ASEE CoED Journal
Published 30 Nov 2021

Use of Open-source Software in Mechatronics and Robotics Engineering Education – Part I: Model Simulation and Analysis

Nima Lotfi[1*], Dave Auslander[2], Luis A. Rodriguez[3], Kenechukwu C Mbanisi[4] and Carlotta A Berry[5]

[1]**Mechanical and Mechatronics Engineering Department, Southern Illinois University Edwardsville, Edwardsville, Il, USA**

[2]**Mechanical Engineering, University of California Berkeley, Berkeley, CA, USA**

[3]**Mechanical Engineering, Milwaukee School of Engineering, Milwaukee, WI, USA**

[4]**Robotics Engineering, Worcester Polytechnic Institute, Worcester, MA, USA**

[5]**Electrical and Computer Engineering, Rose-Hulman Institute of Technology, Terre Haute, IN, USA**

ORIGINAL RESEARCH

Abstract

Open-source Software (OSS) provide attractive solutions for complementing Mechatronics and Robotics Engineering (MRE) education due to their numerous advantages such as free access, customizability and wide community support, increased adoption and utilization in industry, etc. To provide a deeper insight on the current status, limitations, and potentials of the OSS, a summary of the results of an online survey, conducted among various community stakeholders, is included. Furthermore, the two parts of this contribution are intended to provide an exposure to the OSS which have the potential to be used in MRE education. To this end, two specific problems, namely, model simulation and analysis of a DC motor (Part I) and controller implementation for a 2-DOF robot manipulator (Part II), are solved using Python, Java, Modelica, GNU Octave, and Gazebo. The systems chosen for this work are some of the most-commonly encountered systems and the considered problems, i.e. model simulation, analysis, and control implementation, are fundamental problems in the context of MRE. Therefore, this work can help MRE instructors familiar with these OSS to easily integrate them into their respective courses. On the other hand, the students can also greatly benefit from this work as it provides an entry point into applying different OSS in MRE-related courses and projects. Exposure to various ways in which an MRE problem is translated into a software solution enables the students to generalize their problem-solving process across different computational tools. Students equipped with such a skillset would potentially have a flexible mindset and could make well-balanced and informed judgements when devising a solution to a real-world engineering problem. Full code scripts for each of the OSS introduced in this work, along with Matlab which is intended as a point of comparison, are included on the GitHub repository of the paper to provide free access to the community and to help with widespread adoption of the OSS in MRE higher education.

Keywords: Open Source Software, Robotics Engineering Education, Model Simulation, Programming Languages

OPEN ACCESS

Volume
12

Issue
3

Corresponding author
nlotfiy@siue.edu

Submitted *17 Jan 2021*

Accepted *1 Nov 2021*

Citation
Lotfi N., Auslander D., Rodriguez L.A., Mbanisi K.C. and Berry C.A. "Use of Open-source Software in Mechatronics and Robotics Engineering Education – Part I: Model Simulation and Analysis," *Computers in Education Journal*, vol. 12, no. 3, 2021.

1 Introduction

The field of Mechatronics and Robotics Engineering (MRE) has experienced tremendous growth in the past few decades, mainly due to the advancements in integrated circuits and electronics, computers, control systems, and connectivity and networking. Applications such as consumer

electronics, home and building automation, medicine and healthcare, manufacturing and industrial robotics, heavy machinery, autonomous transportation, and aerospace applications, in addition to democratization of access through Maker Movement have further expedited the advancements in the field of MRE.

Considering the prevalence and potentials of MRE, it is of utmost importance to educate the future engineers by providing an interdisciplinary knowledge of mechanical, electrical, computer, software, and systems engineering so that they can undertake and oversee multidisciplinary design and development efforts in this field and to define the future of work at the human-technology frontier. In addition to technical breadth and depth, the MRE educational institutions should also provide the students with a rich hands-on skillset. To this end, in the past decades, numerous software and hardware packages have been developed to further engage the students through laboratory and project-based learning paradigms. Such experiences could give students a deeper understanding of the core MRE concepts as well as an opportunity to practice the application of those concepts – linking knowledge and real-world skills. Students who demonstrate this understanding can confidently study real-world global challenges, devise multiple solutions to these problems, evaluate the feasibility of the various solutions based on the qualitative and quantitative criteria and constraints, and implement the optimal solution while considering the broader societal impacts.

Although commercial products such as Matlab and Simulink, which are targeted towards improving MRE education, have been widely adopted in academia, their acquisition and licensing fees can be a big hurdle for academic institutions which lack the necessary financial infrastructure. Furthermore, the students typically lose access to these resources once they graduate and leave the educational environments. Finally, there is an increasing trend in industry towards the development, distribution, and adoption of open-source platforms as they can reduce the total cost of ownership, reduce dependence on vendors, and facilitate higher flexibility and customizability.

The open-source community has been growing steadily since its first introduction in the early 1980s. The "open-source way", which was initially used in software development, distribution and maintenance, was motivated by the need for better collaboration amongst developers, improved accessibility and reliability of software through a network of contributors. Open-source software (OSS) is defined as software released under a license which grants users the right to study, use, modify, and distribute the source code to anyone and for any purpose [1]. Linux operating system and Apache Web server are some of the early developments of the community. Currently, numerous high-quality open-source alternatives to proprietary software are available in various applications, which have gained widespread adoption.

In higher education, the first exposure to the OSS has traditionally been in introductory programming courses. Different programming languages such as C, C++, Java, etc. have been used to introduce the students to the fundamentals of programming and to develop their computational thinking mindset. In recent years, scripting languages such as Python have attracted more attention in the introductory programming courses, specifically due to their increased popularity, simplicity, and flexibility [2, 3]. Different types of the OSS have also been used in higher-level courses in Computer Science [4] and Software Engineering [5] programs. A detailed overview of the OSS implementation in Software Engineering during a 3-year project along with students' perception and feedback are presented in [6]. The accessibility and customizability of the OSS have also contributed significantly to online and distance education [7, 8]. An empirical study was conducted in [9] to evaluate the common perceptions about the OSS and their applications and to provide a guideline for practitioners who wish to implement or switch to OSS.

The applications of the OSS in MRE education have mainly revolved around the use of the family of C languages, and recently Python, in embedded system programming and hardware interface. Reduced costs and advancements in open-source hardware such as Arduino, Raspberry Pi, BeagleBone, etc. have further increased the use and popularity of these software. The open-source hardware platforms have also enabled the community to develop low-cost laboratory experiments in order to complement the heavy theoretical content of MRE courses such as Embedded Systems [10], Controls [11–13], and Robotics [14–17]. A number of simulation platforms have also

been developed to teach motion dynamics [18] and industrial robotics [19]. Finally, Journal of Open Source Software is an extensive repository that contains articles on the development of specialized OSS tools. Some of such tools related to MRE include: ros_control: A generic and simple control framework for ROS [20], CoreRobotics: An object-oriented C++ library with cross-language wrappers for cross-platform robot control [21], DART: Dynamic Animation and Robotics Toolkit [22], DmpBbo: A versatile Python/C++ library for Function Approximation, Dynamical Movement Primitives, and Black-Box Optimization [23], Phobos: A tool for creating complex robot models [24], Manif: A micro Lie theory library for state estimation in robotics applications [25], SLAM Toolbox: SLAM for the dynamic world [26], and Pybotics: Python Toolbox for Robotics [27].

Utilizing the OSS in MRE education is a feasible option for a wider range of institutions. Furthermore, it can familiarize the students with the development details of real-world systems and therefore, provide a deeper learning experience, which can be very attractive for future employers who are also wishing to utilize OSS. There are, however, a few challenges that need to be resolved before widespread adoption of the OSS in MRE education. These challenges include lack of familiarity among MRE educators with the utilization and application of OSS platforms; lack of tutorials and documentation tailored specifically for the design, analysis, and real-time implementation of MRE systems; the skillset to navigate through vast online resources about OSS for debugging and troubleshooting.

To overcome some of the aforementioned challenges, this work, presented in two parts, aims to increase the MRE community awareness and familiarity with the applications of the OSS. This article, i.e. Part I, addresses model simulation and analysis of a DC motor using some of the commonly-used OSS. DC motors, which are one of the most versatile systems in any MRE system, are governed by a set of linear ordinary differential equations. Familiarity with the simulation and analysis of DC motors using the OSS can pave the way towards their utilization in a wide range of MRE systems. This paper is organized as follows: Section 2 provides background information about the OSS with an extensive online support and the potential to be used in MRE education, namely, Python, Modelica, Java, GNU Octave, and Gazebo. Section 3 summarizes the results of a recent survey conducted among various community stakeholders to gauge the current status, potentials, and limitations of the OSS. Section 4 introduces the dynamics and the parameters of the DC motor used in simulations. In Section5, Python, Modelica, Java, and GNU Octave are used to simulate the dynamics of the introduced DC motor. Although the DC motor simulations can be accomplished in Gazebo, this software package is only considered in Part II of this work for the closed-loop control of a robot manipulator. Code snippets from each software are provided in Section 5 to show the structure of the programs used for the simulations along with the other capabilities of the OSS and resources to learn more about them. This work is not an introduction to programming, neither is it a work on the modeling and control of dynamic systems. It, however, can provide a detailed and thorough exposure to the capabilities and application of some of the common OSS in the context of MRE systems. To the authors' best knowledge, this work is a unique contribution aiming to ease the adoption and application of various OSS in MRE education.

2 Overview of Different Open-source Software

In this section, the OSS with the high potential for use in MRE education and with wide community support are introduced. These software include general programming languages such as Python and Java and more specialized platforms such as Modelica, Gazebo, and GNU Octave. All the open-source software in this work, except for Java, are technically considered scripting languages, as opposed to programming languages, and interpreted languages, as opposed to compiled languages. For brevity and clarity of presentation, the generic term software (and the abbreviation OSS) is used throughout the article to refer to these languages.

2.1 Python

Python is a general-purpose and interpreted programming language, first developed by Guido van Rossum and released in 1991. In order to make it freely accessible to everyone, it is developed

under an open-source license. Python Software Foundation, which is responsible for administering Python's license, was formed "to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers" [28]. Due to its high-level and open-source nature, large community of developers in Python, an abundance of packages developed for Python [29], and easy integration with other programming languages, Python has gained popularity in various applications and disciplines and has turned into an industry standard. Consequently, many valuable resources have been developed for getting started with Python. The website [28] provides a rich repository of documentation about Python installation, learning materials, news and events. In addition to many books written about Python [30, 31], several MOOCs (Massive Open Online Courses) also exist to provide familiarity with the fundamentals, advanced topics, and applications of Python [32, 33].

In the past decade, the Python community has developed numerous packages for mathematics, science, and engineering applications [34, 35]. The following is a list of some of these libraries

- SciPy: A comprehensive library with sub-modules for various mathematical and scientific operations such as Fast Fourier Transform, interpolation, numerical integration, linear algebra, file input/output, optimization and curve-fitting, statistics, and signal processing.
- NumPy: Offers computational capabilities similar to Matlab which also overlaps with some of SciPy's functionalities. It is best suited for fast array operations. Matlab users who wish to learn more about NumPy can refer to [36] for a detailed comparison between Matlab and NumPy.
- Matplotlib: Enables high-quality 2D plotting. It can be used to generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc.
- SymPy: Used for symbolic programming. It is incorporated into a wide range of other libraries such as symbolic statistical modeling, multibody dynamics, linear circuit analysis, etc.
- IPython: Provides an interactive shell similar to Matlab's workspace which is ideal for troubleshooting, interactive data visualization, use of GUI toolkits, and parallel computing.
- pandas and Scikit-learn: Offer an extensive suite of tools for predictive data analytics, modeling, and machine learning.
- Python Control Systems Library: Provides functions and classes to systematically analyze and design feedback control systems along with a Matlab compatibility module for users more familiar with Matlab [37].

As mentioned earlier, Python has received an increased attention in the past few years, mainly due to the fact that while it is a general-purpose, high-level, and interactive programming language, it is also equipped with powerful features and add-on libraries which place it at the same high level as specialized software such as Matlab. To catch up to its fast pace of growth and popularity and to familiarize the future generation, Python is increasingly being introduced in higher education curricula. In the past decade, majority of top CS departments in United States have switched to Python in their introductory computer programming courses [38]. According to a comparative study in [39], among Python, C, and Matlab as the programming language of choice for engineering education, Matlab is determined to be more suitable compared to C, however, Python is selected as the best option due to its clarity and functionality. A similar conclusion about Python being the best choice for engineering education compared to Matlab and C is achieved in [2]. Authors in [40] discuss the motivation and the process of a transition from C to Python in the first computer programming course for Mechanical Engineering students and report a reduced failure rate among the students, among other observations. In the context of MRE education, Python has mainly been used as the programming language for open-source hardware such as Raspberry Pi [41, 42], which is typically used for teaching control engineering. Python has also been used in [43] to develop internet tools to teach the fundamentals of control.

2.2 Java

While Java is not the first programming language environment that comes to mind in an MRE context, it has many properties that make it a very useful tool. The first of these is the language itself. Java is fully object-oriented, has a relatively clean syntax, executes quickly, and operates with no changes (usually) across different computers and operating systems. It can be applied to mathematically based problems (for example, dynamic simulation) as well as used for direct control of physical systems.

Java compiles to an abstract machine code that executes via a Java Virtual Machine (JVM). Thus, a compiled Java program can execute on any system for which a JVM exists. This guarantees that basic properties such as how integer and real numbers are stored, precision associated with different data types, etc., will not change from one environment to another.

Java is open source. The Java Development Kit (JDK) including, compiler, linker, etc. is available from [44]. Commercially licensed versions are also available from Oracle. The two most popular Integrated Development Environments (IDE) are also open source, Netbeans [45] and Eclipse [46].

While the Java language itself provides very effective ways to organize complex problems, solving them often requires additional numerical math tools. For a wide range of problems, the appropriate tools are available through the open source Apache Commons Math Library [47] or its successor, the Hipparchus Math Library [48]. There are also other smaller, more specialized libraries that can be found online.

Java is used as the computing language for the Advanced Placement Computer Science course and is also widely taught in lower division undergraduate courses in computer science as well as other programs. Because of this, there is a large amount of material on learning Java – so much that any listing here would barely scratch the surface. The material consists of books, online tutorials, videos, forums, etc., much of it freely available. In general, this material focuses on learning Java without going very deeply into the subject matter used for examples. For material that is more closely focused on the subject matter of this contribution, a well-documented course on using Java for simulation of physical systems can be found at [49]. A more extensive discourse on the use of Java in simulation has been produced under the auspices of the Open Source Physics project [50].

2.3 Modelica

"The Modelica Language is a non-proprietary, object-oriented, equation-based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents" [51].

Modelica is of particular interest to the mechatronics/robotics community because it provides dynamic simulation of three-dimensional and constrained mechanical systems and energetically correct interactions between the mechanical systems and electrical systems, hydraulic, pneumatic, etc. It does this through an underlying mathematical base that is capable of solving differential-algebraic equations (DAEs) and an underlying computing syntax based on equations rather than algorithmic "store-into" statements. Most users do not interact with Modelica at that level, however, but rather make use of an extensive set of libraries for various physical domains combined with a graphical user interface.

Systems with constraints are known as "acausal". For a system defined by a set of N first-order, nonlinear differential equations, constraints add algebraic equations to the system equations. Acausal systems have fewer than N independent initial conditions because the algebraic constraints reduce the effective order of the system. In mechatronics, almost all mechanical systems of any interest have constraints which is one reason Modelica is so valuable.

The Modelica language is defined by an open source specification maintained by the Modelica Association [modelica.org]. The actual implementation is left to any organization, including both commercial and open source entities. The most important open-source implementation is OpenModelica [52].

Modelica has been heavily used in certain industries, but less so in college-level teaching and, probably, not at all at the high school level. Thus, there is not nearly as much instructional material available as there is for much more widely used languages such as Python or Java. There is, however, some useful material to cite. The Open Source Modelica Consortium [53] has a short, introductory course, as well as a longer, more in-depth course at [54]. "Modelica by Example" is an in-depth, online book on Modelica [55]. It is also available for sale (on a "pay what you can" basis) in various eBook formats. A set of introductory slides that uses many examples from "Modelica by Example" is available at [56]. Because Modelica is explicitly aimed at simulation of physical systems, any instructional material on Modelica is, by definition, also about simulation of physical systems.

2.4 Gazebo

Since its debut in 2007, ROS (Robot Operating System) has grown into the most prominent open-source middleware framework for robotics research and development in industry and academia. Likewise, the Gazebo Simulator has become increasingly popular, thanks in part to its robust integration with the ROS framework [57]. Gazebo is an open-source 3D dynamic simulator that provides a framework for realistic simulation of multiple robots dynamically interacting with each other in complex environments and scenarios [58, 59]. Gazebo comprises a suite of features which include high-performance physics engines (e.g. Open Dynamics Engine, Bullet, etc.), high-fidelity graphical rendering of environments and ability to generate realistic sensory data from the environment [58]. These features allow users to create very realistic simulated environments and scenarios to learn to operate and test their robot designs and control algorithms without the risk of damaging their physical robots. Users can either leverage the rich library of existing robot models available with the software (ranging from mobile aerial and ground robots to state-of-the-art humanoid robots) or design their own robot models from scratch using the Unified robot description format (URDF). As mentioned earlier, Gazebo will be used in the second part of this work to simulate the closed-loop performance of a robot manipulator.

In the context of MRE education, Gazebo and ROS have been used in several ways to help students learn core robotics concepts. A two-part introductory robotics course was presented in [60] using both Gazebo and ROS. A graduate course on mobile robotics and robotic manipulation in the context of Industry 4.0 was developed in [61] using both software. In [62], Gazebo and ROS were both applied in an advanced course on humanoid robotics. In robotics education, open-source simulation tools are particularly relevant because they address the challenge of hardware affordability, making it possible to design labs partially or entirely in simulation [61, 63]. The integration of these open-source software into the curriculum requires that the students are provided with opportunities to learn how to use them. For instance, in [61], the lab included preparatory sessions where the students complete comprehensive tutorials on the software available online [64].

2.5 GNU Octave

GNU Octave [65] is a high-level programing language for numerical computations with built-in plotting and visualization tools. Octave is an open-source software with syntax largely comparable to and compatible with MATLAB. It can be freely redistributed and/or modified under the GNU General Public License. It runs on multiple platforms including GNU/Linux, macOS, BSD, and Windows. Octave was primarily developed by John W. Eaton and was conceived in 1988 as companion software for a chemical reactor design textbook. It is an interpreted programming language designed to solve a variety of engineering problems involving the numerical solution of linear and nonlinear problems and ordinary differential equation problems. It also contains an extensive collection of packages that enhances its core functionality by introducing field specific features. Some of the packages relevant to MRE systems include the Control, Image Acquisition, Image, Optim, and the Arduino package which are used for control design and analysis, image acquisition and processing, optimization, and Arduino control, respectively. Octave also includes both a

graphical user and a traditional command-line interface that provides greater flexibility for novice and advance users. It can also be used to write non-interactive user-defined programs and can call C, C++, and Fortran code within Octave using Oct-Files or using MATLAB-compatible Mex-Files. For new users and developers Octave also provides an active Wiki-page containing installation, development, and documentation resources, in addition to the built-in Octave documentation [66].

In engineering education, GNU Octave has been used as an alternative platform to provide a computational environment to process and visualize data. Its appeal is linked to the fact that it is a free and open-source licensed product that is highly compatible with the MATLAB programming language. Universities have utilized Octave to create line following and robot simulators and to process data from mobile robot projects [67–69]. It has also been combined with open source hardware such as the Raspberry Pi to perform image processing tasks in robot arm sorting applications [70]. One of the main challenges with Octave that educators have pointed out is the lack of repository libraries like MATLAB's mature toolboxes [68]. Nevertheless, Octave has emerged as a promising alternative tool to MATLAB without compromising performance and productivity [70].

3 Community Impressions of the Open-source Software

This section provides a summary of a recent survey conducted in order to understand the extent to which professionals in various engineering disciplines, such as mechanical engineering, electrical and computer engineering, and mechatronics and robotics engineering are familiar with and utilize the OSS. The survey was conducted online and was distributed to community members through personal invitations and posts on academic, technical, and professional email lists and social media. A total of 131 responses were received from various stakeholders comprising of 55 current engineering students, 59 engineering educators, 14 industry professionals, and 2 Other (a mathematics and a retired professor). Figure 1 below shows department affiliation/area of expertise distribution for each of the above categories.

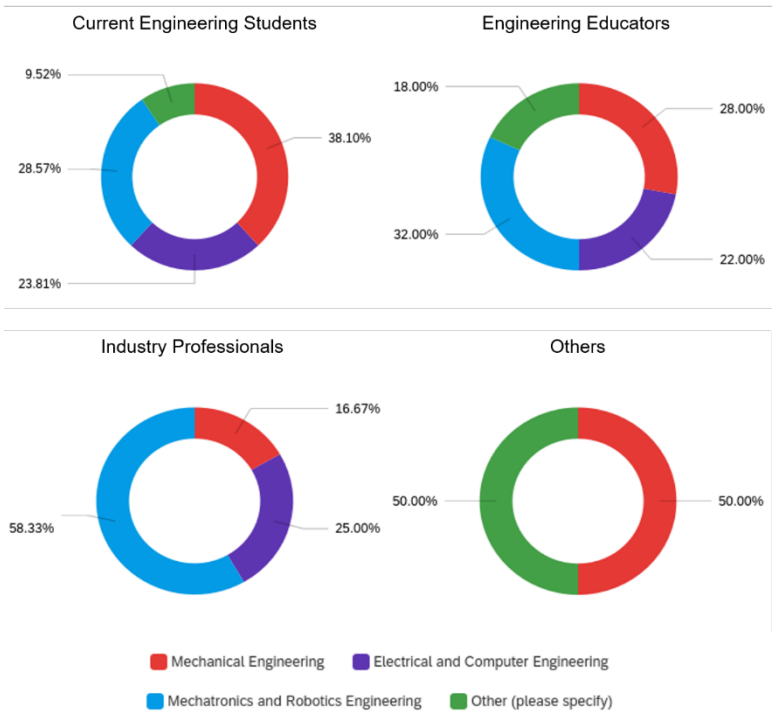

Figure 1. Department affiliation/area of expertise distribution among survey participants.

The Other affiliation in all the categories above comprised mainly of individuals in the field of computer science.

The OSS considered in this survey included Python, Java, Modelica, GNU Octave, Gazebo, and family of C languages. Table 1 below shows the percentage of survey participants with no/slight familiarity with each of these OSS.

Table 1. Percentage of survey participants with no/little familiarity with the OSS.

	Python	Java	Model-ica	GNU Octave	Gazebo	Family of C Languages
Students	50%	72.50%	100%	92.68%	90.25%	26.83%
Educators	35.41%	66.67%	93.62%	61.70%	87.24%	20.41%
Industry Professionals	18.18%	36.36%	90.91%	54.54%	63.63%	9.09%
Other	100%	100%	50%	100%	100%	50%

As can be seen from the results in Table 1 , majority of the survey participants are familiar with the family of C languages which are typically taught as freshman introduction to programming/-computing courses and are mainly used for hardware interface. Considering the focus of this article on model simulation, analysis, and controller implementation, this category of the OSS is not considered here. Although the remainder of the OSS, as discussed in this work, have the potential for use in MRE applications, there seems to be a lack of familiarity with these software, especially Modelica, GNU Octave, and Gazebo.

In response to a question on how the participants learnt the above OSS, 54.79% of the students, 93.44% of the educators, 73.68% of the industry professionals, and 100% of the others described their learning experience as self-taught using a book, a Massive Open Online Course (MOOC), or trial/error, as opposed to learning through a formal required/elective course.

As for prior experience, 56% of educators and 75% of industry professionals have used the OSS in their teaching and careers, respectively. On the other hand, 68% of the educators and 66.67% of the industry professionals definitely plan to use the OSS in their respective sectors in the future, whereas 30% of the faculty and 25% of industry professionals do not have certain plans in this regard.

Finally, through two open-ended questions, the survey inquired about the challenges the participants have faced in learning/using the OSS and their perceptions on the potentials and benefits of the OSS. A summary of the participants' responses is presented below in Table 2 . Repeated responses are eliminated for brevity.

As can be seen from the community survey results, while the OSS have numerous potentials and benefits, there are still several challenges that need to be addressed before their widespread adoption. Although the current work is not intended to address all these challenges, it provides a specialized focus on the application of the OSS in MRE. Developed by the MRE faculty and students for the MRE community, this two-part contribution facilitates ease of entry and can enable MRE educators and students who wish to utilize the OSS in their courses and projects. Such an exposure for the students can further develop their computational mindset and problem-solving skills and therefore, prepare them for the job market. Future work will provide a more detailed analysis of the survey results to build a roadmap for further overcoming the challenges and more widespread adoption of the OSS in higher education.

3.1 Motor Case Study

A DC motor is one of the most commonly used actuators for generating rotational and translational motions in MRE systems. Furthermore, DC motors are perfect examples of an MRE system due to their multidisciplinary nature. The DC motor dynamics are governed by linear differential equations, and therefore, familiarity with numerical simulation of DC motor dynamics can pave the way to study other MRE systems that the students and professionals may encounter in practice.

The circuit diagram below is a typical schematic of an armature-controlled DC motor.

Table 2. Summary of the survey results on community impressions of the open-source software.

Challenges of OSS	Potentials and Benefits of OSS
-Difficulties in initial installation, package additions, and setup - Dependency on specific operating systems	- Based on community learning and collaboration
- Where/how/what to learn for a specific application	- No acquisition costs
- Compatibility with commercial PCs and software	- Source code availability and customizability
- Lack of official technical support and quality documentation	- Security, stability, and privacy
- Specialized support for intermediate users	- Acceptance as industry standard
- Lack of familiarity with proper online search towards troubleshooting specific problems - Initial learning curve	- Strengthened relationship between the user and the software
- Version compatibility and frequent updates	- Accessibility for low-income and unprivileged society groups
- Reliability of the online information	- Applicability to a wider range of projects
- Lack of motivating examples in the classrooms	- Desirable among future employers
- Lack of practice exercises - Multiple off-shoots/flavors	- Speeding up the project due to the available libraries and online solutions
- Potentially higher maintenance costs	- Cost and time savings for industry
- Plagiarism issues	- Promoting and enabling innovation for a wider community
- Security of online forums and websites	- Promoting self-learning
- Conflicts with campus IT	- Long-term accessibility, even after graduation
- Lack of maintenance	- Incorporation of community feedback
- Specialized nature of OSS	- Large number of online tutorials
- Difficulty gauging the maturity, stability, and level of community support for certain OSS	- No licensing hassles
- Managing virtual environments and versions of Python	- Suitable for online education
- Time commitment for self-learning	- Adaptability to certain applications
- Lack of open-source hardware drivers	- Potential to promote learning core skills
- An OSS with Simulink capabilities	- Lack of commercials and marketing noise
- Lack of prior familiarity among the students and faculty	- Avoiding subscription-based billing models
- Lack of course specific material	- Plug and play design
- Technical documentation in non-English languages	- Ease testing multiple methods without a high cost of integration
- Cross-platform issues	- Lack of need for training new hires by the industry
- Difficulty and time commitment to apply the OSS in control systems	
- Lack of university and college support	
- Lack of backward compatibility	
- Unresolved bugs	
- Corporate disapproval to contribute to the open-source community	
- Low-quality user interface	
- Not suitable for short timeframe professional projects	
- Customer acceptance	
- Little emphasis on performance and reliability	
- Lack of high-quality embedded compilers	
- Potential legal issues	

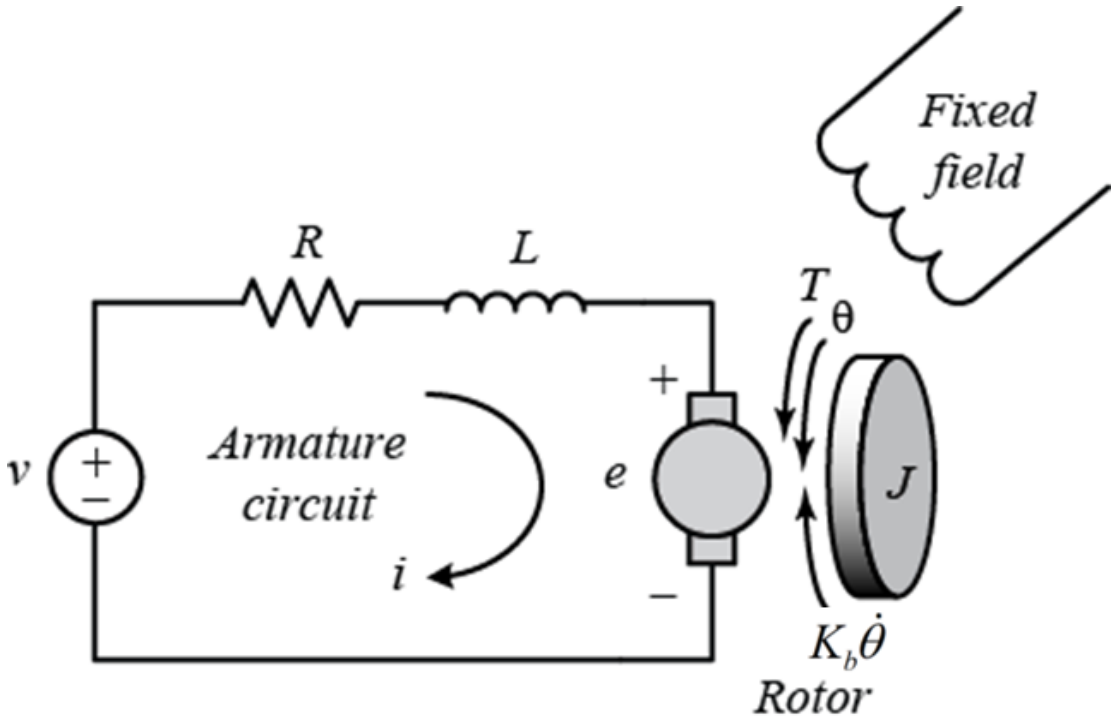

Figure 2. Schematics of an armature-controlled DC motor [71].

In this DC motor, the armature voltage v, is the input to the system whereas the rotational displacement of the motor shaft θ, the angular velocity of the motor shaft ω = dθ/dt, and the armature current i are assumed to be the system outputs. Using Kirchhoff's and Newton's laws, the dynamics of this system can be described by the following set of differential equations:

$$\begin{aligned} J\frac{d^2\theta(t)}{dt^2} + b\frac{d\theta(t)}{dt} &= K_t i(t) \\ L\frac{di(t)}{dt} + Ri(t) &= v(t) - K_b\frac{d\theta(t)}{dt} \end{aligned} \tag{1}$$

where J is the moment of inertia of the motor shaft, b is the viscous friction coefficient of the motor shaft, K_t is the armature constant, L is the armature electrical inductance, R is the armature electrical resistance, and K_b is the back-emf constant. The values of these parameters used here are obtained from the datasheet information and model identification tests conducted on a brushed servomotor, C40-E-500FE, from Advanced Motion Controls [72]. These parameters are reported in Table 3.

Table 3. 3. DC motor parameters used in simulations.

Parameter	J	b	K_t	K_b	R	L	V_{nom}
Value	0.0026 kg.m^2	0.01 N.m.s/rad	0.66 N.m/A	0.66 V/rad/s	2.62 Ω	0.05 H	24 V

To conform to common numerical simulation algorithms required to solve the DC motor dynamics in Equation 1 , the equations first need to be converted to a set of first-order ODEs as shown below:

$$\begin{aligned} \frac{d\theta(t)}{dt} &= \omega(t) \\ \frac{d\omega(t)}{dt} &= \frac{b}{J}\omega(t) + \frac{K_t}{J}i(t) \\ \frac{di(t)}{dt} &= -\frac{K_b}{L}\omega(t) - \frac{R}{L}i(t) + \frac{1}{L}v(t) \end{aligned} \tag{2}$$

These equations can also be written in a state-space form:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}u(t) \end{aligned} \tag{3}$$

where **x**(t) = **y**(t) = $[\theta(t), \omega(t), i(t)]^T$, u(t) = v(t), and the matrices **A**, **B**, **C**, and **D** are:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{b}{J} & \frac{K_t}{J} \\ 0 & \frac{-K_b}{L} & \frac{-R}{L} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{D} = 0 \tag{4}$$

4 DC Motor Model Simulation and Analysis

An important first step in the analysis and control design for any Linear Time-Invariant (LTI) system, including DC motors, is to analyze their time-domain behavior. To this end, the introduced OSS are used in this section to obtain the step response of the DC motor dynamics in Equation 1 . For each OSS, the important code snippets are introduced and explained. Furthermore, other software-specific tools which can be used for further analysis and control design of LTI systems are introduced. Full code scripts can be accessed from the Github repository of the article at [73].

4.1 Python

In order to obtain the step response of the DC motor, the governing ODEs can be directly solved in Python using **solve_ivp** command from the **scipy.integrate** library [74]. The other library required for this simulation includes **numpy** (imported as **np**). The **solve_ivp** command syntax for solving the ODEs in Equation 2 or Equation 3 is:

```
sol=solve_ivp(model,(t0,tf),x0,dense_output=True,vectorized=True,
    args=(v,))
x=sol.sol(t)
```

Code Snippet 1.

where model is a user-defined function. For the ODEs in Equation 2 , this function can be written as:

```
def model(t, x, u):
    theta,omega,i=x
    dxdt = [omega,-(b/J)*omega+(Kt/J)*i, \
        -(Kb/L)*omega-(R/L)*i+(1/L)*u]
    return dxdt
```

Code Snippet 2.

whereas for the ODEs in Equation 3 , the function is:

```
def model(t, x, u):
    A = [[0, 1, 0],[0,-(b/J),(Kt/J)],[0,-(Kb/L),-(R/L)]]
    B = [[0], [0], [1/L]]
    dxdt = np.dot(A,x) + np.dot(B, u)
    return dxdt
```

Code Snippet 3.

In Python, the model parameters can be specified after the definition of the model function. The simulation time is defined using the **arange** command as $0 \leq t \leq 0.5$ with a step size of 0.001 s. Finally, the input voltage v is chosen as the nominal motor voltage, i.e. 24 V, and the initial conditions vector, **x0**, is defined as a 1-dimensional list, **x0 = [0, 0, 0]**.

Python Control Systems Library [37] can provide alternative methods for solving for the step response of the DC motor dynamics. The best way to install this library is through using a pip installer, more information can be found in the official documentation [37]. After the installation, this library needs to be imported to the Python workspace (imported as **ctrl** here.) The step response can then be obtained using the **step_response** command from the Python Control Systems library:

```
t, x = ctrl.step_response(v*ssModel, t)
```

Code Snippet 4.

where **ssModel**, the state-space representation of the system described in Equation 3, is defined as:

```
ssModel = ctrl.ss(A,B,C,D)
```

Code Snippet 5.

While the classes and commands in Python Control Systems Library can be used for simulating the step response, Matlab compatibility module, also included in this library, can provide another convenient method for model simulation, especially for those more familiar with Matlab. Using this module, the step response can be obtained as:

```
ssModel = matlab.ss(A,B,C,D)
x, t = matlab.step(v*ssModel, t)
```

Code Snippet 6.

where matlab is an alias used for importing the control.matlab module. Note that the state vector, **x**, obtained using Matlab Compatibility Module has a dimension of $n \times 3$ whereas the previous methods generate a state vector with a dimension of $3 \times n$, where n is the length of the time vector. Finally, the library matplotlib.pyplot can be used for plotting system states and plot customizations. Figure 3 shows the step response of the DC motor, plotted using **matplotlib.pyplot** library in Python. Although slightly different compared to Matlab, this library provides a wide range of programmatic customization tools for plotting and modifying the appearance of the plots and is easy to implement [75]. The generated figures, however, do not have a lot of the interactive functionalities such as setting cursors, graphical modification of the plots, etc, that Matlab provides.

Other system descriptions such as transfer functions and frequency response data models can also be defined in Python Control Systems Library using **tf** and **frd** commands, respectively. Other features of Python Control Systems Library which can be very helpful in the context of Mechatronics and Robotics Engineering can be found within the online documentation [37]. Examples of some of these capabilities are given in the next section alongside similar features in Octave Control package.

4.2 GNU Octave

To obtain the step response of the DC motor to a constant armature voltage of 24 volts, two approaches are used in this section. The first approach uses Octave's built-in function **ode45** while the second approach implements the **step** function in Octave's Control package to obtain the time response of the governing equations of the motor. For this example, it is assumed that the motor starts from rest.

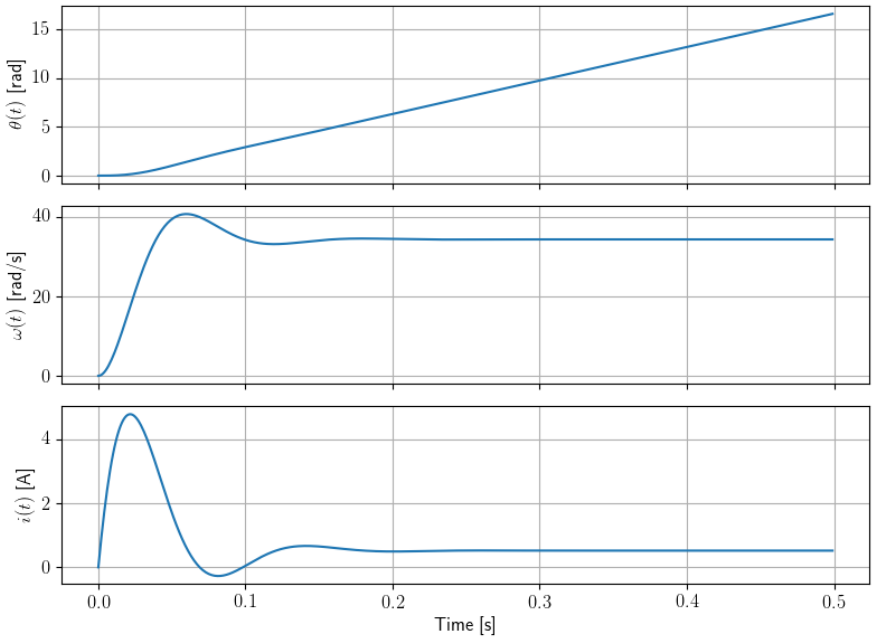

Figure 3. Step response of the DC motor states simulated in Python.

For the first approach, the **ode45** command syntax for solving the ODEs in Equation 2 or Equation 3 is:

```
[~,X]=ode45(@(t,x)motor.model(t,x,v),time,x0,options);
```

Code Snippet 7.

where **time** is the time vector for the simulation, **x0** is the initial condition for the system, options are the solver options, and model is an instance method from the user-defined **DCMotor** object named **motor** which implements the ODEs in Equation 2 as shown below:

```
function xdot = model(obj,t,x,u)
    theta = x(1); omega = x(2); i~= x(3);
    v = u; %~armature voltage
    xdot = [omega, (obj.Kt*i-obj.b*omega)/obj.J, ...
            (v-obj.Kb*omega-obj.R*i)/obj.L];
end
```

Code Snippet 8.

The **obj** refers to the instance of the DCMotor class. To implement the state-space representation as in Equation 3 , the model method is defined as:

```
function [xdot] = model(obj,t,x,u)
  A = [0 1 0; 0 -obj.b/obj.J obj.kt/obj.J; 0- obj.kb/obj.L -obj.R
      /obj.L];
  B = [0;0;1/obj.L];
  C = [1 0 0;0 1 0;0 0 1]; D = 0;
   xdot = A*x+B*u;
end
```

Code Snippet 9.

For the second approach, Octave's Control package [76] needs to be installed and then loaded. Once the package is available, the motor step response can be simulated using the following syntax:

```
ssModel = ss(A,B,C,D);
[~,~,X] = step(ssModel*u,time);
```

Code Snippet 10.

where the **ss** function creates an LTI state-space representation of the motor and the step function determines the step response of the LTI system. The simulated state vector is then stored in the variable **X**. Plotting these state variables results in a figure identical to Figure 3 .

Similar to Python's Control Systems Library, Octave's Control Package provides other system descriptions for defining transfer functions and frequency response data models. It also features other useful control system design and analysis tools. A brief summary of some of the available control-related commands in both Python and Octave are shown in Table 4.

Table 4. 4: Comparison of control-related commands in Python's Control Systems Library and Octave's Control Package

	Python	**Octave**
System Descriptions	tf, frd	tf, frd
Frequency Domain Plotting	bode_plot, nyquist_plot, nichols_plot	bode, nyquist, nichols
Time Domain Simulation	forced_response, impulse_response, initial_response, input_output_respones, phase_plot	lsim, impulse, initial, ode45, quiver
Control System Analysis	stability_margins, pzmap, root_locus, sisotool	margin, pzmap,rlocus, sisotool(not available)
Control System Synthesis	acker, h2syn, hinfsyn, lqr, place	acker, h2syn, hinfsyn, lqr, place

4.3 Modelica – Equation Mode

Using Modelica in its equation mode comes very close to just transcribing the dynamic equations and clicking "GO". Below is the content of the "equation" section of the Modelica model:

```
der(theta) = omega;
der(omega) = ((—b * omega) + Kt * i) / J;
der(i) = ((—Kb * omega) — R * i~+ v) / L;
```

Code Snippet 11.

where **der** is derivative-with-respect-to-time. Note that these are true equations, not computing statements. They can be written in any order and, within each equation, the terms can be rearranged as long as the original algebraic meaning is maintained.

These listings are from OpenModelica's Connection Editor. OpenModelica was used for this case study and all Modelica examples to follow [52].

As with any problem definition, the parameters and initial conditions must also be specified. This is also straightforward in Modelica. The parameters are given by:

```
parameter SI.Resistance R = 2.62 " Ohm";
parameter SI.Voltage v = 24.0 " v";
parameter SI.ElectricalTorqueConstant Kt = 0.66 "N.m/A";
parameter SI.ElectricalTorqueConstant Kb = Kt "N.m/A, equal to Kt
    for consistent units";
parameter SI.MomentOfInertia J = 0.0026 "kg.m^2";
parameter SI.RotationalDampingConstant b = 0.01 "N.m.s/rad";
parameter SI.Inductance L = 0.05 " H";
```

Code Snippet 12.

The **parameter** designation means that the value cannot be changed during simulation. An interesting aspect of Modelica is that the units of these parameters can be specified –SI is a shorthand for "Modelica.SIunits". Using units is optional, but very useful. If the units had not been specified, all of these parameters would have been of type "Real". The part in quotes after the value is a comment; C++, Java style of commenting, "//", can also be used. The units shown in the comment section were written in by the user and, so, must be consistent with the units as defined by Modelica. Having the ability to specify units for physical quantities is part of Modelica's multi-physical-media modeling capability.

The initial values of the state variables are specified in a similar manner:

```
SI.Angle theta(start = 0.0) "Shaft angle, rad";
SI.AngularVelocity omega(start = 0.0) "Shaft angular velocity,
   rad/s";
SI.Current i(start = 0.0) "Current in circuit, A";
```

Code Snippet 13.

The interesting variant here is that for each of the state variables, the initial condition is given as a **start** value. In this case, the problem is a pure ODE (i.e., a causal problem) so the state variables all have independent initial conditions. If the problem were a DAE (Differential-Algebraic Equation, acausal problem) the initial conditions would not all be independent so the **start** values would only be suggestions.

Results from simulating this system match the results presented above with Python. As can be seen in this section, for a problem expressed in equation form, there is essentially no "programming" needed; just copy the equations, list the parameter values, and define the system variables.

4.4 Modelica – Graphical Modeling Mode

The Modelica Standard Library contains a large set of modeling components from a variety of physical media. This is probably the most common way of creating simulations in Modelica. For the DC motor simulation, this means that components are selected from the library, they are given parameter values, and then the simulation is run. Figure 4 shows the model for the DC motor.

This is a semi-representational model in that some of the components look like their physical counterparts but others can be more abstract. The physical orientation of components on the diagram does not have any significance.

In some sense, the most important component in this diagram is the emf unit. It handles the energy conversion between electrical and rotational-mechanical domains. With the emf unit in place, this model can do proper energy/power bookkeeping across the whole system.

Each of the energy domains is identified by its connectors: blue and white squares for the electrical portion and black and white circles for the rotational section. There is an additional energy domain on the diagram, thermal, identified by red squares, but that is not used in this simulation. A right-click on any of the components will bring up a menu for its parameters. These can be entered

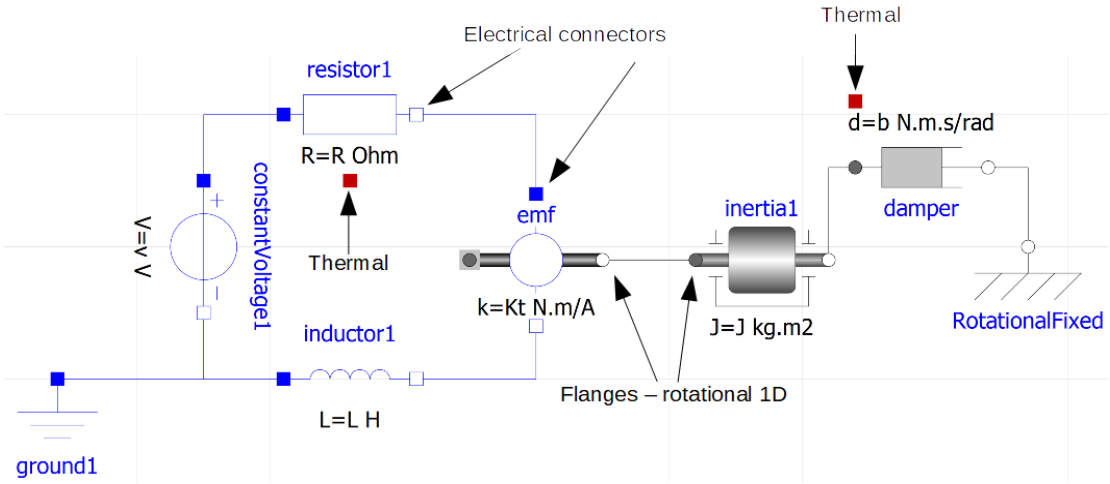

Figure 4. Modelica Graphical Multi-domain Model of the DC Motor

directly as numerical values, but it is often preferable to use named parameter variables (as above) and enter the parameter names. That is what is done in this model.

Again, simulation produces the previous results. The modeling at this high level of abstraction can be very efficient if the library contains the needed components. The Modelica Standard Library has a large selection of components so can often be used successfully for simulation of systems crossing multiple energy domains.

4.5 Java

As mentioned earlier, Java is a general purpose, object-oriented language and is thus a major abstraction level below either of the Modelica solutions. It requires much more detailed programming (real programming, not equations as in Modelica) in return for which it provides the full power of object-oriented programming and much higher speed.

General purpose languages do not usually include the specialized mathematical tools needed for dynamic system simulation. However, most of these languages have a wide variety of libraries available to fill that gap. For these examples, the Hipparchus library is used to supply the ODE solver [48]. As an aside, it is a large library and can meet many other mathematical needs in Java programming. The Hipparchus library is a successor to the Apache Commons Math Library.

As with most ODE solvers, a call-back structure is used where the derivative values of all of the state variables are computed in a function ("method" in Java) and a reference to that function is given to the solver so it can call the derivative function as needed by whatever solution algorithm is being used. In Java, the derivative method is embedded in a class which is what is given to the ODE integration system.

This is what that function looks like for the DC motor system:

```
public double[] computeDerivatives(double t, double[] y)
{
// ODE: d angle / dt = omega; d omega/dt = (torque – b omega)/J
// States: 0 – angle; 1 – omega; 2 – current
    omega = y[1]; // Angular velocity of shaft
    ic = y[2]; // Current in circuit
    vBack = Kb * omega;
    torq = Kt * ic;
    double [] dydt = new double[3];
// Construct derivatives of states
     dydt[0] = omega;
     // d theta / dt = omega
     dydt[1] = (torq – b * omega) / J;
     // d omega/dt=(torque –damping)/inertia
     dydt[2] = (v – vBack – R * ic) / L;
     // di/dt = (voltage – back–emf – R * current) / L
    return dydt;
}
```

Code Snippet 14.

There are many ways to run a simulation using ODE solvers. In this case, a structure is used that is very convenient for problems where control will be applied. In this structure, the ODE solver is called within an event loop and simulated from one event to the next. This simulation does not actually have any control, although it would be natural to add angular velocity or angular position control. The event that is present is data logging, writing the results to a file for later plotting and/or analysis. The event loop code is:

```
while(t <= tf)
{   if(t >= (tNextLog – guard))
   {
       tNextLog += dtLog;
       pw.format("%g %g %g %g %g %n", t, y[0], y[1], y[2], volt);
   }
   tNextSim = tNextLog; // Simulate until next event
   ODEStateAndDerivative finalState = dp853.integrate(ode, state,
       tNextSim);
   y = finalState.getPrimaryState();
   t = tNextSim;
   state = new ODEState(t, y); // setup for next iteration
}
```

Code Snippet 15.

Any number of events can be included in a loop like this. The simulation is run until the next event. These events are all time based. Condition-based events (for example, a state variable crossing a boundary value) are trickier and are best handled inside the solver itself, although they can be done iteratively with a structure such as this one. The **dp853** refers to the Dormand-Prince-8(5,3) ODE integration algorithm [77]. The results are written to a file and plotted using an open-source plotting tool named gnuplot [78]. As noted above, one of the advantages of Java is execution speed. By comparison, the Java version is at least 10 times faster than the Modelica-Equations version to compile and execute.

5 Summary, Conclusions, and Future Work

This article is the first part of a two-part contribution focused on promoting the use of OSS such as Python, GNU Octave, Modelica, Java, and Gazebo in MRE education. The current status, limitations, and potentials of the OSS, as surveyed among the community members, are included in the paper. One of the challenges frequently raised in the survey was lack of specialized educational materials. To this end, a DC motor, one of the most versatile components in MRE systems, is considered to showcase the application of the OSS in model simulation and analysis. Important code snippets are given to demonstrate the structure of the required programs; furthermore, the capabilities and limitations of each of these software are reviewed. The second part of this work introduces the application of the OSS in controller implementation for a slightly more complicated dynamic system, i.e., a robot manipulator. The open-source nature of the introduced software makes them an attractive simulation, design, and analysis solution for a wider range of educational institutions that aim to complement and enrich the quality of education. Furthermore, increased utilization of the OSS in industry, among their other numerous potentials, necessitates the need for further familiarizing the students with these tools. The application showcases and review of the potentials and limitations of each OSS in this work could allow MRE community to make informed judgements about what software to choose for their specific application and consequently, facilitate a wider adoption of these OSS. The complete codes for the discussed examples, along with Matlab scripts which is included as a point of comparison, are provided freely on a GitHub repository to make it accessible to the community. There are still other challenges facing the applications of the OSS in higher education, as highlighted by community survey results. More papers, similar to this work, online webinars, and short courses offered by the MRE professionals for the MRE community could help overcome these challenges and reduce adoption barriers to widespread use of the OSS in MRE higher education.

References

[1] A. M. S. Laurent, "Understanding Open Source and Free Software Licensing," and others, Ed. O'Reilly Media, Inc, 08 2004.

[2] A. Kavianpour and S. Kavianpour, "The First Course of Programming: Python, Matlab, or C?" in *ASEE Annual Conference & Exposition*, 2016.

[3] J. P. Agrawal and O. Farook, "A Case for Python Scripting in Undergraduate Engineering Technology," *ASEE Annual Conference & Exposition*, 2013.

[4] R. Raj and F. Kazemian, "Using Open Source Software in Computer Science Courses," in *36th IEEE Annual Frontiers in Education Conference (FIE)*, 2006.

[5] G. Xing, "Teaching Software Engineering Using Open Source Software," in *Proceedings of the ACM SE '10: ACM Southeast Regional Conference*, 2010. [Online]. Available: https://doi.org/10.1145/1900008.1900085

[6] P. M. Papadopoulos, I. G. Stamelos, and A. Meiszner, "Students' Perspectives on Learning Software Engineering with Open Source Projects: Lessons Learnt After Three Years of Program Operation," in *4th International Conference on Computer Supported Education (CSEDU)*, 2012.

[7] J. M. Pullen, "Low Cost Internet Synchronous Distance Education Using Open Source Software," in *2004 ASEE Annual Conference*, 2004.

[8] P. Jackson and J. Rudaitis, "A Reproducible Solution for Implementing Online Laboratory Systems Through Inexpensive and Open-source Technology,," in *2020 ASEE Virtual Annual Conference Content Access*, 2020.

[9] J. W. Paulson, G. Succi, and A. Eberlein, "An Empirical Study of Open-Source and Closed-Source Software Products," *IEEE Transactions on Software Engineering*, vol. 30, pp. 246–256, 2004.

[10] K. J. Hass and J. Su, "Modernizing the Microcontroller Laboratory with Low-cost and Open-source Tools," in *ASEE Annual Conference & Exposition*, 2012.

[11] M. A. Hopkins and A. M. Kibbe, "Open-Source Hardware in Controls Education," in *2014 ASEE Annual Conference & Exposition*, 2014.

[12] A. Gilmore, T. Daher, and M. S. Peteranetz, "A Multi-year Case Study in Blended Design: Student Experiences in a Blended, Synchronous, Distance Controls Course," in *2020 ASEE Virtual Annual Conference Content Access*, 2020.

[13] S. A. Strom and M. Strom, "Embedded Measurement and Control Applications Utilizing Python on the Pocket BeagleBone," in *2020 ASEE Virtual Annual Conference Content Access*, and others, Ed., 2020.

[14] N. Lotfi, J. A. Novosad, and H. Phan-Van, "A Multidisciplinary Course and the Corresponding Laboratory Platform Development for Teaching the Fundamentals of Advanced Autonomous Vehicles," in *2019 ASEE Annual Conference & Exposition*, 2019.

[15] R. Krauss, "Combining Raspberry Pi and Arduino to Form a Low-cost, Real-Time Autonomous Vehicle Platform," in *2016 American Control Conference*, 2016.

[16] K. A. Khan and J. Ryu, "ROS-based Control of a Manipulator Arm for Balancing a Ball on a Plate," in *2017 ASEE Annual Conference & Exposition*, 2017.

[17] A. Yousuf, C. C. Lehman, M. A. Mustafa, and M. M. Hayder, "Introducing Kinematics with Robot Operating System (ROS)," in *2015 ASEE Annual Conference & Exposition*, 2015.

[18] J. Rivera-Guillen, J. Rangel-Magdaleno, R. Romero-Troncoso, R. Osornio-Rios, and R. Guevara-Gonzalez, "An Open-Access Educational Tool for Teaching Motion Dynamics in Multi-Axis Servomotor Control," *IEEE Transactions on Education*, vol. 55, pp. 218–225, 2012.

[19] J. B. Hooker, V. Druschke, S. A. Kuhl, A. Sergeyev, S. Y. Parmar, M. B. Kinney, N. Alaraje, and M. Highum, "Enhancing Industrial Robotics Education with Open-source Software Paper," in *2017 ASEE Annual Conference & Exposition*, and others, Ed., 2017.

[20] Chitta, "ros_control: A generic and simple control framework for ROS," *Journal of Open Source Software*, vol. 2, no. 20, pp. 456–456, 2017. [Online]. Available: 10.21105/joss.00456

[21] Owan, "CoreRobotics: An object-oriented C++ library with cross-language wrappers for cross-platform robot control," *Journal of Open Source Software*, vol. 3, no. 22, pp. 489–489, 2018. [Online]. Available: 10.21105/joss.00489

[22] Lee, "DART: Dynamic Animation and Robotics Toolkit," *Journal of Open Source Software*, vol. 3, no. 22, pp. 500–500, 2018. [Online]. Available: 10.21105/joss.00500

[23] Stulp, "DmpBbo: A versatile Python/C++ library for Function Approximation, Dynamical Movement Primitives, and Black-Box Optimization," *Journal of Open Source Software*, vol. 4, no. 37, pp. 1225–1225, 2019. [Online]. Available: 10.21105/joss.01225

[24] S. Von, "Phobos: A tool for creating complex robot models," *Journal of Open Source Software*, vol. 5, no. 45, pp. 1326–1326, 2020. [Online]. Available: 10.21105/joss.01326

[25] Deray, "Manif: A micro Lie theory library for state estimation in robotics applications," *Journal of Open Source Software*, vol. 5, no. 46, pp. 1371–1371, 2020. [Online]. Available: 10.21105/joss.01371

[26] Macenski, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, pp. 2783–2783, 2021. [Online]. Available: 10.21105/joss.02783

[27] Nadeau, "Pybotics: Python Toolbox for Robotics," *Journal of Open Source Software*, vol. 4, no. 41, pp. 1738–1738, 2019. [Online]. Available: 10.21105/joss.01738

[28] Python and Org, "Python Website," 2021. [Online]. Available: https://www.python.org

[29] Stxnext Python Powerhouse, "The Most Popular Python Scientific Libraries," 2021. [Online]. Available: https://stxnext.com/blog/2017/04/12/most-popular-python-scientific-libraries/

[30] Importpython, "Books," 2021. [Online]. Available: https://importpython.com/books

[31] K. Reitz and Real Python, "The Hitchhiker's Guide to Python. Learning Python," 2021. [Online]. Available: https://docs.python-guide.org/intro/learning

[32] Massachusetts Institute of Technology, "Courseware Mit Open Courseware," 2021. [Online]. Available: https://ocw.mit.edu/index.htm

[33] Coursera, "Coursera," 2021. [Online]. Available: https://www.coursera.org

[34] Scipy.org, "Scipy," 2021. [Online]. Available: https://www.scipy.org

[35] S. Tiwari, "Python for Scientists and Engineers," 2021. [Online]. Available: https://www.pythonforengineers.com/python-for-scientists-and-engineers/

[36] The NumPy community, "NumPy for Matlab users," 2021. [Online]. Available: https://numpy.org/devdocs/user/numpy-for-matlab-users.html

[37] python-control.org, "Python Control Systems Library," 2021. [Online]. Available: https://python-control.readthedocs.io/en/0.9.0/

[38] P. Guo, "Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities," 2021. [Online]. Available: https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext

[39] H. Fangohr, M. Bubak, G. D. Van Albada, P. M. A. Sloot, and J. Dongarra, "A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering," in *Computational Science - ICCS 2004. ICCS*, vol. 3039. Springer, 2004.

[40] B. J. Furman, S. Ahsan, and E. Wertz, "Making the Move from C to Python with Mechanical Engineering Students," in *2020 ASEE Virtual Annual Conference Content Access*, and others, Ed., 2020.

[41] A. Hoyo, J. L. Guzmán, J. C. Moreno, and M. Berenguel, "Teaching Control Engineering Concepts using Open Source Tools on a Raspberry Pi Board," in *IFAC-PapersOnLine*, ser. IFAC Workshop on Internet Based Control Education IBCE15, and others, Ed., vol. 48, 2015, pp. 99–104.

[42] R. Krauss, "Real-Time Python: Recent Advances in the Raspberry Pi Plus Arduino Real-Time Control Approach," in *American Control Conference*, and others, Ed., 2020.

[43] A. Vergnaud, J. B. Fasquel, L. Autrique, and IFAC Workshop on Internet Based Control Education IBCE15, "Python Based Internet Tools in Control Education," in *IFAC-PapersOnLine*, ser. IFAC Workshop on Internet Based Control Education IBCE15, and others, Ed., vol. 48, 2015, pp. 43–48.

[44] Openjdk, "Openjdk," 2021. [Online]. Available: https://openjdk.java.net

[45] Apache, "Apache Netbeans," 2021. [Online]. Available: https://netbeans.apache.org

[46] Eclipse Foundation, "Eclipse," 2021. [Online]. Available: https://www.eclipse.org

[47] Apache, "The Apache Commons Mathematics Library," 2021. [Online]. Available: https://commons.apache.org/proper/commons-math

[48] Hipparchus.org, "Hipparchus: A Mathematics Library," 2021. [Online]. Available: https://www.hipparchus.org

[49] D. V. Schroeder, "Scientific Computing for Physical Systems," 2021. [Online]. Available: https://physics.weber.edu/schroeder/javacourse/

[50] H. Gould, J. Tobochnik, and W. Christian, "An Introduction to Computer Simulation Methods Third Edition," and others, Ed. CreateSpace Independent Publishing Platform, 09 2016. [Online]. Available: https://www.compadre.org/osp/items/detail.cfm?ID=7375

[51] "The Modelica Association," 2021. [Online]. Available: https://modelica.org/

[52] "Openmodelica," 2021. [Online]. Available: https://openmodelica.org/

[53] "Open Source Modelica Consortium," 2021. [Online]. Available: https://www.openmodelica.org/home/consortium

[54] "Modelica Courses," 2021. [Online]. Available: https://www.openmodelica.org/useresresources/modelica-courses

[55] " Modelica By Example," 2021. [Online]. Available: https://mbe.modelica.university

[56] M. Wetter and T. S. Nouidui, "Introduction to Modelica," 2015. [Online]. Available: https://simulationresearch.lbl.gov/modelica/downloads/workshops/2015-06-22-lbnl/slides/modelica-intro.pdf

[57] Open Source Robotics Foundation, "Tutorial: ROS Integration Overview," 2014. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_overview

[58] "Gazebo," 2021. [Online]. Available: http://gazebosim.org/

[59] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[60] N. Correll, R. Wing, Coleman, and D, "A One-year Introductory Robotics Curriculum for Computer Science Upperclassmen," *IEEE Transactions on Education*, vol. 56, no. 1, pp. 54–60, 2012.

[61] E. Tosello, N. Castaman, and E. Menegatti, "Using Robotics to Train Students for Industry 4.0," in *IFAC-PapersOnLine*, ser. 12th IFAC Symposium on Advances in Control Education ACE 2019, and others, Ed., vol. 52, 2019, pp. 153–158.

[62] S. Michieletto, E. Tosello, E. Pagello, and E. Menegatti, "Teaching Humanoid Robotics by Means of Human Teleoperation through RGB-D Sensors," *Robotics and Autonomous Systems*, vol. 75, pp. 671–678, 2016.

[63] E. Tosello, S. Michieletto, and E. Pagello, "Training Master Students to Program Both Virtual and Real Autonomous Robots in a Teaching Laboratory," *IEEE Global Engineering Education Conference (EDUCON)*, 2016.

[64] "Ros Tutorials," 2021. [Online]. Available: http://wiki.ros.org/ROS/Tutorials

[65] J. W. Eaton, " About GNU Octave," 2021. [Online]. Available: https://www.gnu.org/software/octave/about.html

[66] "Wiki Gnu Octave," 2021. [Online]. Available: https://wiki.octave.org/GNU_Octave_Wiki

[67] R. V. Aroca, F. Y. Watanabe, M. T. D. Vila, and A. C. Hernandes, "Mobile Robotics Integration in Introductory Undergraduate Engineering Courses," in *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, 2016.

[68] R. Balogh, “Using an Open Software for Electronics and Robotics,” in *Proceedings of the conference on Open Software in Research and Education (OSS 2011)*, and others, Ed., 2011.

[69] B. Jakubiec, “Application of Simulation Models for Programming of Robots,” *Society. Integration. Education, International Scientific Conference*, 2018.

[70] V. Pereira, V. A. Fernandes, and J. Sequeira, “Low Cost Object Sorting Robotic Arm Using Raspberry Pi,” in *IEEE Global Humanitarian Technology Conference - South Asia Satellite (GHTC-SAS)*, 2014.

[71] “Control Tutorials for Matlab and Simulink,” 2021. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?aux=Home

[72] “Advanced Motion Controls,” 2021. [Online]. Available: https://www.a-m-c.com

[73] “OSS Paper Code Repository,” 2021. [Online]. Available: https://github.com/nlotfiy/OSS-paper-scripts

[74] The SciPy Community, “API Reference: scipy.integrate.solve_ivp,” 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

[75] matplotlib, “matplotlib,” 2021. [Online]. Available: https://matplotlib.org/

[76] O. F. Community, “Computer-Aided Control System Design,” 2019. [Online]. Available: https://octave.sourceforge.io/control/index.html

[77] CS GROUP, “Dormand-Prince integrator for Ordinary Differential Equations,” 2021. [Online]. Available: https://www.hipparchus.org/apidocs/org/hipparchus/ode/nonstiff/DormandPrince853Integrator.html

[78] GNU, “ Gnuplot Homepage,” 2021. [Online]. Available: http://www.gnuplot.info