Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”
(УНИВЕРСИТЕТ ИТМО)

ЦЕНТР АВТОРИЗОВАННОГО ОБУЧЕНИЯ ИНФОРМАЦИОННЫМ ТЕХНОЛОГИЯМ

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА

Разработка робототехнического комплекса с использованием системы технического зрения для анализа продукции.

Автор Варламов Игорь Андреевич _____
(Фамилия Имя Отчество) (Подпись)

Центр авторизованного обучения информационным технологиям

Наименование программы **«Python-разработчик»**

Руководитель Кузьмин Константин Михайлович _____
(Подпись)

К защите допустить

Заместитель директора ЦАО ИТ, к.т.н. _____ */ Т.В. Зудилова/*

Санкт-Петербург, 2024г.

Обучающийся_____Варламов Игорь Андреевич_____ Группа 124/12
(Фамилия, И. О.)

Работа принята «___» _____ 2024г.

Работа выполнена с оценкой _____

Дата защиты «___» _____ 2024г.

Секретарь ИАК _____ _____
(Фамилия, И. О.) (подпись)

Листов хранения _____

Демонстрационных материалов _____

Оглавление

Введение

Система технического зрения (СТЗ) играет важную роль в управлении промышленным роботом. Она отвечает за распознавание, автоматизацию контроля и анализ объектов на основе изображений.

Эти системы позволяют роботам получать изображения объектов, сцен и рабочих процессов, а затем обрабатывать их с помощью цифровых устройств. Системы технического зрения используются для обнаружения, распознавания или идентификации объектов, определения их местоположения и координат. Полученные данные затем помогают промышленному роботу выполнить задачи, на которые он запрограммирован.

Системы технического зрения (СТЗ) [1] в связке с промышленным роботом или робототехническим комплексом находят широкое применение в различных областях производства. Вот несколько вариантов использования:

1. Контроль качества: СТЗ могут использоваться для контроля качества изготовляемой продукции. Например, обнаружение дефектов на поверхности деталей, коррекция параметров обработки, или проверка правильности сборки изделий.

2. Подбор и сортировка: СТЗ позволяют роботам выполнять задачи по подбору и сортировке предметов на производственной линии. Например, система может распознавать и классифицировать товары или детали, чтобы робот мог правильно распределить их в соответствующие контейнеры или упаковки.

3. Позиционирование и навигация: Системы технического зрения позволяют роботам определять свое местоположение, а также точно определять положение и ориентацию объектов в пространстве. Это особенно важно для задач погрузки, разгрузки и перемещения предметов на производственной площадке.

4. Сбор данных и аналитика: СТЗ могут использоваться для сбора данных о производственных процессах, например, для анализа производственной линии, контроля запасов, или для предупреждения аварийных ситуаций.

5. Сотрудничающие роботы: СТЗ позволяют роботам взаимодействовать с людьми и другими роботами безопасно. Они могут обнаруживать присутствие людей или препятствий, чтобы избегать столкновений и работать в совместном окружении.

Цель данной работы:

Исследование, разработка и реализация робототехнического комплекса (робот Kawasaki RS007), оснащенного системой технического зрения (камера ace Basler), с целью обеспечения автоматизации процесса обнаружения, захвата, транспортировки и сортировки заготовок на производстве. В особенности задачи можно отнести необходимость реализации максимально поддерживаемого программного кода, а также простого и понятно интерфейса взаимодействия с комплексом.

1. Анализ подходов к распознаванию цилиндрических заготовок с метками

1.1. Особенности задачи

В качестве объекта обнаружения в данной задаче выступает заготовка (рисунок 1.1)

Рисунок 1.1 – Объект обнаружения

Так как задачу обнаружения объектов требуется реализовать в условиях плохой освещенности возникает проблема засвета и бликов на металлической поверхности. Это необходимо будет учитывать при выборе алгоритма обнаружения.

Заготовки расположены специальной подложке (рисунок 1.2).

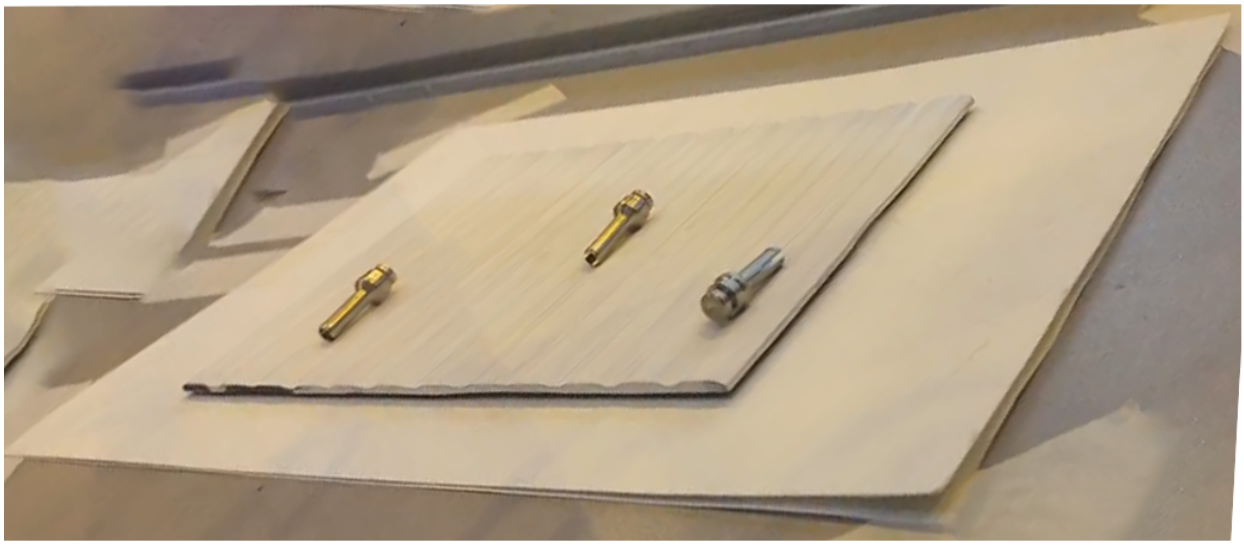


Рисунок 1.2 – Подложка для деталей

Транспортировку заготовок будет выполнять робот Kawasaki RS007 (рисунок 1.3). Передача данных между СТЗ и роботом будет осуществляться по TCP/IP.

Рисунок 1.3 – Робот Kawasaki RS007

1.2. Методы алгоритмического обнаружения при помощи камеры

Глобально для поиска объектов на подготовленном изображении применяется:

1. Использование геометрических признаков: Этот подход основан на распознавании геометрических особенностей формы заготовок. Алгоритмы компьютерного зрения могут быть использованы для анализа формы, размеров и расположения меток на поверхности цилиндрических объектов.

2. Использование цветовой информации: Техническое зрение также позволяет анализировать цветовую информацию и их контраст с поверхностью заготовок. Это может быть особенно полезно в условиях изменяющегося освещения.

Поиск можно реализовать следующими методами:

1. Алгоритм Хафа (Hough Transform): Алгоритм Хафа часто используется для обнаружения форм на изображениях. Он работает путем преобразования изображения в пространство параметров и поиска линейных структур в этом пространстве.

2. Метод Лапласа: Этот метод основан на выделении краевых структур на изображении с использованием оператора Лапласа. После выделения краев, можно применить алгоритм поиска с использованием этих структур.

3. Алгоритм Марр-Хилдет (Marr-Hildreth Algorithm): Этот алгоритм комбинирует оператор Лапласа с гауссовым размытием для поиска форм на изображении.

4. Поиск при помощи нейросетевого обнаружения: Нейросетевые методы обнаружения объектов на изображениях стали очень популярными благодаря своей способности эффективно извлекать признаки и обучаться на больших объемах данных. В основе этого метода лежит использование сверточных нейронных сетей (CNN), которые способны автоматически извлекать признаки из изображений на разных уровнях абстракции.

5. Поиск при помощи анализа контуров: Этот метод основан на анализе контуров объектов на изображении. Контур представляет собой границу объекта, образованную изменениями интенсивности пикселей. Анализ контуров позволяет определять форму и структуру объектов на изображении. Преимущества метода анализа контуров включают простоту реализации, быструю скорость работы и возможность работы с изображениями высокого разрешения. Однако он может быть менее эффективным в условиях сильного шума или низкого контраста на изображении.

По результатам предварительного эксперимента были получены следующие результаты:

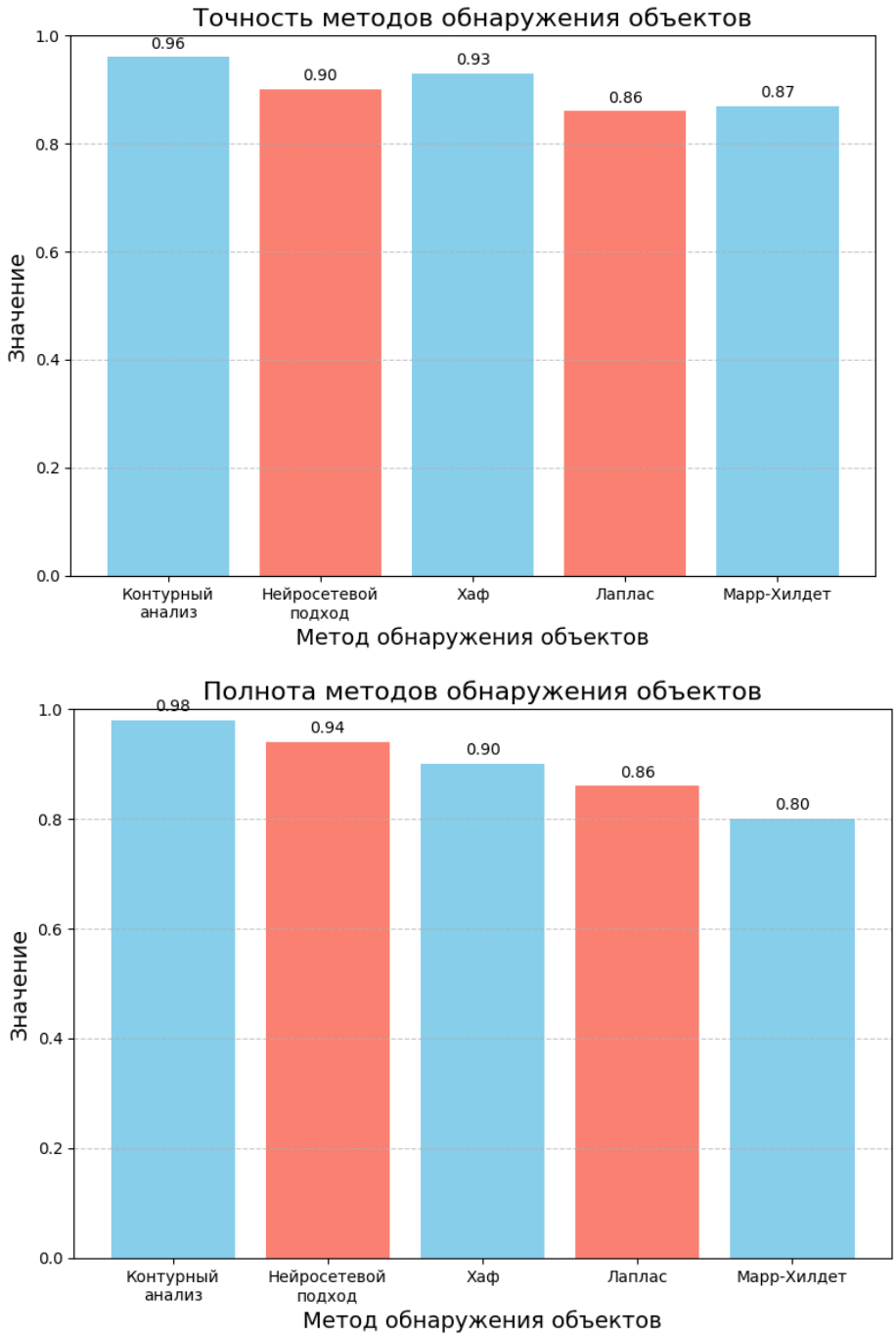


Рисунок 1.4 - Точность и полнота распознавания на подготовленном видео

Под «подготовленным» изображением имеется ввиду качественная настройка параметров захвата камеры, такие как ISO, диафрагма, а также постобработка полученного изображения программно: яркость, контраст, цветокоррекция. Также каждый метод требовал достаточно тонкой настройки путем подбора параметров вручную.

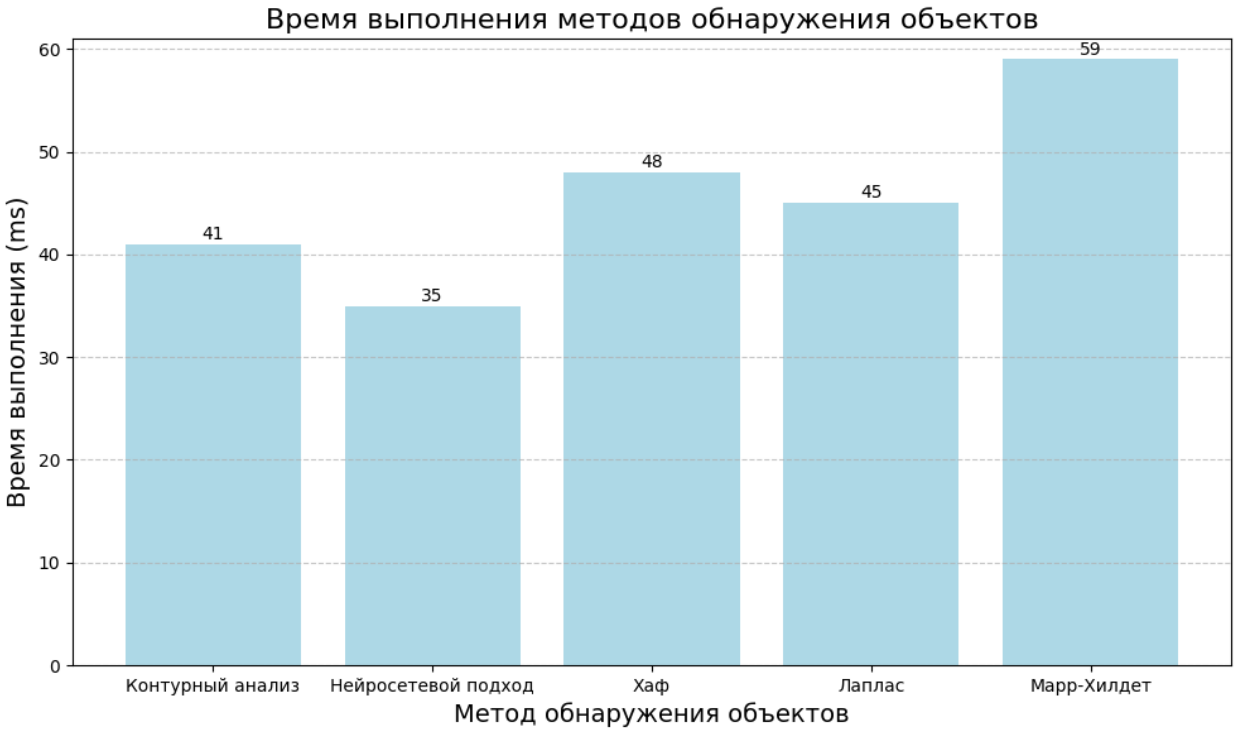


Рисунок 1.5 – Среднее затраченное время на обнаружение

Выше был описан самый простой случай, так как на качество обнаружения не влияли такие факторы, как размытие при движении объектов, изменяющееся в процессе съемки освещение и т.д.

Результаты тестов на «готовом» видео представлены на рисунке 1.4

Итоговые результаты выглядят так:

Контурный анализ – 96.23% точности и 98.15% полноты при среднем времени поиска – 41 ms

Нейросетевой подход – 90.27% точности и 94.84% полноты при среднем времени поиска – 35 ms

Хаф – 93.03% точности и 90.20% полноты при среднем времени поиска – 48 ms

Лаплас – 86.58% точности и 86.01% полноты при среднем времени поиска – 45 ms

Marr-Hildreth Algorith – 87.91% точности и 80.84% полноты при среднем времени поиска – 59 ms

1.3 Методы повышения качества обнаружения

Стоит отметить, что на результат исследования алгоритмов сильно влияет качество ручного подбора параметров, один из вариантов решение этой проблемы – создание алгоритма динамической подстройки параметров метода в зависимости от изменений внешней среды. Такое решение может быть эффективно, но сильно осложнит решение задачи и увеличит время разработки. В данной работе было принято решение отказаться от такого подхода ввиду необходимости создания максимально простого и легко модернизируемого программного решения, и ограничения по времени разработки.

Как уже было сказано ранее, очень важно уделить внимание на подготовку данных:

1. Улучшение четкости и контрастности: Применение фильтров для улучшения четкости и контрастности изображения помогает сделать объекты более выразительными и улучшить различимость текстур и границ объектов.

2. Фильтрация шума: Уменьшение шума на видео поможет улучшить четкость объектов. Применение различных методов фильтрации, таких как медианный фильтр или фильтр Гаусса, может существенно улучшить качество изображения.

3. Нормализация яркости и баланс белого: Регулирование яркости, контрастности и баланса белого может сделать изображение более естественным и улучшить видимость объектов.

4. Использование методов улучшения резкости: Применение методов увеличения резкости изображения помогает сделать границы объектов более четкими, что может быть важно для точного обнаружения объектов.

5. Суперразрешение изображений: Применение техник суперразрешения позволяет улучшить разрешение и детализацию изображения. Это особенно полезно для обнаружения маленьких объектов или объектов с низкой контрастностью.

6. Компенсация движения: При наличии движения на видео можно использовать методы стабилизации изображения для уменьшения размытия и улучшения качества.

7. Коррекция искажений: Если на видео присутствуют искажения, их коррекция (например, дисторсия или афинные искажения) может помочь в улучшении визуального восприятия объектов.

1.4Выводы по главе 1

Для распознавания целесообразно использовать алгоритмические методы.

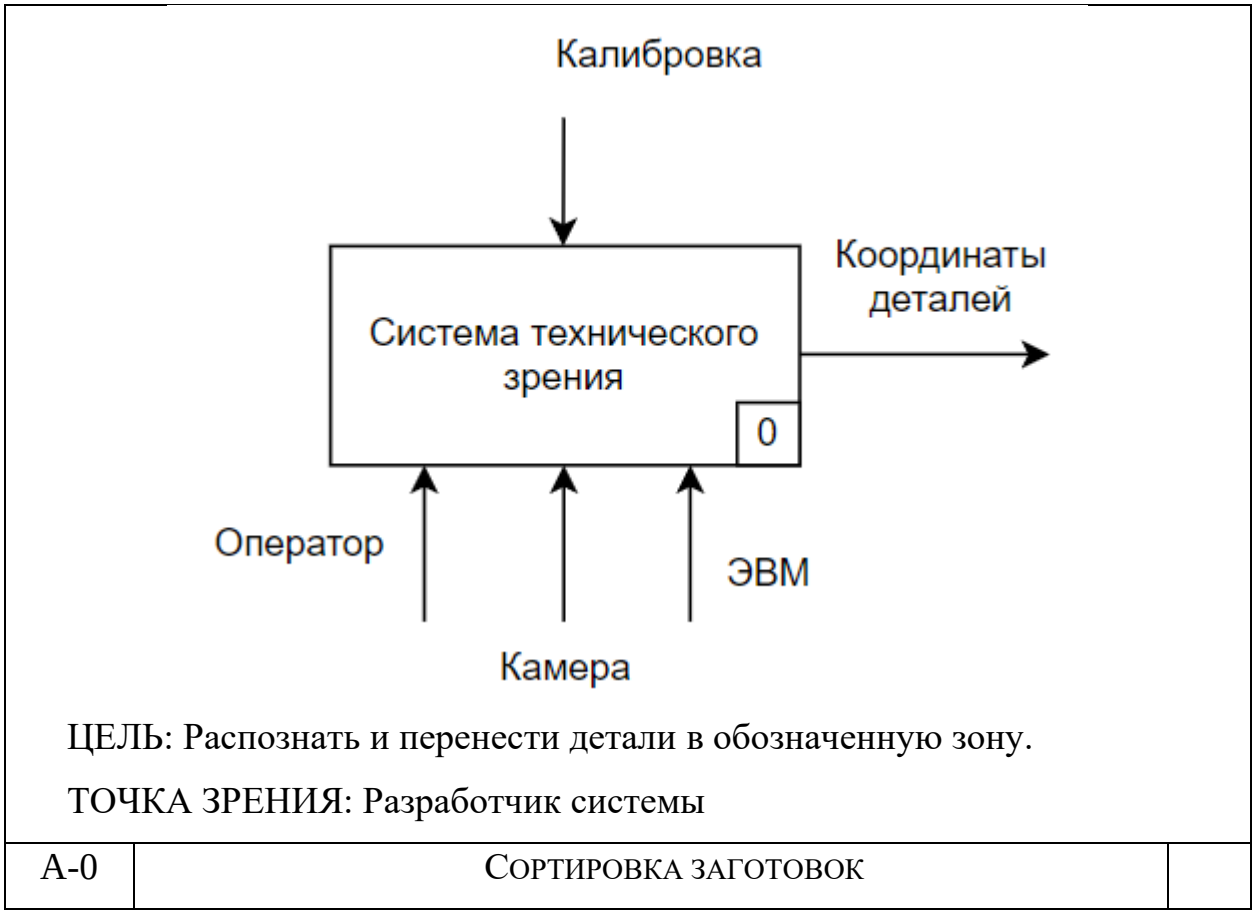


Рисунок 1.6 – Верхний уровень алгоритма

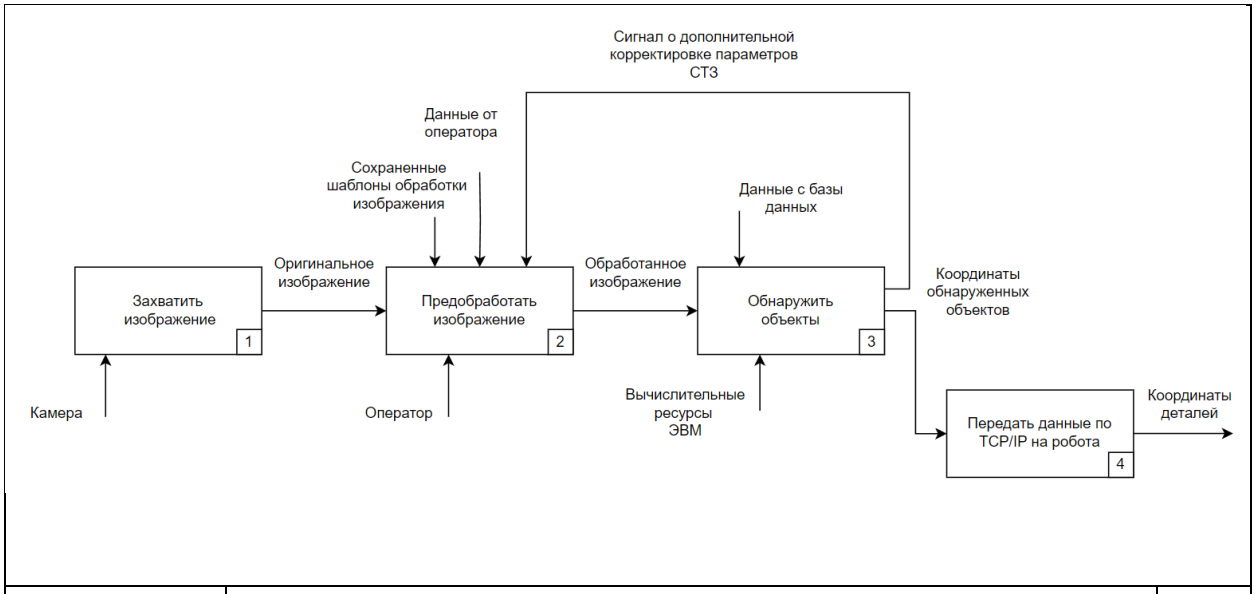


Рисунок 1.7 – Первый уровень декомпозиции алгоритма работы комплекса

Подробнее про декомпозицию системы будет описано в главе 3.

Так же будет необходима точная настройка получаемого изображения под каждое освещение, блики и т.д., для удобной настрой будет реализован пользовательский интерфейс.

Движение робота будет реализовано на встроенном в роботы Kawasaki языке программирование AS, при помощи K-IDE – программного обеспечения робота.

Реализация СТЗ будет выполнена на OpenCV, ввиду наличия и простоты настройки и использования большинства вышеописанных методов и алгоритмов.

2 Математическое описание основных этапов разработки

2.1 Методы определения центра детали

Система распознавания реализована при помощи встроенных функций OpenCV:

cv2.adaptiveThreshold():

Эта функция используется для создания бинарного изображения из оттенков серого с использованием адаптивного порогового преобразования.

Адаптивное пороговое преобразование позволяет использовать разные пороговые значения в разных частях изображения, что особенно полезно при изменяющейся освещенности.

Математически адаптивное пороговое преобразование выглядит следующим образом:

$$T(x, y) = mean\left(I_{gray}(x', y')\right) - c \quad (1)$$

Где T(x,y) - пороговое значение для пикселя (x,y) $I_{gray}(x', y')$ - яркость пикселей в локальной окрестности вокруг (x,y), а c - константа, вычитаемая из среднего значения, чтобы определить порог.

cv2.findContours():

Эта функция используется для поиска контуров на бинарном изображении.

Контур - это кривая, представляющая собой границу между объектом и фоном.

Алгоритм находит набор точек, образующих контур, и возвращает их в виде списка.

Математически поиск контуров основан на алгоритмах сканирования границы, которые последовательно следуют по границе объекта и записывают координаты пикселей, образующих контур.

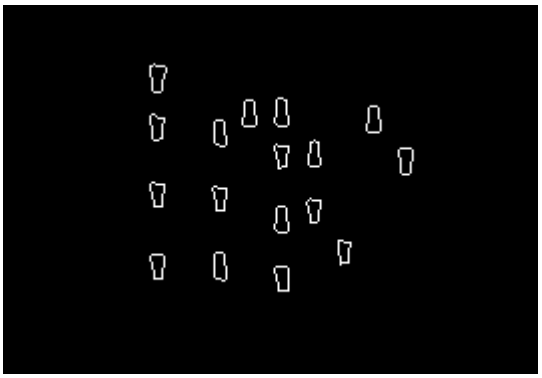


Рисунок 2.1 – Обнаруженные контуры

cv2.moments():

Эта функция используется для вычисления моментов контура, которые представляют собой статистические характеристики формы контура.

Момент M_{ij} определяется следующим образом:

$$M_{ij} = \sum_{x} \sum_{y} x^i y^j I(x,y) \quad (2)$$

Где I(x,y) - яркость пикселя в координатах (x,y), а x и y - координаты пикселя.

cv2.contourArea():

Эта функция используется для вычисления площади контура.

Площадь контура вычисляется как количество пикселей, ограниченных контуром.

Математически площадь контура может быть вычислена с использованием формулы Гаусса для вычисления площади многоугольника по координатам его вершин.

2.2 Методы определения ориентации детали

Система определения ориентации детали реализована при помощи простых математических преобразований:

Поиск ограничивающего прямоугольника: это операция имеет сложность O(n), где n - количество точек контура.

Создание и изменение ROI имеет сложность O(1), так как размер области интереса фиксированный и не зависит от размера контура.

Итерации по области интереса ROI:

Вложенные циклы имеют сложность O(height * width), где height и width - размеры ROI.

Внутри внутреннего цикла выполняются операции доступа к элементам массива, проверки условий и выполнения операций над изображением, которые в целом имеют константную сложность O(1).

Финальная обработка результатов также имеет константную сложность O(1), так как выполняются независимо от размеров изображения или контура.

Общая алгоритмическая сложность данной функции будет зависеть от размеров изображения и областей интереса на нем, но может быть приблизительно оценена как O(n), где n - общее количество пикселей в областях интереса ROI.

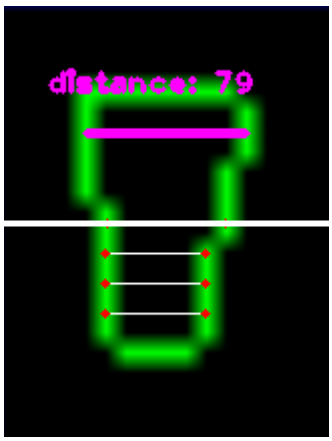


Рисунок 2.2 – Визуализация работы алгоритма определения ориентации

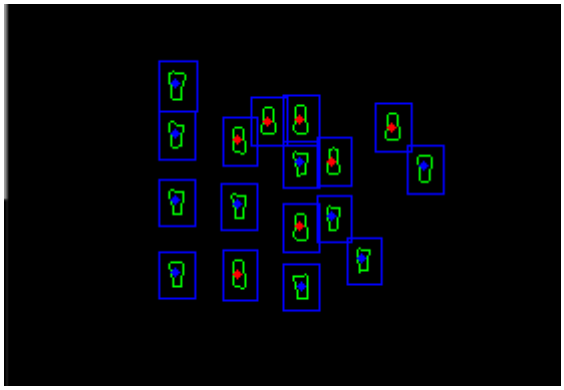


Рисунок 2.3 – Результата работы алгоритма определения ориентации

2.3 Переход в систему координат робота

После отработки всех алгоритмов поиска и прогнозирования необходимо передать данные на робота. В нашем случае данными будут координаты x_1 и y_1 в системе координат OpenCV в пикселях (рисунок 2.6)

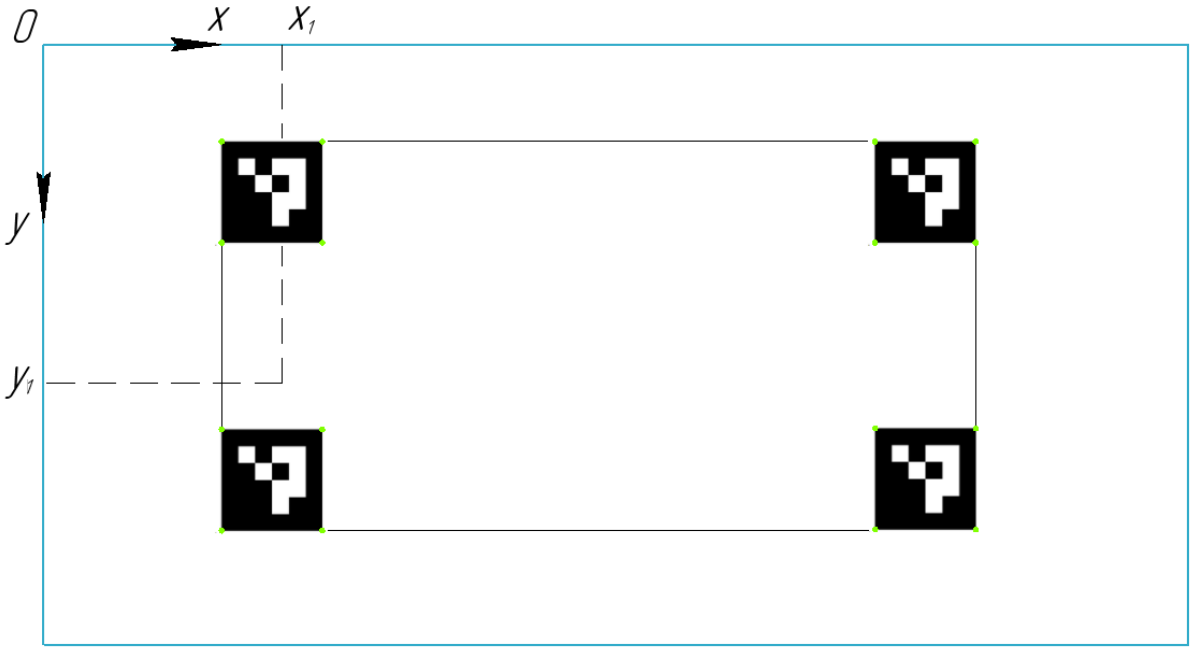


Рисунок 2.6 – Схематичное обозначение системы координат OpenCV

Чтобы передать эти координаты на робота необходимо решить 2 проблемы:

1. Размерность данных должны быть выражена в миллиметрах
2. Необходимо трансформировать координаты из системы координат OpenCV в систему координат робота

Последняя проблема была решена предварительной калибровкой при помощи ArUco Marker (рисунок 2.7)

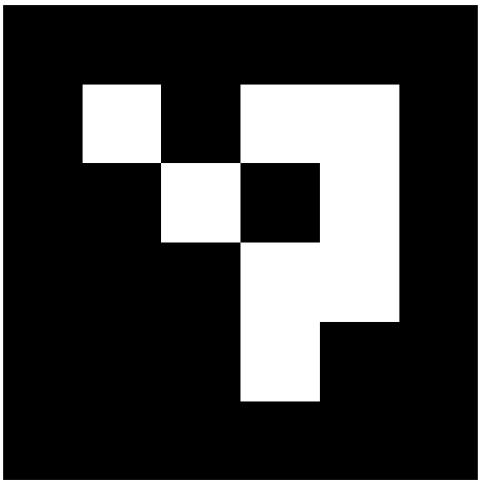


Рисунок 2.7 – ArUco Marker

OpenCV имеет функционал для обнаружения таких маркеров. Для этого расположим маркер так, чтобы его стороны были параллельны сторонам кадра и найдем его углы, а затем найдем расстояние между ними (рисунок 2.8).

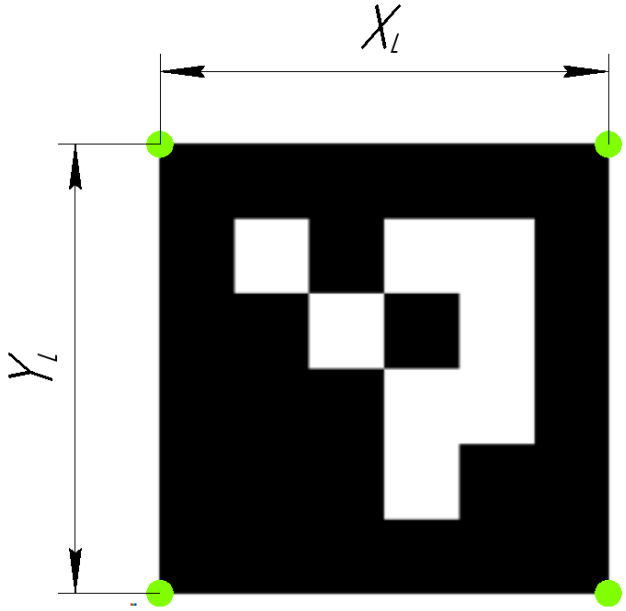


Рисунок 2.8 – Длины сторон маркера

Таким образом мы получили длину его сторон в пикселях – X_L , Y_L

Зная его реальную длину в миллиметрах - X, Y можем высчитать поправочный коэффициент:

$$\frac{X_L}{X} = X_{coef}, \qquad \frac{Y_L}{Y} = Y_{coef} \tag{3}$$

Потом поделим координаты детали на этот коэффициент:

$$\frac{x_1}{X_{coef}} = x_{CV}, \qquad \frac{y_1}{Y_{coef}} = y_{CV} \tag{4}$$

где x_{CV} и y_{CV} – Это координаты детали в миллиметрах в системе координат OpenCV.

Чтобы решить проблемы сопоставления систем координат приведем схему (рисунок 2.9), на нем обозначено: 1 – палета с деталями, 2 – деталь, 3 – камера, 4 – робот с его системой координат и направлением осей y_R, x_R, 5 – точка захвата детали, 6 – зона захвата камеры и направление осей x_{CV}, y_{CV}, 7 – центр фланца инструмента робота.

Задача упрощается тем фактом, что высоту по оси Z можно программно задать на роботе – оставим ее на уровне поверхности палеты (в дальнейшем такое решение поможет при написании программы безопасности – робот не сможет опустить фланец инструмента ниже уровня нуля оси Z).

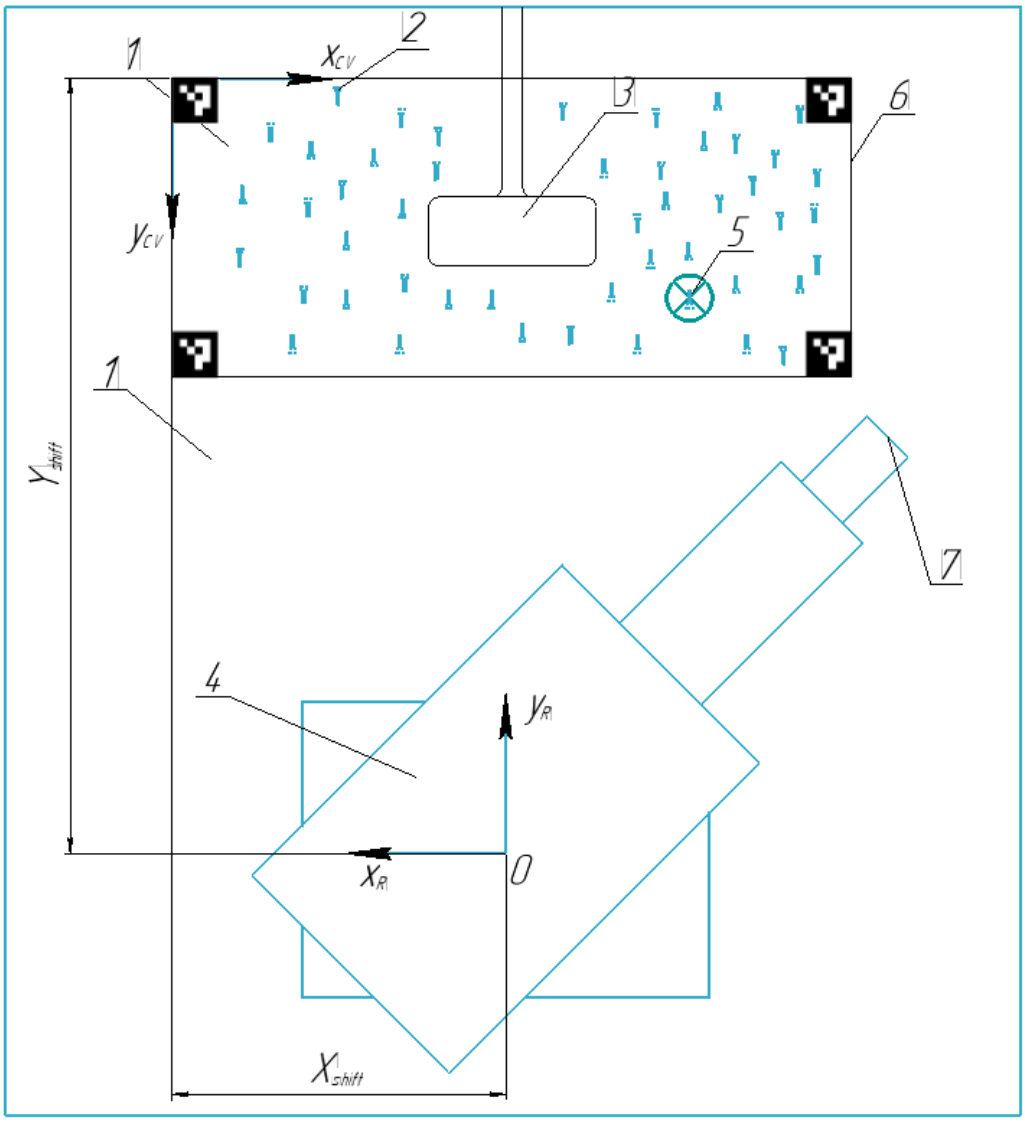


Рисунок 2.9 – Схематичное обозначение рабочей зоны

Чтобы корректно трансформировать координаты x_{CV} и y_{CV} необходимо знать расстояние между началами 2-х систем – эту задачу можно решить при помощи встроенной возможности робота путем перемещения фланца (7) в начало системы отсчета камеры (6), а затем, зная положение фланца относительно начала отсчета системы координат робота, на роботе высчитать расстояние X_{shift}, Y_{shift}.

Далее вычисляем координаты детали в системе координат робота в миллиметрах:

$$y_R = Y_{shift} - y_{CV}, \quad x_R = X_{shift} - x_{CV} \tag{5}$$

2.4 Расчет траектории движения манипулятора

Робот-манипулятор можно рассматривать как набор жестких последовательно соединенных звеньев, каждое из которых оснащено приводом. Робот имеет шесть вращательных кинематических пар (рисунок 2.5.1), причем оси J1, J2, J3 являются переносными, а оси J4, J5, J6 – ориентирующими.

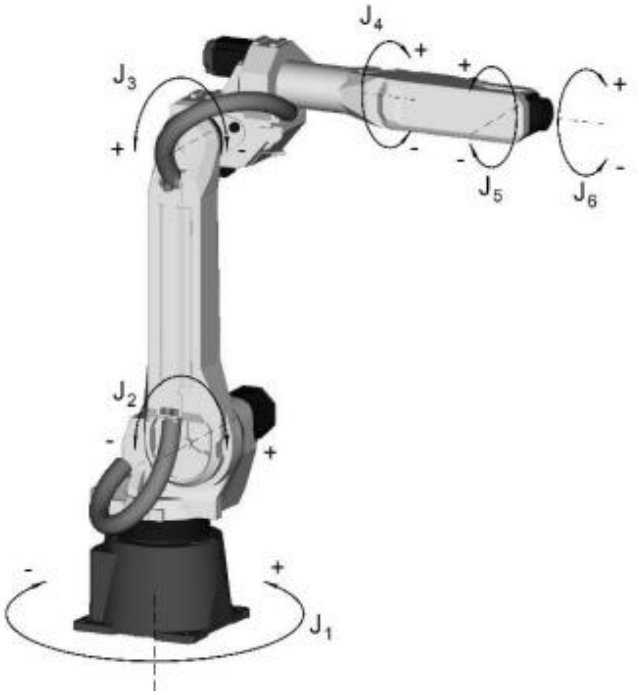


Рисунок 2.10 – 6-ти осевой робот

В общем случае для 6-ти осевого робота, оснащенного позиционером, математическая модель кинематики для обратной задачи можно представить в виде:

$$\begin{aligned} J_1 &= J_1(x, y, z, i, j, k, \alpha_1, \theta_1, a_1, d_1, t) \\ J_2 &= J_2(x, y, z, i, j, k, \alpha_2, \theta_2, a_2, d_2, t) \\ J_3 &= J_3(x, y, z, i, j, k, \alpha_3, \theta_3, a_3, d_3, t) \\ J_4 &= J_4(x, y, z, i, j, k, \alpha_4, \theta_4, a_4, d_4, t) \\ J_5 &= J_5(x, y, z, i, j, k, \alpha_5, \theta_5, a_5, d_5, t) \\ J_6 &= J_6(x, y, z, i, j, k, \alpha_6, \theta_6, a_6, d_6, t) \end{aligned} \quad (15)$$

где x, y, z – декартовы координаты; i, j, k – направляющие орты оси инструмента; α_i , θ_i , a_i , d_i – параметры звеньев; t – время; α_i – угловое смещение – угол, на который нужно повернуть ось z_{i-1} вокруг оси x_i , чтобы

она стала сонаправлена с осью z_i; θ_i – присоединительный угол, на который нужно повернуть ось x_{i-1} вокруг оси z_{i-1}, чтобы она стала сонаправлена с осью x_i; a_i – линейное смещение – расстояния между пересечением оси z_{i-1} с осью x_i и началом i-й системы координат, отсчитываемое вдоль оси x_i; d_i – расстояние между пересечением оси z_{i-1} с осью x_i и началом (i-1)-й системы координат, отсчитываемое вдоль оси z_{i-1}. Для вращательных сочленений параметры α_i, a_i, d_i являются постоянными величинами для каждой конкретной модели робота, характеризующими конструкцию звеньев.

Существует множество методов решения обратной задачи кинематики: метод обратных преобразований, прямые геометрические методы, метод на основе нелинейного математического программирования. Последний получил наибольшую популярность, так как при большом количестве возможных решений можно наложить ограничения, дающие оптимальное решение.

В качестве исходных данных для расчета выступают координаты начальной X_0 и конечной X_1 точек траектории, координаты базы робота O относительно мировой системы координат, которые представлены векторами в следующем виде:

$$X_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ i_0 \\ j_0 \\ k_0 \end{bmatrix}, X_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ i_1 \\ j_1 \\ k_1 \end{bmatrix}, O = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (16)$$

2.5 Предобработка изображения

Очень важно в контексте выбранных алгоритмов обеспечить качественные выходные данные для них. Этого можно достичь при помощи предобработки входного изображения.

Фильтрация в пространственной может рассматриваться, как некоторый оператор действующий на окрестность точки (x,y) на изображении

$$g(x,y) = T[f(x,y)] \quad (17)$$
$$T: R^{mxn} \rightarrow R$$

f – исходное изображение размером NxM,

T– оператор, действующий на область mxn изображения f в окрестности точки (x,y)

g– результирующее фильтрованное изображение

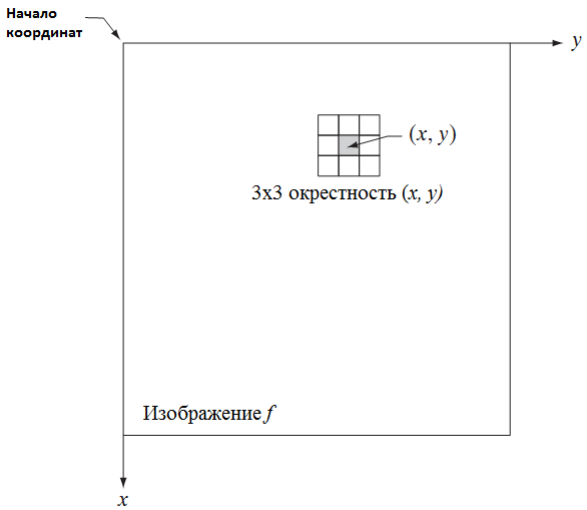


Рисунок 2.11 – Фильтрация в пространственной области

1. Понижение шума:

- Фильтрация: применение фильтров, таких как медианный, гауссовский или билатеральный, для удаления шумов изображения.

2. Улучшение контрастности и резкости:

- Изменение яркости и контраста: коррекция яркости и контраста для улучшения визуального эффекта и облегчения дальнейшего анализа.

- Увеличение резкости: увеличение резкости изображения для более четкого отображения особенностей объектов.

3. Удаление фонов и объектов в заднем плане:

- Бинаризация изображения: преобразование изображения в черно-белое, чтобы выделить объекты от фона.

- Вычитание фона: методы для удаления фона из изображения, например, с помощью вычитания фона или алгоритмов машинного обучения.

4. Устранение искажений и коррекция:

- Искажение перспективы и виньетирование: коррекция искажений, возникающих из-за перспективы и объективов камер.

- Цветовая коррекция: приведение цветового различия между изображениями к стандартным значениям, если нужно.

5. Сегментация объектов:

- Границы и контуры: выделение контуров объектов для дальнейшей сегментации и анализа.

- Кластеризация: разделение изображения на различные кластеры для выделения объектов разного типа или цвета.

6. Улучшение освещения:

- Нормализация освещения: методы для устранения изменчивости освещения и выделения объектов на изображении.

Для достижения удовлетворительного результата, необходимого для корректной и стабильной работы выбранных алгоритмов выберем следующие методы:

Гауссов фильтр [7]– фильтр ядром которого является двумерный гессиан

$$w(i,j) = e^{-\left(\frac{i^2+j^2}{2\sigma}\right)} \quad (18)$$

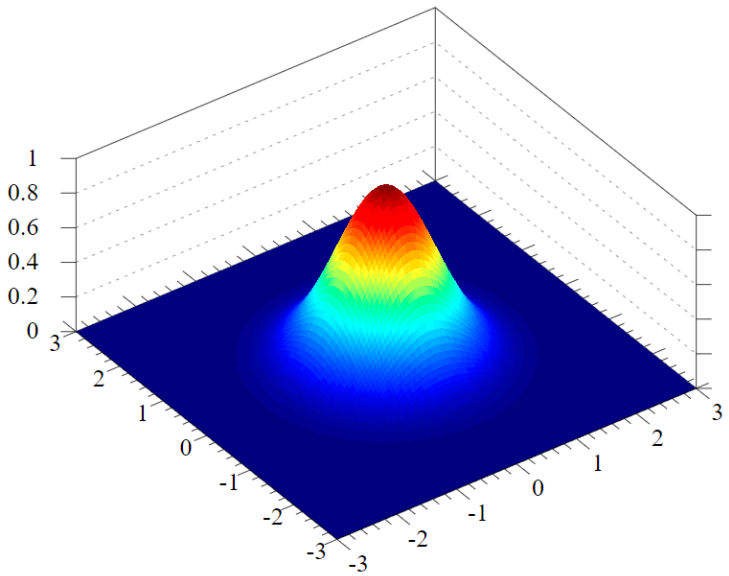


Рисунок 2.12 – График двумерной функции фильтра Гаусса

Сам процесс фильтрации представлен нормированной формулой выше (17).

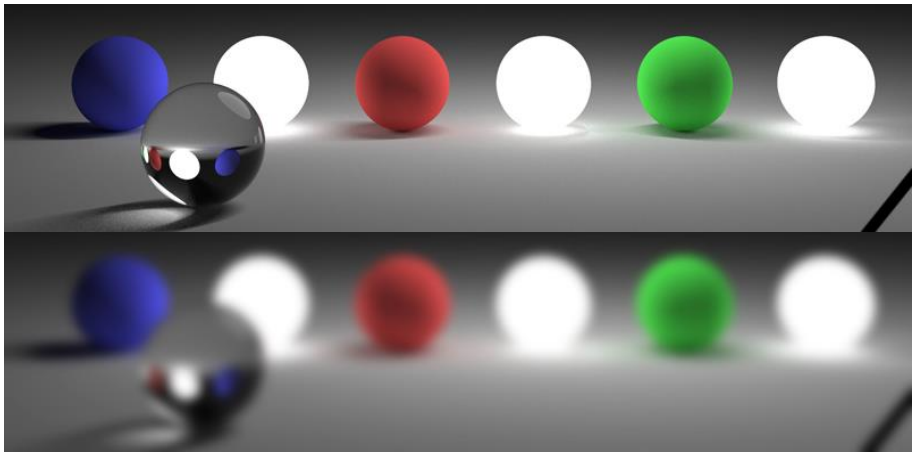


Рисунок 2.13 – Пример фильтрации Гаусса

Фильтрация по уровню интенсивности – на выходе данного фильтра остаются только те пиксели изображения, интенсивность которых попадает в заданный диапазон

$$g(x,y) = \begin{cases} f(x,y), level_{max} \le f(x,y) \le level\_max \\ 0 \end{cases} \quad (19)$$

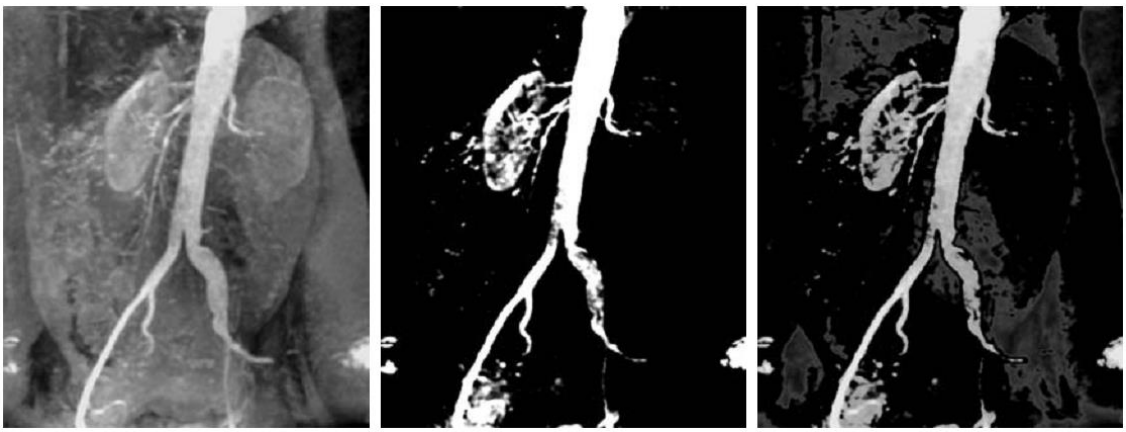


Рисунок 2.14 – Пример фильтрации изображения по уровню (выделение сосудов на снимке)

Нормализация интенсивности изображения – процедура, при которой интенсивность каждого пикселя умножается на константу, приводящее к масштабированию интенсивностей к максимальному диапазону формата представления данных

$$g(x,y) = f(x,y) \cdot \frac{255}{\max f} \quad (20)$$

Изменение средней яркости изображения – добавление фиксированного значения яркости к каждому пикселю

$$g(x,y) = f(x,y) + \mathrm{c} \tag{21}$$

где c – постоянный сдвиг интенсивности.

Гамма коррекция

$$g(x,y) = \mathrm{c} \cdot (f(x,y) + \epsilon)^{\gamma} \tag{22}$$
$$\gamma, c = const > 0$$

– малая величина, если необходим не нулевое значение на выходе при нулевом входе

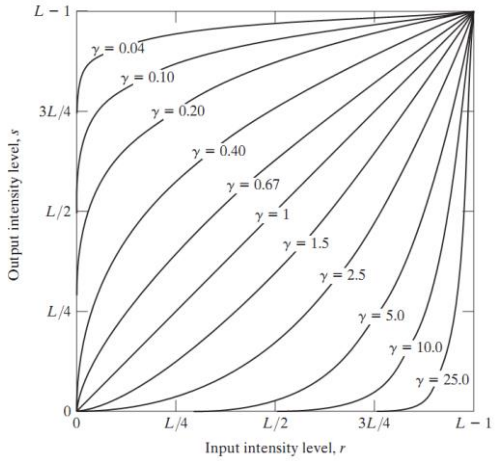


Рисунок 2.15 – Функция гамма преобразования интенсивности

Рисунок 2.16 – Справа исходное изображение, слева – после гамма преобразования

2.5 Выводы по главе 2

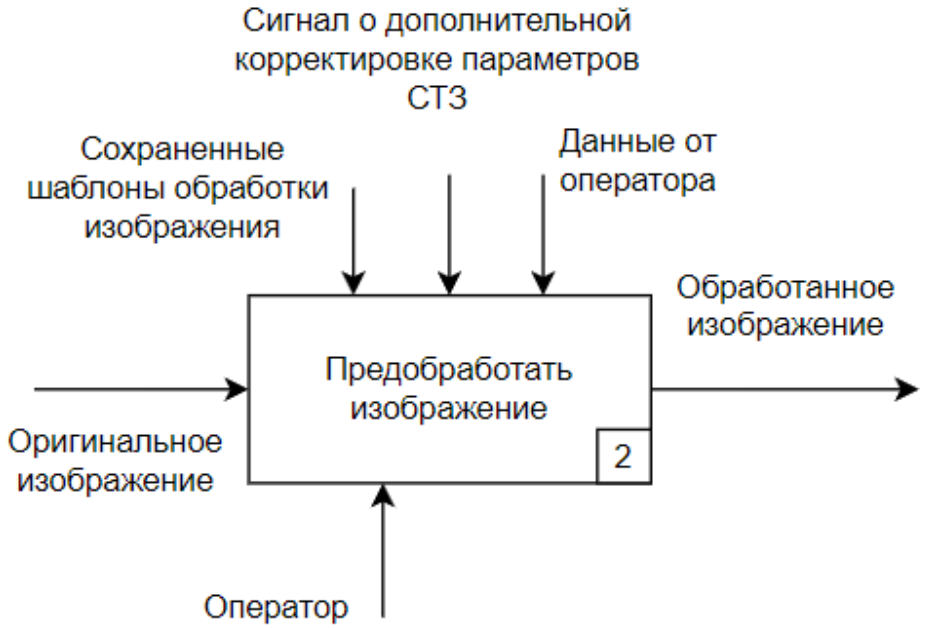
В этой главе были рассмотрены математические представления используемых методов.

Было разработано несколько собственных алгоритмов обработки изображения. Следующий шаг – разработать и реализовать целую систему для обнаружения, настройки, преобразования, отправки данных и движения робота, которая объединит в себе все вышеописанные этапы.

3 Разработка алгоритма СТЗ и управления роботом

3.1Предобработка входного изображения

При реализации каждого блока системы (рисунок 1.7) необходимо определить, что мы будем получать на вход, и какие данные будем передавать следующему блоку (функции или алгоритма). Для четкого определения этих данных воспользуемся методологией функционального моделирования IDEF0 или же SADT (Structural Analysis and Design Technique).



A-2	ПРЕДОБРАБОТАТЬ ИЗОБРАЖЕНИЕ	

Рисунок 3.1 – Верхний уровень алгоритма предобработки изображения

Теперь декомпозируем систему. Необходимый уровень декомпозиции определим так: будем считать систему декомпозированной при достижении уровня описания математической модели, представленной в главе 2.

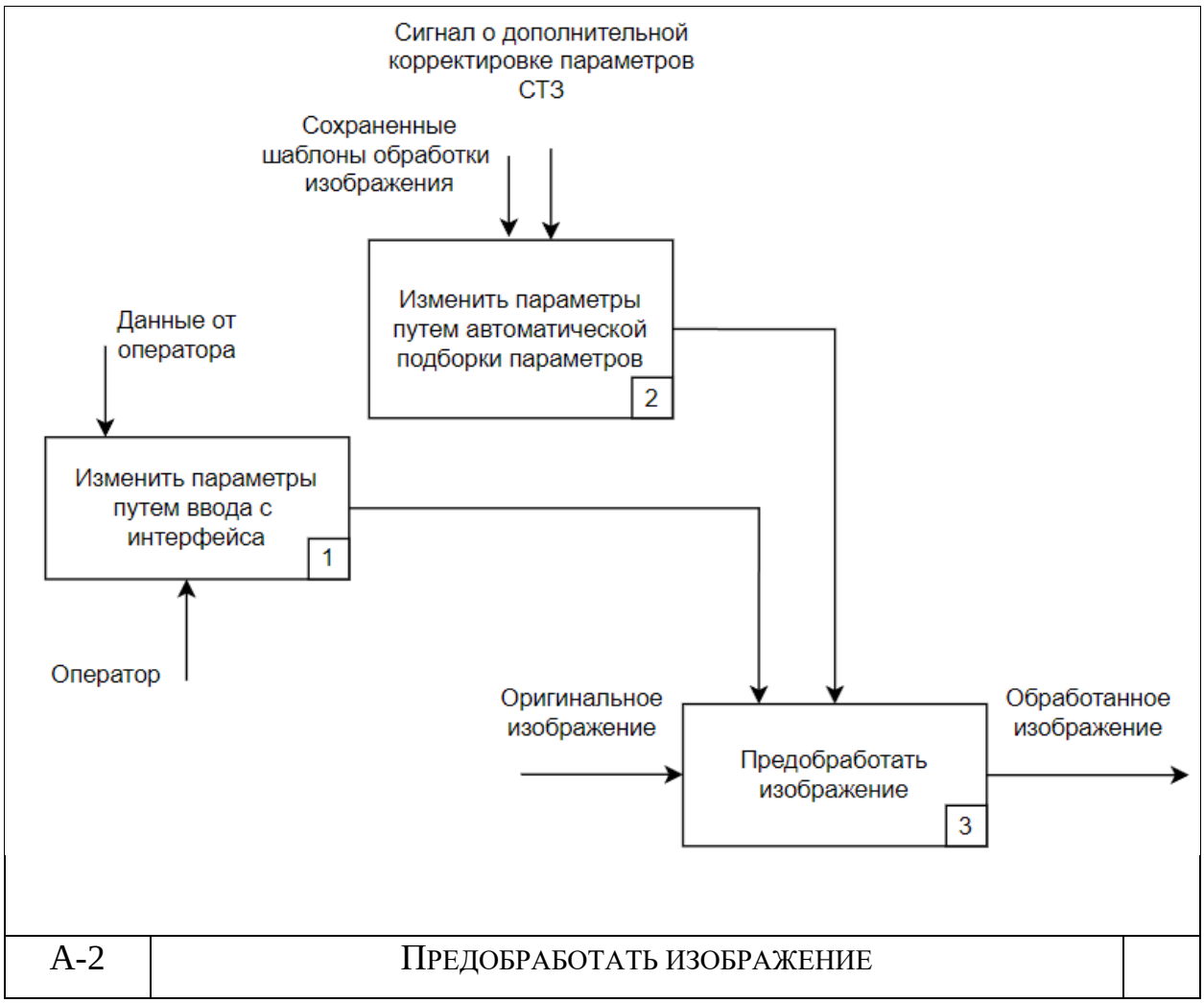


Рисунок 3.2 – Первый уровень декомпозиции алгоритма предобработки изображения

Графический интерфейс реализуется простым инструментом OpenCV при помощи ползунков, значения которых постоянно записываются в отдельный json файл, а также передаются в функцию применения фильтров. Эта функция математически подробно описана в главе 2.6. Получение изображения с камеры также просто реализуется встроенной функцией OpenCV (камера подключена по USB).

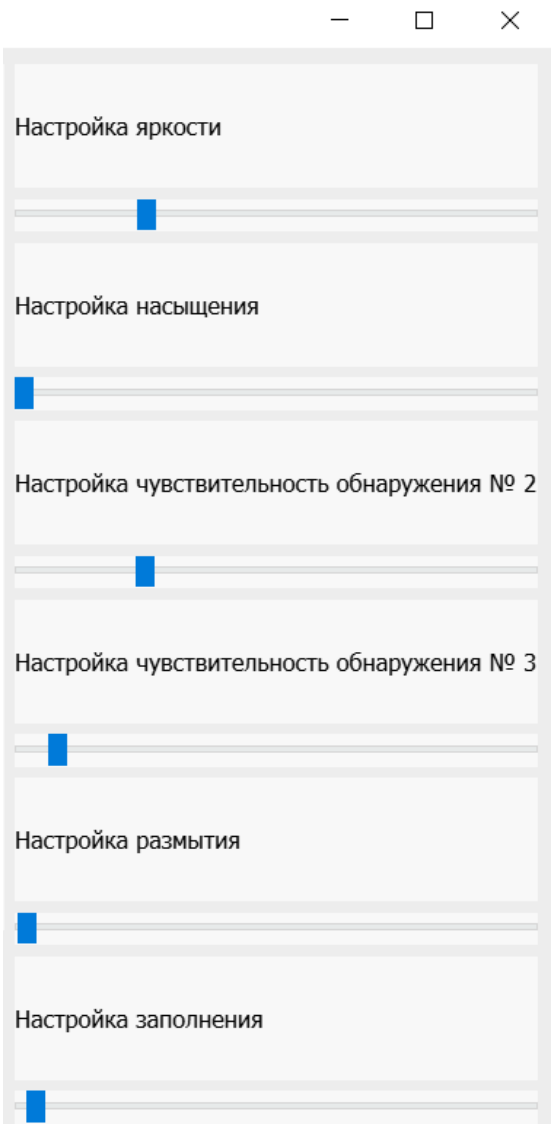


Рисунок 3.3 – Графический интерфейс настройки фильтров изображения

3.2Архитектура системы распознавания

Архитектуру системы распознавания также представим в виде SADT диаграмм.

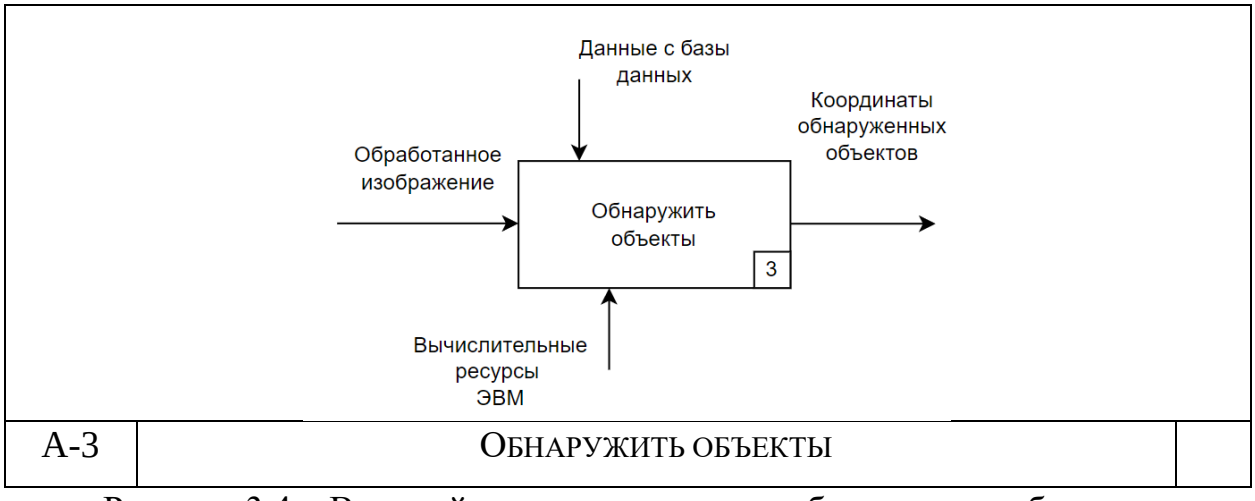
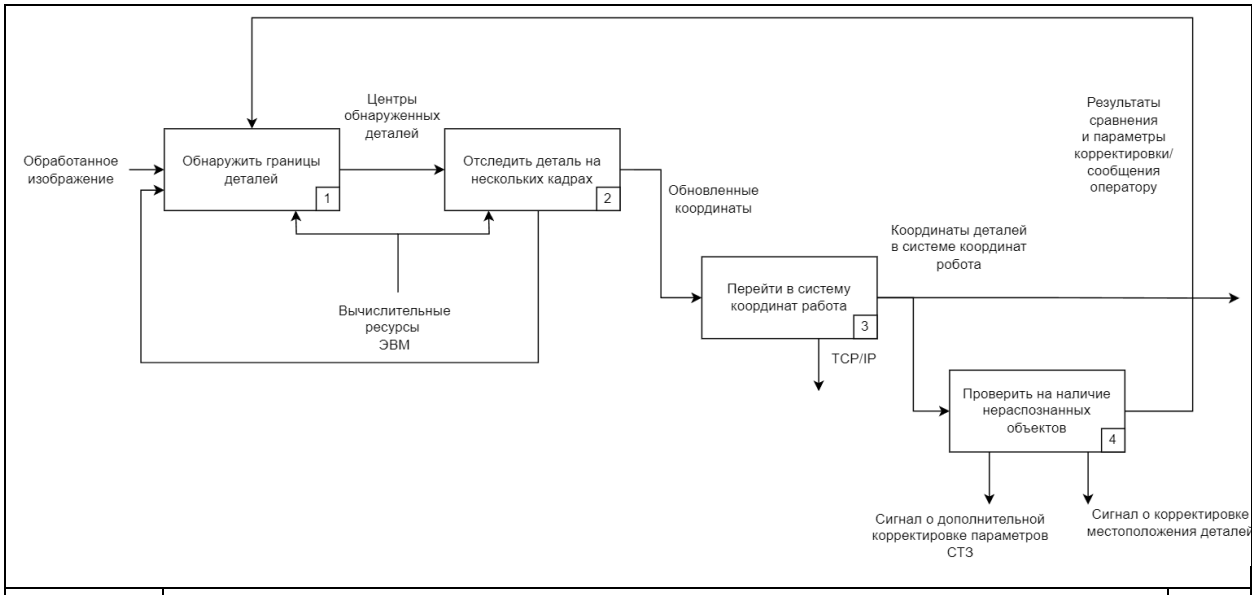


Рисунок 3.4 – Верхний уровень алгоритма обнаружения объектов

Теперь декомпозируем систему.



A-3	ОБНАРУЖИТЬ ОБЪЕКТЫ	

Рисунок 3.5 – Первый уровень декомпозиции алгоритма обнаружения объектов

В блоке А-31 происходит обнаружение контуров на кадре изображения:

1) Инициализация переменных: В начале функции инициализируются списки self.centers, self.angels, а также создаются переменные для работы с изображением.
2) Преобразование изображения: Исходный кадр преобразуется в оттенки серого с помощью cv2.cvtColor().
3) Применение адаптивного порогового преобразования: Применяется адаптивное пороговое преобразование для выделения объектов на изображении.
4) Фильтрация шума и улучшение контраста: Применяются операции медианного размытия, гистограммного выравнивания и дилатации/эрозии для улучшения качества контуров.
5) Поиск контуров: С помощью cv2.findContours() находятся контуры на изображении.

6) Обработка контуров: Происходит обход найденных контуров. Вычисляются их центры масс и площади. Если условия по площади и расположению центра масс удовлетворяют, контур добавляется в список.

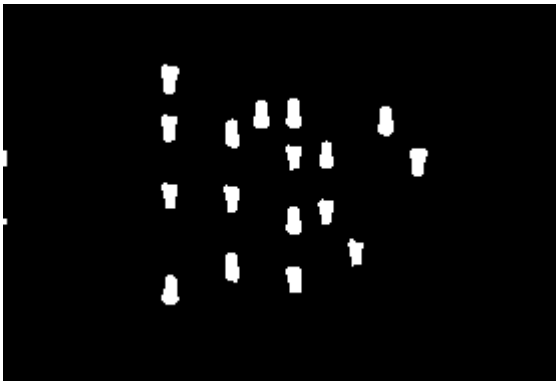


Рисунок 3.4 – Обнаруженные контуры

7) Дополнительная обработка контуров: Для каждого контура определяется его ориентация (направление) и тип. Также контуры рисуются на соответствующих изображениях.
8) Отображение результатов: Результаты отображаются на экране в виде нескольких окон с изображениями.
9) Возврат результатов: Функция возвращает исходный кадр изображения, координаты центров контуров и их ориентации.

Блок А-32 предназначен для уменьшение шанса ошибки обнаружения, сравнивая массивы получившихся координат с нескольких кадров.

Блок А-33 реализован посредством применения заранее записанных параметров. Эти параметры создаются в модулях calibration_zone.py и calibration_chess.py, они работают следующим образом:

calibration_chess.py выполняет калибровку камеры с использованием шахматной доски.

ChessboardCalibrator Class: Этот класс предназначен для калибровки камеры. Параметры калибровки загружаются из файла. При создании экземпляра класса задаются основные параметры: размеры шахматной доски, критерии для поиска углов доски, инициализация массивов для

хранения объектных и изображенческих точек, матрица камеры и коэффициенты дисторсии.

calibrate_image() Method: Этот метод калибрует изображение, используя шахматную доску. Если калибровка еще не проведена, метод пытается найти углы шахматной доски на изображении. Если углы найдены, они уточняются с использованием cornerSubPix(), а затем добавляются в массив объектных точек. После накопления достаточного количества изображений для калибровки, метод вызывает calibrate().

calibrate() Method: Этот метод проводит калибровку камеры. Он использует cv2.calibrateCamera(), чтобы получить матрицу камеры и коэффициенты дисторсии. Если калибровка прошла успешно, параметры сохраняются в JSON-файл.

Main Block: В главном блоке кода открывается камера и начинается захват изображений. Для каждого изображения применяется предварительное преобразование перспективы, чтобы выровнять изображение с шахматной доской. Затем вызывается метод calibrate_image(), чтобы калибровать изображение с использованием шахматной доски. Откалиброванное изображение отображается на экране.

В целом, код выполняет следующие шаги:

Загрузка параметров калибровки из файла.

Нахождение углов шахматной доски на изображениях.

Уточнение углов и добавление их в массивы точек.

Проведение калибровки камеры и сохранение результатов.

Применение калибровки к изображениям с использованием шахматной доски.

calibration_zone.py выполняет обнаружение маркеров ArUco на изображении и преобразование перспективы для выделения области интереса (ROI).

Marker Class: Этот класс представляет маркер ArUco на изображении. Он содержит идентификатор маркера, его центр и углы.

ImageProcessor Class: Этот класс содержит методы для обработки изображения. detectArucoMarkers() обнаруживает маркеры ArUco на изображении с использованием предопределенного словаря и параметров детектора. cropImage() вырезает область интереса (ROI) из изображения на основе переданных точек и сохраняет преобразование перспективы для дальнейшего использования.

Main Block: В главном блоке кода открывается камера и начинается захват изображений. Для каждого захваченного изображения производится обнаружение маркеров ArUco и выделение ROI. Выделенная область отображается на экране.

calibration_zone.py включает обнаружение маркеров ArUco на изображении, а затем вырезает и сохраняет область, которая содержит интересующий нас объект, используя преобразование перспективы.

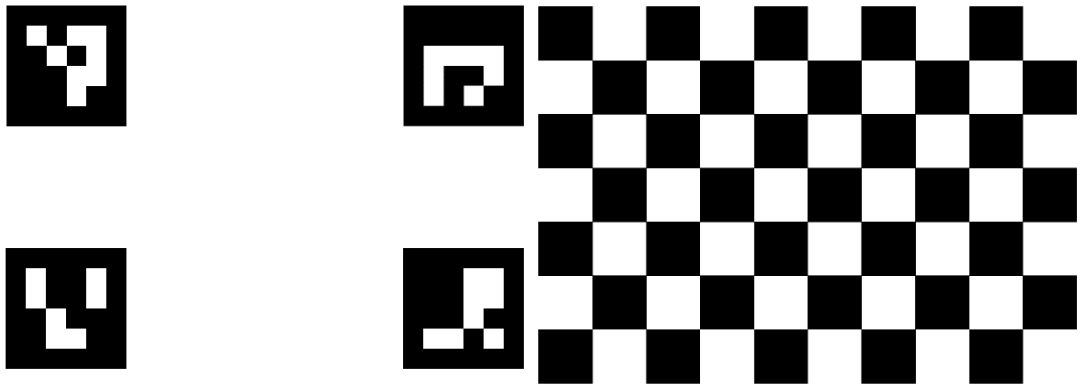


Рисунок 3.5 – Изображения для калибровки

3.3 Разработка алгоритма движения робота

Движение 6-ти осевого промышленного робота Kawasaki реализовано перемещением от «точки к точке» при помощи матриц перемещений, описанных в главе 2. Также робот умеет обрабатывать сигналы, о которых подробнее будет рассказано позже

Расчет траектории происходит автоматически на стороне робота. Задача сводится к правильному создания точек перемещения робота с учетом его геометрии. Всего можно обозначить 5 точек и 2 сигнала:

«home» - точка, обозначающая домашнюю позицию, она находится над зоной захвата, считается безопасной точкой начала и окончания цикла перемещения.

«pregrab» - точка, обозначающая позицию непосредственно перед захватом, располагается на высоте 20 мм над поверхностью палеты, необходима для устранения неопределенности движения робота, при выполнении программы захвата заготовки.

«grab» - точка, обозначающая позицию захвата, когда фланец робота с прикрепленном захватом соприкасается с деталью и включается сигнал «clam_on», который активирует пневмораспределитель, сжимая пальцы захвата.

«zone_0» - точка, обозначающая позицию перед точкой сброса детали в обозначенную зону. Также необходима для устранения неопределенности движения робота, при движении к точке сброса.

«zone_1» - точка сброса заготовки, в ней сигнал «clam_on» выключается, и, как следствие, деталь падает. Располагается на высоте 20 мм над поверхностью для безопасности.

«clam_on» - сигнал, отвечающий за включение и выключение пневмораспределителя.

«coord_add» - сигнал, отвечающий за поступление новых координат на робота, при его включении запускается основная программа.

Весь код по шагам выглядит так:

1) Определение текущего инструмента (TOOL или захват – определяется его геометрия)
2) Установка скорости движения - 100 MM/S
3) Выполнение автоматического запуска программы autostart.pc – программа связи по TCP/IP и получению координат
4) Перемещение в позицию «home»
5) Начало бесконечного цикла
6) Если сигнал «coord_add» равен TRUE, выполнить следующий блок
7) Вызов подпрограммы Calc_coodr.PC – преобразование координат
8) Если координата X больше 190, выполнить следующий блок и если координата Y больше 330, выполнить следующий блок – блок безопасности – если придет значение за пределами этих чисел, то движения не произойдет, так как это будет за пределами контура безопасности.
9) Перемещение в позицию «pre_grab»
10) Установка новой скорости движения - 30 MM/S на следующее действие
11) Включение сигнала «clam_on»
12) Перемещение в позицию «grab»
13) Перемещение в позицию «zone_1»
14) Перемещение в позицию «zone_2»
15) Установка новой скорости движения - 30 MM/S на следующее действие
16) Опустить фланец на 70 мм
17) Выключение сигнала «clam_on»
18) Поднять фланец на 70 мм
19) Выключение сигнала «coord_add»
20) Перемещение в позицию «home»

3.4 Реализация передачи данных

Протоколы TCP/IP (Transmission Control Protocol/Internet Protocol) являются основой современного интернета и сетей компьютеров. В проекте был использован TPC/IP.

TCP/IP следует модели OSI (Open Systems Interconnection), которая разделяет сетевую коммуникацию на уровни. Однако, TCP/IP обычно описывается в контексте четырех основных уровней: уровень доступа к сети, уровень интернета, транспортный уровень и прикладной уровень.

IP-адресация: В TCP/IP каждое устройство в сети имеет уникальный IP-адрес, который используется для идентификации и маршрутизации. IP-адрес состоит из 32 или 128 бит и представляет собой числовое значение, разделенное точками (IPv4) или двоеточиями (IPv6).

Протоколы TCP и UDP: TCP/IP включает в себя два основных протокола транспортного уровня: TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). TCP обеспечивает надежную передачу данных путем установления соединения, подтверждения доставки и контроля потока. UDP, с другой стороны, является более простым и обеспечивает ненадежную доставку данных без установления соединения и контроля ошибок, что не позволяет его использовать, так как необходимо обеспечить синхронизацию и уверенность в том, что как сервер, так и клиент будут знать о состоянии друг друга.

Интернет-адресация и маршрутизация: Протокол IP (Internet Protocol) отвечает за адресацию и маршрутизацию данных в сети. IP использует IP-адреса для определения источника и назначения данных, а также для маршрутизации их через сеть от отправителя к получателю.

Использование портов: TCP и UDP используют порты для идентификации конечных точек в сетевом соединении. Порты позволяют множеству приложений работать на одном устройстве, принимая данные от разных источников. Каждый порт связан с определенным протоколом и может быть открыт для входящих или исходящих соединений.

Код реализует асинхронный сервер, который обрабатывает подключения клиентов и асинхронно принимает и отправляет сообщения. Кроме того, он создает отдельный поток для выполнения этого сервера.

Robot Class: Этот класс представляет сервер. У него есть методы для обработки клиентов (handle_client()), отправки сообщений (send_message()) и запуска сервера (start_server()). Когда клиент подключается, вызывается метод handle_client(), который ожидает данные от клиента и отвечает на них.

run_server() Function: Это функция, которая создает экземпляр класса Robot и запускает его метод start_server() в основном потоке.

В данной реализации связи используются asyncio и потоки (threads).

Asyncio:

В коде используется библиотека asyncio для реализации асинхронного сервера. asyncio позволяет создавать асинхронные функции и корутины, которые могут выполняться параллельно без блокировки потока исполнения.

В методе handle_client() класса Robot используется асинхронное ожидание (await) операций ввода-вывода, таких как чтение данных от клиента. Это позволяет серверу эффективно обрабатывать несколько клиентов одновременно, не блокируя поток выполнения.

Метод start_server() запускает сервер в цикле бесконечно, чтобы он мог принимать подключения клиентов асинхронно.

В функции run_server() мы запускаем цикл событий asyncio, используя функцию asyncio.run(), которая запускает событийный цикл в основном потоке. Это позволяет асинхронному серверу работать параллельно с остальным кодом.

Threads:

Помимо асинхронности, в коде также используется поток (threading.Thread), чтобы запустить асинхронный сервер в отдельном потоке.

Функция run_server() запускается в отдельном потоке с помощью threading.Thread. Это делается для того, чтобы асинхронный сервер мог работать параллельно с остальным кодом в основном потоке. Если бы мы запустили сервер прямо в основном потоке, он мог бы заблокировать выполнение остального кода, так как asyncio.run() блокирует выполнение, пока цикл событий не завершится.

Запуск сервера в отдельном потоке позволяет основному потоку выполнения продолжать работу (например, выполнение других задач или обработка пользовательского ввода), в то время как сервер обрабатывает входящие подключения клиентов в своем собственном потоке.

Ответная часть на роботе реализована на языке AS, специального языка для роботов Kawasaki. Там также существуют несколько связанных потоков, один из которых реализует TCP клиент. Обе части программы обмениваются сообщениями о своем состоянии, робот отправляет флаги (двигаюсь, ожидаю и т.д.).

3.5 Реализация графического интерфейса

Реализован графический интерфейс с использованием библиотеки PyQt6. Основные компоненты интерфейса:

Главное окно (QMainWindow):

Создается класс VideoPlayer, унаследованный от QMainWindow.

В методе __init_main_window создается главное окно приложения, устанавливается в качестве центрального виджета и задаются его размеры и название.

Виджеты (QWidget, QLabel, QPushButton, QSlider):

В методе __init_widgets создаются все виджеты, необходимые для интерфейса приложения.

QLabel используется для отображения видео с камеры (video_label) и информации о деталях (detect_detail_label).

QPushButton используется для кнопок управления (start_button, stop_button, continue_button, statisticks_button, auto_correction_button).

QSlider используется для регулировки параметров обработки видео (brigh_fac_slider, sat_fac_slider, threshold_2_slider, threshold_3_slider, blur_slider, dilate_slider).

Расположение виджетов (QHBoxLayout, QVBoxLayout):

Создаются различные компоновщики для управления расположением виджетов: main_layout, video_and_info_layout, vid_info_and_dop_layout, dop_layout, calibration_layout.

Виджеты добавляются в компоновщики с помощью методов addWidget и addLayout, чтобы распределить их по главному окну.

События и обработчики (clicked.connect, valueChanged.connect):

Для кнопок управления и слайдеров устанавливаются обработчики событий.

Например, при нажатии на кнопку stop_button вызывается метод send_em_stop, который отправляет сообщение остановки роботу.

При изменении значения слайдера вызывается метод on_slider_value_changed, который обновляет параметры обработки видео и сохраняет их в файл.

Обновление виджетов:

Метод update_frame вызывается по таймеру и обновляет изображение в виджете video_label с помощью метода setPixmap.

Также обновляется информация о деталях и другие виджеты, в зависимости от состояния приложения.

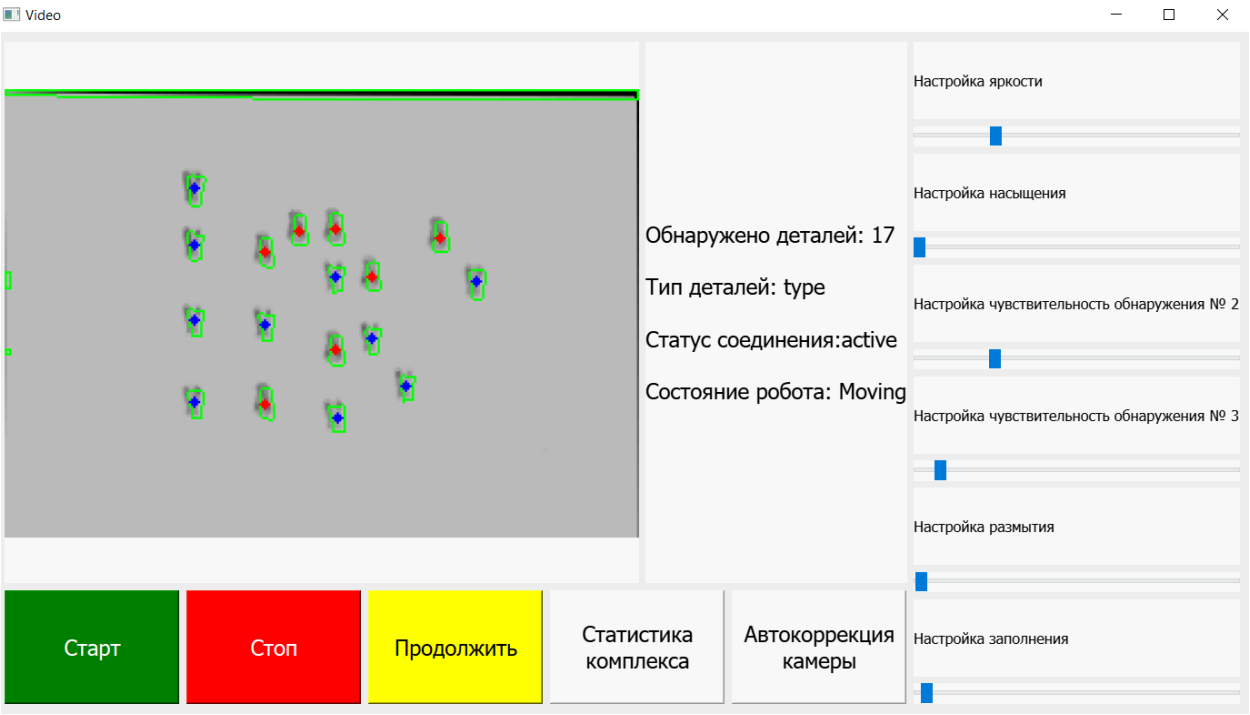


Рисунок 3.5 – Графический интерфейс

3.6 Реализация базы данных

При разработке базы данных для хранения статистики о работе комплекса с использованием СУБД и наличием выделенного сервера, процесс начинается с анализа требований:

База данных должна хранить статистику о работе комплекса:

- Номер партии деталей
- Время начала сортировки
- Время сортировки партии
- Количество деталей в партии
- Количество бракованных деталей
- Количество ошибок во время работы

При отсутствии удаленного подключения к серверу данные для БД записываются в .csv

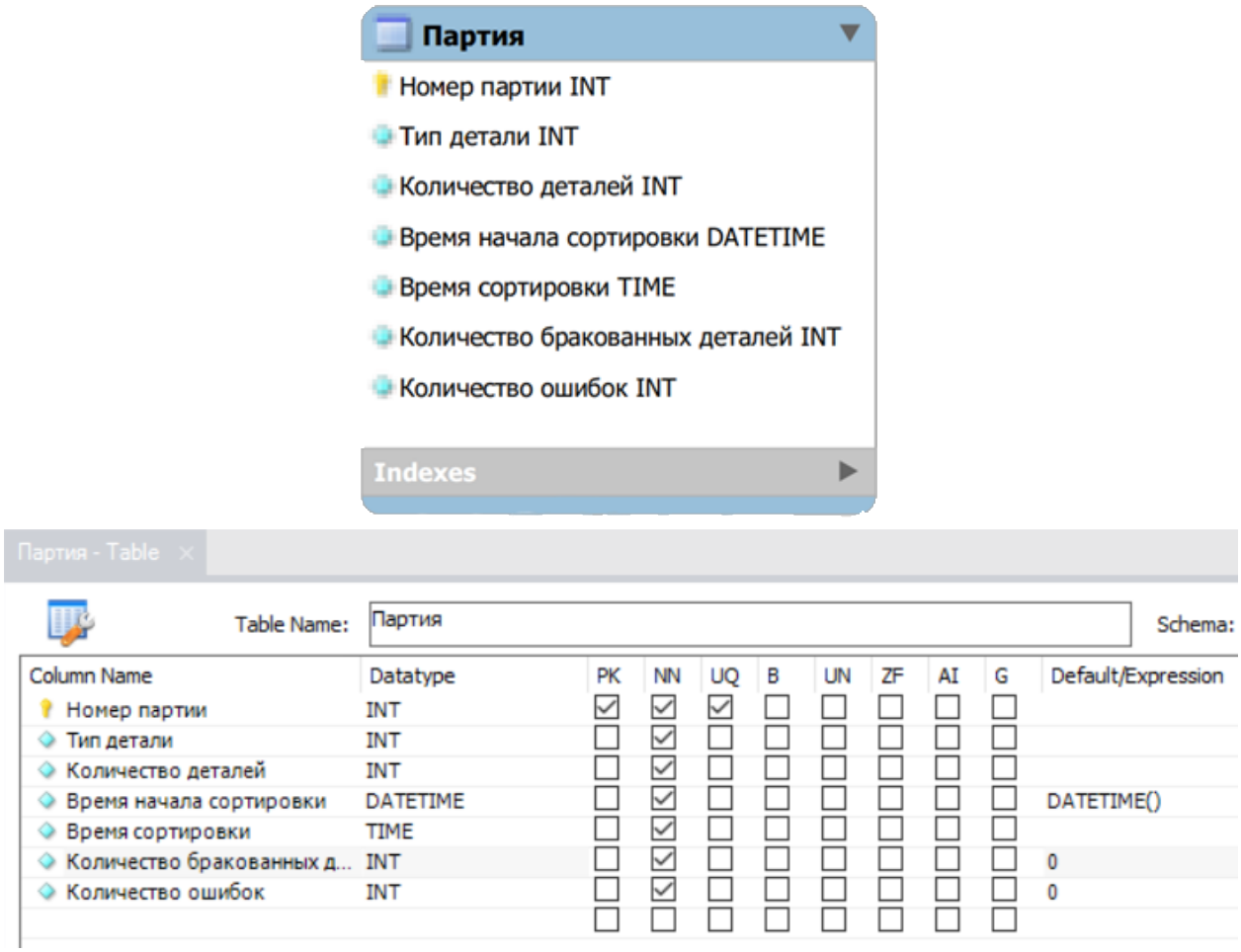


Рисунок 3.6 – Таблица базы данных

Для реализации была выбрана СУБД MySQL, ввиду особенностей относительно других систем управлений.

Простота использования: MySQL, подобно SQLite, обычно считается более простым в использовании и настройке по сравнению с PostgreSQL.

Совместимость с различными языками программирования: MySQL широко используется во множестве языков программирования и сред разработки, что облегчает его интеграцию. SQLite также поддерживается во многих языках, но он чаще используется для приложений, работающих в рамках одного процесса.

Поддержка распределенных систем: MySQL имеет более простую реализацию распределенных систем, что может быть полезно, если в

будущем появится необходимость масштабировать базу данных на несколько серверов.

3.7 Выводы по главе 3

В данной главе был спроектирован робототехнический комплекс с использованием технического зрения. Была проделана следующая работа:

- Была разработана архитектура системы распознавания объектов на видео, а также алгоритм предобработки изображения.
- Была написана программа движения робота и система безопасности его движения.
- Был разработан графический интерфейс для пользователя.
- Была спроектированная база данных
- Была спроектирована система передачи данных

Следующий шаг – экспериментальное исследование разработанных решений.

4 Экспериментальное исследование

4.5 Программа экспериментов

Тестирование проходило несколько часов работы. Необходимо было обнаружить и передать координаты детали для сортировки роботу.

Как было сказано ранее основными показателя качества является скорость сортировки (относительного ручного труда), простота программной реализации и количество потребляемого ресурса ЭВМ, которые и будут оцениваться в ходе экспериментов.

Данные характеристики зависят от следующих параметров:

- Количество отсортированных деталей в час
- Общая скорость выполнения полного цикла сортировки
- Ошибка обнаружения или пропущенная детали
- Объем потребления ресурса ЭВМ

В ходе тестирования были получены результаты, приведенные в таблице 4.2

Параметр	Первый день	Второй день	Третий день	Четвертый день
Количество отсортированных паллет (каждая содержит 40 деталей) с деталями в час (кол-во)	100	80	60	40
Ошибка (кол-во деталей)	43	36	24	19
Неотсортированные детали (%)	0.0107%	0.0112%	0.01%	0.0118%
Общая скорость выполнения (с)	7,80	8,59	7,3	7,85
Общее время сортировки	8 часа 40 минут	7 часов 38 минут	4 часов 52 минут	3 часа 29 минут
Эффективность (деталей в час)	461	436	493	459

Таблица 4.2 - Результаты экспериментов

4.6 Анализ качества алгоритма обнаружения

1) Необходимо провести дополнительные тесты для выявления причин изменений в количестве ошибок и процента неотсортированных деталей.
2) Улучшение алгоритмов обнаружения деталей должно быть направлено на уменьшение ошибок и повышение эффективности сортировки.
3) Регулярное обновление и настройка алгоритмов обнаружения деталей важны для поддержания высокого уровня производительности системы сортировки.
4) Возможно, также стоит рассмотреть внедрение новых технологий или методов обнаружения деталей для улучшения точности и скорости работы системы.

Общий вывод состоит в том, что алгоритм обнаружения деталей играет критическую роль в эффективности и надежности роботизированной системы сортировки, и его постоянное совершенствование является ключевым для успешной работы системы.

4.7 Выводы по главе 4

В ходе экспериментов была доказана целесообразность использования комплекса для обозначенных в первой главе задач, если сравнивать с человеческими возможностями, то:

Человек, в среднем, за 4-х часовую смену будет сортировать 400 деталей в час. Что в общем случае соизмеримо с результатами робота, но преимущество последнего в возможности его эксплуатации без перерывов.

Также проведены эксперименты для оценки показателей качества распознавания.

Заключение

В ходе работы был разработан и протестирован робототехнический комплекс с использованием технического зрения. Для системы были разработаны: система распознавания объектов на видео, а также алгоритм предобработки изображения, программа движения робота и система безопасности его движения, графический интерфейс, реализована связь по TCP/IP для пользователя, а также система была успешно протестирована.

Были рассмотрены несколько алгоритмов для обнаружения необходимых нам деталей, использующие алгоритмический подход. Было проведено сравнение данных различных методов и разобрана их математическая сложность.

Для системы технического зрения разработаны следующие компоненты:

- Предобработка
- Поиск деталей существующими методами
- Определение ориентации деталей
- Отслеживание обнаруженных объектов

Результаты экспериментов по оценке качества показывают рациональность использования данной системы для увеличения эффективности сортировки объектов.

Перспективы дальнейшего развития работы:

- Изучение возможности замены робота на одного линейки высокоскоростных или роботов-сортировщиков
- Реализация алгоритма автоматической коррекции цвета и света на камере

Список использованных источников

1. Дэвид Форсайт, Жан Понс. Компьютерное зрение. Современный подход = Computer Vision: A Modern Approach. — М.: «Вильямс», 2004.
2. Л. Шапиро, Дж. Стокман. Компьютерное зрение = Computer Vision. М.: Бином. Лаборатория знаний, 2006.
3. M. Nielsen, “Neural Networks and Deep Learning”, Determintation Press, 2015 http://neuralnetworksanddeeplearning.com.
4. W. Dai, C. Dai, S. Qu, J. Li, S. Das “Very Deep Convolutional Neural Networks for Raw Waveforms”, ICASSP, 2017 https://arxiv.org/abs/1610.00087
5. J. Yosinski “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop, 2015 http://yosinski.com/deepvis
6. Brief summary of YOLOv8 model structure // Github.com (https://github.com/ultralytics/ultralytics/issues/189https://www.dynamsoft.com/codepool/data-matrix-reading-benchmark-and-comparison.html)
7. Ляхов, П. А. Разработка двумерных сглаживающих фильтров на основе функции специального вида / П. А. Ляхов, Ю. В. Голошубова, Е. А. Попова, М. В. Валуева. — Текст : непосредственный // Молодой ученый. — 2017. — № 22 (156). — С. 48-52. — URL: https://moluch.ru/archive/156/43984/ (дата обращения: 15.01.2024).
8. В. А. Фурсов, С. А. Бибиков, П. Ю. Якимов // Локализация контуров объектов на изображениях при вариациях масштаба с использованием преобразования Хафа, Компьютерная оптика. - 2013
9. BY D. MARRAND E. HILDRETH Theory of edge detection- 1980
10. Сзелиски Р. Computer Vision: Algorithms and Applications: учеб. пособие. - М.: Наука, 2012. - 400 с.
11. Гонсалес Р. С., Вудс Р. Е. Digital Image Processing: учебник. - СПб.: Питер, 2018. - 672 с.
12. Корк П. Robotics, Vision and Control: Fundamental Algorithms in MATLAB: справочник. - Киев: Техника, 2015. - 512

13. Хартли Р., Зиссерман Э. Multiple View Geometry in Computer Vision: учебное пособие. - М.: МГТУ, 2008. - 560 с.
14. Сридхаран М. Motion Vision: Design of Compact Motion Sensing Solutions for Navigation of Autonomous Systems: монография. - М.: Издательство Роботехника, 2019. - 324 с.