Computer Vision

16720-B Fall 2021

16720 (B) Object Tracking in Videos - Assignment 6

```
Instructor: Kris                            TAs: Wen-Hsuan (Lead), Zen, Ya
n, Rawal, Paritosh, Qichen
```

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.

title

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 $\mathbf{x} = [u \;\; v]^T$ and maps pixel values from one place to another in an image $\mathbf{x}' = \left[u' \;\; v'\right]^T$. Simply put, $\mathbf{W}$ maps a pixel with coordinates $\mathbf{x} = [u \;\; v]^T$ to $\mathbf{x}' = \left[u' \;\; v'\right]^T$. 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 $\mathbf{p} = [p_1 \;\; p_2 \;\; p_3 \;\; p_4 \;\; p_5 \;\; p_6]'$.

Answer: For the point $\mathbf{x} = \left[x \quad y\right]^T$, the corresponding warped point is given by

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

Therefore, the derivative $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in terms of the warp parameters $\mathbf{p} = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6]'$ is given by

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial \mathbf{W}_x}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{W}_x}{\partial \mathbf{p}_2} & . & . & . & \frac{\partial \mathbf{W}_x}{\partial \mathbf{p}_6} \\ \frac{\partial \mathbf{W}_y}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{W}_y}{\partial \mathbf{p}_2} & . & . & . & \frac{\partial \mathbf{W}_y}{\partial \mathbf{p}_6} \end{bmatrix}$$

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

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.

Answer:

Firstly, m and n should be equal as $\mathbf{I}$ is generated by warping $\mathbf{T}$ and so, will have same number of pixels.

Going through one iteration of Lucas-Kanade tracking:

(1) Warp $\mathbf{I}$ with $\mathbf{W(x, p)}$ to compute $\mathbf{I(W(x, p))}$

- This takes $\mathbb{O}(np)$ as warping involves multiplying parameters with each pixel coordinates.

(2) Compute the error image $\mathbf{E} = \mathbf{T(x)} - \mathbf{I(W(x, p))}$

- This take $\mathbb{O}(n)$ as it's just finding elementwise difference

(3) Warp the gradient $\nabla \mathbf{I}$ with $\mathbf{W(x, p)}$

- This takes $\mathbb{O}(n)$ assuming that we already have the warped image else $\mathbb{O}(np)$

(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- This takes $\mathbb{O}(np)$ as this is computed per pixel and using each parameter.

(5) Compute the steepest descent images $\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- This takes $\mathbb{O}(np)$ as again this involves manipulation of a n x p matrix.

(6) Compute the Hessian matrix - $\mathbf{H} = \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- This Hessian computation takes $\mathbb{O}(np^2)$ due to the above matrix multiplication - multiplying p x n matrix by n x p matrix can be done fastest in $\mathbb{O}(pnp)$

(7) Compute $\left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \mathbf{E}$

- This takes $\mathbb{O}(np)$ due to the above matrix multiplication - multiplying p x n matrix by n x 1 matrix can be done fastest in $\mathbb{O}(pn)$

(8) Compute $\nabla \mathbf{p} = H^{-1} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \mathbf{E}$

- Inverting the Hessian takes $\mathbb{O}(p^3)$ while multiplying it with the rest would need an additional $\mathbb{O}(p^2)$ - multiplying p x p matrix by p x 1 matrix can be done fastest in $\mathbb{O}(p^2)$

(9) Update the parameters $\mathbf{p}$

- This takes $\mathbb{O}(p)$ as it's a simple update by addition per parameter.

Therefore, the overall time complexity for one iteration of Lucas-Kanade tracking is $\mathbb{O}(p^3 + np^2)$

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?

Answer:

Firstly, m and n should be equal as $\mathbf{I}$ is generated by warping $\mathbf{T}$ and so, will have same number of pixels.

Going through the precomputation of Matthews-Baker tracking:

(1) Evaluate the gradient $\nabla \mathbf{T}$ of the template $\mathbf{T}(\mathbf{x})$

- This takes $\mathbb{O}(n)$ as it would involve iterating over pixels.

(2) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}, \mathbf{0})$

- This takes $\mathbb{O}(np)$ as this is computed per pixel and using each parameter.

(3) Compute the steepest descent images $\nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- This takes $\mathbb{O}(np)$ as again this involves manipulation of a n x p matrix.

(4) Compute the Hessian matrix - $\mathbf{H} = \left[\nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- This Hessian computation takes $\mathbb{O}(np^2)$ due to the above matrix multiplication - multiplying p x n matrix by n x p matrix can be done fastest in $\mathbb{O}(pnp)$

Going through one iteration of Matthews-Baker tracking:

(5) Warp $\mathbf{I}$ with $\mathbf{E} = \mathbf{W}(\mathbf{x}, \mathbf{p})$ to compute $\mathbf{I}(\mathbf{W}(\mathbf{x}, \mathbf{p}))$

- This takes $\mathbb{O}(n)$ assuming that we already have the warped image else $\mathbb{O}(np)$

(6) Compute the error image $\mathbf{E} = \mathbf{I}(\mathbf{W}(\mathbf{x}, \mathbf{p})) - \mathbf{T}(\mathbf{x})$

- This take $\mathbb{O}(n)$ as it's just finding elementwise difference

(7) Compute $\left[\nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \mathbf{E}$

- This takes $\mathbb{O}(np)$ due to the above matrix multiplication - multiplying p x n matrix by n x 1 matrix can be done fastest in $\mathbb{O}(pn)$

(8) Compute $\nabla \mathbf{p} = H^{-1}\left[\nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \mathbf{E}$

- Inverting the Hessian takes $\mathbb{O}(p^3)$ while multiplying it with the rest would need an additional $\mathbb{O}(p^2)$ - multiplying p x p matrix by p x 1 matrix can be done fastest in $\mathbb{O}(p^2)$

(9) Update the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) = \mathbf{W}(\mathbf{x};\mathbf{p}) \cdot \mathbf{W}(\mathbf{x};\nabla\mathbf{p})^{-1}$

- This takes $\mathbb{O}(^2)$

Overall, the precomputation has an overall complexity of $\mathbb{O}(np^2)$ while that for one iteration of Matthews-Baker tracking is $\mathbb{O}(p^3 + np)$

As can be seen the, Matthews-Baker tracking is faster with a smaller time complexity ($\mathbb{O}(p^3 + np)$) as compared to Lucas-Kanade tracking ($\mathbb{O}(p^3 + np^2)$). While the precomputation in Matthews-Baker tracking has a time complexity of $\mathbb{O}(np^2)$, this part only runs once.

So, say for k iterations, Lucas-Kanade traccking would have a time complexity of $\mathbb{O}(kp^3 + knp^2)$ while Matthews-Baker tracking would have $\mathbb{O}(kp^3 + knp) + \mathbb{O}(np^2)$. From this, it can clearly be seen that as the nuber of iterations increases, Matthews-Baker runs way faster than Lucas-Kanade.

Coding Questions Write-up

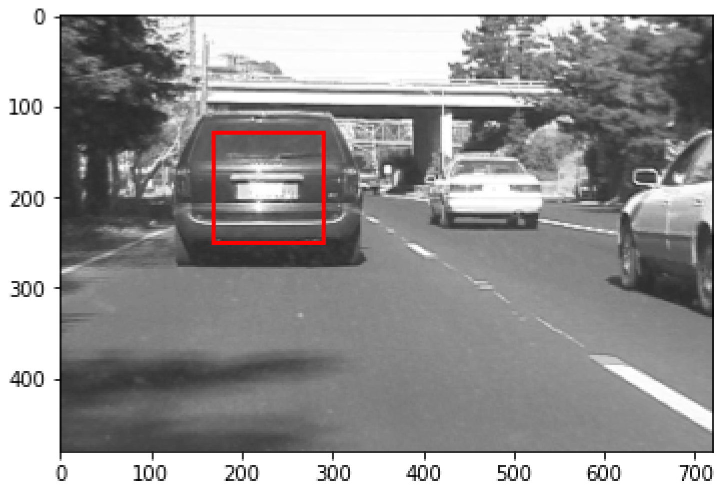
Q1.1

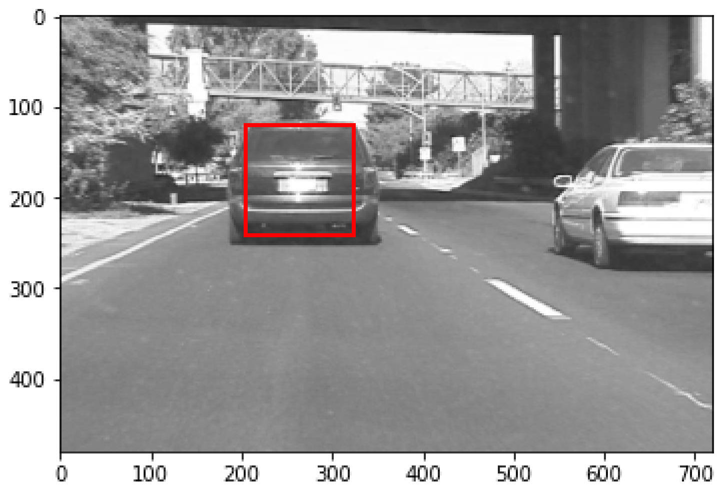
In [ ]:
```python
def LucasKanade(it, it1, rect, thresh=.01, maxIters=100):

    '''
    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
        maxIters: Maximum number of iterations to run

      Outputs:
        p: movement vector dx, dy
    '''

    # Set thresholds (you probably want to play around with the values)
    p = np.zeros(2) # dx, dy
    x1, y1, x2, y2 = rect

    inter_it = RectBivariateSpline(np.arange(it.shape[0]), np.arange(it.shape[1])
    inter_it1 = RectBivariateSpline(np.arange(it1.shape[0]), np.arange(it1.shape

    x0, y0 = np.meshgrid(np.arange(x1, x2 + 0.5), np.arange(y1, y2 + 0.5))
    x0 = x0.flatten()
    y0 = y0.flatten()
    T = inter_it.ev(y0, x0)

    for i in range(maxIters):

        x = x0 + p[0]
        y = y0 + p[1]

        I = inter_it1.ev(y, x)

        # Calculating A
        Ix = inter_it1.ev(y, x, dx=0, dy=1).reshape(-1, 1)
        Iy = inter_it1.ev(y, x, dx=1, dy=0).reshape(-1, 1)
        dI = np.hstack((Ix, Iy))
        dW_dp = np.eye(2)
        A = dI @ dW_dp

        # Calculating b
        b = T - I

        dp = np.linalg.lstsq(A, b, rcond=None)[0]
        p = p + dp

        if np.sqrt(np.sum(dp ** 2)) <= thresh:
            break

    return p
```

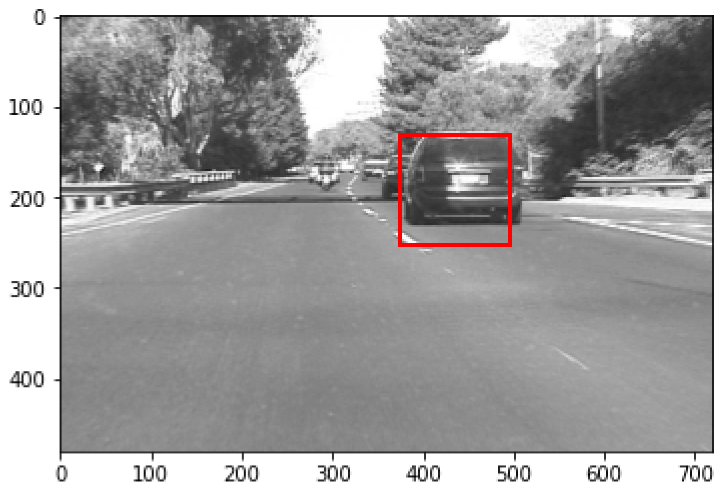
Results:

For each dataset, the first, middle and last frames have been shared

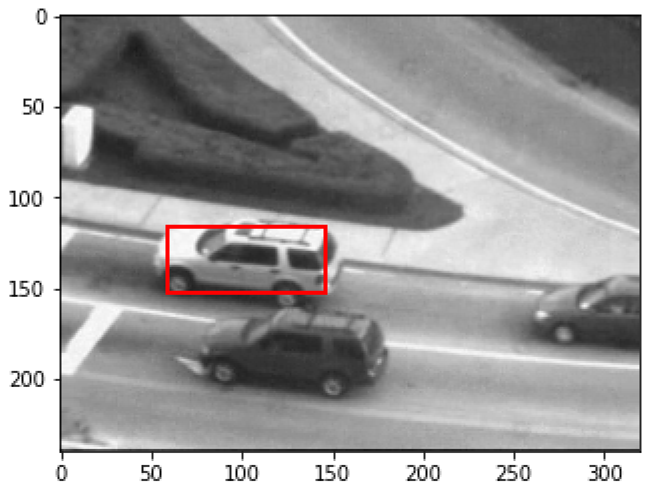
1 - car1

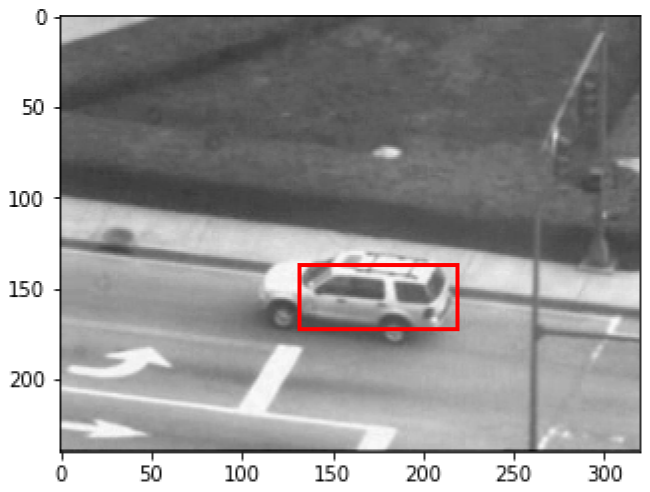

0
100
200
300
400
0
100
200
300
400
500
600
700

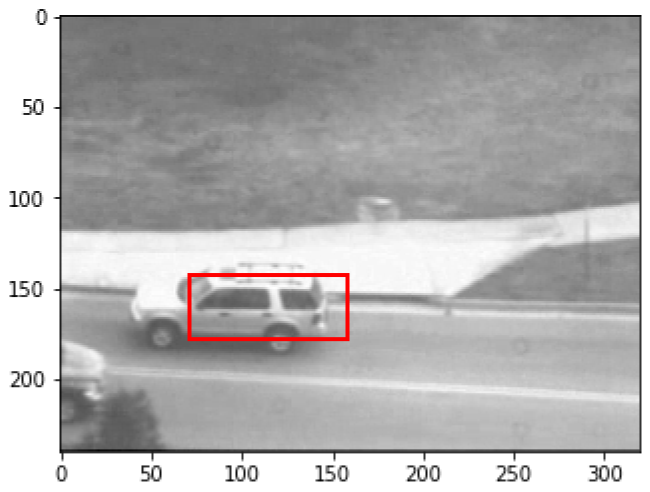

0
100
200
300
400
0
100
200
300
400
500
600
700


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

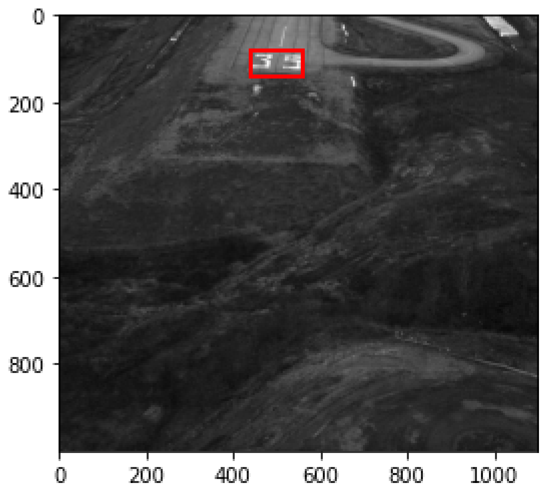
2 - car2

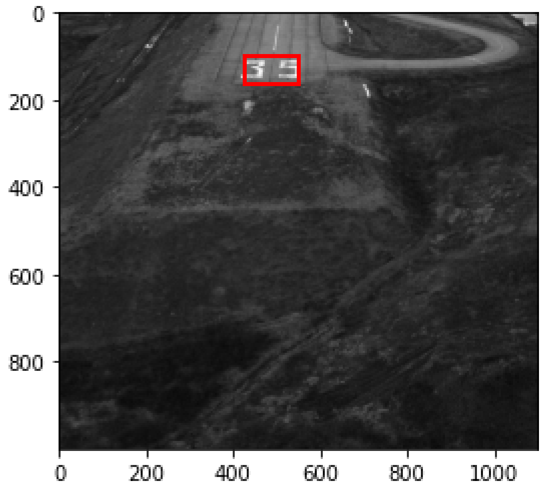


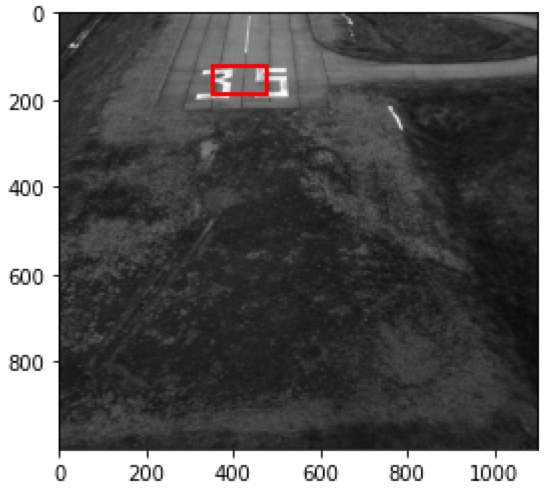




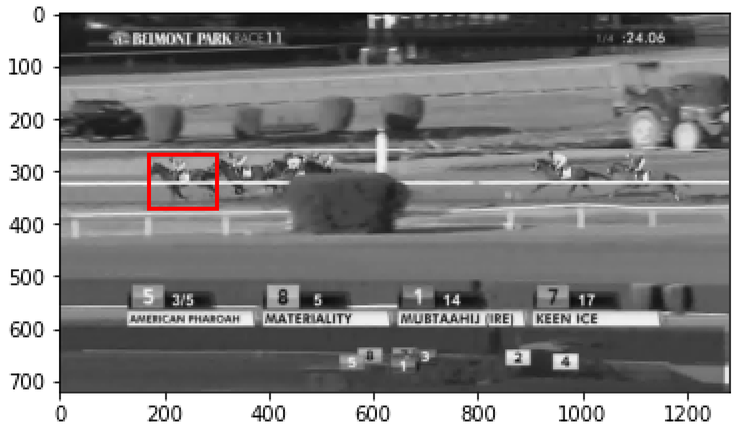
3 - landing

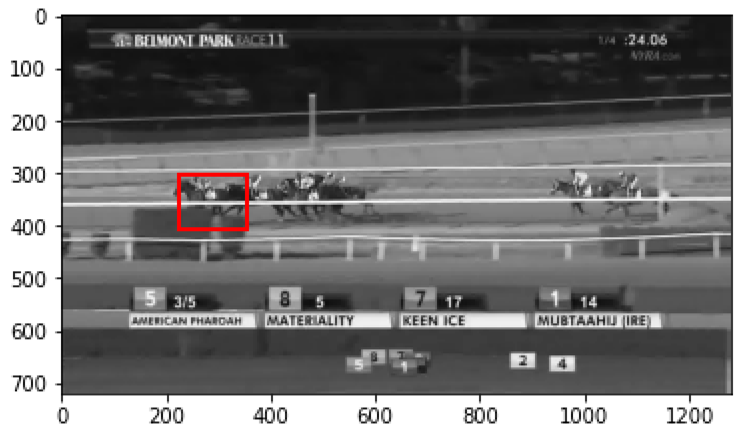

3 5

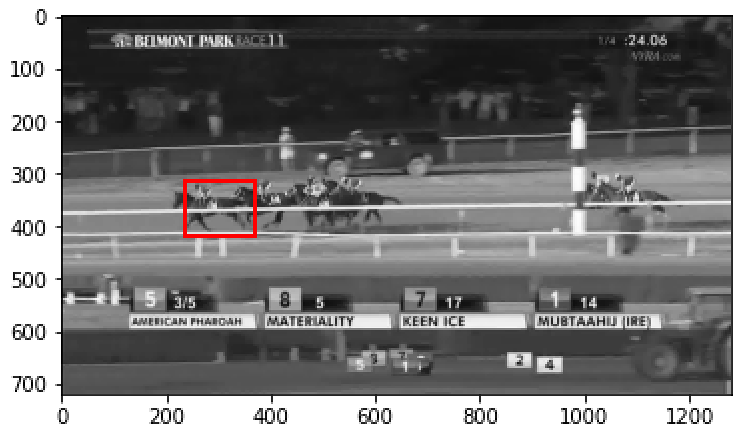

3 5


3 5

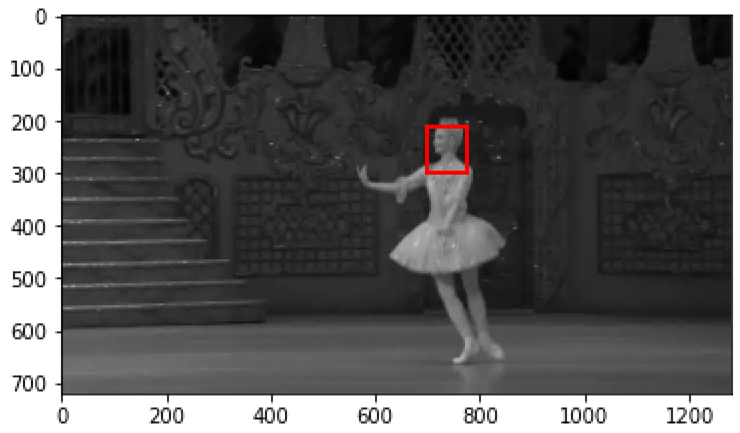
4 - race

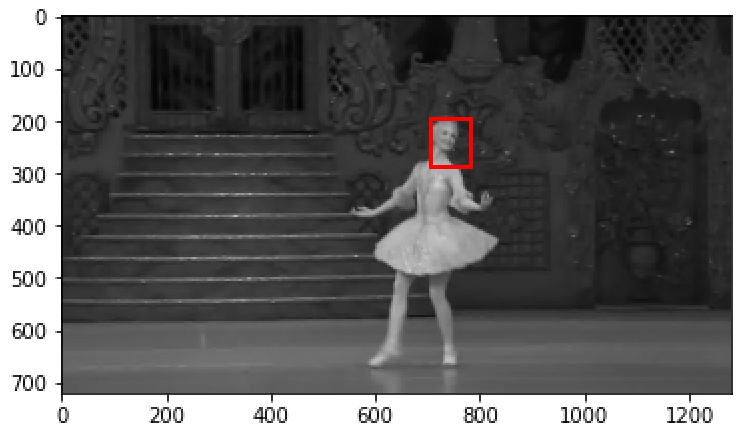

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
8 5
1 14
7 17
AMERICAN PHAROAH
MATERIALITY
MUBTAAHIJ (IRE)
KEEN ICE

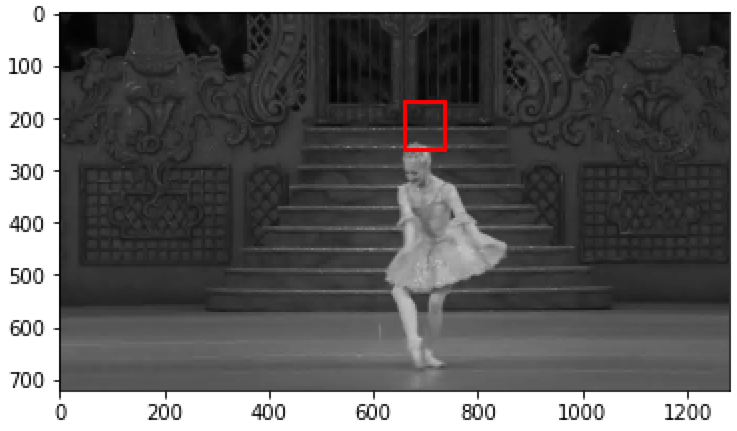

BELMONT PARK RACE 11
1/4 :24.06
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)


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

5 - ballet







Q1.2

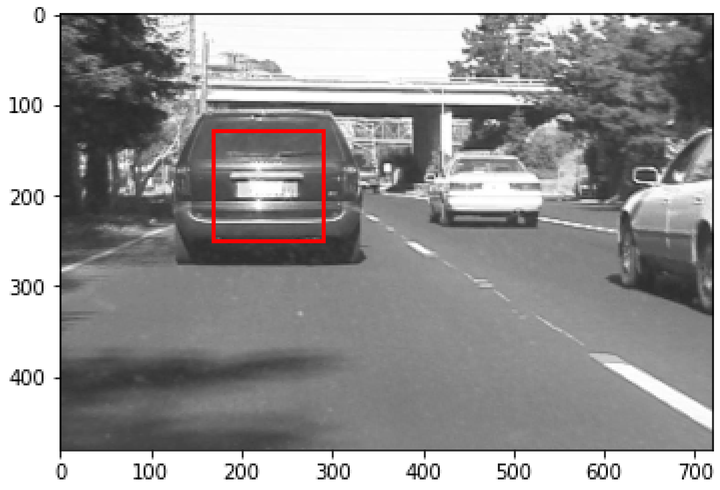
```
In [ ]: def LucasKanadeAffine(it, it1, rect, thresh=.025, maxIters=100):
            '''
            Q1.2: Lucas-Kanade Forward Additive Alignment with Affine MAtrix

              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
                maxIters: Maximum number of iterations to run

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

        #     M = np.zeros((2, 3))
            M = np.hstack((np.eye(2), np.zeros(2).reshape(-1, 1)))
            x1, y1, x2, y2 = rect

            inter_it = RectBivariateSpline(np.arange(it.shape[0]), np.arange(it.shape[1])
            inter_it1 = RectBivariateSpline(np.arange(it1.shape[0]), np.arange(it1.shape[

            x0, y0 = np.meshgrid(np.arange(x1, x2 + 0.5), np.arange(y1, y2 + 0.5))
            x0 = x0.flatten()
            y0 = y0.flatten()

            T = inter_it.ev(y0, x0)
            coords0 = np.hstack((x0.reshape(-1, 1), y0.reshape(-1, 1)))

            for i in range(maxIters):

                coords = M @ (np.hstack((coords0, np.ones(coords0.shape[0]).reshape(-1, 1

                x = coords[0].flatten()
                y = coords[1].flatten()

                I = inter_it1.ev(y, x)

                # Calculating A
                Ix = inter_it1.ev(y, x, dx=0, dy=1)
                Iy = inter_it1.ev(y, x, dx=1, dy=0)

                # A = [x.Ix, x.Iy, y.Ix, y.Iy, Ix, Iy]
                A = np.zeros((x.shape[0], 6))

                A[:, 0] = x * Ix
                A[:, 1] = x * Iy
                A[:, 2] = y * Ix
                A[:, 3] = y * Iy
                A[:, 4] = Ix
                A[:, 5] = Iy

                # Calculating b
                b = T - I

                dp = np.linalg.lstsq(A, b, rcond=None)[0]
                M = M + dp.reshape(np.flip(M.shape)).T

                if np.sqrt(np.sum(dp ** 2)) <= thresh:
                    break
```

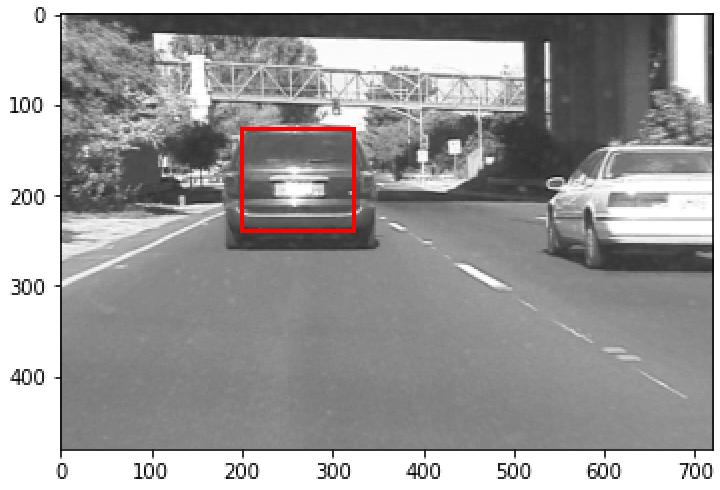
```
    return M
```

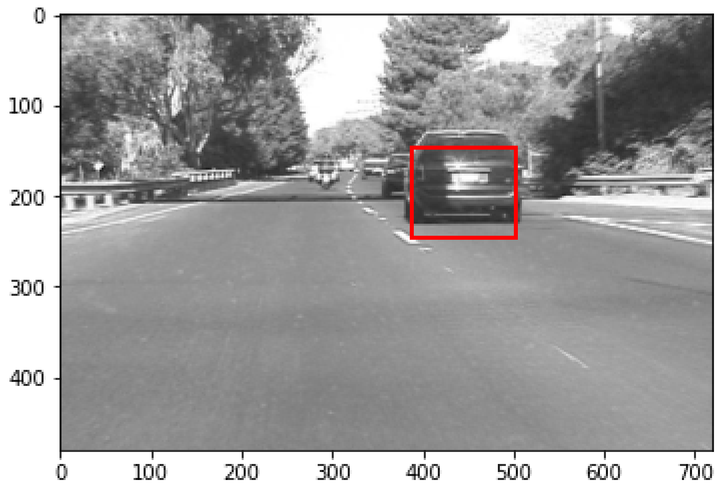
Results:

For each dataset, the first, middle and last frames have been shared

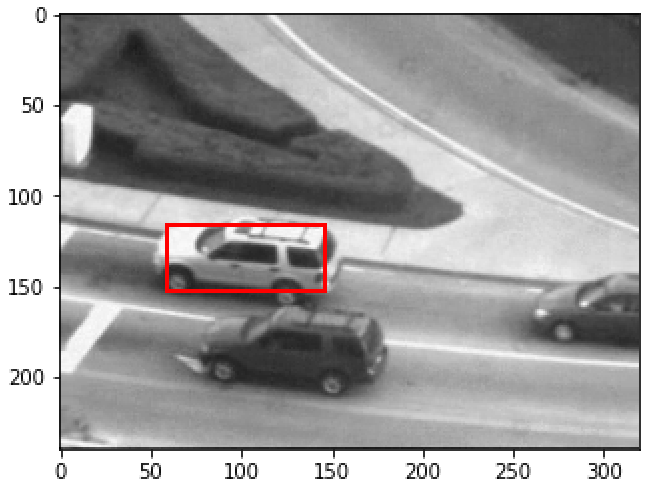
1 - car1

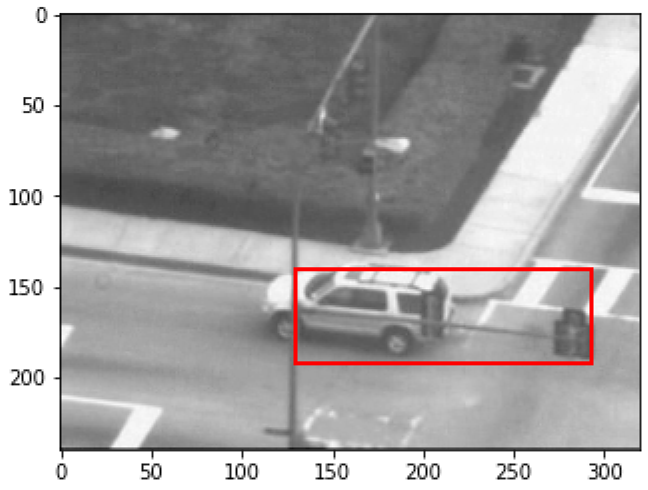




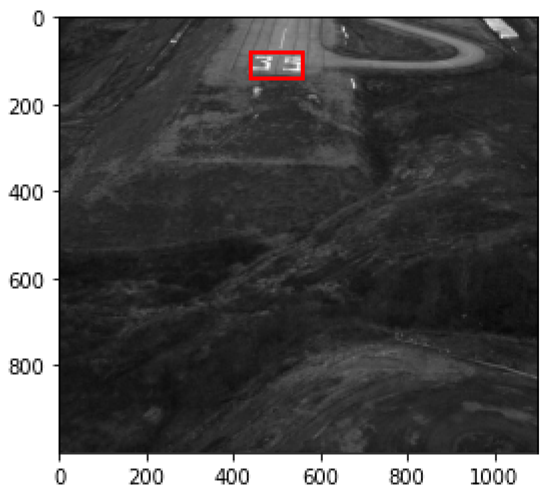


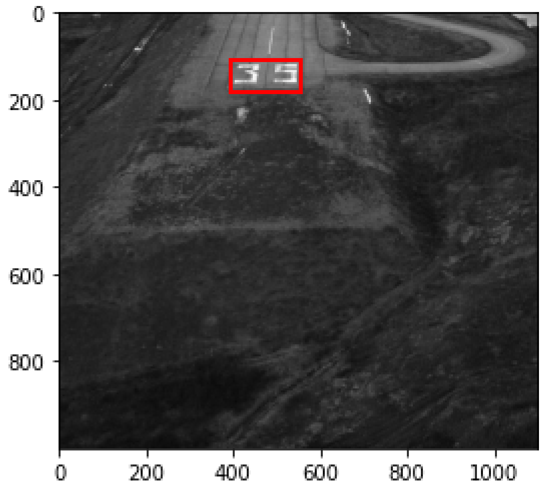
2 - car2

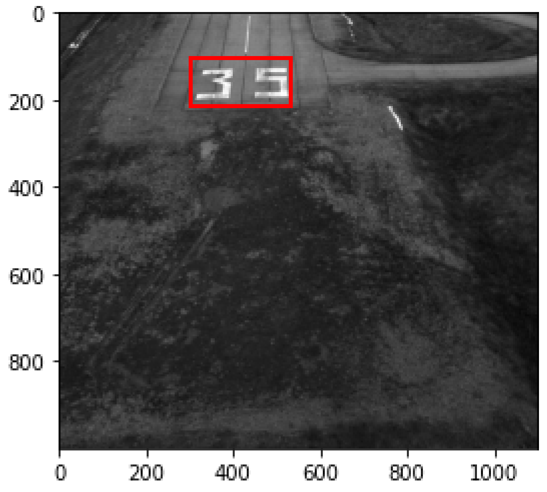




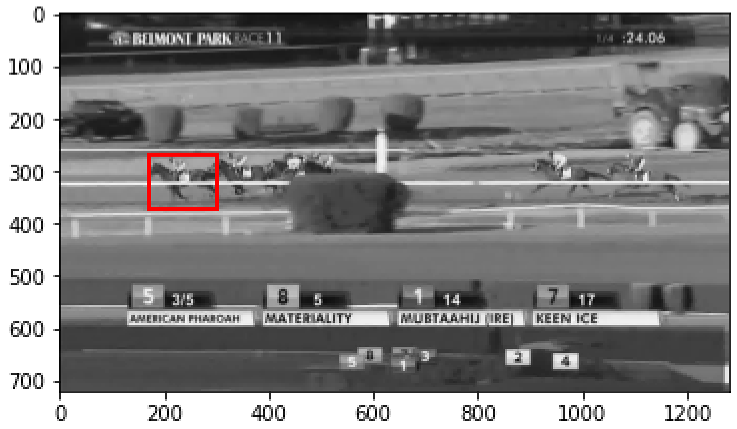
3 - landing

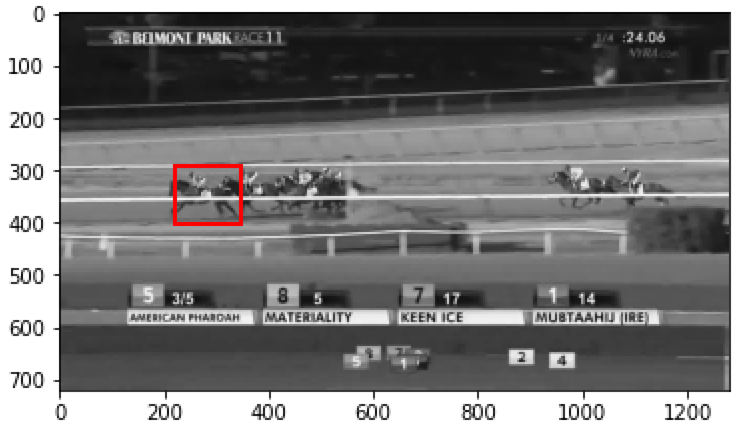




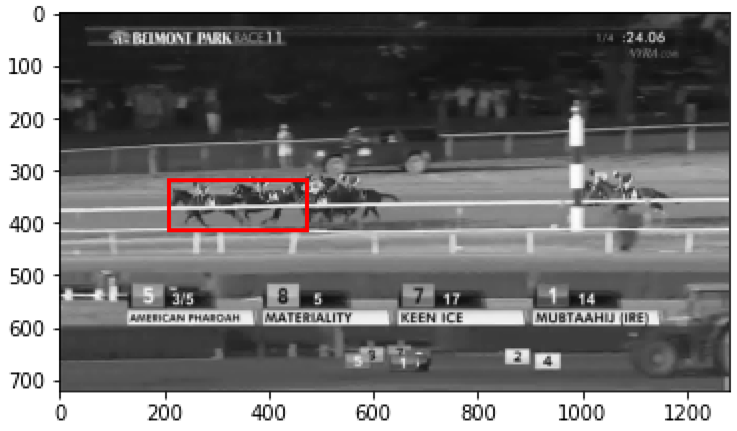


4 - race


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

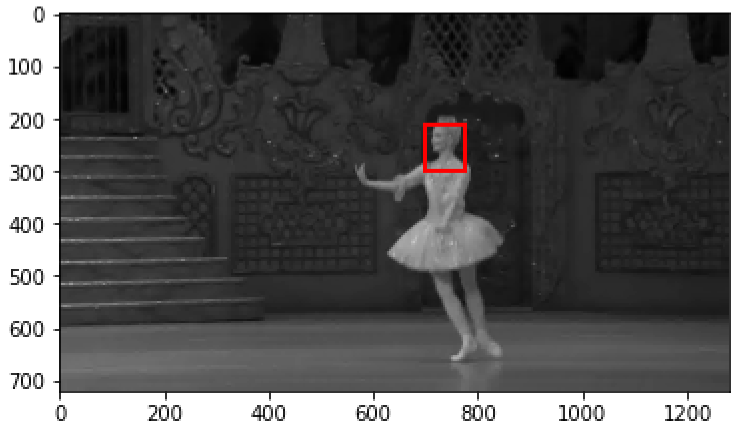


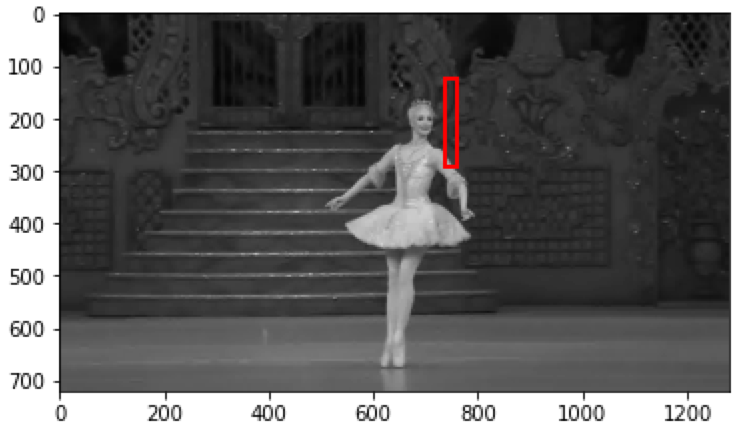
BELMONT PARK RACE 11
1/4 :24.06
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)

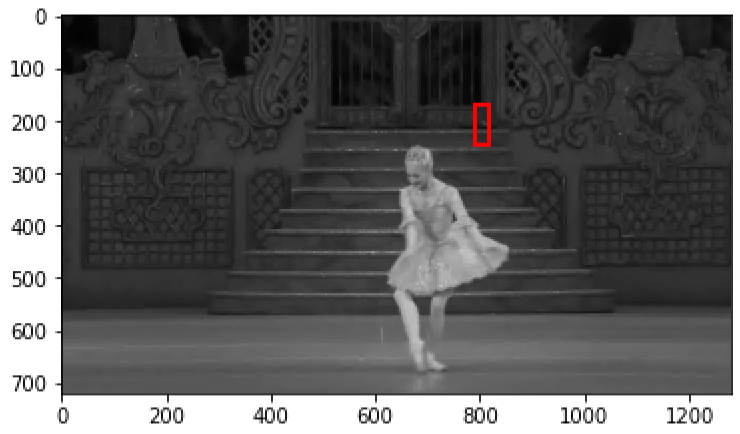


BELMONT PARK RACE 11
1/4 :24.06
5 3/5 AMERICAN PHAROAH
8 5 MATERIALITY
7 17 KEEN ICE
1 14 MUBTAAHIJ (IRE)

5 - ballet







Q2.1

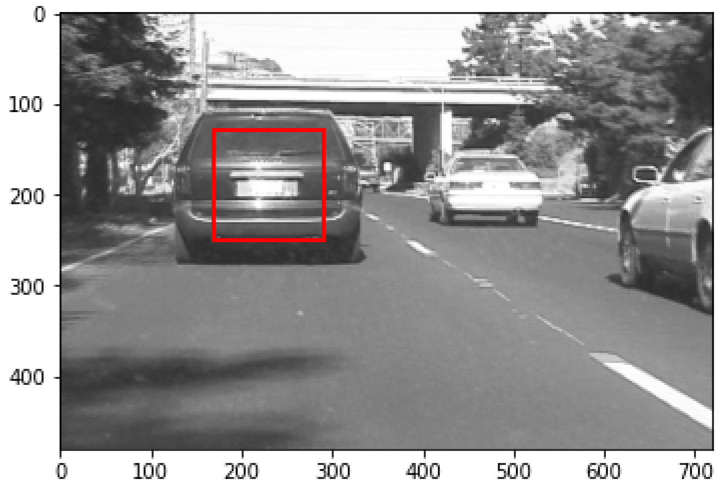
```
In [ ]: def InverseCompositionAffine(it, it1, rect, thresh=.01, maxIters=100):
            '''
            Q2.1: Matthew-Bakers Inverse Compositional Alignment with Affine MAtrix

              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.eye(3)
            p = np.zeros((3, 3))
            x1, y1, x2, y2 = rect

            if x2 < x1 or y2 < y1:
                return M[: 2]

            inter_it = RectBivariateSpline(np.arange(it.shape[0]), np.arange(it.shape[1])
            inter_it1 = RectBivariateSpline(np.arange(it1.shape[0]), np.arange(it1.shape

            x0, y0 = np.meshgrid(np.arange(x1, x2 + 0.1), np.arange(y1, y2 + 0.1))
            x0 = x0.flatten()
            y0 = y0.flatten()

            T = inter_it.ev(y0, x0)
            coords0 = np.hstack((x0.reshape(-1, 1), y0.reshape(-1, 1)))

            Tx = inter_it.ev(y0, x0, dx=0, dy=1)
            Ty = inter_it.ev(y0, x0, dx=1, dy=0)

            # A = [x.Ix, x.Iy, y.Ix, y.Iy, Ix, Iy]
            A = np.zeros((x0.shape[0], 6))

            A[:, 0] = x0 * Tx
            A[:, 1] = y0 * Tx
            A[:, 2] = Tx
            A[:, 3] = x0 * Ty
            A[:, 4] = y0 * Ty
            A[:, 5] = Ty

            for i in range(maxIters):

                coords = M @ (np.hstack((coords0, np.ones(coords0.shape[0]).reshape(-1,

                x = coords[0].flatten()
                y = coords[1].flatten()

                I = inter_it1.ev(y, x)

                # Calculating b
                b = I - T

                dp = np.linalg.lstsq(A, b, rcond=None)[0]
                dp = dp.reshape(2, 3)
```

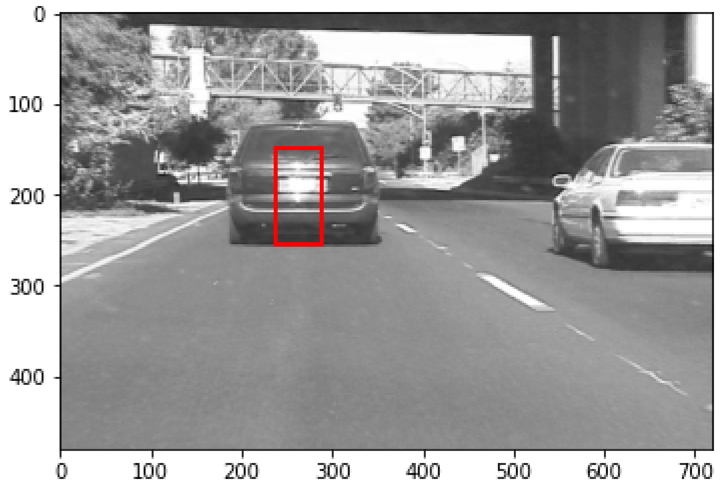
```
        dM = np.eye(3)
        dM = dM + np.vstack((dp, np.array([0, 0, 0])))
        M = M @ np.linalg.pinv(dM)

        if np.linalg.norm(dp) <= thresh:
            break

    return M[: -1]
```

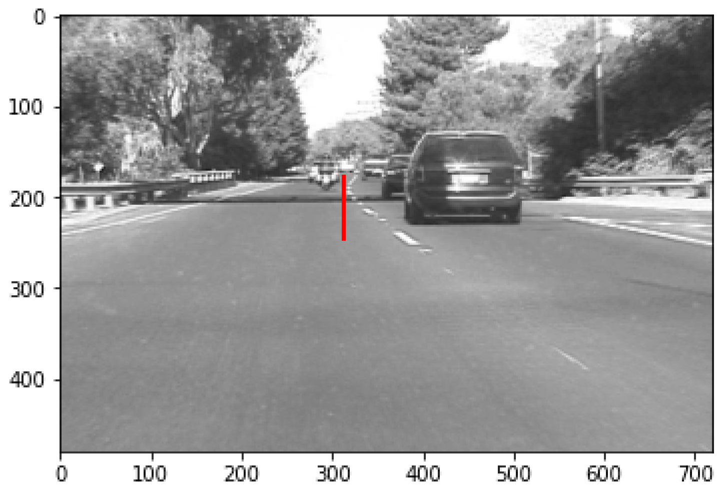
Results:

For each dataset, the first, middle and last frames have been shared

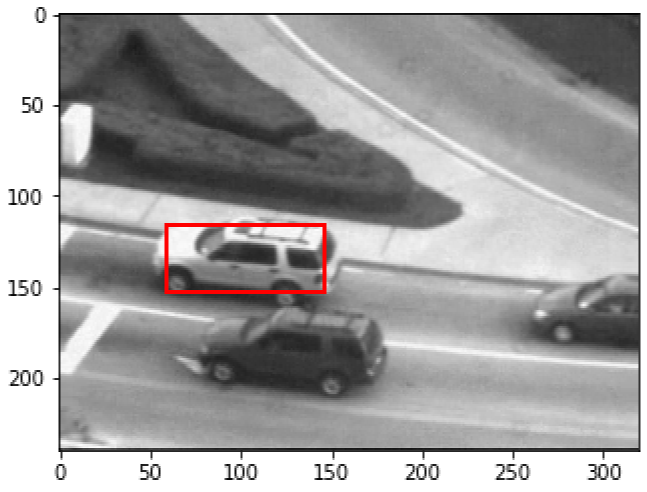
1 - car1



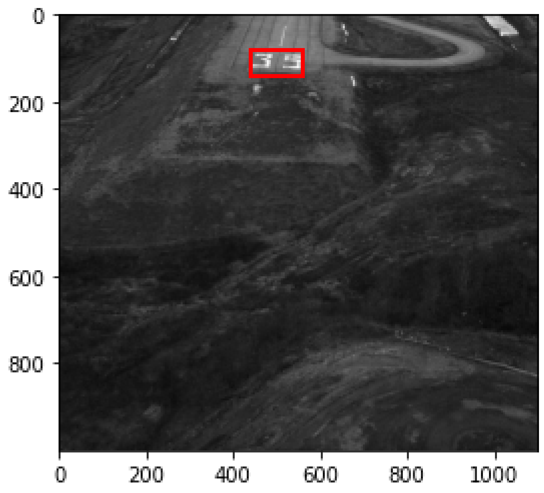


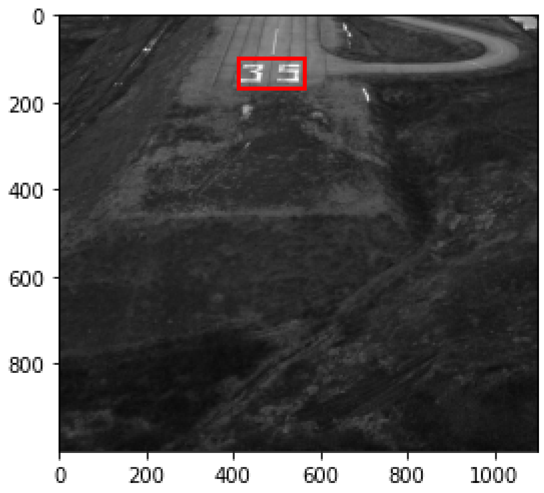


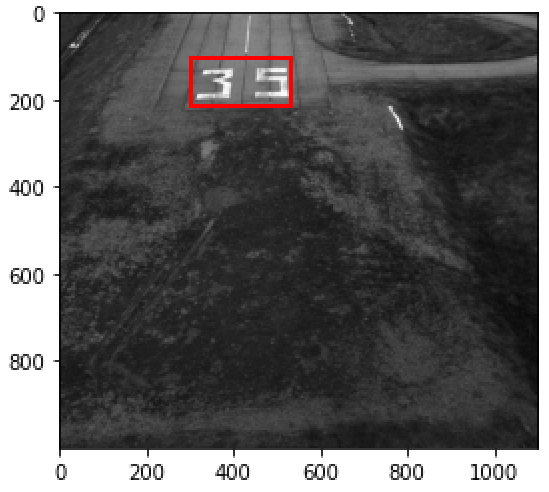
2 - car2



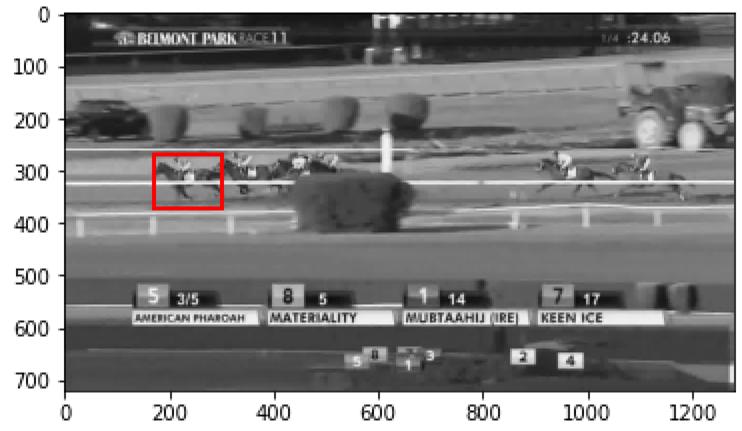
3 - landing

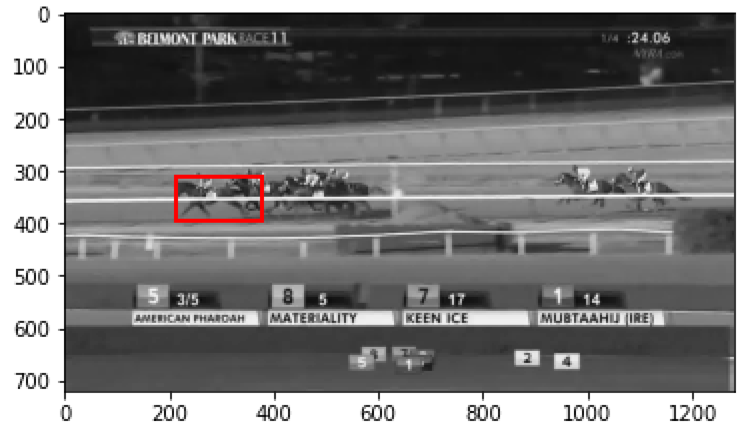


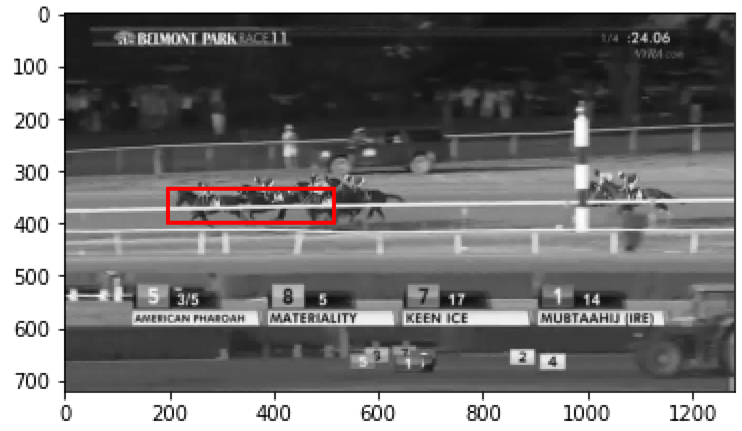




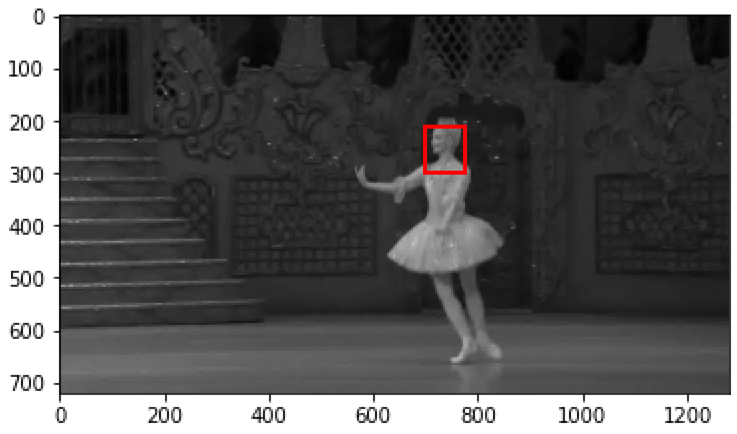
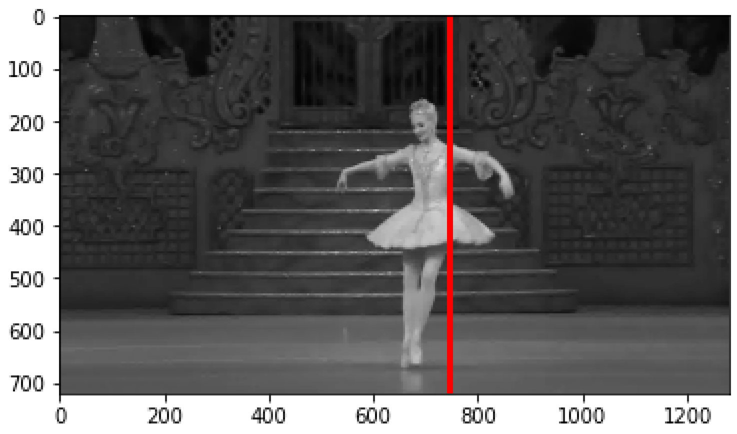
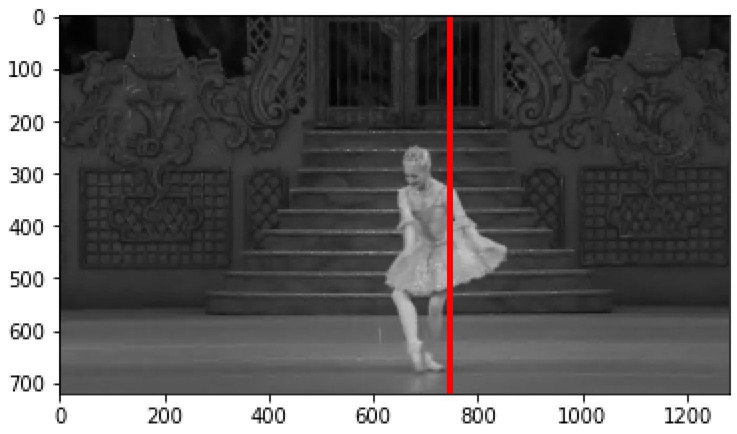
4 - race







5 - ballet

Comparing performance for every dataset

1 - Car1 - The Lucas-Kanade Affine matrix algorithm outperforms the Matthews-Baker tracking. Matthews-Baker algorithm messes up the tracking once the car goes into the shadows.

2 - Car2 - Both Lucas-Kanade Affine and Matthews-Baker tracking seem to perform bad on this dataset. This is mostly because the scaling component of the affine filter learned gets larger and larger when the traffic sign enters the scene as that too was being considered a part of the car. Since this wasn't possible with the Lucas-Kanade translation only implementation, it didnt face this issue.

3 - Landing - Both Lucas-Kanade Affine and Matthews-Baker tracking perform well on this dataset. Since the translation only implementation doesn't have any scaling factor involved, it can't get the right bounding box.

4 - Race - Both Lucas-Kanade Affine and Matthews-Baker tracking have similar performance. The bounding box contains the horse being tracked but also includes part of another horse.

5 - Ballet - Matthews-Baker performs bad on this dataset by again having a completely outstretched bounding box. Lucas-Kanade affine works poorly as well when it comes to tracking.

The lucas-kanade translation only implementation doesn't have any scale factors associated and thus, cant track objects efficiently as their size would change as they would move towards or away from the camera. However, the affine implementation and Matthews-Baker tracking both have this fixed as they find multiple parameters to form the affine filter matrix. Secondly, Matthews-Baker is much faster than Lucas-Kanade as it only needs to compute the Hessian once unlike Lucas-Kanade wherein the Hessian needs to be computed in every iteration.

However, all these algorithms only work in case few conditions are satisfied. 1 - Brightness of points remains constant over time and, 2 - The displacement along x and y between 2 consecutive frames is very small. So, when these conditions are not met - car1 having drastic brightness changes when the car emerges from beneath the shadows and ballet having some moments of drastic movement, the algorithms break.

Q2.2

Comparing performance for every dataset

1 - Car1 - The Lucas-Kanade Affine matrix algorithm outperforms the Matthews-Baker tracking. Matthews-Baker algorithm messes up the tracking once the car goes into the shadows.

2 - Car2 - Both Lucas-Kanade Affine and Matthews-Baker tracking seem to perform bad on this dataset. This is mostly because the scaling component of the affine filter learned gets larger and larger when the traffic sign enters the scene as that too was being considered a part of the car. Since this wasn't possible with the Lucas-Kanade translation only implementation, it didnt face this issue.

3 - Landing - Both Lucas-Kanade Affine and Matthews-Baker tracking perform well on this dataset. Since the translation only implementation doesn't have any scaling factor involved, it can't get the right bounding box.

4 - Race - Both Lucas-Kanade Affine and Matthews-Baker tracking have similar performance. The bounding box contains the horse being tracked but also includes part of another horse.

5 - Ballet - Matthews-Baker performs bad on this dataset by again having a completely outstretched bounding box. Lucas-Kanade affine works poorly as well when it comes to tracking.

The lucas-kanade translation only implementation doesn't have any scale factors associated and thus, cant track objects efficiently as their size would change as they would move towards or away from the camera. However, the affine implementation and Matthews-Baker tracking both have this fixed as they find multiple parameters to form the affine filter matrix. Secondly, Matthews-Baker is much faster than Lucas-Kanade as it only needs to compute the Hessian once unlike Lucas-Kanade wherein the Hessian needs to be computed in every iteration.

However, all these algorithms only work in case few conditions are satisfied. 1 - Brightness of points remains constant over time and, 2 - The displacement along x and y between 2 consecutive frames is very small. So, when these conditions are not met - car1 having drastic brightness changes when the car emerges from beneath the shadows and ballet having some moments of drastic movement, the algorithms break.

Q3

YOUR ANSWER HERE