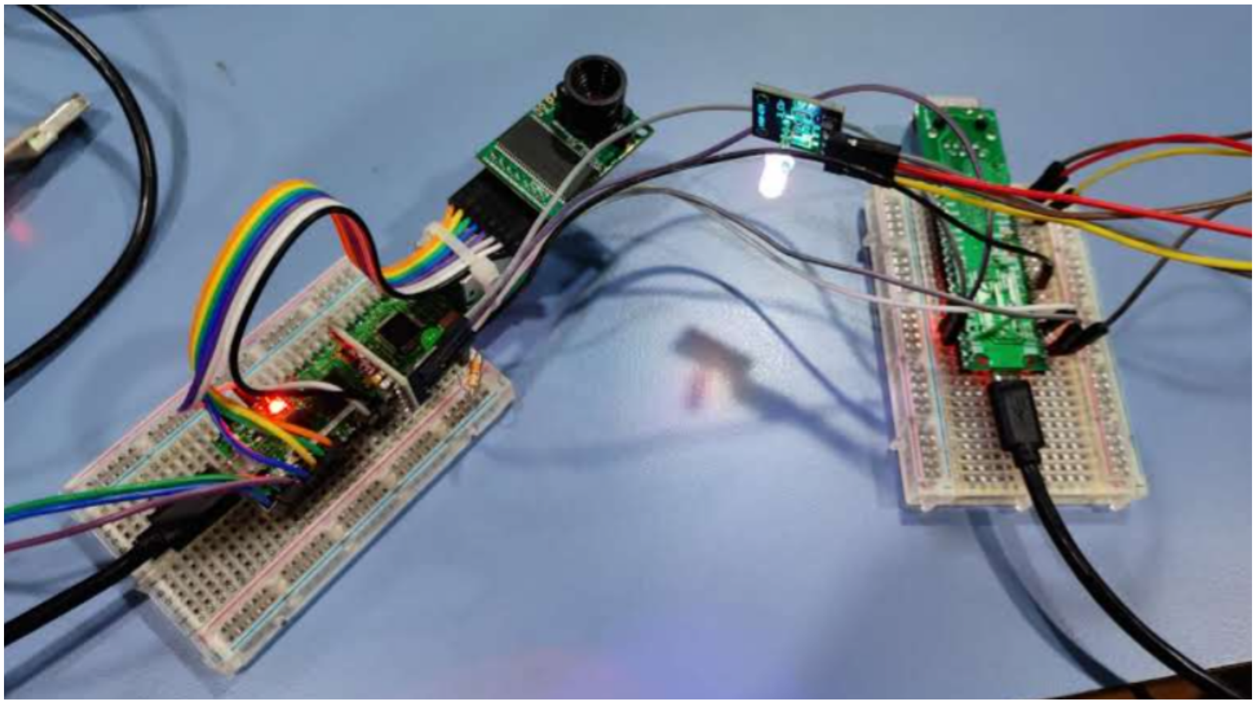
AI + IOT PROJECT REPORT

LampLed

WIZnet



Akshay Singhal 2147209

Akarshi Bansal 2147238

22.06.2022

INTRODUCTION

LED lamp based on hand gesture and setting auto brightness and light color combination according to the environment.

HYPOTHESIS

The LED lamp will use arducam which is supported by a pico board to recognize the hand and then another pico board will use APDS9960 sensor to recognize the gesture and sense the ambient light. The whole model is atomic and can work without any external support of any network. APDS9960 sensor is a kind of Infrared sensor which supports gesture detection, proximity detection,color and lux detection of ambience light.

MATERIALS

1. 2 WIZnet W5100S-EVB-Pico board
2. 2 Breadboard
3. 2 Usb cables
4. Arducam Mini 2MP Plus – OV2640 SPI Camera Module
5. APDS9960 IR sensor
6. Connecting wires
7. Resistors
8. RGB LEDs

PROCEDURE

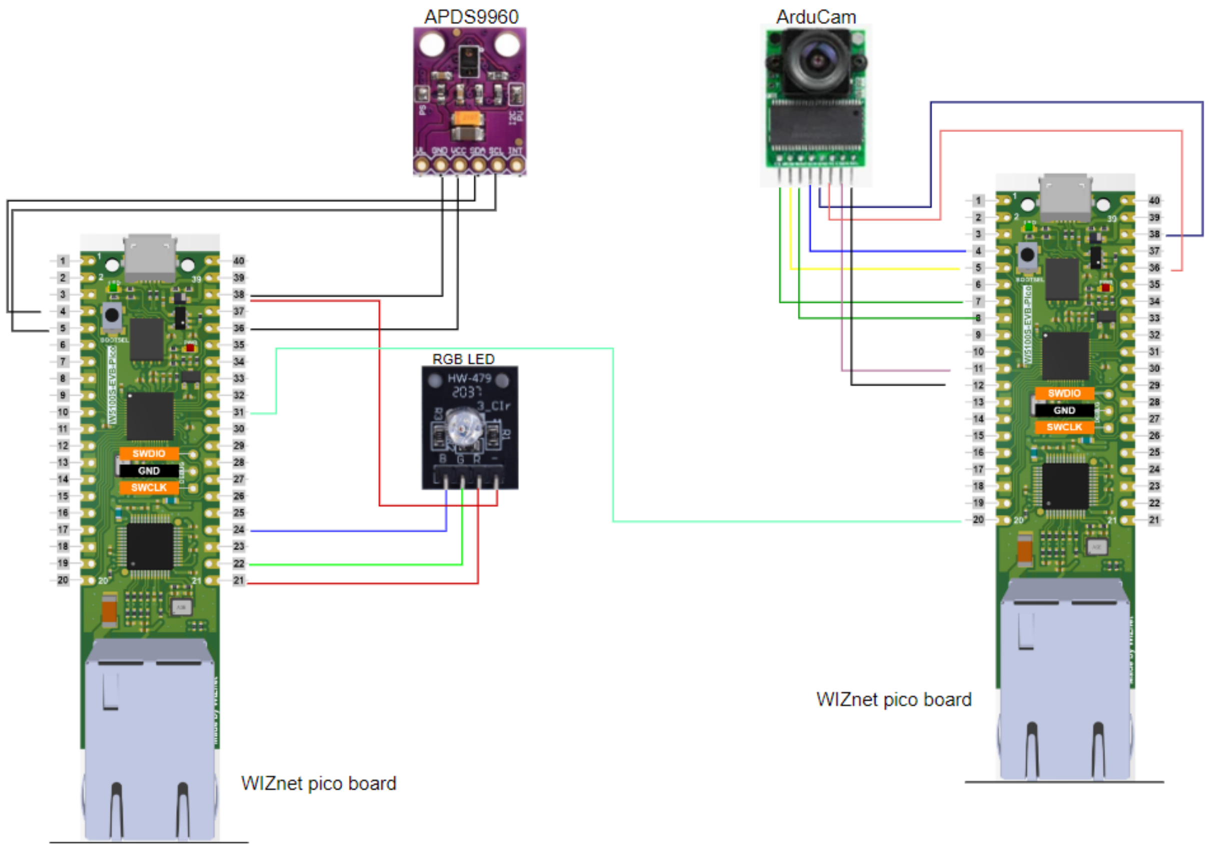
1. ArduCam will recognize the person and thus activate another W5100S-EVB-Pico board which contains an APDS9960 sensor.
2. APDS9960 sensor will detect the hand gesture.

    2.1. If the gesture is either Left or Down it will activate the color and proximity sensor to sense the ambient light.

        2.1.1. According to the LUX and RGB value it will set the brightness and RGB values for the LED and will Turn on the LED light.

    2.2 Else if the gesture is either Right or Up it will turn off the LED.

CIRCUIT DIAGRAM



EXPLANATION

For this model we connected two W5100S-EVB-Pico boards in which one contains an APDS9960 sensor and RGB LED light and another board contains an Arducam and a LED light.

The board which contains an APDS9960 sensor and LED light is termed as PicoBoard A (To avoid confusion), and another pico board as PicoBoard B.

Connections of W5100S-EVB-PicoBoard A with APDS9960

W5100S-EVB-PicoBoard A	APDS9960 sensor
GND	GND
3V3	Vcc
GP2	SDA
GP3	SCL

Connections of PicoBoard A with RGB LED

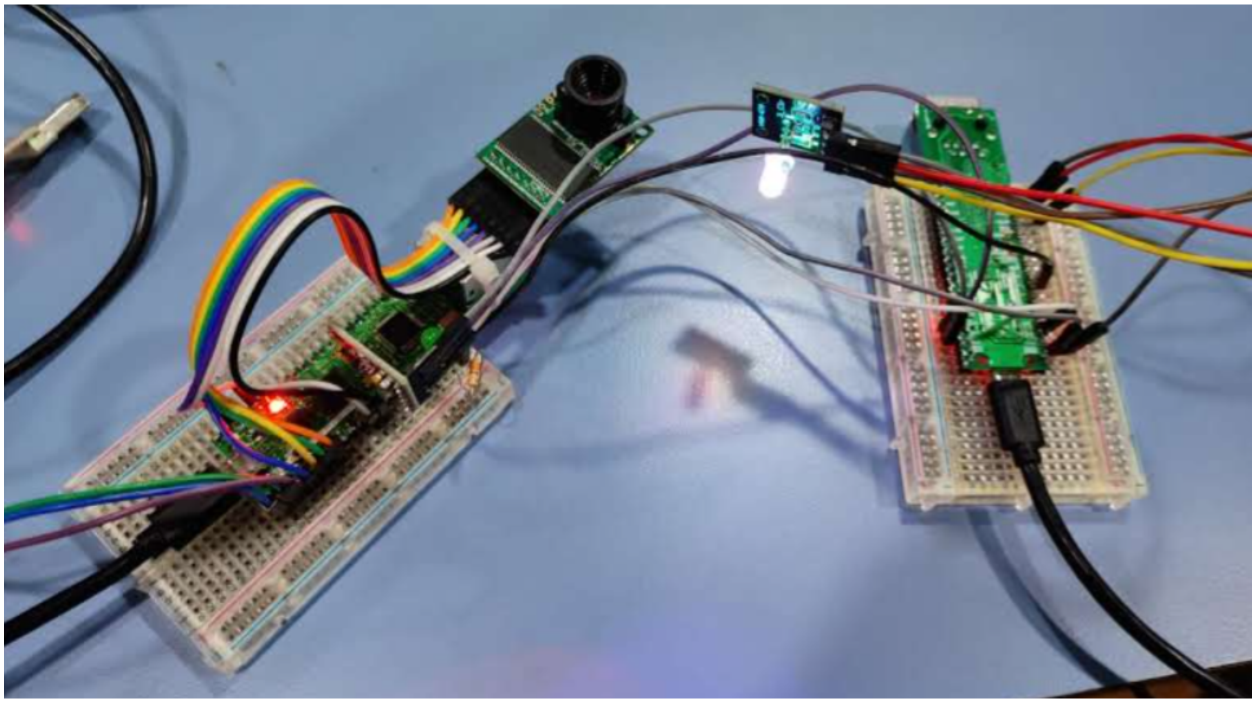
W5100S-EVB-PicoBoard A	RGB LED Light
GND	GND
GP17	R
GP16	G
GP18	B

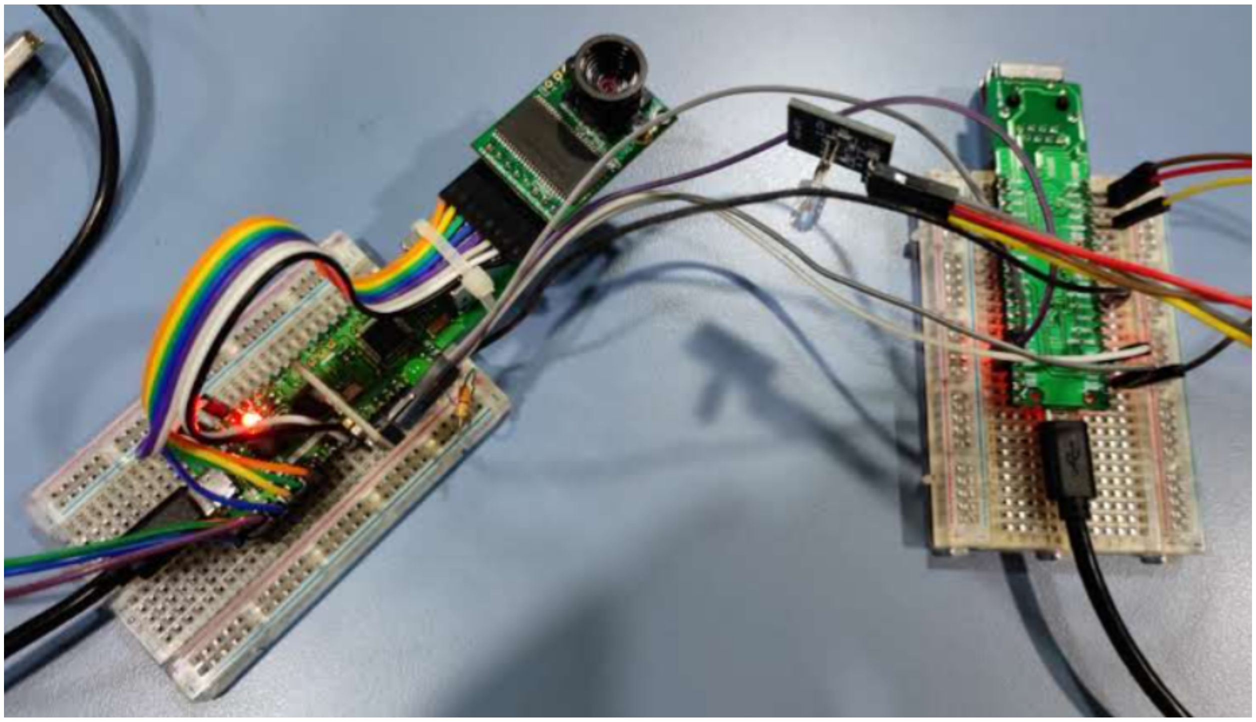
GP26 of PicoBoard A is Connected to GP15 of PicoBoard B to establish connection between the two PicoBoards.

Connection of PicoBoard B with Arducam

W5100S-EVB-PicoBoard B	Arducam
GP5	CS
GP3	MOSI
GP4	MISO
GP2	SCK
GND	GND

3V3	VCC
GP8	SDA
GP9	SCL





CODE EXPLANATION

- circuitPython code for APDS9960 sensor and to set LED values.

```
import board

import busio

import pwmio

import time

import digitalio

from adafruit_apds9960.apds9960 import APDS9960

from adafruit_apds9960 import colorutility

import analogio

adc = analogio.AnalogIn(GP27)

act = digitalio.DigitalInOut(board.GP26)
```

```
ledg = pwmio.PWMOut(board.GP16, frequency=5000, duty_cycle=0)

ledr = pwmio.PWMOut(board.GP17, frequency=5000, duty_cycle=0)

ledb= pwmio.PWMOut(board.GP18, frequency=5000, duty_cycle=0)

brightness = 0

flag = 0
```

- Import **board** module to set a sync between CircuitPython and WIZnet W5100S-EVB-Pico board in order to make the board compatible with the circuit python.
- Import **busio** module which contains an interface for using hardware-driven I2C communication from board and contains classes to support a variety of serial protocols. To enable the access to the I2C interface that will be used by apds-9960 for the communication. I2C is a two-wire protocol for communicating between devices. At the physical level it consists of 2 wires: SCL and SDA, the clock and data lines respectively.
- Import **pwmio** module that contains classes to provide basic pulse IO to set the RGB values of LED.
- Import **time** module which provides functions for working with times such as representing time in various formats, wait during code execution and measuring code efficiency. In order to add time delay in program code, use the *sleep()* function from the time module which helps to suspend execution of code for specified seconds.
- Import **digitalio** module for communicating digital IO with the W5100S-EVB-Pico board.It provides digital IO, which is passed from one pico board of arducam to another pico board. (Will give the values from the arducam whether a person is detected or not).
- Module **adafruit_apds** allows you to easily write Python code that reads the proximity and other readings from the APDS9960 sensor. It is developed by Avago technologies.
- From **adafruit_apds** import **apds9960** which will provide support to apds9960 sensors and all related functionalities. adafruit_apds.apds9960, it can detect simple gestures (left to right, right to left, up to down, down to up are currently supported).
- From **adafruit_apds** import **colorutility** as it will calculate the color temperature, RGB color combination and lux value of ambient light.

- Imported the **analogio** module contains classes to provide access to analog IO typically implemented with digital-to-analog (DAC) and analog-to-digital (ADC) converters, which will thus be passed to RGB light.
- Assign a variable **adc** for analog IO which will operate pin GP27.
- Assign a variable **act** for digital IO which will operate pin GP26.
- Assign a variable **ledg** for pwm output to pin GP16 which is responsible for the green color of the led.
- Assign a variable **ledr** for pwm output to pin GP17 which is responsible for the red color of the led.
- Assigned a variable **ledb** for pwm output to pin GP18 which is responsible for the blue color of the led.
- The initial value of the brightness is set to 0.
- The initial value of the flag is set to 0, which will be used later.

```
def rgb():
    global flag
    apds.enable_color = True
    while not apds.color_data_ready:
        time.sleep(0.005)

    # get the data and print the different channels
    r, g, b, c = apds.color_data
    print("red: ", r)
    print("green: ", g)
    print("blue: ", b)
    print("clear: ", c)

    lx = colorutility.calculate_lux(r, g, b)
    print("color temp
{}".format(colorutility.calculate_color_temperature(r, g, b)))

    print("light lux {}".format(lx))

    apds.enable_color = False
```

Function rgb() is defined to get values of ambient light and thus operate LED accordingly. So it will enable the color sensor by setting apds.enable_color = True. then wait while sensors get ready to sense the values. Following functions are helping to get the values for

- apds.color_data() is used to get the RGB values and clearness of the ambient light.
- colorutility.calculate_lux() is used to get the LUX values of ambient light.
- colorutility.calculate_color_temperature() is used to get the temperature of the ambient light.

```
    if(lx<=750):

        brightness = 65000

    elif(lx>750 and lx<1800):

        brightness = 32000

    elif(lx>=1800):

        brightness = 0

    if(brightness != 0):

        x = brightness#color combination of warm white has RGB ratio of
(35%:34%:31%)

        y = x

        z = x

        y = int(y*0.9715)

        z = int(z*0.8868)

        ledr.duty_cycle = x-r
        ledb.duty_cycle = y-b

        ledg.duty_cycle = z-g

        flag +=1

    else:

        flag = 0

        rgboff()

    print("Values for RGB lights :
",ledr.duty_cycle,ledg.duty_cycle,ledb.duty_cycle)

    apds.enable_proximity = True

    apds.enable_gesture = True

    if(flag == 1):

        time.sleep(2)

        rgb()
```

After calculating the LUX and RGB value of environment light we fix the brightness. If the LUX is more than 1800 brightness will be set to ZERO as there is enough light in the environment. Else if LUX lies between 750-1800, brightness will be set to 32000 as only dim light is needed and IF LUX is less than 750, brightness will be 65000 i.e max light is needed. Then RGB values will be set in a manner that they maintain the R:G:B ratio of 35%:34%:31%. Then providing respective values to RGB LED.

```
def rgboff():

    global flag

    flag = 0

    ledr.duty_cycle = 0

    ledb.duty_cycle = 0

    ledg.duty_cycle = 0
```

Function rgboff() will turn off the LED by setting RGB values to 0 and will zero the flag value. Where the flag is a global variable.

```
i2c = busio.I2C(board.GP3, board.GP2)

apds = APDS9960(i2c)

apds.enable_proximity = True

apds.enable_gesture = True

gesturels = [0x02,0x03]

gestureoff = [0x01,0x04]
```

Initializing i2c ports with pins GP2 and GP3, which are connected with SDA and SCL pins of apds9960 sensor. After setting up the connection with apds9960, enable the proximity and gesture function to detect the gestures for the first time.

Declaring two lists with gesture values which will be used to identify whether to turn on or off the LED. List with variable name "gesturels" contains hex values for LEFT and DOWN gesture, whereas other list contains hex values for RIGHT and UP gesture.

```
def apdss():

    apds.enable_color = False

    apds.enable_proximity = True

    apds.enable_gesture = True

    gesture = apds.gesture()

    if gesture == 0x01:

        print("up")

    elif gesture == 0x02:

        print("down")

    elif gesture == 0x03:

        print("left")

    elif gesture == 0x04:
        print("right")


    if(gesture in gesturels):

        rgb()

    elif(gesture in gestureoff):

        rgboff()
```

Since, we can support max two functions at same time, therefore we will disable the apds9960 function to detect values of ambient light, by setting enable_color = False. And enable the proximity and gesture sensors again to detect the gestures.

Then detecting the gesture using the function apds.gesture(). There will be following cases of gestures :

- Gesture UP has the hex code 0x01.
- Gesture DOWN has the hex code 0x02.
- Gesture LEFT has the hex code 0x03.
- Gesture RIGHT has the hex code 0x04.

Thus if the gesture corresponds to either LEFT or DOWN it will call the rgb() else for RIGHT and UP gesture it will call rgboff(). Where both the functions are user defined functions responsible for setting LED values.

```
while True:

    act.direction = digitalio.Direction.INPUT

    if(act.value == True):

        apdss()
```

Running Code for infinite loop, as “act” variable is defined to take digital input from pin GP26, It will be **True** when arducam will recognize hand/human after which it will call apdss(), which is a user defined function responsible for detecting gestures.

- ❖ EXPLANATION
    - FLOW CONTROL

Importing all the required libraries to support sensor and board compatible with circuitPython.

Define the Analog, digital and i2c pins for the input and output.

Define PWMio for LED outputs.

Take Digital input from another pico board used to support ArduCam, if it is 1 then call function to detect gesture.

Enabling Gesture and Proximity sensor of apds9960 to get the gesture. If the gesture is either left or down, call another function to detect ambient light. Else call function to turn off all the LEDs.

To detect ambient light, enable apds color sensor and detect the conditions and set values accordingly.

Functions Used:

- apdss() is used to enable proximity and gesture sensor in order to detect the gesture.
- rgb() is used to enable color sensors to detect the ambient light conditions and then send appropriate output to the LEDs.
- rgboff() is used to set all LED values to 0 , in order to turn them off.

❖ AI Implementation

- Person Detection Code
    - Explanation:

```
#include "main_functions.h"

#include "detection_responder.h"

#include "image_provider.h"

#include "model_settings.h"

#include "person_detect_model_data.h"

#include "tensorflow/lite/micro/micro_error_reporter.h"
```

```
#include "tensorflow/lite/micro/micro_interpreter.h"

#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"

#include "tensorflow/lite/schema/schema_generated.h"

#include "tensorflow/lite/version.h"

#include "arducam.h"

#include "pico/stdlib.h"

#include "hardware/irq.h"

#include "tensorflow/lite/micro/micro_time.h"

#include <climits>
```

Including necessary files to support the human detection module on arducam.

```

#define TF_LITE_MICRO_EXECUTION_TIME_BEGIN      \

  int32_t start_ticks;                          \

  int32_t duration_ticks;                       \

  int32_t duration_ms;


#define TF_LITE_MICRO_EXECUTION_TIME(reporter, func)
\

  if (tflite::ticks_per_second() == 0) {
\

    TF_LITE_REPORT_ERROR(reporter,
\

                         "no timer implementation found");
\

  }
```

```
\
  start_ticks = tflite::GetCurrentTimeTicks();
\
  func;
\
  duration_ticks = tflite::GetCurrentTimeTicks() - start_ticks;
\
  if (duration_ticks > INT_MAX / 1000) {
\
    duration_ms = duration_ticks / (tflite::ticks_per_second() /
1000); \
  } else {
\
    duration_ms = (duration_ticks * 1000) /
tflite::ticks_per_second(); \
  }
\
  TF_LITE_REPORT_ERROR(reporter, "%s took %d ticks (%d ms)", #func,
\
                                    duration_ticks, duration_ms);
```

Defining macros to keep a timer of execution and detecting images to sync them with frame speed.

```


#define TF_LITE_MICRO_EXECUTION_TIME_SNIPPET_START(reporter)
\
  if (tflite::ticks_per_second() == 0) {
\
    TF_LITE_REPORT_ERROR(reporter,
\
                            "no timer implementation found");
\
  }
\
  start_ticks = tflite::GetCurrentTimeTicks();


#define TF_LITE_MICRO_EXECUTION_TIME_SNIPPET_END(reporter, desc)
\
  duration_ticks = tflite::GetCurrentTimeTicks() - start_ticks;
\
  if (duration_ticks > INT_MAX / 1000) {
\
    duration_ms = duration_ticks / (tflite::ticks_per_second() /
1000); \
  } else {
\
    duration_ms = (duration_ticks * 1000) /
tflite::ticks_per_second(); \
  }
\
  TF_LITE_REPORT_ERROR(reporter, "%s took %d ticks (%d ms)", desc,
\
                                    duration_ticks, duration_ms);
namespace {
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
```

```
TfLiteTensor* input = nullptr;
```

Globals, used for compatibility with Arduino-style sketches.

```
constexpr int kTensorArenaSize = 136 * 1024;

static uint8_t tensor_arena[kTensorArenaSize];

}  // namespace
```

In order to use optimized tensorflow lite kernels, a signed int8_t quantized model is preferred over the legacy unsigned model format. This means that throughout this project, input images must be converted from unsigned to signed format. The easiest and quickest way to convert from unsigned to signed 8-bit integers is to subtract 128 from the unsigned value to get a signed value.

An area of memory to use for input, output, and intermediate arrays.

```
#ifndef DO_NOT_OUTPUT_TO_UART


void on_uart_rx() {

    uint8_t cameraCommand = 0;

    while (uart_is_readable(UART_ID)) {

        cameraCommand = uart_getc(UART_ID);
```

RX interrupt handler.

```
        // Can we send it back?

        if (uart_is_writable(UART_ID)) {

            uart_putc(UART_ID, cameraCommand);

        }

    }
```

```
}
```

Can we send it back?

```
void setup_uart() {

  uint baud = uart_init(UART_ID, BAUD_RATE);
```

Set up our UART with the required speed.

```
  gpio_set_function(UART_TX_PIN, GPIO_FUNC_UART);

  gpio_set_function(UART_RX_PIN, GPIO_FUNC_UART);
```

Set the TX and RX pins by using the function select on the GPIO

Set datasheet for more information on function select

```
  uart_set_format(UART_ID, DATA_BITS, STOP_BITS, PARITY);
```

Set our data format

```
  uart_set_fifo_enabled(UART_ID, false);
```

Turn off FIFO's - we want to do this character by character

```
  int UART_IRQ = UART_ID == uart0 ? UART0_IRQ : UART1_IRQ;
```

Set up a RX interrupt

We need to set up the handler first

Select correct interrupt for the UART we are using

```
  irq_set_exclusive_handler(UART_IRQ, on_uart_rx);

  irq_set_enabled(UART_IRQ, true);
```

And set up and enable the interrupt handlers

```
  uart_set_irq_enables(UART_ID, true, false);

}
```

Now enable the UART to send interrupts - RX only

```
#else
void setup_uart() {}
#endif
void setup() {
  setup_uart();
```

The name of this function is important for Arduino compatibility.

```
  static tflite::MicroErrorReporter micro_error_reporter;
  error_reporter = &micro_error_reporter;
```

Set up logging. Google style is to avoid globals or statics because of lifetime uncertainty, but since this has a trivial destructor it's okay. NOLINTNEXTLINE(runtime-global-variables)

```
  model = tflite::GetModel(g_person_detect_model_data);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
                         "Model provided is schema version %d not
equal "
                         "to supported version %d.",
                         model->version(), TFLITE_SCHEMA_VERSION);
    return;
  }
```

Map the model into a usable data structure. This doesn't involve any copying or parsing, it's a very lightweight operation.

```
  static tflite::MicroMutableOpResolver<5> micro_op_resolver;
  micro_op_resolver.AddAveragePool2D();
  micro_op_resolver.AddConv2D();

  micro_op_resolver.AddDepthwiseConv2D();

  micro_op_resolver.AddReshape();

  micro_op_resolver.AddSoftmax();
```

Pull in only the operation implementations we need.

This relies on a complete list of all the ops needed by this graph.

An easier approach is to just use the AllOpsResolver, but this will incur some penalty in code space for op implementations that are not needed by this graph. tflite::AllOpsResolver resolver;

NOLINTNEXTLINE(runtime-global-variables)

```
  static tflite::MicroInterpreter static_interpreter(

      model, micro_op_resolver, tensor_arena, kTensorArenaSize,
error_reporter);

  interpreter = &static_interpreter;
```

Build an interpreter to run the model with.

NOLINTNEXTLINE(runtime-global-variables)

```
  TfLiteStatus allocate_status = interpreter->AllocateTensors();

  if (allocate_status != kTfLiteOk) {

    TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors()
failed");

    return;

  }
```

Allocate memory from the tensor_arena for the model's tensors.

```
  input = interpreter->input(0);

}
```

Get information about the memory area to use for the model's input.

```
void loop() {

  TF_LITE_MICRO_EXECUTION_TIME_BEGIN


  TF_LITE_MICRO_EXECUTION_TIME_SNIPPET_START(error_reporter)
```

The name of this function is important for Arduino compatibility.

```
  if (kTfLiteOk != GetImage(error_reporter, kNumCols, kNumRows,
kNumChannels,

                            input->data.int8)) {

    TF_LITE_REPORT_ERROR(error_reporter, "Image capture failed.");

  }

  TF_LITE_MICRO_EXECUTION_TIME_SNIPPET_END(error_reporter,
"GetImage")


  TF_LITE_MICRO_EXECUTION_TIME_SNIPPET_START(error_reporter)
```

Get image from provider.

```
  if (kTfLiteOk != interpreter->Invoke()) {

    TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed.");

  }

  TF_LITE_MICRO_EXECUTION_TIME_SNIPPET_END(error_reporter, "Invoke")


  TfLiteTensor* output = interpreter->output(0

);
```

Run the model on this input and make sure it succeeds.

```
  int8_t person_score = output->data.uint8[kPersonIndex];

  int8_t no_person_score = output->data.uint8[kNotAPersonIndex];
```

Process the inference results.

```
const uint LED_PIN = 15;

gpio_init(LED_PIN);

gpio_set_dir(LED_PIN, GPIO_OUT);



  if (person_score > 50){

    gpio_put(LED_PIN,1);

    sleep_ms(250);

  }else{

    gpio_put(LED_PIN,0);

    }

  RespondToDetection(error_reporter, person_score, no_person_score);

}
```

Turning on the Pi if person is detected:

Pin for turning on the RPi4 using Pin 15 from the Pico

Importing tensorflow and all the required libraries to support person detection . We are defining GP15 as the LED pin which will be switched on or off depending on the the person_score value , which indicates whether a hand is detected or not.

If the person_score value is greater than 50 it means that the hand is detected and the LED light will be switched on otherwise the LED light will be switched off.

CONCLUSION

A led lamp controlled by hand gesture which provides led brightness and color according to the ambient light, Thus leading towards sustainable development and provides a better environment for the eyes. The model is completely atomic in nature and does not require any external support of any network or hardware.Here is the link for the video of the working model.

REFERENCES

1. https://github.com/ArduCAM/RPI-Pico-Cam/tree/master/tflmicro/examples
2. https://github.com/adafruit/Adafruit_CircuitPython_APDS9960/tree/main/examples
3. APDS9960 Gesture, Proximity, Light & RGB Sensor Tutorial with Arduino