Self-Flying Aircraft

Prakhar Dubey, Raghav Mehta

***Abstract* - Within the last decade, the advancement in automation of vehicles such as cars and planes promise to fundamentally alter the microeconomics of transporting people and goods. In this paper, we focus on the self-flying aircraft through computer vision. This subset of automated flight would be the most valuable in terms of efficiency, human error reduction and loss of life due to mid-air collisions. We present an analysis of control systems for collision avoidance for an aircraft.**

Keywords - UAV; UAS; UTM; drones; identification; traffic management system; collision avoidance.

I. INTRODUCTION

Advances in computer vision technology have enabled its wide adoption in the auto industry. Today, many vehicles are equipped with backup, front-looking, and side-looking cameras that allow for real time environment analysis and critical decision making.

Many vehicular technologies, particularly those for safety and guidance, can roughly be classified into autonomous (autonomous vehicles) [1, 2, 3, 4] and collaborative (connected vehicles) [5, 6] schemes. In an autonomous scheme, a vehicle relies mostly on its own on-board sensor to produce environmental data for real time manoeuvrings, navigation and collision avoidance.

II. RELATED WORK

II-I. Transfer Learning

A closely related area of work to our study is transfer learning (see Pan and Yang [7] for a review). The early studies of transfer learning with Deep Convolutional Networks successfully demonstrated domain adaptation through pre-training on source data and fine-tuning on target data [9, 10, 11]. Further studies such as the work of Ganin et al. [8] present more sophisticated approaches to domain adaptation. In this work, we apply the most widely used fine-tuning approach to domain adaptation and leave further studies on feature transferability of the synthetic data to the future work.

II-II. Imitation Learning

Imitation learning has been commonly applied to solve different tasks in isolation. This usually requires either careful feature engineering, or a significant number of samples. This is far from what we desire: ideally, robots should be able to learn from very few demonstrations of any given task, and instantly generalise to new situations of the same task, without requiring task-specific engineering. In this paper, we propose a meta-learning framework for achieving such capability, which we call one-shot imitation learning.

Specifically, we consider the setting where there is a very large set of tasks, and each task has many instantiations. For example, a task could be to stack all blocks on a table into a single tower, another task could be to place all blocks on a table into two-block towers, etc. In each case, different instances of the task would consist of different sets of blocks with different initial states. At training time, our algorithm is presented with pairs of demonstrations for a subset of all tasks. A neural net is trained that takes as input one demonstration and the current state (which initially is the initial state of the other demonstration of the pair), and outputs an action with the goal that the resulting sequence of states and actions matches as closely as possible with the second demonstration. At test time, a demonstration of a single instance of a new task is presented, and the neural net is expected to perform well on new instances of this new task. The use of soft attention allows the model to generalise to conditions and tasks unseen in the training data. We anticipate that by training this model on a much greater variety of tasks and settings, we will obtain a general system that can turn any demonstrations into robust policies that can accomplish an overwhelming variety of tasks.

III. DATASET

Fig. 1 Synthetically rendered Environment (up)
Real World Environment (down)

III-I. Authenticity

Our work focuses on controlling an aircraft in a real world flight simulator. It is thus, imperative to gather data appropriately. Since game simulators provide a great benchmark for obtaining training data as can be seen in fig. 1, we used the flight engine in Grand Theft Auto V for generating the frames required for this work.

This platform seems to be an authentic source for coming up with the dataset as noted duly by Will Knight in 'MIT Technology Review' quoting "Self-Driving Cars Can Learn a Lot by Playing Grand Theft Auto. Hyper-realistic computer games may offer an efficient way to teach AI algorithms about the real world. Several research groups are now using the hugely popular game, which features fast cars and various nefarious activities, to train algorithms that might enable a self-driving car to navigate a real road." [12] (MIT technology review, Intelligent Machines by Will Knight)

Similar findings were proposed in the SHAFAEI et al. [13] (Alireza Shafaei, James J. Little and Mark Schmidt. On - Play and Learn: Using Video Games to Train Computer Vision Models, 2016) Over 60,000 synthetic samples were collected from a modern video game with similar conditions to the real-world CamVid and Cityscapes datasets. Several experiments were demonstrated indicating that a convolutional network trained on synthetic data achieved a similar test error to a network that was trained on real-world data for dense image classification. Furthermore, the synthetically generated RGB images could provide similar or better results compared to the real-world datasets if a simple domain adaptation technique was applied.
Instead of monitoring ground traffic, we instead look at the aircraft, its surroundings and seek to achieve successful locomotion after training the model with a CNN architecture.

III-II. Generation

The dataset generation was the bulkiest portion of the work. In addition to perfectly flying the plane in GTA-V repeatedly, several utilities had to be added to ensure no unwanted frames went into the dataset. Grabbing the data frames from the windows DC handle of the process was the foremost task. After taking the win32 handle of the process, the captured frames were converted to bitmap images. These frames were linked with the corresponding keyboard input as labels and appended into a numpy file for storage.
The complete module involved three stages: takeoff, cruise and landing.

All three of the stages required repeated runs of the plane, taking off successfully, cruising through the target circles and finally landing safely on the runway and coming to a halt.
At each step, a start-stop feature for frame capturing was introduced so that only relevant frames entered the data.
Once enough frames for each stage were obtained (~100,000). We stopped doing runs and fed the data into the net.

III-III. Annotation

The data was of the form ([image, keyinput]). After combing through the data for any errors, the keyinput was mapped to a multi one-hot array. The keys corresponded as a [W, A, S, D, 8, 4, 6, 5] list each of which controlled the aircraft in the way a real plane is handled (see fig. 2).

W-S were linked to thrust and brakes. A-D controlled the perpendicular Yaw axis. 8-5 were used for changing the pitch while 4-6 allowed the aircraft to roll longitudinally. The multi-one hot array contained a bit(0/1) linked to each button. It also included a ninth parameter for no keyinput. So any input which was pressed could be mapped as a bit list. For instance, thrusting while rolling left

would be taken as a bit list of [1, 0, 0, 0, 1, 0, 0, 0, 0].

After the data formatting was completed, the dataset was divided into a 95:5 training and test set and pushed into the CNN used.

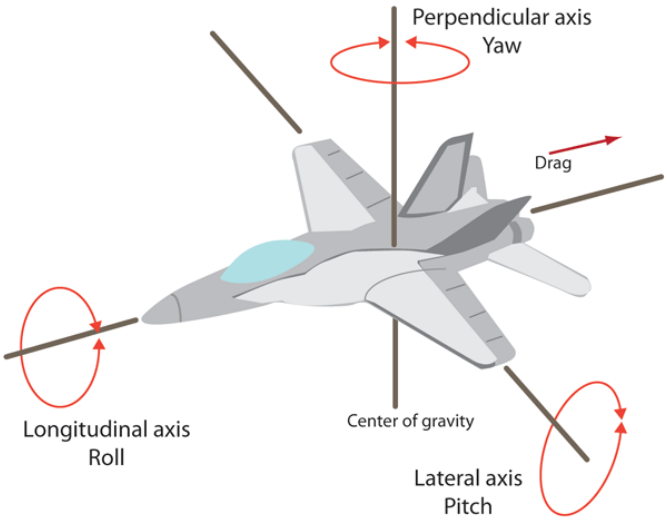


Fig. 2: Motions available to a plane in actual flight.

IV. DATA PROCESSING

IV-I. Convolutional Neural Network

As stated in the paper for AlexNet [3] (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. On - ImageNet Classification with Deep Convolutional Neural Networks) To learn about thousands of objects from millions of images, we need a model with a large learning capacity. The capacity of Convolutional Neural Networks (CNNs) can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Despite the attractive qualities of CNNs, and despite the relative efficiency of their local architecture, they have still been prohibitively expensive to apply in large scale to high-resolution images. Luckily, current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of interestingly-large CNNs.

In the end, the network's size is limited mainly by the amount of memory available on current GPUs and by the amount of training time that we are willing to tolerate. Due to limited power capacity, GPU capabilities and time bounds, we fed the data into the simple yet better performing AlexNet architecture, tweaking it slightly to give relevant results.

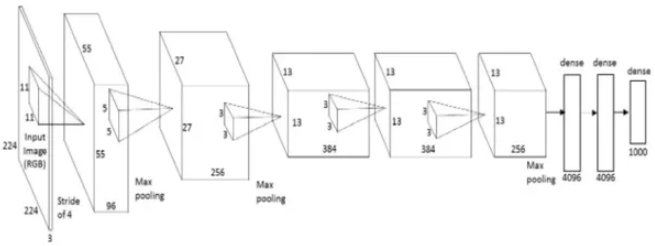


Fig. 3: Alexnet architecture.

IV-II. The Architecture

The net contains eight layers with weights; the first five are convolutional and the remaining three are fully connected. The output of the last fully-connected layer is fed to a 9-way softmax which produces a distribution over the 9 class labels.

The above final layer has been modified to give relevant output to our aircraft as keyboard input for the simulator.

The neurons in the fully connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers, follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size 5 × 5 × 48. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size 3 × 3 × 256 connected to the (normalized, pooled) outputs of the second

convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurones each.

IV-III. Implementation

There are three separate models prepared for takeoff, cruise and landing. Each is trained separately and tested on the divided dataset.

When a module is initiated, it takes the simulator window DC and converts it to bitmap for the image input. Then, corresponding to the image, it predicts whether it needs to thrust or brake, yaw left or right, pitch up or down, roll clockwise or anticlockwise or do nothing. For every frame it outputs a bit list as explained above. This output is in the form of keyboard input which is fed directly to the simulator which then takes the corresponding course of action.

Each of the prepared models are tested, although the accuracy of the model as seen from tensorboard may not justify the performance since it relates to the correct output per frame instead of taking into account whether it was successfully able to takeoff or land the plane.

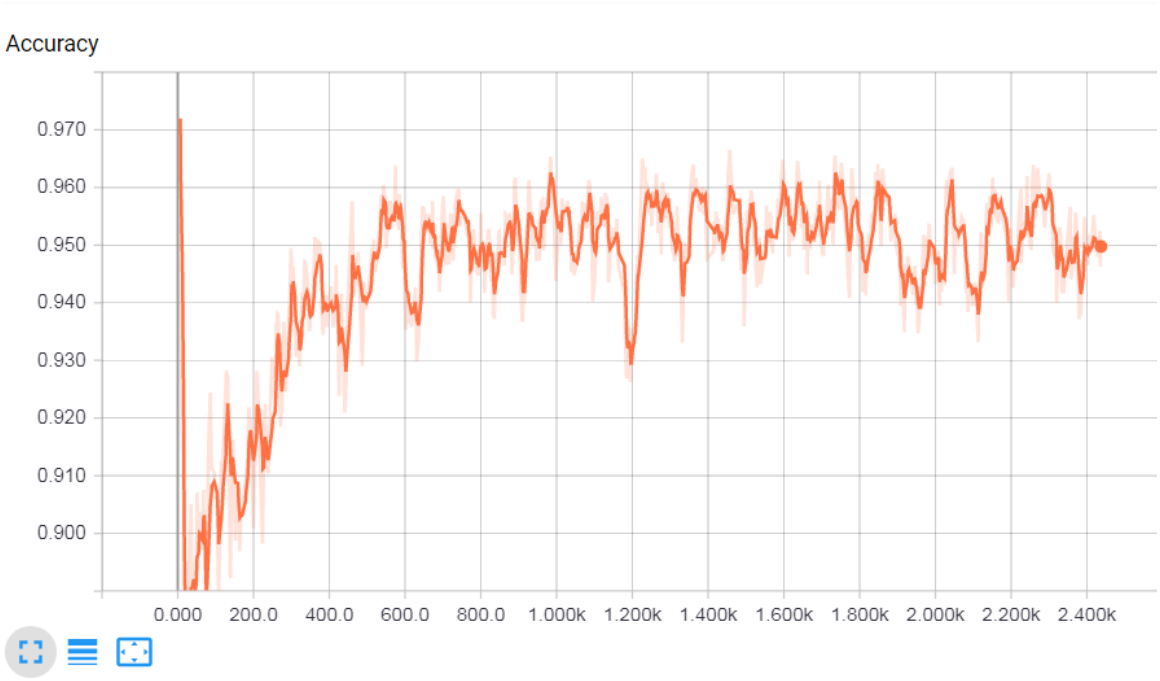
In general though, the plane performs admirably in takeoff and landing phases where most human pilots fail.

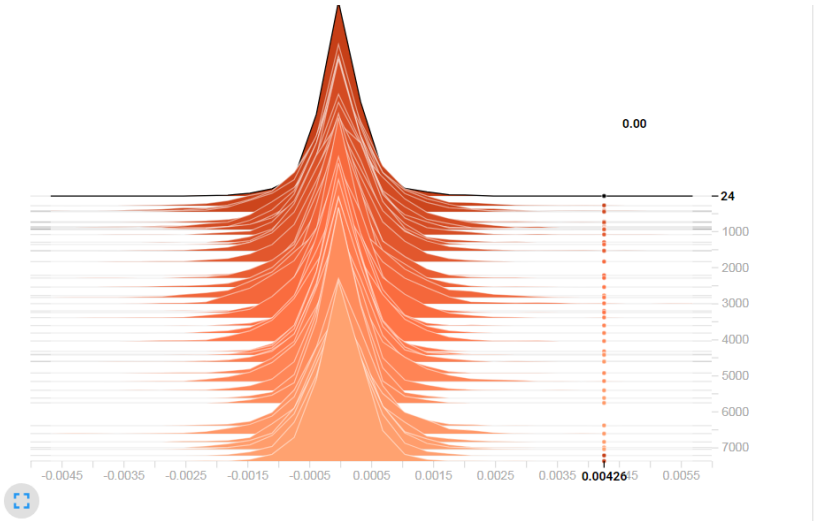
V. CONCLUDING REMARKS

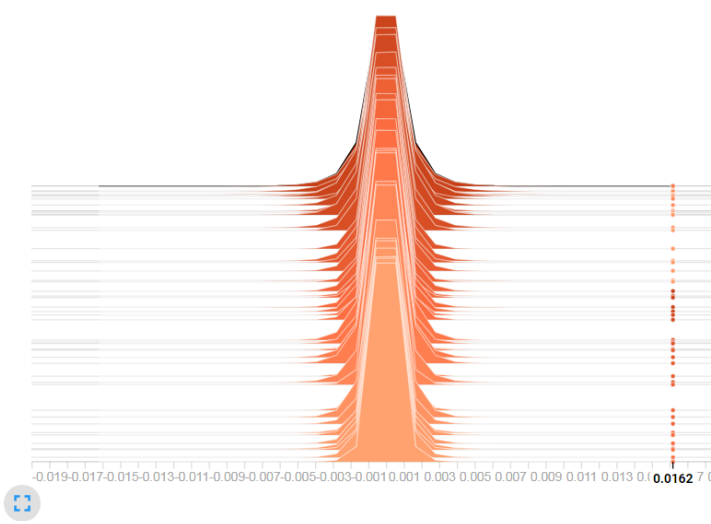
As video games progress towards photorealistic environments, we can also use them for the training of computer vision models at no extra cost. We delivered a proof of concept by exploring the use of synthetic RGB images that we extracted from a video game.[15]

We examined GTA-V as a model to test, train and enhance deep learning in self-flying aircraft research. We found that GTA-V allows researchers to create, train, and test on photo-realistic data to accurately estimate from the driver's perspective the distance to lane markings, distance to cars, and angle of navigation. The best results of the CNN came from estimating horizon and altitude maintenance. The aircraft stability estimation could be improved with more training time,. Additionally, we used GTA-V as a model in creating a more robust training environment that would allow researchers to easily create data for testing and training models on. Using our fabricated environment we were able to recreate critical oil safe training scenarios. We showed that virtual data can be a powerful way to create new data invaluable for training and testing neural networks for autonomous driving. The most important way to further validate this work and to make it truly useful is to combine it with real data to understand what the current limitations are with mixing real and virtual data as well as augmenting data sets that need specific corner cases.

SUPPLEMENTARY GRAPHS







Fully connected Layer Gradients Normalised

REFERENCES

[1] E. Guizzo. How Google's Self-Driving Car Works. IEEE Spectrum, Feb. 2013.

[2] HAVEit. European Union Research Project HAVEit. http://www.haveit-eu.org/, 2015.

[3] Hoeger Reiner and Amditis Angelos and Kunert Martin and Hoess Alfred and Flemish Frank and Krueger Hans-Peter and Bartels Arne and Beutner Achim and Pagle Katia. Highly Automated Vehicles for Intelligent Transport: HAVEit Approach. In Proceedings of the 15th World Congress on ITS, 2008.

[4] I. of Engineering and Technology. Autonomous vehicles from Mercedes, BMW and Audi debut at CES. http://eandt. theiet.org/news/2015/jan/autonomous-cars-ces.cfm, 2015.

[5] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. Mark. Connected Vehicles: Solutions and Challenges. IEEE Internet of Things Journal, Aug. 2014.

[6] E. T. S. I. ETSI. Intelligent Transport Systems Standards. http://www.etsi.org/index.php/ technologies-clusters/technologies/intelligent-transport, 2015.

[7] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 2010.

[8] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In CVPR Workshops, 2014

[9] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In ICML, 2014.

[10] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In NIPS. 2014.

[11] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. JMLR, 2016.

[13] Half-Life 2. http://www.valvesoftware.com/games/hl2.html.

[13] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. TPAMI, 2012.

[14] https://arxiv.org/pdf/1712.01397.pdf

[15] Play and Learn: Using Video Games to Train Computer Vision Models. https://arxiv.org/pdf/ 1608.01745.pdf.