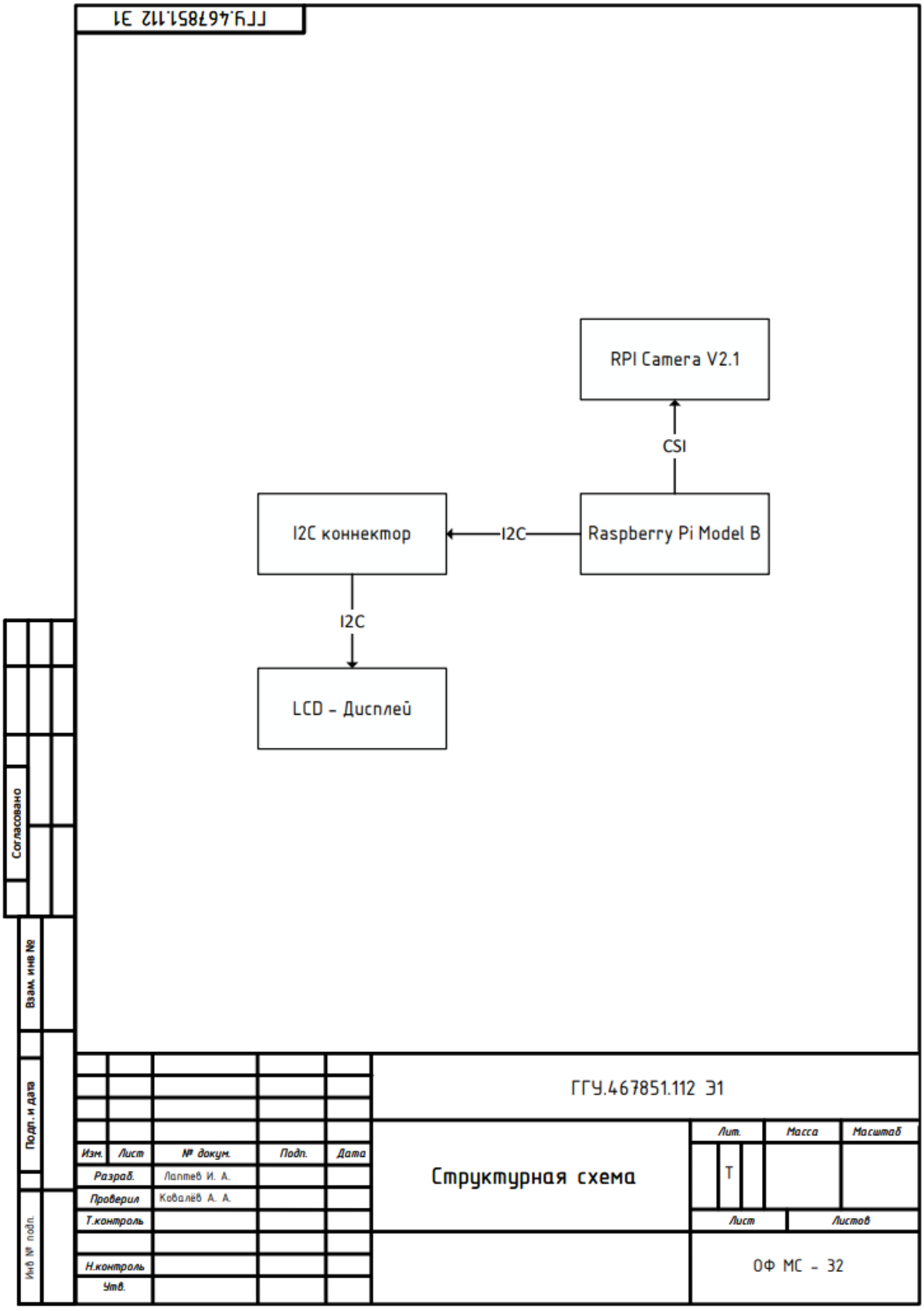
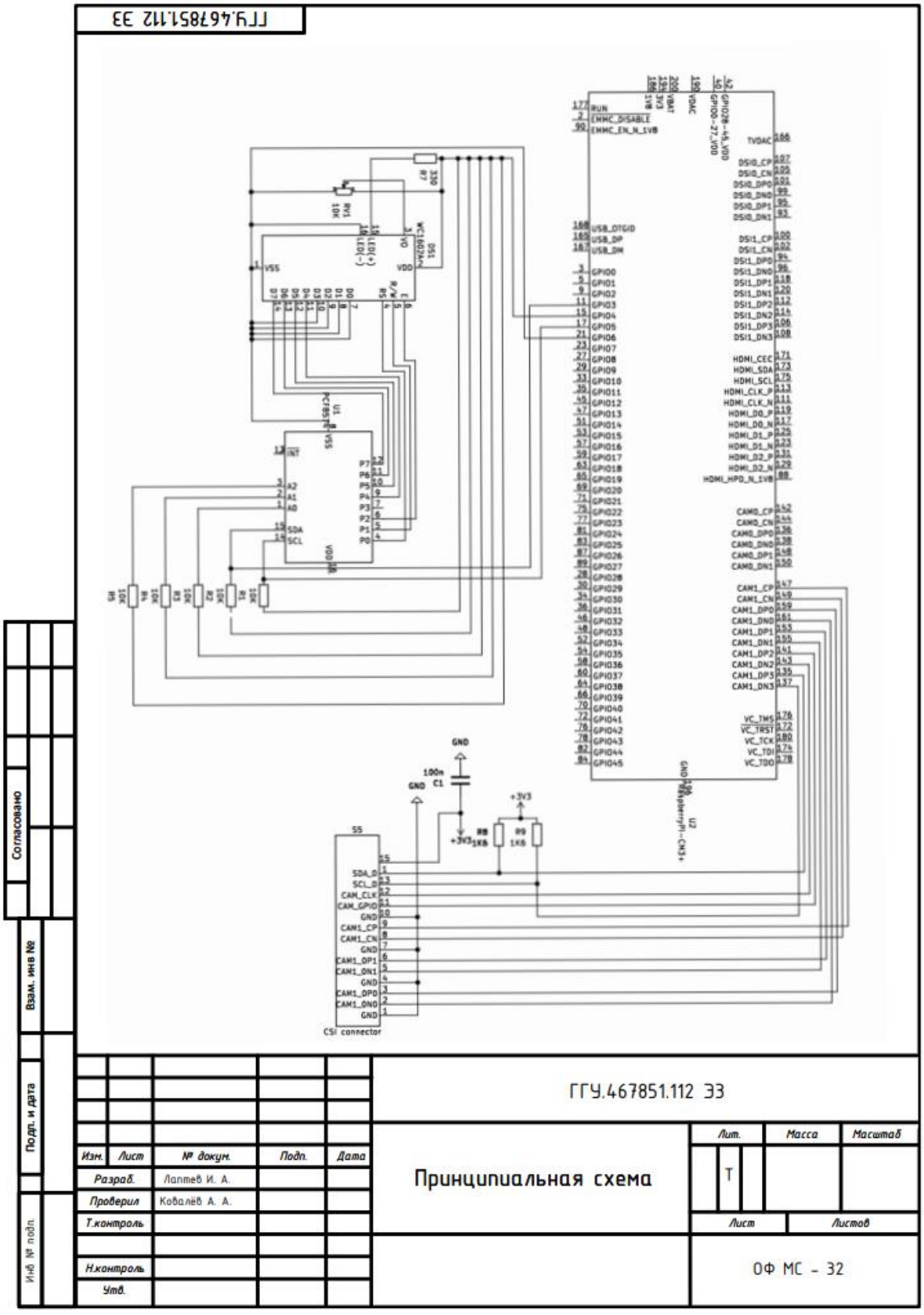
Приложение А

Структурная электрическая схема мобильного устройства распознавания объектов в видеообразе.



Приложение Б

Принципиальная электрическая схема мобильного устройства распознавания объектов в видеообразе.



Приложение В

Код алгоритма по созданию базы HOG дескрипторов.

```
from imutils import paths
import face_recognition
import pickle
import cv2
import os


# в директории Images хранятся папки со всеми изображениями
imagePaths = list(paths.list_images('Images'))
knownEncodings = []
knownNames = []
# перебираем все папки с изображениями
for (i, imagePath) in enumerate(imagePaths):
    # извлекаем имя человека из названия папки
    name = imagePath.split(os.path.sep)[-2]
    # загружаем изображение и конвертируем его из BGR (OpenCV ordering)
    # в dlib ordering (RGB)
    image = cv2.imread(imagePath)
    rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # используем библиотеку Face_recognition для обнаружения лиц
    boxes = face_recognition.face_locations(rgb, model='hog')
    # вычисляем эмбеддинги для каждого лица
    encodings = face_recognition.face_encodings(rgb, boxes)
    # loop over the encodings
    for encoding in encodings:
        knownEncodings.append(encoding)
        knownNames.append(name)
# сохраним эмбеддинги вместе с их именами в формате словаря
data = {"encodings": knownEncodings, "names": knownNames}
# для сохранения данных в файл используем метод pickle
f = open("face_enc", "wb")
f.write(pickle.dumps(data))
f.close()
```

Приложение Г

Код алгоритма для распознавания объектов в видеообразе.

```
import threading
import face_recognition
import pickle
import cv2
import os
from picamera.array import PiRGBArray
from picamera import PiCamera
from RPLCD import *
from RPLCD.i2c import CharLCD
from datetime import datetime


cascPathface = os.path.dirname(cv2.__file__) +
"/data/haarcascade_frontalface_alt2.xml"
faceCascade = cv2.CascadeClassifier(cascPathface)
data = pickle.loads(open('face_enc', "rb").read())

print("Program started")
video_capture = cv2.VideoCapture(0)
ret, frame = video_capture.read()
lcd = CharLCD('PCF8574', 0x27)
faces = []
names = []
framebuffer = ['', '',]


def write_to_lcd(lcd, framebuffer, num_cols):
    lcd.home()
    for row in framebuffer:
        lcd.write_string(row.ljust(num_cols)[:num_cols])
        lcd.write_string('\r\n')


def output_data(text):
    global framebuffer
    global lcd

    lcd.cursor_pos = (1, 0)

    if len(text)<16:
        lcd.write_string(text)
    for i in range(len(text) - 16 + 1):
        framebuffer[1] = text[i:i+16]
        write_to_lcd(lcd, framebuffer, 16)
    sleep(0.2)
    print(text)
```

```
def streaming():
    global frame
    global faces
    global names
    global rawCapture

    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(60, 60),
                                         flags=cv2.CASCADE_SCALE_IMAGE)


def recognition():
    global frame
    global faces

    while True:
        if not len(faces):
            continue
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        encodings = face_recognition.face_encodings(rgb)
        global names
        names = []
        for encoding in encodings:
            matches = face_recognition.compare_faces(data["encodings"],
encoding)
            name = "Unknown"
            if True in matches:
                matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                counts = {}
                for i in matchedIdxs:
                    name = data["names"][i]
                    counts[name] = counts.get(name, 0) + 1
                name = max(counts, key=counts.get)
            names.append(name)




def file_writer():
    global names
    enter = {(0, 0, 0, 0): ['start']}
    current_hour = datetime.now()

    while True:
        current_time = datetime.now()
        time = (current_time.day, current_time.month, current_time.hour,
current_time.minute)
        file_time = "data: " + str(current_time.day) + "." +
str(current_time.month) + " time: " + str(current_time.hour) + ":" +
str(current_time.minute)
```

```
        if len(names) > 0:
            for name in names:
                if time in enter:
                    if name not in enter[time]:
                        enter[time].append(name)
                        temp_str = file_time + " - " + name
                        output_data(temp_str)
                    else:
                        break
                else:
                    enter[time] = [name]
                    temp_str = file_time + " - " + name
                    output_data(temp_str)
        names = []

        if len(enter) > 1:
            if current_time.hour != current_hour.hour:
                file_name = str(current_hour.year) + "_" +
str(current_hour.month) + "_" + str(current_hour.day) + "_" +
str(current_hour.hour) + "_" + str(0) + ".txt"
                with open("log\\" + file_name, 'w') as f:
                    index = 1
                    for key in list(enter.keys())[1:]:
                        key_str = str(key[0]) + "." + str(key[1]) + " " +
str(key[2]) + ":" + str(key[3])
                        for j in enter[key]:
                            output_str = "№" + str(index) + " - data: " +
key_str + " Name :" + str(j) + "\n"
                            f.write(output_str)
                            index += 1
                current_hour = datetime.now()
                enter = {(0, 0, 0, 0): ['start']}


encod_thread = threading.Thread(target=recognition, daemon=True)
encod_thread.start()
write_thread = threading.Thread(target=file_writer, daemon=True)
write_thread.start()


while True:
    streaming()
```