Artificial Nose

Real-Time Systems, Embedded Computing Systems

Luca Belluardo e Andrea Stevanato

a.stevanato.1995@gmail.com, luca.bellu95@gmail.com

January 21, 2019

1 Introduction

In this project a real-time application is developed to recognize smells from an artificial nose. The sensor used for the application is an air quality gas sensor. The application is divided into five periodic tasks. Each task handles a specific functionality of our application.

For the recognizing the image a neural network was trained. For what concerns the neural network the Tensorflow framework was used.

2 The tasks

In our application we have 5 periodic tasks (Figure 1): graphic task, sensor task, neural network task (made with Tensorflow), keyboard task and the store image task.

The main function sets everything up for the tasks, except for the store image task. The keyboard task is in charge to activate the store image task when the `ENTER` key is pressed. If the store image task is already in execution and the `ENTER` key is pressed this it's terminated. Before start the store image task it's possible to write the name of the directory in which the images will be saved; if no name it's writed the images will be saved into *image_neural_network* directory.

The sensor is read by an Arduino M0 pro; the sampled data read by arduino are sent via the serial port to our application and read by the sensor task. All the tasks are terminated by the main when the user presses the `ESC` key.

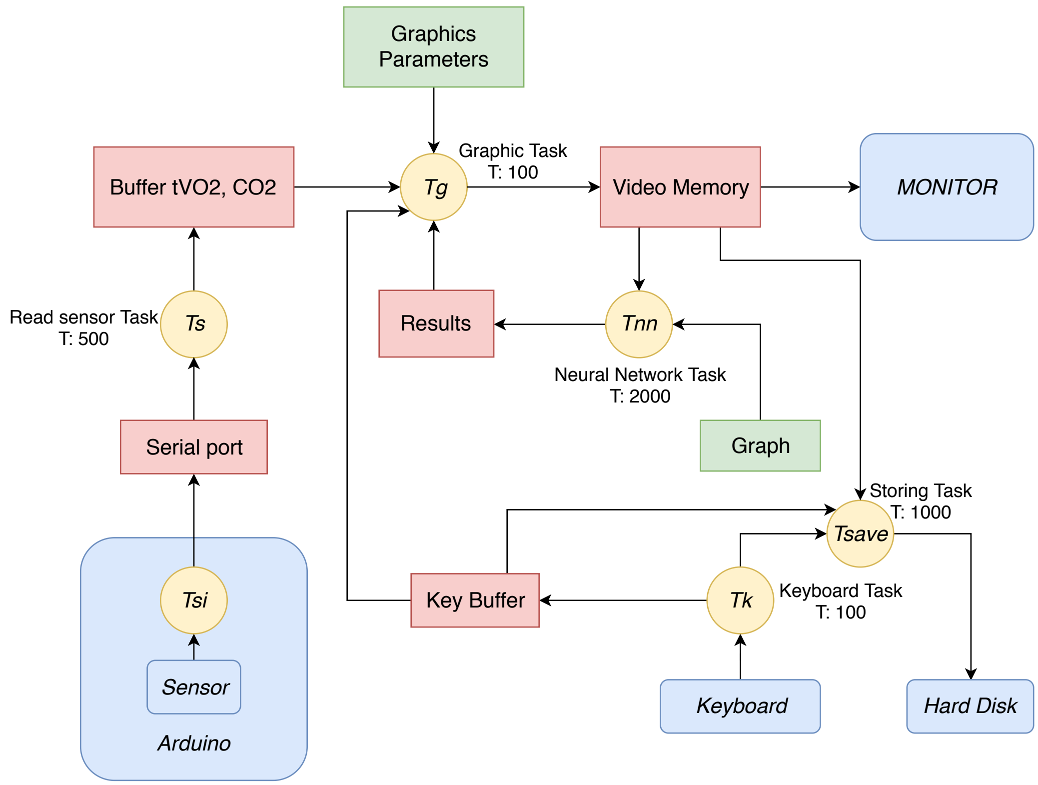

Figure 1: Task diagram

2.1 Main function

In the main function [1] all the tasks, except the store image task, are started and the mutexes initialized. The mutexes are four, one for the buffer that contains the values read from the sensor, one for the results given by the neural network, one to update the deadline miss and WCET values in the task table and one for the buffer that contains the keyboard input. The main also starts allegro and waits for the termination of the keyboard task. Once the keyboard task terminates the main cancels all other task and wait for their termination.

Algorithm 1 Main

$T \leftarrow$ *tasks to be started*
Mutexes and allegro initialization
for $t \in T$ **do**
    start t
end for
repeat
until *wait for termination of keyboard task*
for $t \in T$ **do**
    cancel and join t
end for

2.2 Graphic Task

The graphic task [2] prints the interface (Figure 2) of our application. The interface is divided into different areas which contains for each one: the graph, the image, the results, the legend, the current values, the information about tasks and the current mode (`SAVING` or `WRITING`) followed by, if present, the keyboard input.

Figure 2: Interface of application

Algorithm 2 Graphic task

$p \leftarrow$ *task period*
set activation task
draw interface background
loop
    draw graph and image
    draw results given by neural network
    draw current values read from sensor
    draw keyboard input
    draw task information
    wait for next activation
end loop

Graph

The graph (Figure 3) is made with the values sampled from the sensor. These values are plotted with two different colors: red for the $CO2$ and green for the $tVOC$. The graph contains at most N readings of both values.

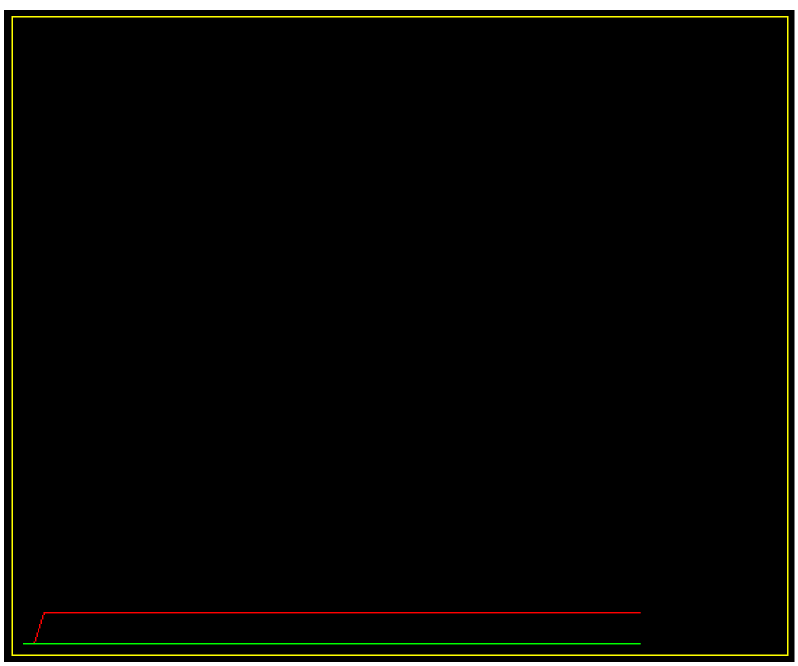

Figure 3: Graph

Image

The image (Figure 4) is made with the last N values sampled from the sensor. The allegro color mode is set to 15-bit. In the 15-bit mode each color is represented by 5-bit.

The sensor samples two data, the CO2 and the tVOC. Each value is a 15-bit number and so we represent the values read as colors in 15-bit. The image is divided in two parts, left and right. In the left side we draw the CO2 and in the right the tVOC. Even if each value could be potentially from 0 to $2^{16} - 1$ the CO2 value is between 400 and 8192 and the tVOC value is between 0 and 1187. These infomations are taken from the datasheet of sensor.

Whenever a new pair of values are read from the sensor, the image is moved one line below, removing one line at the bottom of the image and adding the new line, which contains the new values, on the top of the image.

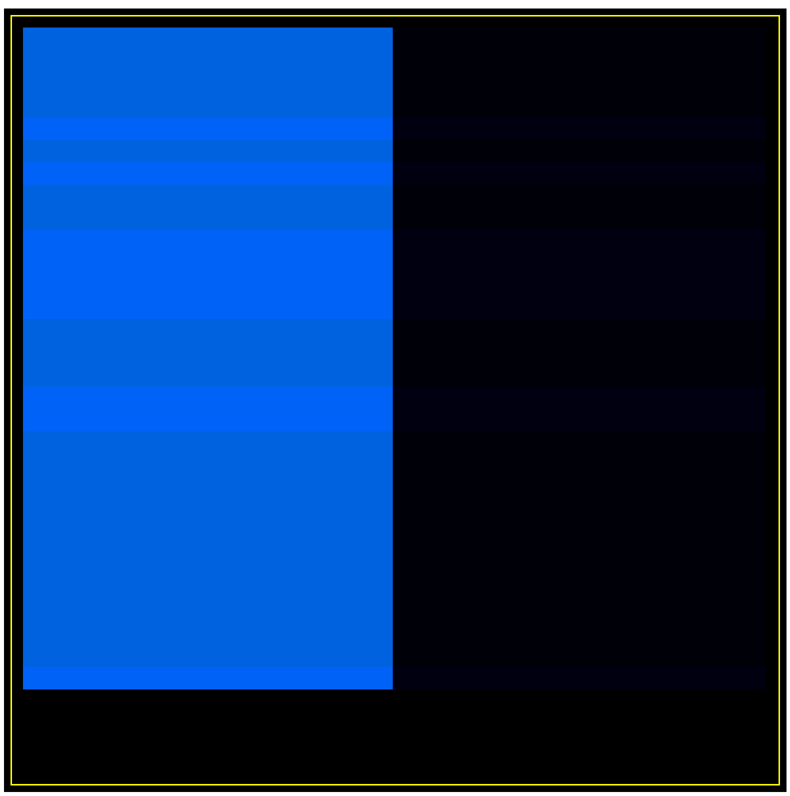

Figure 4: Image

Results

This area shows the results (Figure 5) of neural network having the current image as input. The lines represent the classes for which the neural network was trained and the values are the neural network output.

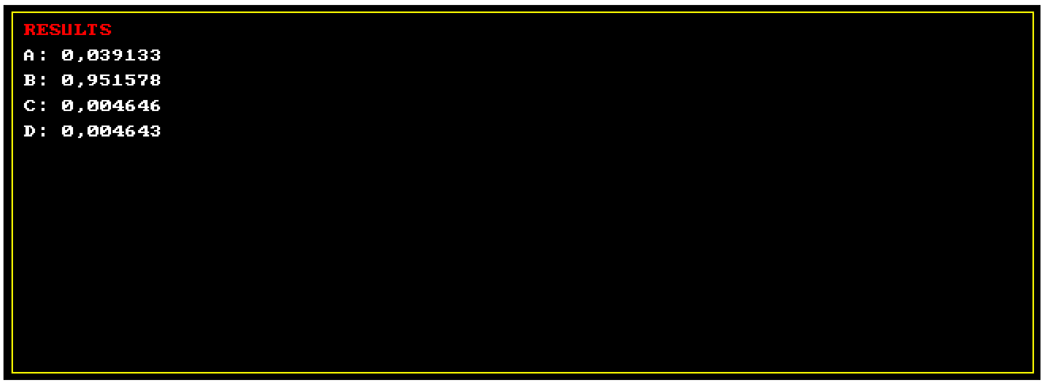

Figure 5: Result

Current values

This area shows the current values (Figure 6) sampled from the sensor.

Figure 6: Current values

Task information

This area (Figure 7) contains the information about deadline misses and WCET for each task of our application. Before a task suspend it self waiting for the next activation, it updates its values in the task table.

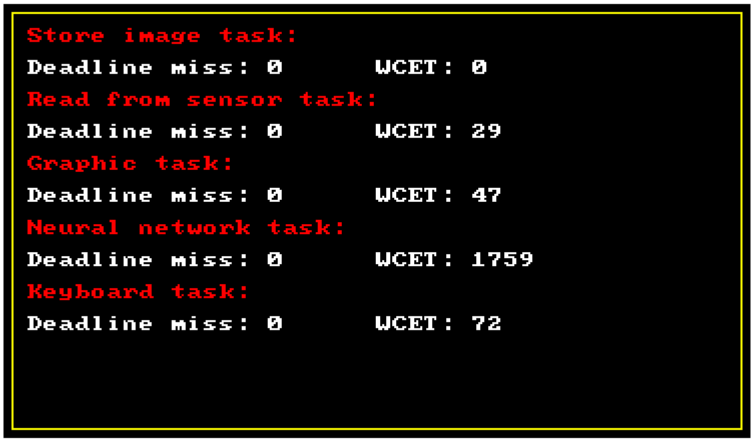

Figure 7: Task information

Current status

This area (Figure 8) gives the information about the current mode, printed on the left side; on the right side there is the name of the directory in which the image are saved.

Figure 8: Saving and writing mode

2.3 Sensor task

The sensor task [3] reads the values from the arduino which sends on serial port the values taken from the sensor. The read values are stored into an array and used by the graphic task to draw the image and the graph; the current read values are also standalone printed by the graphic task.

Algorithm 3 Sensor task

$p \leftarrow$ *task period*
initialization *data_q*
initialization serial port
set activation task
loop
  data_q $\leftarrow$ *data_q* + values read from sensor
  wait for next activation
end loop

2.4 Neural network task

The neural network task [4] recognizes the smells using as input the current image created with the values sampled by the sensor.

Algorithm 4 Neural network task

$p \leftarrow$ *task period*
Tensorflow initialization
set activation task
loop
  image $\leftarrow$ current image
  results $\leftarrow$ use neural network with given *image*
  wait for next activation
end loop

2.5 Keyboard task

The keyboard task [5] takes input from keyboard and puts it into *keyboard buffer*. The keyboard buffer is printed by the graphic task in its area of interface. The string contained into keyboard buffer is the directory, under *image_neural_network*, where the images are saved by the store image task. The keyboard task can be in two different mode: `WRITING` or `SAVING`. The task is started in `WRITING` mode. During this mode it's possible to write the name of the directory in which the images are saved. The name can contain letters, numbers, minus, underscore and point; it's also possible to delete the written characters pressing the `BACKSPACE` key. Pressing the `ENTER` key, the current mode is switched from the `WRITING` to the `SAVING` mode or vice versa. When the `ESC` key is pressed the keyboard task terminates causing the closing of our application.

Algorithm 5 Keyboard task

$p \leftarrow$ *task period*
cur_mode $\leftarrow$ `WRITING`
create *key_buffer*
keyboard initialization
set activation task
repeat
  key_pressed $\leftarrow$ key code from keyboard
  if *key_pressed* $==$ `ENTER` **then**
    if *cur_mode* $==$ `WRITING` **then**
      cur_mode $\leftarrow$ `SAVING`
      start store image task
    else
      cur_mode $\leftarrow$ `WRITING`
      stop store image task and clean key_buffer
    end if
  else if *cur_mode* $==$ `WRITING` **then**
    if *key_pressed* equal to letter, number, minus or point **and** *key_buffer* not full **then**
      key_buffer $\leftarrow$ *key_buffer* $+$ *key_pressed*
    else if *key_buffer* $==$ `BACKSPACE` **and** *key_buffer* not empty **then**
      remove last element from *key_buffer*
    end if
  end if
until *key_pressed* $!=$ `ESC`

2.6 Store image task

The store image task [6] is activated/terminated by keyboard task when the `ENTER` key is pressed. Once the task is activated the image are saved every 300 milliseconds. The images saved by this task are used to train the neural network for the recognizing of smells.

Algorithm 6 Store image task

$p \leftarrow$ *task period*
$dir \leftarrow$ *path to directory where images are saved*
set activation task
loop
    save image to *dir*
    wait for next activation
end loop

2.7 Scheduling

In our application a task table (Listing 1) was created containing all the task that have to be started by the main function. The parameters that characterize a task are respectively: thread identifier, function to be execute, priority, period, number of deadline miss and the WCET. At the beginning the thread identifier is set to -1, once the task is activated the thread identifier is updated. In case of deadline misses the application increments the corresponding value on the task table. The real time scheduling algorithm chosen is `SCHED_FIFO`. The deadline corresponds with the period.

```
Task task_table[] = {
    {-1, store_image_task, 20, 1000, 0, 0},
    {-1, read_from_sensor_task, 30, 500, 0, 0},
    {-1, graphic_task, 30, 100, 0, 0},
    {-1, neural_network_task, 25, 2000, 0, 0},
    {-1, keyboard_task, 30, 100, 0, 0}
};
```

Listing 1: Task table

3 Neural network

Regarding the neural network we used Tensorflow, an open-source software library for dataflow programming across a range of tasks.

For training the neural network was used a python script retrain provided by Tensorflow. The retrain script use the Inception V3 architecture. In this file a lot of configurations are possible. In our application we use the default configurations and choose only the number of training steps.

The output of the retrain script is saved into a file that we have called `graph.pb`. When the neural network task is activated, it loads this file that is used by tensorflow for recognizing the image. Each pixel of image is divided into three elements, one for each color: red, green and blue; the value of each color for each pixel is stored into a third order tensor, which is the input for the neural network.

4 Conclusion

The first experiment was made with wine, vinegar, ethyl alcohol and perfume. Using this smells the sensor goes into saturation and it is not able to discriminate between the smells. The same results was achieved increasing the distance between the smells and the sensor without no positive results.

After the first experiment the neural network was trained for recognizing smells of coffee and orange. This two smells was chosen due to their limited CO2 and tVOC values. However, in our opinion, the sensor used for the application is not able to discriminate between this two smells. Hence the neural network can't produce the hoped results using the images created starting from both values: CO2 and tVOC.