0.1 Sensor Configuration

HTPA32x32d Library

```
[ ]: from periphery import I2C
import time
import numpy as np
import copy
import struct

class HTPA:


        def __init__(self, address):
                self.address = address
                self.i2c = I2C("/dev/i2c-1")
                print("Grabbing EEPROM data")
                eeprom = self.get_eeprom()
                self.extract_eeprom_parameters(eeprom)
                self.eeprom = eeprom
                wakeup_and_blind = self.generate_command(0x01, 0x01)   # wake up␣
 ↪the device
                # set ADC resolution to 16 bits
                adc_res = self.generate_command(0x03, self.mbit_value)# set ADC␣
 ↪resolution in eeprom
                pull_ups = self.generate_command(0x09, self.pu_value)#pu value␣
 ↪in eeprom

                print("Initializing capture settings")

                self.send_command(wakeup_and_blind)
                self.send_command(adc_res)
                self.send_command(pull_ups)

                self.set_bias_current(self.bias_value)   # bias value on eeprom
                self.set_clock_speed(0x050)   # clk value on eeprom self.clk_value
                self.set_cm_current(self.bpa_value)   # BPA value in eeprom

                # initialize offset to zero
                self.offset = np.zeros((32, 32))



        def get_eeprom(self, eeprom_address=0x50):#Talking EEPROM
                query = [I2C.Message([0x00, 0x00]), I2C.Message(
                    [0x00]*8000, read=True)]   # 8 Kbit Data from EEPROM
                self.i2c.transfer(eeprom_address, query)
                return np.array(query[1].data)
```

```
    def extract_eeprom_parameters(self, eeprom):#EEPROM Data
        self.VddCompgrad = eeprom[0x0340:0x0540:2] + (eeprom[0x0341:
↪0x0540:2] << 8)
        self.VddCompoff = eeprom[0x0540:0x0740:2] + (eeprom[0x0541:
↪0x0740:2] << 8)

        ThGrad = eeprom[0x0740:0x0F40:2] + (eeprom[0x0741:0x0F40:2] << 8)
        ThGrad = [tg - 65536 if tg >= 32768 else tg for tg in ThGrad]
        ThGrad = np.reshape(ThGrad, (32, 32))
        ThGrad[16:, :] = np.flipud(ThGrad[16:, :])
        self.ThGrad = ThGrad

        ThOffset = eeprom[0x0F40:0x1740:2] + (eeprom[0x0F41:0x1740:2] <<␣
↪8)

        ThOffset = np.reshape(ThOffset, (32, 32))
        ThOffset[16:, :] = np.flipud(ThOffset[16:, :])
        self.ThOffset = ThOffset

        P = eeprom[0x1740::2] + (eeprom[0x1741::2] << 8)
        P = np.reshape(P, (32, 32))
        P[16:, :] = np.flipud(P[16:, :])
        self.P = P

        epsilon = float(eeprom[0x000D])
        GlobalGain = eeprom[0x0055] + (eeprom[0x0056] << 8)
        Pmin = eeprom[0x0000:0x0004]
        Pmax = eeprom[0x0004:0x0008]
        Pmin = struct.unpack('f', reduce(
            lambda a, b: a+b, [chr(p) for p in Pmin]))[0]
        Pmax = struct.unpack('f', reduce(
            lambda a, b: a+b, [chr(p) for p in Pmax]))[0]
        self.PixC = (P * (Pmax - Pmin) / 65535. + Pmin) * \
                    (epsilon / 100) * float(GlobalGain) / 100
        self.gradScale = eeprom[0x0008]
        self.VddCalib1 = eeprom[0x0046] + (eeprom[0x0047] << 8)
        self.VddCalib = eeprom[0x0046] + (eeprom[0x0047] << 8)
        self.VddCalib2 = eeprom[0x0048] + (eeprom[0x0049] << 8)
        self.Vdd = 3000.0
        self.VddScaling = eeprom[0x004E]
        self.Vddoff = eeprom[0x004F]

        self.PtatCalib1 = eeprom[0x003C] + (eeprom[0x003D] << 8)
        self.PtatCalib2 = eeprom[0x003E] + (eeprom[0x003F] << 8)
        PTATgradient = eeprom[0x0034:0x0038]
        self.PTATgradient = struct.unpack('f', reduce(
```

```
                lambda a, b: a+b, [chr(p) for p in PTATgradient]))[0]
            PTAToffset = eeprom[0x0038:0x003c]
            self.PTAToffset = struct.unpack('f', reduce(
                lambda a, b: a+b, [chr(p) for p in PTAToffset]))[0]
            self.clk_value = eeprom[0x001C]
            self.bias_value = eeprom[0x001B]
            self.pu_value = eeprom[0x001E]
            self.mbit_value = eeprom[0x001A]
            self.bpa_value = eeprom[0x001D]
            self.subt = np.zeros((32, 32))


    def set_clock_speed(self, clk):#set clock speed
            if clk > 63:  # Max 64 Hz
                    clk = 63
            if clk < 0:
                    clk = 0
            clk = int(clk)
            print(clk)
            # The measure time depends on the clock frequency settings.
↪(optimal value)
            clk_speed = self.generate_command(0x06, clk)
            self.send_command(clk_speed)  # send clock data

    # This setting is used to adjust the common mode voltage of the␣
↪preamplifier.


    def set_cm_current(self, cm):
            cm = int(cm)
            cm_top = self.generate_command(0x07, cm)
            cm_bottom = self.generate_command(0x08, cm)

            self.send_command(cm_top)
            self.send_command(cm_bottom)


    def set_bias_current(self, bias):
            bias = int(bias)
            # This setting is used to adjust the bias current of the ADC. A␣
↪faster clock frequency requires a higher bias current setting.
            bias_top = self.generate_command(0x04, bias)
            # This setting is used to adjust the bias current of the ADC. A␣
↪faster clock frequency requires a higher bias current setting.
            bias_bottom = self.generate_command(0x05, bias)
            self.send_command(bias_top)  # send bias top data
            self.send_command(bias_bottom)  # send bias bottom data
```

```
def temperature_compensation(self, im, ptat):#Thermal Offset Calculate
        comp = np.zeros((32,32))
        Ta = np.mean(ptat) * self.PTATgradient + self.PTAToffset
            #     temperature compensated voltage
        comp = ((self.ThGrad * Ta) / pow(2, self.gradScale)) + self.ThOffset
        Vcomp = np.reshape(im,(32, 32)) - comp
        return Vcomp

    def offset_compensation(self, im):#general environment offset send␣
↪offset data
            return im-self.offset

    def sensitivity_compensation(self, im):#object temperature
            return (im*100000000)/self.PixC

    def measure_observed_offset(self):#Measuring observed offsets
            mean_offset = np.zeros((32, 32))
            for i in range(10):
                    print("    frame " + str(i))
                    (p, pt) = self.capture_image()
                    im = self.temperature_compensation(p, pt)
                    mean_offset += (im-10)/10.0
            self.offset = mean_offset


    def Vdd_Comperasition(self,im,ptat):#Vdd Comperasition calculate
            VVddComp=[]
            for i in range(16):
                    for j in range(32):
                            VVddComp.append((((self.
↪VddCompgrad[(j+i*32)%128]*np.mean(ptat))/pow(2, self.VddScaling)+self.
↪VddCompoff[(j+i*32)%128])/pow(2, self.Vddoff))*(self.Vdd-self.VddCalib1-((self.
↪VddCalib2-self.VddCalib1)/(self.PtatCalib2-self.PtatCalib1))*(np.
↪mean(ptat)-self.PtatCalib1)))
            for i in range(16,32):
                    for j in range(32):
                            VVddComp.append((((self.
↪VddCompgrad[(j+i*32)%128+128]*np.mean(ptat))/pow(2, self.VddScaling)+self.
↪VddCompoff[(j+i*32)%128+128])/pow(2, self.Vddoff))*(self.Vdd-self.
↪VddCalib1-((self.VddCalib2-self.VddCalib1)/(self.PtatCalib2-self.
↪PtatCalib1))*(np.mean(ptat)-self.PtatCalib1)))
            self.VVddComp=VVddComp
            return im-np.reshape(self.VVddComp,(32, 32))
```

```
    def measure_electrical_offset(self, blind=True): #measure_electrical_offset
            pixel_values = np.zeros(256)
            ptats = np.zeros(8)

    self.send_command(self.generate_expose_block_command(0, blind=blind), wait=False)

            query = [I2C.Message([0x02]), I2C.Message([0x00], read=True)]

            read_block = [I2C.Message([0x0A]), I2C.Message([0x00]*258, read=True)]
            self.i2c.transfer(self.address, read_block)
            top_data = np.array(copy.copy(read_block[1].data))

            read_block = [I2C.Message([0x0B]), I2C.Message([0x00]*258, read=True)]
            self.i2c.transfer(self.address, read_block)
            bottom_data = np.array(copy.copy(read_block[1].data))

            top_data = top_data[1::2] + (top_data[0::2] << 8)
            bottom_data = bottom_data[1::2] + (bottom_data[0::2] << 8)
    # bottom data is in a weird shape
            pixel_values[0:128] = top_data[1:]
            # bottom data is in a weird shape
            pixel_values[224:256] = bottom_data[1:33]
            pixel_values[192:224] = bottom_data[33:65]
            pixel_values[160:192] = bottom_data[65:97]
            pixel_values[128:160] = bottom_data[97:]
            ptats[block] = top_data[0]
            ptats[7-block] = bottom_data[0]

    self.elloff=pixel_values;


    def electrical_offset(self,im):#electrical offset calculate
    V_new = np.zeros((32,32))
        for i in range(16):
            for j in range(32):
            V_new[i,j]=self.elloff[(j+i*32)%128]
        for i in range(16,32):
            for j in range(32):
            V_new[i,j]=self.elloff[(j+i*32)%128+128]
    self.V_new=V_new
            return im - self.V_new
    def capture_image(self, blind=False):
            pixel_values = np.zeros(1024)
```

```
                ptats = np.zeros(8)

                for block in range(4):
                        print("Exposing block " + str(block))
                        self.send_command(self.
↪generate_expose_block_command(block, blind=blind), wait=False)

                        query = [I2C.Message([0x02]), I2C.Message([0x00],␣
↪read=True)]
                        expected = 1 + (block << 4)

                        done = False

                        while not done:
                                self.i2c.transfer(self.address, query)

                                if not (query[1].data[0] == expected):
                                else:
                                        done = True

                        read_block = [I2C.Message([0x0A]), I2C.
↪Message([0x00]*258, read=True)]
                        self.i2c.transfer(self.address, read_block)
                        top_data = np.array(copy.copy(read_block[1].data))

                        read_block = [I2C.Message([0x0B]), I2C.
↪Message([0x00]*258, read=True)]
                        self.i2c.transfer(self.address, read_block)
                        bottom_data = np.array(copy.copy(read_block[1].data))

                        top_data = top_data[1::2] + (top_data[0::2] << 8)
                        bottom_data = bottom_data[1::2] + (bottom_data[0::2] <<␣
↪8)

                        pixel_values[(0+block*128):(128+block*128)] = top_data[1:
↪]

                        # bottom data is in a weird shape
                        pixel_values[(992-block*128):(1024-block*128)] =␣
↪bottom_data[1:33]
                        pixel_values[(960-block*128):(992-block*128)] =␣
↪bottom_data[33:65]
                        pixel_values[(928-block*128):(960-block*128)] =␣
↪bottom_data[65:97]
                        pixel_values[(896-block*128):(928-block*128)] =␣
↪bottom_data[97:]
```

```
                        ptats[block] = top_data[0]
                        ptats[7-block] = bottom_data[0]

                pixel_values = np.reshape(pixel_values, (32, 32))

                return (pixel_values, ptats)


        def generate_command(self, register, value):#periphery library register␣
↪activate
                return [I2C.Message([register, value])]


        def generate_expose_block_command(self, block, blind=False):#read data␣
↪command
                if blind:
                        return self.generate_command(0x01, 0x0B)
                else:
                        return self.generate_command(0x01, 0x09 + (block << 4))


        def send_command(self, cmd, wait=True):#send data to registers
                self.i2c.transfer(self.address, cmd)
                if wait:
                        time.sleep(0.005) # sleep for 5 ms


        def close(self):#closed device
                sleep = self.generate_command(0x01, 0x00)
                self.send_command(sleep)
```

Capture Image

[ ]:
```
import numpy as np
import cv2
from htpa import *
import pickle
i = 0
k = 0
dev = HTPA(0x1A)

while(True):
    if (i == 5):
    dev.measure_observed_offset()
    dev.measure_electrical_offset()

    pixel_values, ptats) = dev.capture_image()   # Capture Image
```

```
    im = dev.temperature_compensation(pixel_values, ptats)  # thermal offset
    im = dev.offset_compensation(im)  # general offset
    if(k>5):
    im=dev.electrical_offset(im)#electrical offset
    im=dev.Vdd_Comperasition()#Vdd Comperasition
    im = dev.sensitivity_compensation(im)#Sensitivity

    # resize and scale image to make it more viewable on raspberry pi screen
    im = cv2.resize(im, None, fx=12, fy=12)
    im -= np.min(im)
    im /= np.max(im)
    imcolor=cv2.applyColorMap(im,cv2.COLORMAP_JET)

    cv2.imshow('frame', im)
    cv2.imshow('frame1', imcolor)

    i += 1

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

dev.close()

cv2.destroyAllWindows()
```

1 Creating Dataset

Resize Dataset

[ ]:
```
import cv2
import os
from PIL import Image
import numpy as np

src='/open_train/five/'
filenames_train=os.listdir(src)

print(len(filenames_train))
for f_name in filenames_train:
    im=Image.open(src+f_name)
    # Size of the image in pixels (size of orginal image)
    # (This is not mandatory)
#    width, height = im.size

    # Setting the points for cropped image
    # Setting the points for cropped image
```

```
    left = 120
    top = 45
    right = 390
    bottom = 240

    # Cropped image of above dimension
    # (It will not change orginal image)
    #im1 = im.crop((left, top, right, bottom))
    im1=im1.resize((32, 32))
    im1.save('/open_train/five_new/'+f_name)
```

Dataset Augmentation

```
[ ]: # example of images augmentation
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
import os
# Passing the path of the image directory
src='/home/rabikkk/Desktop/final_project/last_dataset/close/'
path1='/home/rabikkk/Desktop/final_project/last_dataset/train/';
filenames_train=os.listdir(src)

print(len(filenames_train))
for f_name in filenames_train:
    # load the image
    img = load_img(src+f_name)
# convert to numpy array
    data = img_to_array(img)
# expand dimension to one sample
    samples = expand_dims(data, 0)
# create image data augmentation generator
    datagen1 = ImageDataGenerator(zoom_range=[0.8,1])
# create image data augmentation generator
    datagen = ImageDataGenerator(brightness_range=[0.6,1.0])
# create image data augmentation generator
    datagen2 = ImageDataGenerator(horizontal_flip=True)
# prepare iterator
    it = datagen.flow(samples, batch_size=1)
    it1 = datagen1.flow(samples, batch_size=1)
    it2 = datagen2.flow(samples, batch_size=1)
    it = datagen.flow(samples, batch_size=4, save_to_dir=path1,␣
 ↪save_prefix='index_test03', save_format='png')
    it1 = datagen1.flow(samples, batch_size=5, save_to_dir=path1,␣
 ↪save_prefix='index_test4', save_format='png')
```

```
    it2= datagen2.flow(samples, batch_size=4, save_to_dir=path1,␣
↪save_prefix='index_test5', save_format='png')
```

2 Static Gesture Learning Model

```
[ ]: import numpy as np
import pandas as pd
import keras
from keras.preprocessing.image import ImageDataGenerator,load_img
#from keras.utils import to_categorical
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import os
import cv2
from PIL import Image,ImageOps
from numpy import asarray
```

Adding Train_set and Test_set

```
[ ]: #Training Set
src='/home/rabikkk/Desktop/final_project/last_dataset/train/'
filenames_train=os.listdir(src)

categories_train=[]
image_train=[]
print(len(filenames_train))
close=0
index=0
last=0
open1=0
for f_name in filenames_train:
    image=Image.open(src+f_name).convert('RGB')
    image=ImageOps.grayscale(image)
    #image=cv2.imread(src+f_name)
    #image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    numpydata = asarray(image)
    image_train.append(numpydata)
    #print(len(image_train))
    category=f_name.split('_')[0]
    if category=='close':
        categories_train.append(0)
        close+=1
    elif category=='index':
        categories_train.append(1)
```

```
        index+=1
    elif category=='last':
        categories_train.append(2)
        last+=1
    else:
        categories_train.append(3)
        open1+=1

df=pd.DataFrame({
    'filename':filenames_train,
    'category':categories_train
})

image_train=np.asarray(image_train)
image_train = image_train.reshape((image_train.shape[0],32, 32,1))
image_train = image_train.astype("float32") / 255.0
categories_train=np.asarray(categories_train)
categories_train=categories_train.reshape(len(filenames_train),1)

print(image_train.shape)
print(categories_train.shape)
print(close)
print(last)
print(open1)
print(index)
#print(categories_train)
```

[ ]:
```
#Test Set
src='/home/rabikkk/Desktop/final_project/last_dataset/test/'
filenames_test=os.listdir(src)

categories_test=[]
image_test=[]
close=0
index=0
last=0
open1=0
print(len(filenames_test))
for f_name in filenames_test:
    image=Image.open(src+f_name).convert('RGB')
    image=ImageOps.grayscale(image)
     #image=cv2.imread(src+f_name)
   #image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    numpydata = asarray(image)
    image_test.append(numpydata)
   # print(len(image_test))
    category=f_name.split('_')[0]
```

```
    if category=='close':
        categories_test.append(0)
        close+=1
    elif category=='index':
        categories_test.append(1)
        index+=1
    elif category=='last':
        categories_test.append(2)
        last+=1
    else:
        categories_test.append(3)
        open1+=1


df=pd.DataFrame({
    'filename':filenames_test,
    'category':categories_test
})
image_test=np.asarray(image_test)
image_test = image_test.reshape((image_test.shape[0], 32, 32,1))
image_test = image_test.astype("float32") / 255.0
#image_test =image_test.reshape(len(image_test),(32,32))
#image_test= np.ndarray(shape=(2050, 32, 32, 1))
#image_test=image_test/255.0
#image_test=image_test.reshape(2008,32,32)
categories_test=np.asarray(categories_test)
categories_test=categories_test.reshape(len(filenames_test),1)

print(image_test.shape)
print(categories_test.shape)
print(close)
print(last)
print(open1)
print(index)
```

Model

[ ]:
```
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D,AveragePooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout,LeakyReLU
from tensorflow.keras.utils import plot_model
#Instantieate an empty model
model = Sequential(name="Static_Gesture_Model")
```

```
#C1 Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',␣
 ↪input_shape=(32, 32,1)))
#S2 Pooling Layer
model.add(MaxPooling2D(strides=2))
#C3 Convolutional Layer
model.add(Conv2D(filters=48, kernel_size=(5,5), padding='valid',␣
 ↪activation='relu'))
#S4 Pooling Layer
model.add(MaxPooling2D(strides=2))
#Flatten the CNN output so that we can connect it with fully connected layers
model.add(Flatten())
#Fully Connected Layer
model.add(Dense(256, activation='tanh'))
#Fully Connected Layer
model.add(Dense(84, activation='tanh'))
#Fully Connected Layer
model.add(Dense(10, activation='tanh'))
#Output Layer with Softmax Activation
model.add(Dense(4, activation='softmax'))
model.summary()
```

Learning

[ ]:
```
import numpy as np
import keras
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
import tensorflow as tf
print("The length of list is: ", len(image_train))
print("The length of list is: ", len(image_test))

# early stopping
early_stop = EarlyStopping(patience=30, monitor='val_loss')
opt = keras.optimizers.Adam(learning_rate=0.000025)
model.compile(optimizer=opt, #'adam'
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model_name = "model"
filepath='/home/rabikkk/Desktop/final_project/learning/' + model_name + '.hdf5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,␣
 ↪save_best_only=True, mode='auto')
logpath = '/home/rabikkk/Desktop/final_project/learning' + model_name + '.log'
csv_logger = keras.callbacks.CSVLogger(logpath)
```

```
callbacks_list = [checkpoint,csv_logger]
history = model.fit(image_train, categories_train,␣
 ↪epochs=100,validation_data=(image_test, categories_test),␣
 ↪callbacks=[callbacks_list,early_stop])#callbacks=[callbacks_list,early_stop]
```

Model Learning Plot

[ ]:
```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(acc))

plt.plot(epochs, acc, label='Training acc')
plt.plot(epochs, val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.ylim(0.9,1)
plt.show()

test_loss, test_acc = model.evaluate(image_test, categories_test, verbose=2)
print(test_loss)
print(test_acc)
```

Model Learning Plot

[ ]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(epochs, loss, label='Training loss')
plt.plot(epochs, val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.ylim(0,0.5)
plt.show()
```

Testing Models

[ ]:
```
from keras.models import load_model
categories_valtest=[]
model = tf.keras.models.load_model('/home/rabikkk/Desktop/final_project/learning/
 ↪deneme.hdf5')
images = []
# Test edeceğin datanı preprocess et trainingde verdiğin input haline getir.
main_folder0 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set1/'
#main_folder1 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set2/'
#main_folder2 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set3/'
#main_folder3 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set4/'
```

```
#main_folder4 ='/home/rabikkk/Desktop/final_project/last_dataset/testing/Set5/'
for f_name in sorted(os.listdir(main_folder0)):
    #image = Image.open(main_folder + f_name)
    image = Image.open(main_folder0 + f_name)
    image=ImageOps.grayscale(image)
    #image_array = asarray(image)
    image_array=np.array(image)
    images.append(image_array)
    category=f_name.split('_')[0]
    if category=='close':
      categories_valtest.append(0)
    elif category=='index':
      categories_valtest.append(1)
    elif category=='last':
      categories_valtest.append(2)
    else:
      categories_valtest.append(3)

df=pd.DataFrame({
    'filename':images,
    'category':categories_valtest
})
categories_valtest=np.asarray(categories_valtest)
categories_valtest=categories_valtest.reshape(len(images),1)
images=np.asarray(images)
images = images.reshape((images.shape[0],32, 32,1))
images = images.astype("float32") / 255.0
print(images.shape)
yhat = model.predict([images])
#print('Predicted: %.3f' % yhat[0])
predictions = model.predict_classes(images)

print(predictions)
# summarize the first 5 cases
count=0;
for i in range(len(images)):
    if predictions[i]==categories_valtest[i]:
        count+=1
    #print('%s%d=> %d (expected %d)' % (f_name,i,predictions[i],␣
 ↪categories_valtest[i]))
print('Result:%f' % ((count/len(categories_valtest))*100))
```