Exercise 1

Task 1

The application will use frame differencing and background subtraction techniques to identify and tag cars on the ‘Main Street’. Frame differencing is a technique where computers check the difference between two video frames. If pixels have changed, apparently something would have changed in the image. Background subtraction is, similarly, the process of identifying moving objects from the difference between the current frame and a reference frame, in this principle called the ‘background frame’.

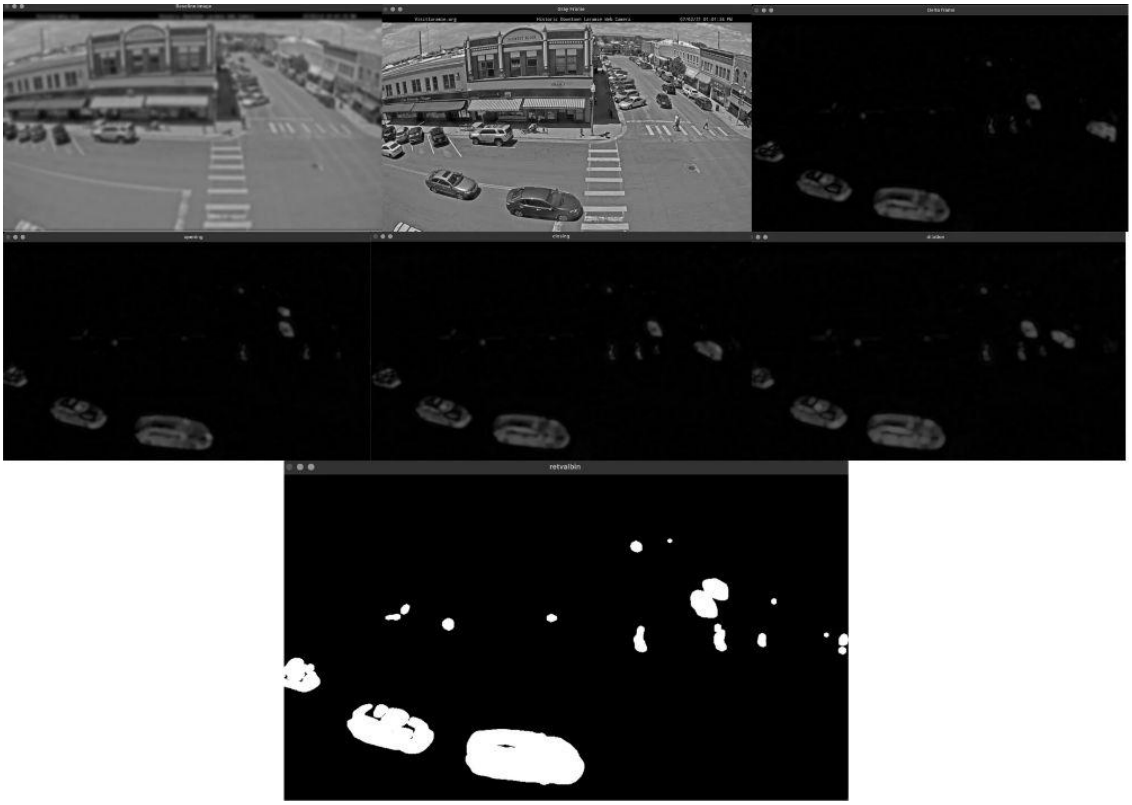
Within a while loop, running the course of the video, we take an initial frame, ‘frame’, and first make it grey. We then blur it to reduce the amount of unnecessary information that comes across. Then our delta frame represents this difference between frames, or the current frame and its ‘background’.

To ensure subtle lighting conditions and noise do not come across as real movement, a series of transformations are applied to the video image. I tweaked about and found the most accurate transforms for tagging vehicles later on. Instead of using the base threshold, I worked up to my own threshold called ‘retvalbin’, which greatly reduced the jittery and incorrect contour tags.

The transformations and frames in code.

```
# gray conversion and noise reduction (smoothening)
gray_frame=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
blur_frame=cv2.GaussianBlur(gray_frame,(25,25),0)

# the first captured frame is the baseline image
if initial_frame is None:
    initial_frame = blur_frame
    continue

# the difference between the baseline and the new frame
delta_frame=cv2.absdiff(initial_frame,blur_frame)

# the difference (the delta_frame) is converted into a binary image
# if a particular pixel value is greater than a certain threshold (specified by us here as 18),
# it will be assigned the value for White (255) else Black(0)
threshold_frame=cv2.threshold(delta_frame,18,255, cv2.THRESH_BINARY)[1]

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))  # kernel to apply to the morphology
closing = cv2.morphologyEx(delta_frame, cv2.MORPH_CLOSE, kernel)
opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
dilation = cv2.dilate(opening, kernel)
retvalbin = cv2.threshold(dilation, 12, 255, cv2.THRESH_BINARY)[1]  # removes the shadows
```

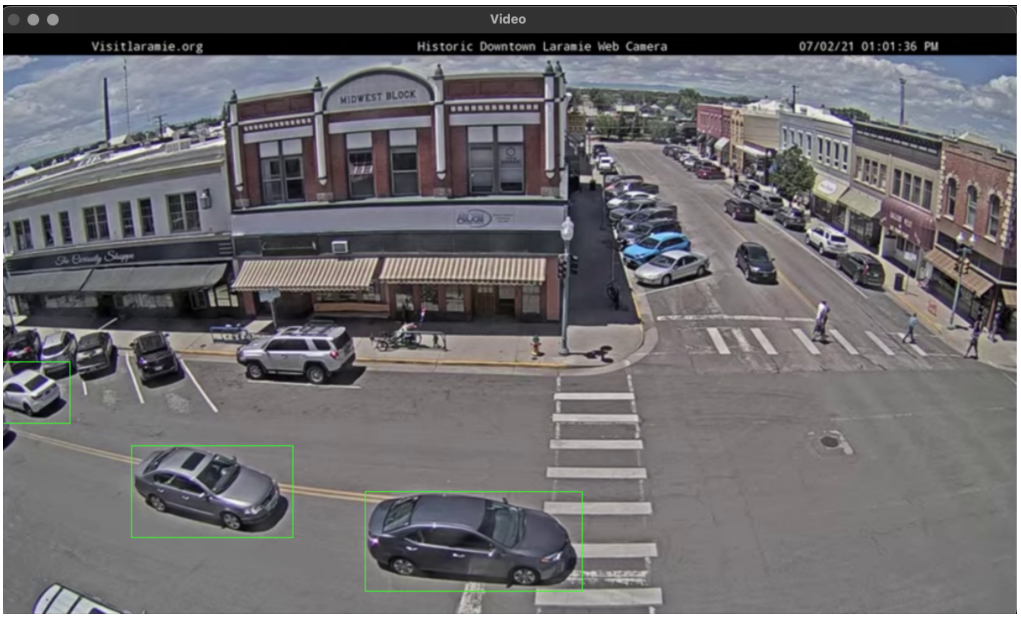
And the resulting frames.



Using the final transformation, we use cv2 to find contours and draw bounding rectangles onto objects of a certain area (>2500, neatly avoiding bicycles and pedestrians) in the original video. To ensure we are only tagging cars on the ‘Main Street’, we also restrict tags to below half the height of the frame, which works quite accurately.

```
# the cv2.findContours() method we will identify all the contours in our image
(contours,_)=cv2.findContours(retvalbin,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

# get height of video frame
height = 0
if video.isOpened():
    height = video.get(cv2.CAP_PROP_FRAME_HEIGHT)

# loop over contours and paint them onto the original video image
for c in contours:
    # contourArea() method filters out any small contours
    if cv2.contourArea(c) < 2500:
        continue
    (x, y, w, h)=cv2.boundingRect(c)
    # if car on main street, roughly bottom half of the video frame, paint contour
    if y > height / 2:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 1)

# to better understand the application, we can visualise the different frames generated
cv2.imshow('Video', frame)
```

Finally, we use imshow to trigger a kernel with the result. The correctly tagged cars on the ‘Main Street’.



The kernel can be stopped at any time using the waitkey ‘q’.

Task 2

Building on Task 1 (using the same transforms and tags), I created a class called box which will act as a collision object, checking whether an object has overlapped with it and increment a counter.

```
# create a class box as our collision object
class Box:
    def __init__(self, start_point, width_height):
        self.start_point = start_point
        self.end_point = (start_point[0] + width_height[0], start_point[1] + width_height[1])
        self.counter = 0
        self.frame_countdown = 0

    def overlap(self, start_point, end_point):
        if self.start_point[0] >= end_point[0] or self.end_point[0] <= start_point[0] or \
                self.start_point[1] >= end_point[1] or self.end_point[1] <= start_point[1]:
            return False
        else:
            return True
```

I append one box, in the lane moving towards the city centre, and later paint the box on the video so we can see it working.

```
# the boxes we want to count moving objects in
boxes = []
boxes.append(Box((300, 350), (10, 80)))
```

```
# let's also insert the boxes
for box in boxes:
    cv2.rectangle(frame, box.start_point, box.end_point, (255, 255, 255), 2)
```

Cv2s Image Moments help calculate some features like the centre of mass of an object, which seemed far more helpful then if just the whole contour passed the box. Using cv2 moments returns a dictionary of all moment values calculated. We index the centre of mass as cX and cY, and check if every contour's centre of mass overlaps the box. This happens within the loop for contours, where our cars are tagged. An overlap increments the counter.

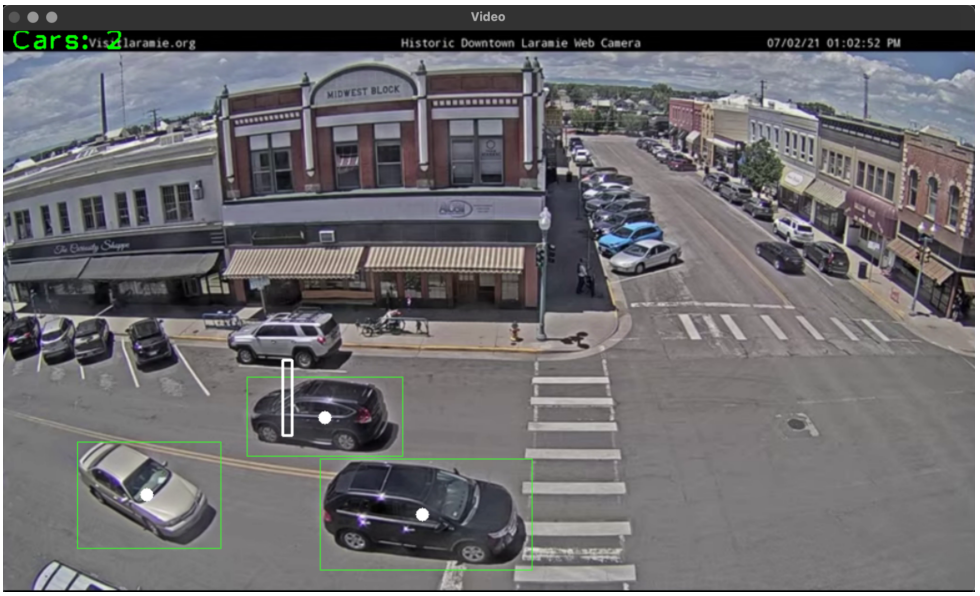
```
M = cv2.moments(c)
cX = int(M["m10"] / M["m00"])
cY = int(M["m01"] / M["m00"])
```

```
# the text string we will build up
    cars = 0
    # go through all the boxes
    for box in boxes:
        box.frame_countdown -= 1
        if box.overlap((cX, cY), (x + w, y + h)):
            if box.frame_countdown <= 0:
                box.counter += 1
            # the number might be adjusted, it is just set based on my settings
            box.frame_countdown = 20
        cars += box.counter
```

We also have a little reference to how many cars have currently passed to the city centre in the video.

```
text = "Cars: " + str(cars)
# set the text string we build up
cv2.putText(frame, text, (10, 20), cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 0), 2)
```

The resulting video looks as such.



The output of the notebook cell, when the video ends, had to include the total number of cars and cars per minute. This is achieved by using the car counter, and a calculated value of cars per minute by dividing the total by the duration of the video.

```
# video duration for cars per minute
frames = video.get(cv2.CAP_PROP_FRAME_COUNT)
fps = int(video.get(cv2.CAP_PROP_FPS))
seconds = int(frames / fps)
minutes = seconds / 60
```

```
else:
    # print total cars
    print("Cars: " + str(cars))
    # print cars per minute of video footage
    print("Cars per minute: " + str(round(cars/minutes, 2)))
    break
```

The output below the cell looks something like this when the video ends.

```
Cars: 6
Cars per minute: 2.03
```

And so, our table.

	Total number of cars	**Cars per minute***
Traffic_Laramie_1.mp4	6	2.03
Traffic_Laramie_2.mp4	4	2.29

*Per minute of video footage, as there is no obvious translation of minutes from the camera's datetime