Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Tehnična dokumentacija projekta

NRP

Aleks Bedenik

Žetale, 14. 1. 2023

1. Uvod

Ta tehnična dokumentacija razloži kako lahko z uporabo STM32IDE in X-CUBE-AI razvijemo program, ki lahko razpozna aktivnost osebe z uporabo podatkov iz STM32F3Discovery merilnika pospeška.
Prav tako opiše kako narediti in natrenirati svoj dataset model.

2. Potrebe

Projekt potrebuje naslednje:
-STM32F3Discovery mikrokrmilnik
-STM32IDE in STM32CUBE-MX
-Knjižnico X-CUBE-AI
-Python
-Poljuben IDE
-Dataset pospeška

3. Potek izdelave

3.1 Kreacija dataseta in treniranje modela

Za kreacijo dataseta, ki nam bo omogočil prepoznavanje potrebujemo vrednosti X,Y,Z iz akselometra. Te vrednosti lahko izmerimo sami, ali pa prenesemo že ustvarjen dataset. V projektu bomo še iz teh vrednosti izračunali povprečno hitrost hoje, ki nam doda možnost prepoznave osebe, s pomočjo te skripte:

```python
# import the necessary libraries
import csv
import numpy as np

# open the CSV file
with open('dataset1.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)

# create an empty list to store the data
    data = []

# read each row of the CSV file
    for row in reader:
        data.append(row)

# convert the data to a numpy array
    data = np.array(data)

# extract the x, y, and z values from the data
    x = data[:,0].astype(float)
    y = data[:,1].astype(float)
    z = data[:,2].astype(float)
    x1 = data[1:,0].astype(float)
    y1 = data[1:,1].astype(float)
    z1 = data[1:,2].astype(float)
    x1 = np.resize(x1,(1000,))
    y1 = np.resize(y1,(1000,))
    z1 = np.resize(z1,(1000,))
    #print("x",x,"x1",x1,"\n")
# calculate the speed from the x, y, and z values
    speed = (((x1) - (x)) + ((y1) - (y)) + ((z1) - (z))) / 0.5

# add the speed column to the CSV file
    data = np.column_stack((data, speed))

# write the data back to a new CSV file
    with open('dataset_with_speed.csv', 'w') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerows(data)
```

Podatke bomo shranili v csv datoteke, iz katerih bomo potem ustvarili učni model. Podatki v csv zgledajo tako(x,y,z,walking_speed):

dataset_with_speed.csv

```
-16,   -12,   1036,8.0
-17,   -12,   1041,-2.0
-16,   -12,   1039,4.0
-17,   -12,   1042,0.0
-16,   -12,   1041,-2.0
-17,   -12,   1041,-6.0
-17,   -12,   1038,6.0
-17,   -12,   1041,0.0
-16,   -12,   1040,2.0
-16,   -12,   1041,-2.0
```

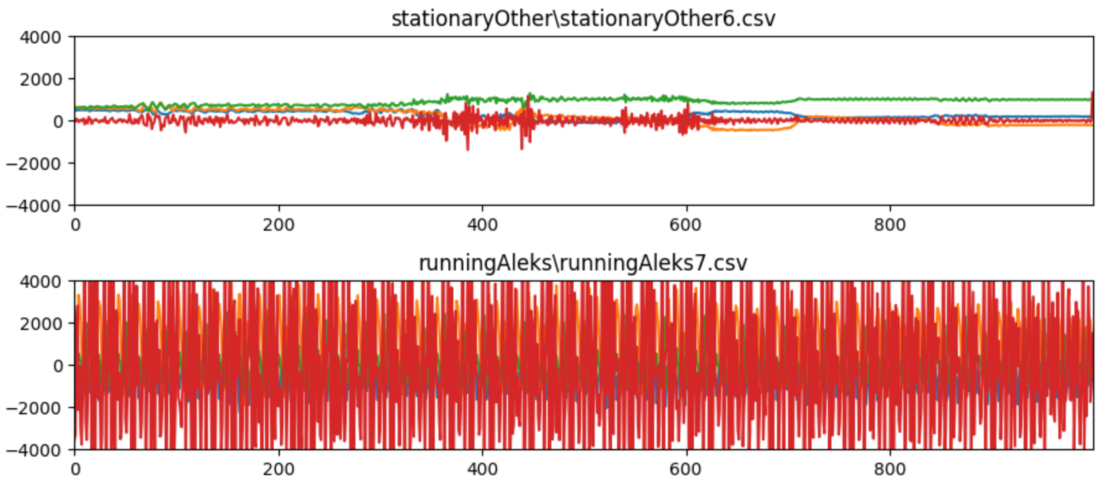
Ko smo zadovoljni z datasetom je čas da kreiramo svoj natreniran model, to lahko storimo s to python skripto:

```python
import glob
import numpy as np
import random
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Load data into memory
labels = ['stationaryAleks', 'walkingAleks', 'runningAleks','stationaryOther', 'walkingOther', 'runningOther']
x_recordings = []
y_recordings = []
recordings_filenames = []
for i, label in enumerate(labels):
    filenames = glob.glob('dataset/' + label + '/*.csv')
    for filename in filenames:
        data = np.loadtxt(filename, delimiter=',')
        x_recordings.append(data)
        y_recordings.append(i)
        recordings_filenames.append(filename)

x_recordings = np.array(x_recordings).reshape(len(x_recordings), -1, 4)
y_recordings = np.array(y_recordings)

print(x_recordings.shape)
print(y_recordings.shape)


# Plot some captures
random.seed(10)
unique_rands = random.sample(range(len(x_recordings)), 10)
plt.figure(figsize=(18, 10))
for i, n in enumerate(unique_rands):
    plt.subplot(5, 2, i + 1)
    plt.margins(x=0, y=-0.25)
    plt.plot(x_recordings[n])
    plt.ylim(-4000, 4000)  # 4000 mg acc. range
    plt.title(recordings_filenames[n].split('/')[-1])
plt.tight_layout()
plt.show()
import numpy as np
```

```python
def frame(x, frame_len, hop_len):
    '''Slice a 3D data array into (overlapping) frames.

    Example
    --------
    >>> x = np.array([[0, 1, 2],
                      [10, 11, 12],
                      [20, 21, 22],
                      [30, 31, 32],
                      [40, 41, 42],
                      [50, 51, 52],
                      [60, 61, 62]])
    >>> frames = x.frame(x, 3, 2)
    >>> x.shape
    (7, 3)
    >>> frames.shape
    (3, 3, 3)
    '''

    assert(x.shape == (len(x), 4))
    assert(x.shape[0] >= frame_len)
    assert(hop_len >= 1)

    n_frames = 1 + (x.shape[0] - frame_len) // hop_len
    shape = (n_frames, frame_len, x.shape[1])
    strides = ((hop_len * x.strides[0],) + x.strides)
    return np.lib.stride_tricks.as_strided(x, shape=shape, strides=strides)

x_frames = []
y_frames = []
for i in range(x_recordings.shape[0]):
    # frames = frame(x_recordings[i], 26, 26) # no overlap
    frames = frame(x_recordings[i], 26, 13) # 50% overlap
    x_frames.append(frames)
    y_frames.append(np.full(frames.shape[0], y_recordings[i]))

print(np.array(x_frames).shape)
x_frames = np.concatenate(x_frames)
y_frames = np.concatenate(y_frames)
print(x_frames.shape)

# Each output label is an integer between 0 and 5:
print(y_frames.shape)
print(labels)

x_frames_normed = x_frames / 4000
```

```
x_train, x_test, y_train, y_test = train_test_split(
    x_frames_normed, y_frames, test_size=0.25)

print("Trainning samples:", x_train.shape)
print("Testing samples:", x_test.shape)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=16, kernel_size=4, activation='relu', input_shape=(26, 4)),
  tf.keras.layers.Conv1D(filters=8, kernel_size=4, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(6, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=30)
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

print("Test loss:", test_loss)
print("Test acc:", test_acc)
model.summary()
Y_pred = model.predict(x_test)
y_pred = np.argmax(Y_pred, axis=1)
confusion_matrix = tf.math.confusion_matrix(y_test, y_pred)

plt.figure()
sns.heatmap(confusion_matrix,
            annot=True,
            xticklabels=labels,
            yticklabels=labels,
            cmap=plt.cm.Blues,
            fmt='d', cbar=False)
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
model.save('model.h5')
```

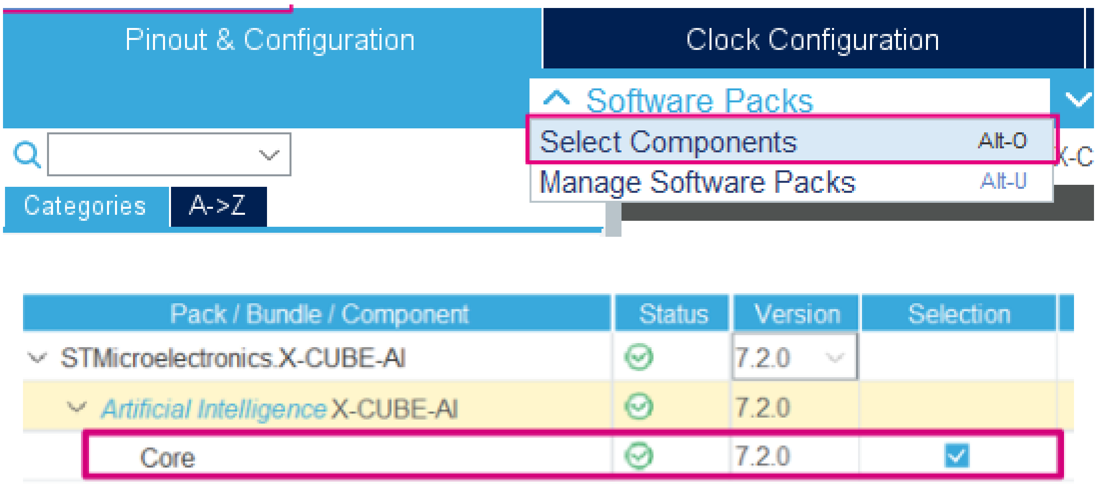
Če zaženemo skripto imamo tudi vizualizacijo naše vrednosti v csv datotekah:



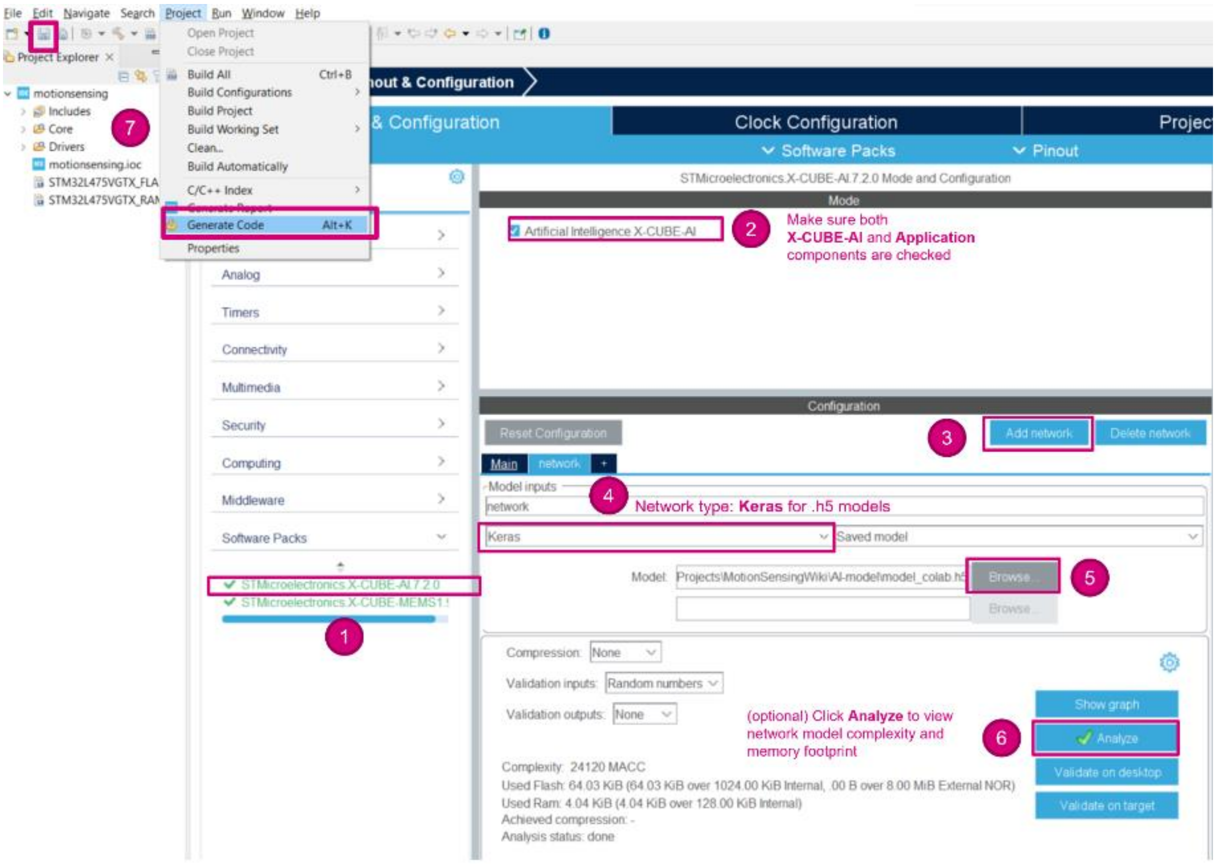
Ko se skripta zaključi dobimo natreniran model 'model.h5'.

3.2 Delo v STM32IDE in STM32-CUBE-MX

Prvo ustvarimo nov projekt v STM32-CUBE-MX za našo ploščico STM32F3Discovery. Omogočimo I2C in nastavimo na Fast mode, za branje vrednosti iz akselometra. Omogočimo USB in nastavimo USB_DEVICE na CDC, za prikaz rezultatov na PC-ju.
Za dodajanje STM32CUBE.AI gremo pod Software Packs->SelectComponents->X-CUBE-AI in omogočimo Core.



Ko dodamo CUBE.AI še moramo vključiti naš model:



STM32 nam analizira model in nam ustvari potrebne knjižnice. Dobimo jih z generiranjem kode.

Ko smo to storili se lahko lotimo glavnega programa.

1. Vključimo knjižnice

```c
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "usb_device.h"
#include "ai_platform.h"    // AI
#include "network.h"        // AI
#include "network_data.h"   // AI
#include "lsm303dlhc.h"
#include <time.h>
```

2. Deklariramo spremenljivke za AI

```c
ai_handle network;
float aiInData[AI_NETWORK_IN_1_SIZE];
float aiOutData[AI_NETWORK_OUT_1_SIZE];
ai_u8 activations[AI_NETWORK_DATA_ACTIVATIONS_SIZE];
const char* activities[AI_NETWORK_OUT_1_SIZE] = {
  "stationaryAleks", "walkingAleks", "runningAleks","stationaryOther", "walkingOther", "runningOther"
};
ai_buffer * ai_input;
ai_buffer * ai_output;
```

3. Dodamo AI bootstrap funkcije

```c
static void AI_Init(void);
static void AI_Run(float *pIn, float *pOut);
static uint32_t argmax(const float * values, uint32_t len);
```

4. Definiramo funkcije

```c
/* USER CODE BEGIN 4 */
static void AI_Init(void)
{
  ai_error err;
  char uart_buf[50];
  int uart_buf_len;
  /* Create a local array with the addresses of the activations buffers */
  const ai_handle act_addr[] = { activations };
  /* Create an instance of the model */
  err = ai_network_create_and_init(&network, act_addr, NULL);
  if (err.type != AI_ERROR_NONE) {
          uart_buf_len = sprintf(uart_buf, "ai_network_create error - type=%d code=%d\r\n", err.type, err.code);
          CDC_Transmit_FS((uint8_t *)uart_buf, uart_buf_len);
    printf("ai_network_create error - type=%d code=%d\r\n", err.type, err.code);
    Error_Handler();
  }
  ai_input = ai_network_inputs_get(network, NULL);
  ai_output = ai_network_outputs_get(network, NULL);
}
static void AI_Run(float *pIn, float *pOut)
{
  ai_i32 batch;
  ai_error err;
  char uart_buf[50];
  int uart_buf_len;
  /* Update IO handlers with the data payload */
  ai_input[0].data = AI_HANDLE_PTR(pIn);
  ai_output[0].data = AI_HANDLE_PTR(pOut);

  batch = ai_network_run(network, ai_input, ai_output);
  if (batch != 1) {
    err = ai_network_get_error(network);
    printf("AI ai_network_run error - type=%d code=%d\r\n", err.type, err.code);
    uart_buf_len = sprintf(uart_buf, "AI ai_network_run error - type=%d code=%d\r\n", err.type, err.code);
    CDC_Transmit_FS((uint8_t *)uart_buf, uart_buf_len);
    Error_Handler();
  }
}
static uint32_t argmax(const float * values, uint32_t len)
{
  float max_value = values[0];
  uint32_t max_index = 0;
  for (uint32_t i = 1; i < len; i++) {
    if (values[i] > max_value) {
      max_value = values[i];
      max_index = i;
    }
  }
  return max_index;
}
/* USER CODE END 4 */
```

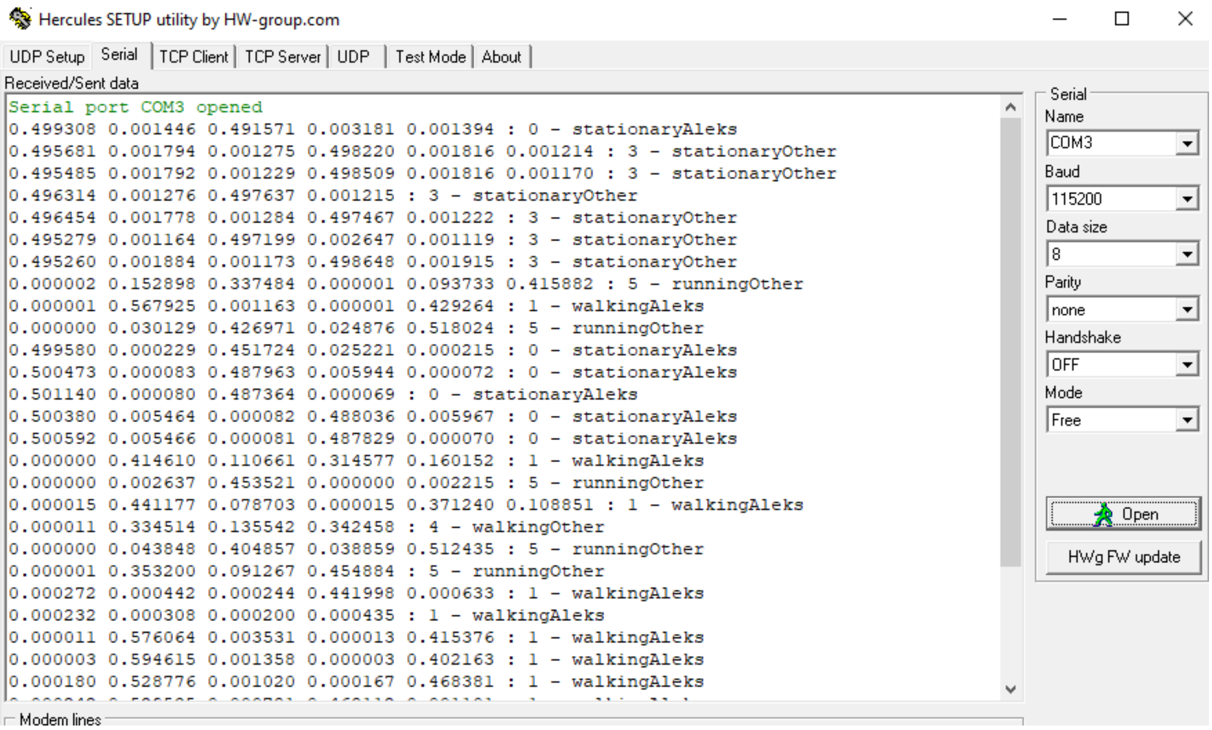
5. Potem kličemo funkcijo AI_INIT(), da inicializiramo omrežje in v neskončni zanki vključimo kodo za pridobivanje akselometer vrednosti in izpis rezultatov omrežja.

```c
    {
        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)){
            start = 1;
            if(switchF == 1 && holdCheck == 0) {
                isPressed = isPressed+ 1;
                if(isPressed == 4)isPressed=0;
                switchF = 0;

            }
            }else{
                holdCheck = 0;
                switchF=1;
            }
        if(start == 1){
        if(isPressed == 0 && holdCheck == 0){
            i2cBinary();
        }
        else if(isPressed == 1 && holdCheck == 0){
        }
        else if(isPressed == 2 && holdCheck == 0){
            xFNext = xF;
            yFNext = yF;
            zFNext = zF;
            i2c1_beriRegistre(0x19,0x28,(uint8_t*)&meritev[1],6);
            i2c1_beriRegistre(0x19,0x29,(uint8_t*)&meritev[2],6);
            i2c1_beriRegistre(0x19,0x2a,(uint8_t*)&meritev[3],6);
            i2c1_beriRegistre(0x19,0x2b,(uint8_t*)&meritev[4],6);
            i2c1_beriRegistre(0x19,0x2c,(uint8_t*)&meritev[5],6);
            i2c1_beriRegistre(0x19,0x2d,(uint8_t*)&meritev[6],6);

            x = meritev[2] << 8 | meritev[1];
            y = meritev[4] << 8 | meritev[3];
            z = meritev[6] << 8 | meritev[5];
            xF = x/9.8;
            yF = y/9.8;
            zF = 200+(z/9.8);
            walkingSpeed = (xF-xFNext + yF-yFNext + zF-zFNext) / 0.5;

            aiInData[write_index + 0] = (float) xF/ 4000.0f;
            aiInData[write_index + 1] = (float) yF / 4000.0f;
            aiInData[write_index + 2] = (float) zF / 4000.0f;
            aiInData[write_index + 3] = (float)walkingSpeed/ 4000.0f;
            write_index += 4;

              if (write_index == AI_NETWORK_IN_1_SIZE) {
                write_index = 0;

                printf("Running inference\r\n");
                AI_Run(aiInData, aiOutData);

                /* Output results */
                for (uint32_t i = 0; i < AI_NETWORK_OUT_1_SIZE; i++) {
                    printf("%8.6f ", aiOutData[i]);
```

```c
                  uart_buf_len = sprintf(uart_buf, "%8.6f ", aiOutData[i]);
                  CDC_Transmit_FS((uint8_t *)uart_buf, uart_buf_len);
              }
              uint32_t class = argmax(aiOutData, AI_NETWORK_OUT_1_SIZE);
              HAL_Delay(100);
              uart_buf_len = sprintf(uart_buf, ": %d - %s\r\n", (int) class, activities[class]);
              CDC_Transmit_FS((uint8_t *)uart_buf, uart_buf_len);

            }
              HAL_Delay(50);
        }
        else if(isPressed == 3 && holdCheck == 0){
            }
        }
```

4. Testiranje

Če naložimo kodo na mikrokrmilnik in mikrokrmilnik povežemo z PC preko USB kabla, lahko ne terminalu vidimo da prepozna aktivnosti če probamo premikati ploščico po različnih hitrostih.



Model je dober pri prepoznavi aktivnosti ampak ni preveč dober pri prepoznavi osebe, saj bi za to morali dodati mnogo več podatkov, ki bi potem prepoznale osebo, ampak koncept izdelave je enaki.

5. Zaključek

Ta dokumentacija opisuje kako se lotiti izdelave programa za mikrokrmilnik STM32F3Discovery, ki prepozna aktivnosti in osebe, ki opravljajo aktivnost. Projekt potrebuje ploščico STM32F3Discovery, STM32IDE, STM32CUBEMX, Python.
Za boljše natančnost prepoznave osebe bi morali dodati datasetu več informacij o osebi.