TERMO KAMERA

Projekt pri predmetu Vhodno-izhodne naprave

Mentor:
Rozman Robert

Avtor:
Jožef Poklukar

Ljubljana, junij 2024

Vsebina

1 Uvod

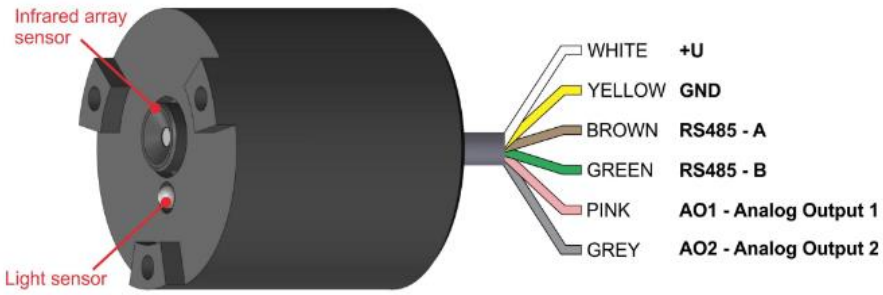
Za projekt sem se s ploščo STM32H750 povezal s termo kamero Smarteh LIR2. Ker s tipalom komuniciraš le s protokolom RS485, za keterega na H7 ni priključka, sem uporabil max485 modul, ki pretvori signal iz RS485 v UART.Tega sem lahko priključil na ploščo. Kamera pošilja temperaturo slikovnih pik, katere sem nato uporabil in prikazal na ekranu H7. Ker je matrika velika le 16x12 sem jo interpoliral, da se barve med seboj lepše prelivajo in nastane očem bolj razumljiva slika.

2 Vezava

2.1 Smarteh LIR2

LIR2 je infrardeča kamera, ki meri toploto med -40 in 300°c z natančnostjo ±1.5°c. Podatke zbira v resoluciji 16x12 slikovnih pik, ki jih lahko beremo iz vhodnih registrov. Z njo komuniciramo preko Modbus RTU (Remote Terminal Unit), torej ji kot Master pošljemo ukaz za branje in nam kot takoimenovan Slave vrne zaprošene podatke. Da kamera komunicira, vidimo z utripajočo zeleno lučjo, ki je okoli vrha tipala.

Ob pogledu na kabel vidimo, da je sestavljen iz šestih manjših. Beli je za napajanje, ki mora biti med 15 in 28 volti, rumeni za uzemljitev, rjav in zelen za RS485 komunikacijo (prvi je A drugi B), roza in siv pa sta analogna izhoda.

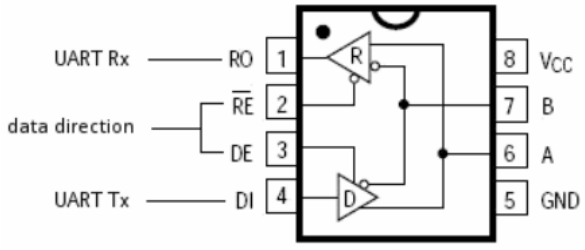


Slika 1: Povezave Smarteh LIR2

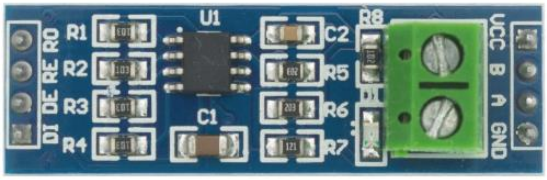
2.2 Max485 modul

Namen modula je, da pretvori signale iz RS485 v UART. Iz kamere dobimo diferencialne signale, torej sta A in B enosmerna ali semi-duplex. Na modul ju vežemo A na A in B na B, poleg teh dveh pinov, najdemo še napajalnega, ki je lahko ali 3.3 V ali 5 V, in ozemljitvenega.

Na nasprotni strani sta sredinska pina za določanje smer prometa. RE (Receiver Enable) bo ob vpostavitvi vklopil branje iz Slave-a, DE (Driver Enable) pa bo vzpostavitvi vklopil pošiljanje ukazov Slave-u. Ker pa je RE že negiran, nam ni potrebno vezati vsakega posebej, ampak ju lahko vežemo skupaj, ter ju tako krmilimo preko le ene povezave. Na levi in desni sta še RO (Receiver Output), ki ga vežemo na UART Rx, ter DI (Driver Input), ki ga vežemo na UART Tx.



Slika 2: Vezava max485 modula

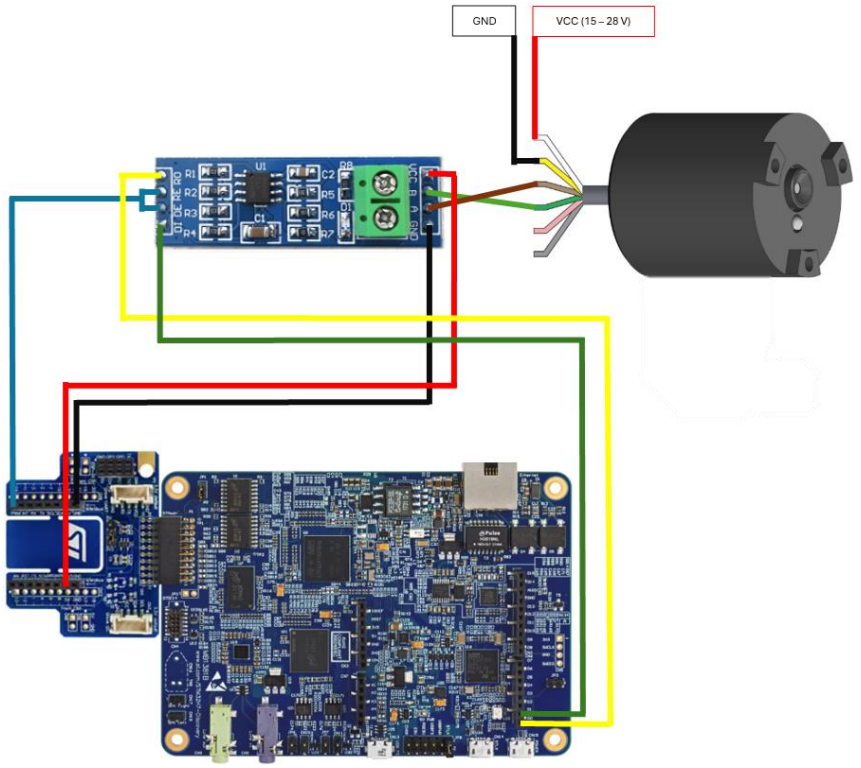


Slika 3: max485

2.3 Skupna vezava

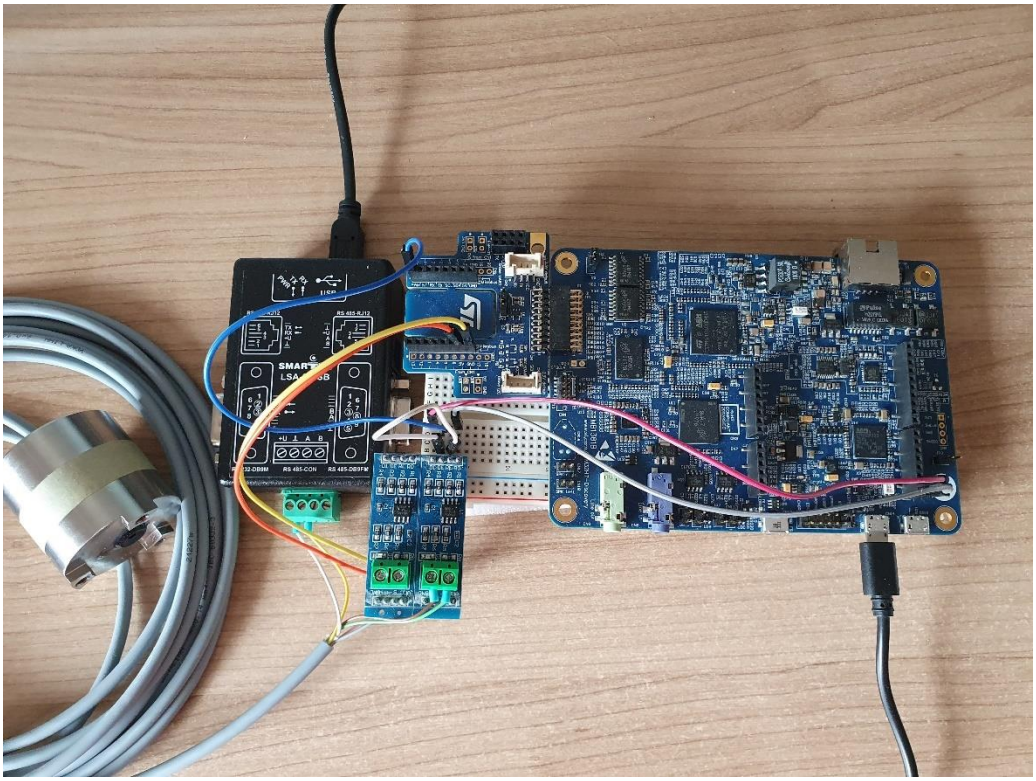
Spodnja slika prikazuje kako sem vezal komponente med seboj. Tipalo sem moral vezati na svoj vir napajanja, ker H7 nima dovolj velike napetosti.

V začetku sem želel uporabiti UART2, ki se nahaja na razširitveni plošči, a mi ta ni deloval niti z enim samim bajtom, zato sem uporabil UART1 na glavni plošči. Imena povezav in pinov so v tabeli.



Slika 4: Prikaz vezave projekta

max485	STM32H750	Smarteh LIR2	Zunanji vir napetosti
RO	**PB6**	**/**	**/**
RE	**PA3**	**/**	**/**
DE	**PA3**	**/**	**/**
DI	**PB7**	**/**	**/**
VCC	**3.3 V**	**/**	**/**
B	**/**	**B** (zelen)	**/**
A	**/**	**A** (rjav)	**/**
GND	**GND**	**/**	**/**
/	**/**	**+U** (bel - vcc)	**15 – 28 V**
/	**/**	**GND** (rumen)	**GND**
/	**/**	**AO1** (roza)	**/**
/	**/**	**AO2** (siv)	**/**

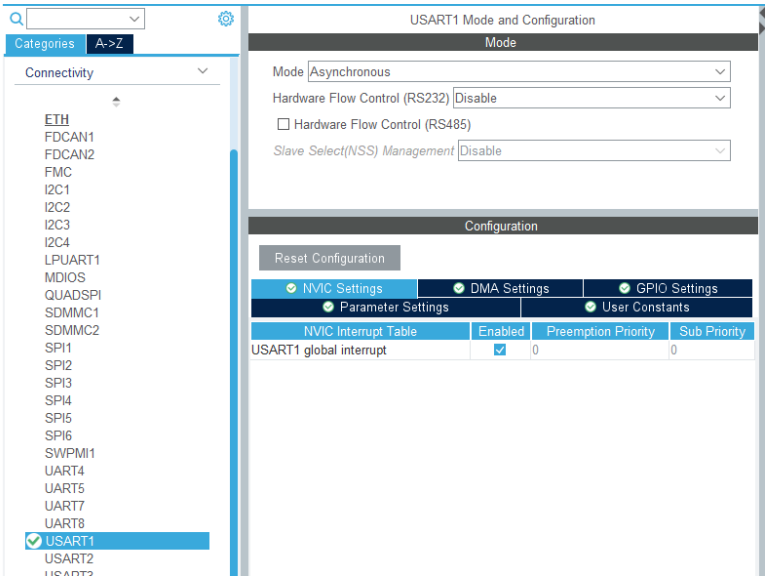


Slika 5: Vezava projekta

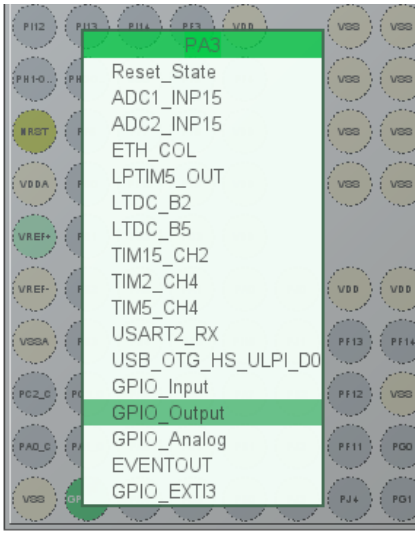
3 Komunikacija

3.1 Inicializacija

Pred začetkom pisanja kode sem v CubeMX vklopil USART1, ga nastavil na asinhronskega in mu vklopil globalno prekinitev. Parametrov UART-a nisem spreminjal, saj so privzeti že pravilni tj. 115200 bit/s, dolžina 8 bitov, en stop bit in brez paritete. Za nastavljanje smeri komunikacije na max485 modulu, sem uporabil pin PA3 in ga nastavil na GPIO Output. Nastavitve sem shranil in zgeneriral kodo



Slika 7: Nastavitve USART1



Slika 6: Vklop pina PA3

V projekt sem uvozil datoteki *modbus_crc.c* in *modbus_crc.h*, ki skrbita za prvilno računanje CRC-ja v naslednjem koraku. Vsebina teh dveh je spodaj.

```
/*
 * modbus_crc.h
 *
 *  Created on: Sep 16, 2022
 *      Author: arunr
 */

#ifndef INC_MODBUS_CRC_H_
#define INC_MODBUS_CRC_H_

uint16_t crc16(uint8_t *buffer, uint16_t buffer_length);


#endif /* INC_MODBUS_CRC_H_ */
```

```
#include "stdint.h"

/* Table of CRC values for high-order byte */
static const uint8_t table_crc_hi[] = {
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
```

```
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
};

/* Table of CRC values for low-order byte */
static const uint8_t table_crc_lo[] = {
    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
    0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
    0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
    0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
    0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
    0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
    0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
    0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
    0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
    0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
    0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
    0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
    0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
    0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
    0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
    0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
    0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
    0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
    0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
    0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
    0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
    0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
    0x43, 0x83, 0x41, 0x81, 0x80, 0x40
};

uint16_t crc16(uint8_t *buffer, uint16_t buffer_length)
{
    uint8_t crc_hi = 0xFF; /* high CRC byte initialized */
    uint8_t crc_lo = 0xFF; /* low CRC byte initialized */
    unsigned int i; /* will index into CRC lookup */
```

```
    /* pass through message buffer */
    while (buffer_length--) {
        i = crc_lo ^ *buffer++; /* calculate the CRC  */
        crc_lo = crc_hi ^ table_crc_hi[i];
        crc_hi = table_crc_lo[i];
    }

    return (crc_hi << 8 | crc_lo);
}
```

Ustvaril sem še novo .c datoteko v katero sem pisal vso kodo, ki je v povezavi s komunikacijo s kamero. Na začetku te datoteke sem definiral parametre, kot so število registrov, ki jih želim naenkrat prenesti, kakšne so dimenzije matrike, spremenljivke za pošiljanje in sprejemanje podatkov in še nekaj spremenljivk za nadzor prenašanje. V glavni funkciji *start_uart*, sem vklopil globalne prekinitve, da se bo klicala željena funkcija, ko bo UART pridobil podatke. Sledi *while* zanka, katere vsebina je v naslednjem poglavju.

```
#include "main.h"
#include "modbus_crc.h"


/* Private variables ---------------------------------------------------------*/
extern UART_HandleTypeDef huart1;


#define BUFSIZE 64
#define MATRIX_ROWS 12
#define MATRIX_COLS 16

uint32_t matrix[MATRIX_ROWS][MATRIX_COLS];

uint8_t rxData[2 * BUFSIZE + 5]; // Receive buffer for UART
uint8_t txData[16];
int n = 0;
int rxReady = 1;

int rxSize = 2 * BUFSIZE + 5;    //each register takes 2 bytes and 5 are for modbus
protocol

int rst = 0;


/* Private function prototypes -----------------------------------------------*/
void sendData (uint8_t *data);

/**
  * @brief  The application entry point.
  * @retval int
  */
int start_uart(void)
{

  __enable_irq();
```

3.2 Master zahteva

Pri uporabi Modbus protokola moramo najprej poslati zahtevo, da prejmemo podatke od naprave Slave. V zahtevi določimo ID naprave Slave, ki zavzame 1 bajt, in funkcijsko kodo, ki prav tako zavzame 1 bajt. Funkcijska koda določa operacijo, ki jo želimo izvesti. V programu sem za naslov uporabil 234, saj je ta na kameri privzeto nastavljen. Za funkcijsko kodo sem vpisal 4, s tem sem sporočil branje vhodnih registrov, kjer se nahajajo vrednosti toplote.

Naslednja dva podatka v sporočilu sta dolga dva bajta. Prvi je začetni register drugi pa število zaporednih registrov, ki jih želimo prebrati. Prvega sem nastavil na devet, drugega pa na le 64. Ker ima modbus omejeno število bajtov, ki jih lahko pošlje z eno zahtevo sem izračunal, da je največje število registrov, ki jih lahko naenkrat preberem 96, kar pomeni da je potrebno brati vsaj trikrat. Odločil sem se da bom bral v treh enako velikih segmentih po 64, zato je branje registrov v zanki. Ta poskrbi, da se master zahteva pošlje trikrat in hkrati nastavi potreben začeten naslov branja.

Zadnja dva bajta v sporočilu sta CRC (Cyclic Redundancy Check), ki z deljenjem dosedanjega sporočila s polinomom določi ostanek in ga pripne k master zahtevi. Ta del sporočila bo Slave uporabil, da preveri ali so se dobljeni podatki prenesli pravilno.

Preden se podatki pošljejo je potrebno sporočiti max485 modulu, da bo Master poslal zahtevo kar sem naredil z nastavljanjem pina PA3.

```
while (1)
  {

    /* USER CODE END WHILE */
        for (int tretjina = 0; tretjina < 3; tretjina++) {
              if (rst == 1) {

                    tretjina = -1;
                    rst = 0;
                    n = 0;

              } else if (rxReady == 1) {
                    HAL_Delay(200);

                    HAL_UARTEx_ReceiveToIdle_IT(&huart1, rxData, rxSize);

                    int addressOffset = tretjina * BUFSIZE;


                    txData[0] = 234;  // slave address
                    txData[1] = 0x04;  // Function code for Read Input Registers

                    txData[2] = 0;
                    txData[3] = 9 + addressOffset;//Reading start with offset

                    txData[4] = 0;
                    txData[5] = BUFSIZE;    //Number of registers we want to read

                    uint16_t crc = crc16(txData, 6);
                    txData[6] = crc&0xFF;    // CRC LOW
                    txData[7] = (crc>>8)&0xFF;  // CRC HIGH
```

```
                    rxReady = 0;
                    n = tretjina;
                    sendData(txData);

                } else {
                    tretjina--;
                }
        }
  }



void sendData (uint8_t *data)
{
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);

      if (HAL_UART_Transmit(&huart1, data, 8, 2000) != HAL_OK) {
            Error_Handler();
      }

      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
}
```

3.3 Prejemanje podatkov

Spodnja funkcija se kliče, ko se prenesejo vsi zaprošeni registri. Pridobljeni podatki so v spremenljivki rxData po 2 bajta na en register, zato jih v dvojni for zanki prepišemo v matriko velikosti slike kamere in združimo tista dva bajta v eno število. Ko je tretji prenos izveden se kliče funkcija za prikaz slike na ekranu.

```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{

      if ((Size < rxSize) || (rxData[0] != 234) || (n > 2)){
          rst = 1;
        } else {
                int count = 3; // first three bytes of response are slave address,
function code, Byte count

                  for (int i = 0; i < 4; i++) {
                        for (int col = MATRIX_COLS - 1; col >= 0; col--) {
                              matrix[i + (n * 4)][col] = rxData[count]<<8 |
rxData[count + 1];
                              count = count + 2;
                        }
                  }
                  if (n == 2) {
                        rst = 1;
                        displayMatrixOnLCD(&matrix);
                  }

        }
      rxReady = 1;
}
```

4 Prikazovanje na ekranu

4.1 Pretvarjenje vrednosti matrike v barve

Prva stvar, ki jo funkcija *displayMatrixOnLCD* stori, je da pretvori vrednosti dobljene iz kamere v barvo. Maksimalno toploto sem omejil na 5000 oz vrednost 50°c, ker sem želel prikazati predvsem sobno in človeško temperaturo.

```
void displayMatrixOnLCD(uint32_t matrix[12][16]) {
    for (int y = 0; y < MATRIX_SIZE_Y; y++) {
        for (int x = 0; x < MATRIX_SIZE_X; x++) {
            uint16_t temp = matrix[y][x];

            // Limit temperature to the range 0 to 5000
            if (temp > 5000) temp = 5000;

            // Convert temperature to color
            matrix[y][x] = tempToColor(temp);
        }
    }
```

Samo pretvorbo stori funkcija *tempToColor*. Vrednost se v if stavkih najde med kateri dve barvi spada in se nato izračuna točen odtenek. Vsaka vrednost RGB se pridobi posebej in je pomnožena z razmerjem oddaljenosti prej določenih dveh barv.

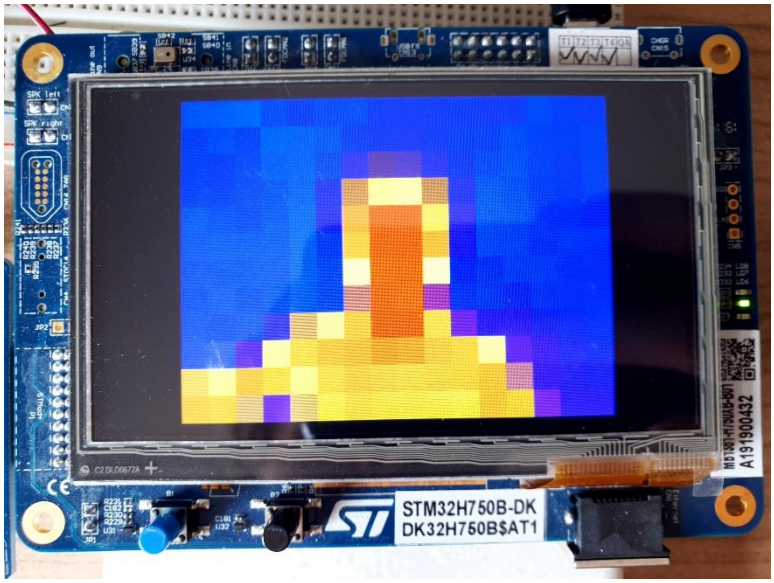
Barv sem dal veliko, zato da so različne temperature kar se da očitne.

```
uint32_t tempToColor(uint16_t temp) {
    // Define color values
    const uint32_t darkBlue = 0xFF000030UL;  // Dark blue color for coldest
    const uint32_t blue = 0xFF02004aUL;     // Blue color for very cold
    const uint32_t lightBlue = 0xFF020098UL;     // Blue color for colder
    const uint32_t lightestBlue = 0xFF5000fdUL;     // Blue color for cold
    const uint32_t purple = 0xFF800080UL;     // Purple color for warm
    const uint32_t yellow = 0xFFFFFF00UL;  // Yellow color for warmer
    const uint32_t orange = 0xFFFFA500UL;  // Orange color for hot
    const uint32_t red = 0xFFFF0000UL;     // Red color for very hot
    const uint32_t white = 0xFFFFFFFFUL;   // White color for extremely hot

    uint8_t redValue, greenValue, blueValue;
    if (temp <= 2200) {
            // Dark Blue to Blue
            float ratio = (float)temp / 2200.0;

            redValue = (uint8_t)((1.0 - ratio) * ((darkBlue >> 16) & 0xFF) + ratio *
((blue >> 16) & 0xFF));
            greenValue = (uint8_t)((1.0 - ratio) * ((darkBlue >> 8) & 0xFF) + ratio *
((blue >> 8) & 0xFF));
            blueValue = (uint8_t)((1.0 - ratio) * (darkBlue & 0xFF) + ratio * (blue &
0xFF));
      } else if (temp <= 2400) {
        // Blue to light Blue
            float ratio = (float)(temp - 2200) / 200.0;
```

```
        redValue = (uint8_t)((1.0 - ratio) * ((blue >> 16) & 0xFF) + ratio *
((lightBlue >> 16) & 0xFF));
        greenValue = (uint8_t)((1.0 - ratio) * ((blue >> 8) & 0xFF) + ratio *
((lightBlue >> 8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (blue & 0xFF) + ratio * (lightBlue &
0xFF));
    } else if (temp <= 2600) {
        // Light Blue to lightest blue
             float ratio = (float)(temp - 2400) / 200.0;

        redValue = (uint8_t)((1.0 - ratio) * ((lightBlue >> 16) & 0xFF) + ratio *
((lightestBlue >> 16) & 0xFF));
        greenValue = (uint8_t)((1.0 - ratio) * ((lightBlue >> 8) & 0xFF) + ratio *
((lightestBlue >> 8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (lightBlue & 0xFF) + ratio *
(lightestBlue & 0xFF));
    } else if (temp <= 2800) {
        // lightest blue to purple
        float ratio = (float)(temp - 2600) / 200.0;

        redValue = (uint8_t)((1.0 - ratio) * ((lightestBlue >> 16) & 0xFF) + ratio *
((purple >> 16) & 0xFF));
        greenValue = (uint8_t)((1.0 - ratio) * ((lightestBlue >> 8) & 0xFF) + ratio *
((purple >> 8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (lightestBlue & 0xFF) + ratio * (purple &
0xFF));
    } else if (temp <= 3000) {
        // purple to Yellow
        float ratio = (float)(temp - 2800) / 200.0;

        redValue = (uint8_t)((1.0 - ratio) * ((purple >> 16) & 0xFF) + ratio * ((yellow
>> 16) & 0xFF));
        greenValue = (uint8_t)((1.0 - ratio) * ((purple >> 8) & 0xFF) + ratio *
((yellow >> 8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (purple & 0xFF) + ratio * (yellow &
0xFF));
    } else if (temp <= 3250) {
        // Yellow to Orange
        float ratio = (float)(temp - 3000) / 200.0;

        redValue = (uint8_t)((1.0 - ratio) * ((yellow >> 16) & 0xFF) + ratio * ((orange
>> 16) & 0xFF));
        greenValue = (uint8_t)((1.0 - ratio) * ((yellow >> 8) & 0xFF) + ratio *
((orange >> 8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (yellow & 0xFF) + ratio * (orange &
0xFF));
    } else if (temp <= 3500) {
        // Orange to Red
        float ratio = (float)(temp - 3250) / 250.0;

        redValue = (uint8_t)((1.0 - ratio) * ((orange >> 16) & 0xFF) + ratio * ((red >>
16) & 0xFF));
        greenValue = (uint8_t)((1.0 - ratio) * ((orange >> 8) & 0xFF) + ratio * ((red
>> 8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (orange & 0xFF) + ratio * (red & 0xFF));
    } else {
        // Red to White
        float ratio = (float)(temp - 3500) / 1500.0;

        redValue = (uint8_t)((1.0 - ratio) * ((red >> 16) & 0xFF) + ratio * ((white >>
16) & 0xFF));
```

```
        greenValue = (uint8_t)((1.0 - ratio) * ((red >> 8) & 0xFF) + ratio * ((white >>
8) & 0xFF));
        blueValue = (uint8_t)((1.0 - ratio) * (red & 0xFF) + ratio * (white & 0xFF));
    }

    // Combine into a single color value
    uint32_t color = (0xFF << 24) | (redValue << 16) | (greenValue << 8) | blueValue;

    return color;
}
```

Na tej točki sem lahko v prvo prikazal sliko na ekran in dobil kar je na spodnji sliki.



Slika 8: Prikaz slike brez interpolacije

4.2 Interpolacija posemeznih slikovnih pik

Nadaljni del funkcije *displayMatrixOnLCD* je računanje barvne vrednosti posamezne slikovne pike. Namen tega dela je predstaviti sliko na čim bolj razumljiv način. Po malo iskanja hitrostno primernega načina, sem se odločil za bilinearno interpolacijo, saj za razliko od bikubične interpolacije, ki potrebuje 16 vrednosti, izračuna le iz štirih. Na rezultatu to lahko opazimo kot slabše prehode med barvami.

Izračunal sem kje se mora slika pojaviti, da bo na sredini ekrana in ne razpotegnjena čez celotnega. Nato sem z dvojno for zanko šel čez vsako slikovno piko, ter določil katera vrednost bi ji pripadala glede na matriko. Za tem sem lahko pridobil še tri potrebne vrednosti matrike in sicer, vrednost na njeni desni, spodnjo ter desno dol (diagonalno). Prav tako sem izračunal faktor oziroma razmerje oddaljenosti slikovne pike od skrajnih točk matrike. Oba faktorja in vse 4 pridobljene vrednosti matrike sem uporabil v funkciji *bilinearInterpolation*, ki izračuna končno barvo slikovne pike.

```
    uint32_t x_size, y_size;
    BSP_LCD_GetXSize(0, &x_size);
    BSP_LCD_GetYSize(0, &y_size);



    // Determine the scaling factors and offsets to maintain aspect ratio
    float scale_x, scale_y, offset_x, offset_y;

      // Screen is wider than the matrix
      scale_y = (float)y_size / (float)MATRIX_SIZE_Y;
      scale_x = scale_y;
      offset_x = (x_size - (MATRIX_SIZE_X * scale_x)) / 2;
      offset_y = 0;

    // Render each pixel on the LCD (image pixel)
    for (uint32_t py = 0; py < y_size; py++) {
        for (uint32_t px = offset_x; px < x_size - offset_x; px++) {
            // Calculate the grid cell the pixel falls into, adjusted by offset
            int cell_x = (int)((px - offset_x) / scale_x);
            int cell_y = (int)((py - offset_y) / scale_y);


            // Calculate the interpolation factors
            float tx = ((px - offset_x) / scale_x) - cell_x;
            float ty = ((py - offset_y) / scale_y) - cell_y;

            // Get the colors of the four surrounding cells
            uint32_t top_left = matrix[cell_y][cell_x];
            uint32_t top_right = (cell_x < MATRIX_SIZE_X - 1) ? matrix[cell_y][cell_x +
1] : top_left;
            uint32_t bottom_left = (cell_y < MATRIX_SIZE_Y - 1) ? matrix[cell_y +
1][cell_x] : top_left;
            uint32_t bottom_right = top_left;//bottom right has more exceptions
            if (cell_x < MATRIX_SIZE_X - 1 && cell_y < MATRIX_SIZE_Y - 1) {
             bottom_right = matrix[cell_y + 1][cell_x + 1];

            } else if (cell_x >= MATRIX_SIZE_X - 1 && cell_y < MATRIX_SIZE_Y - 1) {
             bottom_right = matrix[cell_y + 1][cell_x];

            } else if (cell_x < MATRIX_SIZE_X - 1 && cell_y >= MATRIX_SIZE_Y - 1) {
             bottom_right = matrix[cell_y][cell_x + 1];
            }


            // Interpolate the color
            uint32_t color = bilinearInterpolate(top_left, top_right, bottom_left,
bottom_right, tx, ty);

            // Set the pixel
            UTIL_LCD_SetPixel(px, py, color);

        }
    }
}
```

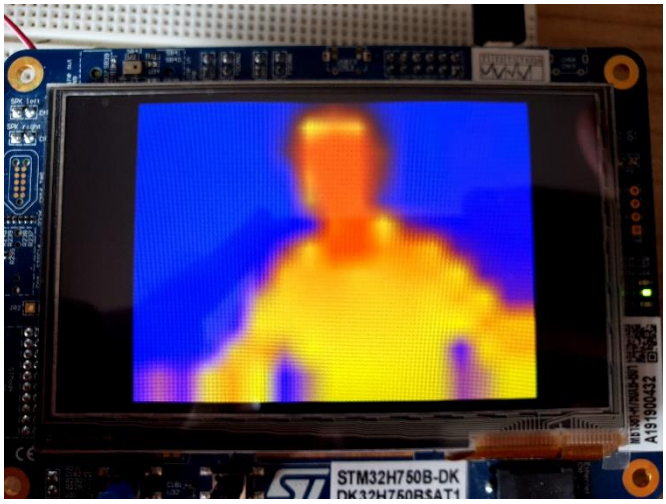
bilinearInterpolation vzame po dve vrednosti matriki naenkrat in izračuna njuni sredinski vrednosti glede na podan faktor. To torej stori trikrat da pride do povrečja vseh štirih vrednosti in končen rezultat vrne.

```
// Helper function to interpolate between two colors
uint32_t interpolateColor(uint32_t c1, uint32_t c2, float t) {
    uint8_t r1 = (c1 >> 16) & 0xFF, g1 = (c1 >> 8) & 0xFF, b1 = c1 & 0xFF;
    uint8_t r2 = (c2 >> 16) & 0xFF, g2 = (c2 >> 8) & 0xFF, b2 = c2 & 0xFF;

    uint8_t r = (uint8_t)((1 - t) * r1 + t * r2);
    uint8_t g = (uint8_t)((1 - t) * g1 + t * g2);
    uint8_t b = (uint8_t)((1 - t) * b1 + t * b2);

    return (0xFF << 24) | (r << 16) | (g << 8) | b;
}



// Bilinear interpolation of colors in a 2x2 grid
uint32_t bilinearInterpolate(uint32_t tl, uint32_t tr, uint32_t bl, uint32_t br, float
tx, float ty) {
    uint32_t top = interpolateColor(tl, tr, tx);
    uint32_t bottom = interpolateColor(bl, br, tx);
    return interpolateColor(top, bottom, ty);
}
```

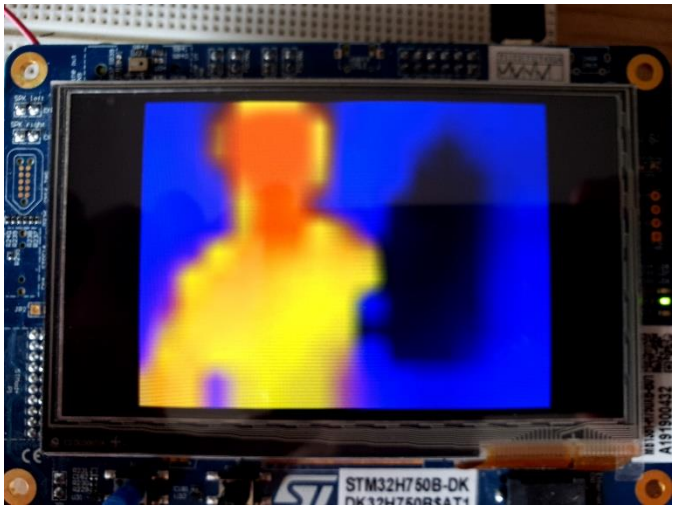
5 Končni rezultat

Končni rezultat lahko prikazuje temperature med 0 in 50°c.

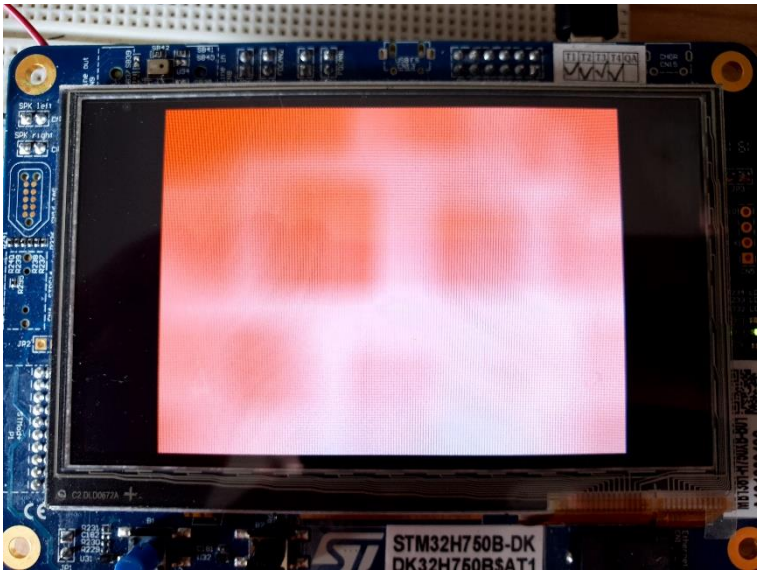
Na sliki 10 sem samo jaz, medtem ko na 11. držim še zmrznjeno plastenko. Na zadnji sliki sem posnel tipkovnico mojega prenosnega računalnika, ki se pod veliko obremenitvijo segreje do skoraj 50°c.



Slika 10: Končna slika - človek



Slika 9: Končna slika - človek z zmrznjeno plastenko



Slika 11: Končna slika - vroča tipkovnica

6 Zaključek

S končnim rezultatom sem zadovoljen. Tekom projekta sem se zagotovo naučil več stvari, kot sem pričakoval, tudi nekaj takih ki jih morda v projektu morda nisem potreboval. Sam izdelek ima še veliko potenciala in možnosti za tako izboljšave kot razširitve.

7 Viri

- https://unilj.sharepoint.com/sites/LAPSYEmbeddedAcademy/Shared%20Documents/Forms/AllItems.aspx?ga=1&id=%2Fsites%2FLAPSYEmbeddedAcademy%2FShared%20Documents%2FDokumentacije%2C%20gradiva%2FTipala%2FSmarteh%20LIR2%20Tipalo
- https://www.st.com/resource/en/schematic_pack/mb1381-h750xb-b01-schematic.pdf
- https://www.st.com/resource/en/technical_note/tn1238-stmod-interface-specification-stmicroelectronics.pdf
- https://www.industrialshields.com/blog/raspberry-pi-for-industry-26/how-to-communicate-raspberry-pi-4-b-with-a-max485-module-137
- https://controllerstech.com/stm32-reads-holding-and-input-registers/
- https://en.wikipedia.org/wiki/Bilinear_interpolation