

# Implementing the Propeller Speed Control of Drone Designed in Solidworks in CoppeliaSim Robotic Simulator Environment with Codes Prepared in Matlab

Bülent SİĞERGÖK<sup>\*1</sup>, Mehmet ÇAVAŞ<sup>2</sup>

\*Corresponding author

<sup>1</sup>Graduate School of Natural and Applied Sciences, Firat University, Elazığ, Turkey, bulentsigergok02010@gmail.com\*

<sup>2</sup>Firat University, Technology Faculty, Mechatronic Engineering, Elazığ, Turkey, mcavas@firat.edu.tr

DOI: 10.13111/2066-8201.2023.15.4.20

Received: 08 October 2023/ Accepted: 17 November 2023/ Published: December 2023

Copyright © 2023. Published by INCAS. This is an “open access” article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

**Abstract:** Real-time applications of autonomous systems are simulated in the computer environment to ensure that they operate error-free or with minimal errors. Coppeliasim, used in this field, is a platform where several sample models, robots, sensors and actuators are used together, a virtual world is created and interacted with it throughout the working period. Having a comprehensive toolbox, autonomous vehicle training and virtual reality, Coppeliasim's compatibility with Solidworks, a very useful design program for drawing, seems to be a great advantage. Due to these features, CoppeliaSim is very important in predicting and solving problems that may arise in many different applications. The propeller movement of the drone, which we designed with the Solidworks 2020 program and transferred to the Coppeliasim platform using the URDF exporter method, was carried out with the Coppeliasim simulator. In our work, Coppeliasim is synchronized with the simulator and MATLAB API codes. While the drone propellers were working on the Coppeliasim platform, angular speed and timing controls were made using the codes we prepared in the MATLAB program. Additionally, this work shows that drones or different autonomous systems can be controlled and designed before real-time operation using the Coppeliasim simulator and the MATLAB program.

**Key Words:** Robotics, Coppeliasim, Propeller Angular Speed Control, Simulation, Matlab

## 1. INTRODUCTION

As in many areas of modern industry, a wide variety of applications such as material transfer, precision assembly, welding and machinery are highly dependent on autonomous systems[1]. An unmanned aerial vehicle (UAV), a vehicle with an autonomous system, is a vehicle that can fly without a human pilot by using aerodynamic effects. The flight of these aircraft can be autonomously or remotely controlled by pilots from the ground, with or without load. Depending on the technological developments, with the development of autonomous systems, UAVs have entered the life of living things. UAVs, which are in the class of autonomous systems, have become actively used in many areas, especially in the military, agricultural and security fields.

Autonomous systems are the design, modeling and production of computer, electronic, mechanical, control systems, etc. These systems are considered to be the product of

interdisciplinary studies covering the subjects. In order to make real-time applications error-free and trouble-free in autonomous systems, the systems must be simulated. Autonomous simulators, developed to solve the problems that may occur, provide the necessary tests for the design, modeling and production of autonomous systems in a virtual environment, and also offer significant benefits and gains for autonomous vehicle training. It is extremely important to design and simulate using parameters from the kinematics, physics and graphics libraries of the simulators in question. At the same time, in terms of engineering and control methodology, the interaction of these parameters and the performance and accuracy of the prepared system are also important in determining what will happen [2].

In robotics, which is a multidisciplinary field, training at all levels has been provided in many fields in recent years. It expands into several engineering and scientific fields as disciplines such as electrical engineering, computer science, control engineering mechanics. This is why robotics is so interesting and versatile from a pedagogical point of view. The primary focus of robotic systems is to create a platform that enables rapid prototyping and validation. Also, the generality of the toolbox makes it valuable in robotics education, human-robot interaction, or other contexts [3] such as developing, testing and validating different cognitive approaches, perception, memory, behavior etc. In order to compare the features of the two simulators, Gazebo and CoppeliaSim, an attempt was made to produce two source-based solutions. Of these, basic, commercial and educational versions, which are free (CoppeliaSim) provided by Coppelia Robotics, are offered as solutions. Some authors working in this field have conducted a survey on the use of robotic simulators. Two of the simulators in the survey were selected according to the results announced. These authors found that Gazebo is the best-known simulator and CoppeliaSim is the best-rated simulator[4].

Today, the CoppeliaSim simulator deserves to be mentioned as one of the most widely used special programs for pedagogical purposes. Moreover, the generality of the toolbox provides values such as robot training, human-robot interaction, or reinforcement learning. CoppeliaSim software is designed to have a low input threshold for moderately complex applications[5]. CoppeliaSim has an integrated development environment (IDE) based on flexible and scripted architecture. Among many platforms, the CoppeliaSim (Virtual Robot Simulation Platform) simulator is designed for the creation of three-dimensional simulation in a short time, a very large object scene and an embedded script, all of which run simultaneously, creating a virtual world and interacting at runtime. It has sensors and actuators along with a large number of sample robotic models, etc. used in this study due to its properties. It provides support for a set of tools that add VR to the prototyping environment of any powerful robot system[6]. CoppeliaSim simulator and MATLAB (matrix laboratory) also offer the opportunity to be used together in real-time applications [7-8]. Scripting algorithm techniques are used to leverage simulator and hardware capabilities to simulate the behavior of real robot functions[9]. In the literature, there are studies on CoppeliaSim simulator and MATLAB-based autonomous systems, but in most of the studies, it is seen that the autonomous vehicles used are not designed and the ones that are ready on the platform were used.

It is seen that studies on drone design and modeling and studies such as speed control of propellers have been done very little. Therefore, in this study, a virtual environment was created for drone speed control by synchronizing with CoppeliaSim and MATLAB and to show the validity and applicability of the recommended organization with MATLAB and CoppeliaSim, important parts of CoppeliaSim, plug-in features of the designed drone, embedded all definitions including types of scripts and remote API (Application Programming Interface) of CoppeliaSim were made and referenced drone simulation was made.

## 2. MATHEMATICAL MODEL OF THE DRONE

The cross-shaped frame of a drone vehicle has four engines, front, rear, right and left, and these engines generate lift in the direction of the rotation axes by driving the propellers. The front and rear propellers rotate counterclockwise, while the left and right propellers rotate clockwise. In this way, when all the propellers rotate at equal speed, the torque applied to the center of the helicopter is balanced and the orientation angle, which is the rotation angle of the helicopter around its axis, does not change. The difference between the speeds of the left and right propellers creates a difference in lift forces and the helicopter's yaw angle changes. By the same logic, the difference in speed between the front and rear propellers causes the pitch angle to change. Increasing or decreasing the speeds of all propellers at the same rate enables the drone vehicle to move in its own  $z$ -axis direction. The direction of rotation of the drone propellers and the lifting forces, rotation angles and movement coordinates due to this rotation are shown in Fig. 1. The yaw ( $\Phi$ ), pitch ( $\theta$ ) and yaw ( $\psi$ ) angles around the  $x$ ,  $y$  and  $z$  axes of the drone vehicle, respectively, the main forces  $F1, F2, F3, F4$  and  $mg$ , and the rotation directions of the four propellers are given.

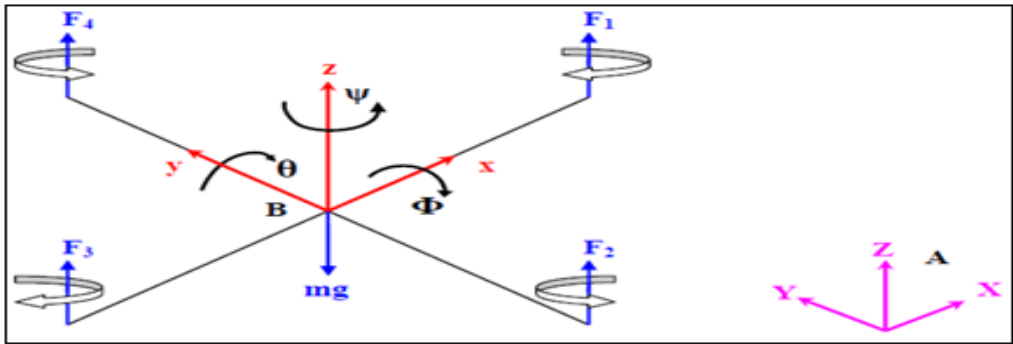


Fig. 1 - Forces acting on the vehicle and earth coordinate axes.

The quadrotor aircraft is modeled using the Newton – Euler equations. The position of  $B$ , the body fixed frame, and  $A$ , the position ( $\zeta$ ) in 3-dimensional space with respect to the inertial frame is expressed as in equation 1.

$$\zeta = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

In equation 2, the velocity of  $B$  with respect to  $A$  is given.

$$v = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{2}$$

In equation 3, the acceleration of the linear motion of the drone vehicle is given.

$$\dot{v} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \tag{3}$$

$\Omega_i$  is the angular velocities of the propellers and ( $i: 1, 2, 3, 4$ ), the lifting force resulting from the rotation of the propellers is expressed in equation 4. The push factor  $b$  here is a constant

value and  $I$  angular momentum. While the total lift force applied to the drone vehicle from the propellers is calculated by equation 5, the acceleration due to this force is expressed as in equation 6 The expression of the total acceleration with respect to the  $A$  frame is expressed as  $R$ , and the force balance is expressed by equation 7.

$$F_i = b\Omega i^2 \tag{4}$$

$$F_i = b\sum_i^4 = I \Omega i^2 \tag{5}$$

$$a_f = \frac{b}{m} \sum_i^4 = I \Omega i^2 \tag{6}$$

$$v = -ge_z + Re_z a_f \dot{\nu} = -ge_z + Re_z a_f \tag{7}$$

$e_z$  in equation 7 is a  $[0 \ 0 \ 1]^T$ -shaped vector and is used to express the magnitudes in the z-axis.  $R$  in Equation 8 represents the rotation matrix.  $C$  stands for Cosine and  $S$  stands for Sine,

$$R = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi - S\psi C\phi & C\psi S\theta C\phi + S\psi S\phi \\ S\psi C\theta & S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi - C\psi S\phi \\ -S\theta & C\theta S\phi & C\theta C\phi \end{bmatrix} \tag{8}$$

Regarding the angular velocity of the rigid body, the relationship between the angular velocities of the rotation matrix and the body frame is defined by equation 9.

$$\dot{R} = RS(\omega) \tag{9}$$

$\omega$  in equation 9 represents the angular velocity vector of the body frame and is given as in equation 10.  $S(\omega)$  is the 3x3 minus symmetric matrix of  $\omega$  and is expressed as in equation 11.

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{10}$$

$$S(\omega) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{11}$$

The axis of rotation of rigid bodies rotating rapidly around its own axis rotates around the vertical axis in the direction of gravity and forms a cone of rotation. A wobble movement occurs when the rotational moment acting on the rotating object changes the direction of the rotation axis of the object. This effect, which occurs in all rotational movements of the drone vehicle, This is called the gyroscopic effect. Since the drone vehicle rotates around its axes with angular velocities of  $\omega$ ,  $X$ ,  $Y$ ,  $Z$  angular momentums are formed and are expressed as in equation 12.

$$I_{X,Y,Z} = I\omega \tag{12}$$

$I$ , a 3x3 matrix in equation 12, is the inertia in the  $x$ ,  $y$  and  $z$  axes of the Drone vehicle body, expressed as in equation 13.

$$I = \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \tag{13}$$

Since torque is the variation of angular momentum with time, the torque due to the angular velocities of the drone vehicle is expressed as in equation 14.

$$T_B = \dot{I} \tag{14}$$

From equations 12 and 14,

$$T_B = \omega \times I \omega + \dot{I} \omega \tag{15}$$

The gyroscopic torque resulting from the rotation of the body of the drone and the propellers around their own axis is expressed as in equation 16. Here  $J$  represents the inertia of one rotor.

$$T_G = b \sum_i^4 \Omega_i^2 = J(\omega \times e_z) \Omega_i (-I)^i \tag{16}$$

The lifting forces due to the rotational movement of each propeller, seen in Figure 1, constitute the torques acting on the drone vehicle. The torque along one axis is equal to the difference between the torques produced by the propellers on the other axis, and the torques caused by the propellers along the  $x$ ,  $y$  and  $z$  axes are expressed as in equation 17. Here,  $l$  is the distance between the rotor and the center of the quadcopter, and  $d$  is the drag factor. The torque balance in equations 15 and 16 is expressed as in equation 18.

$$T_a = \begin{bmatrix} Ib(\Omega_4^2 - \Omega_2^2) \\ Ib(\Omega_3^2 - \Omega_1^2) \\ d(-\Omega_1^2 + \Omega_4^2 - \Omega_2^2 + \Omega_3^2) \end{bmatrix} \tag{17}$$

$$T_G + T_B = T_a \tag{18}$$

From Equation 5.18,  $T_B$  is obtained as in Equation 19.

$$T_B = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} + \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} \tag{19}$$

Torque balance is given as in equation 20.

$$T_B = J \left( \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) (-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4) + \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} + \begin{bmatrix} I_X & 0 & 0 \\ 0 & I_Y & 0 \\ 0 & 0 & I_Z \end{bmatrix} \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} Ib(\Omega_4^2 - \Omega_2^2) \\ Ib(\Omega_3^2 - \Omega_1^2) \\ d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{bmatrix} \tag{20}$$

In this equation, if  $(\theta)$ ,  $\ddot{\phi}$  and  $\psi$  are thrown to the left and the equation is solved, the angular accelerations appear as in equations 21, 22 and 23.

$$\ddot{\phi} = \dot{\psi} \dot{\theta} \left( \frac{I_Y - I_Z}{I_X} \right) - \frac{J}{I_X} \dot{\theta} (-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4) + \frac{l}{I_X} b (\Omega_4^2 - \Omega_2^2) \tag{21}$$

$$\ddot{\theta} = \dot{\psi} \dot{\phi} \left( \frac{I_X - I_Z}{I_Y} \right) + \frac{J}{I_Y} \dot{\phi} (-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4) + \frac{l}{I_Y} b (\Omega_3^2 - \Omega_1^2) \tag{22}$$

$$\ddot{\psi} = \dot{\theta} \dot{\phi} \left( \frac{I_X - I_Y}{I_Z} \right) + \frac{J}{I_Z} (-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \tag{23}$$

$$\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi - S\psi C\phi & C\psi S\theta C\phi + S\psi S\phi \\ S\psi C\theta & S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi - C\psi S\phi \\ -S\theta & C\theta S\phi & C\theta C\phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \frac{b}{m} (\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{24}$$

Equations 25, 26 and 27 give the values of  $\ddot{x}$ ,  $\ddot{y}$  and  $\ddot{z}$ ;

$$\ddot{x} = (C\psi S\theta S\phi + S\psi C\phi) \frac{b}{m} (\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{25}$$

$$\ddot{y} = (S\psi S\theta C\phi + C\psi S\phi) \frac{b}{m} (\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{26}$$

$$\ddot{z} = -g + (C\theta C\phi) \frac{b}{m} (\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{27}$$

When the inputs of the system are selected as in equations 28, 29, 30 and 31 due to their convenience and convenience,

$$U_1 = b ((\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)) \tag{28}$$

$$U_2 = b ((\Omega_4^2 - \Omega_2^2)) \tag{29}$$

$$U_3 = b ((\Omega_3^2 - \Omega_1^2)) \tag{30}$$

$$U_4 = d ((-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)) \tag{31}$$

The complete mathematical model equations of the modeled drone vehicle are obtained as in equations 32, 33 and 34 and used in the drone design.

$$\ddot{\phi} = \dot{\psi} \dot{\theta} \left( \frac{I_Y - I_Z}{I_X} \right) - \frac{J}{I_X} \dot{\theta} (-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4) + \frac{l}{I_X} U_2 \tag{32}$$

$$\ddot{\theta} = \dot{\psi} \dot{\phi} \left( \frac{I_X - I_Z}{I_Y} \right) + \frac{J}{I_Y} \dot{\phi} (-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4) + \frac{l}{I_Y} U_3 \tag{33}$$

$$\ddot{\psi} = \dot{\theta} \dot{\phi} \left( \frac{I_X - I_Y}{I_Z} \right) + \frac{l}{I_Y} U_4 \tag{34}$$

### 3. MATERIALS AND METHODS

We applied the necessary processes and algorithms to transfer the drone prepared in the Solidworks 2020 program with the URDF exporter method. CoppeliaSim is a multi-purpose robot simulation that provides software that allows integration of control algorithms or links with external applications to run models[10]. However, we preferred CoppeliaSim, which is one of the many platforms developed and used for the design and simulation of autonomous systems, in this study because it is one of the most widely used simulators for the training of autonomous systems. The scene objects it hosts are embedded in an embedded script that runs simultaneously, creating a virtual world and interacting at runtime.

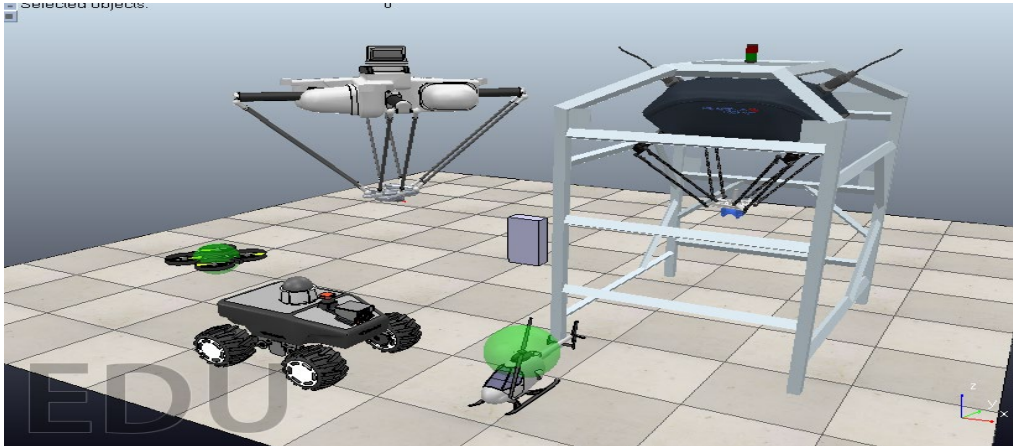


Fig. 2 - General view of the CoppeliaSim simulator, some models of built-in mobile and non-mobile vehicles

A sufficient number of examples have autonomous models, sensors and actuators on its platform. In this simulation study, with CoppeliaSim program tools, drone-path planning, path tracking, tracking-control and speed control of propellers with the necessary commands over the MATLAB program are performed. Below, Figure 2 shows the general view of the CoppeliaSim simulator, and some of its internal mobile and non-mobile models.

To synchronize the CoppeliaSim simulator and MATLAB, a working folder is created to connect MATLAB to CoppeliaSim. This folder view is shown in Figure 3.

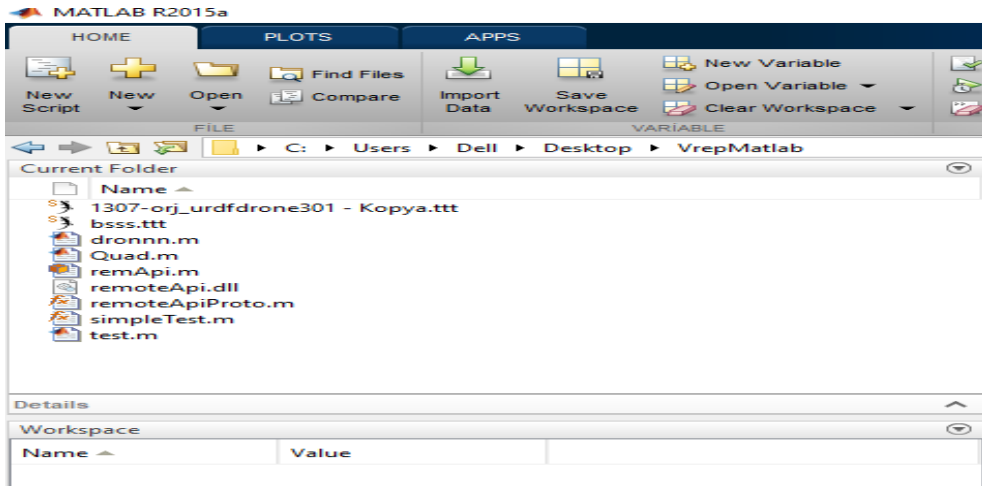


Fig. 3 - Work folder prepared to connect MATLAB to CoppeliaSim

In this folder; In the CoppeliaSim program files and the VrepMatlab folders we prepared;

- \*SimpleTest.m
- \*remoteApi.m
- \*remoteApi.dll
- \*remoteApiProto.m
- \*dronnn.m
- \*1307-orj\_kopya.ttt
- \*.m-file files and our CoppeliaSim simulation file.

The command 'simRemoteApi.start(19999)' found in the simpleTest.m-file in the folder opened in the MATLAB environment allows the connection or communication of the CoppeliaSim program and the MATLAB program to support API function calls that are exactly the same as in the CoppeliaSim script, after running the 1307-orj\_urdfdrone301 - Copy.ttt file in the CoppeliaSim virtual environment that was prepared before making the MATLAB Connection with the CoppeliaSim Program (Figure 4).

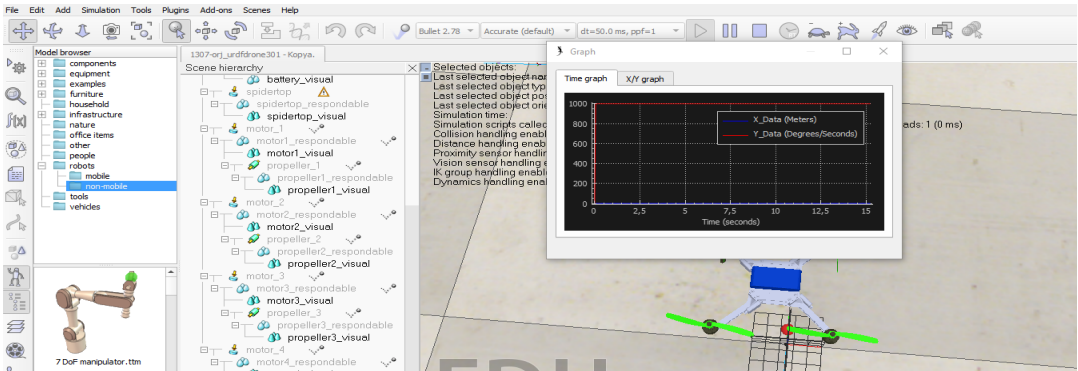


Fig. 4 - 1307-orj\_urdfdrone301 - Executing copy.ttt in CoppeliaSim environment

The simpleTest.m file is run through the VrepMatlab folder defined in the MATLAB environment. Thus, a successful connection is made between the CoppeliaSim program and MATLAB. The image showing successful connection is shown in Figure 5. The graph in the Figure presents the data during the simulation run of the CoppeliaSim file without running the propeller speed control file in the MATLAB environment. The SimpleTest.m file is shown in Figure 6.

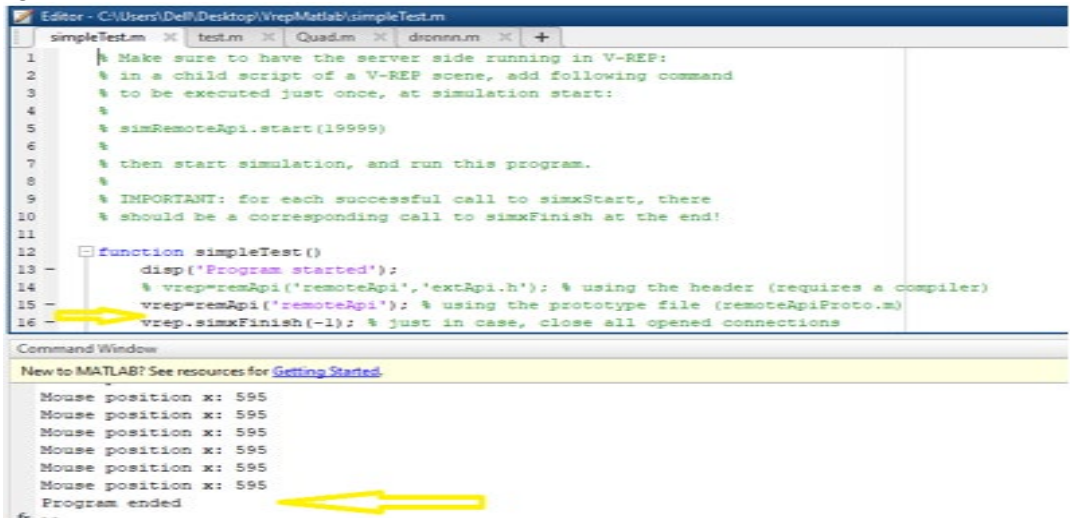


Fig. 5 - Connection of MATLAB-CoppeliaSim with simpleTest.m file

The graph in the Figure presents the data during the simulation run of the CoppeliaSim file without running the propeller speed control file in the MATLAB environment.

### 3. CONTROL OF THE ANGLE SPEED OF DRON PROPELLERS IN COPPELIASIM SIMULATION ENVIRONMENT THROUGH MATLAB PROGRAM

- In this study, MATLAB and CoppeliaSim robot simulator are controlled by providing synchronization. Before the control process, instead of the Quadcopter tool in Figure 6 in the CoppeliaSim program, the drone that we designed in the Solidworks program was placed in the CoppeliaSim environment using the necessary processes and



algorithms. In step 1, the 1307-orj\_urdfdrone301 - Copy – Copy.ttt file designed and prepared in CoppeliaSim environment is run. In the next step, the dronn.m file in the VrepMatlab folder that has been prepared is run in the MATLAB environment. The dronn.m file codes that we prepared in the third step are starting to compile. With the compilation of the codes, the angular velocity of the 2th propeller in the CoppeliaSim virtual environment is reduced to 2 rad/sec. In accordance with the 9th line of the Program Code fragment, the 2th propeller continues at this speed for 5 seconds (timing control). At the end of this period, the angular velocity of the 2th propeller is 12 rad/sec. Propeller 2 continues to rotate at this speed until the end of the simulation. It goes out. An image of these process steps is shown in Figure 7.

```

% simRemoteApi.start(19999)
%
% then start simulation, and run this program.
%
% IMPORTANT: for each successful call to simxStart, there
% should be a corresponding call to simxFinish at the end!

function simpleTest ()
    disp('Program started');
    % vrep=remApi('remoteApi','extApi.h'); % using the header (requires a compiler)
    vrep=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
    vrep.simxFinish(-1); % just in case, close all opened connections
    clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);

    if (clientID>-1)
        disp('Connected to remote API server');

        % Now try to retrieve data in a blocking fashion (i.e. a service call):
[res,objs]=vrep.simxGetObjects(clientID,vrep.sim_handle_all,vrep.simx_opmode_blocki
ng);
        if (res==vrep.sim_return_ok)
            fprintf('Number of objects in the scene: %d\n',length(objs));
        else
            fprintf('Remote API function call returned with error code: %d\n',res);
        end
        pause(2);

        % Now retrieve streaming data (i.e. in a non-blocking fashion):
t=clock;
startTime=t(6);
currentTime=t(6);

vrep.simxGetIntegerParameter(clientID,vrep.sim_intparam_mouse_x,vrep.simx_opmode_st
reaming); % initialize streaming
while (currentTime-startTime < 5)

```

Fig. 6 - SimpleTest.m file Matlab and Vrep connection codes

```

Editor - C:\Users\Del\l\Desktop\VrepMatlab\dronn.m
simpleTest.m x test.m x Quad.m x dronn.m x +
1 - vrep=remApi('remoteApi');
2 - vrep.simxFinish(-1);
3 - clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
4
5 - if (clientID>-1)
6 -     disp('Connected')
7 -     [returnCode,propeller_2]=vrep.simxGetObjectHandle(clientID,'propeller_2',vrep.:
8 -     [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_2,1,vrep.simx_
9 -     pause(8)
10 -    [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_2,12,vrep.simx_
11 -    %code here
12
13 -    vrep.simxFinish(-1);
14 - end
15
16 - vrep.delete();

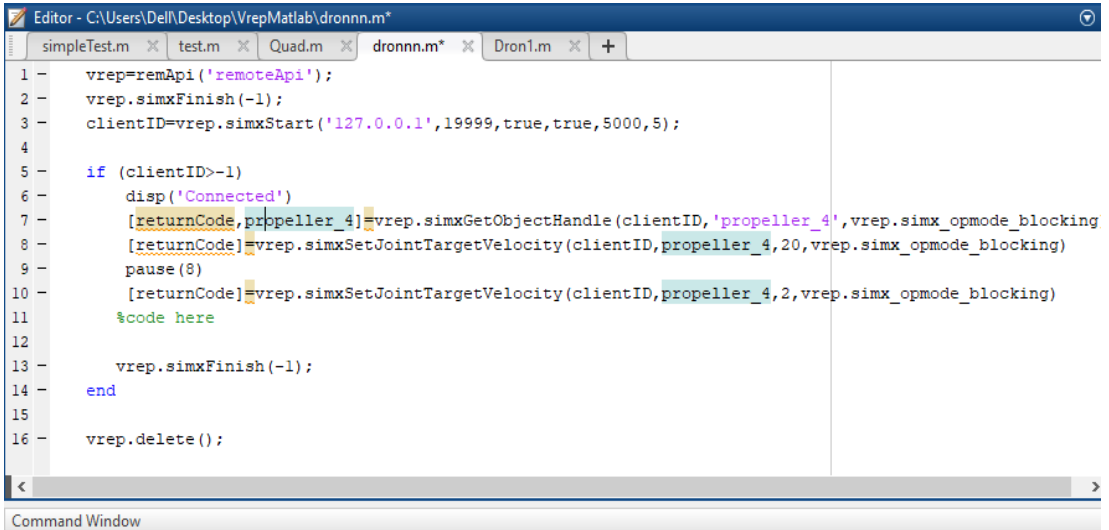
```

Command Window  
New to MATLAB? See resources for [Getting Started](#).

f >>

Fig. 7 - MATLAB code for controlling and changing the angular speed of the 2th propeller

The operations for the speed control of the 2nd propeller are also done for the speed control of the 4th propeller with different speed changes. With the compilation of the codes, the angular velocity of the 4th propeller in the CoppeliaSim virtual environment is reduced to 20 rad/sec. In accordance with the 9th line of the Program Code, the 4th propeller continues at this speed for 8 seconds (timing control). At the end of this period, the angular velocity of the 4th propeller is 2 rad/sec. Propeller 4 continues to rotate at this speed until the end of the simulation. An image of these process steps is shown in Figure 8. These speed changes were made for the 1st and 3rd propellers for different speeds. The code body prepared for the 1st propeller is shown in Figure 9 for the 3rd propeller in Figure 10.



```

Editor - C:\Users\Del\\Desktop\VrepMatlab\dronnn.m
simpleTest.m x test.m x Quad.m x dronnn.m* x Dron1.m x +
1 - vrep=remApi('remoteApi');
2 - vrep.simxFinish(-1);
3 - clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
4
5 - if (clientID>-1)
6 -     disp('Connected')
7 -     [returnCode,propeller_4]=vrep.simxGetObjectHandle(clientID,'propeller_4',vrep.simx_opmode_blocking)
8 -     [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_4,20,vrep.simx_opmode_blocking)
9 -     pause(8)
10 -    [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_4,2,vrep.simx_opmode_blocking)
11 -    %code here
12
13 -    vrep.simxFinish(-1);
14 - end
15
16 - vrep.delete();
Command Window

```

Fig. 8 – Angular speed control of the 4th propeller and the MATLAB code that determines the running time at this speed



```

Editor - C:\Users\Del\\Desktop\VrepMatlab\dronnn.m
simpleTest.m x test.m x Quad.m x dronnn.m x Dron1.m x +
1 - vrep=remApi('remoteApi');
2 - vrep.simxFinish(-1);
3 - clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
4
5 - if (clientID>-1)
6 -     disp('Connected')
7 -     [returnCode,propeller_1]=vrep.simxGetObjectHandle(clientID,'propeller_1',vrep.simx_opmode_blocking)
8 -     [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_1,5,vrep.simx_opmode_blocking)
9 -     pause(8)
10 -    [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_1,15,vrep.simx_opmode_blocking)
11 -    %code here
12
13 -    vrep.simxFinish(-1);
14 - end
15
16 - vrep.delete();

```

Fig. 9 – MATLAB code for controlling and changing the angular speed of the 1st propeller

```

Editor - C:\Users\Dell\Desktop\VrepMatlab\dronnn.m*
simpleTest.m x test.m x Quad.m x dronnn.m* x Dron1.m x +
1 - vrep=remApi('remoteApi');
2 - vrep.simxFinish(-1);
3 - clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
4
5 - if (clientID>-1)
6 -     disp('Connected')
7 -     [returnCode,propeller_3]=vrep.simxGetObjectHandle(clientID,'propeller_3',vrep.simx_opmode_blocking)
8 -     [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_3,7,vrep.simx_opmode_blocking)
9 -     pause(8)
10 -    [returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_3,18,vrep.simx_opmode_blocking)
11 -    %code here
12
13 -    vrep.simxFinish(-1);
14 - end
15
16 - vrep.delete();

```

Fig. 10 – MATLAB code for controlling and changing the angular speed of the 3rd propeller

```

vrep=remApi('remoteApi');
vrep.simxFinish(-1);
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);
if (clientID>-1)
    disp('Connected')
[returnCode,propeller_4]=vrep.simxGetObjectHandle(clientID,'propeller_4',vrep.simx_opmode_blocking)
[returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_4,20,vrep.simx_opmode_blocking)
    pause(10)
[returnCode]=vrep.simxSetJointTargetVelocity(clientID,propeller_4,2,vrep.simx_opmode_blocking)
    %code here
    vrep.simxFinish(-1);
end
vrep.delete();

```

The usage and functioning of some of the codes used;

[returnCode number handle]=simxGetObjectHandle(number clientID,string objectName,vrep.simx\_number operationMode) in the code line;

number handle: The name used by the plugin to be evaluated in the simulation is defined.

simxGetObjectHandle: Since CoppeliaSim has its own function, it is used as vrep.simxGetObjectHandle in MATLAB.

Number operationMode : It is the function used according to the feature and purpose of the plugin to be used.

vrep.simxSetJointTargetVelocity: It is the code that contains the ID information of the evaluated plugin, the plugin name, the targeted speed of the plugin and the usage function values. In the usage format, it is used as 'Targetvelocity' for the 'number jointHandle' code.

'The code 'number targetVelocity' is used as the code where the target speed of the plugin is determined.

### 4. RESULTS

In this study, the graphical information obtained from the MATLAB environment according to the drone simulation with the communication between the CoppeliaSim simulator and MATLAB according to the propeller number, first angular velocity, next angular velocity and the running time of the last active state are shown in Figures 11, 12 and 13.

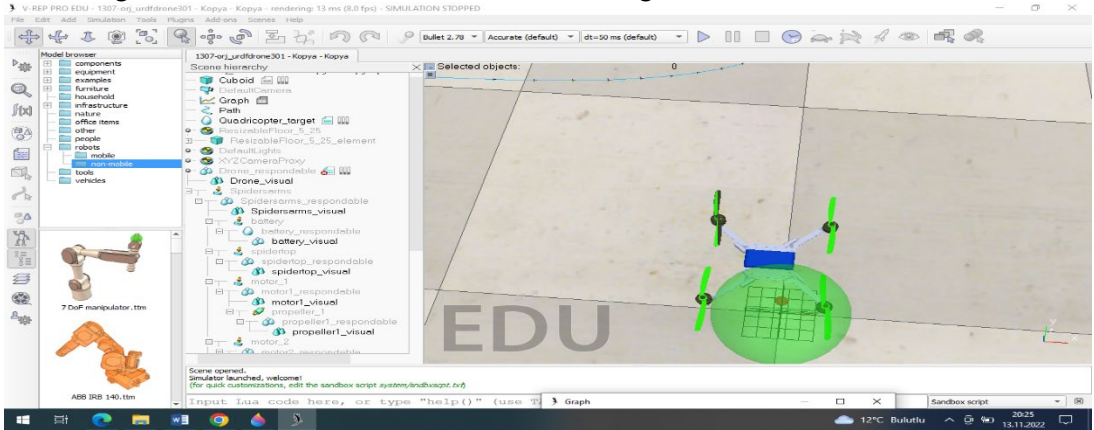


Fig. 11 - General view of CoppeliaSim simulator and designed drone view

From the information given in Figures 11, 12 and 13 we can notice the following: Figure 11 shows only the data of the CoppeliaSim file during the simulation run.

Then upon execution of the file “dronnn.m” in MATLAB, it is reduced to its speed in this file (100 deg/s) and then rotates at reduced speed during the waiting time in the code.

At the end of the waiting period, the propeller continues to rotate at this speed by reaching the increase speed in the code.

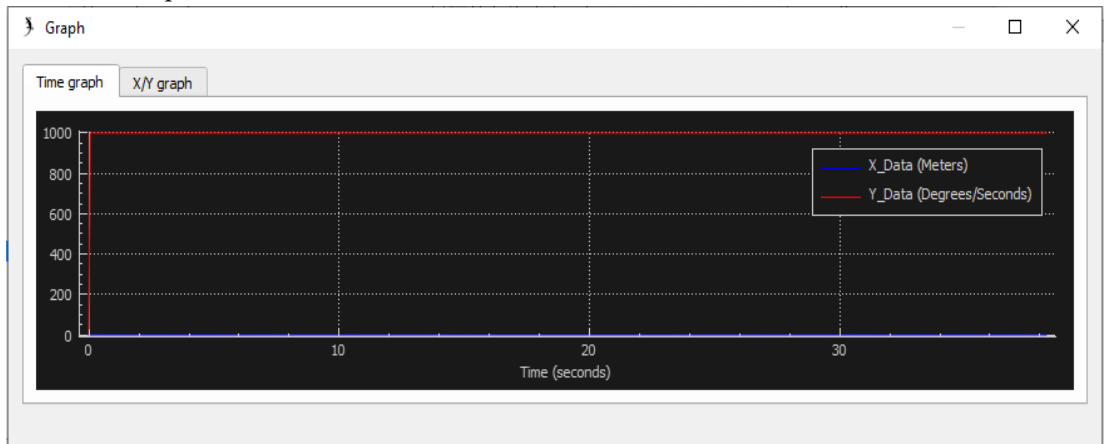


Fig. 12 - Graphic information obtained when the control file is not active for 2nd propeller

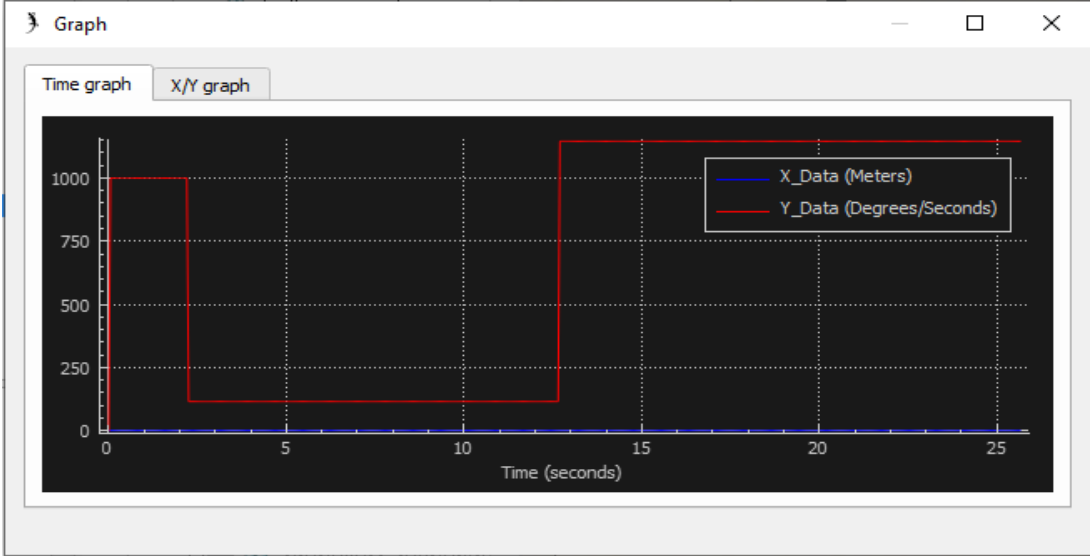


Fig. 13 - Graphic information obtained while the Control file in MATLAB is active for 2nd propeller

Again, from the information given in Figures 14 and 15 below we can state the following: Figure 14 only has the simulation data of the CoppeliaSim file during the simulation run. In Figure 12, while the 1307\_drone\_kopya.ttt file in CoppeliaSim is running (0), the speed of the 2 propeller, which reaches its operating speed very soon after takeoff, is reduced to the speed in this file with the execution of the "dronnn.m" file in MATLAB, then the waiting time in the code rotates at reduced speed for the duration. At the end of the waiting period, the propeller continues to rotate at this speed by reaching the increase speed in the code

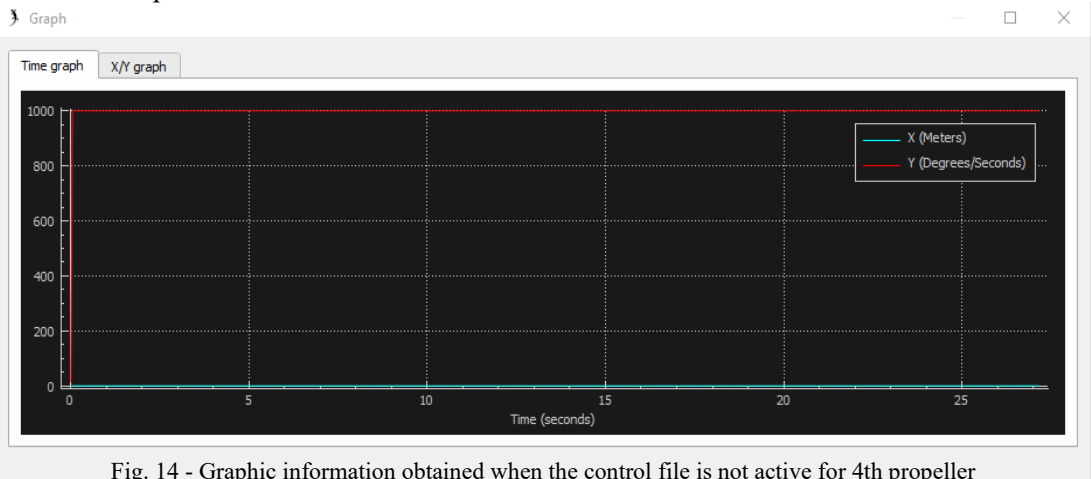


Fig. 14 - Graphic information obtained when the control file is not active for 4th propeller

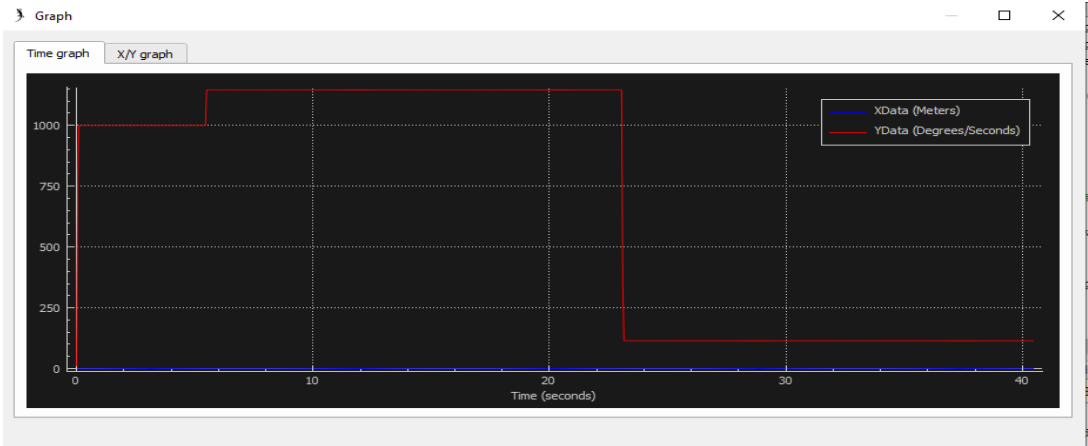


Fig. 15 - Graphic information obtained while the Control file in MATLAB is active for 4th propeller

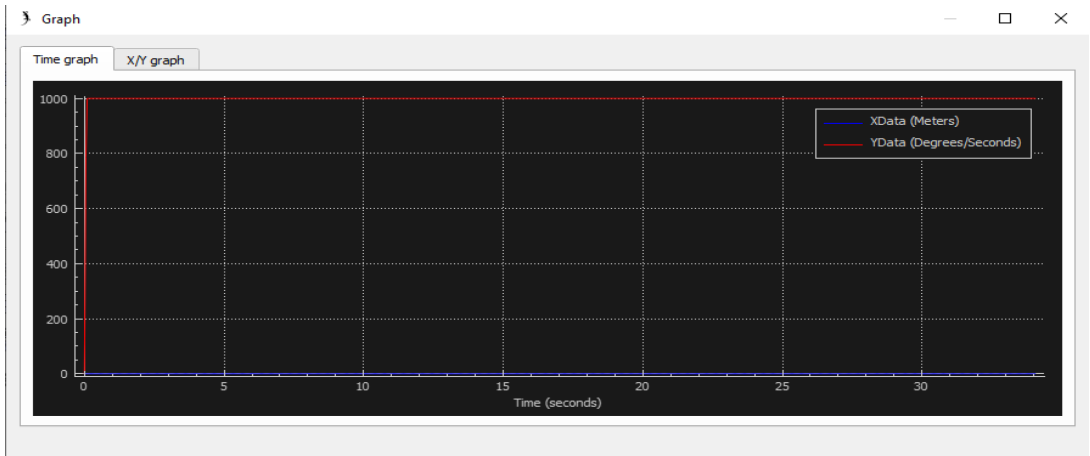


Fig. 16 - Graphic information obtained when the control file is not active for 1st propeller

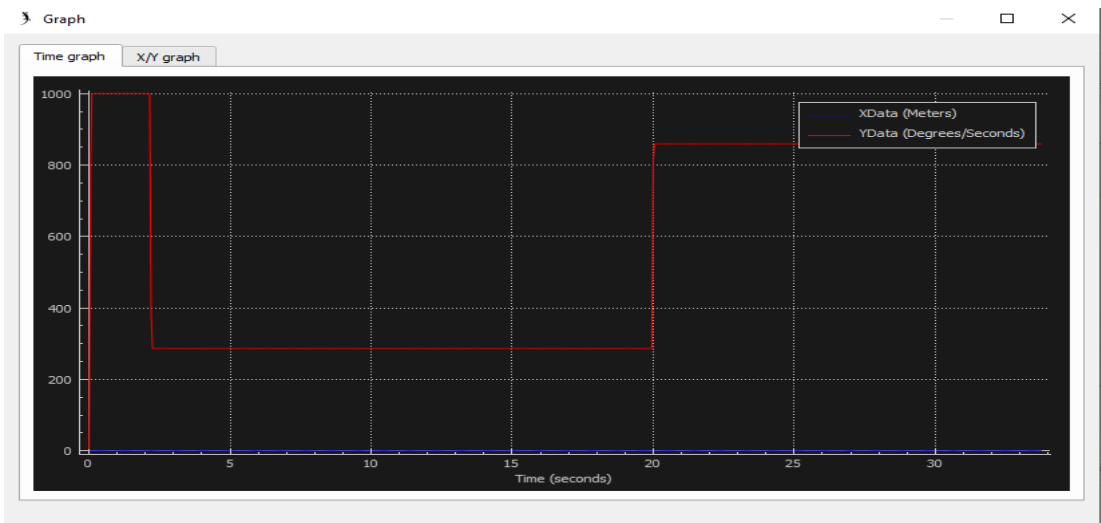


Fig. 17 - Graphic information obtained when the control file in MATLAB is active for 1st propeller

## 5. CONCLUSIONS

In this study, the drone we designed with SolidWorks 2020 was transferred to the CoppeliaSim simulation environment using the URDF exporter method. Afterwards, a successful connection was established between MATLAB and CoppeliaSim simulator. The angular velocity change is shown in Figures 12, 13, 14, 15, 16, , and 17 respectively, by running and without running the CoppeliaSim simulation file and the propeller speed control file in MATLAB at the same time. The MATLAB code body that provides the drone propeller angular speed control and speed change timing is shown in Figure 9. The working compatibility and efficiency between MATLAB and CoppeliaSim, which is another aim of the study, is shown in Figure 5.

When the data obtained as a result of the simulation are evaluated, it has been shown that the communication between the CoppeliaSim simulator and MATLAB is done successfully. At the same time, the drone we designed in the CoppeliaSim simulator environment, which is another aim of our study, was run and it was shown that the propeller angular velocity control of the code prepared in the MATLAB environment was done as desired.

## ACKNOWLEDGEMENT

The whole responsibility for the accuracy of calculations, experimental data and scientific interpretation belongs entirely to the authors.

The authors declare on their own responsibility that the paper has not been previously published elsewhere.

## REFERENCES

- [1] Y. Chen & F. Dong, Robot machining: recent development and future research issues, *Int J Adv Manuf Technol*, vol. **66**, pp.1489–1497, 2013.
- [2] E. Peralta, E. Fabregas, G. Farias, H. Vargas, S. Dormido, Development of a Khepera IV Library for the V-REP Simulator, *IFAC-PapersOnLine*, vol. **49**, no. 6, pp. 081–086, 2016.
- [3] G. Farias, E. Fabregas, E. Peralta, E. Torres, S. Dormido, A Khepera IV library for robotic control education using V-REP, *IFAC PapersOnLine*, vol. **50**, no.1, pp. 9150–9155, 2017.
- [4] L. Nogueira, Comparative Analysis Between Gazebo and V-REP Robotic Simulators, *School of Electrical and Computer Engineering Universidade de Campinas*, December 2014, DOI:10.13140/RG.2.2.18282.36808
- [5] B. Bogaerts, S. Sels, S. Vanlanduit, R. Penne, Connecting the CoppeliaSim robotics simulator to virtual reality, 2020, <https://doi.org/10.1016/j.softx.2020.100426>.
- [6] M. Casini, A. Garulli, MARS: a Matlab simulator for mobile robotics experiments., *IFAC-PapersOnLine* vol. **49**, no. 6, pp. 069–074, 2016.
- [7] E. Rohmer, P. Surya, N. Singh and M. Freese, V-REP: a Versatile and Scalable Robot Simulation Framework. *International Conference on Intelligent Robots and Systems (IROS) IEEE*, pp. 1321-1326, 2013.
- [8] N. Koenig, A. Howard, Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator, *International Conference on Intelligent Robots and Systems (IROS) IEEE*, pp. 2149-2154, 2004.
- [9] S. Rooban, Shaik Dilawar Suraj, Shaik Babji Vali, Nagandla Dhanush, *CoppeliaSim: Adaptable modular robot and its different locomotions simulation framework*, <https://doi.org/10.1016/j.matpr.2021.01.055>.
- [10] M. Ciszewski, T. Buratowski, M. Ciszewski, T. Buratowski, Modeling, Simulation and Control of a Pipe Inspection Mobile Robot. with an Active Adaptation System. *IFAC PapersOnLine*, vol. **51** no. 22, pp.132–137, 2018.