Research Article

INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS

International Journal of Advanced Robotic Systems
January-February 2019: 1–19

DOI: 10.1177/1729881418820425
journals.sagepub.com/home/arx
SAGE

Drone challenge: A platform for promoting programming and robotics skills in K-12 education

Aurelio Bermúdez, Rafael Casado, Guillermo Fernández, María Guijarro and Pablo Olivas

Abstract

The development of skills related to computer programming and robotics and the introduction of computational thinking principles in high schools are worldwide trends today. An effective way of initiating young students in this world consists in proposing them stimulating challenges. This work presents a robotic platform that has been successfully used to develop a competition (called *Drone Challenge*) in which students had to program the navigation system for a simulated unmanned aerial vehicle (or drone). Both the competition and the supporting platform are described in detail. In particular, the article provides a deep technical description of the main components of the platform, namely the drone simulator and the navigation development framework. The results of the survey conducted after the challenge point to the suitability of the working platform deployed.

Keywords

Engineering education, computational thinking, K-12 education, STEM education, ICT/computing skills, educational robotics

Date received: 28 February 2018; accepted: 29 November 2018

Topic: Service Robotics
Topic Editor: Marco Ceccarelli
Associate Editor: Erwin-Christian Lovasz

Introduction

In the literature, many works can be found proposing ways to teach computer programming and, in general, to engage K-12 students with Science, Technology, Engineering, and Mathematics (STEM) subjects. Most of them are focused on robotics programming. For example, He et al.[1] describe a summer K-12 robotics course. Bower[2] discusses strategies for teaching beginner students how to program mobile robots for autonomous operation. In the study by Ilori and Watchorn,[3] a robotics mentorship program mixes undergraduate engineering students with primary and secondary ones. A robotics unit is integrated into a fourth-grade science curriculum.[4] West et al.[5] rely on Arduino-based robot kits and real and simulated exploration rovers. In many cases, these proposals are based on the development of robotics challenges for students, such as the well-known *RoboCupJunior*,[6] the *IT-Adventures*,[7] the *Junior Soccer Simulation*,[8] and the *Zero Robotics*[9] competitions. Game-based learning environments have also been used to motivate young students. This is the case of the LOGO-like environment proposed by Paliokas et al.[10] and the robot

Computing Systems Department, Albacete Research Institute of Informatics, University of Castilla-La Mancha, Campus Universitario, Albacete, Spain

Corresponding author:
Aurelio Bermúdez, Albacete Research Institute of Informatics, University of Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain.
Email: aurelio.bermudez@uclm.es

game environment described by Shim et al.,[11] which combines tangible user interfaces with an educational robot.

At the same time, there is a new trend in K-12 education, referred to as computational thinking. Computational thinking is based on the principle of applying concepts and practices borrowed from the computer programming world to solve problems in any discipline (not only the computing one). Computational concepts include sequences, loops, conditionals, operators, and so on. Some examples of computational practices are experimentation and iteration, testing and debugging, abstracting and modularizing, and reusing and mixing. Since the seminal work of Wing,[12] the number of initiatives introducing these ideas in the teaching/learning process at K-12 level has notably grown. A complete review of such efforts can be found in the study by Heintz et al.[13] Perhaps the most popular approaches are the *Scratch*,[14] *Blocky*,[15] and *Code.org*[16] block-based (or visual) programming environments. Turchi and Malizia[17] suggest extending this kind of environments with tangible user interfaces. Alternative initiatives to block-based coding[18] rely on digital storytelling, and studies in the literature [19,20] propose using robotics (like the current work) for promoting computational thinking. Visual programming and robotics also meet at the popular *Makeblock* platform[21] and in the LEGO-based platform described by Yadagiri et al.[22] Finally, García-Peñalvo and Mendes[23] explore the effects of recent computational thinking experiences in pre-university education.

In this context, and with the goal of promoting skills related to computer programming and robotics among the secondary school students of the Spanish region of Castilla-La Mancha, the Faculty of Computer Science Engineering of the University of Castilla-La Mancha[24] designed a drone programming competition consisting in the development of the automatic navigation system for an unmanned aerial vehicle.[25] The name of this competition is *Drone Challenge*,[26] and its two first editions were carried out from April to June 2017 and from February to April 2018, respectively.

In this work, aside from describing the goals, rules and phases of the competition, and the way in which it was carried out, we present in detail the working platform deployed for the participants, since we consider that it is an excellent baseline for the development of computer programming skills at K-12 level. At a glance, the platform allows the students to design their proposals using a development environment based on Matlab/Simulink [version: R2017a, MathWorks].[27] Then, they can evaluate the performance of their designs in a simulation environment based on the robot operating system (ROS)[28] and Gazebo.[29] Furthermore, the article provides the results of a survey conducted after the competition with the aim of collecting the participants' opinion about the tools employed and the development of the challenge.

To sum up, the main contributions of this work are threefold: (1) the proposal of a complete robotic platform for the development and test of drone navigation systems; (2) the description of a challenge for secondary students based on this platform; (3) an evaluation of both the platform and the competition, based on the opinions provided by the participants in the event.

The rest of this work is structured as follows. First, in the second section, the *Drone Challenge* competition is described. Then, in the third section, the working platform is detailed. After that, in the fourth section, the contents of the survey proposed and an overview of the responses collected are presented. Finally, in the fifth section, conclusions and some future works derived from this work are summarized.

The drone challenge competition

Drone Challenge is a contest for student teams, in which each of them must program the automatic navigation system for an unmanned aerial vehicle (from now on, just "drone"). The drone considered is a quadcopter, since this is the most popular type of drone today. In fact, although navigation systems are tested in a simulation environment, the final prize is a commercial drone for each one of the components of the winner team.

In particular, each team must provide some intelligence to a simulated quadcopter, so that it is able to take off from a base, cross several floating frames in a specific order (red, green, and blue), and finally land over the starting point. Figure 1 shows the competition arena, in which we can observe the colored frames, the drone on its takeoff base, and a golden carpet identifying the allowed flying area. We assume that the drone is equipped with a camera, which helps in the process of locating the frames. Each milestone (crossing a frame, crossing all the frames in the right order, and landing over the starting point) provides a point for the try. In case of a tie, the time of each try is also considered.

Obviously, the solution to the challenge is not unique. Each team can choose its own strategy for the drone navigation system. At the same time, the performance of two implementations for the same strategy may be different, resulting in different flight times. Therefore, the challenge proposed stimulates the creativity of the participants. This concept is very close to the concept of computational thinking. Indeed, the expressions "creative computation" and "creative thinking" are often used as synonyms or complementary of computational thinking.

To evaluate the quality of each proposal, the navigation system developed is tested in different scenarios (or simulated worlds), in which the position of the colored frames, the takeoff base, and the drone's initial orientation vary. Ideally, the drone should exhibit exactly the same behavior (taking off, crossing the frames in the right order, and landing) in all of them.

Although students have to deal with several complex tools, an effort has been made to adapt these tools to their level, not being required to have previous experience with them. Also, there are lots of low-level implementation details such as the drone stability control, the way in which video is processed, and the public–subscribe protocol used

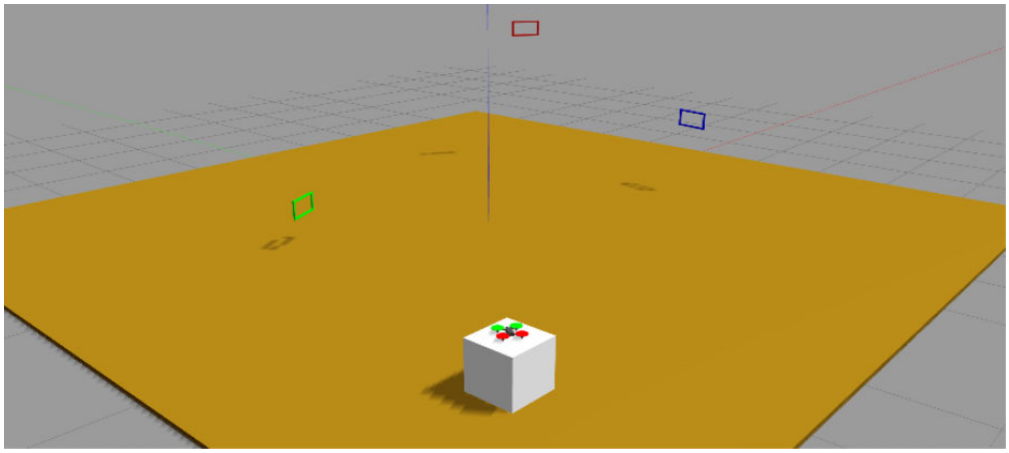


Figure 1. The competition arena.

for the communication between tools, which participants do not need to know to develop their proposals. Third section focuses on all these implementation aspects.

Regarding the phases of the competition, there is an initial phase, which lasts between 8 weeks and 10 weeks, in which the enrolled teams have to develop their proposals in their respective schools. Then, the submitted navigation systems are assessed to determine the set of teams that will finally participate in the final phase of the competition. In the first edition of the competition, 20 teams coming from all the provinces of the region of Castilla-La Mancha were enrolled in this initial phase. In the second edition, the number of enrolled teams grew to 62. Since each team is composed of two, three, or four students supervised by a teacher, more than 80 people in the first edition and 260 people in the second edition were working in their proposals during this phase of the competition.

The final phase took place in the facilities of the Faculty of Computer Science Engineering, on June 30, 2017 and on April 23, 2018. Only 7 teams, involving a total of 30 people, and 14 teams, involving a total of 59 people, participated in this final phase in each edition. Regarding the scheduling of the final phase, during the first hour, teams test their navigation systems in seven different scenarios, and they can make last-minute tunings to their implementations. Then, all the proposals are frozen, and three of the previous scenarios are randomly chosen. During the last hour of the final, proposals are tested in these three scenarios under the supervision of the judges of the competition.

A supporting application, the *Game Monitor*, is available for the participants during the initial phase of the competition, so that they can know and improve the quality of their proposals, and it is also employed by the judges during the final phase. It is a separate Matlab/Simulink program that connects to the simulation environment to measure the time required to complete each try. In particular, while the simulation is running, it provides the time during which the drone engines are on and shows the location of the drone in a top view of the scenario (see Figure 2). This tool is also in charge of checking whether the drone exits the flying scenario (a $10 \times 10 \times 4$ m^3), since this situation must be penalized (see Figure 3), and whether the try has reached the maximum allowed time (set to 3 min).

Apart from the *Game Monitor*, a small client–server application was developed for the final phase so that, while the judges of the competition are assessing proposals, participants can check their "virtual" place in an instantaneous ranking. To have this ranking updated in real time, each judge uses a smartphone to fill in a Web form with the results of each try and sends immediately this information to the server.

Finally, with respect to the interaction with the participants during the competition, it is mainly supported by its "official" blog,[26] which has been conceived as a dynamic tool. In this space, students can access to a set of video tutorials organized in several categories. Figure 4 shows an example. Through these tutorials, the organization provides detailed instructions for installing and configuring the working environment and gives advice and suggestions for the development of the proposals. These tutorials also offer solutions to the problems or doubts that the participants pose as they work on their designs. On average, each of these tutorials has received more than 100 views during the first edition of the challenge and more than 160 views during the second one. Video tutorials are complemented by the traditional "Frequently asked questions" section of the blog.

The robotic platform

The working environment employed by the participants in the *Drone Challenge* competition described in the previous section is composed of two main components: a drone simulator and a navigation development framework. Next, we detail both components. As said above, most of the details provided here are hidden for the students, but they would be available if we wanted, for example, to use this platform for designing a different competition or for teaching purposes.

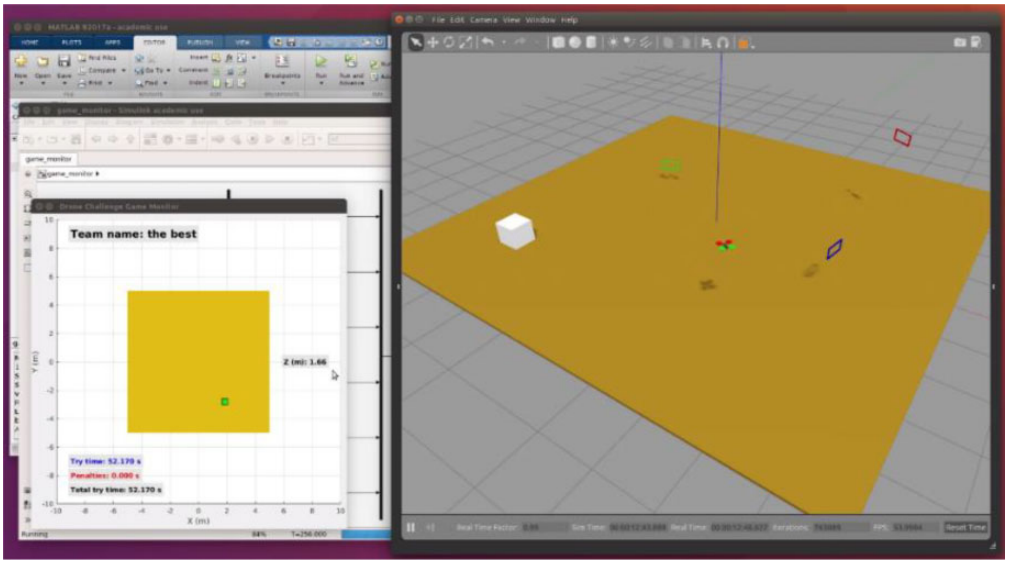


Figure 2. *Game monitor* snapshot.

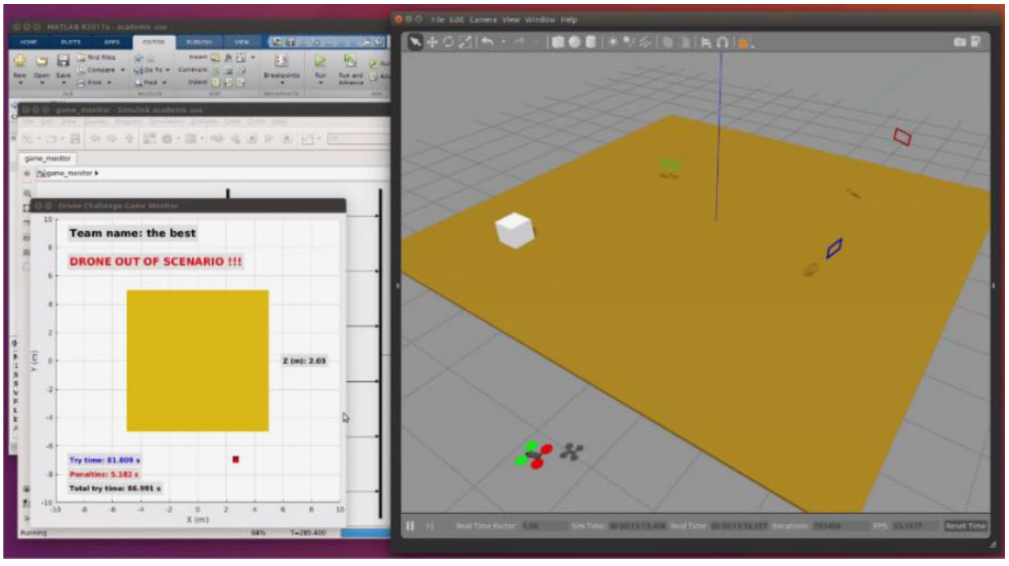


Figure 3. *Game monitor* snapshot (drone out of the scenario).

Drone simulator

The drone simulator (Figure 5) is based on Gazebo,[29] a realistic robotics simulator included in the popular ROS.[28] It runs on Ubuntu and allows testing the navigation system programmed in a simulated scenario. Students just must launch the Gazebo simulator, specifying the name of the scenario in which their program will be tested.

This subsection focuses on the design of the drone model incorporated into the simulator. As we will see, we have used Simulink for solving the model. Then, the solution obtained has been programmed in the ROS/Gazebo plugin managing the simulated drone.

Quadcopter state-space model. To model and control the drone, we have used the state–space theory of linear time-invariant systems.[30] At first, the nonlinear system dynamics is represented by the following state and output equations

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x})$$

Where $\mathbf{u}$ is the input vector, $\mathbf{x}$ is the state vector, and $\mathbf{y}$ is the output vector. Next, we define these vectors.

First, the system input vector $\mathbf{u} = [\Omega_{NE}\ \Omega_{NW}\ \Omega_{SE}\ \Omega_{SW}]^{\mathrm{T}}$ allows us to handle the rotor speed. Note that

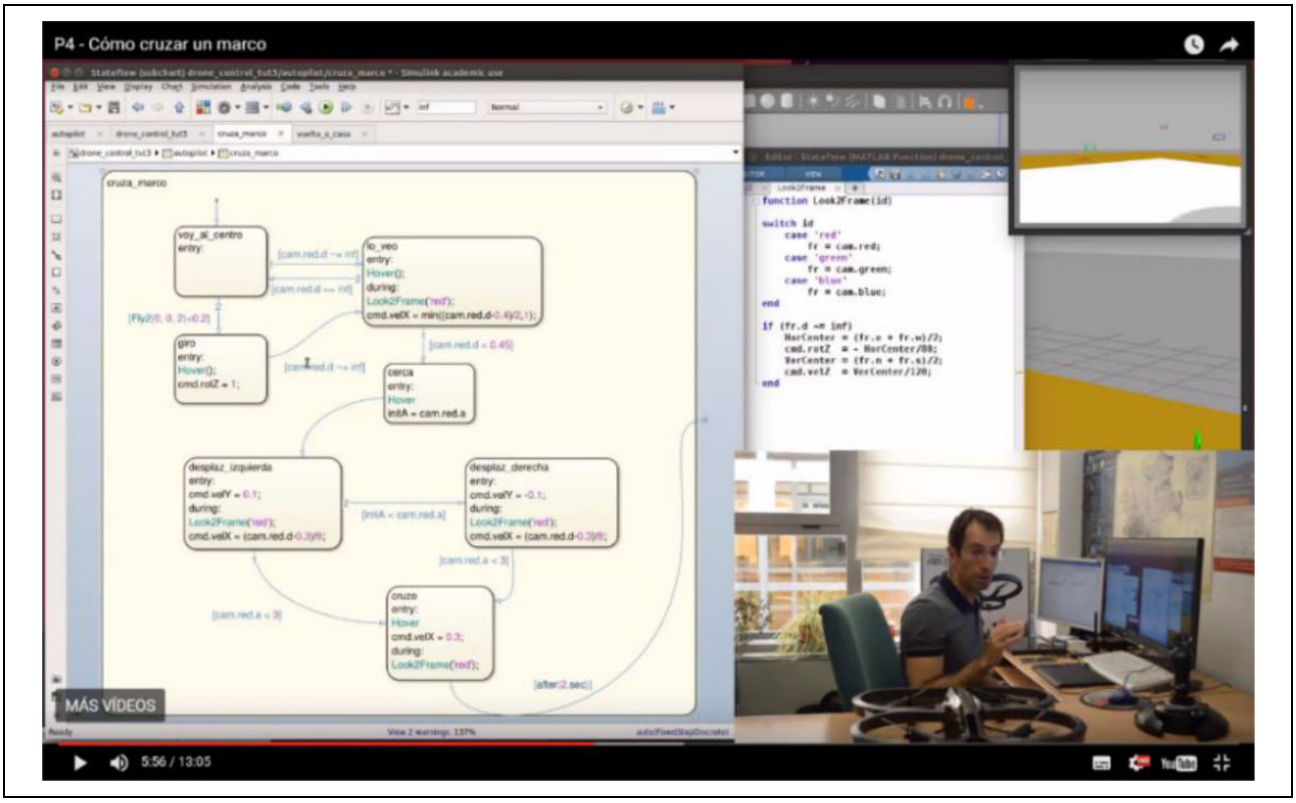


Figure 4. The "*how to cross a frame*" video tutorial.

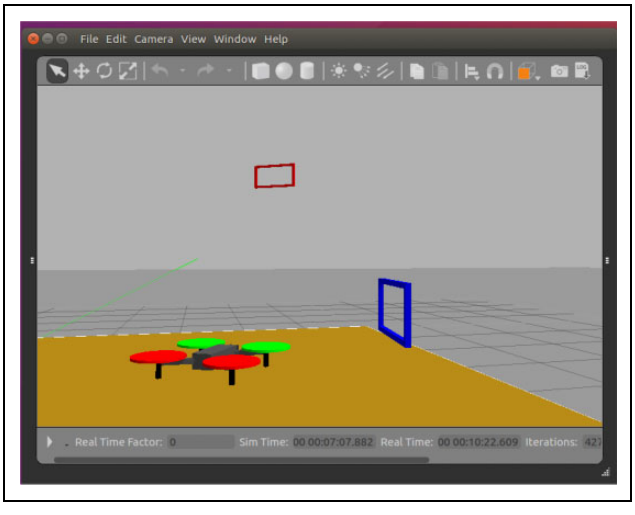


Figure 5. Drone simulator developed in Gazebo.

we use the transpose notation, expressing column vectors as rows.

From now on, we will use the ${}^{e}[]$ notation to refer to the earth coordinate system, where the X and Y axes are located on the floor, and the Z axis remains vertical, so that ${}^{e}X \times {}^{e}Y = {}^{e}Z$. In the same way, we use the ${}^{b}[]$ notation to refer to the drone (or "body") coordinate system, arranged as shown in Figure 6, and equally satisfying ${}^{b}X \times {}^{b}Y = {}^{b}Z$.

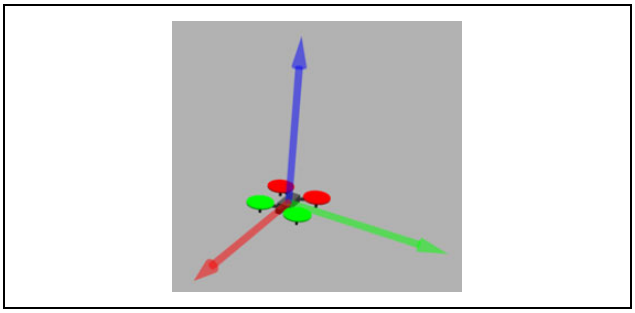


Figure 6. Drone reference system (X axis in red, Y axis in green, and Z axis in blue).

Let vector $\mathbf{p} = {}^{e}[x\ y\ z]^{\mathrm{T}}$ be the drone position according to the earth coordinate system, and let vector $\mathbf{o} = {}^{e}[\phi\ \theta\ \psi\ ]^{\mathrm{T}}$ be the drone orientation with respect to the X (roll), Y (pitch), and Z (yaw) axes of the earth. Let $\mathbf{v} = {}^{b}[\dot{x}\ \dot{y}\ \dot{z}]^{\mathrm{T}}$ and $\boldsymbol{\omega} = {}^{b}[\omega_x\ \omega_y\ \omega_z\ ]^{\mathrm{T}}$ be, respectively, the linear and angular speeds with respect to the drone coordinate system. Starting from this, the complete state of the drone is defined by means of the following vector: $[\mathbf{p}\ \mathbf{o}\ \mathbf{v}\ \boldsymbol{\omega}]^{\mathrm{T}}$. Figure 7 shows the way in which Simulink receives these 12 values from the Gazebo simulator.

Considering that the drone behavior does not depend on its location, from the point of view of the sustentation control we do not need the information provided by

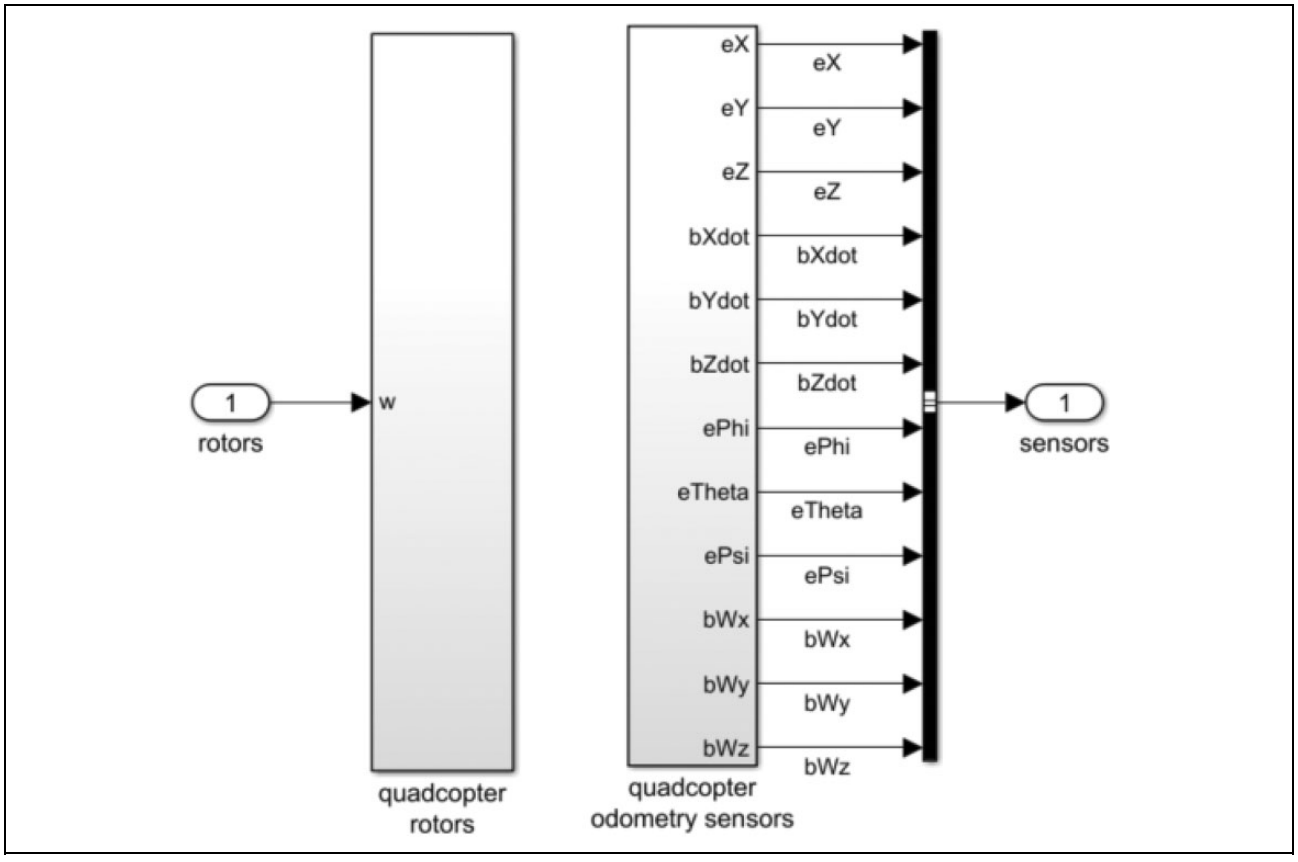


Figure 7. Drone position and orientation (odometry) in Simulink.

${}^{e}[\mathbf{p}\ \psi]^{\mathrm{T}}$. Therefore, the system can be defined by means of the internal state vector $\mathbf{x} = [{}^{e}\phi\ {}^{e}\theta\ {}^{b}\boldsymbol{\omega}\ {}^{b}\mathbf{v}\ ]^{\mathrm{T}}$.

Finally, we define the output vector $\mathbf{y} = {}^{b}[\mathbf{v}\ \omega_z]^{\mathrm{T}}$, so that it represents the low-level drone control. That is, we will be able to obtain the required forward speed in its three axes and the required rotation speed over the vertical axis.

Nonlinear drone dynamics. After defining the input ($\mathbf{u}$), state ($\mathbf{x}$), and output ($\mathbf{y}$) vectors, we have to model the system's behavior by means of the function $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. For this, we apply rigid body dynamics.

Translation movement is given by the set of forces acting over the object, according to the expression $m\,{}^{b}\ddot{\mathbf{p}} = \sum \mathbf{F}$. Similarly, rotation movement is given by the set of moments acting over the object, according to the expression $\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} + \boldsymbol{\omega} \times \mathbf{I}_r\Omega_r = \sum \mathbf{M}$. Fortunately, it is not necessary to handle directly these expressions if we employ the rigid body model with six degrees of freedom (DOF) incorporated into the aerospace Simulink library and shown in Figure 8. The 6-DOF block requires defining the mass and the inertia of the object as well as the different forces and moments that affect it. It provides varied information about the object movement, which we employ to build the output vector ($\mathbf{y}$).

The drone modeled looks like the one shown in left side of Figure 9. Rotors are arranged in a square, and they are located at 25 cm from the center of gravity. Each propeller covers a circumference of 20 cm in diameter. We have considered a mass of 300 g, and a total inertia equal to the inertia of a $40 \times 40 \times 5$ cm^3 box with uniform density. The right side of Figure 9 shows the drone inertial properties, defined according to the simulation description format (SDF) specification.[31]

$$m = 300\ \mathrm{g},\ I = \begin{bmatrix} 10.225 & 0 & 0 \\ 0 & 10.225 & 0 \\ 0 & 0 & 20 \end{bmatrix} \times 10^{-4}$$

Translation equations. The set of forces contributing to the drone translation movement are given by the expression $m\ddot{\mathbf{p}} = \sum \mathbf{F} = \vec{\mathbf{F}}_g + \vec{\mathbf{F}}_T + \vec{\mathbf{F}}_D$ (see Figure 10). Next, we describe these forces, called gravity ($\vec{\mathbf{F}}_g$), aerodynamic thrust ($\vec{\mathbf{F}}_T$), and aerodynamic drag ($\vec{\mathbf{F}}_D$).

The 6-DOF Simulink block requires input forces and moments expressed in the drone coordinate system. However, the gravity is initially defined with respect to the earth coordinate system, as ${}^{e}\vec{\mathbf{F}}_g = {}^{e}[0\ 0\ -mg]^{\mathrm{T}}$. Therefore, we can apply the transformation ${}^{b}\vec{\mathbf{F}}_g = \mathbf{D}\,{}^{e}\vec{\mathbf{F}}_g$, where $\mathbf{D}$ is the direction cousin matrix (DCM) defining the orientation of the drone in the scenario. This matrix is available at the

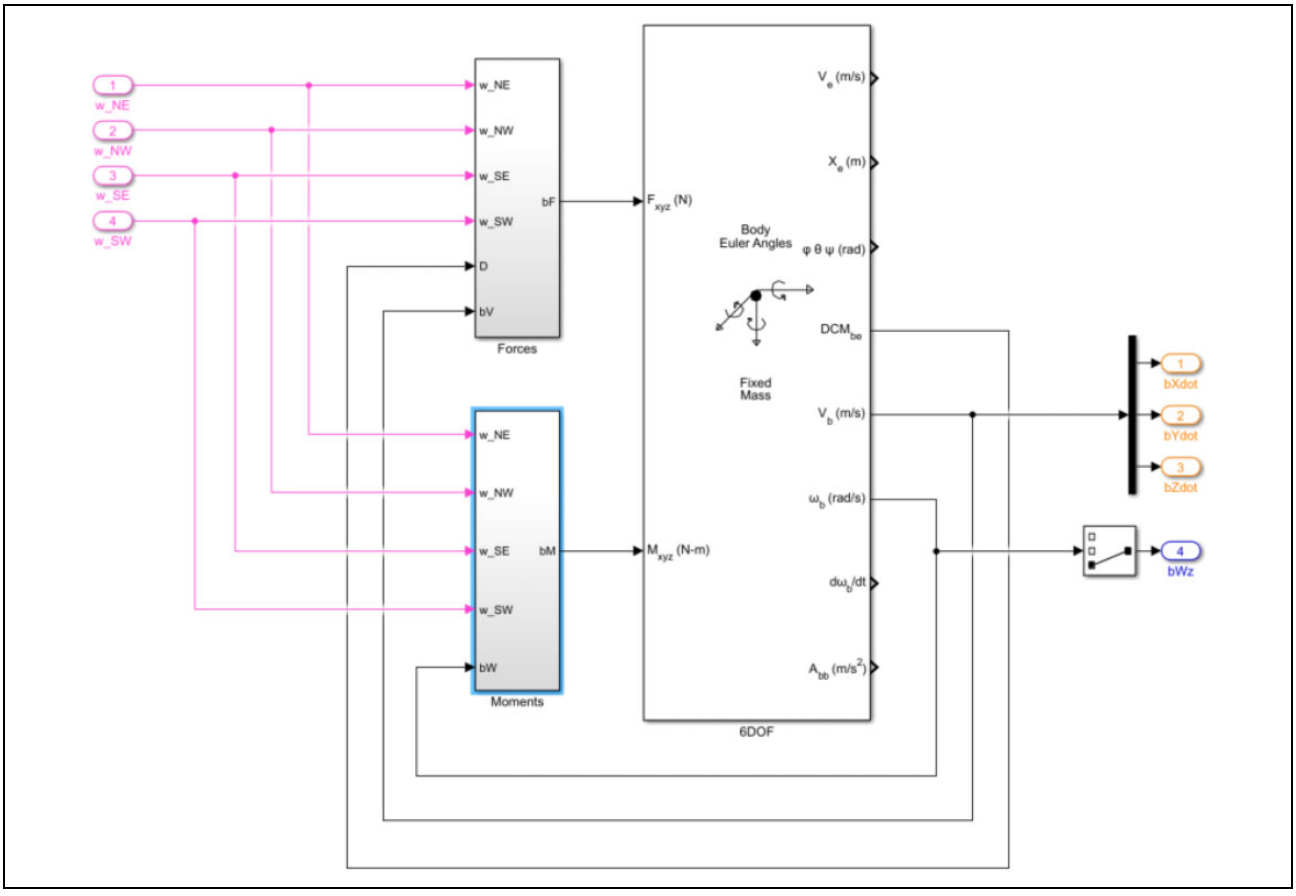
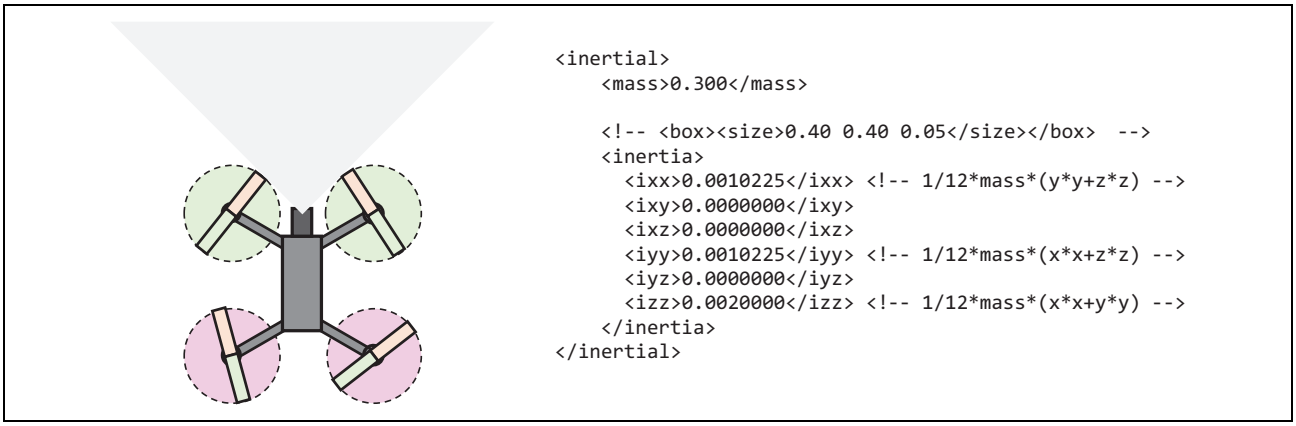


Figure 8. Use of the 6-DOF Simulink block. DOF: degree of freedom.



```
<inertial>
    <mass>0.300</mass>

    <!-- <box><size>0.40 0.40 0.05</size></box>  -->
    <inertia>
      <ixx>0.0010225</ixx> <!-- 1/12*mass*(y*y+z*z) -->
      <ixy>0.0000000</ixy>
      <ixz>0.0000000</ixz>
      <iyy>0.0010225</iyy> <!-- 1/12*mass*(x*x+z*z) -->
      <iyz>0.0000000</iyz>
      <izz>0.0020000</izz> <!-- 1/12*mass*(x*x+y*y) -->
    </inertia>
</inertial>
```

Figure 9. Quadcopter aspect and inertial properties.

DCM_{bo} output of the 6-DOF block (see Figure 8). Figure 11 shows this transformation.

The angular speed of each rotor is $\Omega_i, i \in \{NE, NW, SE, SW\}$, being its maximum value $\Omega_i^{\max} = 15{,}000\,\mathrm{r/min} = 1570.8\,\mathrm{rad/s}$, a typical value in real components.

We define the aerodynamic thrust force T as the one produced (in opposite sense) by the air pushed by a propeller. This force is proportional to the square of the propeller rotation speed Ω, that is, $T = k_{FT}\Omega^2$. Assuming that a rotor can produce a typical maximum force of 1 kg = 9.8 N, we obtain: $k_{FT} = 3.9718\ \mathrm{kg} \times 10^{-6}\mathrm{m}$.

Altogether the four drone rotors will produce a resultant force (${}^b\vec{\mathbf{F}}_T$) in the direction and sense of the drone Z axis

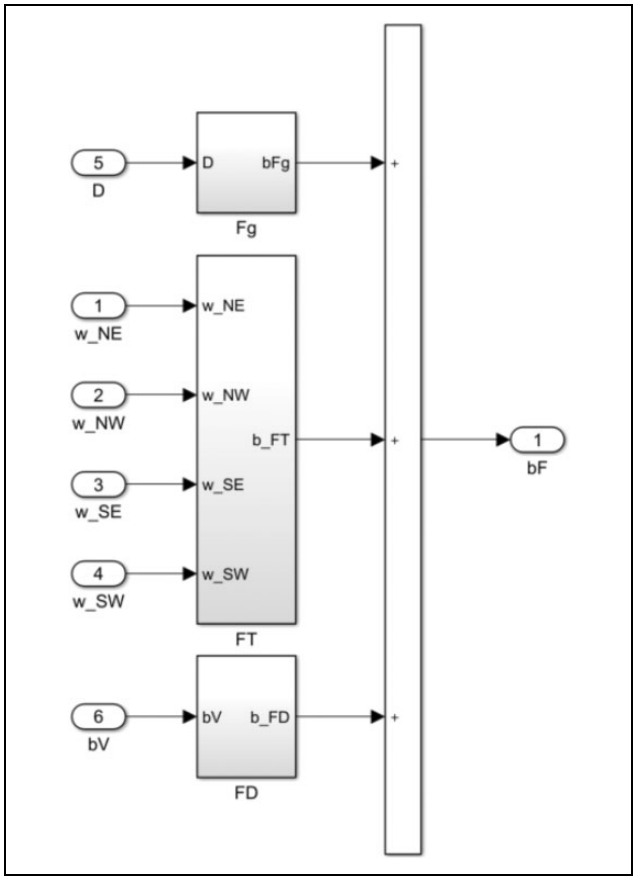


Figure 10. Drone translation movement.

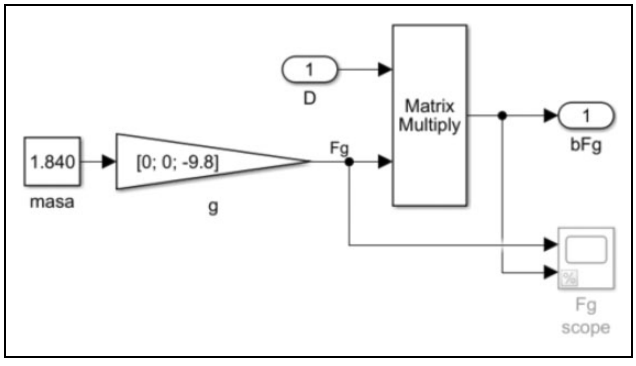


Figure 11. Gravity force in the drone coordinate system.

$$
{}^{b}\vec{\mathbf{F}}_{T} = \begin{bmatrix} 0 \\ 0 \\ \sum\limits_{i}^{\{NE,NW,SE,SW\}} {}^{b}T_{i} \end{bmatrix} = k_{FT} \begin{bmatrix} 0 \\ 0 \\ \Omega_{NE}^{2} + \Omega_{NW}^{2} + \Omega_{SE}^{2} + \Omega_{SW}^{2} \end{bmatrix}
$$

Figure 12 shows the Simulink model implementing this force.

Finally, we define the aerodynamic drag force ($\vec{\mathbf{F}}_{D}$) as the friction with the air produced in opposite sense to the movement of the drone. This force depends on the shape of the object in each axis ($\mathbf{K}_{FD}$) and on the square of the forward speed

$$
{}^{b}\vec{\mathbf{F}}_{D} = -\mathbf{K}_{FD} \begin{bmatrix} \dot{x}|\dot{x}| \\ \dot{y}|\dot{y}| \\ \dot{z}|\dot{z}| \end{bmatrix}, \mathbf{K}_{FD} = \begin{bmatrix} 0.6350 & 0 & 0 \\ 0 & 0.6350 & 0 \\ 0 & 0 & 2.3520 \end{bmatrix}
$$

In our model, the $\mathbf{K}_{FD}$ constant has been set assuming a similar friction in the X and Y axes (although the fuselage is not exactly equal) to obtain a maximum horizontal speed of 20 km/h = 5.55 m/s, and a maximum vertical speed of 3 m/s.

Figure 13 shows the Simulink model implementing this force.

Rotation equations. As said before, the moments contributing to the rotation movement experienced by the drone are given by the expression $\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} + \boldsymbol{\omega} \times \mathbf{I}_{r}\Omega_{r} = \sum \mathbf{M} = {}^{b}\vec{\mathbf{M}}_{T} + {}^{b}\vec{\mathbf{M}}_{DR} + {}^{b}\vec{\mathbf{M}}_{D}$, which has been modeled in Figure 14.

Next, we describe these moments, called aerodynamic thrust (${}^{b}\vec{\mathbf{M}}_{T}$), rotor aerodynamic drag (${}^{b}\vec{\mathbf{M}}_{DR}$), and aerodynamic drag (${}^{b}\vec{\mathbf{M}}_{D}$).

We define the aerodynamic thrust moment (${}^{b}\vec{\mathbf{M}}_{T}$) as the rotation experienced by the drone over the X (roll) and Y (pitch) axes when the four propellers do not generate the same thrust

$$
{}^{b}\vec{\mathbf{M}}_{T} = \begin{bmatrix} l\cos(45) & 0 & 0 \\ 0 & l\sin(45) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} T_{NW} + T_{SW} - T_{NE} - T_{SE} \\ T_{SW} + T_{SE} - T_{NW} - T_{NE} \\ 0 \end{bmatrix}
$$

Regarding the (longitudinal) X axis, rotors producing a positive moment are those located on the left side (NW and SW). On the other hand, regarding the (perpendicular) Y axis, rotors producing a positive moment are those located on the rear side (SW and SE). Since $T = k_{FT}\Omega^{2}$, the previous expression can be represented by the Simulink diagram shown in Figure 15.

The turn of a rotor produces a drone rotation in opposite sense. Rotors can turn at different speeds, according to the sense indicated in Figure 16 left. Collectively, this produces a rotor aerodynamic drag moment (${}^{b}\vec{\mathbf{M}}_{DR}$), defined by the following expression and represented in Simulink as shown in Figure 16 right

$$
{}^{b}\vec{\mathbf{M}}_{DR} = k_{MDR} \begin{bmatrix} 0 \\ 0 \\ \Omega_{NE}^{2} - \Omega_{NW}^{2} - \Omega_{SE}^{2} + \Omega_{SW}^{2} \end{bmatrix},
$$

$$
k_{MDR} = 1.3581 \times 10^{-7}
$$

Finally, we define the aerodynamic drag moment (${}^{b}\vec{\mathbf{M}}_{D}$) as the friction with the air when the drone rotates over itself

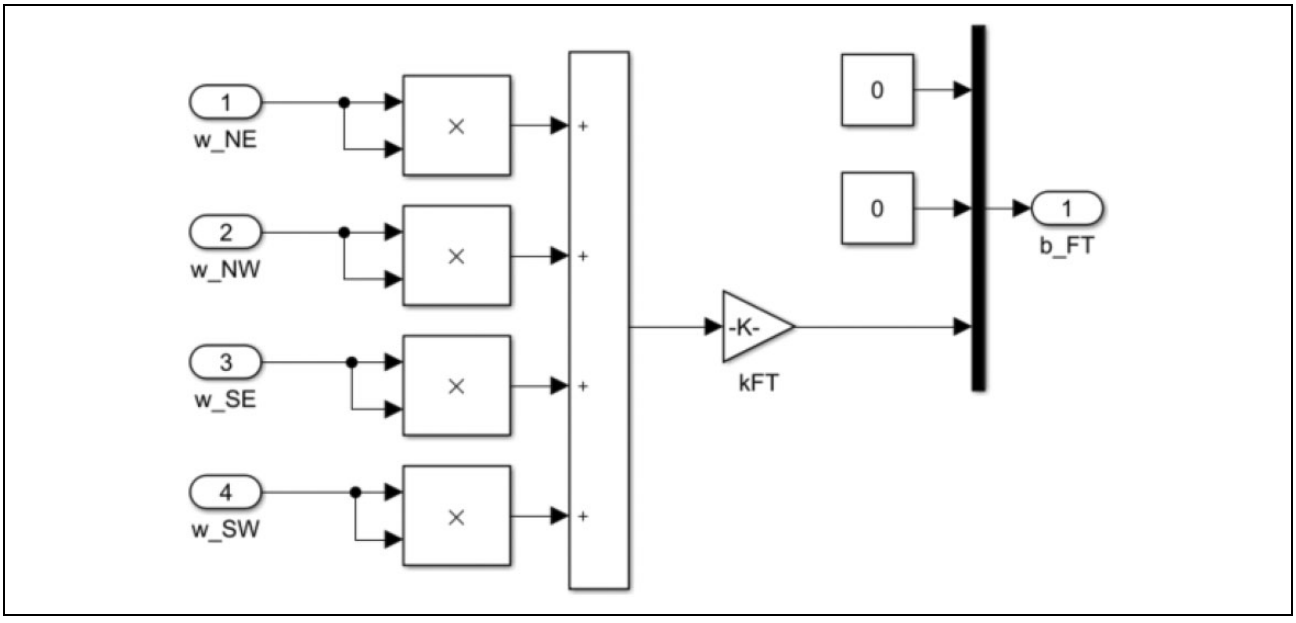


Figure 12. Aerodynamic thrust force.

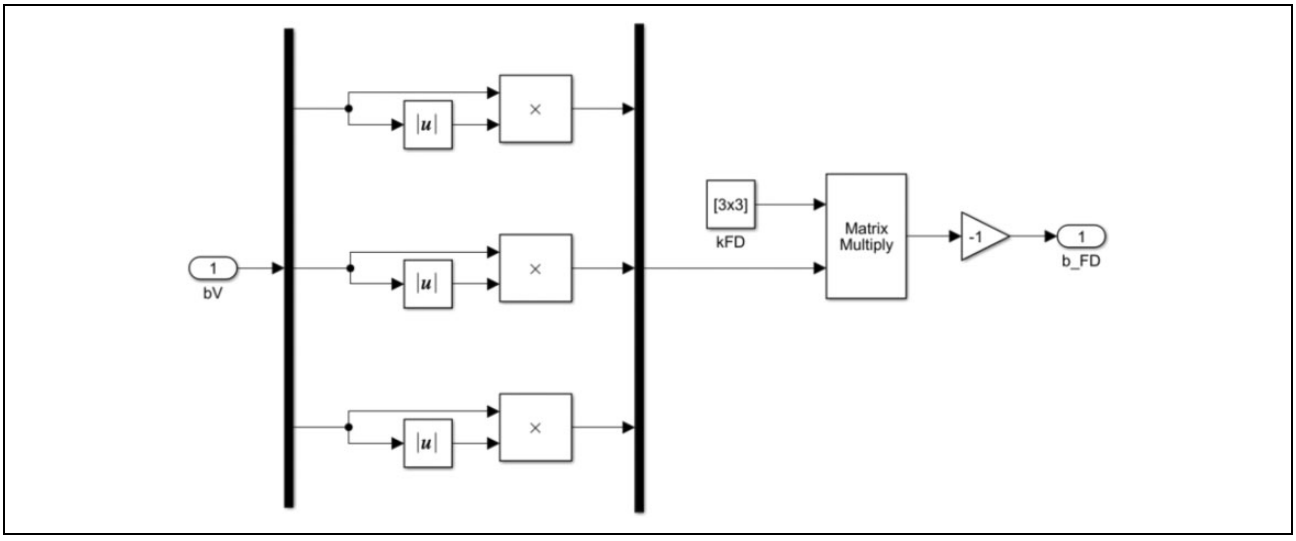


Figure 13. Aerodynamic drag force.

$$
{}^{b}\vec{\mathbf{M}}_D = -\mathbf{K}_{MD}\,{}^{b}\begin{bmatrix} \dot{\phi}|\dot{\phi}| \\ \dot{\theta}|\dot{\theta}| \\ \dot{\psi}|\dot{\psi}| \end{bmatrix},
$$

$$
\mathbf{K}_{MD} = \begin{bmatrix} 0.0621 & 0 & 0 \\ 0 & 0.0621 & 0 \\ 0 & 0 & 0.0039 \end{bmatrix}
$$

Again, we assume similar frictions in the X and Y axes. The $\mathbf{K}_{MD}$ value has been set so that the drone, without gravity, can rotate over itself up to 2 r/min. Figure 17 shows the Simulink model implementing this moment.

Linearization. After modeling the above equations in Simulink, they are linearized using the linear analysis tool. Figure 18 shows the complete system, with the input and the output in open loop. The operating point applied is the drone perfectly balanced, with null linear and angular speeds. The input is similar for all the rotors, so that each one compensates a quarter of the weight of the drone.

After the linearization, we obtain the transformation for the initial system expression, where the values obtained for the state (**A**), input (**B**), and output (**C**) matrices are

$$
\left.\begin{matrix} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \mathbf{y} = \mathbf{g}(\mathbf{x}) \end{matrix}\right\} \rightarrow \left.\begin{matrix} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{C}\mathbf{x} \end{matrix}\right\}
$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g & 0 & 0 & 0 & 0 & 0 & 0 \\ -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -0.2506 & 0.2506 & -0.2506 & -0.2506 \\ -0.2506 & -0.2506 & 0.2506 & -0.2506 \\ 0.0584 & -0.0584 & -0.0584 & 0.0584 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.0114 & 0.0114 & 0.0114 & 0.0114 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

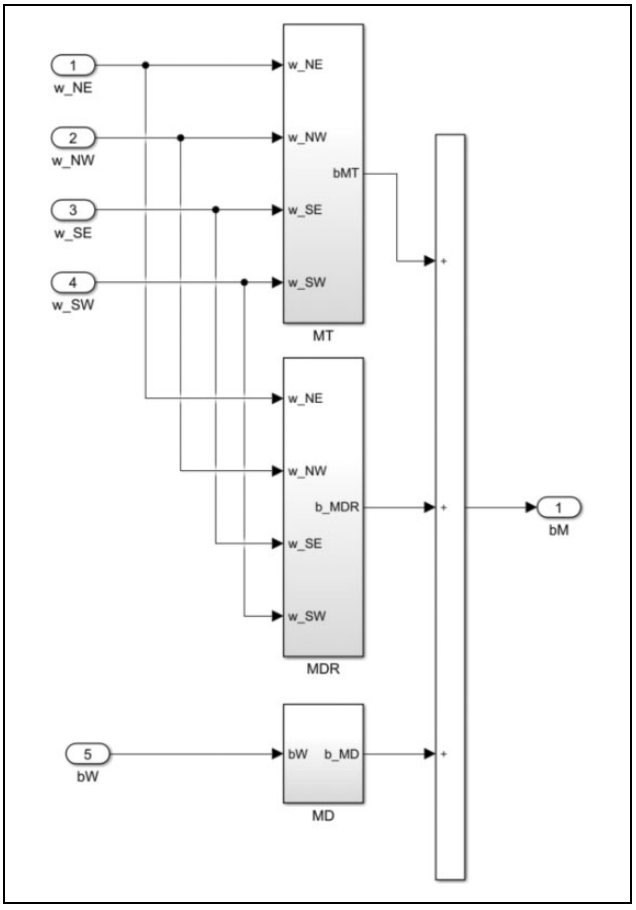


Figure 14. Moments involved in the drone rotation.

Low-level control. We have defined previously the system output as $\mathbf{y} = {}^{b}[\dot{x}\ \dot{y}\ \dot{z}\ \omega_z]^{\mathrm{T}}$, which corresponds to the drone forward speed (in all axes) and rotation speed (in the Z axis). Now, we define a reference to follow ($\mathbf{r}$), as a vector similar to $\mathbf{y}$, so that $\mathbf{y} - \mathbf{r}$ represents the instantaneous error that we are committing ($\dot{\boldsymbol{\xi}}$).

Once we have linearized the nonlinear system modeling of the quadcopter, we apply a controlled feedback of the state and the accumulated error ($\boldsymbol{\xi}$) between the output obtained and the reference to follow, which is defined by the following expressions

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}$$

$$\dot{\boldsymbol{\xi}} = \mathbf{y} - \mathbf{r}$$

$$\mathbf{u} = -\mathbf{K}_{\mathbf{x}}\mathbf{x} - \mathbf{K}_{\mathbf{y}}\boldsymbol{\xi}$$

To compute the control matrices $\mathbf{K}_{\mathbf{x}}$ and $\mathbf{K}_{\mathbf{y}}$, we first represent the previous expressions in matrix notation

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & 0 \\ \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \xi \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ -1 \end{bmatrix} r$$

Then, we define a new system, modeling the follow-up error $\mathbf{e} = \begin{bmatrix} \mathbf{x_e} \\ \xi_{\mathbf{e}} \end{bmatrix}$. The resulting system is

$$\begin{bmatrix} \dot{\mathbf{x}}_{\mathbf{e}} \\ \dot{\xi}_{\mathbf{e}} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{A} & 0 \\ \mathbf{C} & 0 \end{bmatrix} - \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} [\mathbf{K}_{\mathbf{x}} \quad \mathbf{K}_{\mathbf{y}}] \right) \begin{bmatrix} \mathbf{x}_{\mathbf{e}} \\ \xi_{\mathbf{e}} \end{bmatrix}$$

Since the obtained system does not have an input, we can apply the pole placement method. The chosen eigenvalues are $[-6\ -6\ -7\ -7\ -8\ \ -8\ -8\ -8\ -9\ -9\ -9\ -9]$. These values guarantee that the system is stable and does not fluctuate, since they are negative and real numbers. Constants obtained are

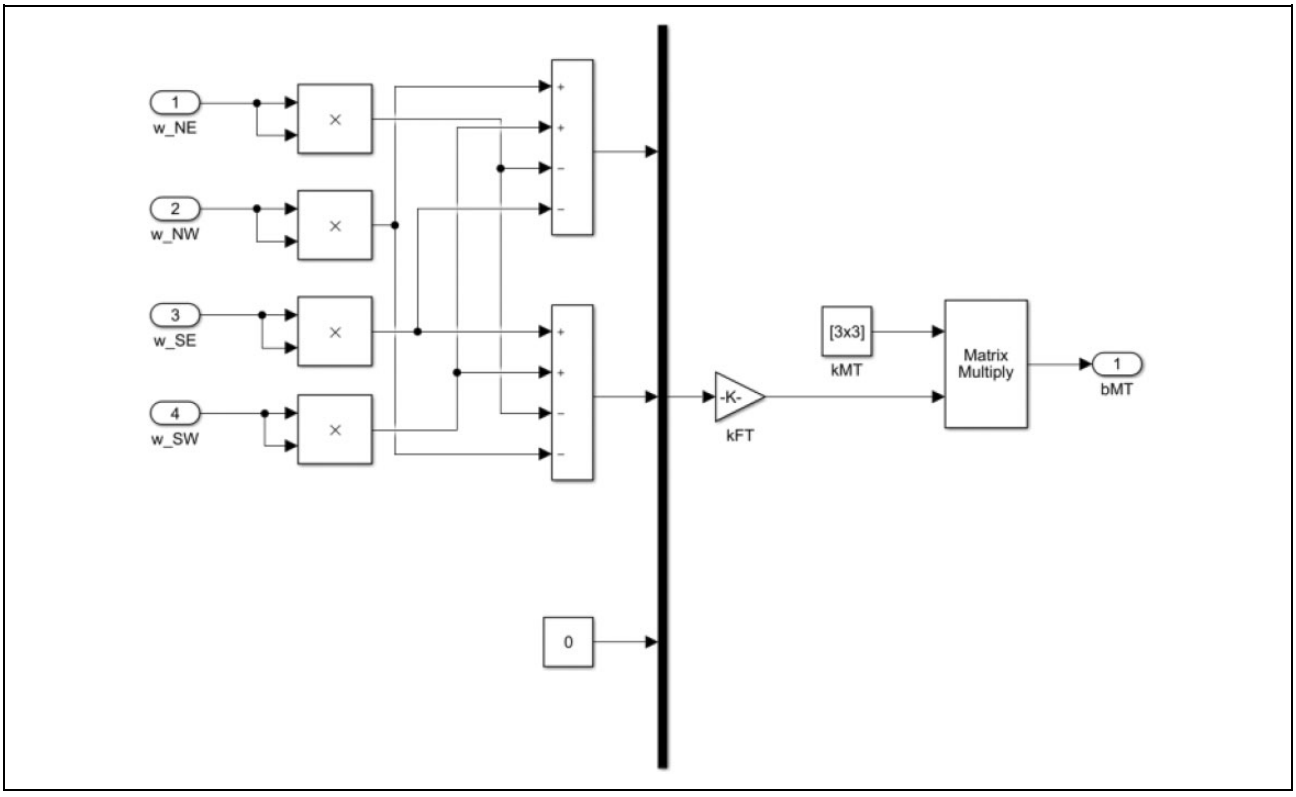


Figure 15. Aerodynamic thrust moment.

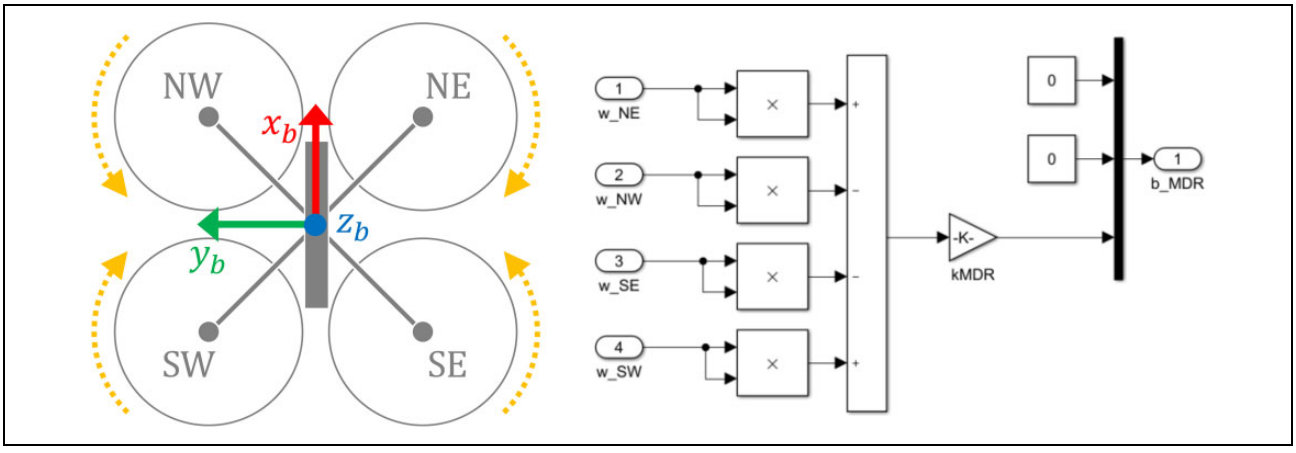


Figure 16. Drone rotors and rotor aerodynamic drag moment.

$$\mathbf{K_x} = \begin{bmatrix} -334.1327 & -334.1327 & -29.9223 & -29.9223 & 72.7456 & -167.9315 & 167.9315 & 334.1327 \\ 0 & 0 & 29.9223 & -29.9223 & -72.7456 & -167.9315 & -167.9315 & 334.1327 \\ 0 & 0 & -29.9223 & 29.9223 & -72.7456 & 167.9315 & 167.9315 & 334.1327 \\ 0 & 0 & 29.9223 & 29.9223 & 72.7456 & 167.9315 & -167.9315 & 334.1327 \end{bmatrix}$$

$$\mathbf{K_y} = \begin{bmatrix} -307.4 & 307.4 & 1580.3 & 308.1 \\ -307.4 & -307.4 & 1580.3 & -308.1 \\ 307.4 & 307.4 & 1580.3 & -308.1 \\ 307.4 & -307.4 & 1580.3 & 308.1 \end{bmatrix}$$

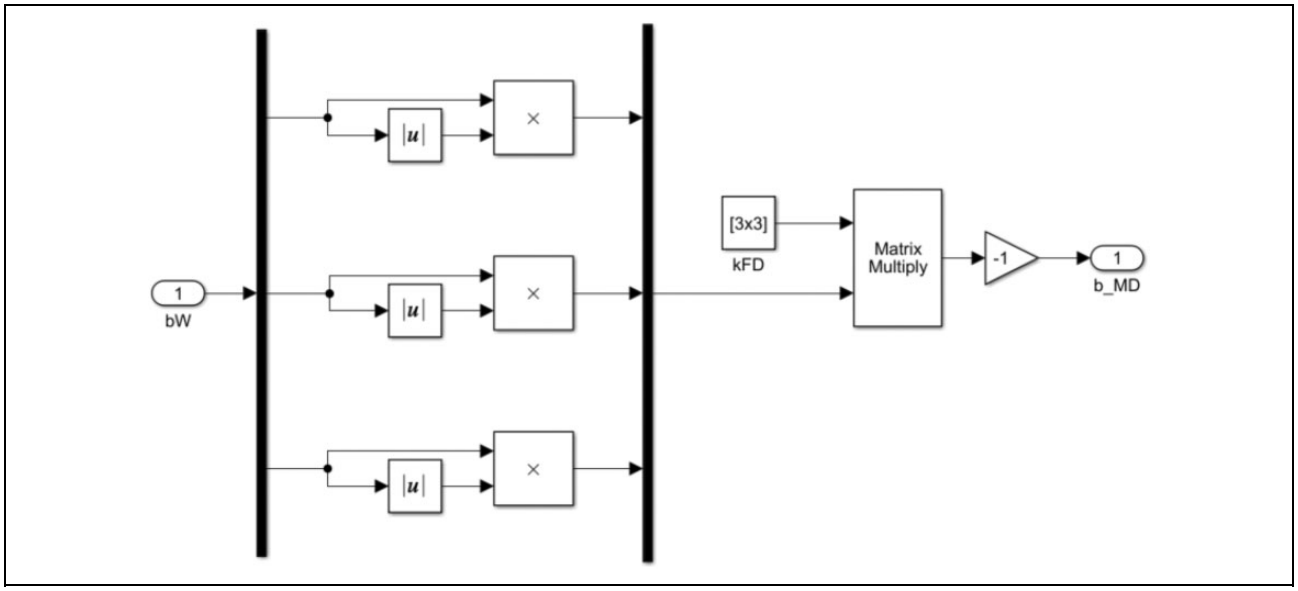


Figure 17. Aerodynamic drag moment.

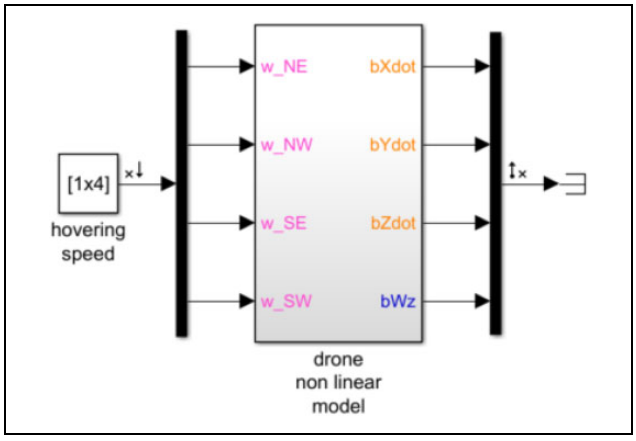


Figure 18. Drone non-linear model.

To conclude this subsection, we will outline the behavior of the ROS/Gazebo plugin developed after solving the model. Each simulation cycle starts by reading the drone X and Y values. Then, considering the proposed reference, the instantaneous error is computed. The result of this computation is used to update the accumulated error. After that, the set of matrix operations previously described are performed, producing the rotation value for all the rotors. As a result, the drone follows the given reference.

Navigation development framework

The navigation development framework (Figure 19, left panel) is based on the popular engineering tool Matlab/Simulink,[27] which can be run over different operating systems (including Microsoft Windows, macOS, and Ubuntu). Students define the drone's behavior by programming the *autopilot* block on the left. As we will see, it can be done in an easy and visual way, by means of a reduced set of commands. Note that it is also possible to control the drone manually using a joystick. The *drone simulator* block on the right employs the Matlab Robotics System Toolbox[32] to connect to Gazebo through several ROS topics (Figure 19 right panel).

Quadcopter odometry. Gazebo continuously publishes on a ROS topic, to which Simulink is subscribed, a vector ${}^{e}[x\ y\ z\ \psi]^{\mathrm{T}}$ with the quadcopter position and orientation (Figure 20 left panel). Internally, these data are managed as a bus called *imu*, composed of four values, referred as *posX*, *poxY*, *posZ*, and *heading*, expressed according to a reference system located in the middle of the simulated scenario. Programmers can read these values and take them into account in their navigation systems. This information is also displayed by means of a Simulink viewer (Figure 20, right panel).

Video image processing. The model also receives the video frames acquired by the drone's built-in camera. This camera presents the following characteristics:

- Refresh rate: 5 Hz
- Resolution: 320 × 240 pixels
- Color range: 24 bits/pixel
- Angle of view: 90°
- Projection plane: 160 cm from the focal point.

All the above means that, if we place a frame on the projection plane, its size in pixels on the image corresponds to its actual size in centimeters.

Each frame is automatically processed, looking for any colored pixel. Figure 21 shows an example of this

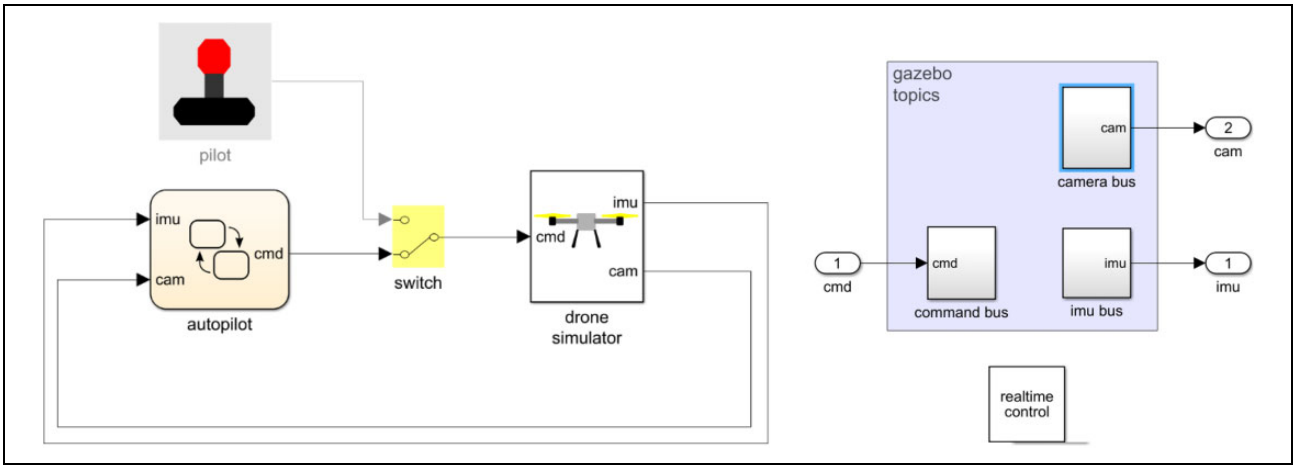


Figure 19. Navigation development framework (high-level view).

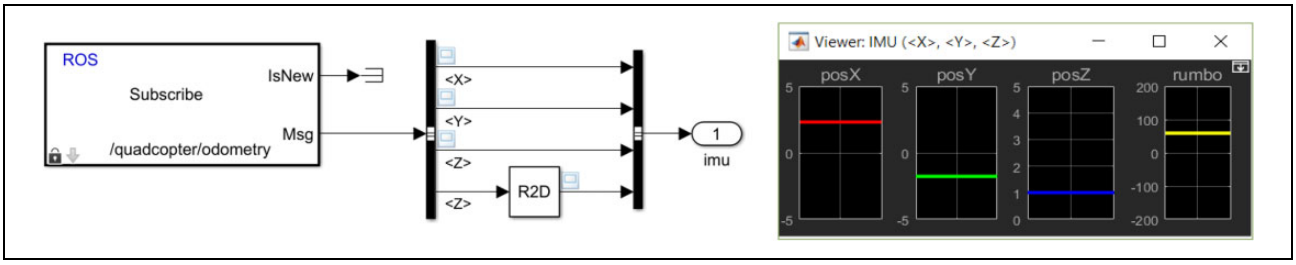


Figure 20. ROS topic publishing quadcopter odometry. ROS: robot operating system.

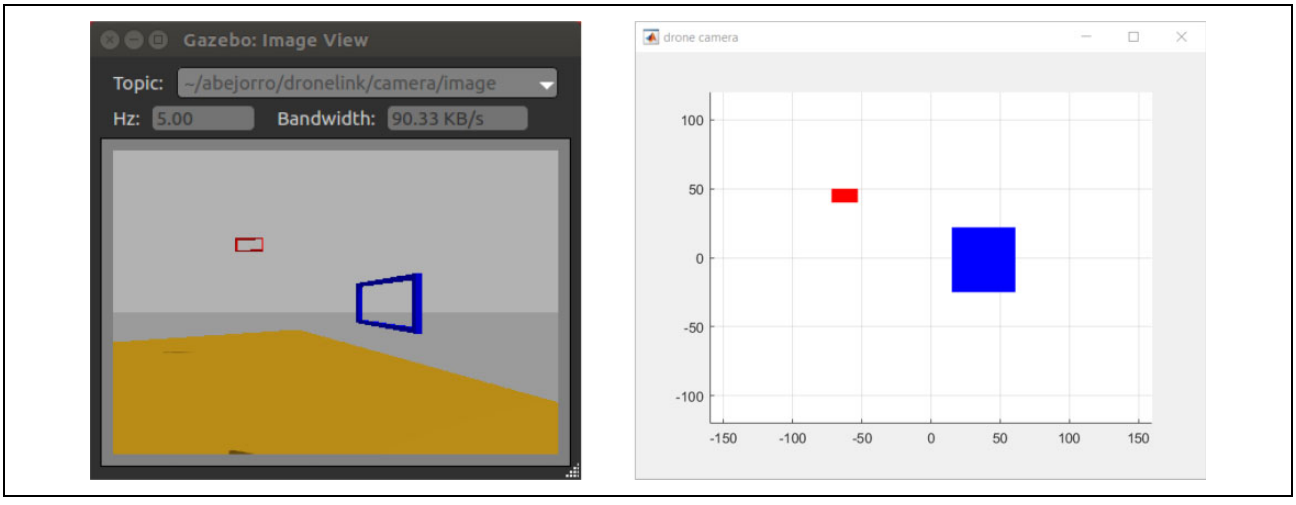


Figure 21. Example of video image processing.

processing. Plot on the right shows the red and blue frames detected in the image shown on the left. The result of this processing is available for the programmers through the *cam* bus. More in detail, this bus is composed of three fields (called, respectively, red, green, and blue) that provide the limits of the image region where the corresponding frame has been located, referred to as N (north), S (south), E (east), and W (west) (see Table 1).

Table 1. Data provided to the programmer in the example of Figure 21.

	N	S	E	W
Red	50	40	−53	−72
Green	−inf	inf	−inf	inf
Blue	22	−25	61	15

N: north; S: south; E: east; W: west.

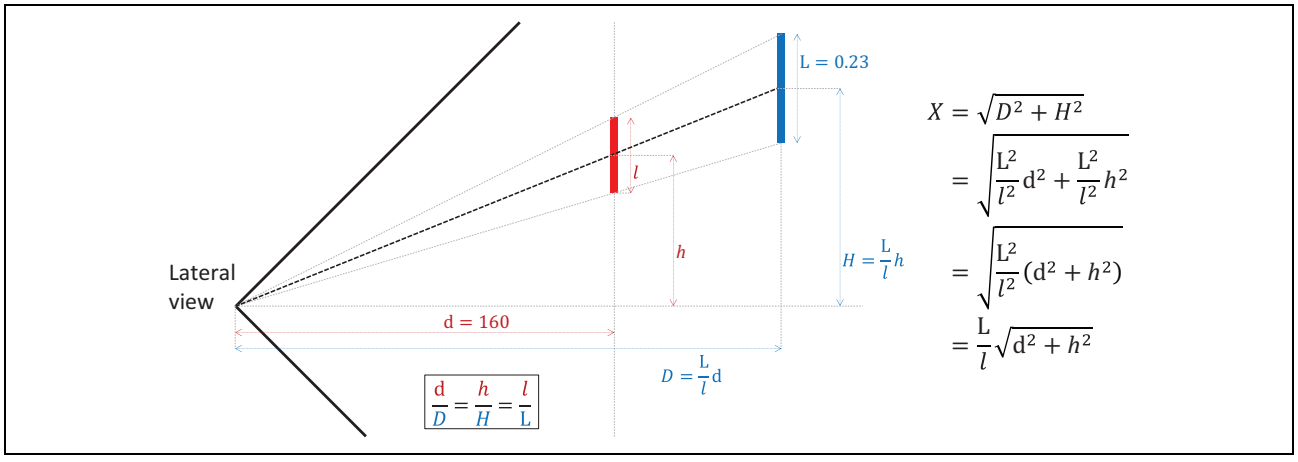


Figure 22. Distance to a frame.

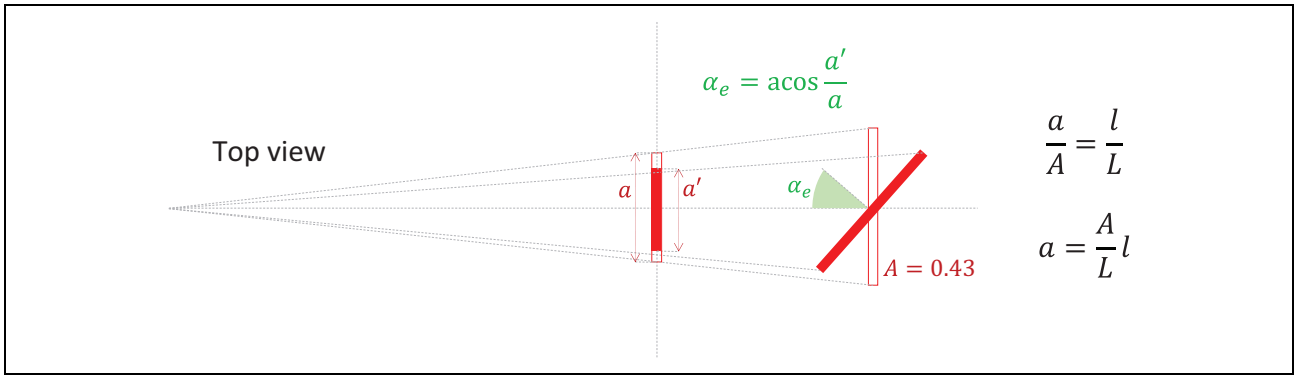


Figure 23. Angle of inclination.

From this information, we can conclude whether a frame is completely displayed on the screen. If so, knowing that the actual height of each frame is 23 cm, we can infer the distance X to it by applying the method shown in Figure 22.

In the same way, once the distance has been computed, and knowing that the actual width of each frame is 43 cm, we can infer the angle of inclination (in absolute value) with which we approach by means of the procedure shown in Figure 23.

The following Matlab code fragment implements these computations:

```
%% Frame position analysis
d2 = 160^2;          % 160^2 pixels
L = 0.23;            % frame height
A = 0.43;            % frame width
if frame.n > -inf && frame.n < 119 && frame.s > -120
    l = frame.n - frame.s;
    h = frame.s + l/2;
    frame.d = L/l * sqrt(d2 + h^2);
    ap = frame.e - frame.w;
    a = max(A/L * l, ap);
    frame.a = acosd(ap/a);
end
```

Quadcopter navigation. Programmers control the movement of the drone by setting five variables composing the *cmd* bus (Figure 24 left). This bus publishes data into a ROS topic to which Gazebo has been previously subscripted. For debugging purposes, this bus is monitored by means of a Simulink viewer (Figure 24 right).

The *on* variable is a binary value used for activating or deactivating the drone engines. If it is set to 0, the drone will drop. Otherwise, it will move according to the rest of control variables. The *velX*, *velY*, and *velZ* variables are real numbers between -1 and $+1$ that provide the drone with different velocities according to the drone reference system (see Figure 6). Finally, the *rotZ* variable is also a real number between -1 and $+1$ that allows to define a horizontal turn (around the Z axis).

Programmers define the value for these variables from a state machine implemented in Stateflow (inside Simulink). The use of this tool is very easy and intuitive. Figure 25 shows two examples. According to the state machine shown on the left, the drone continuously starts the engines, ascends until it reaches an altitude of 2 m, stays in this position for 3 s, descends until it reaches an altitude of 0.7 m, and finally stops the engines. According to the behavior shown on the right, the drone ascends, and then,

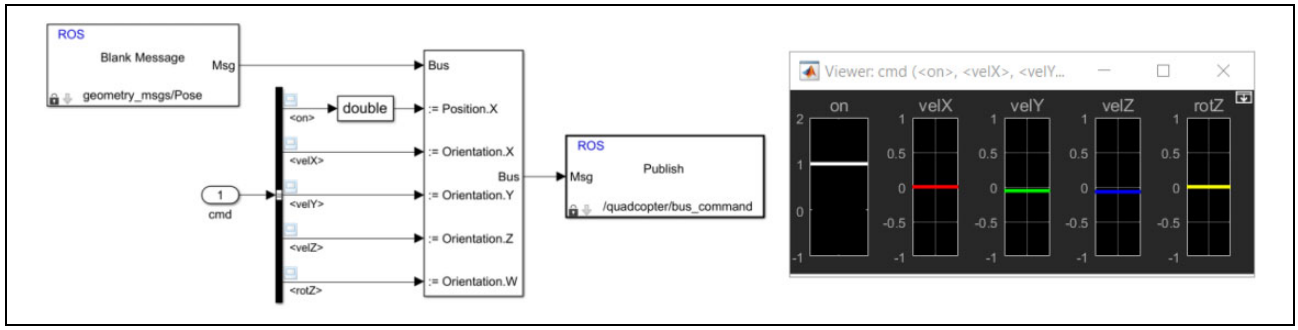


Figure 24. ROS topic publishing quadcopter commands. ROS: robot operating system.

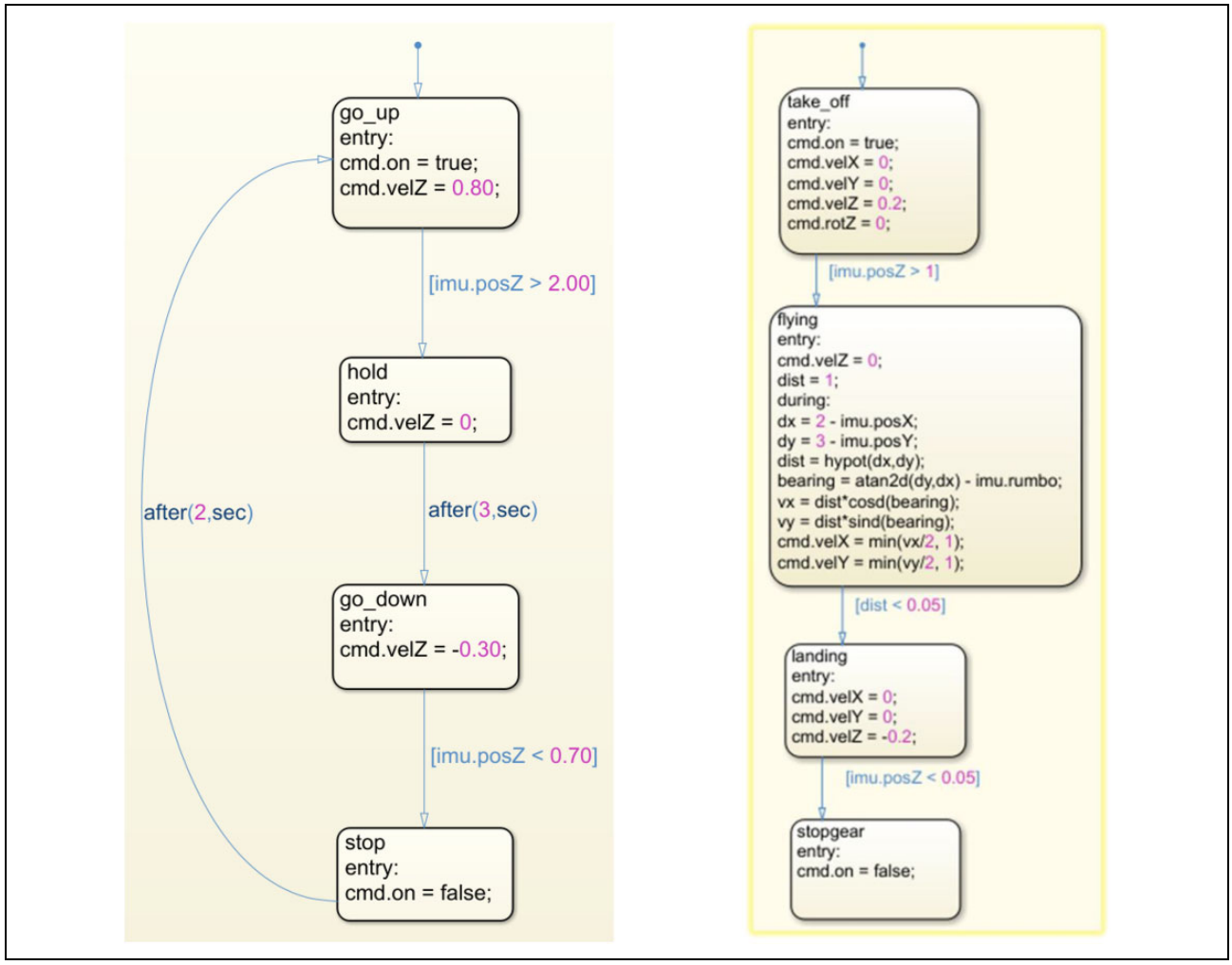


Figure 25. Two examples of drone behavior programmed in stateflow.

maintaining its altitude, it moves to a specific scenario position (defined by $X = 2$ m and $Y = 3$ m). Finally, it lands on the floor.

Survey for participants

After the second edition of the *Drone Challenge* competition, we asked the participants to answer an anonymous and voluntary online survey to collect their opinion about different aspects of the challenge. It was answered by exactly 36 people (19 students and 17 teachers), 34 of whom attended to the final phase of the competition. Therefore, we consider that their opinions can be representative enough.

The survey was composed of two parts. Questions in the first part tried to determine the participant's profile. In

Table 2. Questions related to the participant's profile.

#	Question text	Possible answers
Q1	Indicate whether you are a teacher or a student	Teacher/Student
Q2	If you are a teacher, indicate the level of the students supervised; if you are a student, indicate the level in which are were enrolled	ESO[a]/Bachillerato/ Ciclo Formativo/ Other
Q3	If you are a teacher, indicate whether you teach programming; if you are a student, indicate whether you have received programming classes	Yes/No
Q4	If you are a teacher, indicate whether you teach robotics; if you are a student, indicate whether you have received robotics classes	Yes/No
Q5	If you are a student, indicate whether you are interested in enrolling in a STEM degree	Yes/No
Q6	If you have answered "yes," indicate que name of the degree	(Open answer)

STEM: Science, Technology, Engineering, and Mathematics.
[a]Spanish curriculum at secondary level (high school) considers first a four-year mandatory stage, for students from 12 to 15 years old. This stage is called "Enseñanza Secundaria Obligatoria (ESO)." Then, there is a two-year stage, referred to as "Educación Secundaria Superior," for students between 16 years and 17 years. In this period, students can take the "Bachillerato," oriented to academic degrees, or a "Ciclo Formativo," focused on the labor market.

Table 3. Responses to question Q2 (Educational Level).

	Students			Teachers		
#	*ESO*	*Bachillerato*	*Ciclo Formativo*	*ESO*	*Bachillerato*	*Ciclo Formativo*
Q2	5%	74%	21%	6%	82%	12%

particular, they were asked to indicate their educational level, their background in programming and robotics, and their interest in studying a STEM degree (in case of being a student). Table 2 details the questions composing this part of the survey. As it can be observed in this and the following tables, instead of using a Likert-type scale, most of the questions are *Yes/No* questions. The reason for this decision was that we tried to maintain the concentration of the young students until the end of the survey, composed of 24 questions.

From the responses provided by the participants, we can firstly observe that three quarters of the students come from the "Bachillerato" educational level (see Table 3), that is, they are about to enroll in an academic degree. According to Table 4, we may conclude that less than half of the students are familiar with programming and robotics. On the other hand, most of the teachers have some background in programming, but they are not so familiar with robotics (questions Q3 and Q4). Both findings are not surprising if we analyze the current curriculum at secondary level. Aside from that, nearly all the students expressed their interest in enrolling in an STEM degree (Q5). Most of them were interested in a computing engineering degree.

Table 4. Responses to Questions Related to the Participant's Profile.

	Students		Teachers	
#	*Yes* (%)	*No* (%)	*Yes* (%)	*No* (%)
Q3	47	53	82	18
Q4	42	58	47	53
Q5	95	5	n/a	n/a

Table 5. Questions related to the working environment.

#	Question text	Possible answers
Q7	*The development environment (Matlab/ Simulink) is appropriate*	*Yes/No*
Q8	*I already knew and was familiar with the development environment (Matlab/Simulink)*	*Yes/No*
Q9	*If you have answered "no," indicate whether it has been easy for you to familiarize with Matlab/Simulink*	*Yes/No*
Q10	*The simulation environment (ROS/Gazebo) is appropriate*	*Yes/No*
Q11	*Video tutorials helped me to prepare the entire working environment*	*Yes/No*
Q12	*Video tutorials eased the design of our proposal*	*Yes/No*
Q13	*Indicate (if you wish) those aspects in the working environment that should be improved*	*(Open answer)*

ROS: robot operating system.

Table 6. Responses to questions related to the working environment.

	Students		Teachers	
#	*Yes* (%)	*No* (%)	*Yes* (%)	*No* (%)
Q7	68	32	76	24
Q8	5	95	12	88
Q9	28	72	40	60
Q10	74	26	71	29
Q11	63	37	59	41
Q12	68	32	82	18

The second part of the conducted survey was composed of three blocks. The first block (see Table 5) was focused on polling the opinion of the participants regarding the set of tools composing the working environment and the utility of video tutorials.

According to the responses collected (see Table 6), the general opinion about both the development and simulation environments was positive (Q7 and Q10). However, some

Table 7. Questions related to the evaluation of proposals.

#	Question text	Possible answers
Q14	*I knew the way in which proposals would be evaluated during the final phase of the competition*	*Yes/No*
Q15	*I think that proposals have been evaluated fairly*	*Yes/No*
Q16	*Indicate (if you wish) those aspects in the evaluation procedure that should be improved*	*(Open answer)*

Table 8. Responses to questions related to the evaluation of proposals.

	Students		Teachers	
#	*Yes* (%)	*No* (%)	*Yes* (%)	*No* (%)
Q14	68	32	73	27
Q15	79	21	93	7

Table 9. Questions related to the opinion about the competition.

#	Question text	Possible answers
Q17	*Considering that the competition was addressed to secondary students, the required technical level is appropriate*	*Yes/No*
Q18	*I have received enough information from the organization about the development of the competition*	*Yes/No*
Q19	*The challenge has been stimulating*	*Yes/No*
Q20	*I have enjoyed while we were preparing our proposal*	*Yes/No*
Q21	*If you have participated in the final phase of the challenge, indicate whether you have had fun*	*Yes/No*
Q22	*If you are a student, indicate whether this challenge is motivating to enroll in a STEM degree*	*Yes/No*
Q23	*If you are a student, indicate whether you would like to participate in future editions of this challenge, or in similar competitions*	*Yes/No*
Q24	*Indicate (if you wish) those general aspects of the competition that should be improved*	*(Open answer)*

of the participants complained about the computational power required to run the simulation environment. We believe that the reason is that many of them employed Linux virtual machines for running ROS and Gazebo. Another reason could be the amount of issues when starting Gazebo (in both the real and the physical machine). Finally, most of the participants found video tutorials useful in preparing the working environment and designing their proposals (Q11 and Q12).

Then, the second block (see Table 7) was focused on the opinion of the participants about the way in which their proposals had been assessed. This block of questions was addressed to those teams that reached the final phase of the competition.

Table 10. Responses to questions related to the opinion about the competition.

	Students		Teachers	
#	*Yes* (%)	*No* (%)	*Yes* (%)	*No* (%)
Q17	58	42	53	47
Q18	79	21	82	18
Q19	79	21	88	12
Q20	74	26	82	18
Q21	95	5	80	20
Q22	53	47	n/a	n/a
Q23	84	16	n/a	n/a

From the corresponding responses (see Table 8), we must highlight that a high percentage of both students and teachers believed that their proposals had been evaluated fairly. In addition, most of them knew the assessment procedure.

Finally, the third block (see Table 9) collected the general opinion and feelings about the competition. This block was opened to all the participants in the competition (not only to those attending to the final phase).

The responses obtained in this last block of questions (see Table 10) showed that, although more than a half of the participants considered that the required technical level was appropriate, there was not a complete consensus about it (Q17). Perhaps the reason for this is that at this educational level there is a generalized lack of familiarity with the Matlab/Simulink tool, as we can see in the responses provided to questions Q8 and Q9 (in Table 6). On the other hand, we can see that most of the responses to questions Q19 to Q21 were positive. Therefore, we may conclude that participants enjoyed their participation in the event. Moreover, most of the responses to question Q23 were also positive, which shows that students are willing to participate in future editions of the challenge.

Conclusions and future works

In this work, we have presented a platform for the development of programming and robotics competitions for young students, such as the *Drone Challenge* event described. We consider that this platform is a suitable tool for being used to practice different aspects related to computer programming and aerial robotics at K-12 level. Furthermore, this framework may also be used for supporting robotics subjects in engineering degree programs, since, as we have seen, it allows to work at different abstraction levels.

The results of the survey conducted after the *Drone Challenge* competition indicate that participants had a very

positive opinion about the working platform and the way in which the competition had developed.

As future works, after solving the issues related to the simulation environment (or, alternatively, replacing it by a real drone platform), we plan to use this tool as a starting point for the design of future programming challenges for secondary school students. We also believe that this framework could be useful to develop a research line on autonomous navigation for flying robots.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed the receipt of following financial support for the research, authorship, and/or publication of this article: This work has been partially supported by the TIN2015-66972-C5-2-R (MINECO/FEDER) project.

ORCID iD


Aurelio Bermúdez https://orcid.org/0000-0002-3313-4078

References

1. He S, Maldonado J, Uquillas A, et al. Teaching K-12 students robotics programming in collaboration with the robotics club. In: *IEEE integrated STEM education conference*, Princeton, NJ, USA, 8 March 2014.
2. Bower T. Teaching introductory robotics programming. *IEEE Robot Autom Mag* 2016; 23(2): 67–73.
3. Ilori O and Watchorn A. Inspiring next generation of engineers through service-learning robotics outreach and mentorship programme. *Int J Adv Robot Syst* 2016; 13: 1–7.
4. Eguchi A and Uribe L. Robotics to promote STEM learning: educational robotics unit for 4th grade science. In: *IEEE Integrated STEM Conference (ISEC)*, Princeton, NJ, USA, 11 March 2017.
5. West J, Vadiee N, McMahon A, et al. From classroom Arduinos to missions on mars: making STEM education accessible and effective through remotely operated robotics. In: *IEEE Integrated STEM Conference (ISEC)*, Princeton, NJ, USA, 11 March 2017.
6. Eguchi A. RoboCupJunior for promoting STEM education, 21st century skills, and technological advancement through robotics competition. *Robot Auton Syst* 2016; 75: 692–699.
7. Rursch JA, Luse A, and Jacobson D. IT-adventures: a program to Spark IT interest in high school students using inquiry-based learning with cyber defense, game design, and robotics. *IEEE Trans Educ* 2010; 53(1): 71–79.
8. Nascimento F, Seidel I, and Faria CR. Junior soccer simulation: providing all primary and secondary students access to educational robotics. In: *Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics*, Uberlandia, Brazil, 29–31 October 2015.
9. Liu J, Feenstra W, Hunt M, et al. Students touch space in zero robotics programming competition with free downloadable curriculum. In: *IEEE Aerospace Conference*, Big Sky, Montana, USA, 1–8 March 2014.
10. Paliokas I, Arapidis C, and Mpimpitsos M. Game based early programming education: the more you play, the more you learn. In: Pan Z., et al. (ed.), *Transactions on Edutainment IX. LNCS 7544*. Berlin; Heidelberg: Springer-Verlag, 2013, pp. 115–131.
11. Shim J, Kwon D, and Lee W. The effects of a robot game environment on computer programming education for elementary school students. *IEEE Trans Educ* 2017; 60(2): 164–172.
12. Wing JM. Computational thinking. *Commun ACM* 2006; 49(3): 33–35.
13. Heintz F, Mannila L, and Färnqvist T. A review of models for introducing computational thinking, computer science and computing in K-12 education. In: *Frontiers in Education (FIE)*, 2016, pp. 1–9. IEEE.
14. MIT Media Lab. *Scratch*. https://scratch.mit.edu (accessed 28 February 2018).
15. Google, Inc. *Blockly*. https://developers.google.com/blockly (accessed 28 February 2018).
16. Kalelioglu F. A new way of teaching programming skills to K-12 students: Code.org. *Comput Hum Behav* 2015; 52: 200–210.
17. Turchi T and Malizia A. Fostering computational thinking skills with a tangible blocks programming environment. In: *IEEE symposium on visual languages and human-centric computing (VL/HCC)*, Cambridge, UK, 4–8 September 2016.
18. Kordaki M and Kakavas P. Digital storytelling as an effective framework for the development of computational thinking skills. In: *EDULEARN17 conference*, Barcelona, Spain, 3–5 July 2017.
19. Enríquez C, Aguilar O, and Domínguez F. Using robot to motivate computational thinking in high school students. *IEEE Lat Am Trans* 2016; 14(11): 4620–4625.
20. Witherspoon EB, Higashi RM, Schunn CD, et al. Developing computational thinking through a virtual robotics programming curriculum. *ACM Trans Comput Educ* 2017; 18(1): 1–20.
21. Makeblock Co., Ltd. *Makeblock*. http://www.makeblock.com (accessed 20 May 2018).
22. Yadagiri RG, Krishnamoorthy S, and Kapila V. A blocks-based visual environment to teach robot-programming to K-12 students. In: *Proceedings of the American society for engineering education (ASEE)*, Seattle, Washington, 14–17 June 2015.
23. García-Peñalvo FJ and Mendes AJ. Exploring the computational thinking effects in pre-university education. *Comput Hum Behav* 2018; 80: 407–411.
24. Faculty of Computer Science Engineering (University of Castilla-La Mancha). http://esiiab.uclm.es (accessed 28 February 2018).
25. Valavanis K and Vachtsevanos GJ (eds.). *Handbook of Unmanned Aerial Vehicles*. Berlin: Springer, 2015.
26. Drone Challenge. http://blog.uclm.es/esiidronechallenge (in Spanish) (accessed 28 February 2018).

27. The MathWorks, Inc. *Matlab*. https://www.mathworks.com/products/matlab.html (accessed 28 February 2018).
28. Open Source Robotics Foundation. *Robot Operating System (ROS)*. http://www.ros.org (accessed 28 February 2018).
29. Open Source Robotics Foundation. *Gazebo*. http://gazebosim.org (accessed 28 February 2018).
30. Ogata K. *Modern Control Engineering*, 5th ed. Upper Saddle River: Prentice Hall, 2010.
31. Open Source Robotics Foundation. *SDF*. http://sdformat.org/ (accessed 20 May 2018).
32. The MathWorks, Inc. *Robotics System Toolbox*. https://www.mathworks.com/products/robotics.html (accessed 28 February 2018).