ANN design and implementation for real-time object tracking using quadrotor AR.Drone 2.0

Boufjit, Kamel; Larbes, Cherif; Ramzan, Naeem

Published in:
Journal of Experimental & Theoretical Artificial Intelligence

DOI:
10.1080/0952813X.2018.1509896

E-pub ahead of print: 30/08/2018

Document Version
Peer reviewed version

ANN Design and Implementation for Real-Time Object Tracking Using Quadrotor AR.Drone 2.0

BOUDJIT KAMEL[1], LARBES CHERIF[2], NAEEM RAMZAN[3]
[1] Université des Sciences et de la Technologie Houari Boumediene USTHB, Algiers, Algeria.
kamel.boudjit@g.enp.edu.dz kboudjit@usthb.dz
[2] Ecole Nationale Polytechnique, l'ENP, Algiers, Algeria.
cherif.larbes@g.enp.edu.dz
[3] University of the West of Scotland UWS, Scotland, United of Kingdom.
naeem.ramzan@uws.ac.uk

Abstract. Real-time object detection is crucial for many applications of Unmanned Aerial Vehicles (UAVs) such as reconnaissance and surveillance, search-and-rescue, and infrastructure inspection. In the last few years, Artificial Neural Networks (ANNs) have emerged as a powerful class of models for recognizing image content, and are widely considered in the computer vision community to be the de facto standard approach for most problems. This paper aims to use a visual- based control mechanism to control a quadrotor, which is in pursuit of a target. The nonlinear nature of a quadrotor, on the one hand, and the difficulty of obtaining an exact model for it, on the other hand, constitute two serious challenges in designing a controller for this UAV. A potential solution for such problems is the use of intelligent control methods such as those that rely on artificial neural networks.

A novel technique based on Artificial Neural Networks (ANNs) is proposed in this work for the identification and tracking of targets. Multilayer Perceptron (MLP) is used for this purpose. Experimental results and simulations are shown to demonstrate the feasibility of the proposed method for target tracking.

Keywords: Command and control systems, Artificial intelligence, intelligent systems, Image recognition.

1. Introduction

In recent years, the use of multi-rotor unmanned aerial vehicle (UAV) has gained popularity, especially the four-rotor UAV commonly known as the quadrotor. This is because the multi-rotor UAV is able to perform vertical takeoff and landing, hover and possesses exceptional agility [1, 3]. These abilities allow the multi-rotor UAV to perform a diverse range of tasks such as rescue, surveillance, inspection and military operations

Target tracking is of great interest for applications in diverse areas, such as surveillance system, entertainment, automation and manufacturing, only to mention few [4, 7]. The artificial vision system is based on computational intelligent techniques inspired by biological neural model of human beings [8, 11]. Well known as Artificial Neural Networks (ANN), these techniques are mainly characterized by their ability to learn through experiences, to adapt to adverse conditions, and to be tolerant to noise. These features make the ANN a great field of application study and lead it to reach success in solving real problems such as classification, identification, and digital image processing [12, 15]. To track a certain target, two tasks are required to be implemented. The first one is to recognize the target among many kinds of objects. Another one is to determine the relative pose of the target such as relative distance and angle. The difficulties in tracking objects result from a number of factors [16, 19]:

- Fast maneuvers of the targets;
- Noise in images and communication between camera and receiver;
- Multiple tasks switching such as obstacle avoidance while tracking a target;
- The requirement for following target after UAV comes close to target;

- Real-time data processing.

In this paper, an original method is proposed for robust visual tracking. It is based on a neural network controller. This application uses the visual information from the on-board camera located inside the quadrotor AR.Drone 2.0. The proposed algorithm is based on the detection of the target (tag) and could be used as well to detect objects with different shapes in real-time. The control architecture proposed in this paper was also tested with Multilayer Perceptron (MLP) neural networks. The proposed algorithm is based on a small number of input parameters, what results in low requirements of computational power. Simulations and experimental results are shown to demonstrate the feasibility of our proposed method for target tracking.

This paper presents the formulation of a tracking neural network to allow an unmanned aerial vehicle (UAV) to track a target. Potential application of this scheme lies in missions like surveillance and reconnaissance. The key contributions of this paper are as follows:

- Object detection based on ANN is extremely computationally demanding, typically requiring high-end Graphics Processing Units (GPUs) that require too much power. In our application, experiments on real data shows promising results obtained on a standard CPU, and pave the way towards future research in this direction.
- Target tracking is based on a simple neural network MLP that can be easily generalized to track any visual pattern in dynamic scenes.
- This work aims to use a visual- based control mechanism to control a quadrotor type aerial robot, which is in pursuit of a target. The nonlinear nature of a quadrotor, on the one hand, and the difficulty of obtaining an exact model for it, on the other hand, constitute two serious challenges in designing a controller for this UAV. In addition to the two mentioned problems, another problem that emerges due to the nature of a target is the uncertainty that exists in the target image. By employing an artificial neural network, neural controller MLP has been designed for a quadrotor in search of a target.

2. **The proposed method**

The target tracking system presented in this project was developed by using the Robot Operating System (ROS) framework. The ANN algorithm was written as a ROS node, which receives state information from camera sensors, trains the neural network using the input state and sends velocity and translation commands to a Quadrotor AR.Drone. The simulation environment Gazebo was used to evaluate the behavior of the ANN algorithm in a physical simulation. The RViz package was used to visualize the current environment state during simulation, including the detection of feature markers.

Since the focus of this project consists of object recognition, the vision problem was simplified by using AR tags instead of complex visual features. The AR tags could be placed in the simulation world and reliably tracked by using the ar_track_alvar ROS package. This package can provide reliable pose estimation for multiple detected tags and their associated tag IDs. The detection states for the tags were used as input states to the neural network. In order to simulate the problem of detecting a target given visual features in the environment, the AR tags were used in (feature, target) pairs. Certain tag pairs were assigned to be valid for target discovery. To recognize the border at every conventional marker, a marker detection algorithm is required. ar_track_alvar has been developed and is widely used as a representative method for marker-based tracking. The proposed method is simple and requires small computation power. After we got the data, we developed a system insuring object detection and tracking. Figure 1, shown below, illustrates the development process of the achieved system.

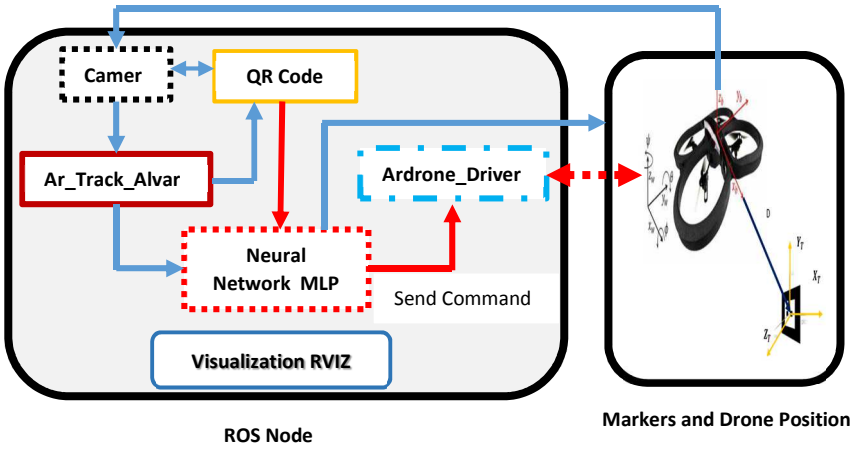


Figure 1. Structure of distributed control system software implementation based on ROS.

The platform runs under Robot Operating System (ROS). ROS is an open source framework for developing robotic control systems. The framework facilitates communications between different modules by using a centralized message passing interface. Multiple modules can be used as components for a larger system.

2.1. Design of the Neural Network

Design of a neural network involves the selection of its model, architecture, learning algorithm, and activation functions for its neurons according to the need of the application [20 – 23]. The objective of our application is to locate and tracking a specific target (tag).

For our work, we use a neural network type MLP (Multilayer Perceptron). MLP and many other neural networks learn to use an algorithm called back propagation. With back propagation, the input data is repeatedly presented to the neural network [24 – 28]. With each presentation, the output of the neural network is compared to the desired output and an error is computed. This error is then sent back to the neural network and used to adjust the weights such that the error decreases with each iteration of the operation and the neural model gets closer and closer to producing the desired output. Thus, for our work a BPNN is selected for the application at hand, and it consists of three layers: one input layer (of source nodes), one hidden layer (with sigmoid activation function), and one output layer.

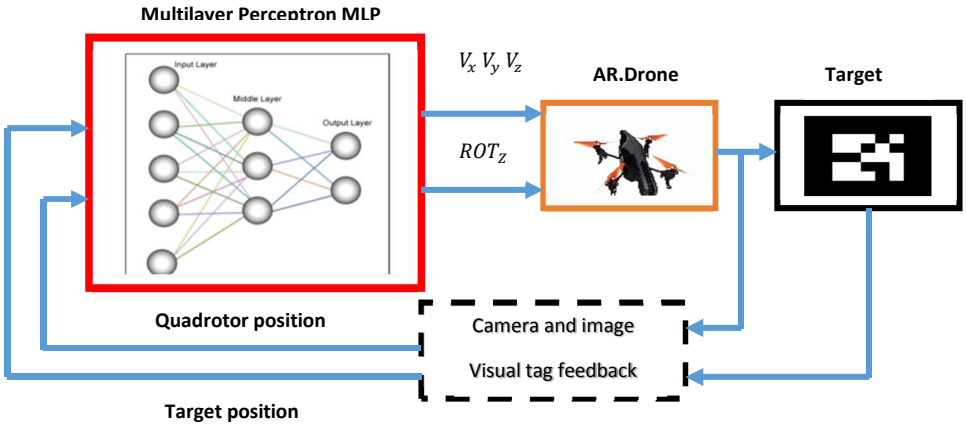


Figure 2. Architecture of the proposed neural network.

For our application, the learning algorithm has two phases.

- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output layer generates the output pattern.

- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.

2.1.A. Input Layer

The input layers of a neural network is determined by the characteristics of the application inputs. In our work, the input layer is the information received by the front camera of the drone and represent the Euclidean position of the target with respect to the drone. These data are described by the following values:

- Pose.Position X: Represents the angle of orientation of the target with respect to the drone. This value is zero if the target is on the x line of the drone. It is negative if the target is to the left of the x line of the drone and it is positive if the target is to the right of the x line of the drone.
- Pose.Position Y: Represents the height of the target relative to the drone. This value is zero if the target is at the same height as the drone. It is negative if the target is above the drone, and it is positive if the target is below the drone.
- Pose.Position Z: Represents the distance that separates the target from the drone. This value is always positive. Its variation depends on the movements of the drone.

Figure 3, represents Euclidean reference points of the drone relative to the target.

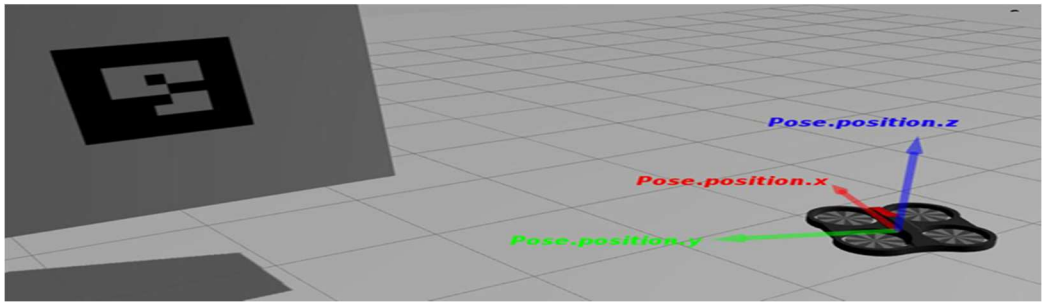


Figure 3. The Euclidean landmarks of the drone.

- Pose.Orientation X: Represents the orientation of the target with respect to the drone, along the x-axis in quaternion.
- Pose.Orientation Y: Represents the orientation of the target with respect to the drone, along the y-axis in a quaternion.

Figure 4, shows the rotation reference points of the drone.

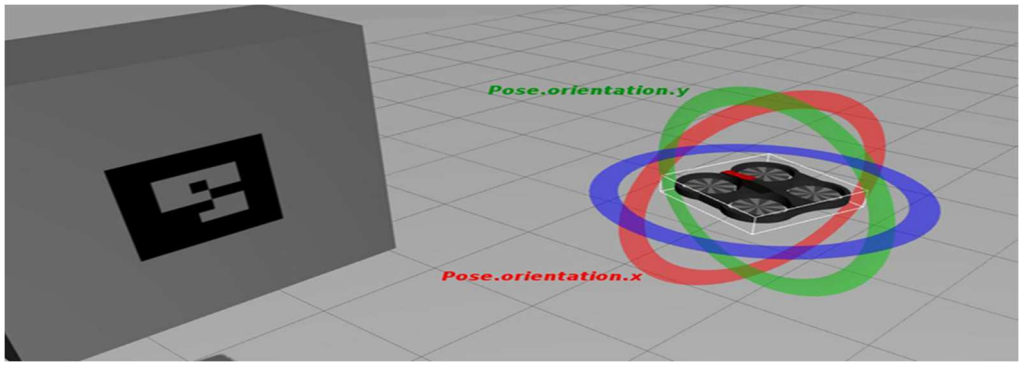


Figure 4. Rotational reference points of the drone.

2.1.B. Output Layer

The output layer of the network is designed according to the need of the application output. The output layer is the information related to the movement of the drone. These data are represented by the following variables:

- Linear Speed X (V_X): Represents the linear velocity of the drone along the x-axis. It allows the drone to move forward or backward.
- Linear Speed Y (V_Y): Represents the linear velocity of the drone along the y-axis. It allows the drone to make a lateral movement to the right or to the left.
- Linear Speed Z (V_Z): Represents the linear velocity of the drone along the y-axis. It allows the drone to move upward or downward.
- Angular velocity Z (Rot_Z): Represents the rotation of the drone along the z-axis in a quaternion. It allows the drone to rotate right or left.

2.1.C. The activation function

In general, a nonlinear function is used for the hidden layer. In the present application, we use the sigmoid function given by equation (1), because it is continuous, nonlinear and differentiable.

$$\emptyset_{(Z)} = \frac{1}{1+e^{-Z}} \tag{1}$$

2.1.D. Training the neural network

In our work, the Back Propagation training algorithm is used of supervised neural network learning. During training, the network is presented with a large amount of input patterns (training set). The desired corresponding outputs are then compared to the actual network output nodes. The error between the desired and actual response is used to update the weights of the network interconnections. This update is performed after each pattern presentation. One runs through the entire pattern set is tanned an epoch. The training process continues for multiple epochs, until a satisfactorily small error is produced. The test phase uses a different set if input patterns (test set). The corresponding outputs are again compared to a set of desired outputs. This error is used to evaluate the network's ability to generalise. Usually, the training set and or architecture needs to be assessed at this point. The rule of modification of the weights or of learning used is that of Widrow and Ho also called a rule of the delta.

To solve the problem of local minimal, we used two approaches. The first is to remake the learning of our neural network several times by changing the initial weights; the second approach is to vary the learning step.

For our learning, we used a learning method containing 644 examples. These were obtained by simulation under ROS Gazebo, the way to proceed was to place the quadrotor in a position relative to the target and guide it manually towards the target so that it is in front of it.

The partitioning of collected data was done according to a ratio of 0.8, that is to say 80% for learning and 20% for the test of learning.

In our application, we started with a learning step equal to 0.1 and we varied this value until we find a good compromise between speed and convergence. Table 1 shows the possible configurations for the learning process.

Let us note that the learning is good if the error rate is less than 0.020. In neural network training results the MSE would result different values, thus we have done nine experiment training to find optimal result.

Table 1. The different configurations for learning process.

Experiment	Maximum Number of Iterations	The Learning Step	Number of hidden layers
Configuration 1	100	0.1	120
Configuration 2	600	0.01	80
Configuration 3	100	0.01	105
Configuration 4	400	0.01	120
Configuration 5	500	0.7	50
Configuration 6	400	0.8	50
Configuration 7	250	0.2	80
Configuration 8	350	0.3	25
Configuration 9	100	0.6	85

Figure 5 shows the results chosen for the learning process. They are presented by a learning curve (in blue) and a learning test curve (in red).

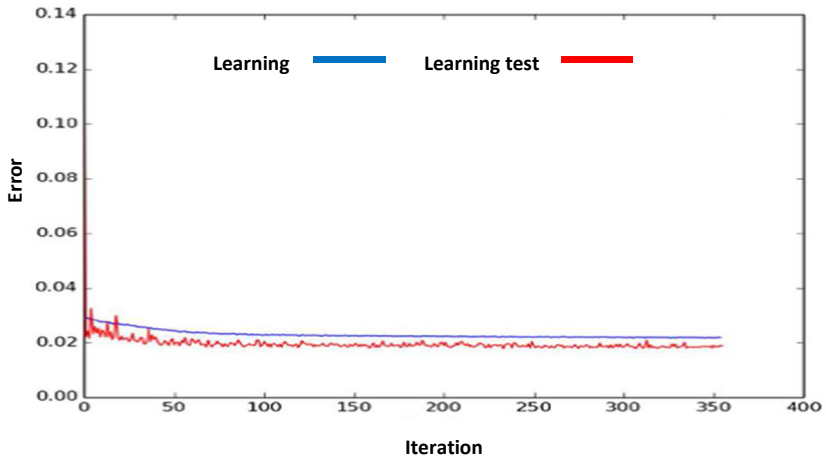


Figure 5. Mean square error curve of the training of configuration 8.

For configuration (8), one notes that the learning curve of the (MLP) converges successfully towards a final solution. The test curve also converges without fluctuations. For our application, we have selected this configuration.

To perform the training process, the neural network requires some parameters, which are listed below:

1- The architecture of the neural network is composed of 3 layers (input, hidden and output layer).

2- The input layer has 5 neurons, the hidden one has 25 neurons and the output one has 4 neurons.

3- The sigmoid activation function is used.

4- The weight initialization is achieved with Widrow and Ho.

5- 350 Maximum epoch.

6- The learning rate is equal to 0.3.

7- The error target is equal to 0.02 .

2.2. AR.Drone-Marker transformation frame

In fact, camera axes do not correspond to drone axes. To better show the necessity of a conversion, figure 6 compares the two reference systems, where (X_D, Y_D, Z_D) denote the drone axes and (X_C, Y_C, Z_C) denote the camera axes.

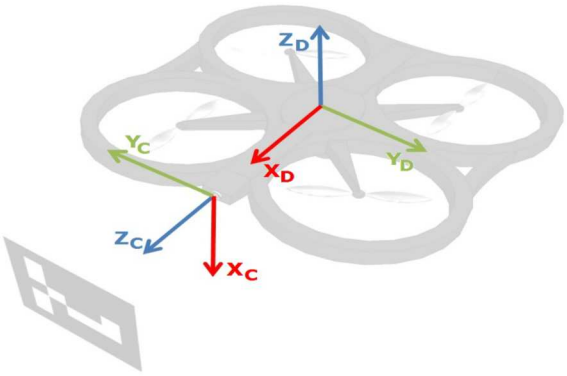


Figure 6. Drone vs camera axes details.

It is clear that a rotor-translation is needed. In particular, two rotations are necessary to convert camera axes to drone axes, as shown in Figure 7.

The first rotation defines X'_C, Y'_C and Z'_C axes through the following rotation matrix:

$$R_{(Z_D,\ 90°)} = \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 & 0 \\ \sin(90°) & -\cos(90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The second rotation defines X''_C, Y''_C and Z''_C through the following rotation matrix:

$$R_{(X_D,\ 90°)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(90°) & -\sin(90°) & 0 \\ 0 & \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

To complete the overlap of the axes a 20cm translation of the camera reference system to the AR.Drone center is required. Thus, the complete roto-translation matrix describes the relationship between the camera reference system and that of the drone, it is defined as:

$$H = R_{(Z_D}, 90°)\, R_{(X_D}, 90°)\, T_{(X_D,\ 20cm)} \tag{4}$$

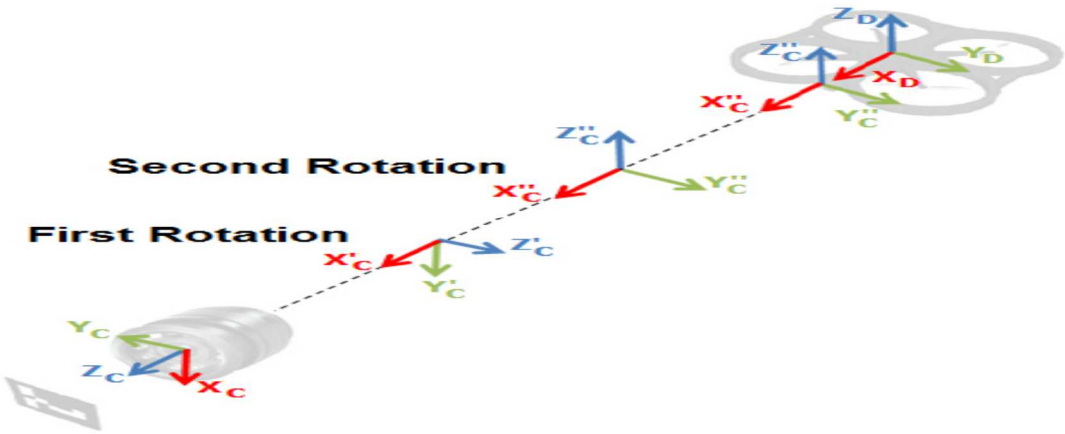


Figure 7. Camera axes roto-translation.

For example, to represent the position of a generic point Q_C in the camera reference system into the drone reference system, it is necessary to multiply its position vector in the camera reference system by the roto-translation matrix H, to obtain Q_C, which defines the position of the point in the drone reference system:

$$Q_D = H \,.\, Q_C \tag{5}$$

Nevertheless, this simple conversion describes the direction of the movements to be executed by the drone, but does not define the real twist to be sent to the drone.

2.3. Marker (Tag) tracking

We have to go through each received image to check if the marker is present. Algorithm 1 illustrates the necessary steps in order to estimate the 2D marker center M_C coordinates.

Algorithm 1: Identification of the marker

Input : Image (Video), Marker (the tracked coded marker)
Output : X_{M_C}, Y_{M_C} (2D pose of the Tracked marker)
Begin
Define marker
for Image do
if Image has marker **then**
Extract marker corners (0,1,2,3);
Find the center of the marker;
Dx1 = x.corner(2)-x.corner(0);
Dy1 = y.corner(2)-y.corner(0);
M1 = dy1/dx1;
C1 = y.corner(0)-m1*x.corner(0);
Dx2 = x.corner(3)-x.corner(1);
Dy2 = y.corner(3)-y.corner(1);
M2 = dy2/dx2;
C2 = y.corner(1)-m2*x.corner(1);
X_{M_C} = (c2-c1) / (m1-m2);
Y_{M_C} = m1*X_{M_C}+c1;
Draw a red square on (X_{M_C}, Y_{M_C});
else
Continue searching;

The results of the marker-tracking algorithm are likewise presented in Figure 8.

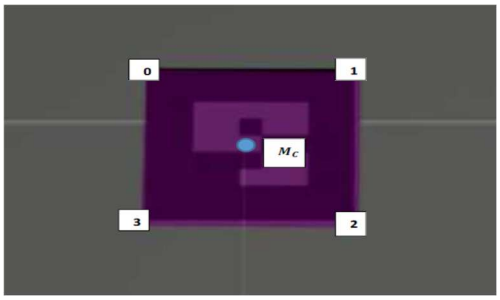


Figure. 8. Marker tracking for target pose estimation.

The main advantage of this method is that we can use different markers with different IDs and environment.

We consider X_Q and Y_Q the axes of the image taken from the quadrotor. In order to locate the marker, we can use one of the presented tracking algorithms to extract the markers positionM_C. The position (in pixels) of M_C along X_Q and Y_Q axes is : X_{M_C} and Y_{M_C}. X_0 and Y_0 are the center (0, 0) of the image (we assume that the center of the image is the center of the quadrotor).

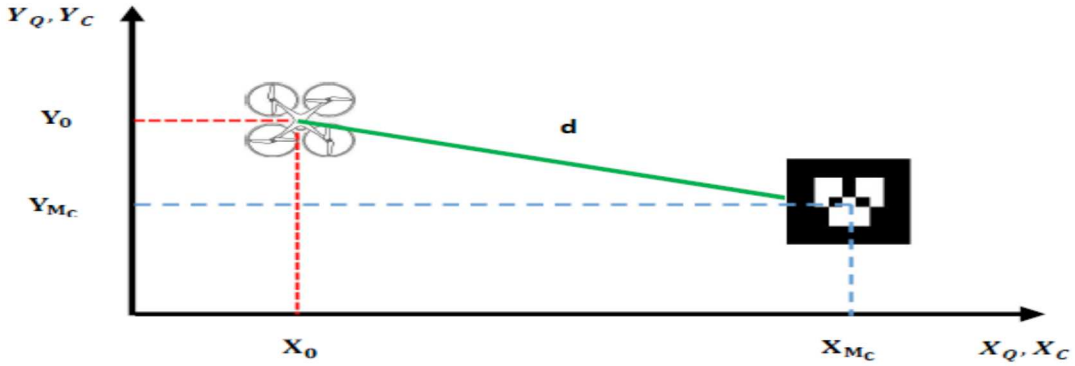


Figure. 9. The target relative position in the image plan.

The distance d that separates the target (Tag) and the center of the quadrotor can be written as follows:

$$\mathrm{d} = \sqrt{(X_0 - X_{M_C})^2 + (Y_0 - Y_{M_C})^2} \tag{6}$$

Where (X_0, Y_0, X_{M_C}) and Y_{M_C} represents respectively the coordinates of the center of the UAV and the marker M_C in its (or cameras) frame, previously supposed to be the same. The pose estimation of the target from the visual input is directly defined in the camera space.

To track marker by the quadrotor during its navigation, we need to control the drone movements along X_Q and Y_Q axes since the marker moves on a 2D plan. We need to develop a navigation controller that takes as input the location of the marker (M_C), and generates the necessary pitch and roll angles to keep the marker at the center of the image plan.

We consider four inputs $(\emptyset_Q, \theta_Q, \omega_Q)$ and Z_Q they represent respectively the desired roll, pitch, yaw angles, and the desired altitude. Note that the quadrotor attitude (roll, pitch, yaw, altitude) is controlled by the internal autopilot. To design the quadrotor controller, the dynamic model of a quadcopter is used [26]. Starting from this dynamic model, the movements along X_Q and Y_Q axes can be described as follows:

$$\ddot{x_Q} = (\cos\emptyset_Q \sin\theta_Q \cos\omega_Q + \sin\emptyset_Q \sin\omega_Q)\frac{1}{m}U_1 \tag{7}$$

$$\ddot{y_Q} = (\cos\emptyset_Q \sin\theta_Q \sin\omega_Q - \sin\emptyset_Q \cos\omega_Q)\frac{1}{m}U_1 \tag{8}$$

$\ddot{x_Q}$ and $\ddot{y_Q}$ correspond to the acceleration along X_Q and Y_Q axes respectively. Eq (7 and 8) can be expressed by :

$$\ddot{x} = uX_Q \frac{1}{m}U_1 \tag{9}$$

$$\ddot{y} = uY_Q \frac{1}{m}U_1 \tag{10}$$

With:

$$u_{X_Q} = (\cos\emptyset_Q \sin\theta_Q \cos\omega_Q + \sin\emptyset_Q \sin\omega_Q) \tag{11}$$

$$u_{Y_Q} = (\cos\emptyset_Q \sin\theta_Q \sin\omega_Q - \sin\emptyset_Q \cos\omega_Q) \tag{12}$$

Where u_{X_Q}, u_{Y_Q}represent the orientation of the total thrust (U_1) responsible for linear motions of the quadrotor along $X_Q$and Y_Q axes, m is the drone weight.

2.4. ROS package "ardrone_autonomy"

A laptop running under Ubuntu 14.04 LTS "Trusty Tahrn" and ROS Indigo-Desktop-Full to control the AR.Drone. The package ardrone_autonomy was used to establish the communication with the vehicle through the Wi-Fi. This package was built for ROS based on the official SDK released by the manufacturer [29 - 31]. The ROS topics listed in Table 2 are the inputs of ardrone_autonomy which are used to control both the simulated and hardware AR.Drone.

Table 2. ROS topics in ardrone_autonomy for quadrotor control.

ROS Topic	Description
/ardrone/navdata/altd	Estimated altitude in millimeters
/cmd_vel/twist/linear/x	Control movement along x-axis
/cmd_vel/twist/linear/y	Control movement along y-axis
/cmd_vel/twist/linear/z	Control movement along z-axis
/cmd_vel/twist/angular/z	Control of rotation along z-axis

2.5. ROS package "NNController"

NNController is the main node in our application and it is integrated inside ROS. This package was developed whit in the aim of simulating and controlling the quadrotor operations. Its main functions is to recognize the target and to sending commands to the quadrotor for tracking.

The software architecture of our target tracking system consists of two components. The first component is a detection marker-using algorithm 1 and the second part is a tracking algorithm using the node NNController. These two algorithms are seamlessly integrated in ROS to ensure smooth and real-time performance. The main algorithm 2 loop is shown in the pseudocode below.

Algorithm 2: Target Tracking

Marker found ← **false**

While true do

m ← new vision

While marker found = = **false do**

Detection (m) ► Invoke detection algorithm ANN

if marker is detected **then**

Marker found ← **true**

endif

end while

Tracking (m) ► send command to drone

if tacking is lost **then**

Marker found ← **false**

endif

end while

3. Results

For target tracking, we used the Tag information to establish the quadrotor maneuvers; the neural network algorithm sends the necessary commands to the drone to track the target. Figure 10 shows the configuration of nodes in ROS to perform tracking control. In this figure, the red colored node represents the tracking target controller node in the 'NNcontroller' package. As inputs, the node takes a calculated image capture relative to the center of the detected target and the target identity descriptor (ID) for the target. With these inputs, the node creates a control vector based on the target ID. ROS Gazebo (blue colored node) was used to simulate the image streams coming from the AR.Drone. Scenes in the simulator were extracted using a camera built into the 'Tum_simulator'. The 'ar_track_alvar' (green colored node) is able to process the images and provide feedback for the 'NNcontroller'. The following graph (figure 10) shows a hierarchical multi-robot control system constructed by simply instantiating multiple navigation stacks, each in its own namespace.

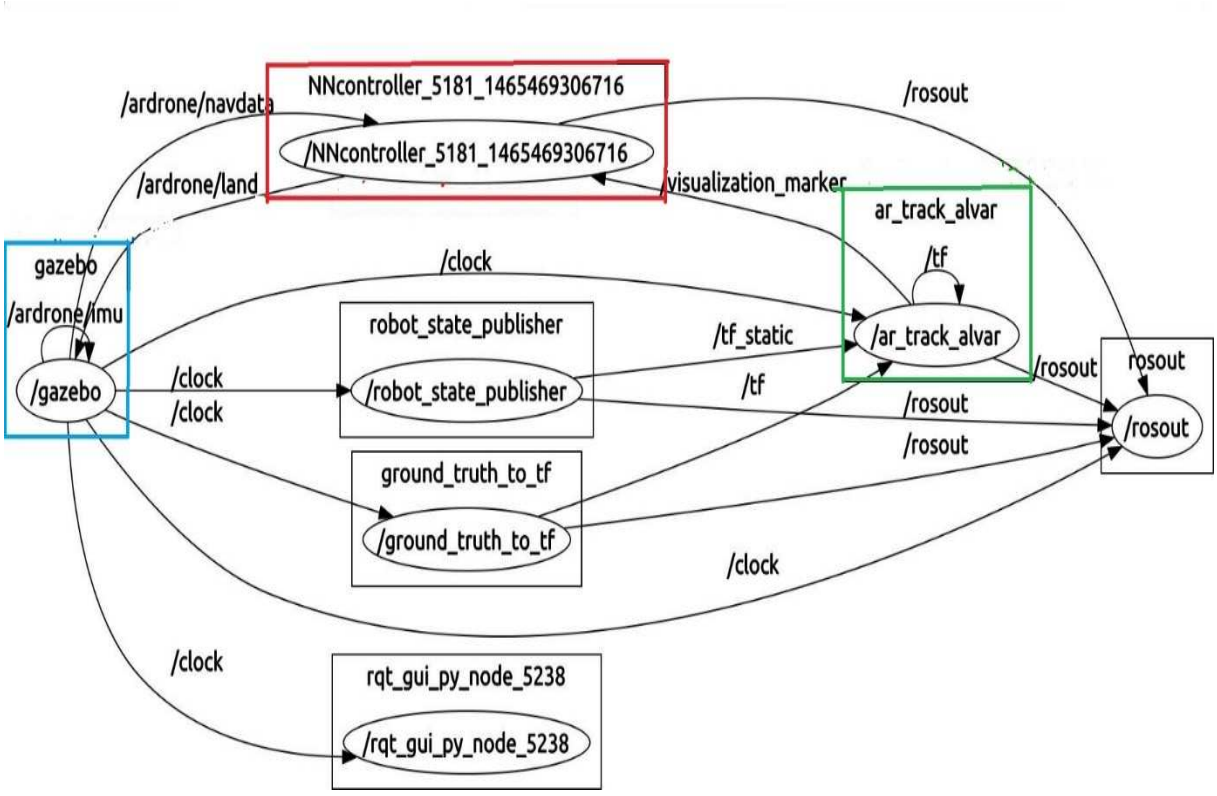


Figure 10. Interaction between all the active processes from Gazebo and neural network in ROS during the simulation tests.

The previous graph was automatically generated by the rxgraph tool, which can inspect and monitor any ROS graph at runtime. Its output renders nodes as ovals, topics as squares, and connectivity as arcs.

After processing the tag, the node "ar_track_alvar" returns the Euclidean position of the tag with its orientations to the node "NNcontroller". The information is transmitted via the topic "/ visualization marker". The neural network receives the input data and calculates the outputs corresponding to the maneuvers of the quadrotor for the tracking target. These data are transmitted to the node "Gazebo" through the topic "/ cmd vel" which is responsible for the movements of the quadrotor.

Many simulation tests were carried out to evaluate the efficiency of the neuron network in different positions of the target with respect to the quadrotor. The aim of these simulation tests is intended to prove that the developed neural network responds favorably to different situations.

3.1. Simulation result

3.1.A. First test: the quadrotor is on the right of the target

In this test, the quadrotor is positioned on the right at the same level of the target as indicated in Figure 11.

Figure 11. Simulation result of the first test.

From the simulation results of the first test, one notes that after take-off of the drone detects the target, makes a rotation to the left, and then it positions itself at the same level as the tag. The quadrotor goes to the target by rotating to the right and left to adjust its trajectory. Once the drone is close to the target, it maintains a distance of 1 m from the tag. In all the period of the flight, the drone uses its frontal camera, when the target is visible; it draws red colored square. Figure 12 shows the path crossed by the drone to follow the target.

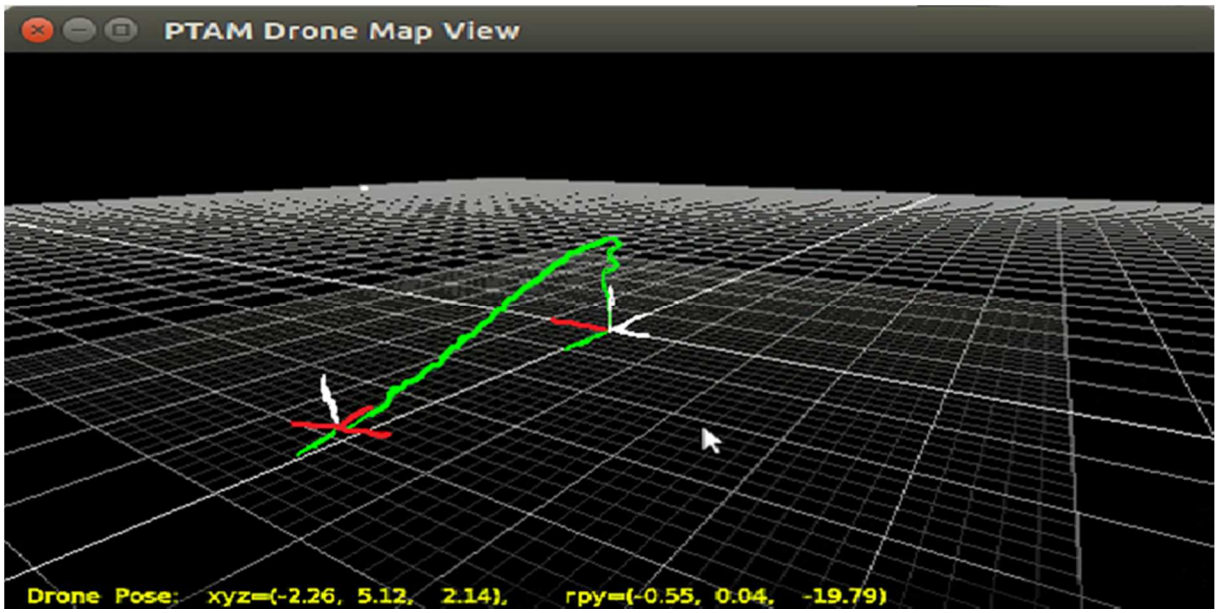
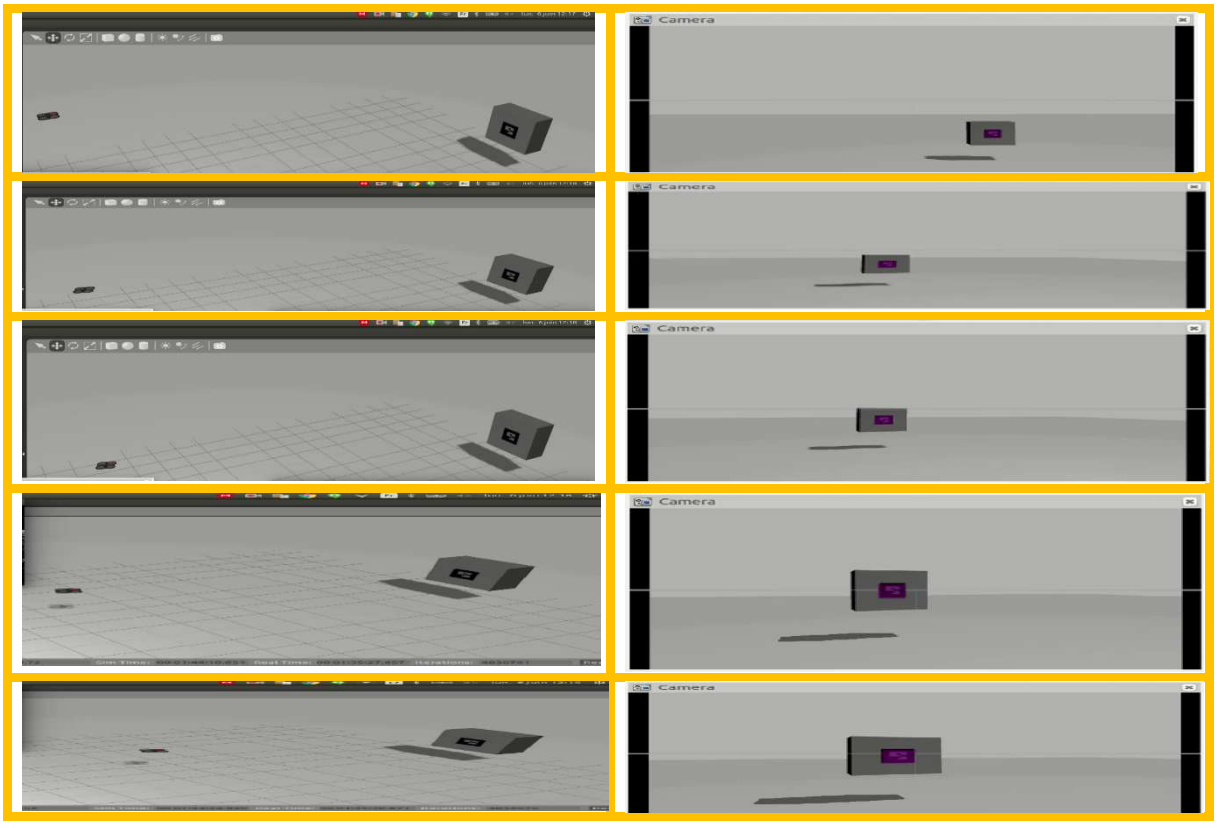


Figure 12. Illustration of a quadrotor trajectory recorded by the Rviz-Gazebo.

In this test, the quadrotor has successfully completed its target-tracking task by performing the following order a yaw, a pitch and another yaw to correct its trajectory. The quadrotor is positioned in front of the target, but remains slightly inclined to the right, but it always keeps a safe distance. The program of the neural network is built to order the quadrotor to do more rolls and yaws to correct the inclination. The results of the first test are found to be acceptable.

3.1.B. Second test: Quadrotor on the left at the same level as the target

In the second test, the quadrotor is positioned on the left at the bottom of the target as shown in figure 13.



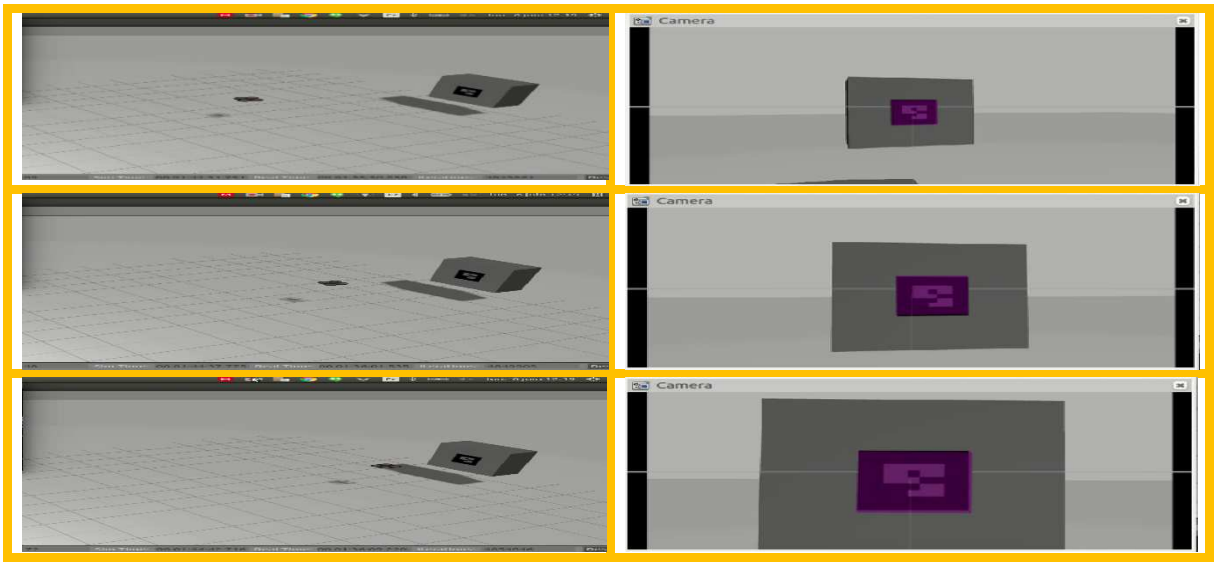


Figure 13. Simulation results of the second test.

From figure 13, the drone detects the target, rotates to the right, and position itself in front of the target. Afterwards, the quadrotor moves towards the target and makes rotations to the right and to left and a translation to correct its trajectory.

In this second test of a simulation, the quadrotor succeeded in accomplishing its mission autonomously by performing in the order a Yaw, a Pitch and a Roll. The quadrotor runs directly to the target and maintains a safe distance once nearby the target. The neural network algorithm provided the quadrotor with the necessary commands to follow the target. The following figure (figure 14) shows the path traveled by the drone during the second test under the Gazebo.

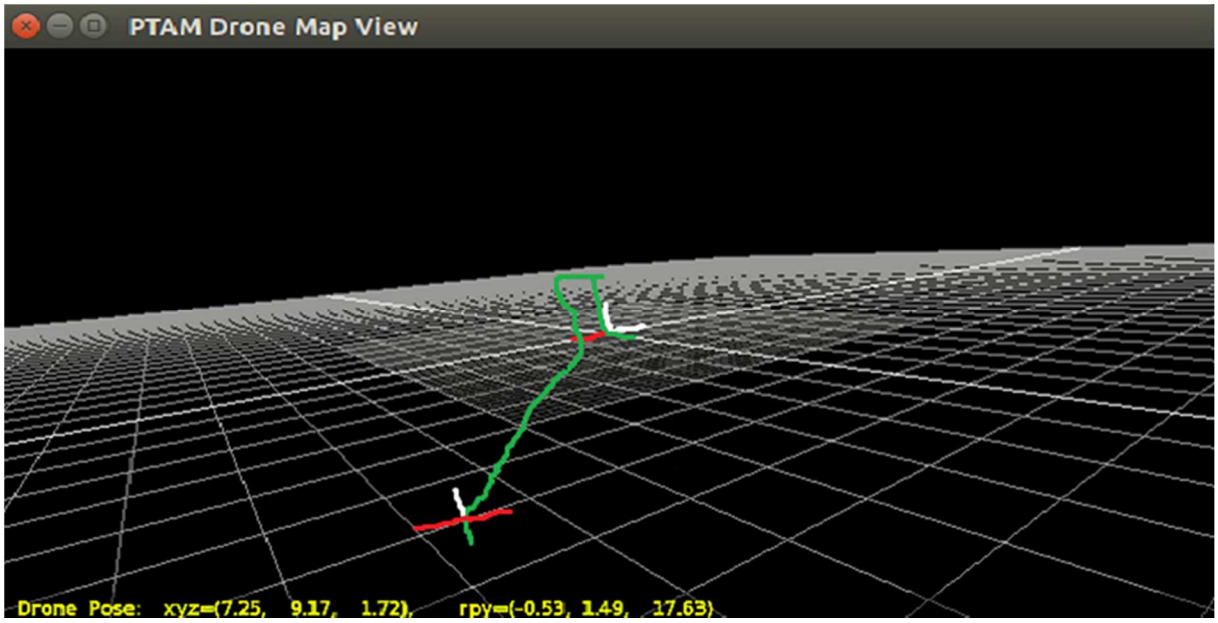


Figure 14. Illustration of the quadrotor trajectory recorded by the Rviz-Gazebo for the second test.

Figure 15 shows the orientations of the quadrotor with respect to the target along the two axes x and y for the first and second simulation tests.

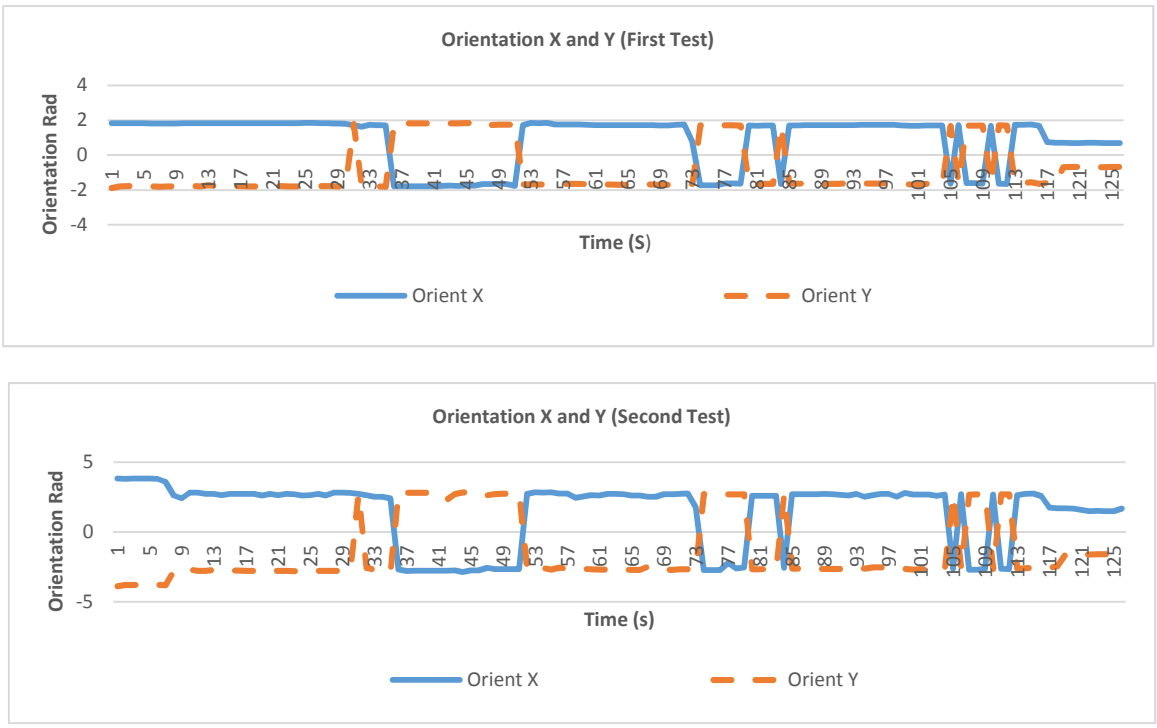


Figure 15. The evolution of the orientations along the two axes X and Y for the tests 1 and 2.

Figure 16 shows the positions of the quadrotor with respect to the target along the three axes x, y and z for the first and second simulation tests.

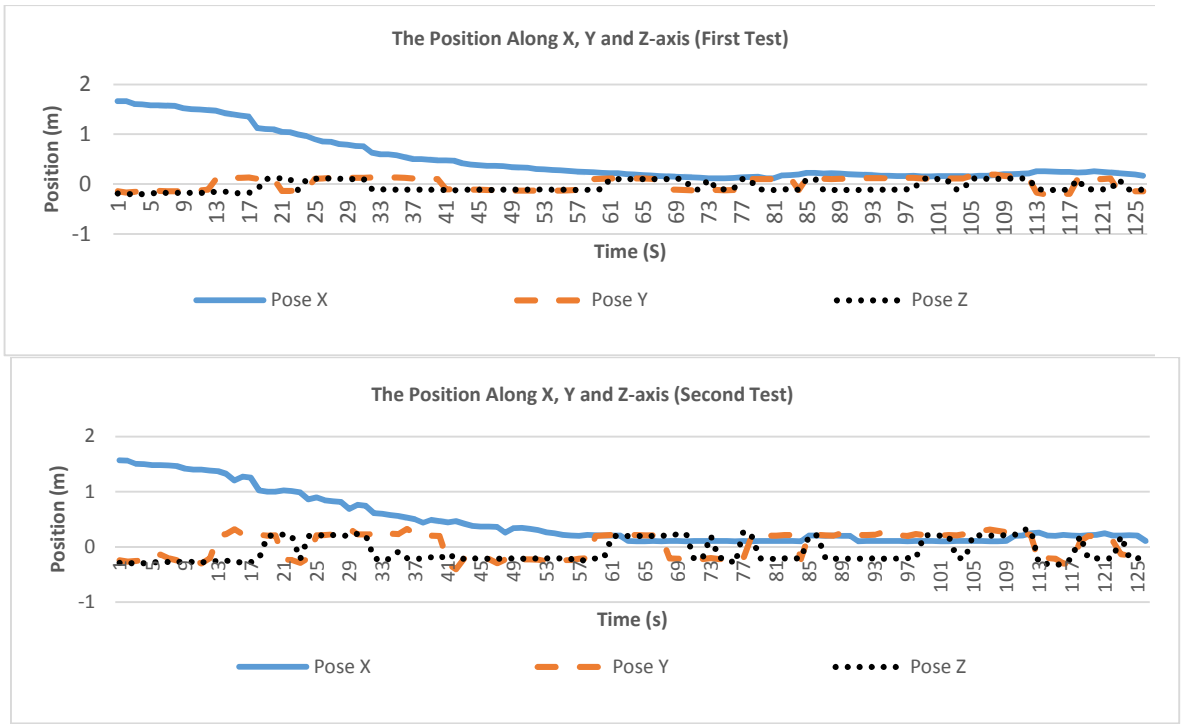


Figure 16. Evolution of the position of the drone along the three axes for the tests 1 and 2.

Figure 17 shows the evolution of the linear velocity of the drone along the three axes x, y and z for the first and second simulation tests.

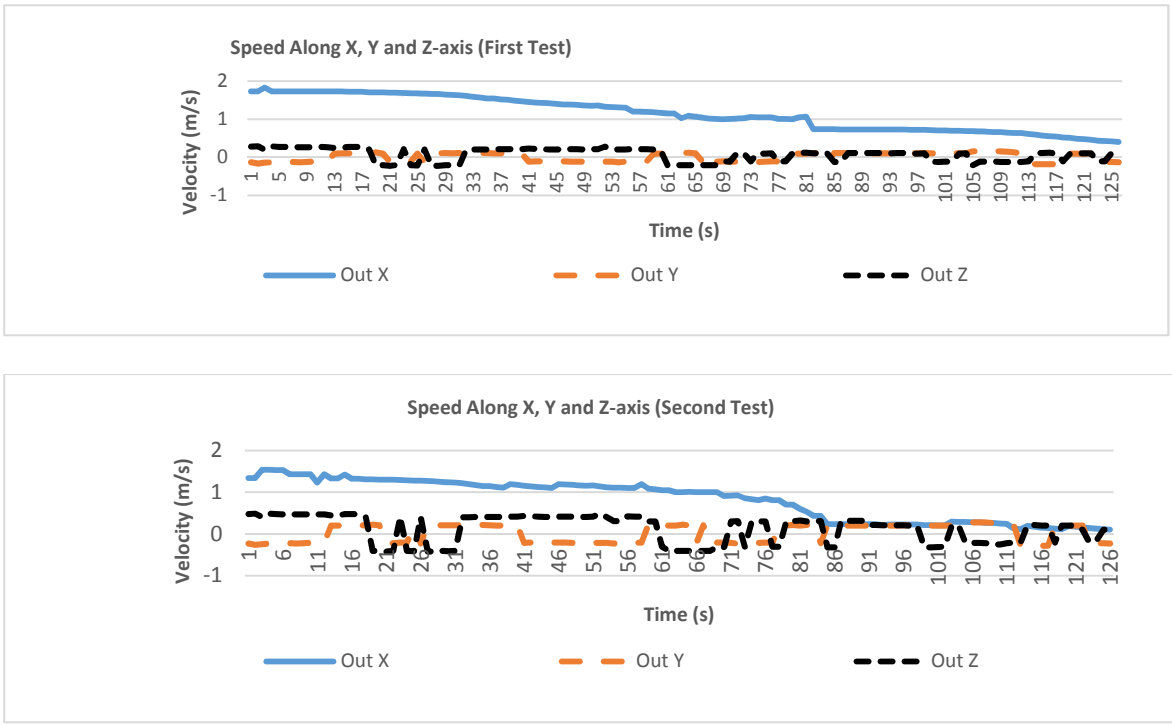


Figure 17. The evolution of the linear velocity along the three axes for the tests 1 and 2.

Figure 18 shows the evolution of the angular velocity of the drone along z-axes for the first and second simulation tests.

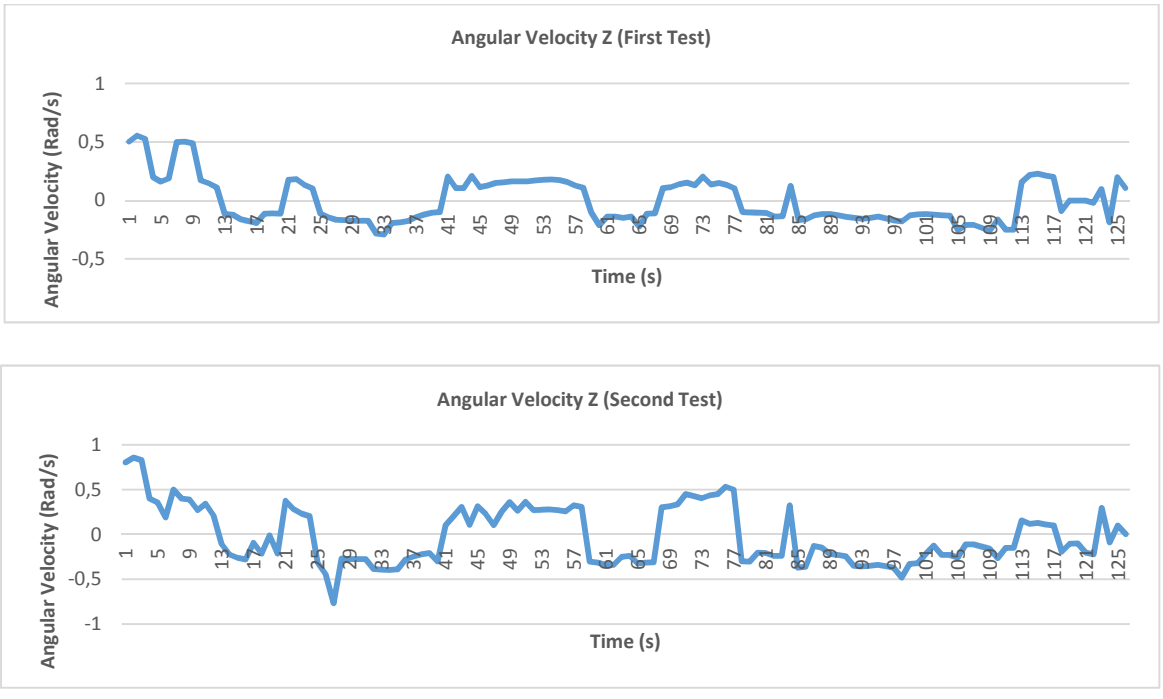


Figure 18. The evolution of the angular velocity along the z-axes for the tests 1 and 2.

An environment in Gazebo has been created, to test the target-tracking algorithms using neural network. All the tests show that after take-off, the quadrotor heads straight to the target. If the target is visible by the quadrotor, AR.Drone starts target tracking. Once the drone is at an acceptable distance for detecting the tag, a red square appears around the tag by using the ‘ar_track_alvar’ ROS software as shown in all

Figures. At this point, coordinates of the tag relative to the camera of the quadrotor, are available to neural network controller.

The performance of the ANN algorithm was tested in the Gazebo simulation environment. The visual camera data was used for detecting markers (tag) placed in the 3D world. The camera data of AR.Drone was used to generate state inputs for the neural network.

Some disturbances are noted in the second test. This is mainly due to the initial location of the quadrotor, which is more complicated and requires many maneuvers in order to locate the target.

3.2. Experimental results

The UAV we used for our experiments is the Parrot AR.Drone 2.0. It is equipped with a 1 GHz ARM Cortex-A8 as the CPU and an embedded version of Linux as its operating system. In addition, two cameras (a HD forward facing camera and a QCIF downward pointing the camera), a sonar height sensor, and an on-board computer. Communication between the AR.Drone and a host machine is performed via Wi-Fi connection.

Our approach consists of many main components as shown in Figure 19. Each component is implemented as a node within the Robot Operating System (ROS) framework, allowing each component to communicate whit the others using the ROS transport protocol.

In order to realize our experiment, a ROS application was developed. The application starts by establishing a connection to the AR.Drone using a Wi-Fi network. After the connection is established, different nodes are launched. The nodes are used for the following purposes:

- Ardrone_driver package: sends commands to the quadrocopter. Commands are sent in a UDP packet every 5 ms, and publishes the video frames received from the quadrotor instantly at the ardrone/front/image_raw topic.
- NNController package: The controller node contains the information of the ANN neural controller and dataset.
- ar_track_alvar package: responsible to identify and detect the target, takes the video frames and sends a signal to the NNController package.

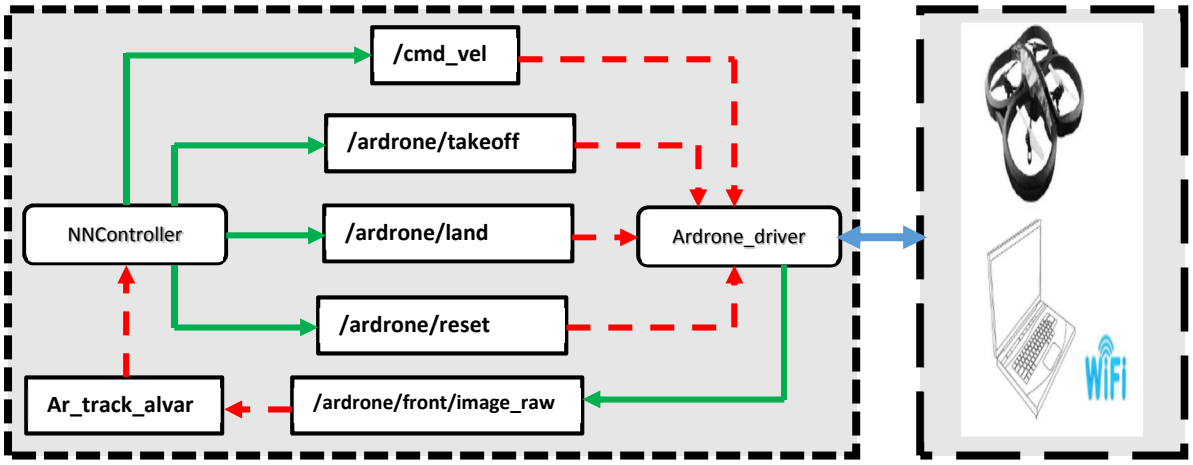


Figure 19. The communication hierarchy between the ardrone_driver and the implemented packages for target detection.

After take-off, we control the quadrocopter using the computer's keyboard. By pressing a key, the ANN is activated and the AR.Drone changes to the autonomous mode. ANN-based controllers are evolved in simulations under Gazebo ROS then transferred to a real AR.Drone. After the connection has been established, the application immediately starts streaming video obtained from the front camera of the

quadrotor. When the target (tag) is visible to the quadrotor, a red frame appears around the target. This will freeze the video stream so that it is possible to draw a bounded box similar to that of simulation results and as accurate as possible, so that it does not contain a lot of background, but only the desired object. Figure 20 shows the video sequences of the practical results for target tracking.

Figure 20. Real time experiments to demonstrate the performance of the proposed target-tracking system.

From the obtained results, one note that the quadrotor follow perfectly the target (tag). We observed that after take-off, the quadrotor moves directly towards the target. Once the target is identified, a red square around the target is drawn. The neural network algorithm acts then on the quadrotor to correct its trajectory and to transmit the necessary commands to track the target. From the experimental tests, we conclude that the experiments produce good results. For the result in real time detection using ROS, one can see the experiment and screen shot at the camera of the drone.

Figure 21 shows the evolution of the target orientation relative to the drone along the x and y axes, called respectively Pose.orientation X and Pose.Orientation Y. These two parameters were used as inputs for the neural network controller.

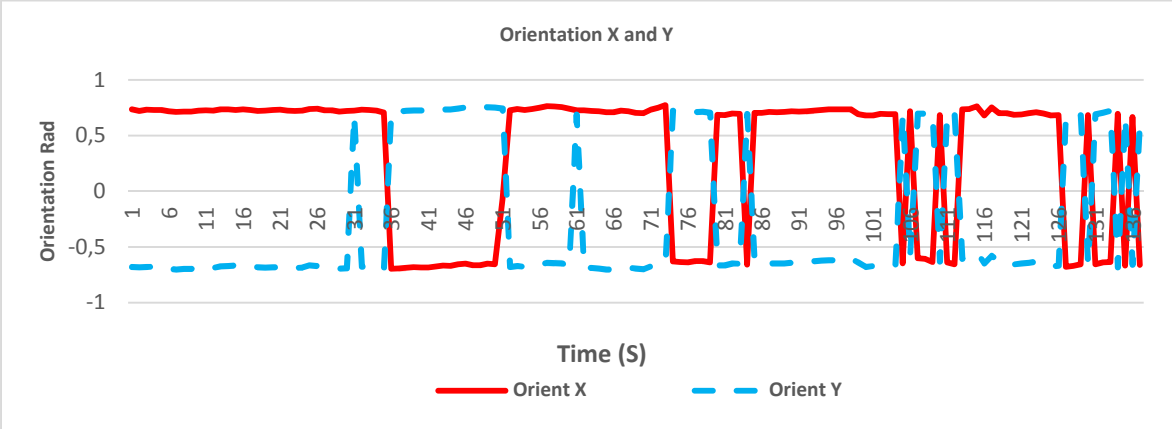


Figure 21. The evolution of the orientations along the two axes X and Y.

From the orientations figure, it can be seen that the drone makes an orientation along the x-axis, it will realize another inverse orientation along the Y-axis.

Figure 22 illustrates the three positions of the drone relative to the target. Three positions were used as inputs for the achieved the neural network controller.

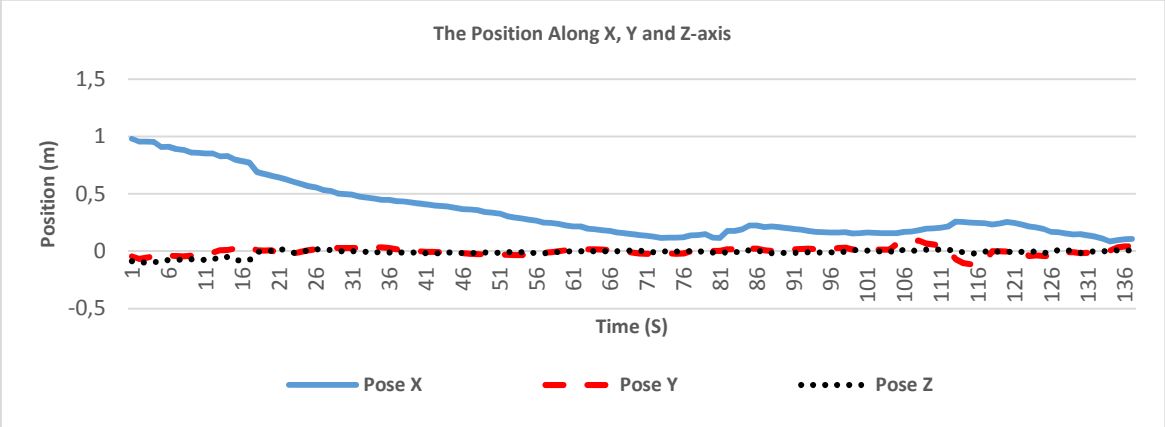


Figure 22. Evolution of the position of the drone along the three axes.

From this figure, it can be seen that the drone moves forward quickly along the x-axis. The evolution of the position of the drone along the other two axes y and z, varies forward slowly and remains unchanged compared the x-axis.

Figure 23 illustrates the variations of the linear speed along the three axes x, y and z. We notice that the quadrotor moves quickly towards the target with a starting speed of 1 m /s, but once the target is near to, the speed decreases. This is justified because the quadrotor must maintain a distance of 1 m from the target.

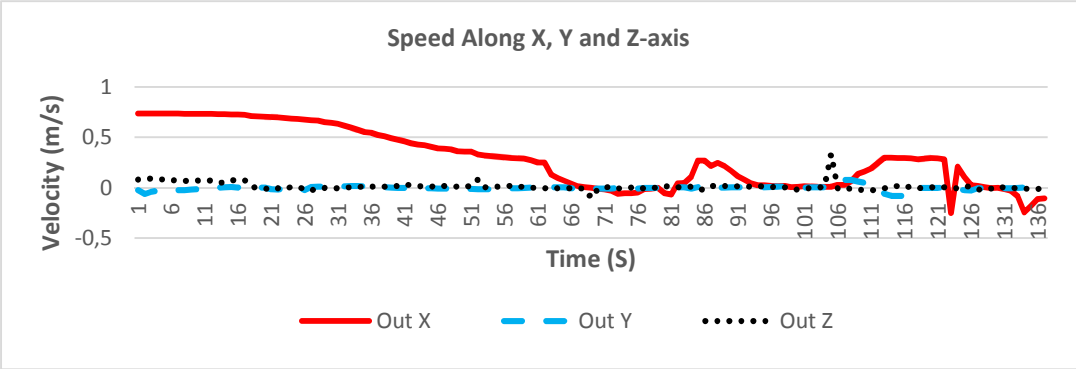


Figure 23. Variation of the linear velocity along x-axis, y-axis and z-axis.

The data of the linear velocity along the three axes is used as output data by the designed neural network controller. Figure 24 illustrates the variations of the angular velocity along the z-axis.

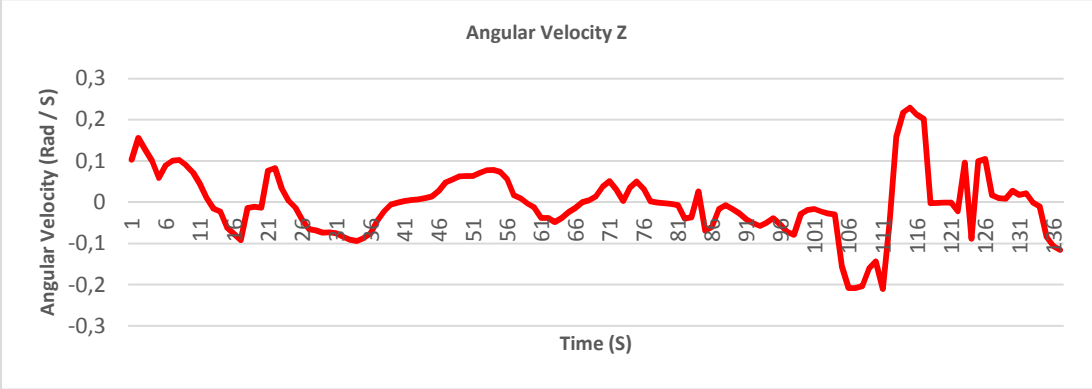


Figure 24. Variations of the angular velocity along the z-axis.

The angular velocity is used as an output by the neural network controller. It can be shown from figure 24 that the quadrotor changes its orientation in order to correct its trajectory.

4. Conclusion and perspectives

In this paper, a vision system for identification and target-tracking using a Parrot AR.Drone 2.0 quadrotor UAV is presented. The proposed technique is based on the detection of a marker by the camera and by using a neural network controller. Then the distance between the quadrotor and the target is estimated, and neural network MLP algorithm is used to fly towards the marker. Once the target position has been reached, the position of the tag is detected again. To correct possible drift while approaching the target, the distance of the quadrotor with respect to the target is estimated and used to correct the quadrotor position.

Based on the obtained results, it can be concluded that the algorithm using artificial neural networks recognizes the target just as the recognition module used by ar_track_alvar in ROS. Thus, the proposed algorithm provides a better performance in the recognition process of the target, which is very important for real time drone applications. Experiment and simulation results show that the implementation of image processing and neural network algorithm for object detection and tracking has been successful.

Future work includes an extensive set of experimental trials of the vision approach using a variety of visual features to be tracked and its integration with the neural network controller. Additionally we expect to extend the strategies presented in this paper, with the purpose of working in different conditions, such as a different target forms. Pattern recognition methods, like machine learning, will be tested with the main task of providing a more robust estimation of the form and location of the target.

References

[1] P. Linci, A. Vinyojita Mohanraj; "Target Identification and Tracking Using UAV"; International Journal of Engineering research & Technology. Vol 4, issue 4, pp. 353-356, April 2015.

[2] R.V. Badu, S. Suresh; and All, "Online Adaptive radial Function Networks for Robust Object Tracking"; Computer Vision and Image Understanding, pp. 297-310, 2010.

[3] H. K. Kavith, S.C.P. Kumar, "Study on Object Tracking Using Neural Network Functions", International Journal of Advanced Research in Computer and Communication Engineering,Vol 3, issue 5, pp. 6424-6427, May 2014.

[4] P. Cheema, S. Luo, P. Gibhens, "Development of a Control and Vision Interface for a AR.Drone", 2016 International Conference on Civil, Architecture and Environmental Engineering (ICCAE 2016), Taipei-Taiwan, November 2016.

[5] P. Prashanth, V.P.S. Naidu, "Target Geo-localization Based on Camera Vision Simulation of UAV", Journal of Optics, Springer, pp. 1-11, April 2017.

[6] T. Dierks, S. Jagannathan, "Output Feedback Control of a Quadrotor UAV Using Neural Networks", IEEE transactions on neural Networks, Vol 21, N°1, pp. 50-66, January 2010.

[7] R. L. Galvez, E. P. Dadios, A. A. Bandala, "Predicting the Motion of Quadrotor Using neural Network", 8th IEEE International Conference Humoid, Nanotechnology Information Technology Communication and Control, Environment and Management (HNICEM) 9-12 December 2015, Philippines.

[8] L. Jangwon, J. Wang, D. Crandall, and all, "Real-Time Object Detection for Unmanned Aerial vehicles Based on Cloud-Based Convolutional Neural Networks", Journal Concurrency and Computation : Practice and Experience, Vol 29, issue 6, 2017.

[9] M. Shirzadeh, A. Amirkhani, A. Jalali, M.R. Mosari, "An Indirect Adaptive Neural Control of a Visual-Based Quadrotor Robot for Pursing a Moving Target", ISA Transactions, Elsevier, October 2015.

[10] A. Borji, S. Frintrop, D. N. Sihite, and all, "Adaptive Object tracking by Learning Background Context", In Proc. IEEE Conf, Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 23-30; 2012.

[11] A. Ernesto, R. S. Cassiano, "Video Target Tracking by Using Competition Neural Networks", WEAS transactions on Signal Processing, issue 8, Vol 8, pp. 420-431, August 2008.

[12] A. D. Mengistu, D. M. Alemayehu, "Robot for Visual Object Tracking Based on Artificial Neural Network", International Journal of Robotics Research and Development (IJRRD), vol 6, issue 1, February 2016.

[13] G. L. Masala, B. Golosio, M. Tistarelli, and all, "2D Recurrent Neural Networks for Robust Visual Tracking of Non-Rigid Bodies" Springer International Publishing Switzerland pp. 18-34, September 2016.

[14] T. Grosz, P. Bodnar, L. Toth, and all, "QR Code Localization Using Deep Neural Networks", 2014 IEEE International Workshop on Machine Learning for Signal Processing, September 2014, French.

[15] W. Yang, Z. Jin, C. Thiem, and all, "Autonomous Target Tracking of UAVs Based on Lower-Power neural Network Hardware" machine Intelligence and Bio-inspired Computation: Theory and Applications VIII, Proc of SPIE, Vol 9119, 2014.

[16] J. Ahmed, M. N. Jafri, M. I. khan, "Design and Implementation of Neural Network for real-Time Object tracking", in proc, Int'l Conf; Machine Vision and Pattern Recognition, pp. 1-4, 2005.

[17] Z. Cao, L. Cheng, C. Zhoo, and all, "Spiking neural Network-Based target tracking Control for Autonomous Mobile Robots" Neural Comput & Applic, Springer, February 2015.

[18] B. Sun, "Fuzzy Neural Network-Based Interacting Multiple Model for Multi Node target tracking Algorithm", Journal Sensors, Vol 16; issue 11, pp. 18-23, 2016.

[19] L. Wang, W. Ouyang, X. Wang, and all; "Visual tracking with Fully Convolutional Networks" 2015 IEEE International Conference on Computer Vision (ICCV), December 2015, USA.

[20] W. K. Lai, M. Y. Too, and all; «Nuchal Translucency Marker Detection Based on Artificial Neural network and Measurement via Bidirectional iteration Forward propagation", WSEAS transactions on Information Science and Applications, vol 7, issue 8, pp. 1025-1036, August 2010.

[21] F. Garcia and E. Araujo, "Visual Multi-Target Tracking by Using Modified Kohonen Neural Networks," in International Joint Conference on Neural Network. IEEE, 2008, pp. 4162–4167.

[22] S. Shams, “Neural Network Optimization for Multi-Target Multi-Sensor Passive Tracking,” Proc. IEEE, vol. 84, pp. 1442–1457, 1996.

[23] P. Gaur, S. Tiwari, “2D QR Barcode Recognition Using texture Features and Neural Network”, International Journal of Research in Advent technology, vol 2, N°5, pp. 433-437, May 2014.

[24] X. Yu and M. R. Azimi-Sadjadi, “Neural Network Directed Bayes Decision Rule for Moving Target Classification,” IEEE Trans. on Aerospace and Electronic Systems, vol. 36, pp. 176–188, 2000.

[25] C. H. El Houssein, F. Guérin, F. Guinand, and All, “Vision Based target tracking Using An Unmanned Aerial Vehicle’, 2015 IEEE International Workshop on Advanced Robotics and Its Social Impacts, Lyon, France July 2015.

[26] I. Szentandrasi, A. Herout, M. Dubska, “Fast Detection and Recognition of QR Codes in High-Resolution Images”, in Proceedings of the 28th Spring Conference on Computer Graphics, New York, USA; SCCG’12, pp. 129-136, 2013.

[27] T. H. Chou, C. S. Ho, Y. F. Kuo, “QR Code Detection Using Convolutional Neural Networks” 2015 International Conference on Advanced Robotics and Intelligent Systems (ARIS), Taipec-Taiwan, 2015.

[28] T. Dierks; “Output Feedback Control of a Quadrotor UAV Using Neural Networks”; IEEE Transactions on Neural Networks, Vol. 21, NO. 1, January 2010.

[29] Z. Neji, F. M. Beji; “Neural Network and Time Series Identification and Prediction”, IEEE-INNS-ENNS International Joint Conference on Neural Networks; 2000.

[30] B. Babenko, M. H. Yang, and S. Belongie, “ Visual tracking With Online Multiple Instance Learning” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011.

[31] C. Caceres, D. Amaya, J. M. Rosario, “Simulation, Model and Control of a Quadrotor AR.Drone 2.0”, International Review of Mechanical Engineering (IREME), vol 10, issue 3; 2016.