Задача сопровождения объектов на видео

Леонид Бейненсон

План:

- Что такое задача сопровождения объекта
  Варианты постановки задачи
- Общая схема алгоритма
- Детали реализации:
  - Вычисление коэффициентов схожести между траекториями и пешеходами
  - Определение соответствий между траекториями и пешеходами

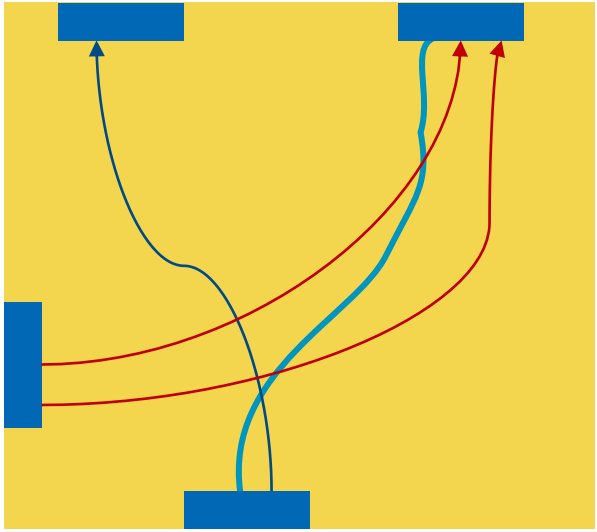
Что такое задача сопровождения объекта

Трекер -- алгоритм сопровождения объекта:

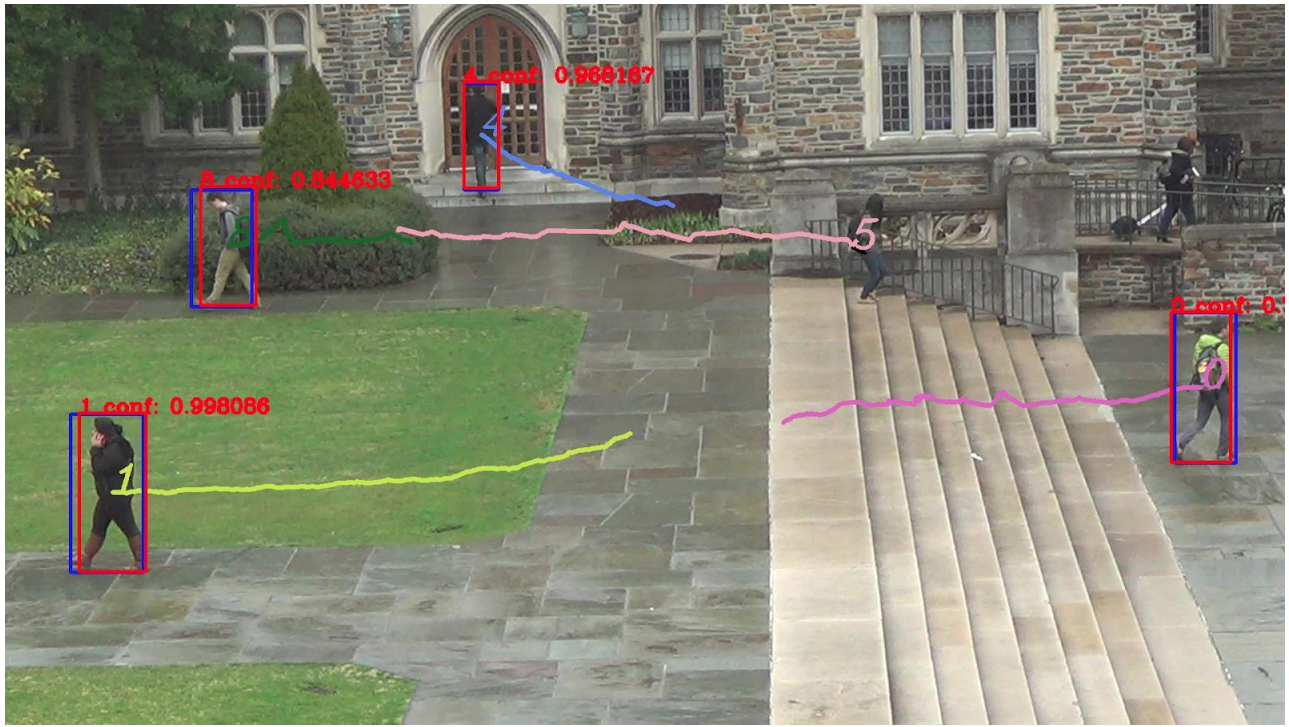
- Трекеру передается фрейм из видеопоследовательности (камера/видео)
- “Некто” (детектор или человек) выделяет объект(ы) на фрейме и передает его трекеру
- Трекеру подаются следующие фреймы из видеопоследовательности
- Трекер должен сказать где объект(ы) на этих фреймах

Пример постановки задачи

- Есть некоторое помещение. Несколько входов, несколько выходов.
- По данным с камеры видеонаблюдения нужно определить сколько людей за день проходит из входа i в выход j
- Решение: строить траектории пешеходов



Пример постановки задачи



Возможные сценарии использования

- Видеонаблюдение (камеры безопасности) на улицах или в помещениях
- Система помощи водителю
  (ADAS = Advanced driver-assistance systems)
  – отслеживание траекторий других машин по одной/нескольким камерам на машине
- «Живой фокус» в камере смартфона

Варианты постановки задачи

- Онлайн или оффлайн трекер
  - Онлайн – обработка «на лету»,
    - получает фрейм, обрабатывает его, и забывает
    - не хранит фреймы, хранит только информацию о траекториях
    - realtime, быстрые алгоритмы,
    - может запускаться прямо в камере видеонаблюдения
  - Оффлайн
    - Использует видео целиком
    - постпроцессинг,
    - «медленный» алгоритм
    - запускается на сервере

Варианты постановки задачи

- Одна камера или несколько?

  Несколько камер – больше информации, можно справляться со случаями «зашел за преграду», но нужно сопоставлять объекты.
  Например:

  - если всего прошло 12 человек, при этом
  - по данным с камеры 1 определили 10 треков
  - по данным с камеры 2 определили 8 треков

  то нужно правильно сопоставить людей с разных камер и определить, что прошло 12 человек, а не 18

Варианты постановки задачи

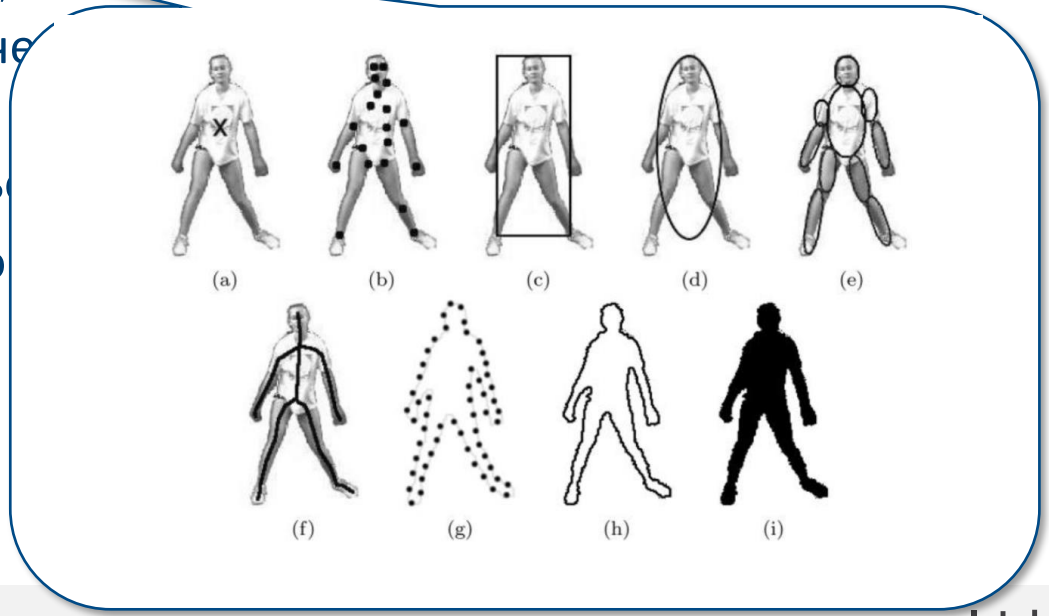
- Объекты одной природы (только люди/машины)? Или произвольные объекты?
  Пример: проследить откуда взялся «ничей» чемодан, потерявшаяся собака, итп
- Если объекты одной природы, то они «жесткие» или «гибкие»?
  - «Жесткие» объекты, которые не могут менять форму, отслеживать намного проще
  - «Гибкие» объекты меняют форму – например, люди могут махать руками и ногами
- Откуда берутся объекты для отслеживания: ручной выбор оператора? или детектор?
  Если детектор – запускается ли он регулярно? или один раз в начале алгоритма?

Варианты постановки задачи

- Как представляются объекты в системе:
  - прямоугольник (bounding box),
  - эллипс,
  - одна центральная точка,
  - несколько ключевых точек,
  - маска,
  - структурная модель объекта (если объекты одной природы)
- Какие характерные черты объектов используются для слежения (расположение на фрейме, внешний вид)

Варианты постановки задачи

- Как представляются объекты в системе:
  - прямоугольник (bounding box),
  - эллипс,
  - одна центральная точка,
  - несколько ключевых точе
  - маска,
  - структурная модель объ
- Какие характерные черты о
  (расположение на фрейме,



Описываемый вариант

Алгоритм из OpenVINO™ pedestrian tracker demo

- Online, realtime (**сравнительно легкий**)
- **Одна камера**
  (**Но в** OpenVINO™ **есть** multi_camera_multi_target_tracking_demo)
- **Объекты – одной природы, только люди**
- **Объекты берутся от детектора**
  детектор запускается регулярно, достаточно часто
  (примерно один раз в каждые три фрейма)
- **Объекты представляются в виде прямоугольников** (bounding box)
- **Характерные черты объектов:**
  расположение объекта, внешний вид объекта

Что такое траектория

Под траекторией T_i мы понимаем последовательность пар описание расположения объекта + индекс фрейма

- $(frame\_index_1, obj\_position_1)$
- $(frame\_index_2, obj\_position_2)$

...

- $(frame\_index_K, obj\_position_K)$

При этом описание расположение объекта может быть как описывалось ранее:
прямоугольник, эллипс, центральная точка, ключевые точки, итп

Что такое траектория

Под траекторией T_i мы понимаем последовательность пар описание расположения объекта + индекс фрейма

- $(frame\_index_1, obj\_position_1)$
- $(frame\_index_2, obj\_position_2)$

...

- $(frame\_index_K, obj\_position_K)$

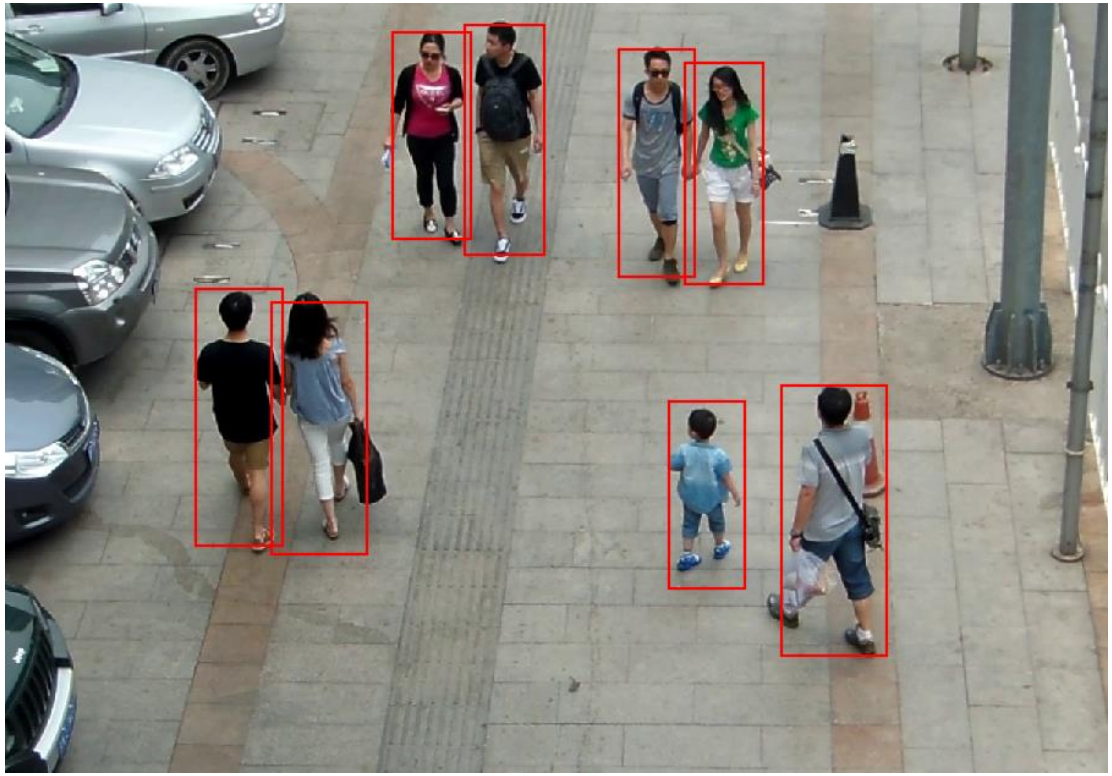
Возможна интерполяция между фреймами

При этом описание расположение объекта может быть как описывалось ранее:
прямоугольник, эллипс, центральная точка, ключевые точки, итп

Детектирование пешеходов

- Для детектирования используются нейронные сети. Например, MobileNet_SSD, натренированная на датасете PASCAL VOC детектирует объекты 20 классов:
    - Аэропланы (индекс класса = 1)
    - Велосипеды (индекс класса = 2)
    - …
    - Пешеходы (индекс класса = 15)
    - …
- Сеть возвращает вектор объектов, каждый из которых описывается как
    - Bounding box – прямоугольник (x_left, y_top, x_right, y_bottom)
    - Индекс класса -- от 1 до 20
    - Confidence (уверенность) – float от 0.0 до 1.0

Детектирование пешеходов



Составление траекторий в pedestrian tracker demo

- Так как хороший детектор с помощью deep learning-а сделать относительно просто, при создании траекторий в pedestrian tracker demo алгоритм рассчитывает на детектор
- Алгоритм считает, что после отсечения «плохих» задетектированных объектов по confidence, детектор выдает точное положение объектов,
- алгоритм составляет траектории из результатов детектора – этот подход называется «tracking by detection»
- При этом подходе алгоритм составляет траектории по индукции

Составление траекторий в pedestrian tracker demo

- Предположим до фрейма № t нашли пешеходов и определили траектории
  $T_1, \ldots, T_N$.
- На фрейме № t+1 нашли пешеходов – прямоугольники $R_1, \ldots, R_D$.
- Как определить траектории?
- Решение: вычислить «схожесть» между траекториями и пешеходами: для каждого i от 1 до N, для каждого j от 1 до D определяем
  a_{ij} – схожесть (affinity) между траекторией T_i и прямоугольником R_j
- $A = (a_{ij})$ – матрица схожести (affinity matrix).
- По этой матрице мы сможем определить какой траектории какой задетектированный пешеход принадлежит.

Схема алгоритма (pedestrian tracker demo)

Алгоритм работает в цикле:

- Получить новый фрейм
- Задетектировать пешеходов на новом фрейме
- Вычислить матрицу схожести A между задетектированными пешеходами и траекториями
- Определить по матрице схожести
    - Каким траекториям какой задетектированный пешеход соответствует
    - Какие пешеходы – новые (не соответствуют ни одной уже существующей траектории)
    - Какие траектории на этом фрейме – без пешехода (ушел, зашел за столб, ошибка детектирования, итп)
- Обновить траектории, в частности удалить (заархивировать) те из них, которым давно не назначали задетектированных пешеходов

Схема алгоритма

Алгори

- Такая схема имеет недостаток: траектории могут рваться (например, если человек зашел за столб, временно вышел из кадра, итп, а потом вернулся).
  Возможное улучшение этой схемы:
  - полученные траектории использовать как «траклеты» – кусочки, из которых собираются более длинные траектории.

  Но это часто делается потом, на этапе постпроцессинга (т.е. алгоритм становится оффлайновым).
  Похожий подход используется и при много-камерном трекинге:
- траклеты с разных камер сопоставляются, чтобы дать каждому человеку уникальный идентификатор; это утяжеляет алгоритм но повышает его качество.

Схема алгоритма

Алгоритм работает в цикле:

- Получить новый фрейм
- Задетектировать пешеходов на новом фрейме
- Вычислить матрицу схожести A между задетектированными пешеходами и траекториями
- Определить по матрице схожести
  - Каким траекториям какой задетектированный пешеход соответствует
  - Какие пешеходы – новые
    (не соответствуют ни одной уже существующей траектории)
  - Какие траектории на этом фрейме – без пешехода
    (ушел, зашел за столб, ошибка детектирования, итп)
- Обновить траектории, в частности удалить старые, которым давно не назначали задетектированных

Как вычисляем матрицу A?

Как по матрице A находим соответствия?

Вычисление матрицы схожести A (1/9)

- Для вычисления «коэффициента схожести» траектории T_i и нового пешехода R_j самый простой способ:
- взять последнее положение траектории T_i в котором мы уверены, – пусть это будет пешеход P_i на фрейме t-k
- и сравнить этого пешехода P_i (задетектированного на фрейме t-k) и нового пешехода R_j
- Как сравнивать:
- По расположению на фрейме
- По размеру
- По внешнему виду

Вычисление матрицы схожести A (1/9)

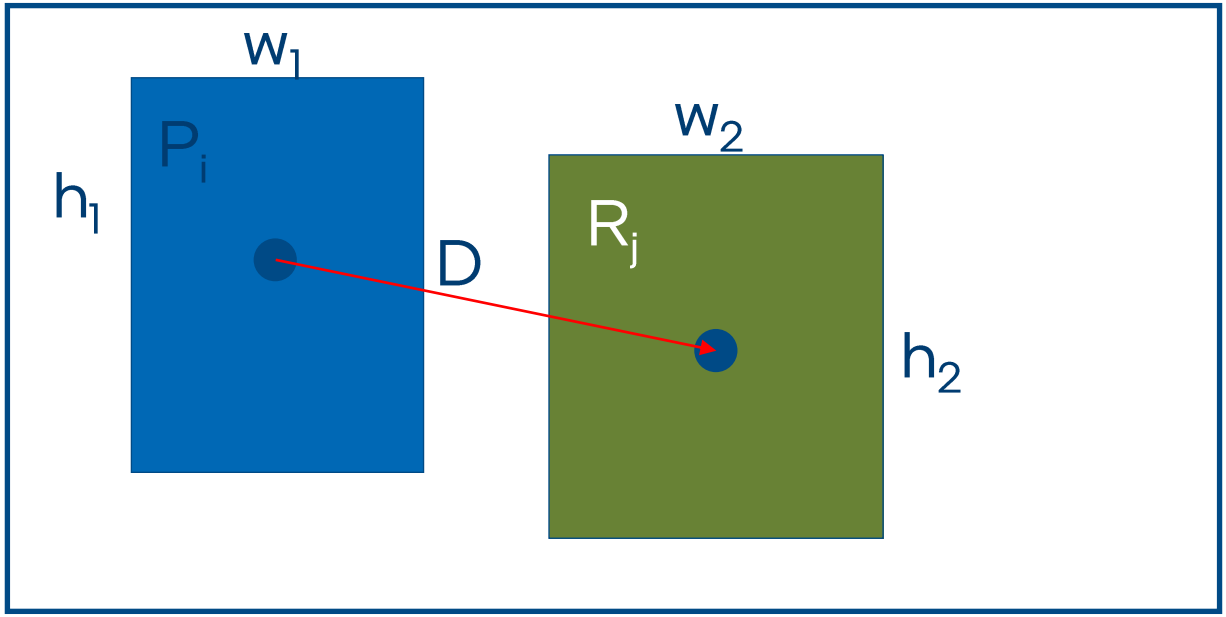
- Для вычисления «коэффициента схожести» траектории T_i и нового пешехода R_j самый простой способ:
- взять последнее положение траект
  – пусть это будет пешеход P_i на фр
- и сравнить этого пешехода P_i (заде
  нового пешехода R_j
- Как сравнивать:
- По расположению на фрейме
- По размеру
- По внешнему виду

Будем вычислять коэффициенты сходства
- affinity_place,
- affinity_shape и
- affinity_appearance

так, чтобы они были $0 \le a \le 1$

Вычисление матрицы схожести A (2/9)

- **Сравниваем пешехода P_i и нового пешехода R_j по положению и размеру.**

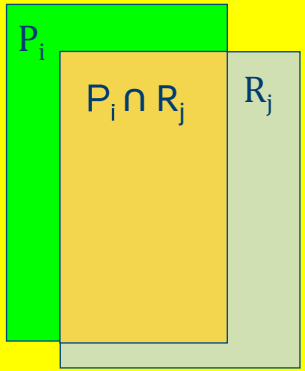


Вычисление матрицы схожести A (3/9)

- **Сравниваем пешехода P_i и нового пешехода R_j по положению и размеру**.
- Пусть D – расстояние между центрами прямоугольников P_i и R_j,
  (w_1,h_1) – ширина и высота прямоугольника P_i
  (w_2, h_2) – ширина и высота прмоугольника R_j
  выберем какое-то число $C_1 > 0$
- Коэффициент сходства по положению на фрейме:
- **Вариант** 1: affinity_place = $\exp(-C_1 * D^2 / (w_1*h_1))$
- **Вариант** 2: affinity_place = $\exp(-C_1 * D / \mathrm{sqrt}(w_1*h_1))$
- **Вариант** 3: affinity_place = **коэффициент** intersection-over-union
- **Вариант** 4: affinity_place = 1.

Вычисление матрицы схожести A (4/9)

- **Сравниваем пешехода P_i и но**
- Пусть D – расстояние между це
  (w_1,h_1) – ширина и высота прям
  (w_2, h_2) – ширина и высота прм
  выберем какое-то число $C_1 > 0$
- Коэффициент сходства по поло
- **Вариант** 1: affinity_place = exp(-C_1 * D / (w_1 h_1))
- **Вариант** 2: affinity_place = exp(-C_1 * D / sqrt(w_1*h_1))
- **Вариант** 3: affinity_place = **коэффициент** intersection-over-union
- **Вариант** 4: affinity_place = 1.



intersection-over-union:

$$\mathrm{IoU}(P_i, R_j) = \frac{area(P_i \cap R_j)}{area(P_i \cup R_j)}$$

– коэффициент от 0 до 1

Вычисление матрицы схожести A (5/9)

- **Сравниваем пешехода P_i и нового пешехода R_j по размеру.**
- Пусть D – расстояние между центрами прямоугольников P_i и R_j,
  (w_1,h_1) – ширина и высота прямоугольника P_j
  (w_2, h_2) – ширина и высота прмоугольника R_j
  выберем какое-то число $C_2 > 0$
- Коэффициент сходства по форме и размеру:
- **Вариант 1:** $affinity\_shapes = \exp(-C_2 * (|w_1-w_2|/w_1 + |h_1-h_2|/h_1))$
- **Вариант 2:** $affinity\_shapes = \exp(-C_2 * |area(P_i) – area(R_j)| / area(P_i))$
- **Вариант 3:** $affinity\_shapes = 1$.

Вычисление матрицы схожести A (6/9)

- **Сравниваем пешехода P_i и нового пешехода R_j по внешнему виду.**

- Вырезать из фрейма t+1 прямоугольник R_j,
- Вырезать из фрейма t-k прямоугольник P_i,
- Уменьшить размеры вырезанных картинок до какого-то заранее установленного размера, например 16x32 (т.е. 16 в ширину, 32 в высоту)
- Вычислить affinity_appearance одним из следующих методов

Простой метод 1:

Выписать все пиксельные значения для P_i в строчку **x** длиной 16*32*3 = 1536, сделать то же для и R_j – получить строчку **y**, вычислить коэффициент **affinity_appearance** = exp(-C * |**x** – **y**|)

Вычисление матрицы схожести A (7/9)

- **Сравниваем пешехода P_i и нового пешехода R_j по внешнему виду.**

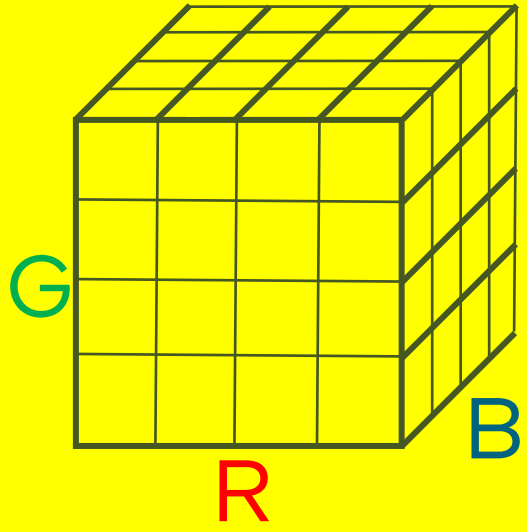
- Вырезать из фрейма t+1 прямоугольник R_j,
- Вырезать из фрейма t-k прямоугольник P_i,
- Уменьшить размеры вырезанных картинок до какого-то заранее установленного размера, например 16x32 (т.е. 16 в ширину, 32 в высоту)
- Вычислить affinity_appearance одним из следующих методов

Простой метод 2:

Посчитать **трехмерную цветовую гистограмму** всех RGB значений пикселов для P_i – получить вектор **x**, сделать то же для и R_j – получить вектор **y**, вычислить коэффициент affinity_appearance как **косинус угла между векторами**

Вы
- **С**

Чтобы посчитать **трехмерную цветовую гистограмму** нужно все разбить куб всех возможных RGB значений, который имеет размер 256 × 256 × 256 на одинаковые кубики размером, например, 64 × 64 × 64
– всего будет 64 таких кубика
Затем нужно посчитать для каждого кубика число – сколько пикселов имеют цвет, попадающий в этот кубик, и выписать эти числа в один вектор длиной 64



Посчитать **трехмерную цветовую гистограмму** всех RGB значений пикселов для P_i – получить вектор **x**, сделать то же для и R_j – получить вектор **y**, вычислить коэффициент **affinity_appearance** как **косинус угла между векторами**

Вычисление матрицы схожести A (8/9)

- **Сравниваем пешехода P_i и нового пешехода R_j по внешнему виду.**

- Вырезать из фрейма t+1 прямоугольник R_j,
- Вырезать из фрейма t-k прямоугольник P_i,
- Уменьшить размеры вырезанных картинок до какого-то заранее установленного размера, например 16x32 (т.е. 16 в ширину, 32 в высоту)
- Вычислить affinity_appearance одним из следующих методов

Сложный метод:

Натренировать **сетку**, которая по вырезанной картинке выдает вектор (например, float[256]), описывающий внешний вид пешехода так, чтобы **косинус угла между векторами** был метрикой сходства.

Вычисление матрицы схожести A (9/9)

- **Собираем общую метрику сходства пешехода P_i и нового пешехода R_j как**
- a_{ij} = affinity_place * affinity_shapes * affinity_appearance
- Это дает нам общую метрику сходства между траекторией и новым задетектированным пешеходом.
- При этом мы выбрали способ вычисления affinity-коэффициентов так, чтобы все affinity-коэффициенты были от 0 до 1:
  $0 \leq a_{ij} \leq 1$.

Нахождение соответствий траектории-пешеходы по матрице схожести A (1/6)

- В простом случае можем найти соответствия сразу.
- Простой случай – это когда:
  - в каждой строке i есть только один максимальный элемент с индексом f(i)
  - Для разных строк эти элементы разные $\forall\, i \neq j$ выполняется $f(i) \neq f(j)$
  - – то есть, каждый новый пешеход подходит только одной траектории, а траектории не конкурируют за новых пешеходов

$$A = \begin{pmatrix} 0.1 & 0.9 & 0.0 & 0.2 \\ 0.2 & 0.0 & 0.6 & 0.2 \\ 0.3 & 0.1 & 0.1 & 0.2 \\ 0.6 & 0.7 & 0.8 & 0.9 \end{pmatrix}$$

f(1) = 2
f(2) = 3
f(3) = 1
f(4) = 4

Нахождение соответствий траектории-пешеходы по матрице схожести A (2/6)

- Дополнительный шаг после того, как нашли соответствия – **не назначать** трактории T_i нового пешехода R_j, если a_{ij} меньше некоторого порога.
- В этом случае будем считать, что траектории **не нашлось соответствий**.
- Если некоторому прямоугольнику R_j не нашлось соответствий – начало **новой** траектории

$$A = \begin{pmatrix} 0.1 & 0.9 & 0.0 & 0.2 \\ 0.2 & 0.0 & 0.6 & 0.2 \\ 0.3 & 0.1 & 0.1 & 0.2 \\ 0.6 & 0.7 & 0.8 & 0.9 \end{pmatrix}$$

Если порог равен 0.5:

f(1) = 2

f(2) = 3

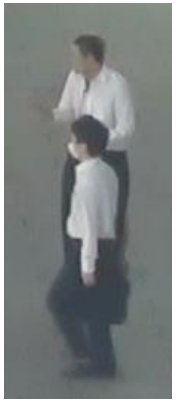
f(3) = None

f(4) = 4

Нахождение соответствий траектории-пешеходы по матрице схожести A (3/6)

- Сложный случай – когда траектории начинают конкурировать за пешеходов или наоборот – пешеходы конкурируют за траектории.
- – нужно выбрать, какие пешеходы будут начинать новые траектории, а какие продолжать старые
  – нужно выбрать какие траектории останутся без пешеходов

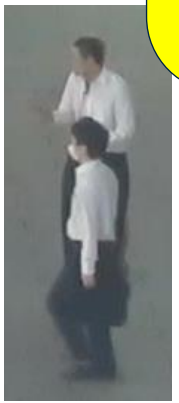
$$A = \begin{pmatrix} 0.1 & 0.9 & 0.0 & 0.2 \\ 0.2 & 0.0 & 0.6 & 0.2 \\ 0.2 & 0.1 & 0.8 & 0.2 \end{pmatrix}$$



Нахождение соответствий траектории-пешеходы по матрице схожести A (3/6)

- Сложный случай – когда траектории начинают конкурировать за пешеходов или наоборот – пешеходы конкурируют за траектории.
- – нужно выбрать, какие пешеходы будут начинать новые траектории, а какие продолжать старые
  – нужно выбрать какие траектории останутся без пешеходов

$A =$



В 80-90% случаев срабатывает простой подход, но качество алгоритма трекинга часто определяется оставшимися 10%

Нахождение соответствий траектории-пешеходы по матрице схожести A (4/6)

- В сложном случае можем сформулировать задачу так: назначить траекториям $T_1,\dots,T_N$ новых пешеходов $R_1, \dots, R_D$ по матрице сходства A так, чтобы
- Одной траектории i соответствовало не более одного нового пешехода f(i), то есть

$$f: \{1, \dots, N\} \to \{1, \dots, D\} \cup \{None\}$$

- Каждый пешеход назначается не более чем одной траектории, то есть

$$\forall i_1 \neq i_2 \quad \text{если } f(i_1) \neq None \text{ и } f(i_2) \neq None, \text{то } f(i_1) \neq f(i_2)$$

- сумма a_{ij} по назначениям была максимальна:

$$\sum_i a_{i,f(i)} \to \max$$

Нахождение соответствий траектории-пешеходы по матрице схожести A (5/6)

В сложном случае задача поиска соответсвий сводится к **задаче о назначениях**.

В канонической формулировке задача о назначениях выглядит так:

- имеется N работ и **то же самое** число *исполнителей*
- любой исполнитель i может быть назначен на выполнение любой (но только одной) работы j = f(i), с затратами a(i,j) ≥ 0
- разным исполнителям назначаются разные работы: $\forall i_1 \neq i_2 \quad f(i_1) \neq f(i_2)$
- нужно выполнить работы с минимальными суммарными затратами

$$\sum_i a(i, f(i)) \rightarrow min$$

Нахождение соответствий траектории-пешеходы по матрице схожести A (5/6)

В сложном случае задача поиска соответсвий сводится к **задаче о назначениях**.

В канонической формулировке задача о назначениях выглядит так:

- имеется N работ и **то же самое** число *исполнителей*
- любой исполнитель i может быть назначен на выполнение любой (но только одной) работы j = f(i), с затратами a(i,j) ≥ 0
- разным исполнителям назначаются разные работы: $\forall i_1 \neq i_2 \quad f(i_1) \neq f(i_2)$
- н боты с минимальными суммарными затратами

Стоимость назначения

$$\sum_i a(i, f(i)) \rightarrow min$$

Нахождение соответствий траектории-пешеходы по матрице схожести A (6/6)

Чтобы свести нашу задачу к задаче о назначениях нужно

1. Сделать матрицу A квадратной
   Для этого мы можем добавить некоторое количество «фиктивных» строк и столбцов, заполненных нулями
2. Перейти от задачи «найти максимум» к задаче «найти минимум».
   Для этого мы воспользуемся тем, что у нас $0 \leq a_{ij} \leq 1$.
   Мы вычтем из 1.0 все значения матрицы A:
   $$a'(i,j) = 1 - a_{ij}$$
   В результате у нас получится неотрицательная матрица A′ и если мы решим задачу минимизации $\sum_i a'(i, f(i)) \rightarrow min$ это будет эквивалентно исходной задаче

Решение задачи о назначениях

Для решения задачи о назначениях можно использовать классический Kuhn-Munkres алгоритм (также известный как «венгерский алгоритм»)

- Трудоемкость $O(N^3)$
- Python:
  - scipy.optimize.linear_sum_assignment из пакета scipy
  - Munkres из пакета munkres3
- C++
  - Google OR-tools https://developers.google.com/optimization/reference/graph/linear_assignment
  - Many custom implementations


intel

Схема алгоритма