# Проект «создание трекера для множества малых неразличимых объектов»

Горбачев Николай, 772е

2 июня 2021 г.

## Введение

В области отслеживание траекторий (далее: треккинга) объектов с видеопотока активно разрабатываются технологии для решения целого спектра задач. Однако такой случай распознавания объектов с видеопотока, в котором объектов большое количество (более десяти), сами объекты неразличимы и малы в масштабе полного изображения, а их движение хаотичное и быстрое (смещение объекта за кадр порядка величины объекта) — задача треккинга при подобных условиях изучена не так глубоко. В данной работе предлагается решить задачу такого рода на примере отслеживания движения когорты дафний из видеопотока.

## Задача

Дан видеоролик на *N* кадров, демонстрирующий движение когорты из M дафний во времени. Необходимо восстановить траектории движения дафний во времени.

## Формализация задачи

На вход треккера поступает массив координат rawCoords, полученный в результате распознавания объектов некоторым алгоритмом детекции D на некотором известном видеоряде V.

Массив rawCoords содержит N строк, где N количество кадров видеоряда V. Размер каждого i-го элемента массива соответствует количеству задетектированных объектов на i-м кадре:

$\forall frame \in rawCoords \Rightarrow frame.size = m_i$

Необходимо получить множество $X = \{x_1 \ldots x_M\}$ последовательностей координат, где $x_j \in X$ – последовательность координат j-го объекта, причем $x_{j_i} \in x_j$ соответствует координатам j-го объекта на i-м кадре. Параметр $M$ в данной задаче заведомо неизвестен в общем случае, его поиск описывается в разделе «»

## Особенности задачи

Данная задача имеет ряд отличительных особенностей:

1. Движение объектов хаотично и негладко из-за чего невозможно интерполировать и экстраполировать траектории объектов в моментах отсутствия выходных данных детектора
2. Объекты малы относительно величины кадра (характерный размер несколько пикселей) в силу чего большинство известных алгоритмов основанных на гипотетической крупности объектов — нерелевантно.
3. Объекты двигаются быстро в сравнении с собственным характерным размером: многие современные алгоритмы детекции учитывают в том числе небольшую скорость изменения положения объект за кадр что также неактуально в данных условиях (сдвиг дафнии за кадр может быть порядке величины дафнии)
4. Объекты неразличимы — то есть при свободном движении дафнии треккер должен однозначно присваивать класс каждой засеченной дафнии на каждом кадре, в то время как после коллизии объектов треккер за неимением точности должен произвольно назначать дафниям классы

## Способ решения

Основной идеей алгоритма детекции является поиск массива ближайших соседей $Y_i$ для всех объектов $Y_{i-1}$ с предыдущего кадра.

Таким образом, будем сопоставлять объект $Y_{i_j}$ с объектом на предыдущем кадре $Y_{i-1_j}$ для каждого кадра i, из чего предлагается восстановить последовательности $X_j = \{Y_{i_j}\}$ координат каждого объекта.

Разберем алгоритм более детально в следующей секции.

## Алгоритм

В данной работе предлагается формировать сперва множество $Y = X^T$, то есть получать временную последовательность из координат $y_i = \{x_{i_j}\}$ для каждого кадра i; затем же предлагается из множества Y получать непосредственно $X = Y^T$

Таким образом, основная проблема при построении алгоритма – связать для i и i-1, $i \in 1 \ldots N$ кадров последовательности координат каждого j-го объекта, $j \in M$.

Предлагаемый алгоритм можно разделить на шесть основных шагов:

1. Копирование данных rawCoords в Y для 1-го кадра, опциональное дозаполнение $Y_1$ до M объектов, отмеченных как «необнаруженный детектором»
2. Для каждого j объекта с i-1 кадра – поиск ближайшего соседа на i-м кадре. Добавление в $Y_{i_j}$ ближайшего соседа, если евклидово расстояние до него меньше параметра $\rho_1$. Иначе – при слишком большом скачке, которое расценивается как смена объекта с j на $j' \neq j$ – отметить $Y_{i_j}$ флагом «необнаруженное» и запомнить координаты $Y_{i-1_j}$ в «память» – присвоить $x(Y_{i_j}) = -x(Y_{i-1_j}); y(Y_{i_j}) = -y(Y_{i-1_j});$ .
3. Если на i-м шаге в массиве задетектированных объектов $rawCoords_i$ менее M объектов, то для каждого объекта из $rawCoords_i$, которые не прошли фильтр по шагу (2.) и не записаны в массив $Y_i$ – произвести попытку найти соответствующий объект в «памяти», т.е. среди объектов $Y_{i_j}$ с отрицательными координатами и записать на их месте соответствующий объект из $rawCoords_i$. Под соответствием между объектом из $rawCoords_i$ и $Y_i$ здесь подразумевается расстояние меньшее чем параметр $\rho_2$.
4. Аналогично пункту (3.): если на i-м шаге в массиве задетектированных объектов $rawCoords_i$ менее M объектов, то для тех объектов из $Y_{i_j}$, для которых не нашлось близкого соседа из $rawCoords_i$ – присвоить ближайшего из возможных. Этот шаг в отличие от 3 подразумевает разрыв связей (прыжок треккера с j на $j' \neq j$ объект на i-м кадре).
5. Если на i-м кадре обнаружено больше объектов чем на i-1, то для всех объектов из rawCoords которые еще не записаны в $Y_{i_j}$, записать дополнительные координаты в $Y_{i_j}$ произвольно (этот шаг также предполагает нарушение непрерывности j траектории)
6. Полученный массив $Y_{i_j}$ транспонировать, либо покадрово вывести на экран.

## Проведение экспериментов

Реализация алгоритмов приведена в файле main.cpp, а также блокноте DafniaTracking_checkpoint-2.ipynb. Использованные при проведении экспериментов входные данные являются результатами детекции дафний посредством MOG2 и MGM. Для запуска main.cpp и необходимо иметь файлы с входными данными coords.txt и coords2.txt соответственно.

## Метод оценки качества

Для сравнения качества трекинга для приведенных детекторов (MOG2 и MGM-детекция) введем несколько величин:

1. $d = \Sigma d_i$ – суммарное количество раз, когда на i-м кадре для j-го объекта нарушается непрерывность траектории, т.е. при потере детекции j-го объекта до i-го кадра и сохранения его координат в памяти, и при нахождении на i-м кадре объекта $obj$, для которого не найдено близких координат ни среди $Y_{i-1_j}$ ни в памяти – алгоритм присваивает $Y_{i_j} = obj$. Эта процедура соответствует (4) шагу алгоритма
2. $c = \Sigma c_i$ – количество раз, когда детектированных координат меньше $M$ (в нашей задаче число искомых объектов $M = 40$), и такие пробелы заполняются заведомо ложными координатами, в случае чего также нарушается непрерывность траекторий – соответствует (5) шагу алгоритма.

## Подбор параметров

Приведенный алгоритм содержит ряд параметров:

1. Пороговое расстояние $\rho_1$ между распознанным объектом $Y_{i-1_j}$ и объектом-кандидатом на $Y_{i_j}$.
2. Расстояние $\rho_2$: для объекта $Y_{i-1_j}$, чья детекция прервана на i-м кадре (при необнаружении кандидатов на расстоянии менее чем $\rho_1$) и чьи координаты сохранены в памяти, назначить в качестве $Y_{i+1_j}$ ближайшего для $Y_{i-1_j}$ соседа с i+1 кадра, если $||Y_{i-1_j} - Y_{i+1_j}|| < \rho_2$

3. Пороговый радиус $\rho_0$, такой, что при разнице центров контуров более чем $\rho_0$, объекты считаются разными.

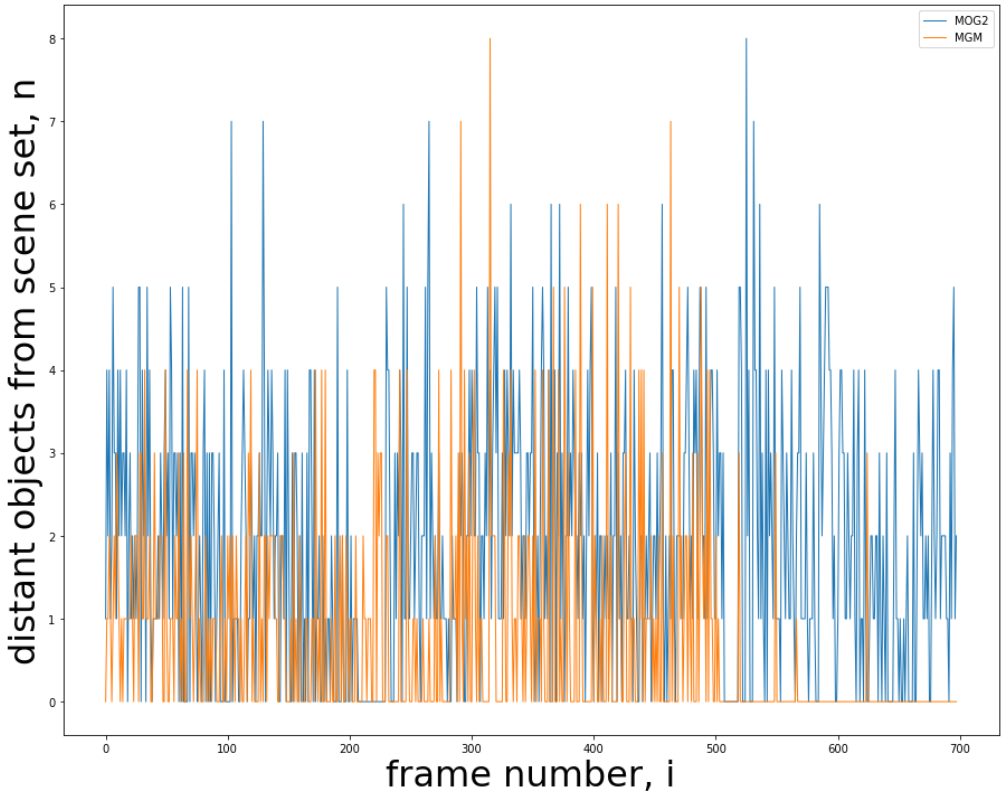
В данной реализации трекера приведенные выше параметры подобраны вручную на основе предположении о движении объектов в задаче: в нашей гипотетической модели будем считать, что за один кадр дафния перемещается на расстояние не более чем на порядок превышающее собственные характерные размеры: $\rho_1 < 50$.
Считаем, что $\rho_2 = 10$: так из памяти переотслеживаются дафнии смещенные не более чем на порядок своей величины. Наконец, примем $\rho_0 = 5$ тк считаем «диаметр» дафнии $\approx 10$.

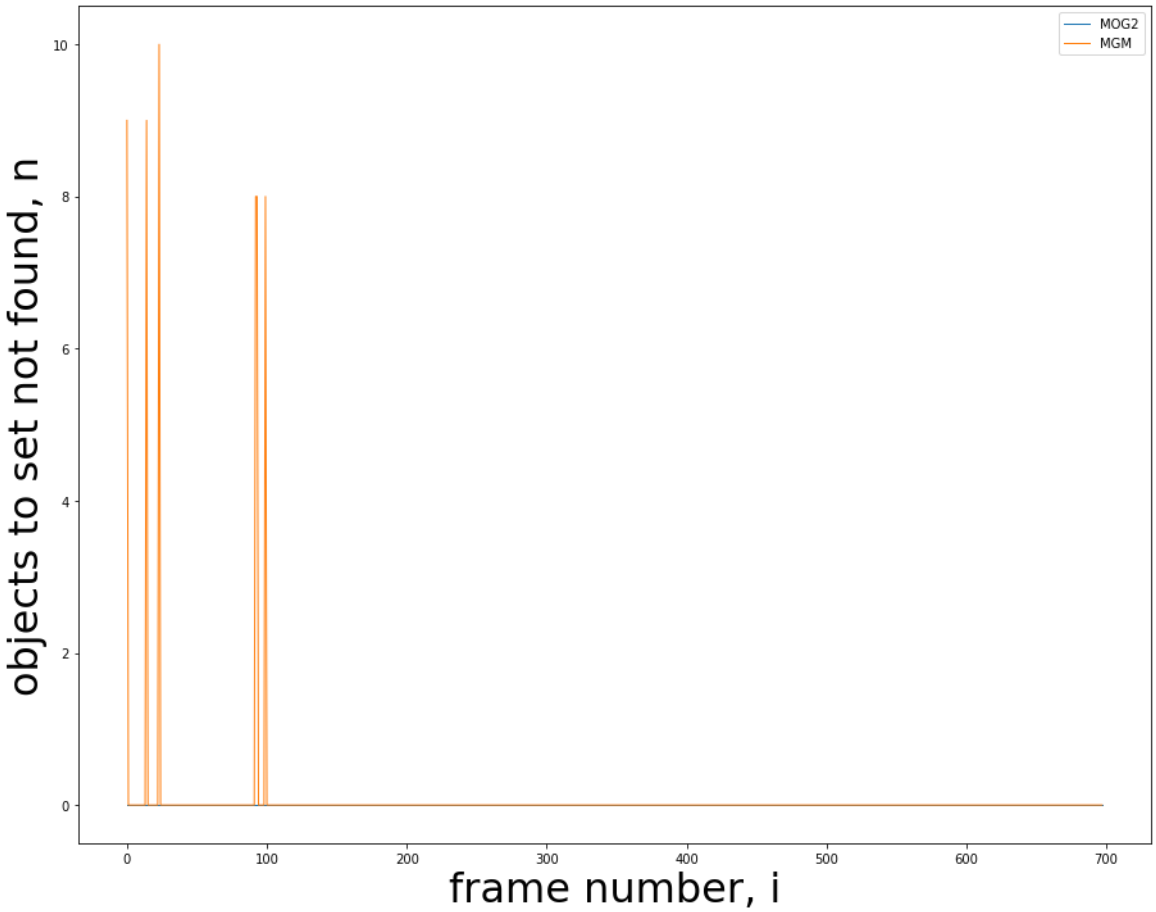
4. Параметр M – количество объектов трекинга – оценивается как среднее по количеству координат распознанных детектором за один кадр.

## Результаты

Для полученых данных детекции M(MOG2) = 40, M(MGM) = 20.

Результаты работы детекции наблюдаются в OpenCV-видеопотоке при запуске main.cpp. Приведем графики величин $d = \Sigma d_i$ и $c = \Sigma c_i$ для MOG2 и MGM -детекции.






Из графиков видно, что переназначение элементов с нарушением непрерывности траектории случается в среднем чаще для алгоритма MOG2-детекции как в абсолютной величине так и относительно среднего количества M детекции на кадре. Алгоритм MGM-детекции также влечет менее качественный трекинг с точки зрения определения числа дафний M, более качественный трекинг в терминах переназначения элементов:

$d(MGM) = 530, d(MOG2) = 1303$

$d(MGM)/M \approx 26.5, d(MOG2)/M \approx 34.03$

То есть, в среднем за кадр переназначение индексов j объектов происходит примерно в два раза чаще для трекинга по MOG2-детекции.

С другой стороны, для нескольких кадров MGM-детекции происходит до-назначение элементов из несуществующих значений (значений ни из памяти ни с данных детекции) – всего таких $c(MGM) = 52$ ошибочных переназначений, что вносит, хотя не такой существенный, вклад.

$c(MOG2) = 0$

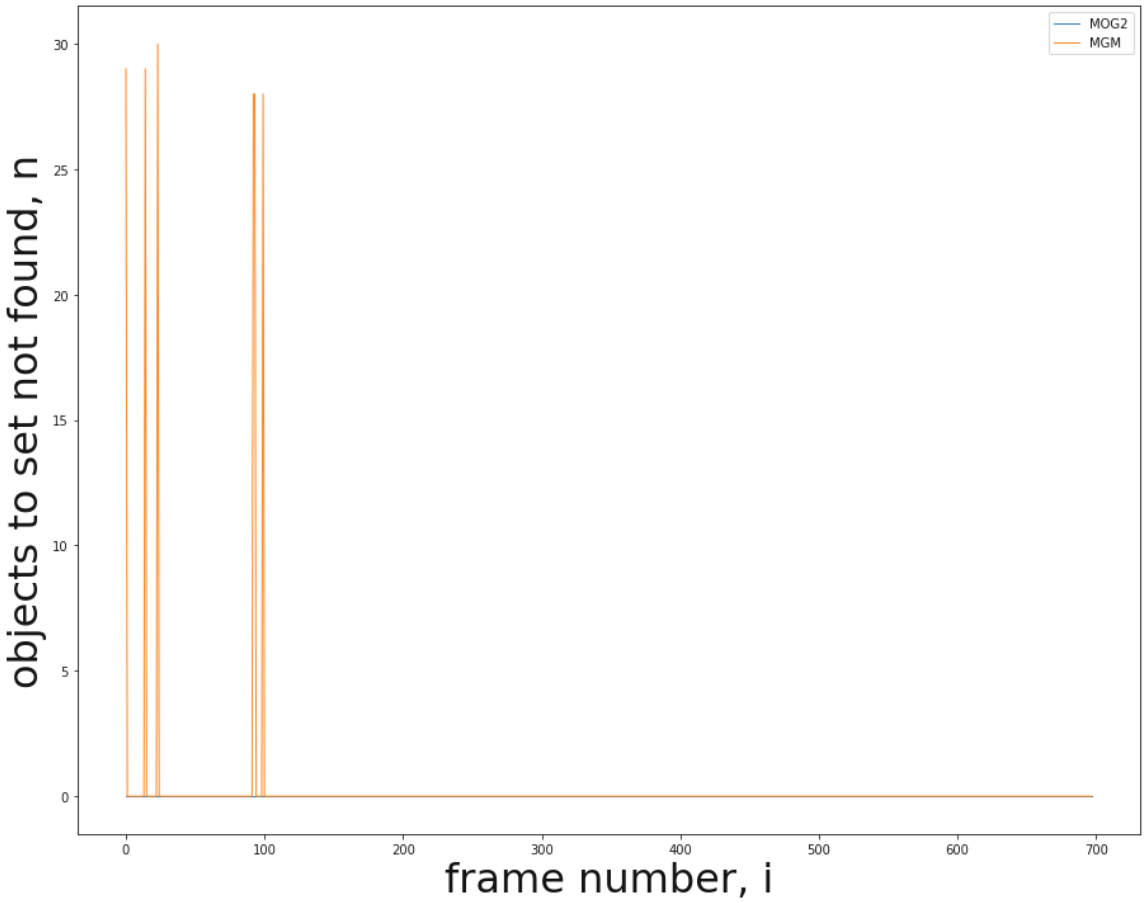



График зависимости c(i)

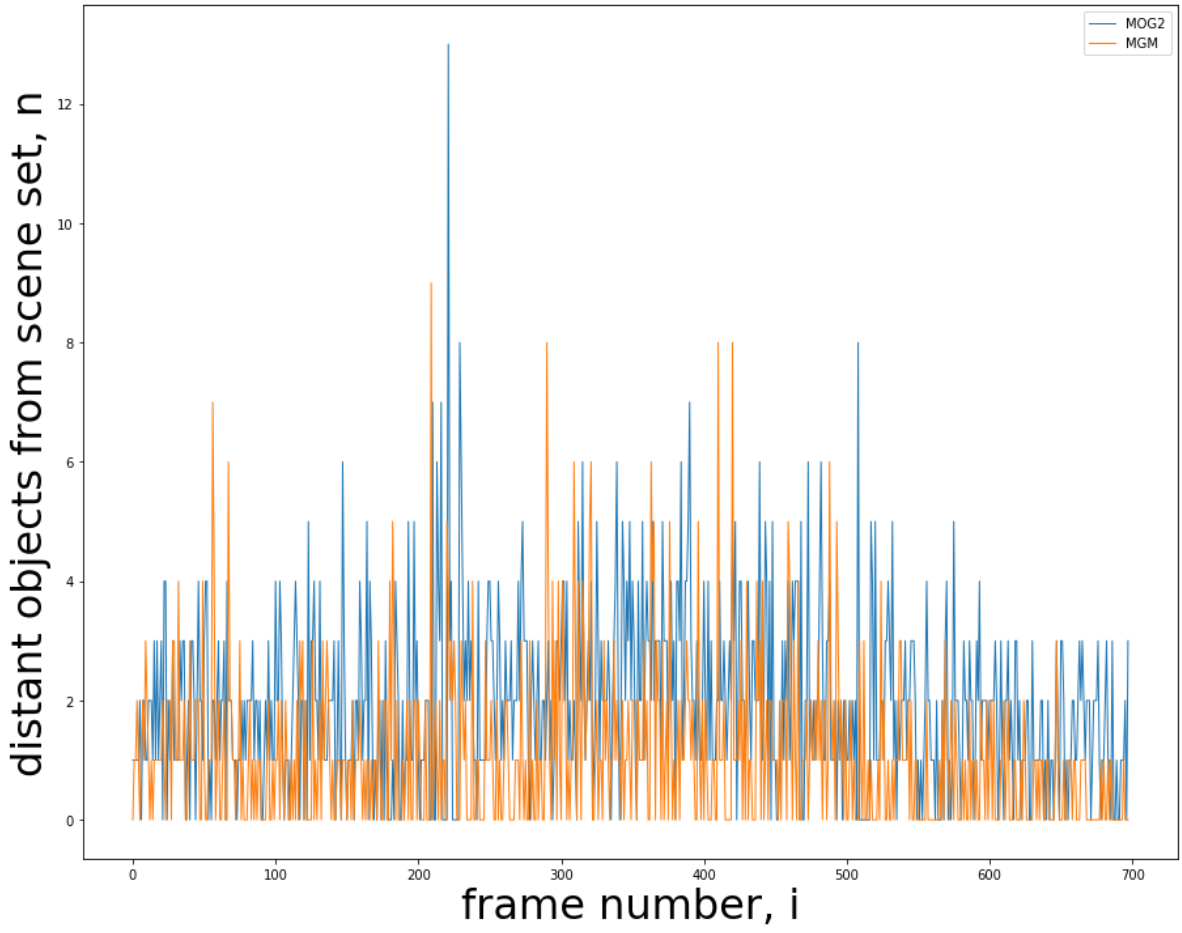



График зависимости d(i)

## Визуализация результатов

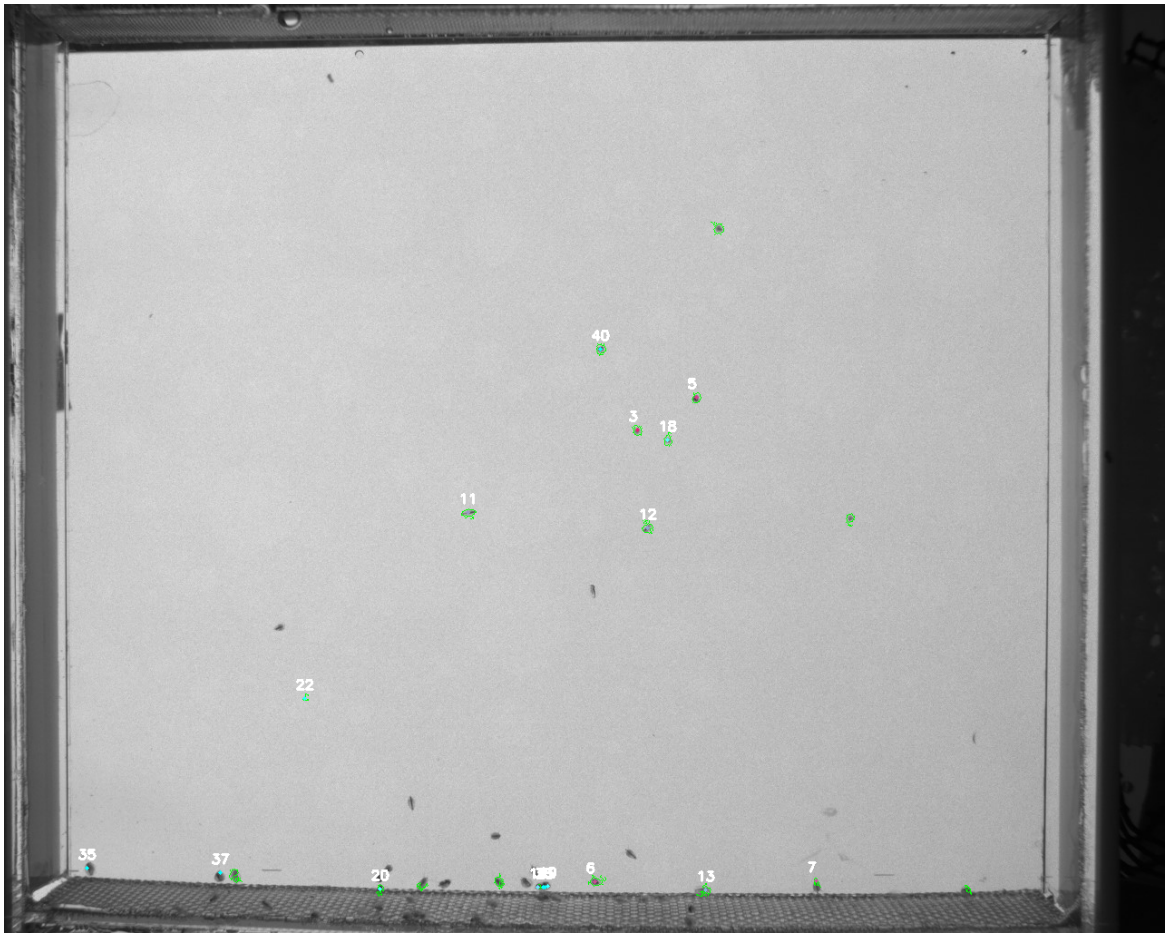
К отчету прилагаются три видеофрагмента:

1. Результат трекинга по более совершенной MGM-детекции
2. Результат трекинга по менее совершенной MOG2-детекции: в этом лучше наглядно демонстрируется работа трекера в тех случаях, когда детектор дает сбой (потеря детекции, детекция чрезмерного числа объектов итд)
3. Результат трекинга с наложенными результатами исходной MOG2-детекции.

Для видео 1. и 2. обозначим данные детектора синими кругами с центром в точке детекции. Результаты треккинга обозначим числами и цветными кругами меньшего радиуса. Для видео 3. Будем обозначать результаты детекции распознанными контурами.

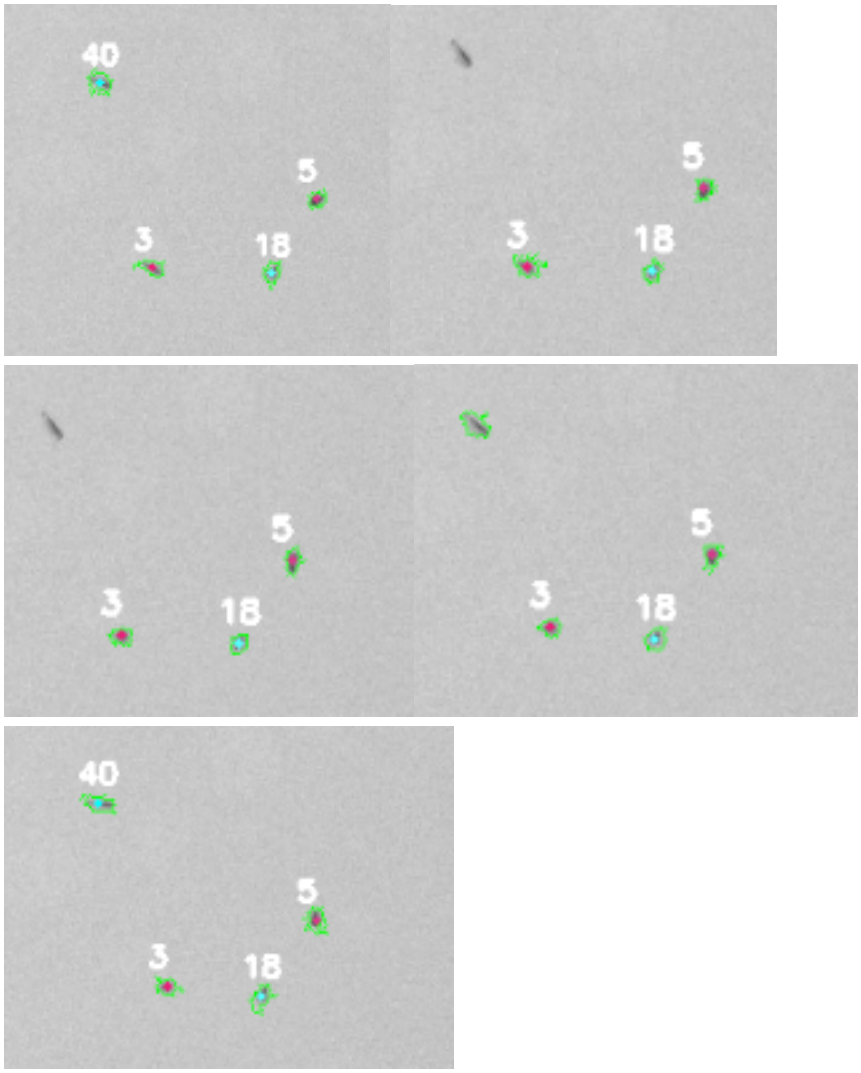
Приведем ниже несколько характерных примеров работы треккера (используя кадры с видео 3.):

1. Работа трекера в общем случае



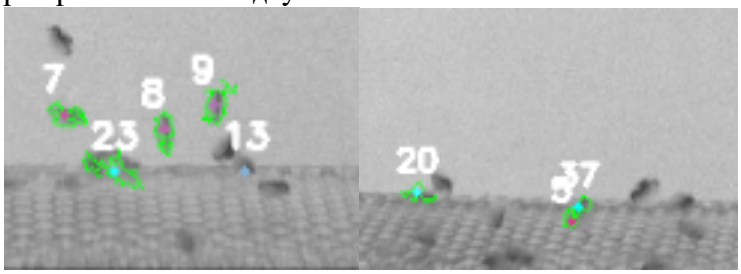
На иллюстрации видно несколько контуров не имеющих подписи. Это может обуславливаться либо непреодолением порога детекции (контур меньшей площади чем считающийся распознанным), либо тем что для соответствующего номера трекер держит в памяти удаленные координаты и не назначает новый контур для сохранения условия непрерывности траектории.

2. Работа «памяти» трекера при потере детекции:




На иллюстрации приведены последовательные кадры при работе трекера. Видно, как детекция 40-го объекта нарушается, но ее координаты хранятся в памяти и при возобновлении детекции новому объекту назначен правильный номер

3. Порог различимости двух объектов




На рисунке проиллюстрированы различные случаи срабатывания порога $\rho_0$: на некоторых кадрах видно, что одному объекту могут соответствовать несколько разных распознанных контуров (например, два контура у дафнии 8 на левой иллюстрации). Чтобы учитывать этот случай вводится пороговый радиус $\rho_0$, такой, что при разнице центров контуров более чем $\rho_0$, объекты считаются разными (напр., объекты 5 и 37 на иллюстрации справа). Для данных экспериментов принят $\rho_0 = 5$.