Design Flow and Implementation of a Vision-Based Gesture-Controlled Drone

Arthur Findelair, Xinrui Yu and Jafar Saniie
Embedded Computing and Signal Processing (ECASP) Research Laboratory (http://ecasp.ece.iit.edu/)
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago IL, U.S.A

***Abstract*—The increasing efficiency of complex Neural Network architecture and the continuous improvement of embedded edge computing has reached a point that allows the deployment of advanced computer vision tasks on some of the most critical embedded applications, such as aerial drones. While similar applications were already possible by moving heavy processing on a ground station, an autonomous and centralized system significantly improves usability and security. The machine is thus self-sufficient and less prone to network attacks. Three main challenges stand-out during the deployment of our complex gesture recognition pipeline: (1) allowing user-defined controls, (2) ensuring robustness, and (3) on-board deployment. These challenges are tackled through handcrafted features to avoid the curse of dimensionality, neural network optimization on GPU-based companion computers, and data augmentation to cover real-life edges cases such as partial inputs.**

***Index Terms*—Machine learning, Gesture control, Embedded software**

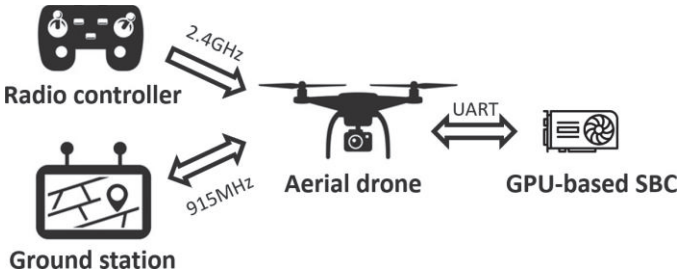


Fig. 1: Overview of the aerial drone augmented with a GPU- based Single Board Computer (SBC). Note that the ground station and the radio controller are used solely for experimentation safety. The whole processing pipeline is deployed on-board.

I. Introduction

There are two crucial characteristics to deploy autonomous drones in more applications. First, drones must perfectly perceive their environment to avoid obstacles and adapt trajectories to reach specific positions. Secondly, the human-machine interaction must be flawless. An assistant drone is hardly useful if it cannot efficiently receive orders from users who could simultaneously perform other tasks. Simplified remote controllers limit the interaction with users equipped with a device, thus known beforehand. Otherwise, drones must perceive orders. Besides speech, the most natural way for humans to communicate is through visual cues. Audio-based solutions are hardly possible due to the loud nature of drones' propulsion systems. This leads us to the latter communication medium: visual cues.

The main objective of this project is to create a fully autonomous gesture-controlled drone. Robustness and expandability are the core characteristics of the implementation. Enforcing these two qualities lead to overcoming three main challenges that encapsulate the main contribution of this paper.

- **Customization & expandability:** Human Machine Interfaces HMIs should evolve following the variety of use- case, and preferences of users. However, the proposed architecture relies on machine learning models that require labor-intensive models.
- **Robustness:** The correctness of the prediction is critical to the operability of the machine. Misinterpreted gestures can lead to unwanted and potentially dangerous behaviors from the drone.
- **On-board deployment:** The computational capability of a small-footprint embedded edge-computing platform is still minimal.

II. System design

A. Hardware

Our goal is to develop a flexible HMI that is easily integrable in embedded applications. It is thus essential to have a computational platform with low power consumption and a small footprint to fit in a large array of scenarios. Still, pose estimation systems, and more generally, deep learning techniques, are particularly computationally heavy. The hardware platform must support parallel computing to operate neural network inference. One of the most common solutions is the Jetson platform from NVidia, which comprises multiple GPU- based single-board computers. We will use the entry offer of this product range as a companion computer, the Jetson Nano B01 4Gb. Even this entry solution will prove powerful enough to handle the gesture-control processing pipeline after some optimization.

As its name suggests, the companion computer is not suited to support autonomous flight operations. We choose the Pixhawk platform, part of the ArduPilot eco-system, to handle this task. The Pixhawk 4 from Holybro implements the FMUv5 design, compatible with the advanced flight controller firmware ArduCopter and the ground-station software Mission

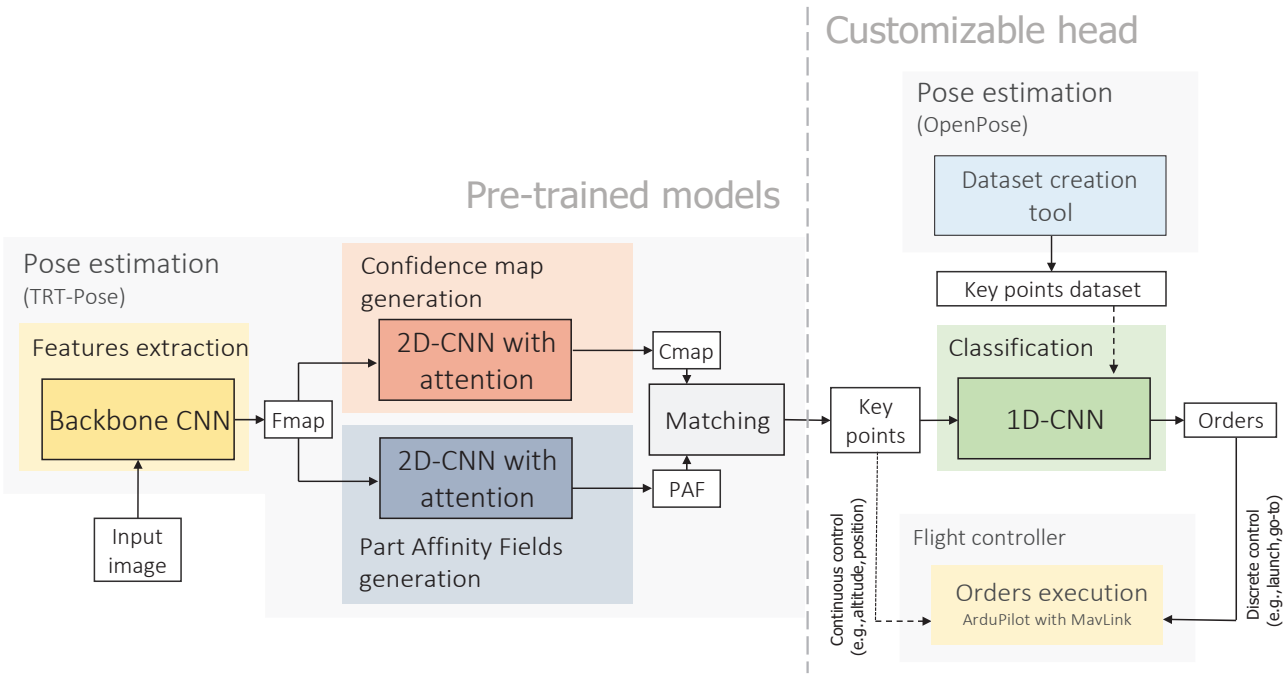


Fig. 2: Flowchart of the processing pipeline. The main contribution of this project lies in the customizable head to ease user-defined HMI. (Fmap – Features map; Cmap – Confidence map; PAF – Part Affinity Fields)

Planner. In addition, a telemetry kit, a radio receiver, and a GNSS & compass module are connected to the flight controller to improve flyability and ease of use. The industry standard MavLink is used for intercommunication between flight controller, ground station, and companion computer. The power distribution is handled by a dedicated power management board connected to the main battery of the drone. The 5V voltage converter of the board is limited to 15W, which is perfectly sound for the flight controller. However, the companion controller requires an additional voltage converter to ensure a stable powering of the flight controller; the Jetson Nano can exceed the consumption of $10W$ to reach a top performance of 472 GFLOPS (FP16) on its Maxwell GPU.

Finally, an IMX477 camera module from Arducam is mounted on the drone. The combination of a 3.9mm lens with a 1/2.3", 12.3MP sensor allows a large horizontal FOV (80°), which matches the camera used on the workstation for dataset creation. This CSI camera is compatible with the Nvidia Encoder (NVENC) included in the Tegra X1 chip used on Nvidia Jetson devices. Images will be captured with the GStreamer plugin included in the Jetpack software suite. It leverages the NVENC to lighten precious computation load on the CPU.

The set of components described above sums up to a mass of 500g (17.6oz). We have mounted the system on a quadrotor in X configuration with an amplitude of 500mm. This ample wheelbase offers plenty of room and a large enough payload to fit in the hardware thanks to a maximum thrust of 1300g per motor with a 6000 mAh, four cells (14.8 V) lithium-polymer battery. The take-off weight sums up to 1.6 kg.

B. Video processing pipeline

Human pose estimation is the task of inferring the precise pose of a person by identifying and locating key-points on the body, such as major joints (elbow, knee, shoulders, etc.). There are two approaches to this problem, known as bottom-up and top-down. The bottom-up approach is more complex than the other to train due to the more generalized approach to keypoints detection and skeleton reconstruction. However, it is way more efficient thanks to the single-shot key-points extraction, especially for crowd analysis. This method is a great fit in our case. It allows scalability of the processing pipeline to multiple cameras and users while keeping a constant inference time. The current most popular bottom-up architecture is based on the fusion of Part Affinity Fields (PAFs), and part Confidence Maps (CMaps) [1]. This architecture is behind some of the most used pose estimation models such as OpenPose [2] and TensorRT-Pose. As shown in figure 2, it is composed of the following blocks:

- The image goes through a backbone CNN which generates a feature map from the input image. Most models submitted to ILSVRC [3] can be used as backbone extractors. These models are all trained to classify images on a general-vision dataset with a couple of thousands of labels and more than 14 million images. Thus, their first layers are extremely efficient at extracting relevant information toward a pseudo-global understanding of images.

- The feature map is fed to a couple of multi-stage CNN that produce Part Confidence Maps (CMap) and Part Affinity Fields (PAF). The CMap represents the probability that a particular human joint can be located in any given pixel. It contains as many channels as the number of types of body joints detected in the image. The PAF is a vector field that encodes the orientation and location of limbs. Again, it contains as many channels as the number of types of limbs (i.e., joint pairs).
- The CMap and the PAF are processed by a greedy bipartite matching algorithm to output the skeleton estimation for each person in the image. Such algorithms have no trainable parameters.
- A final neural network block classifies
- Key-points. The discrete outputs are interpreted as orders transmitted to the drone. The human-machine interface is thus mainly defined by the categories and samples composing the key-point classification model.

Data collection is one of the most time-consuming and critical tasks of a Machine Learning project. A good dataset eases the models' training process and is also vital to the generalization power of the model. The objective is to create a dataset as unbiased as possible. Suppose we were to train the pipeline end-to-end, using images as input. The dataset should consist of various people with different visual features (e.g., body type, skin tone, clothes) in different contexts. Given the complexity of images, it is difficult, even maybe impossible [4], [5], to eliminate all biases. Such limitation is a massive challenge to the customization of our HMI. Instead of training the pipeline as a whole, we deploy state-of-the-art pre-trained models and train the classifier independently on a key-points-based dataset. These handcrafted features reduce the dimension of the dataset space by a thousand-fold and thus avoid the curse of dimensionality [6], [7]. The input space being way smaller, it is easier to eliminate biases: only the positions of the articulations of a person relative to the camera are captured in the dataset. A few hundred samples per class are enough to eliminate most biases from the dataset.

III. Implementation

A. Dataset creation

We have developed an open-source tool 1 to collect body key-points samples based on the OpenPose pre-trained model. It allows the user to efficiently and easily create datasets suited to our classification problem. Using this tool, we have created a base dataset with a total of 20 body dataset classes which contains between 500 and 600 samples each for a total of 10680 entries. Each entry in the dataset is an array of 25 2D coordinates. The mapping of these key-points follows the BODY25 body model, one of the most comprehensive discretized standard body models. However, some pose estimation models, such as the one we will use on the Jetson Nano, use an 18 keypoints representation (BODY18). The seven missing key-points do not strongly influence classification as 6 of them are used for feet representation, and the last one is a central hip key-point.

[1]Code-source: https://github.com/ArthurFDLR/pose-classification-kit

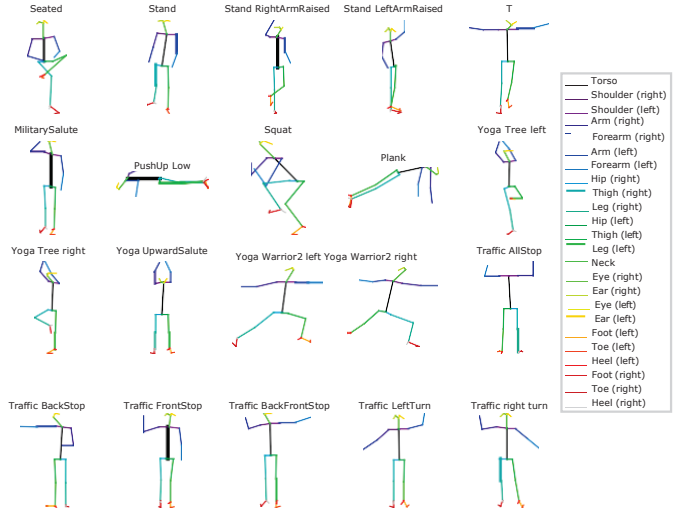


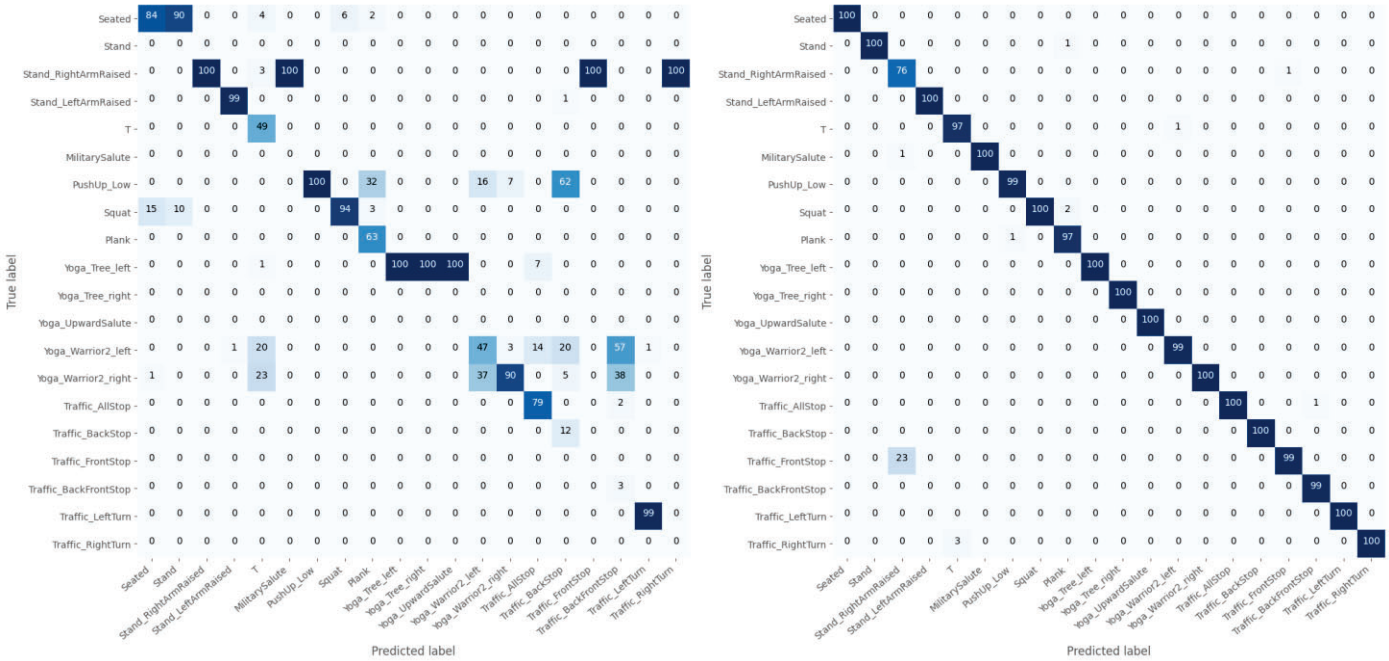
Fig. 3: Random samples from each class of our body gesture dataset (BODY25)

Figure 3 presents samples from all clauses currently included in the dataset. The goal of this initial dataset is to allow several simple classification applications. We can distinguish three main categories: work-out exercise, Yoga pose, and traffic signs. Given our application, we will focus on the traffic hand signals. These are the movement used by police officers or aircraft marshallers to direct traffic. They are thus perfectly suited to control our drone.

B. Model creation

While a classical neural network composed of densely connected layers could map any classification problem, architecture that intrinsically accounts for the particularity of the input generally performs better [8]. Preserving the 2D nature of our key-points and their relative positions can be achieved by 1D convolution layers [9]. It aims at finding patterns in the input. The order of the key-points is thus critical to detect local patterns. Indeed, given a kernel length of 3, the convolutional layer detects templates embedded in the coordinates of 3 consecutive key-points. The most efficient way to order the key-points is thus to form anatomical parts of the body; shoulder–elbow–wrist, and hip–knee–foot are consecutive in our body models. Our best-performing model comprises two 1D-convolution layers with 16 kernels of length 3, two dense layers of 128 neurons each, and a final dense layers matching the number of classes (20); for a total of 57172 learnable parameters. All layers but the last use a ReLu activation function. The output is selected following the softmax function over the 20 logits values. In addition, a couple of regularization methods are used to deal with heavy data augmentation. A dropout rate of 30% and batch normalization are applied to all layers.

The model is trained using the Adam optimizer with a categorical cross-entropy loss function, and a 20-20-60 split



(a) Model trained without data augmentation

(b) Model trained using data augmentation

Fig. 4: Confusion matrices of our best performing models on a synthetic dataset accentuating partial inputs based on our original test fold.

for validation, testing, and training. The loss function reaches a minimum around the 15 epochs, where the validation accuracy is 98.00%. The testing accuracy reaches 98.25%.

While these results are great theoretically speaking, the model's limitation lies in the dataset itself. We have observed inconsistent predictions on partial input during real-life tests. The most common partial inputs (i.e., missing key-points) are caused by body parts not being captured in the original input frame (e.g., obstruction, out of frame), specifically the lower part of the body. Several augmentation techniques are applied to the dataset to improve robustness: scaling, rotation, noise addition, key-point dropping. The dropping is applied randomly and/or on all key-points from the lower part of the body (legs, hip) as a whole. The newly trained model with data augmentation reaches a testing accuracy of 98.05% but is drastically more robust to partial input. Figure 4 presents the confusion matrices of the model trained with or without data augmentation on a synthetic test dataset following similar data augmentation techniques as presented above.

C. On-board deployment

The neural networks composing our pipeline are optimized on the target device (Jetson Nano) to accelerate the inference process. We mainly leverage quantization [10] while compiling the models for our platform using TensorRT, the inference optimization tool for NVIDIA GPUs. The allocation of 33MB GPU memory is enough for each model. In addition, while the pose estimation model reaches far better inference times using FP16 data representation, this is not the case for the pose classification model. This is undoubtedly due to the very low complexity of the model. Still, the FP16–33MB compiled models are selected for deployment. The model compilation process is highly effective as it improved the cumulative inference time of both models from 637ms to 73ms.

The order manager is the most critical part of the embedded processing pipeline as it directly controls the drone's actions. Every movement must be carefully controlled to avoid crashes. The flight controller includes some protections over received orders over MAVLink connections. Such critical application requires careful analysis of the deep learning pipeline output. Temporary misclassification is relatively common and can lead to unwanted actions from the drone. Still, it is improbable that several consecutive frames are similarly misclassified. Thus, orders are selected only when the processing pipeline detects the same pose four times over the seven latest predictions. The order is then transmitted to the flight controller only if it is different from the last selected order to avoid repetition. In addition, the order manager includes a state machine to avoid sending irrational orders, such as taking off when the drone is flying. Also, for safe experimentation, orders are only transmitted if the flight controller is in *guided* mode, which is selected through the radio controller.

IV. Results

Our gesture-controlled drone currently supports the following commands:

- *T*: Arm the drone if it is disarmed and landed; Disarm the drone if it is armed and landed;
- *Traffic AllStop*: Take-off at an altitude of 1.8m if the drone is armed and landed – Land if the drone is in flight;

– *Traffic RightTurn*: Move 4m to the right;
– *Traffic LeftTurn*: Move 4m to the left;
– *Traffic BackFrontStop*: Move 2m backward;
– *Traffic FrontStop*: Move 2m forward;
– *Yoga UpwardSalute*: Return to Launch (RTL);

Fig. 5: Processing pipeline output frame

No delay is perceivable thanks to the processing pipeline's relatively high frame rate, which varies from 9.5 to 12.0 frames per second. The embedded camera's large vertical field of view allows a functional system in flight and on the ground. The user can thus fully control the drone from take-off to landing using gesture control only. However, the system does not include user tracking. The user has to stay in front of the drone to perform gesture control. This is particularly restrictive when the drone operates at a high altitude. The maximum height at which the drone can operate naturally varies according to the camera's orientation and field of view. The limit is approximately 4 meters in our configuration – camera leveled – for the system to also detect gestures while landing.

V. Conclusion

The main objective of this project was to streamline the training and deployment of gesture recognition systems while offering a proof-of-concept of a controlled drone. Following this process, this paper presents the creation and training of two neural networks. The first model is highly efficient and yet reaches a testing accuracy of 98.25%. However, it performs poorly on partial inputs, which are common use-cases in practice. Thus, a second model has been created leveraging heavy data augmentation and regularization techniques. While the testing accuracy remains similar to the first model, it performs very well under challenging conditions.

In addition, we covered the deployment process of the video processing pipeline on resource-limited hardware. An open-source drone platform has been augmented with an embedded Jetson Nano companion computer to allow gesture control. The pipeline optimization allows a processing frame rate exceeding 9.5 FPS – such performance results in a very responsive proof-of-concept.

References

[1] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields, 2017.

[2] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[4] Kaiyu Yang, Klint Qinami, Li Fei-Fei, Jia Deng, and Olga Russakovsky. Towards fairer datasets: Filtering and balancing the distribution of the people subtree in the imagenet hierarchy. In *Conference on Fairness, Accountability, and Transparency*, 2020.

[5] Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei A. Efros, and Antonio Torralba. Undoing the damage of dataset bias. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 158–171, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[6] Loris Nanni, Stefano Ghidoni, and Sheryl Brahnam. Handcrafted vs. non-handcrafted features for computer vision classification. *Pattern Recognition*, 71:158–172, 2017.

[7] Aleš Leonardis and Horst Bischof. Kernel and subspace methods for computer vision. *Pattern Recognition*, 36(9):1925–1927, 2003. Kernel and Subspace Methods for Computer Vision.

[8] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.

[9] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021.

[10] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *CoRR*, abs/2103.13630, 2021.