

Κανιάρης Δημοσθένης, AM : 03116130

Κυριακίδης Δημήτριος, AM : 03117077

ΣΗΜΕΙΩΣΗ:

Για να τρέξετε το παρακάτω πρόγραμμα πρέπει να προσθέσετε στα αρχεία το βίντεο της ομάδας μας "virat.mp4". Αρχικά, επειδή εκτελώντας το πρόγραμμα τυπώνονται πολλές εικόνες με συνολική χωρητικότητα πολύ μεγάλη, είχαμε κρατήσει κάποιες ενδεικτικές φωτογραφίες όπως θα δείτε στο αντίστοιχο μέρος της άσκησης. Στη συνέχεια όμως ειπώθηκε στο μάθημα πως μπορούμε να υποβάλλουμε με σύνδεσμο url για το συγκεκριμένο colab. Για αυτόν τον λόγο τρέξαμε το πρόγραμμα και το αποθήκευσαμε. Στη συνέχεια θα προσπαθήσουμε να το υποβάλλουμε κανονικά και αν δεν επιτραπεί λόγω χωρητικότητας θα υποβάλλουμε τον αντίστοιχο σύνδεσμο.

Θεωρητικό Μέρος

1. Πώς λειτουργεί η πυραμίδα στο Lucas Kanade Feature Tracker και γιά ποιόν λόγο χρησιμοποιείται;

Ο αλγόριθμος πυραμίδας KLT λειτουργεί ως εξής: αρχικά, υπολογίζεται η οπτική ροή (optical flow) και ο μετασχηματισμός ομοιότητας (affine transformation) στο βαθύτατο επίπεδο της πυραμίδας L_m . Υστερα, το αποτέλεσμα μεταφέρεται στο ανώτερο επίπεδο L_{m-1} υπό μορφή εκτίμησης για την μετατόπιση των pixels και τον μετασχηματισμό ομοιότητας. Δεδομένης αυτής υπολογίζεται η οπτική ροή και ο μετασχηματισμός ομοιότητας για το επίπεδο L_{m-1} . Στη συνέχεια το αποτέλεσμα μεταφέρεται στο επίπεδο L_{m-2} και ούτω καθ' εξής έως το επίπεδο 0 (την αυθεντική εικόνα).

Η χρήση της μεθόδου πυραμίδας στον αλγόριθμο Lucas - Kanade Feature Tracker λύνει το πρόβλημα που προκύπτει με την επιλογή του μεγέθους του integration window. Συγκεκριμένα, για να διατηρήσουμε τις λεπτομέρειες των εικόνων χρειάζεται ενα μικρό παράθυρο, όμως σε βίντεο που περιέχουν «μεγάλη κίνηση» (παραβιάζουν, δηλαδή, την υπόθεση της μεθόδου πως μεταξύ διαδοχικών καρέ τα αντικείμενα δεν εκτοπίζονται σημαντικά) χρειάζεται μεγάλο παράθυρο για να αποφευχθεί ο θόρυβος. Μειώνοντας την ανάλυση των εικόνων σε μορφή πυραμίδας και εφαρμόζοντας τον αλγόριθμο διαδοχικά σε κάθε επίπεδο επιτυγχάνεται μεγαλύτερη ακρίβεια. </div>

Ανιχνευτές Γωνιών

2. Στο 1ο frame του βίντεο εφαρμόστε τους ανιχνευτές γωνιών Harris και Shi-Tomasi. Να πειραματιστείτε με τις παραμέτρους των μεθόδων. με στόχο το καλύτερο δυνατό αποτέλεσμα "περιγραφής" των χαρακτηριστικών σημείων (γωνιές) της 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.

```
In [ ]: import cv
import matplotlib.pyplot as plt
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import drive

#drive.mount('/content/gdrive')
```

```
In [ ]: #import os
#print(os.listdir('/content/gdrive/My Drive/Colab Notebooks'))
```

```
In [ ]: # Create a VideoCapture object and read from input file
# If the input is the camera, pass 0 instead of the video file name
#cap = cv2.VideoCapture('/content/gdrive/My Drive/Colab Notebooks/virat.mp4')
cap = cv2.VideoCapture('/content/virat.mp4')
ret, frame = cap.read()

print(cap.get(cv2.CAP_PROP_FPS))
print(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
print(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

print(frame.shape)

23.97
1280.0
720.0
(720, 1280, 3)
```

```
In [ ]: ### Shi-Tomasi Parameters
### Shi-Tomasi
...  
...
```

```

As usual, image should be a grayscale image.
Then you specify number of corners you want to find.
Then you specify the quality level, which is a value between 0-1,
which denotes the minimum quality of corner below which everyone
is rejected. Then we provide the minimum euclidean distance
between corners detected.
...
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 70,
                       qualityLevel = 0.4,
                       minDistance = 8,
                       blockSize = 7 )

feature_params1 = dict( maxCorners = 190,
                        qualityLevel = 0.16,
                        minDistance = 10,
                        blockSize = 7 )

cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame

new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
new_frame1 = np.copy(new_frame)

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)

corners = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv2.circle(new_frame,(x,y),3,255,-1)

corners1 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params1) # useHarrisDetector = True
corners1 = np.int0(corners1)
for i in corners1:
    x,y = i.ravel()
    cv2.circle(new_frame1,(x,y),3,255,-1)

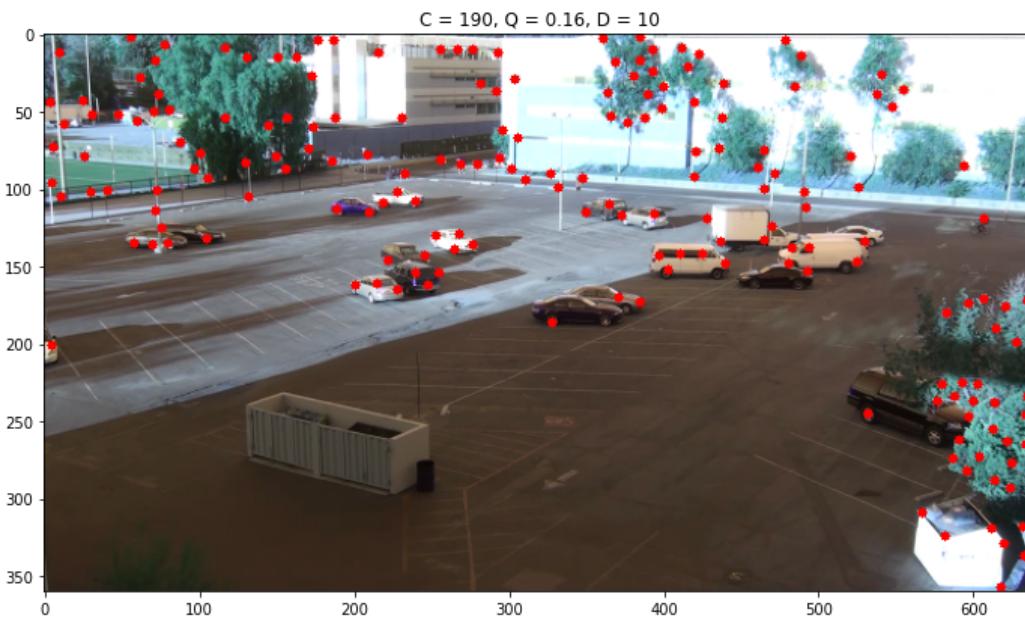
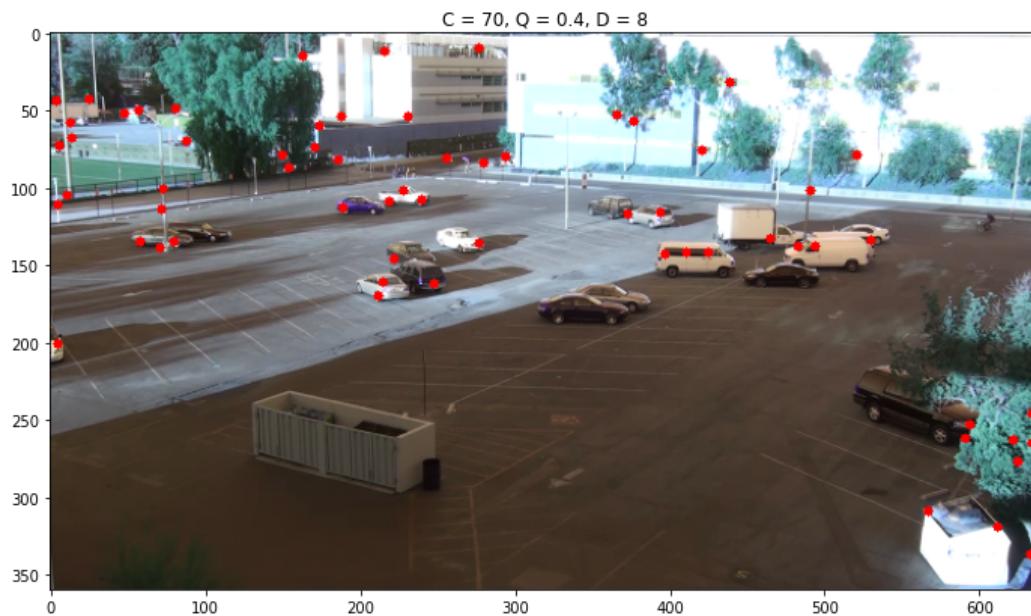
fx, plots = plt.subplots(2, 1, figsize=(15,15))

plots[0].set_title("C = 70, Q = 0.4, D = 8")
plots[0].imshow(new_frame)

plots[1].set_title("C = 190, Q = 0.16, D = 10")
plots[1].imshow(new_frame1)

```

Out[]:



Lucas-Kanade

3. 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 όποιαδήποτε άλλη παράμετρο θεωρείτε εξεις 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;

```
In [ ]: 
### Harris Parameters

### Harris
...
As usual, image should be a grayscale image.
Then you specify number of corners you want to find.
Then you specify the quality level, which is a value between 0-1,
which denotes the minimum quality of corner below which everyone
is rejected. Then we provide the minimum euclidean distance
between corners detected.
...
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 70,
                       qualityLevel = 0.4,
                       minDistance = 8,
                       blockSize = 7 )
```

```

feature_params1 = dict( maxCorners = 210,
                      qualityLevel = 0.03,
                      minDistance = 10,
                      blockSize = 7 )

cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame

new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
new_frame1 = np.copy(new_frame)

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)

corners = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = T
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv2.circle(new_frame,(x,y),3,255,-1)

corners1 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params1) # useHarrisDetector = F
corners1 = np.int0(corners1)
for i in corners1:
    x,y = i.ravel()
    cv2.circle(new_frame1,(x,y),3,255,-1)

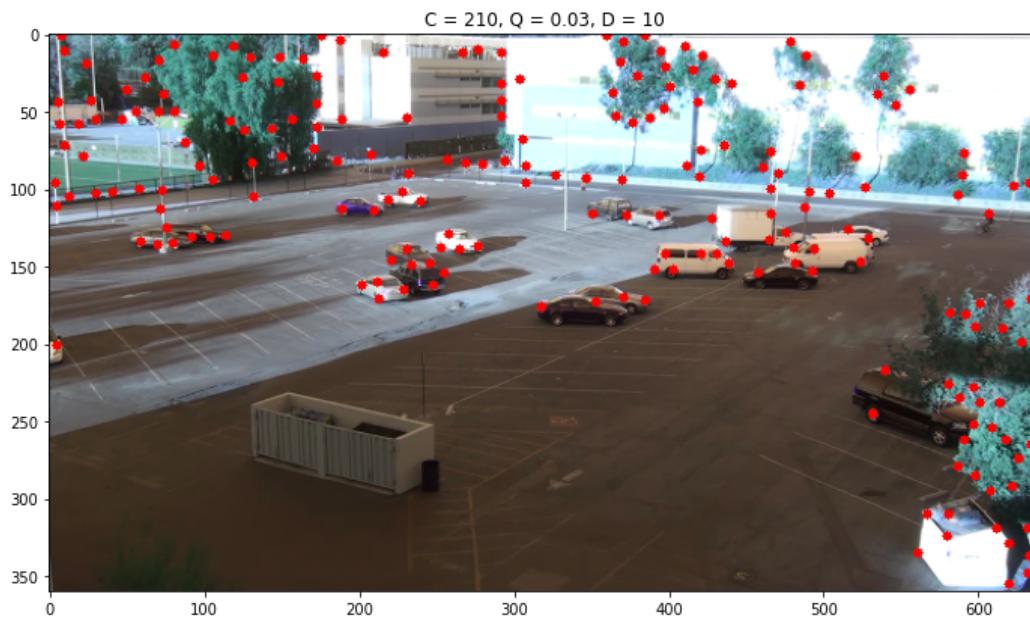
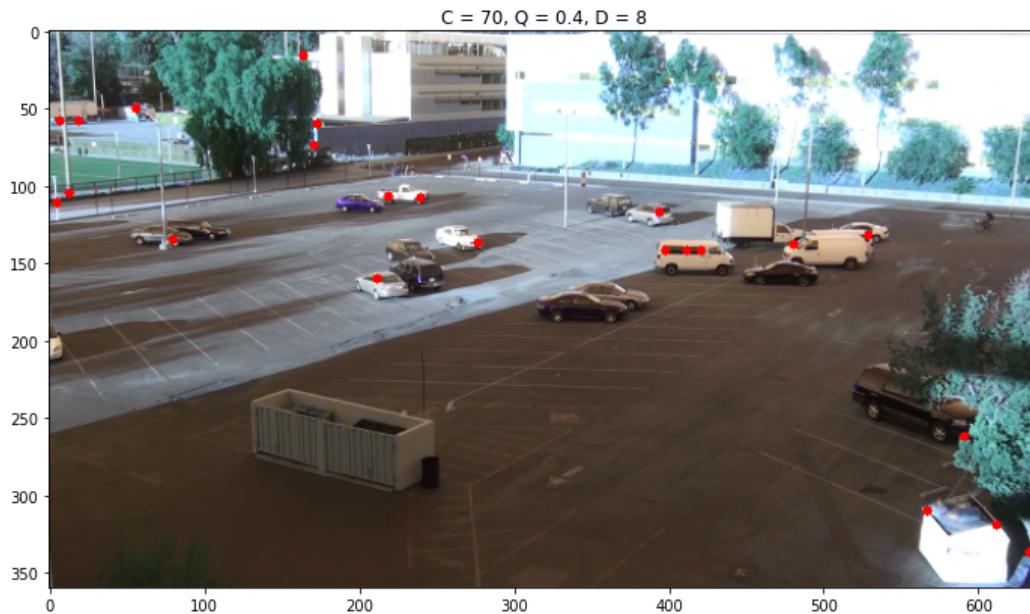
fx, plots = plt.subplots(2, 1, figsize=(15,15))

plots[0].set_title("C = 70, Q = 0.4, D = 8")
plots[0].imshow(new_frame)

plots[1].set_title("C = 210, Q = 0.03, D = 10")
plots[1].imshow(new_frame1)

```

Out[]: <matplotlib.image.AxesImage at 0x7f66684abd90>



Επιλογές παραμέτρων Ανίχνευσης Ακμών

Όσον αφορά το minDistance φάνηκε και στις δύο μεθόδους πρέπει να είναι ανάμεσα στο 6 με 10, καθώς για μικρότερες τιμές επιλέγονται πολλά σημεία που είναι πολύ κοντά μεταξύ τους ενώ για μεγαλύτερες τιμές δεν επιλέγονται στοιχεία που μπορεί να είναι επιθυμητά. Για παράδειγμα για τιμή του minDistance ίση με 2, επιλέγονται σημεία οριακά ταυτόσημα στο λάστιχο του αγροτικού ή στο γήπεδο, ενώ για μεγάλες τιμές δεν επιλέγονται οι άνθρωποι στο πάνω μέρος της εικόνας γιατί είναι κοντά σε πιο ισχυρές γωνίες.

Όσον αφορά το MaxCorners δηλαδή τον μέγιστο αριθμό σημείων (γωνιών), φαίνεται πως χρειάζεται να είναι σχετικά υψηλός, τουλάχιστον 50. Το πρόβλημα είναι πως στο βίντεο μας δεν έχουμε μεγάλη κίνηση. Δεν κινείται κανένα αυτοκίνητο, τα οποία παρουσιάζουν έντονες γωνίες και επιλέγονται πρώτα, με αποτέλεσμα να χρειάζεται η επιλογή μεγάλου πλήθους γωνιών ώστε να προσπαθήσουμε να επιλέξουμε τους ανθρώπους και τα ποδήλατα που εμφανίζονται στο βίντεο. Δυστυχώς ακόμα και με τις επιθυμητές τιμές, οι ποδηλάτες δεν επιλέγονται εύκολα αφού δεν παρουσιάζονται ως δυνατά σημεία γωνιών.

Τέλος, όσον αφορά το QualityLevel χρειάζεται να είναι σχετικά χαμηλά. Οι τιμές πάνω από 0.3 περιορίζουν πολύ τα σημεία και επιλέγουν μόνο μερικά αυτοκίνητα τα οποία όπως είπαμε δεν κινούνται στο βίντεο. Συνεπώς χρειάζεται να έχει χαμηλή τιμή για να επιλέγονται γωνίες που δεν αφορούν αμάξια ή ακίνητα σημεία του γηπέδου.

Αναλυτικότερα:

Harris

Με τη μέθοδο εύρεσης γωνιών του Harris, φαίνεται να μην επιλέγονται πυκνά σημεία. Αυτό έχει σαν αποτέλεσμα να φαίνεται καλύτερη η μέθοδος γιατί με την μείωση του QualityLevel και την αύξηση των γωνιών επιλογής επιλέγονται πάρα πολλά σημεία στα δέντρα και στο γήπεδο με τη μέθοδο των Shi-Tomasi, κάτι που αποφεύγεται με τον Harris.

Shi - Tomasi

Παρατηρούμε πως όσον αφορά το QualityLevel η μέθοδος των Shi-Tomasi επιλέγει πολλά περισσότερα σημεία για μεγάλη τιμή του κατωφλίου. Παραπάνω μπορείτε να συγκρίνετε τη πρώτη φωτογραφί από τη κάθε μέθοδο, που έχουν τις ίδιες παραμέτρους. Φαίνεται πως η μέθοδος αυτή επιλέγει και τα 70 σημεία για τιμή 0.4, ενώ η μέθοδος Harris επιλέγει πολύ λίγα σημεία.

Σύμφωνα με τα παραπάνω επιλέξαμε να έχουμε τις παρακάτω παραμέτρους για να καταφέρουμε να εντοπίσουμε τη πεζό και τον πρώτο ποδηλάτη.

Harris:

```
maxCorners = 210
qualityLevel = 0.03
minDistance = 10
```

Shi-Tomasi:

```
maxCorners = 190
qualityLevel = 0.16
minDistance = 10
```

```
In [1]: ...
### Shi-Tomasi

# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 190,
                      qualityLevel = 0.16,
                      minDistance = 10,
                      blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (9,9),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))

# Create some random colors
color = np.random.randint(0,255,(190,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True

# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)

first_time = 1
printer_check = 0

while(1):
    ret,frame = cap.read()
    if(ret == 0):
        break # reached the end of the video
    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)

    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
```

```

# draw the tracks
for i,(new,old) in enumerate(zip(good_new, good_old)):
    a,b = new.ravel()
    c,d = old.ravel()
    mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
    new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
    img = cv2.add(new_frame,mask)
    printer_check = printer_check + 1
    if (printer_check == 10 or first_time == 1):
        cv2_imshow(img)
        printer_check = 0
        first_time = 0

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
    # Now update the previous frame and previous points
    gray = frame_gray.copy()
    p0 = good_new.reshape(-1,1,2)
    ...

```

```

Out[1]: '''
#### Shi-Tomasi\n\n# params for ShiTomasi corner detection\nfeature_params = dict( maxCorners = 190,\n    qualityLevel = 0.16,\n                minDistance = 10,\n                                blockSize = 7 )\n\n# Parameters f\nor lucas kanade optical flow\nlk_params = dict( winSize = (9,9),\n                maxLevel = 3,\n                    cr\niteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))\n\n# Create some random colors\nicolor = np.random.r\nandint(0,255,(190,3))\ncap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)\nret,frame = cap.read() # get the next frame, ret is T/F\ndepends on if we have a next frame\nnew_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n\n# Take first frame and find corners in it\ngray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\np0 = cv2.goodFeaturesToTrack\n(gray, mask = None, **feature_params) # useHarrisDetector = True\n\n# Create a mask image for drawing purposes\nmask = np.zeros_like(new_frame)\nfirst_time = 1\nprinter_check = 0\nwhile(1):\n    ret,frame = cap.read()\n    if(ret == 0):\n        break # reached the end of the video\n    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\n    # calculate optical flow\n    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)\n    \n    # Select good points\n    if p1 is not Non\n    e:\n        good_new = p1[st==1]\n        good_old = p0[st==1]\n        \n        # draw the tracks\n        for i,(new,old) in enum\nerate(zip(good_new, good_old)):\n            a,b = new.ravel()\n            c,d = old.ravel()\n            mask = cv2.line(mask, (in\nt(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)\n            new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color\n[i].tolist(),-1)\n            img = cv2.add(new_frame,mask)\n            printer_check = printer_check + 1\n            if (printer_check == 10\nor first_time == 1):\n                cv2_imshow(img)\n                printer_check = 0\n                first_time = 0\n            k = cv2.waitKey(30) &\n0xff\n            if k == 27:\n                break\n            # Now update the previous frame and previous points\n            gray = frame_gray.cop\ny()\n            p0 = good_new.reshape(-1,1,2)\n    ...

```

```

In [2]: ...
### Harris

# params for Harris corner detection

feature_params = dict( maxCorners = 210,
                      qualityLevel = 0.03,
                      minDistance = 10,
                      blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (12,12),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))

# Create some random colors
color = np.random.randint(0,255,(210,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True

# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)

first_time = 1
printer_check = 0

while(1):
    ret,frame = cap.read()
    if(ret == 0):
        break # reached the end of the video
    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow

```

```

p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)

# Select good points
if p1 is not None:
    good_new = p1[st==1]
    good_old = p0[st==1]

# draw the tracks
for i,(new,old) in enumerate(zip(good_new, good_old)):
    a,b = new.ravel()
    c,d = old.ravel()
    mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
    new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
img = cv2.add(new_frame,mask)
printer_check = printer_check + 1
if (printer_check == 10 or first_time == 1):
    cv2_imshow(img)
    printer_check = 0
first_time = 0

k = cv2.waitKey(30) & 0xff
if k == 27:
    break
# Now update the previous frame and previous points
gray = frame_gray.copy()
p0 = good_new.reshape(-1,2)
...
'''n### Harris
# params for Harris corner detection
feature_params = dict( maxCorners = 210,
qualityLevel = 0.03, minDistance = 10, blockSize = 7 )
# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (12,12), maxLevel = 3,
criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1) )
# Create some random colors
color = np.random.randint(0,255,(210,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params)
# useHarrisDetector = True
# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)
first_time = 1
printer_check = 0
while(1):
    ret,frame = cap.read()
    if(ret == 0):
        break
    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)
    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
        # draw the tracks
        for i,(new,old) in enumerate(zip(good_new, good_old)):
            a,b = new.ravel()
            c,d = old.ravel()
            mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
            new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
        img = cv2.add(new_frame,mask)
        printer_check = printer_check + 1
        if (printer_check == 10 or first_time == 1):
            cv2_imshow(img)
            printer_check = 0
            first_time = 0
        k = cv2.waitKey(30) & 0xff
        if k == 27:
            break
    # Now update the previous frame and previous points
    gray = frame_gray.copy()
    p0 = good_new.reshape(-1,2)
'''

```

Παρατήρηση Ακμών Συνολικού Βίντεο

4. Να τροποποιήσετε τον κώδικά σας για το βήμα 3, με στόχο την παρακολούθηση γωνιών Harris και Shi-Tomasi που εμφανίστηκαν στο βίντεο μετά το πρώτο frame. Εφαρμόστε τις παραμέτρους που σας έδωσαν το καλύτερο αποτέλεσμα στα βήματα 2 και 3. 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). Καταγράψετε ενδεικτικά screenshots από την εκτέλεση του αλγορίθμου.

Επιλογές παραμέτρων

Να σημειωθεί πως στο συγκεκριμένο ερώτημα εστιάσαμε στη κίνηση του ποδηλάτη στο δεξιό μέρος της εικόνας. Όπως είναι λογικό, όταν ο ποδηλάτης περνάει πίσω από το βανάκι, χάνεται ο εντοπισμός της κίνησης του αφού δεν ανανεώνουμε τα σημεία που παρατηρούμε. Αυτό θα προσπαθήσουμε να επιλύσουμε στο επόμενο ερώτημα όπως ζητείται.

Όσον αφορά το winSize, παρατηρούμε πως για μεγάλα παράθυρα χάνεται η αίσθηση της κίνησης. Για παράδειγμα το σημείο που ακολουθεί τον ποδηλάτη φαίνεται να μη μπορεί να τον ακολουθήσει για μεγάλο χρονικό διάστημα. Από την άλλη μεριά τα μικρά παράθυρα έχουν ως αποτέλεσμα να ακολουθείται πιστά η επιθυμητή κίνηση. Να σημειωθεί όμως πως τα μικρά παράθυρα είναι ευαίσθητα στον θόρυβο ενώ τα μεγάλα παράθυρα το αντιμετωπίζουν αυτό. Στις φωτογραφίες WSmall, WBig βλέπετε τη σύγκριση ενός μικρού και ενός μεγάλου παραθύρου διατηρώντας τα άλλα χαρακτηριστικά ίδια.

Όσον αφορά το μέγιστο επίπεδο της πυραμίδας δε παρατηρείται μεγάλη διαφορά (στην μικρή κίνηση που έχει η εικόνα μας) για τιμές 3 ως 5. Βέβαια όπως αναφέρεται και στη δημοσίευση των Jean-Yves Bouguet δεν έχει ιδιαίτερο νόημα να πάμε πάνω από το επίπεδο 4 ειδικά για εικόνες χαμηλής ανάλυσης όπως οι δικές μας. Από την άλλη παρατηρείται πρόγματι διαφορά μεταξύ των επιπέδων 0 (χωρίς χρήση πυραμίδας) και επιπέδου 5 όπως φαίνεται στις εικόνες Level0, Level5. Η διαφορά αυτή, που προκαλεί μια

περίεργη συμπεριφορά στο σημείο που παρακολουθεί τον ποδηλάτη στα δεξιά της εικόνας, παρατηρήθηκε για μικρά παράθυρα (5x5) ενώ για παράθυρα (15x15) δεν παρατηρήσαμε κάποια διαφορά.

Τέλος, όσον αφορά τα κριτήρια δε παρατηρήσαμε κάποια διαφορά. Βέβαια έχουμε πολύ περιορισμένη κίνηση στο βίντεο μας, οπότε ενδέχεται να μην είναι αντιπροσωπευτικό. Η επιλογή του μέγιστου αριθμού επαναλήψεων πριν τον τερματισμό του αλγορίθμου καθώς και της τιμής του epsilon ως threshold για τον τερματισμό του αλγορίθμου προσφέρουν μεγαλύτερη ακρίβεια καθυστερώντας όμως την υλοποίηση. Για τον λόγο αυτό επιλέξαμε τις τιμές λίγο μεγαλύτερες από τις προτεινόμενες.

Αναλυτικότερα:

Harris

Η μέθοδος Harris φαίνεται να εντοπίζει καλύτερα αρκετές κινήσεις και να περγράφει πιο εύκολα της πορεία τους χωρίς να παρουσιάζει σφάλματα όπως κάποια μικρά σφάλματα που φαίνεται να έχει η μέθοδος Shi-Tomasi. Από την άλλη η μέθοδος Harris δεν μπορεί να εντοπίσει πλήρως τη κίνηση του πρώτου ποδηλάτη μέχρι το βανάκι με όποιες παραμέτρους δοκιμάσαμε, σε αντίθεση με τη μέθοδο Shi-Tomasi που τον εντοπίζει πολύ επιτυχημένα. Να σημειωθεί ακόμα πως η μέθοδος Harris χρειάζεται μεγαλύτερο παράθυρο από τη Shi-Tomasi γιατί για μικρά παράθυρα επιτρέπεται με περίεργο τρόπο και εμφανίζει κινήσεις που δεν υπήρχαν.

Shi - Tomasi

Στη μέθοδο των Shi-Tomasi το μικρό παράθυρο του (9, 9) με επίπεδο πυραμίδας μεγαλύτερο του 3 φάνηκε να εντοπίζει σωστά τα λίγα σημεία που έχουμε ανιχνεύσει και κινούνται στο πρώτο μισό του βίντεο. Όσον αφορά τα κριτήρια δεν παρατηρήσαμε κάποια διαφορά στα 3 σημεία που κινούνται στην εικόνα μας, όμως ενδέχεται να μην είναι αντιπροσωπευτικό το βίντεο γιατί έχουμε πολύ λίγα κινούμενα σημεία τα οποία μάλιστα δεν αποτελούν τις ισχυρότερες γωνίες οι οποίες είναι τα αυτοκίνητα. Συνεπώς επιλέξαμε τα ακόλουθα χαρακτηριστικά για το επόμενο ερώτημα.

Σύμφωνα με τα παραπάνω επιλέξαμε να έχουμε τις παρακάτω παραμέτρους για να καταφέρουμε να εντοπίσουμε τη πεζό και τον πρώτο ποδηλάτη.

Harris:

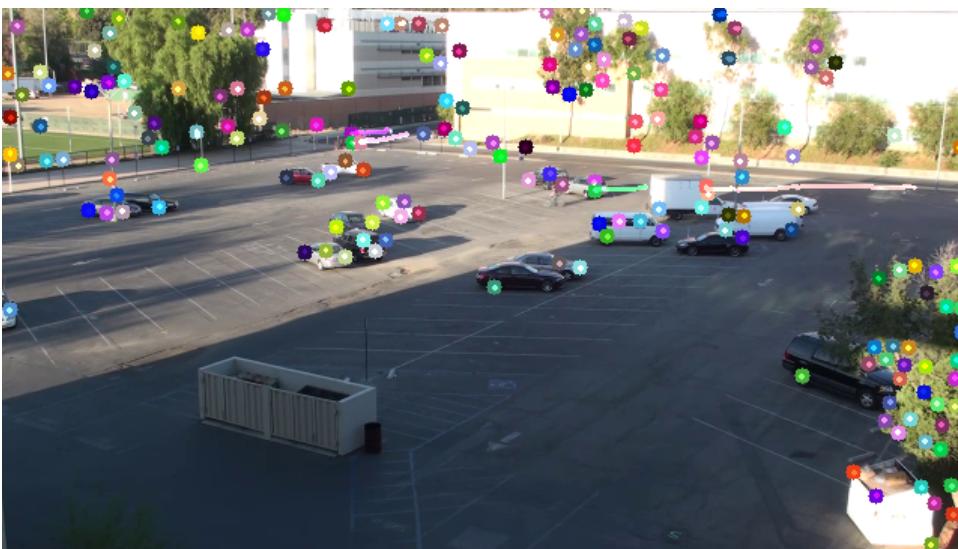
```
winSize = (12, 12)
maxLevel = 3
TERM_CRITERIA_EPS = 0.1
TERM_CRITERIA_COUNT = 20
```

Shi-Tomasi:

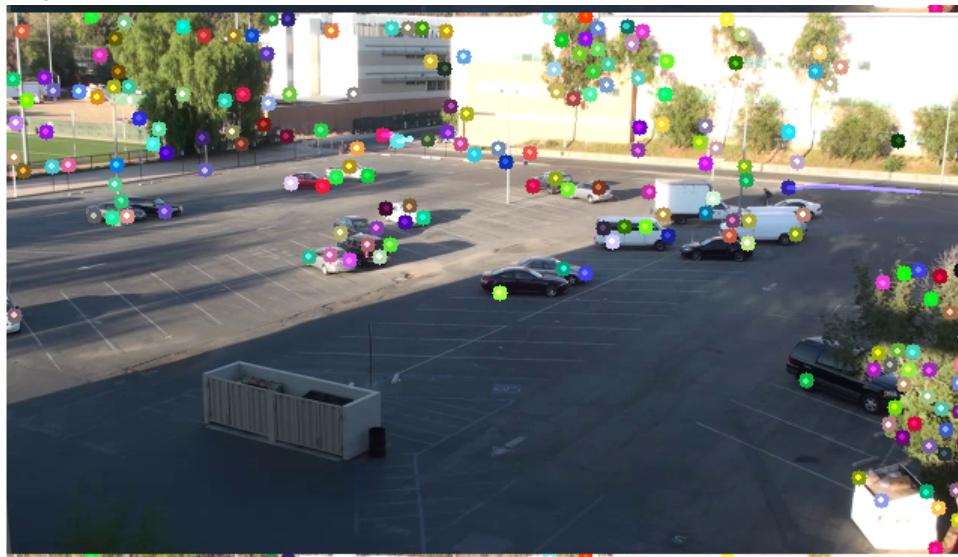
```
winSize = (9, 9)
maxLevel = 3
TERM_CRITERIA_EPS = 0.1
TERM_CRITERIA_COUNT = 20
```

Στις παρακάτω φωτογραφίες Harris, Shi-Tomasi, μπορείτε να δείτε τις τελικές κινήσεις για το συνολικό βίντεο με τις επιλεγμένες παραμέτρους. Δεν πραγματοποιούμε επανεντοπισμό των σημείων οπότε τη κίνηση του δεύτερου ποδηλάτη την ακολουθούμε οριακά καθώς τυχαίνει να περνάει από σημείο που έχει ηδη εντοπιστεί. Βλέπουμε με τη μέθοδο Harris να είναι πιο ομοιόμορφη η παρακολούθηση του δεύτερου ποδηλάτη, όμως η παρακολούθηση του πρώτου ποδηλάτη όπως και κάποιων πεζών χάνεται και δεν εντοπίζεται πλήρως.

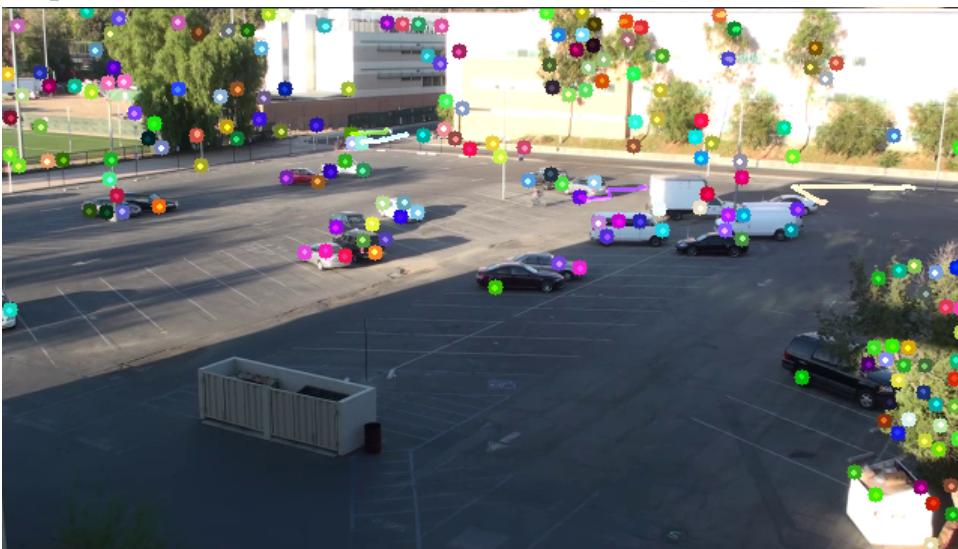
WSmall:



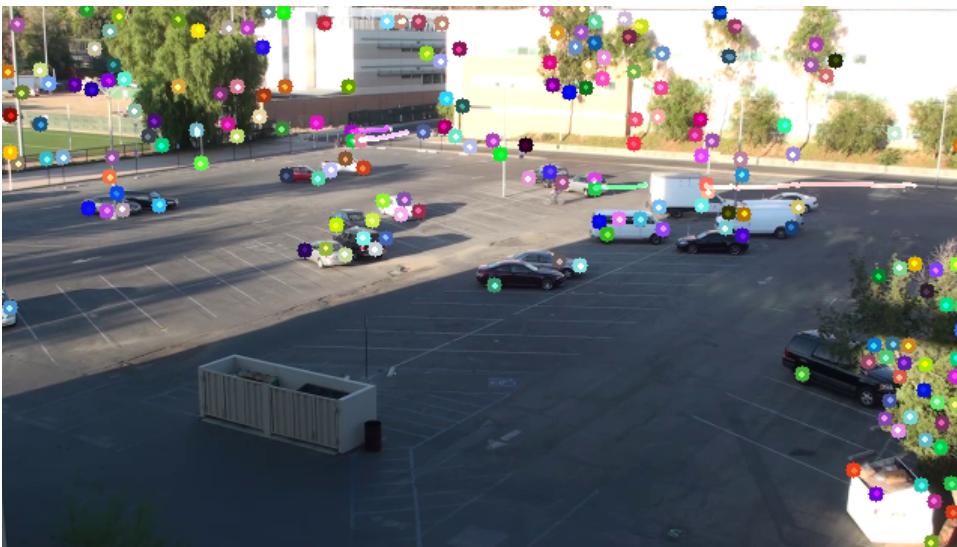
Wbig:



Level_0:



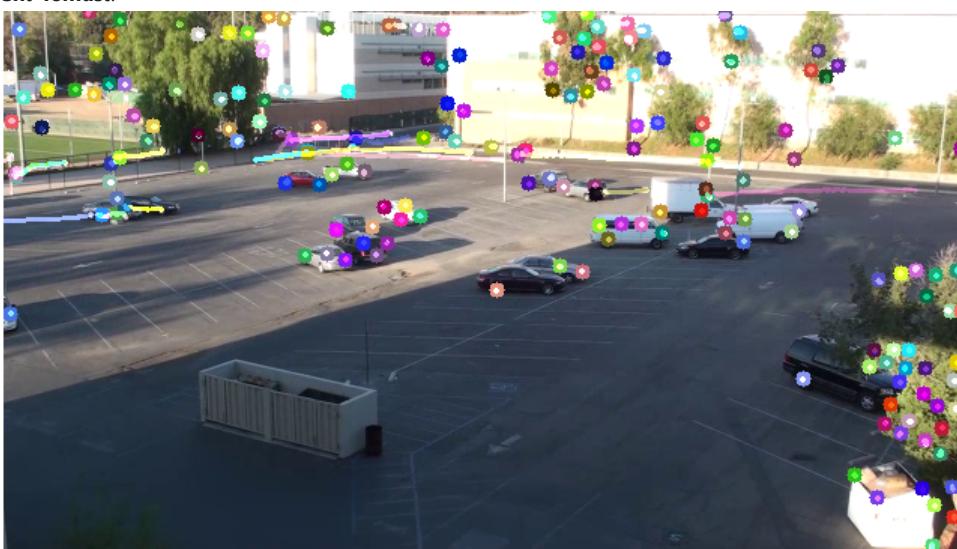
Level_5:



Harris:



Shi-Tomasi:



```
In [ ]: import math
```

```
In [3]: ### Shi-Tomasi  
...  
# params for ShiTomasi corner detection
```

```

feature_params = dict( maxCorners = 190,
                      qualityLevel = 0.16,
                      minDistance = 10,
                      blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (9,9),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
color = np.random.randint(0,255,(190,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True

# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)

first_time = 1
printer_check = 0
count_frames = 0

while(1):
    if (count_frames == 60): # approx 0.5 second
        count_frames = 0
        p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True
        mask = np.zeros_like(new_frame)

    ret,frame = cap.read()
    count_frames = count_frames + 1

    if (ret == 0):
        break # reached the end of the video
    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)

    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
        ##### keep old points
        save_old = good_old.reshape(-1,1,2)
        p0 = good_new.reshape(-1,1,2)

    ### Keep only moving points ###

    del_list = []
    track = 0
    for i in p0:
        if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and
            math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):
            del_list.append(track)
        track = track + 1

    del_list.reverse()
    for i in del_list:
        p0 = np.delete(p0, i, 0)
        save_old = np.delete(save_old, i, 0)

    good_old = save_old.reshape(len(save_old), 2)
    good_new = p0.reshape(len(p0), 2)

    # draw the tracks
    for i,(new,old) in enumerate(zip(good_new, good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
        new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
    img = cv2.add(new_frame,mask)
    printer_check = printer_check + 1
    if (printer_check == 10 or first_time == 1):
        cv2_imshow(img)

```

```

    printer_check = 0
    first_time = 0

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

    # Now update the previous frame and previous points
    gray = frame_gray.copy()
    #p0_test = good_new.reshape(-1,1,2)
    ...

```

```

Out[3]: '\n# params for ShiTomasi corner detection\nfeature_params = dict( maxCorners = 190, \n                                qualityLevel = 0.16,\n                                minDistance = 10,\n                                blockSize = 7 )\n\n# Parameters for lucas kanade optical flow\nlk_params = dict( winSize = (9,9),\n                                maxLevel = 3,\n                                criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))\n\n# Create some random colors\nicolor = np.random.randint(0,255,(190,3))\ncap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)\nret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame\nnew_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n\n# Take first frame and find corners in it\ngray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\np0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True\n\n# Create a mask image for drawing purposes\nmask = np.zeros_like(new_frame)\n\nif first_time == 0:\n    nprintter_check = 0\n    count_frames = 0\n    p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True\n    mask = np.zeros_like(new_frame)\n    ret,frame = cap.read()\n    count_frames = 1\n\n    if (ret == 0):\n        break # reached the end of the video\n    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\n\n    # calculate optical flow\n    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)\n\n    # Select good points\n    if p1 is not None:\n        good_new = p1[st==1]\n        good_old = p0[st==1]\n\n        keep old points\n        save_old = good_old.reshape(-1,1,2)\n        p0 = good_new.reshape(-1,1,2)\n\n        ## Keep only moving points\n        del_list = []\n        track = 0\n        for i in p0:\n            if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0]))) and\n                (math.ceil(i[0][0])) and\n                (math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):\n                del_list.append(track)\n\n            track = track + 1\n\n        del_list.reverse()\n        for i in del_list:\n            p0 = np.delete(p0, i, 0)\n\n        save_old = np.delete(save_old, i, 0)\n\n        good_old = save_old.reshape(len(save_old), 2)\n        good_new = p0.reshape(len(p0), 2)\n\n        # draw the tracks\n        for i,(new,old) in enumerate(zip(good_new, good_old)):\n            a,b = new.ravel()\n            c,d = old.ravel()\n            mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)\n\n        new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)\n\n        img = cv2.add(new_frame,mask)\n\n        printer_check = printer_check + 1\n\n        if (printer_check == 10 or first_time == 1):\n            cv2.imshow(img)\n            printer_check = 0\n            first_time = 0\n\n        k = cv2.waitKey(30) & 0xff\n        if k == 27:\n            break\n\n    # Now update the previous frame and previous points\n    gray = frame_gray.copy()\n    #p0_test = good_new.reshape(-1,1,2)\n'

```

```

In [4]: ...
### Harris

# params for Harris corner detection

feature_params = dict( maxCorners = 210,
                      qualityLevel = 0.03,
                      minDistance = 10,
                      blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (12,12),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))

# Create some random colors
color = np.random.randint(0,255,(210,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True

# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)

first_time = 1
printer_check = 0
count_frames = 0

while(1):
    if (count_frames == 60): # approx 0.5 second
        count_frames = 0
        p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True
        mask = np.zeros_like(new_frame)

    ret,frame = cap.read()

```

```

count_frames = count_frames + 1

if (ret == 0):
    break # reached the end of the video
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))
frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
# calculate optical flow
p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)

# Select good points
if p1 is not None:
    good_new = p1[st==1]
    good_old = p0[st==1]
##### keep old points
save_old = good_old.reshape(-1,1,2)
p0 = good_new.reshape(-1,1,2)

### Keep only moving points ###

del_list = []
track = 0
for i in p0:
    if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and
        math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):
        del_list.append(track)
    track = track + 1

del_list.reverse()
for i in del_list:
    p0 = np.delete(p0, i, 0)
    save_old = np.delete(save_old, i, 0)

good_old = save_old.reshape(len(save_old), 2)
good_new = p0.reshape(len(p0), 2)

# draw the tracks
for i,(new,old) in enumerate(zip(good_new, good_old)):
    a,b = new.ravel()
    c,d = old.ravel()
    mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
    new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
img = cv2.add(new_frame,mask)
printer_check = printer_check + 1
if (printer_check == 10 or first_time == 1):
    cv2_imshow(img)
    printer_check = 0
    first_time = 0

k = cv2.waitKey(30) & 0xff
if k == 27:
    break

# Now update the previous frame and previous points
gray = frame_gray.copy()
#p0_test = good_new.reshape(-1,1,2)
...

```

```

Out[4]: ''' Harris\n#\n# params for Harris corner detection\n\nfeature_params = dict( maxCorners = 210,\n    qualityLevel = 0.03,\n                minDistance = 10,\n                                blockSize = 7 )\n\n# Parameters for\nor lucas kanade optical flow\nlk_params = dict( winSize = (12,12),\n                                         maxLevel = 3,\n                                         criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))\n\n# Create some random colors\nicolor = np.random.randint(0,255,(210,3))\ncap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)\nret,frame = cap.read() # get the next frame, ret is\nT/F depends on if we have a next frame\nnew_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n\n# Take first frame and find corners in it\ngray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\nnp0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True\n\n# Create a mask image for drawing purposes\nmask = np.zeros_like(new_frame)\n\nfirst_time = 1\nprinter_check = 0\nnframes = 0\nwhile (1):\n    if (count_frames == 60): # approx 0.5 second\n        count_frames = 0\n        p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True\n        mask = np.zeros_like(new_frame)\n\n        ret,frame = cap.read()\n        count_frames = count_frames + 1\n        if (ret == 0):\n            break # reached the end of the video\n        new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n        frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\n\n        # calculate optical flow\n        p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)\n\n        # Select good points\n        if p1 is not None:\n            good_new = p1[st==1]\n            good_old = p0[st==1]\n\n            ##### keep old points\n            save_old = good_old.reshape(-1,1,2)\n            p0 = good_new.reshape(-1,1,2)\n\n            #### Keep only moving points ####\n            del_list = []\n            track = 0\n            for i in p0:\n                if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and\n                    math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):\n                    del_list.append(track)\n            track = track + 1\n            del_list.reverse()\n            for i in del_list:\n                p0 = np.delete(p0, i, 0)\n                save_old = np.delete(save_old, i, 0)\n            good_old = save_old.reshape(len(save_old), 2)\n            good_new = p0.reshape(len(p0), 2)\n\n            # draw the tracks\n            for i,(new,old) in enumerate(zip(good_new, good_old)):\n                a,b = new.ravel()\n                c,d = old.ravel()\n                mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)\n                new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)\n\n            img = cv2.add(new_frame,mask)\n            printer_check = printer_check + 1\n            if (printer_check == 10 or first_time == 1):\n                cv2.imshow(img)\n                printer_check = 0\n            first_time = 0\n            k = cv2.waitKey(30) & 0xff\n            if k == 27:\n                break\n\n        # Now update the previous frame and previous points\n        gray = frame_gray.copy()\n        #p0_test = good_new.reshape(-1,1,2)\n

```

Σχολιασμός

Να σημειωθεί πως στο συγκεκριμένο ερώτημα εστιάσαμε ξανά στη κίνηση του ποδηλάτη στο δεξιό μέρος της εικόνας καθώς και σε κάποιους πεζούς στο πάνω αριστερό μέρος της εικόνας όπως και δύο ακόμα ποδηλάτες που εισέρχονται στην εικόνα.

Αρχικά και στις δύο μεθόδους βρίσκουμε τα κρίσιμα σημεία, τις γωνίες που παρακολουθούμε. Στη συνέχεια εντοπίζουμε τη κίνηση και ζωγραφίζουμε μόνο τα σημεία που κινούνται τουλάχιστον ενα πιξελ. Δυστυχώς από την ανάλυση της εικόνας φαίνεται σαν να κινούνται όλα τα σημεία που εμφανίζονται στη προσομοίωση παρ'ότι στην πραγματικότητα δε κινούνται, όπως για παράδειγμα τα παρκαρισμένα αυτοκίνητα. Πράγματι κινούνται τα σημεία αυτά όπως φαίνεται από τις τιμές των πινάκων.

Στη συνέχεια κάθε 60 frames δηλαδή κάθε περίπου δύο δευτερόλεπτα ανανεώνουμε τα σημεία που παρατηρούμε και μηδενίζουμε τη μάσκα ώστε να καταγράψουμε νέα σημεία.

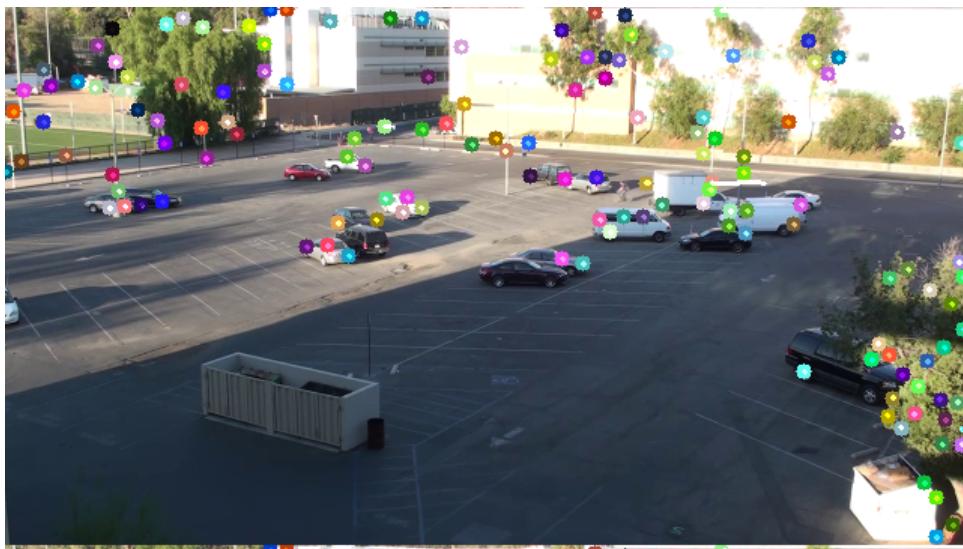
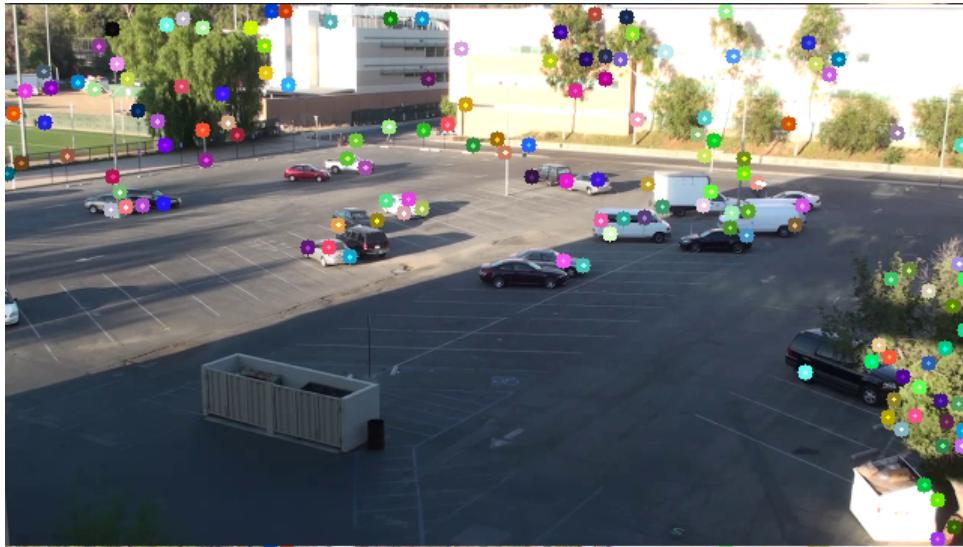
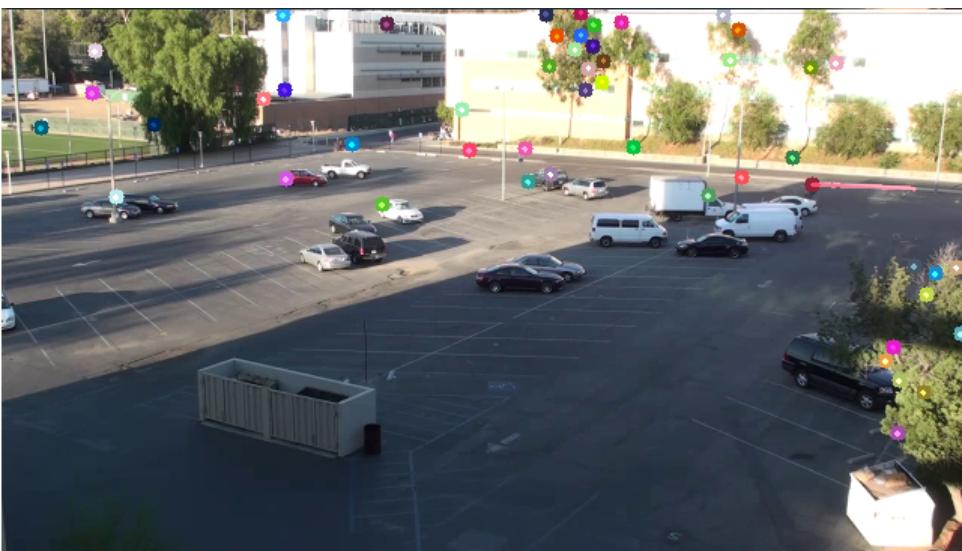
Για να κρατήσουμε μόνο τα σημεία που κινούνται, από τα καλά σημεία ελέγχουμε αν είναι ίσα τα παλιά με τα νέα σημεία και κρατάμε μόνο αυτά που δεν είναι ίσα.

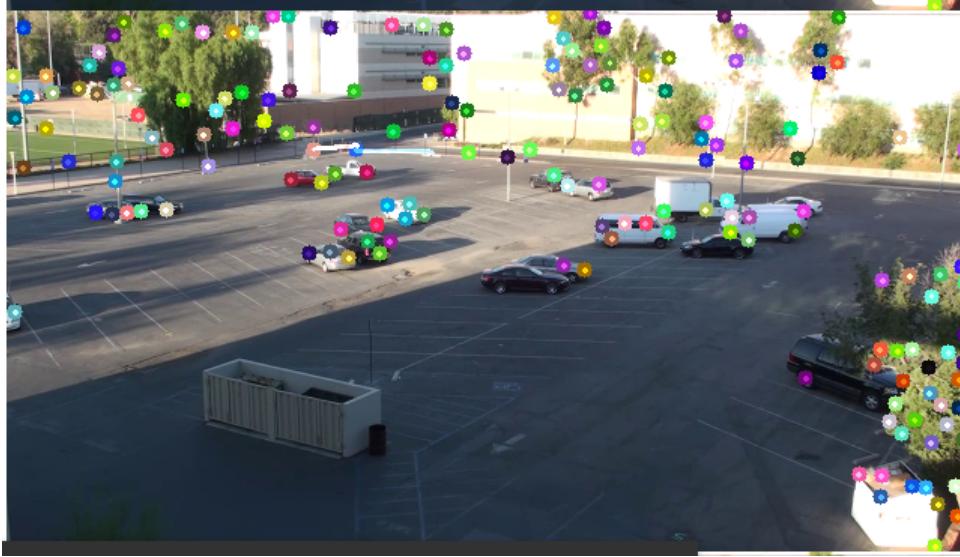
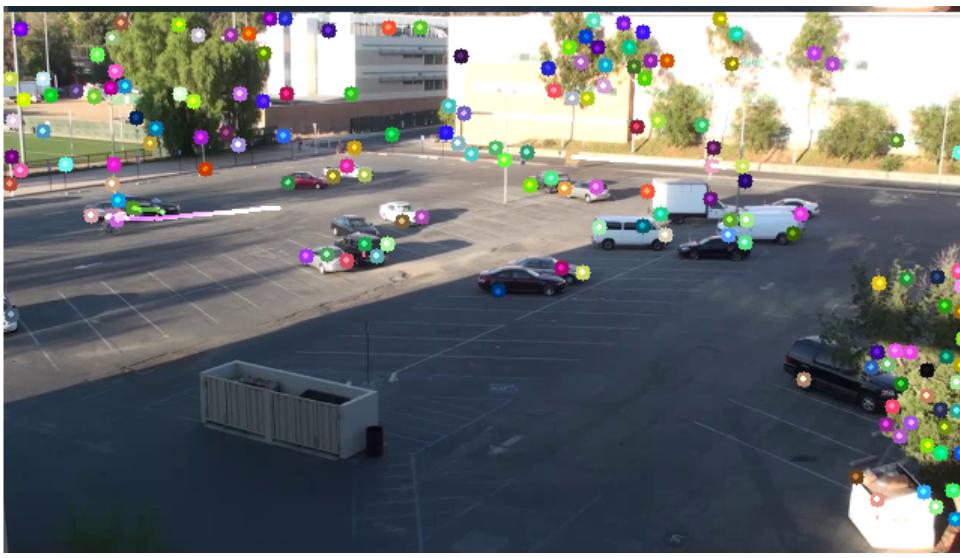
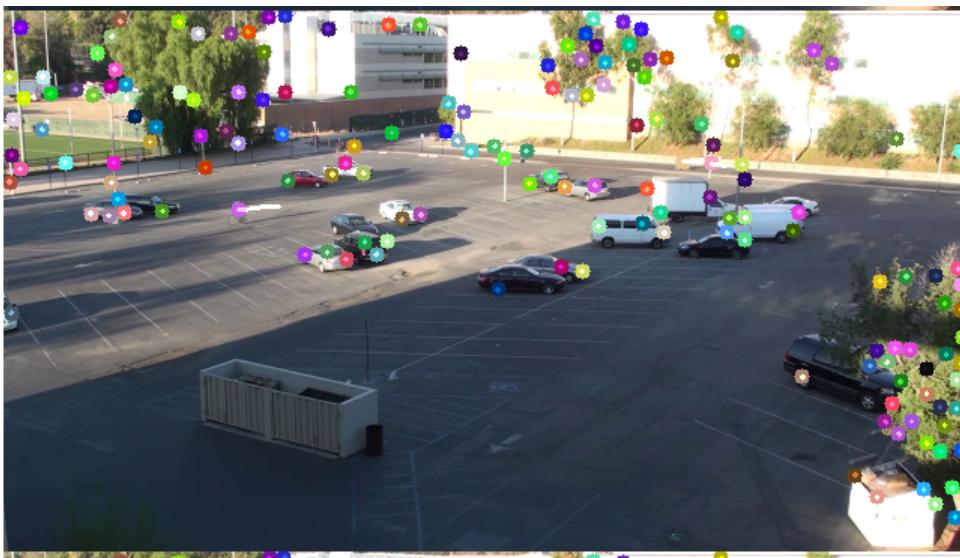
Να σημειωθεί πως η εικόνα μας έχει πολύ περιορισμένη κίνηση και μάλιστα με μικρά και θολά (λόγω της ανάλυσης) χαρακτηριστικά. Συνεπώς δεν εντοπίζεται όλη η κίνηση και δεν είναι πολύ ξεκάθαρη η κίνηση αν δε γνωρίζουμε τι υπάρχει στην εικόνα. Για αυτόν τον λόγο τρέξαμε το προγραμμά μας και με ενα από τα βίντεο του εργαστηρίου για να δείξουμε τη κίνηση που εντοπίζεται πολύ καλύτερα και είναι πιο αντιπροσωπευτική.

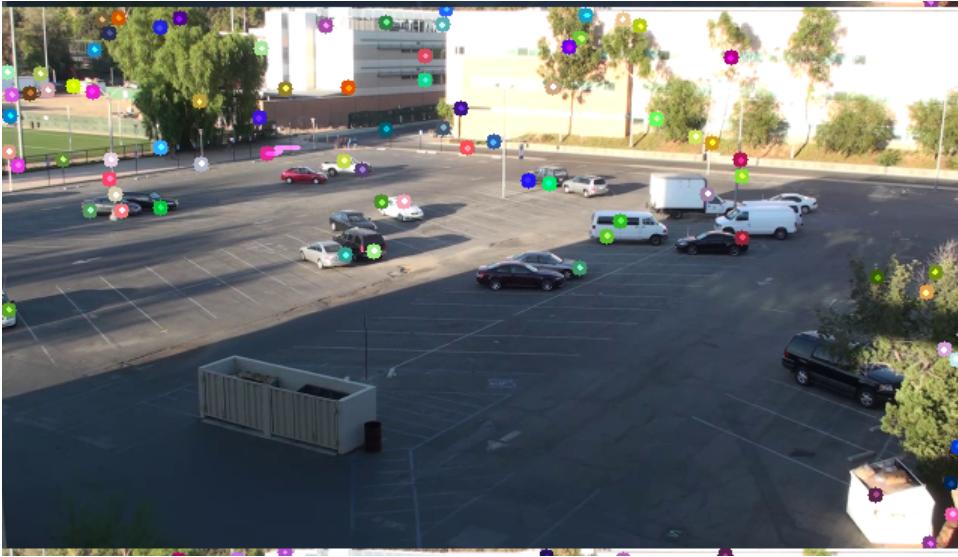
Όσον αφορά τις μεθόδους Harris και Shi-Tomasi, χρησιμοποιήσαμε τα χαρακτηριστικά που επιλέχθηκαν στα προηγούμενα ερωτήματα. Η μέθοδος Harris φαίνεται να μην εντοπίζει πολύ εύκολα τους ποδηλάτες με αποτέλεσμα να χάνει μεγάλο μέρος της κίνησης. Εκτός από αυτό τα υπόλοιπα στοιχεία φαίνονται παρόμοια και για τις δύο μεθόδους.

Παρακάτω φαίνονται εικόνες για τη κάθε μέθοδο καθώς και για το βίντεο από την εργαστηριακή διάλεξη. Στις εικόνες φαίνεται πως αρχικά εντοπίζεται η κίνηση στον ποδηλάτη και ακολουθείτε για λίγο. Στη συνέχεια παρατηρούμε πως η κίνηση που είχε καταγραφθεί διαγράφεται καθώς εντοπίζονται τα καινούργια σημεία και εντοπίζεται η νεα κίνηση. Με αυτόν τον τρόπο βλέπουμε πως μπορεί να εντοπιστεί ξανά η συνέχεια της κίνησης του ποδηλάτη, κάτι που δε μπορούσε να γίνει στο ερώτημα 3. Έπειτα βλέπουμε και τη πεζό που κινείται στο πάνω μέρος της εικόνας όπως και τον καινούργιο ποδηλάτη που εισέρχεται από δεξιά της εικόνας. Δυστυχώς ο ποδηλάτης αυτός εντοπίζεται και από τις δύο μεθόδους όταν βρίσκεται κοντά στην αριστερόστροφη στροφή, δηλαδή χάνεται η αρχική του κίνηση.

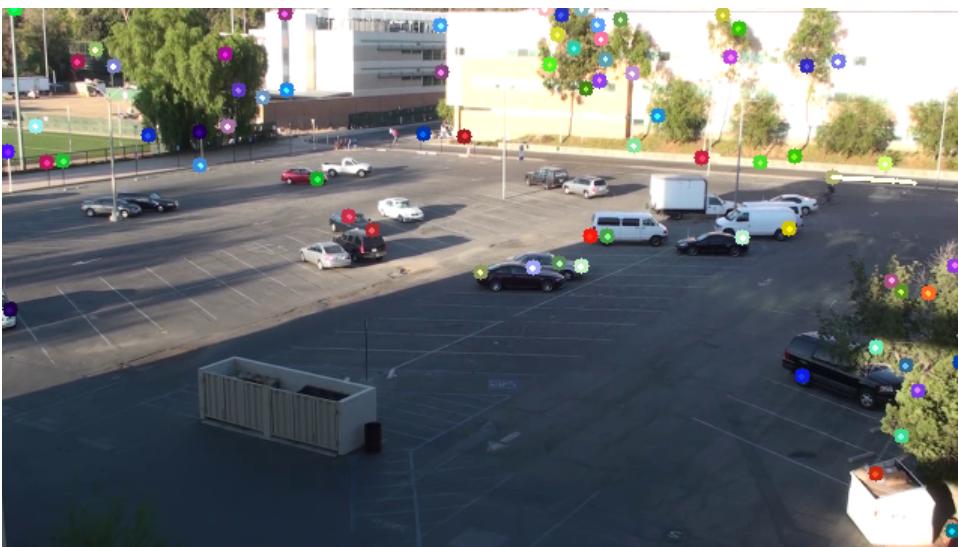
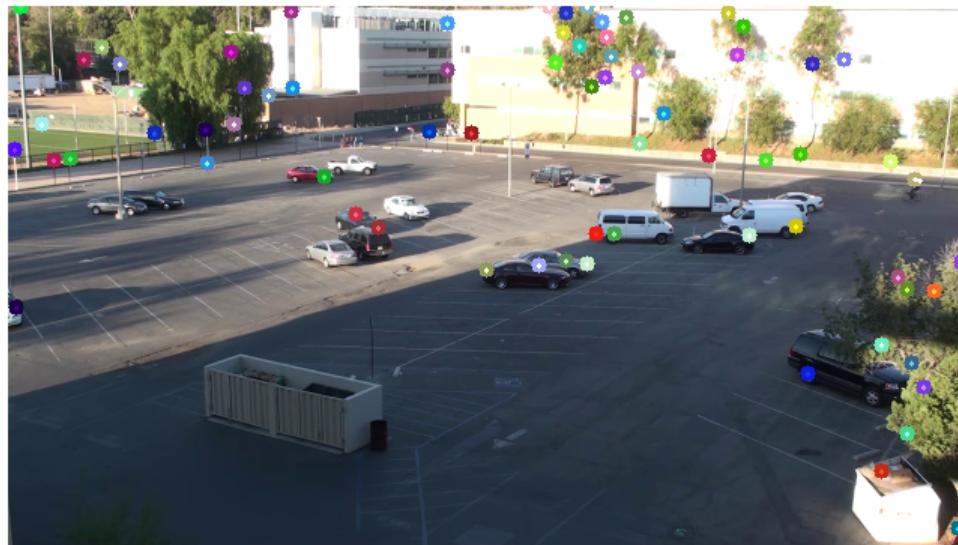
Shi-Tomasi

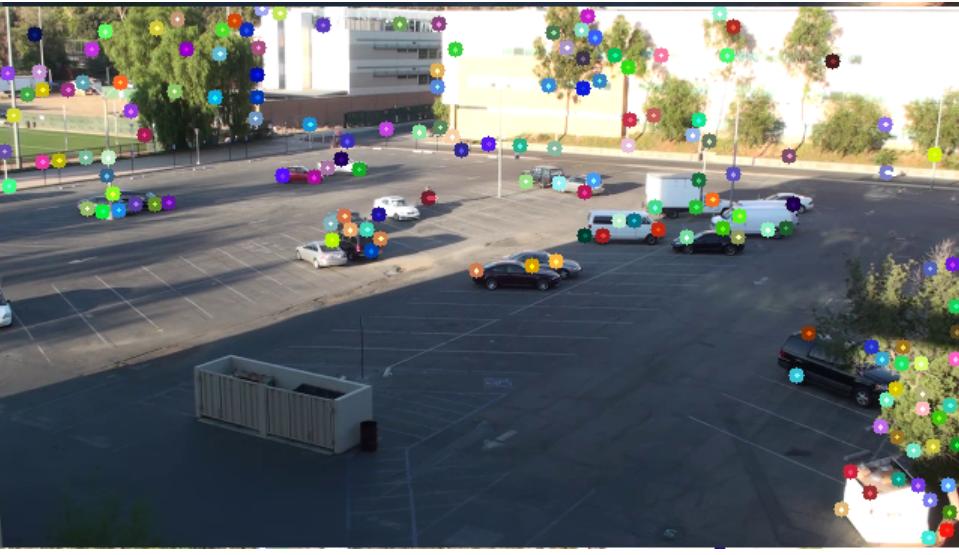
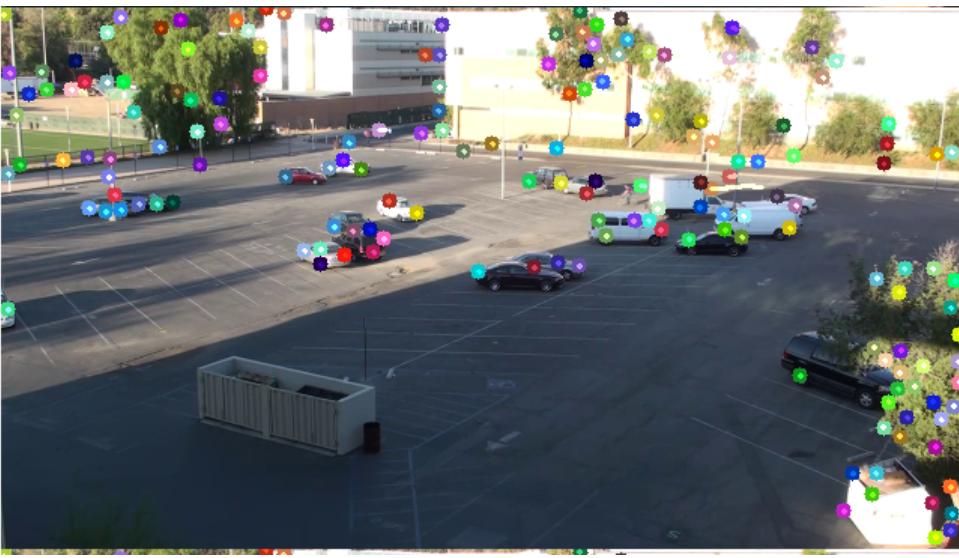
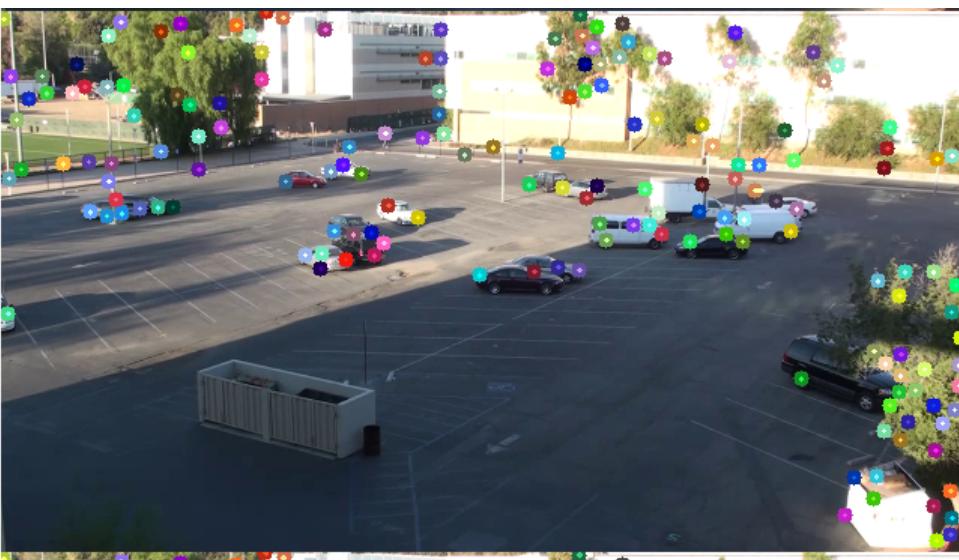


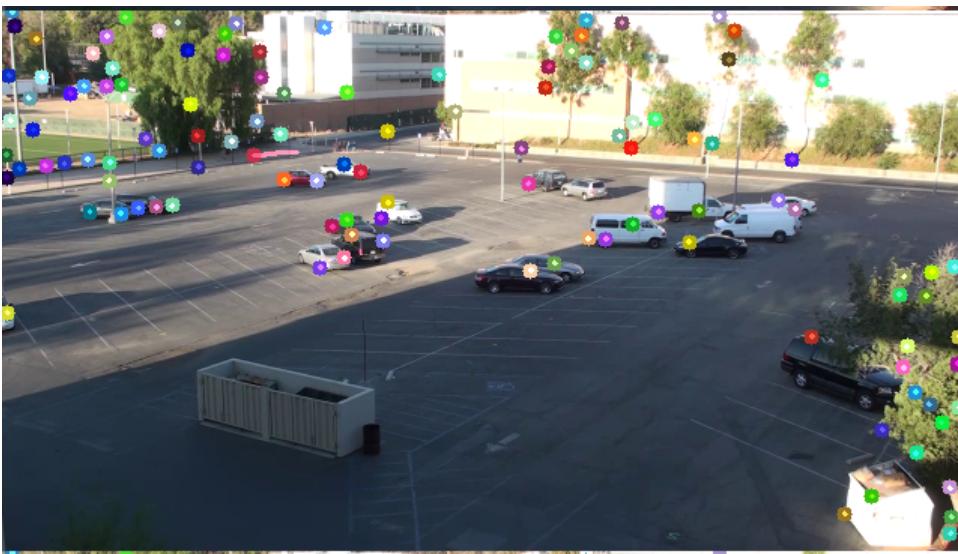
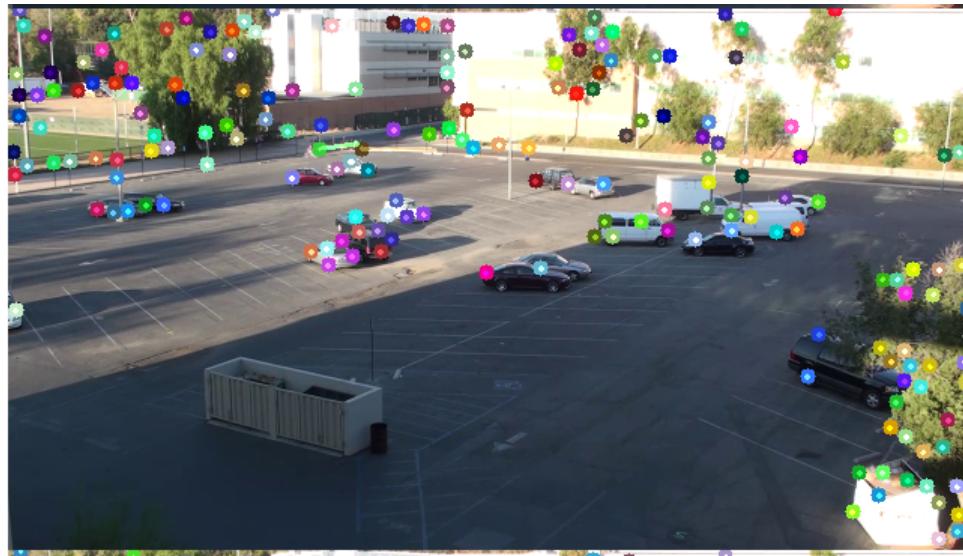
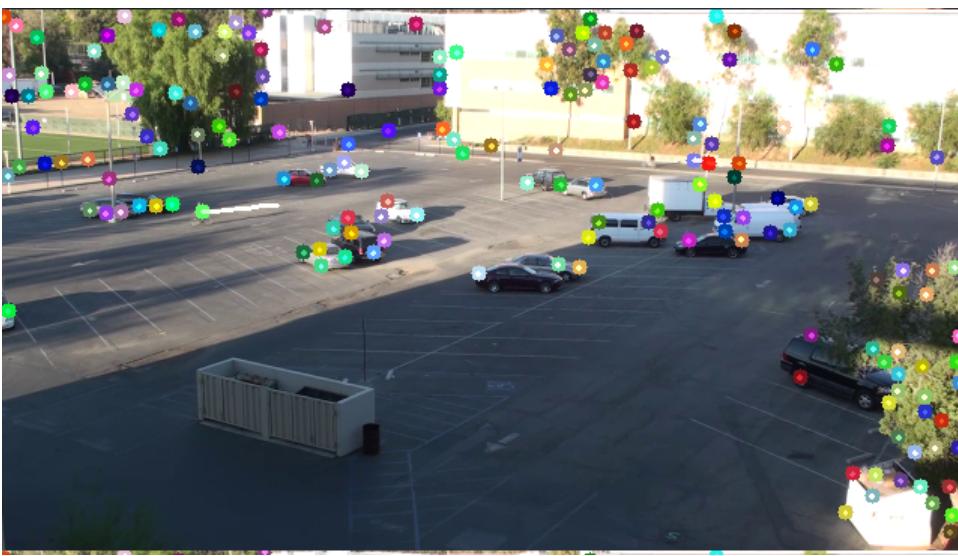




Harris



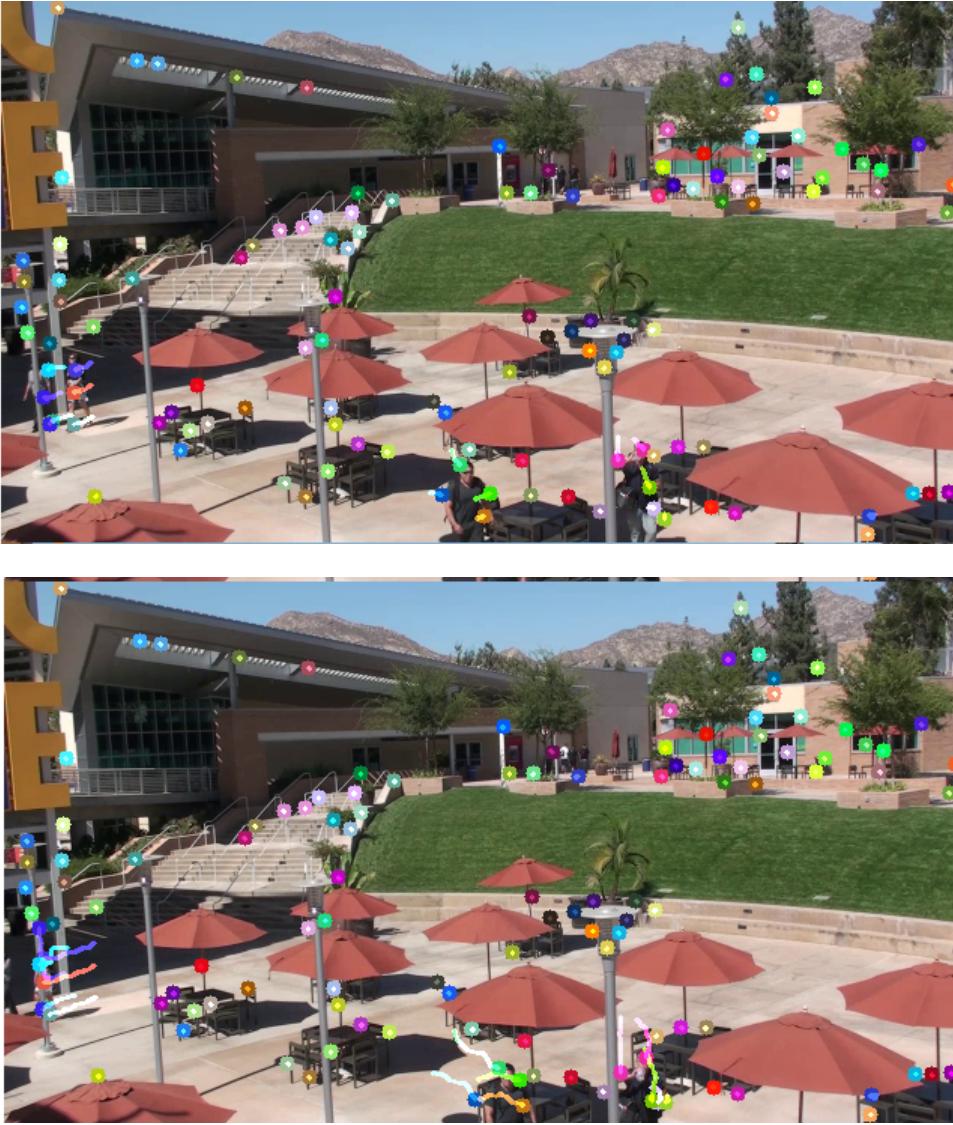




Εργαστηριακό Βίντεο







Θόρυβος

1. Να τροποποιήσετε τον κώδικά σας για το βήμα 4, εισάγοντας Salt and Pepper θόρυβο (δοκιμάστε 2 διαφορετικές τιμές) μετά από κάθε "διάβασμα" ενός frame. Καταγράψετε ενδεικτικά frames και εξηγήστε την επίδραση του θορύβου salt and pepper στο αποτέλεσμα του motion estimation με optical flow.*

```
In [ ]: import math
from skimage.util import random_noise
```

Shi - Tomasi

```
In [5]: ### Shi-Tomasi with noise
...
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 190,
                       qualityLevel = 0.16,
                       minDistance = 10,
                       blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (9,9),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
color = np.random.randint(0,255,(190,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)

ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
```

```

new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

## ADD NOISE
new_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)
new_frame = np.array(255 * new_frame, dtype = 'uint8')

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True

# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)

first_time = 1
printer_check = 0
count_frames = 0

while(1):
    if (count_frames == 60): # approx 0.5 second
        count_frames = 0
    p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True
    mask = np.zeros_like(new_frame)

    ret,frame = cap.read()
    count_frames = count_frames + 1

    if(ret == 0):
        break # reached the end of the video
    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

    ## ADD NOISE
    new_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)
    new_frame = np.array(255 * new_frame, dtype = 'uint8')

    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)

    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
        ##### keep old points
        save_old = good_old.reshape(-1,1,2)
        p0 = good_new.reshape(-1,1,2)

    ### Keep only moving points ###

    del_list = []
    track = 0
    for i in p0:
        if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and
            math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):
            del_list.append(track)
        track = track + 1

    del_list.reverse()
    for i in del_list:
        p0 = np.delete(p0, i, 0)
        save_old = np.delete(save_old, i, 0)

    good_old = save_old.reshape(len(save_old), 2)
    good_new = p0.reshape(len(p0), 2)

    # draw the tracks
    for i,(new,old) in enumerate(zip(good_new, good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
        new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
    img = cv2.add(new_frame,mask)
    printer_check = printer_check + 1
    if (printer_check == 10 or first_time == 1):
        cv2_imshow(img)
        printer_check = 0
        first_time = 0

    k = cv2.waitKey(30) & 0xff
    if k == 27:

```

```

break

# Now update the previous frame and previous points
gray = frame_gray.copy()
#p0_test = good_new.reshape(-1,1,2)
...

```

Out[5]:

```

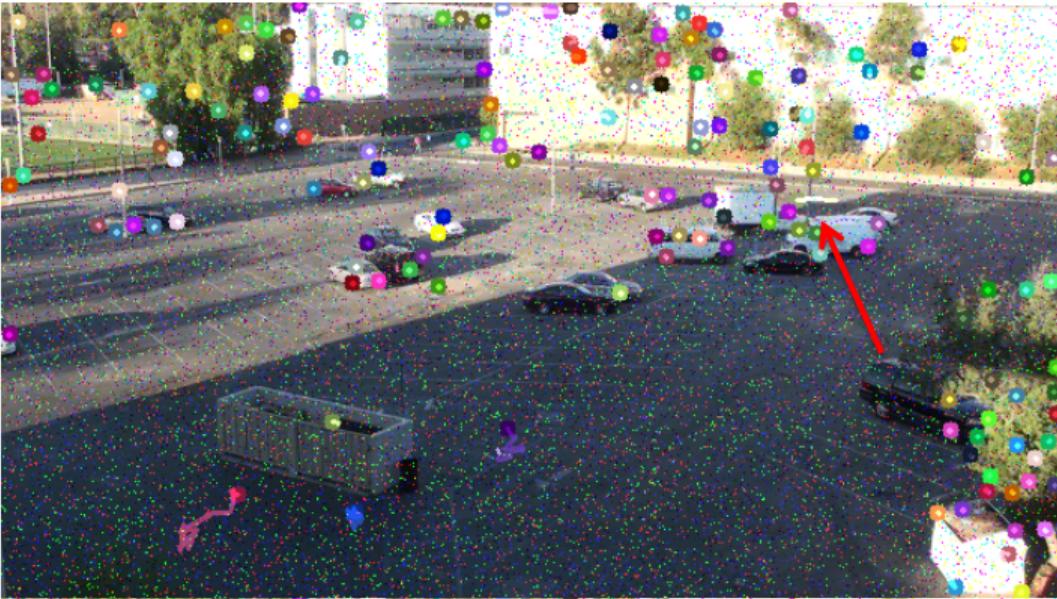
"\n# params for ShiTomasi corner detection\nfeature_params = dict( maxCorners = 190,\n                                qualityLevel\n1 = 0.16,\n                                minDistance = 10,\n                                blockSize = 7 )\n\n# Parameters for lucas ka\nnade optical flow\nlk_params = dict( winSize = (9,9),\n                                maxLevel = 3,\n                                criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))\n\n# Create some random colors\nncolor = np.random.randint(0,255,(190,3))\ncap.set(1, cap.get(cv2.CAP_PROP_FRAME_WIDTH)*9)\nret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame\nnew_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\nn## ADD NOISE\nnew_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)\nnew_frame = np.array(255 * new_frame, dtype = 'uint8')\n\n# Take first frame and find corners in it\ngray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\np0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True\n\n# Create a mask image for drawing purposes\nmask = np.zeros_like(new_frame)\nnfirst_time = 1\nnprinter_check = 0\nncount_frames = 0\nnwhile(1):\n    if (count_frames == 60): # approx 0.5 second\n        count_frames = 0\n        p0 = cv2.goodFeaturesToTrack(gray, mask = None, **feature_params) # useHarrisDetector = True\n        mask = np.zeros_like(new_frame)\nn        ret,frame = cap.read()\nn        count_frames = count_frames + 1\n        if(ret == 0):\n            break # reached the end of the video\n        new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\nn## ADD NOISE\n        new_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)\n        new_frame = np.array(255 * new_frame, dtype = 'uint8')\n        frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\n\n        # calculate optical flow\n        p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)\n\n        # Select good points\n        if p1 is not None:\n            good_new = p1[st==1]\n            good_old = p0[st==1]\n\n            #####\n            # keep old points\n            save_old = good_old.reshape(-1,1,2)\n            p0 = good_new.reshape(-1,1,2)\n\n            ####\n            # Keep only moving points ####\n            del_list = []\n            track = 0\n            for i in p0:\n                if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and \n                    math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):\n                    del_list.append(track)\n                    track = track + 1\n            del_list.reverse()\n            for i in del_list:\n                p0 = np.delete(p0, i, 0)\n                save_old = np.delete(save_old, i, 0)\n                good_old = save_old.reshape(len(save_old), 2)\n                good_new = p0.reshape(len(p0), 2)\n\n                # draw the tracks\n                for i,(new,old) in enumerate(zip(good_new, good_old)):\n                    a,b = new.ravel()\n                    c,d = old.ravel()\n                    mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)\n\n                new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)\n                img = cv2.add(new_frame,mask)\n                printer_check = printer_check + 1\n                if (printer_check == 10 or first_time == 1):\n                    cv2_imshow(img)\n                    printer_check = 0\n                first_time = 0\n                k = cv2.waitKey(30) & 0xff\n                if k == 27:\n                    break\n\n            # Now update the previous frame and previous points\n            gray = frame_gray.copy()\n            #p0_test = good_new.reshape(-1,1,2)\n"

```

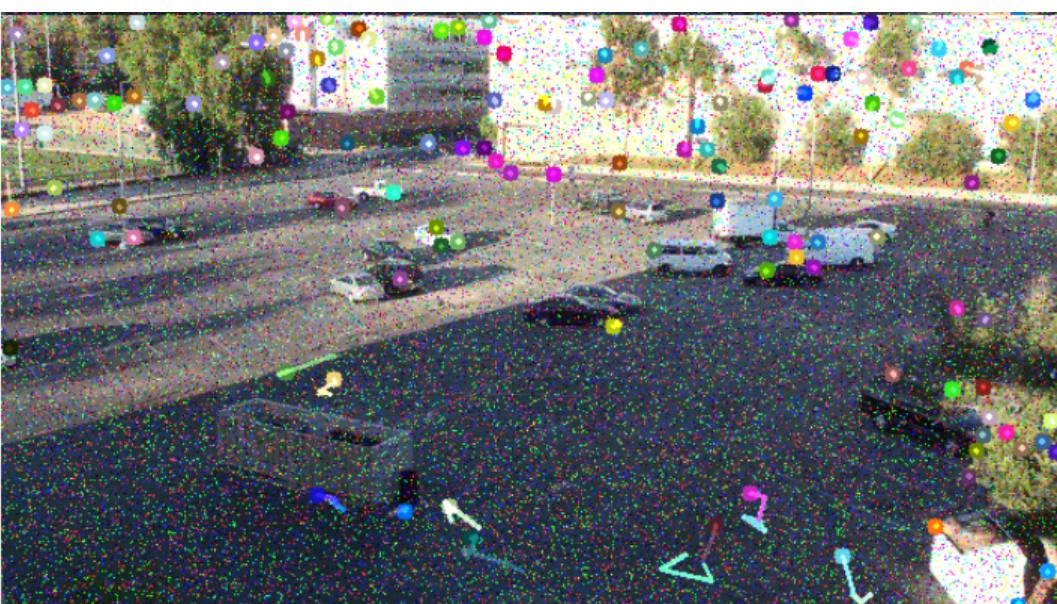
$\alpha = 0.01$



$\alpha = 0.05$



$\alpha = 0.1$



Harris

```
In [6]: ### Harris with noise
...
# params for Harris corner detection

feature_params = dict( maxCorners = 210,
                       qualityLevel = 0.03,
                       minDistance = 10,
                       blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize = (12,12),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))

# Create some random colors
color = np.random.randint(0,255,(190,3))
cap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)
ret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame
new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

## ADD NOISE
new_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)
new_frame = np.array(255 * new_frame, dtype = 'uint8')

# Take first frame and find corners in it
gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True

# Create a mask image for drawing purposes
mask = np.zeros_like(new_frame)

first_time = 1
printer_check = 0
count_frames = 0

while(1):
    if (count_frames == 60): # approx 0.5 second
        count_frames = 0
        p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True
        mask = np.zeros_like(new_frame)

    ret,frame = cap.read()
    count_frames = count_frames + 1

    if(ret == 0):
        break # reached the end of the video
    new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))

    ## ADD NOISE
    new_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)
    new_frame = np.array(255 * new_frame, dtype = 'uint8')

    frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)

    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
        ##### keep old points #####
        save_old = good_old.reshape(-1,1,2)
        p0 = good_new.reshape(-1,1,2)

    #### Keep only moving points ####

    del_list = []
    track = 0
    for i in p0:
        if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and
            (math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):
            del_list.append(track)
        track = track + 1

    del_list.reverse()
    for i in del_list:
```

```

p0 = np.delete(p0, i, 0)
save_old = np.delete(save_old, i, 0)

good_old = save_old.reshape(len(save_old), 2)
good_new = p0.reshape(len(p0), 2)

# draw the tracks
for i,(new,old) in enumerate(zip(good_new, good_old)):
    a,b = new.ravel()
    c,d = old.ravel()
    mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)
    new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)
img = cv2.add(new_frame,mask)
printer_check = printer_check + 1
if (printer_check == 10 or first_time == 1):
    cv2_imshow(img)
    printer_check = 0
first_time = 0

k = cv2.waitKey(30) & 0xff
if k == 27:
    break

# Now update the previous frame and previous points
gray = frame_gray.copy()
#p0_test = good_new.reshape(-1,1,2)
'''
```

Out[6]:

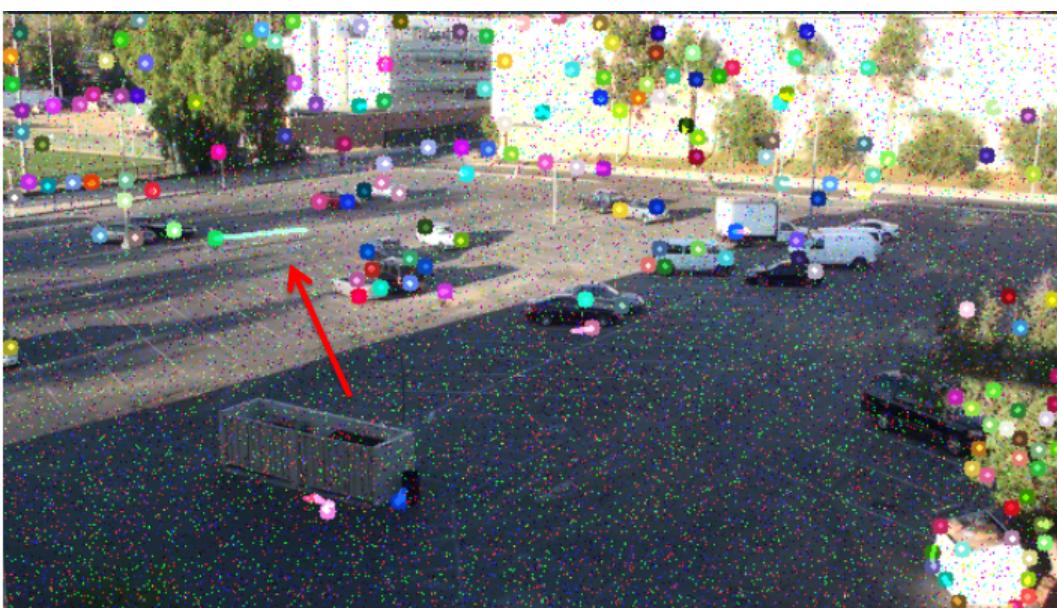
```

"\n# params for Harris corner detection\nfeature_params = dict( maxCorners = 210, \n    qualityLevel = 0.03,\n    minDistance = 10,\n    blockSize = 7 )\n\n# Parameters for lucas kana de optical flow\nlk_params = dict( winSize = (12,12), \n    maxLevel = 3,\n    criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 20, 0.1))\n\n\n# Create some random colors\nncolor = np.random.randint(0, 255, (190,3))\nncap.set(1, cap.get(cv2.CAP_PROP_FPS)*9)\nret,frame = cap.read() # get the next frame, ret is T/F depends on if we have a next frame\nnew_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n\n## ADD NOISE\nnew_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)\nnew_frame = np.array(255 * new_frame, dtype = 'uint8')\n\n# Take first frame and find corners in it\ngray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\np0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True\n\n# Create a mask image for drawing purposes\nmask = np.zeros_like(new_frame)\n\n\nfirst_time = 1\nprinter_check = 0\ncount_frames = 0\nwhile(1):\n    if (count_frames == 60): # approx 0.5 second\n        count_frames = 0\n        p0 = cv2.goodFeaturesToTrack(gray, mask = None, useHarrisDetector = True, **feature_params) # useHarrisDetector = True\n        mask = np.zeros_like(new_frame)\n\n        ret,frame = cap.read()\n        count_frames = count_frames + 1\n        if(ret == 0):\n            break # reached the end of the video\n        new_frame = cv2.resize(frame,(int(frame.shape[1]/2), int(frame.shape[0]/2)))\n\n        ## ADD NOISE\n        new_frame = random_noise(new_frame, mode = 's&p', amount = 0.01)\n        new_frame = np.array(255 * new_frame, dtype = 'uint8')\n\n        frame_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)\n        # calculate optical flow\n        p1, st, err = cv2.calcOpticalFlowPyrLK(gray, frame_gray, p0, None, **lk_params)\n\n        # Select good points\n        if p1 is not None:\n            good_new = p1[st==1]\n            good_old = p0[st==1]\n            ##### keep old points\n            save_old = good_old.reshape(-1,1,2)\n            p0 = good_new.reshape(-1,1,2)\n\n            ### Keep only moving points ####\n            del_list = []\n            track = 0\n            for i in p0:\n                if ((math.ceil(save_old[track][0][0]) == math.ceil(i[0][0])) and\n                    math.ceil((save_old[track][0][1]) == math.ceil(i[0][1]))):\n                    del_list.append(track)\n                track = track + 1\n            del_list.reverse()\n            for i in del_list:\n                p0 = np.delete(p0, i, 0)\n                save_old = np.delete(save_old, i, 0)\n            good_old = save_old.reshape(len(save_old), 2)\n            good_new = p0.reshape(len(p0), 2)\n\n            # draw the tracks\n            for i,(new,old) in enumerate(zip(good_new, good_old)):\n                a,b = new.ravel()\n                c,d = old.ravel()\n                mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].tolist(), 2)\n                new_frame = cv2.circle(new_frame,(int(a),int(b)),5,color[i].tolist(),-1)\n            img = cv2.add(new_frame,mask)\n            printer_check = printer_check + 1\n            if (printer_check == 10 or first_time == 1):\n                cv2_imshow(img)\n                printer_check = 0\n            first_time = 0\n            k = cv2.waitKey(30) & 0xff\n            if k == 27:\n                break\n\n        \n        # Now update the previous frame and previous points\n        gray = frame_gray.copy()\n        #p0_test = good_new.reshape(-1,1,2)\n    "
```

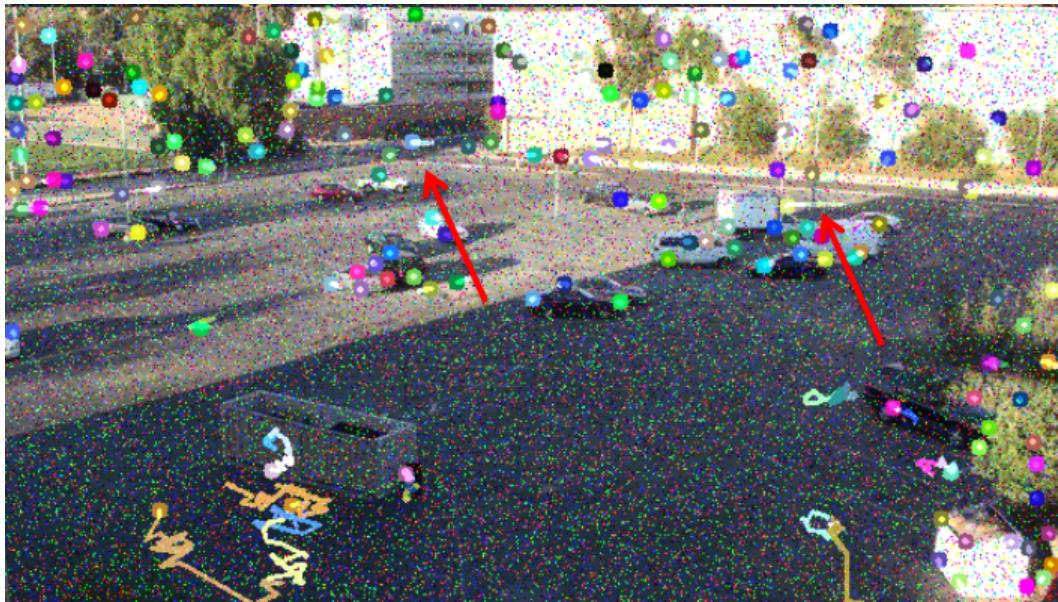
$\alpha = 0.01$



$\alpha = 0.05$



$$\alpha = 0.1$$



Ανεξαρτήτως μεθόδου εντοπισμού γωνιών, η εισαγωγή θορύβου επηρεάζει αισθητά την ικανότητα ανίχνευσης κίνησης. Παρατηρούμε μόνο αποσπάσματα των αναμενόμενων τροχιών. Με χαμηλότερο ποσοστό θορύβου, φαινομενικά ανιχνεύεται καλύτερα η κίνηση των γωνιών κατά Harris. Στο υψηλότερο ποσοστό θορύβου παρατηρείται έντονα το φαινόμενο της λανθασμένης ανίχνευσης κίνησης, σχεδιάζονται, δηλαδή, τροχιές για σημεία τα οποία δεν μετατοπίζονται.