FINAL-TERM ASSIGNMENT REPORT

INTELLIGENT SIGNAL PROCESSING COURSEWORK EXERCISE 1

Contents

Task 1

```python
video_cap = cv2.VideoCapture('Traffic_Laramie_1.mp4')
bg_sub = cv2.createBackgroundSubtractorMOG2(history=12000, varThreshold=20, detectShadows = False)
if not video_cap.isOpened():
    print("Failed to open file")
    exit()
```

To detect objects as required in task 1, we will be using the opencv library “cv2”. We will load the first video and create the background subtractor object which we will use to detect objects. The background subtractor will compare the current frame to the learned background frame to detect objects in the frame.

```python
ret, frame = video_cap.read()

# define area to detect cars
# height: 260 to 600, width: 0 to 1040 of the frame
detection_area = frame[260:600,0:1040]

# subtract background and detect objects
fgmask = bg_sub.apply(detection_area)
_, fgmask = cv2.threshold(fgmask, 254, 255, cv2.THRESH_BINARY)

contours, _ = cv2.findContours(fgmask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Next, as we read each frame in the video, we will define the detection area of each of those frames and apply the background subtractor to those frames. We will be using the default parameters as we do so. After applying the background subtraction, we will convert the frame to a greyscale format by using cv2.threshold. We can then find all the contours (detected objects) in the frame using the findContours() function.

```python
for cnt in contours:
    area = cv2.contourArea(cnt)
    # greater than 4700 so that only contours for vehicles are tracked
    # groups of people walking together or cycling are found to be smaller than 4700 in this cases
    x, y, w, h = cv2.boundingRect(cnt)
```

Next, we will iterate through the contours, calculate each of those contour area and retrieve the x, y, width and height value of the contour.

```python
elif area > 4700:
    # get centroid of contour
    cx = int(x + (x + w) / 2)
    cy = int(y + (y + h) / 2)
    # append centroid info into list
    current_frame_centroids.append((cx, cy))
    # draw rectangle for car
    cv2.rectangle(detection_area, (x,y), (x + w, y + h), (0, 255, 0), 1)
```

We will then check if the area is larger than 4700, before drawing the rectangle.

With this, we can simply include cv2.imshow(“Frame”, frame) to display the video with car detection.

Task 2

To track objects for task 2, we will perform the following steps:

Step 1: Retrieve bounding box coordinates and calculate the centroids for each of them:

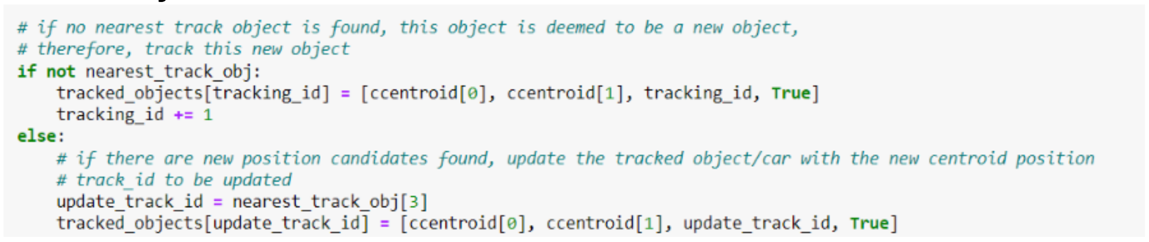
```python
for cnt in contours:
    area = cv2.contourArea(cnt)
    # greater than 4700 so that only contours for vehicles are tracked
    # groups of people walking together or cycling are found to be smaller than 4700 in this cases
    x, y, w, h = cv2.boundingRect(cnt)
    # it is found that overlapping vehicles from x value 0 to 520 and y greater than 130, have areas larger than 13500
    if x < 520 and area > 13500 and h > 130:
        # if contours overlap vertically
        # split and get centroid of both contours of overlapping cars
        # top car
        cx1 = int(x + (x + w) / 2)
        cy1 = int(y + (y + h/2) / 2)
        # bottom car
        cx2 = int(x + (x + w) / 2)
        cy2 = int(y + h/2 + (y + h/2) / 2)

        # append centroid for both cars into centroids list
        current_frame_centroids.append((cx1, cy1))
        current_frame_centroids.append((cx2, cy2))

        # draw rectangle for both cars
        cv2.rectangle(detection_area, (x,y), (x + w, y + int(h/2)), (0, 255, 0), 1)
        cv2.rectangle(detection_area, (x,y + int(h/2)), (x + w, y + h), (0, 255, 0), 1)

    # elif no overlaps
    elif area > 4700:
```

After getting the coordinates of each bounding rectangle, we will check for overlapping cars which I have defined to have an area of greater than 13500 as compared to 4700 for a single car.

Step 2: Compute distance(Euclidean) between tracked centroids with current frame centroids:

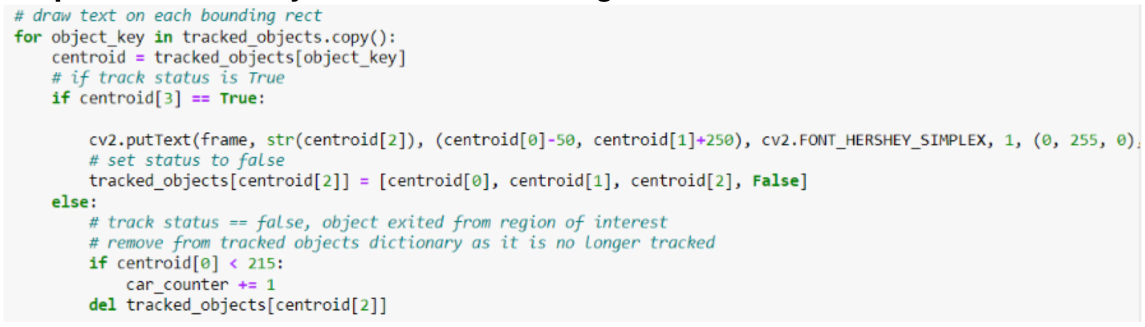
```python
# assign or update tracking id to centroid
for ccentroid in current_frame_centroids:

    # we will be using this list to store centroid information in the following index format:
    # [0] -> centroid x value, [1] -> centroid y value, [2] -> distance, [3] -> tracking_id
    nearest_track_obj = []

    # iterate through tracked objects and find closest object
    # this loop will only run when there are objects added into the list of tracked_objects
    for object_key in tracked_objects:
        # retrieve tracked_obj using the key
        tracked_obj = tracked_objects[object_key]
        # for each iteration, calculate euclidean dist and find closest tracked object to centroid
        # sqrt( (x1 - x2)**2 + (y1 - y2)**2 )
        distance = math.sqrt( (tracked_obj[0]-ccentroid[0])**2 + (tracked_obj[1]-ccentroid[1])**2 )

        # if distance between the centroid and tracked object is smaller than 100,
        # they are likely to be the same object/car
        # this value may actually be smaller but for the purpose of this assignment,
        # 100 is alright as it still works as intended
        if distance < 100:
            # if there is already a new position candidate(nearest_track_obj != []), check if distance is shorter
            # replace nearest_track_obj if conditions met
            if nearest_track_obj != []:
                # if nearest_track_obj distance is larger than new distance (current distance is shorter)
                if nearest_track_obj[2] > distance:
                    # replace with current centroid
                    nearest_track_obj = [ccentroid[0], ccentroid[1], distance, tracked_obj[2]]
            else:
                # if no candidate centroid found previously, assign it to nearest_track_obj
                nearest_track_obj = [ccentroid[0], ccentroid[1], distance, tracked_obj[2]]
        # if distance is >= 100, this tracked object is deemed to have become non visible from the frame
```

Step 3: Update coordinates of tracked objects and track new objects that are not already tracked:



```python
# if no nearest track object is found, this object is deemed to be a new object,
# therefore, track this new object
if not nearest_track_obj:
    tracked_objects[tracking_id] = [ccentroid[0], ccentroid[1], tracking_id, True]
    tracking_id += 1
else:
    # if there are new position candidates found, update the tracked object/car with the new centroid position
    # track_id to be updated
    update_track_id = nearest_track_obj[3]
    tracked_objects[update_track_id] = [ccentroid[0], ccentroid[1], update_track_id, True]
```

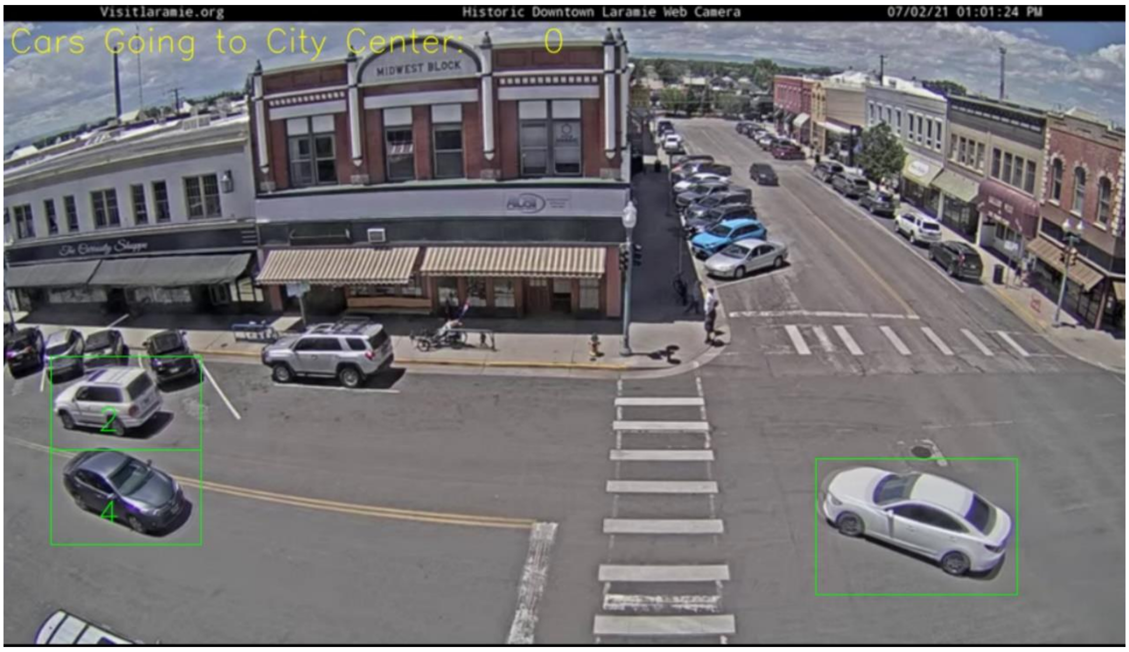
Step 4: Remove objects that are no longer in the frame

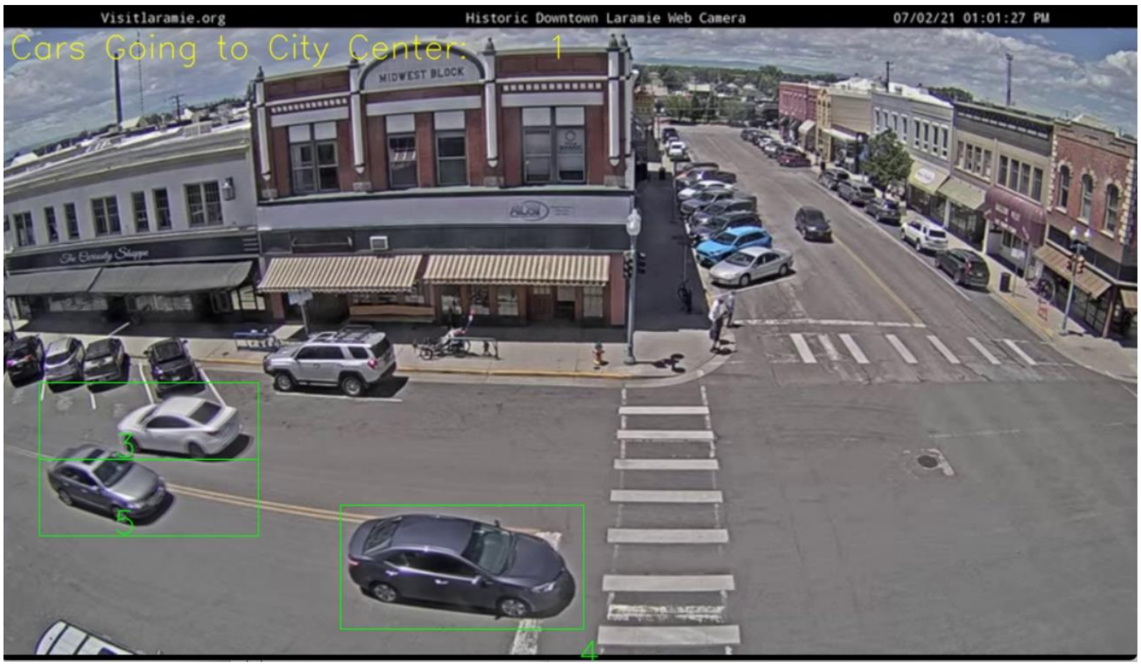


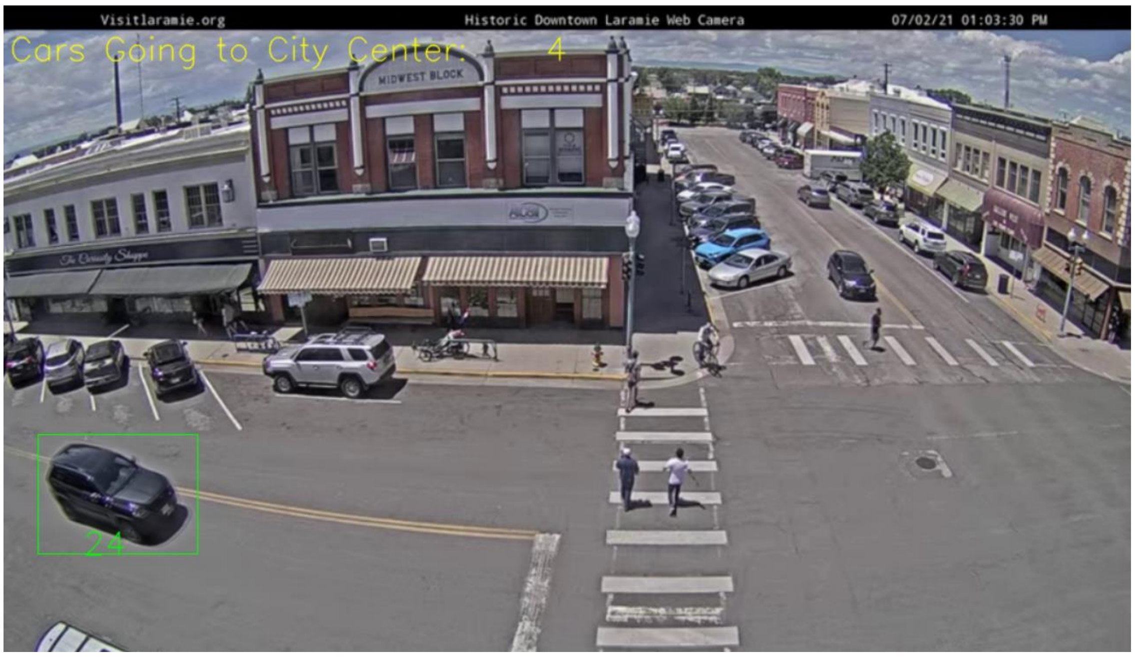
```python
# draw text on each bounding rect
for object_key in tracked_objects.copy():
    centroid = tracked_objects[object_key]
    # if track status is True
    if centroid[3] == True:

        cv2.putText(frame, str(centroid[2]), (centroid[0]-50, centroid[1]+250), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0)
        # set status to false
        tracked_objects[centroid[2]] = [centroid[0], centroid[1], centroid[2], False]
    else:
        # track status == false, object exited from region of interest
        # remove from tracked objects dictionary as it is no longer tracked
        if centroid[0] < 215:
            car_counter += 1
        del tracked_objects[centroid[2]]
```

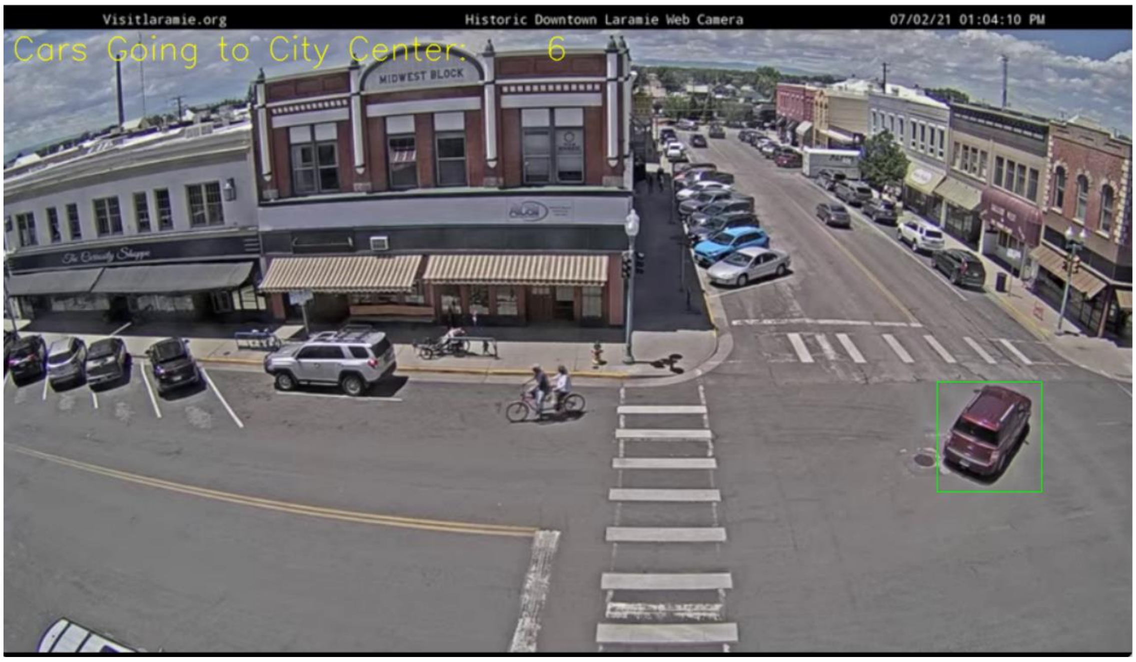
Detailed explanations are given in the source code and are not included here as the report will be too lengthy.

Results:

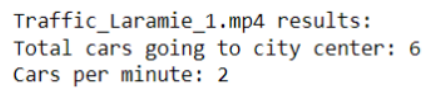

Visitlaramie.org
Historic Downtown Laramie Web Camera
07/02/21 01:01:24 PM
Cars Going to City Center: 0
MIDWEST BLOCK
2
4


Visitlaramie.org
Historic Downtown Laramie Web Camera
07/02/21 01:01:27 PM
Cars Going to City Center: 1
MIDWEST BLOCK
3
5
4

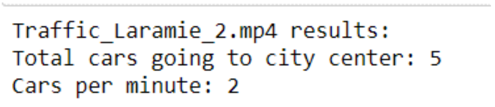

Visitlaramie.org
Historic Downtown Laramie Web Camera
07/02/21 01:03:30 PM
Cars Going to City Center: 4
MIDWEST BLOCK
24


Visitlaramie.org
Historic Downtown Laramie Web Camera
07/02/21 01:04:10 PM
Cars Going to City Center: 6
MIDWEST BLOCK

As seen from above, any overlapping vehicles can be detected as 2 separate cars instead of one and any other cars can be detected even when they have stopped. Only cars that have entered the detection area are detected and humans are not detected. Groups of people walking together or bicycles are also not detected as cars.



```
Traffic_Laramie_1.mp4 results:
Total cars going to city center: 6
Cars per minute: 2
```



```
Traffic_Laramie_2.mp4 results:
Total cars going to city center: 5
Cars per minute: 2
```

	Total number of cars	Cars per minute
Traffic_Laramie_1.mp4	6	2
Traffic_Laramie_2.mp4	5	2

Summary

Capabilities (achieved):

1) Overlapping cars stacked vertically can be split based on this algorithm.

2) All vehicles can be detected within the defined frame of [260:600,0:1040]

3) Stopped vehicles are tracked except for the first 2 stopped vehicles in video1 as the background subtractor fails to detect those 2 vehicles after they have stopped. However, subsequent stopped vehicles are still detected and tracked.

Explanation of the issue in (3): This is due to applying the background subtractor with the default learning rate of -1 “fgmask = bg_sub.apply(detection_area)” which allows the algorithm to select the learning rate automatically. The learning rate can be set to 0, “fgmask = bg_sub.apply(detection_area, learningRate = 0)”, where the first frame is used as the background and no more learning is conducted. This will allow the program to detect every vehicle including the first 2 stopped vehicles, however the accuracy of counting the number of vehicles going towards the city center lowers. For this reason, I have decided to leave the learning rate as the default of -1, sacrificing the ability to continuously track the 2 stopped vehicles but maintaining the accuracy in detecting the number of cars going to the city center for task 2.

Limitations:

1) Vehicles getting too close horizontally will end up being tracked as one object.

2) All body of tracked objects detected with an area larger than 4700 are assumed to be a vehicle.

3) Only overlapping vehicles within a certain defined area can be detected as separate objects

4) Some vehicles nearer to the camera may be incorrectly detected as overlapping vehicles

Improvements:

1) Expand on overlapping handling capabilities (Horizontal)

2) More