МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра 307 «Цифровые технологии и информационные системы»

Домашнее задание №1 по курсу «Мультимедиа технологии»

Тема 16: «Методы выявления прямых линий на изображениях»

Задание подготовил студент группы М3О-412Б-19:

Журавков Владислав

Максимович

_____

дата, подпись

Задание принял:

Минасян Виталий Борисович

_____

дата, подпись

Москва 2023

Оглавление

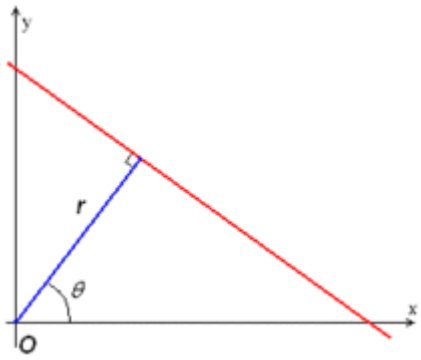
Теоретическая часть

При автоматизированном анализе цифровых изображений часто возникает подзадача обнаружения простых форм, таких как прямые линии, круги или эллипсы. Во многих случаях детектор границ может использоваться в качестве этапа предварительной обработки для получения точек изображения или пикселов изображения, которые находятся на желаемой кривой в пространстве изображения. Однако из-за несовершенства либо данных изображения, либо детектора границ, на желаемых кривых могут отсутствовать точки или пиксели, а также пространственные отклонения между идеальной линией / окружностью / эллипсом и зашумленными точками границ, полученными с помощью детектора границ. По этим причинам часто нетривиально группировать извлеченные граничные объекты в соответствующий набор линий, окружностей или эллипсов. Цель преобразования Хафа - решить эту проблему, сделав возможным группирование граничных точек в объекты-кандидаты путем выполнения явной процедуры голосования по набору параметризованных объектов изображения.

Простейшим случаем преобразования Хафа является обнаружение прямых линий. В общем случае прямая $y = mx + b$ может быть представлена в виде точки (b, m) в пространстве параметров. Однако вертикальные линии создают проблему. Они привели бы к неограниченным значениям параметра наклона m. Таким образом, по вычислительным соображениям было предложено использовать нормальную форму Гессе

$$r = x\cos(\theta) + y\sin(\theta)$$

Где r - расстояние от начала координат до ближайшей точки на прямой, а θ - угол между x осью и линией, соединяющей начало координат с этой ближайшей точкой.



Таким образом, с каждой строкой изображения можно связать пару (r, θ). (r, θ) Плоскость иногда называют пространством Хафа для множества прямых линий в двух измерениях.

Учитывая единственную точку на плоскости, множество всех прямых, проходящих через эту точку, соответствует синусоидальной кривой в плоскости (r, θ), которая уникальна для этой точки. Набор из двух или более точек, образующих прямую линию, создаст синусоиды, пересекающиеся в точке (r, θ). для этой линии. Таким образом, задача обнаружения коллинеарных точек может быть преобразована в задачу нахождения параллельных кривых.

Алгоритм построения прямых линий с помощью преобразования Хафа сводится к следующим шагам:

1. Преобразование входного изображения в градации серого.
2. Применение гауссово размытие к полутоновому изображению для уменьшения шума.
3. Определения краёв на размытом изображении с помощью детектора краев Канни.
4. Установка порогового значение для преобразования Хафа, которое определяет минимальное количество голосов, необходимое для обнаружения линии.
5. Применение преобразование Хафа к изображению краев, в результате чего образуется набор пар (r, θ), которые представляют обнаруженные линии.
6. Для каждой линии, обнаруженной на шаге 5, вычисление координаты ее конечной точки, используя значения (r, θ).
7. Отрисовка линии на исходном входном изображении, используя координаты конечной точки, полученные на шаге 6.

Обсуждение результатов

Разработано программное обеспечение (приложение с пользовательским интерфейсом), имеющее функционал преобразования Хафа.

Рисунок 1 Окно программы

Приложение разделено на 3 секции:

1. Область захвата изображения с камеры
2. Область захвата изображения из файла
3. Панель пользовательского взаимодействия

На панели пользовательского взаимодействия расположены кнопки:

1. Start Video Stream – для начала захвата изображения с камеры
2. Upload File – для начала захвата изображения из файла
3. Save Image – для сохранения изображения с отмеченными линиями (изображения будут сохранены в оригинальных масштабах)
4. Threshold – ползунок для регулировки чувствительности алгоритма (порогового значения, которое определяет минимальное количество голосов, необходимое для обнаружения линии).

При попытке выбрать файл для захвата изображения видим предупреждение:

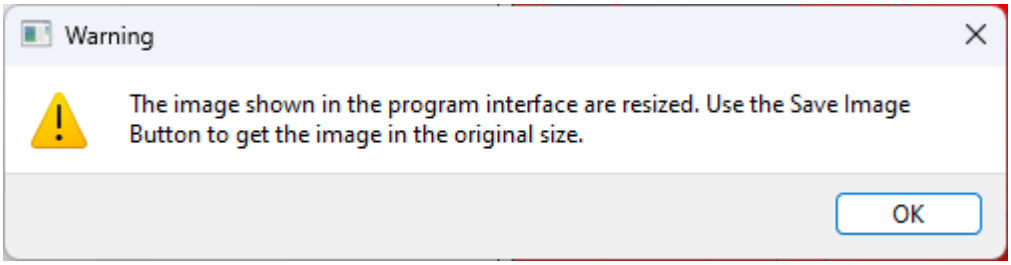


Рисунок 2 Предупреждение о масштабировании изображений в окне программы

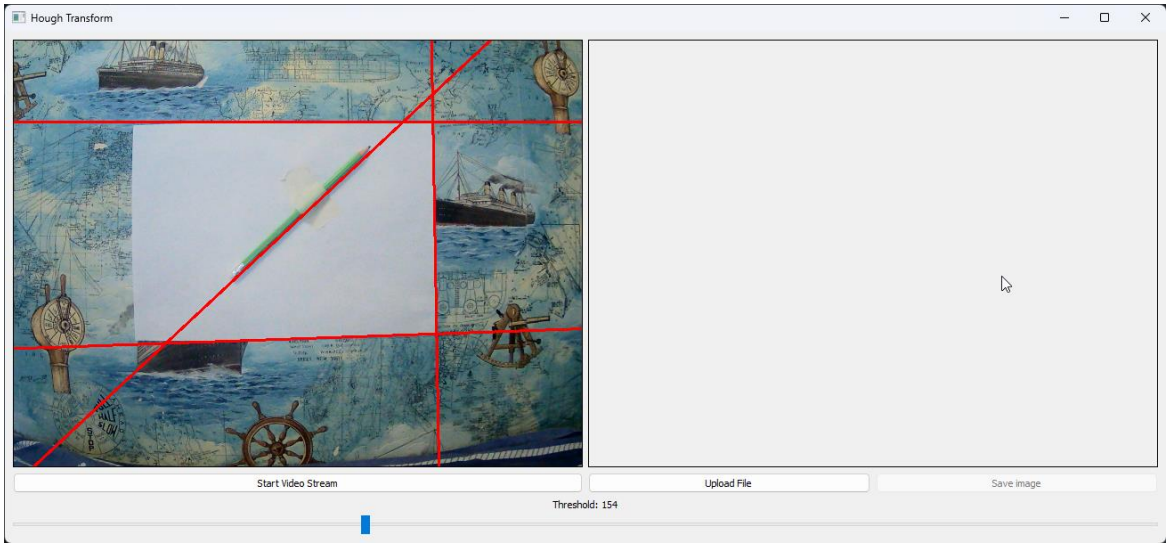


Рисунок 3 Область захвата изображения с камеры. Лист бумаги

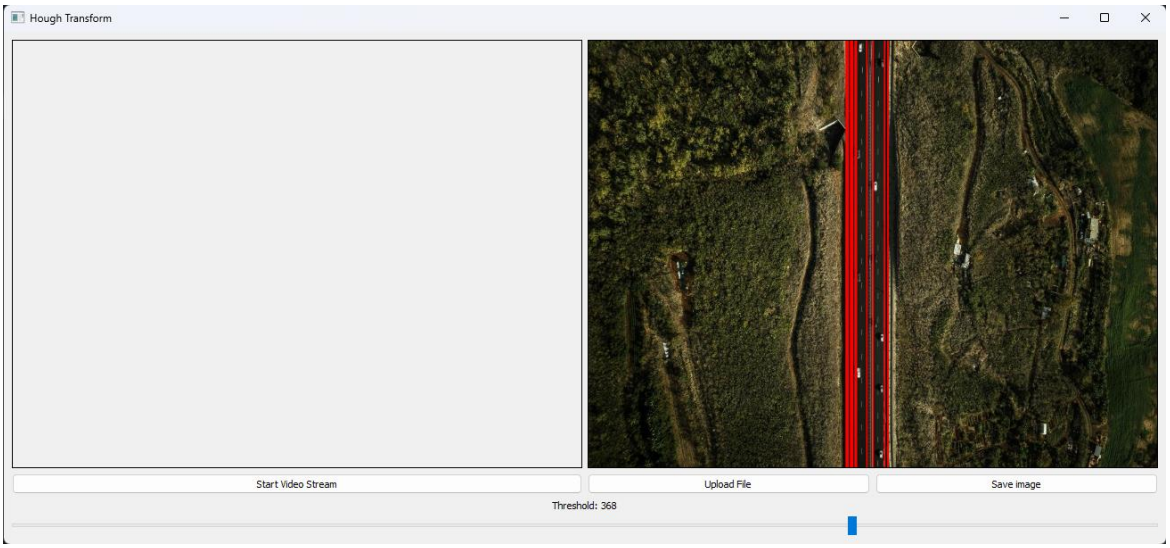


Рисунок 4 Область захвата изображения из файла. Дорога, вид сверху

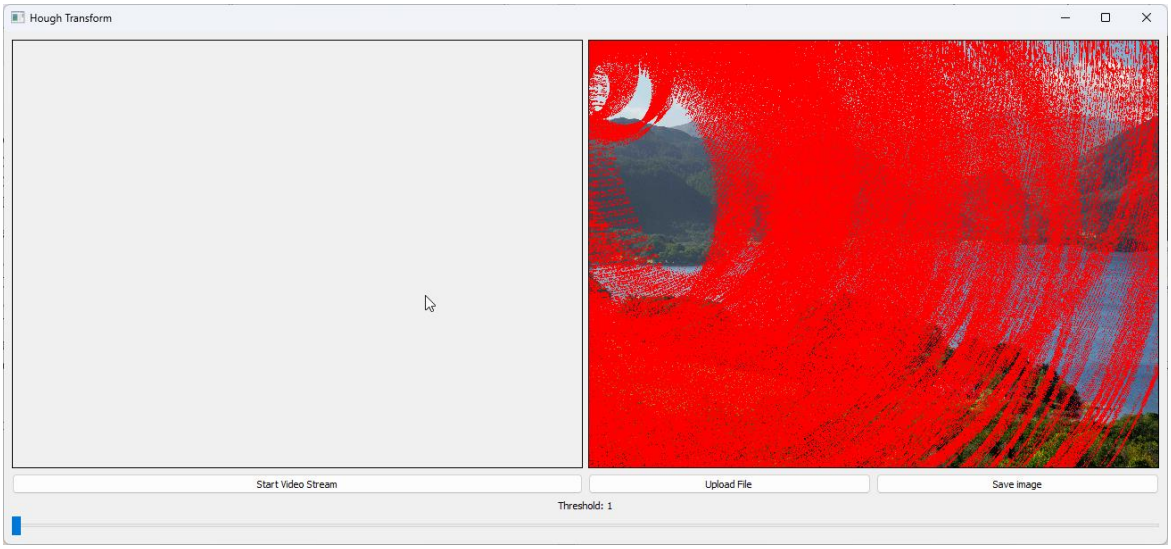


Рисунок 5 Область захвата изображения с камеры. Изображение с высоким разрешением с минимальной чувствительностью

Максимальный размер изображения ограничивается размером буфера вычислительной машины. Для используемой конфигурации этот лимит: 30.000 x 30.000 пикселей (изображение 2.5 гб).
Поддерживаемые форматы изображений: .bmp .png .tif .jpg .hdr
Максимальная глубина цвета: 24 бит на канал.

Конфигурация:
Процессор: 3,3 GHz 8-ядерный процессор Intel Core i7-1360H
Графика: GeForce RTX 3050 Laptop GPU GDDR6 4GB
Память: 16 ГБ 3200 MHz SODIMM
Накопитель: NVMe SSD накопитель SAMSUNG ёмкостью 500 ГБ

Ссылка на репозиторий: https://github.com/Shuich1/media-technology/tree/main/lines%20detection

Для корректной работы программы необходим установленный интерпретатор Python, а также понадобится установить дополнительные зависимости из файла requirements.txt:

```
pip install -r requirements.txt
```

Запуск программы осуществляется командой:

```
python main.py
```

Листинг программы

```
import os
from datetime import datetime

import cv2
import numpy as np
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import (QApplication, QFileDialog, QHBoxLayout, QLabel,
                             QMainWindow, QMessageBox, QPushButton, QSlider,
                             QVBoxLayout, QWidget)


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Hough Transform")

        screen_resolution = QApplication.desktop().screenGeometry()

        self.move(self.frameGeometry().width() // 2,
                  self.frameGeometry().height() // 2)

        self.left_label = QLabel()
        self.left_label.setAlignment(Qt.AlignCenter)
        self.left_label.setMinimumSize(
            screen_resolution.width() // 3, screen_resolution.width() // 4)
        self.left_label.setStyleSheet("border: 1px solid black;")

        self.right_label = QLabel()
        self.right_label.setAlignment(Qt.AlignCenter)
        self.right_label.setMinimumSize(
            screen_resolution.width() // 3, screen_resolution.width() // 4)
        self.right_label.setStyleSheet("border: 1px solid black;")

        self.slider = QSlider(Qt.Horizontal)
        self.slider.setMinimum(1)
        self.slider.setMaximum(500)
        self.slider.setTickInterval(5)
        self.slider.setValue(100)
        self.slider.valueChanged.connect(self.update_slider)

        self.slider_label = QLabel()
        self.slider_label.setAlignment(Qt.AlignCenter)
        self.slider_label.setText(f"Threshold: {self.slider.value()}")

        self.video_button = QPushButton("Start Video Stream")
        self.file_button = QPushButton("Upload File")
        self.save_button = QPushButton("Save image")
        self.save_button.setEnabled(False)
```

```
        self.video_button.clicked.connect(self.on_video_button)
        self.file_button.clicked.connect(self.upload_file)
        self.save_button.clicked.connect(self.save_image)
        self.is_camera_running = False

        self.central_widget = QWidget()
        self.main_layout = QVBoxLayout()
        self.layout = QHBoxLayout()
        self.left_layout = QVBoxLayout()
        self.right_layout = QVBoxLayout()
        self.right_buttons_layout = QHBoxLayout()

        self.left_layout.addWidget(self.left_label)
        self.left_layout.addWidget(self.video_button)

        self.right_layout.addWidget(self.right_label)
        self.right_layout.addLayout(self.right_buttons_layout)

        self.right_buttons_layout.addWidget(self.file_button)
        self.right_buttons_layout.addWidget(self.save_button)

        self.layout.addLayout(self.left_layout)
        self.layout.addLayout(self.right_layout)

        self.main_layout.addLayout(self.layout)
        self.main_layout.addWidget(self.slider_label)
        self.main_layout.addWidget(self.slider)

        self.central_widget.setLayout(self.main_layout)
        self.setCentralWidget(self.central_widget)

        self.camera = cv2.VideoCapture(0)
        self.image_path = None

        self.video_timer = QTimer(self)
        self.video_timer.timeout.connect(self.update_camera)

    def on_video_button(self):
        if not self.camera:
            return

        if not self.is_camera_running:
            self.is_camera_running = True
            self.video_button.setText("Stop Video Stream")
            self.video_timer.start(15)
        else:
            self.is_camera_running = False
            self.video_button.setText("Start Video Stream")
            self.video_timer.stop()
```

```
    def hough_transform(self, frame):
        try:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray, (5, 5), 0)
            edges = cv2.Canny(blur, 50, 150)
            threshold = self.slider.value()
            lines = cv2.HoughLines(edges, 1, np.pi/180, threshold)

            if lines is not None:
                for line in lines:
                    rho, theta = line[0]
                    a = np.cos(theta)
                    b = np.sin(theta)
                    x0 = a*rho
                    y0 = b*rho
                    x1 = int(x0 + 1000*(-b))
                    y1 = int(y0 + 1000*(a))
                    x2 = int(x0 - 1000*(-b))
                    y2 = int(y0 - 1000*(a))
                    cv2.line(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

            return frame

        except Exception as e:
            print("I cant process this image")
            return None

    def upload_file(self):
        self.image_path, _ = QFileDialog.getOpenFileName(
            self, "Open Image", "", "Image Files (*.png *.jpg *.jpeg)")
        if self.image_path:
            QMessageBox.warning(
                self,
                "Warning",
                "The image shown in the program interface are resized. Use the
Save Image Button to get the image in the original size."
            )
            self.update_image()
            self.save_button.setEnabled(True)

    def update_camera(self):
        try:
            _, frame = self.camera.read()
            frame = self.hough_transform(frame)
        except Exception as e:
            return

        if frame is None:
            return
```

```
        qimage = QImage(
            frame.data, frame.shape[1], frame.shape[0],
QImage.Format_RGB888).rgbSwapped()
        pixmap = QPixmap.fromImage(qimage)
        self.left_label.setPixmap(pixmap)

    def update_image(self):
        try:
            frame = cv2.imread(self.image_path)
            frame = self.hough_transform(frame)
        except Exception as e:
            QMessageBox.critical(self, "Error", "Can`t process this image, maybe
it`s too big or have incorrect file extension")
            return

        if frame is None:
            self.image_path = None
            return

        width = self.right_label.width()
        height = self.right_label.height()
        frame = cv2.resize(frame, (width, height))

        qimage = QImage(
            frame.data, frame.shape[1], frame.shape[0],
QImage.Format_RGB888).rgbSwapped()
        pixmap = QPixmap.fromImage(qimage)
        self.right_label.setPixmap(pixmap)

    def save_image(self):
        directory_path = f'{os.path.dirname(os.path.realpath(__file__))}\images'

        if not os.path.exists(directory_path):
            os.mkdir(directory_path)

        frame = cv2.imread(self.image_path)
        frame = self.hough_transform(frame)

        save_path = os.path.join(
            directory_path, f"{datetime.now().strftime('%Y-%m-%d_%H-%M-
%S')}.png")
        cv2.imwrite(save_path, frame)

    def update_slider(self):
        self.slider_label.setText(f"Threshold: {self.slider.value()}")
        if self.image_path:
            self.update_image()

    def closeEvent(self, event):
        self.video_timer.stop()
        self.camera.release()
```

```
if __name__ == "__main__":
    app = QApplication([])
    main_window = MainWindow()
    main_window.show()
    app.exec_()
```