Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Определение траекторий движения объектов в условиях перекрытия

Курсовая работа студента 444 группы

Самарина Алексея Владимировича

Научный руководитель к.ф.-м.н, доц. А.Т.Вахитов

Санкт-Петербург

2014

Оглавление

Введение

В настоящее время, в области криминалистики и в различных системах безопасности, актуальна задача определения траекторий движения объектов по видеозаписи (трекинг). Существует множество подходов к решению данной задачи (см. приложение 1). Тем не менее, почти у всех методов присутствуют недостатки, такие, как определение элементов фона как движущихся объектов, низкая производительность системы трекинга, многочисленные ошибки при перекрытии объектов, игнорирование объектов при их малом размере или малой амплитуде движения, и т. д. К сожалению, одновременного решения всех вышеперечисленных и многих других задач, на данный момент не существует (задача создания универсальной системы трекинга остаётся открытой). Судя по всему, данная проблема является неразрешимой в принципе, так как универсальный трекер должен решать очень широкий класс задач (варьироваться может как тип отслеживаемых объектов, так и их скорость, а так же вероятные траектории их движения). Таким образом, данная работа посвящена решению задачи определения траекторий движения объектов в условиях их перекрытия.

Постановка задачи

Следует отметить что, первым шагом для решения любой задачи трекинга является определение условий, для которых предназначается данная система. Таким образом, нам следует определить тип объектов, их возможную скорость перемещения и возможные типы траекторий, по которым объекты трекинга способны совершать перемещение. Не менее важными, являются параметры калибровки камеры (для нашей системы особенно важны высота крепления и угол наблюдения). В большинстве случаев, определение траекторий применяется к людям, которые перемещаются по комнатам, залам и коридорам зданий, а так же, к участникам различных спортивных состязаний (например, к членам футбольных команд для сбора статистических данных по перемещению игроков). Таким образом, объектами трекинга будут являться передвигающиеся пешком люди, благодаря чему, траектории движения и ситуации перекрытия силуэтов при выбранном ракурсе наблюдения (в случае одной камеры) могут быть самыми разнообразными, но при этом, проблем со скоростью движения объектов быть не должно (в отличие от трекинга различных транспортных средств).

Таким образом, у нас имеется откалиброванная согласно заданным параметрам камера, и произведенная с её помощью видеозапись передвижения пешеходов (сама камера в период фиксации видеопотока является неподвижной). Требуется реализовать программу для максимально точного и быстрого определения траекторий их движения пешеходов с учётом возможных случаев перекрытия силуэтов, а также проанализировать полученный инструмент и его возможные аналоги.

Рис. 1: Пример перекрытия силуэтов пешеходов при трекинге.

Обзор существующих решений

Решение задачи определения траекторий в условиях перекрытия силуэтов, связано с большими дополнительными расходами, которые сильно снижают производительность системы трекинга в целом, уменьшают зоны возможной установки камер, при этом, значительно повышая стоимость системы. Поэтому, большинство существующих систем трекинга не поддерживают данную функциональность, либо проблему такого рода пытаются решить путём выбора ракурса, при котором вероятность перекрытия минимальна.

Тем не менее, возможность восстановления траекторий движения объектов в условиях перекрытия может показаться очень интересной для специалистов различных областей. Благодаря чему, интерес к данной проблеме не угасает. Одним из экспериментальных подходов к решению данной проблемы является метод “Multi-Camera People Tracking with a Probabilistic Occupancy Map”, разработанный группой учёных Francois Fleuret, Jerome Berclaz, Pascal Fua [1].

Multi-Camera People Tracking with a Probabilistic Occupancy Map

Данный метод был разработан для поиска траекторий передвижения пешеходов при условии возможного перекрытия силуэтов. Результаты работы исследовательской группы оказались довольно интересными (был реализован прототип системы трекинга, способный восстанавливать траектории движения нескольких человек внутри комнаты в течении нескольких минут с помощью видеопотока, фиксированного с нескольких камер). Рассмотрим основные принципы работы данного метода.

Начнём с рассмотрения причин возникновения ошибок в случае перекрытия силуэтов пешеходов. Основной предпосылкой к возникновению неточностей здесь является некорректная работа детекторов при возникновении перекрытия силуэтов отслеживаемых объектов.

Процесс трекинга зачастую можно разделить на две составляющие. Первая – это детектирование отслеживаемых объектов, а вторая – сопоставление результатов детектирования в последовательности временных промежутков (поиск траекторий).

Результаты работы детекторов зачастую изобилуют ошибками (определение элементов фона как искомых объектов, ложное детектирование, и т. п.). Наличие шумов того или иного рода зачастую является причиной некорректной работы поиска траекторий. Здесь следует отметить, что именно ситуации перекрытия силуэтов

становятся основной причиной неверной работы детектирования, так как детекторы обучаются на определённых образцах, соответствующих силуэтам людей, в то время, как при перекрытии возникает силуэт, совершенно не похожий на фигуру пешехода, что и становится причиной ошибки (либо детектор вообще не распознаёт человека, либо распознаёт одного вместо двоих, в любом случае результаты детектирования становятся сильно зашумлёнными).

Рассматриваемый в данном разделе подход предлагает следующие методы борьбы с вышеописанными проблемами. Во-первых, при наличии нескольких образцов видеосъёмки с различных откалиброванных заданным образом камер, возможность получить видеопоток с ракурса, на котором отсутствуют перекрытия силуэтов, увеличивается, этот эффект можно усилить путём сопоставления записей видеопотоков, полученных с различных камер. Таким образом, возникает вопрос о том, как реализовать подобного рода сопоставление. Заметим, что просто сопоставив результаты детектирования в видеопотоках с различных камер (а это возможно, так как камеры изначально откалиброваны согласно заданным параметрам), невозможно полностью исключить вероятность одновременного перекрытия силуэтов отслеживаемых объектов относительно всех ракурсов. Поэтому, перед тем, как рассмотреть механизм сопоставления результатов с различных камер, рассмотрим ещё один важный механизм, который используется в данной системе трекинга. В данном подходе используются некоторые эвристики, такие как радиус возможного перемещения каждого отслеживаемого пешехода в период времени от одного момента детектирования до следующего, также, наибольшее внимание уделяется именно тем результатам детектирования, которые согласуются с результатами детектирования, полученными с остальных камер системы. Если принять во внимание описанные выше идеи, то полученная система будет более устойчива к шумам и позволит находить наиболее вероятные траектории движения объектов, даже если результаты детектирования недостаточно достоверны для всех камер системы.

Вышеописанные механизмы приводят к идее использования элементов теории вероятности для определения достоверности нахождения того или иного объекта трекинга в том или ином месте кадра в тот или иной момент времени. Таким образом, задача трекинга сводится к тому, чтобы определить вышеупомянутые вероятностные значения, объединив их в непересекающиеся цепочки с максимальным весом (теме самым получить искомые траектории).

Далее изложим подробнее суть рассматриваемого метода. В первую очередь, на основе результатов детектирования, полученных с различных камер, рассчитываем

вероятности нахождения объектов в той или иной области рассматриваемой территории (локации) в тот или иной момент времени. Таким образом, получается карта с вероятностями, содержащая все вышеописанные данные (Probabilistic Occupancy Map).

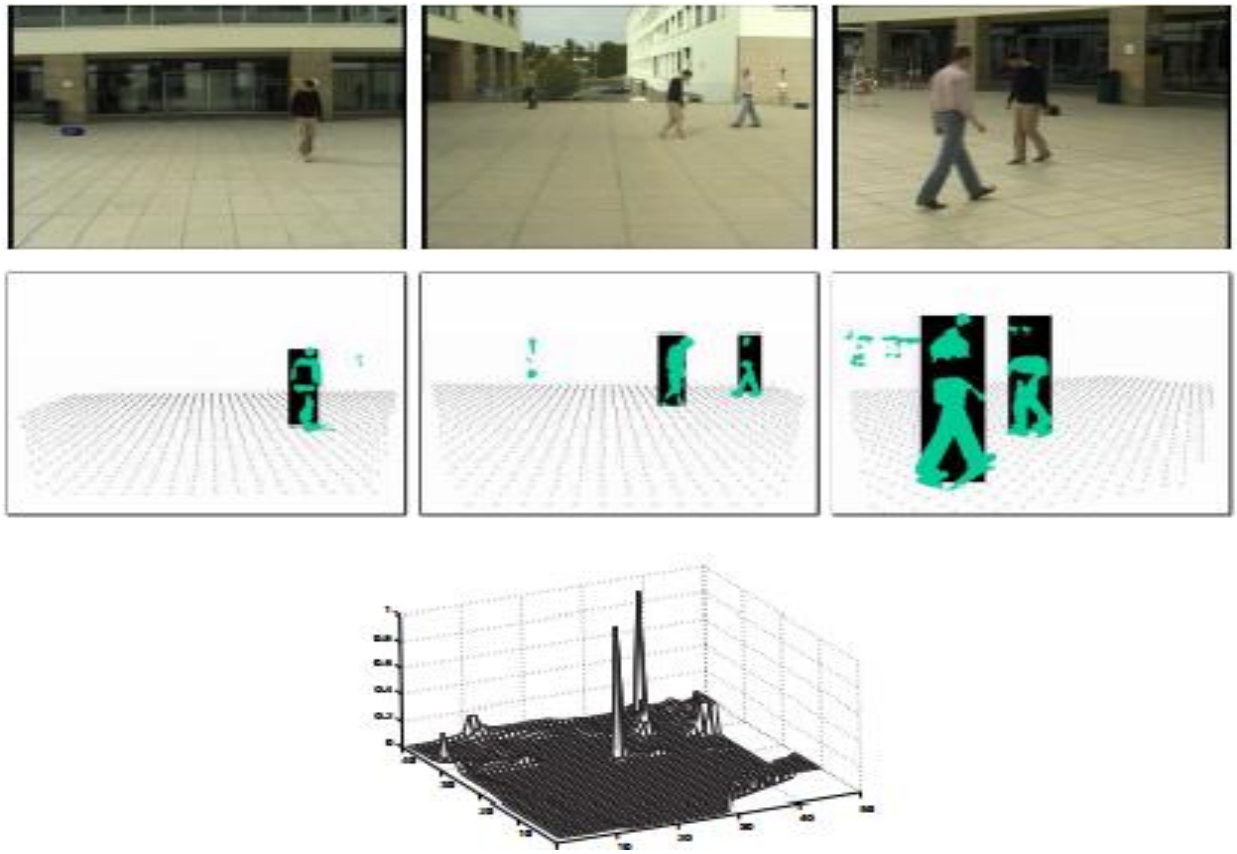

Рис. 2: Probabilistic Occupancy Map. Верхний ряд – кадры видеозаписи, средний ряд – разделение территории перемещения на локации, нижний ряд – график значения вероятности занятости той или иной локации в определённый момент времени (на графике заметно наличие шумов) [1].

После построения вышеупомянутой структуры, осуществляется поиск наиболее вероятных траекторий движения отслеживаемых объектов, который используется и в системе, которая была разработана в рамках настоящей курсовой работы (см. раздел описания решения).

Описание решения

Система, разработанная в рамках данной курсовой работы, использует принципы, описанные в статье [1], но отличается от излагаемого в статье подхода тем, что использует всего одну камеру, что влечёт за собой определённые преимущества и недостатки (см. раздел особенностей данного решения). Итак, наша система трекинга реализована в два этапа. На первом этапе нам необходимо собрать все необходимые данные о вероятности нахождения объектов в различных локациях в тот или иной момент времени, и организовать всё это в некоторую структуру, используя которую, на втором этапе, будет возможно найти оптимальные траектории движения объектов. Рассмотрим этапы нашего решения по порядку.

Probabilistic Occupancy Map

Для описания данного этапа, нам понадобятся следующие обозначения. В первую очередь, нам следует разделить территорию, по которой передвигаются объекты на локации, причём таким образом, чтобы в каждой локации в любой момент времени мог находиться только один объект (данная процедура производится на этапе калибровки системы).

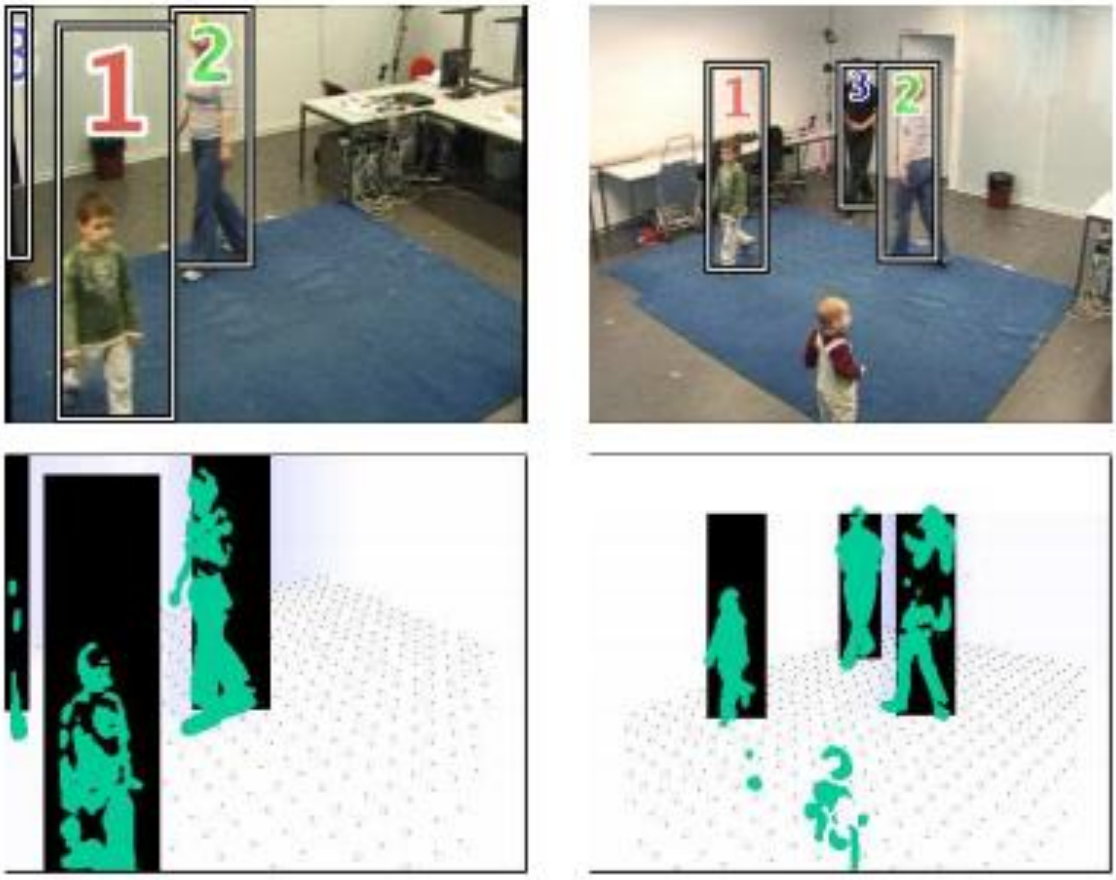

Рис. 3: Пример разбиения территории на локации (нижний ряд, сетка) [1].

После проведения данной процедуры, мы получим набор локаций $L = \{L_1,\ldots,L_n\}$, где n – количество выделенных локаций. Используя выбранный детектор, мы будем собирать данные следующего типа $L_t = \{L_t^1,\ldots,L_t^n\}$, где n – количество выделенных локаций, t – момент времени, L_t^1 – индикатор занятости локации l в момент времени t [1].

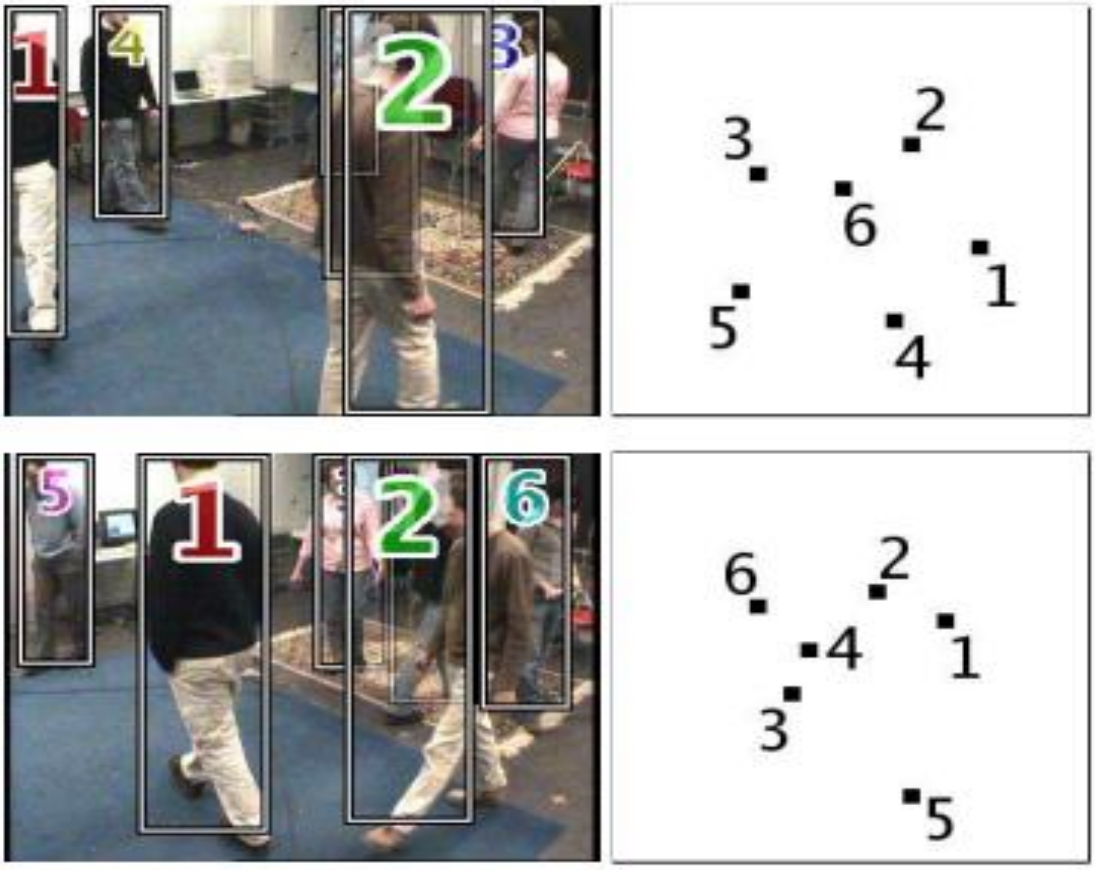

Рис. 4: Пример занятости той или иной локации различными объектами в тот или иной момент времени (справа, вид сверху – схема занятых локаций) [1].

После получения информации о занятости той или иной локации в тот или иной момент времени, мы вычисляем следующего вида величины (которые являются условными вероятностями занятости той или иной локации в той или иной момент времени, то есть в том или ином кадре) [1]:

$$P(L_1^n = l_1^n, \ldots, L_T^n = l_t^n \mid \mathbf{I}_1, \ldots, \mathbf{I}_T) = \frac{P(\mathbf{I}_1, L_1^n = l_1^n, \ldots, \mathbf{I}_T, L_T^n = l_T^n)}{P(\mathbf{I}_1, \ldots, \mathbf{I}_T)}$$

где I_t - номер кадра (~ время детектирования),

$L_t = \{L_t^1, \ldots, L_t^n\}$ – индикатор занятости локаций в тот или иной момент времени, причём:

n – количество выделенных локаций, t – момент времени,

L_t^1 – индикатор занятости локации l в момент времени t.

Рассмотрим подробнее механизм вычисления вышеупомянутой величины. Заметим, что одних данных занятости той или иной локации в тот или иной момент времени $L_t = \{L_t^1,\ldots,L_t^n\}$ не достаточно, так как результаты детектирования, как правило, почти всегда очень зашумлённые, поэтому необходимо ввести следующие эвристики, некоторые из которых уже упоминались в данной статье [1].

Рассмотрим тот факт, что никогда не существует гарантии, что детектор сумеет распознать тот или иной объект, поэтому, каждая локация должна иметь изначально некоторое (небольшое) значение вероятности того, что она будет занята каким-либо объектом в какой-либо промежуток времени, назовём эту величину default_probability_threshold.

Далее, нам следует использовать результаты детектирования $L_t = \{L_t^1,\ldots,L_t^n\}$, и прибавить к значению вероятности default_probability_threshold для каждого момента времени, когда та или иная локация занята значение, которое назовём occupancy_probability_threshold. Следует отметить, что детектор не исключает ошибки ложного распознавания, поэтому полученное на данном этапе значение default_probability_threshold + occupancy_probability_threshold не должно быть очень близко к максимальному значению вероятности – единице.

Следующим шагом мы постараемся снизить влияние шумов и ошибок на значение вероятности. Для этого, нам понадобится параметр motion_amplitude – амплитуда движения – максимальное количество локаций, на которое может переместиться объект за один кадр. Таким образом, для каждых двух результатов занятости локаций с разницей во времени не более, чем в один кадр, вероятность занятости в данный момент времени увеличивается, а при отсутствии в данной временной и пространственной области хотя бы одного результата детектирования объекта, уменьшается на определённую величину, назовём её motion_amplitude_probability_threshold. Таким образом, вероятность занятости той или иной локации увеличивается, при условии присутствия занятых по близости или в недавнем времени локаций, и уменьшается в противном случае.

При вычислении вероятностей, важно отметить тот факт, что вышеупомянутые параметры должны быть подобраны таким образом, чтобы итоговая вероятность посещения той или иной локации в тот или иной момент времени, не оказалась отрицательной, или же большей максимально возможного значения вероятности

(единицы), что ещё раз демонстрирует важность первоначальной настройки и калибровки системы, разработанной и реализованной в рамках данной курсовой работы.

Получив необходимые данные о вероятной занятости той или иной локации в тот или иной момент времени (наборе вероятностей вида) [1]: $P(\mathbf{L}_1, \ldots, \mathbf{L}_T \mid \mathbf{I}_1, \ldots, \mathbf{I}_T)$, мы можем приступить к следующему этапу нашего решения.

Поиск оптимальных траекторий

После выполнения предыдущего этапа нашего решения, мы получили вероятности посещения той или иной локации в каждый момент времени. Далее, рассмотрим способ, с помощью которого можно найти траектории отслеживаемых объектов, используя полученные данные.

Для ясности, опишем общую схему данного этапа решения. Имея на руках вероятности посещения каждой локации в каждый момент времени, нам следует объединить цепочки наиболее вероятных посещений локаций в траектории. Данная задача очевидным образом является задачей линейной оптимизации, и может быть решена такими методами, как метод симплексных таблиц, однако, следует отметить, что одним из наиболее важных характеристик нашей системы трекинга является производительность, принимая во внимание большие объёмы входных данных и большую размерность задачи оптимизации, подобные методы оказываются неприемлемыми. Поэтому, для решения данной задачи, мы введём модель, которая будет представлять собой граф, вершины которого будут являться локациями в тот или иной момент времени, а рёбра будут обозначать возможность перехода из одной локации в другую в следующий момент времени, таким образом, назначив рёбрам и вершинам полученного графа веса, в соответствии с данными, полученными на предыдущем этапе решения, мы получим взвешенный, ориентированный, ациклический граф (DAG). Далее, при обходе графа, нам необходимо найти множество непересекающихся траекторий с максимальным (~ минимальным) весом, полученный набор траекторий и будет искомым нами приближённым решением. Теперь формализуем всё вышесказанное и рассмотрим описанную выше модель подробнее.

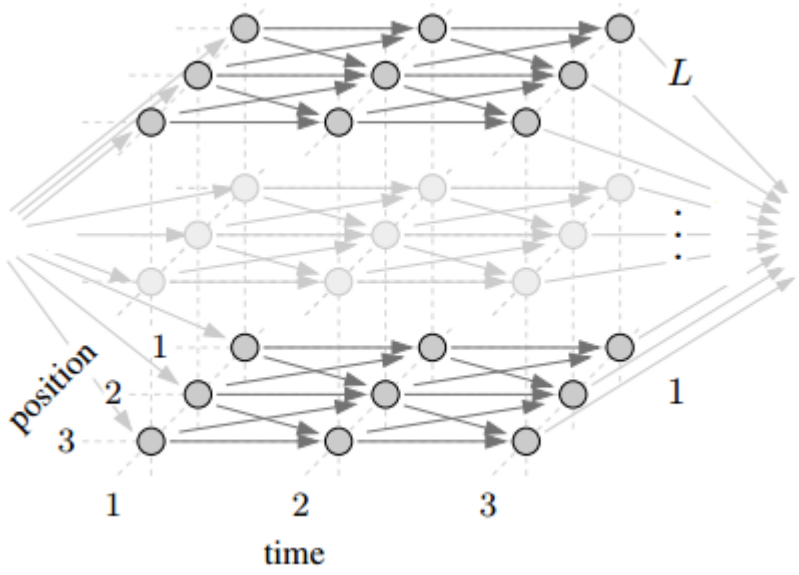

Рис. 5: Схема модели для поиска оптимальных траекторий [2].

Введём обозначения, необходимые для описания данного этапа решения. Пусть у нас имеется K локаций и T кадров. Пусть $\mathcal{N}(k) \subset \{1, \ldots, K\}$ - номера локаций, соседние по отношению к локации k. Таким образом, мы можем представить карту локаций в виде некоторой сети (графа), где достижимые локации соединены рёбрами. Но следует помнить о том, что нам необходимо учитывать фактор течения времени, поэтому, наша структура будет представлять собой граф размерности T x K, вершины и рёбра будут иметь описанные выше значения. Итак, продолжим вводить обозначения. $e_{i,j}(t)$ - ребро, которое соединяет локацию i с локацией j в момент времени t, при условии, что локация j достижима из локации i за временной интервал равный временному промежутку между соответствующими соседними кадрами. $m_k(t)$ - индикатор занятости вершины k в момент времени t. $f_{i,j}(t)$ - индикатор перехода по ребру, ведущему из локации i в локацию j в момент времени t.

Теперь мы можем ввести необходимые для корректности модели условия. Первым условием будет аналог принципа сохранения потока (1). Каждый пешеход переходит из кадра в кадр из локации, в локацию, причём он не может зависнуть во времени [2]:

$$\forall t, j, i \sum_{i:j \in \mathcal{N}(i)} f_{i,j}(t) = m_j(t) = \sum_{k \in \mathcal{N}(j)} f_{j,k}(t+1)$$

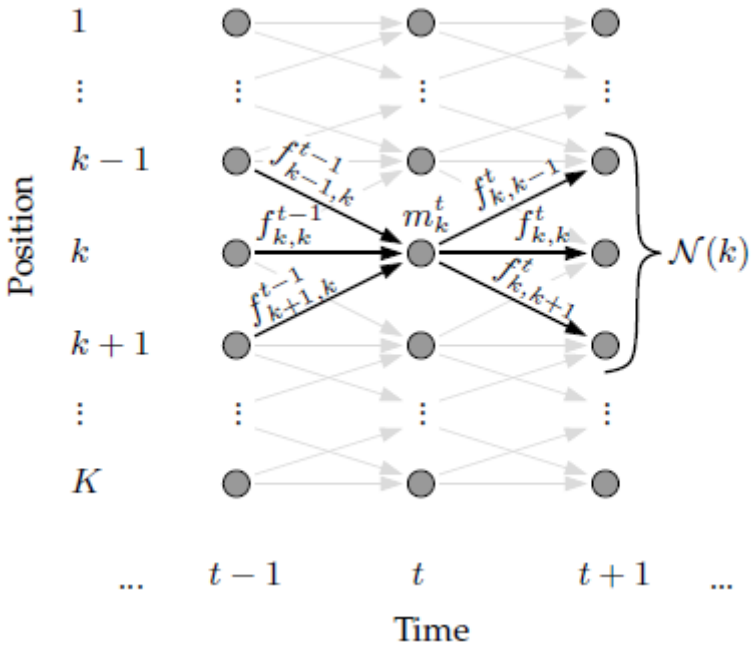

Рис. 6: Иллюстрация к свойству (1) [2].

Следующее условие (2) формализует возможность посещения одним объектом только одной локации в определённый дискретный момент времени [2]:

$$\forall t, k, \sum_{j \in \mathcal{N}(k)} f_{k,j}(t) \leq 1$$

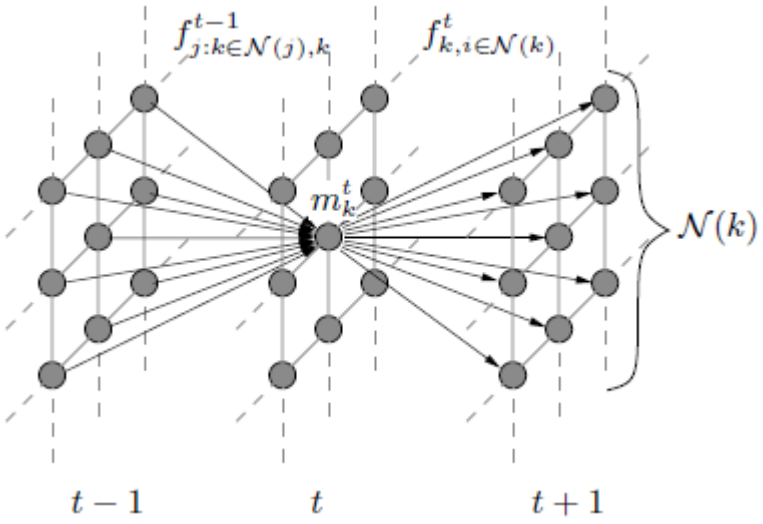

Рис. 7: Иллюстрация к свойству (2), (3) [2].

Аналогичным образом, следует регламентировать возможность перемещения во времени (3) (отслеживаемый объект не может перемещаться в прошлое) [2]:

$$\forall k, j, t,\ \ f_{k,j}(t) \geq 0$$

Исходя из природы нашей модели, мы можем задать возможность появления в кадре. Для описание данного условия (4), мы введём дополнительные вершины: Vsource и Vsink (начальная и конечная локации, соответственно, или просто: исток и сток в терминах потоков). Итак, каждый объект может появляться только из вершины Vsource, или в момент времени t = 0, а исчезать из кадра либо в момент времени t = T, либо из вершины Vsink. При таких условиях, правдоподобность модели достигается теми фактами, что любая вершина в графе соединена как с источником, так и со стоком, точнее (так как граф ориентированный) источник соединён с каждой вершиной, и каждая вершина, в свою очередь, соединена со стоком напрямую. При этом, следующее условие (4), которое мы накладываем на нашу модель, заключается в том, чтобы количество объектов, в ходящих в систему было равно количеству объектов, покинувших систему [2]:

$$\sum_{j \in \mathcal{N}(v_{\text{source}})} f_{v_{\text{source}},j} = \sum_{k:v_{\text{sink}} \in \mathcal{N}(k)} f_{k,v_{\text{sink}}}$$

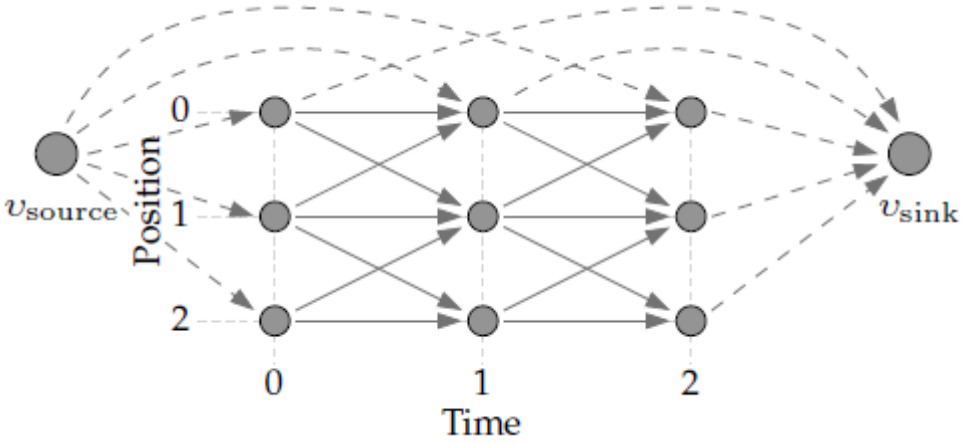

Рис. 8: Иллюстрация к свойству (4) [2].

Теперь, когда требования к нашей модели формализованы, можно заметить, что с точки зрения потоков, наша система является корректной, следовательно, может быть применена к решению поставленной задачи.

Определим вид искомых траекторий в терминах выше описанной модели. Обозначим результаты детектирования следующим образом (в виде вероятности занятости той или иной локации в тот или иной период времени) [2]:

$$\rho_i^t = P(M_i^t = 1 \mid \mathbf{I}^t)$$

Где I^t - событие наблюдения кадра номер t, M_i^t – событие занятости локации i в момент времени t.

Введём ещё несколько обозначений. Пусть m – одно из допустимых (относительно выше перечисленных потоковых условий) распределений вероятностей в POM. Множество всех распределений обозначим, как $\mathfrak{F}$. Текущее распределение вероятностей в POM мы обозначим как M. Тогда искомые траектории будут иметь вид [2]:

$$\mathbf{m}^* = \arg\max_{\mathbf{m}\in\mathfrak{F}} P(\mathbf{M}=\mathbf{m}\,|\,\mathbf{I})$$

То есть мы хотим выбрать такой набор локаций для составления траекторий, чтобы максимизировать вероятность посещения каждой локации вдоль каждой траектории в каждый момент времени.

Преобразуем набор оптимальных траекторий следующим образом [2]:

$$\begin{aligned}
\mathbf{m}^* &= \arg\max_{\mathbf{m}\in\mathfrak{F}} \log\prod_{t,i} P(M_i^t = m_i^t\,|\,\mathbf{I}^t) \\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} \log P(M_i^t = m_i^t\,|\,\mathbf{I}^t) \\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} (1-m_i^t)\log P(M_i^t = 0\,|\,\mathbf{I}^t) \\
&\qquad\qquad + m_i^t \log P(M_i^t = 1\,|\,\mathbf{I}^t) \\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} m_i^t \log\frac{P(M_i^t = 1\,|\,\mathbf{I}^t)}{P(M_i^t = 0\,|\,\mathbf{I}^t)} \\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} \left(\log\frac{\rho_i^t}{1-\rho_i^t}\right) m_i^t\,,
\end{aligned}$$

где m_i^t - индикатор занятости траектории i в момент времени t, M_i^t - выбираемая в траекторию локация в момент времени t.

Таким образом, поиск оптимальных траекторий трансформируется в задачу максимизации [2]:

$$\begin{aligned}
&\sum_{t,i} \log\left(\frac{\rho_i^t}{1-\rho_i^t}\right) \sum_{j\in\mathcal{N}(i)} f_{i,j}^t \\
&\forall t,i,j,\quad f_{i,j}^t \geq 0 \\
&\forall t,i,\quad \sum_{j\in\mathcal{N}(i)} f_{i,j}^t \leq 1 \\
&\forall t,i,\quad \sum_{j\in\mathcal{N}(i)} f_{i,j}^t - \sum_{k:i\in\mathcal{N}(k)} f_{k,i}^{t-1} \leq 0 \\
&\sum_{j\in\mathcal{N}(v_{\text{source}})} f_{v_{\text{source}},j} - \sum_{k:v_{\text{sink}}\in\mathcal{N}(k)} f_{k,v_{\text{sink}}} \leq 0
\end{aligned}$$

Рассмотрим подробнее механизм взвешивания рёбер нашего графа. Обозначим как $e^{t}_{i\,j}$ ребро, обозначающее возможность перехода из локации i в локацию j в момент времени t. Тогда вес, который мы назначим данному ребру будет равен [2]:

$$c(e^t_{i,j}) = -\log\left(\frac{\rho^t_i}{1-\rho^t_i}\right)$$

Соответственно, веса искомых траекторий будут следующими [2]:

$$\mathbf{f}^* = \arg\min_{\mathbf{f}\in\mathfrak{H}} \sum_{t,i} c(e^t_{i,j}) \sum_{j\in\mathcal{N}(i)} f^t_{i,j}$$

Таким образом, для набора траекторий $P_l = \{p_1^*,..,p_l^*\}$ суммарный вес будет следующим [2]:

$$\mathrm{cost}(P_l) = \sum_{i=1}^{l} \mathrm{cost}(p^*_i)$$

где:

$$\mathrm{cost}(p^*_l) = \sum_{e^t_{i,j}\in p^*_l} c(e^t_{i,j})$$

После того, как мы построили описанную выше модель для решения, рассмотрим некоторые подробности реализации данного метода.

Детали реализации

В данном разделе описаны основные технологии и алгоритмы, с помощью которых были созданы программные компоненты, решающие описанные в предыдущих разделах задачи.

Технологии

Для наиболее эффективной реализации программных компонент, было принято решение использовать библиотеку OpenCV, Microsoft Visual Studio 2010, C++. OpenCV – библиотека алгоритмов компьютерного зрения с открытым исходным кодом, содержащая реализацию различных алгоритмов, применимых для детектирования пешеходов, лиц людей, трекинга движущихся объектов, обработки, ввода, вывода различных изображений и видеопотоков, а также, различные структуры данных, упрощающие выполнение вышеупомянутых операций.

Особенности реализации

Разработанная в рамках данной курсовой работы система трекинга состоит из двух компонент. Первая составляющая реализует функциональность построения POM по результатам детектирования. Следует отметить, что видеопоток берётся только с одной камеры, в то время, как подход, предложенный в статье [1], использует минимум две камеры. Данное условие влечёт за собой потребность в более точной калибровке камеры, так как из-за отсутствия избыточности, сбой при калибровке системы, станет причиной её некорректной работы в целом. Ещё одним очевидным следствием использования одной камеры, является более низкая стоимость необходимой аппаратуры. Вдобавок к вышесказанному, можно отметить, что простота и общее время установки и калибровки оборудования также снижается. Несмотря на очевидные преимущества использования одной камеры, такой вариант системы имеет ряд недостатков. Во-первых, при использовании одной камеры, наша система охватывает гораздо меньшую территорию наблюдения. Следующей проблемой является сокращение числа возможных ракурсов и мест возможной установки камер. Помимо вышеперечисленных и некоторых других недостатков, одной из самых главных проблем использования одной камеры для данного подхода, является значительное падение эффективности работы системы определения

траекторий при увеличении площади перекрытия объектов (подробнее см. раздел с результатами данной работы). Причиной этому является увеличение времени, в течении которого отсутствуют корректные результаты детектирования. Конечно, с такой проблемой возможно бороться с помощью выбора наиболее точного детектора, но результат, тем не менее будет не сравним по качеству с результатом работы аналогичной системы, использующей несколько камер.

Помимо инструмента, реализующего построение POM для одной камеры, в рамках данной курсовой работы, реализован инструмент для поиска траекторий по данным, полученным на первом этапе решения. Особенностью реализации данного компонента нашей системы, является тот факт, что поиск траекторий осуществляется с помощью алгоритма k кратчайших путей (см. приложение 2), отметим также, что данная система, позволяет использовать для поиска кратчайших путей такие алгоритмы, как алгоритм Дейкстры, Левита, и т. д. (см. приложение 3). Конечно, проблему поиска оптимальных траекторий можно было бы решить с помощью алгоритма поиска минимального потока, но тогда был бы проигрыш по производительности системы, так как в данном случае, поиск траекторий работал бы за $O(kn^2m\log n)$, где k – число наблюдаемых объектов, m – число рёбер, а n – число вершин построенного графа, в нашем же случае, удалось добиться сложности $O(k(m+n\cdot\log n))$, что значительно повышает производительность разработанной системы.

Заключение

Следующим шагом после реализации нашей системы, является оценка эффективности её работы. Результаты произведённых оценок отражены в следующем разделе.

Сравнение эффективности различных версий системы

Для оценки эффективности и работоспособности различных версий системы, мы используем результаты детектирования, полученные из PETS 2009 [3], а также из Caltech Pedestrian Detection Benchmark [4]. Получив необходимые тестовые наборы данных вместе с результатами детектирования, системами, основанными на самых распространённых методах детектирования, построенных на таких подходах, как HOG, LBP, VJ (см. приложение 4), мы проанализировали и сравнили работу нашей (системы с одной камерой), и многокамерной системой [1]. Результаты тестов описаны в таблице 1:

Способ детектирования	Процент перекрытия силуэтов, при котором система определения траекторий прекращает корректную работу	
	Система, использующая 4 камеры	Система, использующая 1 камеру (реализованная в рамках данной работы)
HOG	~ 43 %	~ 13 %
LBP	~ 37 %	~ 11 %
VJ	~ 29 %	~ 8 %

Таблица. 1: Оценка работоспособности систем определения траекторий в условиях перекрытия.

Результаты работы

Подведём итоги. В результате выполнения данной работы был прочитан ряд статей про алгоритмы компьютерного зрения и методы детектирования и трекинга. Например, методы, основанные на таких подходах, как HOG, LBP, VJ (см. приложение 4). Также был изучен ряд проблем, свойственных современным системам трекинга и детектирования, в результате чего, в силу своей актуальности, для дальнейшего изучения, была выбрана проблема поиска траекторий в условиях перекрытия. В результате работы

над выбранной проблемой, была реализована система поиска траекторий в условиях перекрытия объектов, использующая образцы видеозаписи с одной камеры для детектирования объектов слежения, причём способная использовать различные детекторы. Работа данной системы при использовании различных детекторов, была проанализирована и сравнена с работой аналогичной системы, использующей несколько камер (см. таблицу 1).

Дальнейшее развитие

В дальнейшем возможно продолжить данную разработку в следующий направлениях:

- Оптимизация механизма нахождения вероятностей посещения той или иной локации в тот или иной момент времени, в плане производительности, исправления ошибок детектора и зашумлённости входного потока данных.
- Поиск оптимальных методов детектирования, наиболее подходящих для данной системы трекинга.
- Поиск наиболее оптимального метода поиска траекторий в рамках существующей модели решения.
- Поиск модели решения, позволяющей наилучшим образом смоделировать ситуации перекрытия отслеживаемых силуэтов.

Список литературы

[1] Jerome Berclaz, Francois Fleuret, Pascal Fua: “Multi-Camera People Tracking with a Probabilistic Occupancy Map”

[2] Jerome Berclaz, Francois Fleuret, Engin Turetken, Pascal Fua: “Multiple Object Tracking using K-Shortest Paths Optimisation”

[3] PETS 2009: http://pets2009.net/

[4] Caltech Pedestrian Detection Benchmark: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/

[5] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona: “Pedestrian Detection: An Evaluation of the State of the Art”

[6] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona: “Pedestrian Detection: A Benchmark”

[7] Zhengqiang Jiang, Du Q. Huynh, William Moran, Subhash Challa and Nick Spadaccini: “Multiple Pedestrian Tracking using Colour and Motion Models”

[8] Luis E. Navarro-Serment, Christoph Mertz, and Martial Hebert: “Pedestrian Detection and Tracking Using Three-Dimensional LADAR Data”

[9] Gary Bradski and Adrian Kaehler: “Learning OpenCV”

[10] Bo Wu and Ram Nevatia: “Detection and Tracking of Multiple, Partially Occluded Humans by Bayesian Combination of Edgelet based Part Detectors”

[11] S. Munder, C. Schorr, and D. M. Gavrila: “Pedestrian Detection and Tracking Using a Mixture of View-Based Shape-Texture Models”

[12] Nils T Siebel: “Design And Implementation of People Tracking Algorithms for Visual Surveillance Applications”

[13] Navneet Dalal, Bill Triggs: “Object Detection using Histograms of Oriented Gradients”

[14] Navneet Dalal and Bill Triggs: “Histograms of Oriented Gradients for Human Detection”

[15] Paul Viola and Michael Jones: “Robust Real-time Object Detection”

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: “Introduction to Algorithms”

[17] Bruce D. Lucas and Takeo Kanade: “An Iterative Image Registration Technique with an Application to Stereo Vision”

[18] C. Papageorgiou, M. Oren and T. Poggio: “A General Framework for Object Detection”

[19] А.Н. Алфимцев, Н.А. Демин: “Захват и отслеживание удаленных объектов в видеопотоке”

[20] А. С. Тарасиков: “Программа обнаружения лица на изображении на основе характеристических признаков”

Приложения

Приложение 1 – методы трекинга

В данном приложении мы перечислим основные методы трекинга объектов [18]:

- метод шаблонов движений (Motion Templates) — основан на поиске границ объектов в каждом кадре видеопотока. Смещение границы на новом кадре относительно предыдущего задает вектор движения объекта. Данный алгоритм наиболее эффективен при движении крупных объектов и часто используется для распознавания динамических жестов в человеко-машинных интерфейсах;
- метод сдвига среднего (Mean-Shift) — основан на математической модели, которая заключается в том, что вычисляется локальный экстремум плотности распределения набора характерных точек, т. е. алгоритм отслеживает смещение центра масс точек, определяющих объект слежения, получая на выходе вектор движения объекта.

Высокая эффективность достигается при ярко выраженном цветовом различии объекта и фона;

- метод непрерывно адаптирующегося сдвига (CamShift) — основан на алгоритме сдвига среднего, но отличается тем, что автоматически подстраивает границы и размер окна, в пределах которого расположены характерные точки [3]. Таким образом производится более точное отслеживание объекта, изменяющегося в размерах;
- метод Лукаса – Канаде (Lucas – Kanade) — основан на дифференциальном вычислении оптического потока с помощью анализа пикселей (предполагается, что оптический поток одинаков для пикселей, лежащих в окрестности центра окна слежения), при этом смещение пикселей между соседними кадрами должно быть невелико. Данный алгоритм более двадцати лет активно используется в приложениях компьютерного зрения и уже доказал свою высокую эффективность для широкого круга задач.

Приложение 2 – K-Shortest Paths

В данном приложении мы рассмотрим алгоритм нахождения k кратчайших непересекающихся (в вершинах) траекторий.

Пусть у нас есть ориентированный граф G = (V, E), где G – множество вершин, E – множество рёбер, соответственно. В качестве графа, мы будем рассматривать модель описанную в данной работе, поэтому мы будем искать пути между вершинами графа Vsource и Vsink. Введём ещё некоторые обозначения. l = 1..k – номер некоторого из путей, которые должны быть найдены при выполнении данного алгоритма. P(l) – набор кратчайших путей , найденных на l-й итерации. Отметим, что вес ребра задаёт его направление положительный вес (+) имеет ребро, направленное от Vsource к Vsink, отрицательный вес (-) имеет ребро, направленное от Vsink к Vsource, соответственно. Переход от P(l) к P(l + 1) происходит с помощью рассмотрения пути, обладающего некоторыми свойствами, по отношению к P(l), назовём его p. Рассмотрим вышеупомянутые свойства: 1) p может иметь общее ребро с P(l), только если это ребро отрицательного веса (-), 2) p может иметь общую вершину с P(l), только если эта вершина является смежной с ребром отрицательного веса (-). Сам переход от P(l) к P(l + 1) происходит с помощью добавления к P(l) рёбер из p, имеющих положительные веса (+), и удаления из P(l) рёбер из p, имеющих отрицательные веса (-).

Приведём пример перехода от P(l) к P(l + 1). Пусть P1 = {(Vsource, Vi, Vj, Vsink)}, p = (Vsource, Vm, Vj, Vi, Vn, Vsink), причём знаки весов у p следующие: (+, +, -, +, +), тогда P2 = {(Vsource, Vm, Vj, Vsink), (Vsource, Vi, Vn, Vsink)}.

Помимо преобразования пути p, множества путей P(l), на шаге l, нам так же необходимо преобразовывать сам граф (обозначим полученный граф на l-м шаге, как Gl = (Vl, El). Опишем подробнее преобразование графа. Первая стадия преобразования заключается в замене всех вершин Pl (кроме Vsink и Vsource) на две вершины, причём в одну вершину рёбра только входят, а из другой только выходят, соединим эти вершины вспомогательным ребром (направленном от вершины, в которую рёбра входят). Установим вес вспомогательного ребра равным нулю. На второй стадии мы поменяем направление и знак веса всех рёбер из Pl (включая вспомогательные рёбра) на противоположный.

Возможно ввести ещё одно весовое преобразование к Gl, который, возможно содержит рёбра с негативным весом. С помощью данного преобразования мы получим граф Glc, который будет эквивалентен графу Gl, но не будет содержать рёбер с отрицательным весом. Преобразование такого рода, очевидным образом понизит сложность поиска кратчайших путей на каждом шаге нашего алгоритма. Опишем

подробнее данное преобразование. Оно заключается в следующем: мы изменяем вес каждого ребра путём добавления веса кратчайшего пути от Vsource до его начала, и вычитания из полученной суммы веса кратчайшего пути от Vsource до его конца. Следует отметить, что в данном случае, сложность нашего алгоритма будет O(k*(m + n*log(n))) где k – число отслеживаемых объектов, m – число рёбер, n – число вершин исходного графа.

Для наглядности, ниже приведён псевдокод вышеописанного алгоритма:

```
p(1) <- кратчайший путь от Vsource к Vsink  в графе G
P(1) <- {p1}
for l <- 1 to lmax do
        if l <> 1then
                if cost(P(1)) >= cost(P(l - 1)) then
                        return P(l - 1) = {p(1),…,p(l - 1)}
                end
        end
Gl <- описанная выше двухэтапная процедура преобразования G
Glc <- описанная выше процедура избавления от рёбер отрицательного веса
p(l + 1) <- кратчайший путь от Vsource к Vsink в графе Glc
P(l + 1) <- P(l) ++ p(l + 1)
```

Приложение 3 – алгоритмы поиска кратчайших путей в графе

В данном приложении мы рассмотрим два алгоритма поиска кратчайших путей в графе, которые целесообразно использовать в нашей системе.

Первым таким алгоритмом будет **алгоритм Дейкстры**. Приведём ниже псевдокод данного алгоритма и оценку его сложности. Для начала, введём следующие обозначения:

V — множество вершин графа

E — множество ребер графа

w[ij] — вес (длина) ребра ij

a — вершина, расстояния от которой ищутся

U — множество посещенных вершин

d[u] — по окончании работы алгоритма равно длине кратчайшего пути из a до вершины u

p[u] — по окончании работы алгоритма содержит кратчайший путь из a в u

Теперь рассмотрим псевдокод для данного алгоритма:

Присвоим $d[a] \leftarrow 0,\ p[a] \leftarrow a$

Для всех $u \in V$ отличных от a присвоим $d[u] \leftarrow \infty$

Пока $\exists v \notin U$

Пусть $v \notin U$ – вершина с минимальным $d[v]$

занесём v в U

Для всех $u \notin U$ таких, что $vu \in E$

если $d[u] > d[v] + w[vu]$ то изменим $d[u] \leftarrow d[v] + w[vu]$

изменим $p[u] \leftarrow p[v], u$

Сложность алгоритма Дейкстры зависит от способа нахождения вершины *v*, а также способа хранения множества непосещенных вершин и способа обновления меток. Обозначим через *n* количество вершин, а через *m* — количество ребер в графе *G*. Для разреженных графов (то есть таких, для которых m много меньше n^2) непосещенные вершины можно хранить в двоичной куче, а в качестве ключа использовать значения d[i], тогда время удаления вершины из непосещённых вершин станет log n, при том, что время модификации d[i] возрастет до log n. Так как цикл выполняется порядка n раз, а количество релаксаций (смен меток) не больше m, скорость работы такой реализации O(n * log(n) + m*log(n)).

Следующим алгоритмом, который мы рассмотрим, будет **алгоритм Левита.** Опишем подробнее данный алгоритм. Пусть массив D[1..N] будет содержать текущие кратчайшие

длины путей. Изначально массив D заполнен значениями "бесконечность", кроме D[s] = 0. По окончании работы алгоритма этот массив будет содержать окончательные кратчайшие расстояния. Пусть массив P[1..N] содержит текущих предков. Так же как и массив D, массив P изменяется постепенно по ходу алгоритма и к концу его принимает окончательные значения. Изначально все вершины помещаются в множество M2, кроме вершины V0, которая помещается в множество M1. На каждом шаге алгоритма мы берём вершину из множества M1 (достаём верхний элемент из очереди). Пусть V - это выбранная вершина. Переводим эту вершину во множество M0. Затем просматриваем все рёбра, выходящие из этой вершины. Пусть T - это второй конец текущего ребра (т.е. не равный V), а L - это длина текущего ребра.

- Если T принадлежит M2, то T переносим во множество M1 в конец очереди. DT полагаем равным DV + L.
- Если T принадлежит M1, то пытаемся улучшить значение DT: DT = min (DT, DV + L). Сама вершина T никак не передвигается в очереди.
- Если T принадлежит M0, и если DT можно улучшить (DT > DV + L), то улучшаем DT, а вершину T возвращаем в множество M1, помещая её в начало очереди.

Разумеется, при каждом обновлении массива D следует обновлять и значение в массиве P. Время работы алгоритма на случайных тестах O(M*log(N)), где N – число вершин, M – число рёбер.

В сравнении с методом Дейкстры метод Левита проигрывает на том, что некоторые вершины приходится обрабатывать повторно, а выигрывает на более простых алгоритмах включения и исключения вершин из множества M1. Эксперименты показывают, что для графов с «геометрическим» происхождением, т.е. для графов, построенных на основе транспортных сетей и реальных расстояний, метод Левита оказывается наиболее быстрым. Он выигрывает и по размеру программы. Метод Левита обладает еще и тем преимуществом перед методом Дейкстры, что он применим в случае отрицательных длин дуг.

Приложение 4 – методы детектирования объектов

В данном приложении мы сделаем краткий обзор некоторых способов детектирования.

Первым из таких подходов является метод на основе **HOG (Histogram of Oriented Gradients) дескриптора**. Основной идеей алгоритма является допущение, что внешний вид и форма объекта на участке изображения могут быть описаны распределением градиентов интенсивности или направлением краев. Реализация этих дескрипторов может быть произведена путем разделения изображения на маленькие связные области, именуемые ячейками, и расчетом для каждой ячейки гистограммы направлений градиентов или направлений краев для пикселов, находящихся внутри ячейки. Комбинация этих гистограмм и является дескриптором. Для увеличения точности локальные гистограммы подвергаются нормализации по контрасту. С этой целью вычисляется мера интенсивности на большем фрагменте изображения, который называется блоком, и полученное значение используется для нормализации. Нормализованные дескрипторы обладают лучшей инвариантностью по отношению к освещению.

Дескриптор HOG имеет несколько преимуществ над другими дескрипторами. Поскольку HOG работает локально, метод поддерживает инвариантность геометрических и фотометрических преобразований, за исключением ориентации объекта. Подобные изменения появятся только в больших фрагментах изображения. Более того, как обнаружили Далал и Триггс, грубое разбиение пространства, точное вычисление направлений и сильная локальная фотометрическая нормализация позволяют игнорировать движения пешеходов, если они поддерживают вертикальное положение тела. Дескриптор HOG, таким образом, является хорошим средством нахождения людей на изображениях.

Следующим методом, который мы рассмотрим в данном разделе, будет метод **LBP (Local Binary Patterns)**. Данный алгоритм для каждого проверяемого пикселя изображения в 8-связной области сравнивает яркость пикселя из области с проверяемым пикселем. В случае, если первое значение больше - пикселю области присваивается значение «1», в противном - «0». 8-битное число, получаемое обходом пикселей по часовой стрелке, называется Binary Pattern, и используется для сравнения со значениями в классификаторе. Данный подход можно использовать для обнаружения произвольных объектов. Поскольку алгоритмы такого рода оперируют исключительно яркостью изображения, возможны ложные срабатывания на объектах, схожих по уровню яркости с образцами из тестовой выборки.

Последним подходом, который используется в данной работе, является **метод Виолы-Джонса (Viola–Jones object detection)**, использующий признаки Хаара - сумму яркостей пикселей. Признак Хаара состоит из смежных прямоугольных областей. Эти области позиционируются на изображении, далее происходит суммирование интенсивности пикселей в областях, затем между суммами вычисляется разность. Значение полученной разности и является значением определенного признака, определенного размера, определенным образом расположенного на изображении. На этапе обнаружения в методе Виолы-Джонса используется окно, определенного размера, которое движется по изображению. Признак Хаара рассчитывается для каждой области изображения, над которой проходит окно. Наличие или отсутствие предмета в окне определяется разницей между значением признака и обучаемым порогом. Поскольку признаки Хаара мало подходят для обучения или классификации, для описания объекта с достаточной точностью необходимо большее число признаков. Преимущество использования признаков Хаара является наибольшая, по сравнению с остальными признаками, скорость. При использовании интегрального представления изображения, признаки Хаара могут вычисляться за постоянное время. Недостатки данного метода сходны с недостатками метода LBP.

Приложение 5 – документация реализованной системы

TrajectoryTool/probabilityMapManager – модуль, содержащий описание структуры данных (графа) для поиска оптимальных (наиболее вероятных) траекторий движения отслеживаемых объектов

class Trajectory – класс для хранения искомых траекторий

int length – длина траектории (количество посещённых локаций)

int* areas – список посещённых локаций

float resultWeight – вес траектории (сумма весов рёбер, входящих в траекторию)

Trajectory(int length) – конструктор класса хранения траектории (int length – длина создаваемой траектории)

~Trajectory() – деструктор класса хранения траектории

class Vertex – класс, описывающий локацию в тот или иной момент времени (вершину графа)

float weight – вес вершины (используется для хранения веса кратчайшего пути до данной вершины)

Edge* incomingEdgeList – список входящих рёбер

Edge* outgoingEdgeList – список исходящих рёбер

void addEdgeToOutgoingList(Edge* e) – добавление ребра e в список исходящих рёбер

void deleteEdgeFromOutgoingList(Edge* e) – удаление ребра e из списка исходящих рёбер

class Edge – класс, описывающий ребро, обозначающее возможность перехода меду локациями с течением времени

float weight – вес ребра (зависит от вероятности перехода между определёнными локациями в определённый момент времени)

Vertex* from – начало ребра

Vertex* to – конец ребра

Edge* nextEdge – следующее ребро из списка рёбер

Edge* previousEdge – предыдущее ребро из списка рёбер

void invert() – инвертирование ребра

class MapGraph – класс, описывающий структуру графа, с помощью которого осуществляется поиск оптимальных траекторий

void setCorrectWeights() – устанавливает веса рёбер с учётом избавления от шумов и ошибок

void visitAndUpdate() – производит обновление весов вершин, необходимое для получения оптимальных траекторий

void findDisjointTrajectories() – производит поиск непересекающихся траекторий

void findOptimalTrajectories(float* weights) – производит поиск оптимальных траекторий, согласно заданным весам weights

Trajectory** trajectories – найденные оптимальные траектории

MapGraph(int areasQ, int linksQ, int* from, int* to, int f, int t) – конструктор, areasQ – количество локаций, linksQ – количество рёбер, * from – указатель на исток, * to – указатель на сток, f – номер истока, t – номер стока

~MapGraph() – деструктор

TrajectoryTool/trajectoryToolManager – модуль, содержащий методы для поиска траекторий

class TrajectorySearcher – класс, ответственный за поиск наиболее вероятных траекторий

MapGraph* probabilityMap – граф, с помощью которого производится поиск наиболее вероятных траекторий

float* edgeWeights – веса рёбер графа

int previousArea(int timeSample, int areaNumber) – возвращает предыдущую локацию (int timeSample – текущий кадр, int areaNumber – текущая локация)

int nextArea(int timeSample, int areaNumber) – возвращает следующую локацию (int timeSample – текущий кадр, int areaNumber – текущая локация)

int** possilbleMovements – рёбра графа

int** entranceMarks – метки появления объектов в кадре

int** exitMarks – метки удаления объектов из кадра

float** detectionWeight – веса детектирования

TrajectorySearcher() – конструктор

~TrajectorySearcher() – деструктор

void loadRawData(istream *in) – загрузка данных детектирования из входного потока in

void buildProbabilityGraph() – построение графа для поиска траекторий

void findTrajectories() – поиск оптимальных траекторий

void ouputTrajectories(ostream* out) – вывод искомых траекторий в выходной поток out

pomCreator – модуль для создания вероятностной карты и расчёта вероятностей занятости той или иной локации в тот или иной момент времени по результатам детектирования

void setWidth(int in) – установка ширины карты равной in

void setHeight(int in) – установка высоты карты равной in

void setLocationNumber(int in) – установка количества локаций равному in

void setTimeSampleNumber(int in) – установка числа кадров равному in

void setPresenceThresholdValue(int in) – установка порогового значения вероятности присутствия объекта в некоторой локации, равному in

void setAbsenceThresholdValue(int in) – установка порогового значения вероятности отсутствия объекта в некоторой локации, равному in

void logMapPositions() – вывод взаимного расположения локаций на территории движения объектов

int getLocation(int x, int y) – получение номера локации по координатам точки в кадре, (x, y) – координаты точки в кадре

void updatePom() – заполнение карты с вероятностями по результатам детектирования

Страница проекта: https://code.google.com/p/trajectories-searching-tool/source/browse/trunk/