Санкт-Петербургский государственный университет

Кафедра компьютерных технологий и систем

Голокоз Александр Юрьевич

Выпускная квалификационная работа бакалавра

Детектирование движущихся объектов на видеоизображении с одиночной камеры

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Погожев С. В.

Санкт-Петербург

2016

Содержание

Введение

Детектирование движущихся объектов стало важной частью решения множества прикладных задач в различных сферах. Примером может служить задача слежения за техническим состоянием и функционированием объектов, наблюдением за потоком движения транспорта. Беспилотные летательные аппараты вместе с методами детектирования движущихся объектов могут широко применять в военных целях. Причины очевидны – получение информации о противнике без риска человеческих потерь. Существует ещё большое количество задач и сфер применений летательных аппаратов и алгоритмов детектирования объектов.

В данной работе рассматривается метод определения движущихся объектов и направления их движения на видеоизображении с подвижной камеры. Существует несколько подходов для детектирования объектов: методы, исключающие фон, методы наблюдения за особыми точками, методы формирования фона, модели с автоматическим перемещением. Алгоритмы, использующие стационарные камеры для съёмки, используют подход отделения динамических объектов на кадре от фона кадра. На основе статических точек, не меняющих своего положения на изображении, можно определить движущиеся контуры.

Адаптивные фоновые модели применяются из-за их возможности распознавать изменения на кадрах, которые вызваны изменением освещения в сценах вне помещения или изменением фона из-за движения камеры. Однако эти методы не эффективны при быстром изменении сцены и обычно работают некорректно.

Рассмотрим алгоритм, который основан на алгоритме, предложенным в работе Rodriguez-Canosa G. R., Thomas S., Cerro J. и др. [1] и позволяет опре-

делить движущиеся объекты на видеоизображении в реальном режиме времени.

Постановка задачи

Пусть дан видеофайл, содержащий съёмку поверхности земли, записанный с камеры, жёстко закреплённой на движущемся объекте, например, квадрокоптере. Анализируя кадры видеоизображения, необходимо определить вектор направления движения объекта с камерой в каждый момент времени, и в случае, если на кадрах присутствуют движущиеся (динамические) объекты, определить их положение на кадре.

Результатом работы алгоритма будут являться вектор направления движения камеры и координаты прямоугольников, ограничивающих контуры движущихся объектов.

В данной задаче моментом времени является один кадр видеофайла, поэтому частота выдачи результата алгоритмом в единицу времени будет равняться числу кадров в единицу времени данного видео.

Предполагается, что плоскость полёта движущегося объекта параллельна плоскости земли, а объектив камеры расположен так, что вектор, выходящий из него, перпендикулярен плоскости полёта и направлен к плоскости земли.

Глава 1. Обзор метода

Алгоритм, предложенный в работе [1], сочетает в себе определение движущихся объектов, основанное на выявлении особенностей статических точек, и сравнение оптического потока для выявления точек, принадлежащих динамическим объектам. Главная идея рассматриваемого метода состоит в сравнении искусственного оптического потока, основанного на движении камеры, с действительным оптическим потоком и отслеживании различий в них.

Рассмотрим основные этапы алгоритма:

1. Вычисление действительного оптического потока.
2. Вычисление искусственного оптического потока.
3. Идентификация динамических точек и их фильтрация.

1.1. Вычисление действительного оптического потока

Оптический поток – векторное поле, характеризующее движение объектов, которое возникает из-за движения камеры, относительно сцены. Векторы данного поля показывают как изменилось положение точек в текущем кадре относительно предыдущего.

Данная задача рассматривается для двумерных изображений с учетом времени, поэтому пиксель с координатами (x, y) имеет интенсивность $I(x, y, t)$, где t – переменная времени. За один кадр точка будет перемещена на $\Delta x, \Delta y, \Delta t$ и получается следующее уравнение:

$$I(x, y, t) \approx I(x + \Delta x, y + \Delta y, t + \Delta t).$$

Считая перемещение малым, по формуле Тейлора получим:

$$I(x+\Delta x, y+\Delta y, t+\Delta t) \approx I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t.$$

Следовательно:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0$$

и

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

где (V_x, V_y) – вектор скорости оптического потока в $I(x,y,t)$.

В итоге имеем:

$$I_x V_x + I_y V_y = -I_t$$

или

$$\nabla I^T \cdot \vec{V} = -I_t. \tag{1}$$

Полученное уравнение имеет две неизвестных и не имеет однозначного решения. Решение может быть найдено с помощью добавления ограничений.

Определение действительного оптического потока будет реализовано алгоритмом Лукаса – Канаде [2]. Этот метод является дифференциальным локальным алгоритмом нахождения оптического потока. Неоднозначность уравнения (1) решается за счет информации о соседних пикселях каждой точки. Вводится предположение о том, что оптический поток одинаков в окрестности каждой точки.

Рассмотрим окно с центром в пикселе p. Оптический поток должен быть одинаков для всех точек окна. А вектор (V_x, V_y) оптического потока в точке p должен быть решением системы

$$\begin{cases} I_x(q_1) \cdot V_x + I_y(q_1) \cdot V_y = -I_t(q_1), \\ I_x(q_2) \cdot V_x + I_y(q_2) \cdot V_y = -I_t(q_2), \\ ... \\ I_x(q_n) \cdot V_x + I_y(q_n) \cdot V_y = -I_t(q_n), \end{cases} \tag{2}$$

где $q_1, ..., q_n$ – пиксели в окне, $I_x(q_i), I_y(q_i), I_t(q_i)$ – частные производные I в точке q_i.

Переписав систему (2) в матричной форме, получим

$$Av = b, \tag{3}$$

где

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ ... & ... \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ ... \\ -I_t(q_n) \end{bmatrix}.$$

Решая систему (3) с помощью метода наименьших квадратов, получим систему $A^T Av = A^T b$, откуда $v = (A^T A)^{-1} A^T b$.

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_x(q_i) I_y(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix} \tag{4}$$

Решив систему (4) для каждой точки изображения, найдем оптический поток.

1.2. Вычисление искусственного оптического потока

Искусственный поток представляется положением всех точек, полученных для действительного оптического потока, математически спроецированных на

следующий кадр с учётом движения камеры. Иными словами, вычислив искусственный оптический поток, можно определить вектор направления движения камеры. Движение камеры представляется поворотом и смещением, которые получают в результате работы алгоритма PTAM [3].

Для получения искусственного потока используется гомографическая проекция. Обычно гомография проецирует положение точки из плоскости одной камеры в плоскость другой. В нашем случае используются некоторые предположения для упрощения и ускорения процесса проецирования. Главное предположение состоит в том, что поверхность земли не имеет существенного наклона, т. е. средний наклон можно считать близким к нулю. Небольшие изменения рельефа порядка 40–50 см не повлияют на работоспособность алгоритма.

1.3. Идентификация динамических точек и их фильтрация

В рассматриваемой задаче в качестве представления оптических потоков используются наборы векторов. Стоит отметить, что подавляющее число векторов искусственного оптического потока взаимнопараллельны. Просуммировав их, можно получить результирующий вектор, который будет показывать направление движения камеры в сцене.

Динамический характер точки выясняется в сравнении вышеперечисленных оптических потоков. Динамические объекты идентифицируются набором векторов действительного оптического потока, имеющих отличное от вектора движения камеры направление и/или величину. Для продолжения вычисления необходимо группировать векторы на изображении. Может получиться, что некоторые из векторов окажутся изолированными относительно других.

Часто они могут быть вызваны неконтролируемыми колебаниями или другими ошибками. В общем случае эти векторы не принимаются во внимание. Группа векторов, находящихся в определённой окрестности друг от друга, принимается за динамический объект.

В результате работы алгоритма Лукаса-Канаде получается оптический поток изображения. После вычитания суммарного вектора искусственного из векторов действительного потока получается действительной оптический поток с учетом искусственного, который будет представлен двумя матрицами скоростей V_x и V_y для каждой точки изображения из которых, в свою очередь, можно вычислить матрицы направления скорости и величины скорости M в каждой точке. Каждый элемент матрицы M вычисляется по формуле

$$M_{i,j} = \sqrt{V_{x_{i,j}}^2 + V_{y_{i,j}}^2}$$

По значениям матрицы скорости можно реализовать выделение движущихся объектов на изображении по принципу поиска групп рядом стоящих ненулевых значений матрицы. Скопление таких значений можно найти путем определения контура. Контуром, ограничивающим данное скопление, будет являться прямоугольник, ограниченный параллельными прямыми. Так как поиск будет происходить в матрице, параллельными прямыми здесь будут строки и столбцы матрицы с координатами $j = M_{j_{min}}, M_{j_{max}}$ и $i = M_{i_{min}}, M_{i_{max}}$. Получить эти значения можно по следующему методу:

1. Находится элемент матрицы M со значением $M_{i,j} > 0$. Поиск производится в порядке с первой до последней строки матрицы M.

2. Пока $j > 1$ и $M_{i,j} > 0$ перемещаемся по строке матрицы влево ($j = j-1$).

3. Точка $M_{i,j}$ в данный момент будет принадлежать столбцу $M_{j_{min}}$, ограничивающего контур слева.

4. Пока $M_{i+1,j} == 0$ перемещаемся по строке вправо ($j = j + 1$).

5. Переходим к следующей строке $i = i + 1$.

6. Переходим к шагу 2 и выполняем до тех пор, пока не достигнем края матрицы или состояния, когда справа и снизу от текущей точки будут только нули.

7. Достигнув предельного значения, получим крайнюю левую точку ограничивающего контура движущегося объекта. Таким же образом найдем строку $M_{i_{max}}$ – нижнюю границу и столбец $M_{j_{max}}$ – правую границу. Строка $M_{i_{min}}$, являющаяся границей контура сверху, будет найдена на шаге 1 т. к. поиск ненулевых значений производился сверху-вниз.

Из полученных границ можно получить левую верхнюю и правую нижнюю точку прямоугольника: $(M_{i_{min}}, M_{j_{min}})$ и $(M_{i_{max}}, M_{j_{max}})$ соответственно.

Для предотвращения случая повторного рассмотрения одного и того же контура алгоритмом, нужно применить следующий подход: после нахождения очередного контура необходимо обнулить все значения внутри него и на его границе. В этом случае, при дальнейших итерациях, алгоритм ничего не будет знать о предыдущих значениях и не будет выделять их снова.

Рассмотрим пример. Пусть дана матрица M скоростей изображения размером 7*5 пикселей

$$M = \begin{bmatrix} 0 & 0.7 & 0.9 & 0.8 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 0.7 & 0.4 & 0 & 0 \\ 0 & 0.7 & 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0.4 \end{bmatrix}$$

Матрица M' содержит выделенные цветом элементы. Один цвет характеризует один объект. Таким образом, в данном примере на изображении присутствуют два движущихся объекта, которые будут выделены двумя прямоугольниками с границами: $M_{i_{min}} = 1, M_{j_{min}} = 2, M_{i_{max}} = 3, M_{j_{max}} = 5$ и $M_{i_{min}} = 4, M_{j_{min}} = 6, M_{i_{max}} = 5, M_{j_{max}} = 7$ или с координатами левого верхнего и правого нижнего углов $(1, 2), (3, 5)$ и $(4, 6), (5, 7)$ соответственно.

$$M' = \begin{bmatrix} 0 & 0.7 & 0.9 & 0.8 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 0.7 & 0.4 & 0 & 0 \\ 0 & 0.7 & 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0.4 \end{bmatrix}$$

Стоит отметить, что для фильтрации ошибочных векторов можно использовать нижнее пороговое значение скорости, отличное от нуля. От случая к случаю, корректное пороговое значение будет меняться. Оно зависит от высоты между камерой и объектами, скоростью их движения и скоростью движения камеры.

Кроме того, можно ввести ограничение на минимальную и максимальную площадь пространств, ограниченных контурами. Если площадь текущего контура больше максимально заданной или меньше минимальной, то данный контур просто не учитывается.

Глава 2. Реализация

В данной главе будет рассмотрена реализация алгоритма, представленного в обзоре.

2.1. Выбор инструмента

Выбор инструмента реализации состоял из трёх языков программирования: C++, Python и MATLAB. C++ используется из-за его высокой производительности. Язык Python ускоряет процесс написания программ из-за его простого синтаксиса и из-за того, что язык интерпретируемый. MATLAB использует библиотеки, написанные для большинства популярных процессоров. Библиотеки используют процессорные инструкции, например MMX и SSE2, и другие инструменты процессора для эффективной работы с матричными вычислениями.

Вместе с каждым языком предполагается использовать инструмент для работы с графикой. Для C++ и Python это библиотека OpenCV [4]. Для MATLAB – Copmuter Vision System Toolbox [5].

Несмотря на то, что на C++, при должном подходе, получаются эффективные программы, у этого языка есть недостатки, один из которых – сложность написания качественного кода и сложность поиска ошибок. Язык Python лаконичен и прост, но за это разработчики платят скоростью его работы.

Среда MATLAB была выбрана в качестве инструмента реализации алгоритма. MATLAB не обладает вышеперечисленными недостатками других языков программирования.

2.2. Модификация алгоритма

В первичной реализации решения задачи детектирования движущихся объектов использовался пирамидальный алгоритм Лукаса – Канаде. Скорость работы алгоритма зависит от размера обрабатываемых кадров: чем больше кадр, тем дольше он обрабатывается. С другой стороны, чем качественнее кадр и чем больше его размер, тем точнее будет определён оптический поток. Следовательно, в зависимости от конкретной задачи и имеющихся вычислительных мощностей нужно выбирать, с каким качеством и скоростью будут произведены вычисления.

Причиной неточной работы алгоритма Лукаса – Канаде могут служить различные шумы на изображении. Появление шумов во многом зависит от камеры, производящей съёмку. В алгоритме Лукаса – Канаде перед тем, как обработать изображение, для оптимальности часто его обесцвечивают.

Из-за особенностей алгоритма, перед тем, как работать с изображением, для лучшей производительности, к нему применяют сглаживание. Было опробовано применение сглаживания по Гауссу и медианного фильтра. Однако данные инструменты заметно не повлияли на качество результата.

Вместо пирамидального алгоритма Лукаса – Канаде более эффективным является использование итеративного алгоритма Лукаса – Канаде. Основное его отличие в том, что при работе с текущим кадром в нём используется информация, накопленная за период работы с предыдущими кадрами. Данная разновидность алгоритма работает в несколько раз быстрее и качественнее.

Работа пирамидального алгоритма Лукаса–Канаде представлена на рис. 1 и рис. 3, а итеративного – на рис. 2 и рис. 4. Желтые стрелки – это визуализация оптического потока. Результат работы пирамидального алгоритма содержит большое количество лишних векторов. Результат работы итератив-

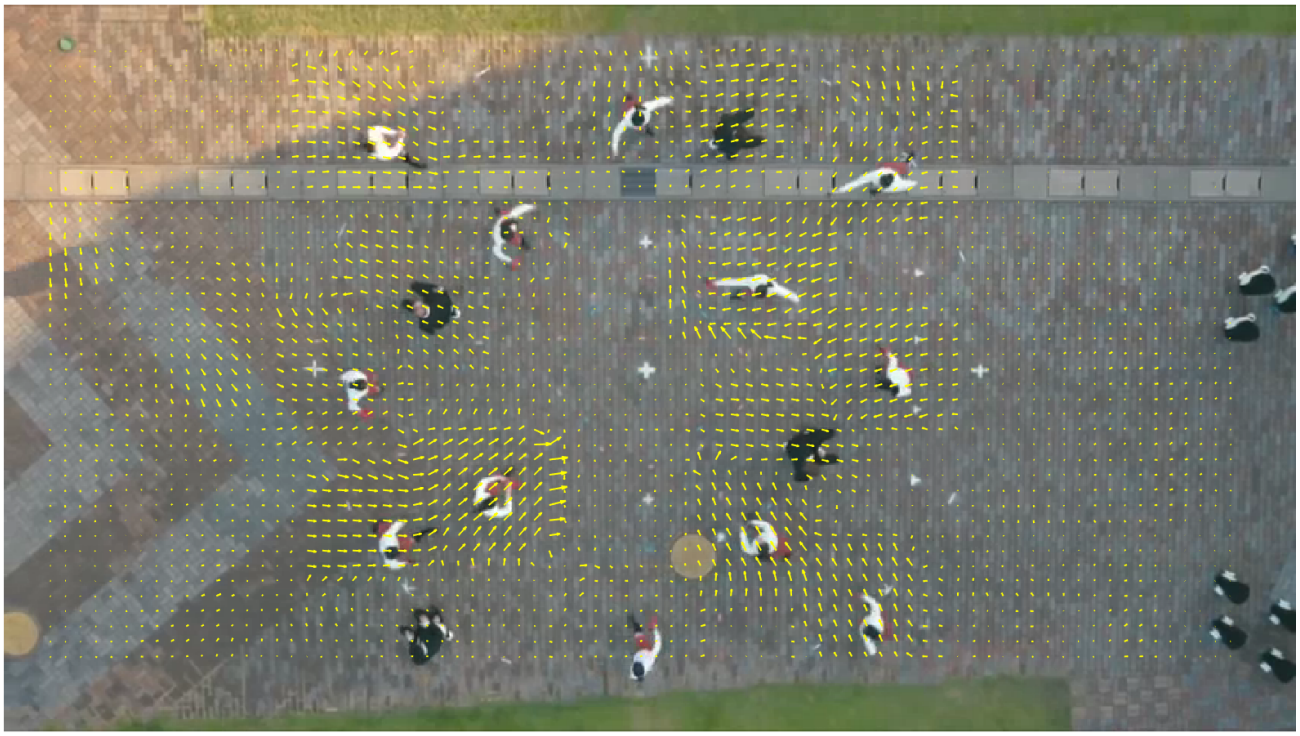

Рис. 1: Пример работы пирамидального алгоритма Лукаса–Канаде.

Рис. 2: Пример работы итеративного алгоритма Лукаса–Канаде.

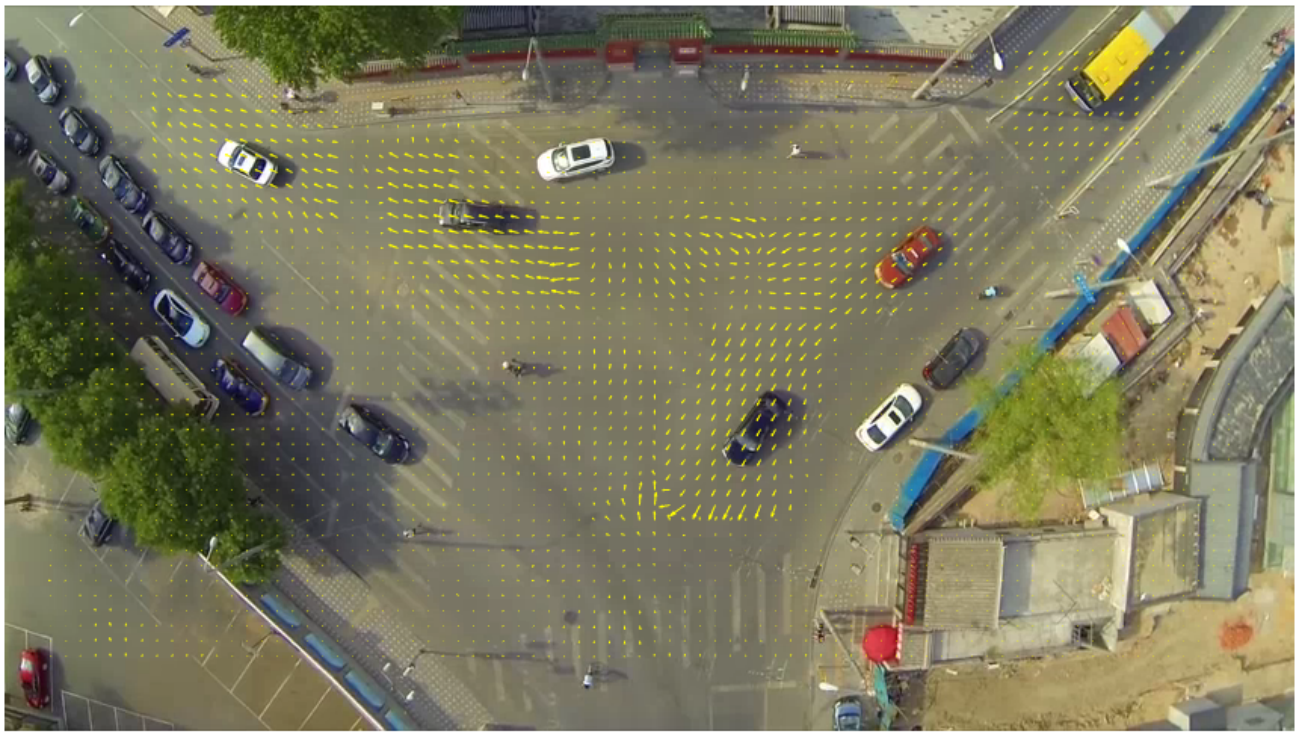

Рис. 3: Пример работы пирамидального алгоритма Лукаса–Канаде.

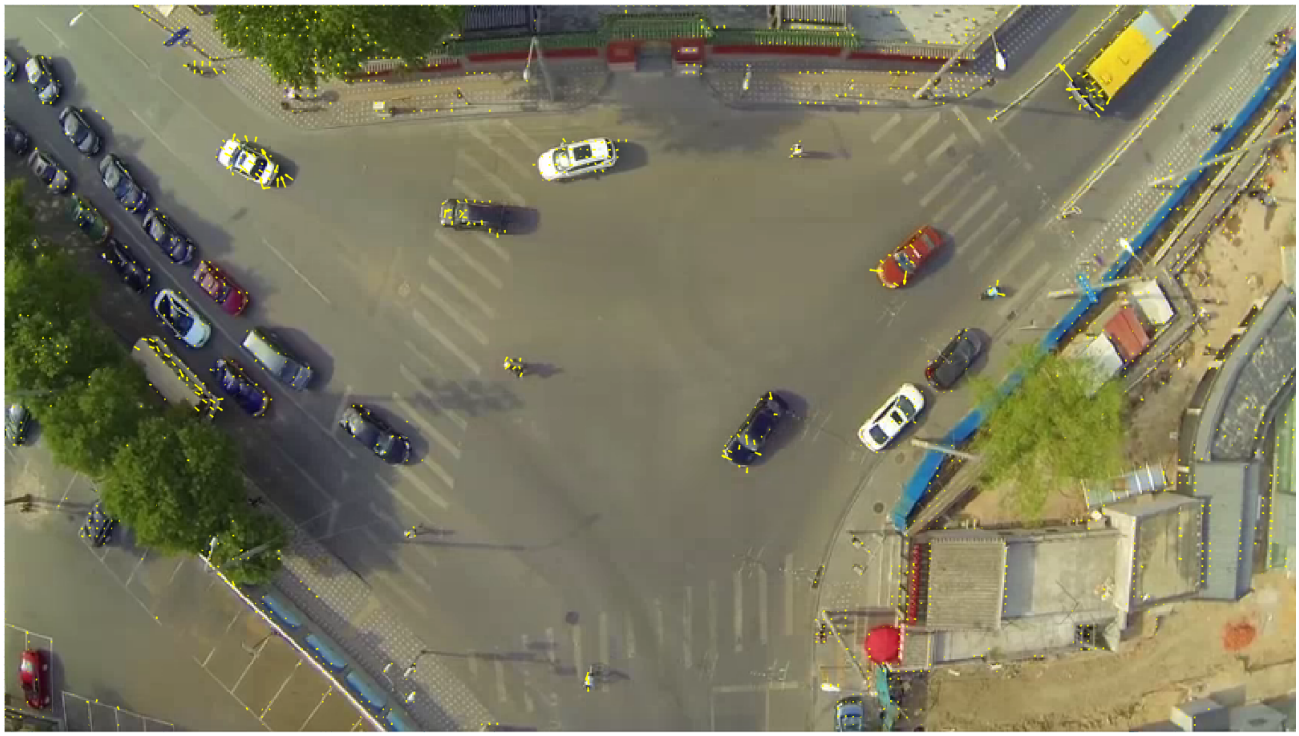

Рис. 4: Пример работы итеративного алгоритма Лукаса–Канаде.

ного алгоритма не имеет данного недостатка. Таким образом, итеративный алгоритм работает качественнее.

Для демонстрации работы алгоритмов для обоих примеров были выбраны видеофайлы с размером кадров 854*480 пикселей. Пирамидальный алгоритм имеет скорость работы 6 кадров в секунду. Итеративный алгоритм, в свою очередь, обрабатывает 24 кадра в секунду. Отсюда следует, что пирамидальный алгоритм работает медленнее итеративного. Измерения производились на компьютере с процессором Intel Core i5-5257U с частотой работы 2.7 ГГц.

Предлагается отказаться от использования алгоритма PTAM в силу его сложности и ресурсозатратности. Задачу, которую решал PTAM алгоритм, будет решать метод, состоящий из следующих этапов:

1. На теукщем и предыдущем кадре по алгоритму из [6] выбираются особые точки.

2. По методу, представленному в [7], подмножеству точек из текущего кадра ставится соответствие подмножество эквивалентных точек из предыдущего кадра.

3. По множеству пар точек, найденных на предыдущем этапе, алгоритм, представленный в [8], определяет геометрическое преобразование, например аффинное, характеризующее движение камеры.

Для визуального выделения движущихся объектов использовался класс BlobAnalysis [9] из пакета Computer Vision System Toolbox.

Рассмотрим пример. На рис. 5 и рис. 6 представлены предыдущий и текущий кадр соответственно. На рис. 7 изображены пары соответствующих точек на предыдущем и текущем кадре. Точки в виде синих окружностей соответствуют особым точкам на предыдущем кадре. Точки в виде красных

Рис. 5: Предыдущий кадр

Рис. 6: Текущий кадр

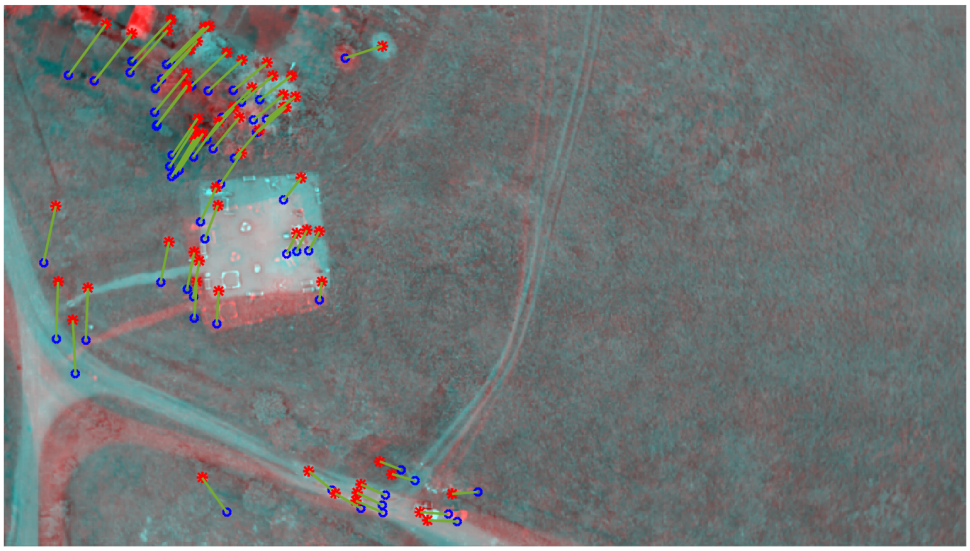

Рис. 7: Пары особых точек.

многоугольников соответствуют особым точкам на текущем кадре. Из последнего этапа алгоритма получается аффинное преобразование. Для данной пары кадров матрица аффинного преобразования будет следующей:

$$\begin{bmatrix} 0.9897 & 0.1022 & 0.0 \\ -0.0979 & 0.9908 & 0.0 \\ 34.0918 & -39.8013 & 1.0 \end{bmatrix}$$

2.3. Иллюстрация применения

На рис. 8 представлен пример работы алгоритма. Кадр взят из видеосъемки квадрокоптера, пролетающего над городской автомобильной дорогой. На изображении есть движущиеся автомобили, которые выделены прямоугольниками.

В правом верхнем углу изображения виден отрезок в окружности, который показывает направление движения объекта с камерой.

Очередной пример представлен на рис. 9, где представлен кадр с видеозаписи квадрокоптера. На кадре видно группу людей, выполняющих элемент танца. Люди, находящиеся в движении, выделены зелеными прямоугольниками.

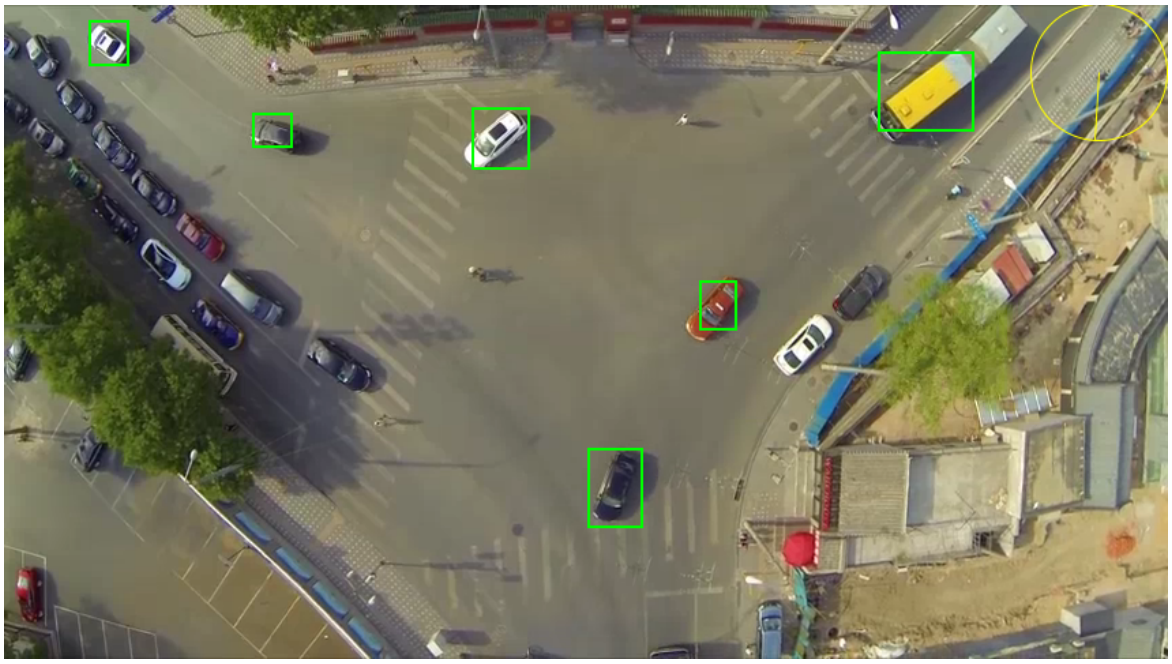

Рис. 8: Вид дороги сверху.

Рис. 9: Кадр видеосъёмки квадрокоптера.

Выводы

Для эффективной реализации алгоритма [1] в MATLAB необходимо было его модифицировать, а именно заменить пирамидальный алгоритм Лукаса–Канаде на итеративный алгоритм Лукаса–Канаде.

Вместо алгоритма PTAM, в силу сложности его реализации, эффективнее использовать метод из [8], реализация которого, в частности, присутствует в пакете Computer Vision System Toolbox среды MATLAB.

Заключение

В работе произведен анализ алгоритма детектирования движущихся объектов с одиночной камеры [1] и осуществлена его реализация в среде MATLAB. Предложена модификация алгоритма, состоящая в использовании итеративного алгоритма Лукаса–Канаде вместо пирамидального и алгоритма MLESAC вместо алгоритма PTAM. Проведено сравнение результатов работы исходного и модифицированного алгоритмов.

Список литературы

[1] Rodriguez-Canosa G. R., Thomas S., Cerro J. et al. A real-time method to detect and track moving objects (DATMO) from unmanned aerial vehicles (UAVs) using a single camera // Remote Sensing. 2012. No 4. P. 1090–1111.

[2] Bouguet J.Y. Pyramidal implementation of the Lucas–Kanade feature tracker. Description of the algorithm; Technical report; Intel Corporation Microprocessor Research Lab: Santa Clara, CA, USA, 1999. `http://robots.stanford.edu/cs223b04/algo_tracking.pdf`

[3] Klein G., Murray D. Parallel tracking and mapping for small AR workspaces // Proceedings of 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. 2007. P. 225–234.

[4] OpenCV Library `http://opencv.org/`

[5] Computer Vision System Toolbox `http://www.mathworks.com/products/computer-vision/`

[6] Bay, H., A. Ess, T. Tuytelaars, and L. Van Gool. SURF:Speeded Up Robust Features. // Computer Vision and Image Understanding (CVIU). 2008. Vol 110. No 3. P. 346-–359.

[7] Lowe, David G. Distinctive Image Features from Scale-Invariant Keypoints // International Journal of Computer Vision. Vol 60. No 2. P. 91-–110.

[8] Torr P. H. S., Zisserman A. MLESAC: a new robust estimator with application to estimating image geometry // Computer Vision and Image Understanding. 2000. No 78. P. 138–156.

[9] Vision Blobanalysis Class http://www.mathworks.com/help/vision/ref/vision.blobanalysis-class.html