دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پروژه‌ی درس آزمایشگاه سخت‌افزار

عنوان:

مستند نهایی پروژه‌ی شناسایی موجود زنده در شب برای اتومبیل

نگارندگان:

علیرضا شاطری، رضا امینی

استاد گرامی:

جناب آقای دکتر اجلالی - جناب آقای دکتر فصحتی

زمستان ۱۴۰۱


بسم الله الرحمن الرحیم

فهرست مطالب

فهرست تصاویر

فهرست جداول

فصل ۱

مقدمه

محصول نهایی این پروژه، یک سیستم دید در شب است که درون اتومبیل قرار می‌گیرد و به راننده در هنگام رانندگی در تاریکی، کمک به سزایی می‌کند. در این سیستم اطلاعات از طریق یک دوربین حرارتی به ماژول رزپری منتقل می‌شود و کدهایی که در رزپری قرار داده شده است با انجا پردازش تصویری ساده، تشخیص خواهد داد که آیا موجود زنده‌ای در میدان دید راننده حضور دارد یا خیر. همچنین از طریق چراغ و صدا نتیجه را به راننده اطلاع می‌دهد.

به صورت دقیق‌تر، این محصول با کمک یک دوربین مادون قرمز، می‌تواند دمای موانع در سر راه راننده را از فاصله‌ی دور تشخیص دهد. سپس اگر دمای قسمتی از تصویر روبه‌رویش نسبت به دمای محیط به مقدار نسبتا قابل ملاحظه‌ای بالاتر باشد، سیستم متوجه حضور یک موجود زنده شده و شروع به هشدار دادن به راننده می‌کند. نکته‌ای که وجود دارد این است که اگر این تغییر دما آن قدر بالا باشد که دیگر نتواند به عنوان دمای واقعی بدن یک موجود زنده در نظر گرفت، آنگاه سیستم نیز موجود زنده‌ای را شناسایی نمی‌کند زیرا تنها محدوده‌ی دمایی خاصی‌ست که می‌توان مربوط به دمای بدن موجودات زنده باشد.

مزیت رقابتی اصلی این محصول هزینه‌ی پایین ساخت آن است که با تغییر در بعضی از ماژول‌های سیستم، حتی می‌توان به هزینه‌ی کمتر نیز رسید. همچنین محصول نهایی بسیار کوچک خواهد بود زیرا به جای چراغ‌هایی که در نمونه‌ی اولیه‌ی ما استفاده شده است، در واقع باید چراغ‌های خود اتومبیل قرار بگیرد و چون روشن کردن این چراغ‌ها از طریق ارتباط با کامپیوتر ماشین ممکن است در نتیجه کافی‌ست

پس از انجام پردازش‌ها توسط رزپری و سنسورها، دستور روشن شدن چراغ‌ها را به کامپیوتر اتومبیل ارسال و آن‌ها را روشن کرد.

فصل ۲

دیتاشیت محصول

محدوده‌ی دمای قابل استفاده	$-۱۰°C - ۷۵°C$
ولتاژ ورودی	۵ ولت
جریان ورودی	۲ آمپر
ابعاد	$۱۵cm * ۱۰cm * ۵cm$
وزن	۴۰۰ گرم
محدوده‌ی دمایی تشخیص موجود زنده	$۳۰°C - ۵۰°C$

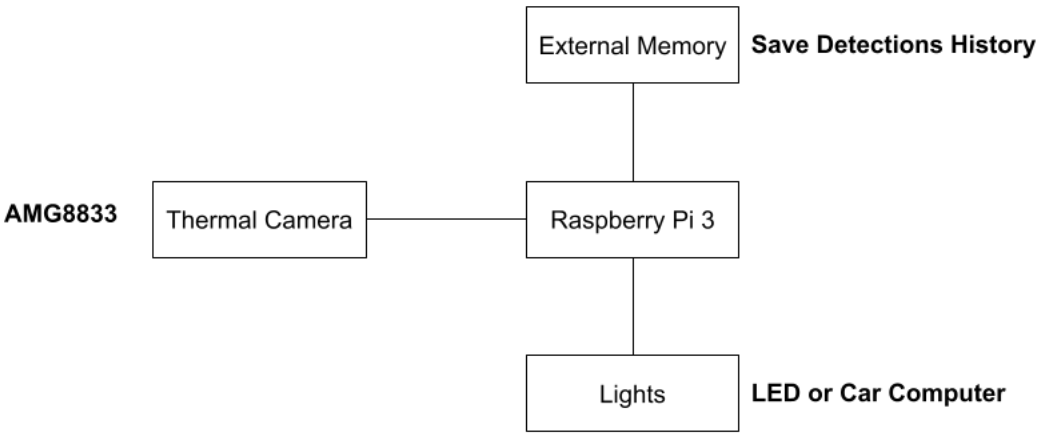
- وزن محصول بسته به شرایط و مواد مورد استفاده در تولید جعبه‌ی آن متغیر است.

فصل ۳

معماری سیستم

سیستم طراحی شده از ۲ قسمت اصلی ساخته شده است. قسمت سخت افزاری که شامل رزپری و سنسور حرارتی و مدارها می‌شود. قسمت نرم افزاری نیز که شامل پیاده‌سازی نرم افزاری است که در رزپری پای اجرا شده و توابع قسمت مختلف را مدیریت می‌کند.

معماری سطح بالای سیستم در شکل ۳-۱ قابل مشاهده است.



شکل ۳-۱: معماری سطح بالای سیستم

۳-۱ طراحی و پیاده‌سازی

اصلی‌ترین قسمت این پروژه، طراحی و پیاده‌سازی قسمت‌های سخت‌افزاری آن است. در زیر لیستی از قطعات سخت‌افزاری مورد استفاده آمده است و پس از آن توضیحاتی در مورد هر یک از سنسورها و نحوه کارکرد و راه‌اندازی آن ذکر شده است.

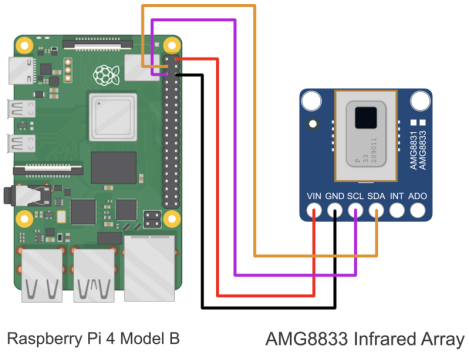
- برد Raspberry Pi 3
- سنسور دوربین حرارتی AMG8833
- چراغ‌های led

۳-۱-۱ دوربین حرارتی

یک آرایه سنسور مادون قرمز کم هزینه است که توسط پاناسونیک توسعه یافته است. برای استفاده با میکروکنترلرها در یک ماژول با شیفترهای سطح و تنظیم کننده ولتاژ یکپارچه شده است که برق و داده ۳ تا ۵ ولت را می دهد.
این سنسور تنها ۶۴ پیکسل (۸×۸) دارد که خیلی زیاد نیست اما برای آزمایش کافی و کار با آن ساده است، همچنین قیمت مناسبی نیز دارد.
ماژول را می‌توان به راحتی به برد متصل کرد و داده‌های دمایی تصویر را دریافت و پردازش نمود.

شکل ۳-۲: AMG8833

در تصویر زیر نیز می‌توانید نحوه‌ی اتصال این ماژول به رزپری را مشاهده کنید.



شکل ۳-۳: اتصال سنسور AMG8833

برای خواندن مقادیر از کتاب‌خانه‌ی smbus [۱] استفاده شده است. البته این کتابخانه مخصوص این ماژول نمی‌باشد و صرفا خواندن سریال از طریق i2c را برایمان راحت کرده است. در نتیجه کد اصلی خواندن دیتا دز دوربین حرارتی پیاده سازی شده است که در ادامه خواهید دید.

کد اصلی مربوط به این قسمت در زیر آورده شده است:

```
import smbus  # i2c bus
```

[۱] https://pypi.org/project/smbus2/

```
#
GE_I2C_ADDRESS = 0X69
RPI_BUS = 0X01
#
################################
# Base Register Addresses
################################
#
# ... You can see the original code for these initializations
#
################################
# Base Write Registers
################################
#
# ... You can see the original code for these initializations
#
####################################
# I2C Bus Initialization and
# Register Read/Write Commands
####################################
#
def get_i2c_device(address, busnum, i2c_interface=None, **kwargs):
    return i2c_driver(address, busnum, i2c_interface, **kwargs)


class i2c_driver(object):
    def __init__(self, address, busnum, i2c_interface=None):
        self._address = address
```

```
        # specify smbus for RPi (smbus 1 for RPi 2,3,4)
        self._bus = smbus.SMBus(busnum)

    def write8(self, register, value):
        # write 8-bits to specified register
        value = value & 0xFF
        self._bus.write_byte_data(self._address, register, value)

    def read16(self, register, little_endian=True):
        # read 16-bits from specified register
        result = self._bus.read_word_data(self._address, register) & 0xFF
        if not little_endian:
            result = ((result << 8) & 0xFF00) + (result >> 8)
        return result


class AMG8833(object):
    def __init__(self, addr=GE_I2C_ADDRESS, bus_num=RPI_BUS):
        self.device = get_i2c_device(addr, bus_num)

        self.set_sensor_mode(GE_PCTL_NORMAL_MODE)  # set sensor mode
        self.reset_flags(GE_RST_INITIAL_RST)  # reset at startup
        self.set_interrupt_mode(GE_INTC_OFF)  # set interrupt mode
        self.set_sample_rate(GE_FPSC_10FPS)  # set sample rate

    def set_sensor_mode(self, mode):
        self.device.write8(GE_POWER_CTL_REG, mode)  # mode
```

```
def reset_flags(self, value):
    self.device.write8(GE_RESET_REG, value)  # reset

def set_sample_rate(self, value):
    self.device.write8(GE_FPSC_REG, value)  # sample rate

def set_interrupt_mode(self, mode):
    self.device.write8(GE_INT_CTL_REG, mode)  # interrupts

def clear_status(self, value):
    self.device.write8(GE_SCLR_REG, value)  # overflows

def read_temp(self, PIXEL_NUM):
    T_arr = []  # temp array
    status = False  # status boolean for errors
    for i in range(0, PIXEL_NUM):
        raw = self.device.read16(GE_PIXEL_BASE + (i << 1))
        converted = self.twos_compl(raw) * 0.25
        if converted < -20 or converted > 100:
            return True, T_arr  # return error if outside temp window
        T_arr.append(converted)
    return status, T_arr

def read_thermistor(self):
    # read thermistor (background temp)
    raw = self.device.read16(GE_TTHL_REG)
    return self.signed_conv(raw) * 0.0625  # scaling values 0.0625
```

```
    @staticmethod
    def twos_compl(val):  # conversion for pixels
        if 0x7FF & val == val:
            return float(val)
        else:
            return float(val - 4096)

    @staticmethod
    def signed_conv(val):  # conversion for thermistor
        if 0x7FF & val == val:
            return float(val)
        else:
            return -float(0x7FF & val)
```

در نهایت تابعی که برای خواندن کل آرایه ۸ در ۸ نهایی استفاده می‌شود، تابع read_temp از کلاس تعریف شده در این کد است. این تابع آرایه‌ای شامل ۶۴ عدد integer برمی‌گرداند که هر کدام از این اعداد دمای یک پیکسل از ۶۴ پیکسل قابل دید توسط دوربین را نمایش می‌دهد. سپس این آرایه به کمک numpy به صورت یک مارتیس ۸ در ۸ در می‌آید که در ادامه خواهیم دید.

۳-۱-۲ چراغ‌های LED

با توجه به اینکه اتصال چراغ‌های LED تنها نیاز به یک ولتاژ صفر و یک ولتاژ فعال دارد، از توضیح نحوه اتصالشان صرف نظر می‌کنیم. در ادامه می‌توانید کد پیاده‌سازی شده برای روشن یا خاموش کردن چراغ‌ها را مشاهده کنید.

```
import RPi.GPIO as GPIO
from enum import Enum

GPIO.setwarnings(False)
```

```
GPIO.setmode(GPIO.BOARD)


class Pin(Enum):
    UP = 8
    DOWN = 10
    LEFT = 13
    RIGHT = 15


class PinHandler:

    def __init__(self):
        GPIO.setup(Pin.UP.value, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(Pin.DOWN.value, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(Pin.LEFT.value, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(Pin.RIGHT.value, GPIO.OUT, initial=GPIO.LOW)

    @staticmethod
    def up_on():
        GPIO.output(Pin.UP.value, GPIO.HIGH)

    @staticmethod
    def up_off():
        GPIO.output(Pin.UP.value, GPIO.LOW)

    @staticmethod
    def down_on():
```

```
        GPIO.output(Pin.DOWN.value, GPIO.HIGH)

    @staticmethod
    def down_off():
        GPIO.output(Pin.DOWN.value, GPIO.LOW)

    @staticmethod
    def left_on():
        GPIO.output(Pin.LEFT.value, GPIO.HIGH)

    @staticmethod
    def left_off():
        GPIO.output(Pin.LEFT.value, GPIO.LOW)

    @staticmethod
    def right_on():
        GPIO.output(Pin.RIGHT.value, GPIO.HIGH)

    @staticmethod
    def right_off():
        GPIO.output(Pin.RIGHT.value, GPIO.LOW)
```

همانطور که در کد می‌توان دید، ۴ تابع روشن کردن و ۴ تابع خاموش کردن چراغ داریم که هر جفت از خاموش و روشن کردن‌ها مربوط به یکی از جهات بالا، پایین، راست و چپ است.

۳-۱-۳ بخش نرم افزاری اصلی

در کنار کدهای قبلی، یک کد اصلی نیز وجود دارد که مغز اصلی سیستم است و با توجه به شرایط و به تناسب از توابع تعریف شده استفاده می‌کند. در این قسمت بخش‌های مختلف این کد را بررسی می‌کنیم.

در ابتدا کتابخانه‌های مورد نیاز را import می‌کنیم. در اینجا از کتابخانه‌ی logging برای نگه داشتن تاریخچه‌ی تشخیص‌های سیستم از موجودات زنده استفاده می‌کنیم. همانطور که می‌بینید این تاریخچه در فایلی با اسم history.log ذخیره می‌شود.فرمت لاگ خروجی را نیز می‌توانید در زیر ببینید:

2022-12-20 16:07:25,426 - LEFT - 30.0625

سپس تعداد متغیر اولیه تنظیم شده است که هرکدام استفاده خاص خود را دارند. به عنوان مثال متغیرهای MIN_TEMP و MAX_TEMP بازه‌ی دمایی موجودات زنده را برای سیستم مشخص می‌کند.

```
import time
import sys
import logging

import numpy as np

import amg8833_i2c
import led

logging.basicConfig(level=logging.INFO, filename='history.log',
                    format='%(asctime)s - %(message)s')

sys.path.append('/')


pin_handler = led.PinHandler()
```

```
LEFT = 0
RIGHT = 1
MID = 2

MIN_TEMP = 29
MAX_TEMP = 50
LOGGING_PERIOD = 1 * 60  # Seconds

last_time = None
```

برای تشخیص موجود زنده دو تابع اصلی وجود دارد. در این تابع پنجره‌ای ۲ در ۲ در نظر گرفته می‌شود و این پنجره روی کل ماتریس ۸ در ۸ حاصل از خواندن داده‌های دوربین حرارتی لغزانده می‌شود و روی هر ۴ درایه از این ماتریس که قرار گرفت، میانگین دماهای این ۴ خانه را محاسبه می‌کند. با توجه به اینکه این میانگین در بازه ی تعریف شده وجود دارد یا نه، تشخیص می‌دهد که موجود زنده جلوی سیستم وجود دارد یا خیر. این میانگین‌گیری برای جلوگیری از خطای احتمالی پیکسل‌های جداگانه است تا سیستم منعطف‌تر کار کند و با کوچیکترین تغییر دما واکنش نشان ندهد. در ادامه می‌توانید کد مربوط به این دو تابع را مشاهده کنید.

```
def generate_submatrices(matrix, sub_size=2):
    submatrices = []
    for i in range(len(matrix) - sub_size + 1):
        for j in range(len(matrix) - sub_size + 1):
            direction = LEFT

            if j == len(matrix) / 2 - 1:
                direction = MID
            elif j >= len(matrix) / 2:
```

```
                direction = RIGHT

            submatrices.append(
                (direction, matrix[i:i + sub_size, j:j + sub_size]))

    return submatrices

def decide_lights(sub_matrices):
    global last_time

    found_left = False
    found_right = False

    for direction, sub_matrix in sub_matrices:
        mean = np.mean(sub_matrix)

        if MIN_TEMP <= mean <= MAX_TEMP:
            if direction == LEFT:
                pin_handler.left_on()
                found_left = True

                if not last_time or time.time() - last_time >= LOGGING_PE
                    log("LEFT", mean, sub_matrix)

            elif direction == RIGHT:
                pin_handler.right_on()
                found_right = True
```

```
                if not last_time or time.time() - last_time >= LOGGING_PE
                    log("RIGHT", mean, sub_matrix)

            elif direction == MID:
                pin_handler.left_on()
                pin_handler.right_on()
                found_right = True
                found_left = True

                if not last_time or time.time() - last_time >= LOGGING_PE
                    log("MID", mean, sub_matrix)

    if not found_left:
        pin_handler.left_off()

    if not found_right:
        pin_handler.right_off()

    if not found_left and not found_right:
        last_time = None
```

در تابع generate_submatrices تمام ماتریس‌های ۲ در ۲ ممکن استخراج می‌شود. همچنین در همین تابع تشخیص داده می‌شود که هرکدام از ماتریس‌های ۲ در ۲ تولید شده، در کدام سمت راننده است، در چپ یا راست. این تشخیص جهت به این دلیل است که چراغ درست روشن شود. اگر موجود زنده در سمت راست بود، چراغ راست و اگر در چپ بود چراغ چپ روشن شود. در تابع decide_lights نیز بر اساس داده‌های تولید شده از تابع قبل، تصمیم گرفته می‌شود که کدام چراغ‌ها روشن و یا خاموش شوند. همچنین فرایند ثبت شناسایی‌ها در تاریخچه نیز در همین تابع انجام می‌گیرد. این کار با صدا زدن تابع log انجام می‌شود. کد مربوط به این تابع را می‌توانید در زیر ببینید.

```
def log(direction, mean, sub_matrix):
    global last_time

    last_time = time.time()
    logging.info(f"{direction} - {mean} - {sub_matrix}")
```

در نهایت یک تابع main وجود دارد که در یک حلقه‌ی بینهایت مقادیر دوربین حرارتی را خوانده و توابع تعریف شده در بالا را صدا می‌زند. این کد را می‌توانید در زیر مشاهده کنید:

```
def main():

    t0 = time.time()
    sensor = []

    while (time.time() - t0) < 1:
        try:
            sensor = amg8833_i2c.AMG8833(addr=0x69)
        except Exception as e:
            sensor = amg8833_i2c.AMG8833(addr=0x68)
        finally:
            pass

    time.sleep(0.1)

    if not sensor:
        print("No AMG8833 Found - Check Your Wiring")
        sys.exit()
```

```
    pixels_resolution = (8, 8)

    pixels_to_read = 64

    while True:
        status, pixels = sensor.read_temp(pixels_to_read)
        if status:
            continue

        T_thermistor = sensor.read_thermistor()

        pixels_reshaped = np.reshape(pixels, pixels_resolution)
        submatrices = generate_submatrices(pixels_reshaped)

        decide_lights(submatrices)
        print(pixels_reshaped)
        print("Thermistor Temperature: {0:2.2f}".format(T_thermistor))


if __name__ == '__main__':
    main()
```

در این تابع ابتدا دوربین حرارتی اتصالش برقرار شده سپس همواره ۶۴ پیکسل از آن خوانده می‌شود و به صورت یک ماتریس ۸ در ۸ تبدیل می‌شود. سپس ماتریس‌های ۲ در ۲ تولید شده از آن به تابع decide_lights داده می‌شود تا تصمیمگیری‌های مربوط به چراغ‌ها را انجام دهد. این فرایند تا زمان خرابی سیستم یا قطع آن توسط کاربر انجام خواهد شد.

۳-۱-۴ بسته بندی

این قسمت هنوز طراحی نشده است.

فصل ۴

قیمت

یکی از مسائل مهم در طراحی محصول قیمت آن است. البته با توجه به این که این محصول به صورت نمونه اولیه طراحی شده است، طبیعتا قیمت تمام شده آن از محصولی که بخواهد تولید عمده بشود بالاتر خواهد بود. در جدول ۴-۱ قیمتی تخمین زده شده و هزینه نهایی پروژه آورده شده است.

ردیف	قطعه	قیمت تخمینی	قیمت نهایی
۱	Raspberry PI 3B	۳۱۰۰	۰
۲	AMG8833	۱۱۰۰	۱۱۰۰
۳	Flash USB	۱۰۰	۱۳۰
۴	LED	۲۵	۰
۵	Board	۲۵	۰
۱۸	Total	۴۳۵۰	۱۲۳۰

جدول ۴-۱: جدول قیمت محصول (قیمت‌ها به واحد هزارتومان)

فصل ۵

جمع‌بندی

در این پروژه به پیاده‌سازی سیستم دید در شب اتومبیل پرداختیم که به راننده برای جلوگیری از سانحه در محیط‌های تاریک کمک می‌کند. در این سیستم با استفاده از سنسور دوربین حرارتی که از طریق جذب مادون قرمز عمل می‌کند و همچنین واحد پردازشی رزپری‌پای، حضور یک موجود زنده جلوی دید راننده را تشخیص و به کمک چراغ‌ها به اون هشدار دادیم.

در کنار طراحی کلی و نحوه پیاده‌سازی، تمام کدهای مربوط به این محصول نیز به صورت متن باز در گیتهاب پروژه قراره گرفته است و هرکسی می‌تواند با کمک این کدها به بهبود و ارتقاء این سیستم کمک کند و یا با الهام از آن، پیاده سازی خاص خودش را ارائه دهد.

آنچه محصول ما را از دیگر محصول‌ها متمایز می‌کند قیمت بسیار پایین تمام‌شده‌ی آن است که می‌توان حتی بیشتر آن را کاهش داد. به عنوان مثال با استفاده از آردوینو به جای رزپری و همچنین استفاده از چراغ‌های خودرو به جای چراغ‌های مجزا برای سیستم.