Machine Learning Engineer Nanodegree

Capstone Project

Rahim Kassanaly
June 12th, 2019

I. Definition

Project Overview

Drones are impacting our society in may ways. We can now accomplish tasks that weren't possible before or that required a lot human intervention. They have completely changed the way we take pictures or gather information of remote areas. Most of the drones are human controlled but with the rise of artificial intelligence we are entering a whole new world.

In this project I created a drone capable to tracking a person based on what it sees. Using the drone camera, it is able to detect a person and move accordingly to that person location. No other sensors are used other than the camera itself.

Problem Statement

For the drone to be able to perform such action it has to be capable of two functions. I'll illustrate this by comparing it to a human logical reasoning.

Let's assume that person A wants to walk towards person B. First, person A has to locate person B in his environment and be able to know where that person is in regards to him. Let's call this function object detection. Once the object of interest is detected, person A has to "activate" his legs to start moving to person B. While walking, person A relative position to person B is changing continuously, so it has to adjust if he's going too slow or going too much to the left, right, etc. Let's call this function motion control.

The drone has the same problems to solve, object detection and motion control.

Let's break out theses two problems and their respective solution:

- Object detection: The drone is equipped with a 720p camera. Using an artificial neural network, specifically convolution networks, we can detect the presence of an object in a picture. With recent architecture like YOLO, we even have an extra information that tells us where that object in located in the picture. It's important to understand that the computer has to have a way to calculate his position in regards to the object of interest to know where the drone should go.
- Motion control: Once we know where the object is located is the picture, we have to tell the drone to move in order to center that object in his camera. We can command the drone to go clockwise, counter-clockwise, up, down, forward or backward.

Metrics

Two metrics were used to determine the solution model:

- Accuracy of the drone position in regards to the center of the object. A squared distance between the coordinate of the drone and the target.
- Time to converge to the target.

I choose these two metrics because they represent what is observable by a human. When the drone is making actions, we can observe how fast it is getting there and when it reaches its final position, we also observe how centered it is to the object of interest. This project is like a selfie drone where we want to center a person while doing it quickly.

II. Analysis

Data Exploration and Visualization

This section will be split in two subsections, one for data of object detection and the other one for the motion control.

Object detection – VOC Dataset

The Pascal VOC challenge is a very popular dataset for building and evaluating algorithms for image classification, object detection, and segmentation.

The dataset contains twenty object classes:

• person

• bird, cat, cow, dog, horse, sheep

• aeroplane, bicycle, boat, bus, car, motorbike, train

• bottle, chair, dining table, potted plant, sofa, tv/monitor

Each record is composed of an image and an xml file that contains information about it. Annotation were done manually by the creator of this dataset.

Details on how this information are used by the model will be described in subsequent sections.

Here is a sample of annotation xml file:

```
<annotation>
                        <folder>Train</folder>
                        <filename>000001.png</filename>
                        <path>/my/path/Train/000001.png</path>
                        <source>
                                <database>Unknown</database>
                        </source>
                        <size>
                                <width>224</width>
                                <height>224</height>
                                <depth>3</depth>
                        </size>
                        <segmented>0</segmented>
                        <object>
                                <name>21</name>
                                <pose>Frontal</pose>
                                <truncated>0</truncated>
                                <difficult>0</difficult>
                                <occluded>0</occluded>
                                <bndbox>
                                        <xmin>82</xmin>
                                        <xmax>172</xmax>
                                        <ymin>88</ymin>
                                        <ymax>146</ymax>
                                </bndbox>
                        </object>
                </annotation>
```

Each xml file refers to a single image. An xml file will list all the classes that are present in the picture and indicate clearly the coordinates of the bounding box of objects in the picture as shown in Figure 1 below.

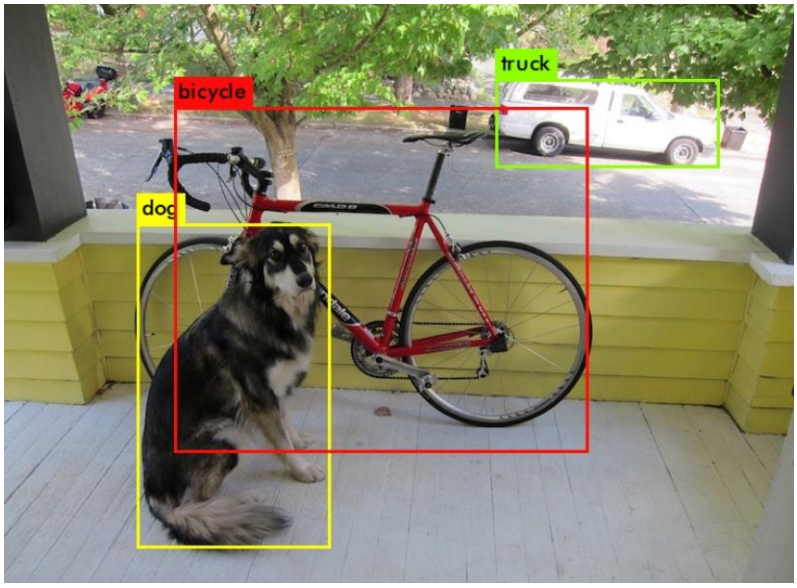


Figure 1: Object detection

The table below shows statistics about different classes :

Table 1: VOC Dataset Statistics

	train		val		trainval		test	
	Images	Objects	Images	Objects	Images	Objects	Images	Objects
Aeroplane	327	432	343	433	670	865	-	-
Bicycle	268	353	284	358	552	711	-	-
Bird	395	560	370	559	765	1119	-	-
Boat	260	426	248	424	508	850	-	-
Bottle	365	629	341	630	706	1259	-	-
Bus	213	292	208	301	421	593	-	-
Car	590	1013	571	1004	1161	2017	-	-
Cat	539	605	541	612	1080	1217	-	-
Chair	566	1178	553	1176	1119	2354	-	-
Cow	151	290	152	298	303	588	-	-
Diningtable	269	304	269	305	538	609	-	-
Dog	632	756	654	759	1286	1515	-	-
Horse	237	350	245	360	482	710	-	-
Motorbike	265	357	261	356	526	713	-	-
Person	1994	4194	2093	4372	4087	8566	-	-
Pottedplant	269	484	258	489	527	973	-	-
Sheep	171	400	154	413	325	813	-	-
Sofa	257	281	250	285	507	566	-	-
Train	273	313	271	315	544	628	-	-
Tvmonitor	290	392	285	392	575	784	-	-
Total	5717	13609	5823	13841	11540	27450	-	-

For this project, only the person class will be of interest. As we can see in the “trainval” section for the class “person”, we have 4087 pictures that contains 8566 persons in them.

Motion Control – Environment

The picture frame received from the drone is a resolution of 960x720 pixels. The center of the frame is always x=480, y=360.

Depending of where the person is located in the picture, we can calculate the center of the bounding box found by the model.

Using the difference between the center of the screen and the center of the bounding box we obtain a relative position that we can use to train a reinforcement learning agent. Please note that x and y only track orientation and altitude of the drone in regards to the target. The distance (z) between the object can’t be found using x and y. To obtain a measure of the distance I used the area of the bounding box. As the object gets closer the area of the box will get bigger and vice versa. The reinforcement learning agent has been only trained to give x, y commands. The distance (z) component is controlled separately with a simple if condition for this project.

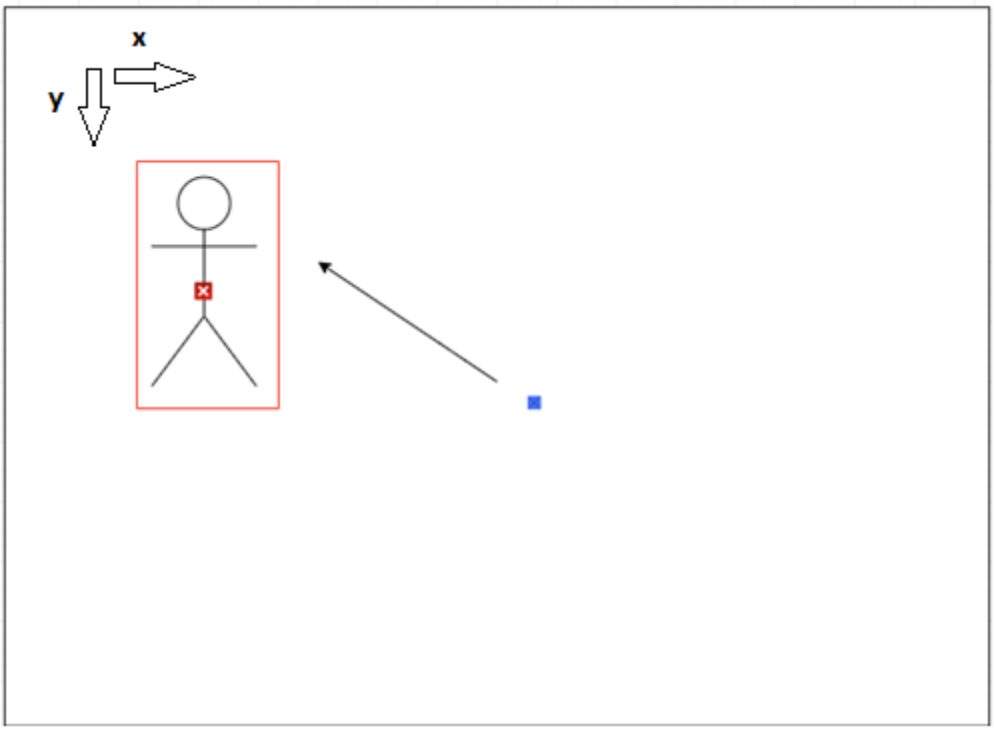


Figure 2: Drone environment

For the training phase I used an environment with the following parameters:

- Observations space: x=0, y=0 ➔ x=960, y=720
- Continuous actions space: -60 < A1 < 60 and -60 < A2 <60.
- Action 1 control the yaw movement and Action 2 controls altitude up or down movement.
- The drone supports speed in each direction that range from -100 to 100.
- I defined areap that is the percentage of the area that the bounding box takes comparing to the full camera screen (areap=area/ (960*720)) *100

Algorithms and Techniques

Again, this section will be split in two subsections, one for the object detection problem and the other one for the motion control.

Object detection – YOLOV3 Architecture

The following information is an extract of the great article of Jonathan Hui about the YOLO algorithm [1] For more information about YOLO, please consult the link in reference.

You Only Look Once (YOLO) is an object detection system targeted for real-time processing.

YOLO divides the input image into an S×S grid. Each grid cell predicts only one object.

For each grid cell,

- it predicts B boundary boxes and each box has one box confidence score,
- it detects one object only regardless of the number of boxes B,
- it predicts C conditional class probabilities (one per class for the likeliness of the object class).

To evaluate VOC dataset, YOLO uses 7×7 grids (S×S), 2 boundary boxes (B) and 20 classes (C). Each boundary box contains 5 elements: (x, y, w, h) and a box confidence score. The confidence score reflects how likely the box contains an object and how accurate is the boundary box. Each cell has 20 conditional class probabilities. In summary, YOLO's prediction has a shape of (S, S, B×5 + C) = (7, 7, 2×5 + 20) = (7, 7, 30).

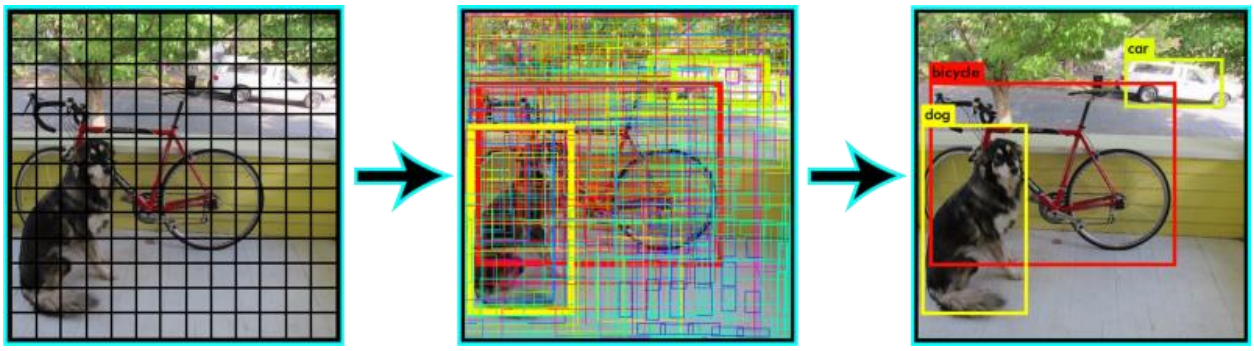


Figure 3: YOLO prediction steps

YOLO will take each 7x7 windows and issue a prediction for each classes and box coordinates. Once all the picture is seen we keep those with high box confidence scores (greater than 0.25) as our final predictions (the right picture). The class confidence score for each prediction box is computed as:

Class confidence score = box confidence score x condition class probability

It measures the confidence on both the classification and the localization (where an object is located).

The following table presents the architecture of the YOLO V3 implementation:

Table 2: YOLOV3 Architecture

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

YOLO is a proven and complex algorithm. We can see that it is outperforming most of his competitors on the COCO dataset:

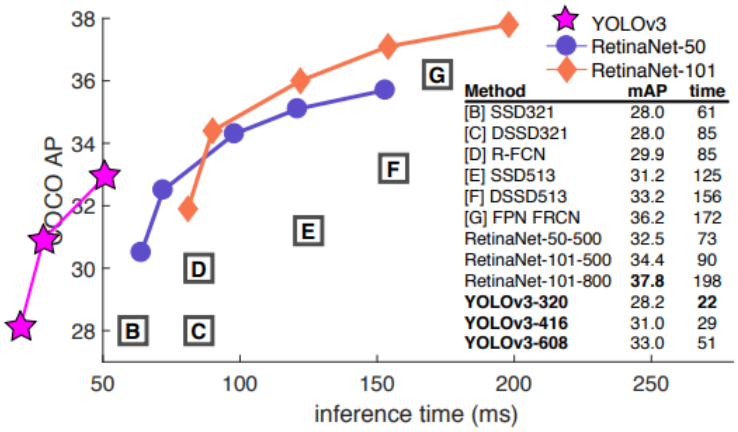


Figure 4: YOLO V3 performance

Motion Control – DDPG Agent

"Deep Deterministic Policy Gradients" (DDPG) is a policy gradient algorithm in reinforcement learning. It uses the actor-critic model as depicted in Figure 3.

In simple words, we have an Actor model that tries different actions that are rated by the Critic model. Some noise is added when the action is chosen to allow exploration and discover states that could have been missed otherwise.

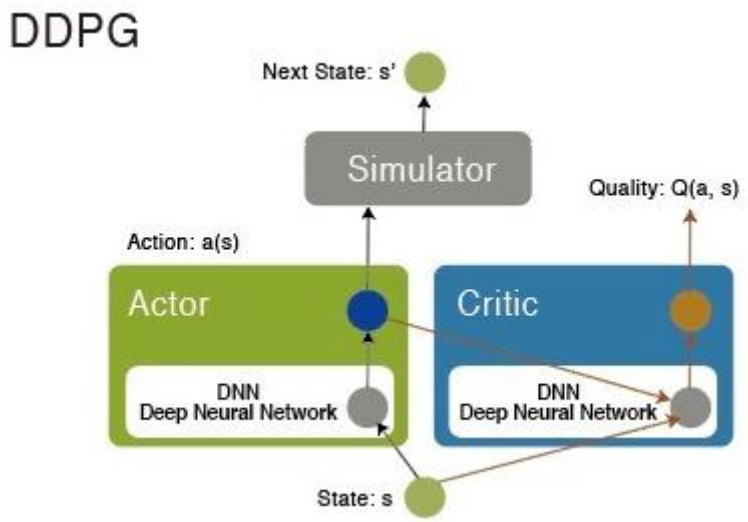


Figure 5: DDPG Algorithm

To train this agent, we need to define a reward function that will let the model know if its going in the right direction. The reward function will have as input the coordinate x and y of the detected bounding box. Details of the reward function will be added in subsequent sections.

Here are the hyperparameters that are available during training:

- Actor & Critic learning rate: gradient learning rate in actor and critic neural networks
- Tau = defines how fast target networks are updated
- Gamma = discount factor
- theta=exploration noise
- mu= exploration noise
- sigma= exploration noise
- nb_max_episode_steps: number of steps before an episode is deemed done
- Number of episodes

Benchmark

The benchmark as described in the proposal is a human that controls the drone.

"videos/Benchmark_test.mp4" video file in the submission shows a human pilot trying to track the human in the picture. From the moment the human is detected (red box) to the time it is centered to the camera took about 15 seconds.

Final position was:

x = 500 and y=420, area_p = 20

I did this test as fast as possible with minimal direction changes.

These results give an error of 4% on x and 16% on y if we compare it to the target position of x=960, y=720.

III. Methodology

This section will be split in two subsections, one for the object detection problem and the other one for the motion control.

Data Preprocessing – Object detection

The VOC dataset itself is already clean and each image is associated with an annotation file.

At the input of the YOLO network, the following preprocessing function is applied to images:

```
def preprocess_input(image, net_h, net_w):
    new_h, new_w, _ = image.shape

    # determine the new size of the image
    if (float(net_w)/new_w) < (float(net_h)/new_h):
        new_h = (new_h * net_w)//new_w
        new_w = net_w
    else:
        new_w = (new_w * net_h)//new_h
        new_h = net_h

    # resize the image to the new size
    resized = cv2.resize(image[:,:,::-1]/255., (new_w, new_h))

    # embed the image into the standard letter box
    new_image = np.ones((net_h, net_w, 3)) * 0.5
    new_image[(net_h-new_h)//2:(net_h+new_h)//2, (net_w-new_w)//2:(net_w+new_w)//2, :] = resized
    new_image = np.expand_dims(new_image, 0)

    return new_image
```

Each image is resized to the input size of the neural network, from a minimal input size to maximal input that is defined in the configuration. Each pixel value is divided by 255 and finally the resulting array in embedded in letter box.

Data is split with 80% for training and 20% for validation.

Data Preprocessing – Motion Control

During the training of the agent, random location of the bounding box are generated, the only preprocessing is to make sure that these values are within the picture boundaries as following:

$$0 < x < 960 \text{ and } 0 < y < 720$$

Implementation – Object Detection

YOLO original model is built in C++ and can only be executed via command line. Since I wanted to grasp in more depth this architecture in a version that I could understand, I reused a YOLOv3 Keras implementation [3]

It is an implementation that have a train and predict function as well as a configuration to manipulate hyperparameters.

Training & Refinement

This the YOLO configuration file that described the different parameters used for training:

- Image input size
- batch_size
- learning_rate
- nb_epochs

With minimum input size of 220 to a maximum of 416:

Nb Epoch	Batch Size	Learning Rate	Average Precision	Early stop (epoch)
100	8	1e-3	0.51	20
100	16	1e-4	0.58	12
100	16	1e-5	0.55	23

With minimum input size of 416:

Nb Epoch	Batch Size	Learning Rate	Average Precision	Early stop (epoch)
100	8	1e-4	0.71	7

I used a virtual machine on Google Cloud to train this model only on the "person" class since this is the only one, I am interested in for this project. Each training exercise range from 4 to 24 hours depending on the input size. I use a preemptible GPU on my VM so my training would be interrupted, and I would have to start over. After giving it, a few tries with the hyperparameters described and changing the input size to 416 I got 0.71 which is decent enough for this project. However, I decided to use a pretrained weights on the entire 20 classes VOC dataset that gives a slightly better 0.78 average precision on the person class. The only tweak was to filter out only predictions of the "person" class.

Once the training done, I imported the weights from the Google Cloud virtual machine to my laptop which has a medium graphic card. Running the YOLO algorithm on a GTX 1051 Ti gives approximately a 250ms detection time which is slow for a real time project. To prevent the detection from slowing down the main loop I executed the YOLO algorithm from a parallel thread that sends detected boxes though a queue. Theses detected boxes appear approximately every 250ms per second frames on a 25 FPS video feed.

Implementation – Motion Control

The drone is controlled via Wifi using UDP commands. I inspired my code from Damien Fuentes [3] GitHub, a python wrapper of the Tello drone. I won't go into details of this library but it allows us to control the drone by sending commands every 50ms. I modified the code and adjusted some parameters to fit my needs for this project. Each direction can be commanded with a value between -100 and 100.

Training & Refinement - Simulation:

For an agent to be trainable, it needes an environment that has basic functions step and reset. I reused the "Continuous Mountain Car" environment from OpenAI Gym.

Step function:

- update the position with the input actions to generate next state.
- Calculate the reward
- Determine if final state (done)
- Return next state, reward and done status

Reset function:

- Reset position to random values within boundaries.
- Initialize internal states, done status

For the DDPG agent I used the keras-rl library [4].

I used the squared distance (dist) of the center of the bounding box and the center of the screen (480, 360) as metrics for the reward function.

Final reward function:

- >100 ➔ reward -= dist*0.25
- dist < 100 ➔ (100 - dist)

These are the conditions to complete an episode:

- Number of steps exceeded 10
- x or y is out of bound (above 960x720)

I had to tweak the reward function a lot in order to obtain good result. It was mostly trial and error. It really helped to plot the position of the bounding box during training using the gym visualization function.

The figure below shows different tries with the done condition and the distance. The orange shows the reward moving average over 100 steps.

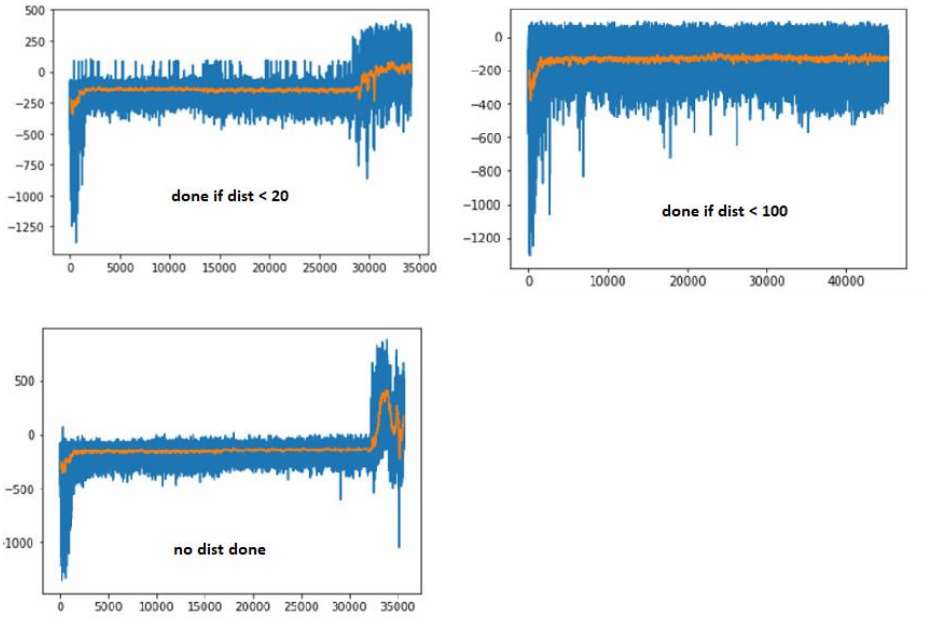


Figure 6 : Reward vs steps

When I added a done status with a distance requirement, the model did not explore states below that number. I needed the model to be able to choose action even really close to the target. I ended choosing not adding a done status with the distance.

Here are the final hyperparameters that were used during training:

- Actor & Critic learning rate: lr=.001
- Tau = 1e-3
- Gamma = 0.99
- theta=0.15
- mu=0
- sigma=0.3
- nb_max_episode_steps=10
- number of episodes: 10000

Distance Control

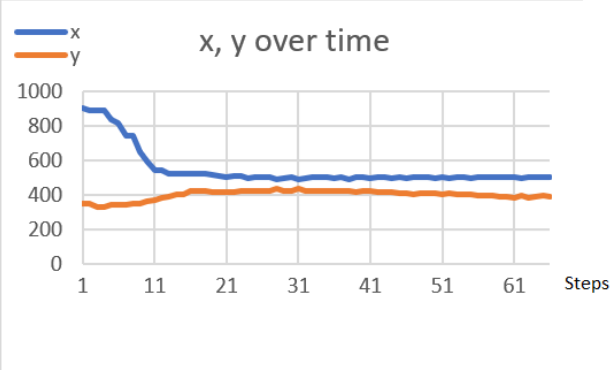
To control the distance, I used the area of the bounding box as metric. As an object closer, the bounding box gets bigger, so it is a good indicate on how the drone is close to the target. As said before, I used I simple if condition to control this:

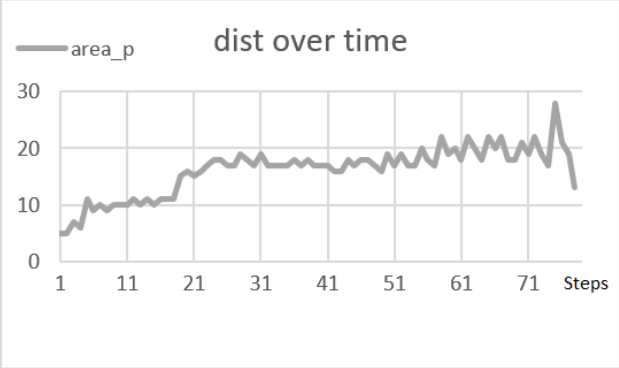
- If area below 25% of total screen size ➔ go forward
- If area above 50% of total screen size ➔ go backward
- If area between 25% and 50% ➔ don't move.

IV. Results

Model Evaluation and Justification

I realised a test under the exact same condition as the benchmark. Refer to the video file "videos/real_test.mp4" that illustrates this test. The drone converges quickly to the target (under 7 secs). The final error is 4.3% on x and 11% on y. Distance is not measure against the benchmark but we can see that is it below 25 as defined earlier. We can say that this model is performing better than the human agent. I am really satisfied with theses results even though the latency caused a lot of issues.





The result is sensitive to the speed of the drone because of the latency introduced by the hardware. I found that the best maximal speed is 60.

The drone behavior is not perfect due to hardware limitation, but results are still quite impressing, and the drone is actually able to identify a human and track it.

V. Conclusion

I added a free test video file in the submission (videos/free_test.mp4). I tested the drone under different conditions and angles.

As we can see in the video the object detection is accurate but there is some latency due to my poor graphic card. The drone is quickly converging to the target but because of the slow response, it is not able to adjust in real time and often cross the target to come back after. The drone is sometimes oscillating at the target, trying to maximize the reward as best as it can. If we removed the jitter that happens because results are satisfying for this level of hardware.

Reflection

The operation of this project can be summarized as followed:

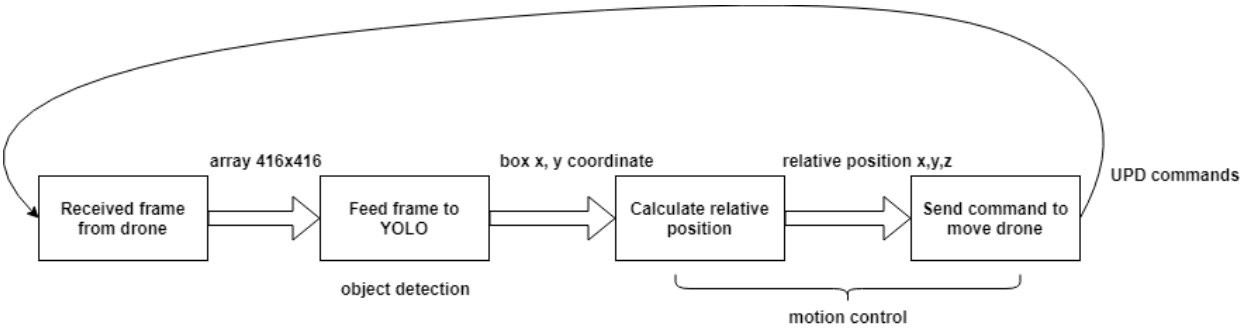


Figure 7: System flowchart

One of the difficult parts if this project was the setup. Since the drone does not have enough processing power, the "brain" has to be on the laptop which introduce a latency.

This project has taught how to use a Linux cloud computing machine and how-to setup a whole CUDA - Keras GPU environment for AI training purposes.

It is impressive to see how the YOLO algorithm is performing and how it has changed the game for object detection project.

Using reinforcement learning seemed a bit overkill for this project as it could have been done with simple if condition. DDPG algorithm basically built a table that associating a relative position with corresponding actions, it was interesting to see how easy it was to implement it using keras-rl libraries.

Improvement

I used a 100$ drone for this project. This project could have got much better results with a high precision drone. As said before, the latency between the drone and the laptop really affected this project, so here are two aspects that could improve this project:

- Real-time drone, i.e higher network speed and computation speed.
- Better graphic card on the laptop to reduce the inference time of detection.

With theses two changes, the drone would know faster the effect of an action. Currently it takes about 0.5sec for the feedback to come which is not considered real-time.

References

[1] : YOLO algorithm explained : https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[2]: A Keras YOLOv3 implementation : https://github.com/experiencor/keras-yolo3

[3]: Damien Fuentes Tello SDK : https://github.com/damiafuentes/DJITelloPy

[4]: Keras reinforcement learning library https://keras-rl.readthedocs.io/en/latest/agents/ddpg/