МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ЭнМИ
Кафедра РМДиПМ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

Направление 15.03.06 Мехатроника и робототехника
(код и наименование)

Образовательная программа Компьютерные технологии управления в робототехнике и мехатронике

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Разработка метода автономной калибровки камер мобильного робота

Студент С-12-18 _____ Алексеев Д.А.
группа подпись фамилия и инициалы

Руководитель ВКР д.ф.-м.н. профессор _____ Кобрин А.И.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Внешний консультант
уч. степень должность подпись фамилия и инициалы

организация

«Работа допущена к защите»

Заведующий кафедрой д.т.н. доцент _____ Меркурьев И.В.
уч. степень звание подпись фамилия и инициалы

Дата 29.06.2022

Москва, 2022

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ЭнМИ
Кафедра РМДиПМ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(БАКАЛАВРСКУЮ РАБОТУ)

Направление 15.03.06 Мехатроника и робототехника
(код и наименование)

Образовательная программа Компьютерные технологии управления в робототехнике и мехатронике

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Разработка метода автономной калибровки камер мобильного робота

Студент С-12-18 Алексеев Д.А.
группа подпись фамилия и инициалы

Руководитель ВКР д.ф.-м.н. профессор Кобрин А.И.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Внешний консультант
уч. степень должность подпись фамилия и инициалы

организация

Заведующий кафедрой д.т.н. доцент Меркурьев И.В.
уч. степень звание подпись фамилия и инициалы

Место выполнения работы ФГБОУ ВО «НИУ «МЭИ»

СОДЕРЖАНИЕ РАЗДЕЛОВ ЗАДАНИЯ И ИСХОДНЫЕ ДАННЫЕ

Целью выпускной бакалаврской работы является разработка метода автономной предварительной калибровки встроенной камеры мобильного робота.
Среди существующих шаблонов наиболее популярным и перспективным методом калибровки камеры является использование плоских координатных меток. В результате должен быть осуществлен обзор систем координатных меток, выбраны потенциальные метки для калибровки камеры.
Главная проблема при калибровке камеры – поиск особых точек на изображении для расчета внутренних и внешних параметров.
Предлагается использовать калибровочный паттерн, на котором однозначно определяется положение точек и на изображении, и в пространстве.

ПЕРЕЧЕНЬ ГРАФИЧЕСКОГО МАТЕРИАЛА

Количество листов _____

Количество слайдов в презентации 12

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Zhengyou Z. A Flexible New Technique for Camera Calibration // Microsoft Research, One Microsoft Way, USA, Radmond. — 1998. – 22p.
2. Hartley R.I. Self-Calibration of Stationary Cameras // G.E. CRD, Schenectady, NY. – 17p.
3. Ю.Б. Блохинов, Е.Э. Андриенко, К.К. Казахмедов, Б.В. Вишняков Автоматическая калибровка системы видеокамер и лидаров для автономных мобильных комплексов, статья, Москва – 2021. – 14стр.
4. Р. Н. Сафин, Р. О. Лавренов, Р. А. Боби, С.К. Саха, Эксперименты по калибровке камер мобильного робота при наличии аппаратных дефектов в системе технического зрения статья, Казань –2018. – 10стр.

Примечания:

1. Задание брошюруется вместе с выпускной работой после титульного листа (страницы задания имеют номера 2, 3).
2. Отзыв руководителя, рецензия(и), отчет о проверке на объем заимствований и согласие студента на размещение работы в открытом доступе вкладываются в конверт (файловую папку) под обложкой работы.

АННОТАЦИЯ

В данной работе решается задача автоматической калибровки камер мобильного робота в случае непредвиденной поломки в процессе его работы. Главная проблема при калибровке камеры – поиск особых точек на изображении для расчета внутренних и внешних параметров. Это определяется отсутствием универсальности детекторов точек [9][16]. С другой стороны, мы используем калибровочный паттерн, на котором однозначно определяется положение точек на изображении и в пространстве. Тогда предлагается использовать объекты на пути следования робота для калибровки. В таком случае нашими задачами становятся: поиск объекта на изображении, нахождение на нем особых точек, нахождение внутренних и внешних параметров камеры.

Для поиска объекта предлагается использовать одноэтапную нейронную сеть семейства YOLO, так как они имеют высокий показатель числа обработки количества кадров в секунду. Для нахождения особых точек предлагается использовать совместную обработку сначала двухэтапной нейронной сетью, затем библиотекой OpenCV. Исследование показывает, что таким способом возможно откалибровать камеру.

ОГЛАВЛЕНИЕ

ОБОСНОВАНИЕ АКТУАЛЬНОСТИ

В современном мире широко используются камеры и стереокамеры, они нужны для определения объектов, для определения расстояния, навигации и так далее. К сожалению, техника, эксплуатируемая в неблагоприятных условиях, имеет тенденцию ломаться и изнашиваться, как, например, случай [11], когда стереопара вышла из строя из-за падения робота, а возможность выровнять механически отсутствует. Тогда можно провести калибровку камеры в «полевых» условиях. Но существующие методы калибровки не способны сделать это. Необходимо иметь калибровочный объект или нас ограничивает точность автоматической калибровки.

Тогда, зная в каких условиях будет использоваться робот, можно заранее загрузить калибровочные объекты, которые встречаются роботизированной платформе на пути и в случае внештатной ситуации использовать их.

Если же говорить про калибровку камеры в целом, то ее нужно производить всегда при пуско-наладочных работах, это исходит из того, что все камеры имеют различные внутренние параметры, чистая случайность, если мы возьмем пару камер одной модели даже из одной партии, и они окажутся идентичными.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОПРЕДЕЛЕНИЙ

Определения

Гомография: в компьютерном зрении отображение одной плоскости в другую.

Датасет: данные для обучения нейронной сети, набор картинок, разделенный по классам, может сопровождаться координатами расположения объектов.

Класс: один из множества образов, который будет выделен при распознавании изображения.

Метрическая камера: камера, используемая в аэрофотосъемке, зачастую имеет большой вес и высокое качество изображения.

Нейросеть: программная реализация математической модели функционирования и организации сети нервных клеток живого организма.

Обучение нейронной сети: процесс подбора весов в искусственной нейронной сети.

Свёрточная нейронная сеть: нейросеть, содержащая слои, наиболее подходящие для распознавания объектов на изображении.

Сокращения

ИИ – искусственный интеллект;

ПО – программное обеспечение;

СНС – свёрточная нейронная сеть;

FPN - Feature Pyramid Network;

ВВЕДЕНИЕ

Всякий раз, когда мы используем камеру в любой системе, например, автопилот автомобиля или камера на роботе манипуляторе, мы обязаны иметь качественную и точную камеру. Вот что из себя должна представлять такая камера:

1) Необходимо иметь достаточное качество изображения для поставленной задачи, именно достаточное, потому что нет необходимости или возможно вычислительной возможности использовать метрическую камеру для обработки информации в режиме реального времени.
2) Изображение, получаемое из камеры, не должно иметь каких-либо искажений или стремиться к тому, чтобы их было минимальное количество.

Все эти аспекты необходимы для получения более точного представления реального мира в цифровом виде.

Калибровка – процесс нахождения внешних и внутренних параметров камеры. Важно уточнить, что каждая камера калибруется индивидуально. Это исходит из того, что существующий уровень технологий не способен производить одинаковую технику. Даже если мы говорим про один и тот же завод, про одну партию, про одну модель, все камеры будут иметь различные внутренние параметры, чистая удача, если мы случайно возьмем несколько камер, и они окажутся действительно идентичными.

Существующие методы калибровки стереопары можно разделить на 2 вида: с использованием калибровочного объекта и без него. В целом, основной проблемой калибровки является поиск особых точек на изображении. Если мы используем калибровочный объект, то тут все понятно, мы заранее пишем ПО под него, но если же мы не пользуемся калибровочным паттерном, то все что у нас есть это набор пикселей и их значений от черного «0» до белого «255»

или аналогично для RGB изображений. К сожалению, на момент написания работы определение особых точек без использования шаблонов остается открытым вопросом.

Калибровка обязательна при пуско-наладочных работах – это факт. Но мобильная платформа, на которой установлена стереокамера, может эксплуатироваться в неблагоприятных условиях, например, на рисунке 1.1 и 1.2 показаны одни из случаев, когда мобильная платформа может повредить стереопару.

Рис. 1.1 Робот-курьер Ровер «Яндекса»

Рис 1.2 Неправильное использование Робота-курьера Ровер

Это может быть падение с бордюра или лестницы, например, для беспилотных автомобилей это может оказаться пропущенная кочка на дороге. При постройке робота, который работает в тесном контакте с людьми, всегда необходимо учитывать человеческий фактор (рисунок 1.2). Именно эта проблема решается в данной работе.

1. ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ КАЛИБРОВКИ КАМЕР

На данный момент существует 3 вида калибровки камер.

1) **Фотограмметрическая калибровка**

Способ фотограмметрической калибровки включает в себя фотографирование тестировочного пространственного или плоского объекта с

множеством точек с известными нам координатами, определением его координат на изображении и вычислением внутренних и внешних параметров камеры [10].

2) Самокалибровка

Самокалибровка проводится без использования калибровочных объектов. В этом способе камера снимает статическую сцену с разных сторон, и определение координат объекта осуществляется путём выравнивания изображений методом связок, после чего идет перевод в координаты камеры и получение внутренней матрицы камеры. [1]

3) Калибровка с помощью коллиматора

На данный момент этот способ редко используется, потому что предполагает собой использование специальной лаборатории и дорогостоящего оборудования для калибровки.

Дисторсия изображения

Дисторсией называют искажение изображения, вызванное искривлением линзы объектива или не параллельностью установки матрицы и линзы.На момент написания работы основной вклад в искажение изображения вносит радиальная дисторсия, благодаря современным технологиям камеры средней ценовой категории практически исключают тангенсальную дисторсию.

На рисунке 1.3 показан пример **радиальной дисторсии**, она бывает двух видов: бочкообразная (Barrel distortion) и подушечная (Pincushion distortion). Искажения, вызванные ею, отсутствуют в центре изображения и усиливаются ближе к краям. Дисторсия вызвана оптическими искажениями линзы.

Рис. 1.3 Радиальная дисторсия

Тангенсальная дисторсия вызвана неправильной установкой линзы. Тогда реальный мир на изображении будет выглядеть более вытянутым или наклоненным. Из-за этого по изображению будет непонятно какой из объектов в действительности находится ближе к камере.

1.1. Алгоритм калибровки Чжан Чжэнъю

Этот простой и достаточно эффективный метод предложил использовать Zhengyou Zhang для того, чтобы обычные пользователи могли распечатать калибровочный лист, состоящий из черных и белых квадратов, и получить внутренние и внешние параметры камеры. [2]

Предложенный метод требует только наличия плоского узора, отображение которого мы будем сохранять в разных ориентациях, на самом деле можно менять ориентацию камеры и держать корректировочный лист в одном положении, так как математически это одна и та же операция. Нет необходимости знать движение калибровочного листа.

Обозначения

Координаты точки на изображении $\mathrm{m} = \begin{bmatrix} u \\ v \end{bmatrix}$.

Координаты трехмерной точки $M = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$.

Переопределенные векторы m и M, путем добавления 1, как последнего элемента: $\widetilde{m} = [u \quad v \quad 1]^T$ и $\widetilde{M} = [X \quad Y \quad Z \quad 1]^T$.

Связь координаты точки на изображении и трехмерной точки задается через уравнение

$$s\widetilde{m} = A[R \quad t]\widetilde{M}, \tag{1.1}$$

где s – произвольный масштабный коэффициент. $[R \quad t]$ – матрица, связывающая мировую систему координат и систему координат камеры, состоит из матрицы поворота R и матрицу смещения t. Внутренняя матрица (intrinsic matrix) задается как A.

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix},$$

где (u_0 , v_0) — координаты основной точки, α и β — коэффициенты сжатия по осям *Ou* и *Ov*, соответственно, и γ — коэффициент асимметрии между осями изображения.

Матрица гомографии

Если взять глобальную систему координат на плоскости коррекционного листа, то можно взять $Z = 0$ и преобразовать систему 1.1 в систему вида

$$s\widetilde{m} = H\widetilde{M}, \tag{1.2}$$

где $\widetilde{M} = [X \quad Y \quad 1]^T$, а $H = A[r_1 \quad r_2 \quad t]$. r_1 и r_2 – соответственно первый и второй столбец матрицы R.

Практически матрица гомографии может использоваться для «склейки» изображений или изменения перспективы объекта на фото. Примеры приведены на рисунке 1.4 и 1.5

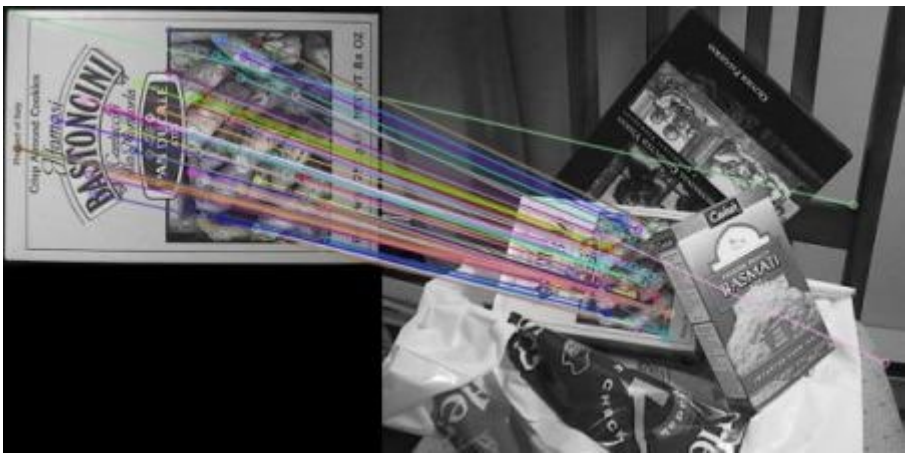


Рис. 1.4 Изменение перспективы объекта на изображении

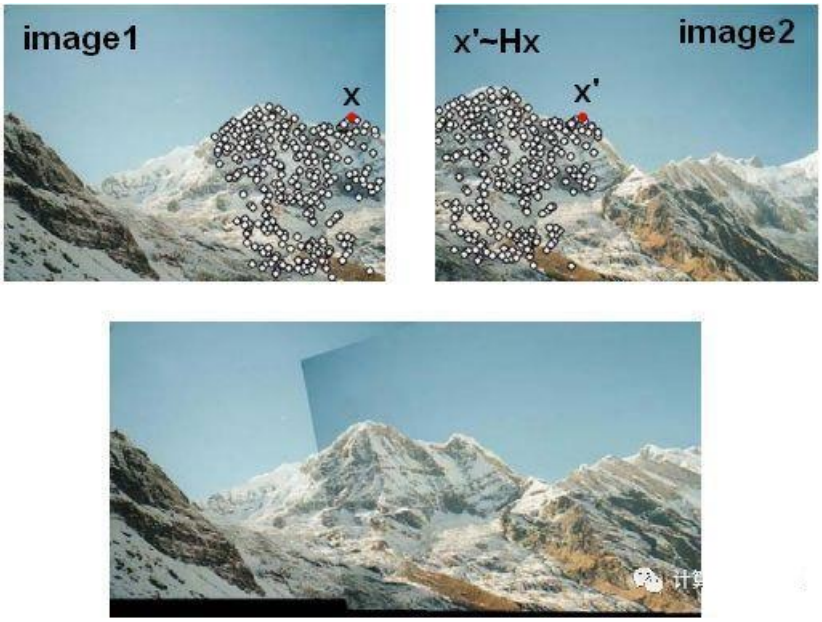


Рис.1.5 «Склейка» изображений

Ограничения внутренних коэффициентов

Матрицу гомографии можно записать в виде $H = [h_1 \quad h_2 \quad h_3]$ из уравнения (1.1) и (1.2) запишем

$$[h_1 \quad h_2 \quad h_3] = \lambda A[r_1 \quad r_2 \quad t],$$

где λ – произвольный скаляр для другого изображения. Зная, что r_1 и r_2 ортогональные векторы [2] ${r_1}^T r_2 = 0$, имеем

$$\mathrm{h_1}^{\mathrm{T}}(\mathrm{A^{-1}})^{\mathrm{T}}\mathrm{A^{-1}}\mathrm{h_2} = 0 \qquad (1.3)$$

$$\mathrm{h_1}^{\mathrm{T}}(\mathrm{A^{-1}})^{\mathrm{T}}\mathrm{A^{-1}}\mathrm{h_1} = \mathrm{h_2}^{\mathrm{T}}(\mathrm{A^{-1}})^{\mathrm{T}}\mathrm{A^{-1}}\mathrm{h_2}. \qquad (1.4)$$

На первый взгляд может показаться, что матрица гомографии имеет 9 степеней свободы, так как имеет размерность 3×3, но это не так. Мы используем однородную систему координат, так что матрица гомографии имеет 8 степеней свободы. [3]

Калибровка камеры

Возьмем матрицу

$$B = (\mathrm{A}^{-1})^{\mathrm{T}}\mathrm{A}^{-1} = \begin{bmatrix} \mathrm{B}_{11} & \mathrm{B}_{12} & \mathrm{B}_{13} \\ \mathrm{B}_{12} & \mathrm{B}_{22} & \mathrm{B}_{23} \\ \mathrm{B}_{13} & \mathrm{B}_{23} & \mathrm{B}_{33} \end{bmatrix} =$$

$$\begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{\mathrm{v}_0\gamma-\mathrm{u}_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2}+\frac{1}{\beta^2} & -\frac{\gamma(\mathrm{v}_0\gamma-\mathrm{u}_0\beta)}{\alpha^2\beta^2}-\frac{\mathrm{v}_0}{\beta^2} \\ \frac{\mathrm{v}_0\gamma-\mathrm{u}_0\beta}{\alpha^2\beta} & -\frac{\gamma(\mathrm{v}_0\gamma-\mathrm{u}_0\beta)}{\alpha^2\beta^2}-\frac{\mathrm{v}_0}{\beta^2} & \frac{(\mathrm{v}_0\gamma-\mathrm{u}_0\beta)^2}{\alpha^2\beta^2}+\left(\frac{\mathrm{v}_0}{\beta^2}\right)^2+1 \end{bmatrix}.$$

Можно заметить, что матрица B симметрична относительно главной диагонали.

Запишем матрицу

$$\mathrm{b} = [B_{11} \quad B_{12} \quad B_{22} \quad B_{13} \quad B_{23} \quad B_{33}]^{\mathrm{T}}. \tag{1.5}$$

Введем индексы для элементов матрицы $H = h_{ij}$, $h_i = [h_{i1} \quad h_{i2} \quad h_{i3}]^T$. Тогда

$$h_i{}^{\mathrm{T}} B h_j = \mathrm{v_{ij}}{}^{\mathrm{T}} b, \tag{1.6}$$

где $v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{21}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}$. Тогда, используя (1.3) и (1.4) получим следующие уравнения для b:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^{\mathrm{T}} \end{bmatrix} = \mathrm{Vb} = 0 \tag{1.7}$$

Таким образом, если мы имеем *n* изображений с разной ориентацией калибровочной плоскости, то система (1.7) содержит *2n* однородных уравнений и V имеет размерность $2n \times 6$. Неизвестных 5, то есть, если $n \geq 3$, мы будем иметь однозначное решение для *b.* Если положить коэффициент асимметрии между осями изображения $\gamma = 0$, то можно обойтись двумя изображениями.

После нахождения элементов матрицы *b* мы можем рассчитать внутренние параметры камеры [2]:

$$v_0 = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2)$$

$$\lambda = B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11}$$

$$\alpha = \sqrt{\lambda/B_{11}}$$

$$\beta = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)}$$

$$\gamma = -B_{12}\alpha^2\beta/\lambda$$

$$u_0 = \gamma v_0/\beta - B_{13}\alpha^2/\lambda.$$

После чего рассчитываются внешние параметры:

$$r_1 = \lambda A^{-1}h_1$$

$$r_2 = \lambda A^{-1}h_2$$

$$r_3 = r_1 \times r_2$$

$t = \lambda A^{-1}h_3$ с $\lambda = 1/\|A^{-1}h_1\| = 1/\|A^{-1}h_2\|$.

1.2. Методы калибровки камеры без шаблона

Все методы калибровки основаны на трех этапах.

1) Поиск на изображениях особых точек.
2) Поиск набора пикселей, соответствующих каждому изображению и одной и той же точке в пространстве.
3) На основе точечных соответствий вычисляется внутренняя матрица и матрицы поворота и смещения относительно мировых координат.

Самая большая проблема заключается в нахождении особых точек.

Особые точки изображения

В работе Ивашечкина А.П. [9] исследуются разные алгоритмы для нахождения особых точек. Это такие методы распознавания особых точек, как Детектор Моравеца, Харриса, Shi-tomasi, FAST, SIFT, SURF, BRISK. Вывод, который делает исследователь, заключается в том, что среди рассмотренных в работе алгоритмов нет универсального, который может работать при разных условиях съемки, таких как шум изображения, съемка в светлое время суток и темное, и увеличение изображения.

Проблема заключается в том, что все рассмотренные алгоритмы неправильно обозначают особые точки. Например, на рисунке 1.6 черной стрелкой обозначена одна из неотмеченных особых точке в повернутом изображении.

Рис. 1.6 Неотмеченная особая точка

Метод Ричарда Харли

Метод Ричарда Харли (Richard I. Hartley) [8] является самым удобным на дынный момент. Он подразумевает собой фотографирование объекта с разными ориентациями камеры, но с одной точки пространства. Автор использует одну камеру вместо того, чтобы брать несколько идентичных. Так как камера находится в одной точке пространства, матрица перемещения нулевая.

Идея алгоритма заключается в нахождении матрицы вращения $P = ARA^{-1}$. Предполагается использование нескольких снимков $J_0, J_1, \dots, J_N$, где $N \geq 2$.

1) Необходимо найти особые точки на изображениях.
2) Для каждого изображения вычислить матрицу поворота относительно первого изображения.

Для этого мы имеем координаты точки на первом изображении $\boldsymbol{u}_i = (u_i \quad v_i \quad 1)^T$ и на другом изображении $\boldsymbol{u}'_i = (u'_i \quad v'_i \quad 1)^T$

Тогда преобразование между координатами точек на изображениях можно записать как $\lambda'(u'_i \quad v'_i \quad 1)^T = P(u_i \quad v_i \quad 1)^T$, где λ неизвестна. Если переписать матрицу P через строки p_1^T, p_2^T, p_3^T, то можно записать систему уравнений:

$$\lambda' u'_i = p_1^T u_i$$
$$\lambda' v'_i = p_2^T u_i$$
$$\lambda' = p_3^T u_i$$

и тогда можно привести её к двум уравнениям

$$p_3^T u_i u'_i = p_1^T u_i$$
$$p_3^T u_i v'_i = p_2^T u_i.$$

Это набор из двух линейных уравнений в элементах P. Четыре таких точечных совпадения обеспечивают набор из восьми уравнений в элементах P. Поскольку P определяется только с точностью до масштаба, этого достаточно для линейного решения уравнений для элементов P. Если имеется более четырех совпадающих точек, то мы имеем переопределенный набор уравнений вида Ap = 0, где p - вектор, состоящий из элементов P. Мы стремимся найти p такой, что || p || = 1 и такой, что || Ap || минимизирован.

3) Для всех изображений найти внутреннюю матрицу A, такую что

$$A^{-1} P_j A = R_j, \tag{1.8}$$

где R_j матрица поворота j-ого изображения относительно первого. Матрицы P_j нам уже известны. Так как $R_J = R_J^{-1}$, мы можем переписать СЛАУ как

$$R_j = A^T\left(P_j^T\right)^{-1}(A^T)^{-1}. \quad (1.9)$$

Приравнивая левую и правую части уравнений 1.8 и 1.9 соответственно имеем

$$(AA^T)\left(P_j^T\right)^{-1} = P_j(AA^T). \quad (1.10)$$

Возьмем $C = AA^T$, тогда уравнение 1.10 дает нам шесть неизвестных, так как размерность матрицы C 3×3 и она симметрична относительно главной диагонали. Так же система 1.10 дает нам 9 линейных уравнений, это значит, что нам нужно минимум 3 различные матрицы P, чтобы найти матрицу C. Эту переопределённую систему можно записать иначе, как $Vc = 0$, где V – матрица размерности $9N \times 6$ ($N \geq 3$), а c – столбец, состоящий из элементов матрицы C. Новую систему можно решить воспользовавшись методом наименьших квадратов или использовать SVD разложение. Затем по найденной матрице C, мы можем найти матрицу A используя метод Холецкого [12], при этом C должна быть положительно определена.

4) Уточнить расчетную матрицу камеры, используя итерационные методы Левенберга-Марквардта.

Калибровка камеры с переменным фокусным расстоянием

Марина Колесник в своей статье [13] предлагает метод, с помощью которого мы можем определить внутренние параметры камеры для нового фокусного расстояния. Эта технология для камеры с зум-объективом позволяет непрерывно обновлять внутренние параметры камеры во время масштабирования. Работа основана на том факте, что оптическая ось и те же две трехмерные точки в стандартной системе координат камеры, остаются неизменным для последующих кадров изображения.

Метод не является новым алгоритмом для калибровки камеры, но он ее упрощает. При этом нахождение внутренних параметров может

осуществляться при условии: 1) камера неподвижна 2) камера выполняет вращение вокруг оптической оси. Помимо этого, необходимо иметь выпрямленное изображение (дисторсия исправлена) и знать внутренние параметры предыдущего фокусного расстояния. Этим методом невозможно рассчитать параметр перекоса, а при условиях, требуемых для метода, два параметра, определяющих центр изображения, остаются прежними. Поэтому в матрице A, мы ищем только фокусное расстояние по горизонтальной и вертикальной оси.

Основа метода: задается матрица проекции P

$$P = \begin{bmatrix} -a_u & 0 & u_0 \\ 0 & -a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (1.11)$$

где u_0, v_0 – координаты центральной точки, a_u, a_v – коэффициенты масштабирования по горизонтальной оси изображения и вертикальной соответственно. Для исходного изображения N матрица P известна, ее коэффициенты будем обозначать без штриха, для изображения с другим фокусным расстоянием $N + 1$ коэффициенты обозначены штрихом и эти параметры можем записать как:

$$\{f' = \mu f\} \rightarrow \{a'_u = \mu a_u, a'_v = \mu a_v, u'_0 = u_0, v'_0 = v_0\}, \quad (1.12)$$

где μ неизвестный параметр.

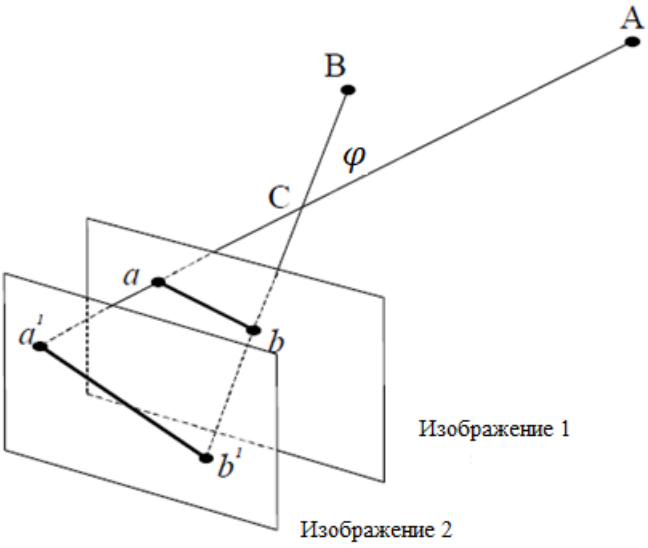


Рис. 1.7 Геометрия двух кадров,
имеющих разное фокусное расстояние

На рисунке 1.7 выражение для оптического луча <C/a> можно записать через координаты пикселя a и оптический центр C

$$A = \lambda P^{-1}\tilde{a}, \tag{1.13}$$

где $\tilde{a} = [a_1, a_{2,}1]$ – вектор однородных координат пикселя a, λ – переменная, лежащая на промежутке от $-\infty$ до $+\infty$.

Физически, угол a^1Cb^1 равен углу $a\ C\ b$, тогда мы можем записать его через скалярное произведение векторов A и B.

$$cos\varphi = \frac{AB}{|A||B|}. \tag{1.14}$$

Подставим переменные с первого и второго изображения в 1.13 и применим их в уравнении 1.14, тогда получим:

$$\frac{(P_1^{-1}\tilde{a})^T(P_1^{-1}\tilde{b})}{|P_1^{-1}\tilde{a}||P_1^{-1}\tilde{b}|} = \frac{(P_2^{-1}\tilde{a}^1)^T(P_2^{-1}\tilde{b}^1)}{|P_2^{-1}\tilde{a}^1||P_2^{-1}\tilde{b}^1|} \tag{1.15}$$

При подстановке 1.12 в 1.15 получим квадратное уравнение. И его решением будет коэффициент μ, он должен быть действительным и положительным

Таким образом алгоритм метода заключается в трех действиях:

1) Вычисление матрицы проекции P для двух изображений.
2) Записать уравнение 1.15 для пары точек.

3) Найти коэффициент изменения масштаба μ.

Самокалибровка камер с евклидовой плоскостью изображения в случае двух видов и известного относительного угла поворота

Евклидова плоскость изображения получается, если мы в камере полагаем параметр перекоса равным нулю, что вполне допустимо для камер средней ценовой категории и выше, и равные стороны изображения. Соответственно, единственные параметры, которые мы ищем – это координаты точки центра изображения и фокус.

Внутренняя матрица A в таком случае будет

$$A = \begin{bmatrix} f & 0 & a \\ 0 & f & b \\ 0 & 0 & 1 \end{bmatrix},$$

где f – фокусное расстояние; a, b -координаты центра изображения.

Для калибровки необходимо иметь минимум два изображения и 7 совпадающих точек на них. Алгоритм работает при:

1) Внутренние параметры камеры совпадают для обоих изображений, или же мы используем одну камеру.
2) Камера имеет евклидову плоскость изображения.
3) Известен угол поворота между изображениями.

Вот как Е. В. Мартышев описывает недостатки [14] своего метода:

Камеры и характерные точки расположены в общем положении, то есть одно изображение должно частично перекрывать другое. Существуют критичные положения камеры при которых самокалибровка невозможна, если не сделаны предположения о движении камеры и об ее внутренних параметрах. Помимо этого, существуют вырожденные конфигурации для точек изображения. В его работе озвучена прекрасная идея нормализовать мировые координаты и координаты на изображении [15].

2. ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РАСПОЗНАВАНИЯ СТАТИЧЕСКИХ СЦЕН

2.1. Принцип работы нейронной сети

Анализ искусственного нейрона

Для начала стоит разобраться, что такое «нейрон». Нейрон в машинном обучении [4] – это некоторая математическая функция, которая, как и в биологическом нейроне, обеспечивает вывод, применяя действие к входным данным. На сегодняшний день можно выделить несколько функций, используемых в нейроне. Их еще называют функцией активации: step, sigmoid, tanh и ReLU. Рассмотрим каждую по отдельности.

STEP или ступенчатая функция активации

$$\boldsymbol{f(x) = \begin{cases} 0\ if\ 0 > x \\ 1\ if\ x \geq\ 0 \end{cases}}$$

Функция подходит для модели в которой применяется один класс и варианты «Да» и «Нет» в качестве вывода результата. Более усложненный вариант с несколькими выходными слоями (классами) будет трудно реализуем. Предположим, мы хотим, чтобы нейрон выдавал только один класс, а остальные выдавал за 0, тогда возможны случаи «активирована на 50%», «активирована на 20%» и так далее. Все эти классы выведут 1, но какой в итоге класс присвоить данному объекту?

Sigmoid или сигмоидальная функция активации

$$\boldsymbol{f(x) = \frac{1}{1 + e^{-x}}}$$

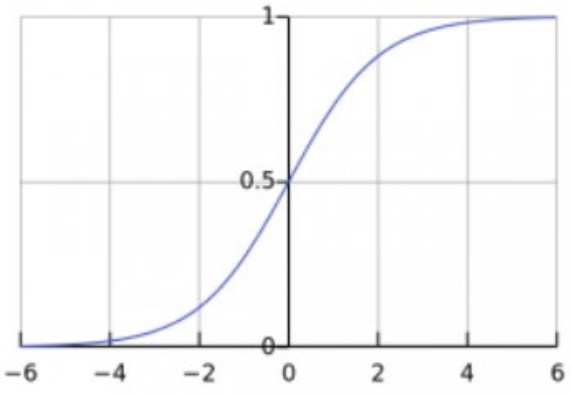


Рис. 2.1 Сигмоидальная функция

Диапазон функции (0,1), ее удобно использовать, когда мы нормализуем наши значения, например, разбив изображение на RGB пиксели и загнав эти данные в массив для подачи в нейрон, мы изменим их размер от [0,1]. Функция не линейна, значит мы можем использовать ее для нескольких классов.

Функция имеет недостаток в виде исчезающего градиента, так как производная мала. Из-за этого, если модель содержит несколько слоев, то усложняется процесс улучшения весов.

Функция гиперболического тангенса

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = 2 \times sigma(2x) - 1$$

Функция работает на диапазоне (-1,1), и она используется вместо сигмоидальной, потому что в основном веса сосредоточены вокруг нуля и более высокий градиент тангенса увеличивает скорость обучения.

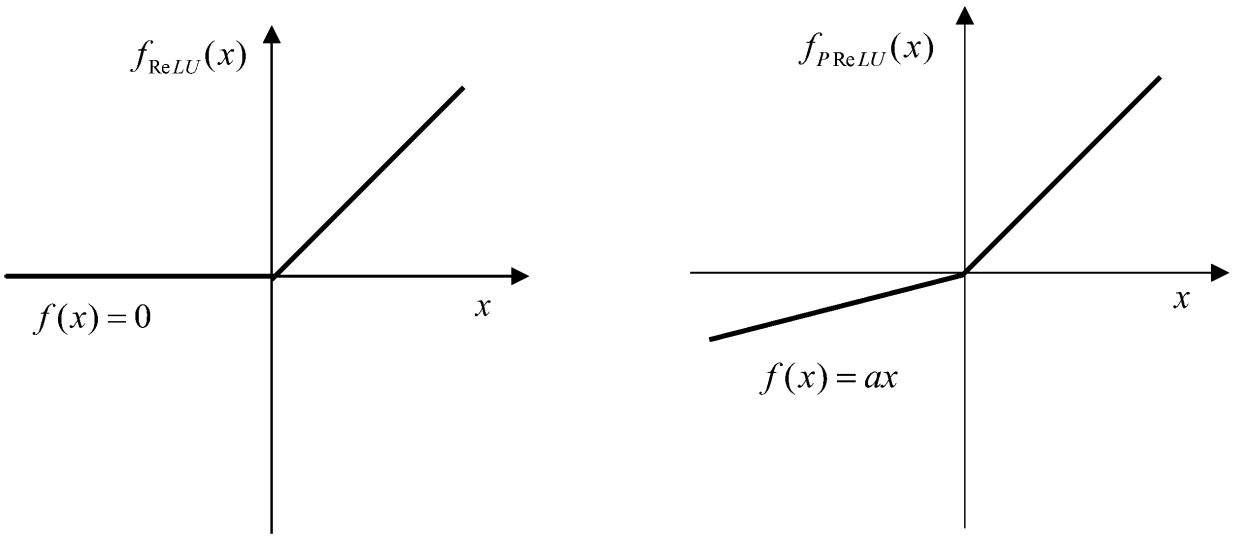
ReLU

Rectified Linear Unit — это наиболее часто используемая функция активации при глубоком обучении. Она возвращает само число при положительном аргументе и 0 при аргументе меньше нуля. Это заметно повышает разреженность, почти в половине случаях нейрон больше не срабатывает. Таким образом, нейроны можно занулить, если они не вносят заметный вклад в прогнозы модели. Заметно ниже вычислительные затраты. Если мы работаем с одним нейроном, то это будет не так важно, но в глубоком

обучении используется много таких нейронов, следовательно ReLU снижает потребность в вычислительных требованиях.

К сожалению, имеются и недостатки. Во-первых, при $x > 0$ производная функции равна 1, что способствует значению нейрона равному сумме входов. Теоретически бесконечный вход выдает бесконечный выход. Во-вторых, появляется проблема «умирающего ReLU» если веса нейрона приближаются к нулю, может случиться так, что они в конечном итоге не смогут оправиться от этого и будут постоянно выводить нули. Это особенно важно, когда сеть плохо собрана или когда входные данные плохо нормализованы. Таким образом, проблема умирающего ReLU – мертвая нейронная сеть из-за обнулившихся в ноль весов.

От второй проблемы можно избавиться, если использовать функцию $f(x) = ax$ при $x < 0$ для небольших значений a.



ReLU PReLU

Рис. 2.2 Функция ReLU

Помимо функции активации в нейроне содержится сумматор, он складывает веса (выходы других нейронов) и передает в функцию активации.

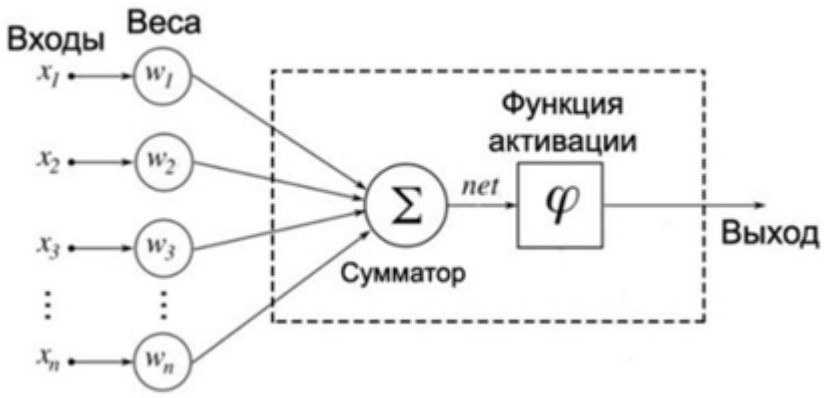


Рис. 2.3 Искусственный нейрон

Анализ слоев в нейронной сети

Совокупность нейронов составляет слой [5]. Для слоя характерно иметь одни и те же функции активации в каждом нейроне. А сама нейронная сеть состоит из совокупности этих слоев (рисунок 2.4), всего их три: **входной слой, скрытые слои и выходной**.

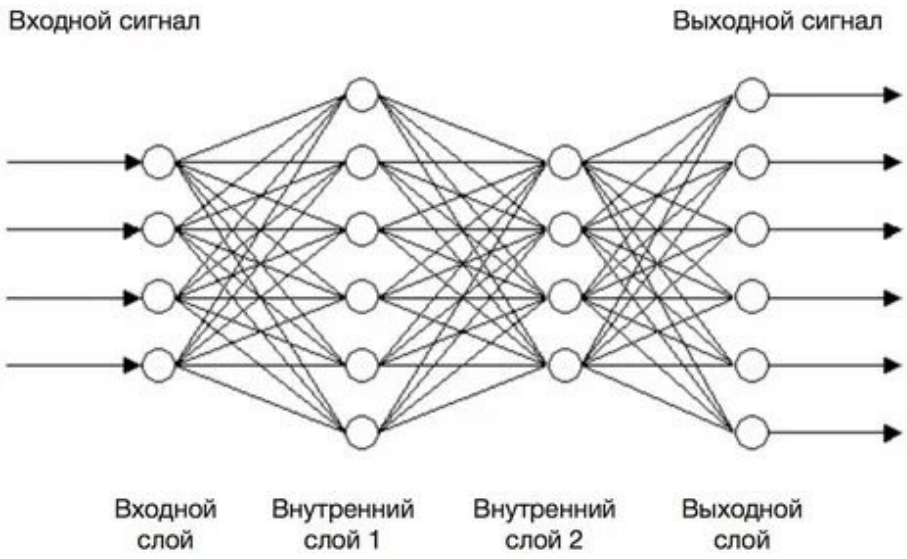


Рис. 2.4 Слои в нейронной сети

Входной слой – это первый слой нейросети, который состоит из поданного нами массива данных, например, в этой работе будет использоваться набор пикселей изображений.

Скрытый слой – слой, состоящий из нейронов, которые суммируют значения весов и применяют некоторую функцию к этим данным. После чего передают сигнал в следующий слой.

Выходной слой – слой, состоящий из нейронов, активация которых указывает на логическое окончание работы нейросети. Если мы говорим про распознавание изображений, то эта классификация, которую мы задали.

Важно упомянуть про такую составляющую нейронных сетей, как использование смещения во входном и скрытых слоях. **Смещение** – это нейрон, который не принимает никакие входные данные, а на его входе всегда единица. Данный параметр позволяет сдвигать функцию активации влево и вправо для того, чтобы сделать модель более гибкой. Его расположение показано на рисунке 2.5

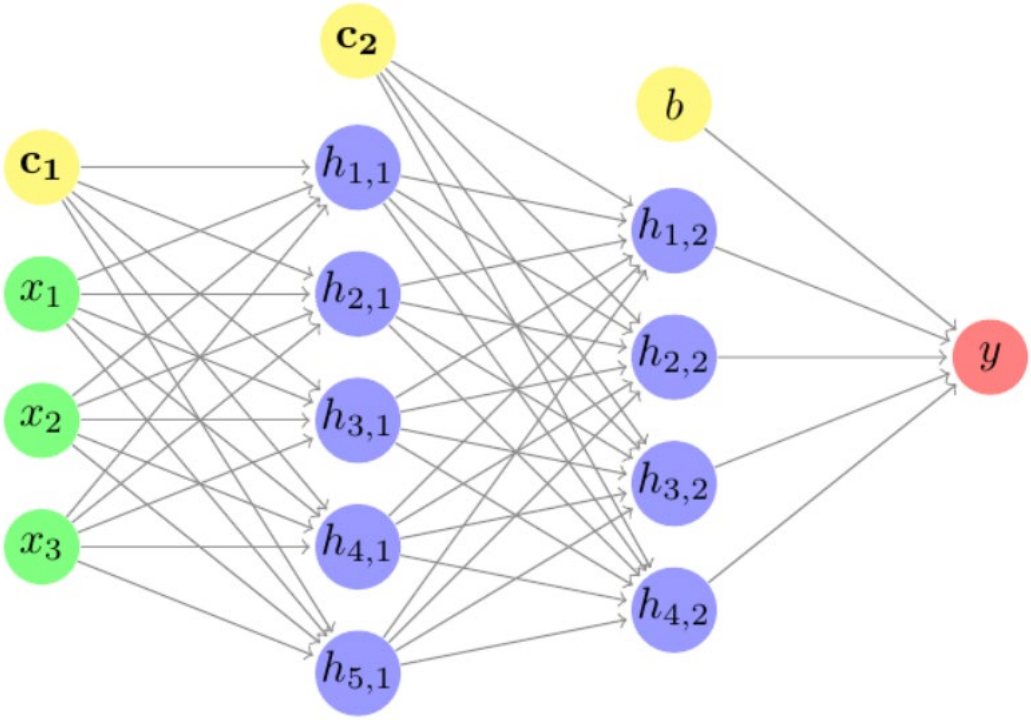


Рис. 2.5 Расположение смещенного нейрона

2.2. Анализ и выбор архитектуры для обработки изображений

Особенности свёрточной нейронной сети

Свёрточная нейросеть содержит преимущественно свёрточные слои и является наиболее эффективной сетью для распознавания образов, входит в состав технологий глубокого обучения. Они используют фильтр размером NxN, где N натуральное число, чтобы выделить существенные признаки объекта. Область «скользит» по изображению и выделяет признаки объекта. Фильтр может иметь разные значения в матрице, в ней за единицу обозначается черный пиксель, за ноль – светлый. Таким образом фильтр может искать участки, содержащие прямые, кривые и другие признаки. Наглядная демонстрация показана на рисунке 2.6, где красным обозначена выделенная область изображения, синим фильтр (ядро), ищущий две перекрестные линии на изображении. Значения, полученные поэлементным перемножением области и фильтра суммируются и в новом массиве данных, имеющем меньший размер, чем предыдущий слой, записывается значение после обработки фильтра.

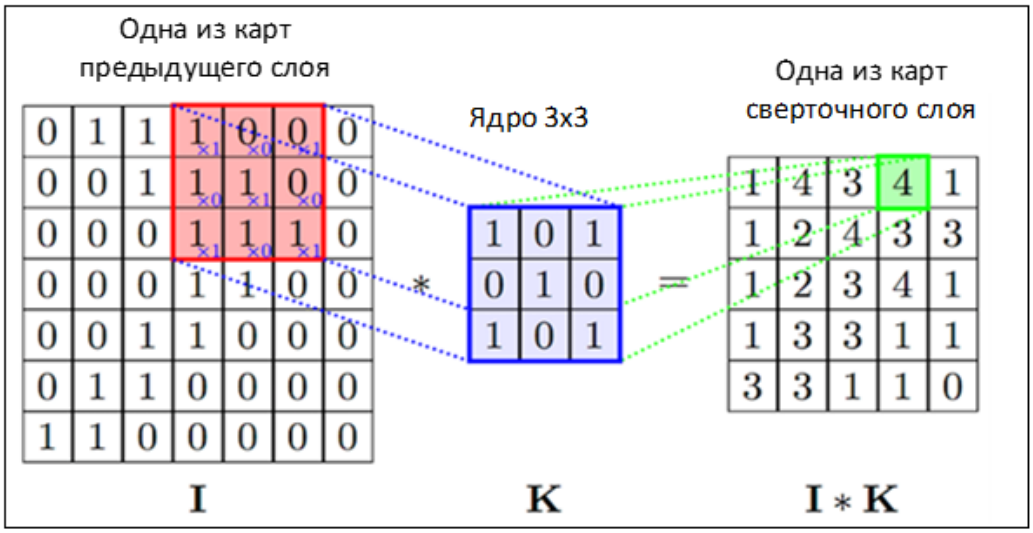


Рис. 2.6 Работа свёрточного слоя

Пример для рисунка 2.6: $1 \times 1 + 0 \times 0 + 1 \times 0 + 0 \times 1 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 1 + 1 \times 1 = 4$.

Mask-RCNN

Mask-RCNN была создана в 2018 году, на момент написания работы она является наилучшей нейросетью для сегментации изображений. Её отличительной особенностью является двухэтапность. На первом этапе происходит классическое детектирование: к изображению применяется связка из нескольких свёрточных слоев и того же количества «расширяющих» слоев. Такую архитектуру называют FPN, наглядная демонстрация показана на рисунке 2.7.

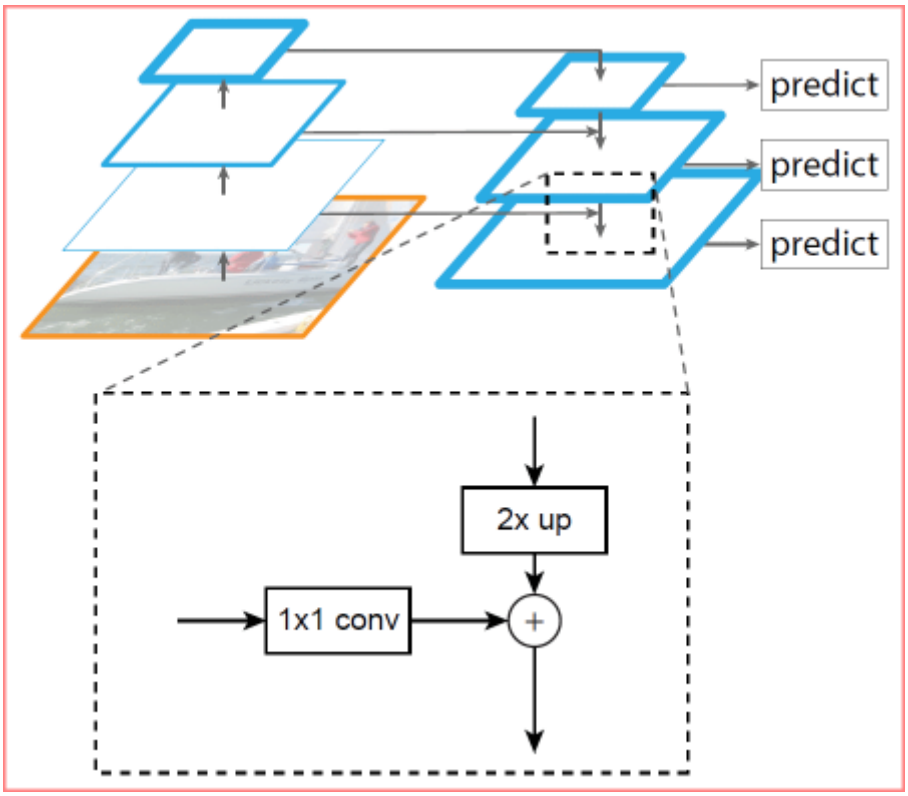


Рис. 2.7 Архитектура FPN

FPN (Feature Pyramid Network) работает следующим образом: к изображению применяется несколько сверточных слоев, изображение после каждого слоя сохраняется, последний слой является началом новой ветви обработки и в этот же момент передается в выходной слой нейросети для предсказания класса. Во

второй ветви изображение расширяется методом ближайшего соседа. На рисунке 3.8 показана приблизительная работа метода.

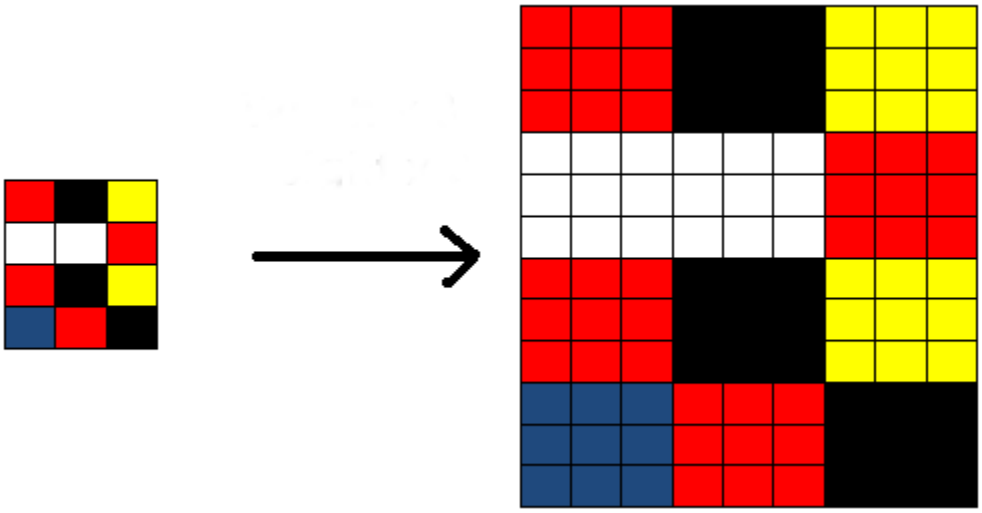


Рис. 2.8 Метод ближайшего соседа

Каждый последующий слой складывается из суммы предыдущего из второй ветви и параллельного ему слоя из первой ветви, прошедший фильтр 1x1 conv. Фильтр необходим, чтобы уменьшить глубину массива данных и соответственно увеличить скорость работы. На каждом слое происходит предсказание объекта на изображении и рамки, обрамляющей его. Далее на каждом из предсказаний идет разделение на классификацию и поиск более точной рамки. Второй этап начинается после FPN модуля и идет параллельно получению обрамляющей рамки и класса. К каждому пикселю применяется сигмоидальная функция и выделение объекта происходит для всех классов, чтобы потом взять маску для каждого объекта из первого класса. Маска вычисляется как изображение, в котором значение пикселя равно единице в случае, если он принадлежит объекту и нулю, если нет.

YOLOv3

В отличии от Mask-RCNN YOLO одноэтапная сеть и она работает не с выборочными рамками, а с изображением целиком. Сначала к изображению применяются свёрточные и редуцирующие слои (метод ближайшего соседа). Через несколько этапов всё, что осталось от изображения делится сохраняется,

далее эта информация будет обрабатываться другими свёрточными слоями. Архитектура показана в оранжевом блоке на рисунке 3.9.

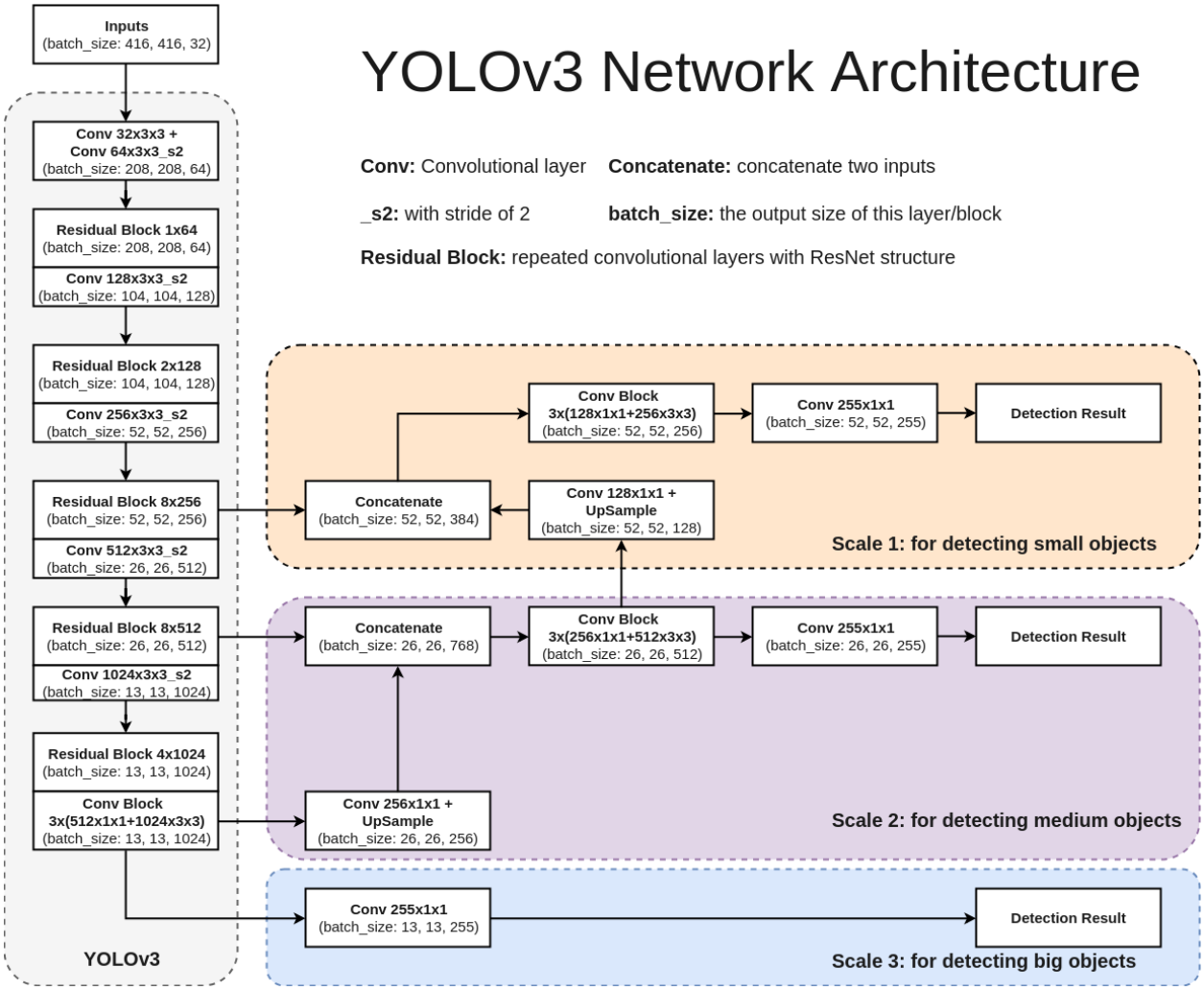


Рис. 2.9 Архитектура YOLOv3

На следующих этапах свертки и редуцирования изображения происходит еще сохранения данных. На заключительном этапе изображение разбивается на рамки из пикселей размером 13x13 и в каждой секции сравниваются веса, чтобы выдать вероятность наличия класса для объекта, занимающего большую часть изображения (на рисунке 3.9 это обозначено синим полем). Предыдущие два сохранения данных нужны для детектирования объектов, имеющих средний размер. Веса из первых сохраненных данных суммируются с последующим и детектирование маленьких объектов осуществляется с помощью рамки 52x52.

2.3. Требования к датасету

Разработка датасета является важным этапом при построении нейронной сети. Точность нейросети напрямую зависит от качества выбранных данных для обучения, если данные плохо размечены, это выльется в ошибочно обученные признаки. Размеченные данные – если мы говорим про детектирование, это набор картинок, классов, координаты верхнего левого угла обрамляющей объект рамки и нижнего угла, данные о сегментации и площади.

Нельзя использовать одни и те же данные для тренировки и теста. В процессе обучения нейросеть выделит признаки из тестового набора и подстроится под них и уже когда мы дадим тестовый датасет на проверку нашей модели, мы получим недостоверный результат обученной модели. Наиболее выгодное процентное распределение данных для тренировки и теста это соответственно 80% и 20% от общего числа картинок. При этом данные для тестирования не должны отличаться от тестовых, например, если мы обучали модель на распознавание кошек, собак и легковых автомобилей, мы не можем иметь в тестовом наборе картинки с лошадьми или самолетами, так как это повлечет за собой ложные выводы о работе нейросети.

Какой же объем данных необходим? Ответ на этот вопрос очень прост, чем больше, тем лучше. 1000 обучающий примеров – хорошо, 10000 – отлично, некоторые компании сейчас обучают свои модели на миллионах изображений, видео и аудиофайлах. Теоретически можно обучить модель на 100 примерах каких-то данных, но результат такой работы будет удручающим. К тому же, для определения одного конкретного человека достаточно и 1000 изображений, ИИ уже выделит и подстроится под все возможные признаки этого человека при всевозможных углах и при разном освещении. Однако, если стоит задача распознавания людей, то набор из 10000 картинок будет

малой частью из всевозможных вариаций поз и типов внешностей. Поэтому для класса с большим количеством признаков требуется большое количество обучающих данных.

3. ПОСТРОЕНИЕ ПРОГРАММЫ И ПОЛУЧЕНИЕ РЕЗУЛЬТАТОВ

Изучив варианты решения задачи калибровки, было выяснено, что все алгоритмы самокалибровки камеры основываются на поиске особых точек. В этом и заключается основная проблема. Ни один метод поиска реперных точек не способен работать в различных условиях, например, в темное и светлое время суток, с пейзажем и в городских условиях. Именно поэтому задача нахождения координат точек объекта на нескольких снимках остается основным.

Автор данной работы предлагает исключить зависимость от детекторов Моравеца, Харриса, Shi-tomasi, FAST, SIFT, SURF, BRISK. Зачастую робот проектируется для конкретной работы в определенных условиях. Соответственно его будут окружать определенные объекты во время его работы. Тогда **требования к программе** будут следующие:

1) Отсеивание изображений или кадров видеопотока на которых нет объекта с известными геометрическими размерами.
2) Определение координат точек на изображении в системе координат изображения.
3) Нахождение внутренней матрицы камеры и коэффициентов дисторсии.
4) Нахождение матрицы поворота и смещения одной камеры относительно другой.
5) Сохранение результатов калибровки.
6) Запуск видеопотока с откалиброванных камер.

Для имитации движения мобильного робота собрана платформа, на которой размещены две камеры рисунок 3.1.

Рис. 3.1 Платформа с камерами

Для вращения платформы используется сервопривод SG-90. Для записи видеопотока применяются 2 камеры LogiTech c270. Сама платформа была разработана с использованием программы SolidWorks и распечатана на 3D принтере с использованием PETG пластика.

Реализация

Выбор калибровочного объекта

В мире существует множество предметов, имеющих всеобщий стандарт. Подход к выбору калибровочного объекта может быть разнообразным. Есть одно условие для его выбора: для решения системы 1.2 и поиска матрицы гомографии необходимо на одном изображении иметь минимум 4 узловые точки. В зависимости от условий работы робота объект выбирается индивидуально, поэтому в качестве учебной модели был выбран

восьмиугольный дорожный знак стоп. Выбор был сделан исходя из доступности датасета. Желательное требование, чтобы объект был контрастирующим с фоном.

Проведя анализ маршрутов, по которым передвигается беспилотный автомобиль, можно выделить следующие объекты для калибровки его камер:

- Дорожный знак «STOP».
- Дорожный знак «Искусственная неровность».
- Дорожный знак «Однопутная железная дорога».
- Регистрационные знаки транспортных средств.
- Полосы пешеходного перехода.

Все объекты были выбраны с учетом того, что автомобиль будет иметь низкую скорость движения при приближении к ним. Это важный параметр, потому что на больших скоростях изображение имеет размытие.

Определение объекта на изображении

Существует несколько вариантов распознать объект на изображении. Самым популярным и точным на данный момент являются нейросети. Помимо этого способа был рассмотрен метод, построенный с использованием библиотеки OpenCV, но как показали тесты, он имеет ту же недостатки, что и существующие детекторы особых точек. Задача этой работы заключается в точном нахождении реперных точек, поэтому было принято решение объединить эти два метода.

Для определения присутствия объекта на фото используется нейросеть YOLOv3 она обладает достаточным для нашей задачи быстродействием, но имеет недостаток – точность.

Так как для калибровки требуется более одного изображения и вычислительная мощность ограничена, для более четкого обнаружения объекта будем использовать нейросеть Mask R-CNN. Исходя из названия, она

накладывает маску на объект и выделяет его на фоне изображения. На рисунке 3.2 показано изображение после обработки нейросетью.

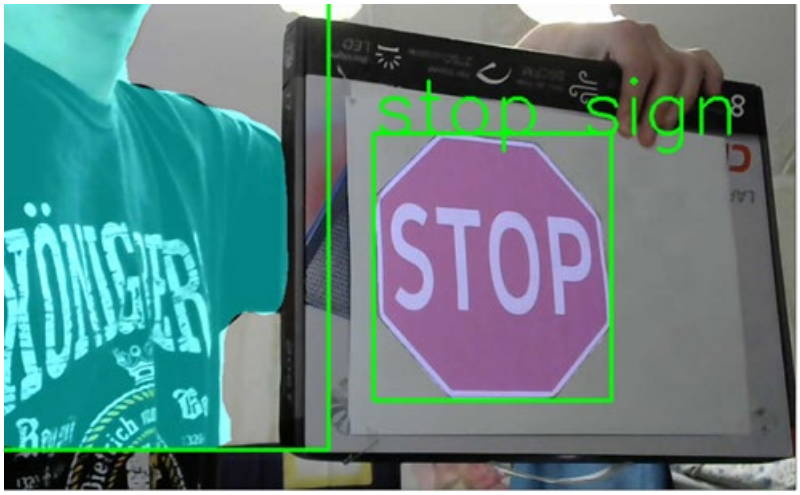


Рис. 3.2 Изображение после обработки нейросетью

Предполагалось, что маска, наложенная на изображение, выделит четкие контуры знака стоп. После нескольких тестов, было выявлено, что маска дает нечеткий контур рисунок 3.3, а если объект находится в движении (калибровка предполагает движение объекта или камеры в пространстве), то использование данного метода оказывается не только бесполезным, но и наносит вред качеству определения координат на изображении.

Рис. 3.3. Неточный контур знака

Тогда мы используем нашу нейросеть для нахождения области, рамки в которой есть наш объект и, пользуясь преимуществом библиотеки OpenCV, более точно определяем координаты на изображении восьми углов дорожного знака. При этом программа переводит цвета пикселей, находящихся за рамкой, в черный цвет для более точного распознавания координат.

Выделение характерных точек

После того, как объект найден на изображении необходимо найти особые точки. Для этого необходимо найти контур знака. В работе используется RGB стереокамера, если бы мы использовали RGBD стереокамеру, то нам так или иначе пришлось бы искать внешний контур объекта. Это исходит из того, что в RGBD камере используется специальная камера глубины, которая по своей сути является лидаром и возвращает облако точек в виде расстояний до объектов, таким образом калибруются RGBD камеры.

Контур знака строится по изменению цвета пикселя на изображении. Для этого кадр переводится из формата RGB в hsv, который более устойчив к

изменениям внешнего освещения. Так как знак содержит внутренние контуры, то из данных, полученных из нейросети, вычисляется площадь и берется контур, занимаемый больше 70% от этой величины. Теперь нам известно примерное положение точек на изображении. Примерное из-за того, что мы выбираем диапазон цвета hsv большим, чтобы расширить спектр применимой освещенности, а смазанное изображение дает градиент от красного до розового, и контур строится по розовому контуру.

Рис 3.4 Расположение контура

Необходимо более точно обнаружить угол, для этого существуют детекторы углов Харриса и более современный ORB, помимо них рассмотрим детектор особых точек SIFT, он больше подходит для распознавания изображений, но необходимо проверить на данном объекте калибровки.

Детектор углов Харриса работает со скользящим окном и его можно настроить для разного удаления предмета от проекции изображения. Для камеры Logitech и размера изображения 1280×720 выбраны расстояния 0.3м от камеры и 1.1м, что соответствует расстоянию для полного масштаба знака $(1.27 - 4.67)$м. Детекторы ORB и SIFT так же были настроены на том же расстоянии.

Границы работы программы

Так как мы напрямую работаем со значениями пикселей, а они сильно зависят от освещенности, необходимо установить границы применимости ПО. Для этого на расстоянии 0.8м от камеры был установлен калибровочный объект и люксметром измерялась освещенность. Из-за того, что мы работаем

с цветом изображения, необходимо проверять распознавание контура при разном световом потоке для теплого 2700К и холодного света 6400К. Тогда на разных уровнях освещенности выбиралось 10 фотографий объекта и каждое из них прорабатывалось программой, чтобы выяснить, какую координату изображения детектор особых точек выделяет за угол. Затем вычислялось расстояние до истинной координаты угла и максимальная погрешность заносилась в таблицу.

Таблица 1

Зависимость обнаружения углов от освещенности для теплого света

	Харрис	ORB	SIFT
Люкс	погрешность пикс.	погрешность пикс.	погрешность пикс.
48	3	3	2
106	2	3	4
145	3	3	3
201	2	3	>5
253	2	4	>5
302	2	3	>5
346	2	5	>5
400	2	>5	>5

Таблица 2

Зависимость обнаружения углов от освещенности для холодного света

	Харрис	ORB	SIFT
Люкс	погрешность пикс.	погрешность пикс.	погрешность пикс.
49	3	3	2
104	3	3	3
149	2	4	5
201	2	4	>5
251	2	4	>5
300	2	5	>5
351	2	>5	>5
402	2	>5	>5

Визуально идеально найденный угол показан зеленым пикселем на рисунке 3.5

Рис 3.5 Угол, найденный детектором Харриса

Как показывают тесты, наилучшим детектором углов в данном случае является алгоритм Харриса. Он дает наименьшую погрешность при работе от 200 люкс. ORB из-за специфики работы и шумов на изображении находит несколько точек возле одного угла, эта ошибка показана на рисунке 3.6

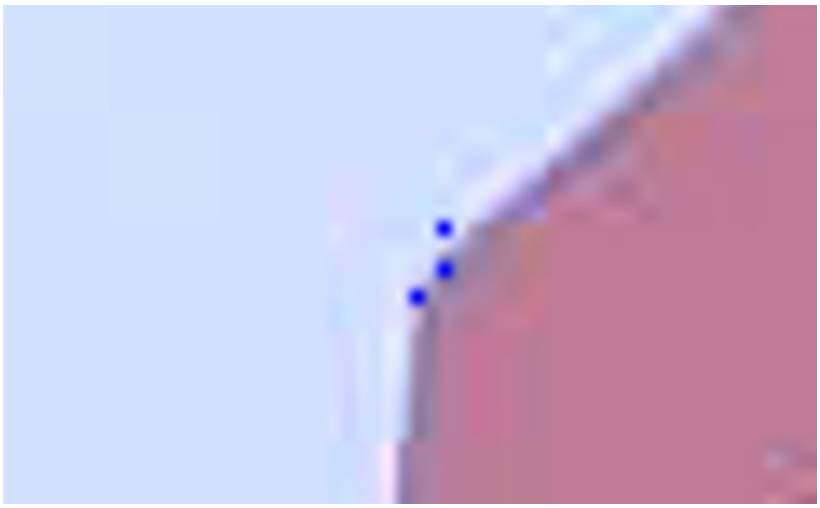


Рис 3.6 Угол, найденный детектором ORB

Детектор SIFT не предназначен для поиска углов, соответствующие результаты выдает программа.

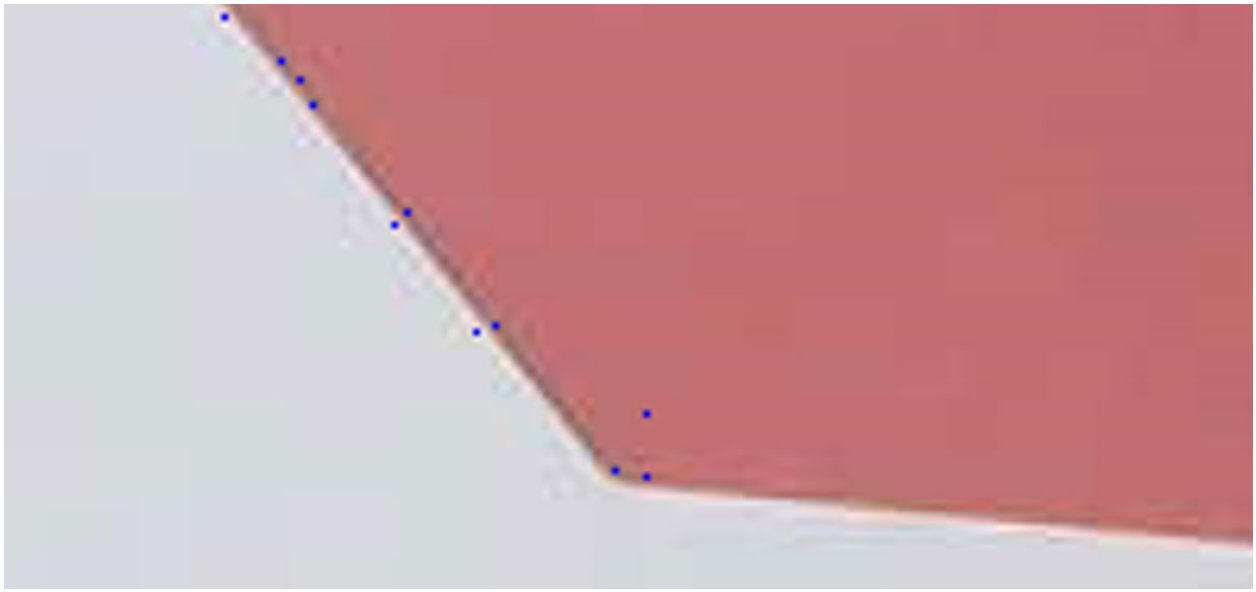


Рис 3.7 Угол, найденный детектором SIFT

Благодаря тому, что нейросеть выдает рамку с объектом, мы можем окрасить остальную часть изображения в черный цвет и осуществлять поиск контура в широком диапазоне красного цвета, и желтый оттенок, который дает лампа с теплым светом никак не влияет на программу.

Структура программы

Программу была разбита на две подпрограммы. Первая подпрограмма отвечает за сбор изображений, которые содержат объект. Вторая обрабатывает сохраненные изображения, ищет реперные точки на объекте, вычисляет внешние и внутренние параметры камеры и сохраняет их. На рисунке 3.8 представлена общая схема подпрограммы, собирающей изображения.

Рис. 3.8 Общая схема

Так как используется 2 камеры то запуск видеопотока производится с каждой камеры отдельно. Затем в цикле нейросеть ищет на изображении объект, пока в папке не будет находиться двадцать изображений.

- Видеопоток запускается с разрешением 1280×720 пикселей.
- Чтобы было равное количество изображений, камеры сохраняют кадр одновременно.
- Запись ведется с частотой 24кадра в секунду. Если объект принадлежит кадру, то сохраняется каждый пятнадцатый кадр.

Рис. 3.9 Общая схема

Обнаружение объекта осуществляется с помощью нейросети Mask R-CNN. Пример обработки показан на рисунках 3.2 и 3.3. Во время обработки функцией get_prediction в переменную pred_boxes заносятся координаты верхнего левого и нижнего правого угла. Эти координаты используются в функции narezka. Все пиксели вне рамки закрашиваются в черный для лучшей работы функции detectionRED. В ней изображение меняет формат с RGB на hsv, чтобы найти контуры красного. На рисунке 3.10 показана обработка изображения функцией detectionRED. Если контур содержит 8 точек, то они запоминаются для дальнейшей калибровки камеры.

Рис. 3.10 Изображение в функции detectionRED

Вычисление внутренних и внешних параметров производится с помощью SVD разложения. Помимо этого, используется встроенная функция для сравнения результатов вычисления.

Далее необходимо вычислить матрицу поворота и смещения правой камеры, относительно левой. Для этого из каждой пары изображений

необходимо взять матрицы поворота и смещения камеры относительно объекта $R_l,\ T_l,\ R_r,\ T_r$ и найти матрицу $R = R_r(R_l)^T$ и $T = T_r\ - RT_l$.

После нахождения преобразования между координатами камеры необходимо вычислить исправления для изображения. Для этого используется метод Bouguet [18]. Из матрицы вращения R вычисляются $r_l,\ r_r\ -$ матрицы поворота каждой камеры так, чтобы выставить камеры компланарно. Необходимо вычислить исправленную внутреннюю матрицу по точкам на обоих изображениях, чтобы выровнять их по вертикали, и с помощью встроенной функции cv2.initUndistortRectifyMap() вычислить преобразование пикселя из исходного изображения в новое, откалиброванное.

После сохранения результатов калибровки можно вывести изображение с двух камер (рисунок 3.11)

Рис. 3.11 стереоизображение после калибровки

В случае идеальной калибровки камер изображение выравнивается с точностью до пикселя и в нем убирается искажение, вызванное дисторсией.

Сравнение результатов калибровки

Фокусные расстояния и координаты фактического центра изображения индивидуальны для каждой камеры и обычно они не пишутся в паспорте камеры. Именно поэтому автор работы сравнивает их с алгоритмом, предложенным Чжан Чжэнъю, ставшим уже классическим. Для поиска особых

точек используется шахматная доска и встроенная в библиотеку OpenCV функция findChessboardCorners(). В таблице 3 показаны данные по калибровке с мною разработанным алгоритмом (наиболее близкие данные) и встроенной функцией OpenCV при одинаковой освещенности и одинаковом удалении объекта от камеры.

Таблица 3

Фокусные расстояния и координаты центра изображения

	Шахматная доска	Дорожный знак
	Левая камера	
fu	1.42973555e+03	1.40634786e+03
fv	1.42485833e+03	1.41378484e+03
Cu	6.30427051e+02	6.67294795e+02
Cv	3.31291706e+02	3.58322760e+02
	Правая камера	
fu	1.43724633e+03	1.42888484e+03
fv	1.43175476e+03	1.43738367e+03
Cu	6.19109449e+02	6.86245448e+02
Cv	3.41614555e+02	3.24060655e+02

Из теста следует, что максимальная разница значения фокуса 1.6% и 11% для координат оптической оси на изображении. Предполагается, что из-за специфики геометрии дорожного знака, точное определение координаты угла равного 135 градусов является более трудоемкой задачей, чем определение углов квадратов шахматной доски или меток AprilTag. В следствии этого появляется погрешность измерения.

ЗАКЛЮЧЕНИЕ

В рамках реализации поставленной цели калибровки камеры в условии недоступности ремонтной мастерской были выполнены следующие задачи:

- Изучены методы калибровки цифровых камер, их преимущества и недостатки, геометрические искажения изображений.
- Исследована методика калибровки камеры по пространственному объекту.
- Выбран калибровочный объект и на его примере исследованы оптимальные способы нахождения особых точек.
- На примере калибровочного объекта проанализированы детекторы углов и их зависимость от освещения.

В результате анализа, наиболее предпочтительным детектором углов оказался алгоритм Харриса, он работает даже при плохой освещенности, дает меньше ложных точек, предполагаемого расположения угла на объекте. Его погрешность остается равной двум пикселям на широком диапазоне освещенности. Однако, нельзя сказать, что он универсальный и подходит для всех видов объектов, хотя заметно превосходит другие детекторы по точности.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Безменов В. М. построение и уравнивание аналитической фототриангуляции // учебно-методическое пособие, Казань – 2009. – 87стр.
2. Zhengyou Z. A Flexible New Technique for Camera Calibration // Microsoft Research, One Microsoft Way, USA, Radmond. — 1998. – P. 1-22
3. Пояснение к матрице гомографии: [сайт]. URL: https://russianblogs.com/article/32831360064/ (дата обращения: 17.05.2022).
4. Рашид. Т. Создаем нейронную сеть. – 2017. – 274с.
5. Haykin S. Neural Networks and Learning Machines. – 2009. – 938p.
6. Созыкин А.В. Обзор методов обучения глубоких нейронных сетей // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. – 2017. – С. 28-59
7. Самое главное о нейронных сетях. Лекция в Яндексе. https://habr.com/ru/company/yandex/blog/307260/ (дата обращения: 05.04.2022).
8. Hartley R.I. Self-Calibration of Stationary Cameras // G.E. CRD, Schenectady, NY. – 17p.
9. Ивашечкин, А.П. Методы нахождения особых точек изображения и их дескрипторов / А.П. Ивашечкин, А.Ю. Василенко, Б.Д. Гончаров. // Молодой ученый. – 2016. - №15(119). – С.138-140
10. Ю.Б. Блохинов, Е.Э. Андриенко, К.К. Казахмедов, Б.В. Вишняков Автоматическая калибровка системы видеокамер и лидаров для автономных мобильных комплексов, статья, Москва – 2021. – 14стр.
11. Р. Н. Сафин, Р. О. Лавренов, Р. А. Боби, С.К. Саха, Е. А. Магид, эксперименты по калибровке камер мобильного робота при наличии аппаратных дефектов в системе технического зрения статья, Казань – 2018. – 10стр.

12. Atkinson K.E. An Introduction to Numerical Analysis, 2nd Edition // John Wiley and Sons, New York — 1989
13. Kolesnik M. Calibration Update Technique for a Zoom Lens // CAIP, 1999, — P.435–443.
14. E.V. Martyushev self-calibration of cameras with Euclidean image plane in case of two views and known relative rotation angle // ECCV 2018, Chelyabinsk – 2018, – 15p.
15. R. Hartley and A. Zisserman, Multiple view geometry in computer vision, Cambridge University Press, 2003. P. 446-447
16. А.С. Макаров, М.В. Болсуновская сравнительный анализ методов обнаружения особых точек на изображениях при различных уровнях освещенности // Научно технические ведомости СПбГПУ, Санкт – Петербург – 2018.
17. Guillermo G. ·Anthony Y. A compact formula for the derivative of a 3-D rotation in exponential coordinates, // Springer, NY – 2014. – 7p.
18. Bouguet, J.-Y., “Camera calibration toolbox.” Web URL: http://www.vision.caltech.edu/bouguetj/calib doc/ (дата обращения: 05.06.2022)

ПРИЛОЖЕНИЕ 1 КОД ДЛЯ КАЛИБРОВКИ КАМЕР

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from PIL import Image
import torch
import torchvision
from torchvision import transforms as T
from cv2 import cv2
import random
import glob
import time
from math import fabs
torch.cuda.empty_cache()

model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
model.eval()
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
```

```
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier','stop sign'
]

count=0
jl=0
jr=0
left = cv2.VideoCapture(1)
right = cv2.VideoCapture(0)
CAMERA_WIDTH = 1280
CAMERA_HEIGHT = 720

left.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)
left.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)
right.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH)
right.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT)


'''def cropHorizontal(image):
    return image[:,
            int((CAMERA_WIDTH-CROP_WIDTH)/2):
            int(CROP_WIDTH+(CAMERA_WIDTH-CROP_WIDTH)/2)]'''
if not left.isOpened() or not right.isOpened():
    print("Cannot open camera")
    exit()
```

```
net = cv2.dnn.readNetFromDarknet('C:/yolov4/yolov3.cfg',
'C:/yolov4/yolov3.weights')
classes = []
with open("C:/yolov4/coco.names","r") as f:
    classes = [line.strip() for line in f.readlines()]

layer_names = net.getLayerNames()
outputlayers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
colors= np.random.uniform(0,255,size=(len(classes),3))

def pred(img,j,f):
    height,width,_ = img.shape
    blob = cv2.dnn.blobFromImage(img,0.00392,(128,128),(0,0,0),True,crop=False)
    net.setInput(blob)
    outs = net.forward(outputlayers)
    class_ids=[]
    confidences=[]
    boxes=[]
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.4:
                center_x= int(detection[0]*width)
                center_y= int(detection[1]*height)
                w = int(detection[2]*width)
                h = int(detection[3]*height)
                x=int(center_x - w/2)
```

```
            y=int(center_y - h/2)
            boxes.append([x,y,w,h]) #put all rectangle areas
            confidences.append(float(confidence)) #how confidence was that object
detected and show that percentage
            class_ids.append(class_id)
    indexes = cv2.dnn.NMSBoxes(boxes,confidences,0.4,0.6)
    font = cv2.FONT_HERSHEY_PLAIN
    for i in range(len(boxes)):
        if i in indexes:
            x,y,w,h = boxes[i]
            label = str(classes[class_ids[i]])
            if label=='stop sign':
                j+=1
                if j%3==0:
                    f=True
            color = colors[i]
    return img,j,f

while True:
    fl=False
    fr=False
    files = os.listdir(path="C:/foto/calibration 2camera/stereoL")
    if not (left.grab() and right.grab()):
        print("No more frames")
        break
    # Capture frame-by-frame 00392
    '''retl, framel = left.read()
    retr, framer = right.read()'''
    _, framel = left.retrieve()
```

```
    _, framer = right.retrieve()
    framl,jl,fl=pred(framel,jl,fl)
    framr,jr,fr=pred(framer,jr,fr)
    if fl and fr:
        cv2.imwrite('C:/foto/calibration 2camera/stereoL/%d.png' % count,framl)
        cv2.imwrite('C:/foto/calibration 2camera/stereoR/%d.png' % count,framr)
    count +=1
    cv2.imshow('windowl', framl)
    cv2.imshow('window', framr)
    if cv2.waitKey(25) & 0xFF == ord('q')or len(files)>9:
        cv2.destroyAllWindows()
        break



    # if frame is read correctly ret is True
    '''if not retl or not retr:
        print("Can't receive frame (stream end?). Exiting ...")
        break'''


    # Display the resulting frame

print('END')
# When everything done, release the capture

left.release()
right.release()
cv2.destroyAllWindows()
```

```
def get_prediction(img_path, threshold=0.5):
  img = Image.open(img_path)
  transform = T.Compose([T.ToTensor()])
  img = transform(img)
  img = img.to(device)
  pred = model([img])
  flag1=False
  #pred_score = list(pred[0]['scores'].detach().numpy())
  pred_score = list(pred[0]['scores'])

  pred_t = [pred_score.index(x) for x in pred_score if x>threshold]
  global pred_boxes, k
  if pred_t!=[]:

    pred_t=pred_t[-1]



    pred_class = [COCO_INSTANCE_CATEGORY_NAMES[i] for i in
list(pred[0]['labels'])]

    pred_boxes = [[(i[0], i[1]), (i[2], i[3])] for i in list(pred[0]['boxes'].detach())]
    pred_boxes = pred_boxes[:pred_t+1]

    k=1000
    for j, c in enumerate(pred_class):
      if c == 'stop sign':
        k=j
        if k<len(pred_boxes):
```

```
            flag1=True
            break

    return pred_boxes,k,flag1

def narezka(img_path,box,l):
    picture = Image.open(img_path)
    width, height = picture.size
    #(797.6748, 504.3219), (1167.3717, 839.1529)
    lx=int(box[l][0][0])
    ly=int(box[l][0][1])
    rx=int(box[l][1][0])
    ry=int(box[l][1][1])

    for x in range(1,width,1):
      for y in range(1,height,1):
          if x<lx-15 or x>rx+15 or y<ly-15 or y>ry+15:
            picture.putpixel( (x,y), (0, 0, 0, 255))

    return cv2.cvtColor(np.asarray(picture), cv2.COLOR_RGB2BGR)

import itertools
def Punkt(img):
    orb = cv2.ORB_create(nfeatures=180)
    kp = orb.detect(img, None)
    kp, _ = orb.compute(img, kp)
    return kp
def sift(img):
    detector = cv2.SIFT_create(nfeatures=180)
```

```
  #descriptor = cv2.DescriptorExtractor_create("SIFT")

  skp = detector.detect(img, None)
  skp, _ = detector.compute(img, skp)
  #skp, sd = descriptor.compute(img, skp)

  #tkp = detector.detect(template)
  #tkp, td = descriptor.compute(template, tkp)
  return skp

def detectionRED(frame, spisok):
    global approx
    img=frame.copy()
    frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    #cv2.imshow('frame_hsv',frame_hsv)
    mask1 = cv2.inRange(frame_hsv, (0,50,50), (30,225,225))
    mask2 = cv2.inRange(frame_hsv, (120,50,50), (255,225,225))
    frame_mask = cv2.bitwise_or(mask1, mask2 )
    #cv2.imshow('frame_mask',frame_mask)
    frame_dilate = cv2.dilate(frame_mask, None, iterations=2)
    #cv2.imshow('frame_dilate',frame_dilate)
    contours,  _  =  cv2.findContours(frame_dilate.copy(),cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    sort = spisok
    flag2=False
    for i in contours:
        area = cv2.contourArea(i)
        if 3000 < area < 100000000:
            sort.append(i)
```

```
    for cnt in sort:
      epsilon = 0.01*cv2.arcLength(cnt, True)
      approx = cv2.approxPolyDP(cnt, epsilon, True) #Функция cv::approxPolyDP аппроксимирует кривую или многоугольник
      # другой кривой/многоугольником с меньшим количеством вершин так, чтобы расстояние между ними было меньше или равно заданной точности.

      if len(approx) == 8:
         flag2=True
         '''cv2.drawContours(img, [approx], 0, (0), 1)
         for i in range(len(approx)):
           img = cv2.circle(img, (approx[i][0][0],approx[i][0][1]), radius=0, color=(0, 0, 0), thickness=1)'''
         for i in range(len(approx)):
              if approx[i][0][0]<10 or approx[i][0][1]<10:
                flag2=False
         break

    '''if len(approx)!=8:
       print(flag2)'''
    #cv2.imshow('img',img)
    #cv2.waitKey(0)
    return img, flag2, approx

def show_neuro_image(img_path, threshold=0.5, rect_th=1):
  boxes,q,flag1 = get_prediction(img_path, threshold)
  l=q  #баг с индексом в boxes
  if flag1:
    img = cv2.imread(img_path)
```

```
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cv2.rectangle(img,
                  (int(boxes[l][0][0]),int(boxes[l][0][1])),
                  (int(boxes[l][1][0]),int(boxes[l][1][1]))
                  ,color=(0, 255, 0), thickness=rect_th)
    plt.figure(figsize=(20,30))
    plt.imshow(img)
    plt.show()

def nearp(cernel,pts):
  d=[]
  r=[]
  flag=True
  for i in range(len(cernel)):
    d.append((cernel[i][0][0],cernel[i][0][1]))
  d=np.array(d)
  for i in range(len(d)):
    distances = np.linalg.norm(pts-d[i], axis=1)
    min_index = np.argmin(distances)
    r.append(list(pts[min_index]))
    if (fabs(d[i][0]-pts[min_index][0]) > 6 or fabs(d[i][1]-pts[min_index][1]) > 6):
      flag=False
  #print(cernel,'\n',r)
  return(r,flag)

def split(c):
  str=c.split('\\')
  return str[1]
def get_impoints(imagePaths,cern):
```

```
razdelenieimg=[]
t=[]
for z, c in enumerate(imagePaths):
  if z>=0:
    b, index, flag1 = get_prediction(c)
    print(c)

    if flag1:
      s=[]
      pix=narezka(c,b,index)
      n=split(c)
      path='C:/foto/calibration 2camera/stlap/'
      cv2.imwrite(path+n,pix)

      #kp = sift(pix)
      img, flag, cernel=detectionRED(pix,s)
      if flag:

        pts = harris(img)
        r,flag=nearp(cernel,pts)
        if flag==0:
          continue
        cern.append(r)   #вернуть для работы
        n=split(c)

        razdelenieimg.append(split(c))
        t.append(True)

        #pts = cv2.KeyPoint_convert(kp)
```

```
        '''for i in range(len(pts)):
          img = cv2.circle(img, (int(pts[i][0]),int(pts[i][1])), radius=0, color=(0, 0,
255), thickness=-1)'''
        for i in range(len(r)):
          img = cv2.circle(img, (int(r[i][0]),int(r[i][1])), radius=0, color=(0, 255, 0),
thickness=-1)
          #img[int(pts[i][0]),int(pts[i][1])]=[0,255,0]

        #img = cv2.drawKeypoints(img, kp, None, color=(0, 255, 0), flags=0)
        cv2.putText(img, "0", (cernel[0][0][0], cernel[0][0][1]),
cv2.FONT_HERSHEY_COMPLEX, 1, 0,2)
        cv2.putText(img, "1", (cernel[1][0][0], cernel[1][0][1]),
cv2.FONT_HERSHEY_COMPLEX, 1, 0,2)

        PIL_image = Image.fromarray(np.uint8(img)).convert('RGB')
        plt.figure(figsize=(20,30))
        plt.imshow(PIL_image)
        plt.show()

    if z>57:
      break
  return cern, razdelenieimg, t
#409, 315

objp=np.array([[[0,0,0],[69,0,0],[118,49,0],[118,118,0],[69,167,0],[0,167,0],[-
49,118,0],[-49,49,0]]])
objpp = np.zeros((1, 8, 3), np.float32)
for i in range(len(objp[0])):
```

```
  objpp[0][i][0] =objp[0][i][0]
  objpp[0][i][1] =objp[0][i][1]

objl=np.array([[[0,0,0],[69,0,0],[118,49,0],[118,118,0],[69,167,0],[0,167,0],[-
49,118,0],[-49,49,0]]])
objll = np.zeros((1, 8, 3), np.float32)
for i in range(len(objp[0])):
  objll[0][i][0] =objl[0][i][0]
  objll[0][i][1] =objl[0][i][1]
Lobjpoints =[]
Robjpoints =[]

def find_centroids(dst, gray):
    ret, dst = cv2.threshold(dst, 0.0001 * dst.max(), 255, 0)
    dst = np.uint8(dst)



    # find centroids
    ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)
    # define the criteria to stop and refine the corners
    criteria = (cv2.TERM_CRITERIA_EPS , 400,
                0.1)
    corners = cv2.cornerSubPix(gray,np.float32(centroids),(10,10),
              (-1,-1),criteria)
    return corners
def harris(image):
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  gray = np.float32(gray)
```

```
  dst = cv2.cornerHarris(gray, 16, 5, 0.04)
  dst = cv2.dilate(dst, None)
  #image[dst > 0.01*dst.max()] = [255, 0, 0]
  corners= find_centroids(dst,gray)
  for corner in corners:
    image[int(corner[1]), int(corner[0])] = [0, 0, 255]
  return corners
  '''cv2.imshow('dst', image)
  cv2.waitKey(0)
  cv2.destroyAllWindows()'''

leftImageDir = 'C:/foto/calibration 2camera/stereoL'
imagePaths = glob.glob("{0}/*.png".format(leftImageDir))
p=[]
_, razdelenieimg,_ = get_impoints(imagePaths,p)
print(razdelenieimg)

leftImageDir = 'C:/foto/calibration 2camera/stereoR'
imagePaths = glob.glob("{0}/*.png".format(leftImageDir))
p=[]
f=[]
for i in razdelenieimg:
  f.append(leftImageDir+'\\'+i)
cernelR,razdelenieimg2,t=get_impoints(f,p)

for i in t:
  if i:
    Robjpoints.append(objpp)
  else:
```

```
    Robjpoints.append(objll)

cer = np.zeros((1, 8, 2), np.float32)

Rimgpoints=[]
for i in range(len(cernelR)):
  z=cer.copy()
  Rimgpoints.append(z)

for i in range(len(cernelR)):
  for j in range(8):
    Rimgpoints[i][0][j][0]=cernelR[i][j][0]
    Rimgpoints[i][0][j][1]=cernelR[i][j][1]

print(len(Rimgpoints))

img = cv2.imread(imagePaths[0])
img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(Robjpoints, Rimgpoints,
img.shape[::-1], None, None)
print(mtxR)

f=[]
for i in razdelenieimg2:
  f.append(leftImageDir+'\\'+i)
print(razdelenieimg2)
print(f)

leftImageDir = 'C:/foto/calibration 2camera/stereoL'
```

```
imagePaths = glob.glob("{0}/*.png".format(leftImageDir))
f=[]
p=[]
for i in razdelenieimg2:
  f.append(leftImageDir+'\\'+i)
cernelL, _,t2 = get_impoints(f,p)

for i in t2:
  if i:
    Lobjpoints.append(objpp)
  else:
    Lobjpoints.append(objll)

print(razdelenieimg)
print(razdelenieimg2)
print(len(razdelenieimg2))

cer = np.zeros((1, 8, 2), np.float32)
Limgpoints=[]
for i in range(len(cernelL)):
  a=cer.copy()
  Limgpoints.append(a)

for i in range(len(cernelL)):
  for j in range(8):
    Limgpoints[i][0][j][0]=cernelL[i][j][0]
    Limgpoints[i][0][j][1]=cernelL[i][j][1]

img = cv2.imread(imagePaths[0])
```

```
img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(Lobjpoints, Limgpoints,
img.shape[::-1], None, None)
print(mtxL)
print(distL)
print(mtxR)
print(distR)

from scipy.linalg import svd
def compute_H(objpoints, imgpoints):
  N = 7

  M = np.zeros((2*N, 9), dtype=np.float64)
  for i in range(N):
    X=objpoints[i][0]
    Y=objpoints[i][1]
    u=imgpoints[i][0]
    v=imgpoints[i][1]
    #print(u)
    row_1 = np.array([ -X, -Y, -1, 0, 0, 0, X*u, Y*u, u])
    row_2 = np.array([ 0, 0, 0, -X, -Y, -1, X*v, Y*v, v])
    M[2*i] = row_1
    M[(2*i) + 1] = row_2
  u, s, vh = np.linalg.svd(M)
  h = vh[np.argmin(s)]
  h = h.reshape(3, 3)
  #нормальное псевдорешение

  return h
```

```
H=[]
for i in range(len(cernelL)):
  #print(i)
  h=compute_H(Lobjpoints[i][0],cernelL[i])
  h.reshape(3, 3)
  H.append(h)
  #print(h,'\n')

def get_intrinsic_parameters(H_r):
    M = len(H_r)
    V = np.zeros((2*M, 6), np.float64)
    def v_pq(p, q, H):
        v = np.array([
                H[0, p]*H[0, q],
                H[0, p]*H[1, q] + H[1, p]*H[0, q],
                H[1, p]*H[1, q],
                H[2, p]*H[0, q] + H[0, p]*H[2, q],
                H[2, p]*H[1, q] + H[1, p]*H[2, q],
                H[2, p]*H[2, q]
            ])
        return v

    for i in range(M):
        H = H_r[i]
        V[2*i] = v_pq(p=0, q=1, H=H)
        V[2*i + 1] = np.subtract(v_pq(p=0, q=0, H=H), v_pq(p=1, q=1, H=H))
    # solve V.b = 0
    u, s, vh = np.linalg.svd(V)
```

```
    # print(u, "\n", s, "\n", vh)
    b = vh[np.argmin(s)]
    print("V.b = 0 Solution : ", b)

    # according to zhangs method
    vc = (b[1]*b[3] - b[0]*b[4])/(b[0]*b[2] - b[1]**2)
    l = b[5] - (b[3]**2 + vc*(b[1]*b[2] - b[0]*b[4]))/b[0]
    alpha = np.sqrt((l/b[0]))
    beta = np.sqrt(((l*b[0])/(b[0]*b[2] - b[1]**2)))
    gamma = -1*((b[1])*(alpha**2) *(beta/l))
    uc = (gamma*vc/beta) - (b[3]*(alpha**2)/l)

    '''print([vc,
          l,
          alpha,
          beta,
          gamma,
       uc])'''

    A = np.array([
          [alpha, gamma, uc],
          [0, beta, vc],
          [0, 0, 1.0],
       ])
    print("Intrinsic Camera Matrix is :")
    #print(A)
    return A
get_intrinsic_parameters(H)
```

```
print('Ro:  ',len(Robjpoints),'\n','Lo:  ',len(Lobjpoints),'\n','Ri:
',len(Rimgpoints),'\n','Li: ',len(Limgpoints),'\n')

print(distL)
print(distR)

distL=np.zeros((1,5))
distR=np.zeros((1,5))

print("Calibrating cameras together...")
(_, _, _, _, _, rotationMatrix, translationVector, _, _) = cv2.stereoCalibrate(
      Robjpoints, Limgpoints, Rimgpoints,
      mtxL, distL,
      mtxR, distR,
      img.shape[::-1], None, None, None, None,
      cv2.CALIB_USE_INTRINSIC_GUESS)


print("Rectifying cameras...")

(leftRectification, rightRectification, leftProjection, rightProjection,
      _, leftROI, rightROI) = cv2.stereoRectify(
            mtxL, distL,
            mtxR, distR,
            img.shape[::-1], rotationMatrix, translationVector,
            None, None, None, None, None,
            cv2.CALIB_ZERO_DISPARITY)

print("Saving calibration...")
```

```
outputFile = 'C:/foto/calibration 2camera/outputFileZnak'
leftMapX, leftMapY = cv2.initUndistortRectifyMap(
        mtxL, distL, leftRectification,
        leftProjection, img.shape[::-1], cv2.CV_32FC1)
rightMapX, rightMapY = cv2.initUndistortRectifyMap(
        mtxR, distR, rightRectification,
        rightProjection, img.shape[::-1], cv2.CV_32FC1)

np.savez_compressed(outputFile, imageSize=img.shape[::-1],
        leftMapX=leftMapX, leftMapY=leftMapY, leftROI=leftROI,
        rightMapX=rightMapX, rightMapY=rightMapY, rightROI=rightROI)
```