|  |  |  |
| --- | --- | --- |
| **КУРСОВОЙ ПРОЕКТ** | | |
| РАСПОЗНАВАНИЕ СВОБОДНЫХ ПАРКОВОЧНЫХ МЕСТ | | |
| по дисциплине «Микропроцессорные системы» | | |
| Выполнили | | |
| студенты гр. 3530904/70106 |  | Исаев В. С.  Куликов Н. C.  Иванов И. А.  Лялюк А. А.  Масленникова А. А. |
| Руководитель | | |
| доцент |  | Круглов С.К. |
| «25» декабря 2019г. | | |

Оглавление

[Условие задачи 3](#_Toc28139850)

[Используемое оборудование 3](#_Toc28139851)

[Анализируемые методы 4](#_Toc28139852)

[Сверточная нейронная сеть 5](#_Toc28139853)

[IoU 5](#_Toc28139854)

[Информирование пользователей о свободных местах 6](#_Toc28139855)

[Блок-схема программы 6](#_Toc28139856)

[Получение кадров с камеры и вывод в окно 7](#_Toc28139857)

[Raspberry Pi 4B. Удаленное управление через SSH 8](#_Toc28139858)

[Запуск проекта 8](#_Toc28139859)

[Результаты работы 9](#_Toc28139860)

[Выводы 9](#_Toc28139861)

[Исходные тексты программ 10](#_Toc28139862)

[Список литературы и электронных источников 14](#_Toc28139863)

# Условие задачи

Поставленная задача: распознать свободные места и оповестить абонентов системы по распознаванию парковочных мест любым доступным способом.

# Используемое оборудование

При выборе основной платы выбрана была Raspberry Pi. В нашем случае в качестве операционной системы используется Raspbian OS — родной дистрибутив для платы. Таким образом на данной плате мы можем не только запустить проект, но и вывести результаты благодаря порту HDMI в конструкции. Ниже представлены технические характеристики:

* Процессор: Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
* оперативная память: 4GB LPDDR4-2400 SDRAM;
* цифровой видеовыход: HDMI;
* композитный выход: 3,5 мм (4 pin);
* USB порты: USB 2.0×4;
* сеть: WiFi 802.11n, 10/100 Мб RJ45 Ethernet;
* Bluetooth: Bluetooth 4.1, Bluetooth Low Energy;
* разъем дисплея: Display Serial Interface (DSI);
* карта памяти: MicroSD;
* порты ввода-вывода: 40;
* разъем видеокамеры: MIPI Camera Serial Interface (CSI-2);

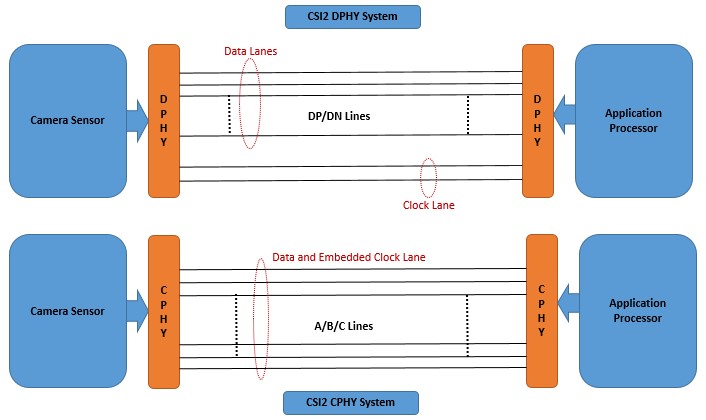
Последовательный интерфейс камеры (CSI) — это спецификация альянса MIPI. Он определяет интерфейс между камерой и хост-процессором. Последними активными спецификациями интерфейса являются CSI-2 v3.0, CSI-3 v1.1 и CCS v1.0, которые были выпущены в 2019, 2014 и 2017 годах соответственно.

MIPI CSI-2 является наиболее широко используемым интерфейсом камеры на мобильных и других рынках. Он получил широкое распространение благодаря своей простоте использования и способности поддерживать широкий спектр высокопроизводительных приложений, включая видео 1080p, 4K, 8K и более, а также фотографии с высоким разрешением.

Основные характеристики:

* Глубина цвета RAW-16 и RAW-24 оптимизирует внутрисценарный динамический диапазон (HDR) и отношение сигнал / шум (SNR), чтобы обеспечить возможности «расширенного обзора» для автономных транспортных средств и систем.
* Возможность использования до 32 виртуальных каналов обеспечивает распространение датчиков изображения с несколькими типами данных и поддерживает объединение датчиков с несколькими экспозициями и несколькими диапазонами для таких приложений Advanced Systems Assistance Systems (ADAS), как улучшенное предотвращение столкновений.
* Снижение задержки и эффективность транспорта (LRTE) обеспечивают агрегацию датчиков изображения без увеличения стоимости системы; облегчает восприятие, обработку и принятие решений в режиме реального времени; и оптимизирует транспорт для уменьшения количества проводов, скорости переключения и энергопотребления
* Дифференциальная импульсная кодовая модуляция (DPCM) Сжатие 12-10-12 сокращает полосу пропускания, обеспечивая превосходные изображения SNR без артефактов сжатия для критически важных приложений зрения
* Скремблирование уменьшает излучение спектральной плотности мощности (PSD), сводит к минимуму радиопомехи и обеспечивает дальнейший доступ к более длинным каналам.

|  |  |  |
| --- | --- | --- |
| **M-PHY speed** | **Clock rate** | **Bit rate** |
| Gear 1 | G1a | 1.25 Gbit/s |
| G1b | 1.49 Gbit/s |
| Gear 2 | G2a | 2.5 Gbit/s |
| G2b | 2.9 Gbit/s |
| Gear 3 | G3a | 5 Gbit/s |
| G3b | 5.8 Gbit/s |



# Анализируемые методы

Сверточная нейронная сеть, модифицированная для ускорения обучения на CUDA,

Так же решалась задача разделения двух парковочных мест методом Intersection Over Union (отношение площади пересечения к сумме площадей)

# Сверточная нейронная сеть

Свёрточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году[и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения. Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв и субдискретизирующих слоёв. Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

Наиболее простым и популярным способом обучения является метод обучения с учителем (на маркированных данных) — метод обратного распространения ошибки и его модификации. Но существует также ряд техник обучения свёрточной сети без учителя. Например, фильтры операции свёртки можно обучить отдельно и автономно, подавая на них вырезанные случайным образом кусочки исходных изображений обучающей выборки и применяя для них любой известный алгоритм обучения без учителя (например, автоассоциатор или даже метод k-средних) — такая техника известна под названием patch-based training. Соответственно, следующий слой свёртки сети будет обучаться на кусочках от уже обученного первого слоя сети. Также можно скомбинировать сверточную нейросеть с другими технологиями глубинного обучения.

В нашем случае нейросеть была обучена в соответствии с указаниями в статье “[YOLOv3: An Incremental Improvement](https://arxiv.org/abs/1804.02767)”

# IoU

В ходе работы мы столкнулись с проблемой, изображенной на рисунке

A car driving down a street

Description automatically generated

- несколько парковочных мест могут перекрывать друг друга. Нам нужно найти способ измерить степень пересечения двух объектов, чтобы искать только «наиболее пустые» рамки.

Мы воспользуемся мерой под названием Intersection Over Union (отношение площади пересечения к сумме площадей) или IoU. IoU можно найти, посчитав количество пикселей, где пересекаются два объекта, и разделить на количество пикселей, занимаемых этими объектами:

A picture containing screenshot

Description automatically generated

Так мы сможем понять, как сильно ограничивающая рамка машины пересекается с рамкой парковочного места. Это позволит легко определить, свободна ли парковка. Если значение IoU низкое, вроде 0.15, значит, машина занимает малую часть парковочного места. А если оно высокое, вроде 0.6, то это значит, что машина занимает большую часть места и там нельзя припарковаться.

Поскольку IoU используется довольно часто в компьютерном зрении, в соответствующих библиотеках с большой вероятностью есть реализация этой меры. В нашей библиотеке Mask R-CNN она реализована в виде функции mrcnn.utils.compute\_overlaps().

# Информирование пользователей о свободных местах

Для информирования была использована библиотека twilio. Twilio — это популярный API, который позволяет отправлять SMS из практически любого языка программирования с помощью всего нескольких строк кода. Чтобы использовать Twilio, мы зарегистрировали пробный аккаунт, и создали виртуальный номер телефона

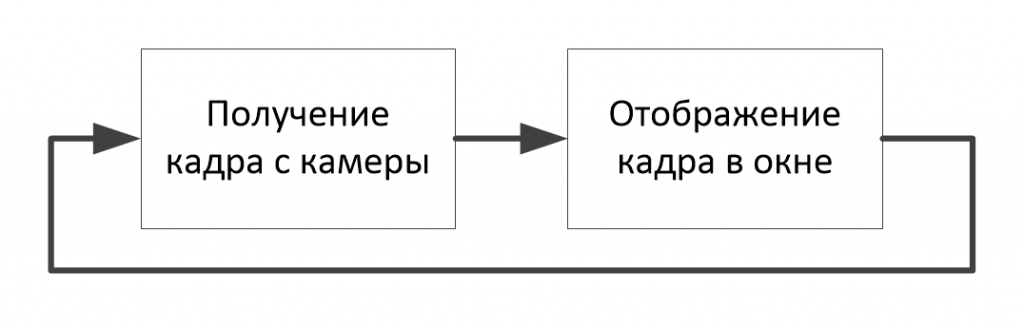
# Блок-схема программы

A screenshot of a cell phone

Description automatically generated

# Получение кадров с камеры и вывод в окно

Наша программа будет в бесконечном цикле получать кадры с камеры и проверять свободыне места, затем отображать их в окне. Алгоритм программы:



Запускаем python-скрипт:

$ python parking.py

Если в программе нет ошибок, откроется окно с видео. Порядок работы программы:

Функция cv2.inRange(src, lowerb, upperb[, dst]) позволяет выделить маску области. Однако резко встает вопрос корректного определения верхней и нижней границы (lowerb и upperb). Для выделения по цвету, достаточно указать диапазон H, а насыщенность и яркость будет сильно варьироваться (из-за разной освещенности, углов скоса объекта и т.д.) На практике обычно делают примерно так: если H составляющая 80 (зеленый цвет), берут нижнюю границу H=80-10 [70, 50, 50], а верхнюю H=80+10 [90,255,255]. Но, в отличие от цветовой модели RGB, диапазон значений H в разных программах может задаваться по-разному и его необходимо пересчитывать.

# Raspberry Pi 4B. Удаленное управление через SSH

SSH (Secure Shell) — сетевой протокол, позволяющий производить удаленное управление операционной системой через шифрованное соединение, а также передавать файлы, транслировать видео- и аудио потоки и сжимать передаваемые данные на лету.

Говоря простым языком, SSH — позволяет нам, через которую можно запустить на компьютере терминал для выполнения консольных команд на другом компьютере.

# Запуск проекта

1. Запускаем плату и подключаемся через SSH. Порт узнаем с помощью команды ifconfig.
2. Подключаемся командой ssh [pi@192.168.1.109](mailto:pi@192.168.1.109), далtе вводим предустановленные на этапе установки ОС логин и пароль
3. Получаем доступ к командной строке используемого raspberry
4. Запуск программы:

Python3 Desktop/main.py

# Результаты работы

A car is lined up in a parking lot full of cars

Description automatically generated

**На рисунке видно, что программа определила занятые и свободные парковочные места.**

# Выводы

В результате разработки получена программа, анализирующая видео с камеры и высылающая уведомление пользователю об освободившемся парковочном месте. Так же программа реализована для запуска на rasberryPi, что дает возможность мобильно устанавливать такие устройства во дворах и оперативно оповещать жильцов о парковке. Так же код находится в свободном доступе (<https://github.com/Vasiliy566/bonch_hack>

) под лицензией GNU General Public License v3.0, что дает возможность другим разработчикам использовать данное приложение.

# 

# Исходные тексты программ

import numpy as np

import cv2

import mrcnn.config

import mrcnn.utils

from mrcnn.model import MaskRCNN

from pathlib import Path

from twilio.rest import Client

# Конфигурация, которую будет использовать библиотека Mask-RCNN.

class MaskRCNNConfig(mrcnn.config.Config):

NAME = "coco\_pretrained\_model\_config"

IMAGES\_PER\_GPU = 1

GPU\_COUNT = 1

NUM\_CLASSES = 1 + 80 # в датасете COCO находится 80 классов + 1 фоновый класс.

DETECTION\_MIN\_CONFIDENCE = 0.6

# Фильтруем список результатов распознавания, чтобы остались только автомобили.

def get\_car\_boxes(boxes, class\_ids):

car\_boxes = []

for i, box in enumerate(boxes):

# Если найденный объект не автомобиль, то пропускаем его.

if class\_ids[i] in [3, 8, 6]:

car\_boxes.append(box)

return np.array(car\_boxes)

# Конфигурация Twilio.

twilio\_account\_sid = 'Ваш Twilio SID'

twilio\_auth\_token = 'Ваш токен аутентификации Twilio'

twilio\_phone\_number = 'Ваш номер телефона Twilio'

destination\_phone\_number = 'Номер, куда придёт сообщение'

client = Client(twilio\_account\_sid, twilio\_auth\_token)

# Корневая директория проекта.

ROOT\_DIR = Path(".")

# Директория для сохранения логов и обученной модели.

MODEL\_DIR = ROOT\_DIR / "logs"

# Локальный путь к файлу с обученными весами.

COCO\_MODEL\_PATH = ROOT\_DIR / "mask\_rcnn\_coco.h5"

# Загружаем датасет COCO при необходимости.

if not COCO\_MODEL\_PATH.exists():

mrcnn.utils.download\_trained\_weights(COCO\_MODEL\_PATH)

# Директория с изображениями для обработки.

IMAGE\_DIR = ROOT\_DIR / "images"

# Видеофайл или камера для обработки — вставьте значение 0, если использовать камеру, а не видеофайл.

VIDEO\_SOURCE = "test\_images/parking.mp4"

# Создаём модель Mask-RCNN в режиме вывода.

model = MaskRCNN(mode="inference", model\_dir=MODEL\_DIR, config=MaskRCNNConfig())

# Загружаем предобученную модель.

model.load\_weights(COCO\_MODEL\_PATH, by\_name=True)

# Местоположение парковочных мест.

parked\_car\_boxes = None

# Загружаем видеофайл, для которого хотим запустить распознавание.

video\_capture = cv2.VideoCapture(VIDEO\_SOURCE)

# Сколько кадров подряд с пустым местом мы уже видели.

free\_space\_frames = 0

# Мы уже отправляли SMS?

sms\_sent = False

# Проходимся в цикле по каждому кадру.

while video\_capture.isOpened():

success, frame = video\_capture.read()

if not success:

break

# Конвертируем изображение из цветовой модели BGR в RGB.

rgb\_image = frame[:, :, ::-1]

# Подаём изображение модели Mask R-CNN для получения результата.

results = model.detect([rgb\_image], verbose=0)

# Mask R-CNN предполагает, что мы распознаём объекты на множественных изображениях.

# Мы передали только одно изображение, поэтому извлекаем только первый результат.

r = results[0]

# Переменная r теперь содержит результаты распознавания:

# - r['rois'] — ограничивающая рамка для каждого распознанного объекта;

# - r['class\_ids'] — идентификатор (тип) объекта;

# - r['scores'] — степень уверенности;

# - r['masks'] — маски объектов (что даёт вам их контур).

if parked\_car\_boxes is None:

# Это первый кадр видео — допустим, что все обнаруженные машины стоят на парковке.

# Сохраняем местоположение каждой машины как парковочное место и переходим к следующему кадру.

parked\_car\_boxes = get\_car\_boxes(r['rois'], r['class\_ids'])

else:

# Мы уже знаем, где места. Проверяем, есть ли свободные.

# Ищем машины на текущем кадре.

car\_boxes = get\_car\_boxes(r['rois'], r['class\_ids'])

# Смотрим, как сильно эти машины пересекаются с известными парковочными местами.

overlaps = mrcnn.utils.compute\_overlaps(parked\_car\_boxes, car\_boxes)

# Предполагаем, что свободных мест нет, пока не найдём хотя бы одно.

free\_space = False

# Проходимся в цикле по каждому известному парковочному месту.

for parking\_area, overlap\_areas in zip(parked\_car\_boxes, overlaps):

# Ищем максимальное значение пересечения с любой обнаруженной

# на кадре машиной (неважно, какой).

max\_IoU\_overlap = np.max(overlap\_areas)

# Получаем верхнюю левую и нижнюю правую координаты парковочного места.

y1, x1, y2, x2 = parking\_area

# Проверяем, свободно ли место, проверив значение IoU.

if max\_IoU\_overlap < 0.15:

# Место свободно! Рисуем зелёную рамку вокруг него.

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 3)

# Отмечаем, что мы нашли как минимум оно свободное место.

free\_space = True

else:

# Место всё ещё занято — рисуем красную рамку.

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 1)

# Записываем значение IoU внутри рамки.

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, f"{max\_IoU\_overlap:0.2}", (x1 + 6, y2 - 6), font, 0.3, (255, 255, 255))

# Если хотя бы одно место было свободным, начинаем считать кадры.

# Это для того, чтобы убедиться, что место действительно свободно

# и не отправить лишний раз уведомление.

if free\_space:

free\_space\_frames += 1

else:

# Если всё занято, обнуляем счётчик.

free\_space\_frames = 0

# Если место свободно на протяжении нескольких кадров, можно сказать, что оно свободно.

if free\_space\_frames > 10:

# Отображаем надпись SPACE AVAILABLE!! вверху экрана.

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, f"SPACE AVAILABLE!", (10, 150), font, 3.0, (0, 255, 0), 2, cv2.FILLED)

# Отправляем сообщение, если ещё не сделали это.

if not sms\_sent:

print("SENDING SMS!!!")

message = client.messages.create(

body="Parking space open - go go go!",

from\_=twilio\_phone\_number,

to=destination\_phone\_number

)

sms\_sent = True

# Показываем кадр на экране.

cv2.imshow('Video', frame)

# Нажмите 'q', чтобы выйти.

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Нажмите 'q', чтобы выйти.

video\_capture.release()

cv2.destroyAllWindows()

# 

# Список литературы и электронных источников

1. <https://tproger.ru/>
2. <https://arxiv.org/abs/1804.02767>
3. <https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started>