SETCMAS: An easy-to-use software stack to facilitate Simulations and Experiments in Teaching Control of Multi-Agent Systems

Sylvain Bertrand

SETCMAS: An easy-to-use software stack to facilitate Simulations and Experiments in Teaching Control of Multi-Agent Systems*

Sylvain Bertrand[1]

Abstract— **This paper presents a software stack named `setcmas`, for Simulation and Experiments in Teaching Control of Multi-Agent Systems. It aims at facilitating simulations and experiments for teachers and students involved in Control Engineering curricula. The stack is available online and can be used freely for Control Education purposes. It leverages the advantages of ROS, the robot operating system, without requiring any knowledge of it to be used. Control algorithms for multi-agent systems can be implemented in simple Python scripts, and then simulated and validated through experiments with ground mobile robots. The paper presents the content of the stack, the possible architecture used for experiments, and some usages already made for teaching, demonstration, and research work.**

I. INTRODUCTION

Many curricula in control engineering now include lectures or modules devoted to multi-agent systems, and more specifically to control architectures and algorithms for multi-agent systems. Many industrial applications have indeed motivated this development of academic and education activities related to modeling and control of network systems, distributed control or state estimation, etc. These applications are for example smart grids, autonomous vehicles, sensor networks, or other applications related to IoT.

Control of multi-agent systems is sometimes taught from the automatic control point of view, as it is the case in Control Engineering curricula. In that case, the effort is often put on theory and analysis, fundamental algorithms such as consensus, etc. with applications in simulations. Sometimes it is taught from the robotics point of view, when related to curricula on robotics or autonomous vehicles. In that case, the emphasize is put on algorithms such as platooning, formation control, etc. with applications in experiments (mobile ground robots [1], scale model autonomous cars [2], drones [3], etc.).

This paper focuses on the context of teaching control of multi-agent systems in Control Engineering curricula, where robotic applications can nevertheless be considered to motivate students, but still by considering the algorithms from the control theory point of view.

In control engineering curricula, teachers and students are not always familiar with simulators and experimental platforms from the robotics community. Sometimes they find too much time consuming to invest in the understanding, use, development of experiments to be integrated as practical work sessions.

It is of course always possible to rely on existing simulators which are widely used and well documented, eg. Webots, Gazebo, MORSE, ARGoS, etc. (see [4], [5] for main features and comparison). They are mainly developed by and for the robotics community and are designed to be representative of the physics of the robots and emulate their sensors. Simulating a multi-robot system for testing control algorithms of multi-agent systems may not always be easy or could require computers with large computational capabilities when the number of simulated robots increases. Although very efficient for research, they can be more difficult to use for control education purposes, especially on students' laptops where specific configuration requirements and installation procedures should remain limited.

Development of simulators more dedicated to education purposes has been investigated, see eg. [6], [7]. But interface to real hardware for experiments is sometimes not accounted for, or very specific platforms (eg. developed by the authors) are considered making it not easy to replicate [8].

Frameworks such as ROS[1], the Robot Operating System, have enabled to decrease the difficulty of moving from simulations to experiments on real hardware ("sim-to-real"). It has motivated the development of software stacks to be used with ROS for education in robotics or autonomous vehicle [9], [10]. A very interesting recent work considered the development of a software stack for multi-aerial vehicles [11]. Although these works provide very valuable material, they can be found difficult to be used by teachers or students from Control Engineering curricula, that may not be familiar with ROS.

The main objective and contribution of this paper is to provide to the Control Engineering education community an easy-to-use framework to facilitate Simulations and Experiments in Teaching Control of Multi-Agent Systems (`setcmas`). It leverages the advantages of ROS but without requiring any knowledge on it by teachers or students. The stack offers the possibility to implement control algorithms in simple Python scripts, just by implementing the "mathematical part", and perform simulations. The proposed simulator is lightweight, allowing to be run for multi-robot systems on classical computers and on a Virtual Machine, which is a solution often well appreciated because not requiring any intrusive installation on students' computers. The proposed software stack also enables to run experiments

*Part of this work has been supported by French grant ANR Delicio (ANR-19-CE23-0006)

[1]Sylvain Bertrand is with Université Paris-Saclay, ONERA, Traitement de l'information et systèmes, 91123, Palaiseau, France `sylvain.bertrand@onera.fr`

[1]`https://www.ros.org/`

with real ground mobile robots. The robots considered in the experiment architecture described in this paper are the Turtlebot3[2]. This choice is motivated by the fact that this type of platform (or previous versions) is often available in many laboratories or universities, and that it is an open source and well-documented platform, which is helpful for people not familiar with robotics and ROS. Note that use of other ground mobile robots would be possible. The stack also provides different examples of control algorithms for multi-agent systems such as distributed consensus, formation tracking or, at a more advanced level, event-triggered distributed consensus.

The `setcmas` stack presented throughout this paper is available online at `https://github.com/bertrandsylv/setcmas` and can be used freely for Control Education purposes.

The paper is organized as follows. The next section presents in details the content of the stack and describes its packages related to simulation, control algorithms, experimentation and data visualization. Examples of usages already made of the stack are then presented, illustrating possibilities offered for teaching, demonstrations and research work.

II. CONTENT OF THE STACK

The `setcmas` software stack has been developed in Python, building upon ROS (ROS1 Noetic distribution). It is composed of three packages which correspond to the simulator, the control algorithms, and the experiments. The control algorithms can be run either in simulation, or using real robots during experiments. In addition, some scripts are provided to visualize and process data recorded during simulations or experiments.

As previously mentioned, no knowledge in ROS is required to work with these package, develop and test distributed control algorithms in simulations or experiments. Only some basic knowledge in Linux/ROS are required for installation of the experimental setup, but the documentation of the Turtlebot3 platforms offer a very comprehensive guide that can be followed by anyone. As explained later in Section III-B, a Virtual Machine provided by the author can be used for working in simulation, hence requiring no specific knowledge for the user.

The next sections describe the simulation package (`setcmas_simu`), the package implementing distributed control algorithms (`setcmas_ctrl`), and the one for experiments (`setcmas_expe`). Some indications are finally provided on how to record, process and visualize data.

A. Simulation

The `setcmas_simu` package allows for simulation of one or several TurtleBot3 ground mobile robots. Contrary to existing simulation packages available for Turtlebot3 that make use of realistic simulators such as Gazebo, accounting for collisions and sensors, the choice has been made here to develop a very lightweight simulation code. Written in Python it simply implements unicycle dynamics for the robot. No sensor are emulated, nor physics such as inertia, friction or collisions. The objective is to keep the code as simple as possible, so that it can easily be used and modified by students. In addition, it can be run without requiring a powerful computer. Use in Virtual Machine is also made possible (see Section III-B), without requiring Linux installation on teachers' and students' PCs. 3D visualization of the robots and of their trajectories is made possible (but not mandatory) using RViz[3]. A color code for the robot and its trajectory is automatically assigned from its number (0: red, 1:green, 2: blue, 4: yellow, etc.). This color code is also used in visualization of recorded data (see Section II-D). The code enables to work with a number of robots ranging from 1 to 8, but this maximum number can be easily changed by modifying the code.

The syntax to launch a mono-robot simulation is: "`roslaunch setcmas_simu simu_mono_robot.launch`" . For a multi-robot simulation with 8 robots: "`roslaunch setcmas_simu simu_multi_robots.launch nb_robots:=8`" . The number of robots is specified as argument. Therefore only one line of command is required from the teacher(s)/student(s) to launch a simulation, without any specific knowledge of ROS.

An example of 3D visualization of a simulation with 8 robots is given in Figure 1, with the color code used for each robot and its trajectory. Recording and exploiting simulation data is made possible using one very simple command line based on the `rosbag`[4] tool from ROS and scripts provided with the simulation package `setcmas_simu` (see Section II-D).

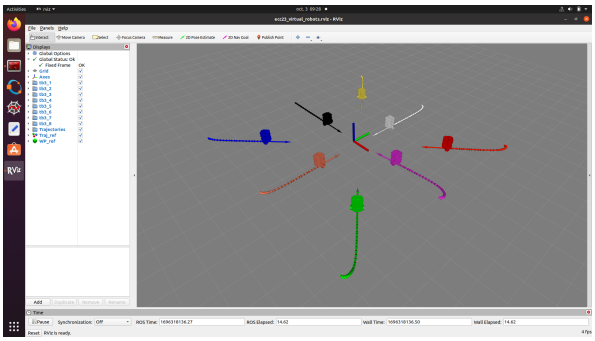

Fig. 1. Visualization in RViz of a simulation of distributed event-triggered consensus with 8 robots.

B. Control algorithms

1) Controller structure: In simulation or experiment, a control law will be evaluated by each robot. The package `setcmas_ctrl` provides scripts written in Python to define the structure of the controller and the control algorithms.

Control inputs of the Turtlebot3 robots, and of their unicycle dynamical model used for simulation, are the linear speed V and angular velocity ω. The objective of this work is to

[2] `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`

[3] `http://wiki.ros.org/rviz`

[4] `http://wiki.ros.org/rosbag`

facilitate application of control algorithms for multi-agent systems. Therefore, it has been chosen to allow for usage of alternative control inputs, in terms of Cartesian velocity vector $v = [v_x\, v_y]^T$. A 2D single integrator dynamics can therefore be considered for control design, which is the type of dynamical model that is very often used when introducing distributed control algorithms for multi-agent systems to students. A Python script (`si_to_uni.py`) is used to convert v into the robot control inputs, by classically assigning V from the module of v, computing a desired orientation from the direction of v and finally using a proportional control law to compute ω from the error between the actual orientation of the robot and the desired one. This script can be modified by teachers and students if needed. It acts as a "low level" controller enabling to implement distributed control algorithms at a higher "guidance" level, by considering control laws for multi-agent systems with single integrator dynamics. In terms of pedagogy, it also enables to introduce to students different concepts usually found in Control Engineering, such as control design considering simplifying models/assumptions, verification/analysis considering the original system (i.e. without simplifying assumptions), guidance and control, inner and outer control loops.

Control algorithms can then be implemented on dedicated scripts. These scripts written in Python only contain mathematical expressions, no code related to ROS. They are the main "interface" proposed to teachers and students. A script consists in a single function which follows the following template:

```
def control(robotNo, nbRobots, poses, t):
    # ...
    # ... control algorithm
    # ...
    return vx, vy
```

The name of the script can be chosen by the user and will be used to run the algorithm. An instance of the script will be run in a distributed way for each robot, to compute its control inputs v_x, v_y, which will then be converted to V and ω. Different arguments of the function can be used, if required, in the algorithm to be implemented by the `control` function:

- `robotNo`: index of the robot which is evaluating its control input.
- `nbRobots`: total number of robots in the multi-agent system.
- `poses`: array with poses information (i.e. positions and orientations) of all the robots. Note that knowledge of the poses of all the robots relates to a centralized strategy. But distributed strategies can be realized by considering in this array only the components corresponding to neighbor agents, that is the information locally available to the current agent (`robotNo`) computing its control input. That can eg. be realized by defining a communication graph between the agents by specifying its adjacency or Laplacian matrix and making use of it in the control algorithm.
- `t`: current time instant.

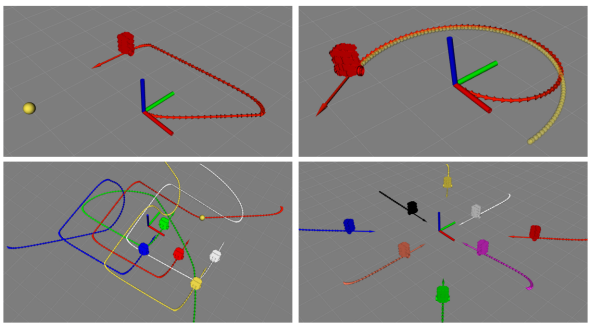

Fig. 2. Examples of simulations with different control algorithms: WP navigation and trajectory tracking for a single robot (respectively top-left and top-right), leader-follower formation and consensus for multi-robots (respectively bottom-left and bottom-right).

As mentioned before, only maths are to be implemented in such script which can be run either in simulation or in experiment with real robots in a totally transparent manner, by teacher(s) and student(s), without knowledge in ROS nor robotic platforms. The name of the script is to be chosen by the user, so that different algorithms can be implemented and easily tested. The name of the script is used as argument of the single command line that has to be run to launch the control algorithm.

2) Examples for single and multi-agent control: For example, for a single robot case, the command line "`roslaunch setcmas_ctrl ctrl_mono_robot.launch algo:=nav_wp`" will run a script named `nav_wp.py` implementing WayPoint navigation control for a single robot. The command line "`roslaunch setcmas_ctrl ctrl_mono_robot.launch algo:=traj_tracking`" will run a script named `traj_tracking.py` implementing trajectory tracking control for a single robot. Visualization of simulations running these two algorithms, provided as examples in the `setcmas_ctrl` package, are presented in the top line of Figure 2. The actual pose, 3D model and past trajectory of the robot are represented in red. The yellow dot and yellow trajectory are visualization markers that can be defined directly, by the user, in an easy way and with pure Python syntax, using ad-hoc functions proposed in the `setcmas_ctrl` package.

Two other examples of simulations involving distributed control algorithms for multi-agent systems are presented in the bottom line of Figure 2, respectively implementing leader-follower formation control with 5 agents (bottom-left) and consensus with 8 agents (bottom-right). In addition to the command line running the simulator with the ad-hoc number of robots (see Section II-A), the two following command lines have respectively been used: "`roslaunch setcmas_ctrl ctrl_multi_robots.launch nb_robots:=5 algo:=leader_follower_formation`" and "`roslaunch setcmas_ctrl ctrl_multi_robots.launch nb_robots:=8 algo:=consensus`" . They refer to the Python scripts `leader_follower_formation.py` and `consensus.py` also provided as examples in the `setcmas_ctrl` package.

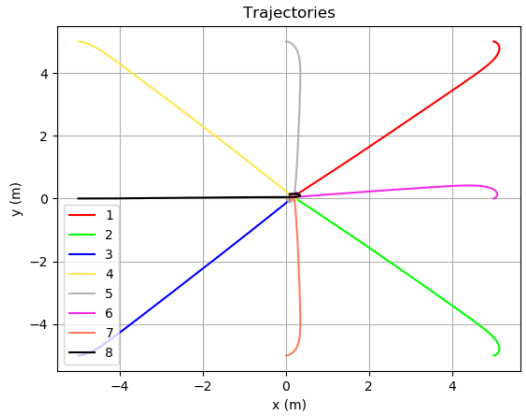

Fig. 3. Event-triggered consensus: trajectories of the robots.

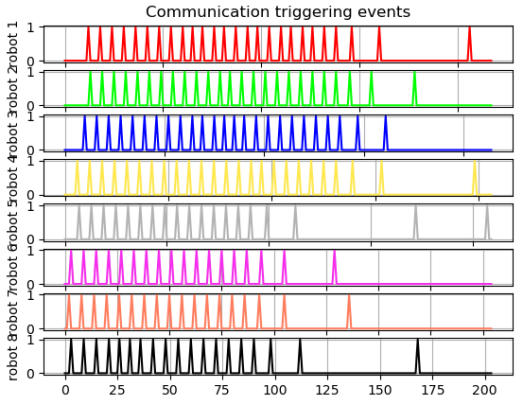

Fig. 4. Event-triggered consensus: communication instants of each robot (0: no communication, 1: communication).

3) Moving to more advanced multi-agent control algorithms: The `setcmas_ctrl` package also enables to work with a class of more advanced control strategies that accounts for communication exchanges between the agents of the network. Event-triggered strategies [12] enable to implement communication decisions and reduce the number of communications compared to periodic communications. They are widely used in control of multi-agent systems and more specifically in distributed schemes, where each agent decides when to broadcast a communication to its neighbors by evaluating a Communication Triggering Condition (CTC), based on information locally available to that agent.
In addition to a script defining the control law of the agent, another Python script (`ctc.py`) can be completed by teacher(s)/student(s) to implement the CTC to be evaluated by each agent. When the condition is verified, the agent communicates its current state information. When not, last received information is used instead (zero-hold) by the control algorithm. An example of CTC implementation, provided in the `setcmas_ctrl` package, is:

```
def ctc(pos, last_pos, t):
    if (numpy.linalg.norm(pos-last_pos)>0.25):
        return True
    else:
        return False
```

A communication is triggered by the agent that evaluates its condition whenever the norm of the difference between its current position and its last transmitted one becomes greater than a given threshold. Note that this is a simple and naive condition, provided here as example, but which can be provided as a starting point to students, to compare with periodic communications and before implementing more efficient CTCs [12].
An example of distributed event-triggered consensus is provided in simulation for 8 agents. Trajectories of the robots, time evolution of their position components and communication instants are presented respectively in Figures 3 and 4. These figures have been obtained as explained later in Section II-D.

C. Experiments

Instead of simulation, experiments with real robots can be launched using the `setcmas_expe` package. Similarly to what is done in simulation, only one command line is required to launch an experiment. The control algorithm is also launched by the same command. For example "`roslaunch setcmas_expe expe_ctrl_mono_robot.launch algo:=nav_wp`" for an experiment with one robot running the Way Point navigation control algorithm, or "`roslaunch setcmas_expe expe_ctrl_multi_robots.launch nb_robots:=8 algo:=consensus`" for an experiment with 8 robots running the distributed consensus control algorithm. Excepted from the installation of the experimental setup, that is detailed below, no knowledge in ROS is required from the user (teacher or student) who would like to run an experiment.

The experiment setup is the one classically used, with an external system providing some localization information (pose, odometry), as presented on Figure 5. A Motion Capture system has been used composed of a network of external IR cameras and markers mounted on top of the robots. A PC receives data flow from the camera network and broadcast high-frequency accurate localization information.

As presented in Figure 5, a Wi-Fi router is used in the architecture of the experiment setup. It ensures network connection between the motion capture system, the robots

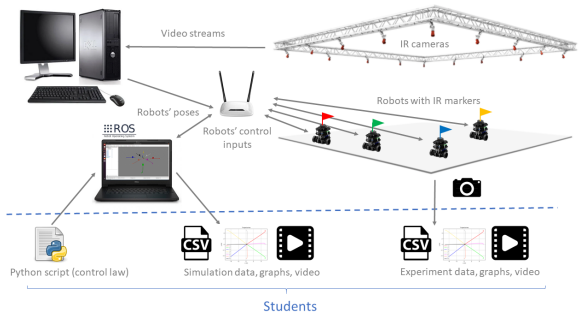

Fig. 5. Architecture for simulation and experiments.

and a laptop. This laptop is used by teacher(s) and student(s) to develop, simulate, and then experiment their control algorithms[5]. Additionally, a joypad can be connected to the PC to start and pause the experiment more easily.

D. Data recording and processing

Since the proposed software packages rely on ROS, and since the objective is to make it easy-to-use by teachers/students who do not have experience in ROS, some scripts are also provided to transform ROS data messages to a format easy to read and process.

ROS messages can be recorded and replayed in a very efficient way using the `rosbag` tool.When performing a simulation or an experiment, teacher(s) or students can therefore record all the information related to the poses of the robots, control inputs, exchanges of messages, etc. using a simple and single command line ("`rosbag record -a`"). A first script (`bag_to_csv.py`) provided with the `setcmas_simu` package allows then to convert the ROS messages into simple standard CSV data files. A second script (`csv_to_plot.py`) can then be used to draw plots, such as the ones presented in Figures 3 and 4.

Therefore, the proposed `setcmas` software stack leverages the advantages of the ROS data system but simplifies its use and exploitation by teachers/students that are not familiar with ROS.

As presented in Figure 5, the interfaces to teacher(s)/student(s) are indeed only Python scripts for the implementation of the control laws, CSV files for the data and visualizations (3D visualization and plots). Of course, students can also be encouraged to take video records of simulations and experiments with real robots.

Only simple command lines are required to run each step of the process (initiate simulation or experiment, run control laws, record and visualize data), making the software package suitable for use within the context of an academic module by teachers and students (eg. in practical work sessions or labs). Some examples of usages are presented in the next section.

III. EXAMPLES OF USAGES OF THE SETCMAS STACK

Three examples of usages of the `setcmas` stack are provided. They illustrate several possibilities for teaching, demonstrations, but also for research.

A. Teaching

The core of `setcmas` has been used in its initial versions for two modules in the Control Engineering curriculum of CentraleSupélec, graduate school of engineering in France. These modules, given at Master 1 and Master 2 levels, concern fleet control and control architectures of multi-agent systems.

The controller architecture from `setcmas` has been used to make students experiment with Turtlebot3 ground mobile robots and develop and validate in practice their own case study (eg. exploration, target tracking, artistic performances or games with robots, etc.) [3], [13]. The same paradigm has been used to make students implement control laws in pure Python scripts, and allow them to move from simulation to experiments without any knowledge on ROS. A picture of an experiment with 6 robots implementing a distributed control law for consensus is presented in Figure 6. It has been realized in the flight arena of CentraleSupélec, equipped with an Optitrack motion capture system.

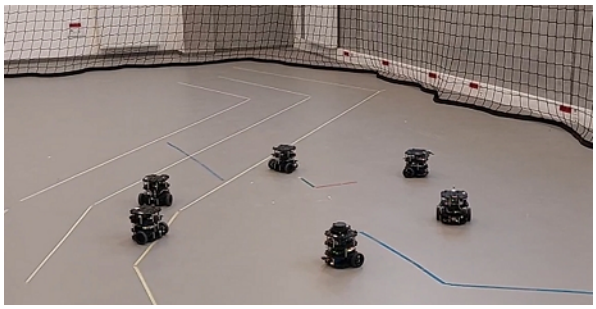

Fig. 6. Distributed consensus of 6 robots during practical work sessions of a Control Engineering curriculum at CentraleSupélec [3], [13].

B. Demonstration during Workshop

The `setcmas_simu` package has been developed for the Workshop "Guidance, navigation and control strategies for small-scale robotic platforms" at the 21st European Control Conference in 2023 [14].

The objective was to provide attendees of the Workshop with a Virtual Machine containing all the codes for simulation of distributed control algorithms for small ground mobile robots. The instructions to download, install and use the Virtual Machine are available in an online document[6]. It contains the previous version of the code made available with this paper (package for experiments is not included). Its packages are named `ecc3_*` instead of `setcmas_*`.

During the Workshop, presentation of how to implement distributed control laws in the Python scripts has been done. Different experiments have been made with the attendees of the Workshop in the conference room equipped with Qualisys motion capture cameras mounted on tripods (see Figure 7). The following experiments have been done with 3 Turtlebot3 robots: leader-follower formation control based on relative positions or distances, consensus, way point navigation, trajectory tracking.

C. Research

The controller structure and codes for Way Point navigation and leader-follower formation tracking of the `setcmas_ctrl` package have been used along with the configuration files of the `setcmas_expe` package to perform experiments in the context of a PhD thesis work. A fleet of 5 Turtlebot3 robots has been used to perform

[5]Note that although simulations can be launched from a Virtual Machine, it is preferable to launch experiments from a Linux/ROS distribution installed on a computer.

[6]`http://bit.ly/ecc23_vm`

Fig. 7. Distributed formation control experiments with 3 robots in a conference room during the Workshop on "Guidance, Navigation and Control Strategies for Small-Scale Robotic Platforms" at ECC 2023 [14].

formation navigation, playing the role of intruders in an area monitoring scenario with a sensor network composed of low-cost cameras mounted on tripods (see Figure 8). This reduce-scale mock-up has been used to successfully demonstrate in practice the performance of new distributed state estimation algorithms. A link to the video of the experiments can be found in [15].

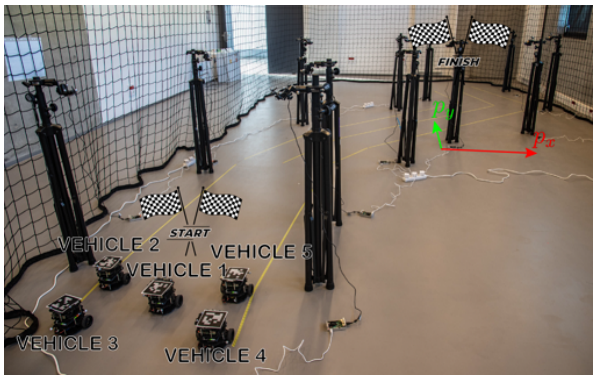

Fig. 8. Distributed formation control with 5 robots in an experiment for research work on distributed state estimation by sensor networks (picture from [15]).

IV. CONCLUSIONS

This paper has presented a software stack named `setcmas` which aims at facilitating simulations and experiments in teaching control of multi-agent systems. It leverages advantages of ROS, the Robot Operating System, but without requiring any knowledge on it for teacher(s) or student(s) using it. Implementation of control algorithms for multi-agent systems is easily done via simple Python scripts. It allows for simulations and experiments with Turtlebot3 mobile robots, and can easily be integrated into practical work sessions of Control Engineering curricula.

Future work will concern update of the experiment package to help working with low-cost localization systems (eg. based on QR Codes, on-board sensors of the robots, etc.). Use of the stack in other curricula dedicated to robotics will be also considered. Collecting feedback from students and other teachers is also intended, to evaluate further the stack and its possible benefits for Control Education.

ACKNOWLEDGMENT

The author would like to thank Cristina Stoica and Aarsh Thakker, from CentraleSupélec/L2S, for their collaboration in the use of the core of the software stack in the context of practical work sessions for graduate students in Control Engineering at CentraleSupélec. The author also would like to thank Florin Stoican, from University Politehnica of Bucharest, and Ionela Prodan, from Univ. Grenoble Alpes / Grenoble INP / LCIS, for the joint organization of the Workshop "Guidance, navigation and control strategies for small-scale robotic platforms" at ECC 2023 conference where the Virtual Machine and the software stack have been used for demonstrations.

REFERENCES

[1] J. McLurkin, J. Rykowski, M. John, Q. Kaseman and A. J. Lynch, "Using Multi-Robot Systems for Engineering Education: Teaching and Outreach With Large Numbers of an Advanced, Low-Cost Robot", in IEEE Transactions on Education, vol.56, no.1, pp.24-33, 2013.

[2] T. R. de Jager, N. K. Meinders, T. A. van Vugt, W. Zomerdijk and R. M. G. Ferrari, "Autonomous RC Cars for Control Research and Education: Implementation of Virtual Potential Based Navigation and Platooning", IEEE Conference on Control Technology and Applications, Trieste, Italy, 2022.

[3] C. Stoica Maniu, C. Vlad, T. Chevet, G. Rousseau, S. Bertrand, S. Olaru, "Modernizing Teaching through Experimentation on UAVs Formations", IFAC-PapersOnLine, vol.52, no.9, pp.144-146, 2019.

[4] F. M. Noori, D. Portugal, R. P. Rocha, M. S. Couceiro, "On 3D Simulators for Multi-Robot Systems in ROS: MORSE or Gazebo?", International Symposium on Safety, Security and Rescue Robotics, Shanghai, China, 2017.

[5] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, M. Dorigo, "ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems", Swarm Intelligence, vol.6, no.4, pp.271-295, 2012.

[6] M. Casini and A. Garulli, "MARS: An Educational Environment for Multiagent Robot Simulations", Modelling and Simulation in Engineering, 2016.

[7] E. Ferrera, J. C. Fernandez, M. Stampa, P. J. Marron, "MiMicS: a Multi-robot simulator for teaching, rapid prototyping and large scale evaluations", ACM SAC Conference, Pau, France, 2018.

[8] J. McLurkin, J. Rykowski, M. John, Q. Kaseman, A. J. Lynch, "Using Multi-Robot Systems for Engineering Education: Teaching and Outreach With Large Numbers of an Advanced, Low-Cost Robot", in IEEE Transactions on Education, vol.56, no.1, pp.24-33, 2013.

[9] J. M. Canas, E. Perdices, L. Garcia-Perez, J. Fernandez-Conde, "A ROS-Based Open Tool for Intelligent Robotics Education", in Applied Sciences, vol.10, no.7419, 2020.

[10] M. A. Chunab-Rodríguez, A. Santana-Diaz, J. Rodríguez-Arce, E. Sanchez-Tapia, C. A. Balbuena-Campuzano, "A Free Simulation Environment Based on ROS for Teaching Autonomous Vehicle Navigation Algorithms", in Applied Sciences, vol.12, no.7277, 2022.

[11] M. Fernandez-Cortizas, M. Molina, P. Arias-Perez, R. Perez-Segui, D. Perez-Saura, P. Campoy, "Aerostack2: A Software Framework for Developing Multi-robot Aerial Systems", arXiv 2023, arXiv:2303.18237.

[12] L. Ding, Q.-L. Han, X. Ge, X.-M. Zhang, "An Overview of Recent Advances in Event-Triggered Consensus of Multiagent Systems", IEEE Transactions on Cybernetics, vol.48, no.4, pp.1110-1123, 2018.

[13] S. Bertrand, C. Stoica Maniu, C. Vlad, "Teaching by Practice the Basis of Consensus for Multi-Agent Systems", IFAC Workshop on Aerospace Education, Milano, Italy, 2021.

[14] F. Stoican, S. Bertrand and I. Prodan, "Guidance, navigation and control strategies for small-scale robotic platforms" Workshop at the 21st European Control Conference, Bucharest, Romania, 2023.

[15] A.Venturino, C. Stoica Maniu, S. Bertrand, T.Alamo, E. F. Camacho, "Multi-vehicle localization by distributed MHE over a sensor network with sporadic measurements: Further developments and experimental results", in Control Engineering Practice, vol. 132, 2023.