Τεχνολογία Και Ανάλυση Εικόνων και Βίντεο

Άσκηση 1: Επεξεργασία εικόνας, φίλτρα, ακμές και εκτίμηση κίνησης

Γεώργιος Σκουρτσίδης (03114307)

Ιωαννης Βονδικάκης (031131860

ΕΡΩΤΗΜΑ 2

1. Να κατεβάσετε τοπικά το αρχείο βίντεο που αντιστοιχεί στην ομάδα σας. Κάθε frame που διαβάζετε από το βίντεο θα πρέπει να γίνεται **resize** στη **μισή ανάλυση από την αρχική**.

```
cap = cv.VideoCapture("ss.mp4")

color = (0, 255, 0)

ret, first_frame = cap.read()

#first_frame = util.random_noise(first_frame, mode='s&p', seed=7, amount = 0.37777777778)


print('Original Dimensions : ',first_frame.shape)

scale_percent = 50 # percent of original size
width = int(first_frame.shape[1] * scale_percent / 100)
height = int(first_frame.shape[0] * scale_percent / 100)
dim = (width, height)
# resize image
first_frame = cv.resize(first_frame, dim, interpolation = cv.INTER_AREA)

print('Resized Dimensions : ',first_frame.shape)
```

Έξοδος:

```
In [2]: runfile('C:/Users/home/Desktop/
Εικονες και Βιντεο/ερωτημα2ασκ1py.py',
wdir='C:/Users/home/Desktop/Εικονες και
Βιντεο')
Original Dimensions :  (720, 1280, 3)
Resized Dimensions :  (360, 640, 3)

In [3]:
```

2. Να επιλέξετε τον κατάλληλο χρωματικό χώρο για την εικόνα εισόδου σας.

```
# Converts frame to grayscale because we only need the luminance channel for detecting edges
# - less computationally expensive
prev_gray = cv.cvtColor(first_frame, cv.COLOR_BGR2GRAY)
```

3. Να εφαρμόσετε Harris και Shi-Tomasi corner detectors στο πρώτο frame του βίντεο. Να πειραματιστείτε με τις παραμέτρους των μεθόδων, με στόχο το καλύτερο δυνατό αποτέλεσμα "περιγραφής" των αντικειμένων στην εικόνα.

- Πειραματιστείτε με τις παραμέτρους maxCorners, qualityLevel και minDistance.
  Να καταγράψετε το αποτέλεσμα για κάθε εφαρμογή των μεθόδων και να σχολιάσετε τις διαφορές μεταξύ τους, καθώς και με ποιες παραμέτρους πήρατε το καλύτερο αποτέλεσμα. Για την καταγραφή θα πρέπει να υλοποιηθεί τρόπος επισημείωσης των corners στο frame, με τις μεθόδους που παρέχει η OpenCV.

Shi-Tomasi :

```
# Parameters for Shi-Tomasi corner detection
feature_params = dict(maxCorners = 300, qualityLevel = 0.05, minDistance = 20, blockSize = 7,
                      useHarrisDetector = False)
```

Για την μέθοδο Harris αλλάζουμε την τιμη της παραμέτρου useHarrisDetector σε True

```
# Parameters for Shi-Tomasi corner detection
feature_params = dict(maxCorners = 300, qualityLevel = 0.05, minDistance = 20, blockSize = 7,
                      useHarrisDetector = True)
```

Εφαρμόζουμε τις μεθόδους στο πρώτο frame

```
# Finds the strongest corners in the first frame by Shi-Tomasi / Harris method -
# we will track the optical flow for these corners
prev = cv.goodFeaturesToTrack(prev_gray, mask = None, **feature_params)
```

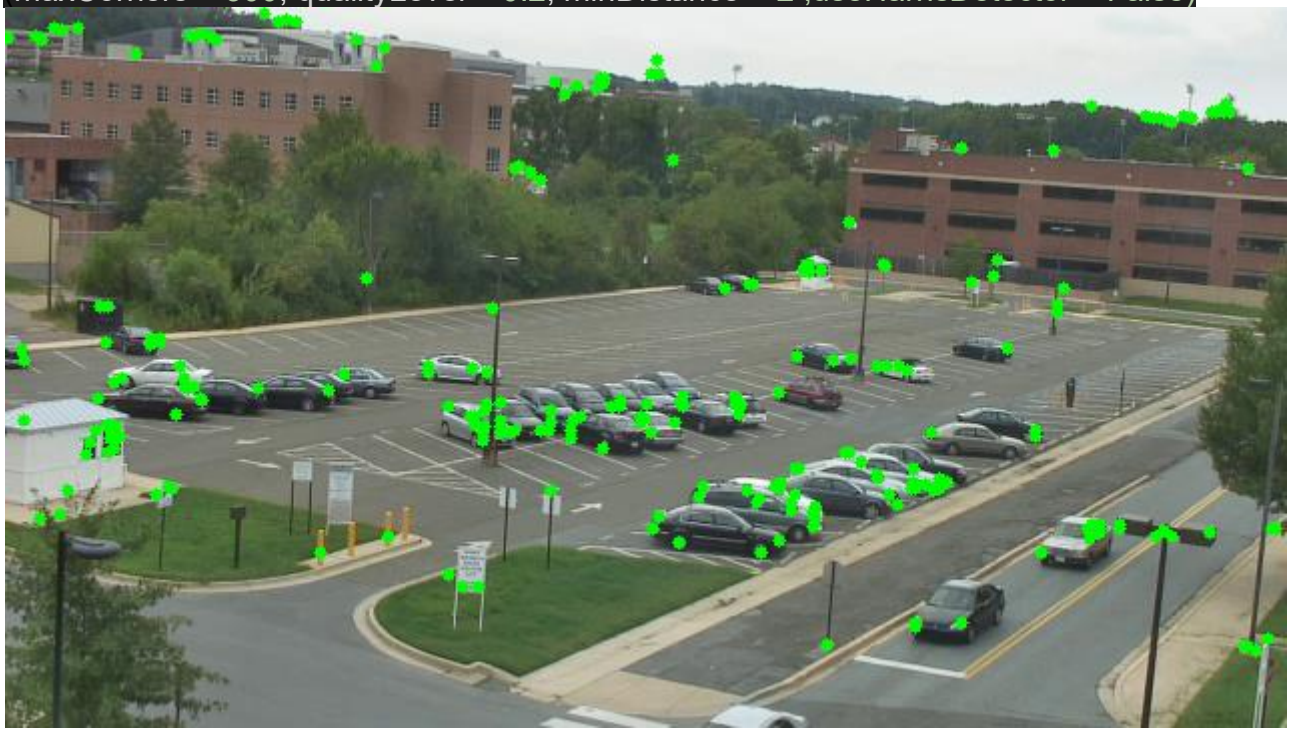
maxCorners – ο μέγιστος αριθμός γωνιών που θα επιστρέψει, αν έχουμε περισσότερες θα επιστρέψει τις εντονότερες.

qualityLevel – παράμετρος χαρακτηρίζει την ελάχιστη αποδεκτή ποιότητα γωνιών εικόνας.

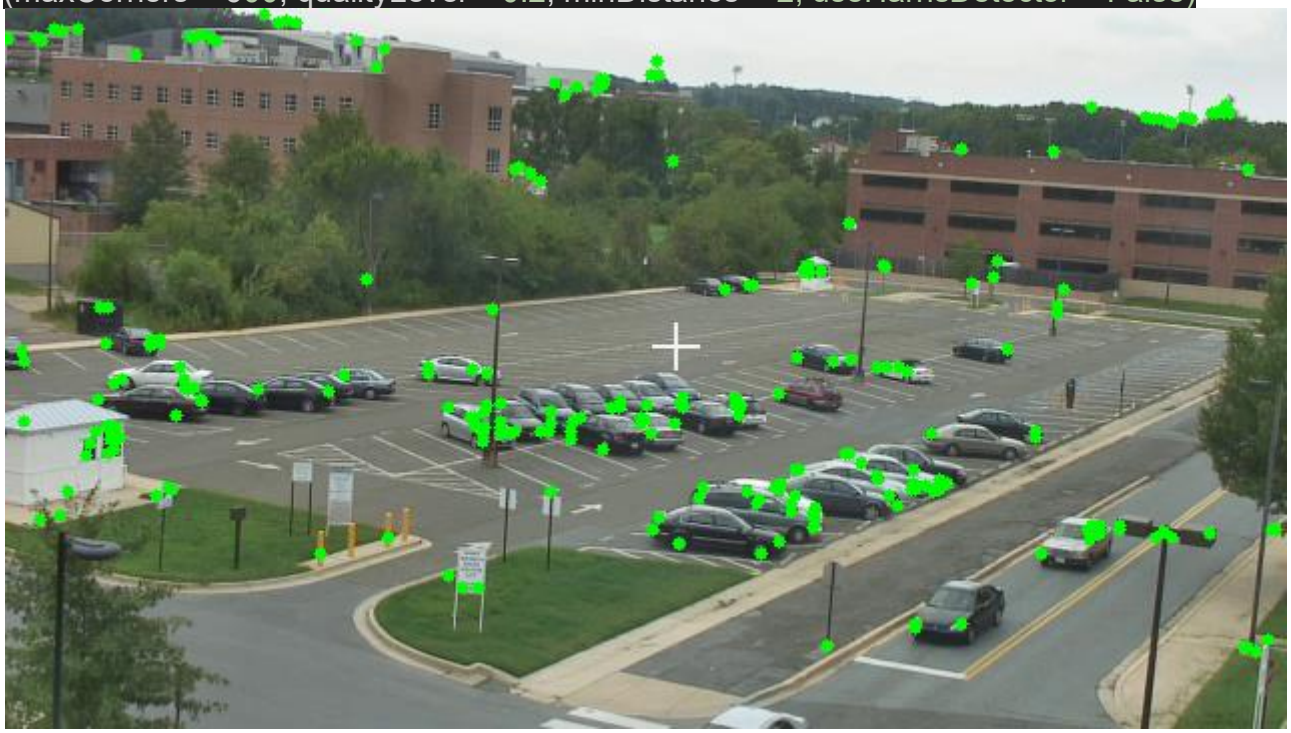
minDistance– ελάχιστη δυνατή Ευκλείδεια απόσταση μεταξύ των επιστρεφόμενων γωνιών.

Γενικές παρατηρήσεις:
Παρατηρούμε ότι όσο αυξομειώνουμε το maxCorners αυξομειώνονται και οι γωνίες ,μειώνοντας το qualityLevel ο αλγόριθμος αναγνωρίζει περισσότερες γωνίες και έχουμε καλύτερη περιγραφή. Τέλος σε μικρό minDistance έχουμε μεγάλη συγκέντρωση γωνιών ενώ αντίθετα όσο το μεγαλώνουμε έχουμε μεγαλύτερη διασπορά τον γωνιών κάτι που σε συνδυασμό με μικρό maxCorners μας δίνει κακή περιγραφή της εικόνας.

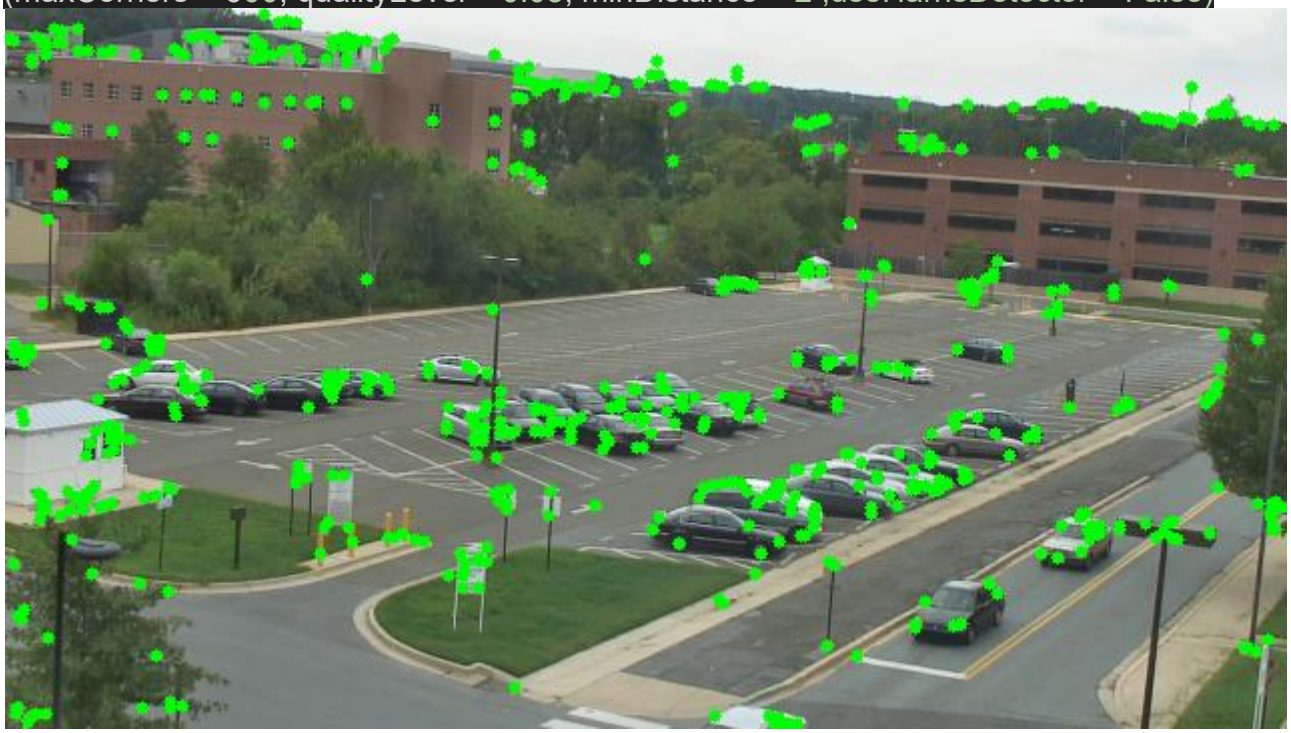
(maxCorners = 300, qualityLevel = 0.2, minDistance = 2 ,useHarrisDetector = False)



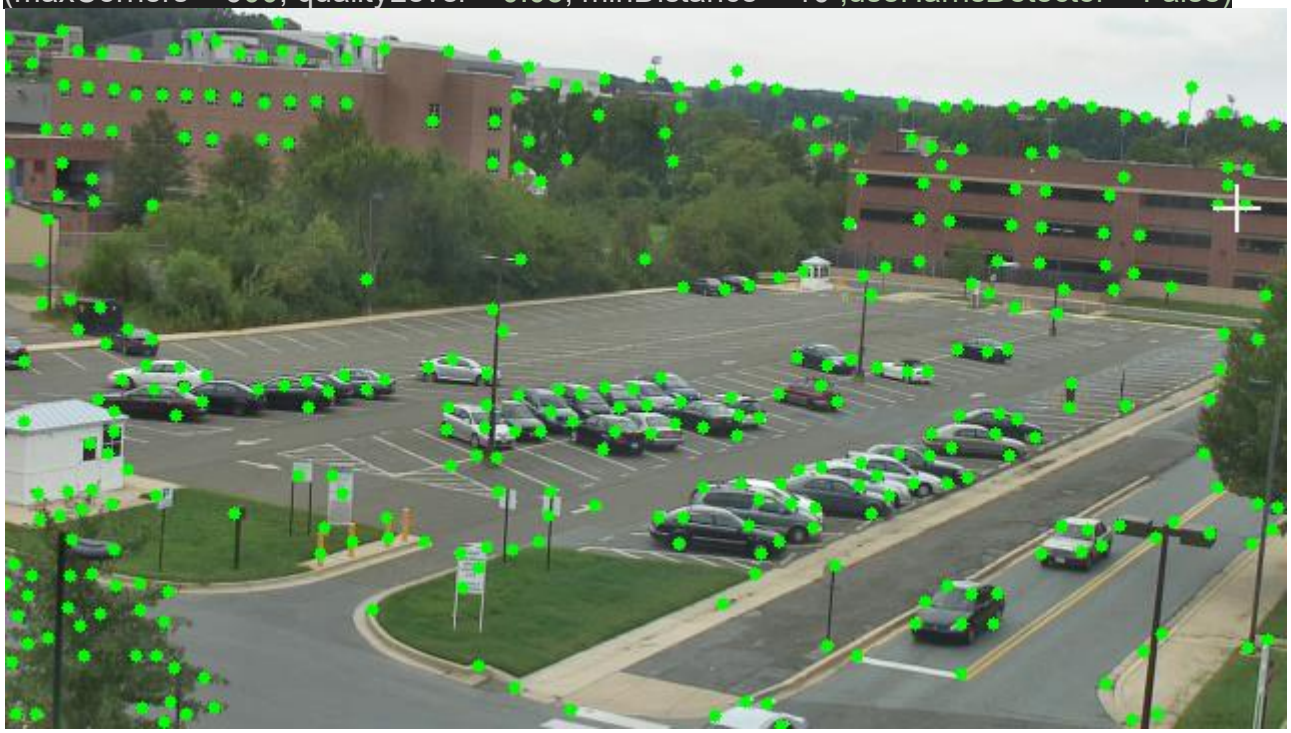
(maxCorners = 600, qualityLevel = 0.2, minDistance = 2, useHarrisDetector = False)



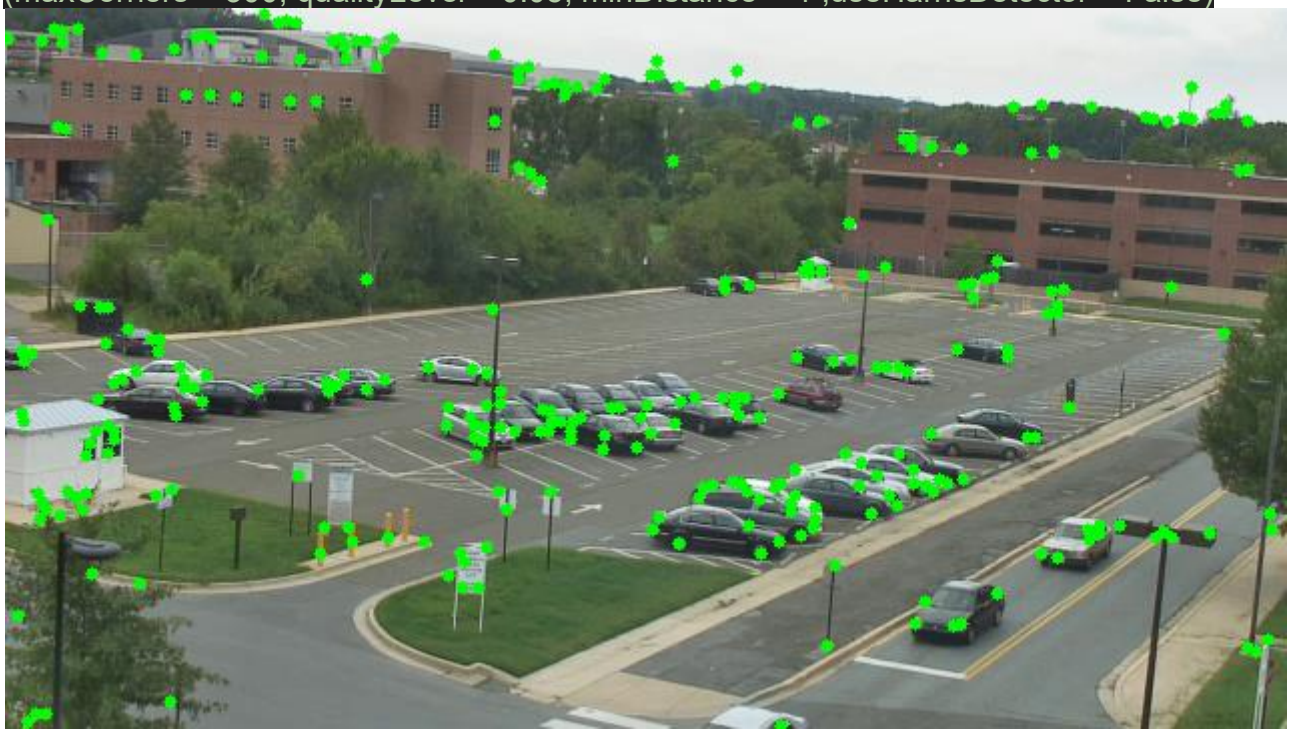
(maxCorners = 600, qualityLevel = 0.05, minDistance = 2 ,useHarrisDetector = False)



(maxCorners = 600, qualityLevel = 0.05, minDistance = 10 ,useHarrisDetector = False)

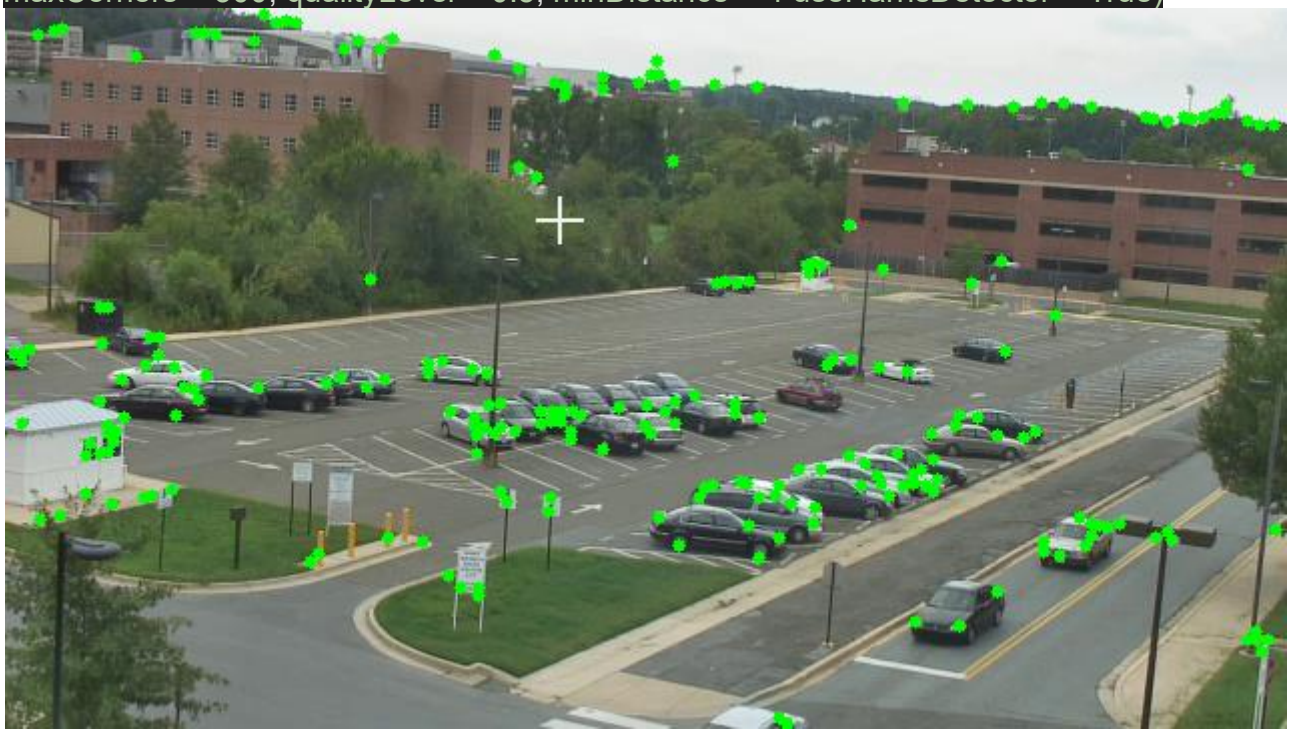


(maxCorners = 300, qualityLevel = 0.05, minDistance = 4 ,useHarrisDetector = False)



Για την μέθοδο Harris τρέχουμε τις ίδιες παραμέτρους:

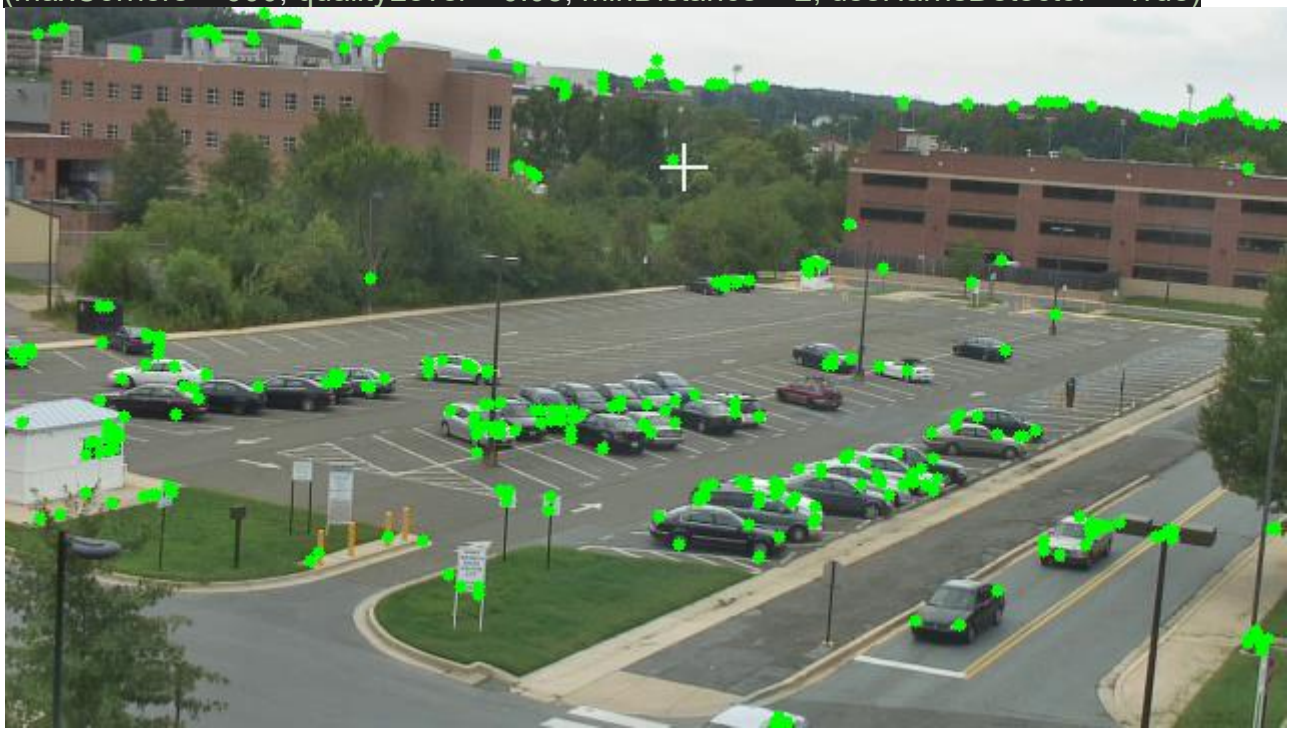
maxCorners = 300, qualityLevel = 0.5, minDistance = 4 useHarrisDetector = True)



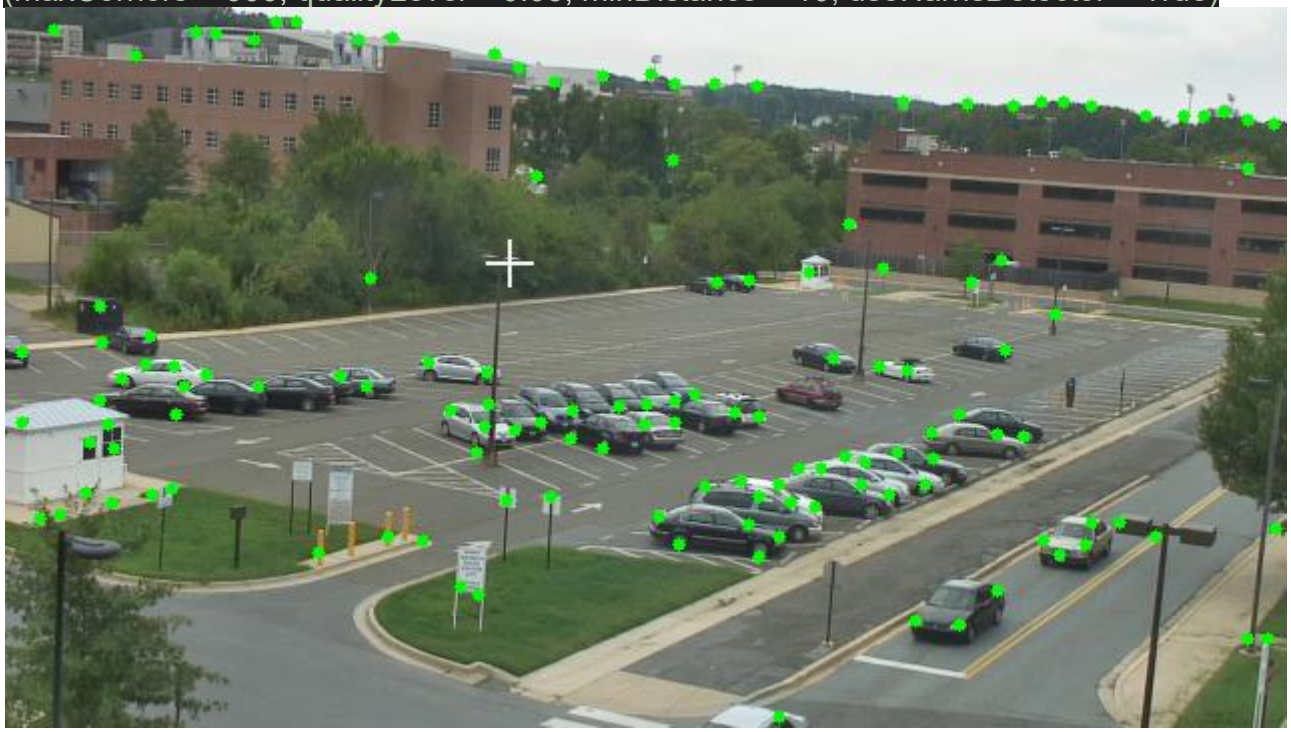
(maxCorners =300, qualityLevel = 0.2, minDistance = 2, useHarrisDetector = True)

(maxCorners = 600, qualityLevel = 0.2, minDistance = 2, useHarrisDetector = True)

(maxCorners = 600, qualityLevel = 0.05, minDistance = 2, useHarrisDetector = True)



(maxCorners = 600, qualityLevel = 0.05, minDistance = 10, useHarrisDetector = True)



(maxCorners = 1000, qualityLevel = 0.01, minDistance = 4, useHarrisDetector = True)

4. Να εφαρμόσετε τον αλγόριθμο Lucas-Kanade για υπολογισμό optical flow στα Harris και Shi-Tomasi σημεία του προηγούμενου βήματος. Να πειραματιστείτε με τις παραμέτρους

των μεθόδων, με στόχο το καλύτερο δυνατό αποτέλεσμα στο βίντεο της ομάδας σας.

- Πειραματιστείτε με τις παραμέτρους `winSize`, `maxLevel` και `criteria`, καθώς και με οποιαδήποτε άλλη παράμετρο θεωρείτε εσείς ότι χρειάζεται.
- Θα πρέπει να υλοποιήσετε τρόπο επισημείωσης της κίνησης, όπως είδαμε στο αντίστοιχο εργαστήριο.
- Να καταγράψετε ενδεικτικά screenshots από την εκτέλεση του αλγορίθμου και να συγκρίνετε τα αποτελέσματα των πειραματισμών σας. Ποιο σενάριο δίνει το καλύτερο αποτέλεσμα;

```
# Parameters for Lucas-Kanade optical flow
lk_params = dict(winSize = (15,15), maxLevel = 10, criteria = (cv.TERM_CRITERIA_EPS |
                                                               cv.TERM_CRITERIA_COUNT, 10, 0.03))

# Converts frame to grayscale because we only need the luminance channel for detecting edges
# - less computationally expensive
prev_gray = cv.cvtColor(first_frame, cv.COLOR_BGR2GRAY)


# Finds the strongest corners in the first frame by Shi-Tomasi / Harris method -
# we will track the optical flow for these corners
prev = cv.goodFeaturesToTrack(prev_gray, mask = None, **feature_params)



# Creates an image filled with zero intensities with the same dimensions as the frame
#- for later drawing purposes
mask = np.zeros_like(first_frame)

j = 0

while(cap.isOpened()):

    j = j+1

    ret, frame = cap.read()

    #frame = util.random_noise(frame, mode='s&p', seed=7, amount = 0.37777777778)

    scale_percent = 50 # percent of original size
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    # resize image
    frame = cv.resize(frame, dim, interpolation = cv.INTER_AREA)

    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)


    # Calculates sparse optical flow by Lucas-Kanade method
    next, status, error = cv.calcOpticalFlowPyrLK(prev_gray, gray, prev, None, **lk_params)

    # Selects good feature points for previous position
    good_old = prev[status == 1]

    # Selects good feature points for next position
    good_new = next[status == 1]

    # Draws the optical flow tracks
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        # a, b = coordinates of new point
        a, b = new.ravel()

        # a, b = coordinates of old point
        c, d = old.ravel()

        # Draws line between new and old position with green color and 2 thickness
        mask = cv.line(mask, (a, b), (c, d), color, 2)

        # Draws filled circle (thickness of -1) at new position with green color and radius of 3
        frame = cv.circle(frame, (a, b), 3, color, -1)

    # Overlays the optical flow tracks on the original frame
    output = cv.add(frame, mask)

    # Updates previous frame
    prev_gray = gray.copy()

    # Updates previous good feature points
    prev = good_new.reshape(-1, 1, 2)

    # Opens a new window and displays the output frame
    cv.imshow("sparse optical flow", output)

    # Frames are read by intervals of 10 milliseconds. The programs breaks out of the while loop when the user p
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

# The following frees up resources and closes all windows
cap.release()
cv.destroyAllWindows()
```

```
    # Opens a new window and displays the output frame
    cv.imshow("sparse optical flow", output)

    # Frames are read by intervals of 10 milliseconds. The programs breaks out of the while loop when the user p
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

# The following frees up resources and closes all windows
cap.release()
cv.destroyAllWindows()


    # Opens a new window and displays the output frame
    cv.imshow("sparse optical flow", output)

    # Frames are read by intervals of 10 milliseconds. The programs breaks out of the while loop when the user p
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

# The following frees up resources and closes all windows
cap.release()
cv.destroyAllWindows()
```

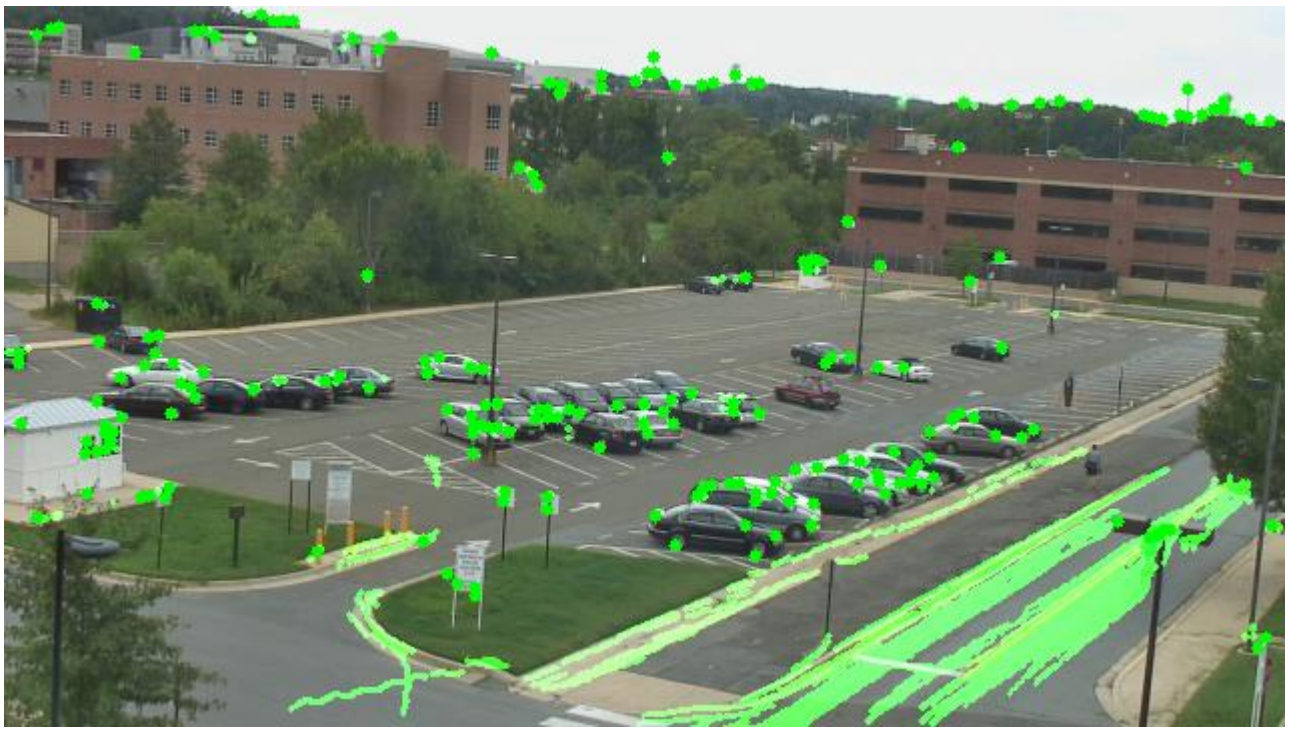
winSize : μέγεθος του παραθύρου αναζήτησης σε κάθε επίπεδο πυραμίδας.
maxLevel : ο αριθμός μέγιστου επιπέδου πυραμίδας με βάση 0. Εάν οριστεί σε 0, δεν χρησιμοποιούνται πυραμίδες (μονό επίπεδο), αν οριστεί σε 1, χρησιμοποιούνται δύο επίπεδα και ούτω καθεξής. εάν οι πυραμίδες μεταβιβαστούν στην είσοδο, τότε ο αλγόριθμος θα χρησιμοποιεί τόσα πολλά επίπεδα όπως οι πυραμίδες έχουν αλλά όχι περισσότερο από το maxLevel.
criteria : η παράμετρος καθορίζει τα κριτήρια τερματισμού του επαναληπτικού αλγορίθμου αναζήτησης μετά τον καθορισμένο μέγιστο αριθμό κριτηρίων επαναλήψεων .maxCount ή όταν το παράθυρο αναζήτησης κινείται με λιγότερα από criteria.epsilon.
criteria.maxCount: καθορίζει τον μέγιστο αριθμό επαναλήψεων που θα κάνει, η διαδικασία θα σταματήσει εάν φτάσει σε αυτόν τον αριθμό.
criteria.epsilon: καθορίζει το ελάχιστο παράθυρο αναζήτησης το οποίο μπορεί να επεξεργαστεί η διαδικασία θα σταματήσει όταν το παράθυρο αναζήτησης κινείται λιγότερο από αυτό.

Επιλέγουμε τους παρακάτω συνδυασμούς παραμέτρων από το 3 ερώτημα διότι θέλουμε να εντοπίζει όλους του ανθρώπους και τα αυτοκίνητα χωρίς όμως να έχουμε υπερβολικά πολλές γωνίες

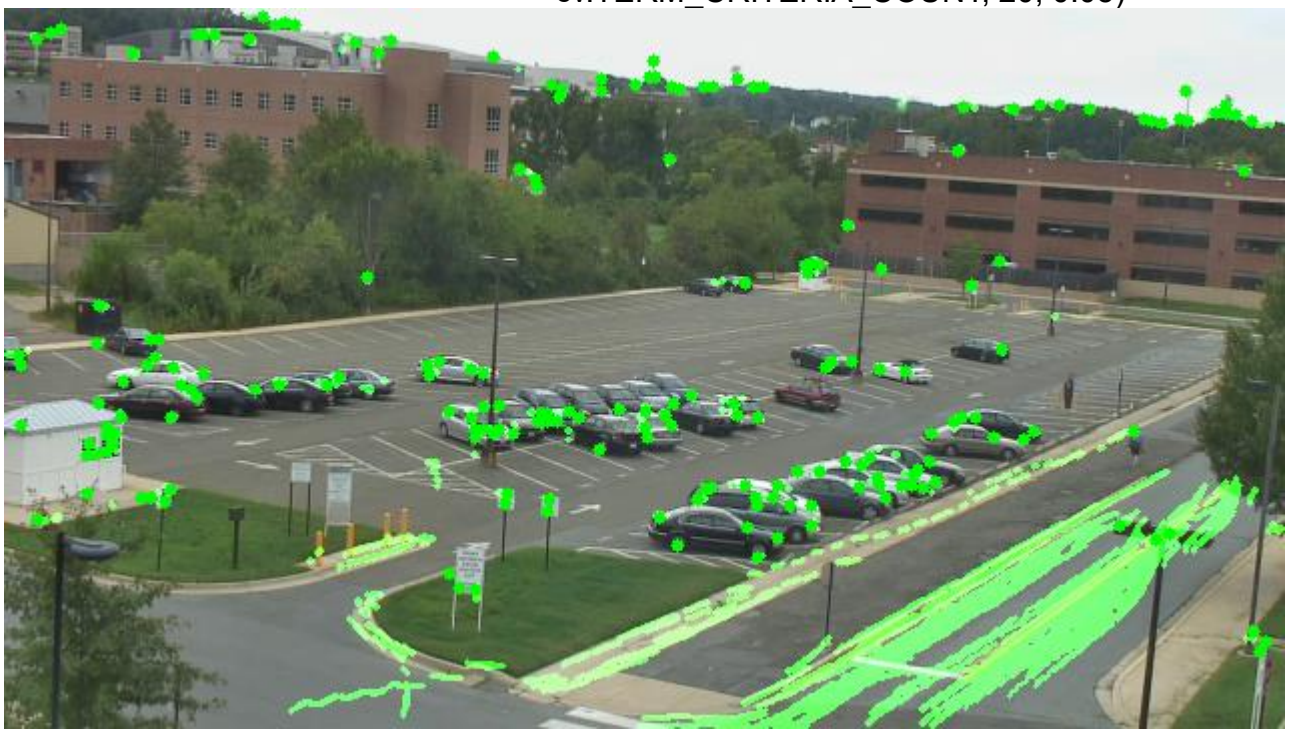
(maxCorners = 300, qualityLevel = 0.05, minDistance = 4 ,useHarrisDetector = False)

maxCorners = 300, qualityLevel = 0.5, minDistance = 4 useHarrisDetector = True)

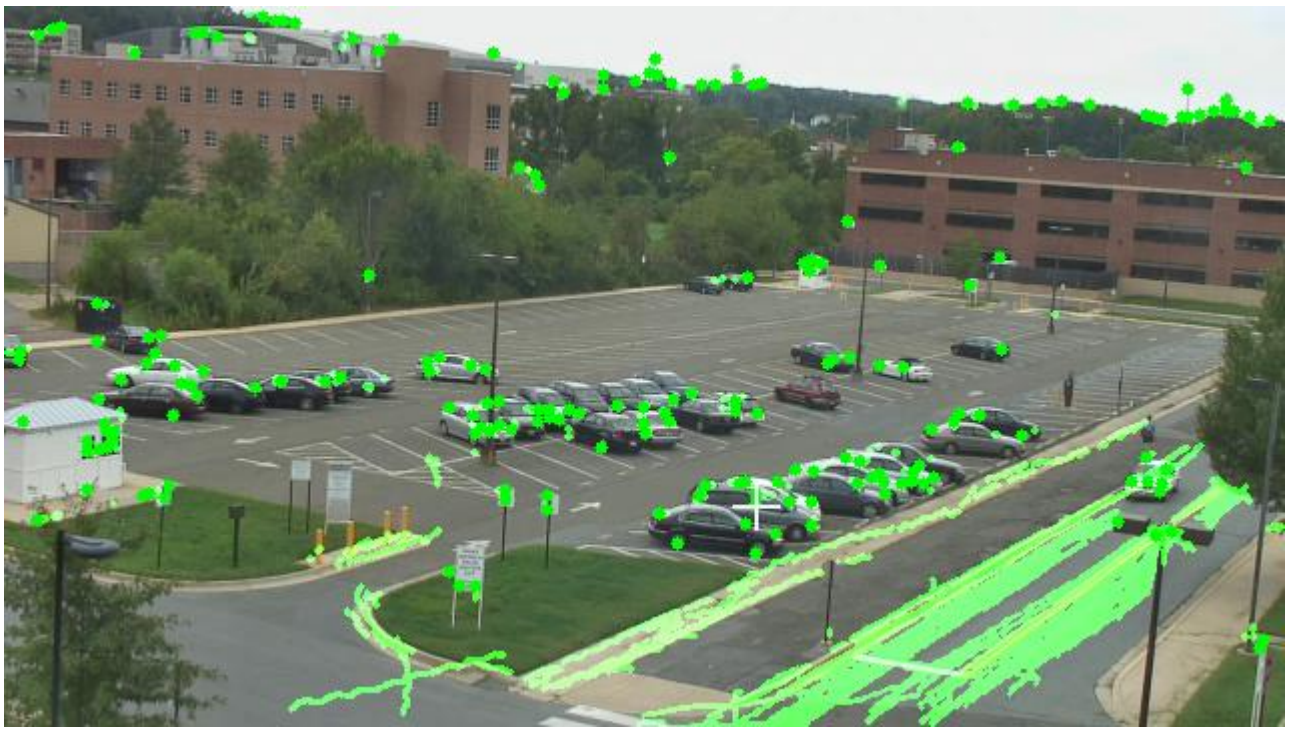
(winSize = (15,15), maxLevel = 2, criteria = (cv.TERM_CRITERIA_EPS |
cv.TERM_CRITERIA_COUNT, 10, 0.03))



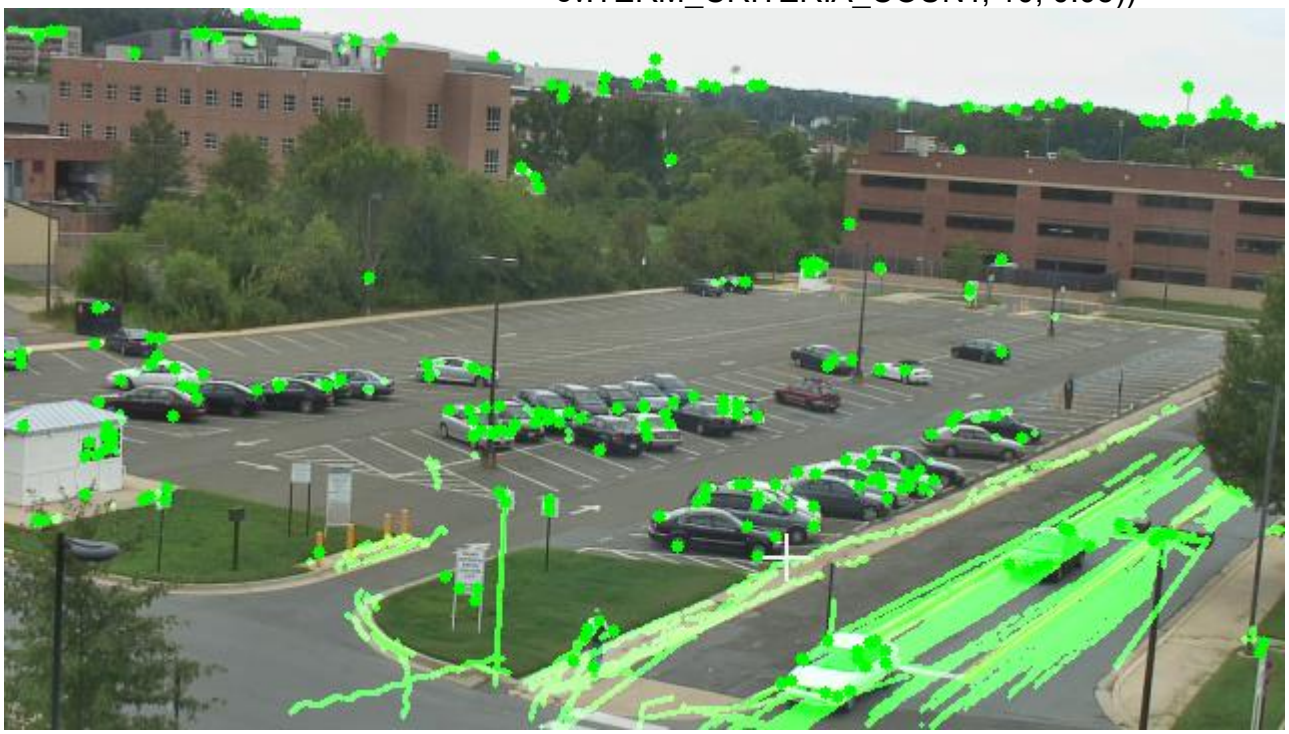
(winSize = (30,30), maxLevel = 5, criteria = (cv.TERM_CRITERIA_EPS |
cv.TERM_CRITERIA_COUNT, 20, 0.05)



(winSize = (5,5), maxLevel = 0, criteria = (cv.TERM_CRITERIA_EPS |
cv.TERM_CRITERIA_COUNT, 5, 0.01))

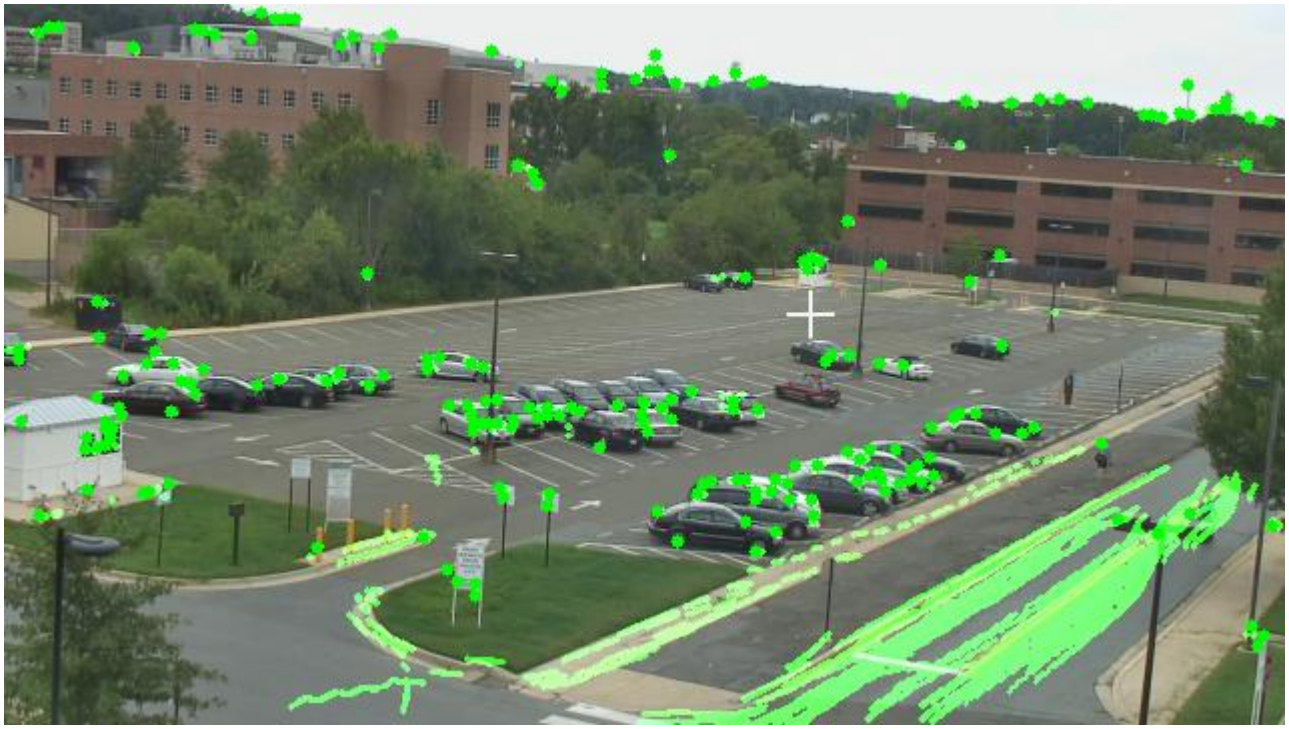


lk_params = dict(winSize = (5,5), maxLevel = 3, criteria = (cv.TERM_CRITERIA_EPS |
cv.TERM_CRITERIA_COUNT, 10, 0.05))



(η χειρότερη δοκιμή σχηματιστήκαν 2 τυχαίες διαδρομές χωρίς να υπάρχει κίνηση)

winSize = (30,30), maxLevel = 1, criteria = (cv.TERM_CRITERIA_EPS |
cv.TERM_CRITERIA_COUNT, 6, 0.07))



Με τις παρακάτω 2 δοκιμές είχαμε την πιο ομαλή παρακολούθηση των κινούμενων αντικείμενων. Χωρίς ωστόσο να έχουμε σημαντικές διάφορες μεταξύ των δοκίμων
(winSize = (5,5), maxLevel = 0, criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 5, 0.01))
(winSize = (15,15), maxLevel = 2, criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))

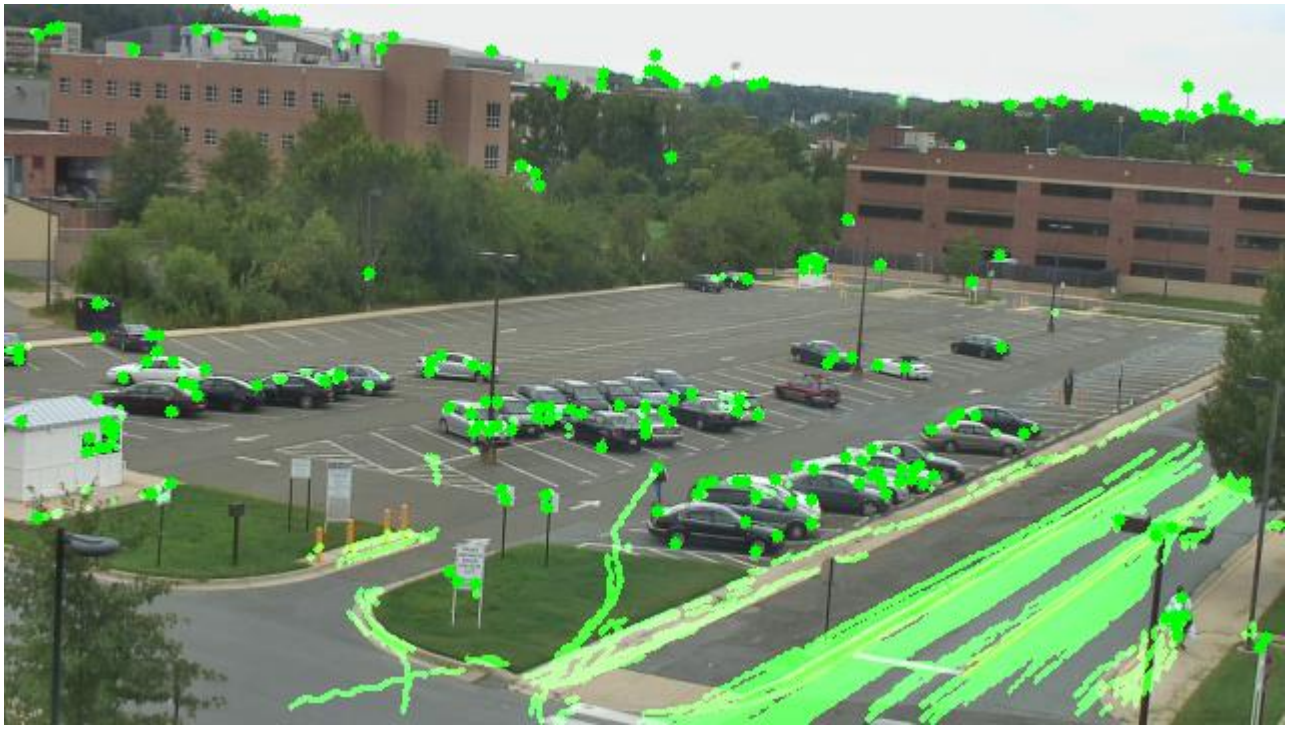
5.Να τροποποιήσετε τον κώδικά σας για το βήμα 4, με στόχο την παρακολούθηση γωνιών **Harris** και **Shi-Tomasi** που εμφανίσθηκαν στο βίντεο **μετά** το πρώτο frame. Ακολουθήστε τις παρακάτω οδηγίες:

- Εφαρμόστε τις παραμέτρους που σας έδωσαν το καλύτερο αποτέλεσμα στα βήματα 3 και 4.
- Θα πρέπει να **ενημερώνετε** τα σημεία που παρακολουθούνται ανά τακτικά διαστήματα, αλλά όχι σε κάθε frame.
- Προσπαθήστε να επισημειώσετε **μόνο** τα σημεία που αλλάζουν σημαντικά θέση (μετακίνηση πάνω από ένα pixel σε μια από τις διαστάσεις)

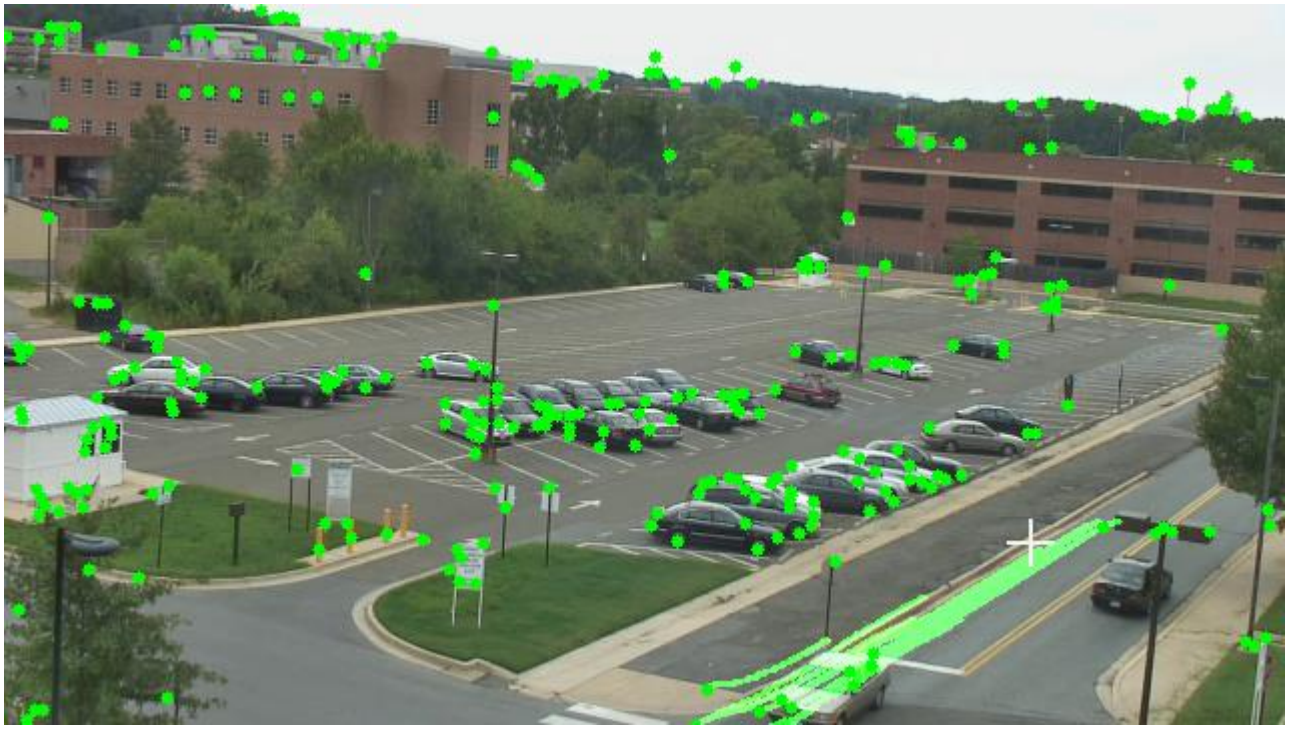
Να καταγράψετε ενδεικτικά screenshots από την εκτέλεση του αλγορίθμου.

```
j = 0

while(cap.isOpened()):

    j = j+1

    ret, frame = cap.read()

    #frame = util.random_noise(frame, mode='s&p', seed=7, amount = 0.37777777778)

    scale_percent = 50 # percent of original size
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    # resize image
    frame = cv.resize(frame, dim, interpolation = cv.INTER_AREA)

    if j==30 :

        j=0

        # Converts frame to grayscale because we only need the luminance channel for detecting edges - less compu
        prev_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)


        # Finds the strongest corners in the  frame by Shi-Tomasi / Harris method - we will track the optical flo

        prev = cv.goodFeaturesToTrack(prev_gray, mask = None, **feature_params)

    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
```

Για να παρακολουθήσουμε την κίνηση τον γωνιών που εμφανίζονται μετα το πρώτο φραμε μέσα στην while βάζουμε μια if ώστε ανά κάποια καρέ (ανάλογα την μέγιστη τιμή του j) και εφαρμόζουμε Harris ή Shi-Tomasi corner detectors ξανά.



Χωρίς να εφαρμόζουμε ξανά corner detectors:



6.Να τροποποιήσετε τον κώδικά σας για το βήμα 5, εισάγοντας **Salt and Pepper θόρυβο** μετά από κάθε "διάβασμα" ενός frame.

- Ακολουθήστε την ίδια διαδικασία με το **βήμα 3 του Ερωτήματος 1**.
- Καταγράψτε ενδεικτικά frames και εξηγήστε την επίδραση του θορύβου salt and pepper στο αποτέλεσμα του motion estimation με optical flow.

Εφαρμόζουμε θόρυβο salt & papper μετά από κάθε ανάγνωση :

```
ret, first_frame = cap.read()

first_frame = util.random_noise(first_frame, mode='s&p')
#first_frame = filters.rank.median(first_frame, neighborhood)
```

Ωστόσο μας βγάζει το παρακάτω error και δεν μπορέσαμε να τρέξουμε ολόκληρο τον κώδικα

```
error: OpenCV(4.2.0) c:\projects
\opencv-python\opencv\modules\imgproc
\src\color.simd_helpers.hpp:94:
error: (-2:Unspecified error) in
function '__cdecl
cv::impl::`anonymous-
namespace'::CvtHelper<struct
cv::impl::`anonymous
namespace'::Set<3,4,-1>,struct
cv::impl::A0xe227985e::Set<1,-1,-1>,s
truct
cv::impl::A0xe227985e::Set<0,2,5>,2>:
:CvtHelper(const class
cv::_InputArray &,const class
cv::_OutputArray &,int)'
> Unsupported depth of input image:
>     'VDepth::contains(depth)'
> where
>     'depth' is 6 (CV_64F)
```

7.Στον κώδικα του βήματος 6, να εφαρμόστε κατάλληλη αποθορυβοποίηση, όπως αυτήν που σας έδωσε το καλύτερο αποτέλεσμα στο **βήμα 4 του Ερωτήματος 1**, πριν την εισαγωγή του κάθε frame στον αλγόριθμο. Καταγράψτε ενδεικτικά frames και συγκρίνετε το αποτέλεσμα με το αντίστοιχο του βήματος 5.

```
    ret, frame = cap.read()

    frame = util.random_noise(frame, mode='s&p')
    frame = filters.rank.median(frame, neighborhood)
```

Χρησιμοποιώντας την αποθορυβοποίηση με φίλτρο median θα περνάμε καθαρή εικόνα,καθώς το φίλτρο median παρέχει τα καλύτερα αποτελέσματα για Salt and Pepper θόρυβο συγκριτικά με όλες τις άλλες μεθόδους,όπως άλλωστε έχουμε διαπιστώσει και στο πρώτο ερώτημα της εργασίας.