2016 7th International Conference on Intelligent Systems, Modelling and Simulation

Simulators as Drivers of Cutting Edge Research

Muhammad Adil Raja*, Sarmad Ali*, Aisha Mahmood†
*Department of Computer Science
†Department of Electrical Engineering
Namal College
Mianwali, Pakistan
{adil.raja, sarmad.ali, aisha.mahmood}@namal.edu.pk

***Abstract* — Undertaking engineering research can be compounding for beginning graduate students and thwarting even for seasoned researchers. With a wealth of academic literature and a myriad of software development and open-source computing tools available online, the life of a researcher should have become much easier. However, the overwhelming body of knowledge, that is increasing rapidly by the day, quite frequently often confuses a researcher about things that should be done. It is normal to find beginning graduate students who are clueless about how they should conduct their research. Similarly, part of the onus shifts to the academic supervisors too. They are equally oblivious about delivering worthwhile problems and adequate supervision to their students.**

This position paper is written in this wake. The motive is to highlight important ideas about how good research should be conducted in today's multidisciplinary academic world. A particular intention is to highlight the importance of employing domain-specific simulators for successful research. More specifically, it is argued that in order to conduct good quality research in various scientific and academic disciplines, it is not only important, rather inevitable to employ simulators for conducting cutting-edge research. It is hoped that taking heed from this paper would lead to better research and make research a more fulfilling and enjoyable experience for the researcher.

Keywords* — *Simulators; machine learning; software integration;

I. Introduction

As students enrol in graduate school in order to advance in their academic and professional careers, most of them can be clueless about what they would really like to do in life. This is normally true about most students, irrespective of where they come from. And this behaviour can be verified with personal observation as well as a review of academic literature [1]. Add to this the plight of a student who enrols in graduate school or a research facility to undertake research as a professional career choice. It is quite true that at least during the first year of developing a thorough research experience, a typical student can be quite oblivious of what he (or she) really wants to accomplish during his tenure as a researcher. Normally, this manifests itself either as indifference about personal career objectives and desires or too much overwork and frustration in the hope of getting somewhere at some point in future.

Consider in addition the enormous amount of academic literature that is present online and that is published regularly. Massive academic literature exists online on almost any conceivable scholarly discipline. As a starting student is exposed to the overwhelming amount of literature, it simply adds to the commotion of an already mused mind.

The intent behind writing this paper is to highlight the importance of employing domain-specific simulators of real-world phenomena that may help in developing more robust and reliable systems. This paper also argues that employing simulators in certain feats of research and development can help in shortening the system development life-cycle. Moreover, it is argued that employing simulators would also help a starting researcher to clearly envision his research questions and their solutions. Knowledge about why simulators should be employed in certain types of problems, and how they can be helpful can help the starting researcher to easily crunch his problems without wasting large amounts of valuable time.

The rest of this paper is organized as follows. Section II presents reasons for why using simulators has become a tradition among computer science students. In section III, it is argued that why simulators should be employed in modelling various real-world phenomena. Section IV states examples of how simulators can be used for inventing cutting-edge technologies. Section V presents some concerns about software automation and integration that ensue from employing simulators to develop solutions for real-world problems. Finally, section VI concludes the paper.

II. Of Simulators and Escapism

A decade ago a student of computer science would enrol in a course on communication networks not primarily because of a keen interest in the subject matter, but also in the hope of escaping from the vast amount of programming exercises and projects he would have to undertake in any other course, such as operating systems or an advance course on programming. Various courses in computer networks offer the liberties of passively and verbally brain storming ideas and solving simple numericals for computing traffic flow rates and congestion control. The student could strive for a good grade due to the format of the examination. Normally a well designed curriculum requires students to indulge in a practical hands-on software development part. As students of communication networks, this requirement could easily be evaded by simulating a real-world scenario pertaining to communication networks. So a student confronted with the challenge of completing course project could simply download a network simulator, perform a bunch of simulations about a network topology or a scenario and get away with it by writing a report about the whole experience. NS-2 [2] and Opnet [3] used to be favourite candidates to fulfil these requirements.

This tendency to evade programming exists among our students today as well. However, they are not confined to enrolling in a course on communications networks simply due to the possibility of exploiting the power of a couple of simulators. The reality is that powerful simulators exist for almost any conceivable engineering or scientific feat now. This implies that students have many optional courses to enrol in order to evade the necessity to program. However, these are not the reasons for which one should adopt simulators. Moreover, these are not the reasons for which this paper has been written.

III. A Rationale for Adopting Simulators

Advanced simulation software and frameworks exist for many scientific and engineering disciplines nowadays. There are certain advantages that sophisticated simulators can lend which would be unattainable if systems or technological frameworks were built without them. Traditionally, most of the engineering feats have required specialized expertise on the part of the practitioner. An engineer spends years in learning a particular set of skills before delivering the engineering marvel he has been dreaming to develop for so long. The manufacturing process, whether automated or requiring manual labour, has to address all the underpinnings of the successful development of the technology under consideration.

The recent advances in computing technologies have changed this. The arrival of cloud computing technologies, coupled with advancements in computing hardware, have begun to enable the realization of the numerous fantastic engineering gadgets that were possible only theoretically previously.

Consider the development of panels for solar energy systems as an example. The solar energy panels were developed previously by domain-experts, mostly electrical engineers, according to a certain product specification which addressed the electrical and physical properties of the panel to be developed. At the very best an electrical engineer had access to a simulation software that could be used to simulate a solar panel of desired specification. If the simulated panel fulfilled the design and development criteria set by the specifications and expected by the end users, the design would be sent to the factory for fabrication. Designing such panels would nonetheless be a black-art requiring a lot of expertise on the part of the electrical engineer responsible for the whole project.

Current advances in computing technologies have changed this trend. Given the vast processing power an average cloud computing platform offers, simulators can be designed and developed that allow the target technology to be emulated to best match the target specifications. Moreover, if the simulator is augmented with an appropriate machine learning or artificial intelligence algorithm, the process of developing highly innovative products and technologies can be entirely automated and sped up.

As a matter of fact, it is the ability to include a machine learning algorithm in the product development life-cycle that basically automates the whole process and makes human intervention completely dispensable.

A brief reflection on machine learning is in order. Machine learning is a sub-field of artificial intelligence in which a computing machine is expected to learn to find solutions to a user-specified problem all by itself with minimal human intervention. In quite layman terms, this simply implies that a machine is provided with a specification of the problem, along with relevant data, and it is expected to find a solution for the problem all by itself. Machine learning algorithms come in all hues and colors and are quite well-known for automated problem solving. Machine learning algorithms can be categorized according to how they work or according to the analogues from real-life systems from which they are adapted. Thus, algorithms that are inspired from biological systems can be broadly termed as bio-inspired algorithms. Similarly, algorithms that are inspired from biological evolutionary systems are known as evolutionary algorithms.

Inclusion of machine learning algorithms in the overall software development life-cycle has actually helped from various vantage points. Firstly, as stated earlier, the overall development life-cycle can be automated, making human intervention mostly unnecessary. Secondly, as the development life-cycle is mostly automated, workable prototypes of desired products can be produced with a lot more agility. Thirdly, despite the fact that machine learning algorithms are known for producing counter-intuitive models, achieved models are often a lot more efficient than what could have been achieved by a hand-crafted counterpart.

Machine learning algorithms should, nonetheless, work with simulators or emulators. The reason for this is that as machine learning algorithms make progress in finding an optimal solution for the problem at hand, they do so by making errors. As a matter of fact, machine learning algorithms have to be allowed to make errors as they proceed to find optimal solutions. There are obvious reasons for this that can be read elsewhere in the literature. One of the reasons that machine learning algorithms should be allowed to make errors in pursuit of finding optimal solutions is that by making errors they tend to escape the regions of sub-optimal solutions and move to more optimal regions of the solution space.

Given this feature of machine learning algorithms to make errors in pursuit of finding more optimal solutions by escaping the regions of local optimum solutions, it becomes prohibitive to employ them to real systems to find solutions. A nice example can be taken from the general domain of vehicle routing problems [4]. A machine learning algorithm can be used to address a vehicle routing problem in which the algorithm trains a vehicle to successfully traverse a path by fulfilling certain user specified objectives. In finding the optimal solution, a typical algorithm will normally generate a large number of candidate solutions randomly and improve them (or their successors) iteratively over time or several generations. Since the initial solutions are chosen randomly, they can be expected to be erroneous, having a high probability that the vehicle they govern will crash as it tries to cruise to the destination. It is only due to the guided nature of the underlying search algorithm that solutions begin to improve after a certain number of iterations and a routing pattern begins to emerge that can lead the vehicles to the desired destinations successfully.

Most machine learning algorithms work in this way. Moreover, it is more true about the general class of meta-heuristic algorithms, which deliberately allow the solutions to err in pursuit of finding the ideal solution. Evolutionary algorithm and, generally, all of the bio-inspired algorithms function like this. As a matter of fact, their power of finding befitting solutions for intricate problems ensues from their ability to deliberately err as they pursue to find the ideal solution. Evolutionary algorithms, for instance, work by generating a large population of candidate solutions to a problem. The solutions are evaluated and then recombined using evolutionary search operators to create offspring. It is due to the very nature of the evolutionary operators that a diversity of offspring is induced in the solution space. The offspring are anticipated to deviate from their parents in terms of behaviour, which manifests itself as offspring being generated on various other parts of the solution space that were not occupied by their parents. Due to this the offspring become able to traverse those parts of the search space that were unexplored by their parents. The newly discovered parts of the search space may have better or worse solutions in them depending on the very nature of the latter. The offspring that do better than their parents are kept for evolutionary recombination for the latter generations. The offspring that perform poorly are littered. This is how training normally happens in machine learning in general, and in evolutionary algorithms and other meta-heuristic algorithms in particular.

Now assume what would happen if such an algorithm was run to address the vehicle routing problem using real vehicles. Clearly, as erroneous solutions are deliberately produced and tested with real vehicles, quite a lot of them would end up in catastrophic crashes. This can make the whole enterprise of solving vehicle routing problems unaffordable.

Even if the deliberate creation of erroneous solutions were not a problem, finding an optimal solution by employing real vehicles would still be prohibitive. Consider that an evolutionary algorithm is employed in finding a solution to route a vehicle from one place to another. Such an algorithm normally begins with creating a large population of individuals as candidate solutions for the optimal route. The population size is normally maintained during the course of an evolutionary run, spanning the number of generations required to complete a typical evolutionary process in an evolutionary algorithm. Each of these solutions would in turn have to be evaluated for its feasibility. If real vehicles are employed, the time it would take to evaluate a solution to see whether or not it routes a vehicle successfully, by fulfilling all the intermediate constraints, to its destination would be quite prohibitive even if the vehicle did not crash on its way to the destination.

Evolutionary algorithms are quite compute intensive due to their very nature of requiring to maintain and manipulate a large population of individual solutions and their offspring over the course of evolutionary search. Consider employing a more efficient algorithm such as simulated annealing and/or gradient descent, as in the case of back propagation networks. As these algorithms run and create intermediate solutions to find the optimal solution, they are also expected to be able to test the viability of those intermediate solutions. In case of the vehicle routing problem, if real vehicles are employed to test the viability of these solutions, the time required for evaluation would still become prohibitive for these algorithms to run effectively simply for the reasons mentioned above.

A conclusion from the aforementioned discourse clearly requires an alternate means to train and test models for vehicle routing. The only viable option left to ponder about as a solution is to find and employ a realistic simulator for vehicles to be trained. If a well designed simulator for vehicles and the environments on which they are run could be dovetailed with an appropriate machine learning algorithm, the process of discovery of solutions can be simplified without incurring huge costs. Once a desirable solution to the problem at hand is found using the simulator, it may then be ported to the real system, which is a vehicle in this case.

A. *Simulators for Other Feats*

The rationale for employing simulators can be considered to hold for many other feats. Consider the problem of discovering an optimal design for a solar energy panel. Clearly, one approach could be to design a new panel from scratch by hand. Such a design would naturally be a function of the whims and caprices of the designer. Another approach could be to employ a suitable machine learning algorithm to do the job of design discovery.

In adopting this second approach, a natural tendency could be to somehow employ real hardware that should nonetheless emerge into a novel design. This approach can be quite prohibitive due to the reasons mentioned above. Apart from that, take into account the fact that machine learning algorithm does not know a priori as to how to get a final design into order. The steps required for intermediate assembly of various parts of panels as the final design evolves can not only be very expensive, they are quite difficult even to conceive.

A more viable approach could be to find a way to simulate solar panels in software. The simulator should have the ability to draw a design of a panel with desired electrical and physical properties. Such simulators are not uncommon these days. The only thing that needs to be done is to dovetail the simulator with a suitable machine learning algorithm. The algorithm may then discover a nice design for a solar panel with wonderful properties that a human being could never have conceived otherwise. The designed panel can finally be developed using a suitable manufacturing process.

This approach has gained much popularity nowadays. There are many accounts in the academic literature in which machine learning algorithms, and especially evolutionary algorithms, have been employed for discovery of novel designs of various systems and processes [5]. Some examples are quoted in section IV.

B. How Good is the Simulated Stuff

Once we realize that simulation can be a nice alternative for novelty detection, a natural question arises that how good are the objects that have been simulated? This is to ask that how well do the simulated systems function in the real world? A simple and straight-forward answer to this question is that it all depends on how good are the simulators in terms of how accurately they model the real world. This is to say that the performance accuracy of the simulated systems depends on how exactly the simulators, that have been used to discover them, model the real world.

In the case of the vehicle routing problem, a vehicular system should model everything as perfectly as possible to mimic the real world. This means that it should have realistic models of various vehicles. Simulated vehicles should function as accurately as possible as its real counterpart. Its engine, throttle, gearbox, brakes and accelerator should be modelled as perfectly as possible with their real counterpart.

Moreover, the environment of the vehicle should also be modelled as perfectly as possible. The roads, bends, curves should be modelled as perfectly as possible to represent real-world scenarios. Along with that, other features such as road friction, aerial drag, obstacles, behaviour of other vehicles on the road, behaviour of the vehicle while reacting to stimuli, such as applications of brakes, or appearance of an obstacle should be proportional and modelled as accurately as possible to the real-world scenarios.

In certain scenarios, simulated phenomena turns out to be a better choice than employing the actual phenomena. This can lead to more accurate analysis of a system under test. Consider the problem of modelling the evolutionary dynamics of a biological ecosystem. What is now known about evolution cannot be observed together in most real world scenarios. Moreover, it is difficult to keep track of all the dynamics of a real ecosystem at various time scales and different landmasses. The inherent randomness and whimsical behaviour on the part of the living species can lead to a lot of obscurantism. The randomness could be about how various species interact with each other, as well as their ecosystem. Once a certain behaviour is observed in certain species at a particular point in time, it might be quite difficult to replicate or be observed again. This could simply happen because the species want to behave in altogether different ways from that point onwards.

However, if various species could be modelled and simulated in an ecosystem modelling tool, studying ecosystem dynamics could be simplified. The inherent randomness, whims and caprices of the species under observation would have to be simulated in software. Moreover, the simulation software could also address the complex interactions of species with each other as well as with their environment. How evolutionary dynamics proceed as a function of such interactions could then be modelled with the simulator.

Such simulators are becoming common by the day. Advances in computing power is making things possible that were only limited to the imaginations of scientists in the past. To this end, EcoSim [6] is a novel simulator that models the ecosystem dynamics by employing a predator and prey model, along with more traditional elements of a typical ecosystem modelling system. The answer to the question that how good is the simulated stuff, again, is that how accurately the simulator has modelled the real world.

Another example of where it would be beneficial to employ a simulator instead of a real system is the modelling of various aspects of communications networks. In certain scenarios, it would be better to employ a simulator as opposed to a real network to make measurements about a network. One such scenario is the modelling of end-to-end delay, jitter and packet loss induced by a packet network on Voice over Internet Protocol (VoIP). While measuring these metrics, a natural tendency could be to employ a real network by initiating a VoIP call from one part of the world and storing the packets being sent to a remote place. Both parts could be geographically far apart and contain a complex heterogeneous network along all paths of the call traffic.

However, employing a real network could be a bottleneck due to various reasons. For one thing, given the timing of the analysis and some bad luck, the modeller might not be able to accumulate enough data about a particular type of desired network behaviour. For instance, although the underlying network is widely spread and has all sorts of problems inherent in it that are worth a scrutiny, the observed traffic might not be reflective of all sorts of possible scenarios the analyst wants to model. Consider the problem of observing the quality of the VoIP call as the function of end-to-end packet delay and packet loss. Even if the network had the ability to exhibit all kinds of possible scenarios, it might become prohibitive for the modeller to observe and analyse a particular type of behaviour. This is to say that, it might become quite time consuming, and even impossible for the modeller, to gather enough traffic about a VoIP call that exhibits end to end delay and/or packet loss in a certain desirable range of values. Employing a realistic simulator in such scenarios could be a better option. The simulator could emulate all types of realistic scenarios emerging in a typical communications network, and the analyst could perform calculations using that.

From the above discourse, it is apparent that time is another compelling reason for choosing a simulator to study VoIP call behaviour or network traffic analysis. By employing a good simulator, the analyst could save time on performing the desirable calculations, which he would have wasted in getting to grips with a real network. An example of where a simulator was applied successfully to emulate VoIP network traffic is given in [7].

Fortunately, as stated earlier, such simulators are not uncommon nowadays. And as work continues to happen in the fields of artificial reality and agent-based modelling systems, such simulation systems are becoming more sophisticated by the day.

This advocacy in favour of using simulators, however, is not to suggest that real systems be totally neglected or abandoned. A scrutiny of real systems is, nonetheless, important to gather information about how they work. Realistic expectations of how various real systems work, or are expected to work, is only possible after a careful analysis of them. Where it becomes impossible to leverage from the properties of real systems to design, enhance or model solutions for them, simulators should be developed and employed.

IV. Innovation Through Simulation

This section presents brief details of a few recent innovations that happened with simulation studies.

A. Unmanned Aerial Vehicles

Considerable work has been done in the area of developing autonomous unmanned aerial vehicles. Most of the achievements have been made by employing a simulation-based methodology. The UAV market is expected to grow tremendously by the next decade. Moreover, it is anticipated that the level of autonomy reach by UAV systems shall also become quite sophisticated. Work on developing fully autonomous, self-coordinating UAV is already under way, and the field is expected to become quite mature by the passage of time.

One of the auspicious aspects of developing this type of sophisticated technology is that novel models for fully autonomous and self-coordinating fleets of UAVs is accomplished through simulation studies. Cited literature reports use of simulation software such as FlightGear [8] and X-plane [9] for development. Moreover, such simulators can be dovetailed with machine learning algorithms such as genetic algorithms [10], genetic programming [11], their variants, gene expression programming, differential evolution, artificial neural networks, simulated annealing, particle swarm optimization and ant colony optimization [12]. FlightGear is also capable of hardware in the loop simulations. A hardware in the loop simulation simply means that a precise mathematical representation of the system under development is incorporated in the simulation environment. It also has access to precise electrical emulation of sensors and actuators [13].

Dovetailing flight simulation software with such machine learning algorithm automates the process of novelty detection for finding optimal sets of controllers for UAVs that could fly them successfully in unprecedented environments and scenarios.

B. Solar Energy Systems

Machine learning algorithms have been applied for finding solutions for various problems in solar energy systems. Various types of problems arise in this discipline and each one of them is addressed differently. However, the use of appropriate simulators dovetailed with appropriate machine learning algorithms is quite commonplace in academic literature.

One such problem is to innovate the design of solar panels so as to find more energy efficient solutions with a longer life span. In this regard, two examples are particularly noteworthy.

Jeffrey Grossman, a theoretical physicist at MIT, conceived the idea that if solar panels had three dimensional (3-D) shapes, such as the shapes of trees, they could be more effective in terms of producing energy than the currently existing two dimensional (2-D), flat solar panels. In order to investigate this, the scientists employed a genetic algorithm to design solar panels in a computer simulation. Their scheme randomly generated jumbles of flat, double-sided solar panels. An analysis was followed to figure out which structures generated the most power as a virtual sun moved across the sky. The best performers were then genetically recombined to create offspring. The process was repeated for millions of generations, in the hope of evolving better structures. The 3-D solar panels were as easy to implement as flat ones, while generating more energy [14].

In a seemingly similar approach, researchers have tried to evolve the design of organic, flat solar panels [15]. In the problem they have addressed, a polymeric solar panel is made of two layers of a polymeric material. One of them is an absorption layer, and the other a scattering layer. The scattering layer sits on top of the absorption layer, and is exposed to direct sunlight. The absorption layer is responsible for absorbing photons from the sunlight and use them to generate electricity, as in traditional solar panels. The scattering layer, on the other hand, is responsible for gathering the sunlight in itself and scattering it to the absorption layer beneath it steadily.

The researchers were confronted with the problem of finding an optimal design for the scattering layer. The design criteria were driven by the performance expected by the layer. The problem was to find an optimal design for the scattering layer, whereby the layer could absorb as much sunlight as possible, and disseminate it to the absorption layer, below it, slowly and steadily so that the latter would not get depleted of sunlight during the day, or much after sunset. Clearly, any person would be clueless of what would such an optimal design look like. Feeding their design criteria to their simulators, the researchers found the optimal design by employing a genetic algorithm.

C. Social and Ecosystem Modelling

Like many other disciplines, modelling the complexities of societies as well as ecosystems is also becoming common. A few examples were given in the previous section. Integrating social and ecosystem simulators with machine learning algorithms such as variants of evolutionary algorithms has also become quite common [6], [16].

D. Vehicle Routing

Vehicle routing is considered a formidable problem in computer science. Developing solutions for smart and intelligent vehicular systems is becoming a vogue. Several simulators exist for vehicular systems. Among these, TORCS is possibly the most well known [4]. Numerous examples exist in open literature that describe how TORCS was dovetailed with other machine learning algorithms to evolve controllers for racing cars [17]. Like FlightGear, TORCS is also capable of simulating hardware in the loop.

V. Software Dovetailing Issues

As the technical underpinnings a typical simulator addresses widen, it becomes an issue to provision simulation systems with appropriate hardware resources and software platforms to run on. With the current advancement of cloud computing technologies, such problems also tend to become trivial. Nowadays it is quite easy to deploy a private cloud on-campus, which may in turn be used for running quite compute-intensive simulators. The simulator that would take a long time to load on the single CPU, may now be run on a cloud. The underlying cloud may in turn distribute various parts of the simulator on its underlying hardware systems, and run it effectively. To this end, OpenStack suddenly comes to the mind. OpenStack is a new cloud operating system that is also much hyped. Its hype is also quite justified. For one thing, it is open-source, meaning that it can be deployed, used and altered freely. Moreover, it comes along with a suite of tools which make the process of software deployment and operation quite easy. Juju, for instance, is one such set of tools that is responsible for service orchestration on the cloud. Juju comes with its charms, which are small computer programs, that address the problems of deploying software and services on an OpenStack cloud.

A major problem that needs to be addressed is how to dovetail domain-specific simulators with various machine learning algorithms. It would be ideal to come up with a general application integration framework that accepts a pair of a randomly chosen simulator and a machine learning algorithm and dovetails them. Integration should be such that the machine learning algorithm should be able to invoke various objects in the simulator, perform computations on them and adapt its states according to the performance of those objects. As various machine learning algorithms implement some sort of an iterative mechanism, a natural requirement could be to have a means of data transfer between the pair of applications. The data transfer can be required to test the solutions generated by the learning algorithm using the simulator. So, a repetitive data transfer back and forth between the two applications should be naturally sought.

There are other objectives a nice dovetailing application should also address. For instance, it would be quite worthwhile if the dovetailing application can exploit the computing power of a distributed computing environment, such as a cloud. Such issues would have to be taken into account while developing the dovetailing application, and should be included in its software design issues to be addressed.

At present such dovetailing applications do not exist that integrate a randomly chosen pair of a machine learning algorithm and a domain-specific simulator. As a matter of fact, it is quite hard to find a specific application online that works for a given pair of such applications. Authors tried to search the world wide web but failed to find one.

VI. Conclusion

This position paper argues about the viability of employing simulators for cutting-edge research. Recent advancements in computing systems are opening endless possibilities to simulate real-world phenomena that were traditionally only possible in human imagination and thought experiments. With the sophistication that is being achieved by simulation frameworks, it is now more convenient to simulate various types of problems. The real world artefacts can be created afterwards, after simulations have succeeded. This is not only supposed to reduce production costs, but a handful of other benefits are conceived and presented.

References

[1] G. Graff, *Clueless in academe: How schooling obscures the life of the mind.* Yale University Press, 2008.

[2] S. McCanne, S. Floyd, K. Fall, K. Varadhan *et al.*, "Network simulator ns-2," 1997.

[3] X. Chang, "Network simulations with opnet," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1.* ACM, 1999, pp. 307–314.

[4] B. Wymann, E. Espi, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," 2013.

[5] D. Pelusi, "Pid and intelligent controllers for optimal timing performances of industrial actuators," *International Journal of Simulation: Systems, Science and Technology*, vol. 13, no. 2, pp. 65–71, 2012.

[6] R. Gras, A. Golestani, A. P. Hendry, and M. E. Cristescu, "Speciation without pre-defined fitness functions," *PLoS ONE*, vol. 10, 09 2015.

[7] A. Raja, R. M. A. Azad, C. Flanagan, and C. Ryan, "A methodology for deriving VoIP equipment impairment factors for a mixed NB/WB context," *IEEE Transactions on Multimedia*, vol. 10, no. 6, pp. 1046–1058, Oct. 2008.

[8] B. Michael, S. Martin, and B. Stuart, "The flightgear manual," 2014.

[9] A. Bittar, N. M. F. de Oliveira, and H. V. de Figueiredo, "Hardware-in-the-loop simulation with x-plane of attitude control of a suav exploring atmospheric conditions," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 271–287, 2014.

[10] G. Varela, P. Caamaño, F. Orjales, Á. Deibe, F. López-Peña, and R. J. Duro, "Autonomous uav based search operations using constrained sampling evolutionary algorithms," *Neurocomputing*, vol. 132, pp. 54–67, 2014.

[11] I. Buzhinsky, D. Chivilikhin, V. Ulyantsev, and F. Tsarev, "Improving the quality of supervised finite-state machine construction using real-valued variables," in *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion.* ACM, 2014, pp. 1037–1040.

[12] H. Duan and P. Li, *Bio-inspired computation in unmanned aerial vehicles.* Springer, 2014.

[13] H.-P. Halvorsen, "Introduction to hardware-in-the-loop simulation," *Telemark University College*, 2012.

[14] M. Bernardi, N. Ferralis, J. H. Wan, R. Villalon, and J. C. Grossman, "Solar energy generation in three dimensions," *Energy Environ. Sci.*, vol. 5, pp. 6880–6884, 2012. [Online]. Available: http://dx.doi.org/10.1039/C2EE21170J

[15] C. Wang, S. Yu, W. Chen, and C. Sun, "Highly efficient light-trapping structure design inspired by natural evolution," *Scientific reports*, vol. 3, 2013.

[16] C. Cioffi-Revilla, "A methodology for complex social simulations," *Journal of Artificial Societies and Social Simulation*, vol. 13, no. 1, p. 7, 2010.

[17] M. Hansen-Schwartz, "Evolving race car drivers using grammatical evolution."