Министерство науки и высшего образования Российской Федерации Национальный исследовательский технологический университет «МИСиС»Институт информационных технологий и компьютерных наук
Кафедра Инженерной Кибернетики

Курсовая работа
по дисциплине «Методы и средства обработки изображений» на тему «Приложение для детектирования дорожных конусов на фотографии»

Выполнил:студент гр. БПМ-18-1
Петкин. Д.В.

Проверил:доцент кафедры ИК, к.т.н.
Полевой Д.В.

Москва, 2022

Оглавление

Задача

Реализовать на базе библиотеки OpenCV программу, в функционал которой входит детектирование дорожных конусов на изображении.

Основные возможности и функции:

1. Чтение изображений
2. Контуры конусов с центром в формате выходного изображения

Техническое задание:

1. Язык программирования C++
2. Основная библиотека OpenCV

Описание используемого алгоритма

Алгоритм действует следующим образом: на вход программы подается изображение, которое представляет из себя фото, на котором находится некоторое количество конусов, которые нужно детектировать.

Программа получила начальное изображение

Рисунок 1. Пример входного изображения

Так как дорожные конусы исключительно красные, проведем первоначальную обработку, преобразуя изображение в цветовую модель HSV.

```
cv::cvtColor(imgOriginal, imgHSV, CV_BGR2HSV);
```

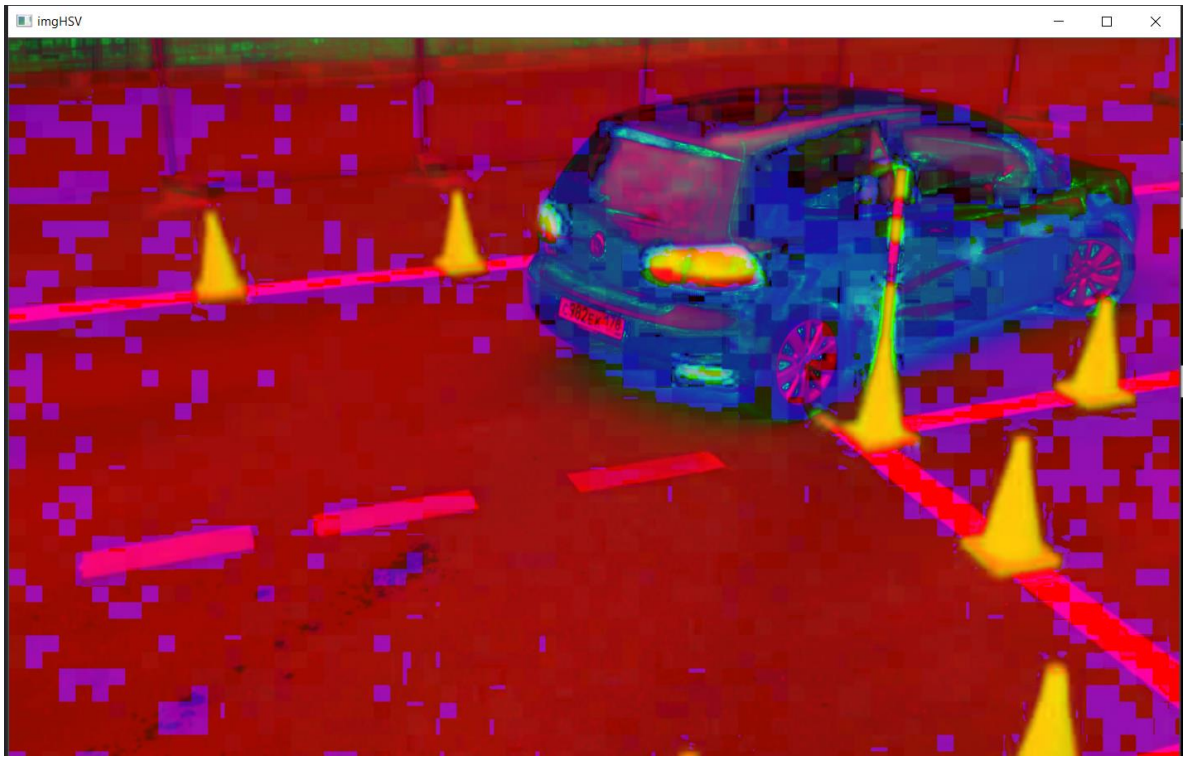

Рисунок 2. HSV Изображение

Далее используем функцию OpenCV для выделения цветового диапазона с низким и высоким пороговым значением для красного цвета, а затем объединим полученные изображения функцией add.

```
cv::inRange(imgHSV, cv::Scalar(0, 135, 135), cv::Scalar(15, 255, 255), imgThreshLow);
cv::inRange(imgHSV, cv::Scalar(159, 135, 135),cv::Scalar(179, 255, 255),imgThreshHigh);
cv::add(imgThreshLow, imgThreshHigh, imgThresh);
```

Рисунок 3. Результат сложения изображений с разным пороговым значением

Далее мы применяем к изображению морфологические преобразования – эрозию и расширение, затем сглаживаем изображение размытием по Гауссу, и только потом применяем алгоритм Кенни, для детектирования границ.

```
cv::erode(imgThreshSmoothed, imgThreshSmoothed, structuringElement3x3);
cv::dilate(imgThreshSmoothed, imgThreshSmoothed, structuringElement3x3);
cv::GaussianBlur(imgThreshSmoothed, imgThreshSmoothed, cv::Size(3, 3), 0);
cv::Canny(imgThreshSmoothed, imgCanny, 80, 160);
```

Рисунок 4. Результат приведенных преобразований

Найдем и отрисуем контуры.

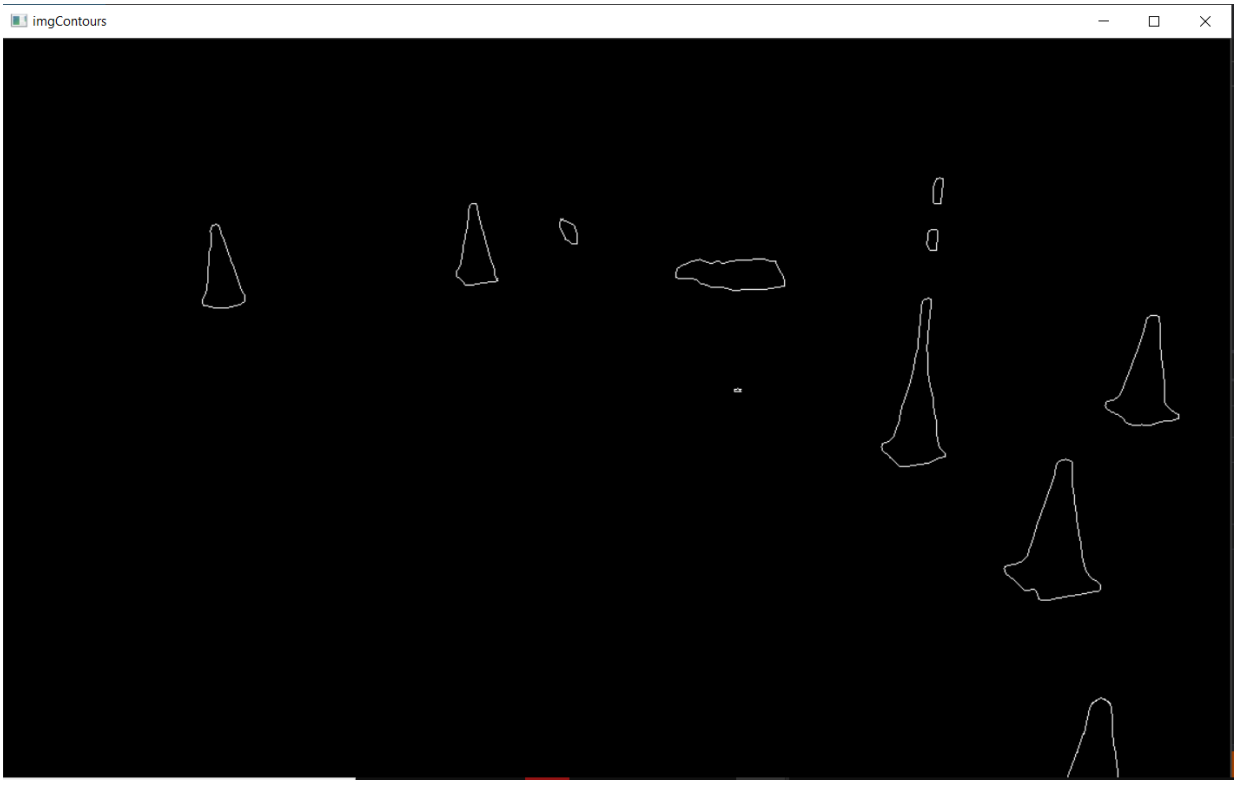

Рисунок 5. Контуры изображений

После этого найдем выпуклые оболочки, на кадре.

```
cv::convexHull(contours[i], allConvexHulls[i]);
```

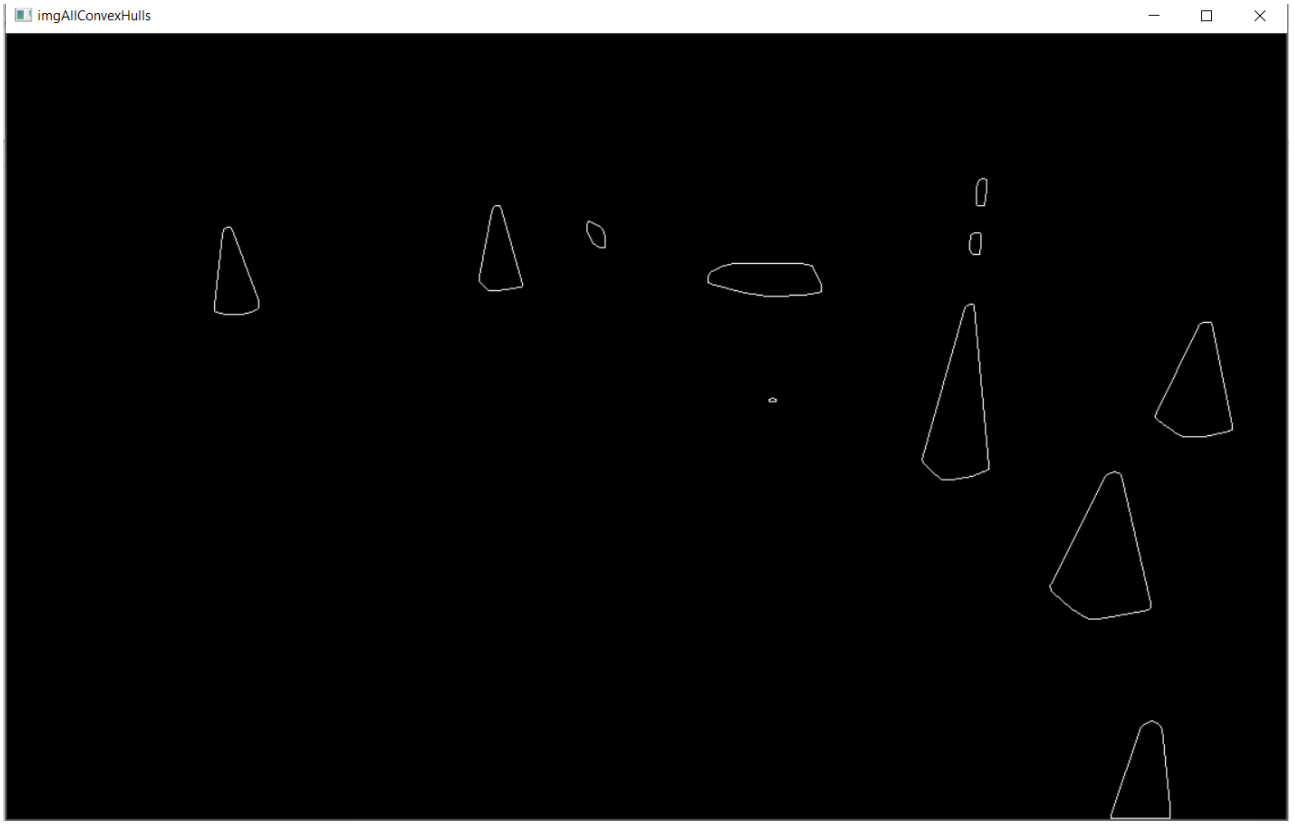

Рисунок 7. Результат convex hulls

Затем перебираем выпуклые оболочки, чтобы определить являются ли они дорожным конусом, для этого:

1) проводим грубую проверку размеров

2) определяем направлена ли выпуклая оболочка вверх, для этого находим все точки ниже и выше центра, затем проверяем нет ли среди точек выше центра таких, что они левее/правее найденных самых левых/правых точек ниже центра.

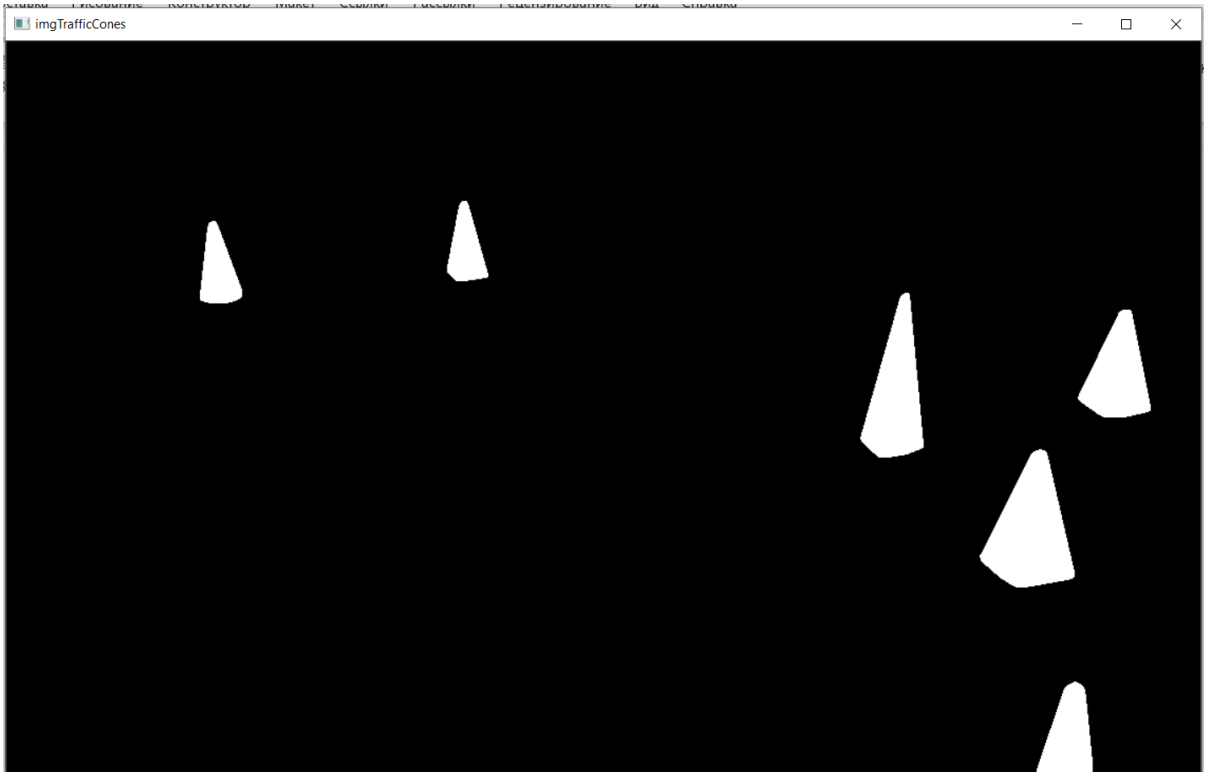

Рисунок 8. Результат нахождения дорожных конусов

Теперь добавляем оставшиеся после проверки контуры на исходное изображение, выводим их количество на экран, а также отрисовываем центры найденных дорожных конусов.

Рисунок 9. Результат работы

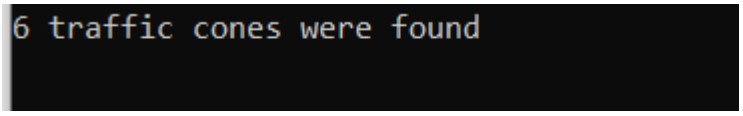

Рисунок 10. Вывод количества объектов на экран

Пользовательское описание

1. Чтобы загрузить изображение, на котором требуется детектировать дорожные конусы, пользовательдолжен просто загрузить его в папку build
2. После того как пользователь загрузил изображения, следует запустить программу Результат работы программы появится на экране

Рисунок 11. Входное изображение

Рисунок 12. Выходное изображение

Количественная оценка

IoU (Intersection over Union или пересечение над объединением) — это метод, используемый в Non-maximal Suppression для сравнения того, насколько близки два разных ограничивающих многоугольника. Это просто показано на следующем рисунке:

Рисунок 13. Оценка IOU

Чем выше IoU, тем ближе ограничивающие рамки. IoU, равное 1, означает, что две ограничивающиерамки совпали, а IoU, равное 0, означает, что они не пересекаются.

Инструкция по сборке

1. Загрузите исходные файлы из репозитория: https://github.com/Danil-petkin1/coursework
2. Для сборки данного решения у пользователя должен быть установлен CMake версии 3.8
3. Запустите Cmake (cmake-gui)
4. В верхнем поле в появившемся окне укажите путь к скачанной из репозитория папке, внижнем поле укажите путь к директории сборки
5. Перенесите папку с изображениями в папку проекта

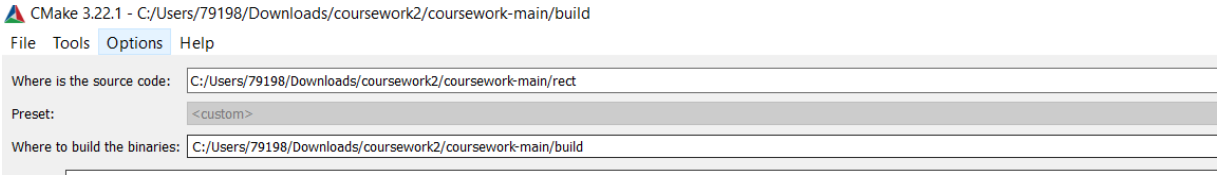

Рисунок 14. Настройка полей в Cmake (cmake-gui)

6. Нажмите на кнопку “Configure”, после чего на “Generate” и откройте проект (нажмите либо

на «Open project», либо в папке build найдите файл coursework.sln и запустите его.

7. Запустите готовое решение

Выводы по курсовой работе (анализ результатов)

В ходе курсовой работы был реализован алгоритм, позволяющий детектировать дорожные конусы по фото. Для эксперимента нам необходим эталон (маска, представляющая собой многоугольник с границами конуса на изображении), к которому должен стремиться алгоритм, сгенерировав маску программно.

Можно отметить, что алгоритм в разработанном приложении работает не совсем идеально. Из-за качества исходного изображения и расстояния до объекта, может не выдать результат, ожидаемый пользователем.

Для оценки работы алгоритма возьмём два результата выделения границ конуса, созданных по его контуру. Первый – создан вручную и принят за эталон.

Данные для отрисовки конутра-эталона для первого примера:

```
std::vector <cv::Point> conepoint = { Point(180,257),Point(194, 170),Point(231, 252)};
```

Второй – получен в результате работы алгоритма.

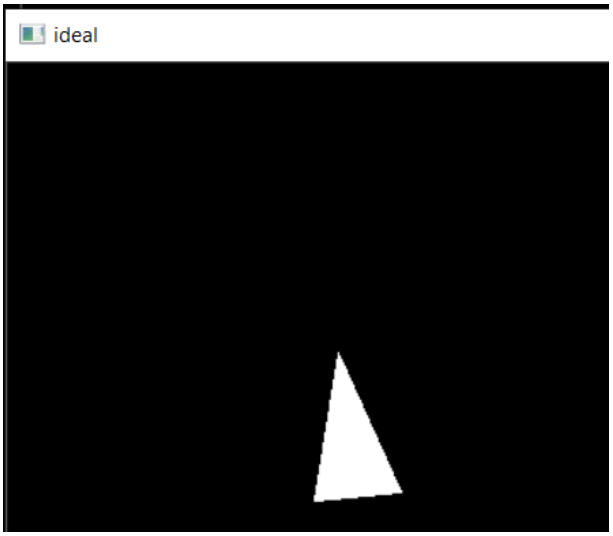

Рисунок 15. Эталон_1

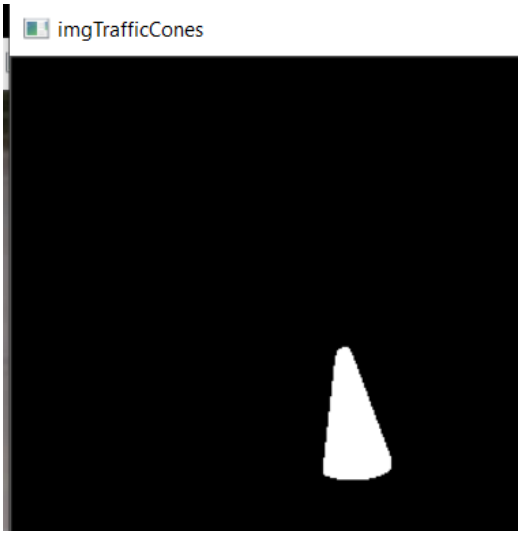

Рисунок 16. Маска, найденная алгоритмом

В данном случае **IOU** = 0.78975.

На исходном изображении хорошо виден искомый объект, поэтому алгоритму удалось определить контур объекта.

Этот результат достаточно неплох, однако, так как размер конуса невелик, результат количественной оценки всего около 80%.

Похожие результаты показал алгоритм при обработке фотографии с конусом большего размера на траве.

Данные для отрисовки контура-эталона для этого примера

```
std::vector <cv::Point> conepoint = { Point(255,435),Point(266, 435),Point(276,
504),Point(292, 511), Point(218, 513), Point(236, 503) };
```

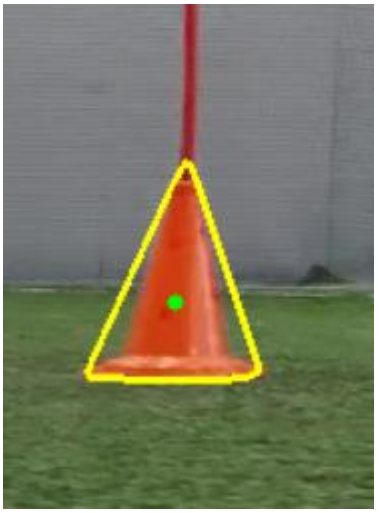

Рисунок 17. Эталон_2

Рисунок 18. Маска, найденная алгоритмом

В данном случае **IOU** = 0.89268.

Теперь возьмем другую исходную фотографию.
Данные для отрисовки контура-эталона для этого примера:

```
std::vector <cv::Point> conepoint = { Point(615,977),Point(449, 442),Point(386, 445) , Point(254, 973)
};
```

Рисунок 19. Входное изображение

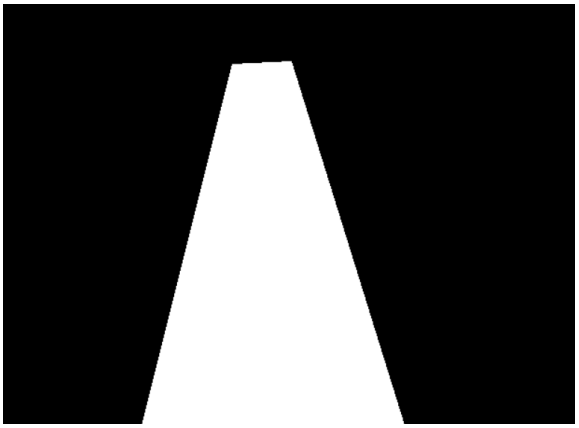

Рисунок 20. Эталон_3

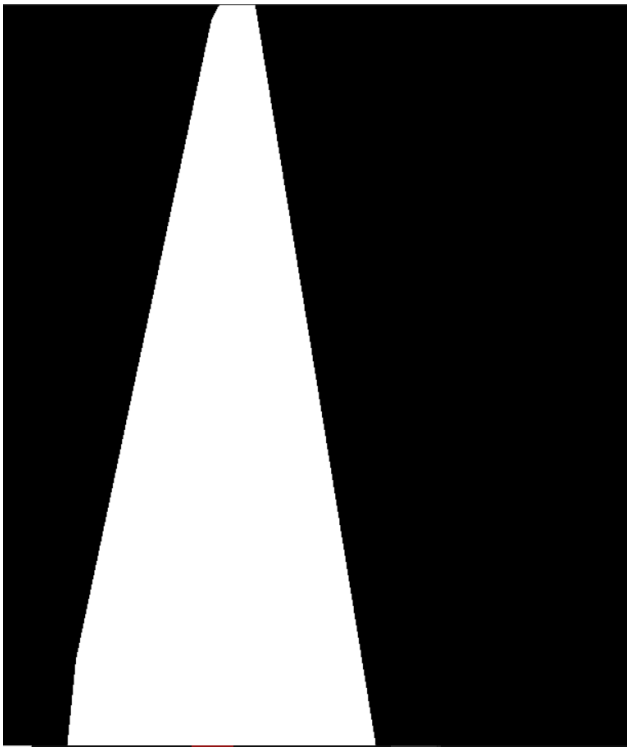

Рисунок 21. Маска, найденная алгоритмом

В этом случае **IOU** = 0.51268.

Этот результат был достаточно очевиден, так как на исходном изображении конусы находятся друг за другом. Поэтому алгоритм выделяет этот набор конусов в один большой контур, что сильно снижает **IOU**.

Таким образом, алгоритм эффективно находит дорожные конусы на фотографии (для первого и второго примера IOU больше 78%), однако если контуры конусов пересекаются между собой на исходном изображении (третий пример), алгоритм может объединить их в один, что, в свою очередь, снижает качество работы (IOU около 0.5).

Списки использованных источников

1. Техническая документация OpenCV в электронном формате
   https://docs.opencv.org/master/
2. Распознавание объектов через контуры
   https://vc.ru/dev/286152-poisk-obektov-cherez-opredelenie-ih-konturov-sredstvami-opencv
3. Объединение изображений
   https://stackoverflow.com/questions/19222343/filling-contours-with-opencv-python