**СОДЕРЖАНИЕ**

Введение ...................................................................................................................................4

[1. Анализ предметной области. Постановка задачи 5](#_Toc122304741)

[2. Разработка и описание алгоритма трекинга выбранных объектов в видеопотоке 6](#_Toc122304742)

[2.1. Инициализация графического интерфейса 7](#_Toc122304743)

[2.2. Отслеживание выбранного объекта в реальном времени через веб-камеру 7](#_Toc122304744)

[2.3. Отслеживание выбранного объекта из видеофайла 8](#_Toc122304745)

[3. Разработка программы 10](#_Toc122304746)

[4. Тестирование 18](#_Toc122304747)

[4.1. Подготовка к тестированию 18](#_Toc122304748)

[4.2. Тестирование 19](#_Toc122304749)

[Заключение 23](#_Toc122304750)

[Список использованной литературы 24](#_Toc122304751)

**ПРИЛОЖЕНИЕ А ТЕКСТ ПРОГРАММЫ**

**ПРИЛОЖЕНИЕ Б ГРАФИЧЕСКИЙ МАТЕРИАЛ. СХЕМА ТРЕКИНГА ВЫБРАННЫХ ОБЪЕКТОВ В ВИДЕОПОТОКЕ**

**ВВЕДЕНИЕ**

В медиасфере motion tracking – это процесс отслеживания движения объекта в кадре. По окончании трекинга программа «запоминает» и сохраняет траекторию движения.

Трекингом называют задачу сопровождения объектов. От детекции задача отличается тем, что нам не только надо задетектировать объект в первом кадре, но и понять, где он находится в следующем. Алгоритм анализирует кадры видео и выдает положение движущихся целевых объектов относительно кадра.

Что можно трекать?

* Людей.
* Животных
* Машины и прочую технику

Трекинг людей нужен чаще всего. Самая стандартная задача, на которой больше всего решений — трекинг людей на улице. Другая задача — подсчёт числа посетителей торговых точек, аналитика по торговому залу. Сюда же можно отнести задачи плана подсчёт входящих/выходящих людей в транспорте.

В основном трекинг животных применяется на фермах. В природе делают скорее детекцию, чтобы посчитать статистику. А на ферме очень полезно знать какая корова не отдоилась и куда она пошла. Как ни странно, но таких фирм очень много.

Машины и прочие самолёты обычно трекать очень просто. Они ездят по практически идеальным траекториям, что сильно упрощает работу алгоритма.

Конечно, можно трекать что угодно (точки на пальцах, или на лице), но 99% задач из вышеперечисленных категорий.

Основная проблема в трекинге состоит в сопоставлении положений целевого объекта на последовательности кадров, особенно если объект движется быстро относительно частоты кадров. Таким образом, системы трекинга обычно используют модель движения, которая описывает как может изменяться изображение целевого объекта при всевозможных различных его движениях.

изображение деформируемого объекта может быть покрыто сеткой (mesh), движение объекта задается положением вершин этой сетки

# Анализ предметной области. Постановка задачи

В данном курсовом проекте необходимо разработать приложение, позволяющее выполнять отслеживание выбранных объектов в видеопотоке. Для решения можно использовать любые современные нейросетевые модели. Можно использовать любые объекты для отслеживания (лицо, человек, кот, футбольный мяч и т.д.). Трекинг производить в реальном времени. Продемонстрировать работу приложения, выполнив сеанс отслеживания для выбранного видеофрагмента и объекта. Дополнительно задаются следующие исходные данные:

1. Язык программирования – Python 3.9.5.
2. Программная база – редактор кода VS Code.
3. Библиотеки для работы с изображениями – OpenCV2.
4. Фреймворк для работы с нейросетями – Tensorflow 2.

# Разработка и описание алгоритма трекинга выбранных объектов в видеопотоке

Перед реализацией программной системы необходимо установить и подключить необходимые библиотеки: OpenCV2, PyQt5, numpy, TensorFlow 2, Pillow, а также при наличии видеокарты NVidia рекомендуется установить драйвер CUDNN, CUDA Toolkit и видеодрайвер последней версии (ускорение обучения и обработки видеоряда будет происходить при помощи графического процессора). Для этого в терминале необходимо прописать команды “pip install --ignore-installed --upgrade tensorflow==2.5.0”, “pip install pillow”, запустить установщик CUDA Toolkit последней версии, запустить установщик CUDNN последней версии, добавить в системные переменную PATH среды Windows следующие строки:

* <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\bin
* <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\libnvvp
* <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\include
* <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\ CUDA \v11.2\ extras\ CUPTI\lib64
* <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v11.2\cuda\bin.

Создайте новую папку по выбранному вами пути и назовите ее TensorFlow. (например, C:\Users\andrey\Documents\TensorFlow).

С вашего терминала перейдите в каталог TensorFlow.

Чтобы загрузить модели, вы можете либо использовать Git для клонирования репозитория моделей TensorFlow внутри папки TensorFlow, либо просто загрузить его в виде ZIP-файла и извлечь его содержимое в папку TensorFlow. Чтобы сохранить согласованность, в последнем случае вам придется переименовать извлеченную папку models-master в models.

Теперь у вас должна быть одна папка с именем models в вашей папке TensorFlow, которая содержит еще 4 папки как таковые:

TensorFlow/

└─ models/

├─ community/

├─ official/

├─ orbit/

├─ research/

└── ...

Далее необходимо запустить команды “pip install cython”, “pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI”, после чего запустить команды “cp object\_detection/packages/tf2/setup.py .” , “python -m pip install --use-feature=2020-resolver .”. Установка необходимых библиотек завершена.

В ходе разработки программной системы были использованы следующие процедуры и функции:

1. Инициализация графического интерфейса
2. Отслеживание выбранного объекта в реальном времени через веб-камеру
3. Отслеживание выбранного объекта из видеофайла

## 2.1. Инициализация графического интерфейса

Выходные данные: вывод графического интерфейса на экран компьютера.

1. Создание процесса Qt.
2. Создание объекта класса главного окна
3. Инициализация кнопок и надписей в главном окне
4. Отрисовка рамки видеокадра и замена содержимого временным текстом “Waiting for command”
5. Ожидание нажатие пользователя на одну из пяти кнопок:
   1. Кнопка выхода
   2. Кнопка остановки любых обработок видео
   3. Кнопка выбора объекта
   4. Кнопка выбора видеофайла
   5. Кнопка запуска обработки видео с веб-камеры компьютера

## 2.2. Отслеживание выбранного объекта в реальном времени через веб-камеру

Входные данные: изображение видеокадра с веб-камеры компьютера.

Выходные данные: изображение с наложенным поверх трэком (путём объекта за прошедшее время) и рамка объекта с подписью и уверенностью нейросети в этой подписи.

1. Создание отдельного потока QThread для обработки видео
2. Выделение памяти на видеокарте (-ах)
3. Загрузка файла подписей к объектам (будут отображаться около рамки в видеокадре)
4. Загрузка натренированной нейронной сети для выделения в видеокадре определённого объекта и его классификации
5. Подключение к веб-камере и создание потока видео
6. Инициализация класса наложения кадра на изображение (для отрисовки пути выбранного объекта на изображении)
7. Отклик потока QThread главному потоку о готовности к обработке, в этот момент главный поток включает видимость окну с видеопотоком с веб-камеры.
8. Запуск цикла while True:
   1. Считывание кадра с потока веб-камеры
   2. Подгонка кадра к размеру 640х480 (для улучшенного быстродействия)
   3. Передача кадра в обработку нейронной сети, она возвращает координаты рамок объектов, их подписи, цвет рамки и уверенность в них
   4. Копирование кадра и отрисовка на этой копии кадра рамок и надписей объектов в которых нейронная сеть уверена более чем на 60% (включительно) при помощи модуля viz\_utils фреймворка TensorFlow
   5. Обработка выделенных объектов:
      1. Получение позиций центра рамок объектов и их цвет
      2. Запись текущих позиций в массив данных
      3. Если объект отслеживания был изменён (выбран другой) то идёт очистка массива данных с позициями объектов во времени
   6. За каждую запись в массиве позиций объектов отрисовка пути (трека) именно выбранного объекта в цвет рамки данного объекта
   7. Отправка обработанного кадра с рамками объектов, надписями и путём (треком) в главный поток, отрисовка его в главном окне программы
   8. Если установлен флаг завершения обработки, завершается обработка и останавливается поток QThread обработки изображения

## Отслеживание выбранного объекта из видеофайла

Входные данные: видеокадр из файла с объектом отслеживания.

Выходные данные: изображение с наложенным поверх трэком (путём объекта за прошедшее время) и рамка объекта с подписью и уверенностью нейросети в этой подписи

1. Получение названия файла и его пути из диалогового окна QFileDialog, выбранного пользователем
2. Создание отдельного потока QThread для обработки видео и отправка названия файла этому потоку
3. Выделение памяти на видеокарте (-ах)
4. Загрузка файла подписей к объектам (будут отображаться около рамки в видеокадре)
5. Загрузка натренированной нейронной сети для выделения в видеокадре определённого объекта и его классификации
6. Подключение к файловой системе компьютера и создание потока видео из видеофайла
7. Инициализация класса наложения кадра на изображение (для отрисовки пути выбранного объекта на изображении)
8. Отклик потока QThread главному потоку о готовности к обработке, в этот момент главный поток включает видимость окну с видеопотоком из файла.
9. Запуск цикла while True:
   1. Считывание кадра с потока файла
   2. Подгонка кадра к размеру 640х480 (для улучшенного быстродействия)
   3. Передача кадра в обработку нейронной сети, она возвращает координаты рамок объектов, их подписи, цвет рамки и уверенность в них
   4. Копирование кадра и отрисовка на этой копии кадра рамок и надписей объектов в которых нейронная сеть уверена более чем на 60% (включительно) при помощи модуля viz\_utils фреймворка TensorFlow
   5. Обработка выделенных объектов:
      1. Получение позиций центра рамок объектов и их цвет
      2. Запись текущих позиций в массив данных
      3. Если объект отслеживания был изменён (выбран другой) то идёт очистка массива данных с позициями объектов во времени
   6. За каждую запись в массиве позиций объектов отрисовка пути (трека) именно выбранного объекта в цвет рамки данного объекта
   7. Отправка обработанного кадра с рамками объектов, надписями и путём (треком) в главный поток, отрисовка его в главном окне программы
   8. Если установлен флаг завершения обработки, завершается обработка и останавливается поток QThread обработки изображения

# Разработка программы

Среда разработки: Microsoft VS Code.

Задание выполнялось на языке программирования Python.

Операционная система: Windows 10 x64 Home.

Для выполнения задачи потребовалось подключение библиотек OpenCV2, PyQt5, numpy, tensorflow, pillow.

OpenCV2 – вспомогательная библиотека компьютерного зрения, предназначенная расширить возможности программного обеспечения по взаимодействию с изображениями и видео в огромном количестве форматов и кодировок.

PyQt5 – библиотека языка Python, использующаяся для создания графического пользовательского интерфейса к программному обеспечению, написанном на языке программирования Python.

Numpy – модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций.

TensorFlow - программная библиотека для машинного обучения, предназначенная для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

pillow – вспомогательная библиотека языка Python, предназначенная для работы с растровой графикой;

Разработка модуля:

**Инициализация графического интерфейса**

Назначение: взаимодействие программы с пользователем;

Выходные данные: окно Windows системы с различными кнопками взаимодействия.

class Placeholder(QLabel):

def \_\_init\_\_(self, parent):

super(Placeholder,self).\_\_init\_\_(parent)

self.i = 0

self.timer=QTimer()

self.timer.setInterval(400)

self.timer.timeout.connect(self.animateText)

def stopTimer(self):

QtTest.QTest.qWait(40000)

self.timer.stop()

def animateText(self):

self.i += 1

self.i = self.i%4

self.setText("Loading" + ("."\*self.i))

class App(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title = 'PyQt5 Video'

self.left = 100

self.top = 100

self.width = 640

self.height = 500

self.initUI()

def notFoundText(self, msg):

if msg == 0:

self.labelNotFound.setText("")

else:

self.labelNotFound.setText("Selected object not found")

@pyqtSlot(QImage)

def setImage(self, image):

self.label.setPixmap(QPixmap.fromImage(image))

def initUI(self):

self.setWindowTitle(self.title)

self.setGeometry(self.left, self.top, self.width, self.height)

self.resize(850, 550)

self.labelExit = QLabel(self)

self.labelExit.setText("Press to exit")

self.labelExit.setGeometry(30,420,120,16)

self.buttonExit = QPushButton(self)

self.buttonExit.clicked.connect(self.close)

self.buttonExit.setGeometry(30,440,140,50)

self.buttonExit.setText("Exit")

self.buttonExit.show()

self.labelSelect = QLabel(self)

self.labelSelect.setText("Select item to track")

self.labelSelect.setGeometry(30,320,120,16)

self.selection = QComboBox(self)

self.selection.setGeometry(30,340,140,50)

for item in self.listOptions():

self.selection.addItem(item)

self.selection.activated[str].connect(self.onChanged)

self.labelCamera = QLabel(self)

self.labelCamera.setText("Press to load camera feed")

self.labelCamera.setGeometry(30,220,140,16)

self.buttonCamera = QPushButton(self)

self.buttonCamera.clicked.connect(self.openCamera)

self.buttonCamera.setGeometry(30,240,140,50)

self.buttonCamera.setText("Webcam")

self.buttonCamera.show()

self.labelFile = QLabel(self)

self.labelFile.setText("Press to open a video file")

self.labelFile.setGeometry(30,120,120,16)

self.buttonFile = QPushButton(self)

self.buttonFile.clicked.connect(self.openFile)

self.buttonFile.setGeometry(30,140,140,50)

self.buttonFile.setText("Video file")

self.buttonFile.show()

self.labelStop = QLabel(self)

self.labelStop.setText("Press to stop everything")

self.labelStop.setGeometry(30,10,140,16)

self.buttonStop = QPushButton(self)

self.buttonStop.clicked.connect(self.stopAll)

self.buttonStop.setGeometry(30,30,140,50)

self.buttonStop.setText("Stop")

self.buttonStop.show()

self.placeholder = Placeholder(self)

self.placeholder.setText("Waiting for command")

self.placeholder.setGeometry(191,11,638,478)

self.placeholder.setStyleSheet('''QWidget{text-align: center; font-size: 30px; border: 1px solid black;}''')

self.placeholder.show()

self.label = QLabel(self)

self.label.setStyleSheet('''QWidget{border: 1px solid black;}''')

self.label.move(190, 10)

self.label.resize(640, 480)

self.show()

def onChanged(self, text):

global TRACKED\_NAME

TRACKED\_NAME = text

def listOptions(self):

li = []

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

for a in category\_index:

li.append(category\_index.get(a).get('name'))

return li

app = QApplication(sys.argv)

application = App()

app.exec\_()

**Отслеживание выбранного объекта из видеофайла**

Назначение: трекинг какого-либо объекта, отрисовка его пути на экране компьютера;

Входные данные: название файла с видеоизображением;

Выходные данные: путь выбранного объекта во времени, отрисованный на экране компьютера.

MODELS\_DIR = os.path.join(os.path.join(os.getcwd(), 'data'), 'models')

MODEL\_NAME = 'efficientdet'

PATH\_TO\_CKPT = os.path.join(MODELS\_DIR, os.path.join(MODEL\_NAME, 'checkpoint/'))

PATH\_TO\_CFG = os.path.join(MODELS\_DIR, os.path.join(MODEL\_NAME, 'pipeline.config'))

PATH\_TO\_LABELS = os.path.join(MODELS\_DIR, os.path.join(MODEL\_NAME, 'label\_map.pbtxt'))

TRACKED\_NAME = 'person'

LAST\_TRACKED\_NAME = 'person'

class Thread(QThread):

changePixmap = pyqtSignal(QImage)

loaded = pyqtSignal()

source = 0

stopvar = 0

def stop(self):

self.stopvar = 1

def run(self):

gpus = tf.config.experimental.list\_physical\_devices('GPU')

for gpu in gpus:

tf.config.experimental.set\_memory\_growth(gpu, True)

configs = config\_util.get\_configs\_from\_pipeline\_file(PATH\_TO\_CFG)

model\_config = configs['model']

detection\_model = model\_builder.build(model\_config=model\_config, is\_training=False)

ckpt = tf.compat.v2.train.Checkpoint(model=detection\_model)

ckpt.restore(os.path.join(PATH\_TO\_CKPT, 'ckpt-0')).expect\_partial()

@tf.function

def detect\_fn(image):

image, shapes = detection\_model.preprocess(image)

prediction\_dict = detection\_model.predict(image, shapes)

detections = detection\_model.postprocess(prediction\_dict, shapes)

return detections, prediction\_dict, tf.reshape(shapes, [-1])

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

cap = cv2.VideoCapture(0)

detect\_history = []

class Overlay():

lines = []

def \_\_init\_\_(self, x1, y1, x2, y2, color):

self.x1 = x1

self.y1 = y1

self.x2 = x2

self.y2 = y2

self.color = color

def addLine(self):

Overlay.lines.append([self.x1, self.y1, self.x2, self.y2, self.color])

def saveDetect(name, curX, curY, color):

global TRACKED\_NAME

global LAST\_TRACKED\_NAME

if TRACKED\_NAME != LAST\_TRACKED\_NAME:

Overlay.lines.clear()

LAST\_TRACKED\_NAME = TRACKED\_NAME

return

if name == LAST\_TRACKED\_NAME:

if len(detect\_history) == 0:

detect\_history.append([name, curX, curY, color])

else:

buf = 0

for i in range(len(detect\_history)):

if detect\_history[i][0] == name:

prevX = detect\_history[i][1]

prevY = detect\_history[i][2]

color = detect\_history[i][3]

Overlay.addLine(Overlay(prevY, prevX, curY, curX, color))

detect\_history[i][1] = curX

detect\_history[i][2] = curY

buf = i

if buf == (len(detect\_history)-1):

detect\_history.append([name, curX, curY, color])

cap = cv2.VideoCapture(self.source)

self.loaded.emit()

while True:

ret, image\_np = cap.read()

image\_np = cv2.resize(image\_np, (640,480))

input\_tensor = tf.convert\_to\_tensor(value=np.expand\_dims(image\_np, 0), dtype=tf.float32)

detections, predictions\_dict, shapes = detect\_fn(input\_tensor)

label\_id\_offset = 1

image\_np\_with\_detections = image\_np.copy()

viz\_utils.visualize\_boxes\_and\_labels\_on\_image\_array(

image\_np\_with\_detections, detections['detection\_boxes'][0].numpy(), (detections['detection\_classes'][0].numpy() + label\_id\_offset).astype(int),

detections['detection\_scores'][0].numpy(), category\_index, use\_normalized\_coordinates=True,

max\_boxes\_to\_draw=10, min\_score\_thresh=.6, agnostic\_mode=False)

for i in range(6):

if (detections['detection\_scores'][0][i].numpy() > 0.6):

#print("name: ", category\_index.get((detections['detection\_classes'][0][i].numpy() + label\_id\_offset).astype(int)).get('name'))

yCur = (detections['detection\_boxes'][0][i][0].numpy() + detections['detection\_boxes'][0][i][2].numpy()) / 2

xCur = (detections['detection\_boxes'][0][i][1].numpy() + detections['detection\_boxes'][0][i][3].numpy()) / 2

box\_color = ImageColor.getrgb(STANDARD\_COLORS[((detections['detection\_classes'][0].numpy() + label\_id\_offset).astype(int)[i])% len(STANDARD\_COLORS)])

box\_title = category\_index.get((detections['detection\_classes'][0][i].numpy() + label\_id\_offset).astype(int)).get('name')

saveDetect(box\_title, round(yCur\*480), round(xCur\*640), box\_color)

for line in Overlay.lines:

cv2.line(image\_np\_with\_detections, (line[0],line[1]), (line[2],line[3]), line[4], 3)

cv2.putText(image\_np\_with\_detections , 'KAMISARAU, PO-7, 2022', (20,460), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255,255,255), 1, 2)

if cv2.waitKey(25) & 0xFF == ord('q'):

break

if self.stopvar == 1:

break

if ret:

# https://stackoverflow.com/a/55468544/6622587

rgbImage = cv2.cvtColor(image\_np\_with\_detections, cv2.COLOR\_BGR2RGB)

h, w, ch = rgbImage.shape

bytesPerLine = ch \* w

convertToQtFormat = QImage(rgbImage.data, w, h, bytesPerLine, QImage.Format\_RGB888)

p = convertToQtFormat.scaled(640, 480, Qt.KeepAspectRatio)

self.changePixmap.emit(p)

cap.release()

print("finished thread")

self.stopvar = 0

...

class App(QWidget):

...

@pyqtSlot(QImage)

def setImage(self, image):

self.label.setPixmap(QPixmap.fromImage(image))

def \_\_init\_\_():

...

self.thF = Thread(self)

self.thF.loaded.connect(self.placeholder.stopTimer)

self.thF.changePixmap.connect(self.setImage)

def openFileNameDialog(self):

options = QFileDialog.Options()

options |= QFileDialog.DontUseNativeDialog

fileName, \_ = QFileDialog.getOpenFileName(self,"QFileDialog.getOpenFileName()", "","Video files (\*.mp4)", options=options)

if fileName:

return fileName

def openFile(self):

file = self.openFileNameDialog()

self.thF.source = file

self.label.show()

self.placeholder.timer.start()

self.thC.stop()

self.thF.stopvar = 0

self.thF.start()

def stopAll(self):

self.placeholder.setText("Waiting for command")

self.label.hide()

...

self.thF.stop()

def onChanged(self, text):

global TRACKED\_NAME

TRACKED\_NAME = text

def listOptions(self):

li = []

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

for a in category\_index:

li.append(category\_index.get(a).get('name'))

return li

**Отслеживание выбранного объекта из в реальном времени через веб-камеру**

Назначение: трекинг какого-либо объекта, отрисовка его пути на экране компьютера;

Входные данные: очередной видеокадр взятый в данный момент с веб-камеры;

Выходные данные: путь выбранного объекта во времени, отрисованный на экране компьютера.

MODELS\_DIR = ...

...

LAST\_TRACKED\_NAME = ...

class Thread(QThread):

changePixmap = pyqtSignal(QImage)

...

def run(self):

...

cap = cv2.VideoCapture(self.source)

...

class App(QWidget):

...

@pyqtSlot(QImage)

def setImage(self, image):

self.label.setPixmap(QPixmap.fromImage(image))

def initUI(self):

...

self.thC = Thread(self)

self.thC.source = 0

self.thC.loaded.connect(self.placeholder.stopTimer)

self.thC.changePixmap.connect(self.setImage)

...

def openCamera(self):

self.label.show()

self.placeholder.timer.start()

self.thF.stop()

self.thC.stopvar = 0

self.thC.start()

def onChanged(self, text):

global TRACKED\_NAME

TRACKED\_NAME = text

def listOptions(self):

li = []

category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)

for a in category\_index:

li.append(category\_index.get(a).get('name'))

return li

# 4. Тестирование

## 4.1. Подготовка к тестированию

Минимальные требования для Microsoft VS Code: Процессор x64 или x32 структуры с минимальной тактовой частотой 1.6 гигагерц или быстрее, а также установленное ОЗУ с объёмом памяти не менее 1 гигабайта. VS Code не имеет практически ничего общего с похожим продуктом от всё той же компании Microsoft - Visual Studio. Редактор очень легко установить на любую платформу: на официальном сайте есть подробные инструкции для Windows, Mac и Linux.

Изначально в VS Code отсутствует поддержка запуска приложений написанных на языке программирования Python, однако существует возможность установить расширение с поддержкой приложений Python, оно даёт:

* Поддержка Python 3.4 и выше, а также Python 2.7
* Автоматическое дополнение кода с помощью IntelliSense
* Линтинг
* Отладка
* Сниппеты
* Модульное тестирование
* Автоматическое использование conda и виртуальных сред
* Редактирование кода в средах Jupyter и Jupyter Notebooks

Тестирование программы осуществлялось во встроенном терминале редактора кода Microsoft VS Code на 64-битной операционной системе Windows 10 Домашняя версии 21H2. Аппаратное обеспечение устройства, на котором происходило тестирование:

* Процессор: Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz
* Оперативная память: 8,00 ГБ (доступно: 7,78 ГБ)
* Тип системы: 64-разрядная операционная система Windows 10 Домашняя версии 21H2, процессор x64

## Тестирование

Программа запускается путём запуска через интерпретатор исходного кода, находящегося в файле main.py. При запуске данного файла перед пользователем открывается консоль с дополнительной информацией, но самое главное – открывается окно графического интерфейса, через который пользователь может и будет взаимодействовать с программой, путём нажатия различных кнопок: выход из программы, закрыть или свернуть окно, выбрать отслеживаемый объект, создать видеопоток с веб-камеры, создать видеопоток с содержимым видеофайла, либо остановить воспроизведение всех видеопотоков.

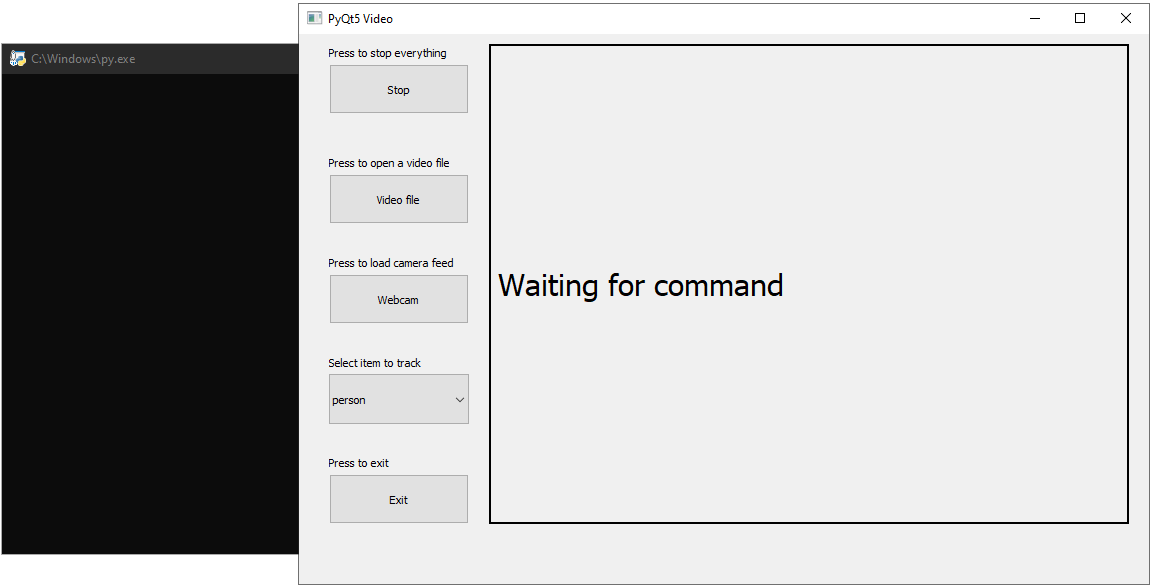


Рисунок 4.2.1 – Окно программы с консолью на заднем плане, сверху вниз кнопки: остановить все видеопотоки, открыть файл, запустить веб-камеру, выбор объекта для отслеживания, выход из программы.

При нажатии на кнопку выбора файла открывается диалоговое окно выбора файла, с которого будем отслеживать объекты.

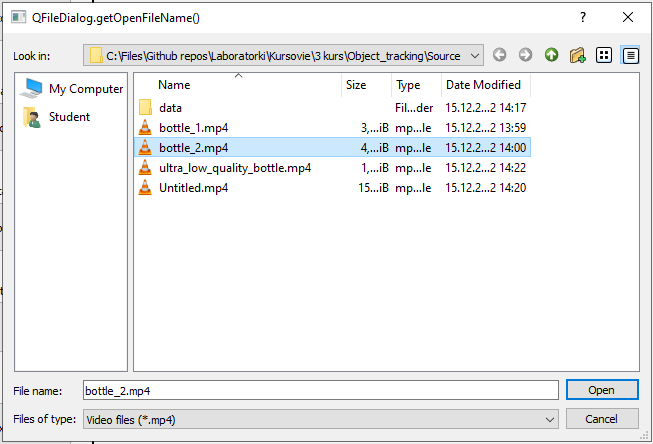


Рисунок 4.2.2 – Диалоговое окно для выбора видеофайла.

После выбора файла на окно с временным текстом “Waiting for command” выводится видео, в котором отслеживаются объекты, а также записывается траектория выбранного объекта в пространстве.

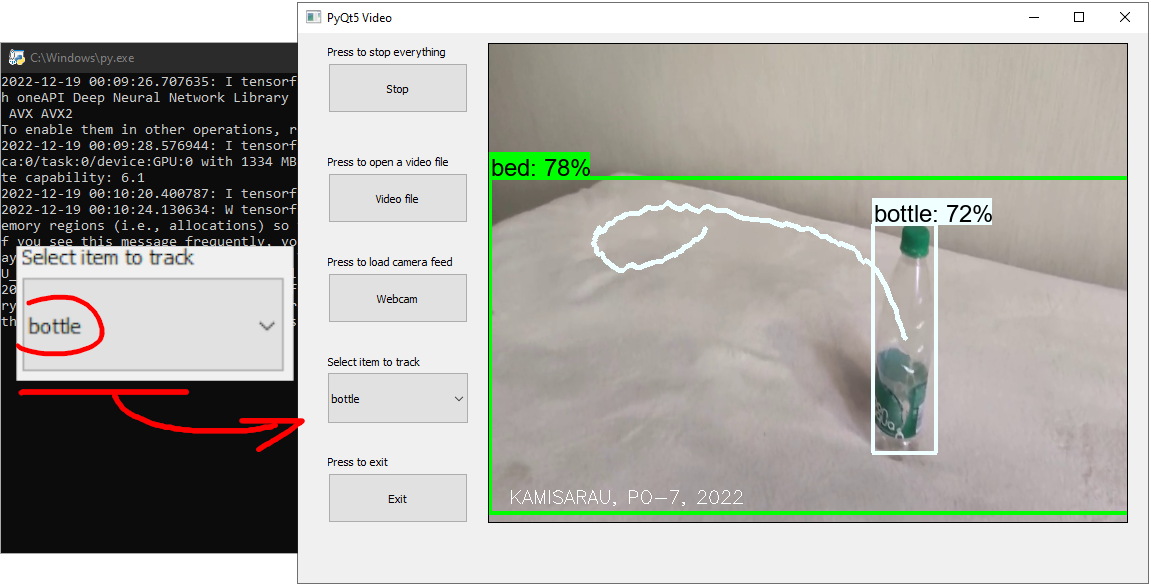


Рисунок 4.2.3 – В окне показан кадр изображения, отрисованный путь выбранного на нём объекта (белая полоса), процент уверенности нейронной сети и рамка объекта. В данный момент выбранный объект – бутылка (bottle).

Выбор отслеживаемого объекта происходит путём нажатия на элемент выпадающего списка подписанного “Select item to track”. Элементы в данный список заносятся путём чтения файла со всеми подписями предметов, обнаружению которых обучали нейронную сеть.

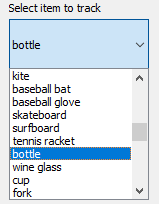


Рисунок 4.2.4 – Выпадающий список с выбором объекта отслеживания.

При остановке видеопотока, собственно видеопоток останавливается и видео перестаёт воспроизводиться и обрабатываться.

После нажатия на кнопку открытия видео с веб-камеры сразу происходит воспроизведение и обработка видеопотока с веб-камеры компьютера, никаких дополнительных окон подтверждения не открывается.

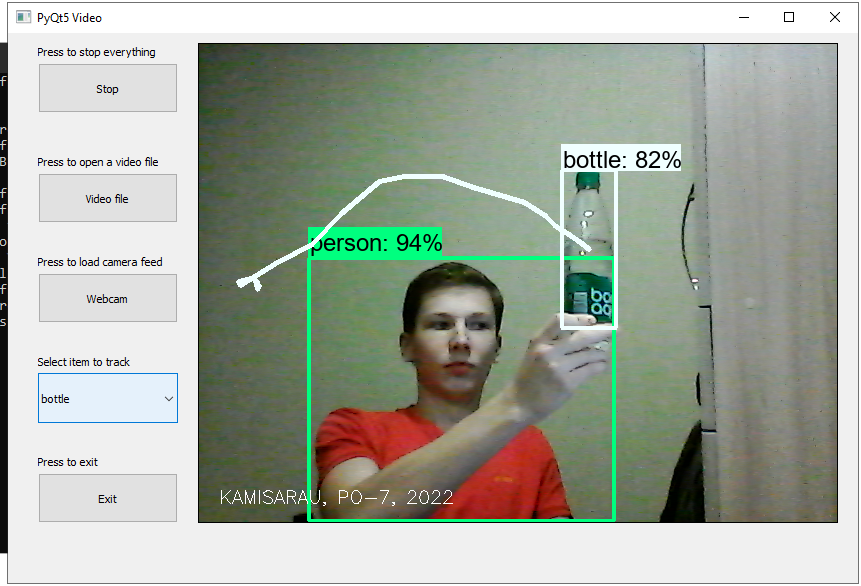


Рисунок 4.2.5 – Отслеживание объекта в реальном времени через веб-камеру компьютера. Выбранный объект – бутылка (bottle).

При потере выбранного объекта, по истечении некоторого времени программа сообщает пользователю внизу окна красным цветом “Selected object not found”, что в переводе значит “Выбранный объект не обнаружен”.

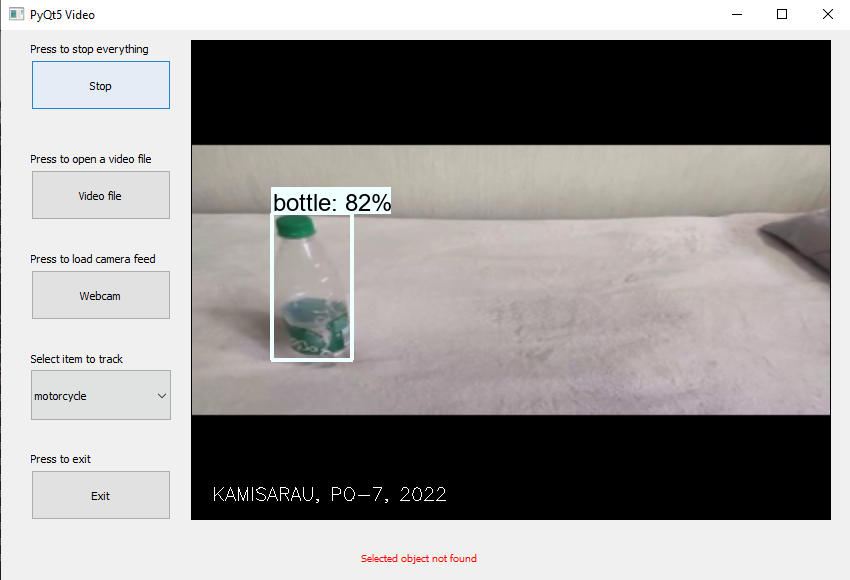


Рисунок 4.2.6 – “Selected object not found”

Одновременно программа может определять на экране до 6 объектов, отслеживать – только выбранный.

Успешность определения и отслеживания объектов на видео достаточно высокая, программа умеет определять в общей сложности 90 видов объектов, в т.ч. людей, различных животных и автомобили.

Тестирование программного обеспечения разработанного в ходе данной курсовой работы, при вышеуказанном аппаратном обеспечении, в среднем показало скорость обработки видео через веб-камеру – 3.16 кадра в секунду, через видеофайл – 1.38 кадра в секунду. Такое различие в показателях производительности может быть обусловлено дополнительной нагрузкой на запоминающее устройство (SSD-хранилище) компьютера и более высокие размеры кадра при открытии видеофайла.

В программе была использована предобученная на датасете COCO 2017 нейронная сеть EfficientDet D0 512x512 из набора моделей TF2 Detection Model Zoo.

# Заключение

Сейчас большую роль в развитии ПО в области общественной безопасности играют технологии computer vision. Основными задачами такого ПО являются распознавание и слежение за объектами, последняя из которых представляется более сложной, т.к. условия съемки часто бывают неблагоприятными.

В данном курсовом проекте мною были получены базовые теоретические и практические знания применения алгоритмов детекции и отслеживания местоположения различных объектов в видеокадре, применены знания по детеционным нейронным сетям.

В результате выполнения курсового проекта были выполнены все необходимые условия, проведено тестирование и замер производительности разработанного программного продукта.

# Список использованной литературы

1. ГОСТ 19.103-77. ЕСПД. Обозначения программ и программных документов.
2. ГОСТ 19.504-79. Единая система программной документации ЕСПД. Руководство программиста. Требования к содержанию и оформлению.
3. ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению.
4. ГОСТ 19.101-77. ЕСПД. Виды программ и программных документов.
5. ГОСТ 19.102-77. ЕСПД. Стадии разработки.
6. ГОСТ 2.105-95. Единая система конструкторской документации (ЕСКД). Общие требования к текстовым документам.
7. ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.
8. ГОСТ 19.402-78. ЕСПД. Описание программы.
9. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.
10. ГОСТ 19.005-85. ЕСПД. Р-схемы алгоритмов и программ. Обозначения условные графические и правила выполнения.