



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

Институт кибернетики
Кафедра проблем управления

ОТЧЁТ ПО ПРАКТИКЕ по ППУ и ОПД

**Тема практики: Детекция типовых объектов с применением
генетического алгоритма**

Отчет представлен к
рассмотрению:

Студент группы
КРБО-01-17

П.Р. Кабанов
«__» _____ 2020 г.

Отчет утвержден
Допущен к защите:

Руководитель практики
от кафедры

С.А.К. Диане
«__» _____ 2020 г.

Руководитель практики
от университета

А. А. Сухоленцева
«__» _____ 2020 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт кибернетики

(наименование института, филиала)

Кафедра проблем управления

(наименование кафедры)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ ПО ППУ И ОПД

(указать вид практики)

Студенту 4 курса учебной группы КРБО-01-17

Кабанову Павлу Романовичу

Место и время практики: лаборатория для самостоятельных занятий кафедры
проблем управления

Должность на практике: -

1. ЦЕЛЕВАЯ УСТАНОВКА: Детекция типовых объектов с применением
генетического алгоритма

2. СОДЕРЖАНИЕ ПРАКТИКИ:

2.1 Изучить: Алгоритмы предобработки изображений и эволюционные
алгоритмы

2.2 Практически выполнить: программно реализовать эволюционный
алгоритм определения параметров типового объекта на изображении

2.3 Ознакомиться с библиотеками обработки изображений в составе языка
программирования Python

3. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ: -

4. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:

Заведующий кафедрой:

«__»____20__г. _____

М.П. Романов

СОГЛАСОВАНО:

Руководитель практики от кафедры:

«__»____20__г. _____

С.А.К. Диане

Руководитель практики от Университета:

«__»____20__г. _____

А.А. Сухоленцева

Задание получил:

«__»____20__г. _____

П.Р. Кабанов

Проведенные инструктажи:

Охрана труда:

Инструктирующий

Подпись


Расшифровка, должность

Инструктируемый

Подпись


Расшифровка

«__»____20__г.
доцент С.А.К. Диане

П.Р. Кабанов

Техника безопасности:


Инструктирующий

Подпись


Расшифровка, должность

Инструктируемый

Подпись


Расшифровка

«__»____20__г.
доцент С.А.К. Диане

П.Р. Кабанов

Пожарная безопасность:

Инструктирующий

Подпись


Расшифровка, должность

Инструктируемый

Подпись


Расшифровка

«__»____20__г.
доцент С.А.К. Диане

П.Р. Кабанов

С правилами внутреннего распорядка ознакомлен:


Подпись

Расшифровка

«__»____20__г.
П.Р. Кабанов



МИНОБРНАУКИ РОССИИ




Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

**РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ
ПРАКТИКИ ПО ППУ И ОПД**

студента 4 курса группы **КРБО-01-17** очной формы обучения, обучающегося
по направлению подготовки «Мехатроника и робототехника», профиль
«Автономные роботы»

Неделя	Сроки выполнения	Этап	Отметка о выполнении
1-4	02.09.20 - 29.09.20	Изучение алгоритмов предобработки изображений и эволюционных алгоритмов	
5-8	30.09.20 - 27.10.20	Программная реализация эволюционного алгоритма определения параметров типового объекта на изображении	
9-16	28.10.20 - 16.12.20	Ознакомление с библиотеками обработки изображений в составе языка программирования Python	

Согласовано:

Заведующий кафедрой

М.П. Романов

Руководитель практики
от кафедры



С.А.К. Диане

Руководитель практики
от Университета

А.А. Сухоленцева

Обучающийся



П.Р. Кабанов

ОТЧЕТ
по преддипломной практике
студента 4 курса учебной группы КРБО-01-17
Института кибернетики

Кабанова Павла Романовича

1. Практику проходил с 02.09.2020г. по 16.12.2020г.

В научно-исследовательской лаборатории кафедры проблем управления
Российского технологического университета

2. Задание на практику выполнил в полном объеме

Не выполнены следующие задания: -

Подробное содержание выполненной на практике работы и достигнутые
результаты: Изучил алгоритмы предобработки изображений и
эволюционных алгоритмов; Программно реализовал эволюционный
алгоритм определения параметров типового объекта на изображении;
Ознакомился с библиотеками обработки изображений в составе языка
программирования Python

Предложения по совершенствованию организации и прохождения практики:
предложений нет

Студент _____ (П.Р. Кабанов)
(подпись)

«__» _____ 2020г.

Заключение руководителя практики от профильной организации:

Приобрел следующие профессиональные навыки:

- способность разрабатывать программное обеспечение, необходимое
для обработки информации и управления в мехатронных и
робототехнических системах, а также для их проектирования
- готовность участвовать в составлении аналитических обзоров и
научно-технических отчетов по результатам выполненной работы, в
подготовке публикаций по результатам исследований и разработок

Проявил себя как: организованный, целеустремленный специалист,
способный самостоятельно решать поставленные перед ним инженерно-
технические задачи и представить отчет по практике в полном объеме и в
установленный срок и заслуживает оценки «отлично».

Руководитель практики от кафедры проблем управления

К.Т.Н., доцент

(подпись)

С.А.К. Диане

Отчет проверил:

Руководитель практики от Университета

А.А. Сухоленцева

Оглавление

1 Введение	7
2 Постановка задачи	7
3 Подготовка изображения	8
4 Эволюционные алгоритмы	13
4.1 Математическая модель особи и определение функции приспособленности ..	16
4.2 Оценка популяции и селекция	17
4.3 Скрещивание и формирование нового поколения	18
4.4 Мутация	19
4.5 Завершение работы алгоритма	19
5 Результаты работы алгоритма	19
6 Вывод	25
7 Список источников	27
8 Приложение	28

1 Введение

Системы технического зрения применяются во многих областях робототехники: начиная от мобильных роботов, заканчивая стационарными промышленными манипуляторами. Одна из основных задач систем машинного зрения – детектирование объектов на изображении. В последнее время для этих целей чаще всего применяют глубокие сверточные нейронные сети, обученные на наборе данных под определенную задачу. Такой подход позволяет быстро классифицировать и детектировать объекты сложных форм, однако, это требует больших вычислительных мощностей. Вместе с этим, обучение модели занимает длительное время, и для качественного результата требуется большой объем обучающего набора с как можно большим количеством возможных случаев, которые могут попасться модели в процессе работы.

2 Постановка задачи

В системах конвейерного снабжения зачастую возникает задача определения габаритных характеристик объектов прямоугольной формы, таких как коробки, печатные платы, отверстия в деталях и т.д., а также существует задача детектирования дверных проемов в работе мобильных роботов. В обоих случаях специфика задачи заключается в неопределенности: известно только то, что нужно найти на изображении объект прямоугольной формы с произвольным соотношением сторон.

В таких условиях использование сверточных нейросетей становится нецелесообразным, так как в данном случае задача сводится к классификации, где одним классом является один вариант соотношения сторон прямоугольника. При условиях, что соотношение сторон в задаче может быть любым, количество классов становится бесконечным. Также при свертке будет недостаточно необходимых признаков для обучения нейросети. В качественно обученных системах к ним, как правило, относятся контуры, цвет, текстура и другие признаки. В целом можно сказать, что решать данную задачу с помощью сверточных нейросетей неоправданно сложно.

В качестве альтернативы рассмотрим использование эволюционных алгоритмов, которые должны находить определенную форму, которую можно описать математической моделью.

3 Подготовка изображения

В основе работы алгоритма лежит поиск прямоугольника среди контуров изображения. Прежде чем начать определять контуры объекта, нужно отфильтровать шумы на изображении. Для этого зачастую применяется размытие по Гауссу, однако в данной работе был применен способ очистки от шумов через билатеральный фильтр.

Основное отличие билатерального фильтра от фильтра Гаусса заключается в том, что веса в ядре свертки зависят не только от расстояния от центра ядра, но и от разницы в цветовом пространстве между пикселями. Это позволяет избавиться от шумов, но при этом сохранить границы и контуры объектов на изображении.



Рисунок 1. Отличие билатерального фильтра от фильтра Гаусса

Дадим определение свертке по функции Гаусса:

$$GC[I]_p = \sum_{q \in S} G_{\sigma}(\|p - q\|) I_q \quad (1)$$

$$G_{\sigma}(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \quad (2)$$

Фильтр Гаусса представляет собой средневзвешенное значение яркостей соседних пикселей в области матрицы ядра с весом, уменьшающимся с расстоянием от центра ядра p . Вес пикселя определяется $G\sigma$ ($\|p - q\|$), L2 нормой матрицы (Евклидовой нормой), которая представляет собой геометрическое расстояние между точками в пространстве, где σ – параметр, определяющий размер окрестности. Сила этого влияния зависит только от пространственного расстояния между пикселями, но не от их значений. Так, яркий пиксель имеет сильное влияние на соседний темный пиксель, хотя значения яркостей этих пикселей сильно отличаются. В результате, края изображения размываются, так как пиксели на неоднородных участках усредняются все вместе.

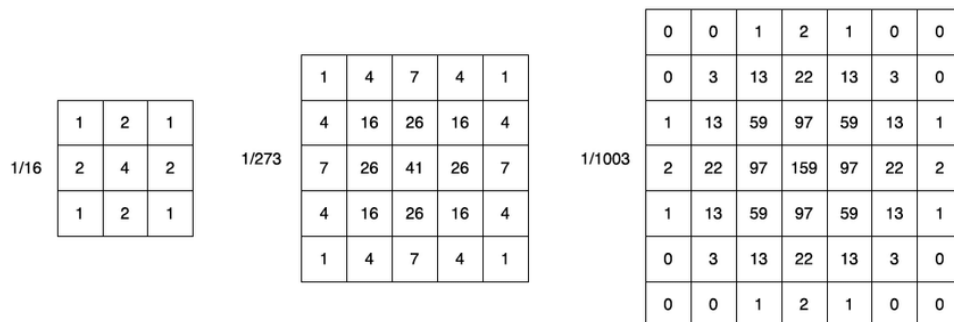


Рисунок 2. Примеры ядер фильтра Гаусса с размерами 3x3, 5x5 и 7x7

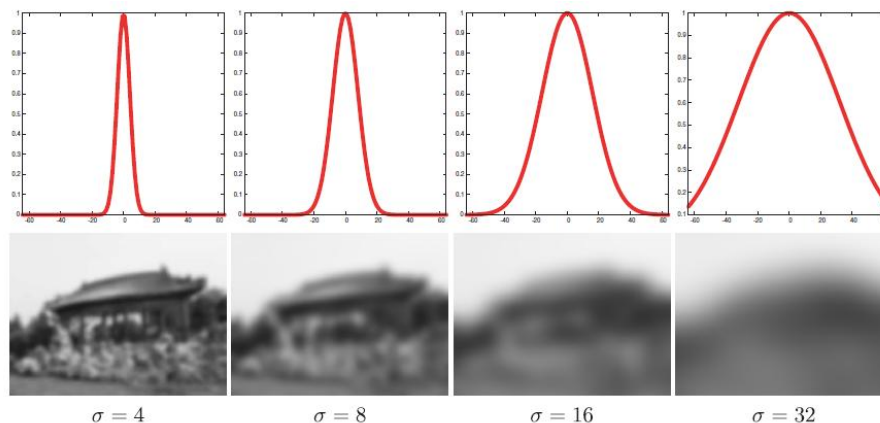


Рисунок 3. Пример Гауссова размытия при разных размерах ядра

Билатеральный фильтр берет за основу средневзвешенное пикселей в окрестности, как при свертке по Гауссу, однако учитывает значение яркостей между

соседями, вследствие чего изменяются веса в свертке, что позволяет защитить границы и края объектов от размытия. Ключевая идея заключается в том, что для влияния одного пикселя на другой, они не только должны находиться рядом друг с другом, но и иметь схожее значение яркости.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{q_s}(\|p - q\|) G_{q_r}(|I_p - I_q|) I_q \quad (3),$$

где фактор нормализации W_p

$$W_p = \sum_{q \in S} G_{q_s}(\|p - q\|) G_{q_r}(|I_p - I_q|) \quad (4)$$

Параметры σ_s и σ_r определяют степень фильтрации изображения I . Уравнение (3) – нормализованное среднее взвешенное, где G_{σ_s} – пространственное взвешивание, как в фильтре Гаусса, уменьшающееся с удалением от центра ядра, G_{σ_r} – яркостное взвешивание, уменьшающее влияние пикселей q , когда их яркость отличается от I_p . Рис. 4 иллюстрирует изменение весов для пикселя около края.

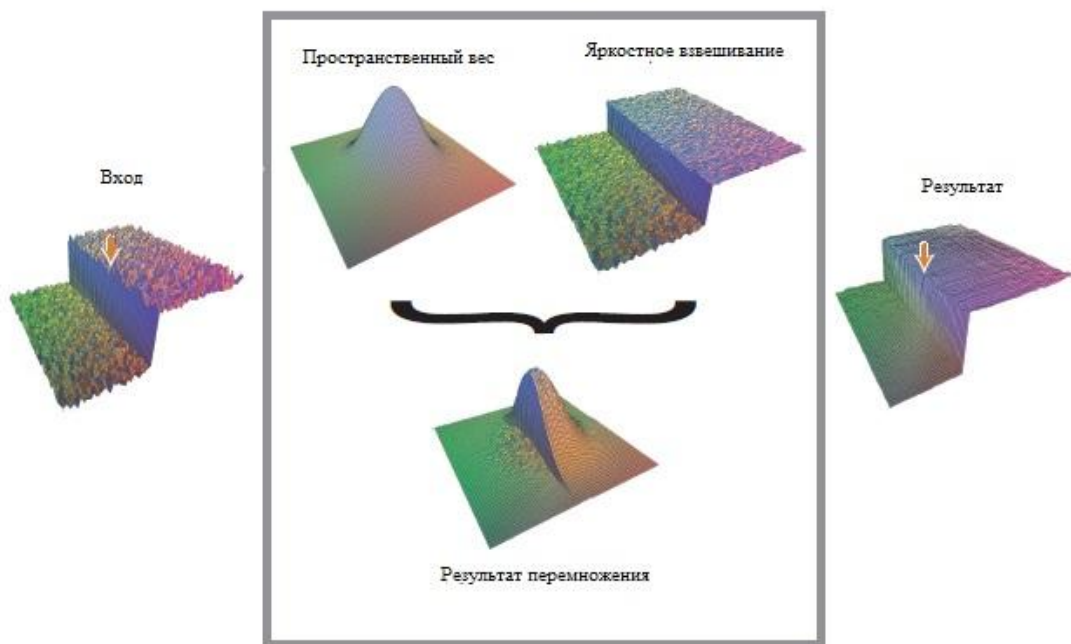


Рисунок 4. Билатеральное размытие возле края объекта

После фильтрации изображения следует перейти к выделению границ объекта. Для этого используется оператор Кэнни.

Алгоритм можно разбить на 5 этапов:

1. Сглаживание изображения
2. Поиск градиентов. Границы отмечаются там, где градиент изображения приобретает максимальное значение
3. Подавление не-максимумов. Только локальные максимумы отмечаются как границы.
4. Двойная пороговая фильтрация. Потенциальные границы определяются порогами.
5. Трассировка области неоднозначности. Итоговые границы определяются путем подавления всех краев, несвязанных с сильными границами.

Первый пункт был выполнен с помощью билатерального фильтра. Поиск градиента осуществляется следующим образом. Границы отмечаются там, где градиент изображения приобретает максимальное значение. Они могут иметь различное направление, поэтому алгоритм Кэнни использует четыре фильтра для обнаружения горизонтальных, вертикальных и диагональных ребер в размытом изображении.

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctg\left(\frac{G_y}{G_x}\right)$$

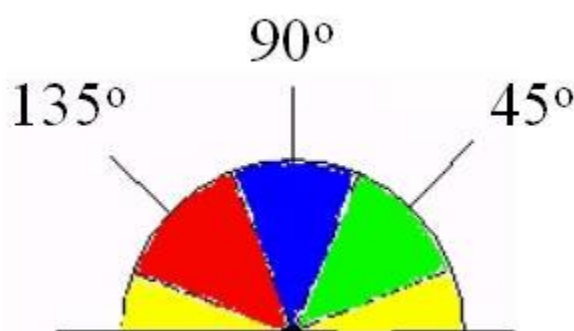


Рисунок 5. Определение направления градиента

Угол направления вектора градиента округляется и может принимать значения 0, 45, 90 и 135 градусов. Направление края, попадающего в каждую цветовую область будет установлено на определенное значение угла.

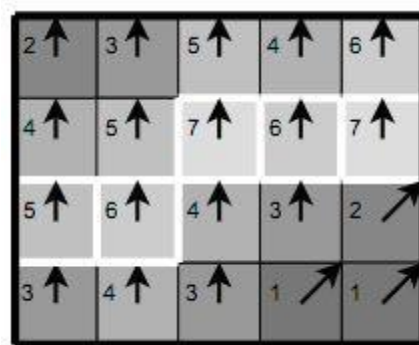


Рисунок 6. Визуализация градиента

Подавление не-максимумов — техника уменьшения толщины границ. Применяется для нахождения областей с резким изменением яркости. Алгоритм для каждого пикселя в градиентах:

1. Сравнение силы границ текущего пикселя с силой границ пикселей в положительных и отрицательных направлениях градиента
2. Если сила границ текущего пикселя больше значения в маске с пикселями с таким же направлением градиента, значение будет сохранено, в противном случае — подавлено.

Сила границ пикселя — производная первого порядка, например, величина градиента.

После подавления не-максимумов, оставшиеся пиксели дают более точное представление границ на изображении. Однако некоторые пиксели остаются из-за шума и цветового разброса. Чтобы отбросить эти выбросы, важно отфильтровать пиксели со слабым значением градиента и сохранить пиксели с сильным значением градиента. Для этого выбираются верхнее и нижнее пороговые значения. Если значение градиента краевого пикселя выше верхнего порогового значения, он помечается как сильный краевой пиксель. Если значение градиента краевого пикселя меньше верхнего порогового значения и больше нижнего значения, он помечается как слабый. Если значение меньше нижнего порога, оно будет подавлено. Пороговые значения определяются эмпирически и будут зависеть от содержимого изображения.

Пиксели с сильными границами обязательно будут задействованы в конечном результате, поскольку они извлекаются из истинных границ изображения. Однако, возникает спорный момент с пикселями со слабыми границами, так как они могут быть извлечены и из истинных границ, и из шума. Для достижения более точного результата следует избавиться от пикселей вызванных шумом. Обычно пиксель со слабым значением, полученный от истинных границ будет связан с пикселем с сильным значением, в то время как шумовые характеристики ни с чем не связаны. Чтобы отследить связь, рассматривают слабый пиксель и соединенные с ним соседние 8 пикселей. Если есть хотя бы один сильный пиксель в этом квадрате, слабый пиксель можно сохранить.

В результате работы всех вышеперечисленных методов можно получить четкие контуры изображения.

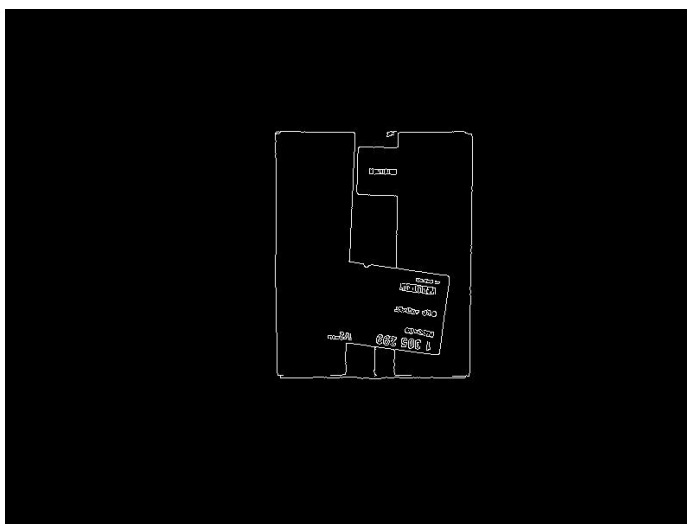


Рисунок 7. Контуры изображения

4 Эволюционные алгоритмы

Эволюционные алгоритмы – направление в искусственном интеллекте, которое использует и моделирует процессы естественного отбора. В данной работе с помощью генетического алгоритма будет осуществляться поиск прямоугольного объекта на изображении.

Генетический алгоритм – эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем случайного подбора,

комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе. На Рис. 8 представлена блок-схема алгоритма.



Рисунок 8. Блок-схема генетического алгоритма

4.1 Математическая модель особи и определение функции приспособленности

Зададим прямоугольник на плоскости следующими параметрами:

x, y – координаты левого нижнего угла прямоугольника,

w – ширина,

h – высота,

θ – угол поворота прямоугольника относительно его центра.

Для корректной работы алгоритма необходимо как-то оценивать прямоугольники. Для этого введем понятие функции приспособленности.

Функция приспособленности – вещественная функция, подлежащая оптимизации в результате работы генетического алгоритма.

Так как в настоящей задаче используется поиск среди контуров изображения, зададим прямоугольнику окрестность, в которой будет осуществляться анализ попавших в нее пикселей. В случае, когда нужно чтобы в окрестность попало наибольшее количество точек контура коробки, можно обратиться к алгоритму RANSAC (Random Sample Acquaintance). Этот алгоритм позволяет путем случайно выбранных точек аппроксимировать модель, качественно отфильтровав выбросы. Основной идеей является поиск такой модели, чтобы в окрестность функции попадало как можно большее количество точек из общего облака. Поэтому зададим функцию приспособленности уравнением:

$$f = \frac{N_{inliers}}{N_{outliers} + 1},$$

где $N_{inliers}$ – количество точек попавших в окрестность, $N_{outliers}$ – число точек, не попавших в нее, равно N всех точек на изображении – $N_{inliers}$. В результате работы алгоритма значение функции должно стремиться к максимальному значению.

После определения модели и функции можно перейти к инициализации начальной популяции размера S . Параметры прямоугольника выбираются случайным образом.

4.2 Оценка популяции и селекция

После инициализации каждую особь нужно оценить по определенной выше функции приспособленности. В результате нужно составить таблицу соответствий особи и ее значением. После этого нужно провести селекцию. Существует множество методов селекции, перечислим самые распространенные:

1) Рулетка – вероятностный способ отбора. По результатам оценки популяции, для каждой особи вычисляется значение вероятности попадания в следующее поколение.

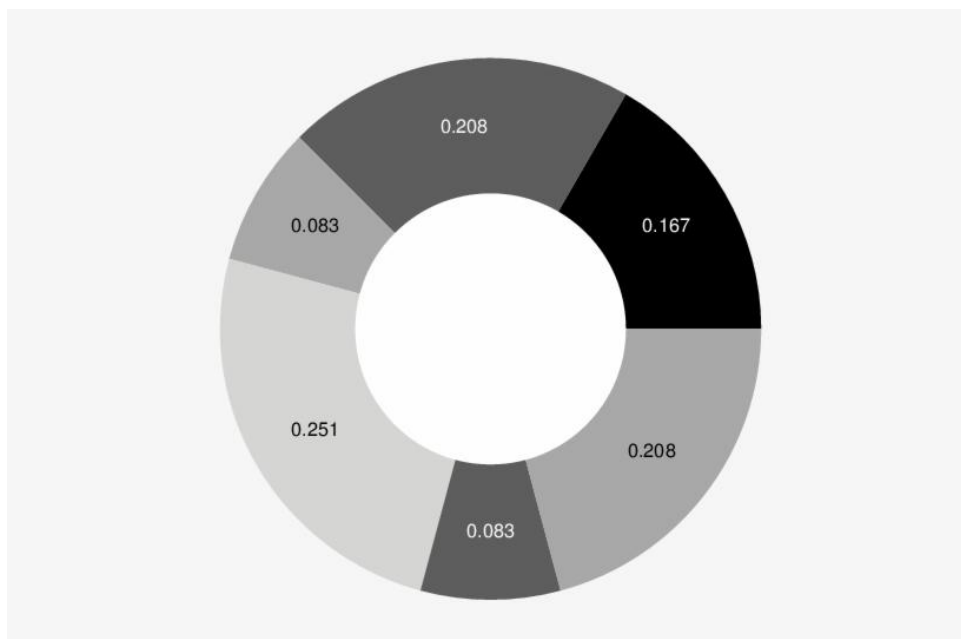


Рисунок 9. Вероятности пропорциональны значениям функции приспособленности

После этого для каждого члена популяции случайно выбирается число от 0 до 1. Сравнивая полученные значения с результатами кумулятивной суммы всех вероятностей, переставляется порядок особей в поколении, так, чтобы самыми первыми были те, кто попал в диапазон между первыми двумя элементами массива из кумулятивных сумм.

Такой метод позволяет разнообразить популяцию, что впоследствии приводит к большему числу вероятных решений и возможностям для мутации.

2) Отбор по рангу – метод, когда происходит сортировка особей по убыванию их значения приспособленности. Такой способ позволяет быстро оценить всю

популяцию и не потерять ценные особи, однако может привести к малой вариативности возможных гипотез.

3) Турнирная селекция – особи разбиваются на «команды». Победитель состязаний из каждой «команды» попадает в следующее поколение.

4) Элитизм - Зачастую для получения наилучших параметров используются стратегии с частичным воспроизведением. Небольшая часть лучших особей из последнего поколения без каких-либо изменений передается следующему.

В данной работе был использован метод отбора по рангу. После сортировки особей, вторая половина удаляется. Их место займут потомки.

4.3 Скрещивание и формирование нового поколения

Размножение в генетических алгоритмах требует для производства потомка нескольких родителей, обычно двух.

Можно выделить несколько операторов выбора родителей:

1. Панмиксия – оба родителя выбираются случайно, каждая особь популяции имеет равные шансы быть выбранной
2. Инбридинг – первый родитель выбирается случайно, а вторым выбирается такой, который наиболее похож на первого родителя
3. Аутбридинг – первый родитель выбирается случайно, а вторым выбирается такой, который наименее похож на первого родителя

Инбридинг и аутбридинг бывают в двух формах: фенотипной и генотипной. В случае фенотипной формы похожесть измеряется в зависимости от значения функции приспособленности (чем ближе значения целевой функции, тем особи более похожи), а в случае генотипной формы похожесть измеряется в зависимости от представления генотипа (чем меньше отличий между генотипами особей, тем особи сильнее похожи). Также существует несколько вариантов вероятностного скрещивания.

В работе был использован метод панмиксии. Скрещивание осуществляется путем сложения параметров родителей, но каждое слагаемое умножается на случайно сгенерированный весовой коэффициент в диапазоне от 0 до 1.

После того как скрещивание завершилось, потомки добавляются в конец списка, восполняя собой удаленные ранее особи.

4.4 Мутация

В генетическом алгоритме мутация – вероятностный процесс. Как правило, частота и сила мутаций определяется специальным параметром, который выбирает n -е количество параметров из параметров всех особей и случайным образом меняет их. Такой способ позволяет регулировать степень изменения генофонда популяции и изменить ее для получения лучшего результата. В данной работе мутация осуществлялась для каждой особи, чтобы таким образом ускорить поиск подходящего решения и компенсировать малую вариативность при селекции.

4.5 Завершение работы алгоритма

После окончания мутаций, особи в популяции перемешиваются случайным образом и цикл начинается заново. Чтобы не потерять хорошие гипотезы, нужно сохранять особи с самым высоким значением функции приспособленности и обновлять каждый раз, как только будет найдена новая гипотеза с лучшим значением.

5 Результаты работы алгоритма

На Рис 10-16 показаны результаты работы программы. Желтый прямоугольник – модель, красные вокруг него – окрестность. Тестирование осуществлялось при 20 особях и 300 шагах цикла.

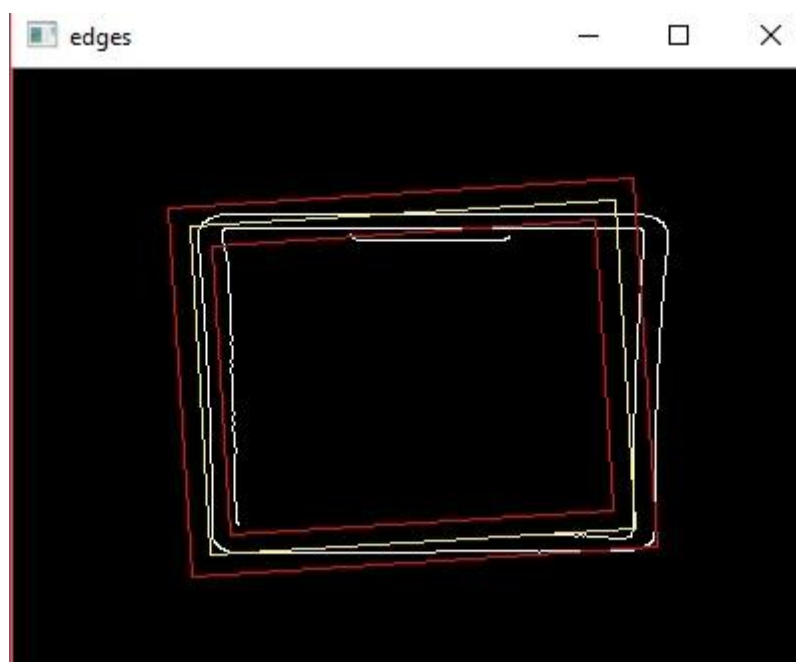


Рисунок 10

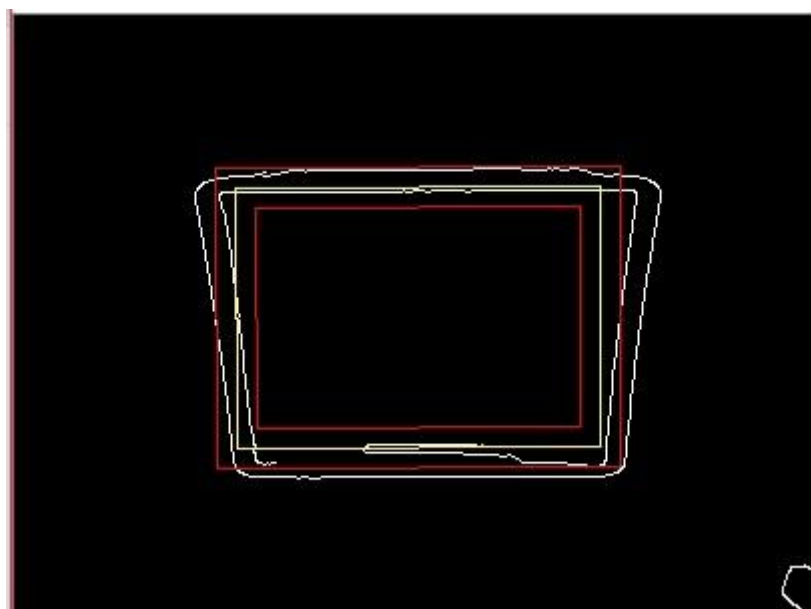


Рисунок 11

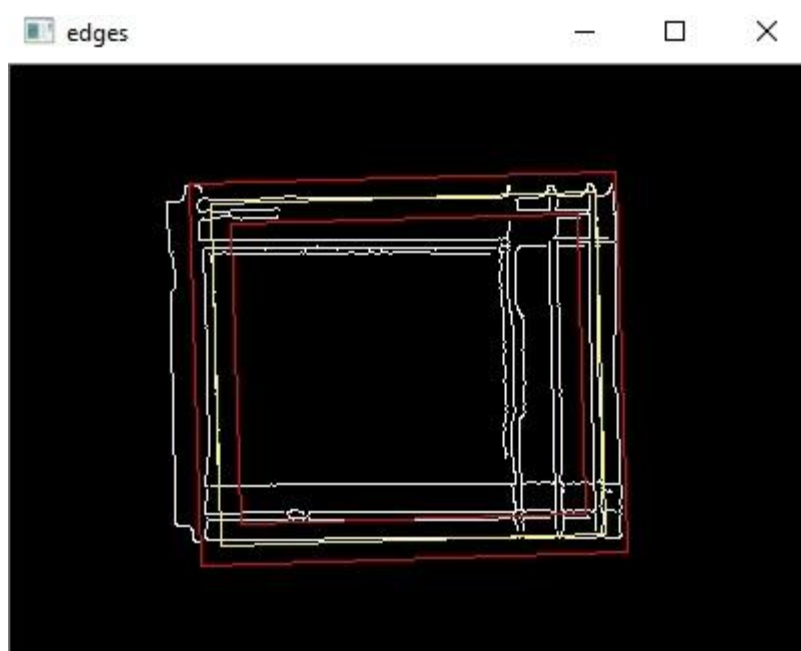


Рисунок 12

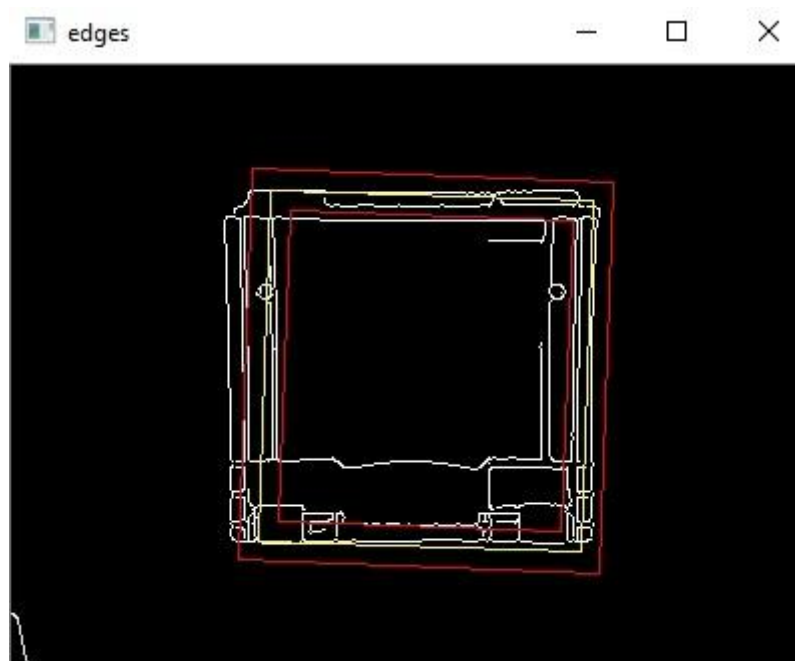


Рисунок 13

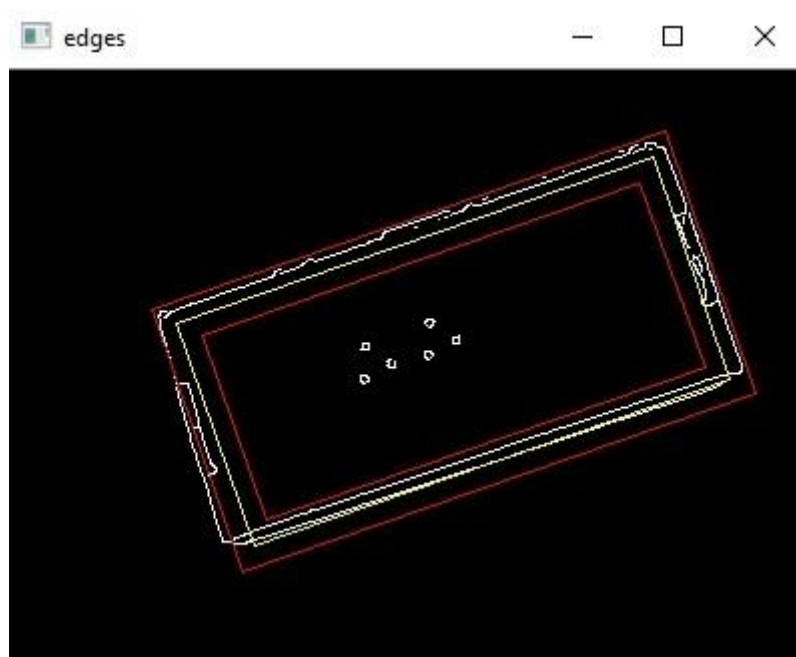


Рисунок 14

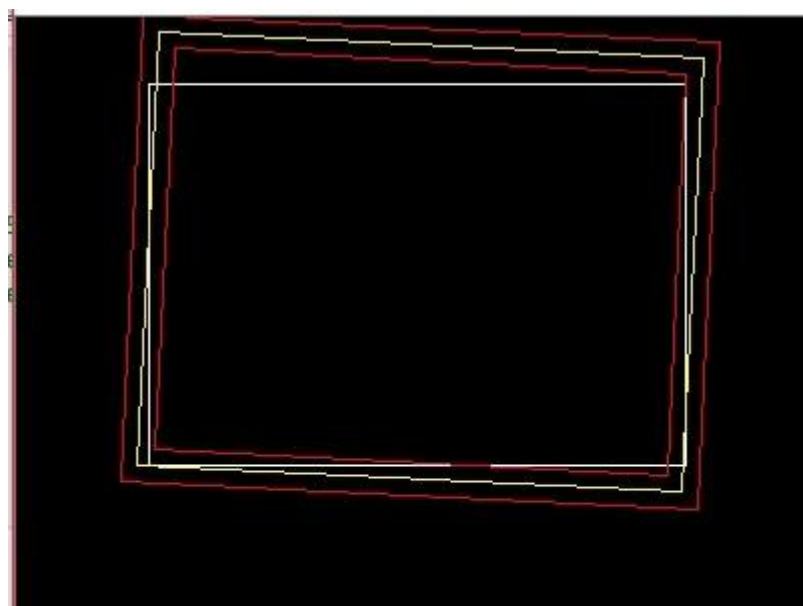


Рисунок 15

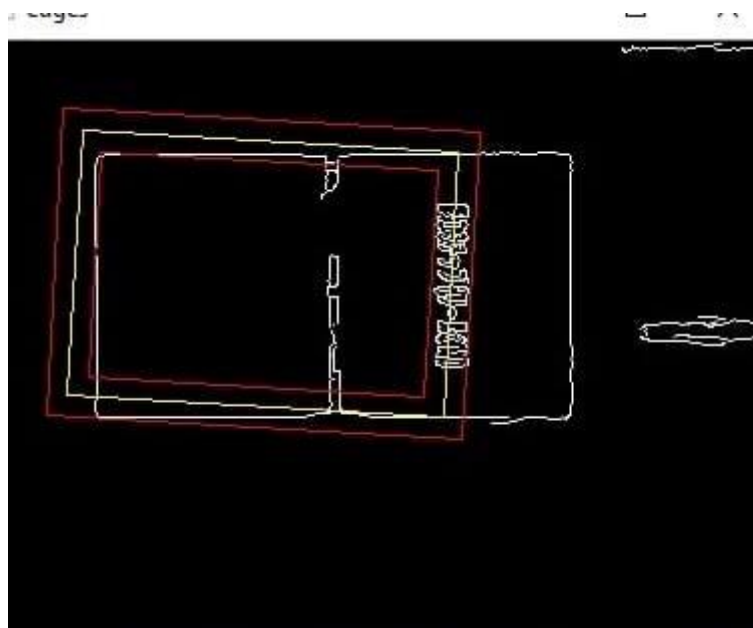


Рисунок 16

Видно, что в большинстве случаев алгоритму удастся определить область, где находится коробка. Однако, на Рис.16 изображен плохой случай, когда в окрестность прямоугольника попали буквы, которые не удалось удалить фильтром Кэнни. Их контуры состоят из большего количества пикселей, чем те, что осталось обвести.

Также была попытка определить дверной проем на изображении.



Рисунок 17

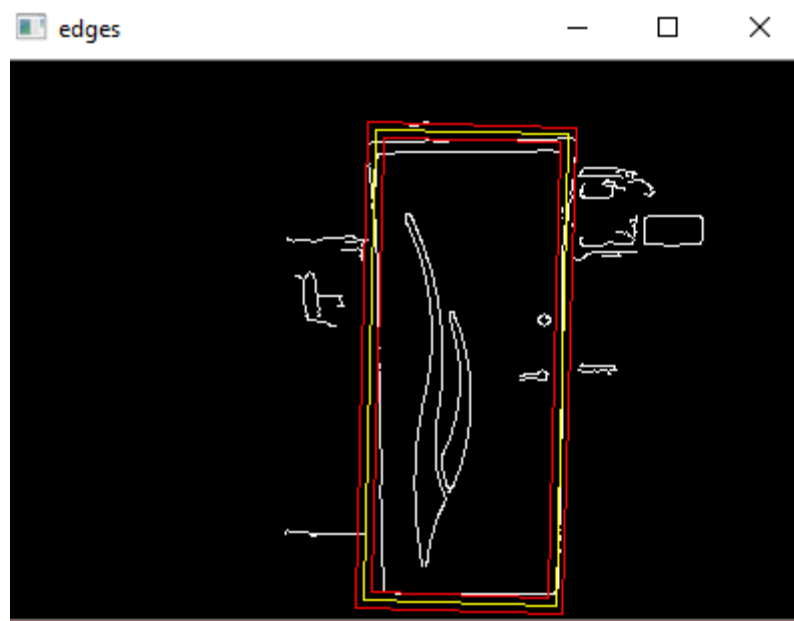


Рисунок 18

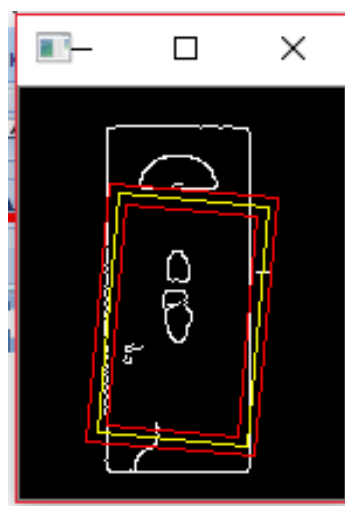


Рисунок 19

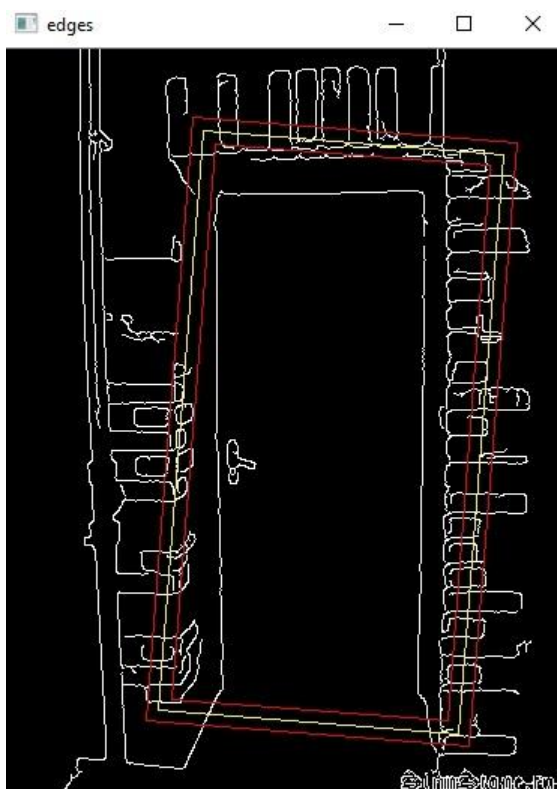


Рисунок 20

Как видно по изображениям, алгоритм относительно неплохо работает, когда в кадре нет контуров кроме искомого объекта, либо их суммарный периметр меньше периметра контура искомого объекта.

6 Вывод

Написанная программа показала неплохие результаты при детектировании коробок. С каждым новым запуском программы также удастся обнаружить коробку, то есть присутствует повторяемость эксперимента. Однако, в дальнейшем можно улучшить работу алгоритма несколькими способами.

Во-первых, следует изменить функцию оценки пригодности. Можно опробовать функцию, которая оценивает непрерывность линии, которую составляют пиксели, попавшие в окрестность прямоугольника. Это поможет избавиться от гипотез, которые вместо одной непрерывной линии вдоль прямоугольника накрывают множество прерывистых линий, расположенных поперек.

Во-вторых, стоит переработать сам алгоритм. В частности из-за малой степени случайности, популяции зачастую получаются недостаточно разнообразными, и через какое-то время все гипотезы сводятся к приблизительно одной. Чтобы отсрочить это время, в первую очередь стоит заменить функцию селекции на более случайную. Затем при скрещивании также стоит внести долю случайности. Вместо того, чтобы скрещивать все особи, оставшиеся в поколении, можно задать коэффициент частоты скрещивания числом от 0 до 1 и для каждой особи генерировать случайное число в этом диапазоне. Те особи, у которых значение превысило коэффициент, будут скрещиваться между собой, остальные нет. В самом скрещивании параметров также стоит избегать каких-либо фиксированных значений. Стоит случайно выбирать параметры, которые мы хотим скрестить. И скрещивание также может быть разным: в данном случае использовалась сумма параметров помноженных на весовые коэффициенты, однако можно использовать и обмен параметрами.

В-третьих, процесс мутации также нужно сделать настраиваемым, чтобы можно было эмпирически выбрать оптимальное значение степени мутации. В разделе «Мутация» подробнее описаны возможные альтернативы.

Улучшив генетический алгоритм, можно в перспективе перейти к поиску трехмерных форм-примитивов в облаке точек. Это может позволить системам ИИ «понимать» из каких форм-примитивов состоит тот или иной объект, а не воспринимать его как непрерывную поверхность. Такой подход может позволить более эффективно производить классификацию трехмерных объектов при обучении без учителя, так, чтобы система могла самостоятельно обучаться «рассматривая» новые объекты и добавляя новые признаки в виде составных форм.

7 Список источников

1. D. Hermawanto Genetic Algorithm for Solving Simple Mathematical Equality Problem // Indonesian Institute of Sciences (LIPI) – 2013.
2. N. Saini Review of Selection Methods in Genetic Algorithms // International Journal Of Engineering And Computer Science – 2013. – №6
3. S. Paris, P. Kornprobst, J. Tumblin, F. Durand Bilateral Filtering: Theory and Applications // Foundations and Trends in Computer Graphics and Vision – 2008. – №4
4. Genetic algorithm [Электронный ресурс]: – Электрон. текстовые дан. – 2020г. – Режим доступа: https://en.wikipedia.org/wiki/Genetic_algorithm
5. Canny edge detector [Электронный ресурс]: – Электрон. текстовые дан. – 2020г. – Режим доступа: https://en.wikipedia.org/wiki/Canny_edge_detector

8 Приложение

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import sys
import random

# Фильтрация изображения
def filter_image(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blurred = cv2.bilateralFilter(gray,6,75,75)
    edges = cv2.Canny(blurred,0,150)
    edges = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    return edges

# Класс прямоугольника
class rectangle:
    def __init__(self, x, y, w, h, theta):
        thresh = 4
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.theta = theta
        s = np.sin(theta)
        c = np.cos(theta)
        centerx = x + w/2
        centery = y + h/2
        self.x1 = x
        self.y1 = y+h
        self.x2 = x+w
        self.y2 = y+h
        self.x3 = x+w
        self.y3 = y
        x0r = centerx + (self.x - centerx)*c - (self.y - centery)*s
```

```

y0r = centery + (self.y - centery)*c + (self.x - centerx)*s
x1r = centerx + (self.x1 - centerx)*c - (self.y1 - centery)*s
y1r = centery + (self.y1 - centery)*c + (self.x1 - centerx)*s
x2r = centerx + (self.x2 - centerx)*c - (self.y2 - centery)*s
y2r = centery + (self.y2 - centery)*c + (self.x2 - centerx)*s
x3r = centerx + (self.x3 - centerx)*c - (self.y3 - centery)*s
y3r = centery + (self.y3 - centery)*c + (self.x3 - centerx)*s
self.pts = np.array([[x0r,y0r],[x1r,y1r],[x2r,y2r],[x3r,y3r]], np.int32)

```

```

# Внешний четырехугольник

```

```

out_x0 = self.x - thresh
out_y0 = self.y - thresh
out_x1 = self.x1 - thresh
out_y1 = self.y1 + thresh
out_x2 = self.x2 + thresh
out_y2 = self.y2 + thresh
out_x3 = self.x3 + thresh
out_y3 = self.y3 - thresh

```

```

x0r = centerx + (out_x0 - centerx)*c - (out_y0 - centery)*s
y0r = centery + (out_y0 - centery)*c + (out_x0 - centerx)*s
x1r = centerx + (out_x1 - centerx)*c - (out_y1 - centery)*s
y1r = centery + (out_y1 - centery)*c + (out_x1 - centerx)*s
x2r = centerx + (out_x2 - centerx)*c - (out_y2 - centery)*s
y2r = centery + (out_y2 - centery)*c + (out_x2 - centerx)*s
x3r = centerx + (out_x3 - centerx)*c - (out_y3 - centery)*s
y3r = centery + (out_y3 - centery)*c + (out_x3 - centerx)*s
self.pts_outer = np.array([[x0r,y0r],[x1r,y1r],[x2r,y2r],[x3r,y3r]], np.

```

```

int32)

```

```

# Внутренний четырехугольник

```

```

int_x0 = self.x + thresh
int_y0 = self.y + thresh
int_x1 = self.x1 + thresh

```

```

int_y1 = self.y1 - thresh
int_x2 = self.x2 - thresh
int_y2 = self.y2 - thresh
int_x3 = self.x3 - thresh
int_y3 = self.y3 + thresh

x0r = centerx + (int_x0 - centerx)*c - (int_y0 - centery)*s
y0r = centery + (int_y0 - centery)*c + (int_x0 - centerx)*s
x1r = centerx + (int_x1 - centerx)*c - (int_y1 - centery)*s
y1r = centery + (int_y1 - centery)*c + (int_x1 - centerx)*s
x2r = centerx + (int_x2 - centerx)*c - (int_y2 - centery)*s
y2r = centery + (int_y2 - centery)*c + (int_x2 - centerx)*s
x3r = centerx + (int_x3 - centerx)*c - (int_y3 - centery)*s
y3r = centery + (int_y3 - centery)*c + (int_x3 - centerx)*s
self.pts_inner = np.array([[x0r,y0r],[x1r,y1r],[x2r,y2r],[x3r,y3r]], np.
int32)

self.S_thresh = cv2.contourArea(self.pts_outer) -
cv2.contourArea(self.pts_inner)
self.num_whites = 0

# Класс генетического алгоритма
class GA:
    def __init__(self, image):
        self.population=[]
        self.image = image
        self.img_dims = image.shape[:2]
        self.fitness =[]
        self.num_whites_total = self.image[self.image==(255,255,255)].size
        print(self.num_whites_total)

    def init_population(self, size):
        self.size = size
        for _ in range(self.size):
            rand_x = np.random.randint(0, self.img_dims[0]-10,1)
            rand_y = np.random.randint(0, self.img_dims[1]-10,1)

```

```

        rand_w = np.random.randint(10, self.img_dims[0]*0.9,1)
        rand_h = np.random.randint(10, self.img_dims[1]*0.9,1)
        rand_theta = np.random.uniform(0, 2*np.pi)
        self.population.append(rectangle(rand_x,rand_y,rand_w,rand_h,rand_theta))

    def calc_axis_line(self, point_1, point_2):
        k = (point_1[1] - point_2[1])/(point_1[0]-
point_2[0]+ sys.float_info.epsilon)
        b = point_2[1]-k*point_2[0]
        return k, b

    def find_pxls_thresh(self):
        for i in range(self.size):
            mask_img = np.zeros(self.image.shape[:2], np.uint8)
            cv2.fillPoly(mask_img, pts = [self.population[i].pts_outer], color=(
255,255,255))
            cv2.fillPoly(mask_img, pts = [self.population[i].pts_inner], color=(
0,0,0))

            masked = cv2.bitwise_and(self.image, self.image, mask=mask_img)
            self.population[i].num_whites = masked[masked==(255,255,255)].size

    def calc_fitness(self):
        self.fitness = []
        weighted_sides = []
        for i in range(self.size):
            self.fitness.append( 100*(self.population[i].num_whites/(self.num_wh
ites_total-self.population[i].num_whites+1)))
        x = zip(self.fitness,self.population)
        xs = sorted(x,reverse = True, key=lambda tup: tup[0])
        self.fitness = [x[0] for x in xs]
        self.population = [x[1] for x in xs]

    def selection(self):
        self.population = self.population[:int(len(self.population)/2)]

```

```

def crossover(self, r1, r2):
    a = np.random.random(1)
    b = 1 - a
    x_new_1 = a * r1.x + b * r2.x
    y_new_1 = a * r1.y + b * r2.y
    w_new_1 = a * r1.w + b * r2.w
    h_new_1 = a * r1.h + b * r2.h
    theta_new_1 = a * r1.theta + b * r2.theta
    self.rect_new_1 = rectangle(x_new_1,y_new_1,w_new_1,h_new_1,theta_new_1)
    a = 1 - a
    b = 1 - b
    x_new_2 = a * r1.x + b * r2.x
    y_new_2 = a * r1.y + b * r2.y
    w_new_2 = a * r1.w + b * r2.w
    h_new_2 = a * r1.h + b * r2.h
    theta_new_2 = a * r1.theta + b * r2.theta
    self.rect_new_2 = rectangle(x_new_2,y_new_2,w_new_2,h_new_2,theta_new_2)
    return self.rect_new_1, self.rect_new_2

def repopulate(self):
    self.offspring = []
    for _ in range(len(self.population)):
        index_1 = np.random.randint(0,len(self.population)-1,1)
        index_2 = np.random.randint(index_1,len(self.population)-1,1)
        child_1, child_2 = self.crossover(self.population[index_1[0]],self.p
opulation[index_2[0]])
        self.offspring.append(child_1)
    self.population = self.population + self.offspring

def mutate(self, rect):
    rect.x += int(10*(1-2*np.random.random(1)[0]))
    rect.y += int(10*(1-2*np.random.random(1)[0]))
    rect.w += int(10*(1-2*np.random.random(1)[0]))
    rect.h += int(10*(1-2*np.random.random(1)[0]))
    rect.theta += 10*(1-2*np.random.random(1)[0])
    return rect

```



```

def mutate_population(self):
    for i in range(len(self.population)):
        self.population[i] = self.mutate(self.population[i])

if __name__ == "__main__":
    #np.random.seed(14)
    file = cv2.imread('Test/box1.jpg')
    scale_percent = 70
    width = int(file.shape[1] * scale_percent / 100)
    height = int(file.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(file, dim)
    edges = filter_image(resized)
    edges_cpy = edges.copy()
    population_len = 30
    algorithm = GA(edges_cpy)
    number_steps = 500
    algorithm.init_population(population_len)
    best_fit = 0
    best_model = None
    for _ in range(number_steps):
        edges_cpy = edges.copy()
        random.shuffle(algorithm.population)
        for i in range(population_len):
            cv2.polylines(edges_cpy, [algorithm.population[i].pts], True, (0, 255, 25
5))
            cv2.polylines(edges_cpy, [algorithm.population[i].pts_outer], True, (0,
0, 255))
            cv2.polylines(edges_cpy, [algorithm.population[i].pts_inner], True, (0,
0, 255))
        algorithm.find_pxls_thresh()
        algorithm.calc_fitness()
        if algorithm.fitness[0] > best_fit:
            best_fit = algorithm.fitness[0]
            print('Found better fitness: ', best_fit)

```

```

        best_model = algorithm.population[0]
    algorithm.selection()
    algorithm.repopulate()
    algorithm.mutate_population()
    cv2.imshow('edges', edges_cpy)
    if cv2.waitKey(27) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
edges_cpy = edges.copy()
print('Best fitness: ', best_fit )
cv2.polylines(edges_cpy,[best_model.pts],True,(0,255,255))
cv2.polylines(edges_cpy,[best_model.pts_outer],True,(0,0,255))
cv2.polylines(edges_cpy,[best_model.pts_inner],True,(0,0,255))
cv2.imshow('edges', edges_cpy)
cv2.waitKey()
cv2.destroyAllWindows()

```