Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **Специального машиностроения**

КАФЕДРА **СМ11 «Подводные роботы и аппараты»**

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент Андреев Евгений Викторович
фамилия, имя, отчество

Группа СМ11-11М

Тип практики **Научно-исследовательская работа**

Название предприятия **НУК СМ МГТУ им. Н.Э. Баумана**

Студент _____ Андреев Е. В.
подпись, дата *фамилия, и.о.*

Руководитель практики _____ Макашов А. А.
подпись, дата *фамилия, и.о.*

Оценка _____

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Кафедра **«Подводные роботы и аппараты» (СМ11)**

З А Д А Н И Е
на выполнение научно-исследовательской работы (производственной практики)

на предприятии **НУК СМ МГТУ им. Н.Э. Баумана**

Студент Андреев Евгений Викторович, СМ11-11М
(фамилия, имя, отчество; индекс группы)

Тема научно-исследовательской работы:

Использование каскадного детектора для построения системы позиционирования подводного аппарата

Дата выдачи задания «» февраля 2020 г.

Руководитель НИР _____ / Макашов А. А.
(подпись, дата)

Студент _____ / Андреев Е. В.
(подпись, дата) (Фамилия И.О.)

РЕФЕРАТ

Отчёт на 46 стр., 5 ч., 15 рис., 12 источников, 2 таблицы.

ИСПОЛЬЗОВАНИЕ КАСКАДНОГО ДЕТЕКТОРА ДЛЯ ПОСТРОЕНИЯ СИСТЕМЫ ПОЗИЦИОНИРОВАНИЯ ПОДВОДНОГО АППАРАТА

Перечень ключевых слов: АНПА, видеокамера, опорный маркер специального вида, каскадный детектор Хаара, донная зарядная станция, проблема оценки перспективы по N точкам.

Целью данной работы является исследование возможности использования каскадного классификатора Хаара для позиционирования АНПА у донной зарядной станции с помощью опорных маркеров специального вида.

В процессе работы был проведён сбор и систематизация информации по подготовке данных и обучению каскадного классификатора Хаара методом Виолы-Джонса. Были спроектированы опорные маркеры специального вида двух типов и изготовлен макет донной зарядной станции. Обучены несколько каскадных детекторов различных конфигураций для обнаружения маркеров каждого типа. Проведена калибровка камеры, получена матрица линейной модели камеры и вектор коэффициентов дисторсии, разработан пример продвинутой фильтрации информации, полученной на выходе каскадного детектора. Предложены алгоритмы упорядочивания выходных данных каскадов и вычисления наклонной дальности.

В результате исследования была предложена оптимальная методика по подготовке набора данных для обучения каскадного классификатора, проверена применимость библиотечного метода для решения задачи оценки перспективы по N точкам на основе данных, полученных от нескольких каскадных детекторов. Проведено сравнение производительности алгоритмов позиционирования на базе каскадов Хаара и (Least Binary Patterns) LBP в зависимости от их комбинаций и разрешения видеопотока.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 6

ВВЕДЕНИЕ 7

1 Каскадный классификатор Хаара 8

2 Используемые маркеры и конфигурация донной станции 10

2.1 Разработка маркеров специального вида 10

2.2 Описание макета донной зарядной станции 11

3 Определение положения камеры 13

3.1 Формулировка задачи 13

3.1 Калибровка камеры 15

4 Обучение каскадного детектора 17

4.1 Этап подготовки данных 17

4.2 Этап обучения каскада 20

5 Применение каскадного детектора для определения координат 25

5.1 Описание алгоритма фильтрации 25

5.2 Формула для пересчёта расстояния 29

5.3 Упорядочивание маркеров 30

5.4 Вычисление наклонной дальности 33

5.5 Оценка положения камеры 34

5.6 Сравнение каскдов Хаара и LBP 37

ЗАКЛЮЧЕНИЕ 41

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42

Приложение А. Сценарий для администрирования большого количества изображений в процессе обучения каскадного классификатора 44

Приложение Б. Сценарий для наполнения файла фоновых изображений..... 46

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете о НИР применяются следующие сокращения и обозначения:

LBP – Least binary patterns;

АНПА – Автономный необитаемый подводный аппарат;

ВК – Видеокамера;

ДЗС – Донная зарядная станция;

СК – система координат.

ВВЕДЕНИЕ

В настоящее время всё большее применение находят автономные необитаемые подводные аппараты (АНПА). К видам технических работ, осуществляемых аппаратами, можно отнести обследование трубопроводов и кабелей, проверку точности карт, фото- и видеосъёмку, в том числе маршрутную, осмотр опор эстакад и платформ и много другое.

В этой связи актуальной задачей является разработка методов позиционирования подводного аппарата по данным видеосистемы. Для наведения предлагается использовать маркеры специального вида, исследование проводилось применительно к задаче стыковки АНПА с донной зарядной станцией.

Предполагается, что аппарат оснащён всеми необходимыми измерителями параметров движения, вопросы маневрирования в данной работе не рассматриваются.

Цель работы – исследование каскадного детектора для построения системы видеопозиционирования АНПА.

Задачи:

- сбор и систематизация информации по подготовке данных и обучению каскадного классификатора Хаара;
- разработка маркеров специального вида для построения системы видеопозиционирования;
- сравнение производительности различных каскадов;
- исследование готовых программных решений в библиотеке OpenCV для определения положения камеры относительно объекта.

1 Каскадный классификатор Хаара

Каскадный классификатор Хаара представляет собой специальный детектор границ, построенный на каскадах решающих деревьев, каждое из которых содержит в своих листьях один из примитивов Хаара (см. рисунок 1) и два пороговых значения соответствующие ему.

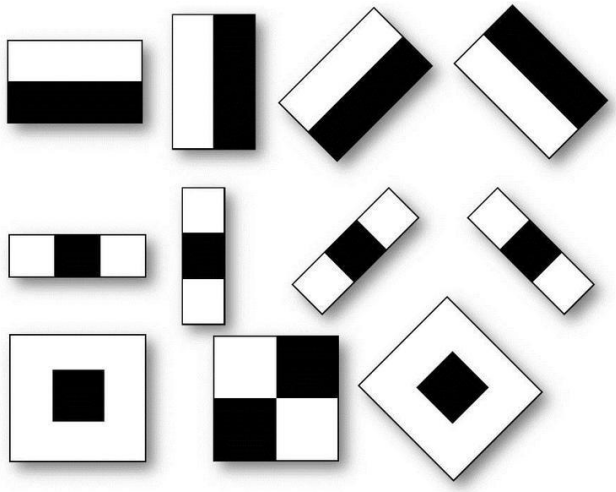

Рисунок 1 – Некоторые примитивы Хаара

Обучение каскада методом Виолы-Джонса [1, 2] построено на однократном вычислении интегрального изображения и последующим последовательном переборе примитивов и расчёте значений признаков как разницы между суммарной яркостью пикселей, покрытых былой областью текущего примитива и суммарной яркости пикселей чёрной области. Сложность вычисления признака так же как и получения значения пикселя остается O(1): значение каждой подобласти можно вычислить скомбинировав 4 значения интегрального представления. Для определения наличия или отсутствия объекта на изображении в каждом каскаде находится сумма значений слабых классификаторов этого каскада. Каждый слабый классификатор выдает два значения в зависимости от того больше или меньше заданного порога значение признака, принадлежащего этому классификатору. В конце сумма значений слабых классификаторов сравнивается с порогом каскада и выносится решения

найден объект или нет данным каскадом. Авторам статьи [1] удалось создать быстрый алгоритм поиска объектов, который пользуется успехом уже больше десятилетия.

2 Используемые маркеры и конфигурация донной станции

Использование каскада Хаара позволит задействовать несколько небольших специально спроектированных опорных маркеров, разместив их на грани донной зарядной станции в отдалении друг от друга. Тем самым удаётся обойти ограничения, связанные с габаритами ARuCo-маркеров, для которых имеется готовая реализация алгоритмов видеопозиционирования в OpenCV.

2.1 Разработка маркеров специального вида

Для решения задачи навигации были разработаны 2 типа маркеров специального вида, показанных на рисунке 2. Все они имеют размеры 30х30 мм.

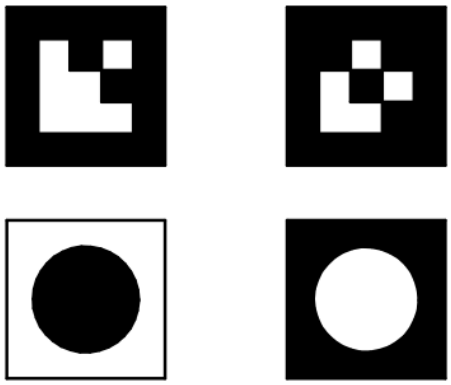

Рисунок 2 – Различные варианты опорных меток

Первый ряд представляет собой ARuCo-подобные маркеры с укрупнёнными внутренними структурными элементами. Стандартный модуль “aruco”, входящий в состав библиотеки OpenCV предлагает пользователю на выбор предопределённые словари меток размерами 4 x 4, 5 x 5 или 6 x 6 клеток. Больший размер внутренних элементов предполагает более устойчивое детектирование или же возможность уменьшить габариты маркера при неизменном расстоянии обнаружения. Однако же нет никакой необходимости

равняться на алгоритмы, заложенные в известную библиотеку, тем более что поставлена задача использовать машинное обучение.

Маркеры второго ряда состоят из ещё более простых геометрических фигур и их комбинаций, именно на эти маркеры пал окончательный выбор, так как их преимущество над ARuCo и им подобными заключается в наличии круга в центре опорной метки. В дальнейшем планируется применять преобразование Хафа для продвинутой фильтрации объектов, распознанных каскадом, и уточнения центров маркеров.

2.2 Описание макета донной зарядной станции

Для удобства тестирования и отладки разрабатываемых алгоритмов был собран небольшой макет донной зарядной станции, показанный на рисунке 3. Пропорции станции приблизительно соответствуют пропорциям реального объекта: усечённая пирамида со стороной нижнего основания в 250 мм, верхнего – в 125 мм и высотой 125 мм.

Введём систему координат (СК) OXYZ, связанную с донной зарядной станцией. Её центр расположим в точке, где предполагается наличие стыковочного порта, ось абсцисс разместим в плоскости грани и направим горизонтально направо, ось ординат – в плоскости грани вверх. Ось OZ смотрит наружу и перпендикулярна грани.

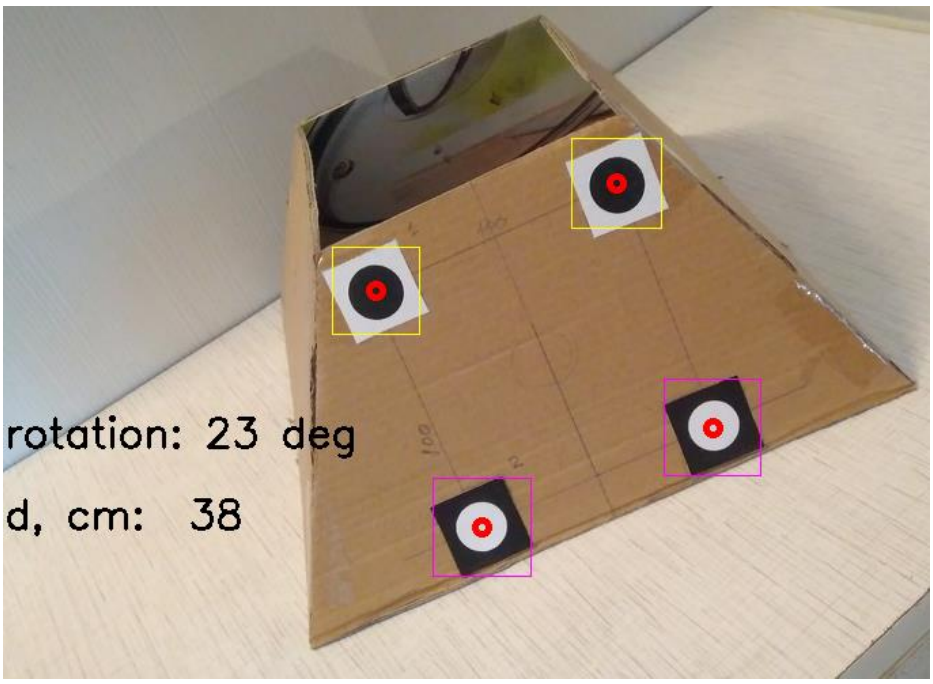

Рисунок 3 – Макет донной зарядной станции

Выбранные маркеры размерами 30 х 30 мм расположим в углах квадрата со стороной 100 мм и геометрическим центром в начале введённой СК.

3 Определение положения камеры

Задача определения положения камеры относительно объекта, для которого известны экранные и трёхмерные координаты минимум 4-х точек, исследована достаточно хорошо [3, 4] и в англоязычной литературе называется Perspective-n-Point Camera Pose Estimation.

3.1 Формулировка задачи

Для определения положения камеры понадобится следующая информация:

1. 3D-координаты точек в некоторой глобальной СК. Логичнее всего связать эту СК с самим объектом.
2. 2D-координаты тех же точек на изображении *в правильном порядке*, их предоставят каскадные детекторы после соответствующей фильтрации и упорядочивания.
3. Внутренние параметры камеры: фокусное расстояние, координаты оптического центра и параметры радиальной дисторсии. Для их получения необходимо камеру откалибровать. Однако, как описано в [5], в упрощённых случаях фокусное расстояние может быть аппроксимировано шириной кадра в пикселях, координаты оптического центра заменены координатами центра изображения, а радиальную дисторсию не учитывать.

На рисунке 4 показано преобразование трёхмерных координат в двумерные для модели камеры-обскуры.

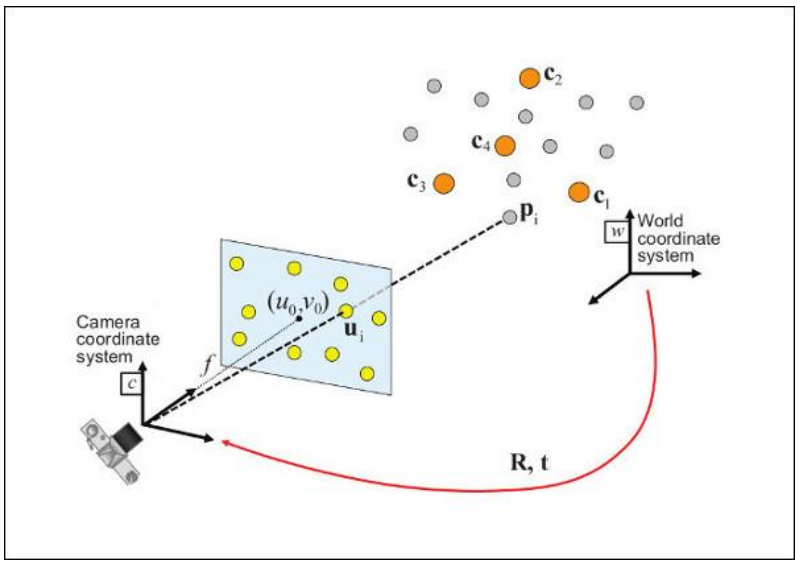

Рисунок 4 – Пояснения к задаче определения положения камеры

Учитывая соответствия между точками в глобальной СК (p_i) и их проекциями (u_i), необходимо определить положение связанной с камерой системы координат в глобальной СК согласно уравнению:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

где

r_{ij} и t_i – координаты векторов переноса и вращения, подлежащие определению;

f_x, f_y, c_x, c_y – параметры внутренней матрицы камеры, получаемые в результате калибровки;

s – масштабный коэффициент, также подлежащий определению;

X, Y, Z – трёхмерные координаты точки p_i;

u, v – координаты её проекции в пикселях.

Подробнее об определении положения камеры с учётом радиальной дисторсии можно прочесть на странице официальной документации модуля `calib3d` [6].

3.1 Калибровка камеры

Для более точного определения реальных координат объекта необходимо произвести калибровку камеры с целью получения матрицы камеры и устранения различных аберраций оптической системы. Матрица камеры используется для проецирования точек трёхмерного пространства на плоскость изображения. Нелинейные параметры внутренней калибровки, такие как коэффициенты дисторсии, также имеют важное значение, хотя и не могут быть включены в линейную модель, описываемую матрицей внутренней калибровки. Большинство современных алгоритмов калибровки камеры определяет их вместе с параметрами линейной части модели. Параметры внутренней калибровки относятся только к камере, но не к сцене, поэтому они изменяются только в том случае, когда меняются соответствующие настройки камеры.

При использовании камеры свет из снимаемой сцены фокусируется и захватывается. Этот процесс уменьшает число измерений у данных, получаемых камерой, с трёх до двух (свет из трёхмерной сцены преобразуется в двухмерное изображение). Поэтому каждый пиксель на полученном изображении соответствует лучу света исходной сцены. Во время калибровки камеры происходит поиск соответствия между трёхмерными точками сцены и пикселями изображения.

В случае идеальной камеры-обскуры для задания такого соответствия достаточно одной матрицы проекции. Однако в случае более сложных камер искажения, вносимые линзами, могут сильно повлиять на результат. Таким образом, функция проецирования принимает более сложный вид и часто записывается как последовательность преобразований.

Для получения матрицы камеры и коэффициентов дисторсии воспользуемся приложением `opencv_interactive-calibration.exe`, запустив программу с ключами `-t=chessboard -sz=25.125`.

В качестве шаблона для калибровки было выбрано изображение шахматной доски с шириной клетки в 25,125 мм. На рисунке 5 показан процесс интерактивной калибровки, в левой части отображены захваченные точки на текущем и предыдущих кадрах, а в правой – зоны покрытия кадра тестовым изображением.

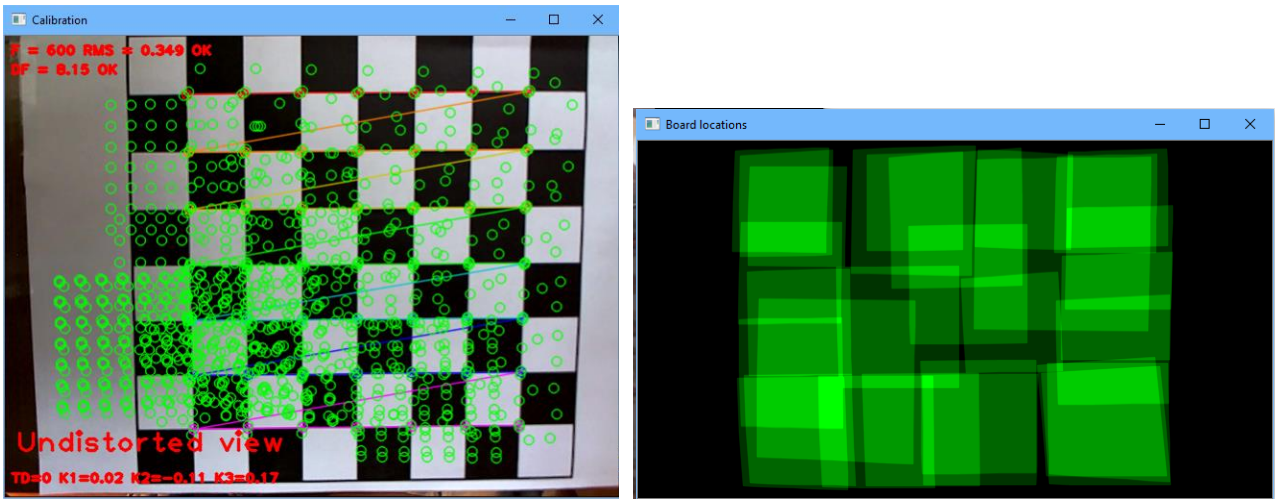

Рисунок 5 – Процесс интерактивной калибровки

При наличии достаточного количества данных программа выдаст сообщение об успешной калибровке, а вычисленные матрица камеры и коэффициенты дисторсии будут сохранены в файле `cameraParameters.xml`. Подробное руководство по интерактивной калибровке камеры описано в [7].

4 Обучение каскадного детектора

Значительную часть исследования занял сбор информации об обучении каскадного классификатора и формулирование эффективных подходов к работе с большим объёмом обучающих данных.

4.1 Этап подготовки данных

Приведём последовательность действий для подготовки изображений и их разметки

1. Для тренировки каскада потребуется два набора данных: с объектом и без него. Допускается наличие в кадре одновременно нескольких объектов. Удобнее всего не снимать последовательно несколько сотен кадров, заснять два видео. Подобный подход позволит существенно сэкономить время, а также одновременно учесть все негативные эффекты, такие как размытия, расфокусировка, возникающие во время движения реального робота.
2. Полученные видеофайлы разбиваются на кадры. Например, это можно сделать с помощью бесплатной утилиты “Free Video to Jpeg Converter”. Наибольшее количество *уникальных* кадров определяется по формуле:

   $$F * T,$$

   где

   F – частота, кадров/с;

   T - длительность видеоролика, с.
3. Для удобства предлагается переименовать изображения, оставив короткий псевдоним из латинских символов и порядковый номер (далее будет обоснована важность этого шага). Например, IMG_20200509_144931.jpg превращается в with0001.jpg. Используется бесплатное ПО FastStone Image Viewer, имеющее инструменты пакетной обработки фотографий.

4. Для тренировки алгоритма, как правило, используются изображения небольшого разрешения (1280 x 720 пикселей, 640 x 480 пикселей или меньше). В противном случае обучение займёт чрезвычайно много времени. В этом же ПО после переименования можно отмасштабировать изображения с уменьшением разрешения до указанных выше значений.
5. С помощью утилиты `opencv_createsamples.exe` (входит в пакет поставки фреймворка OpenCV) на позитивных изображениях необходимо вручную указать объект(-ы). Запуск утилиты производится командой:

   ```
   opencv_annotation.exe --annotations=good_2.dat
   --images=./extracted_images/with_original
   ```

   Ключ «`annotations`» с параметром «`good_2.dat`» задаёт выходной файл, в который будет сохранена информация о разметке, ключ «`images`» определяет папку с изображениями, содержащими требуемый объект. На рисунке 6 показан процесс аннотации изображения, содержащего макет донной зарядной станции с нанесёнными на неё маркерами специального вида. Предполагается осуществлять навигацию автономного необитаемого подводного аппарата по системе технического зрения в процессе его стыковки с зарядной станцией.

   О том, как взаимодействовать с программой разметки [8], после запуска напоминает подсказка в командной строке:

   ```
   * mark rectangles with the left mouse button,
   * press 'c' to accept a selection,
   * press 'd' to delete the latest selection,
   * press 'n' to proceed with next image,
   * press 'esc' to stop.
   ```

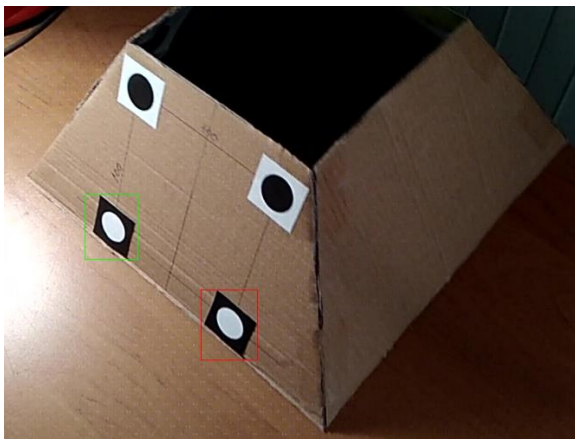

Рисунок 6 – Процесс разметки изображений. Подлежащие детектированию объекты выделены рамками. Слева – подтверждённая аннотация (после нажатия клавиши «c»), справа – не подтверждённая

Программа не будет откликаться на действия пользователя, если выбрана отличная от английской раскладка клавиатуры. По нажатии клавиши «Escape» приложение завершит работу и по указанному пути появится файл с содержимым, показанным на рисунке 7.

```
1  ./extracted_images/with_original\pyramid0289.jpg 2 491 325 63 85 642 417 80 99
2  ./extracted_images/with_original\pyramid0290.jpg 2 456 319 70 87 603 413 80 102
3  ./extracted_images/with_original\pyramid0291.jpg 2 421 334 74 84 566 433 80 94
4  ./extracted_images/with_original\pyramid0292.jpg 2 408 335 77 85 555 434 72 103
5  ./extracted images/with original\pyramid0293.jpg 2 422 336 65 83 552 438 77 95
```

Рисунок 7 – Вид содержимого файла аннотаций изображений с объектом

Каждая строчка соответствует отдельной аннотации, сначала указан путь к файлу (относительный или абсолютный, с зависимости от указанных при запуске программы разметки аргументов); следующий параметр – количество объектов на изображении, в данном примере на каждом кадре по два маркера одного вида; оставшиеся цифры обозначают координаты прямоугольников: левый верхний угол, затем правый нижний. Порядок описания прямоугольников соответствует порядку разметки пользователем.

Вообще говоря, файл негативных примеров может содержать любые изображения, включая и те, которые роботу не доведётся встретить в процессе выполнения задачи. Однако детектор будет функционировать намного лучше,

если в качестве негативных примеров использовать изображения обстановки, в которой предстоит вести обнаружение. На рисунке 8 показан фрагмент такого файла. Он не содержит никакой лишней информации, кроме относительного пути к изображениям.

```
4614  ../preparing data/extracted_images/without_advanced/advanced1658.jpg
4615  ../preparing data/extracted_images/without_advanced/advanced1659.jpg
4616  ../preparing data/extracted_images/without_advanced/advanced1660.jpg
4617  ../preparing data/extracted_images/without_advanced/advanced1661.jpg
```

Рисунок 8 – Вид содержимого файла с описанием фоновых изображений

4.2 Этап обучения каскада

Для обучения каскада [8, 9] требуется передать последнему файл-вектор, представляющий собой бинарный набор сжатых изображений объекта. Рассмотрим подробнее команду создания вектора:

```
opencv_createsamples.exe -info good_1_fixed.dat -vec
samples.vec -num 1274 -w 24 -h 24 -show
```

Данное действие выполняет программа `opencv_createsamples.exe`, также входящая в пакет поставки фреймворка компьютерного зрения. С ключом «`info`» указывается путь к файлу с аннотациями, полученному ранее. Ключ «`vec`» обозначает имя выходного вектор-файла, «`num`» — это количество позитивных изображений, а параметры «`w`» и «`h`» специфицируют ширину и высоту в пикселях. Чем больше размеры объекта, тем дольше будет проходить обучение, но тем точнее будет детектирование. Важно, чтобы отношение указанных величин соответствовало пропорциям реального объекта: к примеру, для поиска на изображении конфеты или болта стоит указать `-w 40 -h 10`, а для обнаружения маркера как на рисунке 2 лучше оставить квадратные пропорции.

Запуск обучения каскада выполняется командой

```
opencv_traincascade.exe -data haar_output -vec samples.vec
-bg bad.dat -numStages 16 -numThreads 12 -w 24 -h 24 -
numPos 1100 -numNeg 2955 -mode ALL,
```

где

`data` – путь к каталогу куда будут сохранены результаты;

`vec` – входной вектор-файл;

`bg` – файл фоновых изображений (background);

`numStages` – количество итераций (каскадов);

`numThreads` – количество задействованных потоков процессора;

`w, h` – ширина и высота изображений вектора в пикселях;

`numPos` – количество позитивных примеров, необходимых для обучения каскада. Это число обязательно должно быть меньше числа изображений в vec-файле. Причина будет объяснена ниже;

`numNeg` – количество фоновых изображений, чем больше, тем лучше;

`mode` – режим на стандартном наборе примитивов Хаара или же на расширенном (включая повёрнутые на 90º).

Программа отобразит следующий вывод:

```
PARAMETERS:
cascadeDirName: haar_output
vecFileName: samples.vec
bgFileName: bad_navigation.dat
numPos: 1100
numNeg: 2955
numStages: 16
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
acceptanceRatioBreakValue : -1
stageType: BOOST
featureType: HAAR
sampleWidth: 24
```

```
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL
Number of unique features given windowSize [24,24] :
261600
```

Здесь важными параметрами являются `minHitRate` и `maxFalseAlarmRate`. Первый обозначает точность определения, которую должен достичь каскад на каждой стадии. Значение 0,995 означает, что из 1000 изображений 5 будут ложноположительными срабатываниями. Второй – какое количество изображений, не содержащих объект, будет отсеяно на каждой стадии. К примеру, для обученного каскада из 8 стадий будет отсеиваться $1 - 0{,}5^8 = 0{,}996 = 99{,}6\%$ всех негативных изображений.

По завершении обучения каждой стадии программа будет выдавать подобные сообщения:

```
===== TRAINING 12-stage =====
<BEGIN
POS count : consumed   1100 : 1195
NEG count : acceptanceRatio    2955 : 2.89746e-05
Precalculation time: 4.876
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|        1|
+----+---------+---------+
|   4|        1|        1|
+----+---------+---------+
|   5| 0.999333| 0.836887|
+----+---------+---------+
|   6| 0.999333| 0.843316|
```

```
+----+---------+---------+
|   7| 0.998667| 0.649746|
+----+---------+---------+
|   8| 0.996667| 0.581726|
+----+---------+---------+
|   9| 0.996667| 0.461591|
+----+---------+---------+
END>
Training until now has taken 0 days 1 hours 23
minutes 3 seconds.
```

Третий столбец – значение `FalseAlarmRate`, переход к следующей стадии происходит, когда оно становится менее 0,5. Фраза «`Required leaf false alarm rate achieved. Branch training terminated.`» сообщит об успешном завершении обучения классификатора [4]. Работа обученного детектора по распознаванию маркеров специального вида показана на рисунке 9.

Рисунок 9 – Работа каскадного классификатора Хаара. Слева видно одно ложноположительное срабатывание

На каждом этапе каскад совершает ошибки, принимая некоторые фоновые изображения за изображения с объектом. Для тренировки каждой последующей стадии берутся количество изображений, соответствующих параметру `numPos`

плюс все ошибочные с предыдущей стадии. В примере вывода выше для 12-й стадии обучения присутствует строчка `POS count : consumed  1100 : 1195`, сообщающая о том, что для тренировки нынешней стадии было взято 1195 изображений, 1100 из которых изначально переданы в vec-файле. Следовательно, на предыдущем этапе ошибочно были распознаны 95 изображений.

Приблизительно количество позитивных изображений, необходимых для создания vec-файла, можно оценить по формуле:

$$V \geq numPos + (numStages - 1) * (1 - minHitRate) * numPos + S,$$

где

V – число изображений, использованных для создания вектора;

S – суммарное количество нераспознанных изображений на всех стадиях.

По приведённой выше формуле также можно оценить необходимое число объектов, передаваемых `opencv_traincascade.exe` в параметре `-numPos 1100`. При меньшем числе пользователь получит ошибку «`Bad argument (Can not get new positive sample. The most possible reason is insufficient count of samples in given vec-file).`»

5 Применение каскадного детектора для определения координат

Каждый каскад (первый - для верхних, второй - для нижних) возвращает массив прямоугольников, внутри которых с определённой вероятностью на изображении находится искомый объект. Необходимо отсеивать ложноположительные срабатывания, к тому же, границы прямоугольников не всегда соответствуют границам объекта, объект может иметь сложную форму или быть искажённым в зависимости от угла зрения. К примеру, красные кружочки на рисунке 3 обозначают центры соответствующих прямоугольников, а не центры маркеров.

5.1 Описание алгоритма фильтрации

Для получения относительно устойчивого детектирования необходимо было разработать процедуру фильтрации данных, получаемых во время работы каскадных детекторов. Возможны 4 варианта работы классификатора:

1. объект присутствует на изображении и был обнаружен;
2. объект отсутствует и не был обнаружен;
3. объект присутствует, но обнаружен не был;
4. объект отсутствует, но был обнаружен.

Первые 2 случая соответствуют ожидаемой работе детектора, последние два – ложно отрицательные и ложно положительные срабатывания. Поскольку в данном исследовании мы в основном опираемся на работу каскада Хаара, то уменьшить вероятность наступления 3-го случая без изменения параметров каскада не представляется возможным, в то время как для ложно положительных срабатываний можно предложить усовершенствованный алгоритм распознавания.

Блок-схема первой части алгоритма фильтрации представлена на рисунке 10.

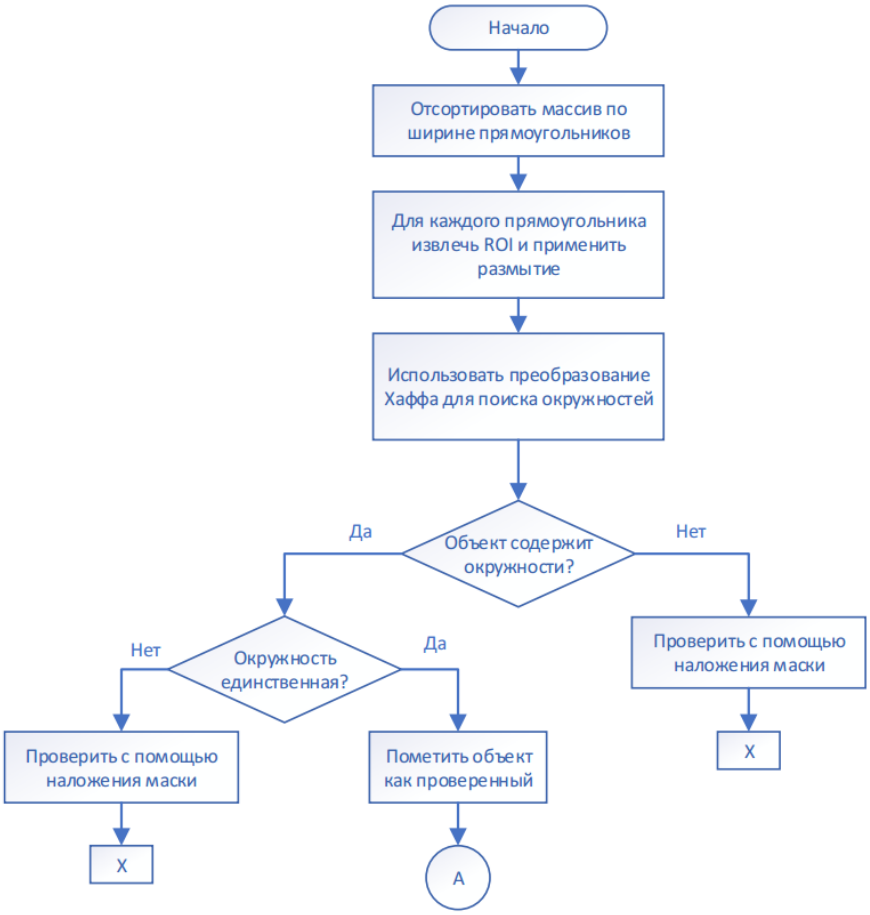

Рисунок 10 – Блок-схема фильтрации данных каскада, часть 1

Как уже было упомянуто выше, каждый каскад возвращает массив объектов класса `cv::Rect`, внутри которых с определённой вероятностью на изображении находится искомый маркер. В начале данный массив сортируется по убыванию по ширине прямоугольника. С помощью координат прямоугольника из исходного изображения извлекается ROI (Region of Interest) и применяется сглаживание для уменьшения влияния шумов. Внутри каждого ROI производится поиск окружностей с использованием преобразования Хафа. Если каскад «утверждает». что в данной области находится искомый объект, при этом в ней не обнаружено ни одной окружности или же окружностей больше одной, допустимо произвести проверку с помощью наложения маски.

Маска (см. рисунок 11, правая часть) представляет собой динамически генерируемый шаблон, предназначенный для сравнения с исходным изображением (рис. 11 слева).

Рисунок 11 – ROI с распознанным маркером и сгенерированная маска

В случае если зафиксировано совпадение, объект также был бы помечен как проверенный.

На блок-схеме, представленной на рисунке 10, соответствующие ветви помечены знаком «X». Это означает что в текущем состоянии научно-исследовательской работы эти варианты не реализованы в виду их чрезмерной трудоёмкости и необходимости предварительного сбора и статистической обработки большого объёма калибровочных данных. Ниже приведён фрагмент кода, демонстрирующий возможный вариант использования данного подхода.

```
...
else if (circles.size() > 1) {

	Mat t;
	if (m_type == markerType::black_circle) {
		t = Marker::get_template_t1(roi.rows, roi.cols);
		threshold(roi, roi, 200, 255, 0);
		absdiff(roi, t, roi);
		int nonZero = countNonZero(roi);

		//if (nonZero < 0.1 * frame_gray.cols) {
		//    hough_valid.push_back(objects[i]);
		//}
	}
	else {
		t = Marker::get_template_t2(roi.rows, roi.cols);
		threshold(roi, roi, 200, 255, 0);
		absdiff(roi, t, roi);
		int nonZero = countNonZero(roi);
		//if (nonZero < 0.15 * frame_gray.cols) {
		//    hough_valid.push_back(objects[i]);
		//}
	}
} ...
```

Статические методы `Marker::get_template_t1(...)` и `Marker::get_template_t1(...)` возвращают сгенерированный шаблон с указанными параметрами ширины и высоты. В качестве упрощения принято считать, что окружность радиуса 0,35 от ширины прямоугольника находится в центре маркера, проективные искажения не учитываем. Далее ищется матрица абсолютных разностей бинаризованного чёрно-белого изображения объекта и его двухтоновой маски. По количеству ненулевых элементов можно делать вывод о присутствии на изображении маркера.

Вторая часть алгоритма фильтрации производится только для элементов, помеченных как проверенные, и представляет отдельный цикл. Его блок-схема показана на рисунке 12.

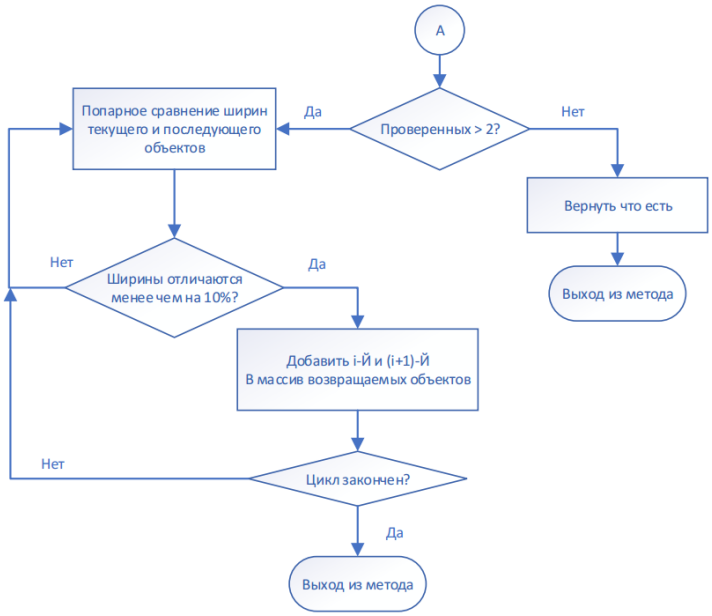

Рисунок 12 – Блок-схема фильтрации данных каскада, часть 2

Здесь производится сравнение ширины охватывающего прямоугольника текущего и последующего объектов. Их размеры не должны различаться более чем на 10% (установлено экспериментальным путём). Объекты, прошедшие проверку, возвращаются методом.

5.2 Формула для пересчёта расстояния

Для измерения расстояния по видеопотоку требуется произвести калибровку камеры по расстоянию. С этой целью был изготовлен стенд, повторяющий конфигурацию маркеров ДЗС, и отснята серия изображений, показанных на рисунке 13.

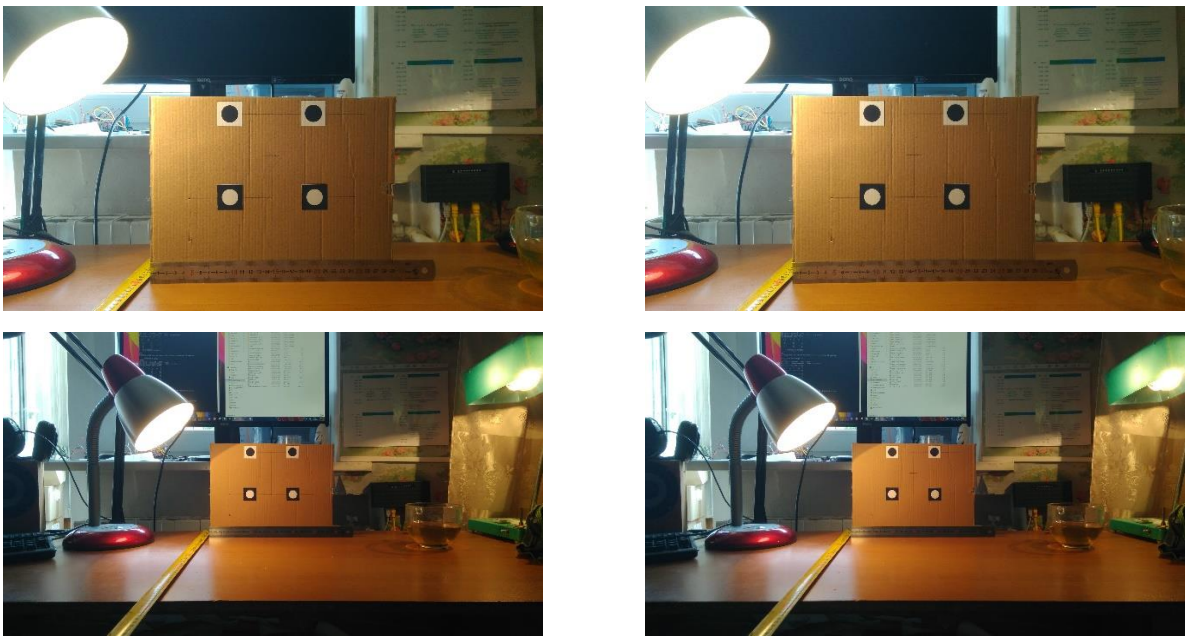

Рисунок 13 – Изображения для калибровки камеры. Расстояние на фотографиях сверху составляет 500 мм, снизу – 1000 мм.

Плоскость с маркерами расположена строго перпендикулярно лучу, соединяющему центр объектива и центр квадрата, сторона квадрата составляет 100 мм. У детектированных каскадами маркеров находился геометрический центр, представляющий собой пересечение диагоналей прямоугольника. Расстояния между каждой парой вершин были усреднены, результаты измерения приведены в таблице 1. Исходные изображения имели высокое разрешение (3840 x 2160) для уменьшения погрешностей измерения.

Таблица 1 – Результаты калибровки камеры по расстоянию

Расстояние, мм	Значение, пикселей
500	600
1000	300

В пересчёте по ширине на разрешение 640 x 480 пикселей имеем 100 px при 500 мм и 50 px при 1000 мм. Итоговая формула для пересчёта расстояния имеет вид:

$$d = \frac{50 * 100 * \frac{w}{640}}{a},$$

где

w – ширина кадра в пикселях;

a –длина ребра квадрата в пикселях;

d – расстояние по прямой, см.

5.3 Упорядочивание маркеров

В процессе работы оба каскадных детектора определяют маркеры на каждом кадре в произвольном порядке, однако для правильного оценивания положения камеры относительно донной зарядной станции необходимо поддерживать упорядоченное состояние в соответствии с выбранной ранее конфигурацией опорных меток (см. рисунок 3) вне зависимости от крена аппарата.

Для упорядочивания маркеров необходимо прежде вычислить угол поворота ДЗС вокруг оси OZ. Приведённый ниже код вычисляет угол, основываясь на данных детектирования нижних маркеров (белый круг на чёрном фоне), поскольку те изначально имели несколько меньше ложных срабатываний.

```
void AUV::rotate_over_normal(Mat& frame, vector<Rect> m1, vector<Rect> m2) {

	static double delta_x = 0;
	static double delta_y = 0;
	static double alpha = 0;

	if (m2.size() == 2) {

		//PointA
		delta_x = abs(m2[0].x - m2[1].x);
		delta_y = abs(m2[0].y - m2[1].y);

		//не известно, в каком порядке детектируются точки: сначала левая, а потом
		//правая, или наоборот
		// Если ОДНА точка левее и выше, то считаем поворот по ч.с. со знаком "+"
		if ((m2[0].x < m2[1].x && m2[0].y > m2[1].y) || (m2[1].x < m2[0].x &&
m2[1].y > m2[0].y)) {
			alpha = atan(delta_y / delta_x);
		}
		else {
			alpha = -atan(delta_y / delta_x);
		}
	}

	if (abs(alpha) >= 0 && abs(alpha) < 2 * 3.1415926535) {
		this->d_roll = alpha * 180 / 3.1415926535;
	}

	ostringstream strstream;
	strstream << setprecision(2);
	strstream << "rotation: ";
	strstream << this->d_roll;
	strstream << " deg";

	//string str = "Rotation over " + to_string(degs) + "degs";

	String text(strstream.str());
	int text_y = int(frame.rows * 0.6);
	putText(frame, text, Point(100, text_y), 0, 1, BLK, 2);

	return;
}
```

Поскольку пропорции реальной зарядной станции были известны весьма приблизительно, а значит, двугранный угол наклона боковой грани усечённой пирамиды точно неизвестен, в выше приведённом коде угол поворота вокруг нормали к грани условно назван `d_roll`, хотя таковым в действительности и не является.

После определения угла можно упорядочить маркеры с помощью следующего фрагмента кода:

```
void AUV::arrange_markers(Mat& frame) {

	//assert(m1.size() == 2 && m2.size() == 2);
	if ((m1.size() == 2 && m2.size() == 2)) {

		if (this->d_roll > 0) {

			if (m1[0].y < m1[1].y) {
				swap(m1[0], m1[1]);
			}
			if (m2[0].y < m2[1].y) {
				swap(m2[0], m2[1]);
			}
		}
		else if (this->d_roll <= 0) {

			if (m1[0].y > m1[1].y) {
				swap(m1[0], m1[1]);
			}
			if (m2[0].y > m2[1].y) {
				swap(m2[0], m2[1]);
			}
		}

		Scalar COLOR;
		if (frame.channels() == 1) {
			COLOR = WHT;
		}
		else {
			COLOR = RED;
		}

	putText(frame, String("11"), Point(m1[0].x + 10, m1[0].y - 10), 1, 1, COLOR);
	putText(frame, String("12"), Point(m1[1].x + 10, m1[1].y - 10), 1, 1, COLOR);
	putText(frame, String("21"), Point(m2[0].x + 10, m2[0].y - 10), 1, 1, COLOR);
	putText(frame, String("22"), Point(m2[1].x + 10, m2[1].y - 10), 1, 1, COLOR);
	}
}
```

Результат работы метода приведён на рисунке 14, где индексы представляют положение соответствующего маркера в матрице 2 x 2.

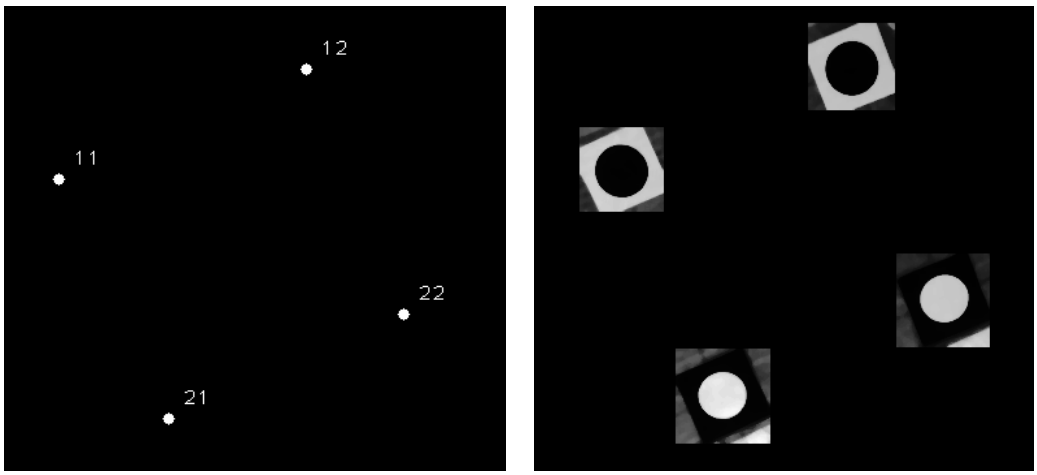

Рисунок 14 – Упорядоченные маркеры (слева) и отладочное изображение в градациях серого

5.4 Вычисление наклонной дальности

Проще всего воспользоваться полученной выше информацией и определить расстояние по прямой от аппарата до начала системы координат, связанной с донной станцией, то есть наклонной дальности. Вычисление производит следующий метод:

```
void AUV::calculate_distance(Mat& frame, bool debug) {

	if (m1.size() == 2) {
		//Point a, b;
		upper = sqrt(pow(abs(m1[0].x - m1[1].x), 2) + pow(abs(m1[0].y - m1[1].y),
2));
	}

	if (m2.size() == 2) {
		lower = sqrt(pow(abs(m2[0].x - m2[1].x), 2) + pow(abs(m2[0].y - m2[1].y),
2));
	}

	if (debug) {
		cout << "upper = " << upper << " lower = " << lower << "\n";
	}

	int w = frame.cols;
	float scale = float(w) / 640;

	double average = (upper + lower) / 2;
	double calculated_distance = 50 * 100 / (average / scale);

	if (calculated_distance > 0 && calculated_distance <= 200)
		this->dist = calculated_distance;
```

```
        ostringstream strstream;
        //strstream << setprecision(0);
        strstream << "d, cm: ";
        strstream << setw(3) << int(dist);// << " " << setw(3) << int(lower);

        String text(strstream.str());
        //putText(frame, text, Point(10, 400), 0, 1, Scalar(255, 255, 255), 2);
        int text_y = int(frame.rows * 0.5);
        putText(frame, text, Point(100, text_y), 0, 1, BLK, 2);
}
```

В коде выше после входных проверок вычисляются длины верхнего и нижнего рёбер квадрата, в углах которого располагаются опорные маркеры. Находится их среднее значение и определяется итоговое расстояние согласно формуле, полученной в разделе 5.2. Таким образом достигается коррекция измерений по углу дифферента АНПА, коррекция по углу курса может быть осуществлена аналогичным образом с использованием длин боковых рёбер. Однако учёт обоих искажений одновременно является нетривиальной задачей и за выходит за рамки темы данной научной работы.

Показанный выше код был разработан по большей части для целей отладки и, как будет показано в дальнейшем, для сравнения с готовым библиотечным решением.

5.5 Оценка положения камеры

Большая предварительно проделанная работа позволит наконец использовать готовый метод `solvePnP(…)` [11] для определения положения камеры, описываемого векторами переноса и вращения.

OpenCV предоставляет четыре метода решения задачи оценки перспективы по N точкам: итеративный, EPNP, P3P и DLS. В зависимости от типа приложения методы оценки будет отличаться. В том случае, когда мы разрабатываем приложение реального времени, более подходящими методами являются EPNP и P3P, поскольку они быстрее находят оптимальное решение,

чем итерационные и DLS. Однако EPNP и P3P не слишком надежны при использовании плоских поверхностей, и иногда оценка положения имеет зеркальный эффект. Поэтому в данном случае используется итерационный метод.

Матрицы камеры и дисторсии, а также вектор координат маркеров в системе координат станции являются членами класса `AUV`. Их инициализация происходит в конструкторе класса:

```
AUV::AUV(string path1, string path2) {

marker_1_path = path1;
marker_2_path = path2;

m1.resize(2);
m2.resize(2);

/*
Simplified solution

double focal_length = frame_gray.cols; // Approximate focal length.
Point2d center = cv::Point2d(frame_gray.cols / 2, frame_gray.rows / 2);
cv::Mat camera_matrix = (cv::Mat_<double>(3, 3) << focal_length, 0, center.x, 0,
focal_length, center.y, 0, 0, 1);
*/

cMatrix640 = (Mat_<double>(3, 3) << 5.3226273868525448e+02, 0, 3.2590522394049350e+02,
                                    0, 5.3226273868525448e+02, 2.6946997900677803e+02,
                                    0, 0, 1);

cMatrix1280 = (Mat_<double>(3, 3) << 8.6155235630774325e+02, 0, 6.2961522415048103e+02,
                                     0, 8.6155235630774325e+02, 3.9881978167213623e+02,
                                     0, 0, 1);


distortion640 = (Mat_<double>(1, 5) << 0, -6.1539772782054671e-02, 0, 0,
1.7618036793466491e-02);

distortion1280 = (Mat_<double>(1, 5) << 0, -6.5524123635067169e-02, 0, 0, 0 );

// Задание координат маркеров
model_points.push_back(cv::Point3d(-50, 50, 0));  // left up corner
model_points.push_back(cv::Point3d(50, 50, 0));   // right up corner
model_points.push_back(cv::Point3d(50, -50, 0));  // left up corner
model_points.push_back(cv::Point3d(-50, -50, 0)); // left down corner

...

}
```

Метод, осуществляющий вызов `solvePnP(...)` приведён ниже:

```
void AUV::estimatePos(Mat &frame, bool draw_perp) {
```

```
        if (this->m1.size() == 2 && this->m2.size() == 2) {

        vector<Point2d> corners = {
                                Point2d(m1[0].x, m1[0].y),
                                Point2d(m1[1].x, m1[1].y),
                                Point2d(m2[1].x, m2[1].y),
                                Point2d(m2[0].x, m2[0].y)
        };


        // Solve for pose
        solvePnP(model_points, corners, cMatrix640, distortion640, Rvec, Tvec);
        //solvePnP(model_points, corners, camera_matrix, distortion640, Rvec, Tvec);


        cout << setprecision(5);

        for (int j = 0; j < Tvec.rows; j++) {
                cout << setw(8) << Tvec.at<double>(j, 0);
        }
        cout << "\n";

        if (draw_perp) {

                vector<Point2d> perpendicular_point2D;
                vector<Point3d> perpendicular_point3D;
                perpendicular_point3D.push_back(Point3d(0, 0, 0));
                perpendicular_point3D.push_back(Point3d(0, 0, 100));

                projectPoints(perpendicular_point3D, Rvec, Tvec, cMatrix640,
                distortion640, perpendicular_point2D);

                line(frame, perpendicular_point2D[0], perpendicular_point2D[1], BLU, 2);
                }
        }
        else {
                cout << "Less than 4 markers\n";
                cout << m1.size() << " "  << m2.size() << "\n";
        }
}
```

Здесь на каждом кадре после предварительной проверки происходит задание экранных координат центров маркеров в нужном порядке и вызов метода `solvePnP(...)`. Вектора переноса и вращения `Rvec` и `Tvec` имеют тип `cv::Mat`, являются также членами класса и передаются по ссылке. При желании разработчик также может отобразить положение нормали, как показано на рисунке 15. Отметим, что данные, полученные в результате оценки положения камеры, также требуют большой фильтрации.

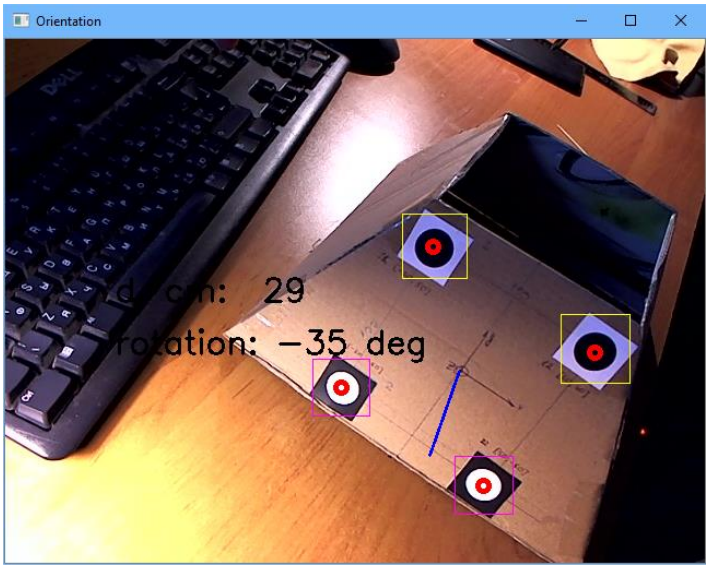

Рисунок 15 – Демонстрация работы алгоритма оценки положения камеры

В процессе исследования было выявлено, что устойчивое детектирование достигается лишь при угле сектора не превышающем 60º по вертикали и горизонтали.

Поскольку проверка точного соответствия измеряемых и реальных координат вызывает некоторые сложности, оценивалась суммарная погрешность алгоритма по измерению наклонной дальности. Между камерой и ДЗС устанавливалась линейка: погрешность определения расстояния методом оценки перспективы по N точкам составляла приблизительно 15 мм на 250 мм дистанции, а расчёт согласно пункту 5.4 – приблизительно 8-10 мм.

5.6 Сравнение каскадов Хаара и LBP

Утилита «opencv_traincascade.exe» позволяет обучать как каскады Хаара, так и LBP (Local Binary Patterns). В открытых источниках [12] упоминается, что 2-й тип каскадного детектора в несколько раз быстрее каскада Хаара и на 15-20% менее точен. Было проведено собственное сравнение, результаты которого представлены в таблице 2.

Таблица 2 — Сравнение производительности алгоритма детектирования на основе каскадов Хаара и LBP

Тип	Источник видеоряда	Скорость, кадров/с
LBP	Тестовое видео, 1280 x 720 пикселей	12,5
HAAR		11
LBP	Потоковое видео с домашней веб-камеры, 640 x 480, разрешение по умолчанию для OpenCV	17
HAAR		15
LBP	Тестовое видео, 1280 x 720 пикселей, детектируется только один тип маркера (1-й тип - чёрный круг на белом фоне)	17
HAAR		15
LBP	Потоковое видео с домашней веб-камеры, 640 x 480, детектируется только один тип маркера (1-й тип - чёрный круг на белом фоне)	20
HAAR		19

По данным таблицы 2 ожидаемой разницы в производительности не наблюдается. Однако в выводе программы «opencv_traincascade.exe» при равных размерах поискового окна количество уникальных комбинаций примитивов, их размеров и положений сильно различается:

```
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL
```

```
stageType: BOOST
featureType: LBP
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
```

```
Number of unique features given windowSize [24,24] : 261600
```

```
Number of unique features given windowSize [24,24] : 8464
```

Сравнете значения, выделенные жирным. Автор связывает подобное поведение с особенностью разработанных маркеров: каскадам необходимо детектировать плоское контрастное изображение из простых геометрических фигур, в результате чего большое количество примитивов отсеивается на ранних стадиях. Таким образом, для детектирования достаточно весьма ограниченного набора примитивов и соответствующих им слабых классификаторов. Однако время полного обучения каскадов различается значительно:

```
featureType: HAAR

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   1100 : 1113
NEG count : acceptanceRatio 4617 : 3.13382e-05
Precalculation time: 5.091
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1| 0.565735|
+----+---------+---------+
|   4|        1| 0.565735|
+----+---------+---------+
|   5|        1| 0.286766|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 41 minutes 35 seconds.

===== TRAINING 6-stage =====
<BEGIN
POS count : consumed   1100 : 1113
NEG count : acceptanceRatio   2 : 1.01725e-05
```

```
featureType: LBP

===== TRAINING 5-stage =====
<BEGIN
POS count : consumed   1100 : 1107
NEG count : acceptanceRatio 4617 : 8.73515e-05
Precalculation time: 0.33
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1| 0.347412|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 2 minutes 8 seconds.

===== TRAINING 6-stage =====
<BEGIN
POS count : consumed   1100 : 1107
NEG count : acceptanceRatio   0 : 0
Required leaf false alarm rate achieved.   Branch   training terminated.
```

```
Required leaf false alarm rate
achieved.    Branch    training
terminated.
```

ЗАКЛЮЧЕНИЕ

В результате проведённого исследования была исследована возможность применения каскадного детектора Хаара, обученного по методу Виолы-Джонса, в совокупности с опорными маркерами специального вида, для построения системы видеопозиционирования подводного аппарата у донного объекта. Выработаны рекомендации по подготовке данных и обучению каскадного классификатора, приведена формула для оценочного расчёта требуемого количества позитивных изображений, представлены сценарии для наполнения файлов аннотаций и администрирования большого количества изображений. Произведена калибровка камеры и предложен алгоритм фильтрации выходных данных каскада, позволяющий значительно повысить стабильность детектирования, уменьшив процент ложных срабатываний. Проанализирована производительность алгоритма детектирования для двух типов каскадов (Хаара и LBP) в зависимости от разрешения видеопотока и выбранной комбинации каскадов. Представлен алгоритм расстановки маркеров после предварительной фильтрации данных каскада, показан упрощённый метод определения наклонной дальности до объекта и проверена применимость библиотечного метода для решения задачи оценки перспективы по N точкам на основе обработанной информации от каскадного детектора.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Rapid Object Detection using a Boosted Cascade of Simple Features, Paul Viola, Michael Jeffrey Jones. [электронный ресурс]. URL: https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features (дата обращения: 10.05.2020).
2. Метод Виолы-Джонса (Viola-Jones) как основа для распознавания лиц [электронный ресурс]. URL: https://habr.com/ru/post/133826/ (дата обращения: 10.05.2020).
3. Страница документации фреймворка OpenCV. Real Time pose estimation of a textured object [электронный ресурс]. URL: https://docs.opencv.org/master/dc/d2c/tutorial_real_time_pose.html (дата обращения: 12.05.2020).
4. EPnP: An Accurate O(n) Solution to the PnP Problem. Vincent Lepetit, Francesc Moreno-Noguer, Pascal Fua, [электронный ресурс]. URL: https://link.springer.com/article/10.1007/s11263-008-0152-6 (дата обращения: 12.05.2020).
5. Head Pose Estimation using OpenCV and Dlib [электронный ресурс]. URL: https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/ (дата обращения: 12.05.2020).
6. Страница документации молуля `calib3d` фреймворка OpenCV [электронный ресурс]. URL: https://docs.opencv.org/master/d9/d0c/group__calib3d.html (дата обращения: 12.05.2020).
7. Руководство про интерактивной калибровке камеры. Официальный сайт фреймворка OpenCV [электронный ресурс]. URL: https://docs.opencv.org/3.4.9/d7/d21/tutorial_interactive_calibration.html (дата обращения: 12.05.2020).
8. Страница документации фреймворка OpenCV по обучению каскадного классификатора [электронный ресурс]. URL:

https://docs.opencv.org/4.1.1/dc/d88/tutorial_traincascade.html (дата обращения: 12.05.2020).

9. Обучение каскада Хаара на примере поиска символов автомобильного номера OpenCV [электронный ресурс]. URL: https://kostyakulakov.ru/opencv-%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BA%D0%B0%D1%81%D0%BA%D0%B0%D0%B4%D0%B0-%D1%85%D0%B0%D0%B0%D1%80%D0%B0/ (дата обращения: 12.05.2020).
10. Страница форума stackoverflow.com [электронный ресурс]. URL: https://stackoverflow.com/questions/50186866/opencv-cascade-training-fast-fails-required-leaf-false-alarm-rate-achieved-br (дата обращения: 16.05.2020).
11. Страница документации фреймворка OpenCV. Описание метода solvePnP() [электронный ресурс]. URL: https://docs.opencv.org/3.4.9/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d (дата обращения: 12.05.2020).
12. Страница форума stackoverflow.com. Вопрос о выборе типа каскадного детектора [электронный ресурс]. URL: https://stackoverflow.com/questions/8791178/haar-cascades-vs-lbp-cascades-in-face-detection (дата обращения: 16.05.2020).

Приложение А. Сценарий для администрирования большого количества изображений в процессе обучения каскадного классификатора

```
abs_path = 'Your_absolute_path_to_base_folder/'
which_file = 2

if which_file == 1:
  f = open(abs_path+'good_1.dat')
  f_fixed = open(abs_path+'good_1_fixed.dat', 'a')
else:
  f = open(abs_path + 'good_2.dat')
  f_fixed = open(abs_path + 'good_2_fixed.dat', 'a')

data = f.readlines()
print(type(data))

for line in data:

  first_part = line.strip().split()[0]
  #print(first_part)
  file_name = first_part.split("\\")[-1]
  print(file_name)

  if which_file == 1:
    try:
      shutil.move(abs_path+"from_folder/"+file_name,
      abs_path+" /to_folder/"+file_name)
      line = line.strip().split('\\')
      new_line = line[0] + "/to_folder/" + line[1] + "\n"
      print(new_line, end='')
      f_fixed.write(new_line)
    except FileNotFoundError:
      print(file_name, " does not exist")
      input()
  else:
    try:
      shutil.move(abs_path+"from_folder_2/"+file_name,
      abs_path+"to_folder_2/"+file_name)
        line = line.strip().split('\\')
        new_line = line[0] + "/to_folder_2/" + line[1] +
"\n"
        print(new_line, end='')
        f_fixed.write(new_line)
```

```
except FileNotFoundError:
    print(file_name, " does not exist")
    input()
```

Приложение Б. Сценарий для наполнения файла фоновых изображений

```
import os

f = open('bad.dat', 'w')

for i in range(1, 2956):
    s   =   './extracted_images/without_800/advanced'   +
'{:04d}'.format(i) + ".jpg\n"
    f.write(s)
    # print(s, end = '')
```