Improving Human-Robot Interactions with Deep Reinforcement Learning

(Final Article)

Felipe Teixeira da Silva*, Aloïs Blarre*, Caroline Chanel†‡ and Nicolas Drougard‡
*Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: {Felipe.TEIXEIRA-DA-SILVA,Alois.BLARRE}@student.isae-supaero.fr
†Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: Caroline.CHANEL,Nicolas.DROUGARD@isae-supaero.fr

***Abstract*—The recent developments in machine learning, more specifically convolutional neural networks, fostered new applications in computer vision, mainly image recognition. Generative adversarial networks and furthermore deep generative adversarial networks are frames that enabled improvements when working with unlabeled data. Such algorithms found substantial results when applied to continuous real valued data. However, the development of frameworks to be applied in discrete data still lag far behind and are poorly discussed in the literature. In this paper we apply DCGAN in a database which contains discrete data and analyse its performance. A method to assess the framework results was specifically designed to the case of the "The Firefighter robot mission" to validate the results.**

I. Introduction

To pursue the study of the interaction of human and automated systems a huge amount of data is necessary, since the model of the control system that will manage this interaction must be able to perform properly in scenarios in which the actions prompted by the human are not always deterministic, i.e. the actions performed by the human are not predictable and sometimes difficult to model mathematically. To study how stress influences human behavior and to develop a model of when and how an automated model should take control over the action, the "The firefighter robot mission" game simulation was built. Through this environment, data about human making process was collected to further prompt the development of a supervision policy. The Markov Decision Process (MDP) was used to manage the interaction human-robot along the mission. The study showed that the data collected from humans only helps to improve the supervision policy, thus delivering better results. However, to collect this data a huge number of hours playing the game had been necessary.

Recent development in the field of machine learning prompted the investigation of the applicability of deep learning models to reduce the need of real hours of simulation, i.e. simulations with human beings playing the game.

Recently, Goodfellow [5] published his paper in which the generative adversarial network framework is introduced. The main idea behind such model is to have two different networks competing against each other. One is responsible for generating artificial data, while the second should discriminate whether the data presented is real or if it was artificially generated. The game reaches its ends when the discriminator cannot further discriminate whether the data came from the real set or from the generator. If we follow the notation used by Goodfellow in his paper and represent the scalar probability output of the discriminator by D(x), which means the probability that the image x is a real one. And let z be vector sampled from a gaussian law, the goal of the generator is to map G(z) to a generated data, thereby finding at the end one distribution that matches the one of the real images. The figure 1 below show the idea behind GAN.

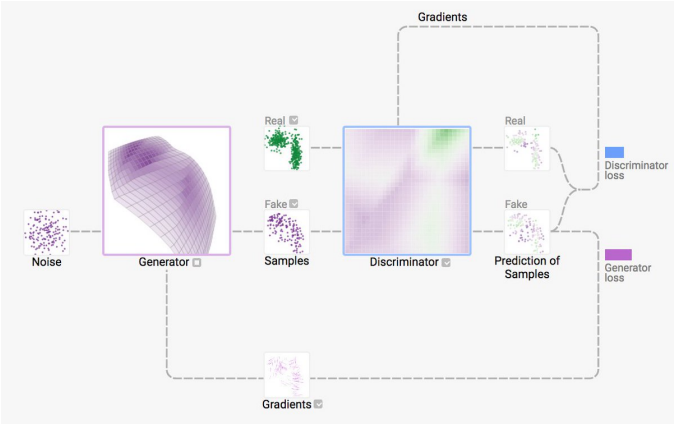

Fig. 1. An illustration of GAN's dynamic by Kahng et al. Available in:poloclub.github.io/ganlab/

The generator produces its data from a random distribution, and the discriminator tries to differentiate whether the data comes from the generator or from the real samples. From the results, the gradients indicating how both networks should be changed are calculated, i.e: how the generator should be updated to generate better data, and how the discriminator should be updated to better discern between real and fake data.

The function 1 mathematically describe the game played by the discriminator and the generator.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \quad (1)$$

It is important to highlight that GANs are only able to model continuous data [5]. GANs works by having the gradient of

discriminator's output calculated, therefore indicating how to change the generator's output to improve the model.Moreover, although thrilling the framework presented above is, it had some difficulties delivering realistic results and had suffered of instability in calculation.

GANs were further developed by Radford et al in [2], where the same framework is studied, but this time both the generator and the discriminator are made of convolutional neural networks, hence the name deep convolutional generative adversarial networks (DCGAN). Such model offers state of the art results when working with unlabeled data sets. In [4] a comprehensive study of DCGAN's was made, even intermediary results produced by each layer were studied.

II. Objectives

Major applications and studies related to DCGANs are restricted to the field of image processing and recognition and other applications still lags behind. The aim of this research project is to go a step further and investigate whether the recent deep learning algorithms can be applied to generate artificial results of simulations and evaluate its performances, thereby facilitating the development of solutions to other field than image processing.

More specifically, the question to be addressed is if such generative models will be able to perform within discrete data whose features are not as evident.

To assess the performance of the model, some metrics will have to be developed. Differently from images, numerical data set evaluations are not as straightforward.

III. Understanding the Data

A. Data acquisition

The data has been acquired through a campaign in which players joined a simulation that lasted maximum 10 minutes or until they lost the game. Figure 2 displays the interface of the "Firefighter robot mission".

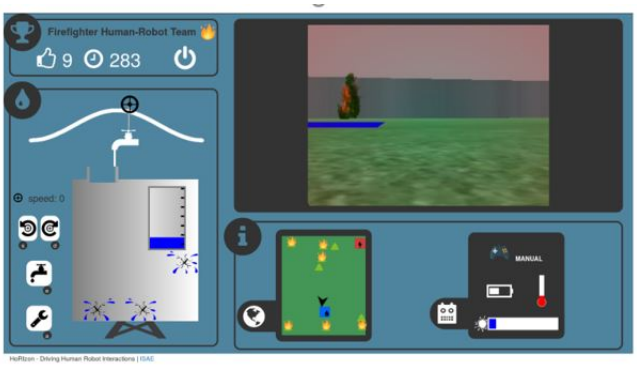

Fig. 2. Graphical user interface of the Firefighter Robot mission

The objective of the simulation is to acquire data to pursue the study on how stress and task load influence the performance of a human operator. The final goal is to have an optimal managing policy between human and autonomous systems.

The game consists of 9 trees that can catch on fire at any moment randomly. Moreover, the robot has a tank of water in which several leaks might appear (up to a maximum of 9) throughout the game. The main objective is to put out the fire, for this reason, attention must be payed to the water level inside the tank. Thereby the leaks have to be fixed during the simulation and water refilled. To have the tank filled, the robot must be at the water station - blue region seen in the picture.

There are two modes during the game, one autonomous in which the robot is controlled by the machine and another mode in which is the human is responsible for controlling the actions. The switch is made randomly every 10 seconds, following a uniform distribution.

Several actions can be made during the game: the robot is controlled, when in human mode, with the arrows. When near a tree on fire, the fire can be extinguished by pressing the space bar. When in autonomous mode, the human can only fix the leaks that appear on the tank. The robot is programmed to drive itself, shoot water on the trees and even recharge itself when necessary,

To have the tank fixed, the operator must click with the mouse on the leak and press "a". With respect to the water level inside the tank, it is indicated by a blue scale that can be seen in the left side of picture 2. More leaks the tank has, faster it will lose water. To get water inside the tank, when over the water station region, the tap can be activated by pressing "e". In addition, the tap must be at the peak of the curve seen in the Fig. 2 to be able to fill the tank. Keys "g" and "d" should be used to control the position of the tap by displacing it to the right or the left respectively.

Additionally, the robot's battery level and temperature must be monitored. If the battery level reaches zero the mission fails. To have the battery recharged, the robot must be at the charging station, the red corner seen in Fig. 2. The temperature is a major factor as well. When close to trees on fire, the temperature increases, if it passes a specific threshold, the mission fails.

B. Original Data

Every second of simulation is recorded in a ".csv" file. A total of 1191 games were played corresponding to over 85 hours of simulation. However it should be highlighted, that some of the games were not played until the end or the actions are not really representative of players who were engaged in the game. Therefore, some of the action performed by the humans must be seen with care when analyzing the results.

Each second, 16 different parameters are recorded. Some to describe the situation such as: the remaining time of simulation, the robot mode, the position and orientation of the robot, the state of the forest, battery level, robot's temperature, tank's water level, leak states, the ground station's water level. Then some parameter to describe the action took by the human, such as : keys pressed, clicks on the interface, number of errors - not valid keys - and number of shortcuts used.

To get a complete description of how the data was stored, why these parameters were collected and further information about the gathering of data or the game itself, refer to [3].

Every ".csv" file contains a header, thus the information of the nth second of simulation is stored at the line n+1. If no premature game over occurs, the file will have a maximum of 601 lines. The mode of the robot is accounted by a binary variable, 1 for autonomous mode and 0 for manual mode. The position and orientation $(x_t, y_t, \theta_t) \in ([-20, 20]^2 \times] - \pi, \pi])$. The forest state is stored in a binary string in which 1 stands for tree on fire. The leak states are stored in the same fashion, in a binary string in which 1 stands for leak in one of the respective possible leak's locations.

Picture 3 displays one extract from the dataset in which we can visualize all the parameters, the transition from the manual mode to the autonomous one.

	remaining_time	robot_mode	alarm	robot_x	robot_y	robot_theta	forest_state	battery_level	temperature	water_tank	water_gound_tank	leaks_state	keys	clicks	errors	shortcuts
12	590	1	-1	-0.36174	-10.2939	-1.599	100000000	95	20	70	30	0	front	-1	-1	-1
13	589	1	-1	-0.36174	-10.2939	-1.599	100000000	95	20	90	10	0	-1	left	-1	1
14	588	1	-1	-0.36939	-10.6781	-1.2	100000000	94	20	100	0	0	-1	-1	-1	-1
15	587	1	6	-0.36576	-10.6782	-0.709	100000000	94	20	100	0	0	-1	-1	-1	-1
16	586	1	-1	-0.36383	-10.681	-0.221	100000000	93	20	100	0	0	-1	right-right	-1	2
17	585	1	-1	-0.36214	-10.6822	0.259	101000000	93	20	100	0	100000	-1	rm_alarm	click-click	-1
18	584	1	-1	-0.36124	-10.6817	0.7387	101000000	92	20	100	0	100000	-1	push-push-left-left	-1	4
19	583	1	-1	-0.35978	-10.6823	1.2109	101000000	92	20	100	0.8016	100000	-1	-1	-1	-1
20	582	1	-1	-0.11835	-9.68327	1.357	101000000	91	20	100	4.0952	100000	-1	wrench	-1	1
21	581	1	-1	0.23907	-8.05468	1.3534	101000000	91	20	100	3.873	0	-1	push-leak_4-right	-1	1
22	580	0	-1	0.59919	-6.4251	1.3527	101000000	90	20	100	3.873	0	-1	right	-1	2
23	579	0	-1	0.95823	-4.7956	1.3582	101000001	90	20	100	3.873	0	-1	push	-1	1
24	578	0	-1	1.3916	-2.79099	1.3595	101000001	89	20	100	3.873	0	front-front-front-front-fr	-1	-1	-1
25	577	0	-1	2.00594	0.00607	1.3509	101000001	89	20	100	3.873	0	front-front-front-front-fr	-1	-1	-1
26	576	0	-1	2.62918	2.80559	1.3506	101001001	88	20	100	3.873	0	front-front-front-front-fr	right-right	-1	2
27	575	0	-1	3.25776	5.60415	1.3479	101001001	88	20	100	3.873	100000	front-front-front-front	-1	-1	-1
28	574	0	-1	3.64221	7.86202	1.4647	101001001	87	20	100	3.3175	100000	front-left-front-front	-1	-1	-1
29	573	0	-1	3.93981	10.7146	1.4646	101001001	87	20	100	2.7619	100000	front-front-front	-1	-1	-1
30	572	0	-1	4.22566	13.3766	1.4315	101001001	86	31.12	100	2.373	100000	front-front-right	-1	-1	-1
31	571	0	-1	4.22522	13.3763	1.2141	1001001	86	27.787	90	1.9841	100000	space	-1	-1	-1

Fig. 3. Original representation of the data as collected from the simulation

C. Data treatment

The raw data, as it was collected, cannot be directly understood by the DCGAN's current framework. Thus, the data had to be worked before we could use it.

The changes consisted mainly in normalization of the continuous variable to have them centered in zero.

Due to the DCGAN's gradient calculations, this framework is limited to work with continuous data, therefore a major step in our data treatment concerned how to adapt our discrete data such as the inputs from the human operator and the forest and leak's states.

With respect to the forest and leak's states, it has been decided to have the data split in nine different variables and have each one of them centered in zero. Moreover, since the states are assigned randomly accordingly to a uniform distribution, we will let DCGAN treat them as uniform variables, and if the output of the network is positive, the value one will be assigned at the end. On the other hand, if the output is negative, the value zero will be assigned at the end.

IV. DCGAN Implementation

The approach proposed by Nathan Inkawhich [6] was followed and the python script modified to attend the needs and characteristics of our dataset. Therefore, pytorch library was chosen to support our machine learning implementation.

The first part was to have the code modified so it could properly read our dataset which is composed of several csv files, as discussed above.

Most of the research on neural network is about finding the correct architecture and parameters so the train can occur in feasible time delivering suitable results. Thus, moving on to the architecture of the network, each parameter of the network had to be studied so it could later be tuned accordingly. Moreover the architecture of the network itself had to be adapted. These changes presented itself as quite a challenge, because the literature is scarce about such problems.

A. Parameters

1) Batch size: the term is vastly used in machine learning and refers to the number of training examples used in one iteration.

Taking into consideration the recommendations found in literature, mini-batches were used to train the discriminator: one mini batch which consists of real samples are passed forward through the network and their gradient calculated and the backward pass executed; another mini batch consists of generated samples which are forward passed through the discriminator and their gradient calculated as well. These gradients are then accumulated and propagated backward. Finally, the discriminator's optimizer is called. For the discriminator, we have their samples classified using the discriminator, then we compute the losses and the gradients. In the last step,the generator's optimizer is called.

From GAN's standpoint, the strategy described above is essential to avoid the generator to collapse and produce always the same output [1].

By using mini-batches instead of the whole samples available, we work with fewer samples at time, thereby requiring less memory capacity from the machine. A drawback is the lost of accuracy in the estimation of the gradient.

2) Number of Epochs: influences directly the quality of the final output, more epochs produced more sharp results [4]. However, if the model was correctly tuned, the results are expected to stabilize and no substantial differences should arise from further training.

3) Learning rate and beta: The learning rate is a major parameter and it influences directly how the network will perform. Throughout the implementation, Adam optimizer was adopted. Literature refers to standard values about 0.001 for the learning rate and $\beta = 0.9$ for Adam optimizer hyperparameter. However, we have found along our test, that these parameter did not apply well. At the end, we changed the learning rate to 0.002 and $\beta = 0.5$.

B. Discriminator's architecture

The discriminator is responsible for telling whether or not a given sample is real. Its input is a "csv" file.

The figure 4 show's the discriminator's architecture. Several different models were tested, however we could not tune it properly and the losses diverged in most of the cases. With the model presented below the model converged.

The layers used to build the model are vastly discussed in [6] and [4]. In this project, the layers were rearranged and some were added to give proper results.

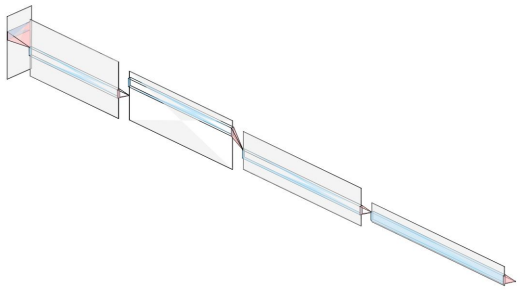

Fig. 4. Discriminator's network scheme

In the description below, "nc" stands for the number of channels in the input and "ndf" stands for the number of channels in the output of the first layer of convolution. Furthermore, the number inside parentheses is the $(kernel_{height}, kernel_{width})$. The stride were always 1 and no bias was considered.

1) nn.Conv2d(nc,ndf,(3,3),1,0,bias=False),
2) nn.LeakyReLu
3) nn.Conv2d(ndf,ndf·2,(3,1),1,0,bias=False),
4) nn.BatchNorm2d(ndf·2),
5) nn.LeakyReLU
6) nn.Conv2d(ndf·2, ndf·4,(3,1),1,0,bias=False),
7) nn.BatchNorm2d(ndf·4),
8) nn.LeakyReLU
9) nn.Conv2d(ndf·4, ndf·8,(3,1),1,0,bias=False),
10) nn.BatchNorm2d(ndf·8),
11) nn.LeakyReLU
12) nn.Conv2d(ndf·8,1,(2,1),1,0,bias=False),
13) nn.Sigmoid()

C. Generator's architecture

The generator is responsible for producing samples as close to the real one as possible. To do so it relies on the orientation provided by the discriminator. For this reason, both the genarator and discriminator should work together to improve network's results. Therefore, as we is currently adopted in the literature, the discriminator and generator have similar architectures.

The figure 4 show's the generator's architecture. As well as with the discriminator, several different models were tested, however we could not tune it properly and the losses diverged in most of the cases. With the model presented below the model converged.

The layers used to build the model are also discussed in [6] and [4]. In this project, the layers were rearranged and some were added to give proper results.

In the description below, "nz" stands for the size of the latent vector in the input and "ngf" stands for the number of channels in the output of the first layer of convolution. Furthermore, the number inside parentheses is the $(kernel_{height}, kernel_{width})$. The stride were always 1 and no bias was considered.

1) nn.ConvTranspose2d(nz, ngf·8,(2,1),1,0,bias=False),

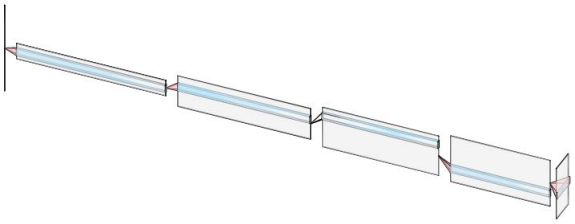

Fig. 5. Generator's network scheme

2) nn.BatchNorm2d(ngf·8),
3) nn.ReLU,
4) nn.ConvTranspose2d(ngf·8,ngf·4,(3,1),1,0,bias=False),
5) nn.BatchNorm2d(ngf·4),
6) nn.ReLU,
7) nn.ConvTranspose2d(ngf·4,ngf·2,(3,1),1,0,bias=False),
8) nn.BatchNorm2d(ngf·2),
9) nn.ReLU,
10) nn.ConvTranspose2d(ngf·2,ngf,(3,1),1,0,bias=False),
11) nn.BatchNorm2d(ngf),
12) nn.ReLU,
13) nn.ConvTranspose2d(ngf,1,(3,3),1,0,bias=False),
14) nn.Tanh()

V. Results and Discussion

We aim at applying DCGANs in a more complex situation, to generate a whole dataset including all parameters presented in the "Firefighter robot mission". Nevertheless, to tune such a model becomes complicated and we can easily lose track on were our model is having problems.

To avoid this problem, as a first approach to the problem we divided our dataset in two sub-datasets : autonomous and human to work them separately. This methodology facilitates when we need to assess the model's outputs, because the robot's movement, when in autonomous mode, is hard coded, thus easier to compare to our results. Differently, the movements made by a human operator are erratic and may not follow any logic, consequently more difficult to evaluate whether or not our model works properly.

A. First approach

In a first time, we used $1 \times 10 \times 18$ tensors representing 10 second sequences of gameplay as input for the discriminator. 16 of those columns represented the situation (robot position and angle, tree states, tank levels, temperature and battery level) and two columns represented the player's input (the robot direction and distance of displacement). On the other side, the generator was also given a $1 \times 10 \times 18$ tensor with 16 columns representing the situation and 2 random columns. The generator output a $1 \times 10 \times 2$ tensor representing the player's input which was then concatenated with the $1 \times 10 \times 18$ tensor representing the situation before being fed to the discriminator. We tried many different architectures for the generator and the discriminator but we couldn't get this first approach to work correctly, the generator's loss diverged every time.

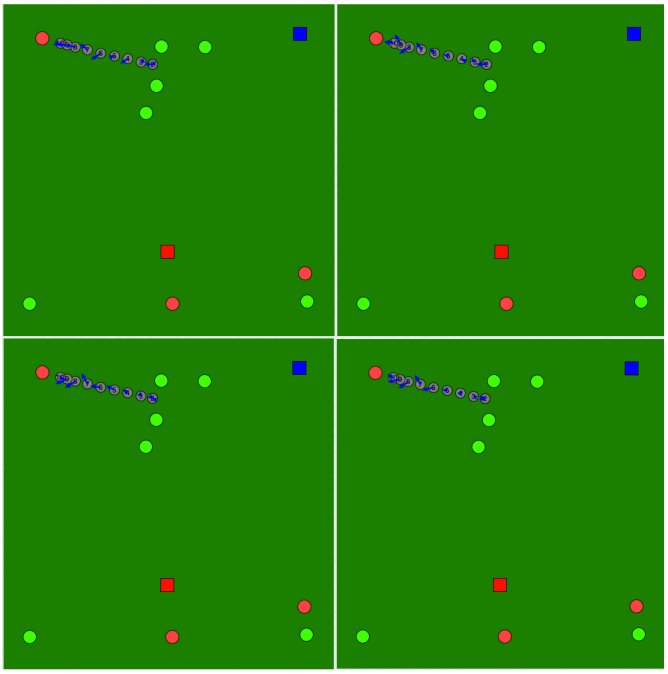

Fig. 6. From left to right, top to bottom : epochs 1, 5, 10, 15

The figure above represents the same 10 second sequence of the game at different epochs of training. The blue arrows represent the direction and distance output by the generator for this particular situation (with the same noise as input as well) for these different epochs. As we can see, the generated moves do not seem to improve with the epochs.

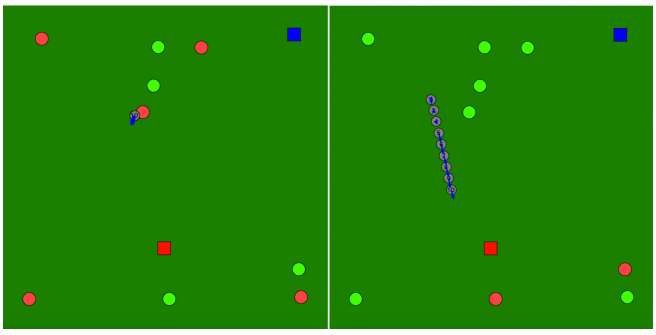

Fig. 7. Two situations after training

The figure above shows the output of the generator on two different situations after training. In the left image, the robot is stationary but the generator outputs a non-zero distance as robot displacement which clearly shows an inconsistency in the generator. On the right image however, the generator outputs a distance and a direction that match the trajectory of the robot.

As we did not manage to get satisfactory results with this approach, we tried an other, simplified, one.

B. Second approach

In a second time, we tried generating 10-second trajectories of the robot. To do so, we planned on tuning our DCGAN model on the autonomous data, so we could visually check that the generated data resembled the original data (autonomous trajectories are most of the time going straight towards a tree) and then using the same model on the human data, assuming it would work similarly to the autonomous data.

The architecture we chose for this approach was more standard than the one we used in our first approach. The input of the generator is a $100 \times 1 \times 1$ random tensor and the output is a $3 \times 10 \times 1$ tensor that can directly be fed to the discriminator. The 10 lines represent the 10 seconds of the simulation and the 3 dimensions represent the position of the robot (x, y, θ). To compare this architecture to the working DCGANs on images [2], we would be working on 10×1 images with 3 colors channels representing (x, y, θ).

Once again with this architecture we couldn't get satisfactory results, and the generator loss diverged every time, no matter which parameters we chose. The generated trajectory did not resemble the autonomous behavior as they were most of the time not straight and the robot orientation did not match the movement of the robot.

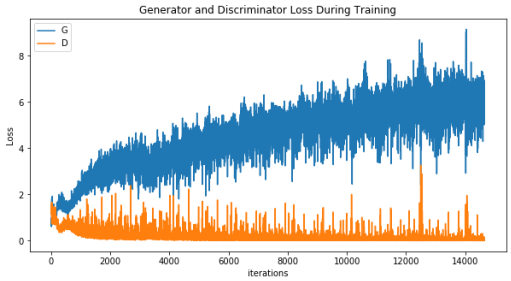

Fig. 8. Evolution of the loss of the generator and the discriminator throughout training

Fig. 9 is representative of the average generator's output : positions are grouped together in the middle right portion of the map, with no order and the orientation vectors do not match the movement.

C. Third approach

This latest approach is similar to the second approach but surprisingly gave better results. We here used used $1 \times 10 \times 3$ tensors instead of $3 \times 10 \times 1$ tensors (second approach) to represent (x, y, θ). Again, to compare this to an image, it would be equivalent to working with a 10×3 image with only one color channel. After tuning the model, we managed to get a stagnant loss for the generator.

The result we obtained were also much better, with straight and ordered trajectories in most cases. The main issue remains the orientation, which doesn't match the movement of the robot.

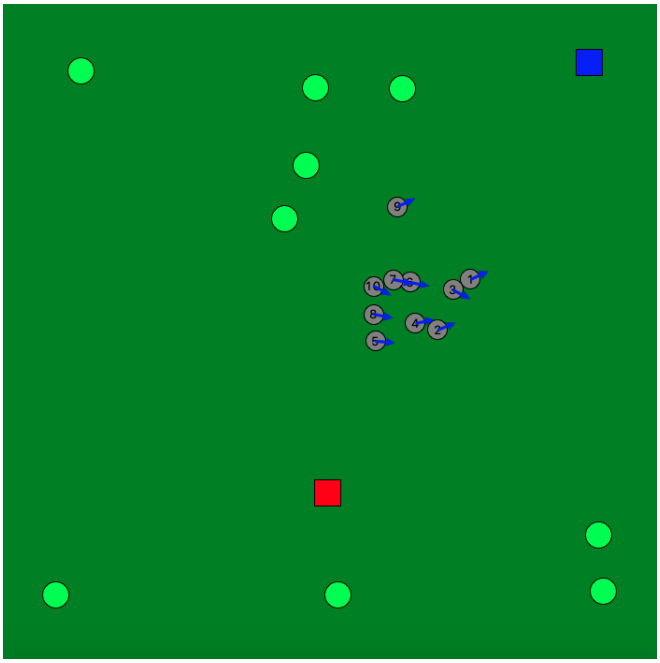

Fig. 9. Generated 10 second trajectory

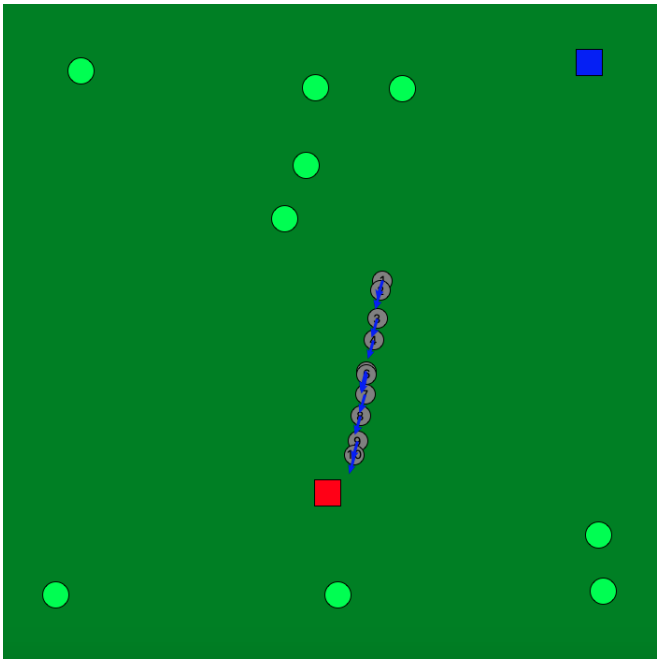

Fig. 11. Best 10 second trajectory generated

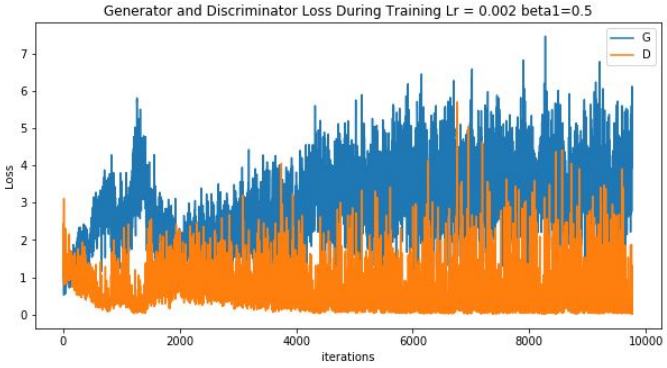

Fig. 10. Evolution of the loss of the generator and the discriminator throughout training

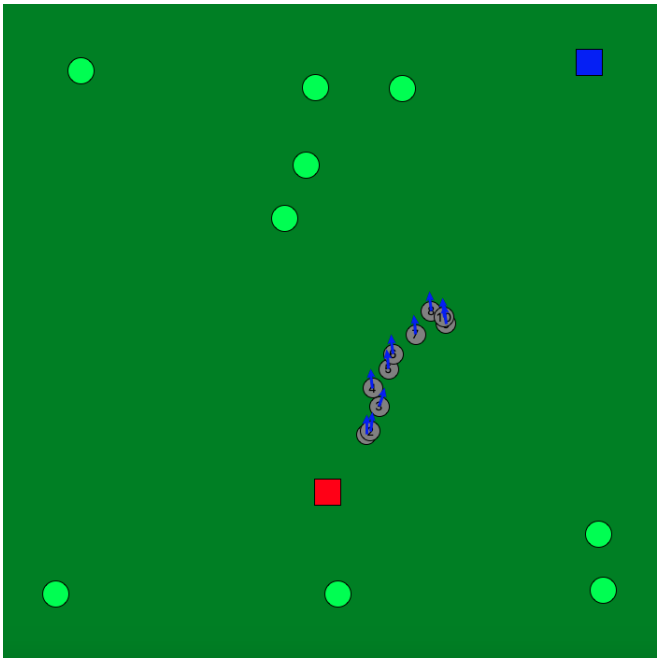

Fig. 12. Generated 10 second trajectory representing the average output of the generator

VI. Validation

One major challenge when it comes to validate our model is the lack of a proper metric to evaluates its performance. Most of the works that have applied GANs to up to this time, lies on a human operator to assess the end results. We decided to approach this problem in a similar fashion, and from our final output generate an image in which we reproduce the trajectory of the robot. Thereby, the final results can be evaluated by looking if the trajectory generated is coherent, that means, whether or not the points produced are aligned and if they followed some specific target, i.e the water ground tank, the energy station.

Moreover, the problem of not having a objective metric still remains, for that reason we have not conducted a formal study on how each parameter altered the quality of our network's output.

VII. Conclusion

The results obtained were not conclusive with respect to DCGAN's applicability. DCGAN's main difficulties arise in tuning the parameters accordingly to the problem.

Although not representative as a measure of the network's quality, we were able to find a configuration of parameters in which the losses stabilized after a few epochs. This is

an indicator showing that the network was able to learn something along the training. The case of 50% accuracy for the discriminator is hard to achieve, and should not necessarily be attended.

It should be highlighted that in one of the cases the network actually seemed to have memorized some points, instead of learning how to create new ones. This problem occurs when the gradient of the discriminator points to a single point making it difficult to the network to learn something other than these points.

The results obtained so far, show that in some cases the network is capable of delivering proper results. The last approach can be further developed to give trajectories that resemble the ones of the robot, because it offers the most straightforward approach to scale up the model to have all the variables of the simulation including modelling the human behavior.

With respect to the validation method, a visual resource offer the necessary information about the results to enable a human to assess the quality of the network's final output. However, due to the lack of a metric, we are not able to quantify how the evolution of each parameter influence model's results.

VIII. Future work

Starting from the work we have accomplished, it would be interesting to try and work on the DCGAN's parameters (batch size, numbers of feature maps, learning rate, ...) and the architecture of the generator and the discriminator. After many tries we've achieved a stagnant loss for the generator and we believe that it is possible to find parameters that achieve a loss converging towards zero.

It may also be very beneficial to study the gradient to expose where the learning issues come from and be able to solve them.

Once the right parameters found, we could, little by little, add parameters (such as tree state, leak state, remaining time, alarms, ...) and adjust the DCGAN's architecture at every step. When all the parameters are taken into account and the training is done on the autonomous dataset and validated (the robot goes towards trees in fire for instance), we can run the same DCGAN on the human dataset and assume that the output of the generator is representative of the human behavior.

Finally, when the model is validated for the autonomous case, we can generate the human behavior in any situation of the game and therefore run deep learning algorithms to determine when to set off alarms and when to put the robot in autonomous mode to improve the mission overall success.

Acknowledgment

The authors would like to thank DCAS group of ISAE- SUPAERO and Dassault Aviation for the opportunity of joining the research. And specially Mrs. Chanel and Mr. Drougard for the support along the project.

References

[1] Ian Goodfellow Vicki Cheung Wojciech Zaremba Xi Chen Alec Radford, Tim Salimans.

[2] Luke Metz Alec Radford and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[3] Jack-Antoine Charles, Caroline P. C. Chanel, Corentin Chauffaut, Pascal Chauvin, and Nicolas Drougard. Human-agent interaction model learning based on crowdsourcing. In *Proceedings of the 6th International Conference on Human-Agent Interaction - HAI 18*. ACM Press, 2018.

[4] Hendrik Strobelt Bolei Zhou Joshua B. Tenenbaum William T. Freeman David Bau, Jun-Yan Zhu and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. 1811.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets.

[6] Nathan Inkawhich. Dcgans faces tutorial, pytorch, 2017.