MaxMovingRegion
(поиск наибольшего движущегося объекта)

1. Алгоритм получения бинарного изображения (по двум соседним кадрам), определяющее наличие или отсутствие движения в пикселе изображения:

1.1. Сглаживание кадров фильтром Гаусса (cv::GaussianBlur), чтобы избавиться от шумов.

1.2. Из сглаженного текущего кадра вычитается сглаженный предыдущий (cv::absdiff).

1.3. Сравнение значения полученной разности с некоторым пороговым значением – Main::MOTION_THRESHOLD (cv::threshold). Если значение пикселя больше порогового, то этот пиксель принадлежит движущемуся объекту, иначе отбрасываем его. Теперь мы получили бинарное изображение, где нуль означает, что пиксель не движется, отличное от нуля значение – пиксель движется.

1.4. Применение морфологических операций закрытия и открытия (cv::morphologyEx), чтобы избавиться от движущихся регионов малого размера (шумы камеры).

2. Определение местоположения наибольшего движущегося объекта (Main::maxMovingRegion):

2.1 Ожидание движения в видеопотоке:

а) Считывание кадра (imAcqGetImg) и отображение (Gui::showImage).

б) Перевод изображения в оттенки серого (cv::cvtColor).

в) Определение наличия или отсутствия движения (Main::isMotionInBox - определение присутствия ненулевых точек на бинароном изображении, полученном из текущего и предыдущего кадров (см. пункт 1.).

2.2. Накопление информации о движении:

а) Считывание кадра (imAcqGetImg) и отображение (Gui::showImage).

б) Перевод изображения в оттенки серого (cv::cvtColor).

в) Наложение бинарного изображения, полученного из текущего и предыдущего кадров (см. пункт 1.) на изображение истории движения – motionHistoryImage (cv::updateMotionHistory). На нём

нарисованы движущиеся контуры за последние время – motionHistoryDuration (в милисекундах). Интенсивность пикселей контура обратно пропорциональна времени, которое прошло от измерения контура до данного момента. Т.е. чем раньше был получен движущийся контур, тем он бледнее изображён на изображении истории движения.

2.3. Выделение регионов с различными движениями на изображение истории движения (cvSegmentMotion).

Отбрасываем все регионы, хотя бы одна сторона которых меньше некоторого значения – Main::MIN_BOX_SIDE.

Объединение пересекающихся регионов (Main::unionRectangls). Для этого используется выделение связных компонент графа (вершина – регион, ребро – отношение пересечения регионов) с помощью алгоритма поиска в ширину.

Выбор наибольшего (по площади) региона среди регионов, отстоящих от края кадра на 5 пикселей. Если ни одного региона не осталось, то возвращаемся к пункту 2.2.

2.4. Уточнение размеров найденного региона (Это необходимо потому, что объект может значительно сдвинуться в течение времени накопления информации о движении):

а) Поиск контуров (cv::findContours) и объединение их в один в найденном регионе на последнем бинарном изображении движения.

б) Поиск рамки (ограничивающего прямоугольника) по контуру (cv::boundingRect).

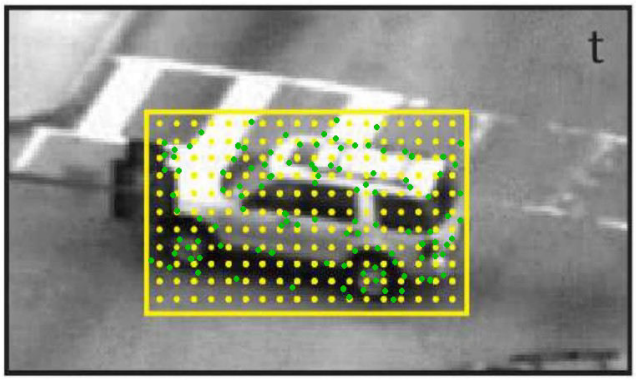
2.5. Если хотя бы одна сторона рамки меньше некоторого значения – Main::MIN_BOX_SIDE, то возвращаемся к пункту 1, в противном случае объект найден.

MedianFlowTracker
(трекер медианного потока)

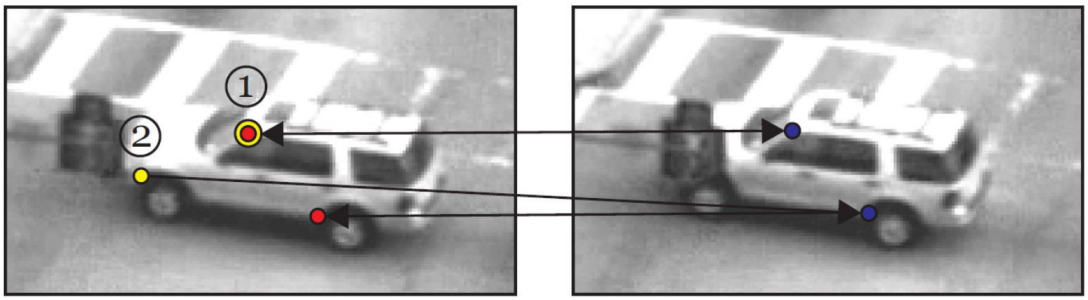
Поиск ограничивающего прямоугольника (рамки) объекта на текущем кадре (tld::MedianFlowTracker::track):

1. Поиск рамки на текущем изображении (fbtrack):

1.1. Выбор точек в рамке на предыдущем изображении для слежения: получение наиболее видных углов (cvGoodFeaturesToTrack) (зеленые на рисунке)не более 50 и узлов сетки (getFilledBBPoints) (желтые на рисунке) так, чтобы было в сумме 100 точек.



1.2. Слежение за точками от предыдущего кадра к текущему и обратно (trackLK):



Для слежения за точками используется метод вычисления оптического потока алгоритм Лукаса-Канаде (cvCalcOpticalFlowPyrLK). Далее точки, для которых не найден результат хотя бы в одну сторону, не рассматриваются.

Для каждой точки на предыдущем кадре (желтые на рисунке) и соответствующей ей на текущем кадре (синие на рисунке) подсчитывается нормированная кросс-корреляция в области размером 10 X 10 (normCrossCorrelation).

Также подсчитывается расстояние между начальными точками (желтые на рисунке) и соответствующими им точкам, прошедшие от предыдущего кадра к текущему кадру и обратно (красные на рисунке).

1.3. Вычисляются медианы (getMedian) для массива нормированных кросс-корреляций и массива расстояний (ошибки прямого и обратного направлений слежения). Далее рассматриваются точки, для которых

ошибки прямого и обратного направлений слежения меньше (соответствующей) медианы и нормированная кросс-корреляция больше (соответствующей) медианы.

1.4. Предсказание рамки на текущем кадре по найденным точкам (predict). Изменяет размер и сдвигает (вычисляются медианы (getMedian) смещений по x, по y и медиана коэффициентов масштабируемости для каждой точек) предыдущей рамки, получая новую.

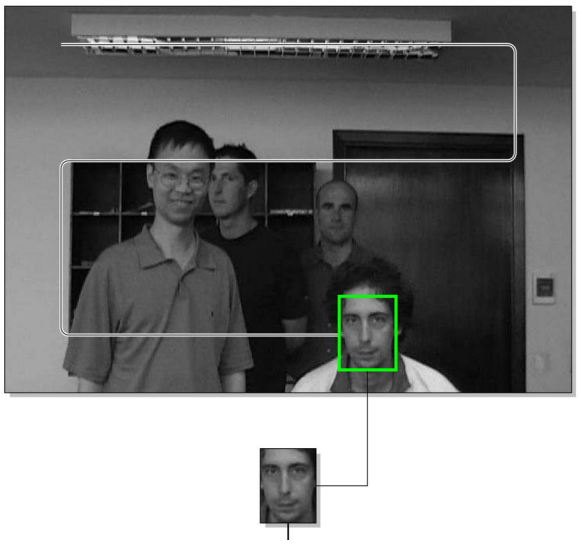
2. Проверка успешности (медиана массива ошибок прямого и обратного направлений слежения должна быть <= 10) и корректности рамки (каждая из строн > 0, и она не выходит за границы изображения).

Algorithm 1 Recursive Tracking

Input: B_I, I, J

$p_1 \dots p_n \leftarrow \text{generatePoints}(B_I)$
for all p_i **do**
$p'_i \leftarrow LK(p_i)$
$p''_i \leftarrow LK(p'_i)$
$\varepsilon_i \leftarrow |p_i - p''_i|$
$\eta_i \leftarrow NCC(W(p_i), W(p'_i))$
end for
$med_{NCC} \leftarrow \text{median}(\eta_1 \dots \eta_n)$
$med_{FB} \leftarrow \text{median}(\varepsilon_1 \dots \varepsilon_n)$
if $med_{FB} > \theta_{FB}$ **then**
$B_J = \emptyset$
else
$C \leftarrow \{(p_i, p'_i) \mid p'_i \neq \emptyset, \varepsilon_i \leq med_{FB}, \eta_i \geq med_{ncc}\}$
$B_J \leftarrow \text{transform}(B_I, C)$
end if

DetectorCascade
(каскадный детектор)



Поиск рамки среди множества скользящих окон различных масштабов на текущем кадре (tld::DetectorCascade::detect).

Поля dx и dy между двумя смежными окнами одинакового масштаба равны 1/10 от значения стороны исходной рамки.

Для генерации окон различных масштабов используется масштабный коэффициент $s = 1.2^a$, $a \in \{-10 \ldots 10\}$ для исходной рамки.

Минимальная сторона рассматриваемых окон - 25.

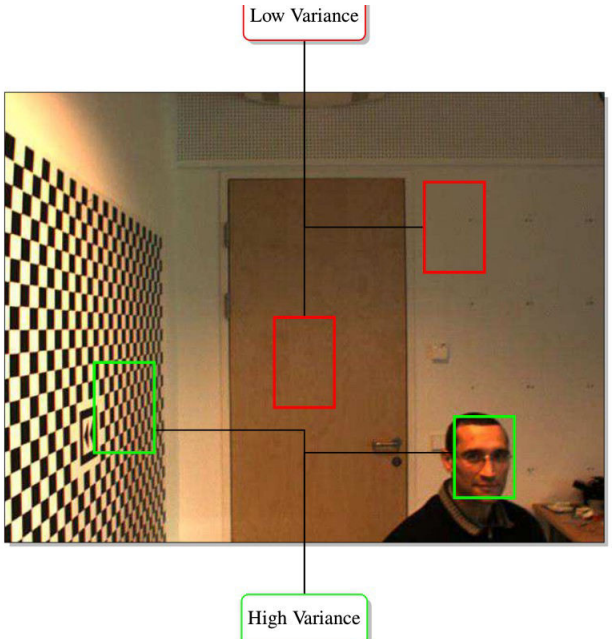
Каждое окно последовательно проходит через каскад: VarianceFilter, EnsembleClassifier, NNClassifier и оставшиеся объединяются Clustering.

Algorithm 3 Detection Cascade

$D_t \leftarrow \emptyset$

$I' \leftarrow \text{integralImage}(I)$;
$I'' \leftarrow \text{integralImage}(I^2)$;
for all $B \in \mathcal{R}$ **do**
  if $\text{isInside}(B, F)$; **then**
    if $\text{calcVariance}(I'(B), I''(B)) > \sigma^2_{min}$ **then**
      if $\text{classifyPatch}(I(B)) > 0.5$ **then**
        $P \leftarrow \text{resize}(I(B), 15, 15)$
        if $\text{matchTemplate}(I(B)) > \theta^+$ **then**
          $D_t \leftarrow D_t \bigcup B$
        end if
      end if
    end if
  end if
end for
$D_t \leftarrow \text{cluster}(D_t)$

VarianceFilter
(дисперсионный фильтр)



1. Установка кадра (изображение I(x,y)) в качестве текущего (tld::VarianceFilter::nextIteration) с подсчетом интегральной матрицы ($I'(x,y)$) и интегральной матрицы ($I''(x,y)$) для квадратов значений пикселей (tld::IntegralImage::calcIntImg).

Формула по определению:

$$I'(x,y) = \sum_{x' \le x, y' \le y} I(x', y').$$

Рекурсивная формула:

$$I'(x,y) = I(x,y) + I'(x-1,y) + I'(x,y-1) - I'(x-1,y-1).$$

где $I'(0,y)=0$, $I'(x,0)=0$.

2. Фильтрация окна по однородности (tld::VarianceFilter::filter): подсчет дисперсии (σ^2) в окне (tld::VarianceFilter::calcVariance) по интегральным матрицам и сравнение ее с минимальным значением (minVar).

$I'(B)$ - сумма пикселей внутри прямоугольника B, заданный параметрами x, y, w ,h:

$$\sum_{i=1}^{n} x_i = I'(x-1,y-1) - I'(x+w,y-1) - I'(x-1,y+h) + I'(x+h,y+w).$$

Формула для подсчета дисперсии:

$$\sigma^2 = \frac{1}{n} I''(B) - [\frac{1}{n} I'(B)]^2$$

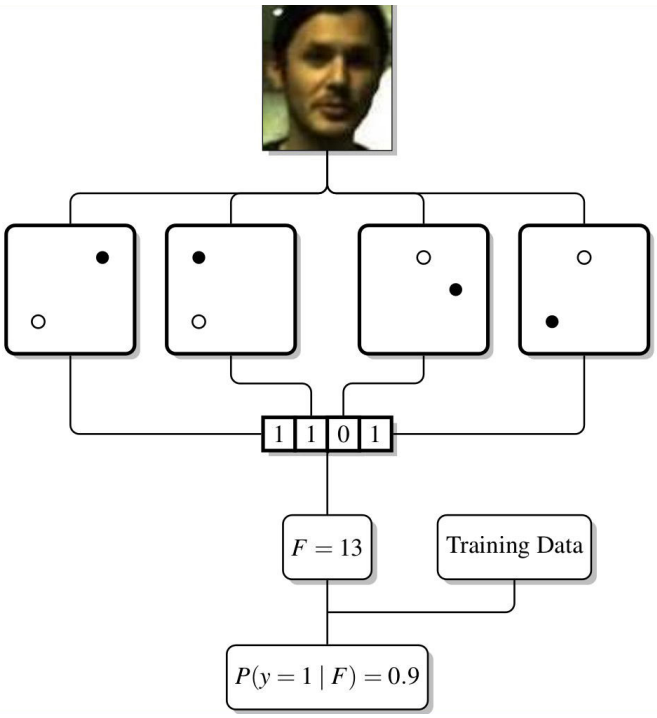
EnsembleClassifier
(ансамблевый классификатор)

1. Установка текущего изображения в ансаблевый классификатор (tld::EnsembleClassifier::nextIteration).

2. Фильтрация окна (tld::EnsembleClassifier::filter): классификация окна (tld::EnsembleClassifier::classifyWindow) по решению деревьев.

Этот классификатор имеет форму рандомизированного леса (множество деревьев) и основывает свое решение на значениях интенсивности нескольких пикселей.

Каждое дерево дает отклик на окно (tld::EnsembleClassifier::calcFernFeature) в виде вектора признаков (в десятичном представлении это номер ветки в дереве, по которой проходит окно). По этим векторам вычисляется вероятность P_{pos} (EnsembleClassifier::calcConfidence) принадлежности окна к изображению объекта. Если эта вероятность меньше 0.5, то окно отфильтровывается.



В на рисунке выше показана схема отклика дерева на окно. В каждом из четырех полей (обозначение результата измерения признака), расположенных в схеме под окном, изображены черная и белая точки. Каждая из этих точек соответствует пикселю в окне. Для каждого поля пиксель в позиции белой точки более ярок, чем пиксель в позиции черной точки. Позиции этих точек (в относительных координатах (от 0 до 1) в окне) определяются один раз при запуске (tld::EnsembleClassifier::initFeatureLocations).

Algorithm 2 Efficient Fern Feature Calculation

Input: I

Output: F

$F \leftarrow 0$

for $i = 1 \ldots S$ **do**

$F \leftarrow 2 \times F$

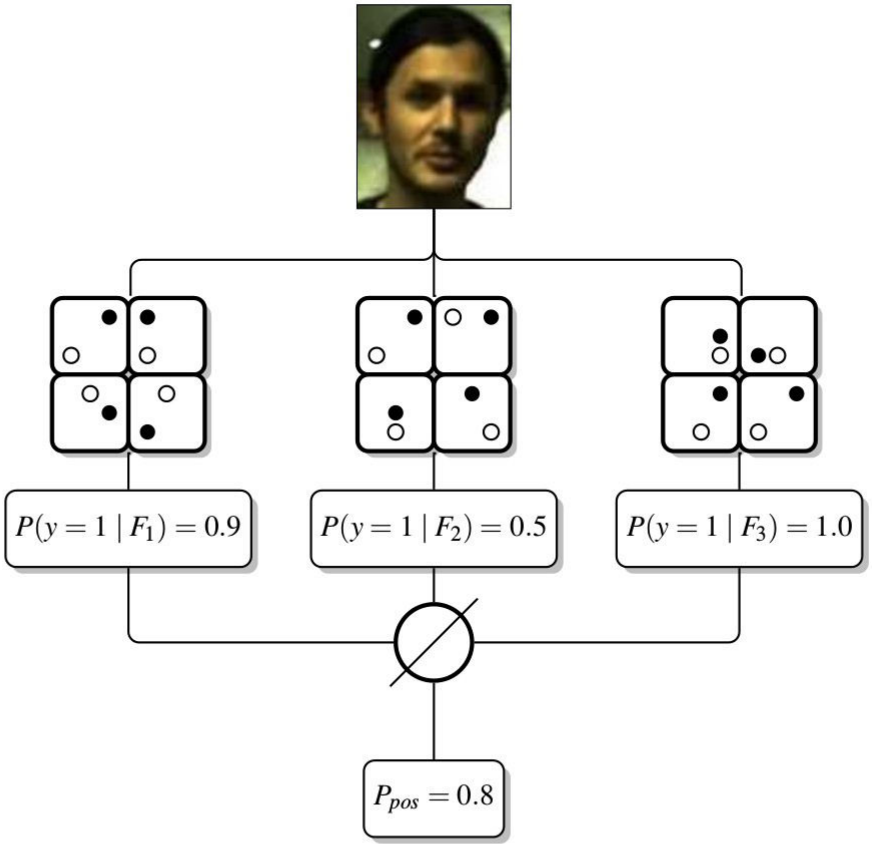
if $I(d_{i,1}) < I(d_{i,2})$ **then**

$F \leftarrow F + 1$

end if

end for

В на рисунке ниже показана схема (в каждом из 3-х деревьев по 4 измерения признака) для получения вероятности P_{pos} (среднее арифметические из вероятностей, полученных каждым деревом).



В приложении используется 10 деревьев, в каждом из них измеряется 13 признаков.

NNClassifier
(классификатор ближайшего соседа)

Этот классификатор фильтрует окно (tld::NNClassifier::filter), используя метод сравнения с положительными и отрицательными примерами (шаблонами, патчами).

(a) Positive Examples

(b) Negative Examples

При классификации окна (NNClassifier::classifyWindow), оно сжимается до размера 15×15 и нормируется относительно среднего (tld::tldNormalizeImg), получается патч (tld::tldExtractNormalizedPatch).

Для сравнения двух патчей P_1 и P_2 используется нормализованный коэффициент корреляции (NNClassifier::ncc):

$$\mathrm{ncc}(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x) - \mu_1)(P_2(x) - \mu_2)}{\sigma_1 \sigma_2},$$

где μ_1, μ_2, σ_1 и σ_2 - средние и стандартные отклонения патчей P_1 и P_2.

Для определения расстояния (от 0 до 1) между двумя патчами используется формула:

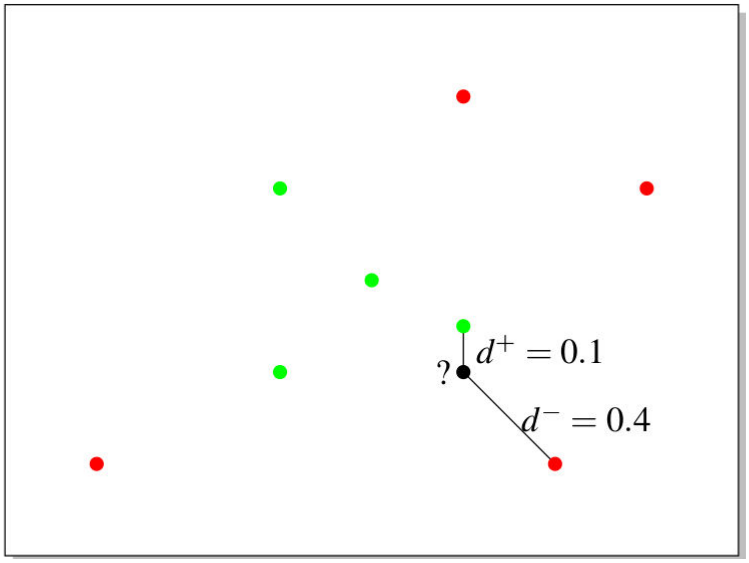
$$\mathrm{d}(P_1, P_2) = 1 - \frac{1}{2}(\mathrm{ncc}(P_1, P_2) + 1).$$

Далее, определяются наименьшие расстояния от патча (полученого из окна) до положительных (d^+) и отрицательных (d^-) примеров и объединяются в одно значение (p^+), характеризующее уверенность нахождения объекта в патче (tld::NNClassifier::classifyPatch):

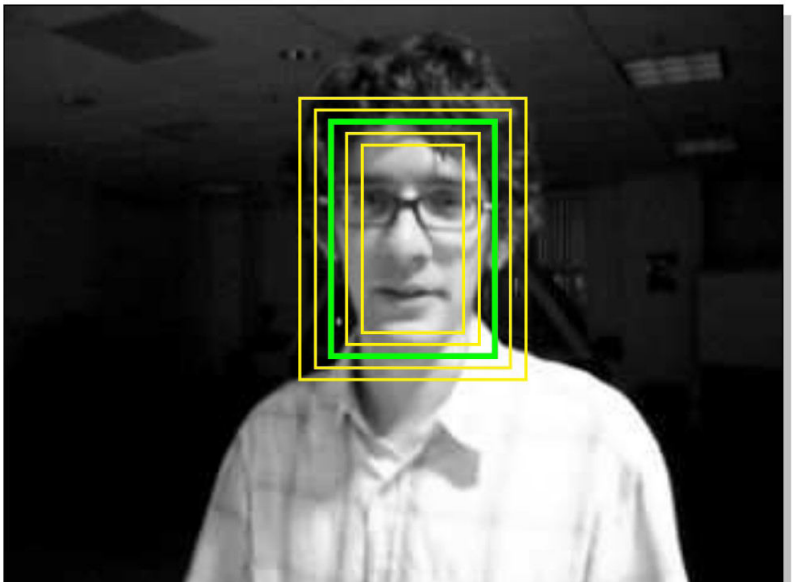
$$p^+ = \frac{d^-}{d^- + d^+}.$$

Если p^+ больше, чем $\theta^+=0.65$, то окно не отфильтровывается.

На рисунке ниже зеленые точки соответствуют положительным примерам, красные точки соответствуют отрицательным примерам. Черная точка, отмеченная вопросительным знаком, соответствует патчу, полученному из исследуемого окна.



Clustering
(кластеризация)



На рисунке выше изображена ситуация многозначного обнаружения объекта. Правильное обнаружение отмечено зеленым, лишние - желтым.

Окна, прошедшие ранее фильтрацию, объединяются в одно (tld::Clustering::clusterConfidentIndices):

1. Определение числа кластеров и распределение окон по ним (tld::Clustering::cluster). Если кластеров обнаружено несколько, то считается, что объект на изображении не найден.

Принципы кластеризации:

1.1 Вычисляются попарно перекрытия между всеми окнами (tld::Clustering::calcDistances). Измерения перекрытия (от 0 до 1) между двумя окнами B_1 и B_2 (tld::tldOverlapOne) происходит по формуле:

$$overlap = \frac{B_1 \cap B_2}{B_1 \cup B_2} = \frac{I}{(B_1 + B_2 - I)}.$$

1.2 Для каждого окна ищется самое близкое (в смысле перекрытия) окно.

1.3. Распределяем каждое окно по кластерам:

а) Если рассматриваемое окно и самое близкое к нему окно не распределены:

при перекрытии меньше, чем порог (cutoff=0.5), окна распределяются в один кластер, в противном случае - в различные кластеры.

б) Если одно из этих окон распределено:

при перекрытии меньше, чем порог (cutoff=0.5), нераспределенное окно помещяется в тот же кластер, что и распределенное, в противном случае - в новый кластер.

в) Если оба из этих окон распределены:

при перекрытии меньше, чем определенный порог (cutoff=0.5), кластеры объединяются.

2. Усреднение всех параметров окон (x, y, w, h) принадлежащих кластеру (tld::Clustering::calcMeanRect).

fuseHypotheses
(объединение предположений)

Объединение рамок-кадидатов, полученных в блоках MedianFlowTracker (trackerBB) и DetectorCascade (detectorBB) (tld::TLD::fuseHypotheses):

1. Подсчет уверенностей p^+ для рамок-кандидатов (NNClassifier::classifyBB) при их наличии с помощью классификатора ближайшего соседа, см. раздел NNClassifier.

2. Возможны 4 случая:

1) отсутствие trackerBB, отсутствие detectorBB.

Результат отсутствует.

2) отсутствие trackerBB, наличие detectorBB.

Результат detectorBB.

3) наличие trackerBB, отсутствие detectorBB.

Результат trackerBB. Подсчет допустимости для определения необходимости обучения. Если уверенность p^+ для рамки trackerBB больше, чем $\theta^+=0.65$; или больше, чем $\theta^-=0.5$, и предыдущая допустимость имела значение истина, то текущая допустимость получает значение истина. По умолчанию допустимость имеет значение ложь.

4) наличие trackerBB, наличие detectorBB.

При уверенности рамки-кандидата detectorBB больше, чем уверенность рамки-кандидата trackerBB, и перекрытии этих рамок (tld::tldOverlapRectRect) не более, чем на 0.5, результатом является detectorBB. В противном случае - как в случае 3).

learn
(обучение)

Если допустимость имеет значение истина (см. раздел fuseHypotheses), то происходит обучение (tld::TLD::learn):

1. Подсчет перекрытий между текущей рамкой и всеми окнами (tld::tldOverlapRect). Отбор окон с перекрытием > 0.6 в качестве положительных кандидатов для обучения. Сортировка этих окон по перекрытию (std::sort с параметром tld::tldSortByOverlapDesc). Отбор окон с перекрытием < 0.2 и с вероятность (полученной ранее в EnsembleClassifier) > 0.5 в качестве отрицательных кандидатов для обучения.

2. 10 первых положительных кандидитов и все отрицательные кандидаты используются для обучения ансамблевого классификатора (tld::EnsembleClassifier::learn):

Для положительных кандидатов, имеющих вероятность < 0.5, и для отрицательных кандидатов, имеющих вероятность > 0.5, (вероятности пересчитываются — tld::EnsembleClassifier::calcConfidence) происходит обновление позитивных и негативных веток и соответственно вердиктов деревьев (tld::EnsembleClassifier::updatePosteriors).

3. Патчи (tld::tldExtractNormalizedPatch), полученные из текущей рамки (в качестве положительного кандидата) и всех отрицательных кандидатов, используются для обучения классификатора ближайшего соседа (tld::NNClassifier::learn):

Положительный патч, имеющий уверенность $p^+ <= \theta^+=0.65$, и отрицательные патчи, имеющие уверенность $p^+ >= \theta^-=0.5$, (уверенности пересчитываются — tld::NNClassifier::classifyPatch) заносятся в положительные и отрицательные примеры классификатора ближайшего соседа.

Algorithm 5 Applying P/N-constraints

Input: I, B_t

for all $B \in \mathcal{R}$ **do**
  if $\text{overlap}(B, B_t) > 0.6$ **and** $\text{classifyPatch}(I(B)) < 0.5$ **then**
    for $k = 1 \ldots M$ **do**
      $F \leftarrow \text{calcFernFeatures}(I(B), k)$
      $p_{F_k}[F] \leftarrow p_{F_k}[F] + 1$
    end for
  else if $\text{overlap}(B, B_t) < 0.2$ **and** $\text{classifyPatch}(I(B)) > 0.5$ **then**
    for $k = 1 \ldots M$ **do**
      $F \leftarrow \text{calcFernFeatures}(I(B), k)$
      $n_{F_k}[F] \leftarrow n_{F_k}[F] + 1$
    end for
    if $p^+_{B_t} > \theta^-$ **then**
      $\mathcal{P}^- \leftarrow \mathcal{P}^- \cup I(B)$
    end if
  end if
end for
if $p^+_{B_t} < \theta^+$ **then**
  $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup I(B_t)$
end if