\mathbf{x}' = \begin{bmatrix} u' & v' \end{bmatrix}^T$. Simply put, $\mathbf{W}$ maps a pixel with coordinates $\mathbf{x} = \begin{bmatrix} u & v \end{bmatrix}^T$ to $\mathbf{x}' = \begin{bmatrix} u' & v' \end{bmatrix}^T$.

Translation, rotation, and scaling are all examples of warps. We denote the parameters of the warp function $\mathbf{W}$ by $\mathbf{p}$:

$$\mathbf{x}' = \mathbf{W}(\mathbf{x};\mathbf{p})$$

Affine Warp

An affine warp is a particular kind of warp that can include any combination of translation, scaling, and rotations. An affine warp can be represented by 6 parameters $\mathbf{p} = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]^T$. One of the most convenient things about an affine warp is that it is linear; its action on a point with coordinates $\mathbf{x} = [u\ \ v]^T$ can be described as a matrix operation by a 3×3 matrix $\mathbf{W}(\mathbf{p})$:,

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \mathbf{W}(\mathbf{p}) \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$\mathbf{W}(\mathbf{p}) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Note: For convenience, when we want to refer to the warp as a function, we will use $\mathbf{W}(\mathbf{x};\mathbf{p})$ and when we want to refer to the matrix for an affine warp, we will use $\mathbf{W}(\mathbf{p})$. We will use affine warp and affine transformation interchangeably.

Theory Questions (30 pts)

Before implementing the trackers, let's study some simple problems that will be useful during the implementation first. The answers to the below questions should be relatively short, consisting of a few lines of math and text.

Q1.1

Assuming the affine warp model defined above, derive the expression for the $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in terms of the warp parameters $\mathbf{p} = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]'$.

YOUR ANSWER HERE

$$\mathbf{W}(\mathbf{X},\mathbf{p}) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$\implies$

$$\mathbf{W}(\mathbf{X},\mathbf{p}) = \begin{bmatrix} (1+p_1)u + p_3 v + p_5 \\ p_2 u + (1+p_4)v + p_6 \\ 1 \end{bmatrix}$$

$$\frac{\partial W}{\partial p} = \begin{bmatrix} \frac{\partial W_1}{\partial p} \\ \frac{\partial W_2}{\partial p} \end{bmatrix} = \begin{bmatrix} u & 0 & v & 0 & 1 & 0 \\ 0 & u & 0 & v & 0 & 1 \end{bmatrix} \tag{1}$$

Q1.2

Find the computational complexity (Big O notation) for each runtime iteration (computing $\mathbf{J}$ and $\mathbf{H}^{-1}$) of the Lucas Kanade method. Express your answers in terms of n, m and p where n is the number of pixels in thetemplate $\mathbf{T}$, m is the number of pixels in an input image $\mathbf{I}$ and p is the number of parameters used to describe the warp W.

You may refer to the supplementary PDF for more detailed descriptions of the algorithm.

YOUR ANSWER HERE

1. For image warping by an affine warp (on every pixel): $O(mp)$
2. $O(np)$ for finding the error by taking the difference.
3. For warping the gradient by an affine warp: $O(mp)$
4. To evaluate the jacobian: $O(np)$ (n pixels and p parameters)
5. For getting the steepest descent, we multiply ∇I and $\frac{\partial W}{\partial p}$. Therefore: $O(np)$
6. Commputation of Hessian: $O\left(n^2 p\right)$
7. For getting the difference that has to be updated, we multiply three matrices. Therefore, $O\left(n^3\right)$
8. Final update step: $O(np)$

Therefore overall complexity will be given by: $O\left(n^3 + n^2 p\right)$

Q1.3

Find the computational complexity (Big O notation) for the initialization step (Precomputing $\mathbf{J}$ and $\mathbf{H}^{-1}$) and for each runtime iteration of the Matthews-Baker method. Express your answers in terms of n, m and p where n is the number of pixels in the template $\mathbf{T}$, m is the number of pixels in an input image $\mathbf{I}$ and p is the number of parameters used to describe the warp W. You may refer to the supplementary PDF for more detailed descriptions of the algorithm.

How does this compare to the run time of the regular Lucas-Kanade method?

YOUR ANSWER HERE
Before the loop:

1. $= O(n)$ to calculate gradient of template image

2. $O(np)$ for Jcobian
3. For getting the steepest descent: $O\left(n^2p\right)$
4. To calculate the Hessian matrix: $O\left(n^2p\right)$

Inside the loop:

- For image warping by an affine warp (on every pixel): $O(mp)$
- $O(np)$ for finding the error by taking the difference.
- To multiply J^T and E: $O(mp)$
- For getting δp, we multiply three matrices. Therefore, $O\left(n^3\right)$
- Final warp update is of the order $O\left(p^2\right)$

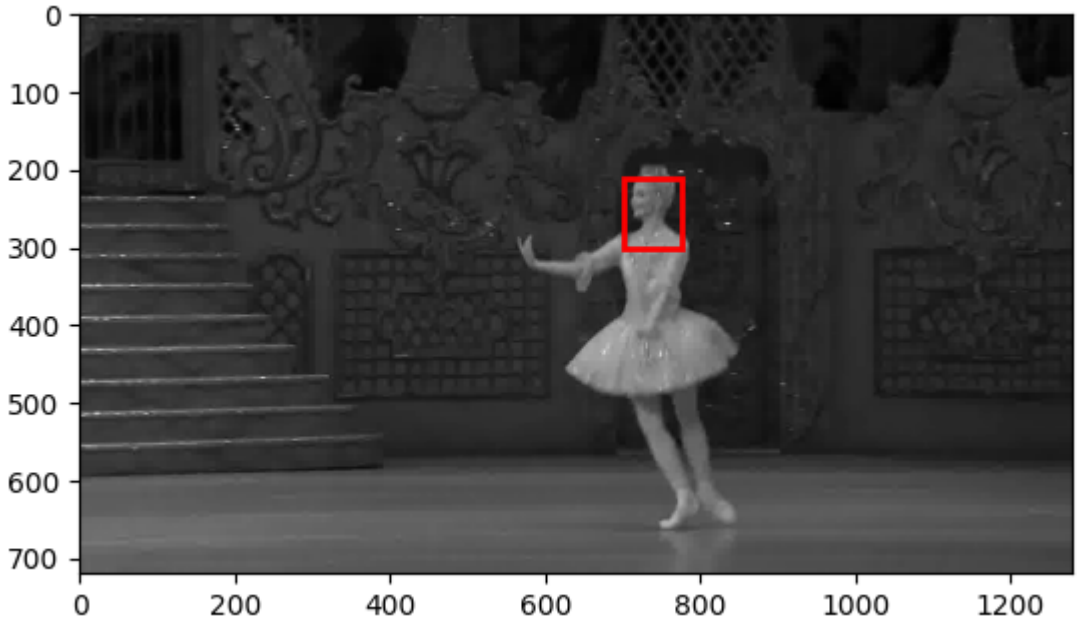
So before the loop: $O\left(n^2p\right)$ and in loop computation is of the order: $O\left(mp+p^3\right)$

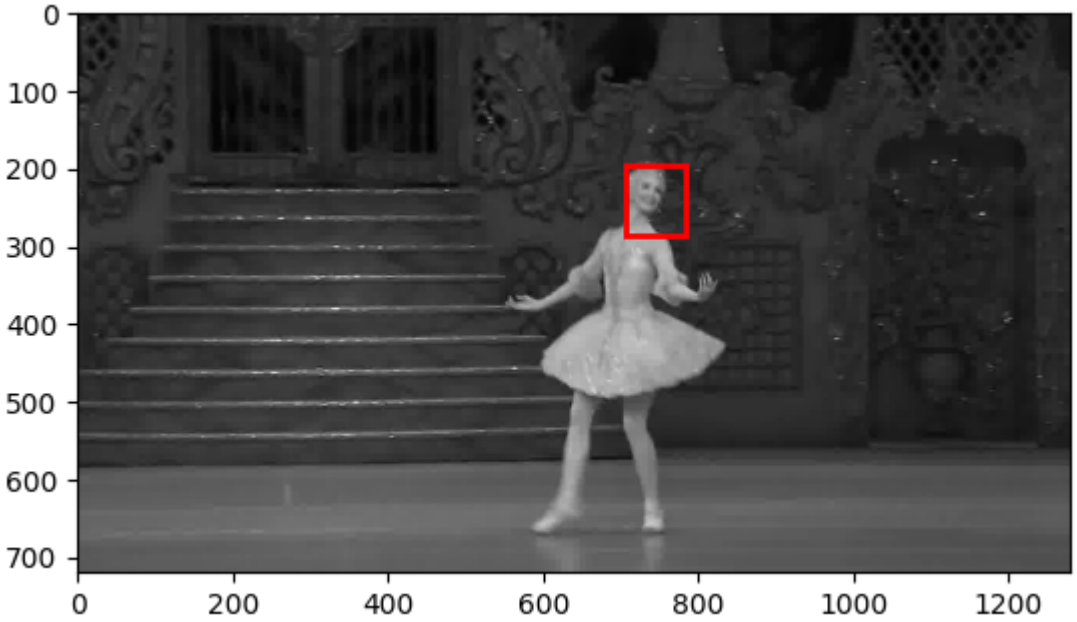
If we compare this to the previous one (Lucas-Kanade), we can see that this is significantly smaller! (as p << n).

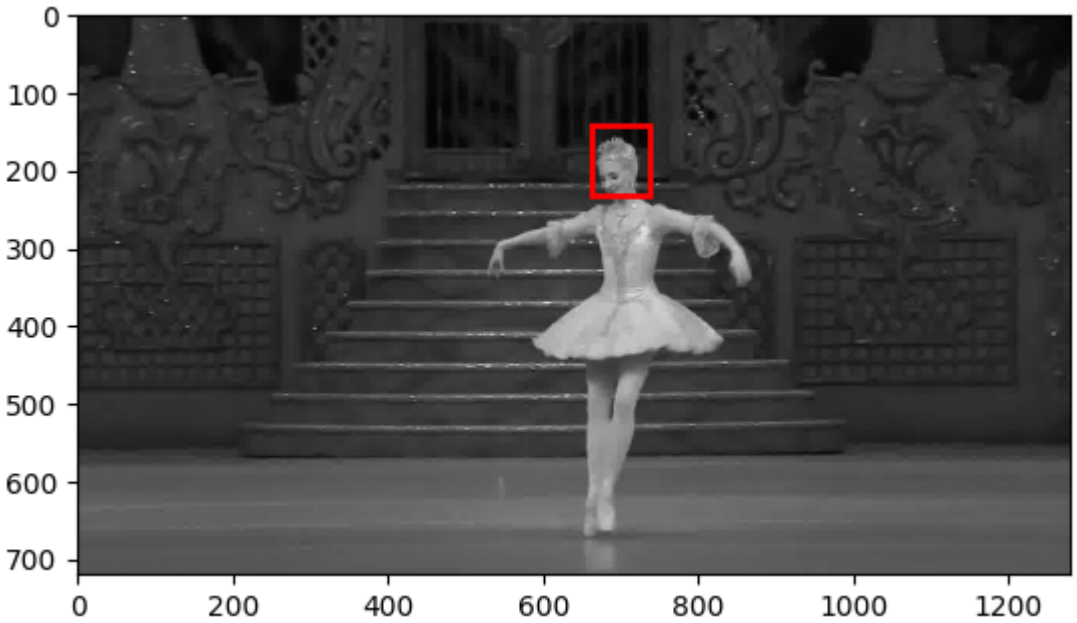
Coding Questions Write-up

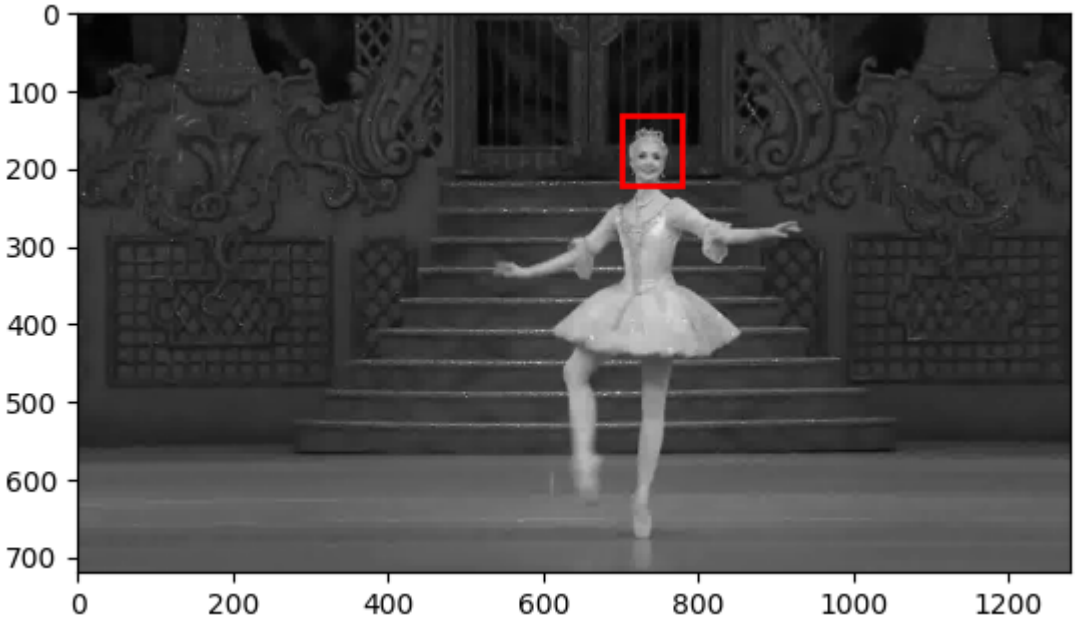
Q1.1

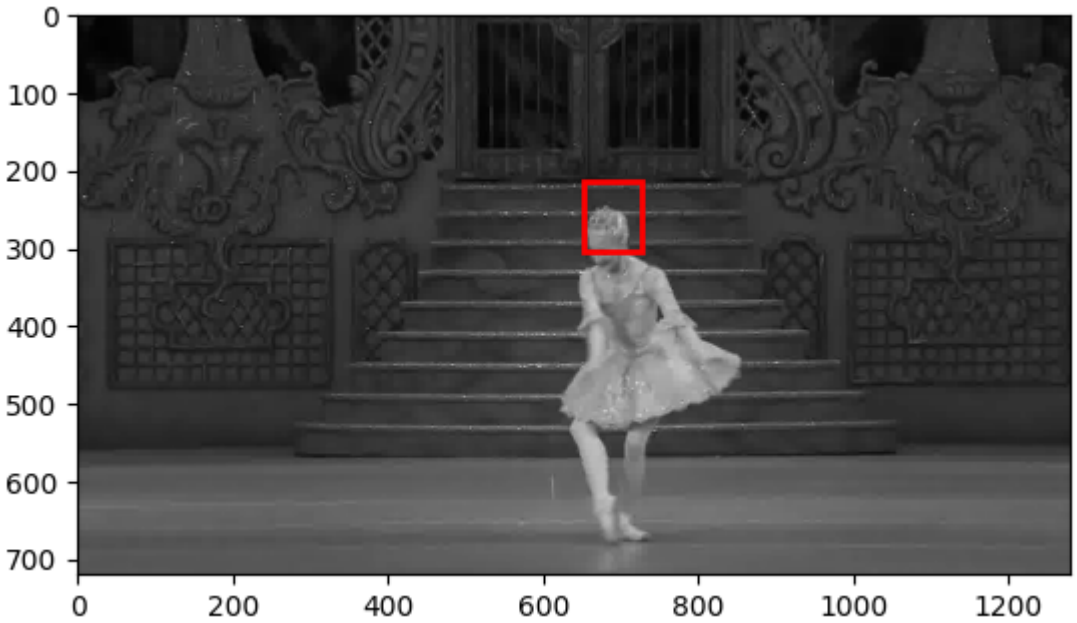
YOUR ANSWER HERE
Results for ballet.npy:



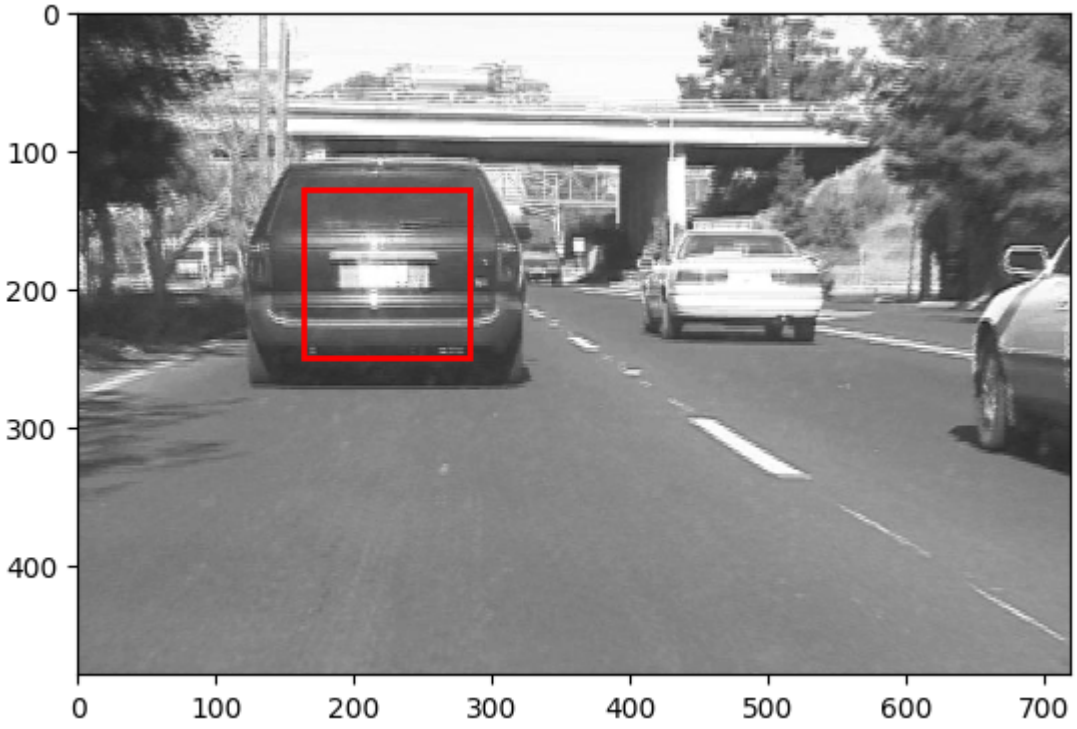

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Results for car1.npy:




0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

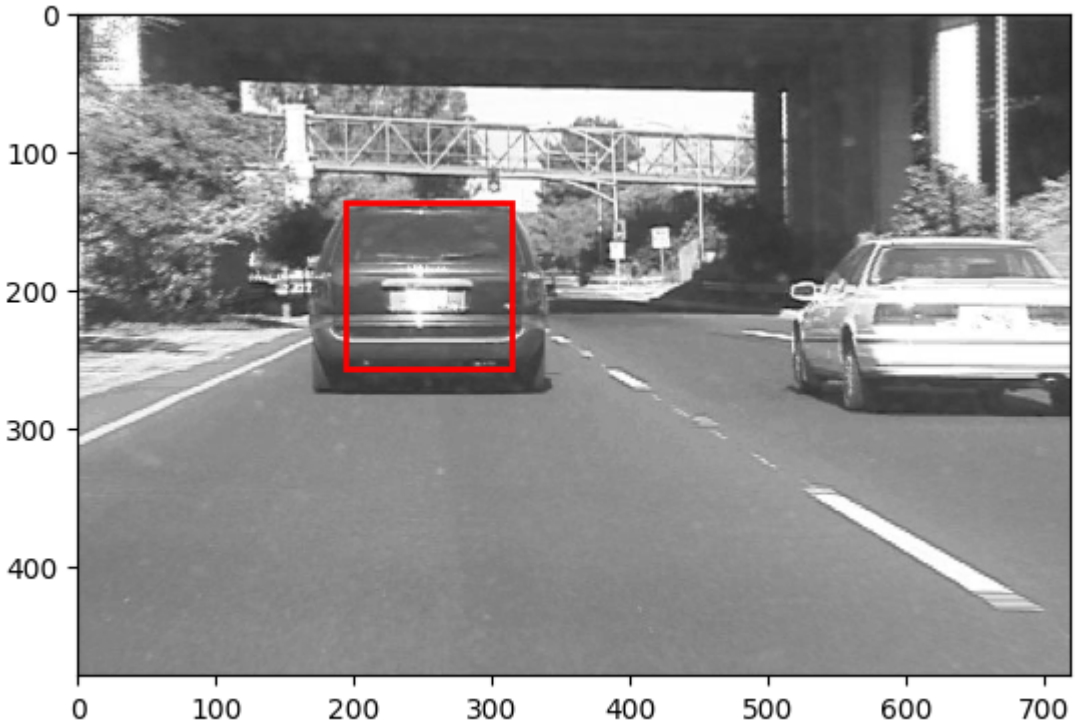

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

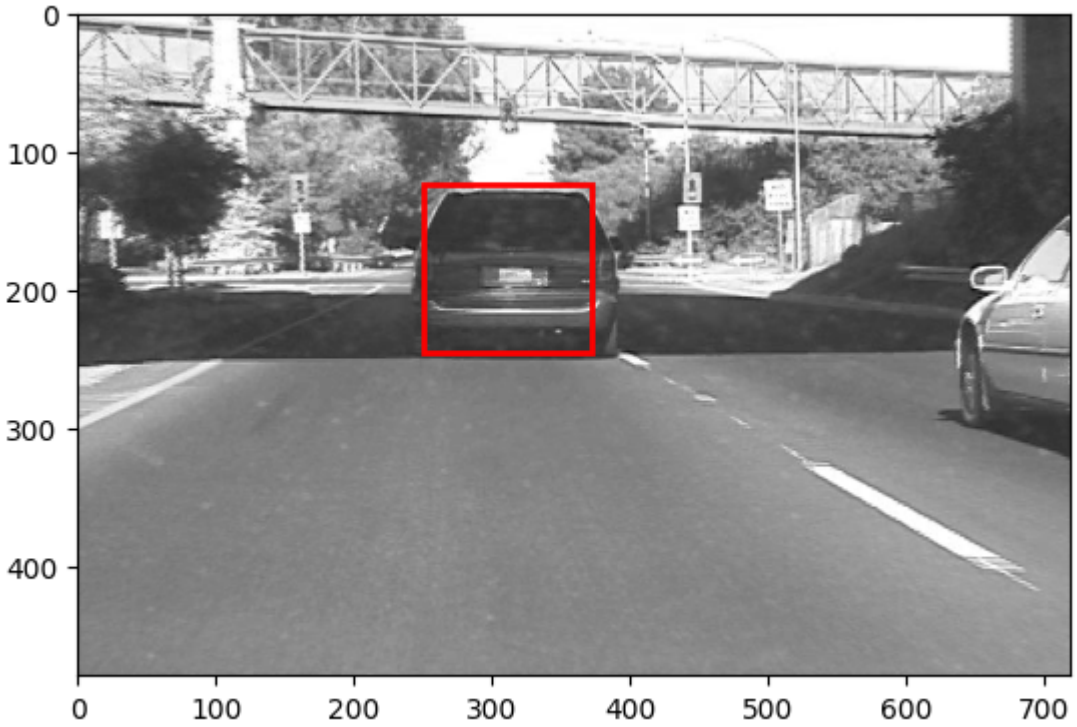

0
100
200
300
400
0
100
200
300
400
500
600
700

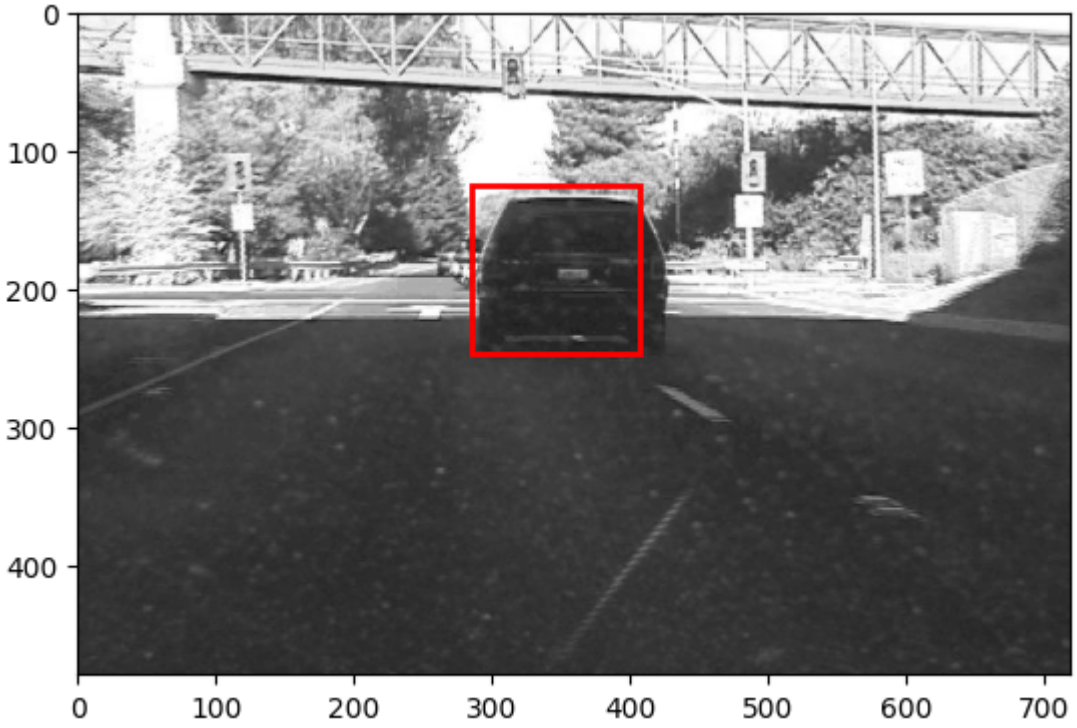

0
100
200
300
400
0
100
200
300
400
500
600
700

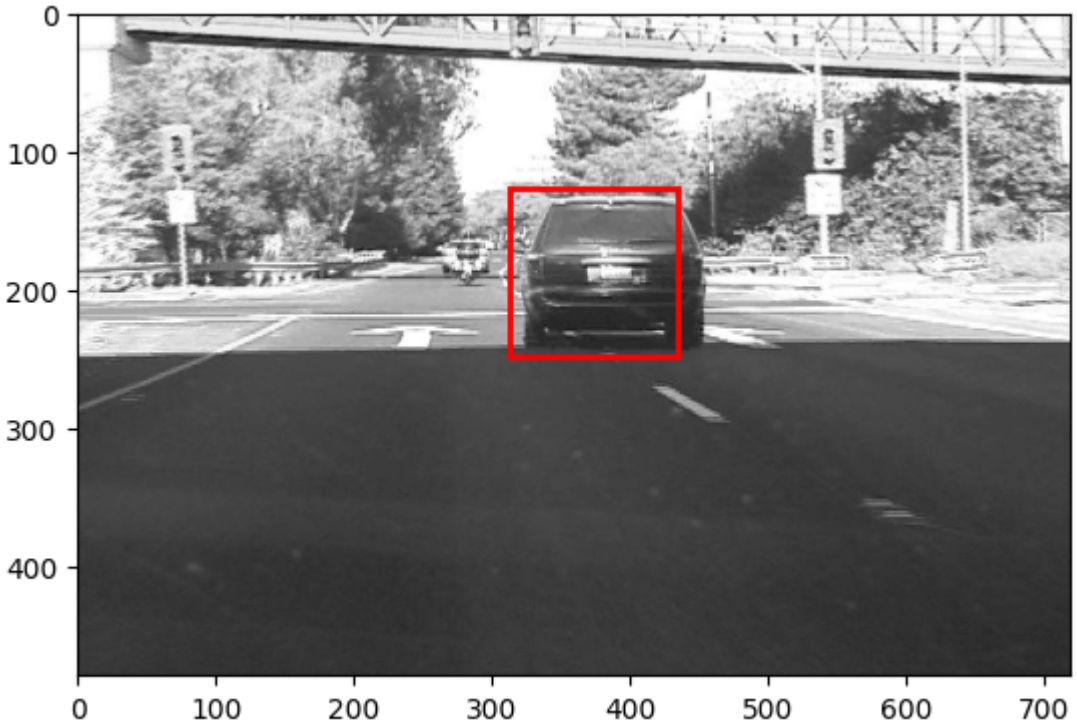

0
100
200
300
400
0
100
200
300
400
500
600
700

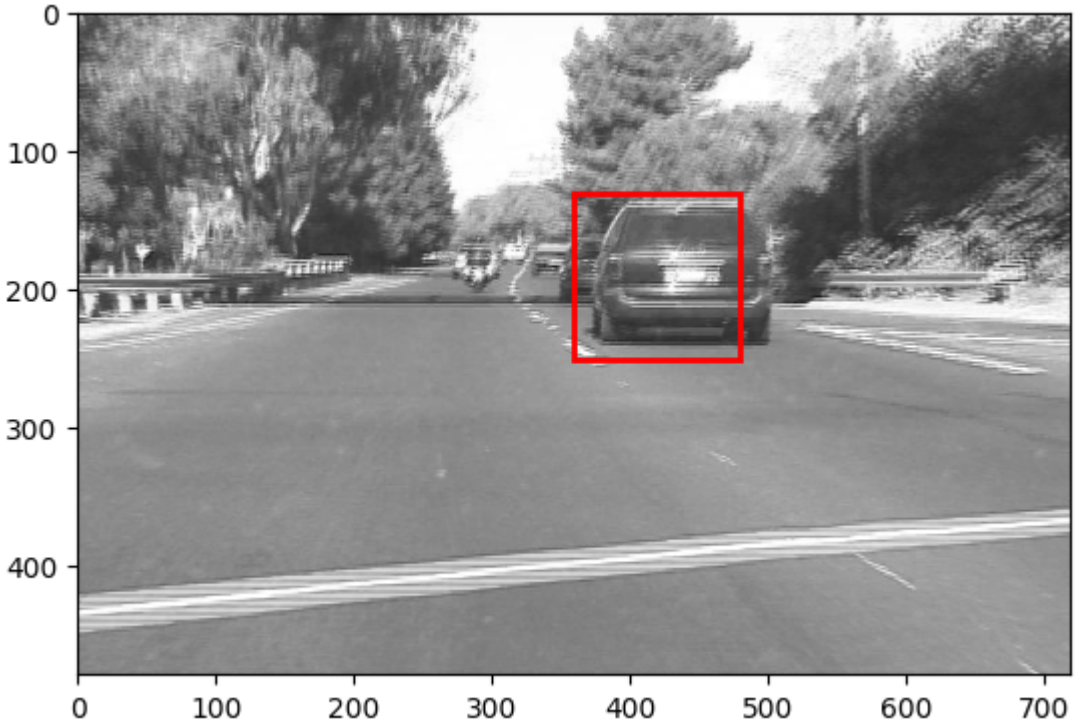

0
100
200
300
400
0
100
200
300
400
500
600
700

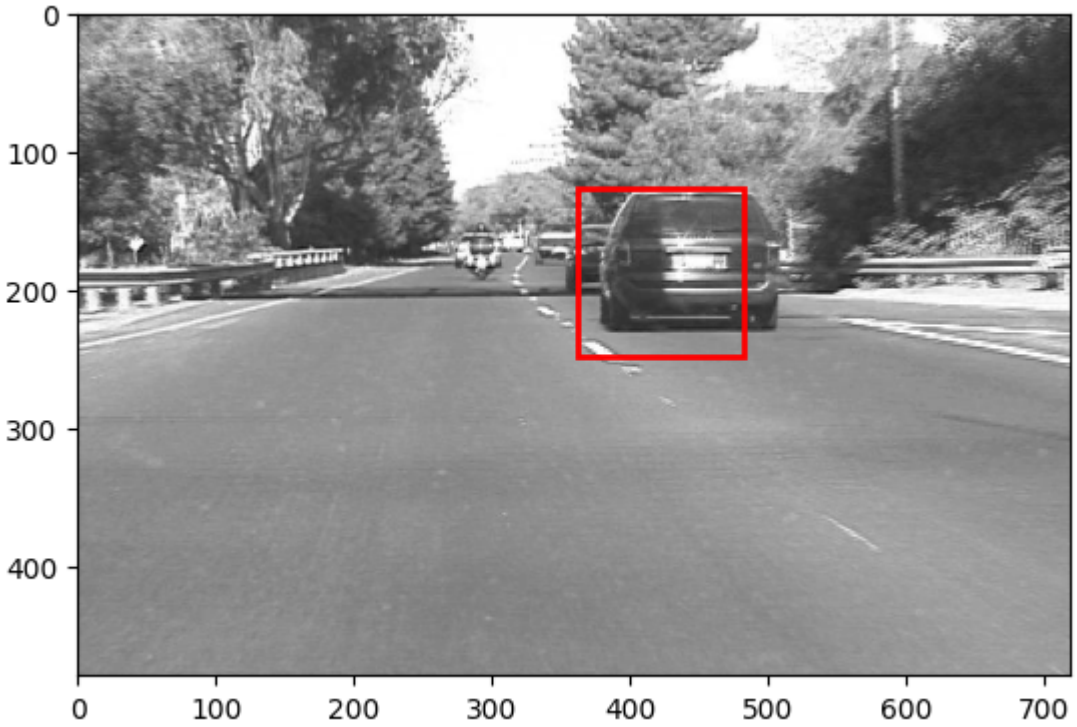

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

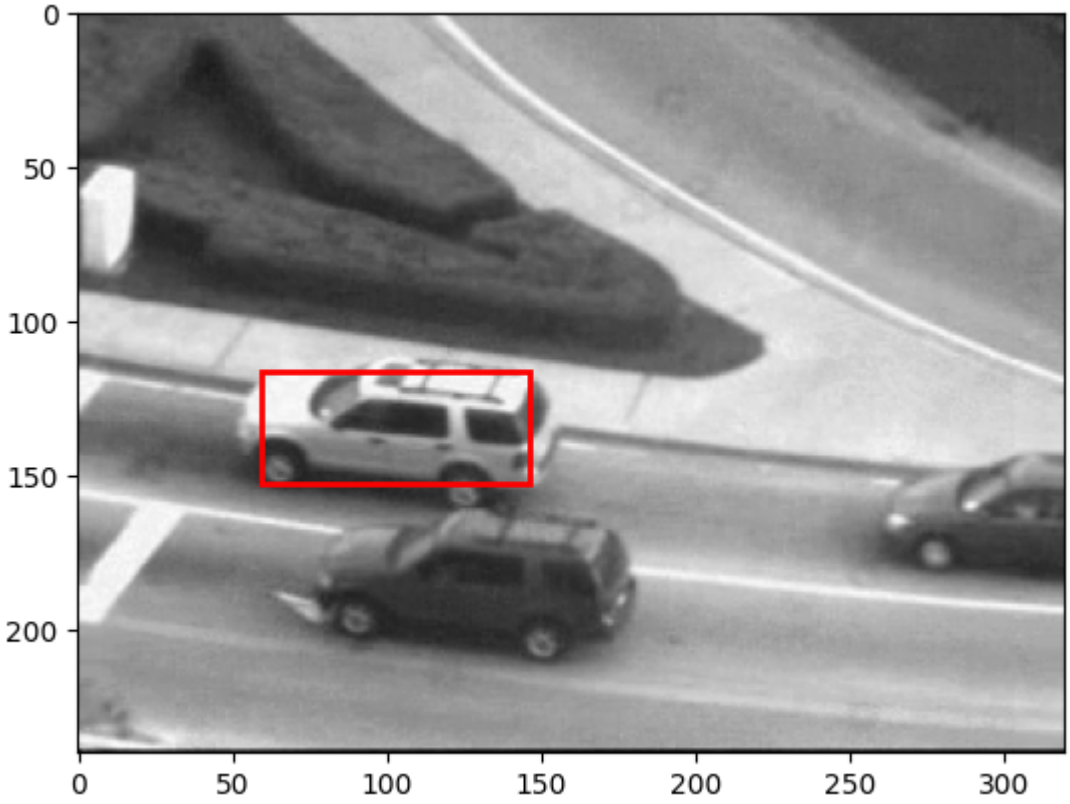

0
100
200
300
400
0
100
200
300
400
500
600
700

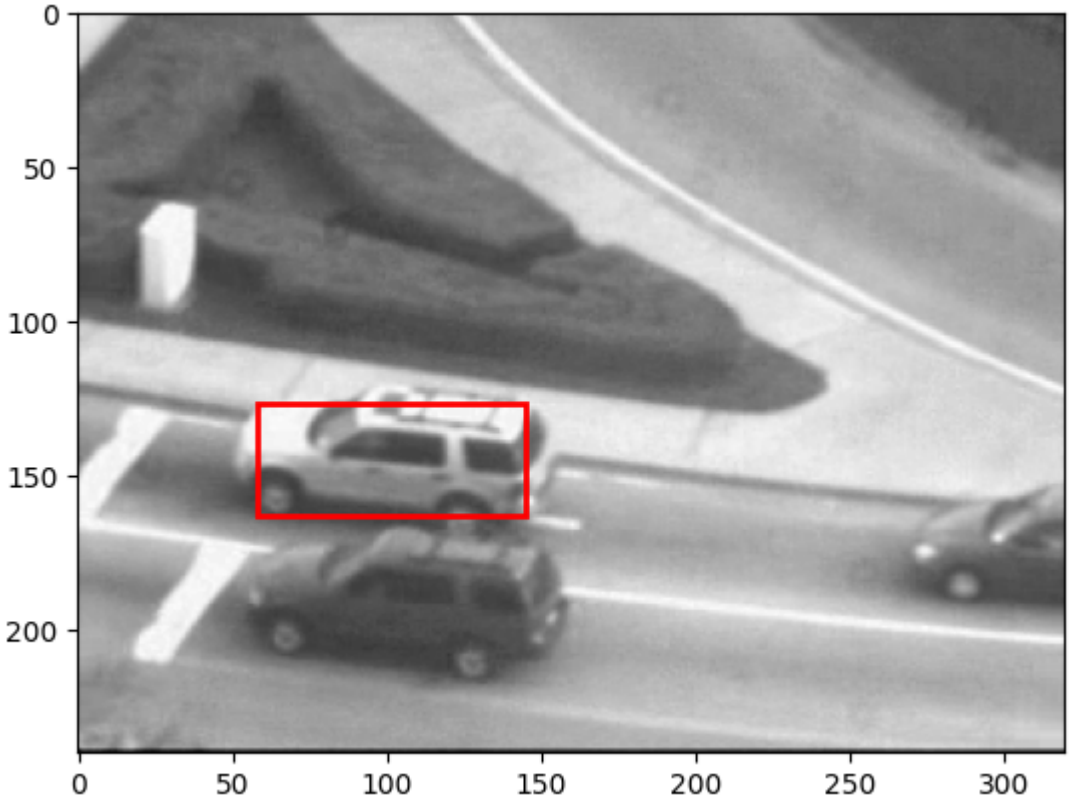

0
100
200
300
400
0
100
200
300
400
500
600
700

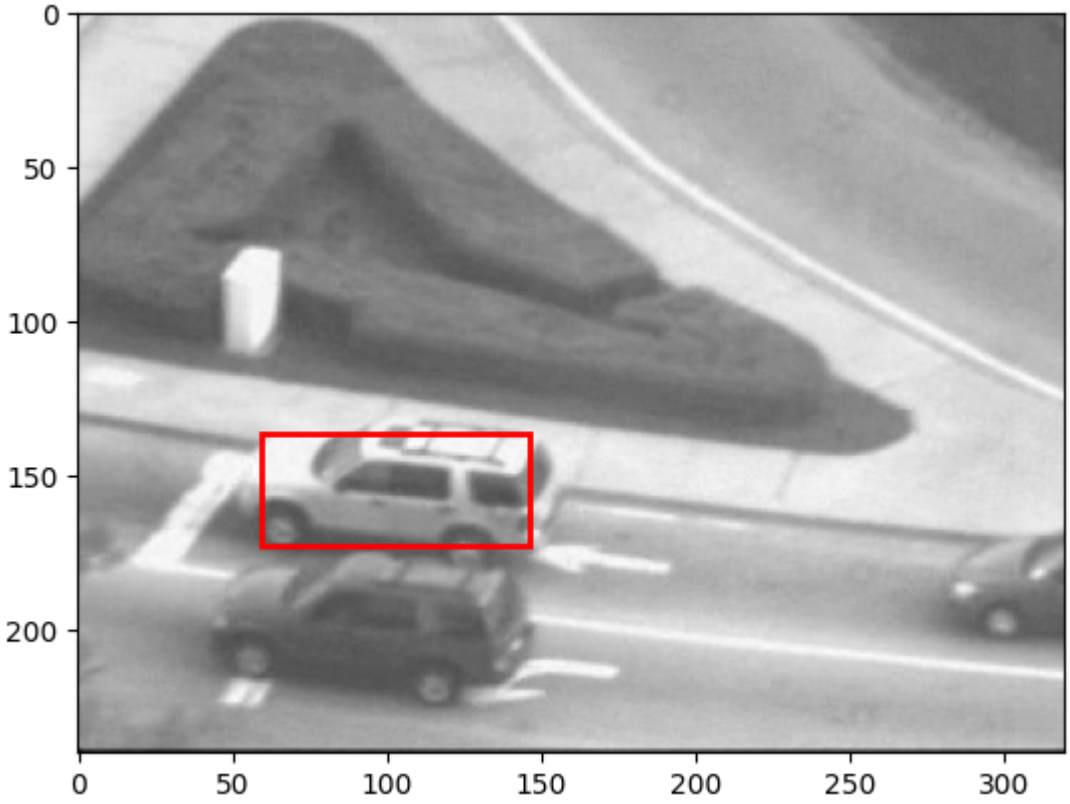

0
100
200
300
400
0
100
200
300
400
500
600
700

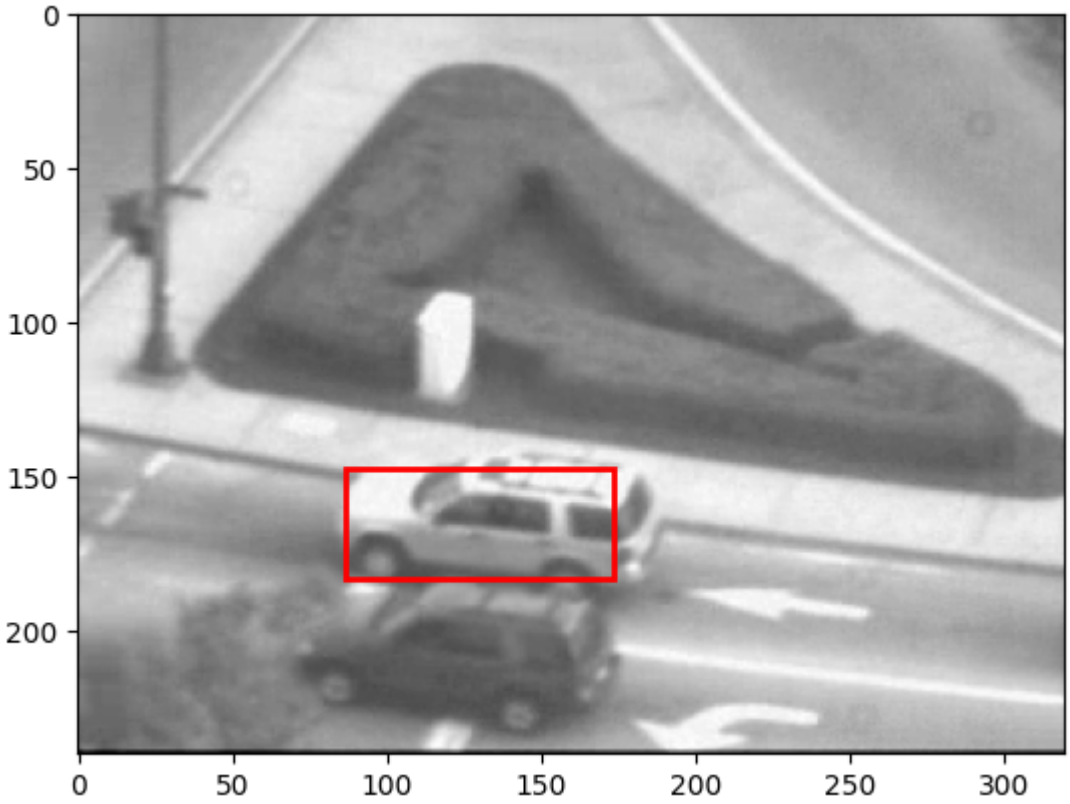

0
100
200
300
400
0
100
200
300
400
500
600
700

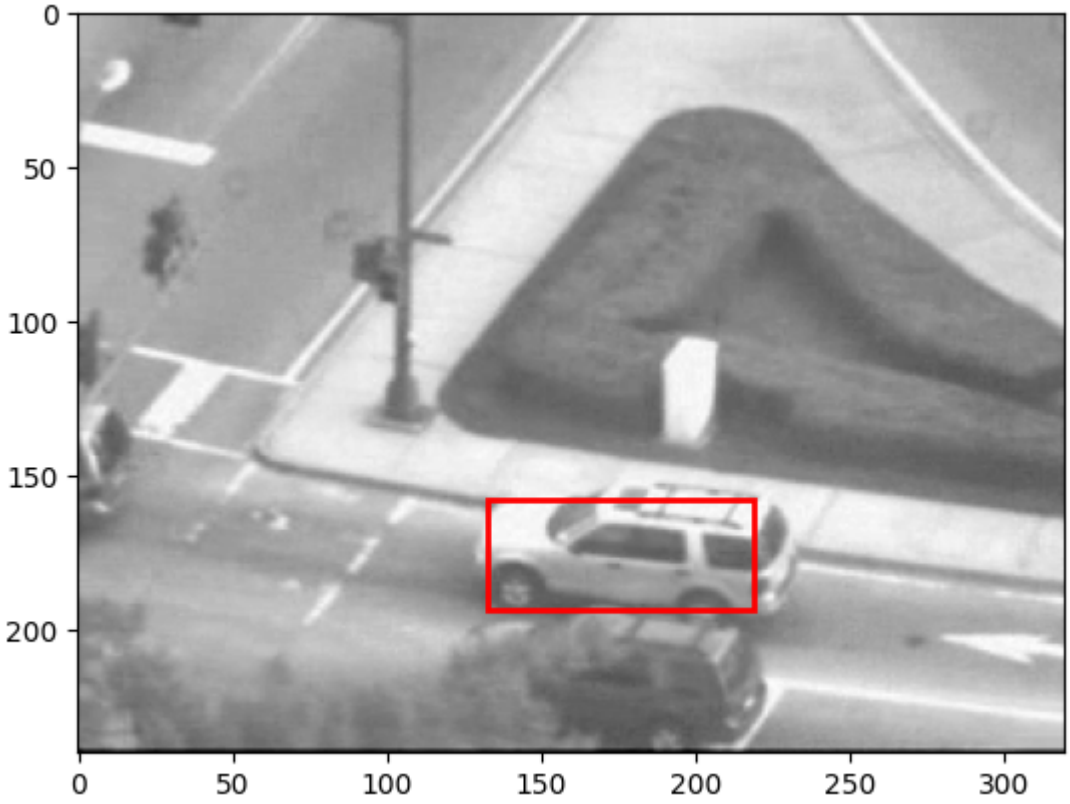
Results for car2.npy



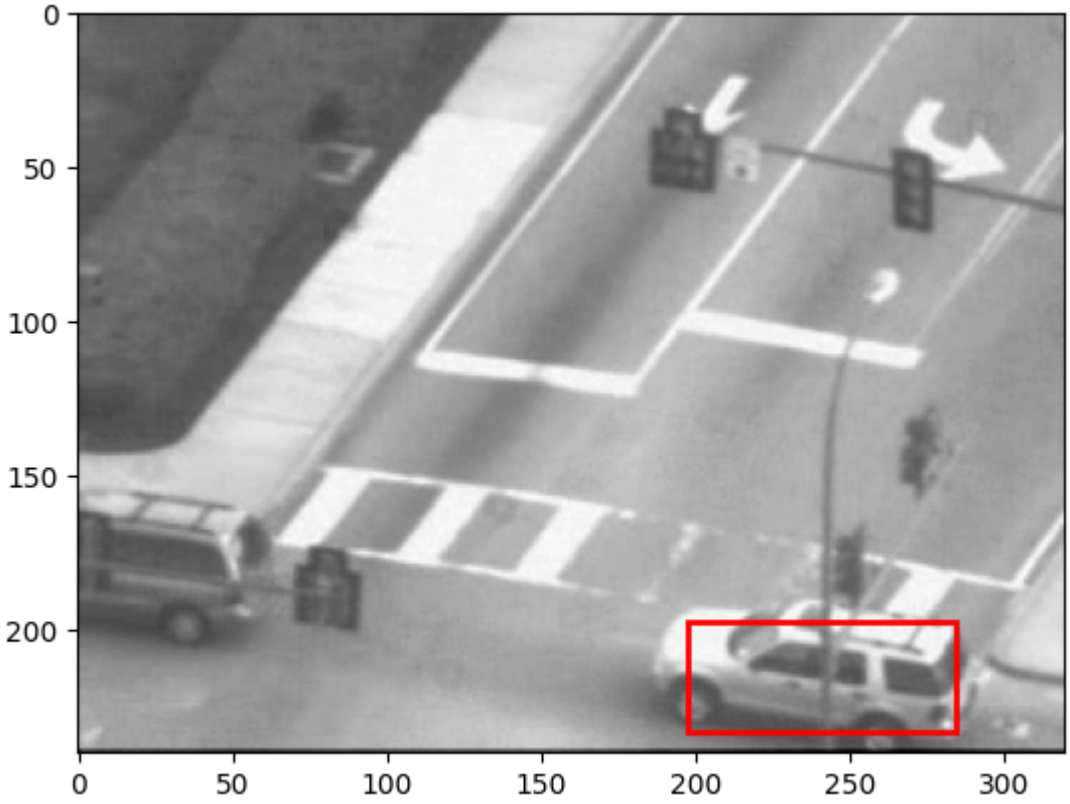


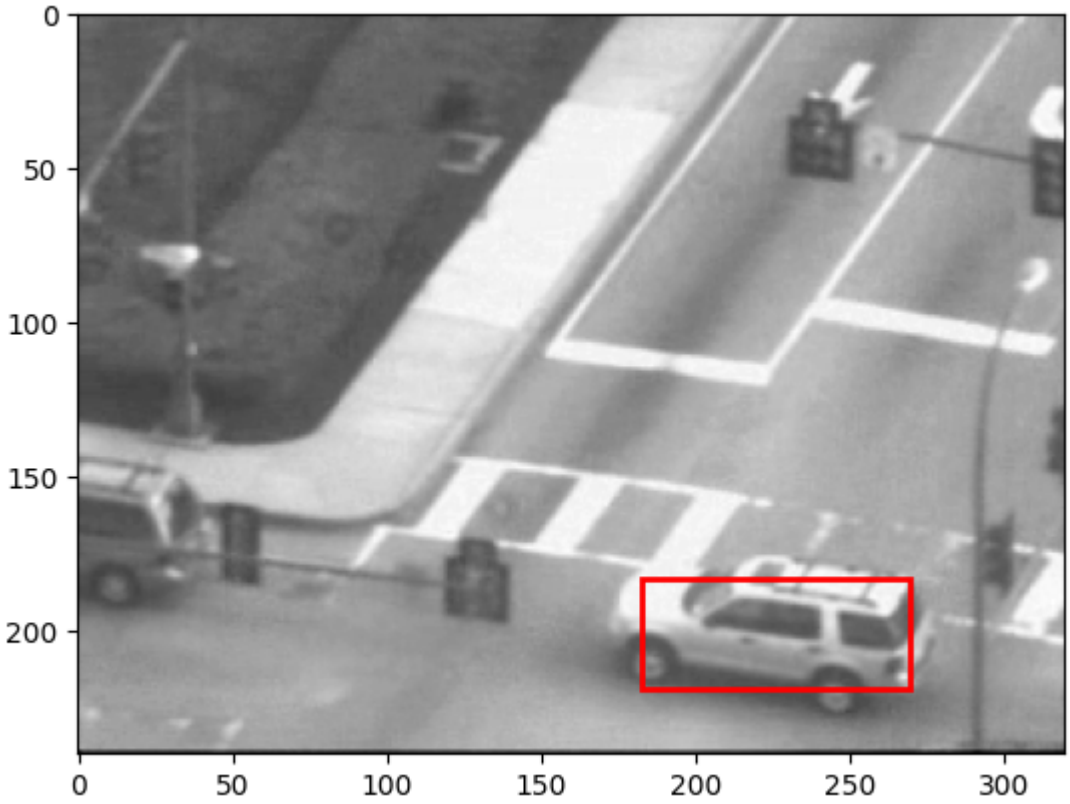

0
50
100
150
200
0
50
100
150
200
250
300

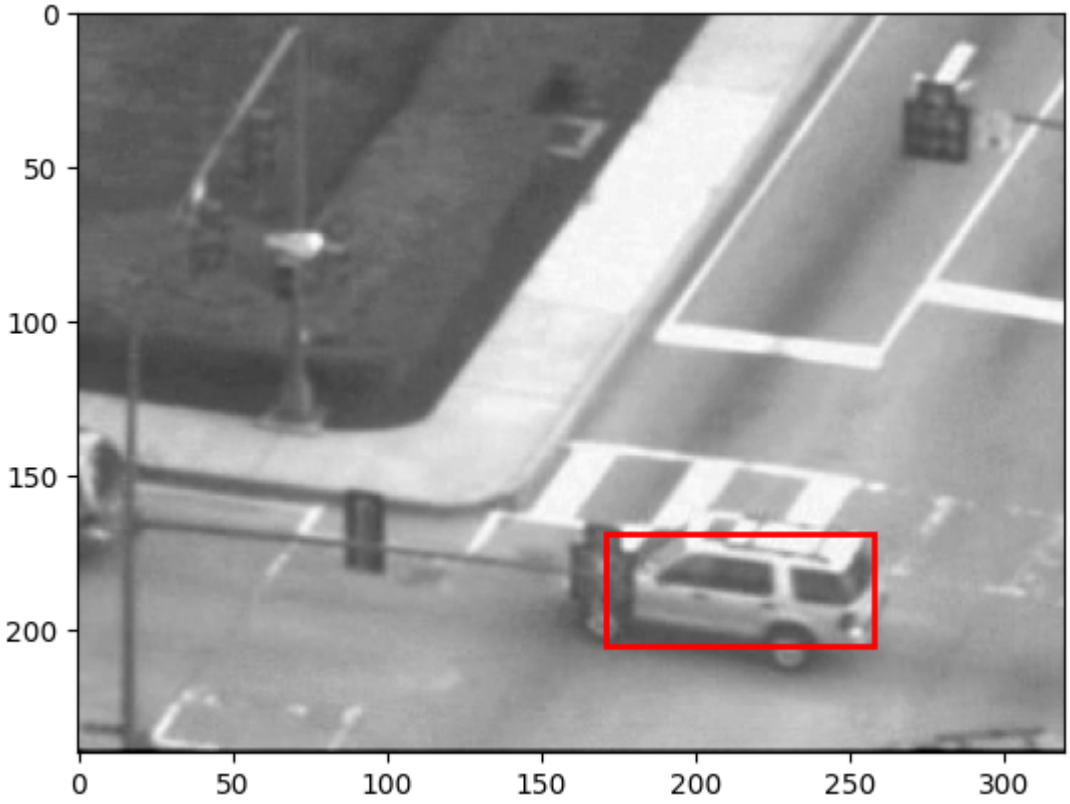

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

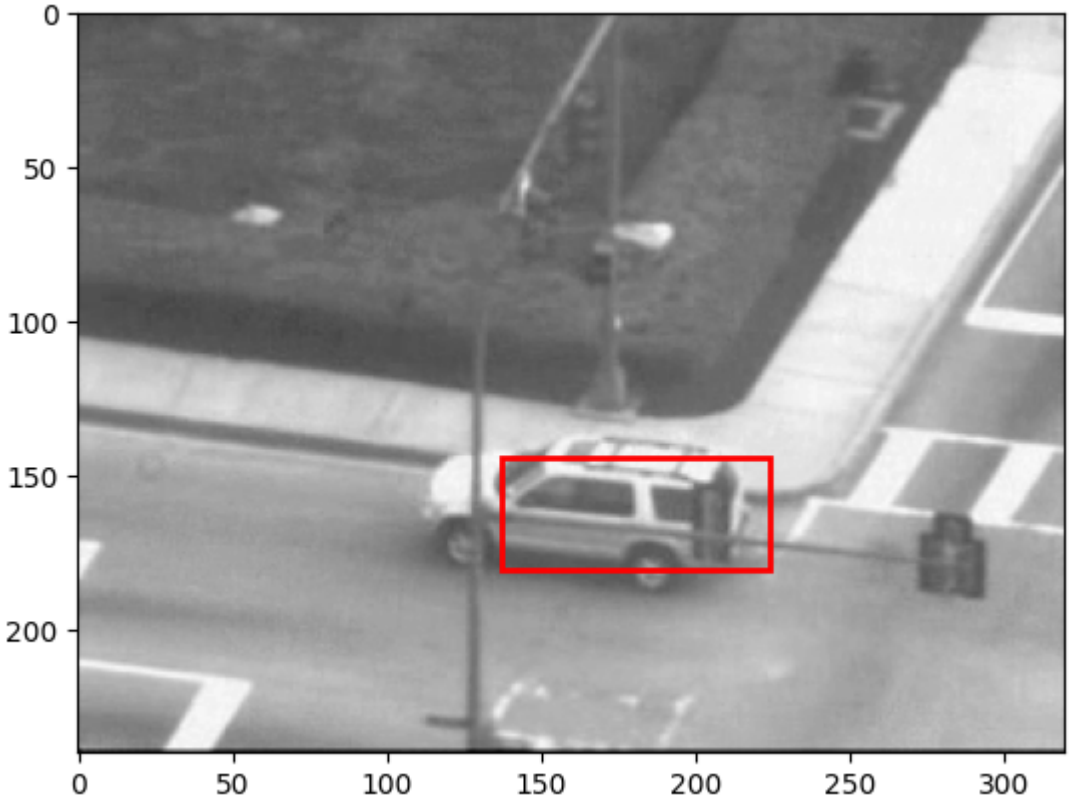

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

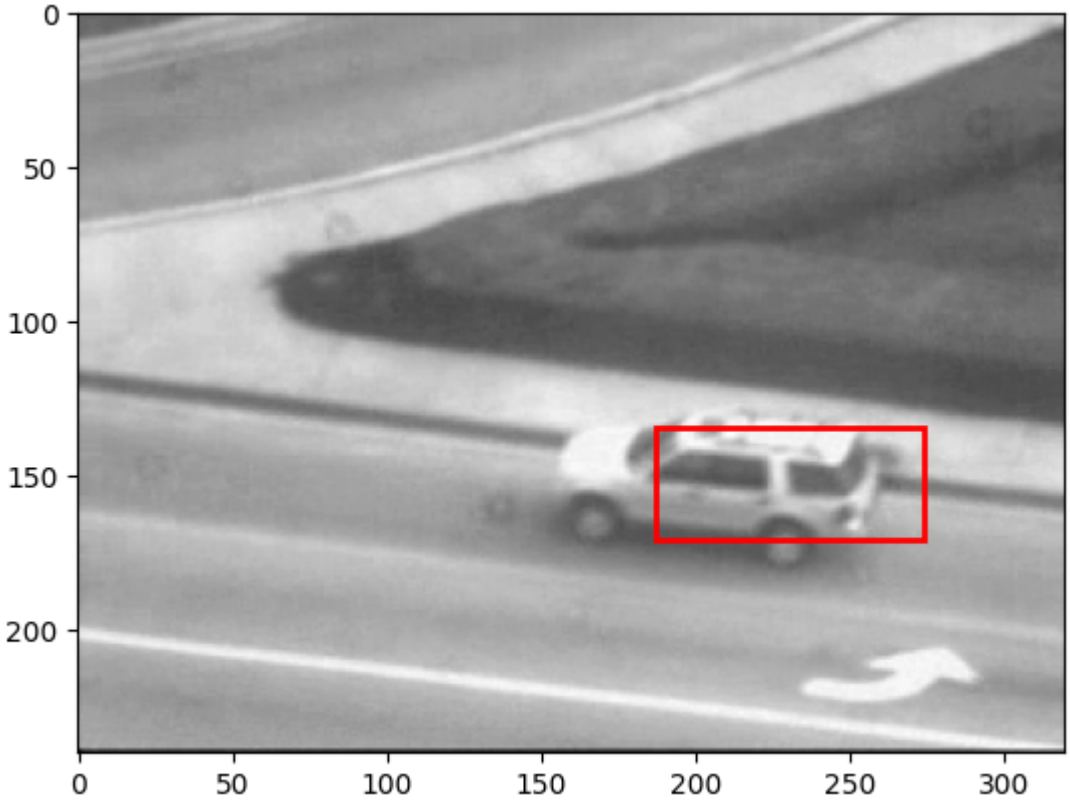

0
50
100
150
200
0
50
100
150
200
250
300

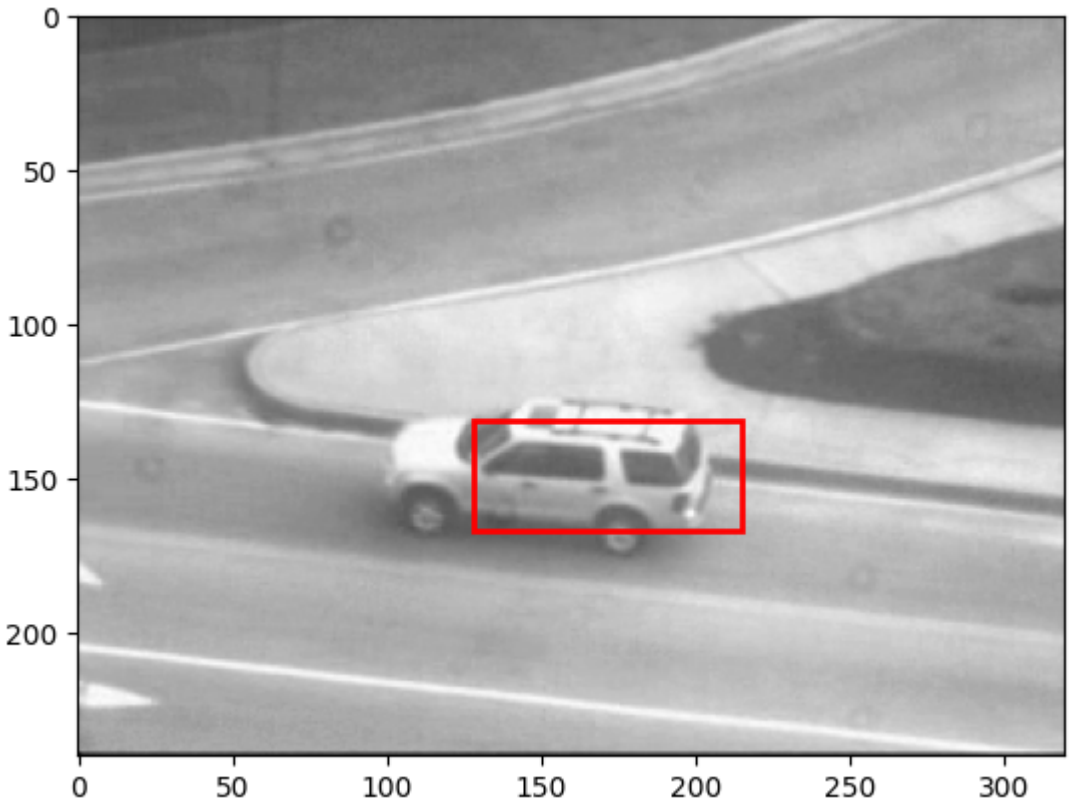

0
50
100
150
200
0
50
100
150
200
250
300

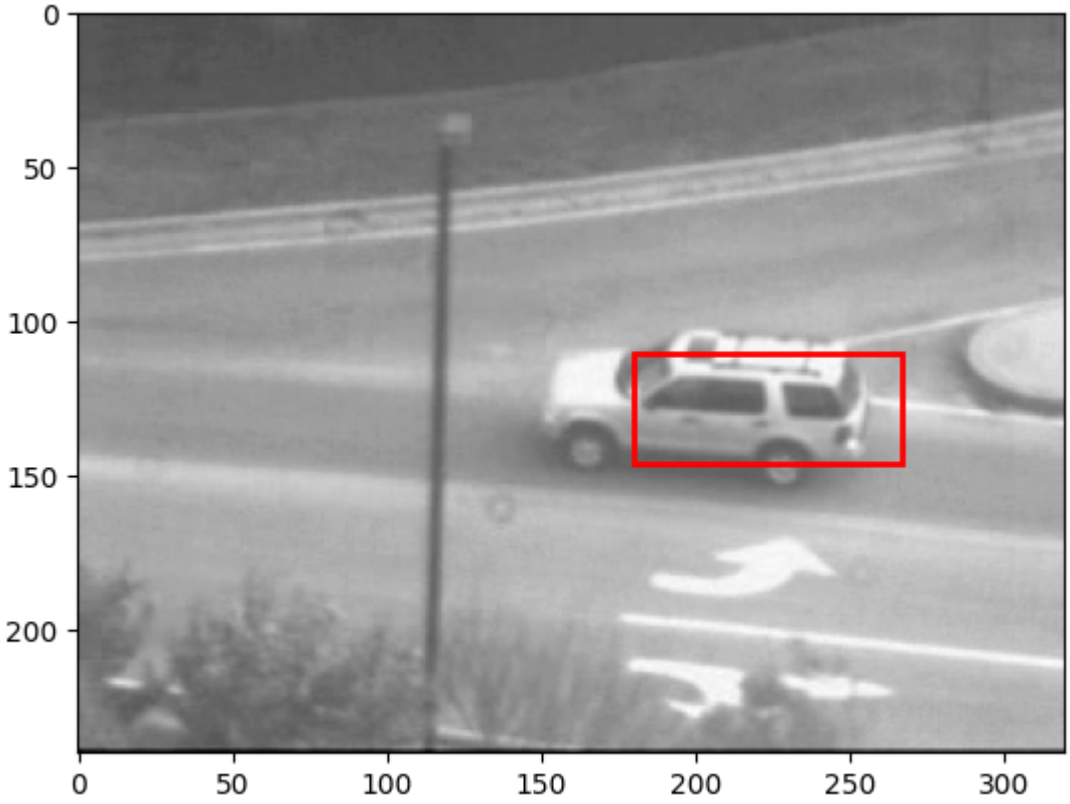

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

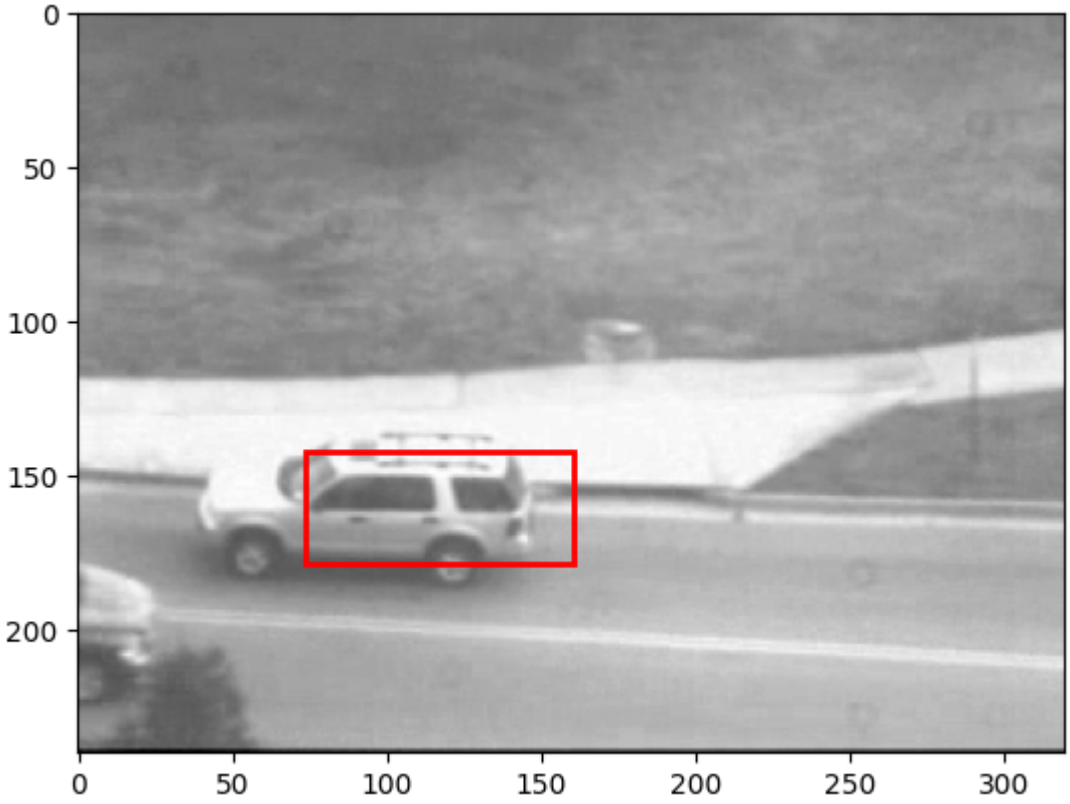

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

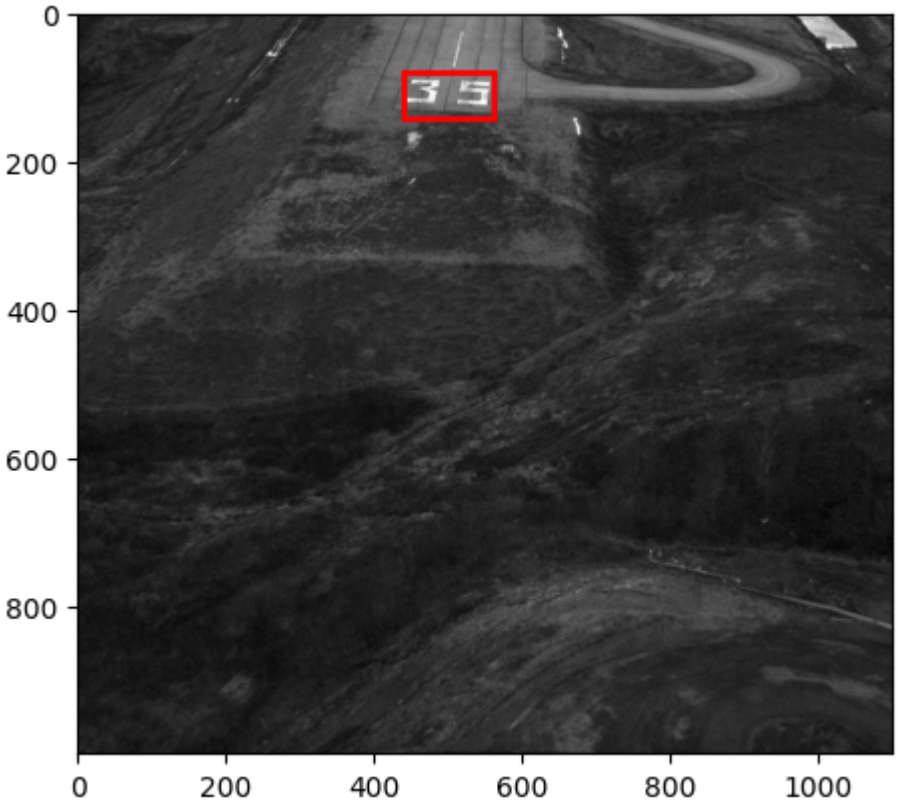

0
50
100
150
200
0
50
100
150
200
250
300

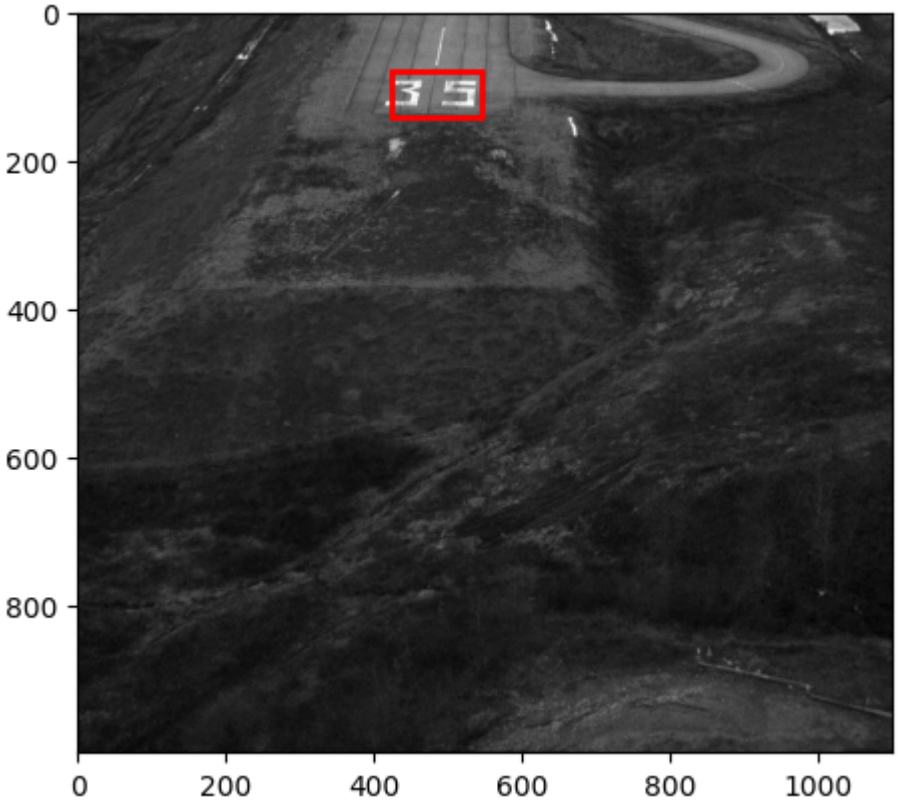

0
50
100
150
200
0
50
100
150
200
250
300

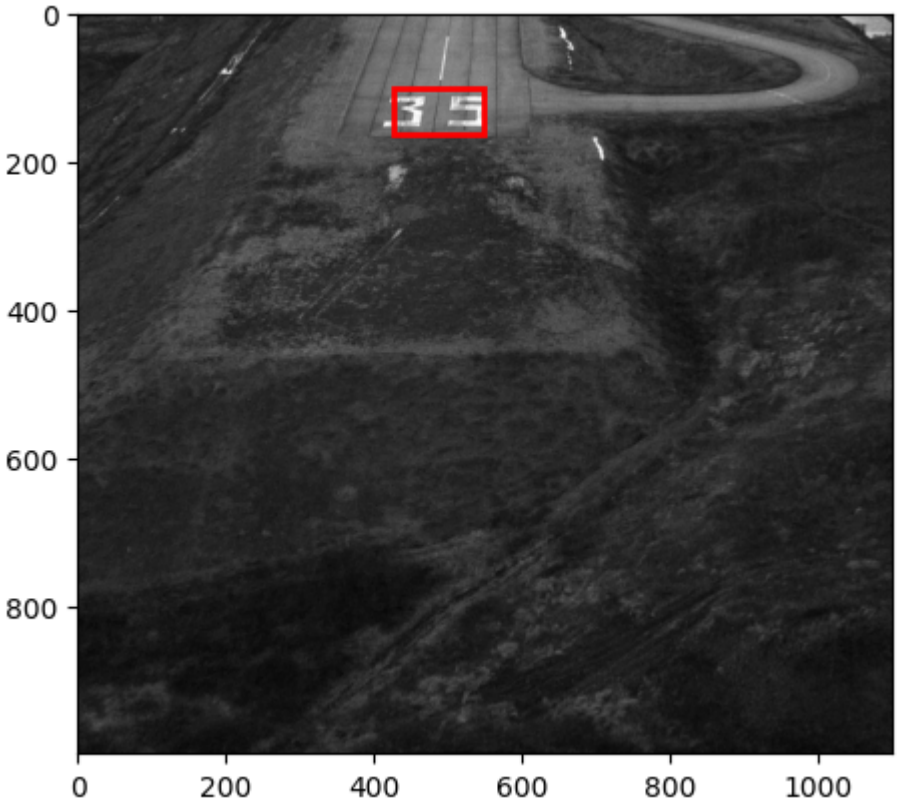

0
50
100
150
200
0
50
100
150
200
250
300

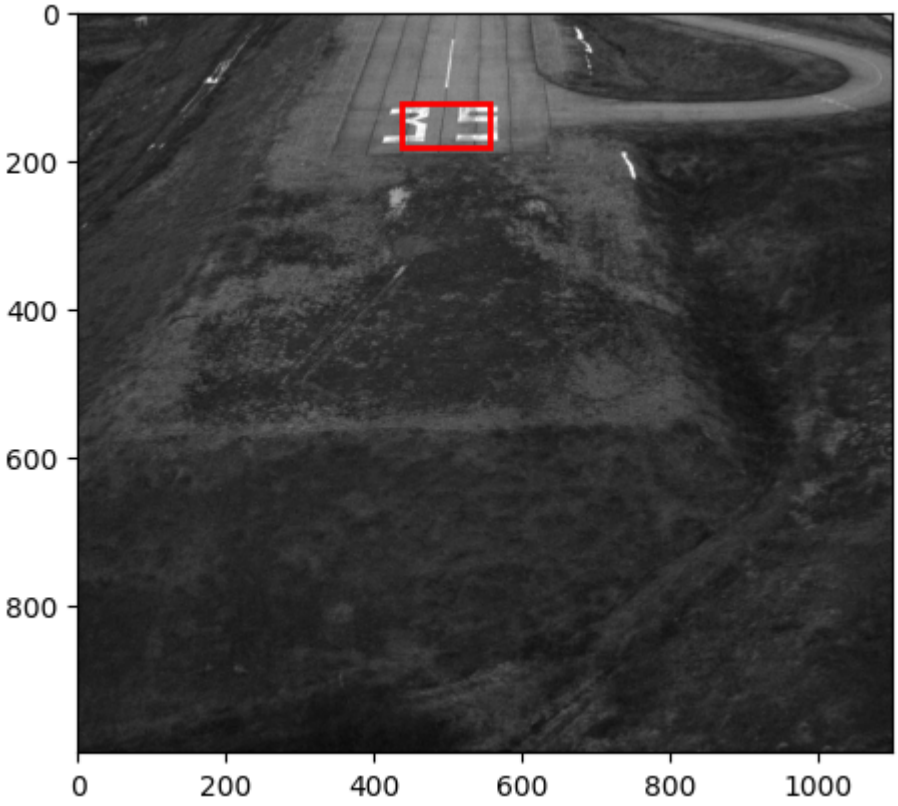


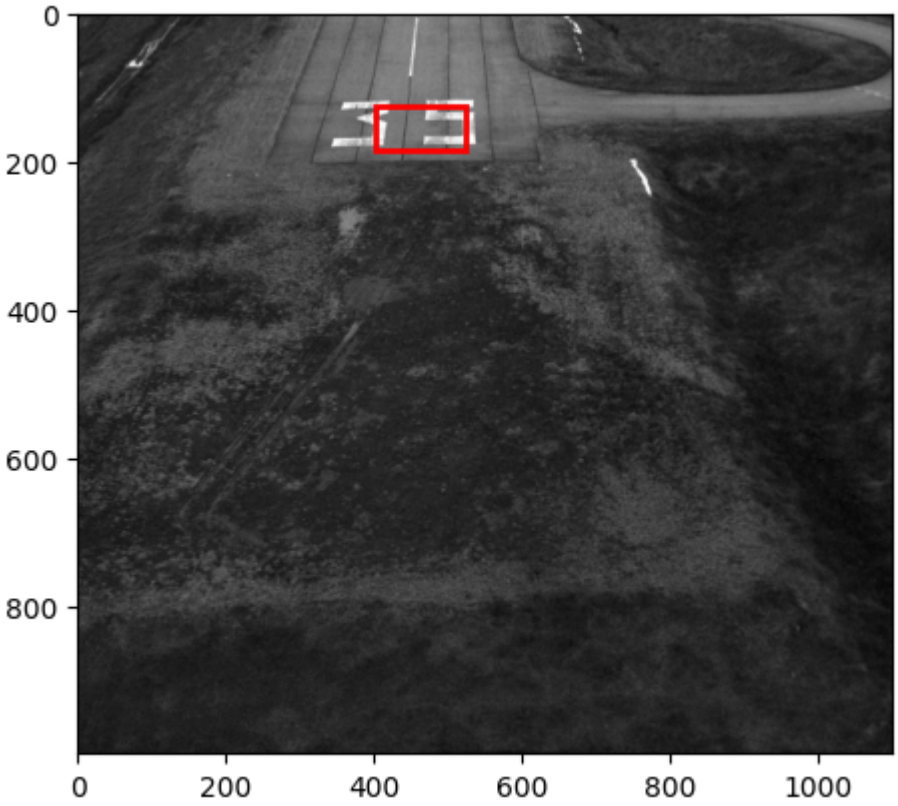
Results for landing.npy

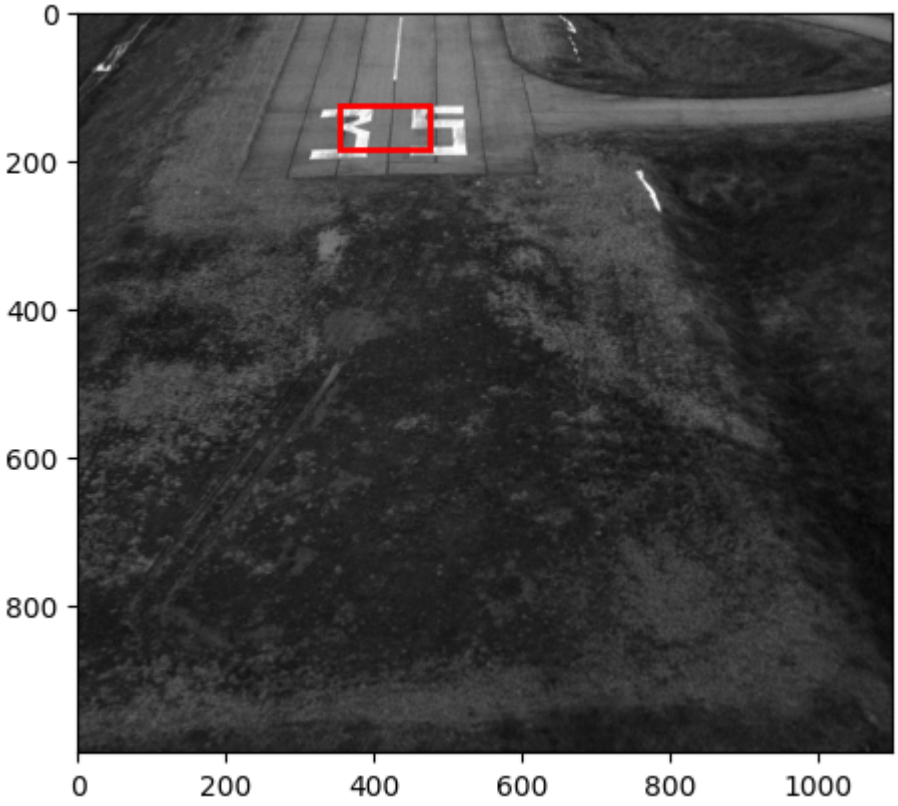



0
200
400
600
800
0
200
400
600
800
1000

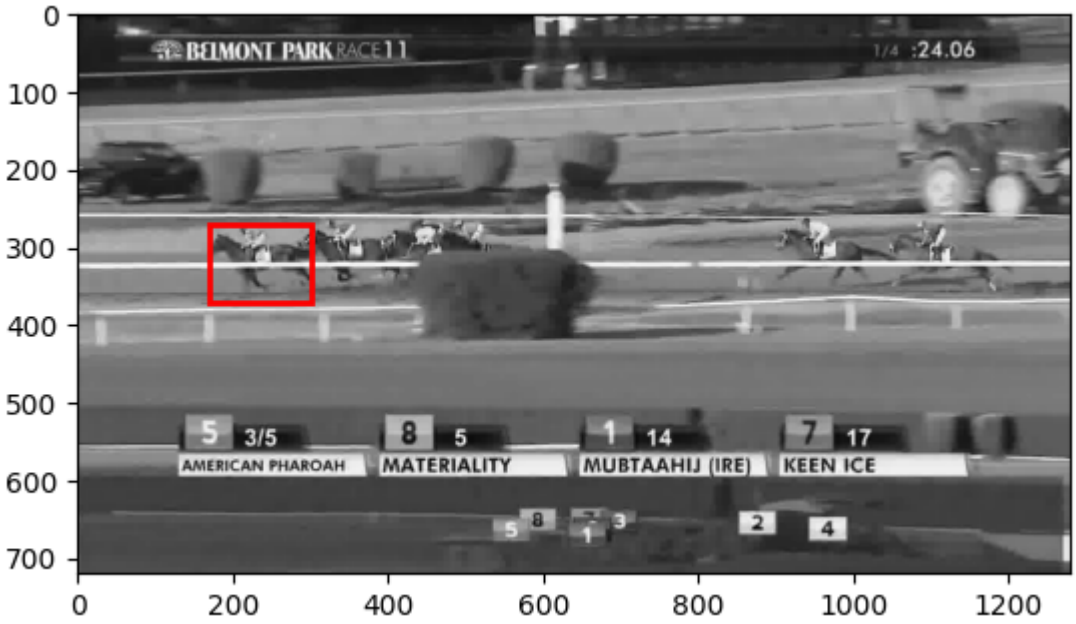

0
200
400
600
800
0
200
400
600
800
1000

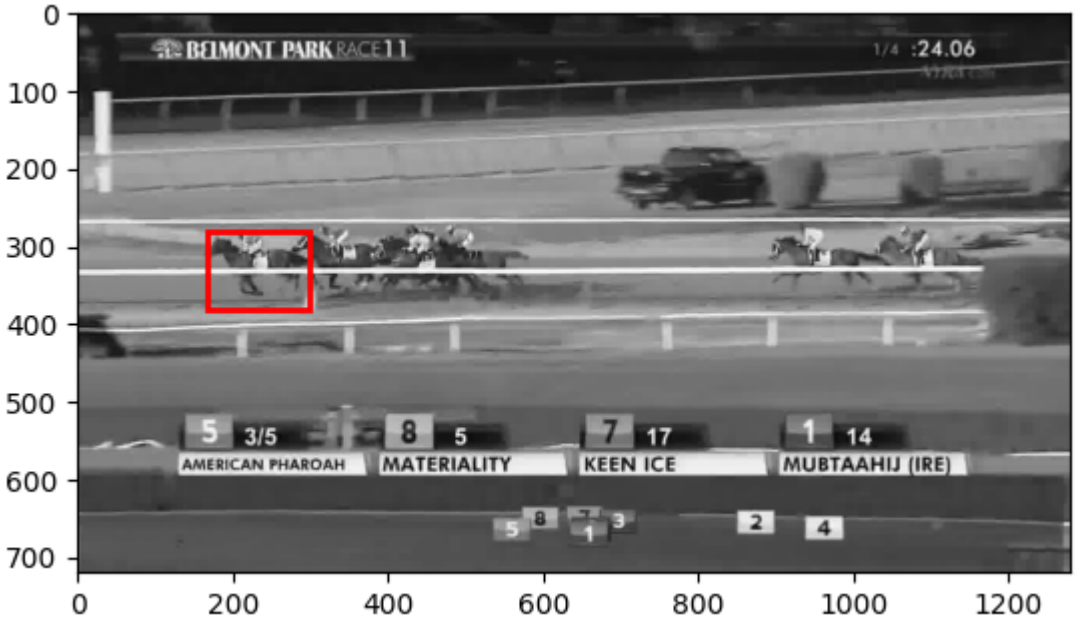

0
200
400
600
800
0
200
400
600
800
1000

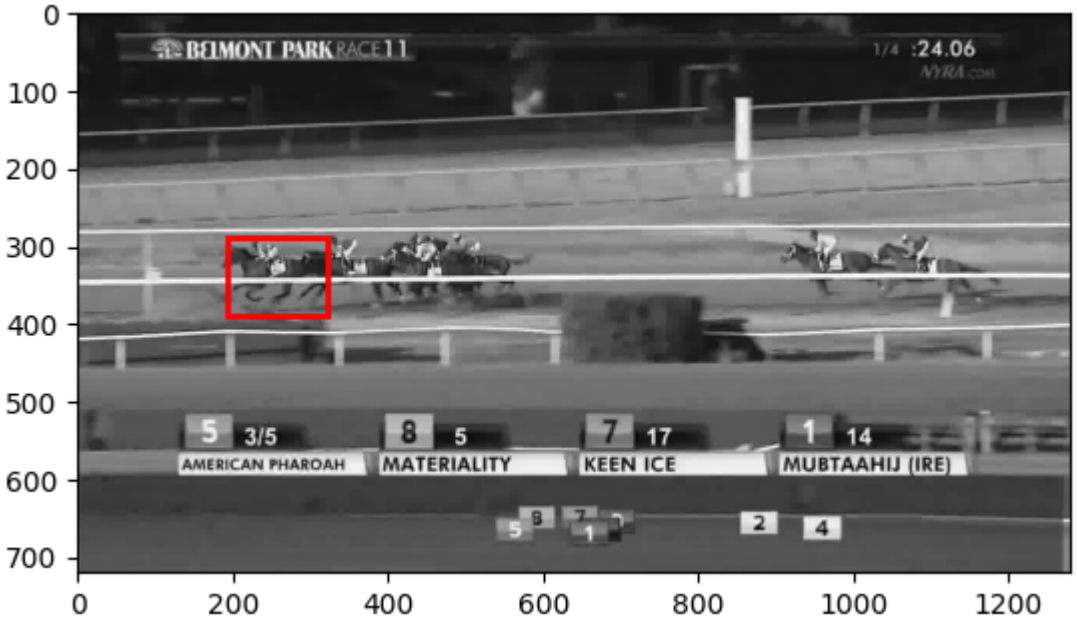

0
200
400
600
800
0
200
400
600
800
1000

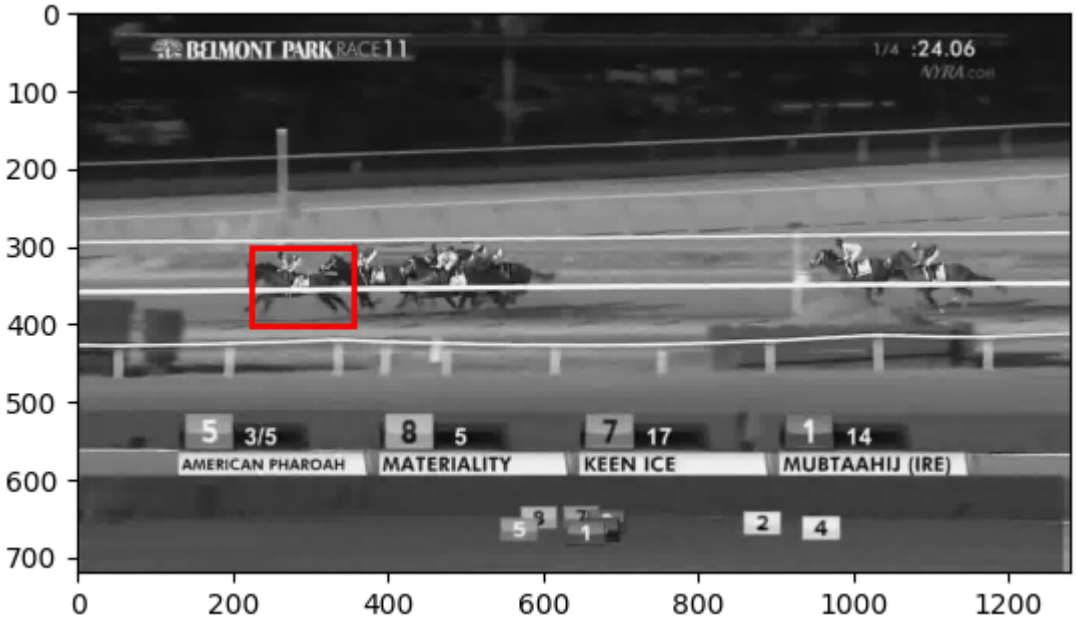


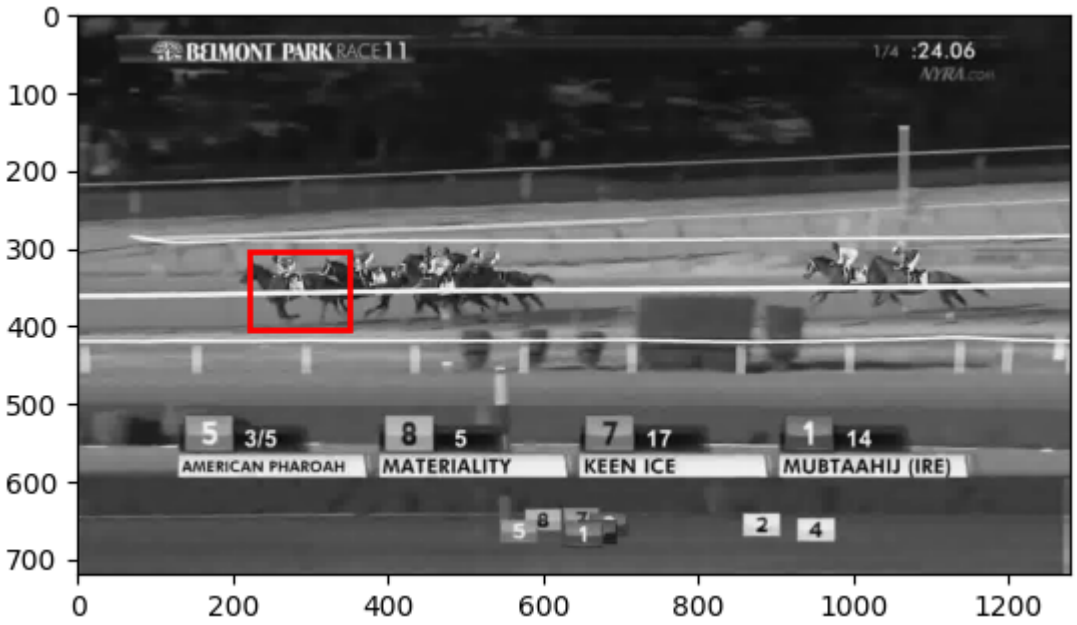
Results for race.npy

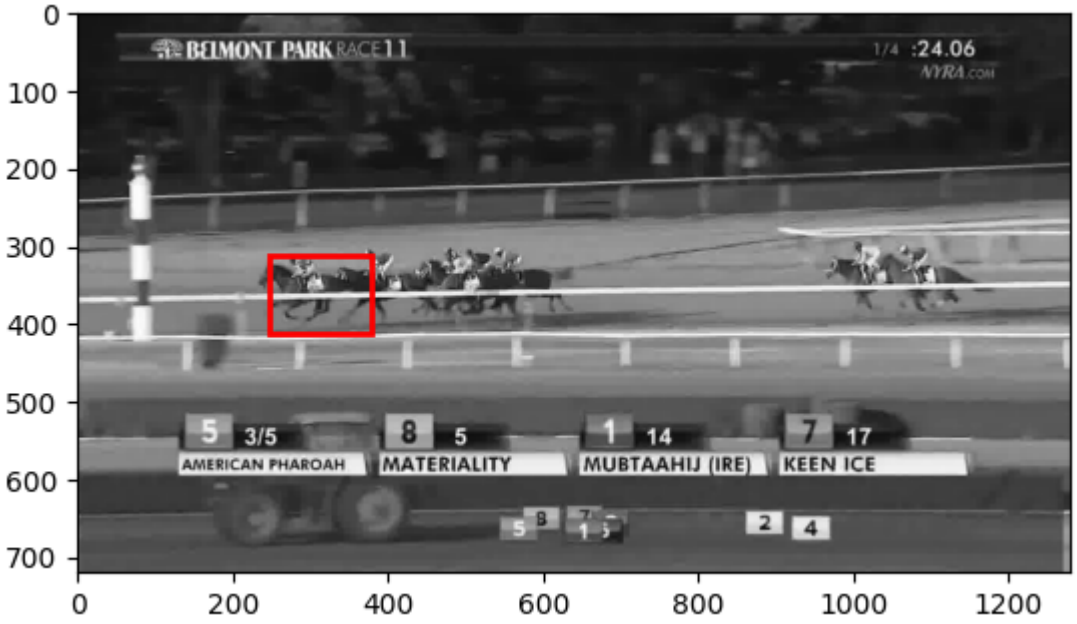


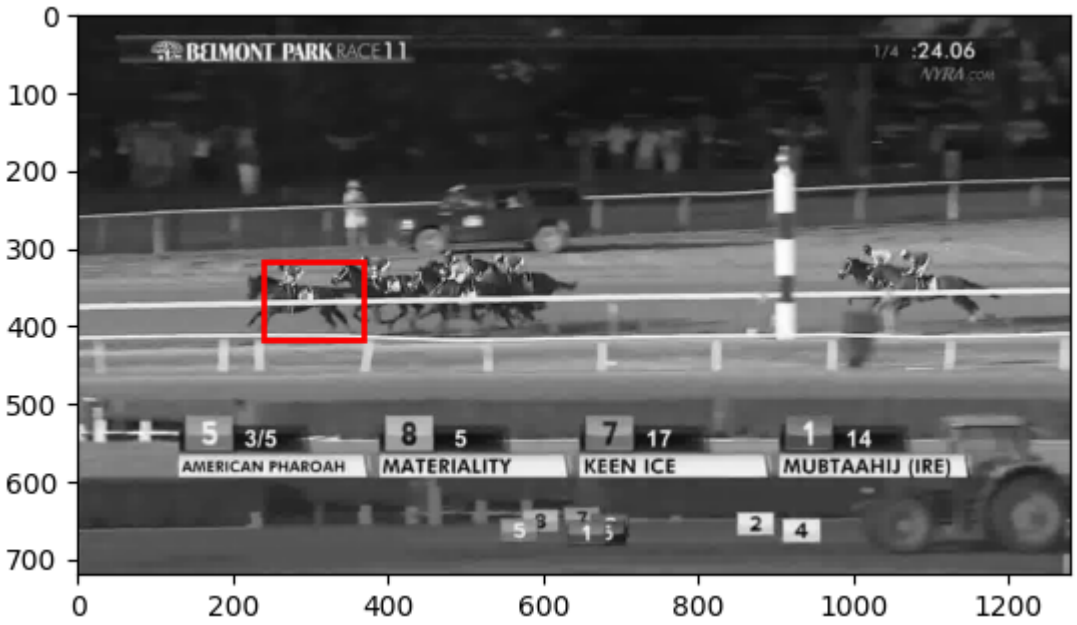

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200





Code:

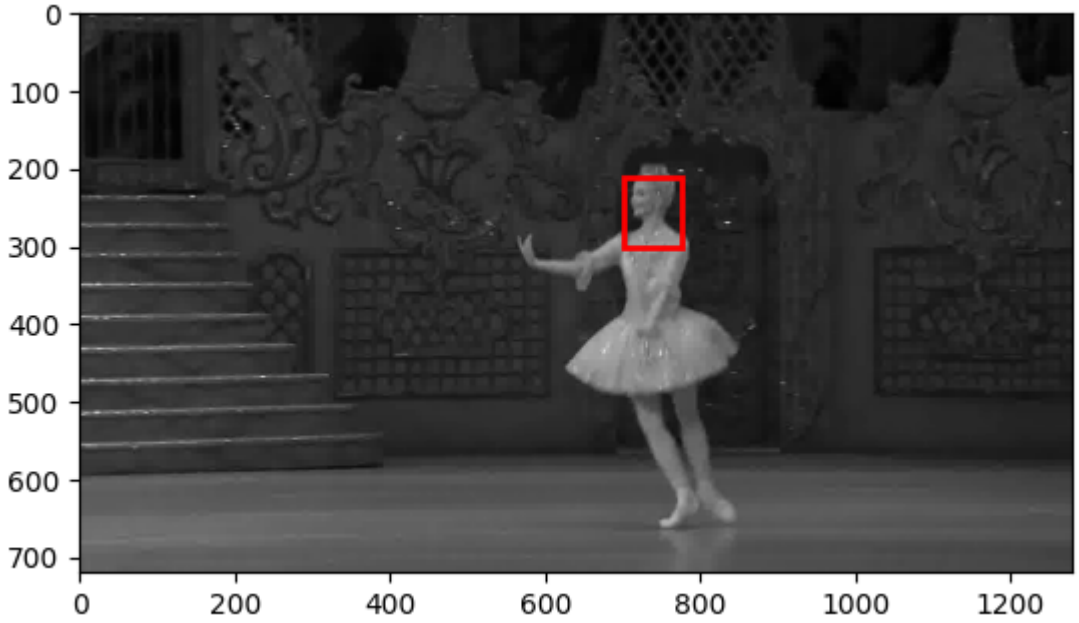
In [ ]:
```
def LucasKanade(It, It1, rect, thresh=.025, maxIt=100):

    '''
    Q1.1: Lucas-Kanade Forward Additive Alignment with Translation Only

      Inputs:
        It: template image
        It1: Current image
        rect: Current position of the object
        (top left, bottom right coordinates, x1, y1, x2, y2)
        thresh: Stop condition when dp is too small
        maxIt: Maximum number of iterations to run
```

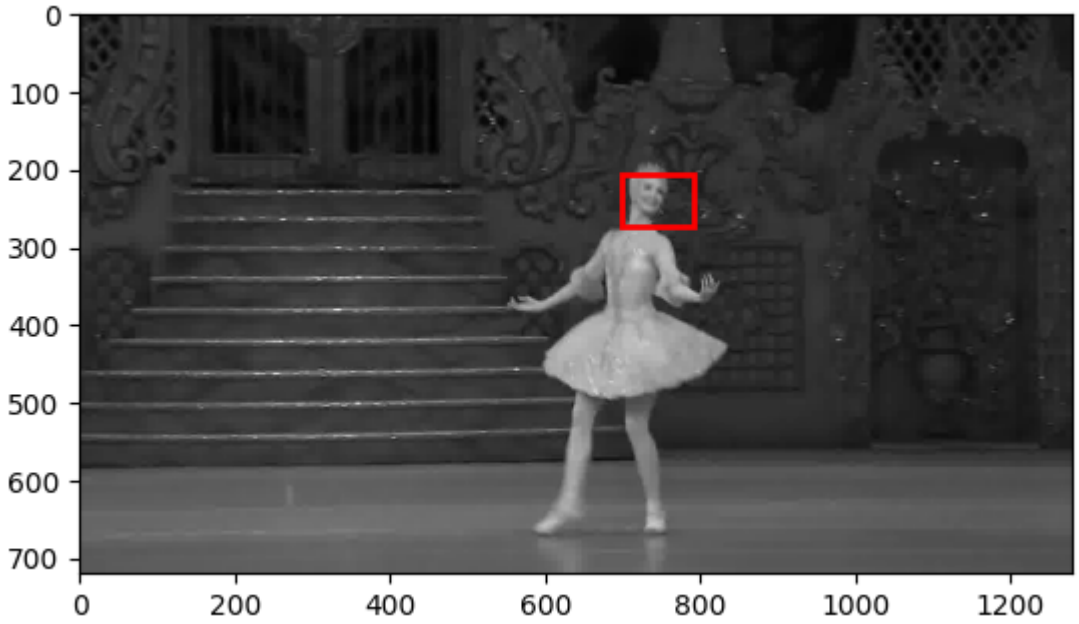
```
    Outputs:
      p: movement vector dx, dy
  '''
  # print(It.shape)
  # print(It1.shape)
  # Set thresholds (you probably want to play around with the values)
  p = np.zeros(2) # dx, dy
  threshold = thresh
  maxIters = maxIt
  i = 0
  x1, y1, x2, y2 = rect

  # ----- TODO -----
  # YOUR CODE HERE
  # del_Px, del_Py = 1, 1
  del_p = np.ones(2,)
  img_t_spline = RectBivariateSpline(np.arange(It.shape[0]), np.arange(It.sha
  mesh_x, mesh_y = np.meshgrid(np.arange(x1,x2), np.arange(y1,y2))
  wrapped_img_t_spline = (img_t_spline.ev(mesh_y ,mesh_x)).flatten()

  img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arange(It1
  grady_img_t1, gradx_img_t1 = np.gradient(It1)
  gradx_img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arang
  grady_img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arang

  curr_itr = 0
  while np.linalg.norm(del_p) > threshold and curr_itr < maxIters:
  # while curr_itr < maxIters:

    del_Px = p[0]
    del_Py = p[1]
    mesh_x_temp = mesh_x + del_Px
    mesh_y_temp = mesh_y + del_Py

    wrapped_img_t1_spline = (img_t1_spline.ev(mesh_y_temp, mesh_x_temp)).fla

    b = wrapped_img_t_spline - wrapped_img_t1_spline

    wrapped_gradx = gradx_img_t1_spline.ev(mesh_y_temp, mesh_x_temp).flatten
    wrapped_grady = grady_img_t1_spline.ev(mesh_y_temp, mesh_x_temp).flatten

    # A = np.column_stack((wrapped_gradx, wrapped_grady))
    A = np.stack((wrapped_gradx, wrapped_grady), axis = 1)
    # A = np.vstack((wrapped_gradx, wrapped_grady))
    # A = A.reshape((2,1))

    del_p = np.linalg.lstsq(A, b, rcond=-1)[0]
    # del_p = del_p[0]
    # print(del_p.shape)
    # if np.linalg.norm(del_p) > threshold:
    #   break
    # else:
    p += del_p
    curr_itr += 1
```

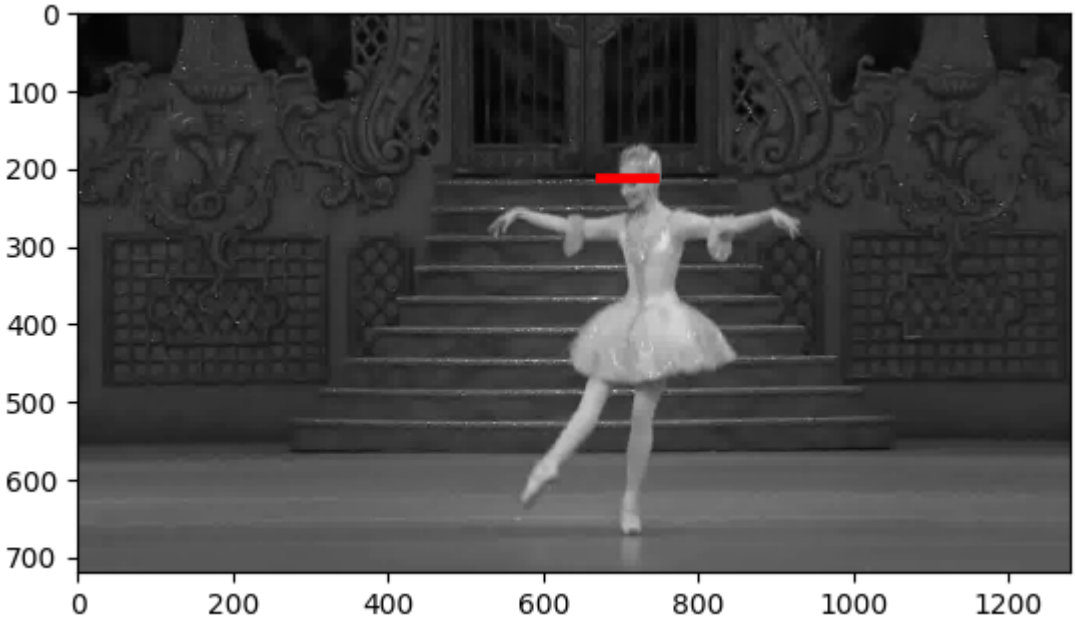
```
    # raise NotImplementedError()

    return p
```

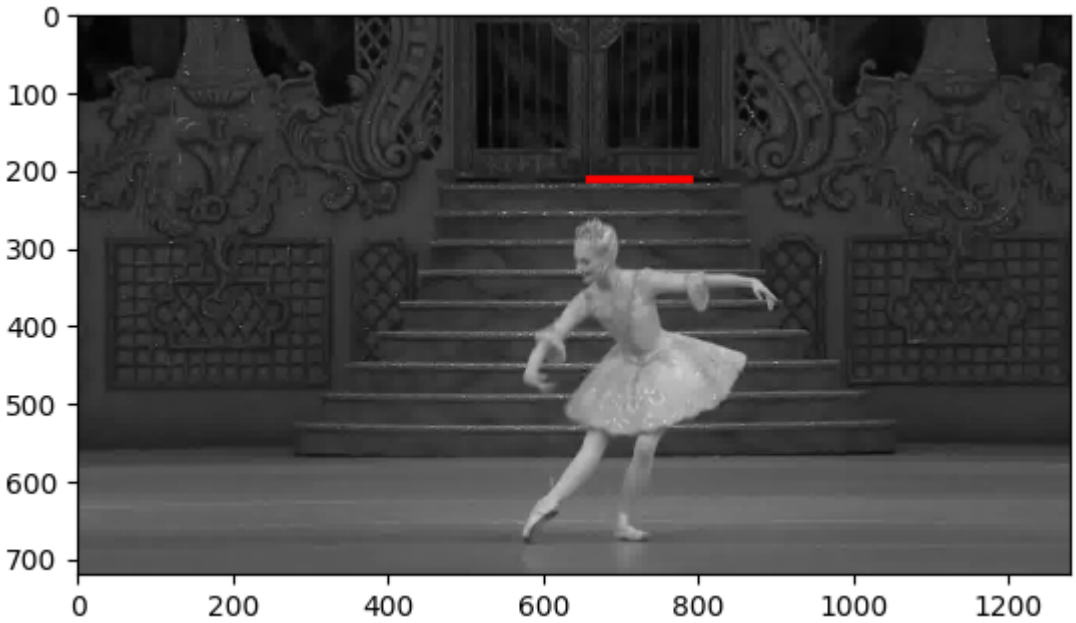
Q1.2

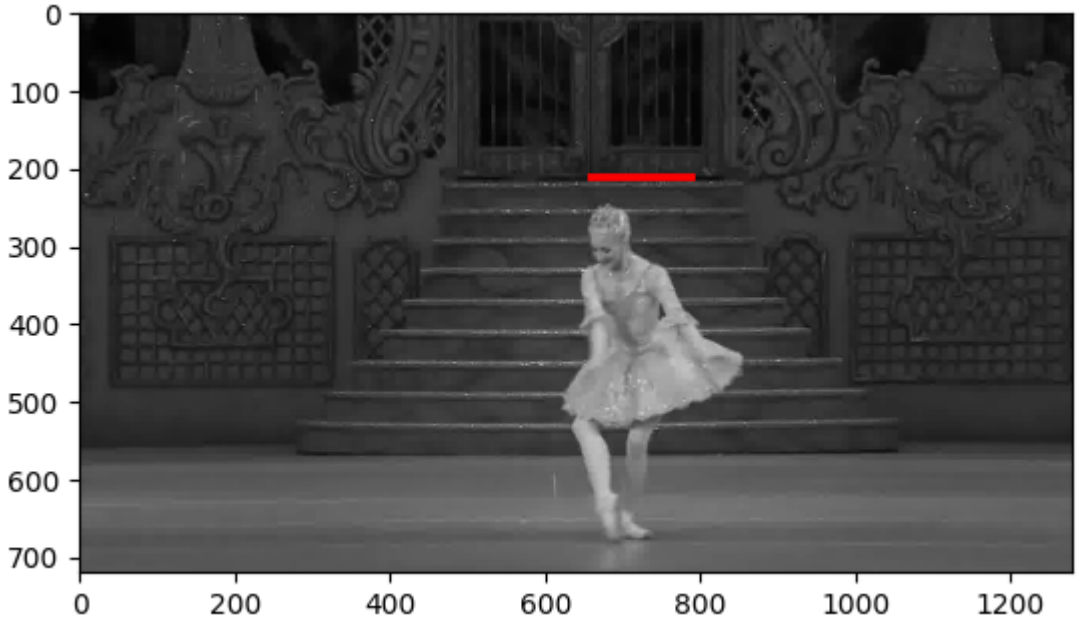
YOUR ANSWER HERE Results for ballet.npy:



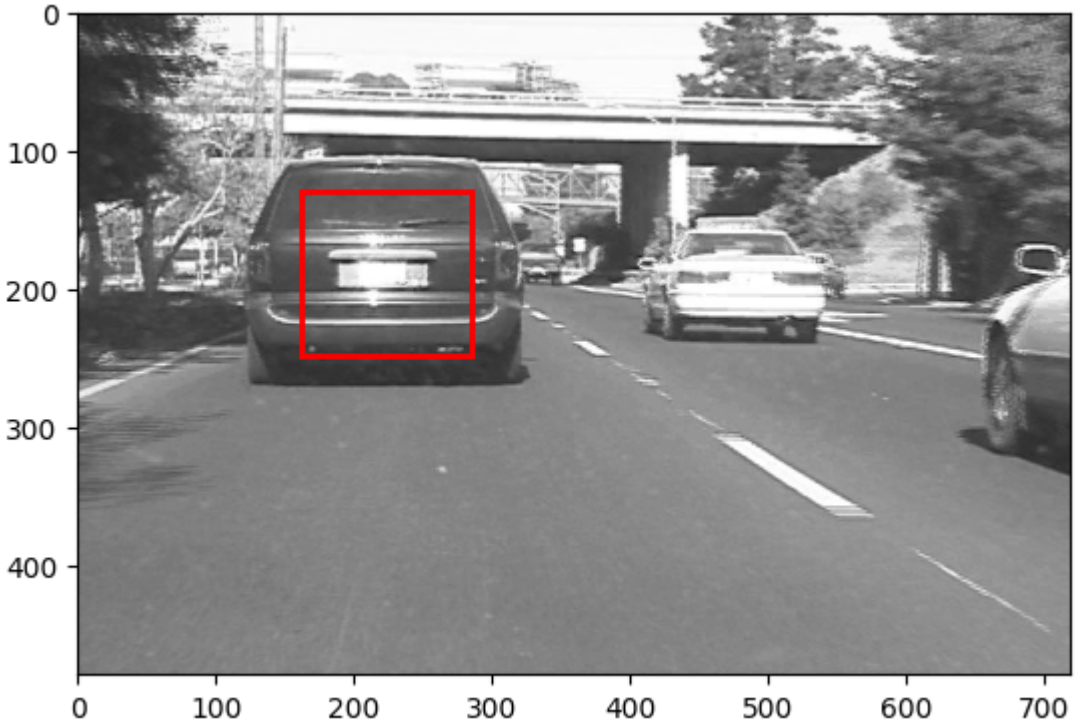


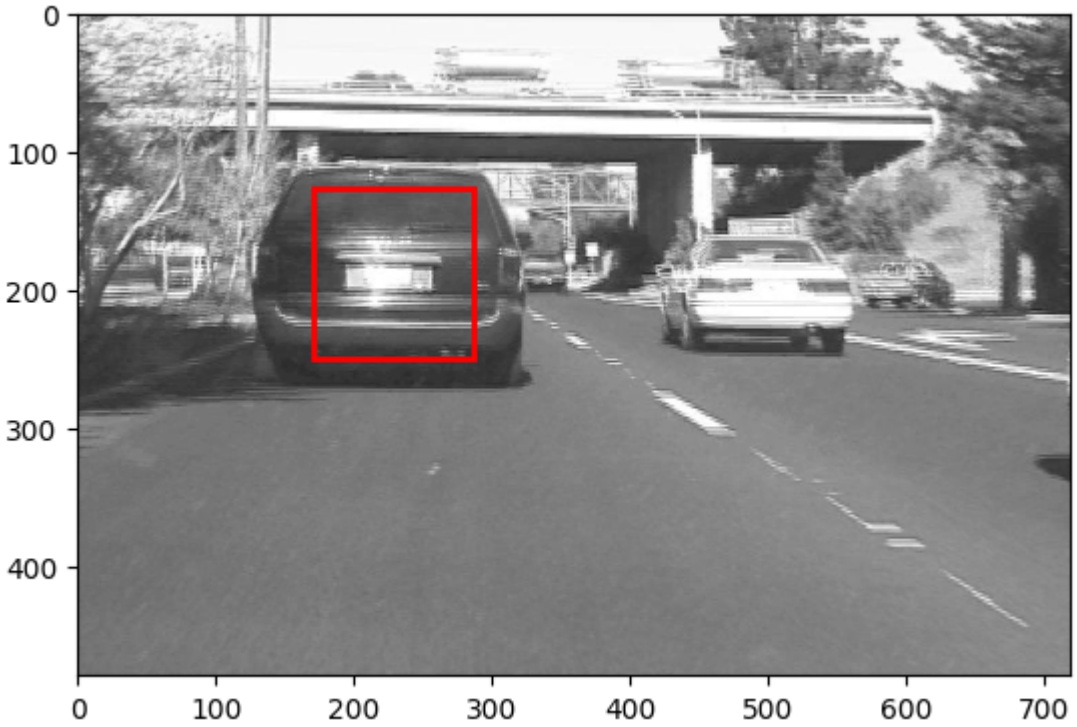

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



Results for car1.npy:


0
100
200
300
400
0
100
200
300
400
500
600
700

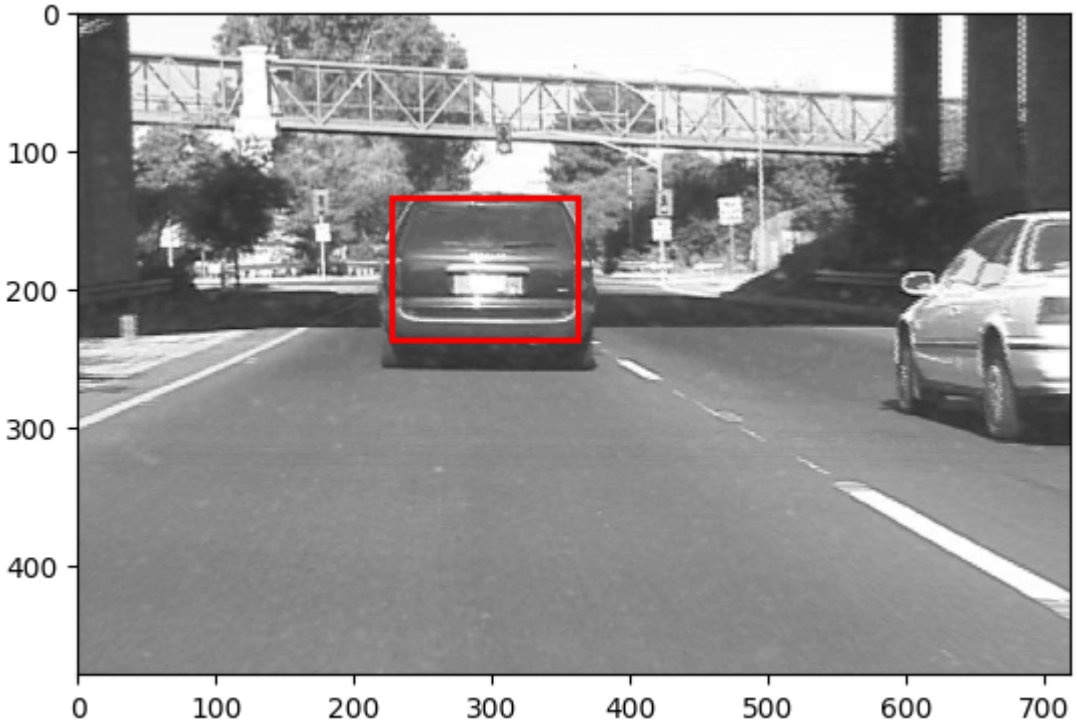

0
100
200
300
400
0
100
200
300
400
500
600
700

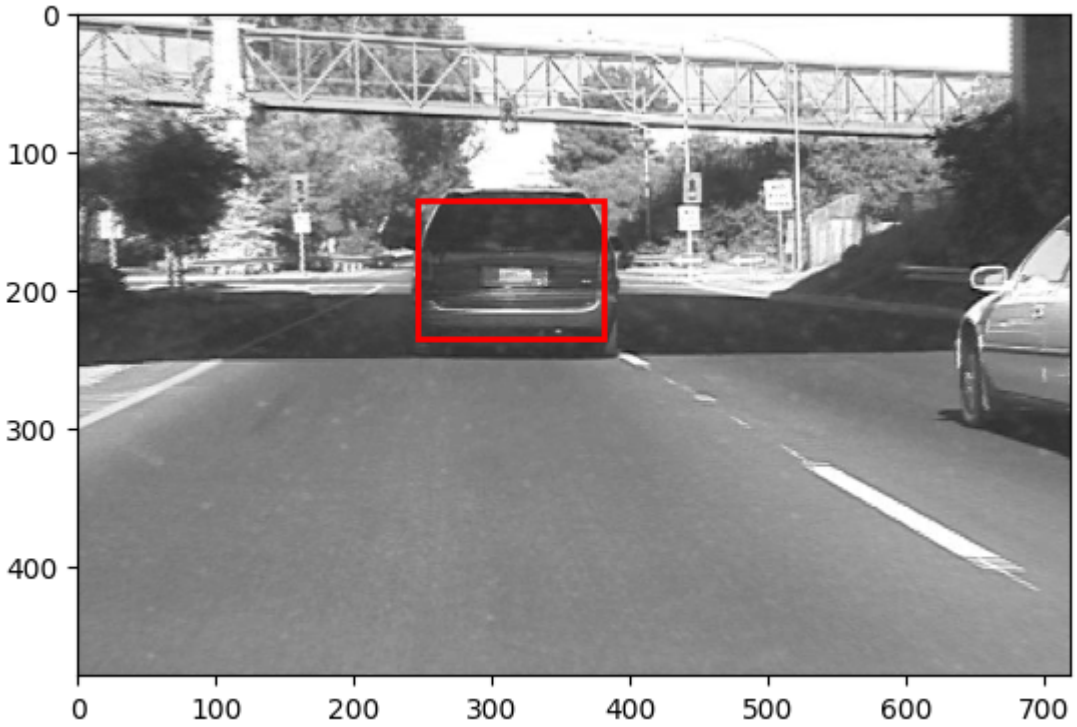

0
100
200
300
400
0
100
200
300
400
500
600
700

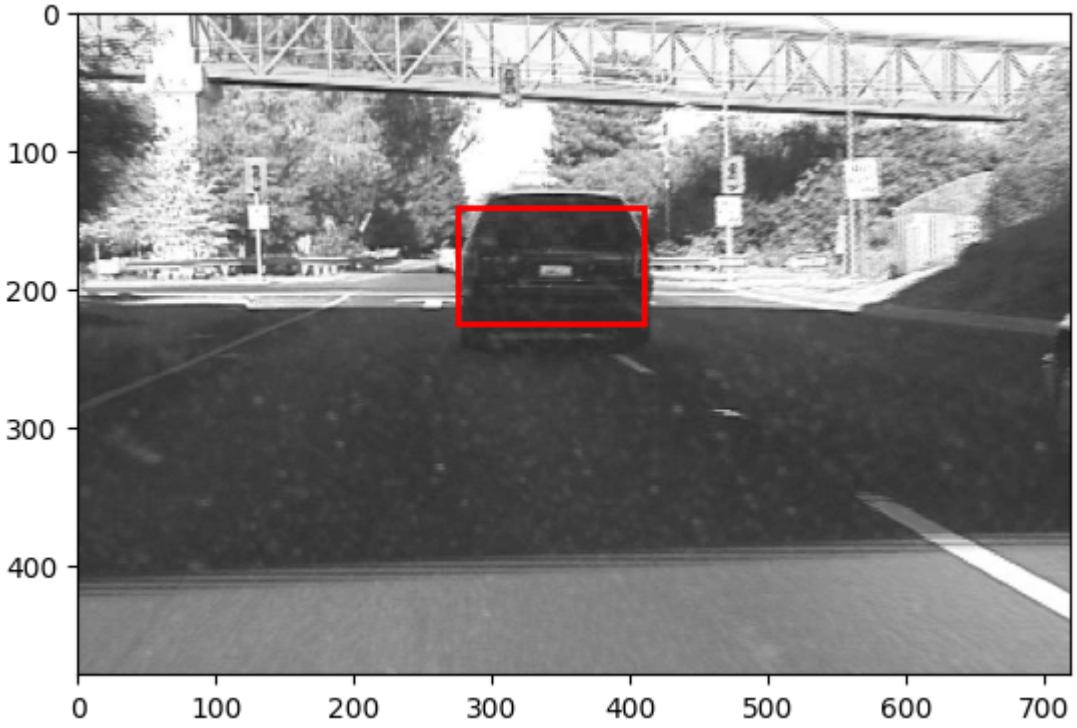

0
100
200
300
400
0
100
200
300
400
500
600
700

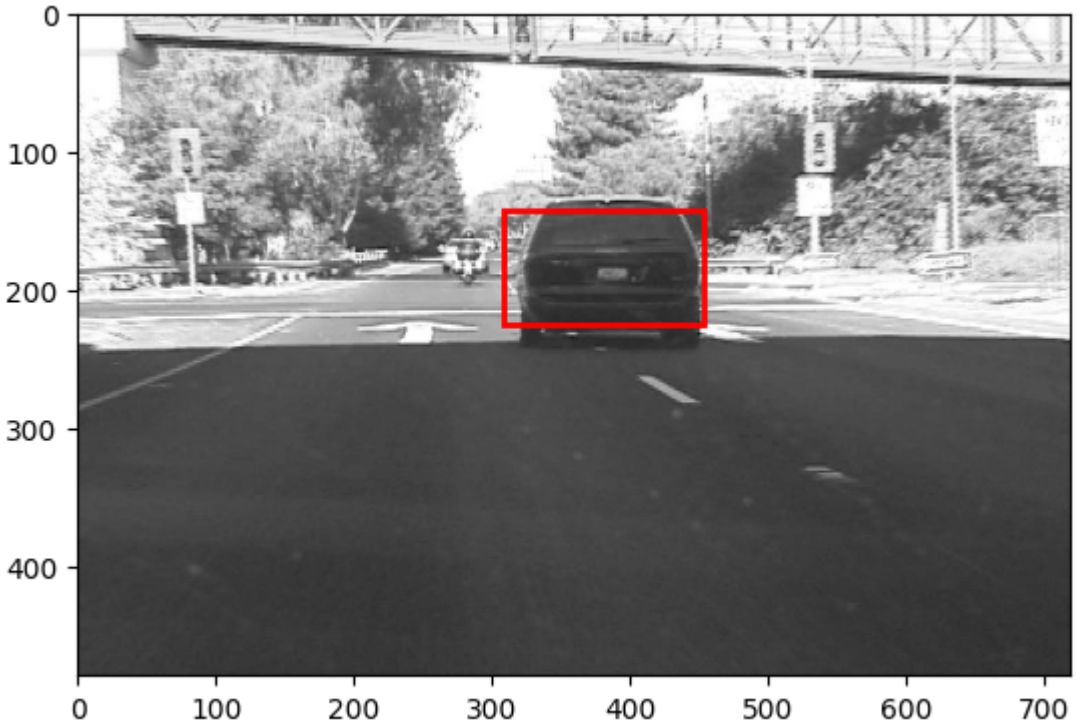

0
100
200
300
400
0
100
200
300
400
500
600
700

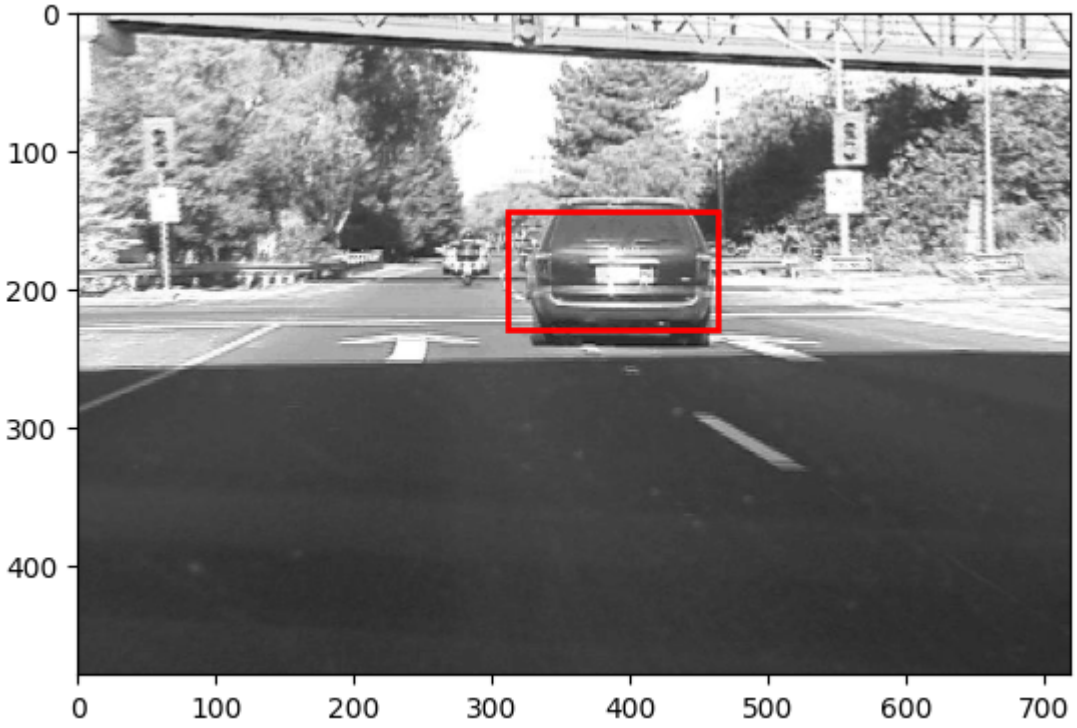

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

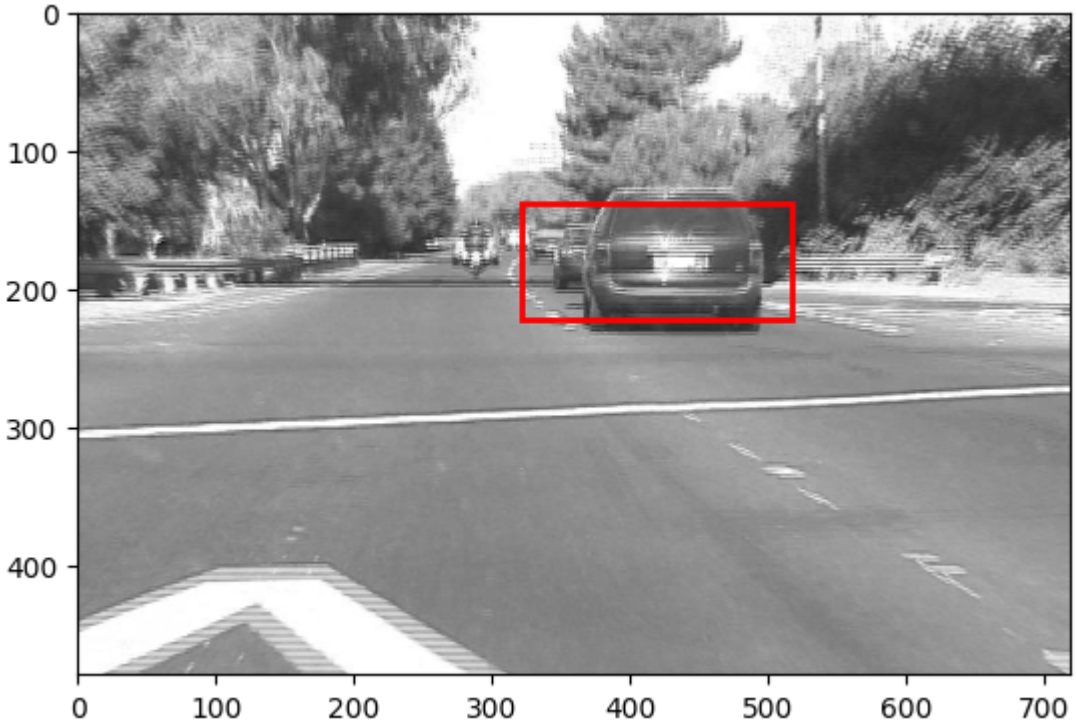

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

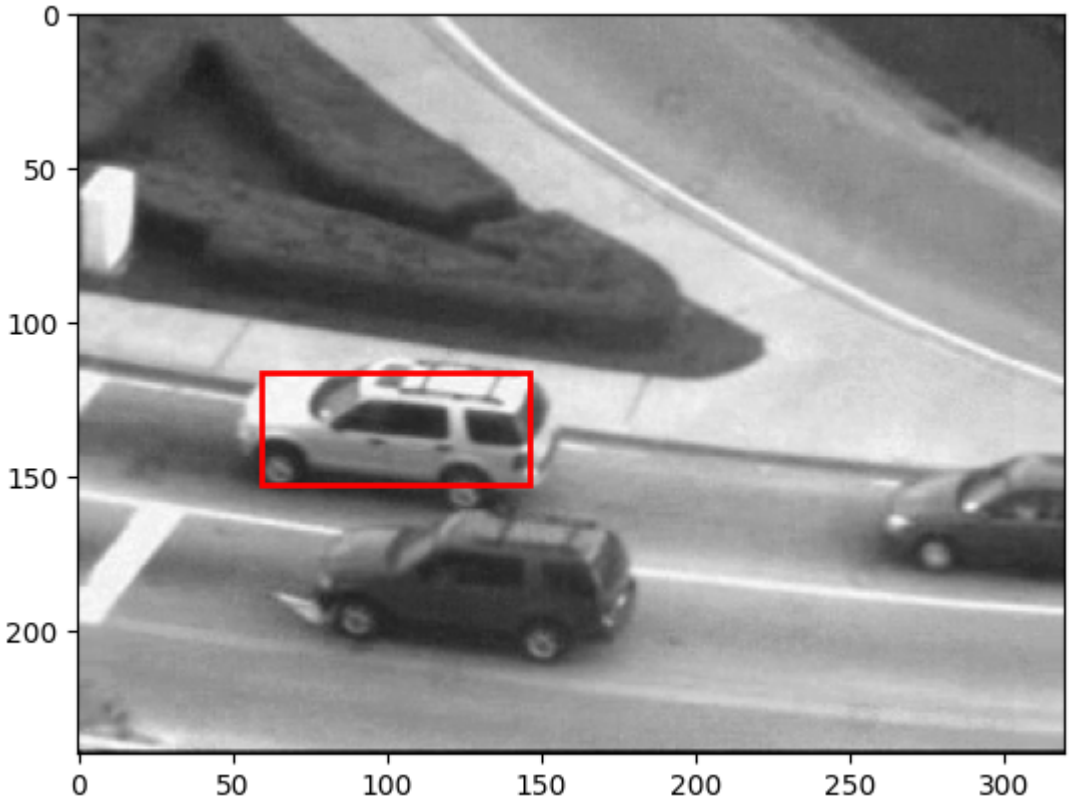

0
100
200
300
400
0
100
200
300
400
500
600
700

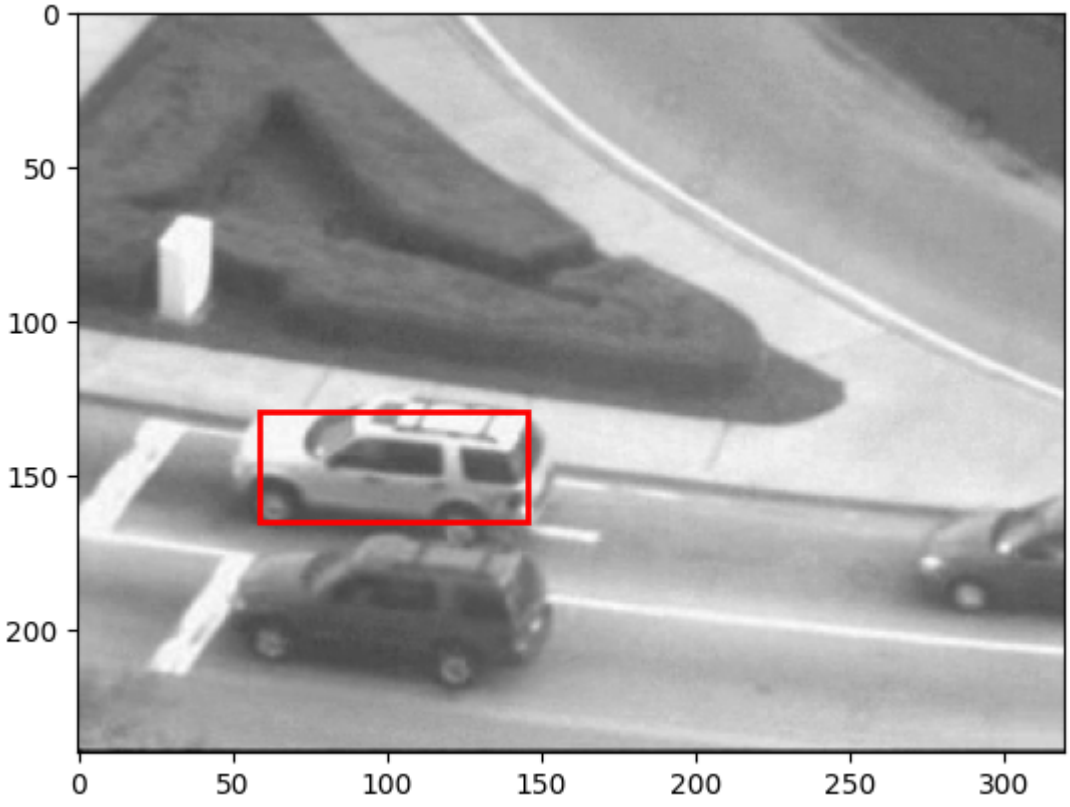

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

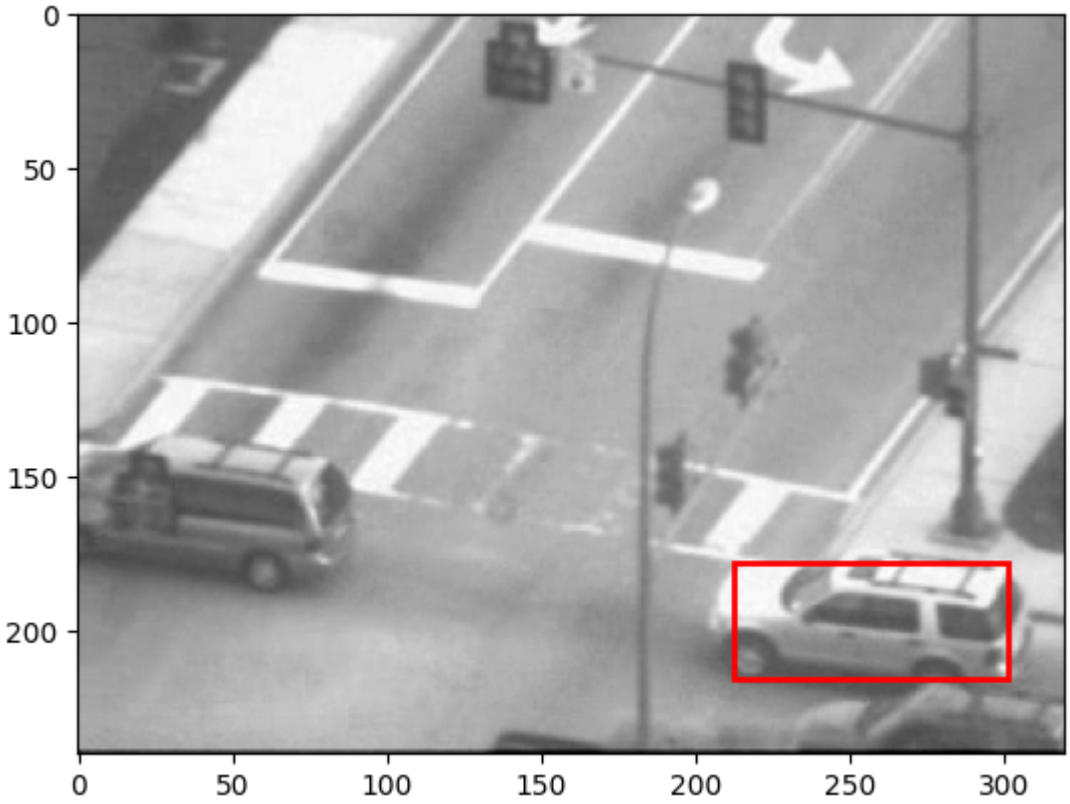

0
100
200
300
400
0
100
200
300
400
500
600
700

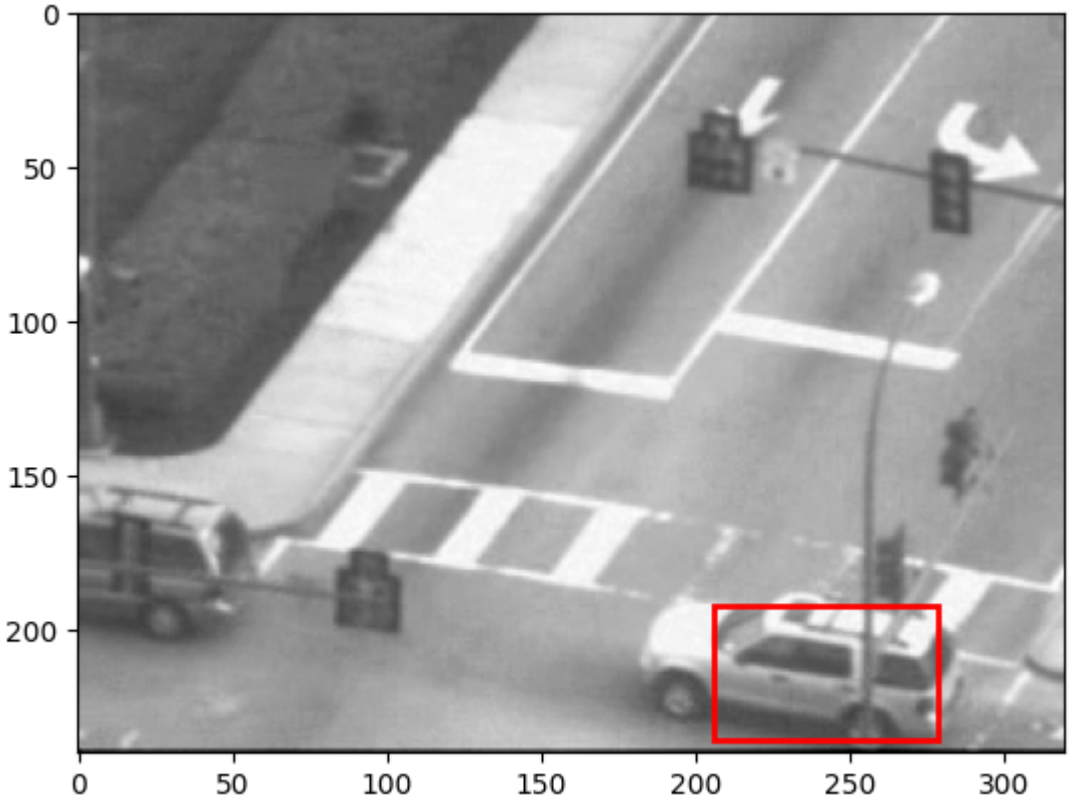
Results for car2.npy

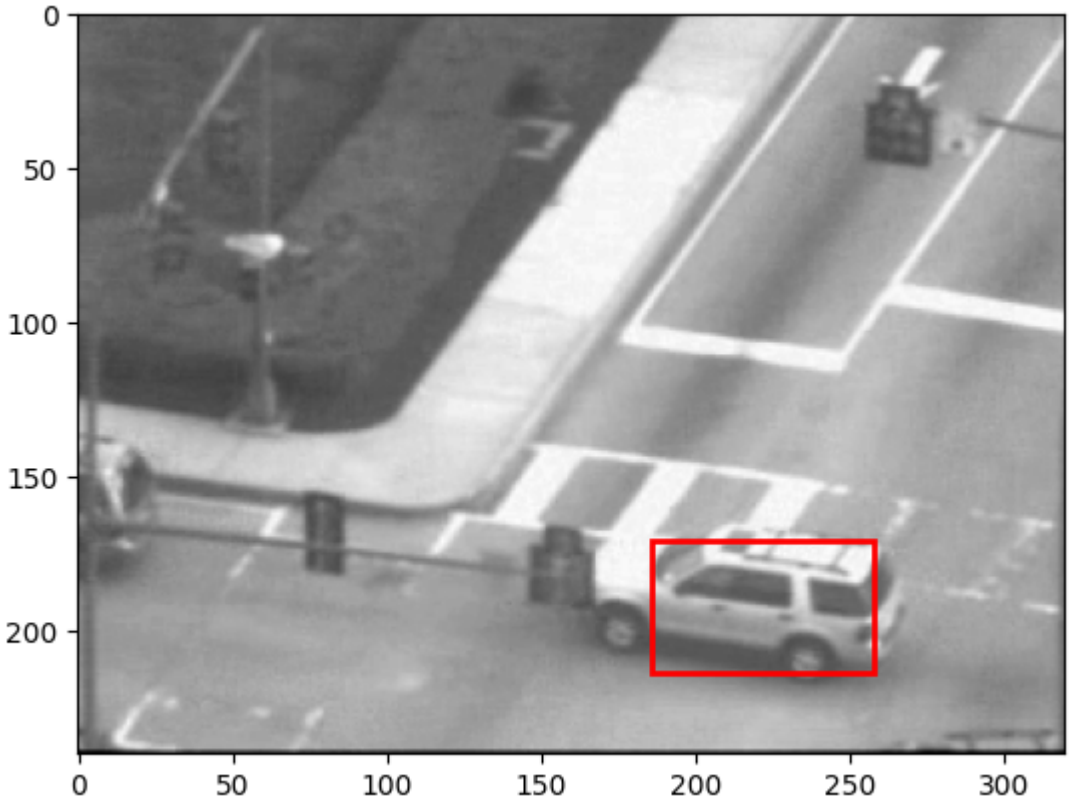


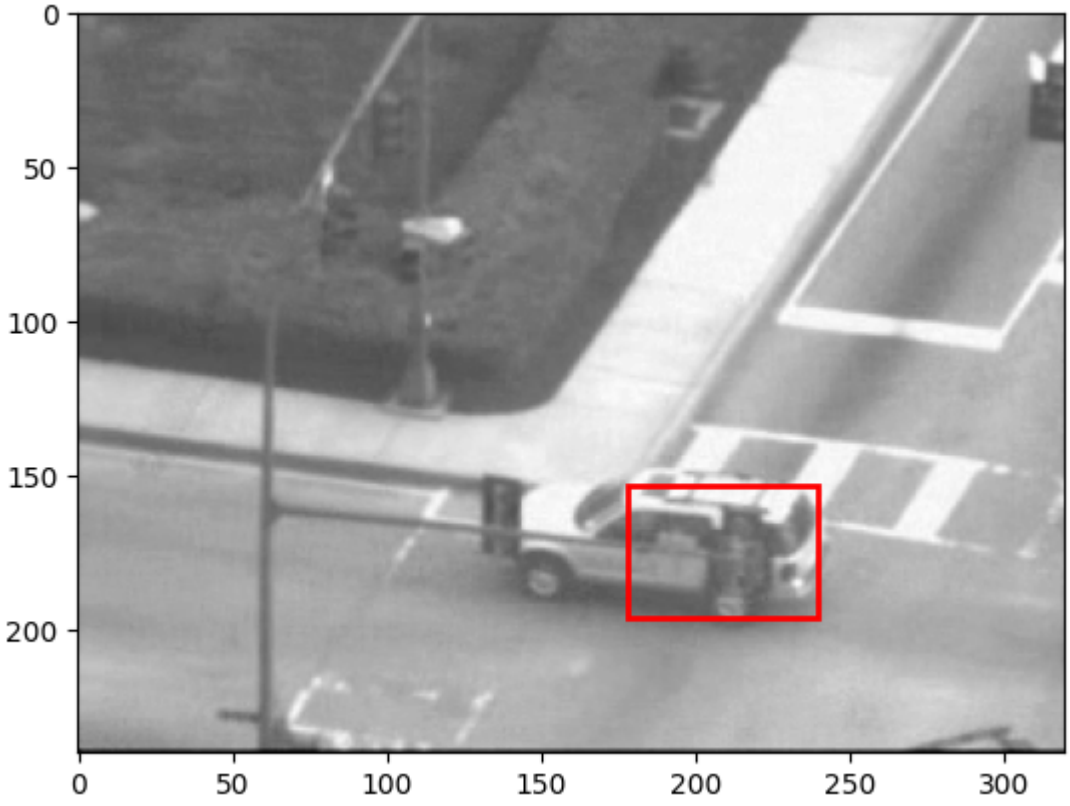


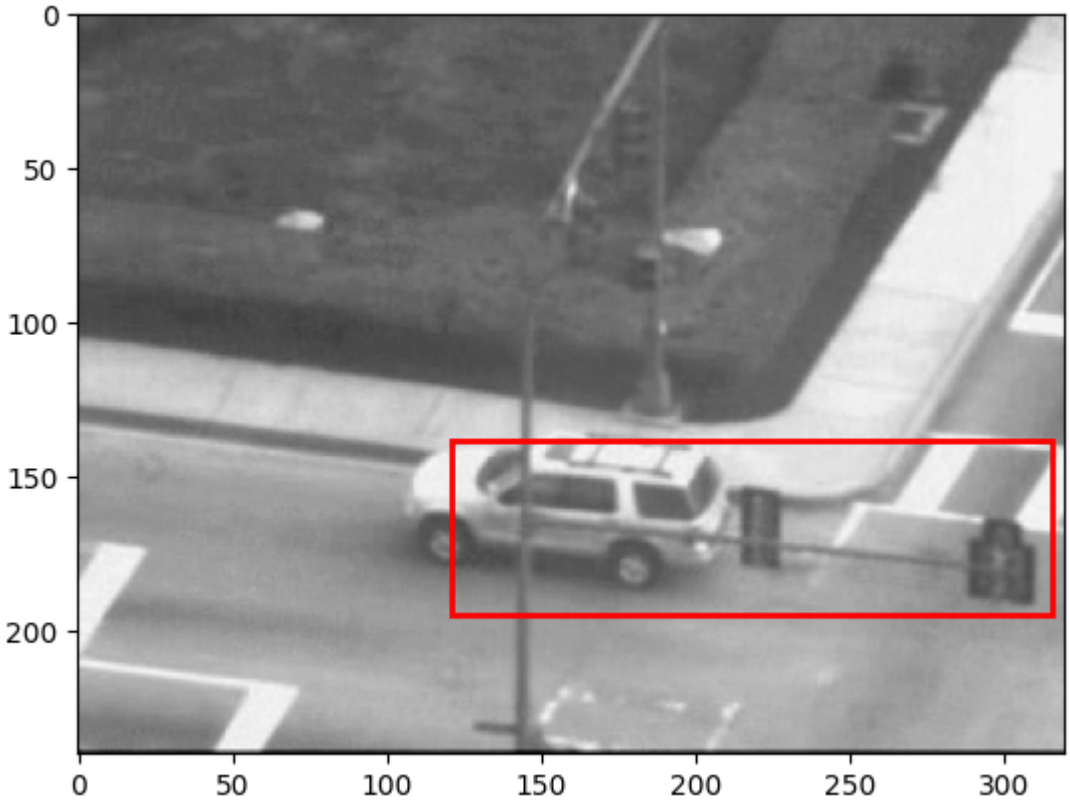

0
50
100
150
200
0
50
100
150
200
250
300

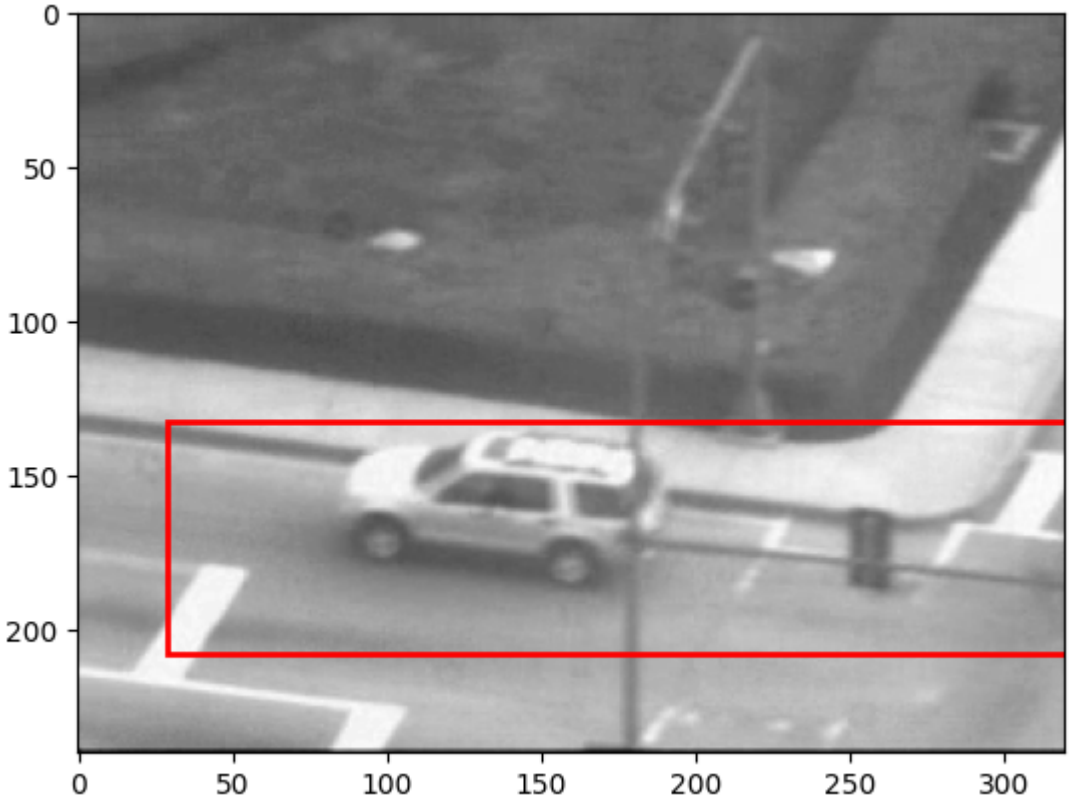

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

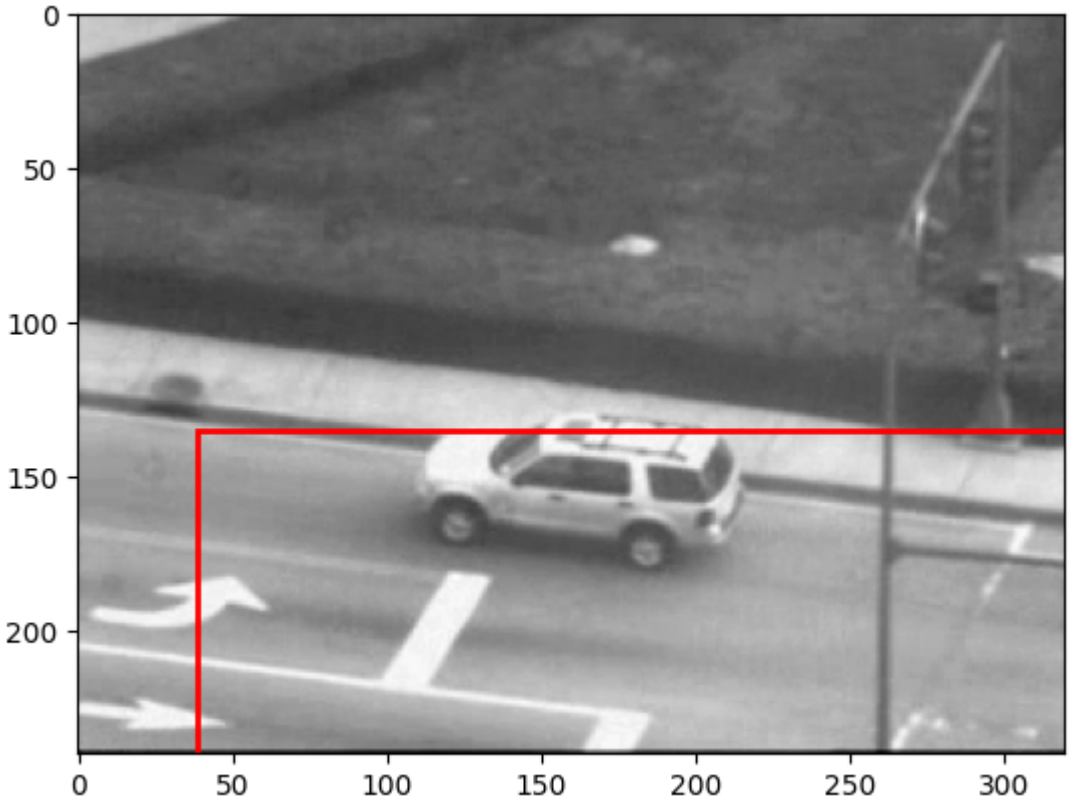

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

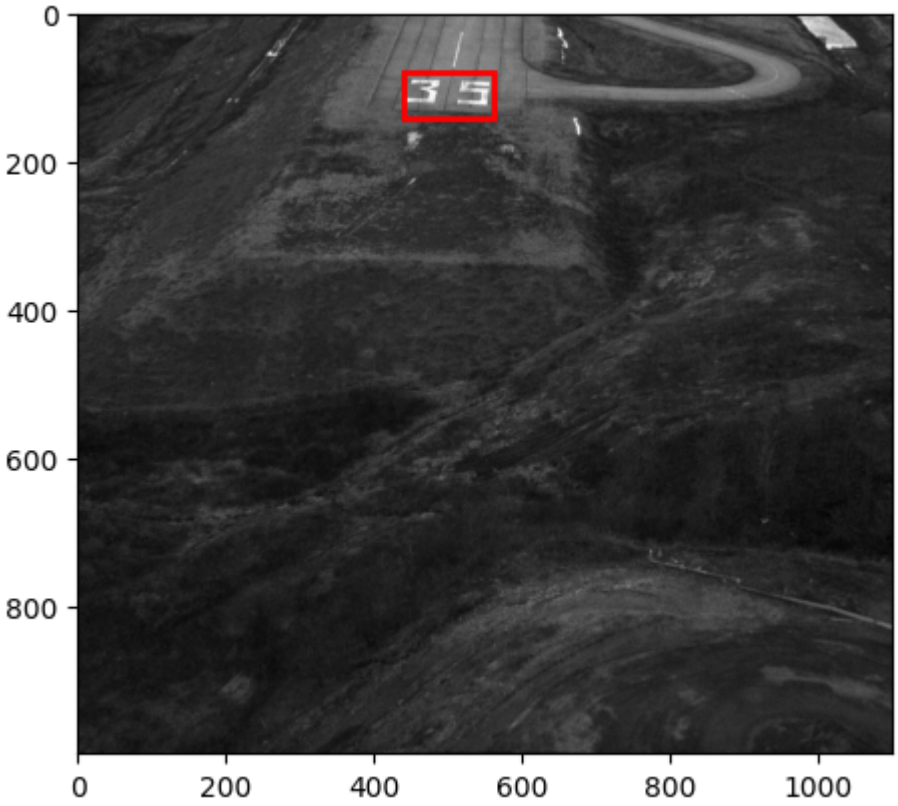

0
50
100
150
200
0
50
100
150
200
250
300

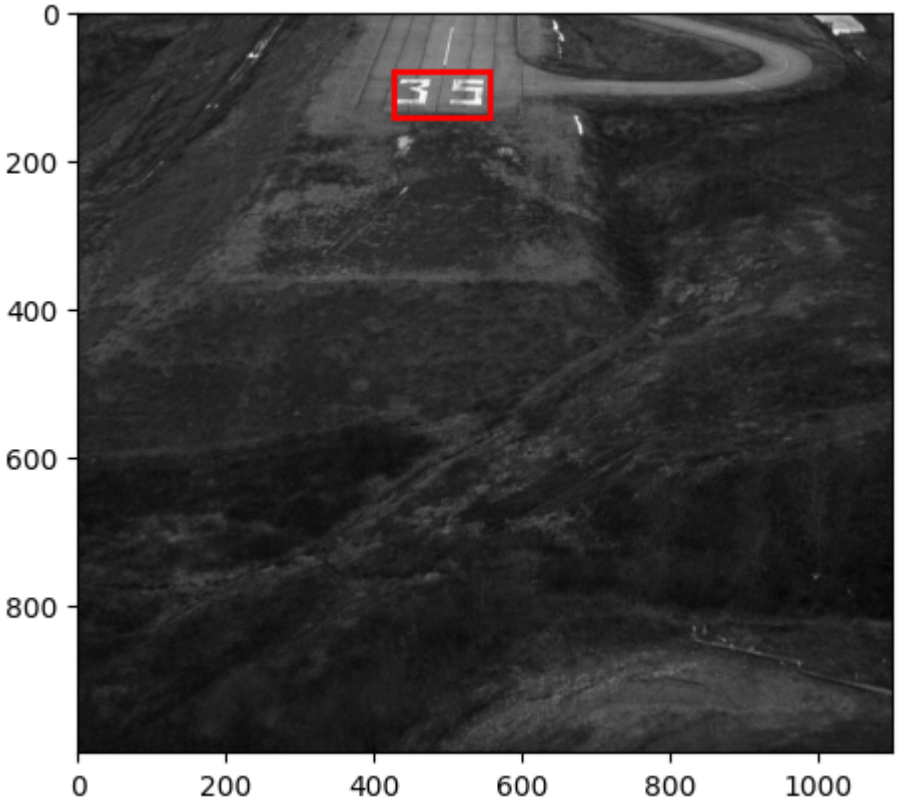

0
50
100
150
200
0
50
100
150
200
250
300

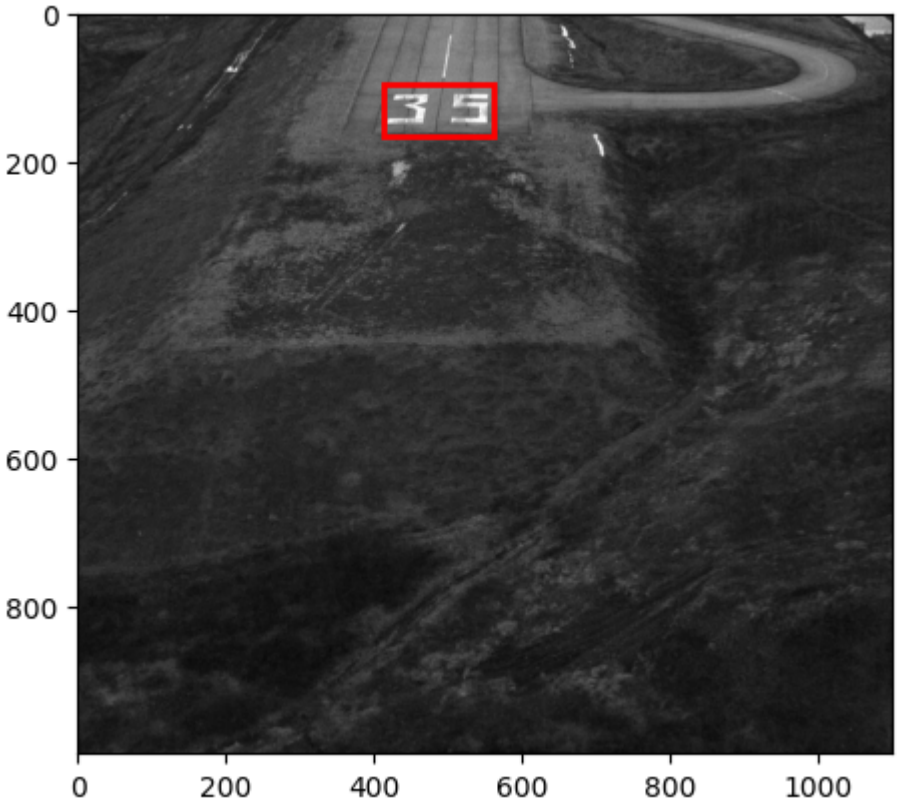

0
50
100
150
200
0
50
100
150
200
250
300

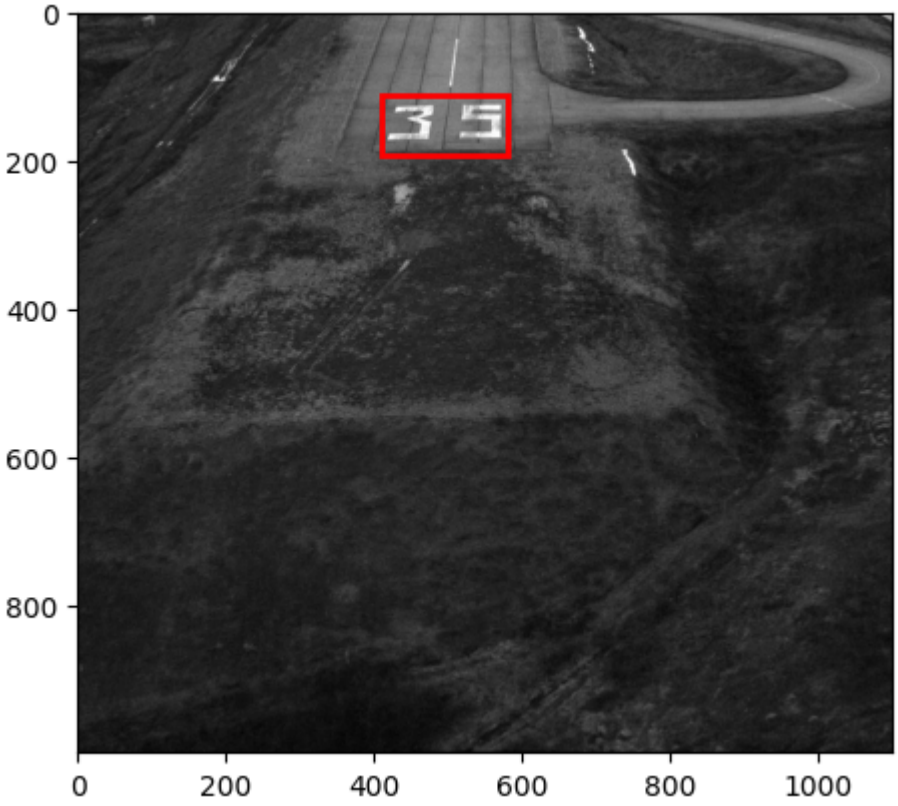

0
50
100
150
200
0
50
100
150
200
250
300

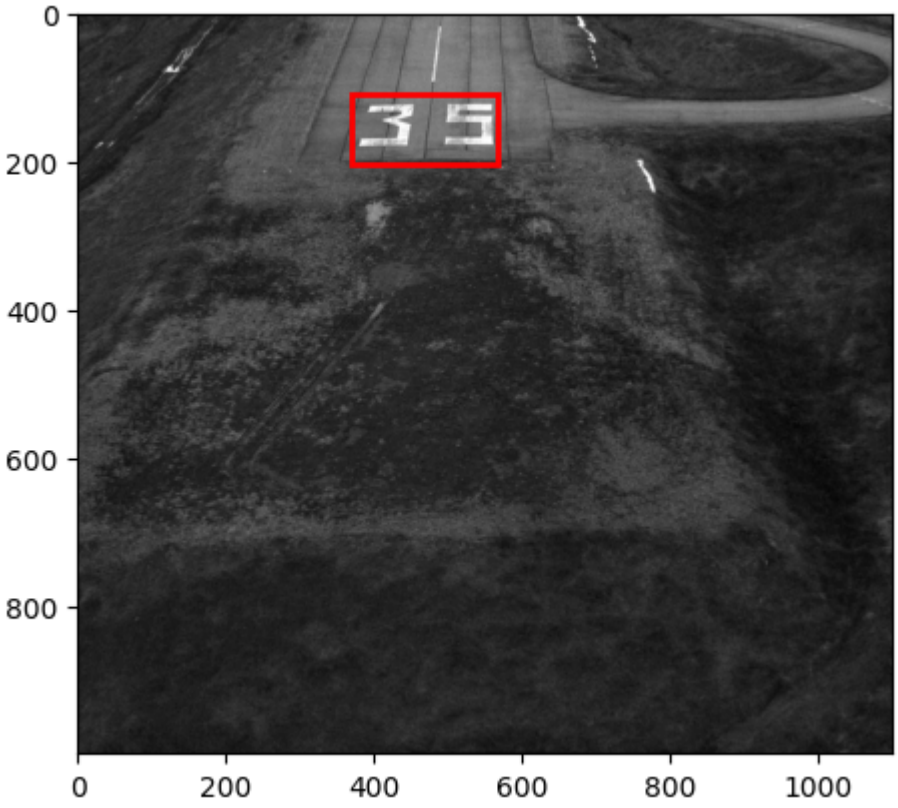
Results for landing.npy

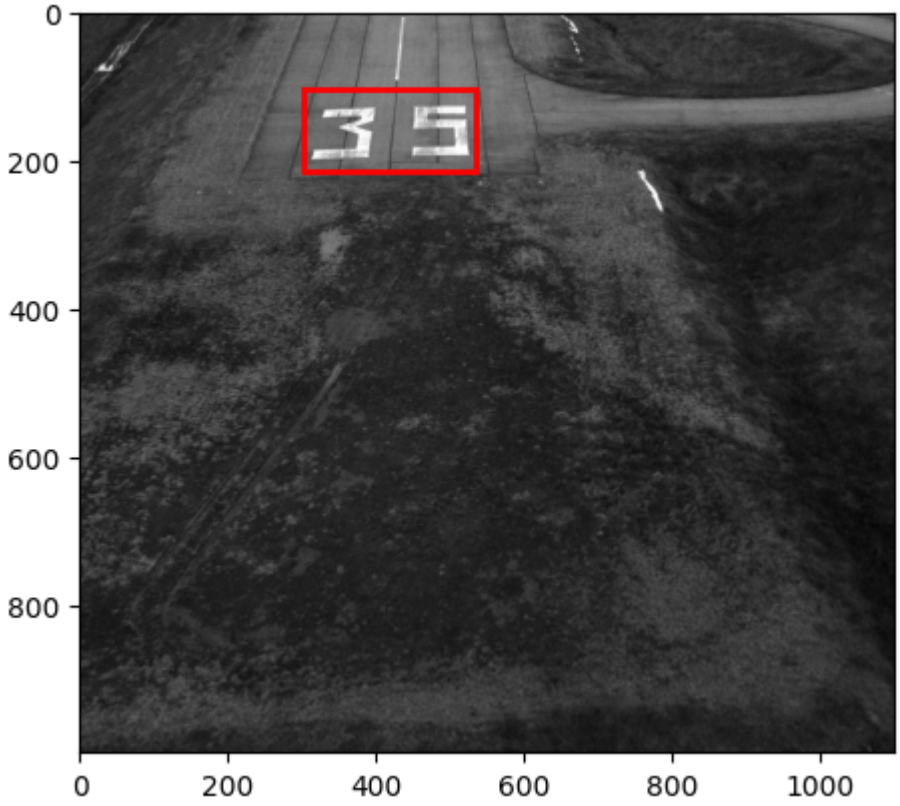



0
200
400
600
800
0
200
400
600
800
1000

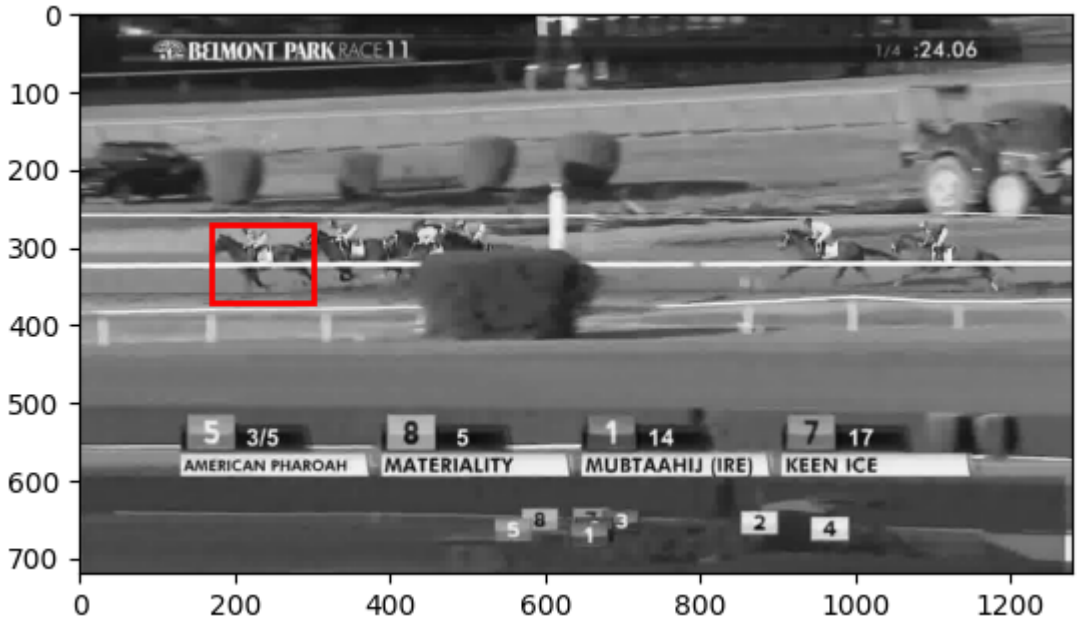

0
200
400
600
800
0
200
400
600
800
1000

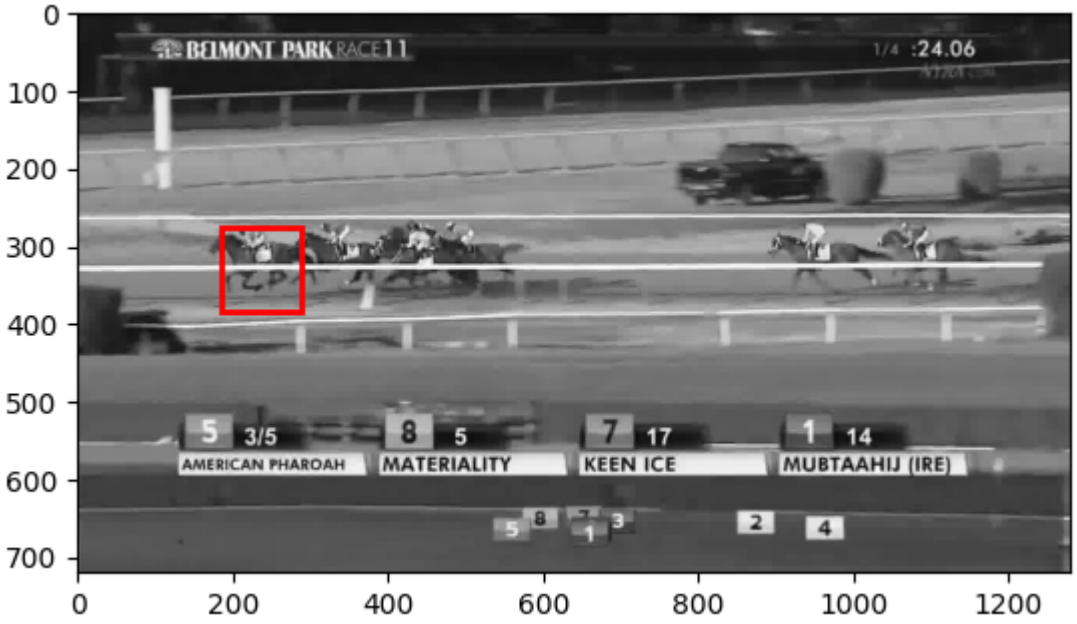

0
200
400
600
800
0
200
400
600
800
1000

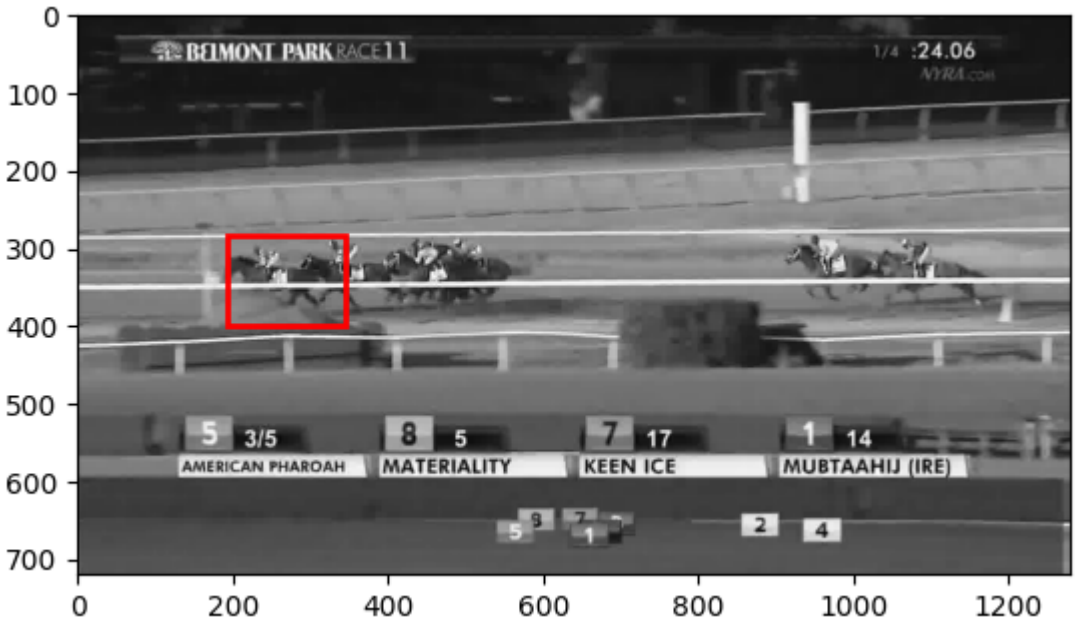

0
200
400
600
800
0
200
400
600
800
1000

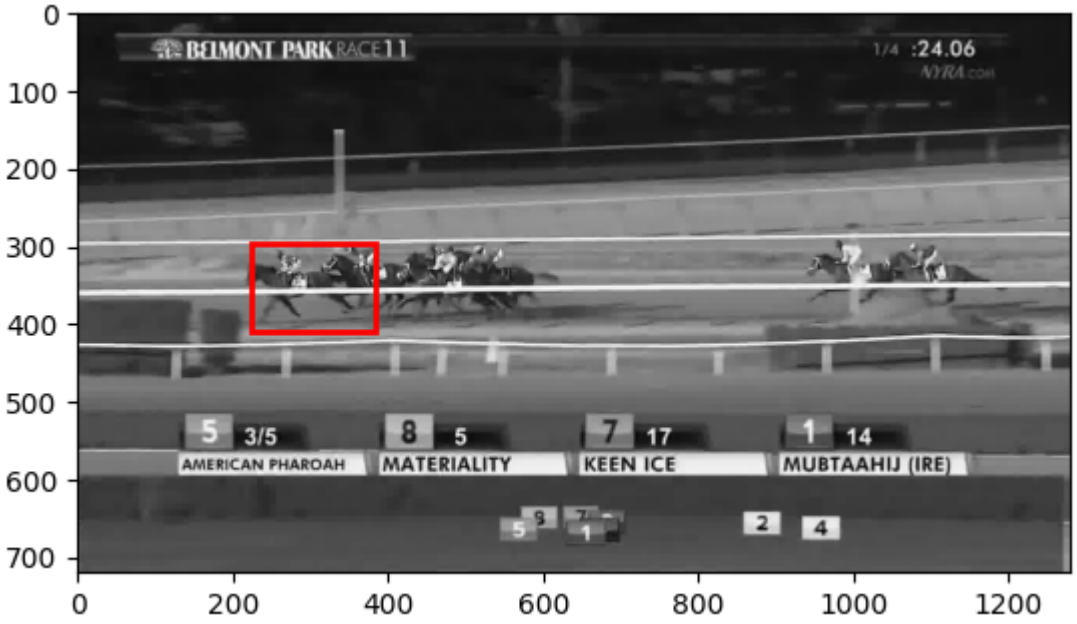


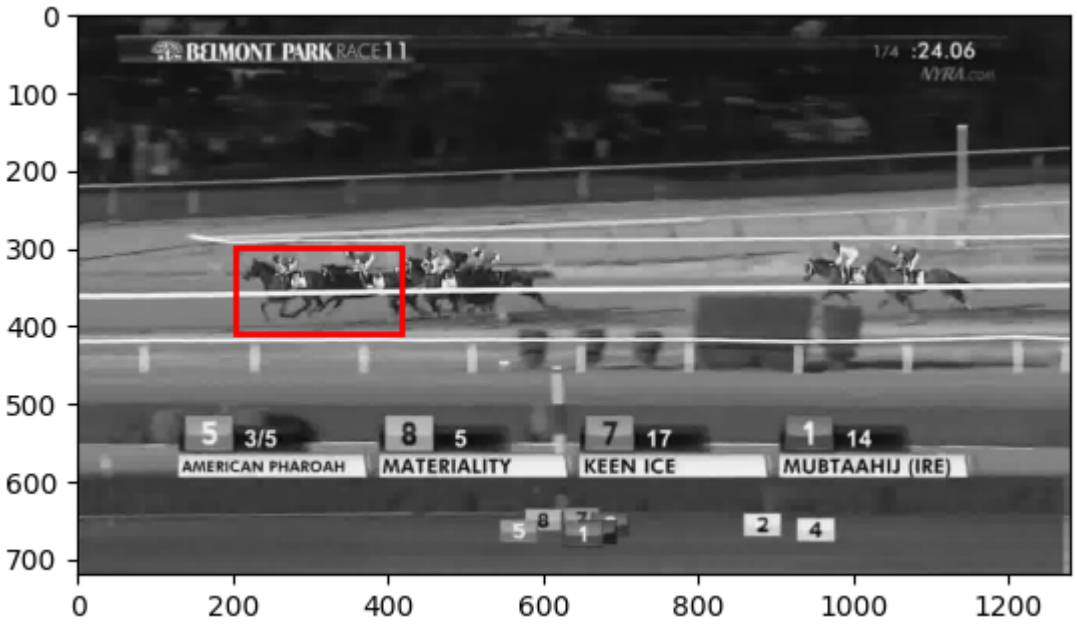
Results for race.npy

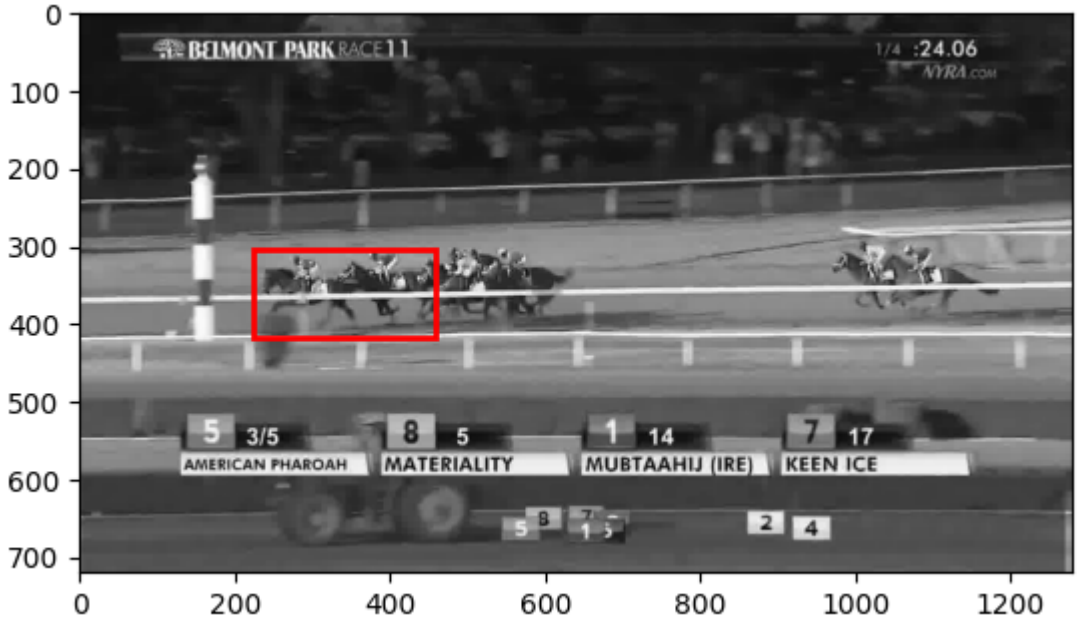


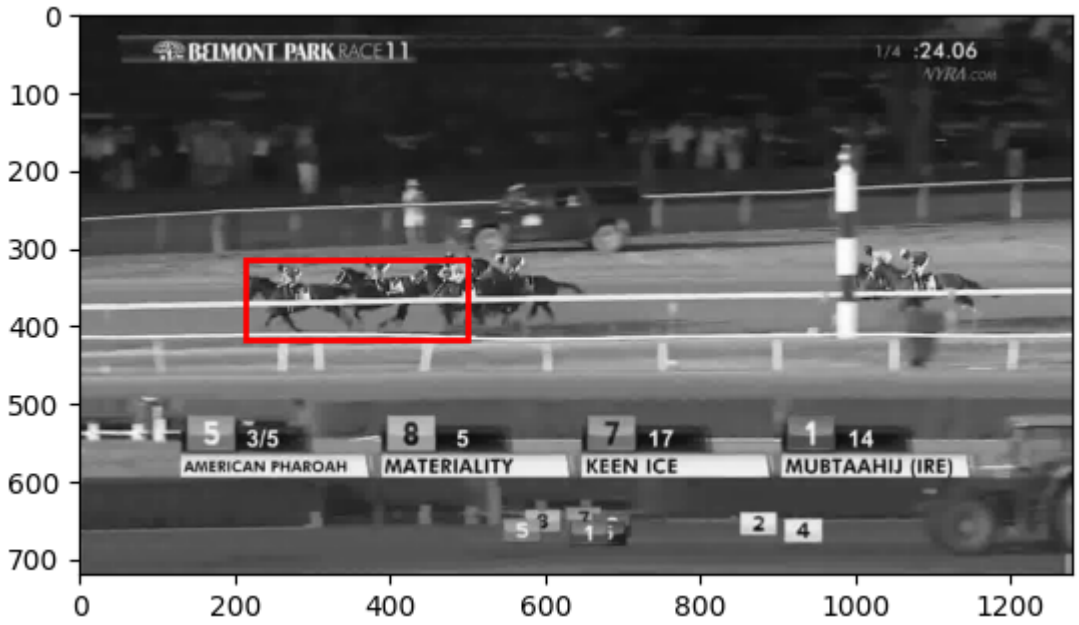

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200





Code:

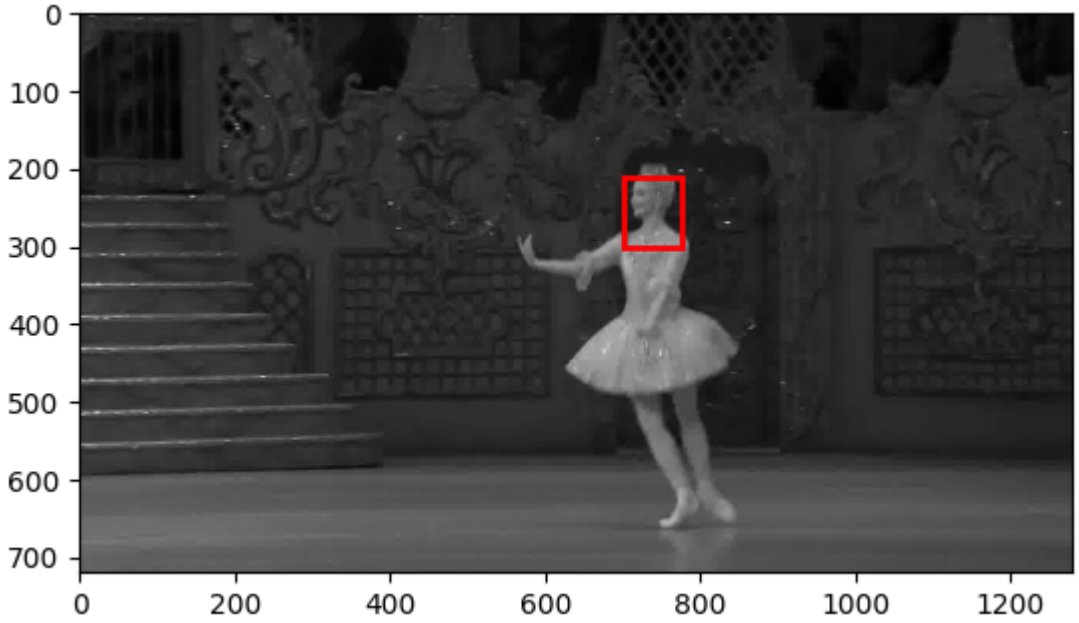
In [ ]:
```
def LucasKanadeAffine(It, It1, rect, thresh=.025, maxIt=100):
    '''
    Q1.2: Lucas-Kanade Forward Additive Alignment with Affine MAtrix

      Inputs:
        It: template image
        It1: Current image
        rect: Current position of the object
        (top left, bottom right coordinates, x1, y1, x2, y2)
        thresh: Stop condition when dp is too small
        maxIt: Maximum number of iterations to run

      Outputs:
```

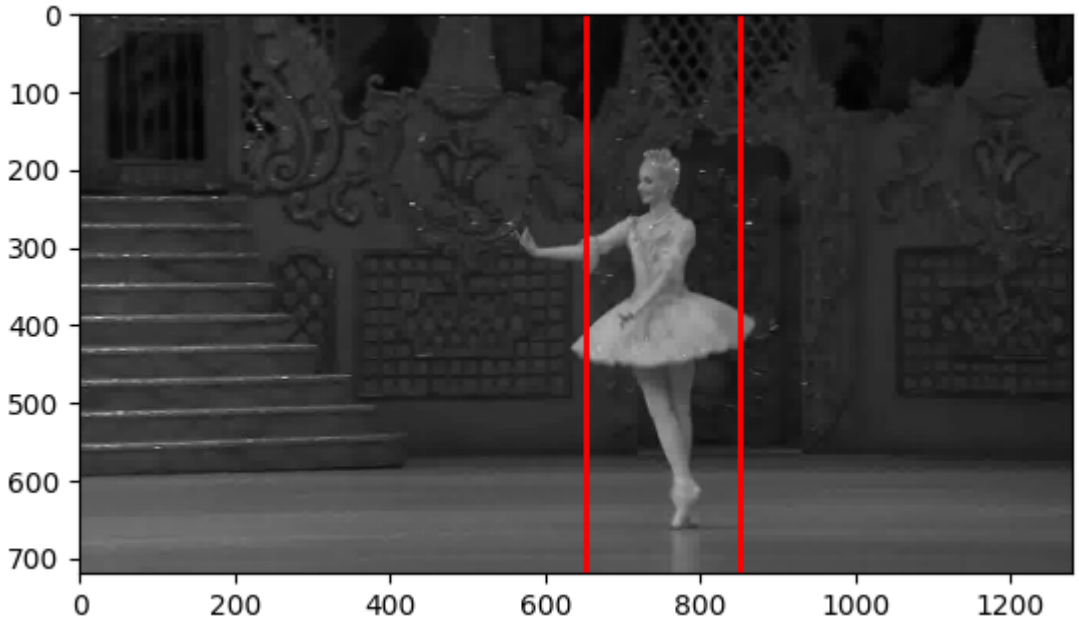
```
        M: Affine mtarix (2x3)
    '''

    # Set thresholds (you probably want to play around with the values)
    M = np.zeros((2,3))
    threshold = thresh
    maxIters = maxIt
    i = 0
    x1, y1, x2, y2 = rect

    # ----- TODO -----
    # YOUR CODE HERE
    p = np.zeros(6) # dx, dy
    del_p = np.ones(6,)
    img_t_spline = RectBivariateSpline(np.arange(It.shape[0]), np.arange(It.sha
    mesh_x, mesh_y = np.meshgrid(np.arange(x1,x2), np.arange(y1,y2))
    wrapped_img_t_spline = img_t_spline.ev(mesh_y ,mesh_x).flatten()

    img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arange(It1
    grady_img_t1, gradx_img_t1 = np.gradient(It1)
    gradx_img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arang
    grady_img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arang

    homo_mesh = np.vstack((mesh_x.flatten(), mesh_y.flatten(), np.ones(mesh_x.s
    curr_itr = 0
    while np.linalg.norm(del_p) > threshold and curr_itr < maxIters:

      W = np.array([[1,0,0],[0,1,0]])+np.array([[p[0], p[2], p[4]], [p[1], p[3
      homo = np.array([0,0,1])
      W = np.vstack((W, homo))
      warped_block = np.matmul(W, homo_mesh)
      wrapped_img_t1_spline = img_t1_spline.ev(warped_block[1,:], warped_block

      b = wrapped_img_t_spline - wrapped_img_t1_spline

      wrapped_gradx = gradx_img_t1_spline.ev(warped_block[1,:], warped_block[0
      wrapped_grady = grady_img_t1_spline.ev(warped_block[1,:], warped_block[0
      wrapped_gradx = np.expand_dims(wrapped_gradx, axis=1)
      wrapped_grady = np.expand_dims(wrapped_grady, axis=1)
      # print(wrapped_gradx.shape)
      grad = np.dstack((wrapped_gradx, wrapped_grady))
      x_coords = np.expand_dims(warped_block[0,:], axis=1)
      y_coords = np.expand_dims(warped_block[1,:], axis=1)
      # print(y_coords.shape)
      # print("zerp",np.zeros((x_coords.shape)).shape)
      dhoW_dhoP1 = np.dstack((x_coords, np.zeros((x_coords.shape)), y_coords, n
      dhoW_dhoP2 = np.dstack((np.zeros((x_coords.shape)), x_coords, np.zeros((x
      # print("check", dhoW_dhoP2.shape)
      # dhoW_dhoP = np.stack((dhoW_dhoP1, dhoW_dhoP2), axis=1)
      dhoW_dhoP = np.hstack((dhoW_dhoP1, dhoW_dhoP2))
      # print("grad",grad.shape)
      # print("dhow", dhoW_dhoP.shape)
      # raise NotImplementedError()
      A = np.matmul(grad, dhoW_dhoP)
      A = A[:,0,:]

      del_p = np.linalg.lstsq(A, b, rcond=None)[0]
```

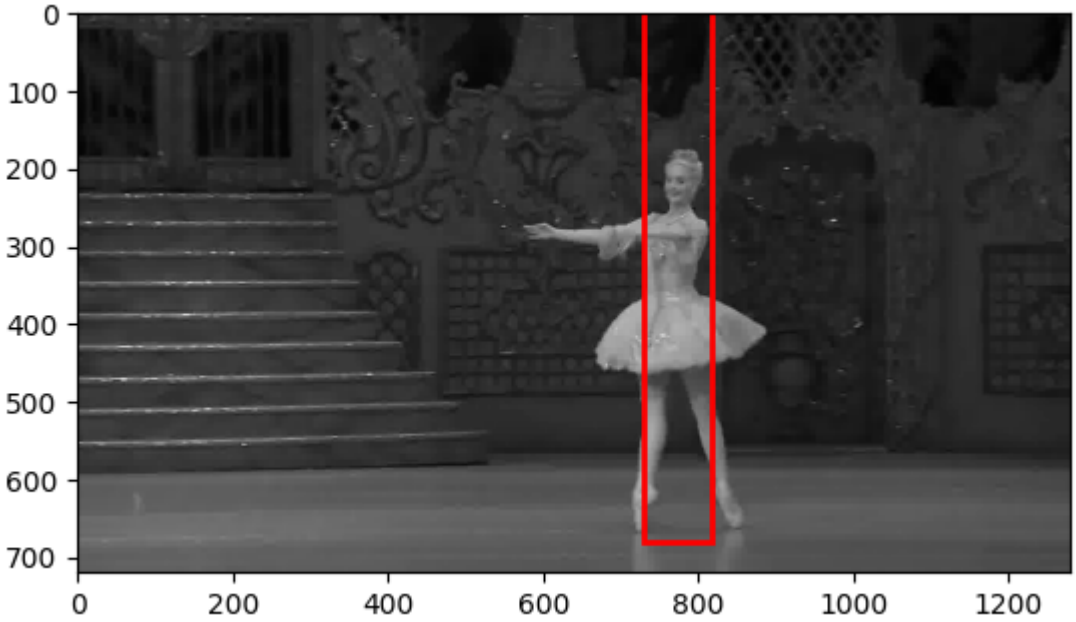
```
        p += del_p
        curr_itr += 1
    M = np.array([[1 + p[0], p[2], p[4]], [p[1], 1+p[3], p[5]]])
    # raise NotImplementedError()

    return M
```

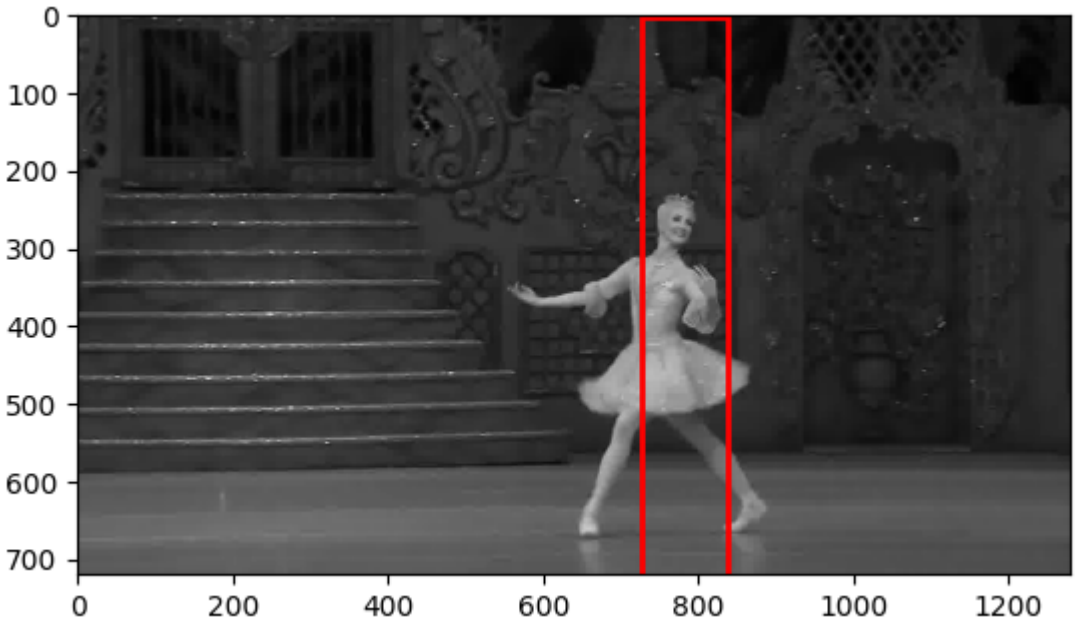
Q2.1

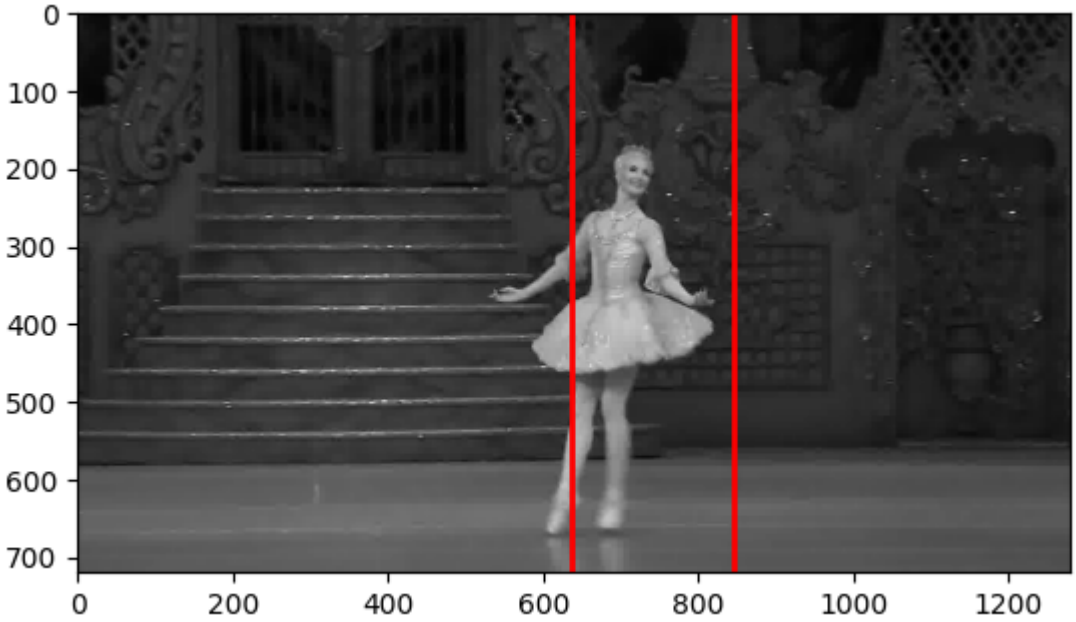
YOUR ANSWER HERE Results for ballet.npy:

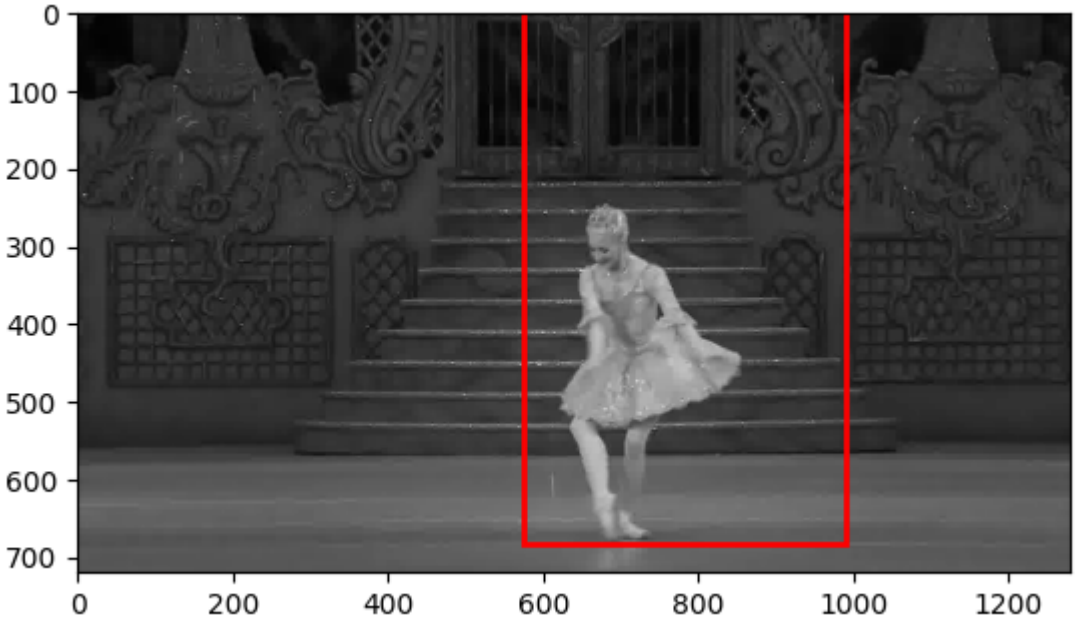




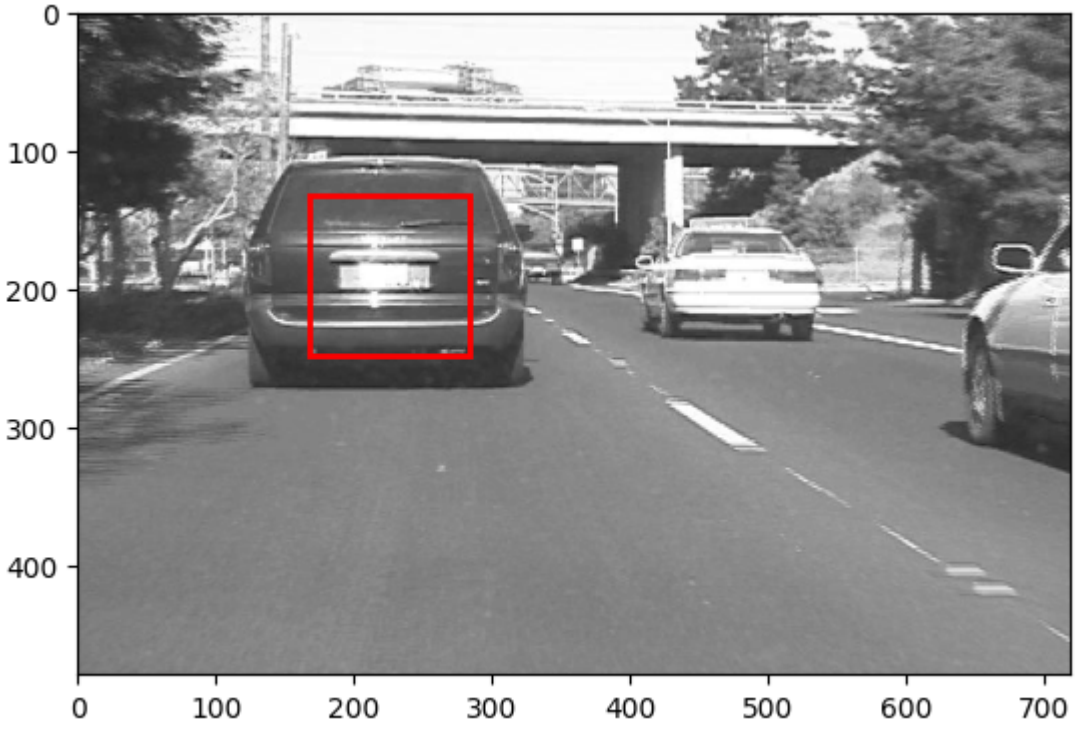

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Results for car1.npy:




0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

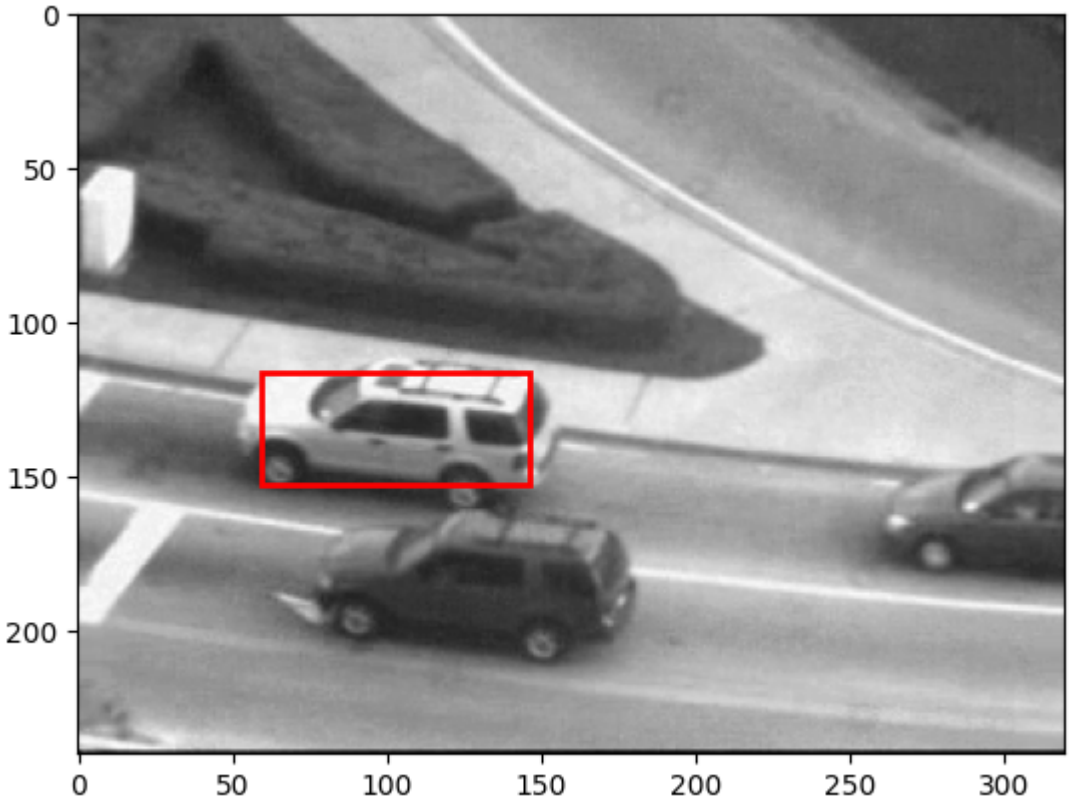

0
100
200
300
400
0
100
200
300
400
500
600
700

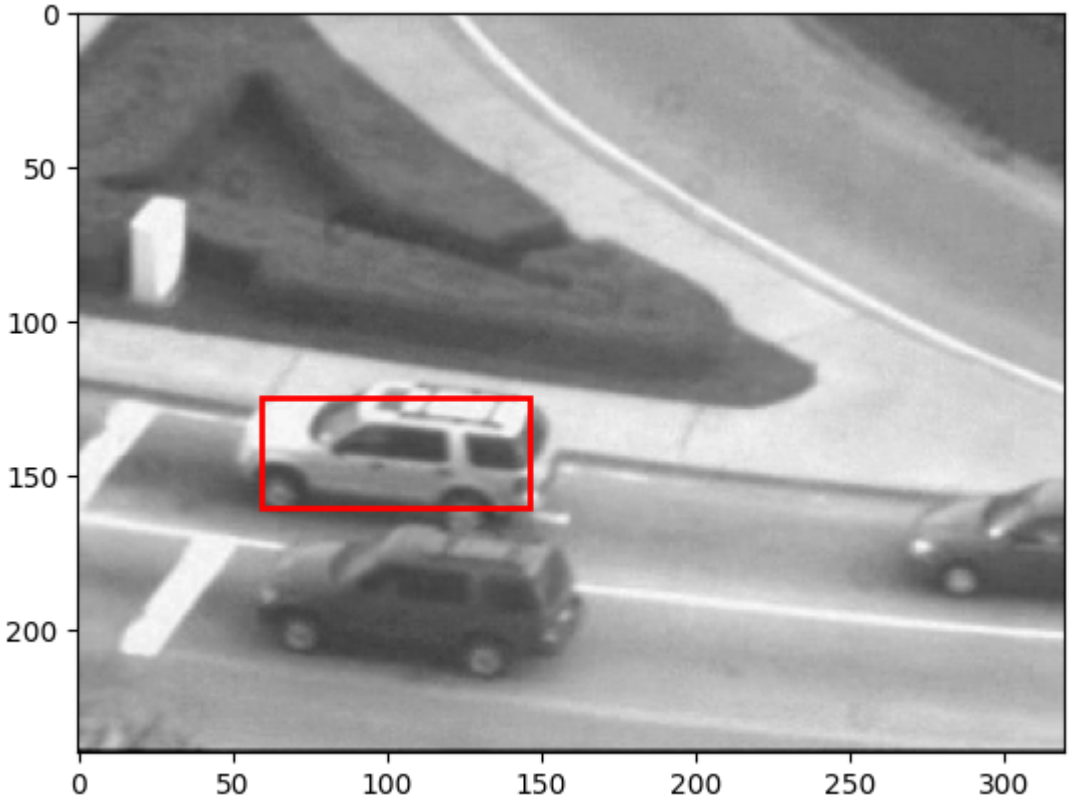

0
100
200
300
400
0
100
200
300
400
500
600
700

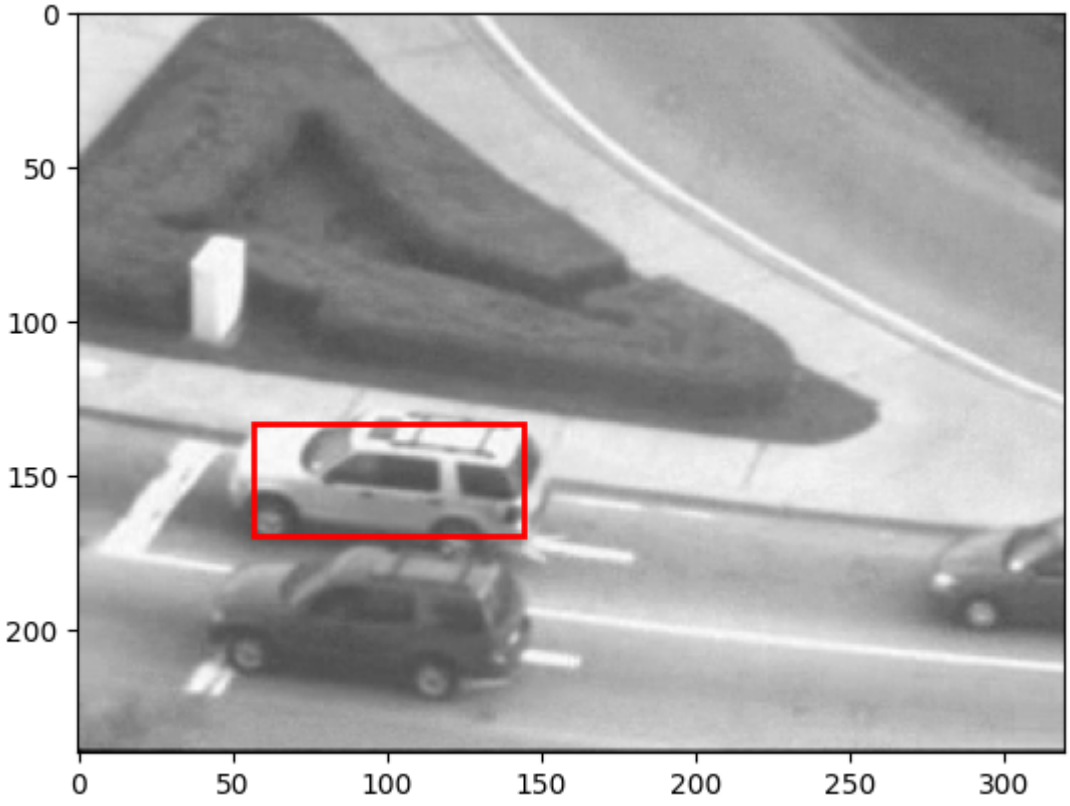

0
100
200
300
400
0
100
200
300
400
500
600
700

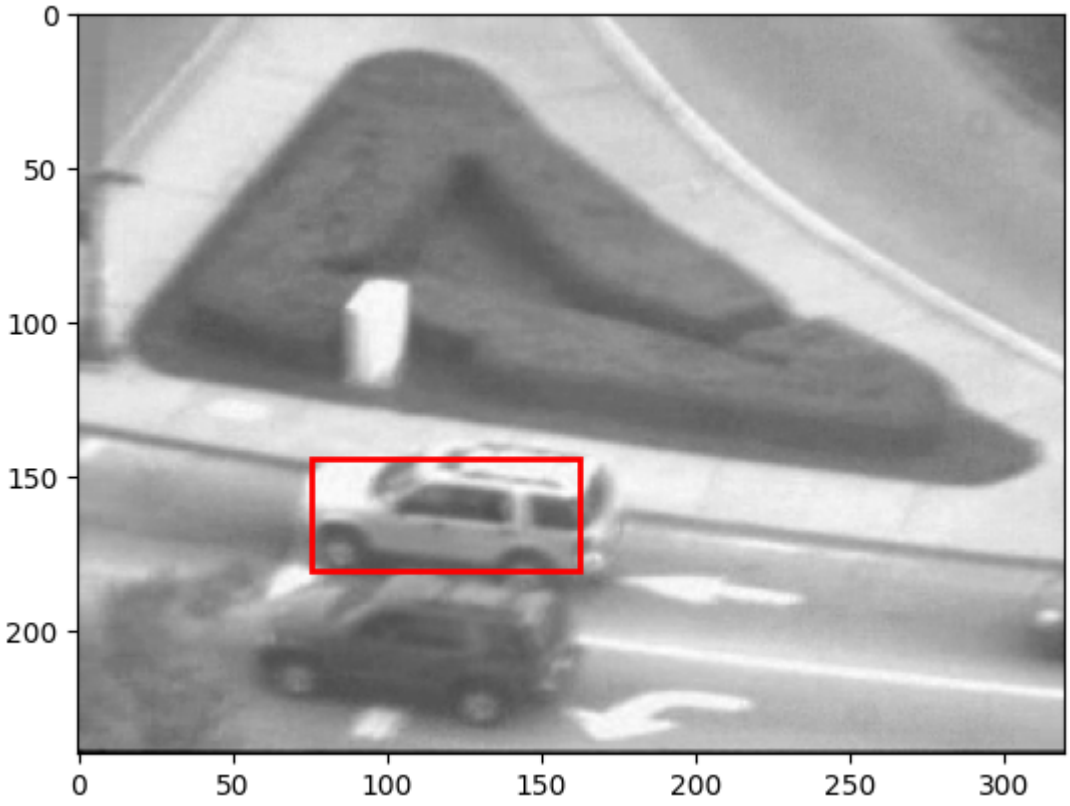

0
100
200
300
400
0
100
200
300
400
500
600
700

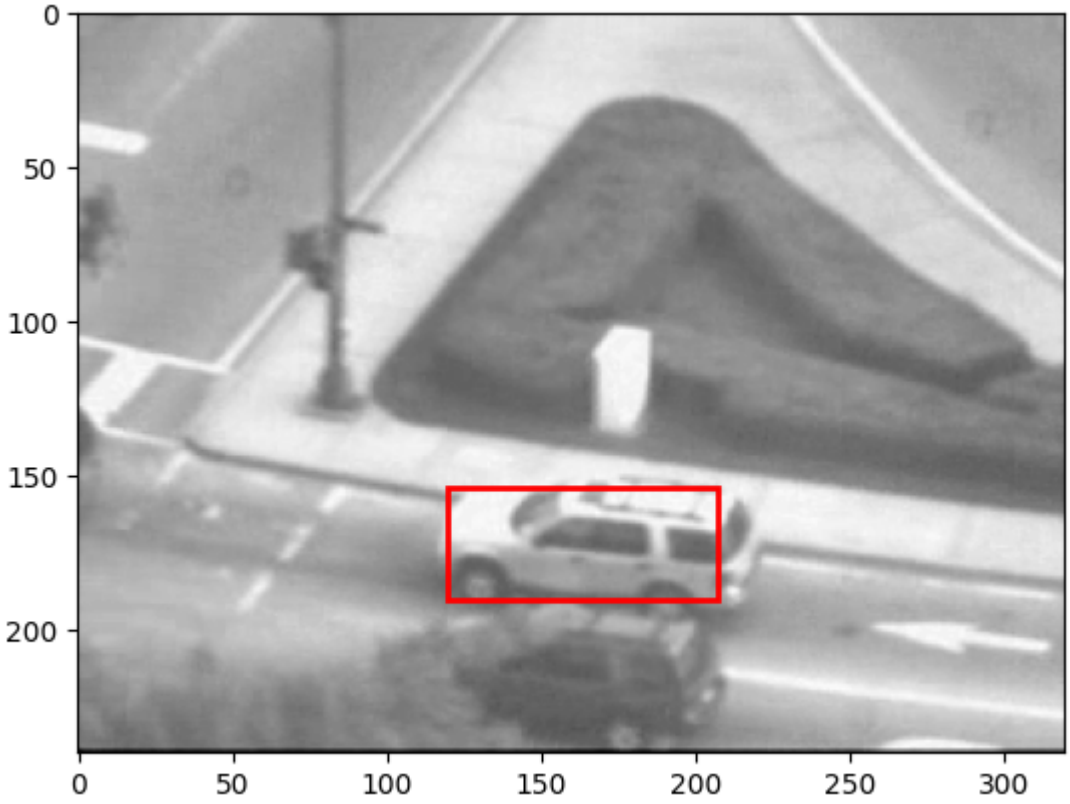
Results for car2.npy



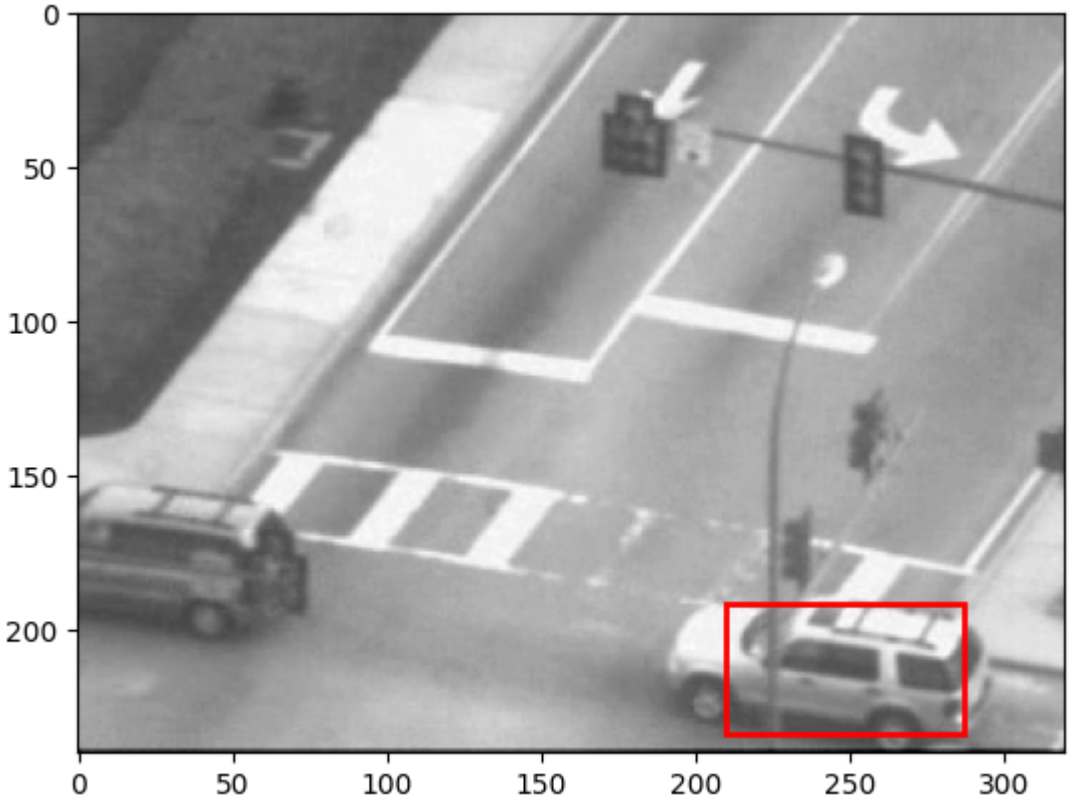


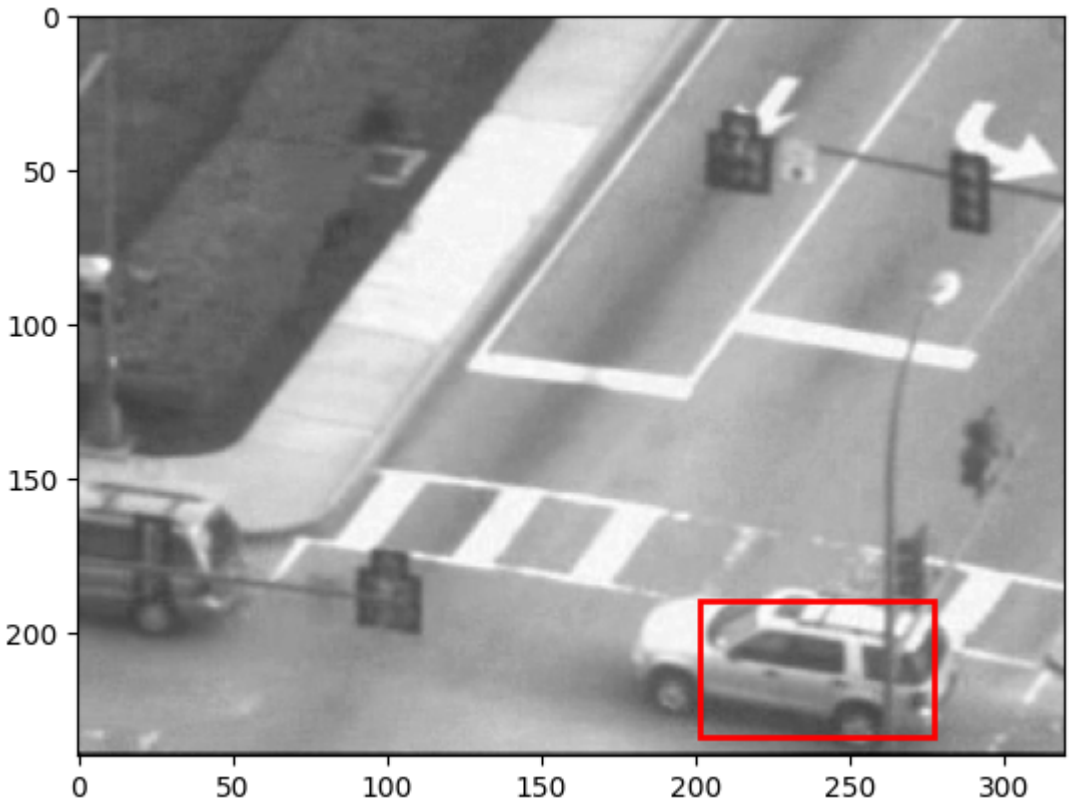

0
50
100
150
200
0
50
100
150
200
250
300

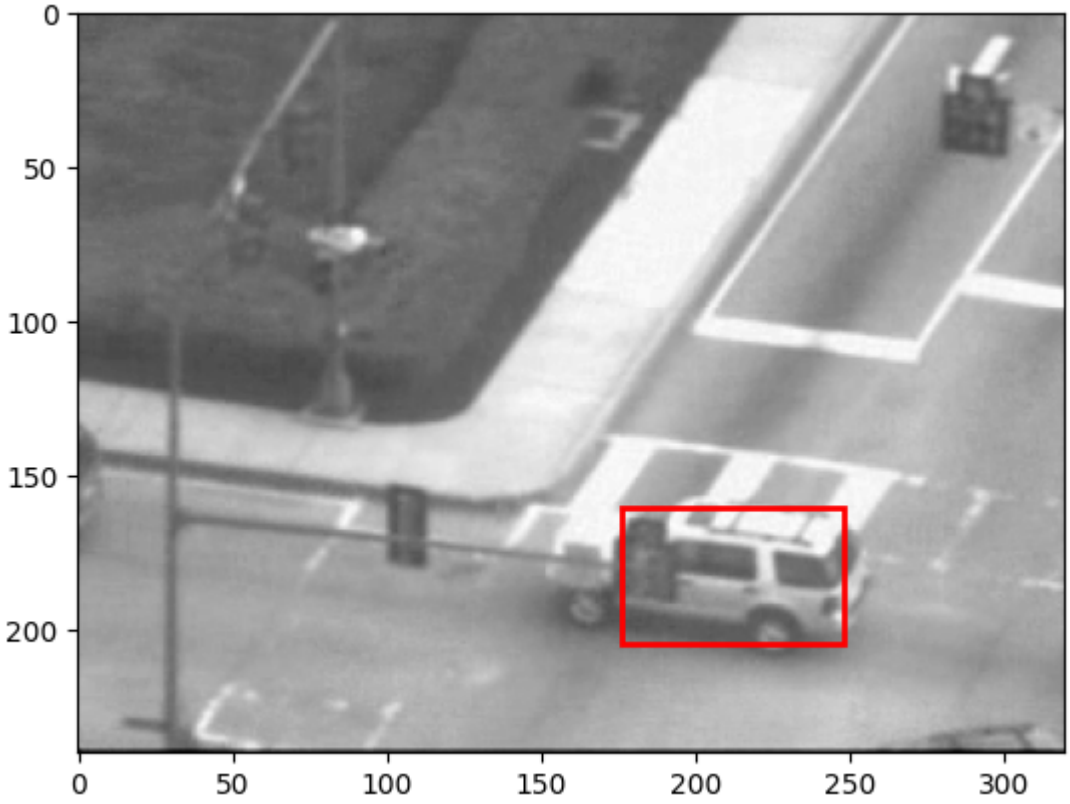

0
50
100
150
200
0
50
100
150
200
250
300

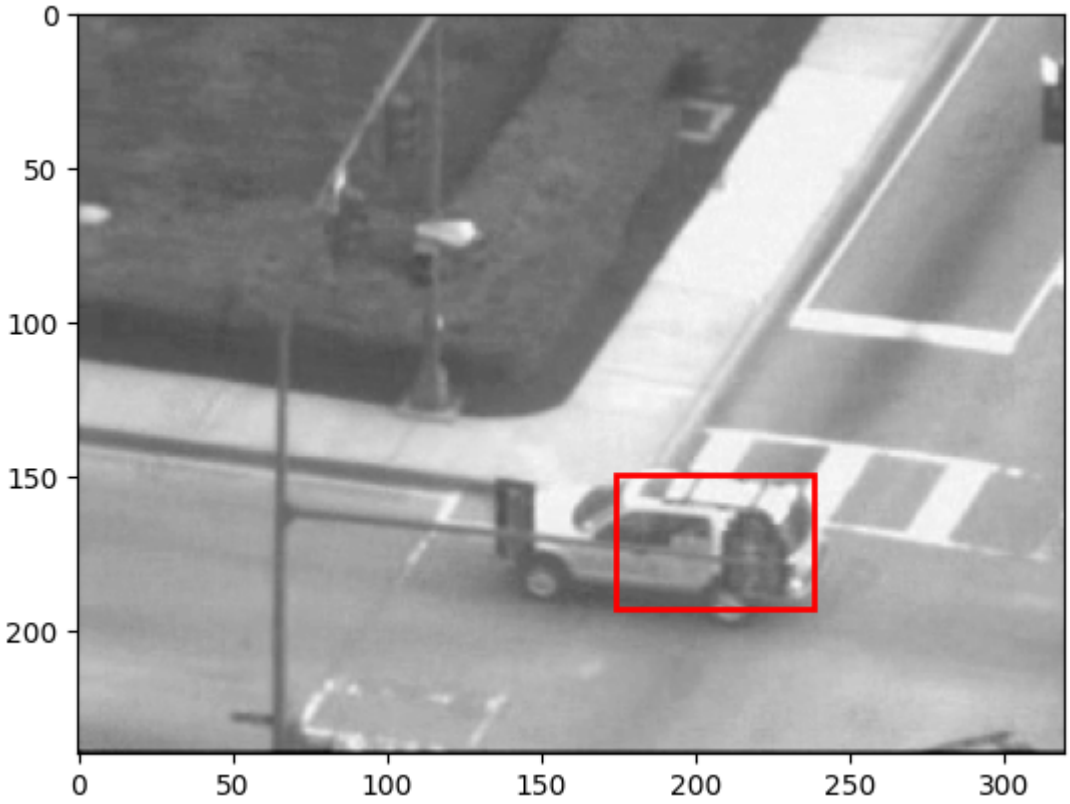

0
50
100
150
200
0
50
100
150
200
250
300

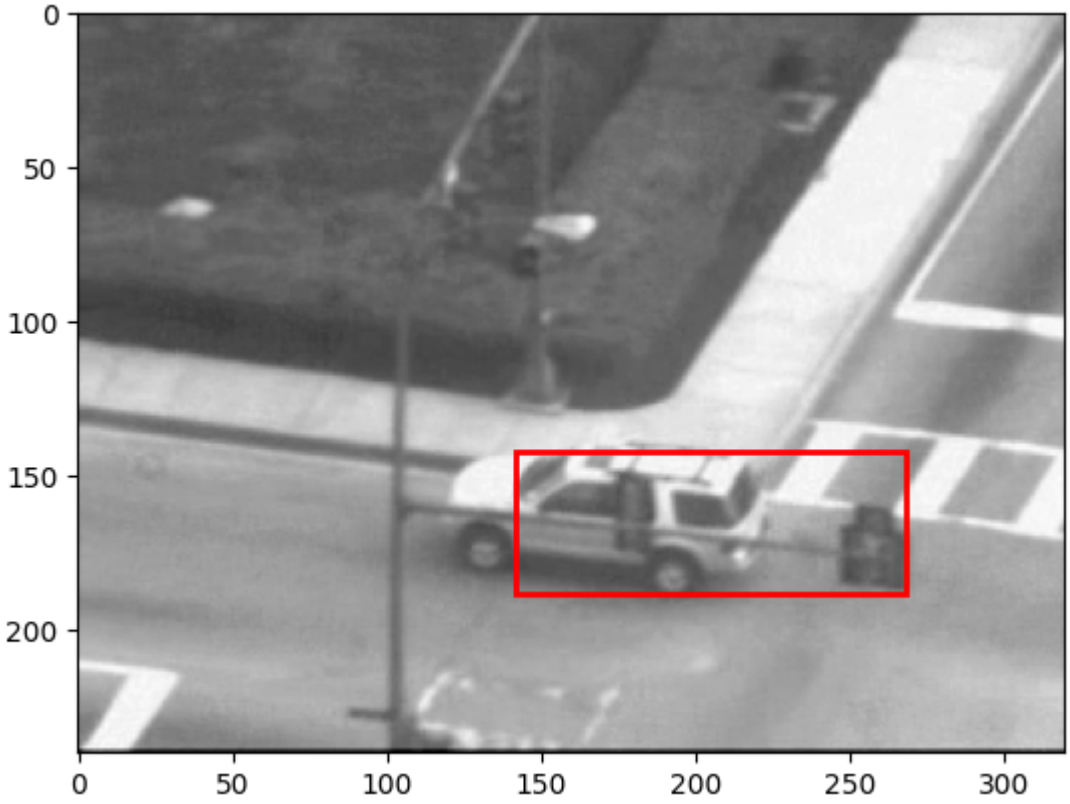

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

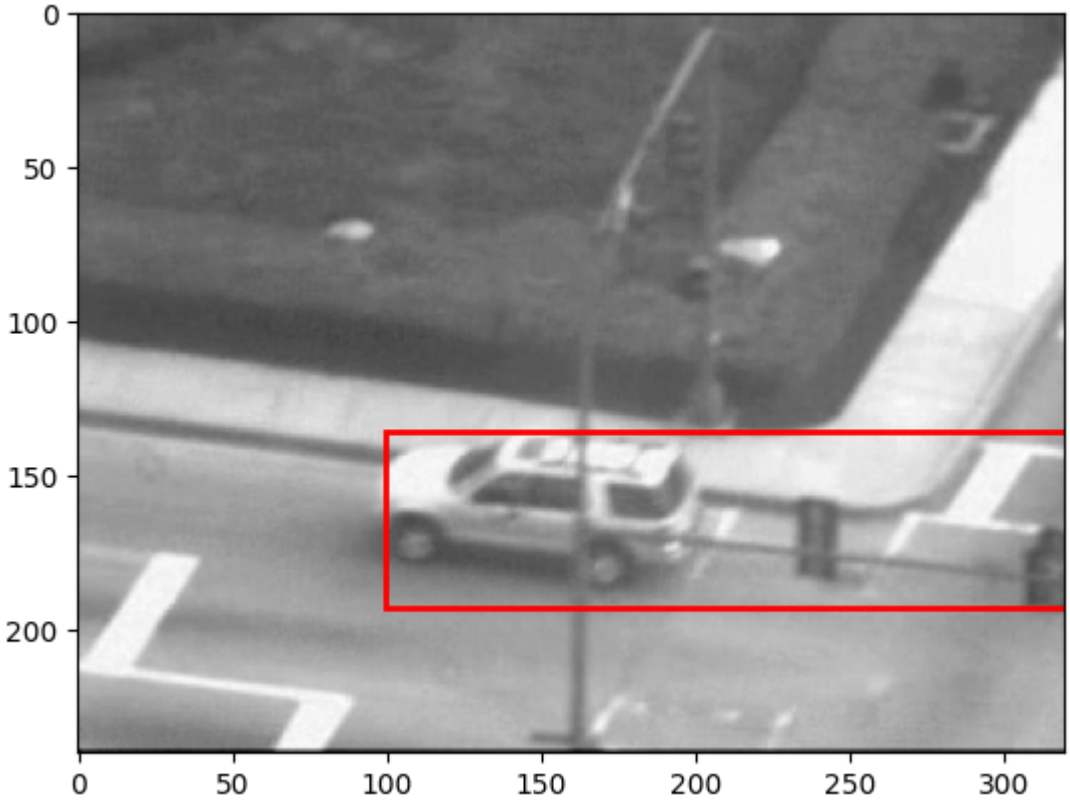

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

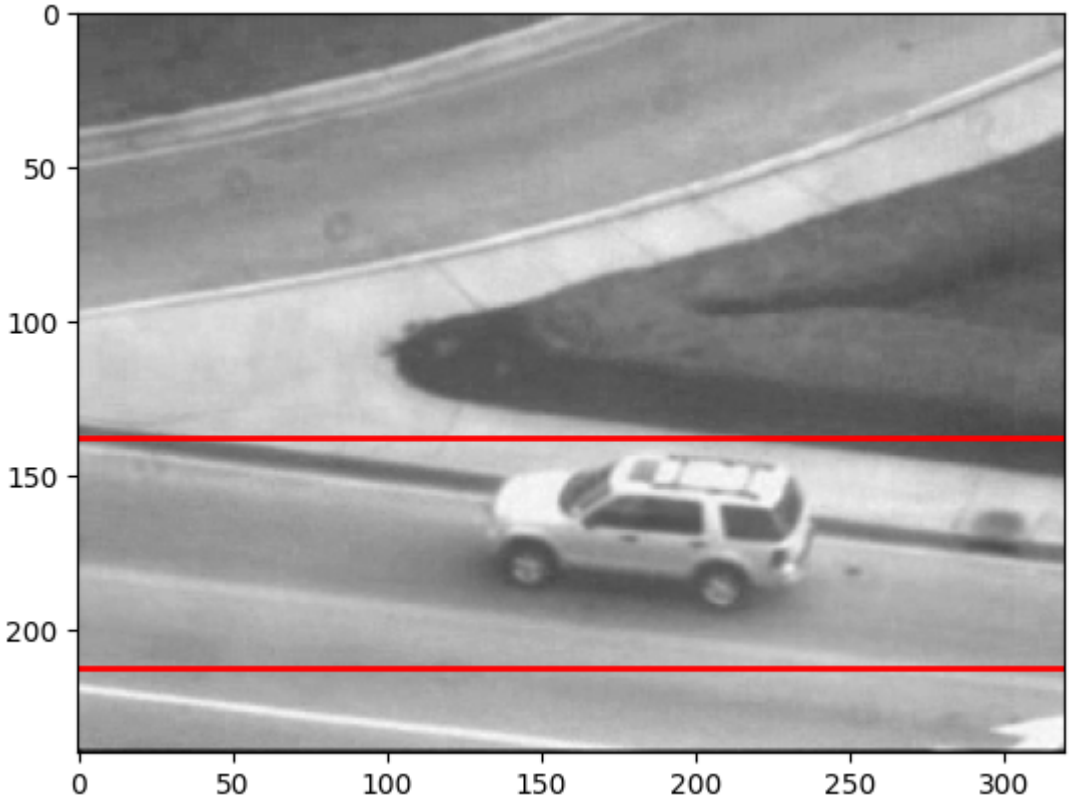

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

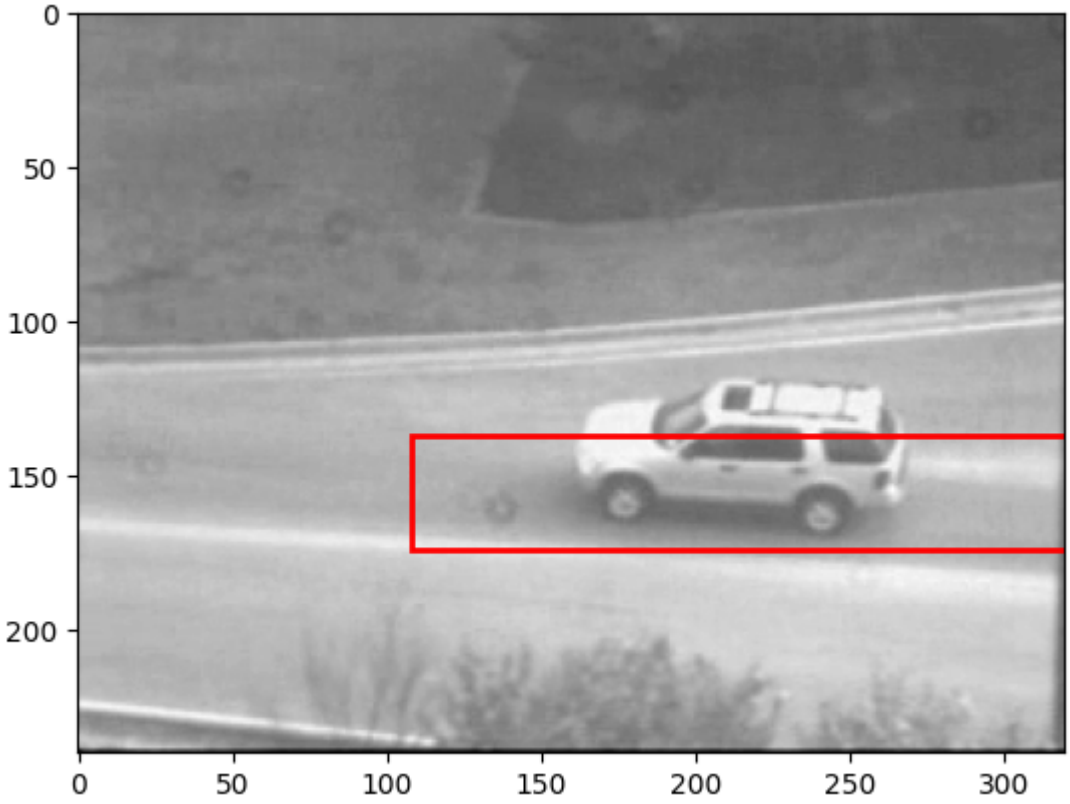

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

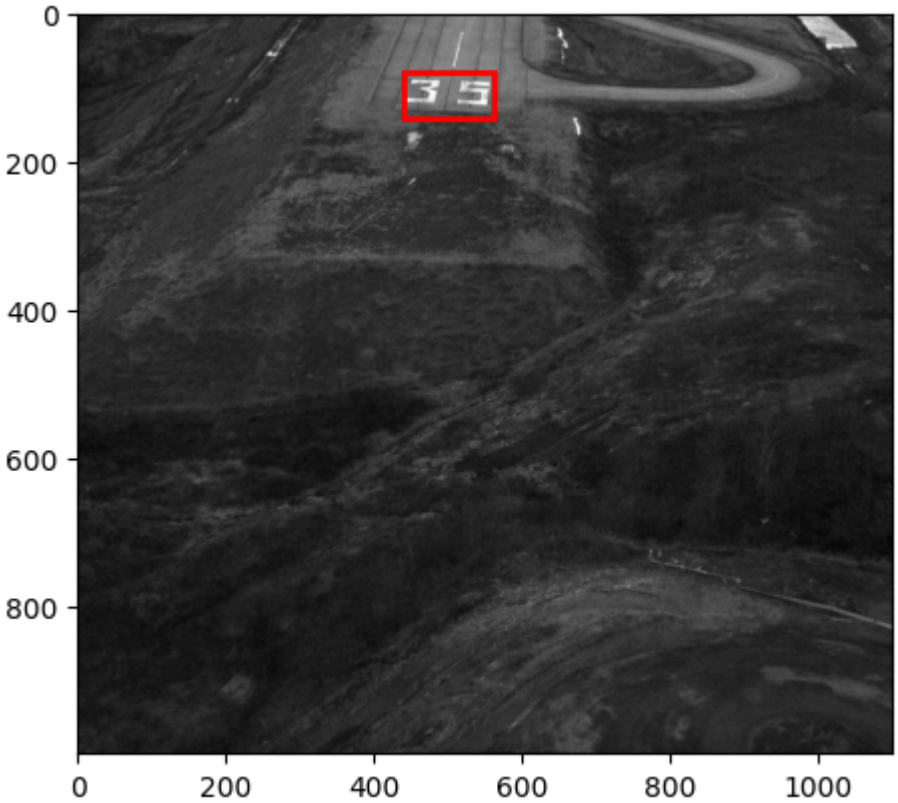

0
50
100
150
200
0
50
100
150
200
250
300

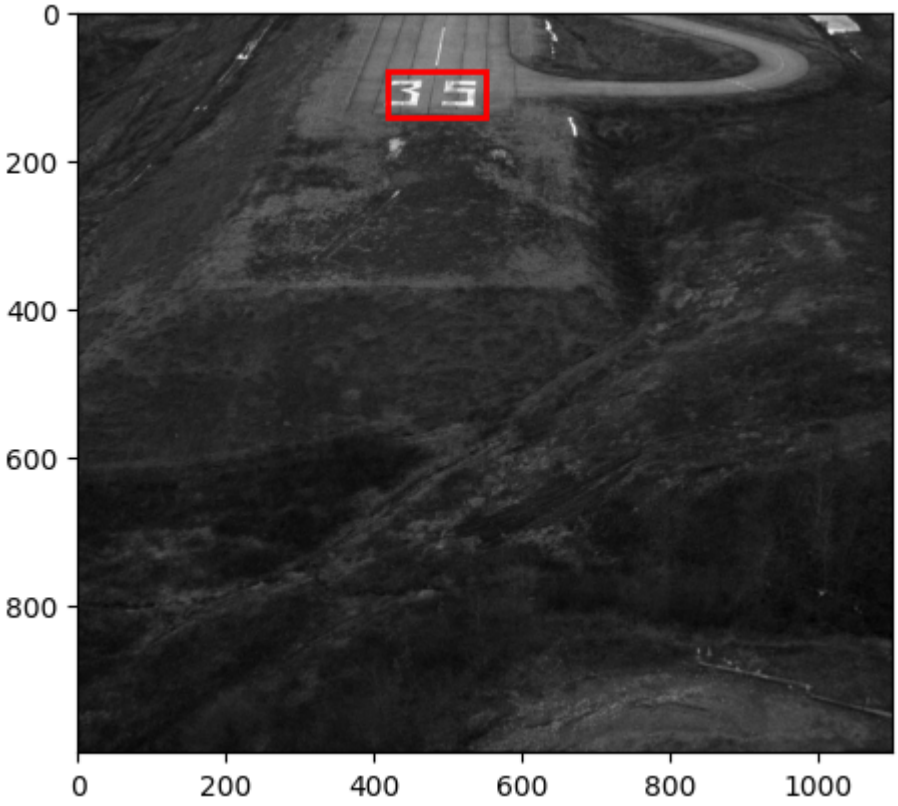

0
50
100
150
200
0
50
100
150
200
250
300

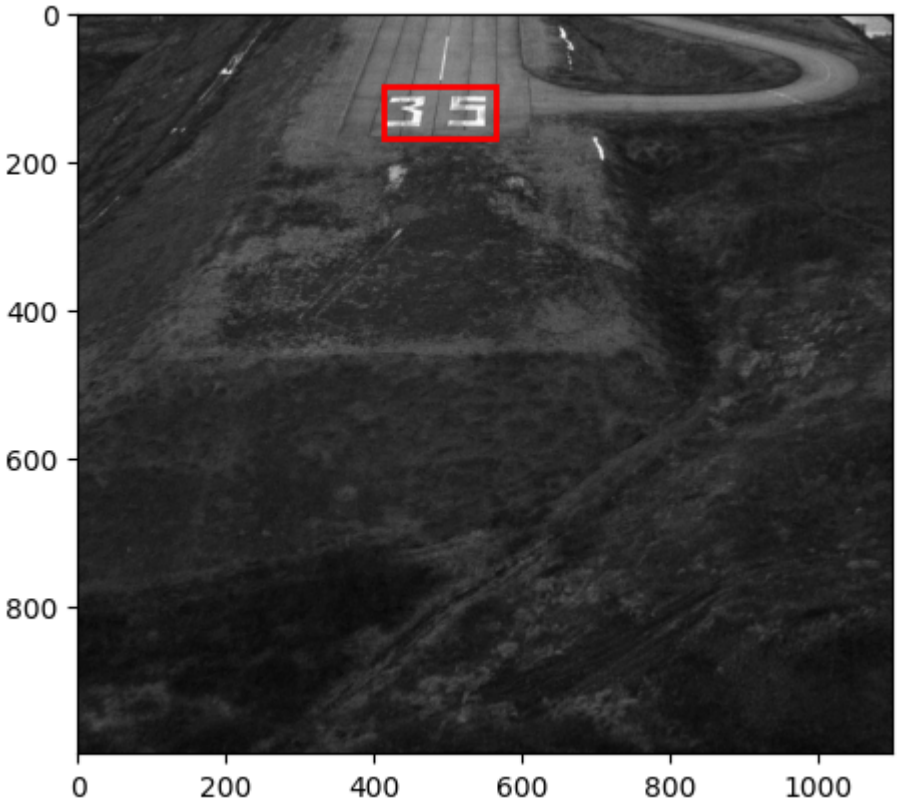

0
50
100
150
200
0
50
100
150
200
250
300

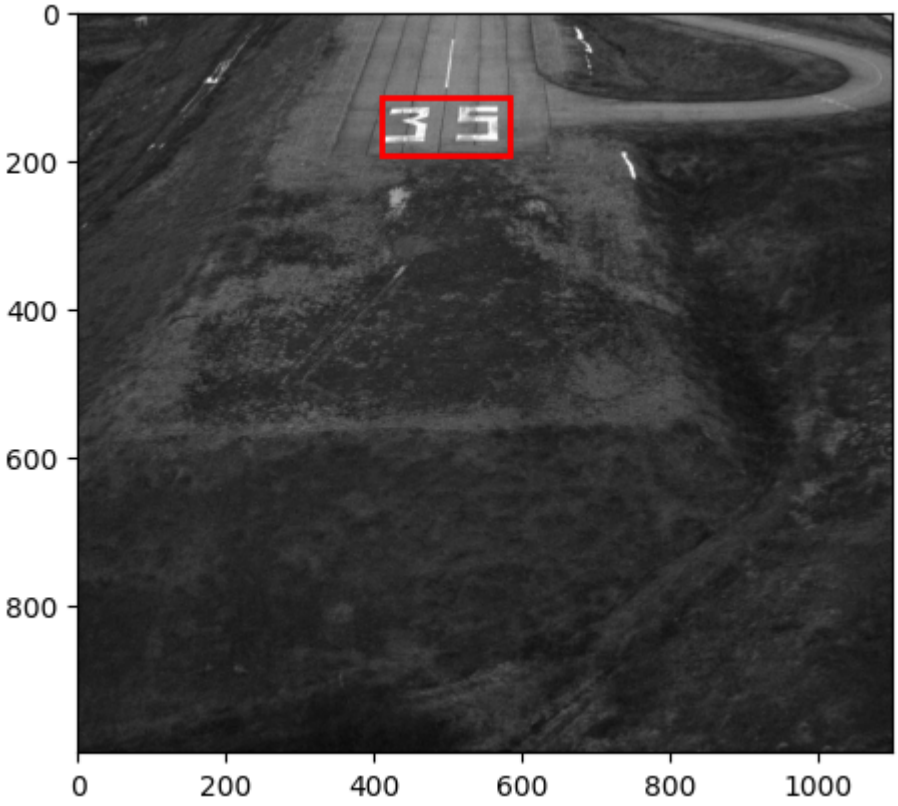


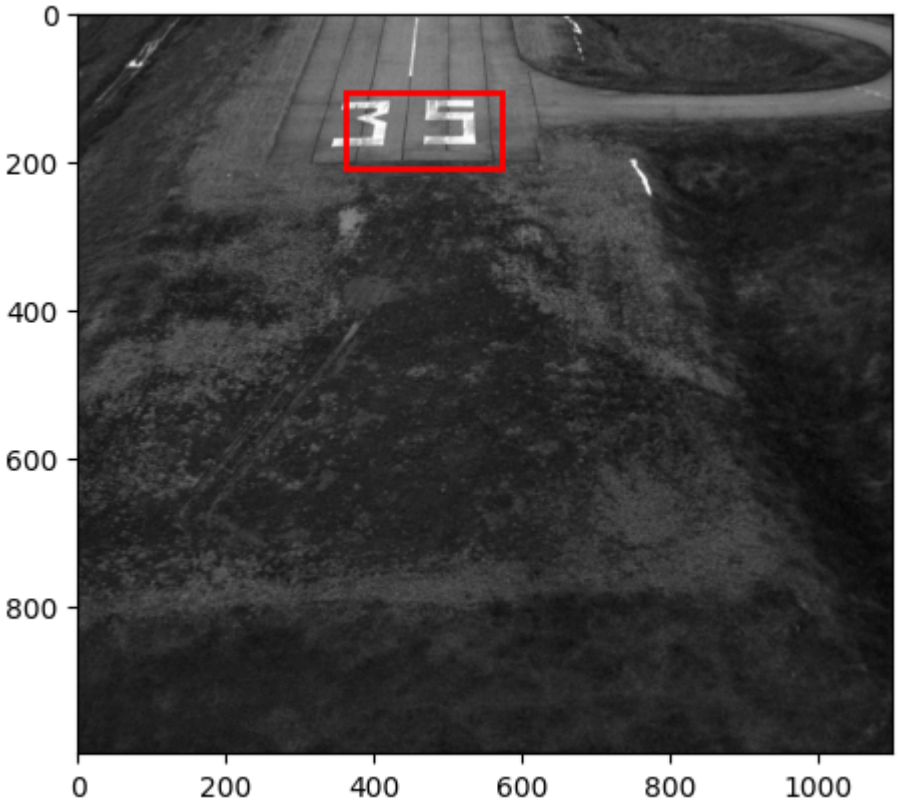
Results for landing.npy

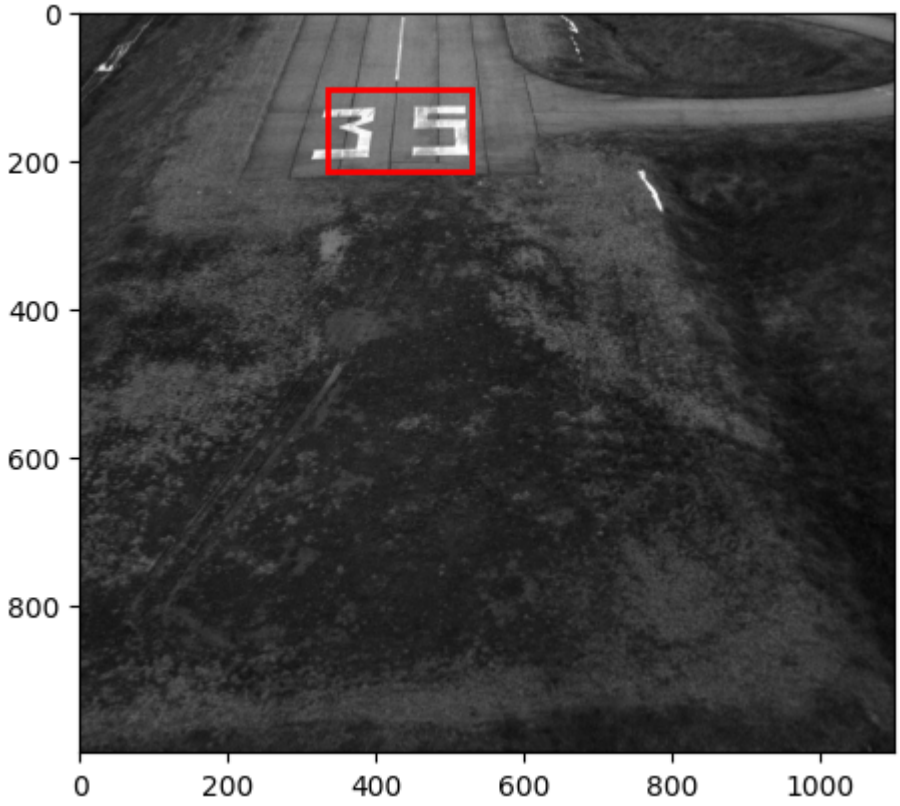



0
200
400
600
800
0
200
400
600
800
1000

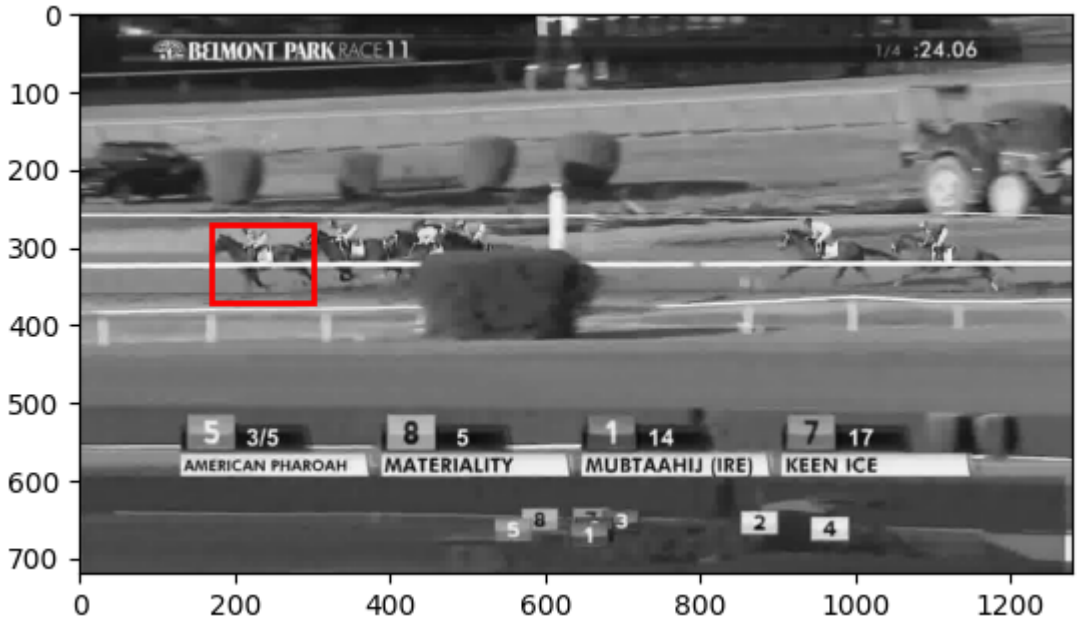

0
200
400
600
800
0
200
400
600
800
1000

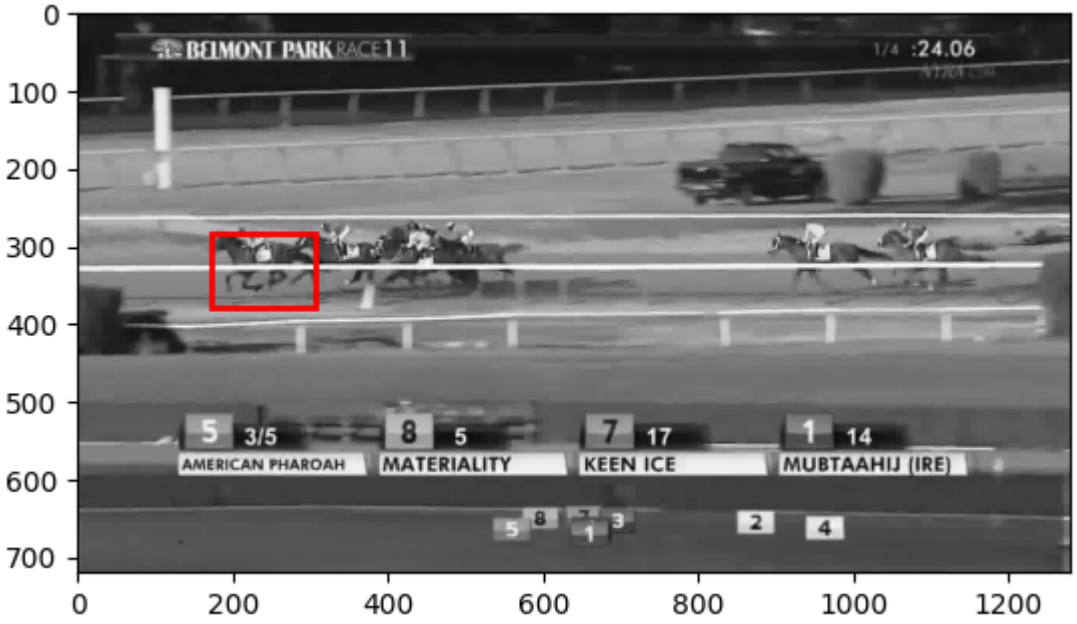

0
200
400
600
800
0
200
400
600
800
1000

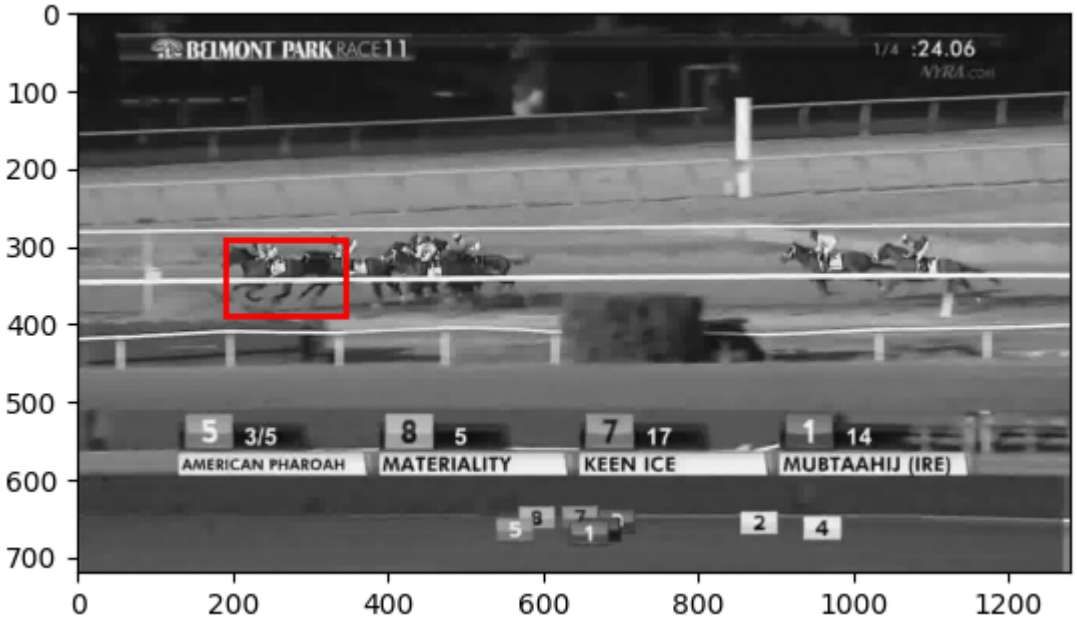

0
200
400
600
800
0
200
400
600
800
1000

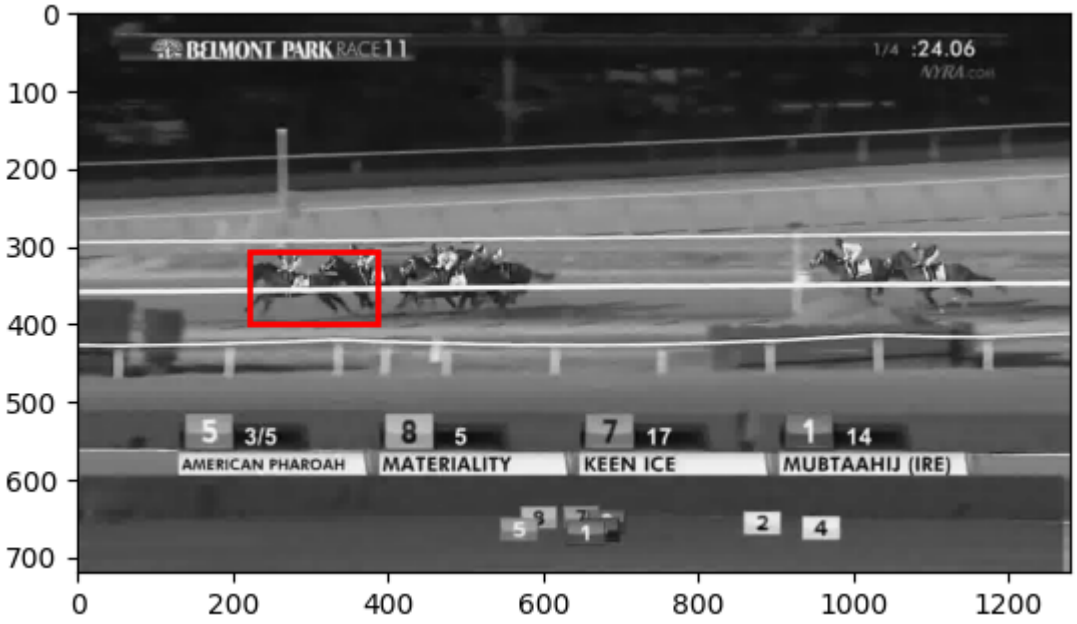


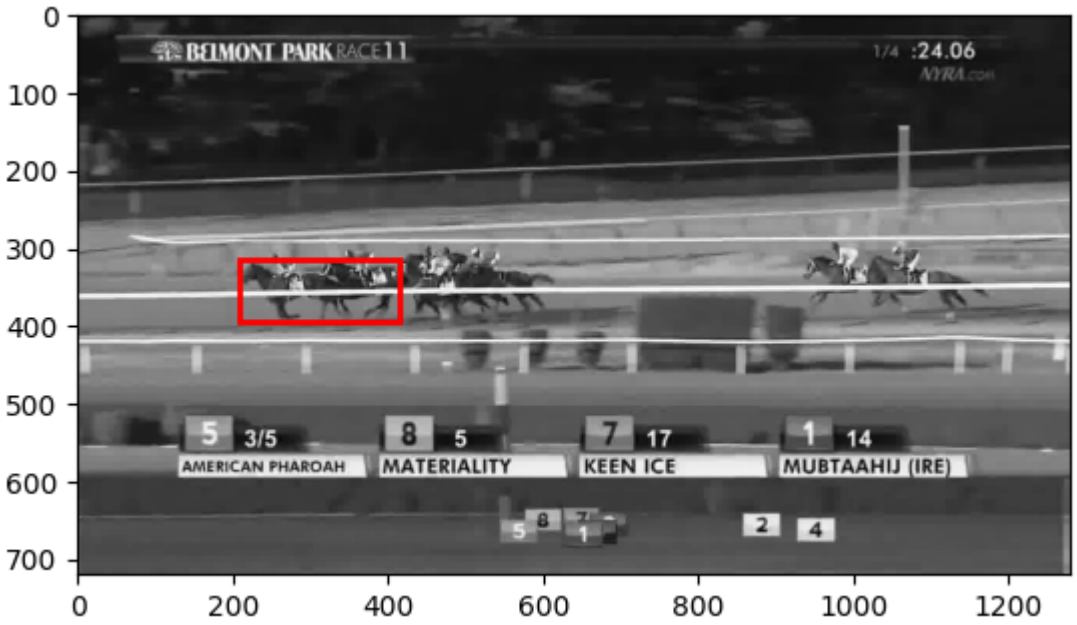
Results for race.npy

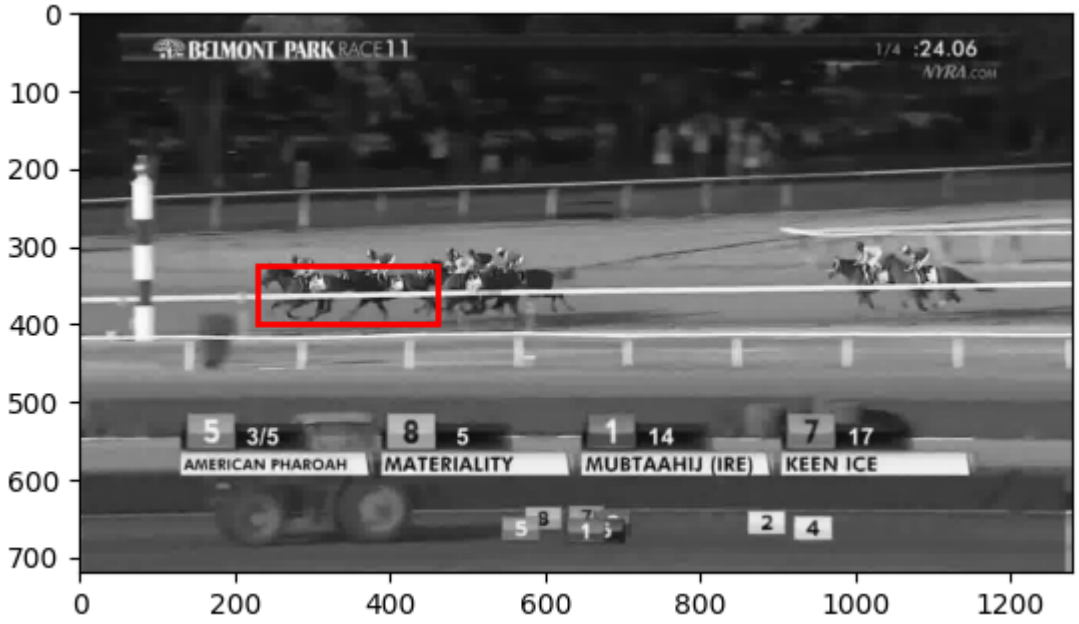


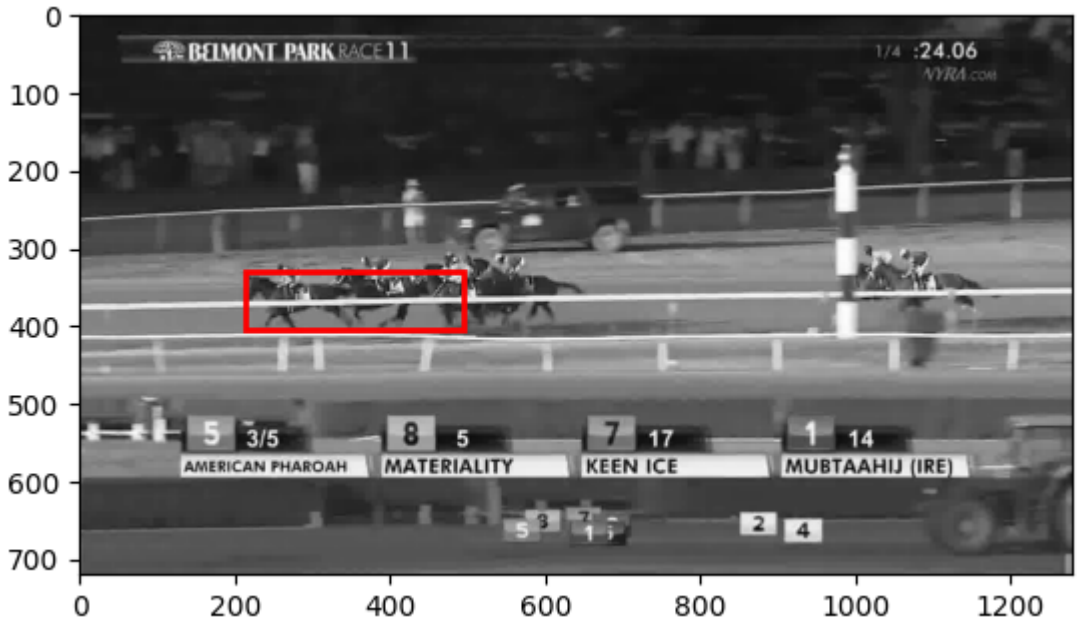

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)


BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)
0 100 200 300 400 500 600 700
0 200 400 600 800 1000 1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)
0 100 200 300 400 500 600 700
0 200 400 600 800 1000 1200





Code:

In [ ]:
```
def InverseCompositionAffine(It, It1, rect, thresh=.025, maxIt=100):
    '''
    Q2.1: Matthew-Bakers Inverse Compositional Alignment with Affine MAtrix

      Inputs:
        It: template image
        It1: Current image
        rect: Current position of the object
        (top left, bottom right coordinates, x1, y1, x2, y2)
        thresh: Stop condition when dp is too small
        maxIt: Maximum number of iterations to run

      Outputs:
```

```
        M: Affine mtarix (2x3)
    '''

    # Set thresholds (you probably want to play around with the values)
    M = np.zeros((2,3))
    threshold = thresh
    maxIters = maxIt
    i = 0
    x1, y1, x2, y2 = rect

    # ----- TODO -----
    # YOUR CODE HERE
    p = np.zeros(6,)
    # del_p = np.ones(6,)

    img_t_spline = RectBivariateSpline(np.arange(It.shape[0]), np.arange(It.sha
    mesh_x, mesh_y = np.meshgrid(np.linspace(x1, x2, int(x2-x1)+1), np.linspace
    wrapped_img_t_spline = img_t_spline.ev(mesh_y ,mesh_x).flatten()

    img_t1_spline = RectBivariateSpline(np.arange(It1.shape[0]), np.arange(It1
    homo_mesh = np.vstack((mesh_x.flatten(), mesh_y.flatten(), np.ones(mesh_x.s
    gradx_img_t1_spline = img_t_spline.ev(homo_mesh[1,:], homo_mesh[0,:], dy=1
    grady_img_t1_spline = img_t_spline.ev(homo_mesh[1,:], homo_mesh[0,:], dx=1

    gradx_img_t1_spline = np.expand_dims(gradx_img_t1_spline, axis=1)
    grady_img_t1_spline = np.expand_dims(grady_img_t1_spline, axis=1)
    # print("gradx_img_t1", gradx_img_t1_spline.shape)
    grad = np.dstack((gradx_img_t1_spline, grady_img_t1_spline))
    x_coords = np.expand_dims(homo_mesh[0,:], axis=1)
    y_coords = np.expand_dims(homo_mesh[1,:], axis=1)
    dhoW_dhoP1 = np.dstack((x_coords, np.zeros((x_coords.shape)), y_coords, np
    # print(dhoW_dhoP1.shape)
    dhoW_dhoP2 = np.dstack((np.zeros((x_coords.shape)), x_coords, np.zeros((x_
    dhoW_dhoP = np.hstack((dhoW_dhoP1, dhoW_dhoP2))

    D = np.matmul(grad, dhoW_dhoP)
    D = D[:,0,:]

    Hess = np.matmul(D.transpose(), D)
    Mat = np.matmul(np.linalg.inv(Hess), D.transpose())

    curr_itr = 0
    # while np.linalg.norm(del_p) < threshold and curr_itr < maxIters:
    while curr_itr < maxIters:
      W = np.array([[1,0,0],[0,1,0]])+np.array([[p[0], p[2], p[4]], [p[1], p[3
      homo = np.array([0,0,1])
      W = np.vstack((W, homo))
      warped_block = np.matmul(W, homo_mesh)
      wrapped_img_t1_spline = img_t1_spline.ev(warped_block[1,:], warped_block

      error = -wrapped_img_t_spline + wrapped_img_t1_spline
      error = np.expand_dims(error, axis=1)
      # print(error.shape)
      # del_p = np.matmul(Mat, error).reshape((-1,))
      del_p = (np.matmul(Mat, error)).flatten()
      # print(del_p.shape)
```

```
    inv_warp = np.array([-del_p[0]-(del_p[0]*del_p[3])+(del_p[1]*del_p[2]),
    inv_warp = inv_warp/((1+del_p[0])*(1+del_p[3]) - (del_p[1]*del_p[2]))
    Warp_inv = np.array([[1,0,0],[0,1,0],[0,0,1]]) + np.concatenate((inv_war
    W_fin = np.matmul(W, Warp_inv)

    p[0] = W_fin[0,0] - 1
    p[1] = W_fin[1,0]
    p[2] = W_fin[0,1]
    p[3] = W_fin[1,1] - 1
    p[4] = W_fin[0,2]
    p[5] = W_fin[1,2]

    if np.linalg.norm(del_p) < threshold:
      break
    curr_itr += 1
  M = np.array([[1 + p[0], p[2], p[4]], [p[1], 1 + p[3], p[5]]])
  # raise NotImplementedError()

  return M
```

Q2.2

YOUR ANSWER HERE

As far as the compuutation speed is concerned, the Matthew-Baker Tracker is much better than either of the Lucas-Kanade (transaltion only, affine) algorithms. This is because in Matthew-Baker we are not calculating the the Hessian and Jacobian at every iteration. However, the accuracy in this case is compromised to a certain extent becuase the errors start to increase when the bounding boxes go out of the dimension and we are supposed to extrapolate the image.

Now, in case of translation only Lucas-Kanade (LK) algorithm, we could see that the performance was best as compared to the other two (affine LK, and Matthew-Baker). I think this is because the videos that we are considering here are mostly representing translation of the objects of interest. So, not considering the affine transformations eliminates any errors that could assimilate due to those parameters and not considering them is fine becuase of the kinds of videos we are using. However, in case of landing when the object of interest was moving closer/farther from the camera, the size of the object kept changing and the tanslation only LK could not take into factor that change in size. Although it could still track the patch in general.

For the affine LK, we can take into factor the changes in the sizes of the object of interest! However, when there is a sudden movement/change, the bounding boxe blows up. We can see this in the ballet video example. When the dancer moves her head suddenly, the bounding box expands suddenly. This happens because while doing gradient descent the slope(direction) of the gradient approaches zero or infinity. And hence the solver is not able to solve for such a jacobian.

Q3

YOUR ANSWER HERE

Q4 (Extra Credit) Short notes on important optical flow papers (15 points)

In this section we will go over three important optical flow papers and summarize them. For each paper, please follow these guidelines:

- Please read the papers in detail, focussing on the method described in the paper.
- For each paper, write 5 itemized points (6 max, 4 min) describing the method the authors use to solve the problem.
- Each point should have *no more* than *2 medium length sentences* and *a math equation*. You will **lose points for verbose descriptions**.
- By reading your summary, a person who is motivated about the optic flow problem and has the background on what optic flow is, should understand how the authors posed and solved the problem.
- You may add one point (not exceeding max 6 points) to mention something interesting about the results of the paper. E.g. how well does it generalize, how much real world data it needs etc.

Paper 1: GOTURN (5 Pts)

Learning to Track at 100 FPS with Deep Regression Networks. Held, Savarese and Thrun. ECCV'16

Additional material: A PyTorch implementation

Answer:

- From frame t-1, crop an image centered at c=(c_x, c_y), with height=k_1h, width=k_1w, where k1 determines context we want to be part of the target object to be tracked.
- From frame t, crop an image centered at c=(c_x, x_y), with height=k2h, width=k_2w, where k2 is usually 2, denoting the search space for the current time step.
- Pass these two images through two conv nets converting into two embeddings.
- Pass this embedding through an FCNN which predicts the center, width, and height of the bounding box for current frame.
- Interestingly, the model learns a generic relationship between object motion and appearance and can be used to track novel objects that do not appear in the training set.
- However, the model finds difficulty in dealing with occlusions and needs tuning for high speed moving objects.

Paper 2: RAFT (5 Pts)

RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. Teed and Deng ECCV'20

Additional material: Implementation, Talk (unofficial)

Answer:

- Pass a given a pair of consecutive RGB images I_1, I_2 through feature extractor g_\theta, such that resolution decreases 8 folds, and channels increases to D=256 (choosen by authors).
- Additional, pass I_1 through context network h_\theta to get additional features.
- Form correlation volume between image pairs by taking a dot product between feature output by g_\theta.
- Form a correlation pyramid by polling the last two dimensions by kernel sizes (1, 2, 4, 8)
- Using an LSTM + conv filters predict f_{k+1} = f_{k+1} + \Delta f, where \Delta f is predicted from the correlation volume, hidden state, and f_{k+1}. Initialize f_{k+1} to zero.
- For upscaling, take a convex combination of a 3x3 grid of coarse resolution neighbors.

Paper 3: GM Flow (5 Pts)

GMFlow: Learning Optical Flow via Global Matching. Xu et al. CVPR'22

Additional material: Implementation

Answer:

- This method might produce unreliable results in occluded regions.
-
-
-
-

Talk by Utsav Prabhu

1. Even though the concept of "fairness" is not new, but still it is a big and unsolved problem. Infact this field had seen a rapid explosion recently (2017).
2. People sometimes consider accuracy and fairness as mutually exclusive. However research has shown that we can both together! We can have high accuracy of a model along with considering the fairness of the model.
3. One of the major reasons for a bias being developed in a model, is due to the dataset it is being trained upon, since the datasets are biased themselves!!
4. Fairness is largely classified into "Equal treatement" and "Equal outcomes" of the models. These are further broken down into "Group fairness", "Individual Fairness" under "Equal outcome"; and "Counterfactual Fairness", "Interventional Fairness" under "Equal treatement". And currently a majority of research is being conducted in the field of Group fairness.

5. In the last part of the lecture Utsav touched upon how we can solve this issue using learning, and some tools that can be used for datasets compositional statics. He mentioned that although people have made some progress in solving this problem but still there are more unknown unknowns! And adversarial testing can reveal previously unknown failure modes and undesired system behaviors.

FiftyOne

This is just one single screenshot but there are >5 images that are incorrectly labled in this one screenshot itself. Like,

1. Ship - Frog
2. Airplane - Ship
3. Ship - Deer
4. Truck - Cat
5. Dog - Cat