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

Институт кибернетики
Кафедра проблем управления

ОТЧЁТ ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Тема работы: Алгоритмы оценки положения и формы объектов по облаку точек

Отчет представлен к рассмотрению:

Студент группы КРБО-01-17 _____ П.Р. Кабанов

«__» _____ 2021 г.

Отчет утвержден
Допущен к защите:

Руководитель работы от кафедры _____ С.А.К. Диане
«__» _____ 2021 г.

Руководитель работы от университета _____ А. А. Сухоленцева

«__» _____ 2021 г.

Оглавление

Введение

Системы технического зрения необходимой составляющей для автономной работы РТС. Для построения визуальной одометрии используют камеры с глубиной, способные выдавать RGB-D изображение, представляющее собой RGB-изображение, в котором каждый пиксель имеет значение расстояния от объектива камеры. В конечном счете, это можно представить как облако точек в трехмерном пространстве. С помощью полученной информации можно составить картину об окружающей местности и объектах. В данной работе будет рассмотрен способ поиска объектов с использованием сверточных нейронных сетей и эволюционного алгоритма.

Постановка задачи

Большую часть объектов в трехмерном пространстве можно разбить на составные части набором примитивов, таких как параллелепипед, эллипс, пирамида, цилиндр и т.д. Это позволит как обнаруживать некоторые простые объекты в пространстве, так и сегментировать сложные объекты для последующей классификации и семантического описания. Для поиска целевого объекта на RGB-изображении можно использовать нейросеть для уменьшения области поиска в карте глубины. Для более качественного составления и описания визуальной одометрии с камеры можно сегментировать целевой объект для получения данных о его положении и ориентации в пространстве. В качестве решения в данной работе используется эволюционный алгоритм. Его преимуществом является то, что он не требует данных для обучения и больших вычислительных мощностей, в отличие от нейронных сетей, что делает его универсальным при определении объектов в трехмерном облаке точек.

Детектирование объекта на RGB-изображении

В задачах детектирования объектов и классификации изображений хорошие результаты показывают сверточные нейронные сети. В отличие от нейронных сетей прямого распространения, помимо полносвязных слоев в

них присутствуют сверточные слои и слои подвыборки, что позволяет определять больше признаков на входном изображении.

Операция свертки в случае работы с изображениями представляет собой вычисление нового значения заданного пикселя, при котором учитываются значения окружающих его соседних пикселей. Главным элементом свертки является ядро, которое представляет собой квадратную матрицу нечетного размера, по умолчанию 3x3. Ядро является матрицей весов, которая «скользит» над двумерным изображением, выполняя поэлементное умножение с той частью, над которой оно находится в данный момент, суммируя затем полученные значения в новый пиксель. Независимо от того, попадает ли входной признак в то же место, он определяется в зависимости от того, находится он в зоне ядра, создающего выходные данные, или нет. Это значит, что размер ядра сверточной нейронной сети определяет количество признаков, которые будут объединены для получения нового признака на выходе. Такой способ перемещения называется паддинг (padding), что позволяет сохранить исходный размер матрицы входного изображения проходя по каждому пикселю.

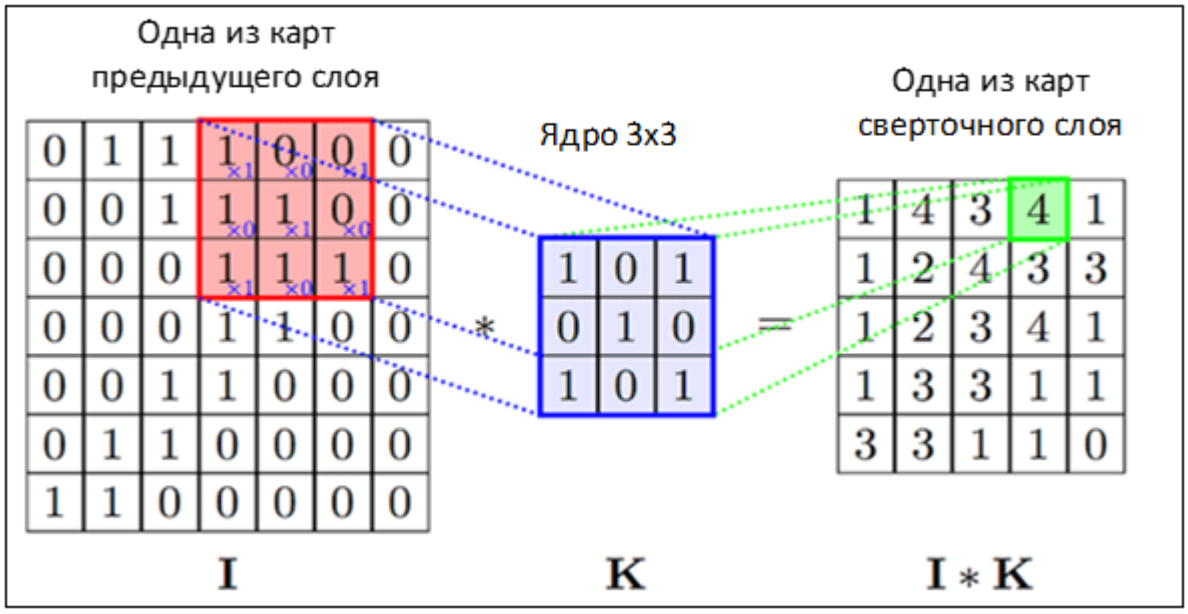

Рисунок 1. Пример свертки

Упрощенно слой можно описать формулой

$$x^l = f(x^{l-1} * k^l + b^l),$$

где

$x^l -$ *выход слоя* l,

$f(x) -$ *функция активации*,

$b^l -$ *коэффициент сдвига слоя* l,

$*$ $-$*операция свертки входа* x *с ядром* k.

Цель слоя подвыборки (pooling layer) – уменьшение размерности карт сверточного слоя. Если на предыдущей операции свертки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает модели избежать переобучения. Как и в сверточном слое, в слое подвыборки есть ядро, как правило, размером 2x2, которая позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Карта признаков разделяется на ячейки размером 2x2 элемента, из которых выбирается один. Зачастую применяют Max-Pool – выбор наибольшего значения в ячейке. Обычно в подвыборочном слое применяется функция активации ReLU.

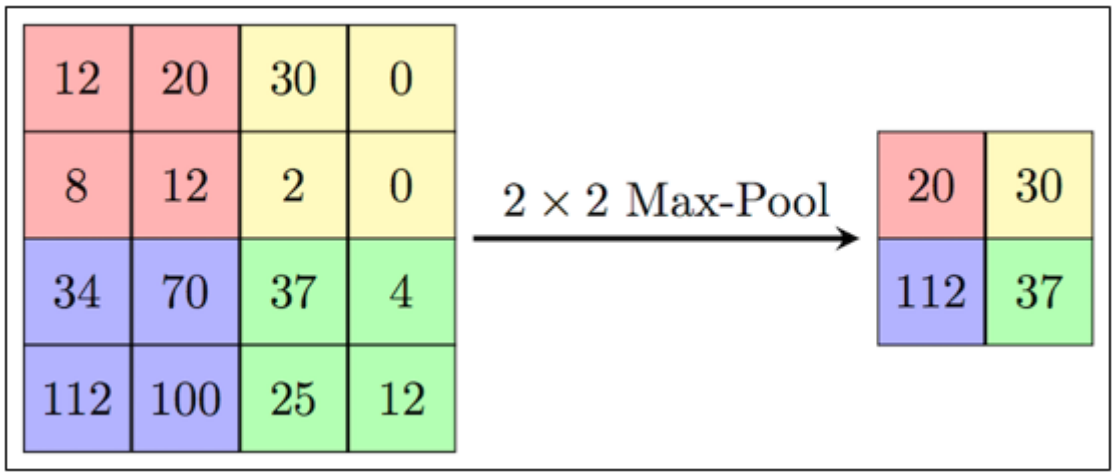

Рисунок 2. Пример работы слоя подвыборки

Формально подвыборочный слой можно описать формулой

$$x^l = f(a^l * subsample(x^{l-1}) + b^l),$$

где

$x^l -$ *выход слоя* l,

$f() -$ *функция активации,*

$a^l, b^l -$ *коэффициент сдвига слоя* l*,*

$subsample$ $-$ *операция выборки локальных максимальных значений.*

Полносвязный слой представляет собой слой обычного многослойного перцептрона, основной целью которого является классификация.

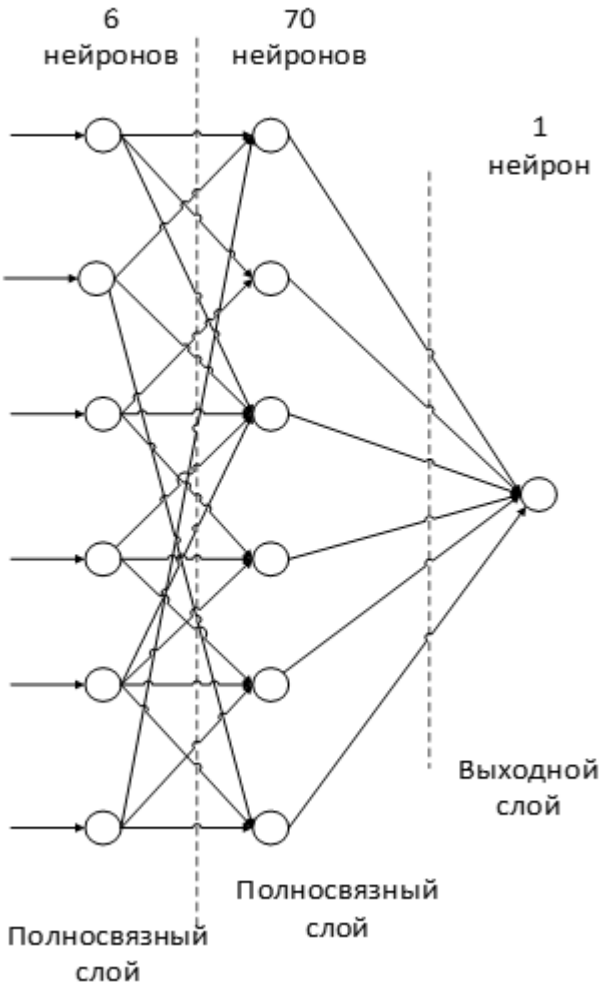

Рисунок 3. Пример полносвязных слоев

Математически полносвязный слой можно описать формулой:

$$x_j^l = f\left(\sum_i x_i^{l-1} * w_{i,j}^{l-1} + b_j^{l-1}\right),$$

где

$x_j^l -$ *выход предыдущего слоя* l*,*

$f -$ *функция активации,*

$b_j^{l-1} -$ *коэффициент сдвига слоя* l,

$w_{i,j}^{l} -$ *матрица весовых коэффициентов слоя* l.

Функции активации представляет собой непрерывную функцию, принимающую на вход вещественное число, а на выходе дает значение в интервале от -1 до 1.

Примерами функций активации являются сигмоида, гиперболический тангенс, ступенчатая функция, softsign, ReLU (Rectified Linear Unit) и др. В современных моделях широко применяется ReLU.

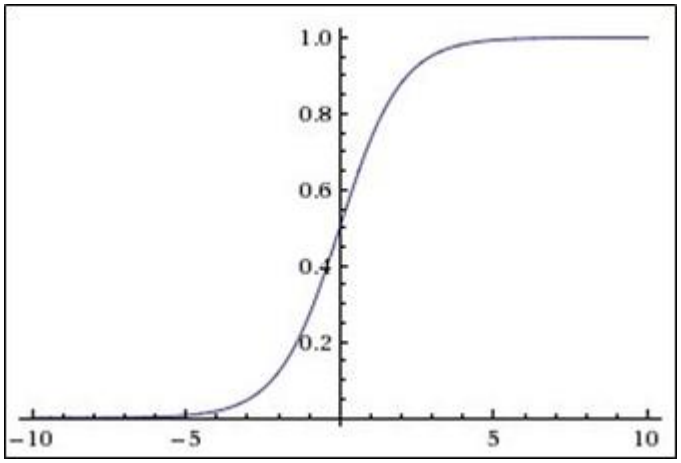

Рисунок 4. Пример сигмоидальной функции

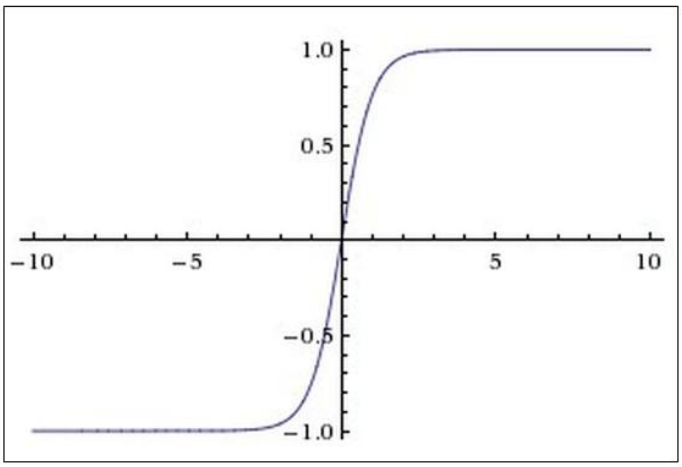

Рисунок 5. Пример функции гиперболического тангенса

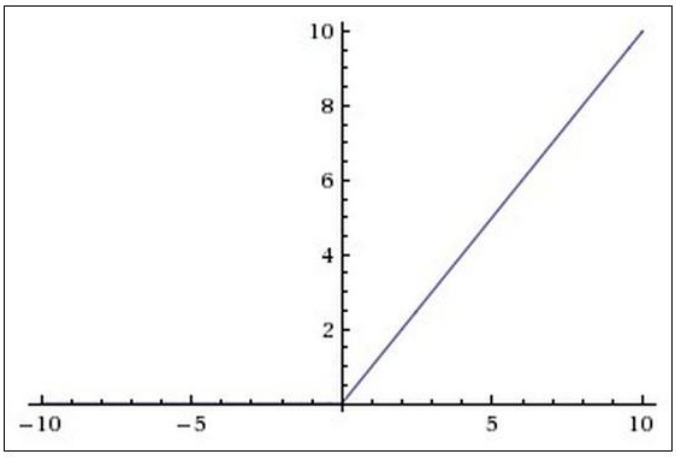

Рисунок 6. Пример функции ReLU

В общем виде топология сверточной нейронной сети имеет следующий вид:

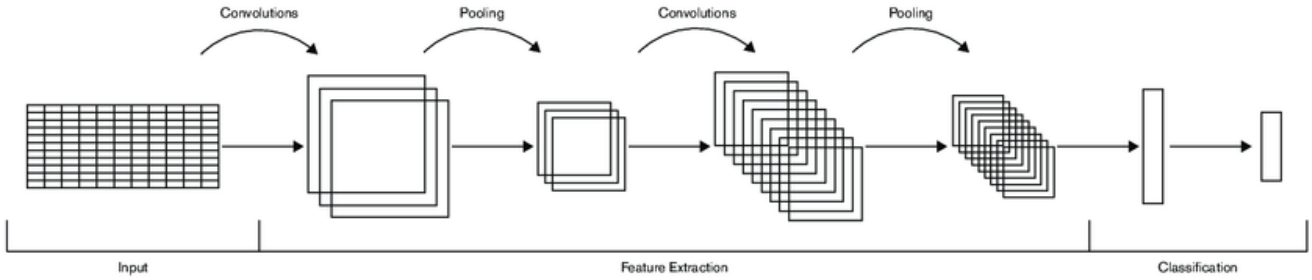

Рисунок 7. Топология сверточной нейронной сети

Одной из самых эффективных и быстродействующих моделей для детектирования объектов на изображении и сегментации является SSD MobileNetV2, разработанная специально для встраеваемых систем и активно использующаяся в мобильной робототехнике.

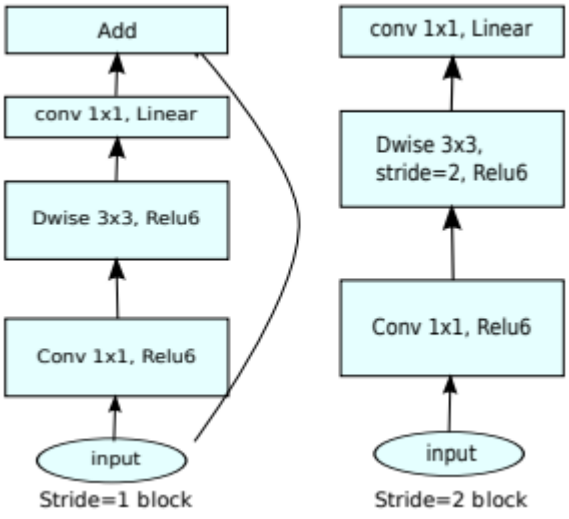

Рисунок 8. Блочная схема SSD MobileNetV2

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Рисунок 9. Архитектура SSD MobileNetV2

MobileNetV2 использует принцип Single Shot Detection (SSD). По изображению располагаются рамки (далее - bounding box) различных размеров, накрывающие участки пикселей, в которых потенциально может находиться целевой объект.

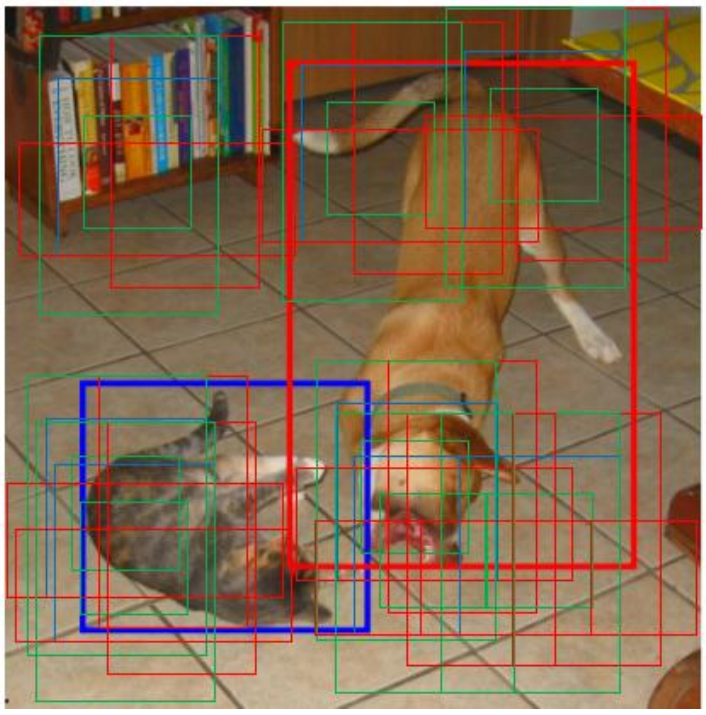

Рисунок 10. Single Shot Detection

В процессе обучения рамки по по умолчанию сравниваются по соотношению сторон, местоположению и масштабу с истинными. Выбираются блоки с наибольшим перекрытием с блоком искомого объекта. Пересечение над объединением (IoU – Intersection over Union) между предсказанными рамками и истинными должно быть больше 0.5. В итоге выбирается bounding box с наибольшим попаданием в рамку искомого объекта.

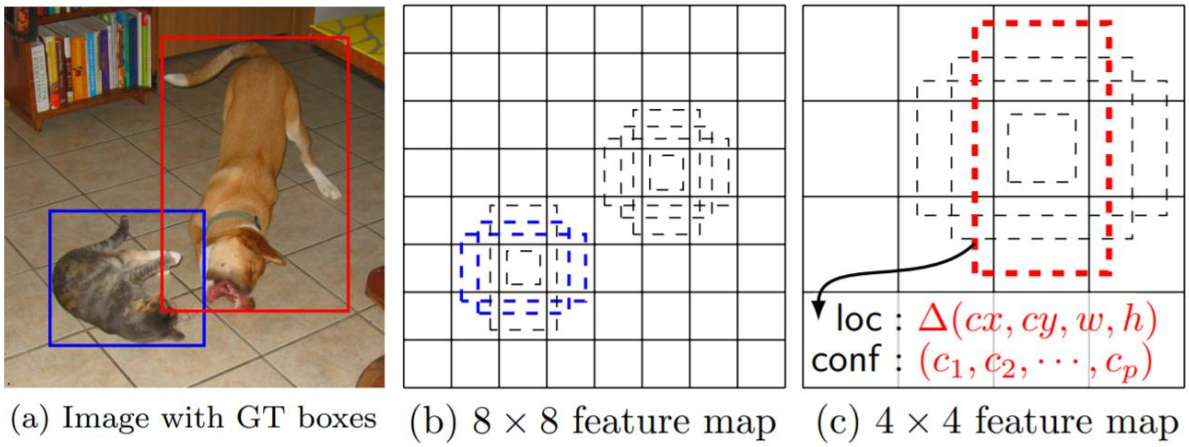

Рисунок 11. Пример работы SSD

Каждая гипотеза состоит из:

- Рамки со смещением положения. Δcx, Δcy, h и w представляют смещение от центра поля по умолчанию, его высоту и ширину.
- Уверенность в правильности найденного класса. Класс 0 зарезервирован под отсутствие объекта.

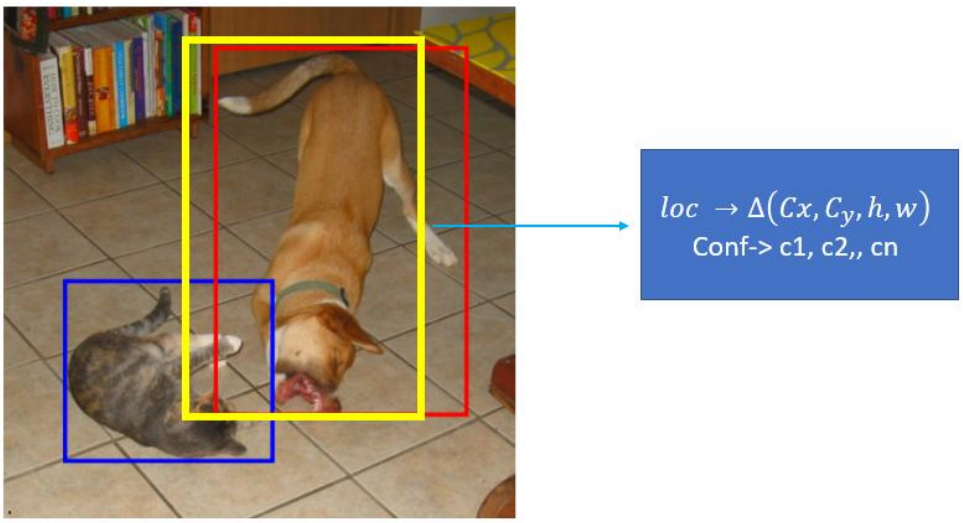

Рисунок 12. Гипотеза в SSD

Функция затрат в SSD называется MultiBox loss и состоит из confidence loss и localization loss.

SSD MobileNetV2 реализована во фреймворке Tensorflow Object Detection API и обучена на датасете COCO. С его помощью можно быстро находить объекты с видеопотока камеры мобильного робота.

Рисунок 13. Пример работы SSD MobileNevV2

Поиск объекта в трехмерном облаке точек

Для поиска объектов в трехмерном облаке точек был использован эволюционный алгоритм из предыдущей ППУ и ОПД. Он представляет

собой генетический алгоритм, в котором в качестве функции оценки особи используется функция из алгоритма RANSAC

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

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

В ходе работы был реализован поиск кубического объекта, а в качестве особи используется класс параллелепипеда с параметрами начальной точки, высоты, ширины, длины и наклона по осям X, Y, Z.

Программа была реализована на языке python3 с использованием фреймворка Open3D для визуализации.

Результаты работы

На рисунках ниже представлены результаты работы алгоритма

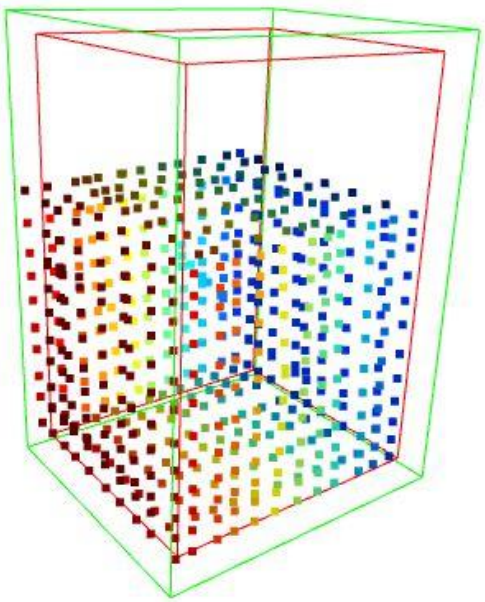

Рисунок 14. Результаты эксперимента 1

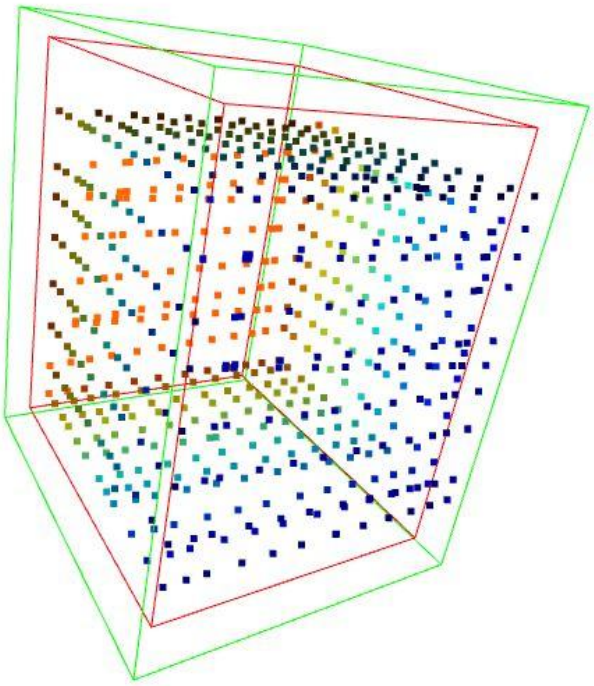

Рисунок 15. Результаты эксперимента 2

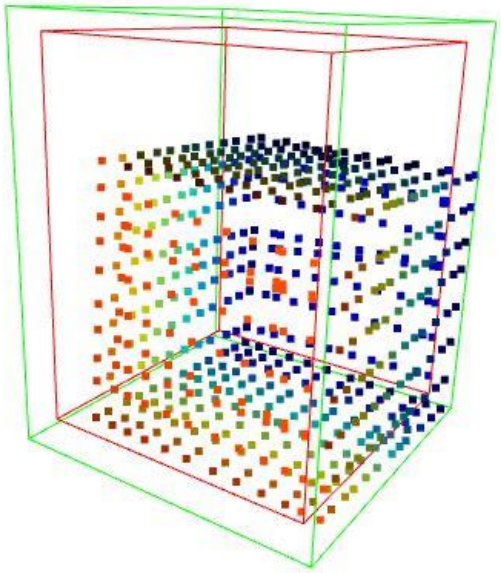

Рисунок 16. Результаты эксперимента 3

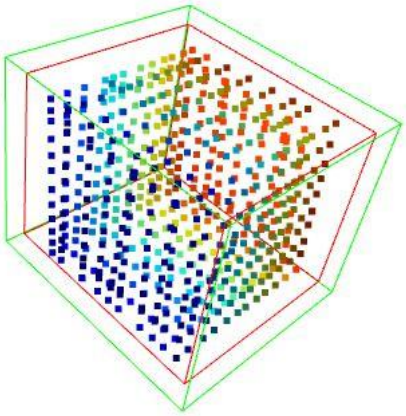

Рисунок 17. Результаты эксперимента 4

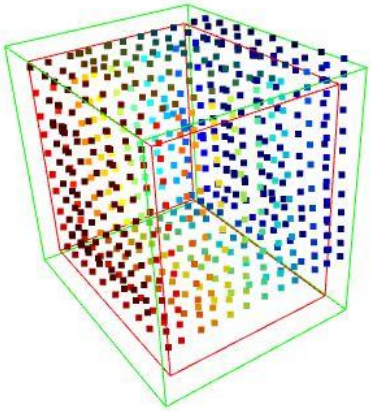

Рисунок 18. Результаты эксперимента 5

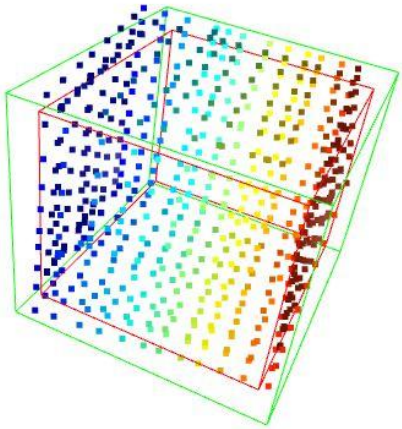

Рисунок 19. Результаты эксперимента 6

Вывод

Как видно из результатов эксперимента, с помощью генетического алгоритма удается достичь качественного оконтуривания объекта в трехмерном пространстве. По такому же принципу можно работать с эллиптическими, пирамидальными и цилиндрическими формами. Однако реализация алгоритма в данной научно-исследовательской работе хорошо показывает себя при работе с объектов, у которого нет большого количества шумов. Т.к. алгоритм RANSAC изначально предназначен для аппроксимации линейной функции в двумерном облаке точек, он предполагает сравнение количества лежащих в окрестности прямой точек с количеством не попавших в нее. Если схожий подход применять для полного RGB-D изображения, в котором присутствует большое количество точек за пределами искомого объекта (т.е. превышает количество точек, из которых состоит сам объект), результаты будут неудовлетворительными, так как фигура своей окрестностью будет пытаться охватить наибольшее количество точек на всем изображении, а не на искомом объекте.

Для упрощения задачи следует, во-первых, использовать обрезку вглубь RGB-D изображения bounding box'ом с RGB изображения, найденного через нейросеть. Для еще более качественного результата можно использовать Mask-RCNN, который позволяет четко выделить контуры изображения, а не использовать bounding box, однако это может повлечь за собой потерю важных точек объекта при обрезании вглубь облака. Во-вторых, следует разработать новую функцию оценки особи, которая бы не зависела от количества точек вне окрестности фигуры, и в целом, вероятно, использовать несколько иной подход. Нужна абсолютная метрика оценки особи, которая позволит понять, что определенная часть облака точек была описано нужным примитивом. В качестве примера, можно рассмотреть случай, когда облако сферической формы ошибочно описывается кубической, так как лучшего варианта алгоритмом найдено не было. Т.е. в случае ненахождения достаточно подходящей гипотезы предпочтительно не

делать ее вообще, чем делать ошибочную. Также дополнительно следует разработать способ оптимального поиска нескольких примитивов в облаке, например, использовать многопоточность либо последовательный поиск. В обоих случаях все еще нужна абсолютная метрика.

Список источников

1. Y. Zhang A deep Convolutional Neural Network for topology optimization with strong generalization ability // Department of Bridge Engineering, Tongji University – 2019
2. M. Sandler MobileNetV2: Inverted Residuals and Linear Bottlenecks // Google.inc – 2019
3. W. Liu SSD: Single Shot MultiBox Detector // Google.inc – 2016
4. SSD: Single Shot Detector for object detection using MultiBox [Электронный ресурс] – Электрон. текстовые дан. – 2019г. – Режим доступа: https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca
5. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество [Электронный ресурс] – Электрон. текстовые дан. – 2018г. – Режим доступа: https://habr.com/ru/post/348000/
6. П. Кабанов Отчет по практике по ППУ и ОПД //КПУ, РТУ МИРЭА – 2020

Приложение

```
import open3d as o3d
import numpy as np
import os
import sys
import math
import random
from time import sleep

class cube:
    def __init__(self, x, y, z, w, l, h, thetaX, thetaY, thetaZ):
        self.x = x
        self.y = y
        self.z = z
        self.w = w
        self.l = l
        self.h = h
        thresh = 0.15
        self.thetaX = thetaX
        self.thetaY = thetaY
        self.thetaZ = thetaZ

        self.centerx = x + w/2
        self.centery = y + l/2
        self.centerz = z + h/2
        self.pts_total = 0
        self.inliers_total =0
        ptA = [x, y, z]
        ptB = [x+self.w, y, z]
        ptC = [x, y+self.l, z]
        ptD = [x+self.w, y+self.l, z]
        ptE = [x, y, z+self.h]
        ptF = [x+self.w, y, z+self.h]
        ptG = [x, y+self.l, z+self.h]
        ptH = [x+self.w, y+self.l, z+self.h]

        pointsInit = [
            ptA,
            ptB,
            ptC,
            ptD,
            ptE,
            ptF,
            ptG,
            ptH
        ]
        ptA1 = [x-thresh, y-thresh, z-thresh]
        ptB1 = [x+self.w+thresh, y-thresh, z-thresh]
        ptC1 = [x-thresh, y+self.l+thresh, z-thresh]
        ptD1 = [x+self.w+thresh, y+self.l+thresh, z-thresh]
        ptE1 = [x-thresh, y-thresh, z+self.h+thresh]
        ptF1 = [x+self.w+thresh, y-thresh, z+self.h+thresh]
        ptG1 = [x-thresh, y+self.l+thresh, z+self.h+thresh]
        ptH1 = [x+self.w+thresh, y+self.l+thresh, z+self.h+thresh]

        pointsInit1 = [
            ptA1,
            ptB1,
            ptC1,
            ptD1,
            ptE1,
            ptF1,
            ptG1,
```

```
            ptH1
        ]
        lines = [
            [0, 1],
            [0, 2],
            [1, 3],
            [2, 3],
            [4, 5],
            [4, 6],
            [5, 7],
            [6, 7],
            [0, 4],
            [1, 5],
            [2, 6],
            [3, 7],
        ]
        lines1 = [
            [0, 1],
            [0, 2],
            [1, 3],
            [2, 3],
            [4, 5],
            [4, 6],
            [5, 7],
            [6, 7],
            [0, 4],
            [1, 5],
            [2, 6],
            [3, 7],
        ]

        colors = [[1, 0, 0] for i in range(len(lines))]
        colors1 = [[0, 1, 0] for i in range(len(lines1))]

        self.line_set = o3d.geometry.LineSet(
            points=o3d.utility.Vector3dVector(pointsInit),
            lines=o3d.utility.Vector2iVector(lines),
        )
        self.line_set1 = o3d.geometry.LineSet(
            points=o3d.utility.Vector3dVector(pointsInit1),
            lines=o3d.utility.Vector2iVector(lines1),
        )
        self.line_set.colors = o3d.utility.Vector3dVector(colors)
        self.line_set1.colors = o3d.utility.Vector3dVector(colors1)

rotation=self.line_set.get_rotation_matrix_from_xyz((math.radians(self.thetaX
),math.radians(self.thetaY),math.radians(self.thetaZ)))
        self.line_set.rotate(rotation,
(self.centerx,self.centery,self.centerz))
        self.line_set1.rotate(rotation,
(self.centerx,self.centery,self.centerz))

    def check_pts(self, points):
        pts_tfd = np.asarray(self.line_set.points)
        ptA, ptB, ptC, ptD, ptE, ptF, ptG, ptH = pts_tfd
        pts_tfd1 = np.asarray(self.line_set1.points)
        ptA1, ptB1, ptC1, ptD1, ptE1, ptF1, ptG1, ptH1 = pts_tfd1

        dir1 = (ptB-ptA)
        size1 = np.linalg.norm(dir1)
        dir1 = dir1 / size1

        dir2 = (ptC-ptA)
```

```
        size2 = np.linalg.norm(dir2)
        dir2 = dir2 / size2

        dir3 = (ptE-ptA)
        size3 = np.linalg.norm(dir3)
        dir3 = dir3 / size3

        dir11 = (ptB1-ptA1)
        size11 = np.linalg.norm(dir11)
        dir11 = dir11 / size11

        dir21 = (ptC1-ptA1)
        size21 = np.linalg.norm(dir21)
        dir21 = dir21 / size21

        dir31 = (ptE1-ptA1)
        size31 = np.linalg.norm(dir31)
        dir31 = dir31 / size31

        cube_center = np.array([self.centerx, self.centery,
self.centerz]).reshape(1,3)
        self.pts_total = points.shape[0]
        pts_indices = np.arange(points.shape[0])
        indices = pts_indices.tolist()
        dir_vec =points - cube_center
        res1 = np.where( (np.absolute(np.dot(dir_vec, dir1)) * 2) >= size1
)[0]
        res2 = np.where( (np.absolute(np.dot(dir_vec, dir2)) * 2) >= size2
)[0]
        res3 = np.where( (np.absolute(np.dot(dir_vec, dir3)) * 2) >= size3
)[0]

        res11 = np.where( (np.absolute(np.dot(dir_vec, dir11)) * 2) >= size11
)[0]
        res21 = np.where( (np.absolute(np.dot(dir_vec, dir21)) * 2) >= size21
)[0]
        res31 = np.where( (np.absolute(np.dot(dir_vec, dir31)) * 2) >= size31
)[0]

        fits =   list( set().union(res1, res2, res3) )
        fits1 = set(indices)- set(list( set().union(res11, res21, res31) ))
        self.inliers_total = len(np.intersect1d(fits,list(fits1)))

        return self.inliers_total

class GA:
    def __init__(self):
        self.population=[]
        self.fitness =[]

    def init_popultation(self, size):
        self.size = size
        for _ in range(self.size):
            rand_x = random.uniform(-3, 3)
            rand_y = random.uniform(-3, 3)
            rand_z = random.uniform(-3, 3)
            rand_w = random.uniform(0.2, 4)
            rand_l = random.uniform(0.2, 4)
            rand_h = random.uniform(0.2, 4)
            rand_thetaX = random.uniform(0, 360)
            rand_thetaY = random.uniform(0, 360)
            rand_thetaZ = random.uniform(0, 360)
```

```
self.population.append(cube(rand_x,rand_y,rand_z,rand_w,rand_l,rand_h,rand_th
etaX,rand_thetaY,rand_thetaZ))

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

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

    def crossover(self, r1, r2):
        a = np.random.random(1)
        b = 1 - a

        x_new_1 = a * r1.x + b * r2.x
        y_new_1 = a * r1.y + b * r2.y
        z_new_1 = a * r1.y + b * r2.y
        w_new_1 = a * r1.w + b * r2.w
        l_new_1 = a * r1.l + b * r2.l
        h_new_1 = a * r1.h + b * r2.h
        thetaX_new_1 = a * r1.thetaX + b * r2.thetaX
        thetaY_new_1 = a * r1.thetaY + b * r2.thetaY
        thetaZ_new_1 = a * r1.thetaZ + b * r2.thetaZ
        self.cube_new_1 =
cube(x_new_1,y_new_1,y_new_1,w_new_1,l_new_1,h_new_1,thetaX_new_1,thetaY_new_
1,thetaZ_new_1)
        a = 1 - a
        b = 1 - b
        x_new_2 = a * r1.x + b * r2.x
        y_new_2 = a * r1.y + b * r2.y
        z_new_2 = a * r1.z + b * r2.z
        w_new_2 = a * r1.w + b * r2.w
        l_new_2 = a * r1.l + b * r2.l
        h_new_2 = a * r1.h + b * r2.h
        thetaX_new_2 = a * r1.thetaX + b * r2.thetaX
        thetaY_new_2 = a * r1.thetaY + b * r2.thetaY
        thetaZ_new_2 = a * r1.thetaZ + b * r2.thetaZ
        self.cube_new_2 =
cube(x_new_2,y_new_2,y_new_2,w_new_2,l_new_2,h_new_2,thetaX_new_2,thetaY_new_
2,thetaZ_new_2)

        return self.cube_new_1, self.cube_new_2

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

    def mutate(self, cube):
        cube.x += int(0.9*(1-1.1*np.random.random(1)[0]))
```

```
            cube.y += int(0.9*(1-1.1*np.random.random(1)[0]))
            cube.z += int(0.9*(1-1.1*np.random.random(1)[0]))
            cube.w += int(0.9*(1-1.1*np.random.random(1)[0]))
            cube.l += int(0.9*(1-1.1*np.random.random(1)[0]))
            cube.h += int(0.9*(1-1.1*np.random.random(1)[0]))
            cube.thetaX += 0.9*(1-1.1*np.random.random(1)[0])
            cube.thetaY += 0.9*(1-1.1*np.random.random(1)[0])
            cube.thetaZ += 0.9*(1-1.1*np.random.random(1)[0])
            return cube

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

    if __name__ == '__main__':
        vis = o3d.visualization.Visualizer()
        number_steps = 70
        best_fit = 0
        best_model = None
        algorithm = GA()
        algorithm.init_popultation(70)
        vis.create_window(width = 800, height = 600)
        pcd = o3d.io.read_point_cloud("models/cube_test1.ply")
        vis.add_geometry(pcd)
        geometry_list = list()

        for _ in range(number_steps):
            #random.shuffle(algorithm.population)
            vis.add_geometry(pcd)
            for i in range(len(algorithm.population)):
                #vis.add_geometry(algorithm.population[i].line_set)
                #vis.add_geometry(algorithm.population[i].line_set1)
                inliers =
    algorithm.population[i].check_pts(np.asarray(pcd.points))
            algorithm.calc_fitness()
            if algorithm.fitness[0]>best_fit:
                best_fit = algorithm.fitness[0]
                print('Found better fintess: ',best_fit, 'at', _ )
                best_model = algorithm.population[0]

            algorithm.selection()
            algorithm.repopulate()
            algorithm.mutate_population()
            #vis.poll_events()
            #vis.update_renderer()
            #vis.clear_geometries()


        vis.add_geometry(pcd)
        vis.add_geometry(best_model.line_set)
        vis.add_geometry(best_model.line_set1)
        vis.run()
        vis.destroy_window()
```