Preprints of the 2012 20th Mediterranean Conference on Control & Automation (MED), Barcelona, Spain, July 3-6, 2012

AVLib: A Simulink® Library for Multi-Agent Systems Research

Damjan Miklic, Stjepan Bogdan, and Rafael Fierro

Abstract— **This paper presents a library of Simulink blocks, aimed at facilitating research and education in the area of multi-vehicle systems. The inherent complexity involved in modeling, simulating and experimenting with multi-vehicle systems presents significant challenges for researchers, educators and students. There is a need for re-usable building blocks implementing standard vehicle models and control laws, as well as convenient visualization components and interfaces. Furthermore, well-designed system building blocks make the transition from simulation to actual hardware experiments as seamless as possible. The described library, called AVLib, provides such tools, implemented as a set of Simulink blocks that are easy to use, can be extended with other Simulink blocks, and are scalable to model systems comprised of several hundreds of agents. We present several case studies showing how the AVLib library can facilitate research of multi-vehicle systems.**

I. INTRODUCTION

A group of autonomous vehicles, or agents, acting in a coordinated effort towards the completion of a common goal, can by far outperform a single unit. Some of the potential advantages include increased fault tolerance, greater area coverage, distributed sensing and coordinated manipulation of large objects. Due to the obvious advantages and numerous potential application areas, ranging from search and rescue missions to warehouse automation, coordination and control mechanisms for multi-agent systems have been an active research area for over a decade [1]-[2]. The enhanced capabilities of multi-agent systems do not come without a price and the control problems related to such systems exhibit a new dimension of complexity. The fundamental issues of system stability, convergence to the desired state and robustness must be considered. High system dimensionality, complex interactions, inherent parallelism and uncertainties make analysis and control synthesis a challenging task, therefore, simulation is an indispensable tool in multi-agent research. A major requirement on any simulation tool used for multi-agent systems is *scalability*, as the number of simulated entities can range from only a few, to several hundreds of units. Furthermore, as control and coordination algorithms should ultimately be experimentally verified on actual hardware, it is important to enable the transition between simulations and experiments with only minimal changes to the controller code.

S. Bogdan and D. Miklic are with Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia {damjan.miklic,stjepan.bogdan}@fer.hr

R. Fierro is with the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131, USA rfierro@ece.unm.edu

Today's robotics researchers have a wide array of simulation tools at their disposal [3] - [4]. A good overview and comparison of existing tools can be found in [5]. Simulators range from simpler 2D tools aimed at simulating a larger number of robots, such as Stage [3], to full-blown 3D environment simulators with high-fidelity physical engines such as USARSim [6]. Some are commercial software packages that include a development environment and scripting or graphical tools for robot programming, for example Webots [4] and Microsoft Robotics Develper Studio [7], while others are open source projects, such as Gazebo [8]. Most of the aforementioned tools require a significant amount of programming proficiency in order to implement robot controllers. Robotics Toolbox [9] and iCreate Toolbox [10] are Matlab®-based packages for simulation and controller development. The former is a well-established education tool, but more oriented towards manipulators, with a small number of vehicle models, while the latter is aimed at controller development for the iRobot Create® mobile base.

In spite of the relative abundance of simulation and development tools, researchers still often resort to developing their own custom made models and controllers. Many roboticists, at least in the initial prototyping phase, prefer working with Matlab/Simulink, a tool they are familiar and efficient with. To the best of the authors' knowledge, there is currently no publicly available toolbox for Matlab/Simulink offering a wide range of models and controllers that are standardly used in research on multi-vehicle systems. As a result, researchers often end up "re-inventing the wheel", i.e., re-implementing standard components. In addition to providing a set of standardized building blocks, such as vehicle models and commonly used control schemes, a library should provide several other components in order to be useful as a research and teaching tool. It should be easily extensible, scalable to systems comprised of hundreds of units, and it should make the transition from simulations to experiments as seamless as possible. In this paper, we describe AVLib, a library of Simulink blocks, whose goal is to alleviate the above needs.

The paper is organized as follows. In Section II we discuss the challenges in modeling, simulating and experimenting with multi-agent systems, and outline the requirements for an useful simulation tool. The design of AVLib is described in Section III. In Section IV, we present four case studies of conducted research projects that benefited from the AVLib library. Concluding remarks are given in Section V.

978-1-4673-2529-5/12/$31.00 ©2012 IEEE

II. Implementation challenges in multi-vehicle systems research

High dimensionality, complex interactions, parallelism of actions and uncertainties are some inherent properties of multi-agent systems. These properties make multi-agent and multi-vehicle systems an interesting research topic, and at the same time they pose significant implementation challenges. A library of software components should have the following properties, in order to be a useful tool for multi-vehicle system research:

- modularity and re-usability,
- extensibility,
- scalability,
- seamless transition between simulations and experiments.

Multi-agent systems are comprised of a large number of individual units, interconnected by a communication and control structure[1], where each unit is an instance from a limited set of basic models. The first and foremost requirement on any software library dealing with multi-agent systems is to provide a collection of such basic models that can be re-used and combined into more complex structure. Having these standard building blocks at hand, the user can focus on building his system by interconnecting the available components. Extending the system with new elements, e.g., vehicle models or control laws, should be straightforward, by connecting components through standardized, intuitive interfaces.

Closely linked to the modularity and re-usability requirements is the issue of scalability, which needs to be taken into account from two aspects. On one hand, it should be easy for the user to scale his simulation by one, or even more orders of magnitude, without making major changes in system structure. For example, having set up a simulation with 15 units, extending it to 150 or 300 units should require only changing system parameters (e.g., providing additional initial conditions and constraints), without making any changes to its structure. On the other hand, increasing the number of simulated units must be implemented in such a way, that it has the smallest possible impact on performance.

Finally, in most cases, experiments on real hardware are an essential part of research validation, and simulations are just a stepping stone used for initial proof-of-concept validation and algorithm debugging. Therefore, the ability to transfer control algorithms from simulation to experiment with only minimal modifications is of paramount importance. Ideally, it should be possible to simply remove the models of physical entities (such as vehicles), and replace them by interfaces to actual hardware.

To address all of the above requirements, Simulink was chosen as the platform for developing AVLib. It is a tool well-known to a wide audience of researchers, because its use is part of standard curricula in most control-oriented courses. Due to its intuitive graphical nature and versatility, it is commonly used for modeling and prototyping a wide range of engineering problems. The AVLib block library extends existing Simulink libraries with a set of standard vehicle models, commonly used controller structures, visualization and environment modeling tools, and a set of interfaces towards external software and hardware, that enable advanced visualization capabilities and the inclusion of hardware-in-the loop.

[1]This structure can be physical, implying a physically interconnected system, but most of the time the connection exists only at the level of information exchange, such as a wireless communication link, or sensory information.

III. Simulink Library Design

The AVLib block library implements five basic groups of blocks for prototyping multi-vehicle systems: utility blocks, vehicle models, control algorithms, environment modeling and visualization and external interfaces. The first group, utility blocks, are provided as a convenience for performing commonly used operations, such as angle wrapping and coordinate transforms. The other four groups are described in more detail in the rest of this section. Throughout the library, extensive use is made of the concept of *vectorization* [11], in order to make the models easily scalable and numerically as efficient as possible.

A. Vehicle models

The basic building blocks of any multi-vehicle scenario are vehicle models. Ranging from simple first-order kinematics, to more complex models that include vehicle mass and moment of inertia, a library of efficiently implemented and tested models can significantly speed up system development. Standardized interfaces make it straightforward to test algorithms with different vehicle types. Several models are currently implemented within the AVLib library, such as first order unicycles, second order unicycles, Ackermann-steered vehicles, quadrotor and blimps. First order unicycles (1) are the most elementary models, commonly used in developing multi-vehicle coordination mechanisms because of their simplicity:

$$\begin{aligned} \dot{x}_n &= v_n \cos\theta_n, \\ \dot{y}_n &= v_n \sin\theta_n, \\ \dot{\theta}_n &= \omega_n, \end{aligned} \tag{1}$$

where the n-th vehicle pose in two dimensions is given by the $(x_n,\ y_n,\ \theta_n) \in SE(2)$ triplet, while linear and angular speed $(v_n,\ \omega_n) \in U$ represent system inputs. Second order unicycles extend the first order model by also considering vehicle mass m_n and moment of inertia J_n, and making the force f_n and torque τ_n system inputs:

$$\begin{aligned} \dot{v}_n &= \frac{1}{m_n} f_n, \\ \dot{\omega}_n &= \frac{1}{J_n} \tau_n. \end{aligned}$$

Ackermann-steered vehicles are implemented according to the kinematic models described in [12]. The model used for quadrotors is described in [13], while blimps are modeled after [14]. Currently, the ground vehicle models are vectorized,

meaning that a single block can represent an arbitrary number of vehicles, while UAV models still require one block per unit. Vectorization of UAV models is currently under way.

B. Control algorithms

The library currently contains a limited set of control algorithms, directly applicable to vehicles with unicycle kinematics and holonomic vehicles (e.g., quadrotors). The following elementary behaviors are supported:

- Point-to-point motion,
- Random walk,
- Leader-following.

The point-to-point controller, based on feedback linearization, enables a vehicle to be guided through a set of waypoints. By properly choosing the waypoints, the vehicle can navigate even through a cluttered environment. The random walk controller emulates Brownian motion by changing the vehicle's speed and heading in random time intervals, according to a user-defined distribution (normal, uniform or exponential). Such behavior can be useful for instance in searching scenarios [15].

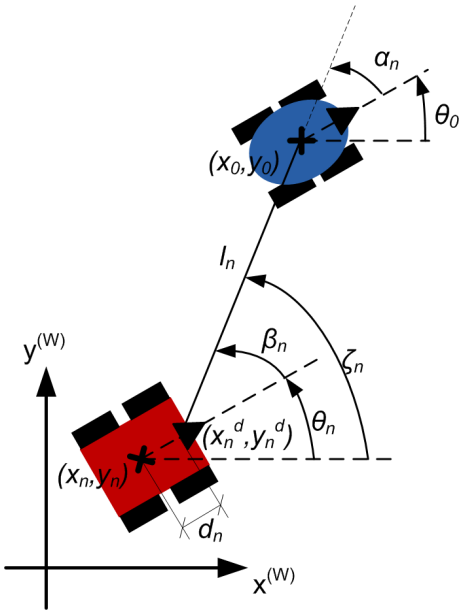


Fig. 1: Relevant parameters of the implemented leader-follower controller.

The leader-following controller is implemented according to the feedback linearization scheme described in [16]. Relevant system parameters for leader-following are shown in Figure 1. The transformation between the Cartesian coordinates of vehicle n and its Euclidean distance $l_n \in \mathbb{R}_{\geq 0}$ and angles $\alpha_n,\ \beta_n \in (-\pi,\ \pi]$ with respect to the leader (indexed with 0) is given by

$$\begin{aligned} l_n &= \sqrt{(x_0 - x_n^d)^2 + (y_0 - y_n^d)^2}, \\ \alpha_n &= \zeta_n - \theta_0, \\ \beta_n &= \zeta_n - \theta_n, \end{aligned} \tag{2}$$

where $\zeta_n = \operatorname{atan2}\left(y_0 - y_n^d, x_0 - x_n^d\right)$, $x_n^d = x_n + d\cos\theta_n$, $y_n^d = y_n + d\sin\theta_n$. The offset d defines a point off the vehicle axis, that is being controlled by the feedback linearization algorithm. From the point of view of the controller, vehicle state is described by

$$\mathbf{s}_n(t) = \begin{bmatrix} l_n(t) \\ \alpha_n(t) \end{bmatrix}. \tag{3}$$

The control law can then be written as

$$\begin{bmatrix} v_n(t) \\ \omega_n(t) \end{bmatrix} = \mathbf{F}^{-1}\left(\begin{bmatrix} u_{nl} \\ u_{n\alpha} \end{bmatrix} - \begin{bmatrix} \cos\alpha_n & 0 \\ -\frac{\sin\alpha_n}{l_n} & 1 \end{bmatrix}\begin{bmatrix} v_0(t) \\ \omega_0(t) \end{bmatrix}\right), \tag{4}$$

where $u_{nl} = K_{nl}\left(l_n^{ref} - l_n\right)$ and $u_{n\alpha} = K_{n\alpha}\left(\alpha_n^{ref} - \alpha_n\right)$ are error signals multiplied by controller gains and

$$\mathbf{F}^{-1} = \begin{bmatrix} -\cos\beta_n & l_n\sin\beta_n \\ -\frac{\sin\beta_n}{d} & -\frac{l_n\cos\beta_n}{d} \end{bmatrix}. \tag{5}$$

This control scheme forces the follower vehicle to track the leader at distance l_n^{ref} and tracking angle α_n^{ref}. By cascading such blocks, arbitrarily complex leader-follower formations can be designed. An example of a multi-vehicle formation

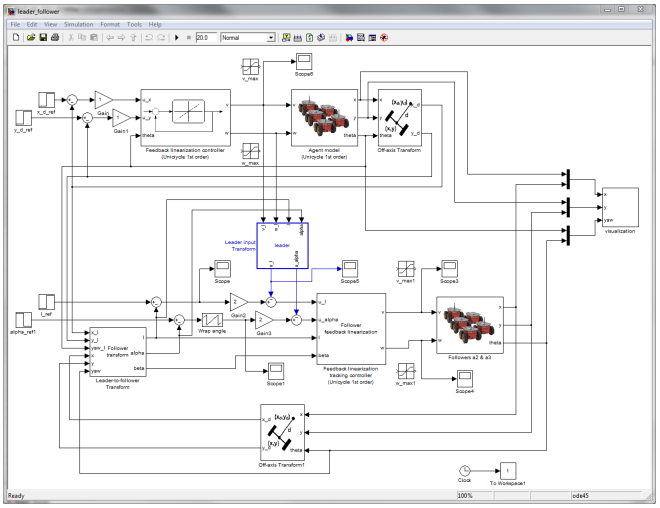


Fig. 2: Simulink model of a leader-follower control scheme. The model scales to an arbitrary number of followers.

model in Simulink, with a single leader and an arbitrary number of followers is shown in Figure 2.

Using AVLib, we have implemented some other control schemes, for example potential field controllers for flocking. However, at this point the interfaces to these controllers were not general enough to be included as library blocks.

C. Environments and visualization

The environment modeling and visualization subset enables the user to visualize vehicles, customize backgrounds, define obstacles, and detect collisions. The Visualization block provided by the library offers a convenient way to visualize an arbitrary number of vehicles in 2D space[2]. Its inputs are three vectors of vehicle poses, interpreted as

[2]Visualization functionality in 3D space is currently under developed.

$(x_n,\ y_n,\ \theta_n)$ triplets. For each vehicle pose, the user can specify a custom drawing function called at each simulation time step. There is also a default drawing function that visualizes the vehicle as an arrow to indicate position and orientation. The block can also plot vehicle vehicle paths, as it moves. In order to provide context to the visualization, the background can be customized by inserting arbitrary bitmaps. Finally, the block provides an option to store the images taken at each simulation step and make a movie of the experiment.

The environment modeling block allows the user to specify an arbitrary number of polygonal obstacles. During simulation, it uses the user-defined vehicle collision model (a polygon, centered at the location of the vehicle) to detect collisions with the environment. This information is then provided as a vector of boolean values at the block's output, and can be used to stop the vehicle when a collision occurs.

D. External interfaces

The main purpose of AVLib is to allow quick and efficient prototyping of multi-agent scenarios and control algorithm debugging. External interface blocks extend this functionality to provide advanced 3D visualization, image processing and control of robotic hardware. The blocks are designed to provide the same interface as purely Simulink-based blocks. For instance, the block to control a differential-drive mobile robot provides the unicycle interface, having v_n and ω_n as inputs, and (x_n, y_n, θ_N) as outputs, with the symbols referring to Equation (1). This enables users to simply replace the vehicle model based on differential equations with the interface to the actual vehicle, and try their algorithms on real hardware, without any changes to the controller structure.

The interface to USARSim [6] provides High-fidelity 3D visualization and a physics-based simulation engine. It can be used to create effective experiment visualizations, or as an in-between step before actual hardware experimentation. Furthermore, it can be used to simulate scenarios not possible with available hardware, for instance with a greater number of robots, or in a custom-designed environment. USARSim interfaces are implemented as m-Language S-functions, and communication with the simulator is implemented based on the work described in [17].

The interface to OpenCV [18] provides access to state-of-the art image processing algorithms. As the library provides a wealth of algorithms and not all can be easily transferred to a Simulink-based implementation, we have currently implemented simple color-based tracking, which provides information on the location of the biggest blob of a particular color in the image. The interface is written as an S-function in C++, and can be used as a template for other algorithms.

To enable a seamless transition between simulations and experiments, an interface using the Player framework [19] has been implemented. It currently supports the Pioneer P3 differential drive mobile platforms, which can be fully controlled from a Simulink model. To provide more reliable localization than the onboard encoders, we have also implemented and interface to the Vicon motion capture system, which can provide sub-millimeter precision vehicle localization. However, because it is linked to proprietary libraries, this interface can not be re-distributed

Controlling objects that are running in another simulation engine, or real hardware, requires real-time synchronization in order to make the control algorithms applicable. For this purpose, we are using the Simulink Real-Time block [20] to synchronize Simulink simulation time to real time. It relies on the assumption that the Simulink model is running much faster than real-time, and just spends the extra time in an idle loop. Although it does not provide any hard real-time guarantees, the performance is satisfactory as long as the sample time is sufficiently large.

IV. Facilitating multi-agent system research with AVLib

In this section, we present several case-studies of using AVLib in multi-agent system research. The case studies range from simple efficiency improvements in simulated scenarios, to performing experiments on actual robotic hardware.

A. Increasing simulation scalability

The first case-study outlines efficiency improvements achieved by replacing legacy code with AVLib. During our jprevious research on multi-agent systems, we have developed a Matlab based simulator with a GUI for in-house use. The simulator had the capability of specifying initial positions for an arbitrary number of robots, and choosing several other parameters, such as simulation duration, time step, and some controller parameters. The actual controller code and vehicle models were hard-coded inside the simulator. Trying to extend the results of our research proved difficult, because there was no flexible way of changing the controller implementation. Essentially, each new controller implementation resulted in a new application. Furthermore, the simulation duration would start to exceed the simulated time, i.e., the simulation was running slower than real-time, already for several dozen agents. The controllers were re-implemented using AVLib within several weeks, and performance improvements were substantial, as depicted in Figure 3. Simulations were run in Matlab R2009a on an Intel® Core2 Duo CPU at 2.53 GHz with 4 GB of RAM, running Windows7 64-bit. Simulated time is 30 s.

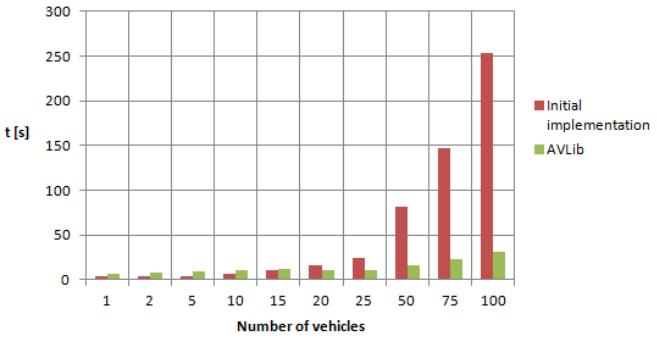


Fig. 3: Scalability comparison between legacy code and AVLib.

B. Improving simulation flexibility

Similarly to the previous example, we had some legacy code used to simulate an environment-search scenario, as described in [21]. In this scenario, loosely motivated by behavioral patterns observed in honeybees, we were simulating about a dozen agents, doing a random search of a restricted area, looking for a hazardous substance. When trying to extend the code to also simulate information propagation in the ”nest”, where agent density was much greater (some 500 units), simulation times became unreasonably large. Again, re-implementing the scenario with AVLib took a modest amount of time, and resulted in a much cleaner, more maintainable and much more efficient implementation. A detail of scout agents performing information propagation through the nest is shown in Figure 4. Results of the extended research were published in [15].

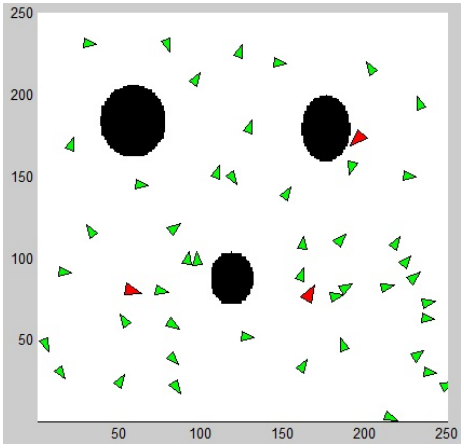


Fig. 4: Scout agents propagating information through the nest.

C. Interfacing with 3D simulation for visual feedback

This scenario was created as a tutorial for the NATO ASI 2010 workshop on Advanced All-Terrain Autonomous Systems[3]. Its purpose is to demonstrate the integration of OpenCV and USARSim simulations into Simulink. The scenario features a Quadrotor performing color-based visual tracking of an unmanned ground vehicle, in a cluttered environment. USARSim is used to simulate the vehicles and provide visual feedback. Simulation snapshots are shown in Figure 5.

Another freely available tool for interfacing USARSim and MATLAB is the USARSim Toolbox [22], developed by the SAS Lab of Drexel University. This tool provides interface functions for the m-Language, whereas AVLib provides Simulink blocks.

D. Transitioning from simulation to experiments

Finally, highlighting what we consider the most prominent feature of AVLib, we describe how it can be used

[3]More information, including workshop materials available at http://robotics.ece.drexel.edu/events/nato/2010/

(a) Camera view.

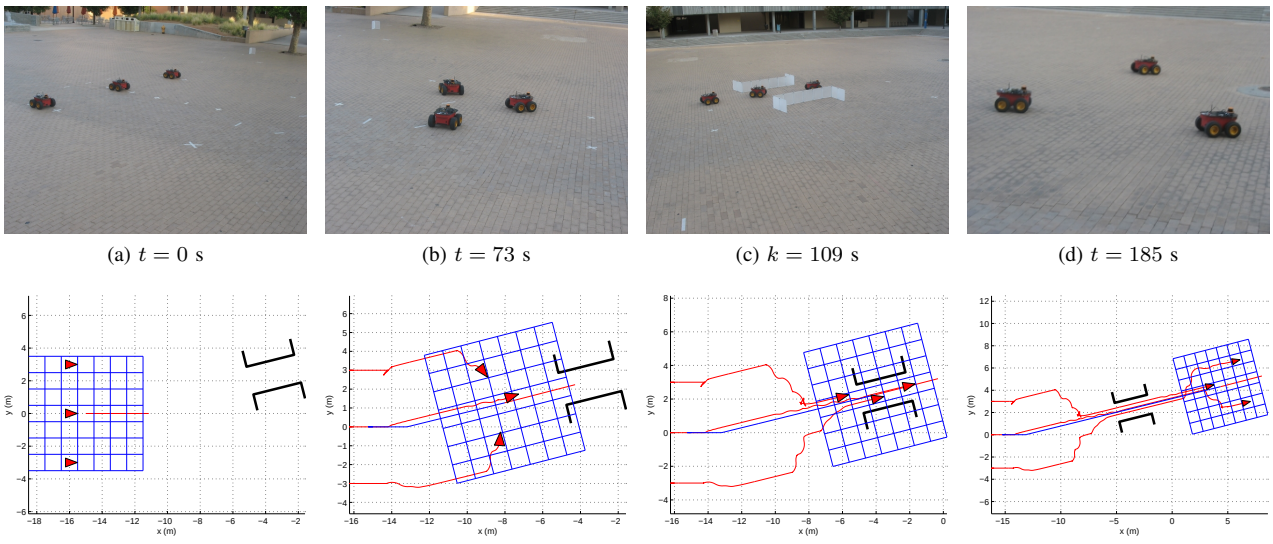
(b) USARSim scene.

Fig. 5: Target following experiment in USARSim.

to seamlessly transition from simulations to experiments. The application considered is a formation control problem for autonomous ground vehicles. A virtual grid structure is introduced as a control abstraction, enabling the vehicles to keep a formation, and switch between different predefined formations, in order to adapt to events in the environment, such as approaching an obstacle. More details on this formation control approach can be found in [23]. Snapshots from an experiment featuring three Pioneer 3-AT robots, changing formation to avoid a corridor-like obstacle, are shown in Figure 6. The control scheme combines point-to-point navigation with leader-following to maintain the formations. A supervisory discrete event-based control layer, implemented in Sateflow®, coordinates the transitioning between formations. The whole control scheme was first developed and debugged in simulation, and then tested on the actual robots. Transitioning from simulations to experiments did not require any changes to the controller code. Simulated vehicles were simply replaced by the Player-based interface blocks, which provide the same inputs and outputs.

V. CONCLUSIONS

We have described AVLib, a library of Simulink blocks implementing standard models, controllers, visualization components and external interfaces, useful for conducting research of multi-vehicle systems. The library provides a convenient way to quickly model a multi-vehicle setup, run simulations that can be scaled to several hundreds of units, and visualize the results. A key feature of the library is the



(a) $t = 0$ s (b) $t = 73$ s (c) $k = 109$ s (d) $t = 185$ s

(e) $t = 0$ s (f) $t = 73$ s (g) $t = 109$ s (h) $t = 185$ s

Fig. 6: Experiment snapshots and corresponding vehicle configurations, plotted in Matlab.

ability to transition from simulations to hardware experiments seamlessly, without requiring any changes in the controller code. This feature enables an efficient experimentation process, where system parameters are tuned through iterative simulations running in accelerated time, and are finally validated experimentally, by simply replacing the vehicle model with an external interface communicating with an actual vehicle. The usefulness of the library is demonstrated through four case studies, ranging from increasing the speed and scalability of simulations, to seamlessly transitioning between simulations and experiments.

REFERENCES

[1] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 6, pp. 926–939, Dec 1998.

[2] N. Michael and V. Kumar, "Planning and Control of Ensembles of Robots with Non-holonomic Constraints," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 962–975, 2009.

[3] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, no. 2, pp. 189–208, 2008.

[4] Webots, "http://www.cyberbotics.com," commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[5] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A survey of commercial & open source unmanned vehicle simulators," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 852–857.

[6] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1400–1405.

[7] J. Jackson, "Microsoft robotics studio: A technical introduction," *IEEE Robotics & Automation Magazine*, vol. 14, no. 4, pp. 82–87, December 2007.

[8] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[9] P. Corke, "MATLAB toolboxes: robotics and vision for students and teachers," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 16–17, 2007.

[10] J. M. Esposito, O. Barton, and J. Kohler, "Matlab Toolbox for the iRobot Create," www.usna.edu/Users/weapsys/esposito/roomba.matlab/, 2011.

[11] MathWorks, "Tech note 1109 - code vectorization guide," Online, January 2012, http://www.mathworks.com/support/tech-notes/1100/1109.html.

[12] A. De Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," *Robot motion planning and control*, pp. 171–253, 1998.

[13] M. Orsag and S. Bogdan, "Hybrid control of quadrotor," in *Control and Automation, 2009. MED'09. 17th Mediterranean Conference on*. IEEE, 2009, pp. 1239–1244.

[14] D. Namlić, "LQR contrl algorithm for a Zeppelin-type UAV," Master's thesis, University of Zagreb, 2010.

[15] M. Varga, S. Bogdan, M. Dragojevic, and D. Miklic, "Collective search and decision-making for target localization," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 18, no. 1, pp. 51–65, 2012. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/13873954.2011.601424

[16] O. Orqueda and R. Fierro, "Robust vision-based nonlinear formation control," in *American Control Conference, ACC*, 2006.

[17] A. Mathis, K. Fregene, and B. Satterfield, "Creating High Quality Interactive Simulations Using MATLAB® and USARSim," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop Robots, Games and Research: Success Stories in USARSim*, 2009.

[18] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[19] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, 2003, pp. 317–323.

[20] L. Daga, "Real-Time blockset for Simulink," Online, May 2007, http://www.mathworks.com/matlabcentral/fileexchange/3175-real-time-blockset-7-1-for-simulink.

[21] M. Varga, Z. Piskovic, and S. Bogdan, "Multi-agent swarm based localization of hazardous events," in *Control and Automation (ICCA), 2010 8th IEEE International Conference on*. IEEE, 2010, pp. 1864–1869.

[22] SAS Lab, "MATLAB USARSim Toolbox," 2011. [Online]. Available: http://robotics.mem.drexel.edu/USAR/

[23] D. Miklic, S. Bogdan, R. Fierro, and Y. Song, "A grid-based approach to formation reconfiguration for robots with non-holonomic constraints," *European Journal of Control*, 2012, to appear.