# СОДЕРЖАНИЕ

ВВЕДЕНИЕ....4
1 Основная часть....5
1.1 Диаграмма IDEF0....5
1.1.1 Выявление точек интереса....7
1.1.2 Вычисление оптического потока....8
1.1.3 Построение схемы....10
1.2 Требования к программному обеспечению....15
1.3 Входные и выходные данные....16
1.4 Реализации алгоритмов....17
1.4.1 Выявление точек интереса....18
1.4.2 Вычисление оптического потока....18
1.4.3 Построение схемы....19
1.5 Тестирование отрисовки карты-схемы....21
1.6 Взаимодействие с программой....24
1.7 Инструкции для запуска....24
1.8 Пример работы....24
1.9 Введение в исследование....26
1.10 Характеристики компьютера....27
1.11 Характеристики камеры....27
1.12 Предмет исследования....28
1.13 Исследование зависимости количества точек интереса от освещённости сцены....28
1.14 Исследование точности карты-схемы от конфигурации сцены....29
1.14.1 Однородная сцена....30

1.14.2 Сцена низкой сложности.....31
1.14.3 Сцена средней сложности.....34
1.14.4 Сцена высокой сложности.....35
1.15 Выводы из исследования.....37
ЗАКЛЮЧЕНИЕ.....38
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....39
ПРИЛОЖЕНИЕ А.....41

# ВВЕДЕНИЕ

Потребность в навигации и картографировании местности вокруг себя всегда была присуща человеку. Изначально составление карты требовало высокой квалификации, точных математических расчётов вручную. В настоящее же время для составления различных карт и схем местности активно используются возможности компьютеров: специальное программное обеспечение, различные камеры и датчики. Стоит отметить, что компьютерная (или цифровая) картография не является самостоятельным разделом, собственно, картографии. Она просто изменила способы визуализации данных. Если раньше это были чернила на бумаге, то сейчас это полностью экран компьютера.

С возникновением автономных роботов остро встала проблема их ориентации в пространстве (открытом или закрытом). Роботу-пылесосу, например, необходимо определить расстояние до какого-либо препятствия, на основании этого построить свой маршрут и запомнить его. Также при возникновении чрезвычайной ситуации в здании человеку нужно знать актуальную схему помещения, что зачастую не всегда так.

Также построение карты-схемы помещения может быть полезным на различных фазах строительства (от проектирования до сдачи в эксплуатацию), ведь было бы удобно не имея на руках специального оборудования и не обладая какой-либо квалификацией сфотографировать помещение на камеру телефона и получить актуальную карту.

Во время выполнения выпускной квалификационной работы был разработан метод картографирования помещений на основе визуальной одометрии. Также была проведена апробация метода картографирования помещений с помощью визуальной одометрии для установления зависимостей результатов работы от различных параметров системы.

# 1 Основная часть

В разделе приведены IDEF0-диаграммы, проведён анализ входных данных и их ограничений, требуемая функциональность метода, особенности его реализации.

## 1.1 Диаграмма IDEF0

На рисунках 1, 2 и 3 изображены этапы построения карты помещения в виде диаграммы IDEF0.

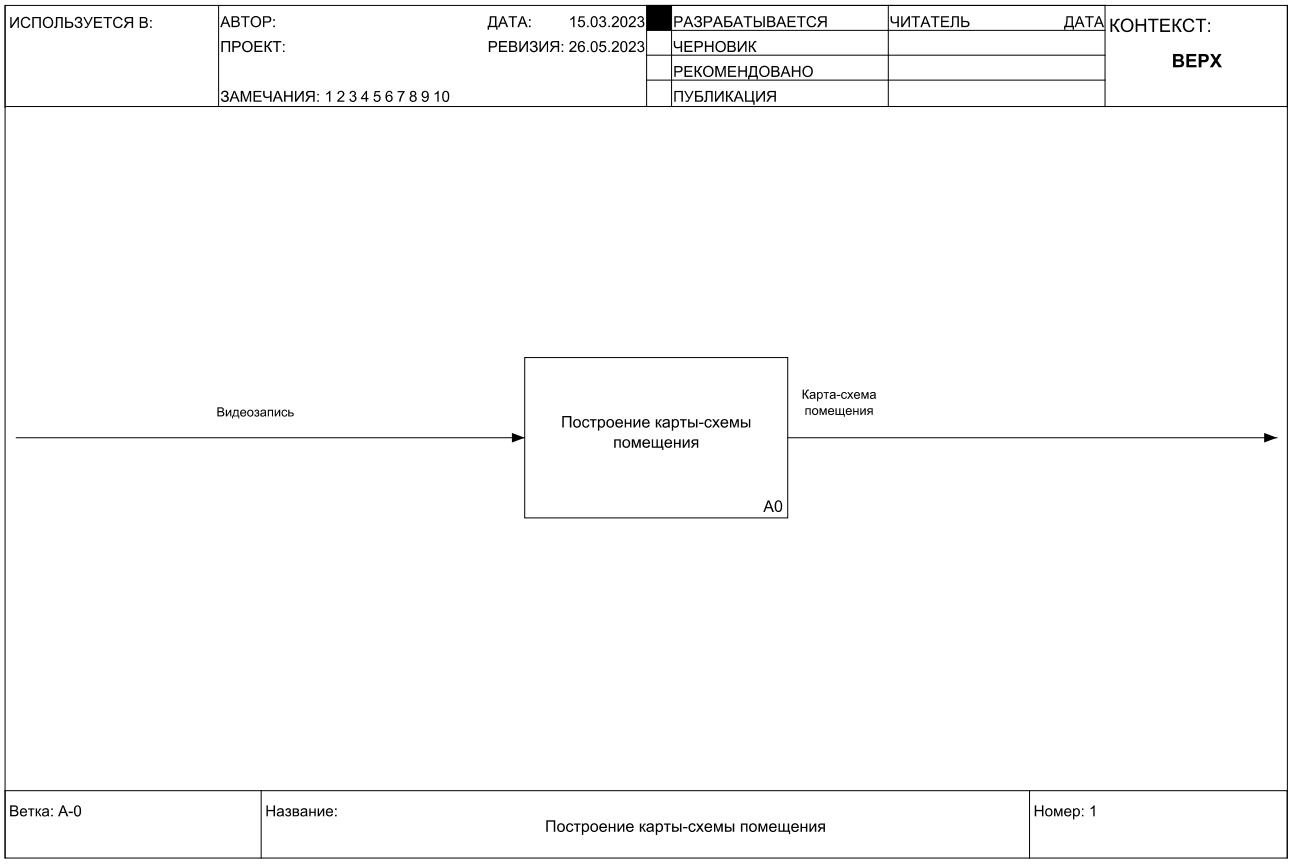



Рисунок 1 — Задача построения карты помещения

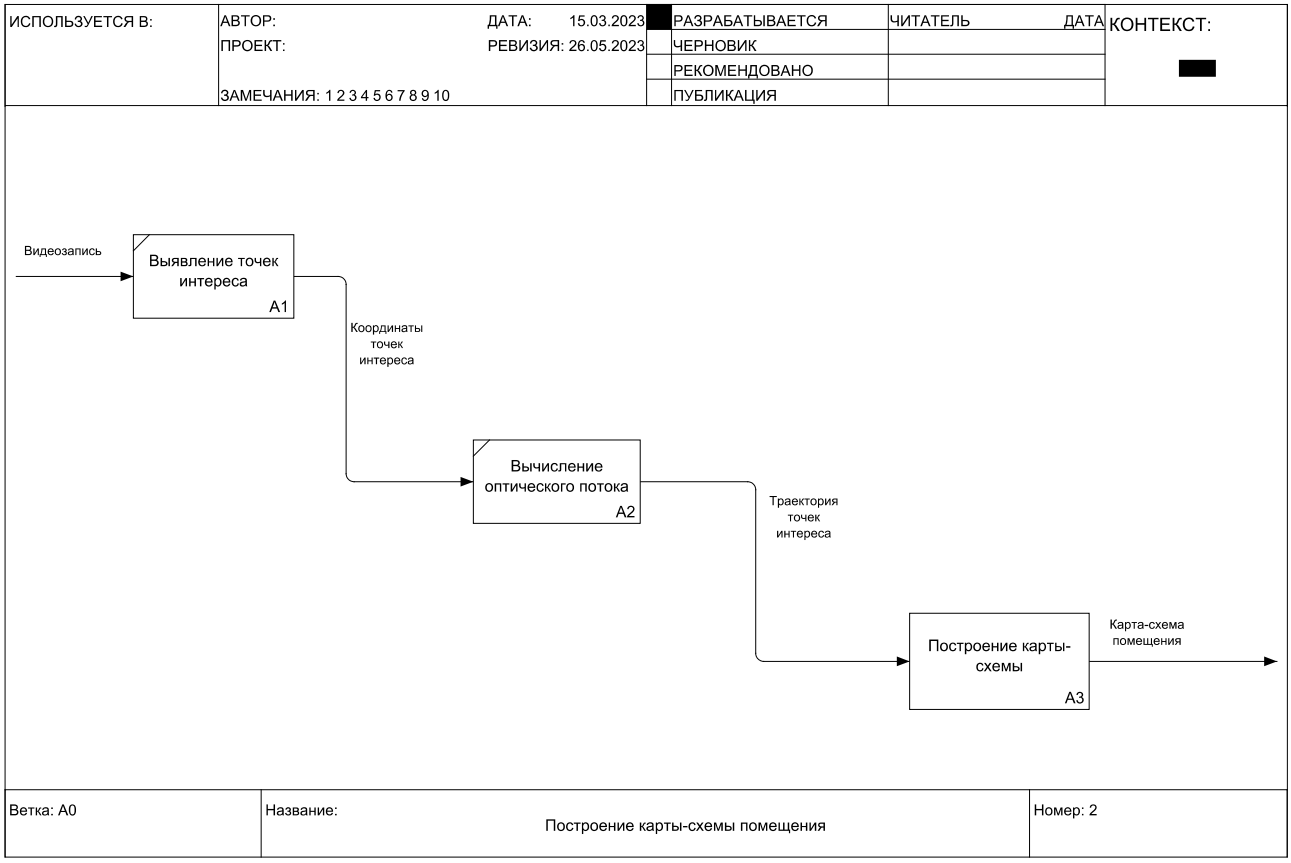



Рисунок 2 — Этапы построения карты помещения

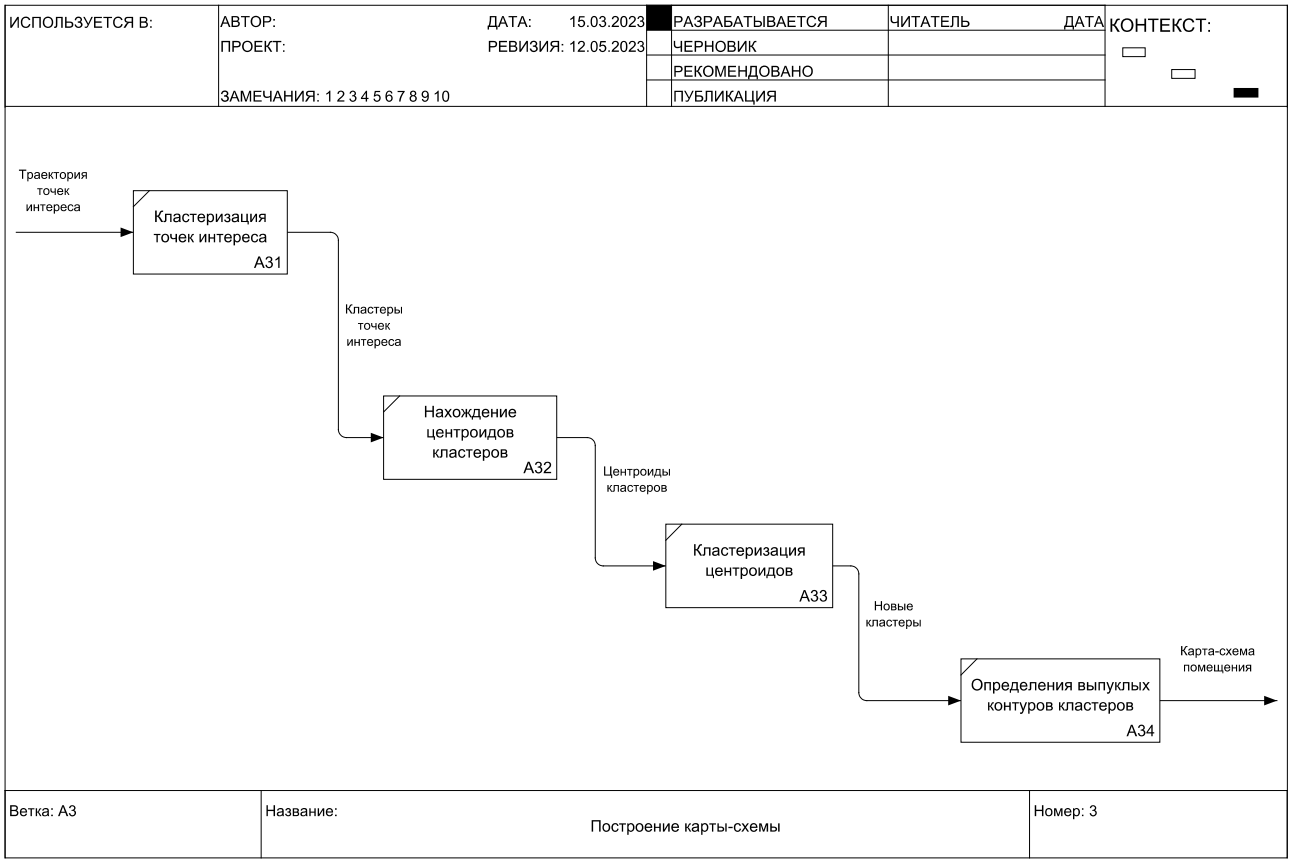



Рисунок 3 — Этапы формирования карты помещения

### 1.1.1 Выявление точек интереса

Чтобы отслеживать характерные признаки кадра и его индивидуальные особенности, на нём выделяются так называемые точки интереса, или «углы». Обычно они представляют собой, собственно, углы стен, предметов.

Для решения задачи поставленной задачи выделяется несколько этапов [1]:

1. Вычисление собственных чисел и собственных векторов блоков изображения. Для каждого пикселя $p$ определяется некоторая окрестность $\mathrm{S}(p)$ заданного размера. Затем вычисляется ковариационная матрица производных функции яркости изображения $I$ по координатам $x$ и $y$:

$$M = \begin{bmatrix} \sum_{S(p)} \left(\frac{dI}{dx}\right)^2 & \sum_{S(p)} \frac{1}{dxdy} \\ \sum_{S(p)} \frac{1}{dxdy} & \sum_{S(p)} \left(\frac{dI}{dy}\right)^2 \end{bmatrix}, \quad (1)$$

где: $x$ и $y$ — координаты пикселя в системе координат кадра. Из матрицы $M$ рассчитывается минимальное собственное значение, называемое картой точности $Q(x, y)$. Каждую точку такого изображения называют коэффициентом точности *qualityLevel*.

2. Подавляются немаксимумы изображения, причём локальные максимумы в окрестности $3 \times 3$ сохраняются.
3. Углы с собственным значением меньше, чем произведение $qualityLevel \cdot \max_{x,y}(Q(x, y))$, отбрасываются. Здесь *qualityLevel* — заданный коэффициент точности.
4. Оставшиеся углы (они же точки интереса) сортируется по соответствующим им коэффициентам точности в порядке убывания.
5. Отбрасывается каждый угол, для которого существует более точный угол на расстоянии меньше заданного минимального.

Таким образом формируется вектор координат углов (точек интереса) исходного изображения.

### 1.1.2 Вычисление оптического потока

Для решения задачи нахождения оптического потока широко используется дифференциальный алгоритм Лукаса — Канаде. Он позволяет найти смещение каждого пикселя между двумя кадрами с помощью частных производных по двум направлениям изображения. [2]

Допустим, что в окрестности пикселя значение оптического потока практически неизменно, то есть значения пикселя переходят из одного кадра в другой без каких-либо изменений. Тогда возможно записать функцию яркости пикселя $I$ от его координат и времени:

$$I(x,y,t)=I(x+u_x,y+u_y,t+1), \tag{2}$$

где: $x$ и $y$ — координаты пикселя в системе координат кадра, $t$ — номер кадра в последовательности, причём «расстояние» между кадрами равно единице. Таким образом, вектор оптического потока $(V_x,V_y)$ пикселя $p$ равен:

$$\begin{cases} I_x(q_1)V_x+I_y(q_1)V_y=-I_t(q_1) \\ I_x(q_2)V_x+I_y(q_2)V_y=-I_t(q_2), \\ \cdots \\ I_x(q_n)V_x+I_y(q_n)V_y=-I_t(q_n) \end{cases} \tag{3}$$

где $q_1$, $q_2$, …, $q_n$ — пиксели внутри окна с центром в пикселе $p$, а $I_x(q_i)$, $I_y(q_i)$, $I_t(q_i)$ — частные производные изображения по координатам и времени в точке $q_i$ . Переписав (3) в матричной форме и применив метод наименьших квадратов, получим вектор оптического потока:

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

Одним из недостатков обычного алгоритма Лукаса — Канаде является его *локальность*, то есть невозможно определить смещение пикселей внутри таких участков кадра, размер которых больше окрестности пикселя.

Пирамидальный вариант этого алгоритма призван устранить эту проблему [3]. При обработке изображения размерности $N\times N$ строится так называемая гауссовская пирамида изображений — упорядоченное множество $L$ -ый элемент которого это матрица $2^{M-L}\times 2^{M-L}$ пикселей, где $L\in[0,M]$, $M=\log_2 N$. Последующий элемент этой пирамиды формируется фильтрацией и прореживанием предыдущего в два раза по строке и столбцу.

На рисунке 4 показана схема пирамидального алгоритма Лукаса — Канабе.


Рисунок 4 — Схема пирамидального алгоритма Лукаса — Канабе

### 1.1.3 Построение схемы

Зная координаты перемещения точек интереса от кадра к кадру входной последовательности изображений можно соединить их определённым образом и таким образом построить контуры объектов на фотографиях. Эти контуры и будут в дальнейшем считаться картой-схемой.

Однако отсутствие должной стабилизации камеры и тряска значительно осложняют эту задачу — слишком близко расположенные углы образуют «кучи» точек, через которые невозможно провести предполагаемый контур предмета. Для устранения эффекта тряски было решено алгоритмы кластеризации и определения так называемых центроидов.

Кластер — это подмножество объектов в выборке, которые похожи друг на друга в пространстве. Метрику сходства необходимо выбирать под конкретную задачу, но чаще всего используется евклидово расстояние.

Для этапа «усреднения» значений точек интереса будет применён алгоритм кластеризации $k$-средних ($k$-means). Этот алгоритм, относящийся к числу плоских алгоритмов кластеризации, строит заданное число кластеров, расположенных как можно дальше друг от друга. Одни и те же объекты можно по-разному разбить на кластеры, это зависит от начального положения центроидов. Чаще всего выбирается $k$ случайных объектов из выборки, которые становятся центроидами. Целью алгоритма является минимизация дисперсии внутри кластера. Алгоритм $k$-средних:

1. Берутся случайные точки в пространстве, которые принимаются за центры кластеров (центроиды $c_i$).
2. Для каждого объекта в выборке находится ближайший к нему центроид.
3. Каждому центроиду соответствует множество ближайших объектов. Для каждого образованного кластера находится его центроид:

$$c_i = \frac{\sum_{j=1}^{N_i} x_j}{N_i}, \tag{5}$$

где $N_i$ — количество объектов в кластере.

4. Центроиды переходят в найденный центр кластера. Далее алгоритм повторяется, пока центроиды не перестанут менять положение.

Функция стоимости алгоритма $L$ выглядит следующим образом:

$$\mathrm{L}(c_1, c_2, \ldots, c_k) = \sum_{i=1}^{k} \sum_{j=1}^{N_i} \|x_j - c_i\| \to \min, \quad (6)$$

где $c_i$ — центроиды кластеров; $k$ — количество кластеров; $N_i$ — количество объектов в кластере; $x_j$ — объекты в кластере.

Схема работы алгоритма $k$-средних изображена на рисунке 5.


Рисунок 5 — Схема работы алгоритма $k$-средних.

Иерархическая агломеративная кластеризация будет использована для того, чтобы объединить центроиды, найденные на предыдущем этапе в новые кластеры. «Агломеративная» (то есть, «восходящая») означает, что алгоритм начнёт работу с одного объекта и постепенно объединит их в кластеры побольше. Для начала вычисляется матрица расстояний между каждой парой объектов. Далее выбирается пара объектов с минимальным расстоянием и вычисляется среднее значение, после чего снова рассчитывается новая матрица расстояний, с учётом того, что пара объектов становится одним новым объектом. Таким образом алгоритм повторяется, пока не останется только одно значение. Иерархия контуров предметов на видео означает иерархию соответствующих им кластеров, ведь вложенность очертаний предметов в друг друга при взгляде сверху очевидна.

Выпуклую оболочку получившихся кластеров будем считать контуром соответствующего объекта на видео, а совокупность таких контуров — картой-схемой. Вообще, выпуклой оболочкой некоторого множества точек является такое выпуклое множество (в нём все точки отрезка между двумя точками множества также должны принадлежать этому множеству), которое содержит заданную фигуру и которое само содержится в любом другом выпуклом множестве [4]. Построение выпуклой оболочки будет осуществляться алгоритмом быстрой оболочки, в основе которого лежит быстрая сортировка Хоара [5]. Метод быстрой оболочки предполагает следующие геометрические построения:

1. Пусть дано некоторое множество точек. Выберем такие две точки этого множества — $A$ и $B$, которые будут крайней левой и крайней правой соответственно. Проведём прямую через эти точки и обозначим подмножество точек выше или на прямой как $M_1$, а ниже или на ней — $M_2$ .
2. Теперь рассмотрим $M_1$. Выберем точку $P$ такую, что расстояние между ней и прямой $AB$ будет максимальным. Эта точка является вершиной выпуклой оболочки множества. Проведём отрезки $AP$ и $BP$. Точки внутри

образовавшегося треугольника $APB$ не будут рассматриваться в дальнейшем, так как находятся внутри границы оболочки.

3. Аналогично для множеств точек слева от $AP$ и справа от $BP$ находим самую удалённую точку от этих прямых и строим выпуклую оболочку.
4. Аналогично для $M_2$.

## 1.2 Требования к программному обеспечению

Разработанное программное обеспечение должно предоставлять следующие возможности:

- загрузку видеозаписи;
- просмотр промежуточных результатов (распределения точек по кластерам) в виде изображений;
- сохранение промежуточных результатов (распределения точек по кластерам) в виде изображений;
- просмотр получившейся карты-схемы;
- сохранение получившейся карты-схемы.

На рисунке 6 представлена диаграмма вариантов использования.

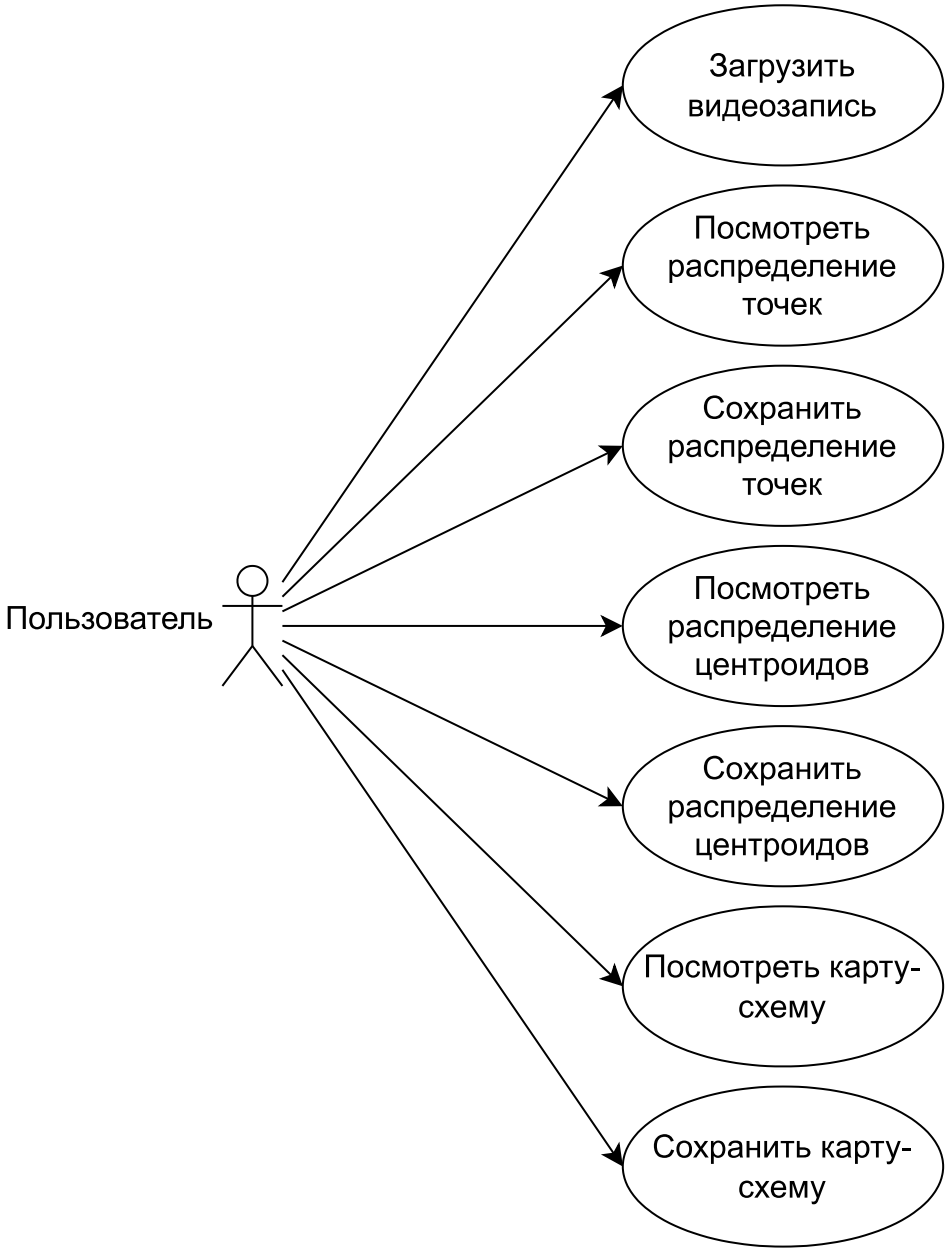



Рисунок 6 — Диаграмма вариантов использования

## 1.3 Входные и выходные данные

На вход программе подаётся видеозапись, формат которой может быть `AVI` или `MP4`. Ниже перечислены рекомендации для того, чтобы получившаяся карта-схема была как можно более точной:

- камера должна быть направлена вниз и охватывать как можно большее пространство картируемого помещения;
- быть неподвижной;
- длительность съёмки — 5 секунд.

Программные проверки, например, на предмет и длительность съёмки не планируются.

На выходе будет формироваться изображение, представляющее собой карту-схему сцены на входной видеозаписи. Оно имеет такие же размеры как и исходное видео, в нём на чёрном фоне отмечены контуры объектов (выпуклые оболочки кластеров), чей цвет соответствует цвету своих кластеров.

Для реализации разработанного метода был выбран язык Python [6]. В качестве среды разработки была выбрана Visual Studio Code [7].

## 1.4 Реализации алгоритмов

Поскольку язык Python не навязывает определённую структуру разрабатываемого на нём ПО, то в состав разработанной программы вошли как классы, так и отдельные функции.

Поиск точек интереса в кадрах и вычисление оптического потока выделены в класс `App`. Он имеет несколько полей:

- `track_len`. Представляет собой целое число, равное длине списка точек для вычисления оптического потока. При инициализации экземпляра класса равно 5;
- `detect_interval`. Представляет собой целое число, является шагом для нахождения точек интереса в кадре. Таким образом, для каждого $n$-ого кадра будут найдены точки интереса. При инициализации экземпляра класса равно 5;
- `tracks`. Представляет собой список координат точек интереса, чьё местоположение меняется от кадра к кадру. Каждый элемент такого списка — это кортеж из абсциссы и ординаты точки интереса в пространстве кадра. При инициализации экземпляра класса список создаётся пустым;
- `cam`. Это поле хранит дескриптор входного файла с видео формата, который указан в пункте 1.3;

– `frame_idx`. Представляет собой целое число, равное номеру текущего кадра. Поскольку поиск точек интереса и вычисление оптического потока происходит в цикле по всем кадрам видеозаписи, целесообразно хранить индекс кадра. При инициализации экземпляра класса равно 0.

Единственным методом класса `App` (помимо конструктора `__init__`, который единственным явным аргументом принимает исходный видеофайл) является метод `run`, реализующий функции поиска необходимых точек интереса. Этот метод возвращаемого значения не имеет, как и явных аргументов.

### 1.4.1 Выявление точек интереса

Получение последовательных изображений осуществляется посредством класса `VideoCapture` библиотеки OpenCV [8], затем цветовое пространство каждого кадра с помощью функции `cvtColor` [9] из традиционного (красный, зелёный, синий) в режим оттенков серого. За непосредственно определение интересующих точек отвечает функция `goodFeaturesToTrack`, в которую, помимо прочих, передаются такие параметры, как: максимальное количество обнаружаемых точек интереса (или, углов), характеристика минимального показателя качества обнаруженных углов, размер окрестности пикселя, в которой ведётся поиск угла и т. д. [10]. В последствие, точки интереса будут обнаруживаться через наперёд заданное количество кадров, так как функция не проверяет, насколько корректны следующие ключевые точки. Поэтому даже если какая-либо точка исчезает на изображении, есть вероятность, что оптический поток найдёт следующую точку, которая может выглядеть близко к ней.

### 1.4.2 Вычисление оптического потока

Имея два последовательных кадра видеозаписи и координаты точек интереса на них, можно, сравнив их, вычислить оптический поток. Достигается это засчёт использования функции `calcOpticalFlowPyrLK` [11], вычисляющей оптический поток по алгоритму Лукаса — Канаде (его принцип

действия был разобран в пункте 1.1.1), причём двоекратного. Вычисляется оптический поток точек интереса как от предыдущего кадра к последующему, так и в обратном направлении. Такие меры нужны, чтобы минимизировать возможную ошибку, когда новое положение точки интереса вычислить невозможно.

Координаты найденных таким образом точек заносятся в CSV-файл [12] (comma-separated values, с англ. «значения, разделённые запятыми»), а по самим точкам отрисовывается их траектория в видео. Отрисовка различных линий, окружностей реализована средствами всё той же библиотеки OpenCV [13].

### 1.4.3 Построение схемы

Помимо вышеобозначенного класса `App`, в программе имеется функция `draw_map`. Её аргументами являются переменные `file` и `example_frame`, а возвращаемым значением — `None`. Первый аргумент `file` представляет собой CSV-файл, хранящий координаты точек интереса в каждом обработанном кадре. Второй аргумент — первый кадр из видео, на размерах и цветовом профиле которого будут все промежуточные изображения и конечная карта-схема. Назначение данной функции — отрисовка карты-схемы объектов в видео.

Стоит отметить, что библиотеки Scipy и Scikit основаны на Numpy, поэтому типы выходных данных их методов и функций представляют типы данных из Numpy, например: тип данных `ndarray` (от англ. *n*-dimensional array — *n*-мерный массив). [14]

Первым этапом из файла выгружаются координаты точек в двумерный список. Таким образом данные становятся более удобными для их последующей обработки.

Вторым этапом осуществляется сама кластеризация точек. За кластеризацию отвечает класс `KMeans` из библиотеки `scikit-learn`. [15] Явные параметры, использовавшиеся при его инициализации: `n_clusters` равен числу, введённому пользователем, `n_init='auto'`, `max_iter=300`.

Число `n_init` означает количество итераций алгоритма *k*-средних для разных центроидов. В данном случае оно равно 1, так как используется не классический вариант алгоритма, а «жадный». В «жадном» варианте *k*-средних несколько раз выбирается «лучший» центроид в терминах вероятностного распределения точек. Параметр `max_iter` отвечает за максимальное количество итераций за один проход алгоритма.

Так как пользователю дана возможность просматривать изображения, формирующегося на каждом этапе, то — для наглядности — каждый кластер окрашен в разный цвет. Этого удалось достичь засчёт ассоциативного массива (он же «словарь»), в котором сопоставлены номера кластеров и их цвет в модели КЗС. Результирующее изображение составляется посредством функции `circle` из библиотеки OpenCV. Здесь следует добавить, что для того, чтобы хранить информацию о том, какая точка относится к какому кластеру, у экземпляра класса `KMeans` есть атрибут `labels_`, чей тип данных — `ndarray` и который хранит номера кластеров в порядке соответствующих им точек в исходном наборе данных.

Следующим этапом построения карты-схемы является кластеризация центроидов имеющихся точек. В библиотеке Scikit тоже есть подходящий класс для иерархической восходящей кластеризации — `AgglomerativeClustering`. [16] В разработанной программе он явно инициализируется следующими параметрами: `n_clusters` и `linkage`. Если первый отвечает за количество обнаружаемых кластеров и задаётся пользователем, то второй указывает, какой тип связи, т. е. какое расстояние между наборами данных будет использоваться при построении матрицы сходства. Наиболее подходящим значением будет `'average'` — среднее расстояние между элементами этих кластеров. Набором же данных для текущего этапа будет набор центроидов, получаемых из атрибута `cluster_centers_` объекта `KMeans`. Визуализация строится аналогично визуализации на предыдущем этапе, однако перед передачей координат точек

функции circle, нужно преобразовать их из типа `float` (число с плавающей запятой) в тип `int` (целое число).

Для нахождения выпуклых оболочек кластеров были использованы два ассоциативных массива. Первый хранит пары «кластер – набор входящих в него точек», а второй «кластер – его выпуклая оболочка». За построения последней отвечал объект `ConvexHull` библиотеки Scipy [17]. Для его инициализации нужен список типа `ndarray` из точек, для которых и строится выпуклая оболочка. Используя возможности OpenCV, а именно функцию `line`, можно построить контуры объектов, снова преобразовав рациональные координаты в целочисленные.

## 1.5 Тестирование отрисовки карты-схемы

Отрисовка карты-схемы по вычисленным координатам является такой же важной частью программы, как и само их вычисление. При виде непонятных линий, не соответствующих чертам объектов в видео, кажется, что возможная ошибка может где угодно, но только не в тривиальной операции прорисовки граней выпуклой оболочки. Именно поэтому важно устранить любое неожиданное поведение функции отрисовки карты-схемы.

Можно выделить следующие классы эквивалентности:

— множества, содержащие меньше, чем 3 точки. В этом случае оболочки будут вырожденными. В случае одной точки — собственно, в точку, двух точек — в отрезок. Более того, построение таких «оболочек» не допускается объектом `ConvexHull`.

— множества, содержащие больше, чем 2 точки и не имеющие внутренних. Здесь оболочка не будет являться вырожденной построить корректно.

— множества, содержащие больше, чем 2 точки и имеющие внутренние точки, не попадающие на границу множества.

Была написана отдельная программа, рисующая выпуклую оболочку заданного множества точек. Её листинг представлен ниже (листинг 1).

Листинг 1 — Программа для тестирования

```
      import cv2 as cv
      import numpy as np
      from scipy.spatial import ConvexHull


      def draw_str(dst, target, s) → None:
          x, y = target
           cv.putText(dst, s, (x+1, y+1), cv.FONT_HERSHEY_PLAIN, 1.0, (0, 0,
0), thickness=2, lineType=cv.LINE_AA)
           cv.putText(dst, s, (x, y), cv.FONT_HERSHEY_PLAIN, 1.0, (255, 255,
255), lineType=cv.LINE_AA)


      zeros_like = cv.imread('1.png')
      colored_features_img = np.zeros_like(zeros_like)


      points1 = np.array([[10, 10]])
      points2 = np.array([[10, 10], [500, 10]])
      points3 = np.array([[15, 15], [505, 15], [505, 505]])
      points4 = np.array([[600, 600], [910, 600], [910, 910], [600, 910]])
      points5 = np.array([[25, 600], [515, 600], [515, 900], [25, 900], [245,
745]])
      points = [points1, points2, points3, points4, points5]


      for array in points:
          for point in array:
               cv.circle(colored_features_img, (point[0], point[1]), 0, (255,
255, 255), 10)
                    draw_str(colored_features_img, (point[0]+5, point[1]+5),
f'{point[0]};{point[1]}')


          if array.shape[0] > 2:
              hull = ConvexHull(array)
              for simplex in hull.simplices:
                  x1 = int(hull.points[simplex[0]][0])
                  y1 = int(hull.points[simplex[0]][1])
                  x2 = int(hull.points[simplex[1]][0])
                  y2 = int(hull.points[simplex[1]][1])
```

```
                    cv.line(colored_features_img, (x1, y1), (x2, y2), (0, 255,
0))
      cv.imwrite('1.png', colored_features_img)
```

Рисунок 7 представляет собой получившееся изображение. Проверка количества точек множества предотвращает ошибку программы во время построения оболочки, а взятие координат вершин оболочки через обращение к объекту позволяет в случае задания двух разных контуров в одном наборе данных корректно её изобразить.

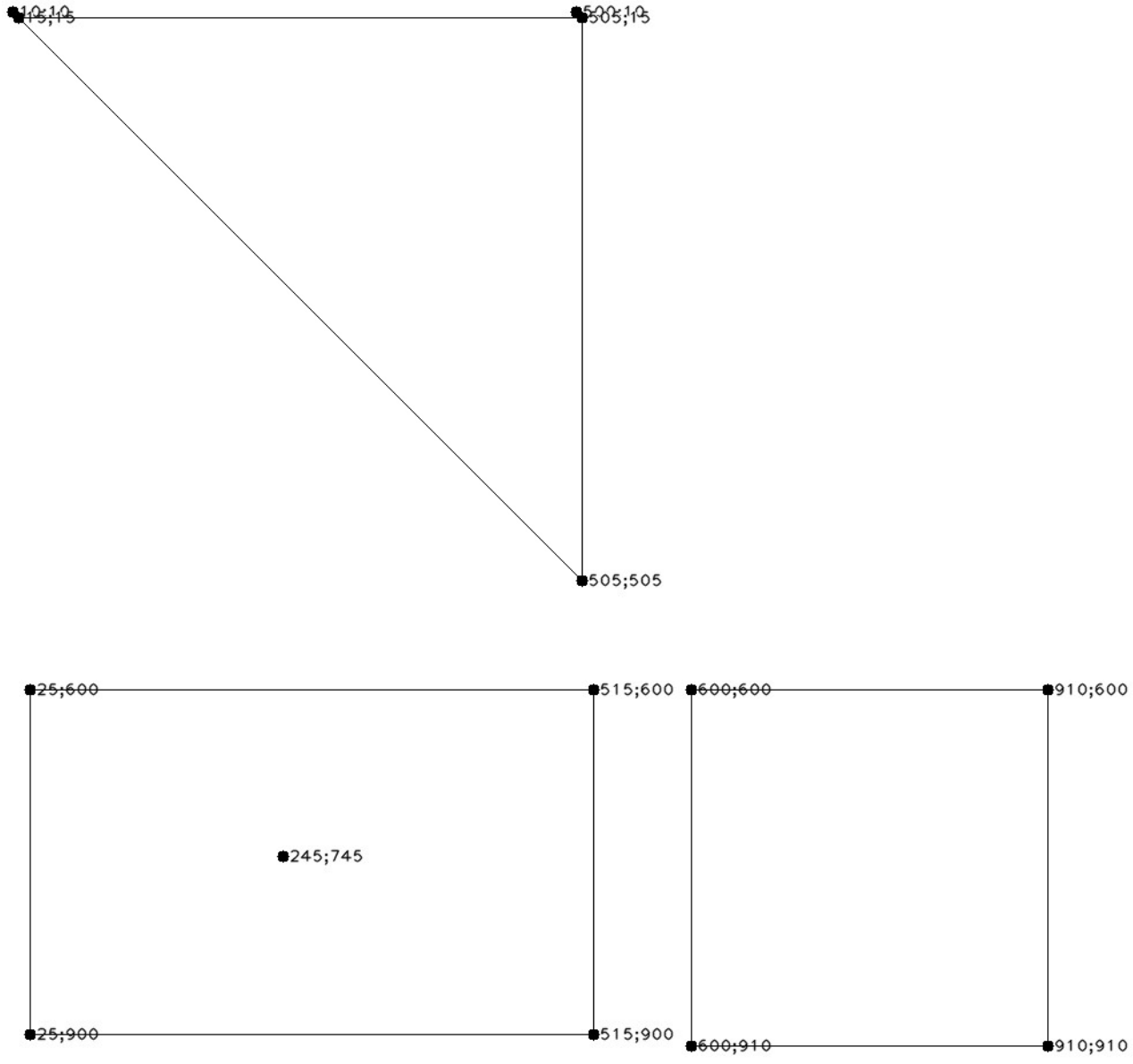



Рисунок 7 — Результаты тестирования

## 1.6 Взаимодействие с программой

Взаимодействие с программой осуществляется через графический пользовательский интерфейс, созданный с помощью библиотеки Tkinter [18]. Единственное окно представлено на рисунке 8.

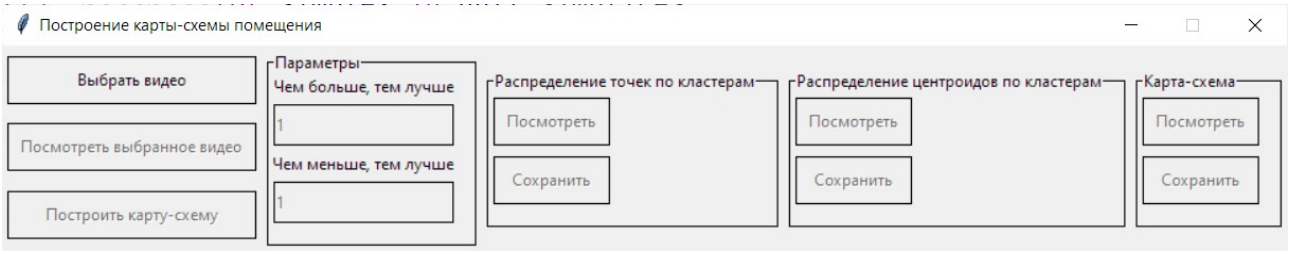



Рисунок 8 — Пользовательский интерфейс

Окно можно поделить на 5 частей. В самой левой находятся кнопки для загрузки видео в программу, его просмотра и, собственно, построение по видео карты-схемы. Справа находится раздел с двумя параметрами, которые влияют на степень детализации и целостности карты. Первый параметр задаёт количество кластеров для алгоритма $k$-средних++ и должен быть равен хотя бы 2. Второй параметр задаёт количество кластеров для агломеративной кластеризации и должен быть равен хотя бы 1. Далее располагаются кнопки для просмотра и сохранения изображений, полученных последующих этапах анализа точек интереса.

## 1.7 Инструкции для запуска

Для запуска разработанной программы необходимо установить интерпретатор языка Python не ниже версии 3.8. Также необходимо установить через пакетный менеджер `pip` следующие пакеты: `opencv-python`, `numpy`, `scipy`, `scikit-learn`.

## 1.8 Пример работы

Пусть пользователь после запуска программы выбрал некоторую видеозапись с учётом требований пункта 1.3 и значения параметров: мера точности карты — 100, кол-во объектов в видео — 2. После нажатия на кнопку «Построить карту-схему» появляется окно, в котором воспроизводится исходное видео с отмеченными на нём зелёным цветом точками интереса и

которое закрывается автоматически после окончания видео. Пример этого окна изображён на рисунке 9.


Рисунок 9 — Вычисленный оптический поток точек интереса (отмечены зелёным цветом)

Кадр из входного видео представлен на рисунке 10 (а). На рисунке 10 (б) изображено распределение точек интереса из всех кадров входного видео по кластерам (алгоритм $k$-средних). На рисунке 10 (в) изображено распределение найденных центроидов в соответствии иерархического аггломеративного алгоритма. Рисунок 10 (г) — это получившаяся карта-схема объектов из видеозаписи. Как уже было сказано, программа не проверяет наличие на видео именно помещения, поэтому фактически в видео может оказаться всё, что угодно.

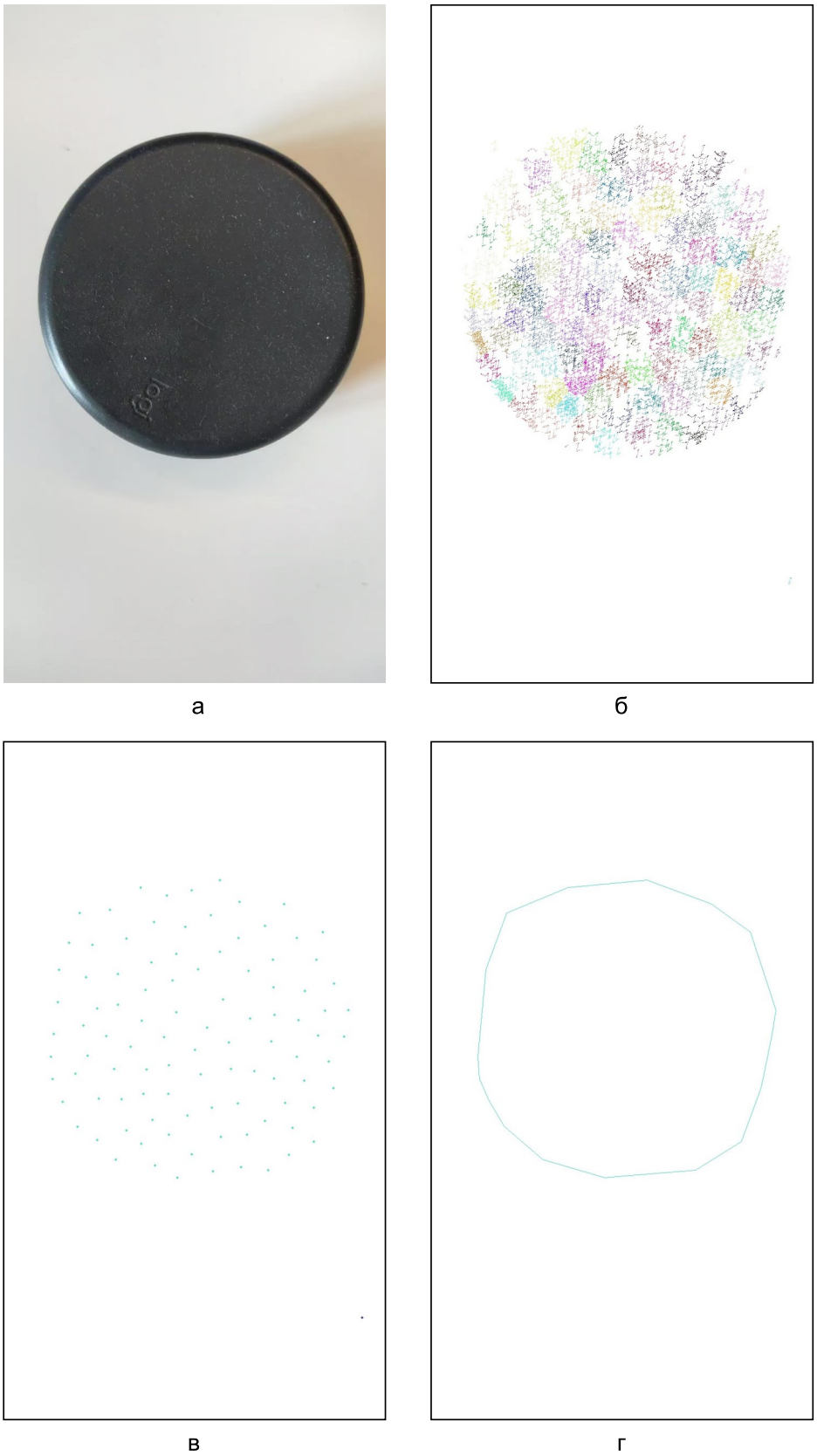



Рисунок 10 — Входное (а) и выходные (б, в, г) изображения

## 1.9 Введение в исследование

В данном разделе проводится исследование эффективности разработанного метода картографирования на основе визуальной одометрии,

также изучено влияние конфигурации, освещённости сцены в видео на строящуюся карту.

Значение результатов этих исследований сложно недооценивать: освещённость сцены и количество объектов на ней имеют большое влияние на её качество (или, правдоподобность). В свою очередь правдоподобность карты-схемы критична в самых разных условиях эксплуатации разработанного метода. К ним относятся, например, оценка качества уборки роботом-пылесосом. Поскольку устройство подобных роботов подходит только для сбора пыли, то, проезжая, по жидкости или веществу вязкой консистенции на полу остаётся след, а нижняя часть робота, включая ходовую, повреждается. Предложенный метод картирования пола, где будет находится робот, позволит своевременно предупредить опасную ситуацию для механизмов автономного аппарата. Также карта может быть полезна в аддитивных технологиях и трёхмерной печати.

## 1.10 Характеристики компьютера

Характеристики компьютера, на котором будет проводиться исследование:

— операционная система: Windows 10 Домашняя для одного языка 22H2, 64-разрядная;

— центральный процессор: Intel® Core™ i7-8550U 1.80ГГц 1.99 ГГц;

— количество ядер: 4;

— количество потоков: 8;

— объём оперативного запоминающего устройства: 8 ГБ.

## 1.11 Характеристики камеры

Характеристики камеры, на которую велась видеосъёмка:

— сенсор: 48 мегапикселей (Sony IMX 586);

— оптическая стабилизация;

— диафрагма: $f/1.7$.

## 1.12 Предмет исследования

Термин «правдоподобность карты» означает оценку того, насколько совпадают выявленные выпуклые контуры кластеров-объектов с реальными очертаниями объектов на видео. Также немаловажным является количество точек интереса, выявленных алгоритмом Лукаса — Канаде на протяжении всего видео. Таким образом, количество точек интереса и степень точности построенных контуров и будут критериями правдоподобности карты-схемы.

## 1.13 Исследование зависимости количества точек интереса от освещённости сцены

В этом исследовании освещённость сцены во входном видео экспертным образом разделена на три уровня: низкая, средняя и высокая. Соответствующие этим уровням кадры показаны на рисунке 11.

а б в

Рисунок 11 — Кадры высокой (а), средней (б) и низкой (в) контрастности (инвертирован)

Рисунок 12 демонстрирует результаты исследования. На видео с высокой освещённостью ожидаемо было найдено больше точек, чем на видео с низкой: более отчётливы контуры объектов и различные особенности их поверхности.

Однако же видео с низкой освещённостью сцены неожиданно показало самое большое количество точек интереса. Это связано не с недостатками алгоритма вычисления оптического потока, а с многочисленными артефактами съёмки в тёмном помещении: зернистость, экспозиция и т. д. Для проверки этого предположения было вручную создано видео с абсолютно чёрным фоном и также проанализировано на предмет точек интереса.

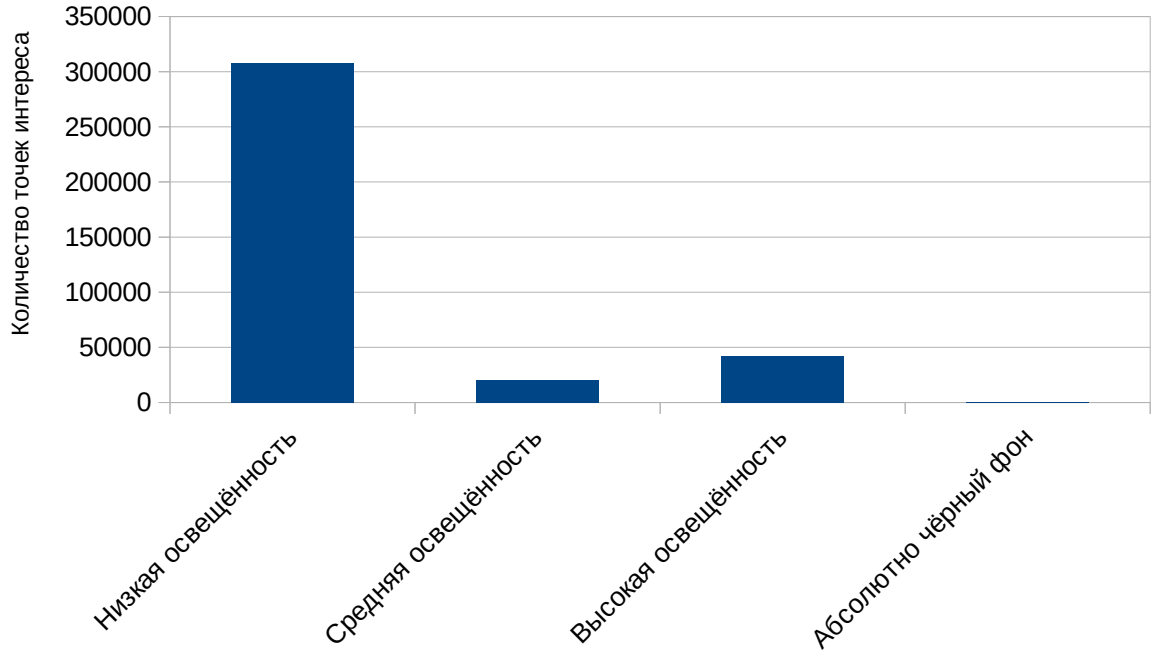



Рисунок 12 — Зависимость количества точек интереса от освещённости сцены видео

## 1.14 Исследование точности карты-схемы от конфигурации сцены

Поскольку назначение разработанного программного обеспечения — построение карты-схемы по видеозаписи и поскольку *точность* зависит от двух выявленных в ходе разработки метода параметров, то было бы целесообразно узнать, насколько значения этих параметров влияют на карту-схему.

Предварительно были выделены несколько классов эквивалентности для сцены входного видео:

— однородная сцена. Сцена представляет собой сплошной фон без выделяющихся объектов. Допустимы различные тени или блики света.

— сцена низкой сложности. На сцене расположены от одного до пяти крупных, хорошо выделяющихся объектов несложной формы (круг, прямоугольник и т. д.).
— сцена средней сложности. В этом случае на сцене расположены один или несколько самопересекающихся контуров объектов, например, скрученные провода.
— сцена высокой сложности. На такой сцене могут быть расположены в большом количестве мелкие объекты как и примитивной, так и неправильной формы: клавиатура, бумага с текстом, цветная узорчатая ткань.

Далее по каждым результатам исследования будет приведено 4 изображения. Рисунок «а» демонстрирует кадр входного видео (оно, согласно рекомендациям из пункта 1.3, статично), «б» — выявленные алгоритмом Лукаса — Канаде точки интереса, «в» — обнаруженные алгоритмом «$k$-средних++» центроиды кластеров, «г» — карта-схема, получающаяся методом вычисления выпуклой оболочки кластеризованных точек из изображения «в».

### 1.14.1 Однородная сцена

Введя минимальные допустимые параметры (мера точности карты: 2, кол-во объектов на видео: 1), получаем следующие изображения (рисунок 13). Из рис. 13 (а) ясно, что сцена представляла из себя пустую поверхность стола белого цвета с малыми неровностями и отблесками света. Как и ожидалось, точек интереса было найдено немного (755), что вместе с заданным количеством кластеров сделало невозможным отрисовку какого-либо контура.

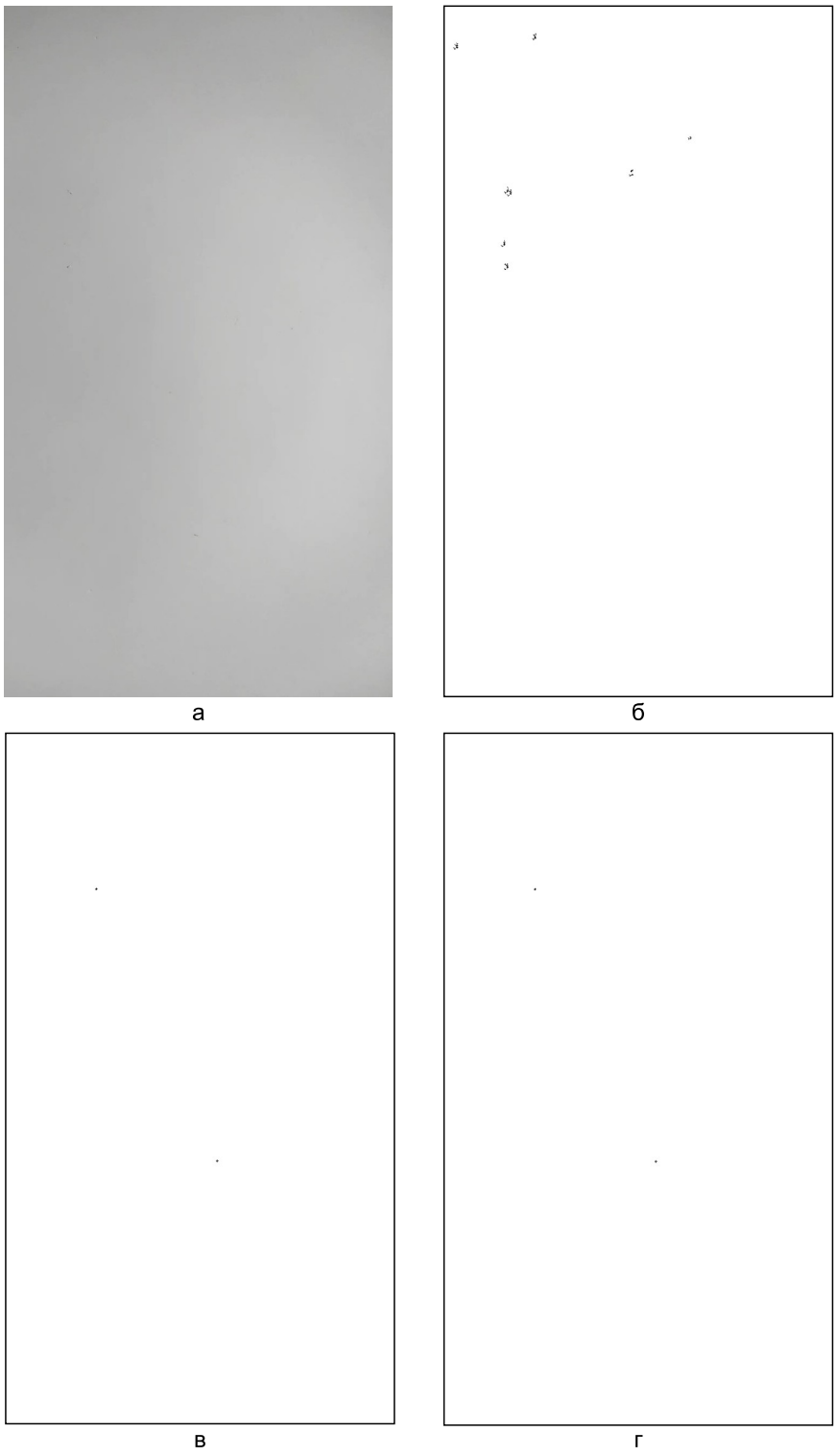
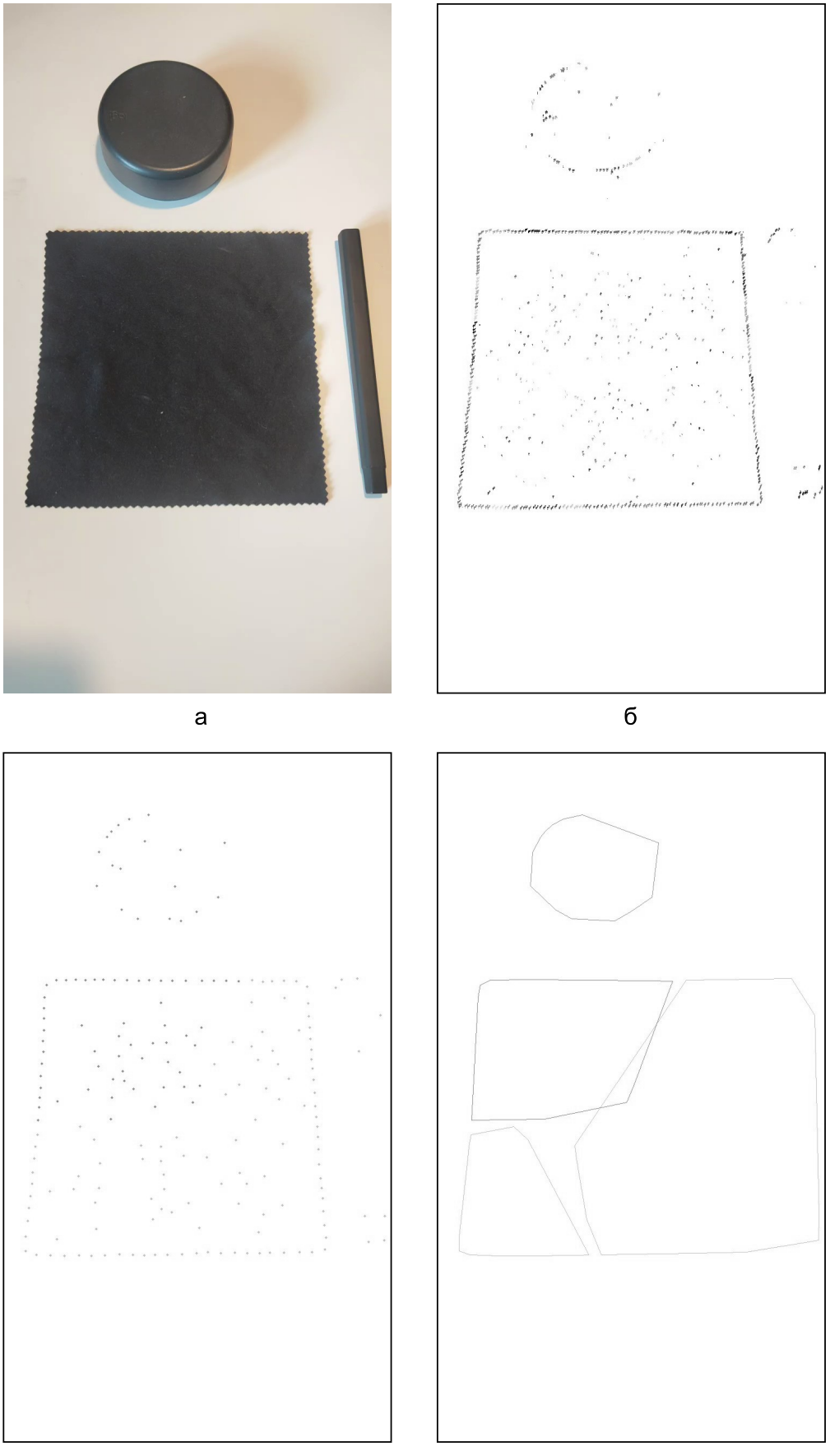



Рисунок 13 — Карта однородной сцены с параметрами 2 и 1

### 1.14.2 Сцена низкой сложности

Сцена низкой сложности, а также получившиеся изображения представлены далее (рисунок 14). Из рис. 14 (а) видно, что на хорошо освещённой сцене расположены три ясно различимых предмета правильной формы. Окончательные значения параметров 200 и 4 были выбраны методом

подбора, так как с ними карта-схемы получается наиболее точной. По итогу карта получилась довольно точной: очертания объектов почти без искажений. Неровность очертаний круга и «поглощение» контура ручки справа можно объяснить отсутствием на их теневых сторонах необходимых точек интереса, ведь в области тени яркость соседних пикселей меняется постепенно, что критично для алгоритма Лукаса — Канаде.



а б в г

Рисунок 15 — Карта сцены низкой сложности с параметрами 200 и 4

### 1.14.3 Сцена средней сложности

Сцена средней сложности предполагает наличие пересекающихся контуров предметов, объектов неконтрастных цветов в отличие от предыдущих испытаний, где использованы только чёрные и белые цвета. Результаты можно увидеть на рисунке 15. Значения, использованные при построении карты в данном исследовании (200 и 50), — яркие примеры того, что задаваемое количество объектов в видео может далеко не соответствовать реальным (очевидно, что объект в реальности один, а не 50). Дело в том, что параметр «количество объектов в видео» на самом деле задаёт количество кластеров, в которые объединяются вычисленные центроиды точек интереса.

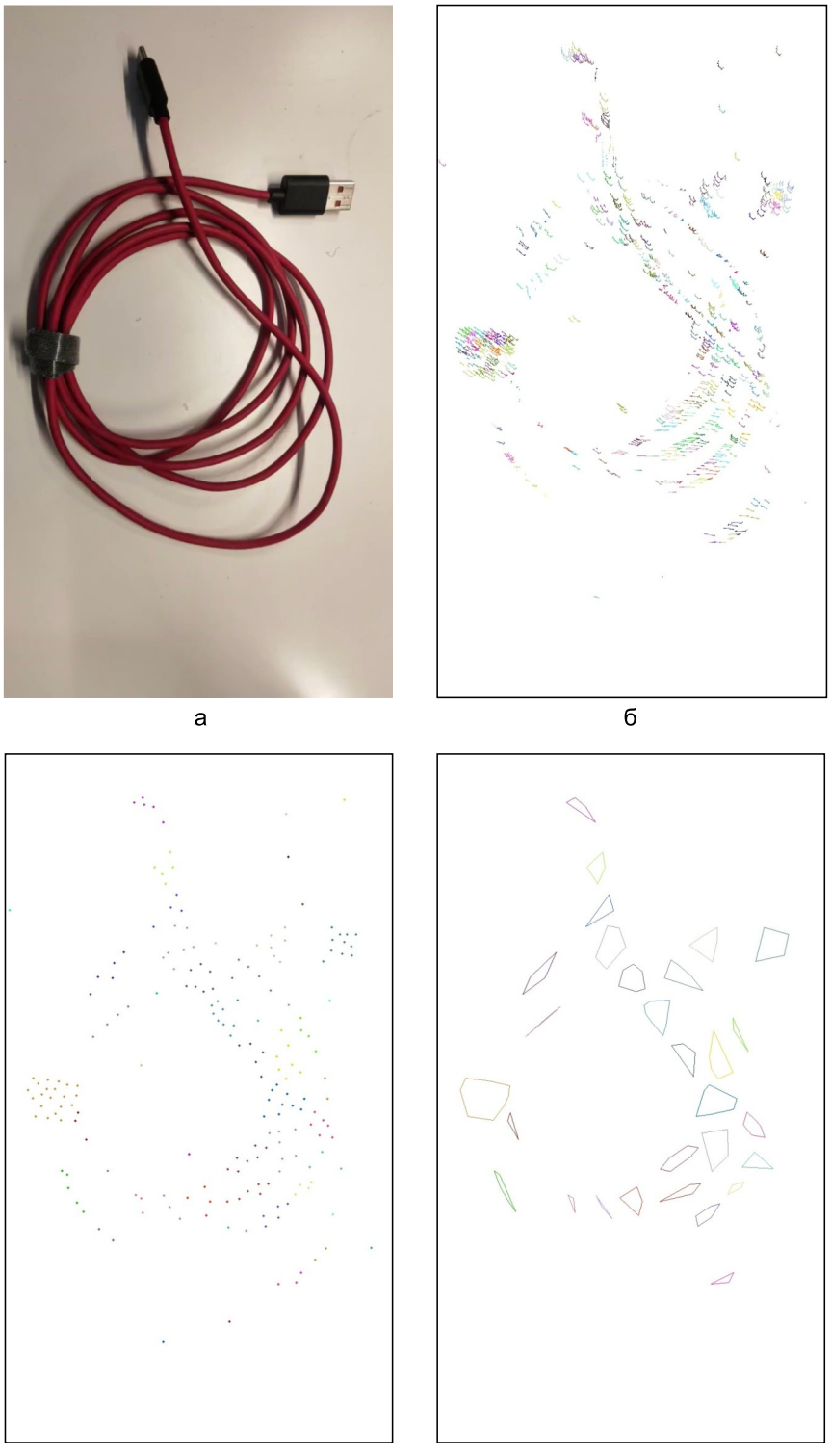



Рисунок 15 — Карта сцены средней сложности с параметрами 200 и 50

### 1.14.4 Сцена высокой сложности

В этом исследовании на видео будет показана клавиатура — объект *сложной* формы с вложенными контурами, т. е. контурами клавиш, иногда с подсветкой. Особую сложность ещё представляют надписи на клавишах — обычно имеющие белый цвет с чёрным фоном. Все эти мелкие детали требуют

тщательного подбора двух входных параметров, чьи значения — как было показано в предыдущем исследовании — не обязательно могут совпадать с реальными. Кадр из входного видео и результаты исследования продемонстрированы на рисунке 16.

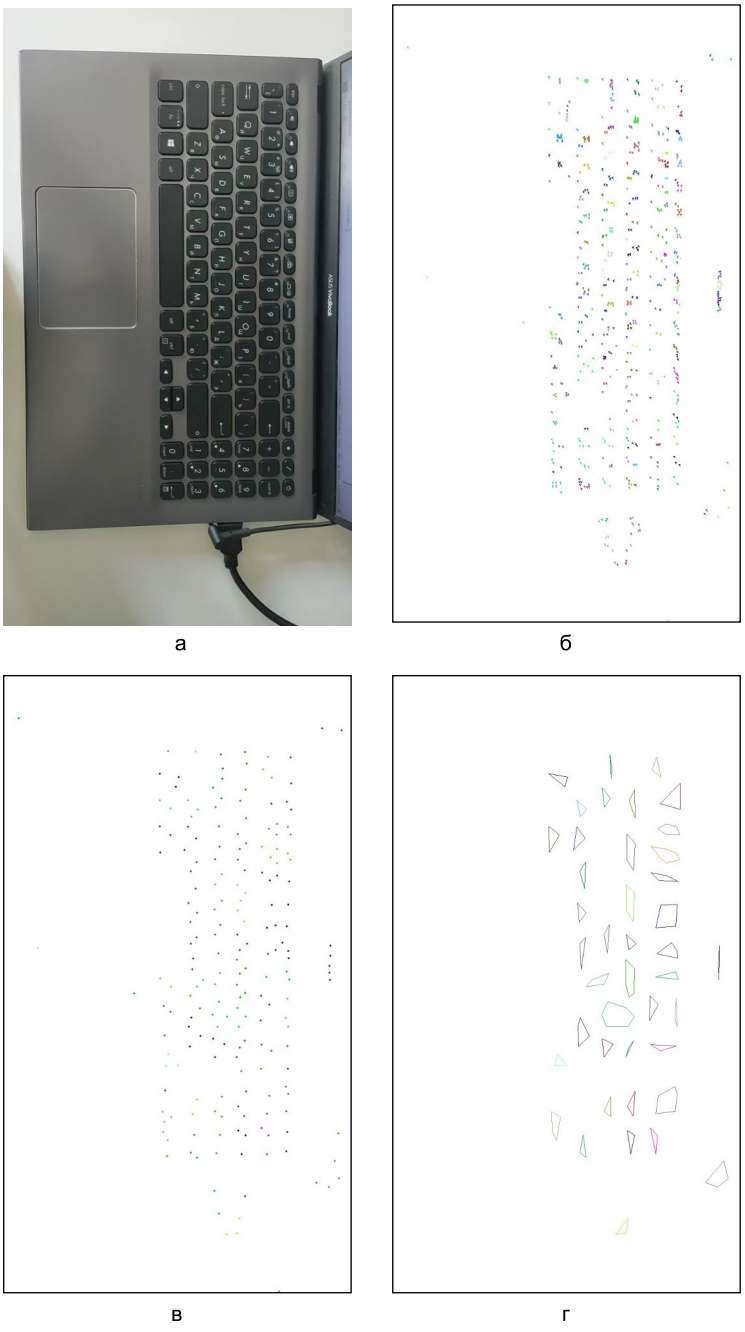



Рисунок 16 — Карта сцены высокой сложности с параметрами 200 и 70

Программа распознала большое количество многоугольных контуров-клавиш, однако общие черты прямоугольного корпуса ноутбука не угадываются: точки интереса по краям корпуса находятся слишком далеко для объединения в один кластер иерархическим алгоритмом.

## 1.15 Выводы из исследования

В разделе были исследованы важные для разработанного метода зависимости: количества точек интереса от освещённости сцены и точности построения карты-схемы от конфигурации сцены и особенностей предметов на ней. В результате было выявлено, что:

- чем выше освещённость сцены, тем больше будет распознано точек интереса (на 48,65%, т. е. почти вполовину), поскольку очертания объектов и их теней чётче, а значит соседние пиксели на их границах будут иметь большую разницу в яркости;
- т. к. идеальных условий освещения не существует в принципе, то точки интереса будут детектироваться в реальных условиях **всегда**, даже в сильно затемнённом помещении. Виной тому неправильно настроенная аппаратура (экспозиция, баланс белого и т. д.);
- чем больше предметов расположено на сцене, чем чаще пересекаются их контуры, тем сложнее добиться правдоподобности карты-схемы: в случае близко расположенных предметов кластеризация их отцентрированных точек интереса невозможна иерархическим алгоритмом, ввиду их близости;
- названия предложенных вводных параметров могут не в полной мере отражать их реальные значения в силу разнообразия подаваемых на вход сцен;
- тем не менее, варьирование требуемой меры точности карты-схемы и количества объектов на сцене позволяют в некоторой степени устранить несоответствия, отмеченные в предыдущем пункте.

# ЗАКЛЮЧЕНИЕ

В соответствии с индивидуальным заданием было разработано программное обеспечение, демонстрирующее практическую осуществимость метода, разработанного в ходе выполнения выпускной квалификационной работы. Даны диаграммы формата IDEF0, описаны требования к ПО, форматы входных и выходных данных, проведено тестирование. Также был продемонстрирован сценарий взаимодействия с программой, приведены инструкции для её запуска.

В дальнейшем было проведено исследование важных для реализованного метода зависимостей: количества точек интереса от освещённости сцены и точности построения карты-схемы от конфигурации сцены и особенностей предметов на ней.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Jianbo Shi and Tomasi, "Good features to track," 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 1994, pp. 593-600, doi: 10.1109/CVPR.1994.323794.
2. Fleet, D., Weiss, Y. (2006). Optical Flow Estimation. In: Paragios, N., Chen, Y., Faugeras, O. (eds) Handbook of Mathematical Models in Computer Vision. Springer, Boston, MA. https://doi.org/10.1007/0-387-28831-7_15
3. Смирнов С.А., Бабаян П.В., Ершов М.Д., Муравьев В.С. РАЗРАБОТКА И ОПТИМИЗАЦИЯ АЛГОРИТМА СЛЕЖЕНИЯ ЗА ТРАНСПОРТНЫМ СРЕДСТВОМ В ВИДЕОПОТОКЕ НА ОСНОВЕ ПИРАМИДАЛЬНОГО МЕТОДА ЛУКАСА - КАНАДЕ // ВК. 2020. №2 (38). URL: https://cyberleninka.ru/article/n/razrabotka-i-optimizatsiya-algoritma-slezheniya-za-transportnym-sredstvom-v-videopotoke-na-osnove-piramidalnogo-metoda-lukasa (дата обращения: 02.05.2023).
4. Половинкин Е. С, Балашов М. В. Элементы выпуклого и сильно выпуклого анализа. — М.: Физматлит, 2004.
5. Greenfield,, Jonathan Scott, "A Proof for a QuickHull Algorithm" (1990). Electrical Engineering and Computer Science - Technical Reports. 65. https://surface.syr.edu/eecs_techreports/65
6. Welcome to Python.org. [Электронный ресурс]. Режим доступа: https://www.python.org/
7. Visual Studio Code - Code Editing. Redefined. [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/
8. OpenCV: cv::VideoCapture Class Reference [Электронный ресурс]. Режим доступа: https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html
9. OpenCV: Color Space Conversions. [Электронный ресурс]. Режим доступа: https://docs.opencv.org/4.x/d8/d01/group__imgproc__color__conversions.html#ga397ae87e1288a81d2363b61574eb8cab

10. OpenCV: Feature Detection [Электронный ресурс]. Режим доступа: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541
11. OpenCV: Object Tracking [Электронный ресурс]. Режим доступа: https://docs.opencv.org/4.x/dc/d6b/group__video__track.html#ga473e4b886d0bcc6b65831eb88ed93323
12. RFC 4180 - Common Format and MIME Type for Comma-Separated Values (CSV) Files [Электронный ресурс]. Режим доступа: https://datatracker.ietf.org/doc/html/rfc4180
13. OpenCV: Drawing Functions [Электронный ресурс]. Режим доступа: https://docs.opencv.org/4.x/d6/d6e/group__imgproc__draw.html
14. numpy.ndarray — NumPy v1.24 Manual [Электронный ресурс]. Режим доступа: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html
15. sklearn.cluster.KMeans — scikit-learn 1.2.2 documentation [Электронный ресурс]. Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans
16. sklearn.cluster.AgglomerativeClustering — scikit-learn 1.2.2 documentation [Электронный ресурс]. Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering
17. scipy.spatial.ConvexHull — SciPy v1.10.1 Manual [Электронный ресурс]. Режим доступа: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html
18. tkinter — Python interface to Tcl/Tk — Python 3.11.3 documentation [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/tkinter.html

# ПРИЛОЖЕНИЕ А