Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский политехнический университет»

**факультет информационных технологий**

**Кафедра СМАРТ-технологий**

Дисциплина: Нейронные сети в задачах технического зрения и управления

Отчёт по лабораторной работе №4

«Подготовка изображений с использованием языка Python»

Работу выполнил\_и

студент\_ка 3 курса

очного отделения

<ФИО>

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<ФИО>

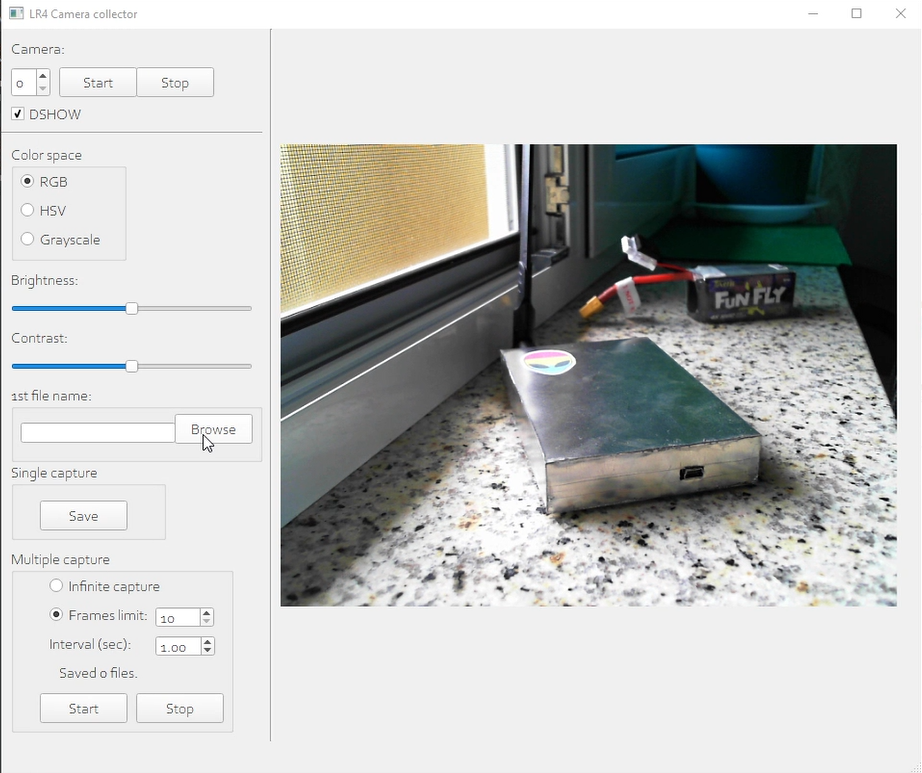
**Цель работы**

Разработать приложение для загрузки изображений и предварительной обработки с использованием языка Python и библиотеки OpenCV.

**Задачи**

* Подготовить приложение для генерации набора изображения для построения обучающей выборки сохраняющей данные с веб-камеры;
* Разработать интерфейс для просмотра изображений перед сохранением и применения базовых преобразований (бинаризация, изменение палитры, смена цветового пространства, обрезка, масштабирование);
* Разработать функциональность, обеспечивающую сохранение серии изображений (интервальная фотография) с использованием заданных фильтров (обрезка, яркость, преобразование палитры) с последовательными именами по шаблону.

**Ход работы**



Для использования приложения необходимо выбрать индекс камеры. Имеется возможность установки флага DSHOW для прямого отображения картинки с камеры (требуется для корректной работы многих камер)

**if** self.check\_dshow.isChecked():  
 self.cv\_cap = cv2.VideoCapture(self.camera\_id.value(), cv2.CAP\_DSHOW)  
**else**:  
 self.cv\_cap = cv2.VideoCapture(self.camera\_id.value())

После нажатия кнопки “Старт” запускается фоновый процесс чтения изображения с камеры, обработки и вывода в поле предпросомтра.

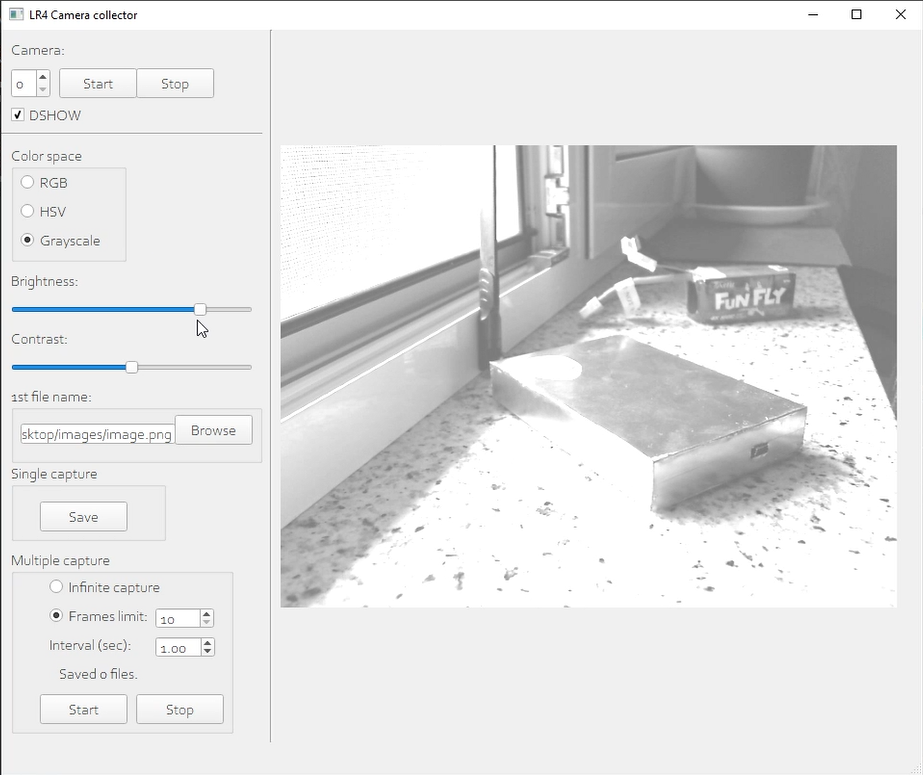
Имеется возможность изменять цветовое пространство (RGB, HSV, Ч/Б)

**if** self.radio\_color\_hsv.isChecked():  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
**if** self.radio\_color\_grayscale.isChecked():  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

Регулировка яркости и контрастности производится по формуле:

Новый\_пиксель = контраст \* старый\_пиксель + яркость

self.current\_frame = (self.slider\_contrast.value() / 50) \* self.current\_frame \  
 + ((self.slider\_brightness.value() - 50) \* 4)  
self.current\_frame = np.clip(self.current\_frame, 0, 255)



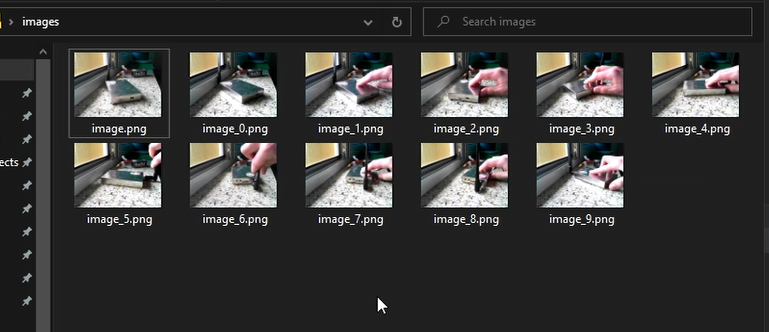
Сохранение в файл производится с помощью функции:

cv2.imwrite(filename, self.current\_frame)

Имеется возможность выбрать имя и расширение файла для сохранения.

Также существует алгоритм циклического сохранения через заданный интервал времени. Сохранения прекращаются или по счётчику или по нажатию кнопки.

Имена файлов автоматически инкрементируются



**Вывод**

В ходе данной работы было создано приложение на языке Python с использованием библиотеки PyQT5, способное создавать выборку изображений из кадров, получаемых с камеры.

**Исходный код**

**import** os  
**import** threading  
**import** time  
  
**import** numpy **as** np  
**import** cv2  
**import** sys  
**from** PyQt5 **import** QtWidgets  
**from** PyQt5.QtGui **import** QPixmap  
**import** qimage2ndarray  
**import** random  
**from** tkinter **import** filedialog  
**import** tkinter  
  
root = tkinter.Tk()  
root.withdraw()  
  
**import** gui\_4  
  
  
**def** valmap(value, istart, istop, ostart, ostop):  
 **return** ostart + (ostop - ostart) \* ((value - istart) / (istop - istart))  
  
  
**class** LR4(QtWidgets.QMainWindow, gui\_4.Ui\_MainWindow):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.btn\_camera\_start.clicked.connect(self.camera\_start)  
 self.btn\_camera\_stop.clicked.connect(self.camera\_stop)  
 self.btn\_browse.clicked.connect(self.browse\_file)  
 self.btn\_save\_current.clicked.connect(self.save\_current)  
 self.btn\_start\_multiple.clicked.connect(self.start\_multiple)  
 self.btn\_stop\_multiple.clicked.connect(self.stop\_multiple)  
  
 self.cv\_cap = **None** self.camera\_running = **False** self.save\_running = **False** self.current\_frame = np.zeros((480, 640, 3), np.uint8)  
  
 **def** camera\_start(self):  
 self.camera\_running = **True  
 if** self.check\_dshow.isChecked():  
 self.cv\_cap = cv2.VideoCapture(self.camera\_id.value(), cv2.CAP\_DSHOW)  
 **else**:  
 self.cv\_cap = cv2.VideoCapture(self.camera\_id.value())  
 thread = threading.Thread(target=self.cv\_thread)  
 thread.start()  
 **pass  
  
 def** camera\_stop(self):  
 self.camera\_running = **False  
 pass  
  
 def** browse\_file(self):  
 files = [(**'PNG Image'**, **'\*.png'**),  
 (**'JPG Image'**, **'\*.jpg'**)]  
 file = filedialog.asksaveasfilename(filetypes=files, defaultextension=files)  
  
 **if** file **is not None and** len(file) > 0:  
 self.line\_file.setText(file)  
  
 **def** save\_current(self):  
 **if** self.camera\_running:  
 **if** len(self.line\_file.text()) > 0:  
 cv2.imwrite(self.line\_file.text(), self.current\_frame)  
 print(**'File '** + str(self.line\_file.text()) + **' saved.'**)  
 **else**:  
 print(**'Empty filename!'**)  
 **else**:  
 print(**'Camera not started!'**)  
  
 **def** start\_multiple(self):  
 **if** self.camera\_running:  
 **if** len(self.line\_file.text()) > 0:  
 self.btn\_start\_multiple.setEnabled(**False**)  
 self.save\_running = **True** thread = threading.Thread(target=self.multiple\_saving)  
 thread.start()  
 **else**:  
 print(**'Empty filename!'**)  
 **else**:  
 print(**'Camera not started!'**)  
  
 **def** multiple\_saving(self):  
 iterations\_counter = 0  
 files\_counter = 0  
 *# radio\_infinite, radio\_limit, spin\_limit, spin\_interval* **while** self.save\_running:  
 filename\_base = self.line\_file.text()  
 filename = os.path.splitext(filename\_base)[0] + **'\_'** + str(files\_counter) + \  
 os.path.splitext(filename\_base)[1]  
  
 cv2.imwrite(filename, self.current\_frame)  
  
 self.label\_saved\_files.setText(**'Saved '** + str(files\_counter + 1) + **' files.'**)  
 print(**'File '** + filename + **' saved.'**)  
  
 **if** self.radio\_limit.isChecked():  
 self.label\_saved\_files.setText(**'Passed '** + str(iterations\_counter + 1) + **'/'** + str(self.spin\_limit.value()) + **' iterations.'**)  
 time.sleep(self.spin\_interval.value() / 2)  
 iterations\_counter += 1  
 **if** iterations\_counter == self.spin\_limit.value():  
 self.stop\_multiple()  
 print(**'Done.'**)  
  
 self.label\_saved\_files.setText(**'Saved '** + str(files\_counter + 1) + **' files.'**)  
 **if** self.radio\_limit.isChecked():  
 time.sleep(self.spin\_interval.value() / 2)  
 **else**:  
 time.sleep(self.spin\_interval.value())  
 files\_counter += 1  
  
 **def** stop\_multiple(self):  
 self.save\_running = **False** self.btn\_start\_multiple.setEnabled(**True**)  
  
 **def** cv\_thread(self):  
 **while** self.camera\_running:  
 ret, img = self.cv\_cap.read()  
  
 *# Color space* **if** self.radio\_color\_hsv.isChecked():  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
 **if** self.radio\_color\_grayscale.isChecked():  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
 *# Brightness + Contrast* self.current\_frame = img.copy().astype(**'float32'**)  
 self.current\_frame = (self.slider\_contrast.value() / 50) \* self.current\_frame \  
 + ((self.slider\_brightness.value() - 50) \* 4)  
 self.current\_frame = np.clip(self.current\_frame, 0, 255)  
 self.current\_frame = self.current\_frame.astype(**'uint8'**)  
  
 **if** self.radio\_color\_rgb.isChecked():  
 self.cvl\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.cvtColor(self.current\_frame, cv2.COLOR\_BGR2RGB))))  
 **else**:  
 self.cvl\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(self.current\_frame)))  
  
 self.cv\_cap.release()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QtWidgets.QApplication(sys.argv)  
 app.setStyle(**"fusion"**)  
 window = LR4()  
 window.show()  
 app.exec\_()