2022 Intel Cup Undergraduate Electronic Design Contest
- Embedded System Design Invitational Contest

Final Report

Intel Cup Embedded System Design Contest

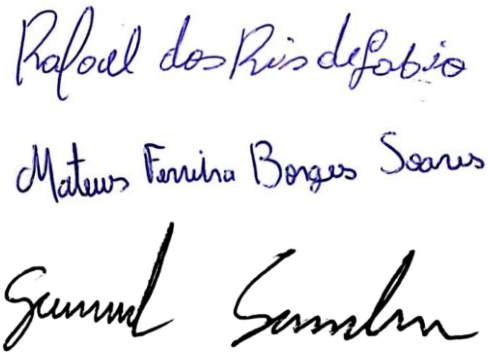
Project Name： Eyes of the River: Object Recognition on The Edge to Map Garbage Foci Along River Water

Students: Rafael dos reis de labio
Mateus Ferreira Borges Soares
Gabriel Almeida Schneider
Faculty: Centro de Informática
University: Universidade Federal de Pernambuco

2022 Intel Cup Undergraduate Electronic Design Contest
- Embedded System Design Invitational Contest

Declaration of Originality

We hereby declare that this thesis and the work reported herein was composed and originated entirely by ourselves. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the references.

Team Members Signature: 

Name (in Block Letters): RAFAEL DOS REIS DE LABIO
MATEUS FERREIRA BORGES SOARES
GABRIEL ALMEIDA SCHNEIDER

Date: 26/07/2022

EYES OF THE RIVER: OBJECT RECOGNITION ON THE EDGE TO MAP GARBAGE FOCI ALONG RIVER WATER

ABSTRACT

Water is a primordial substance for the existence of life on earth: plants, animals and human beings depend on it to survive and thrive. Rivers are a particularly notable source of water for their positive influence on the lives of the local communities but also their influence on a global scope, since they are directly linked to the oceans, and, thus, their health also affects the health of the oceans. One of the biggests sources of river degradation is pollution with plastic and other types of man-made materials, such as metal cans and food packaging. The first step for caring for a rivers' well-being, hence, is understanding where trash accumulates so that cleaning can take efficient action. Mapping garbage foci however, can be a challenging task to be done manually by humans on big rivers, and, thus, a solution that helps automate this work could lead to a better efficiency of cleaning efforts. In this report, we describe a system that utilizes artificial intelligence to perform object detection and help map garbage location. We consider two different machine learning models and two different hardware architectures to base our object detection module on, and we test and analyze their results in order to find the best suited solution for application on the edge.

Key words: machine learning, object detection, HOG, SVM, CNN, FPGA, edge, AI

Content

Chapter 1 - Introduction

1.1 Motivation

According to *The Ocean Cleanup*[1], rivers are a major source of plastic waste in the oceans. Locally, rivers often provide the means of living for riverside communities and are essential for the well-being of the nearby ecosystem. In general, rivers are of great importance for human beings and the environment. A lot of rivers, however, suffer from pollution by man-made objects that degrades its health and damages nearby life.

Manual efforts to map garbage foci, although a step in the right direction, are slow and ineffective. Thus, there is the need for a system that helps automate the process of finding where garbage is located and present this information in a way that helps decision-makers understand how they should allocate work to organize cleaning efforts more efficiently.

1.2 Goal

Our goal is to design a system that is capable of covering a great area of a river in the smallest amount of time possible, while capturing and processing all the necessary information to map garbage foci without the need for an internet connection, since some rivers might be in areas where cellular signal is not available. We set real-time processing of the gathered data as a requirement, since storing raw data in hopes of processing them later might lead to high memory requirements of on-board components, which might be impracticable in a context of an embedded system. Processing the data in real time also enables the system to adapt itself to dynamic conditions, which can be a decisive trait in some situations.

Chapter 2 - Proposed Solution

1.1 System Design Overview

We propose a system based on the GNS-V40 that commands a drone to fly along a river taking top-down pictures from above, sends those pictures to an android smartphone that acts as a bridge and then sends the pictures to the GNS-V40, that then runs one of the three available inference algorithms/architectures (HOG SVM on x86, HOG SVM on FPGA, CNN on x86) and detects the presence and location of 3 classes of garbage on the image (bottle, drink can, and milk carton).

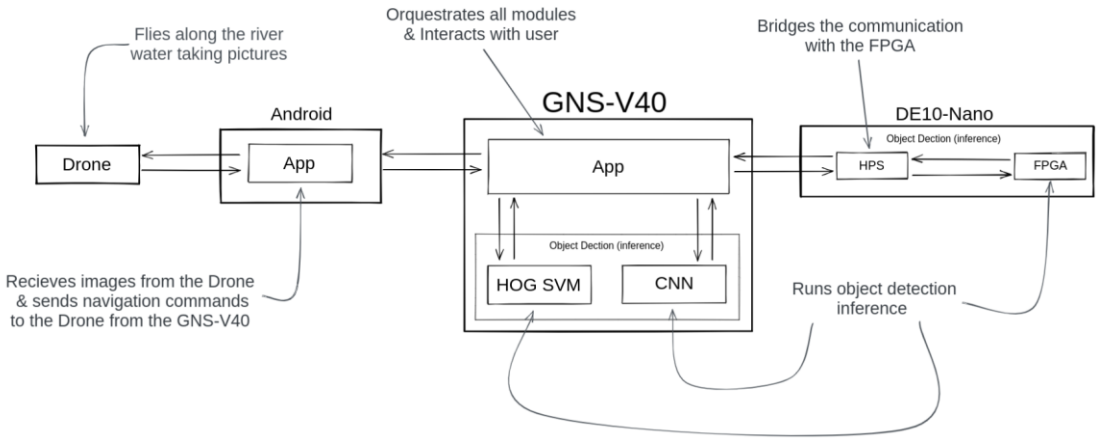


Fig. 1 - Block Diagram of the system.

The android application, *ControlePorMensageria*[2] is a third party project that turns the smartphone into a proxy between the drone and another agent, in this case, the GNS-V40, such as to enable this third agent (the GNS-V40) to send http requests to the app with movement commands as a way to control the drone indirectly and also receive photos taken by the drone, as well as its location at the time the photo was taken.

After receiving the image and location information, and after running one of the three inference algorithms/architectures available, if garbage is found on the image, the GNS-V40 pins the corresponding location on a map together with the photos taken, so that the user can see visually where are the garbage foci and what exactly was the type of garbage found on each place.

The design proposes that everything must be run locally on the edge, without any connection to the internet, and only connecting to local networks such as the one created by the smartphone, which ensures that the system can remain reliable while on locations without internet access.

Chapter 3 - Implementation

1. Communication Bridges

a. Android - GNS-V40 Bridge

The Android smartphone communicates with the GNS-V40via a Wi-Fi network that's established using the phone as a router. The images are then sent from the smartphone to the GNS-V40 through the HTTP protocol using the Wi-Fi network that was previously established. The Application Layer logic is handled by a NodeJS[3] server on the GNS-V40: it waits for images to be sent whilst handling other tasks, such as communicating with the HPS on the DE10-Nano and processing images.

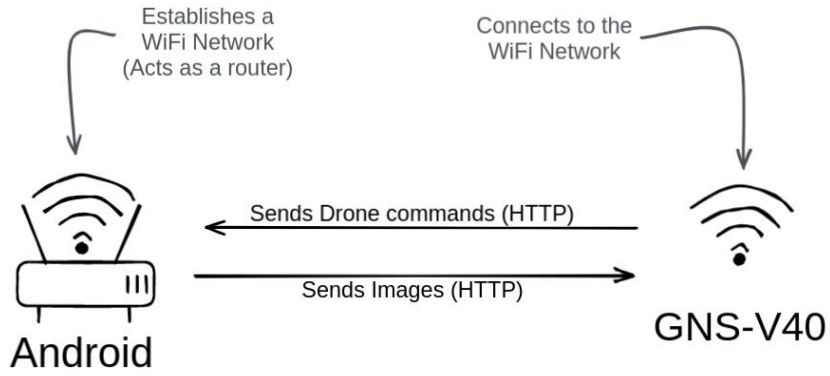


Fig. 2 - WiFi Communication between Android smartphone and GNS-V40.

b. GNS-V40 - DE10-Nano Bridge

The GNS-V40 computer sends images via an USART connection to the HPS in the DE10-Nano which then sends them to the FPGA fabric. Images are encoded in BASE64 encoding in the GNS-V40 and in the DE10-Nano's ARM Processor (Hard Processor System - HPS) are decoded back to binary format. The BASE64 encoding was chosen because it's a plain text format and by using so we can leverage tools and libraries that are made to send text via USART. When the HPS reads back the inference results from the FPGA, it sends to the GNS-V40 in text format also via the USART connection.

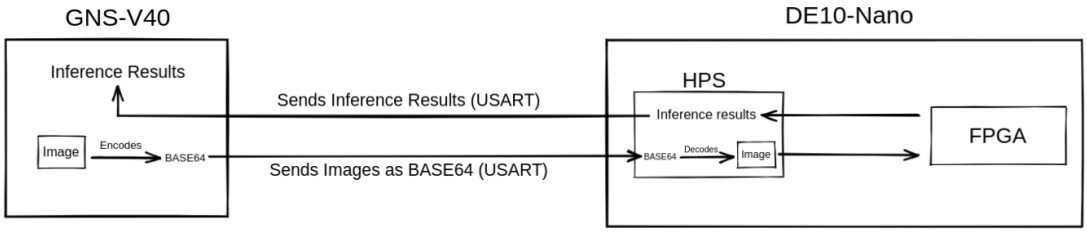


Fig. 3 - USART Communication between GNS-V40 and DE10-Nano.

c. (Inside DE10-Nano) HPS - FPGA bridge

The HPS and the FPGA communicate inside the DE10-Nano via a connection between an AXI BUS Master (on the HPS side) and an Avalon Bus Slave (on the FPGA side), which enables HPS→FPGA communication with feedback[4] .We developed a protocol to exchange data between them in an orderly way by utilizing a control word that refers to what state the state machine running inside the HPS should be in. Images

are stored in RAM in the FPGA's fabric alongside a control word (On-Chip memory, where the first word is for control and the second word onwards is for data). The control word is used to dictate when the FPGA or HPS should read or write. The speed of transfer is bound by the underlying AXI bridge and by the packeting that's needed to overcome the space limitations in the FPGA's RAM.

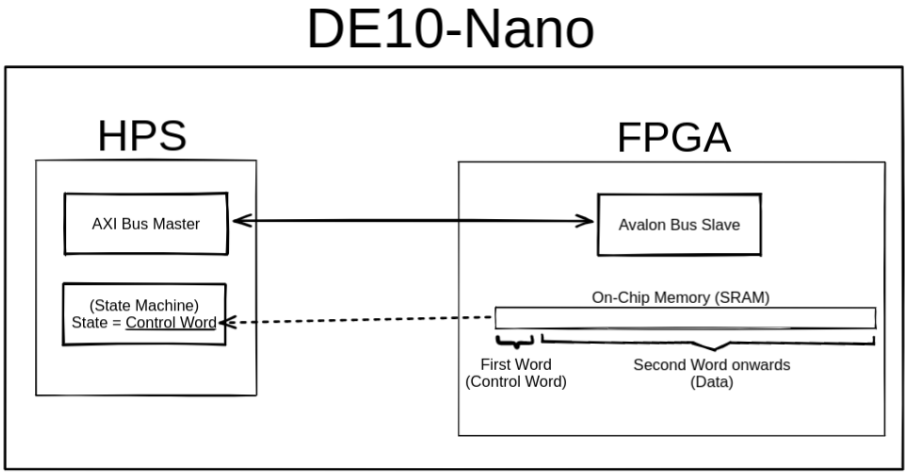


Fig. 4 - Bus communication inside DE10-Nano.

2. Dataset

a. Virtual dataset generation with Blender

Due to the very specific nature of our object detection use case, no readily available datasets fulfilled our needs. We considered manually creating a dataset with real photos, but ultimately decided to utilize virtual generation in order to achieve a high level of control over the objects and the environment, which would lead us to a high variability of lighting, position of objects and general configuration of the scene.

We expanded upon the *blender-dataset*[5] toolkit to create our renders, and implemented scenes with multiple lighting conditions and three objects: a milk carton, a drinking can and a bottle, which are types of garbage commonly found in rivers. We generated close-up scenes of objects and multiple-objects scenes, each one aimed at a different training model, as explained further in the report.

Fig. 5 - Close up of a milk carton.

Fig. 6 - Multi-object scene.

b. Creating multiple datasets

Since the virtual generation is very flexible, we experimented populating our dataset with different types of lighting and texture shaders, in order to give the dataset a high variability and

ensure it is robust enough to yield good results when running the inference with real-life photos.

Fig. 7 – Forest ambient lighting.

Fig. 8 – City ambient lighting.

3. HOG-SVM Model

a. Implementation and how it works

Histogram of Oriented Gradients (HOG) is an image feature extraction technique that, according to *Wikipedia*[6] "counts occurrences of gradient orientation in the localized portion of an image". It is a way of representing relevant features for object detection related to the edges of the object.

Support Vector Machine (SVM) is a machine learning algorithm that analyzes data for classification and regression analysis. It is a way of classifying an image as one of two possible categories, given a previous training where the correct classification was provided.

We utilized the *OpenCV Documentation HOG-SVM example code*[7] as a starting point to train a model with this technique, based on our dataset of close-ups: in the training step, close-ups of individual objects were assigned as of the "positive" class, while close-ups that contained only water were designated as of the "negative" class. The result model of this training is a .xml file that can be used together with an sliding window object detection software to infer the position, on a given never-seen image, of the objects referred as "positives" during the training.

b. Running on the GNS-V40 (x86 architecture)

Since the GNS-V40 has a CPU, we can utilize the same code from *OpenCV's* documentation to infer the results for any new image, since the sliding window technique is also implemented as a "test" step during the training process. We modularized this code so it could run outside of the context of training, and so it could be integrated into the end-software running on the GNS-V40.

c. Running on the FPGA (custom hardware)

Aiming to find what alternative would suit best the edge constraints, we decided to also run our HOG SVM model inside an FPGA. We used a *third party, unreleased HDL module developed in our research group* [8] to run through our dataset and find out what are the advantages and disadvantages of running this inference technique on a CPU vs running it on a FPGA.

4. CNN Transfer Learning Model

a. Implementation and how it works

A Convolutional Neural Network (CNN) is a very popular type of Neural Network that is common in use-cases of object detection. Transfer Learning is the process of, given an already existing Neural Network, removing its last layers, where new layers, originated by a new dataset, will be created and cemented on. This makes it possible to take advantage of a model that already exists and has good results, teaching it new information, so it behaves as if it was trained, from scratch, solely by this new dataset.

For this type of training, we used our virtually generated dataset that was originally multi object per scene, and with the help of the *Roboflow*[9] online tool, we manually labeled, for each image, the three types of objects, each belonging to a specific and consistent class. The *Roboflow* tool was also helpful in performing data augmentation, which increased the size of our dataset.

Fig. 9 - Manual object labeling performed in *Roboflow*

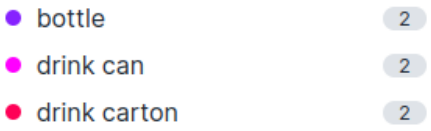


Fig. 10 - Labels utilized in manual object labeling in Roboflow

We utilized the *YOLOV3-tiny model*10] as our base-model for the transfer-learning of our virtual dataset. The transfer learning process was made on a NVIDIA GPU Tesla T4, allocated inside an instance of a third party *Google Collaboratory Notebook*[10], in which the fine tuning of the base model was performed.

b. Running on Google Colab

The inference process was already implemented after the transfer learning in the mentioned Google Collaboratory project, hence, we got the metrics of its performance.

c. Running on the GNS-V40

In order to run the output model of the transfer learning process, to infer on new images inside the GNS-V40, we ran the *darknet project*[11] locally and also got to observe the metrics of accuracy and performance of the model.

Chapter 4 - Results

1. Datasets

The validation process of the datasets was two-fold: first, we manually selected the datasets that presented a higher variability of visuals and scenery. Second, we trained, and transfer learned models with these datasets and then chose the ones that gave the best results in all training algorithms and all platform architectures. Datasets with too much "aggressive" data augmentation, that distorted too much the original images actually performed worse in most cases then datasets without any data augmentation at all. The best combination of factors for detecting a multi object scene under a determined light condition was a model trained with a dataset also constrained to that specific type of lighting.

2. HOG SVM Model

Running on the GNS-V40, the HOG SVM showed a mean time of about 33 milliseconds to infer one image, while the time of inferring one image on the FPGA detection module was about 3 milliseconds and sometimes lower. There were no notable differences in the accuracy of the model running in both hardware architectures.

3. CNN Transfer Learning Model

While there was no notable difference in the accuracy of the CNN transfer learning model under different hardware, the performance boost that running on a dedicated GPU brings was very notable: the inference of one image on the GNS-V40 took about 1300 milliseconds average, while the Google Colab computer with a powerful dedicated GPU took only about 4.6 milliseconds average.

4. CNN vs HOG SVM under same dataset

Under the same dataset, the CNN-based solution had a better accuracy than the HOG SVM one. The HOG SVM, however, tends to be faster under comparable hardware conditions (CNN on CPU only compared to HOG-SVM on CPU only and CNN on GPU compared to HOG-SVM on FPGA).

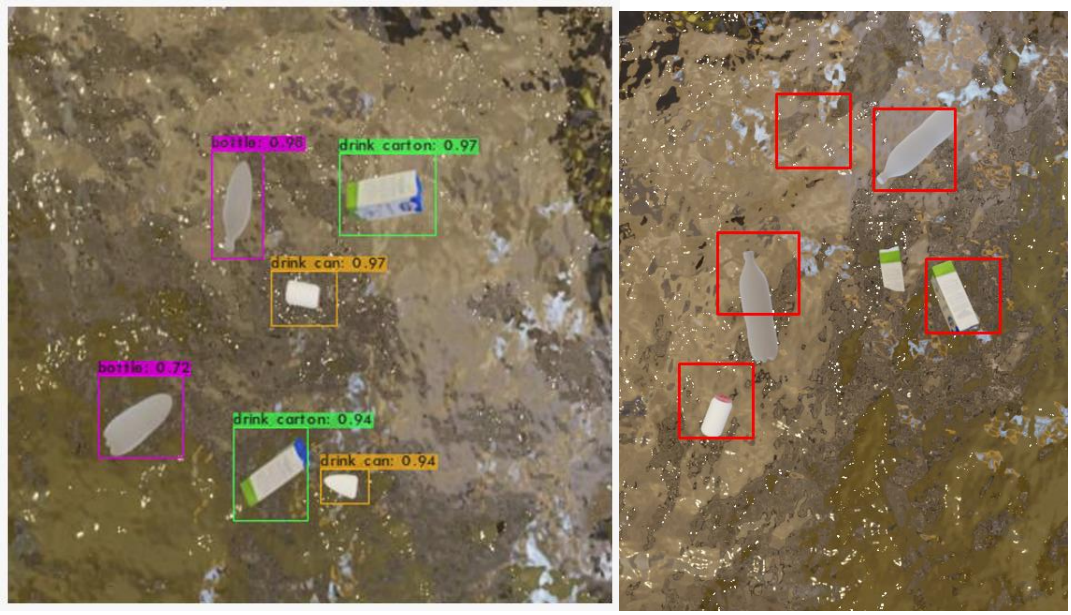


Fig. 11 - Comparing the inference accuracy of the CNN-based solution (left) vs HOG-SVM-based solution (right).

Chapter 5 - Conclusion

As the results demonstrated, a CNN-based object detection model is more flexible and yields better accuracy results than the HOG-SVM counterpart when compared under the same dataset. The HOG-SVM technique, however, is faster on the CPU, when compared to the CNN on the CPU, and can be even faster than the CNN on a GPU, when executed on an FPGA. The optimal design for a object detection solution on the edge, thus, must choose between higher portability plus higher speed but generally worse, or more difficult to train, results (HOG-SVM on an FPGA) and worse portability plus lower speed, but generally higher accuracy that's more easy to train (CNN on a CPU, or GPU).

The proposed solution, thus, not only fulfills its purpose but also shows that it is flexible, by having multiple possible ways of implementation, depending on the specific needs of the user.

References

[1] The Ocean Cleanup: https://theoceancleanup.com/sources/

[2] ControlePorMensageria: https://github.com/bcfreitas/ControlePorMensageria

[3] NodeJs: https://nodejs.org/en/

[4] DE1-SoC: ARM HPS and FPGA - Addresses and Communication - Cornell ece5760: https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/HPS_peripherials/FPGA_addr_index.html

[5] Blender-dataset: https://github.com/ivan-alles/blender-dataset
[6] Histogram of oriented gradients: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

[7]samples/cpp/train_HOG.cpp: https://docs.opencv.org/3.4/d0/df8/samples_2cpp_2train_HOG_8cpp-example.html

[8] Lucas Cambuim and Edna Barros, FPGA-Based Pedestrian Detection for Collision Prediction System, Sensors, 2022

[9] Roboflow: https://app.roboflow.com/

[10] Darknet tyni: https://github.com/AlexeyAB/darknet#how-to-train-tiny-yolo-to-detect-your-custom-objects

[11] Darknet Project: https://pjreddie.com/darknet/