Problem Description

Name: Prabhdeep Singh Sethi
Email: prabhdeepsethi0@gmail.com
Mob.: +919112360030
Place: Nagpur, Maharashtra, India
Date: 15/12/21 (15th December 2021)

1. Objective:

a) The objective of the project is to make a real-time embedded drone detection system from the infrared data. The data will be collected with different UAVs, altitude, distance and lighting conditions after that team will develop a model which will be able to detect the UAVs on new data. The proposed system will process in the following ways:
   i. Pre-processing of the data received from the IR camera.
   ii. Defining the ROI to start the detection of UAV.
   iii. Detection of the UAV with confidence index above defined threshold.
   iv. Continuing the detection and/or tracking with guided search.
   v. The output of the system would be the a co-ordinate of the centroid of bounding box of UAV as well as the confidence index.
   vi. System will be deployed on a custom PCB which needs to fit in a 70 mm diameter space.
   vii. System should work in a dynamic environment with high body rates, velocity and non-co-operative situations.
   viii. System should be able to process 50 frames per seconds (FPS).

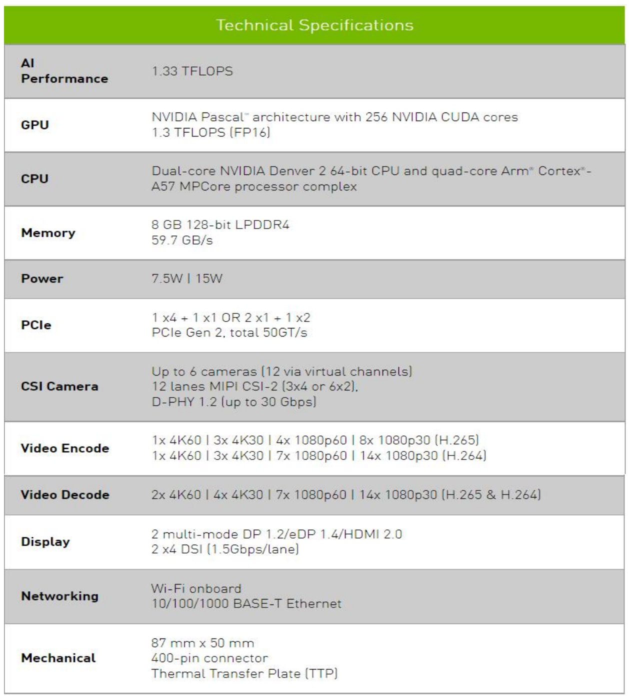
b) A gist of it is: Detecting UAV in IR video at 50 FPS and the system would be deployed on a custom PCB of size 70mm diameter.

2. Key Specifications:

a) Hardware size: 70mm diameter / 49 mm square
b) Inference time: 20 ms
c) Start time / Boot time Required: 2s
d) Other environmental condition:
   i. Shock proof
   ii. Operational temperature: Military grade (-55°C to +125°C)
   iii. Electromagnetic interference
e) Power Availability: (Data not available)

3. Current Approach to solve the problem:

a) Problem: To find out the hardware required for running our desired software
b) Solution:
   i. First we started with a hardware readily available with us so that we can benchmark it and find out the requirement accordingly
   ii. We had with us a Nvidia Jetson TX2. Attaching the device specifications overview



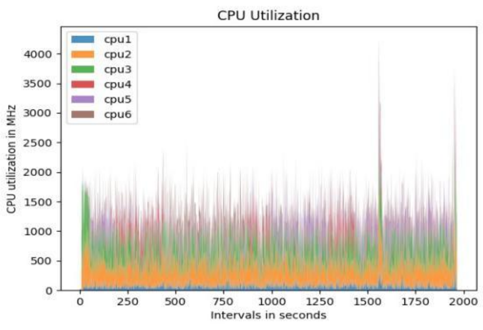
Technical Specifications	
AI Performance	1.33 TFLOPS
GPU	NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores 1.3 TFLOPS (FP16)
CPU	Dual-core NVIDIA Denver 2 64-bit CPU and quad-core Arm® Cortex®-A57 MPCore processor complex
Memory	8 GB 128-bit LPDDR4 59.7 GB/s
Power	7.5W | 15W
PCIe	1 x4 + 1 x1 OR 2 x1 + 1 x2 PCIe Gen 2, total 50GT/s
CSI Camera	Up to 6 cameras (12 via virtual channels) 12 lanes MIPI CSI-2 (3x4 or 6x2), D-PHY 1.2 (up to 30 Gbps)
Video Encode	1x 4K60 | 3x 4K30 | 4x 1080p60 | 8x 1080p30 (H.265) 1x 4K60 | 3x 4K30 | 7x 1080p60 | 14x 1080p30 (H.264)
Video Decode	2x 4K60 | 4x 4K30 | 7x 1080p60 | 14x 1080p30 (H.265 & H.264)
Display	2 multi-mode DP 1.2/eDP 1.4/HDMI 2.0 2 x4 DSI (1.5Gbps/lane)
Networking	Wi-Fi onboard 10/100/1000 BASE-T Ethernet
Mechanical	87 mm x 50 mm 400-pin connector Thermal Transfer Plate (TTP)

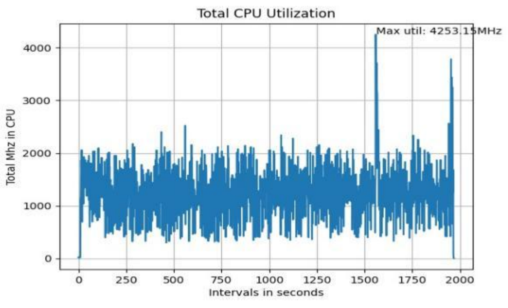
iii. Our approach was to run a Deep Learning model similar to we would run in production

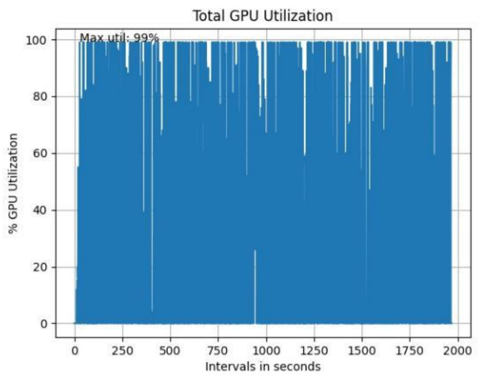
iv. We were using YOLOv5s object detection here. In the final code there would be more complexities as well as some higher order logic but the deep learning process would be the most compute intensive part of the project. So we tried to benchmark it.

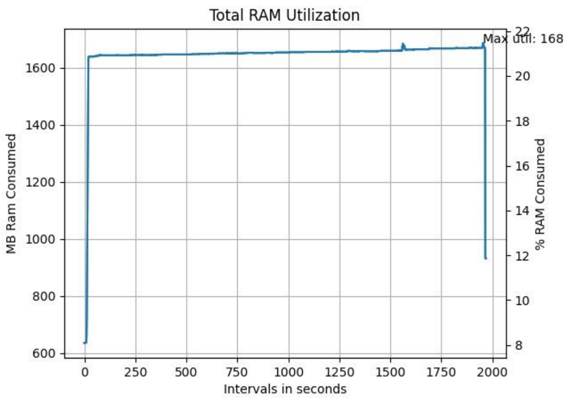
v. Once our deep learning model was ready we optimized it by pruning/sparsification and quantization using Nvidia TensorRT.

vi. Following are the resource utilization when running YOLOv5s on Tx2

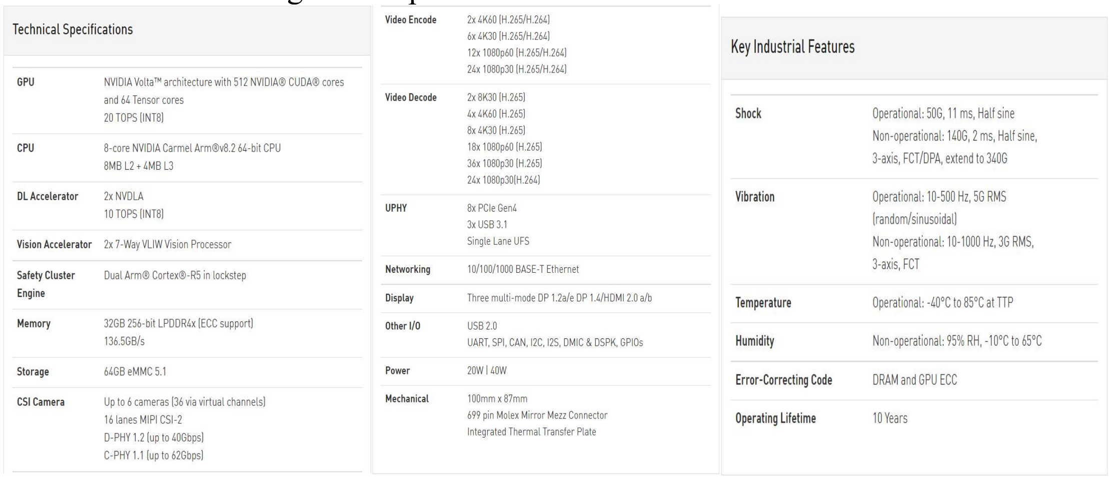








vii. Finally we after all the optimizations we were getting 37 FPS on Nvidia Jetson TX2.
viii. We figured we needed an embedded device with a more powerful GPU as that was the bottleneck for us in order to achieve 50FPS and remember in this we were getting data from saved video file whereas in actual condition we will be getting data from an IR camera live. Also, more code complexity would be added.
ix. So based on these we identified the following hardware: Nvidia Jetson Xavier AGX Industrial. We feel that this hardware has enough specification to take the load of our approach.
x. Following are its specifications:



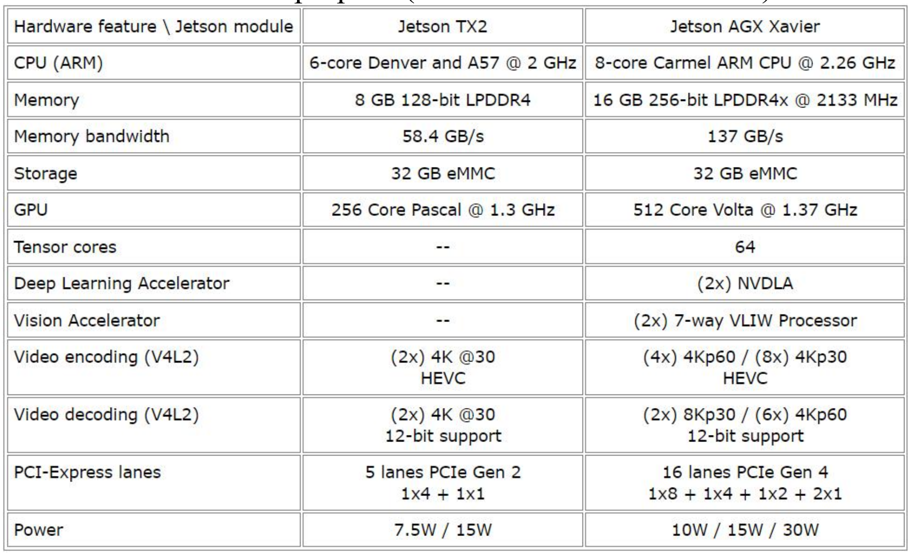
Technical Specifications

GPU	NVIDIA Volta™ architecture with 512 NVIDIA® CUDA® cores and 64 Tensor cores 20 TOPS (INT8)
CPU	8-core NVIDIA Carmel Arm®v8.2 64-bit CPU 8MB L2 + 4MB L3
DL Accelerator	2x NVDLA 10 TOPS (INT8)
Vision Accelerator	2x 7-Way VLIW Vision Processor
Safety Cluster Engine	Dual Arm® Cortex®-R5 in lockstep
Memory	32GB 256-bit LPDDR4x (ECC support) 136.5GB/s
Storage	64GB eMMC 5.1
CSI Camera	Up to 6 cameras (36 via virtual channels) 16 lanes MIPI CSI-2 D-PHY 1.2 (up to 40Gbps) C-PHY 1.1 (up to 62Gbps)
Video Encode	2x 4K60 (H.265/H.264) 6x 4K30 (H.265/H.264) 12x 1080p60 (H.265/H.264) 24x 1080p30 (H.265/H.264)
Video Decode	2x 8K30 (H.265) 4x 4K60 (H.265) 8x 4K30 (H.265) 18x 1080p60 (H.265) 36x 1080p30 (H.265) 24x 1080p30(H.264)
UPHY	8x PCIe Gen4 3x USB 3.1 Single Lane UFS
Networking	10/100/1000 BASE-T Ethernet
Display	Three multi-mode DP 1.2a/e DP 1.4/HDMI 2.0 a/b
Other I/O	USB 2.0 UART, SPI, CAN, I2C, I2S, DMIC & DSPK, GPIOs
Power	20W | 40W
Mechanical	100mm x 87mm 699 pin Molex Mirror Mezz Connector Integrated Thermal Transfer Plate

Key Industrial Features

Shock	Operational: 50G, 11 ms, Half sine Non-operational: 140G, 2 ms, Half sine, 3-axis, FCT/DPA, extend to 340G
Vibration	Operational: 10-500 Hz, 5G RMS (random/sinusoidal) Non-operational: 10-1000 Hz, 3G RMS, 3-axis, FCT
Temperature	Operational: -40°C to 85°C at TTP
Humidity	Non-operational: 95% RH, -10°C to 65°C
Error-Correcting Code	DRAM and GPU ECC
Operating Lifetime	10 Years

xi. Following is a broad comparison between our hardware we used for benchmarking (Nvidia Jetson Tx2) vs the hardware we feel would be able to solve our purpose (Nvidia Jetson Xavier AGX):



Hardware feature \ Jetson module	Jetson TX2	Jetson AGX Xavier
CPU (ARM)	6-core Denver and A57 @ 2 GHz	8-core Carmel ARM CPU @ 2.26 GHz
Memory	8 GB 128-bit LPDDR4	16 GB 256-bit LPDDR4x @ 2133 MHz
Memory bandwidth	58.4 GB/s	137 GB/s
Storage	32 GB eMMC	32 GB eMMC
GPU	256 Core Pascal @ 1.3 GHz	512 Core Volta @ 1.37 GHz
Tensor cores	--	64
Deep Learning Accelerator	--	(2x) NVDLA
Vision Accelerator	--	(2x) 7-way VLIW Processor
Video encoding (V4L2)	(2x) 4K @30 HEVC	(4x) 4Kp60 / (8x) 4Kp30 HEVC
Video decoding (V4L2)	(2x) 4K @30 12-bit support	(2x) 8Kp30 / (6x) 4Kp60 12-bit support
PCI-Express lanes	5 lanes PCIe Gen 2 1x4 + 1x1	16 lanes PCIe Gen 4 1x8 + 1x4 + 1x2 + 2x1
Power	7.5W / 15W	10W / 15W / 30W

xii. So the objective of this phase was to find out the appropriate hardware that would solve the problem and we came up with one.

4. Problems with existing system:

a) Now our model is running with good accuracy but we cannot deploy this system due to the following points

i. <u>Physical constraint (size):</u>

        1. The size of our system should fit in either a 70 mm diameter PCB or a length could be extended to 115mm but width has to be in that 70mm range.
        2. Now, this Jetson AGX is of the size 100mm*87mm and already very dense so there is hardly any scope for reduction of its size.
        3. Although, we can find out individual parts of this hardware like the GPU and CPU and build a custom PCB.
    ii. <u>General purpose vs specialized hardware:</u>
        1. Now, the Jetson AGX is a general purpose hardware which is not built specifically for our purpose.
        2. A point has been raised by the hardware team that in such a case the reliability in terms of operation of the system is hard to determine.
        3. Also, the reliability in different environmental conditions is also hard to determine and require lots of test in different conditions.
    iii. <u>Booting time:</u>
        1. The booting time of the complete system which is the time from which the system boots up, the program starts, the YOLOV5s model loads and inference starts on Jetson TX2 is around 10-15 seconds.
        2. Here, we do assume we will get a better boot up time of the coplete system with Jetson AGX but we still think it would be a very challenging task to optimize it to 2-3 seconds.

5. Proposed system by the hardware team:

a) <u>Proposal:</u>
    i. Will find out the absolute resources required by current software. Amount of instructions sent to GPU per cycle for parallel processing, RAM consumed, CPU utilization as well as other key resources by individual layers of our deep learning model (YOLOV5s)
    ii. Once we have the complete resource utilization will replace GPU with FPGAs and DSPs for parallel processing.
    iii. Based on that a custom specialized hardware will be designed.

b) <u>Problems:</u>
    i. Finding out the absolute requirements is a complex task and would require a lot of time whereas we are on a constrained timeline
    ii. The amount of community support and frameworks/libraries available for Nvidia GPU support is way way more compared to FPGAs, where either we drop the idea of deep learning to drop FPGA or we have to reinvent the wheel by writing multiple custom codes/libraries for parallel processing where we lack both in terms of proficiency and time.

6. Help required in following:

a) Designing a hardware which could run a YOLOv5s model or any deep learning model for that purpose.

b) The hardware should abide by the physical constraints as well as it should be powerful enough to process the 50 FPS.

c) If both these things are not possible we will have to drop the deep learning approach and shift to a pure image processing based system. But then solving the UAV detection problem with purely image processing robustly is not possible or would not fit in our time frame of developing the system in 6 months. We can also, process the complete deep learning on CPUs but that also requires a lot of engineering and would not be possible in time frame and is not guaranteed to work.

d) Also, if a Nvidia based hardware doesn't work, help is required in designing a FPGA based hardware for faster processing. Figuring out following related to FPGA:
   i. No. of I/Os
   ii. Manufacturer
   iii. Level of complexity
   iv. Heatsinks, etc.

7. If we go from an approach other than a Nvidia GPU based approach, what would be the impact.

a) The core reason behind adopting a Nvidia based system is that it has direct compatibility with high level deep learning frameworks like Pytorch (we are using this right now) or Tensorflow.
b) It uses Cuda Toolkit and Cudnn which interacts with our framework to manage all the GPU cores and assign task to it rapidly.
c) The system is extremely tightly bounded and works in real-time
d) If we move to any other hardware for parallelization the major problem is we loose the compatibility and we have to design custom libraries which takes a lot of time.
e) We have seen some implementations of FPGAs with deep learning but they are still in a very nascent stage.