Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (Национальный исследовательский университет)»   
(МГТУ им. Н.Э. Баумана)

Факультет Информатики и систем управления (ИУ)

Кафедра Информационные системы и телекоммуникации (ИУ-3)

**Отчёт**

**по домашнему заданию**

**«Разработка программы отслеживания объектов в видеопотоке на основе метода Лукаса-Канаде (ЛК) и создание оригинального алгоритма регенерации характерных признаков ЛК»**

**по курсу**

**«Цифровая обработка изображений»,**

**1 курс, 2-й семестр.**

Выполнил: студентка группы ИУ3-21М

Смирнова К.Д.,

Проверил: Алфимцев А.Н.

Москва, 2020

**Задание**

1. В среде Spyder (сборка Anaconda) на языке Python 3.х создать проект и подключить библиотеку opencv.
2. Из л.р. 1-3 взять модули загрузки цветного цифрового изображения и обработки пикселей.
3. Провести информационный поиск в Интернете и дать пошаговое описание работы метода Лукаса-Канаде.
4. Подготовить видеопоток с медленно движущимся объектом по горизонтали (в соответствии с вариантом внизу, найти в Интернете или сделать видео самостоятельно, не более 1 мин, камера не двигается).
5. Запрограммировать отслеживание объекта с использованием метода Лукаса-Канаде (для этого использовать функцию cv2.calcOpticalFlowPyrLK() из библиотеки opencv).
6. Сделать перевод разделов III и IV статьи Tracking of Moving Objects With Regeneration of Object Feature Points.
7. Подготовить второй видеопоток на основе п. 4 для создания условий потери характерных точек ЛК при движении объекта (искусственные вертикальные черные полосы по аналогии с рис.1-3 из статьи из п. 6).
8. Разработать и запрограммировать оригинальный алгоритм регенерации характерных признаков ЛК (по аналогии со статьей из п. 6).
9. Провести эксперимент по регенерации характерных признаков ЛК с визуализацией результатов. Построить график ошибок первого и второго рода по результатам регенерации.
10. Провести эксперимент по отслеживанию объектов с визуализацией результатов (в том числе визуализацией применения характерных признаков на изображении). Сравнить надежность отслеживания с регенерацией и без нее. Построить график ошибок первого и второго рода по результатам отслеживания объекта.

Вариант:

|  |  |
| --- | --- |
| **№** | **Движущийся объект** |
| 12 | Птица |

**Описание метода Лукаса-Канаде. Постановка задачи.**

Метод Лукаса-Канаде является алгоритмом вычисления оптического потока. Понятием оптического потока формализуется движение на видео объектов. Таким образом, нахождение оптического потока на массиве кадров представляет собой выявление смещения определённых точек на кадре относительно предыдущего, что позволяет, как минимум, решить задачу отслеживания объекта в видеопотоке. В основу вычисления закладывается предположение, что значения пикселей переходят из предыдущего кадра в следующий без изменений, то есть пиксели, относящиеся к одному и тому же объекту, могут сместиться в какую-либо сторону, но их значение останется неизменным.

Функция интенсивности пикселей от положения в кадре и времени записывается как:

,

где и – координаты пикселя;

и – смещение;

– номер кадра.

Применив аппроксимацию ряда Тейлора, получается:

В получившимся уравнении два неизвестных смещения и *.* Для их нахождения, основываясь на этом уравнении, используются различные методы, в том числе и метод Лукаса-Канаде.

В общем, алгоритм Лукаса-Канаде относится к тем алгоритмам, которые позволяют получить смещения для каждого отдельного пикселя, и применяются тогда, когда скорость не столь критична.

Работа метода существует с предположениями, что смещение точек на двух изображении незначительное и смещение точек в окрестности некоторой точки одинаково. Поэтому на первом этапе определяется окно 3x3 пикселя вокруг отслеживаемой точки, для которых смещение одинаково, и записывается для них система из девяти уравнений с двумя неизвестными:

где - частные производные изображения I по координатам x, y и времени t, вычисленные в точке p.

Системе соответствует матричное уравнение:

Далее полученная система решается методом наименьших квадратов:

Решение имеет вид:

В вычисление можно добавить весовые коэффициенты пикселей окна. Чем ближе пиксель к отслеживаемой точке, тем выше коэффициент. Тогда в матричном уравнении появится диагональная квадратная матрица весов. Решение преобразуется в:

Если не удается вычислить оптический поток за одну итерацию, то можно применять итеративный метод. Вычислив смещения на первой итерации, каждый пиксель кадра перемещается в противоположную сторону так, чтобы это смещение компенсировать. На следующей итерации вместо исходного кадра будет использоваться его искаженный вариант. И так далее, пока на очередной итерации все полученные смещения не окажутся меньше заданного порогового значения. Итоговое смещение для каждого конкретного пикселя получается как сумма его смещений на всех итерациях.

В иерархическом методе берётся два изображения — исходное и следующее и строится пирамида масштабов — гауссова пирамида. Рассматривается изображение с самым низким разрешением. Максимальное движение на самом низком разрешении происходит в пределах одного пикселя. Далее с помощью итеративного варианта метода Лукаса — Канаде вычисляется оптический поток для изображения с самым низким разрешением. Полученное значение используется в изображениях с более высоким разрешением и т. д.

**Перевод пунктов 3 и 4 статьи “Tracking of Moving Objects With Regeneration of Object Feature Points”**

ПОСТАНОВКА ЗАДАЧИ

Отслеживание разрозненного оптического потока [21-23] заключается в обнаружении основных моментов в видеокадре и последующего отслеживания обнаруженных точек функции в видеосъемке. В связи с несовершенством алгоритмов слежения, некоторые функциональные точки могут быть потеряны. Первоначально, несколько точек могут быть обнаружены в пределах контура движущегося объекта. Значительная часть фрагмента изображения вокруг такой «особенной» точки принадлежит фону. При отслеживании данная точка функции может покинуть область объекта и войти в фоновую область, что вносит неопределенность в отслеживание движущихся объектов. Кроме того, непрозрачные препятствия могут скрывать некоторые части движущегося объекта и затруднять отслеживание движущегося объекта. Для того, чтобы увеличить точность отслеживания, мы предложили метод автоматического восстановления утраченной функции точки с помощью обнаружения новых особенностей в пределах области объекта. Основная идея метода восстановления основана на эксплуатации принципов биологической регенерации.

ВОССТАНОВЛЕНИЕ ФУНКЦИОНАЛЬНЫХ ТОЧЕК НА ОСНОВЕ БИОЛОГИЧЕСКОЙ РЕГЕНЕРАЦИИ

Регенерация – это особая способность биологического организма к восстановлению некоторых потерянных частей его тела. Рассмотрим три известных принципа биологической регенерации [24]: эпиморфоз, морфаллаксис и эндоморфоз. Эпиморфоз состоит в уменьшении тканей тела, прилегающих к раненой поверхности, рост клеток и их последующая дифференцировка. Морфаллаксис в основном сходна с эпиморфозом, за исключением того, что остальные клетки организма должны быть перестроены. Эндоморфоз заключается в восстановлении поврежденной органной массы на основе деления и гипертрофия клеток. Вышеупомянутые принципы регенерации могут быть адаптированы к восстановлению утраченных функциональных точек в задаче отслеживания объектов. Эпиморфозы сохраняют оригинальную геометрическую форму параметров движущихся объектов. Основной критерий для присвоения точки функции движущемуся объекту - это расстояние от этой точки до ближайшего центра объекта. Главное представление о морфаллаксисе в основном сходно с эпиморфозом за исключением того, что дальнейшая оптимизация осуществляется для того, чтобы уменьшить количество плохо отслеживаемых объектов. При эндоморфозе учитывается количество точек объекта, которые в настоящее время отслеживаются в поле "Объекты".

В этой работе мы предлагаем использовать эпиморфозы в качестве основного принципа восстановления опознавательных точек. Пользователь указывает порог, при котором количество точек объекта будет отслеживаться как входной параметр метода. Особенные точки изображения отслеживаются в каждом видеокадре с помощью оптического потока Лукаса-Канаде [19] и границы объекта вычисляются с помощью метода слежения за движущимся объектом, представленном в следующем разделе. Когда количество особенных точек в пределах ограничивающего объекта уменьшается ниже заданного пользователем порога, точки новых функций будут обнаружены следующим образом. Сначала мы применяем детектор точек опознавания Ши-Томаси для обнаружения новых точек опознавания внутри окна, ограничивающего объект. Во-вторых, мы отбрасываем точки, расположенные на расстоянии менее 5 пикселей до исходных особенных точек объекта, которые отслеживаются в данный момент. Наконец, выбираем необходимое количество точек объекта, которые являются ближайшими к центру окна, ограничивающего объект в текущем видеокадре.

**Исследуемый видеопоток**

Задание данной работы – произвести отслеживание объекта (пешехода) на протяжении видео на основе вычисления оптического потока характерных точек, принадлежащих объекту.

В качестве исследуемого видеопотока был подготовлен видеофайл, на котором человек проходит из одной части кадра в другую. Камера при этом неподвижна. На рисунке 1 представлены фрагменты видео.

Видео и листинги доступны на <https://github.com/karinoizerr/Digital-image-processing/tree/master/DZ>



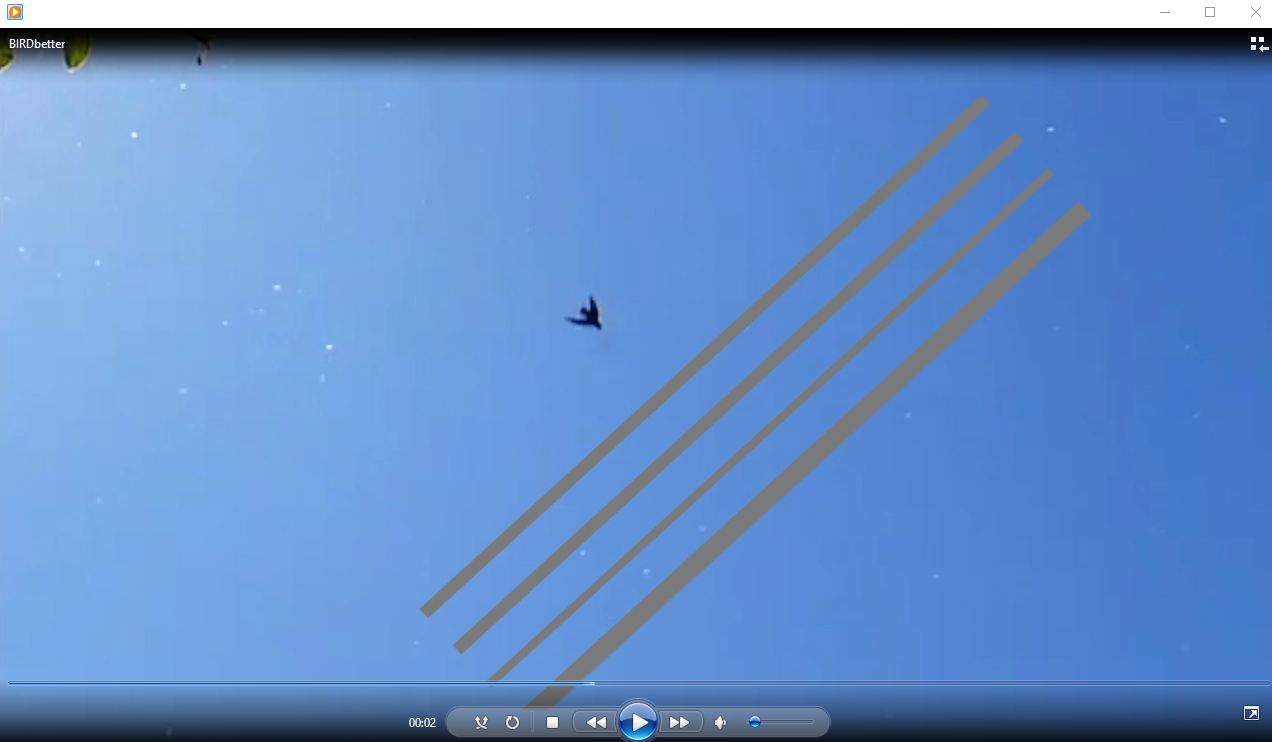




Рисунок 1 – Кадры видеофайла

**Создание характерных точек**

Так как пешеход в любом случае будет представлять угловатый объект с чётким контуром, для создания характерных точек применён алгоритм нахождения углов Ши-Томаси, который реализуется методом goodFeaturesToTrack из Python библиотеки opencv. Для оптимального результата можно варьировать максимальное количество точек, уровень качества, минимальное расстояние между точками. На рисунке 2 изображён фрагмент видео с нанесёнными найденными характерными точками.

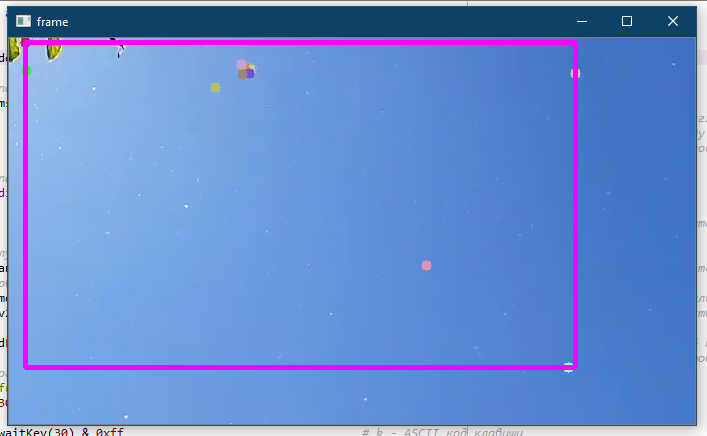


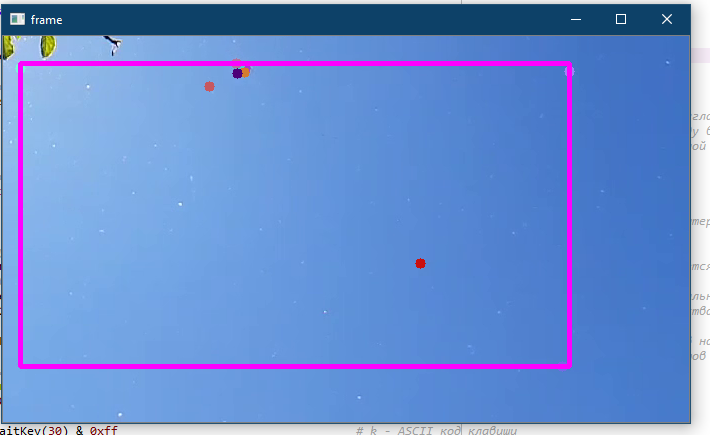
Рисунок 2 – Фрагмент видео с нанесёнными найденными характерными точками

**Вычисление оптического потока**

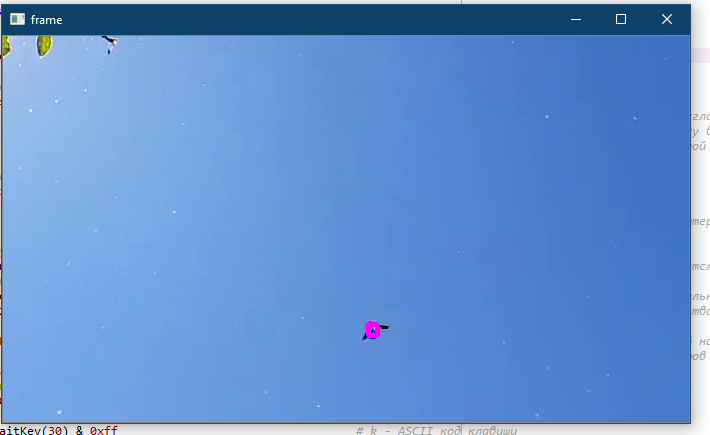
Естественно, углами на видео обладает не только объект, но и элементы фона. Поэтому, условимся, что движение в кадре осуществляет только отслеживаемый объект, а значит, далее находиться оптический поток будет только для точек, совершающих на первых кадрах смещение выше порогового значения. То есть, на некотором количестве первых кадров (около десяти) выбираются движущиеся точки, а статические отбрасываются, после чего на остальных кадрах эти точки являются закреплёнными за объектом и обрабатываются в полной мере.

Оптический поток вычисляется методом calcOpticalFlowPyrLK из Python библиотеки opencv.

На основе точек, которые мы определили за объектом и для которых действителен оптический поток, нарисуем прямоугольную область объекта. Результат показан на рисунке 3. На нём видно, как на первых кадрах прямоугольная область объекта доопределяется на основе только движущихся точек и трансформируется до определённой формы.



Начало движения птицы



Конец движения птицы

Рисунок 3 – Результат вычисления оптического потока

Программный код на языке Python, реализующий данное отслеживание приведён в листинге DZ.py.

**Создание условий потери характерных точек**

В качестве условий для потери характерных точек на видеофрагмент были нанесены серые вертикальные полосы. На рисунке 4 видно, что характерные точки действительно теряются у объекта.



Рисунок 4 – Потеря характерных точек

**Алгоритм регенерации характерных точек**

В процессе обработки не начальных кадров (в которых точки уже закреплены за объектом и определена прямоугольная область) добавлен контроль количества характерных точек внутри прямоугольной области объекта. В случае потери точек из области происходит регенерация характерных точек алгоритмом Ши-Томаси. Далее из всех точек на кадре выбираются ближайшие к центру прямоугольной области объекта в количестве изначально закреплённых за объектом точек. Работа алогритма продемонстрирована на рисунке 5.

Программный код на языке Python, реализующий данный алгоритм приведён в листинге DZbetter.py.

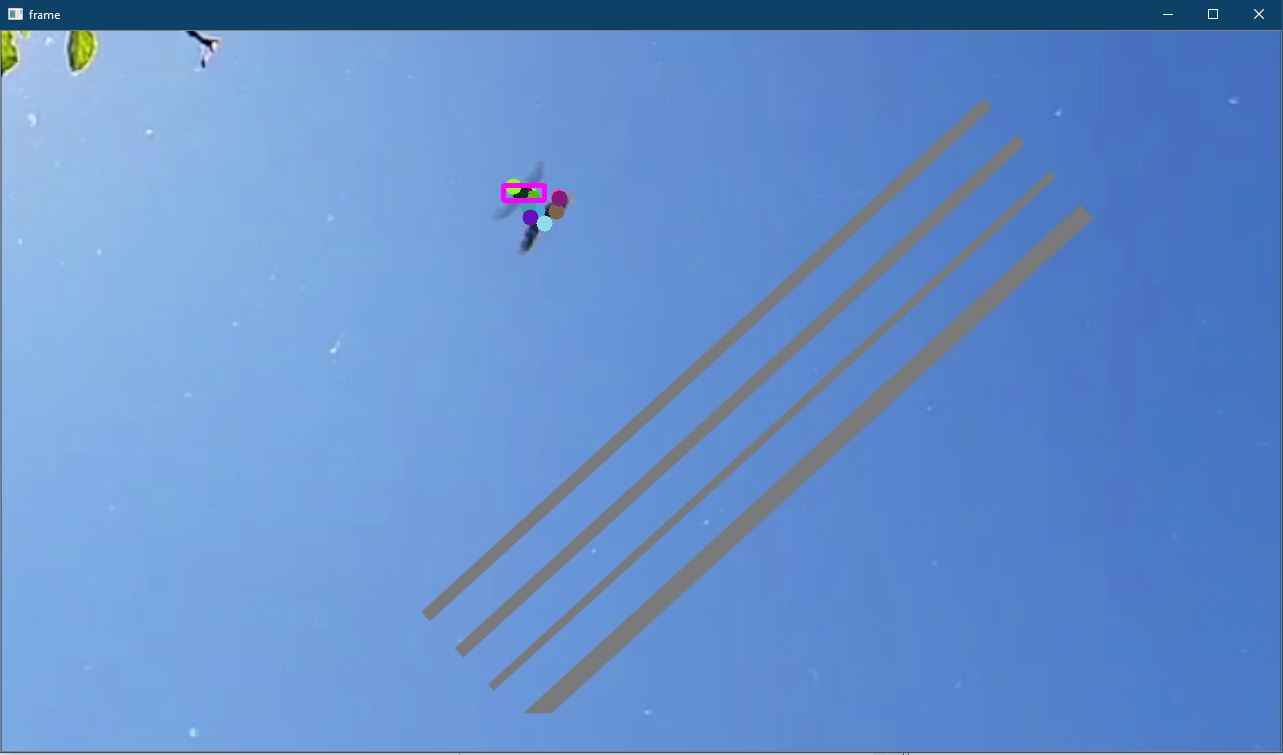












Рисунок 5 – Сохранение отслеживания объекта с условиями потери характерных точек с помощью алгоритма регенерации

У каждого кадра высчитывается количество точек, для которых вычислен оптический поток. Эта цифра соответствует количеству точек на объекте в предыдущем кадре. Далее фиксируется количество точек, оставшихся в области. Таким образом, разность этих двух величин покажет число потерянных или неверно отвергнутых точек, что будет соответствовать ошибке первого рода, график которой вынесен на рисунок 6. Всплески значений заметны на кадрах, в которых объект проходит через созданные серые полосы.

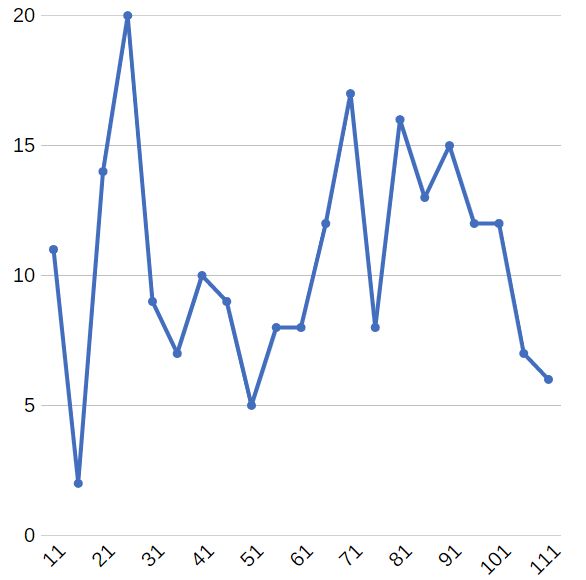


Рисунок 6 – График зависимости потерянных точек от кадра

**Вывод**

Результатом выполнения домашнего задания является разработанная программа обработки видеоматериала с целью установления перемещения заданного объекта методами обнаружения углов Ши-Томаси и вычисления оптического потока Лукаса-Канаде. Были достигнуты цели изучения работы вышеперечисленных методов.

На практике установлена необходимость алгоритма регенерации характерных особенностей для преодоления условий потери. Разработанный алгоритм регенерации срабатывает удовлетворительно в предложенных условиях.

**DZ.py**

import numpy as np

import cv2

cap = cv2.VideoCapture('BIRDbetter1.mp4') # Захват видеофайла

# Словарь - параметры для метода нахождения углов Ши-Томаси

feature\_params = dict( maxCorners = 100, # Максимальное количество найденных углов.

qualityLevel = 0.01, # Характеризует минимально допустимое качество угла. Умножается на показатель качества самого "лучшего" угла.

minDistance = 1, # Минимально возможное евклидово расстояние между возвращаемыми углами.

blockSize = 7 ) # Размер среднего блока для вычисления производной ковариационной матрицы по каждому соседству пикселей.

# Словарь - параметры для вычисления оптического потока методом Лукаса-Канаде.

lk\_params = dict( winSize = (15,15), # Размер окна поиска на каждом уровне

maxLevel = 2, # Номер максимального уровня пирамиды

criteria = (cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03)) # Критерий остановки итераций алгоритма. Остановить, если достигнута точность 0.03 или после 10 итераций.

# Создание случайного набора цветов

color = np.random.randint(0,255,(100,3)) # Диапазон случайной величины 0 - 255. Возвращается матрица цветов размерами 100x3.

# Берётся первый кадр и находятся углы

ret, old\_frame = cap.read() # Возвращет true в ret, если кадр читается правильно. old\_frame содержит текущий кадр

old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY) # Преобразует изображение из цветового пространства BGR в полутоновое.

p0 = cv2.goodFeaturesToTrack(old\_gray, mask = None, \*\*feature\_params) # Функция нахождения N самых сильных углов на изображении методом Ши-Томази. Передаётся кадр old\_gray, параметры из feature\_params.

print(p0) # Возвращает массив из пар-координатов углов (2D точек).

# Вывод первого кадра

cv2.imshow('frame', old\_frame) # Демоснтрация frame

cv2.waitKey(30) & 0xff

while True: # Цикл для запуска видео по нажатию клавиши

k = cv2.waitKey(30) & 0xff # k - ASCII код клавиши

if k == 27: # Запуск видео - нажатие ESC

break

count = 0 # Счётчик обработанных кадров

cadr = 50 # Количество начальных кадров, по которым определяется область объекта.

# Бесконечный цикл считывания кадров видео

while(1):

ret,frame = cap.read() # Считывание кадра. Возвращет true в ret, если кадр читается правильно. frame содержит текущий кадр

if not ret: # выход из цикла, если не удалось считать кадр

break

frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) # Преобразует кадр из цветового пространства BGR в полутоновое.

# Вычисление оптического потока

p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params) # Вычисляет оптический поток объектов p0 (входной вектор 2D точек) на основе первого кадра old\_gray и текущего кадра frame\_gray

print(p1) # с параметрами из lk\_params.

# p1 - выходной вектор 2D точек, содержащий рассчитанные новые позиции входных объектов в текущем кадре.

# st - вектор состояний для каждой 2D точки. 1 - если оптический поток найден для точки. 0 - если не найден.

if count <= cadr: # Обработка первых кадров

L = p1 - p0 # Расстояние, на которое перемещается точка с предыдущего кадра.

move = []

print('это эль', L)#ВЫВОЖУ ЭЛЬ # Создание пустого списка движущихся точек.

for i in range(len(L)): # Цикл для заполнения списка точками, переместившимися по сравнению с предыдущем кадром.

if (abs(L[i][0][0]) >= 0.01) or (abs(L[i][0][1] >= 0.01)): # Если точка переметсилась на пиксель по любому направлению.

move.append(p1[i][0]) # Присоединеие точки к списку.

move = np.array(move) # Создание массива из списка.

new\_center = np.median(move, axis=0) # Рассчёт нового центра прямоугольной области объекта, как медиана движущихся точек.

print(move)

max\_x = move[0][0] # Нахождение X-координаты самой отдалённой точки от (0;0)

for i in range(len(move)):

if move[i][0] > max\_x:

max\_x = move[i][0]

max\_y = move[0][1] # Нахождение Y-координаты самой отдалённой точки от (0;0)

for i in range(len(move)):

if move[i][1] > max\_y:

max\_y = move[i][1]

min\_x = move[0][0] # Нахождение X-координаты самой приближённой точки к (0;0)

for i in range(len(move)):

if move[i][0] < min\_x:

min\_x = move[i][0]

min\_y = move[0][1] # Нахождение Y-координаты самой приближённой точки к (0;0)

for i in range(len(move)):

if move[i][1] < min\_y:

min\_y = move[i][1]

# Расчёт координат нижнего и верхнего углов прямоугольной области с объектом

new\_up = (new\_center + (max\_x, max\_y) - new\_center).reshape(-1)

new\_down = (new\_center - new\_center + (min\_x, min\_y)).reshape(-1)

# Найденные отклоенния от центра

new\_x = max\_x - new\_center[0]

new\_y = max\_y - new\_center[1]

# Прорисовка движущихся точек на кадре.

for i,new in enumerate(move):

a,b = new.ravel()

frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1) # Рисование точки

if count > cadr: # Обработка последующих кадров

good\_new = p1[st==1] # Дальшнейшая обработка точек, для которых найден оптический поток.

new\_center = np.mean(good\_new, axis=0) # Рассчёт нового центра прямоугольной области объекта, как медиана точек.

# Расчёт координат нижнего и верхнего углов прямоугольной области с объектом

new\_up = (new\_center + (new\_x, new\_y)).reshape(-1)

new\_down = (new\_center - (new\_x, new\_y)).reshape(-1)

# Прорисовка точек на кадре.

for i,new in enumerate(good\_new):

a,b = new.ravel()

frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1) # Рисование точки

# Приведение размерности координат прямоугольной области с объектом для передачи их в функцию отрисовки прямоугольника

nd = new\_down.astype(np.integer).reshape(-1)

nu = new\_up.astype(np.integer).reshape(-1)

if frame is not None:

frame = cv2.rectangle(frame, tuple(nd), tuple(nu), (255, 0, 255), 4) # Прорисовка прямоугольной области объекта

cv2.imshow('frame',frame) # Демоснтрация кадра

k = cv2.waitKey(30) & 0xff # Задержка изорбражения 30 милисекунд. В k возвращается нажатая клавиша клавиша.

if k == 27: # Выход по нажатию ESC

break

# Старый кадр теперь ссылается на текущий кадр

old\_gray = frame\_gray.copy()

# Для первых кадров в цикле точки для последующего кадра берутся из move

if count <= cadr:

p0 = move.reshape(-1,1,2)

# Для последующих кадров в цикле точки для последующего кадра берутся из good\_new

if count > cadr:

p0 = good\_new.reshape(-1,1,2)

count += 1 # Инкрементация счётчика

# Особождение ресурсов

cv2.destroyAllWindows()

cap.release()

**DZbetter.py – Алгоритм регенерации**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Jun 13 18:59:11 2020

@author: karina

"""

import numpy as np

import cv2

cap = cv2.VideoCapture('BIRDbetter1.mp4') # Захват видеофайла

# Словарь - параметры для метода нахождения углов Ши-Томаси

feature\_params = dict( maxCorners = 100, # Максимальное количество найденных углов.

qualityLevel = 0.01, # Характеризует минимально допустимое качество угла. Умножается на показатель качества самого "лучшего" угла.

minDistance = 1, # Минимально возможное евклидово расстояние между возвращаемыми углами.

blockSize = 7 ) # Размер среднего блока для вычисления производной ковариационной матрицы по каждому соседству пикселей.

# Словарь - параметры для вычисления оптического потока методом Лукаса-Канаде.

lk\_params = dict( winSize = (15,15), # Размер окна поиска на каждом уровне

maxLevel = 2, # Номер максимального уровня пирамиды

criteria = (cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03)) # Критерий остановки итераций алгоритма. Остановить, если достигнута точность 0.03 или после 10 итераций.

# Создание случайного набора цветов

color = np.random.randint(0,255,(100,3)) # Диапазон случайной величины 0 - 255. Возвращается матрица цветов размерами 100x3.

# Берётся первый кадр и находятся углы

ret, old\_frame = cap.read() # Возвращет true в ret, если кадр читается правильно. old\_frame содержит текущий кадр

old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY) # Преобразует изображение из цветового пространства BGR в полутоновое.

p0 = cv2.goodFeaturesToTrack(old\_gray, mask = None, \*\*feature\_params) # Функция нахождения N самых сильных углов на изображении методом Ши-Томази. Передаётся кадр old\_gray, параметры из feature\_params.

# Возвращает массив из пар-координатов углов (2D точек).

# Вывод первого кадра

cv2.imshow('frame', old\_frame) # Демоснтрация frame

cv2.waitKey(30) & 0xff

while True: # Цикл для запуска видео по нажатию клавиши

k = cv2.waitKey(30) & 0xff # k - ASCII код клавиши

if k == 27: # Запуск видео - нажатие ESC

break

count = 0 # Счётчик обработанных кадров

count\_2 = 0 # Счётчик последующих кадров, в которых возможна регенерация точек.

cadr = 10 # Количество начальных кадров, по которым определяется область объекта.

# Бесконечный цикл считывания кадров видео

while(1):

# print("Кадр: ", count)

ret,frame = cap.read() # Считывание кадра. Возвращет true в ret, если кадр читается правильно. frame содержит текущий кадр

if not ret: # выход из цикла, если не удалось считать кадр

break

frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) # Преобразует кадр из цветового пространства BGR в полутоновое.

# Вычисление оптического потока

p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params) # Вычисляет оптический поток объектов p0 (входной вектор 2D точек) на основе первого кадра old\_gray и текущего кадра frame\_gray

# с параметрами из lk\_params.

# p1 - выходной вектор 2D точек, содержащий рассчитанные новые позиции входных объектов в текущем кадре.

# st - вектор состояний для каждой 2D точки. 1 - если оптический поток найден для точки. 0 - если не найден.

# print("Оптический поток изначально вычилсен для: ", len(p1)," точек")

if count <= cadr: # Обработка первых кадров

L = p1 - p0 # Расстояние, на которое перемещается точка с предыдущего кадра.

move = [] # Создание пустого списка движущихся точек.

for i in range(len(L)): # Цикл для заполнения списка точками, переместившимися по сравнению с предыдущем кадром.

if (abs(L[i][0][0]) >= 1) or (abs(L[i][0][1] >= 1)): # Если точка переметсилась на пиксель по любому направлению.

move.append(p1[i][0]) # Присоединеие точки к списку.

move = np.array(move) # Создание массива из списка.

number = move.shape[0] # Количество точек на объекте

new\_center = np.median(move, axis=0) # Рассчёт нового центра прямоугольной области объекта, как медиана движущихся точек.

#print("На объекте: ", number," точек")

max\_x = move[0][0] # Нахождение X-координаты самой отдалённой точки от (0;0)

for i in range(len(move)):

if move[i][0] > max\_x:

max\_x = move[i][0]

max\_y = move[0][1] # Нахождение Y-координаты самой отдалённой точки от (0;0)

for i in range(len(move)):

if move[i][1] > max\_y:

max\_y = move[i][1]

min\_x = move[0][0] # Нахождение X-координаты самой приближённой точки к (0;0)

for i in range(len(move)):

if move[i][0] < min\_x:

min\_x = move[i][0]

min\_y = move[0][1] # Нахождение Y-координаты самой приближённой точки к (0;0)

for i in range(len(move)):

if move[i][1] < min\_y:

min\_y = move[i][1]

# Расчёт координат нижнего и верхнего углов прямоугольной области с объектом

new\_up = (new\_center + (max\_x, max\_y) - new\_center).reshape(-1)

new\_down = (new\_center - new\_center + (min\_x, min\_y)).reshape(-1)

# Найденные отклоенния от центра

new\_x = max\_x - new\_center[0]

new\_y = max\_y - new\_center[1]

# Прорисовка движущихся точек на кадре.

for i,new in enumerate(move):

a,b = new.ravel()

frame = cv2.circle(frame,(a,b),8,color[i].tolist(),-1) # Рисование точки

if count > cadr: # Обработка последующих кадров

count\_2 += 1 # Счёт кадра с возможной регенрацией

good\_new = p1[st==1] # Дальшнейшая обработка точек, для которых найден оптический поток

new\_center = np.median(good\_new, axis=0) # Рассчёт нового центра прямоугольной области объекта, как медиана точек

# Расчёт координат нижнего и верхнего углов прямоугольной области с объектом

new\_up = (new\_center + (new\_x, new\_y)).reshape(-1)

new\_down = (new\_center - (new\_x, new\_y)).reshape(-1)

# Расчёт числа характерных точек в прямоугольной области с объектом

in\_box = 0

for (x, y) in good\_new:

xy = np.array((x, y))

if (xy >= new\_down).all() and (xy <= new\_up).all():

in\_box += 1

#print("В области ", in\_box," точек")

# Регенерация новых характерные точек, если количество имеющихся в области объекта уменьшилось

if in\_box < number:

# Изменение размерности массива для упрощения вычислений

old\_p0 = p0.reshape(p0.shape[0], p0.shape[2])

# Регенерация новых характерных точек

p0 = cv2.goodFeaturesToTrack(frame\_gray, mask = None, \*\*feature\_params)

#print("Регенерировано до ", p0.shape[0], " точек")

if p0.shape[0] > number:

# Формирование массива расстояний до центра прямоугольной области с объектом

distances = []

for xy in p0:

distances.append(np.linalg.norm(xy - new\_center))

# Выбор number точек, ближайших к центру прямоугольной области

p0 = p0[np.argpartition(distances, number)[:number]]

else:

p0 = p0

# Расчёт оптического потока с новыми характерными точками

p1, st, err = cv2.calcOpticalFlowPyrLK(old\_gray, frame\_gray, p0, None, \*\*lk\_params)

good\_new = p1[st==1]

if count\_2 > 1:

L = p1 - p0 # Расстояние, на которое перемещается точка с предыдущего кадра.

good\_new = [] # Создание пустого списка движущихся точек.

for i in range(len(L)): # Цикл для заполнения списка точками, переместившимися по сравнению с предыдущем кадром.

if (abs(L[i][0][0]) >= 0.1) or (abs(L[i][0][1] >= 0.1)): # Если точка переметсилась на пиксель по любому направлению.

good\_new.append(p1[i][0]) # Присоединеие точки к списку.

good\_new = np.array(good\_new)

#print("В итоге на объекте ", good\_new.shape[0], " точек")

# Прорисовка точек на кадре.

for i,new in enumerate(good\_new):

a,b = new.ravel()

frame = cv2.circle(frame,(a,b),8,color[i].tolist(),-1) # Рисование точки

# Приведение размерности координат прямоугольной области с объектом для передачи их в функцию отрисовки прямоугольника

nd = new\_down.astype(np.integer).reshape(-1)

nu = new\_up.astype(np.integer).reshape(-1)

if frame is not None:

frame = cv2.rectangle(frame, tuple(nd), tuple(nu), (255, 0, 255), 4) # Прорисовка прямоугольной области объекта

cv2.imshow('frame',frame) # Демоснтрация кадра

k = cv2.waitKey(30) & 0xff # Задержка изорбражения 30 милисекунд. В k возвращается нажатая клавиша клавиша.

if k == 27: # Выход по нажатию ESC

break

# Старый кадр теперь ссылается на текущий кадр

old\_gray = frame\_gray.copy()

# Для первых кадров в цикле точки для последующего кадра берутся из move

if count <= cadr:

p0 = move.reshape(-1,1,2)

# Для последующих кадров в цикле точки для последующего кадра берутся из good\_new

if count > cadr:

p0 = good\_new.reshape(-1,1,2)

count += 1 # Инкрементация счётчика

#print(" ")

# Особождение ресурсов

cv2.destroyAllWindows()

cap.release()