Platooning LEGOs: An Open Physical Exemplar for Engineering Self-Adaptive Cyber-Physical Systems-of-Systems

Yong-Jun Shin, Lingjun Liu, Sangwon Hyun, Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
{yjshin, riensha, swhyun, bae}@se.kaist.ac.kr

***Abstract*—Many modern systems interact with both cyber and physical environments. They are complex systems in which multiple constituent systems work together to achieve higher-level goals. These systems are called cyber-physical systems of systems (CPSoS). As the interest in CPSoS, such as platooning vehicles and robot-based smart factories, increases, engineering for adaptive goal achievement of CPSoS is needed. Common exemplars of a research community can facilitate research; however, existing exemplars of CPSoS are mostly based on virtual simulations. Although this allows researchers to share experimental scenarios and environments, it has the limitation that it is difficult to conduct experiments that reflect actual physical environments. To overcome this limitation, we propose a physical exemplar of an industrial CPSoS, called *Platooning LEGOs*, which employs platooning technology that is actively being developed by the autonomous driving industry. A platoon, in which independent vehicles drive together, achieves SoS-level goals through adaptive behavioral decisions of the vehicles. This exemplar provides a physical experimental environment that can be implemented with LEGOs. A simple LEGO assembly allows the use of real data from sensors and actuators, facilitating a focus on software engineering without considerable mechanical knowledge. Moreover, as this is an open exemplar, researchers can implement the same physical experimental environment with a limited budget and expand its physical or software elements. We provide system descriptions, physical and software implementation manuals, and sample experimental results of *Platooning LEGOs*.**

***Index Terms*—Platooning LEGOs, Self-adaptive system, Cyber-Physical System, System-of-Systems, Cyber-physical System-of-Systems, Lego, Exemplar, Experimental environment**

I. Introduction

Cyber-physical systems (CPS), such as autonomous vehicles, play an increasingly important role in modern society, attracting considerable interest in CPS engineering [1]. Because not only virtual information but also physical conditions or people must be considered, it is difficult to fully anticipate uncertainties in the environment of a CPS at the time of designing. Therefore, a CPS essentially requires adaptation functionality that can consistently achieve system goals in uncertain environments.

Another characteristic of some modern systems is that they form systems-of-systems (SoS) in which multiple independent systems cooperate to achieve higher-level goals that cannot be achieved by a single system [2]. Examples of SoS are clusters of vehicles or drones, smart factories where many robotic systems work together, and complex defense systems with multiple weapon systems. As the size and influence of SoS increase, an important objective of SoS engineering is to ensure that SoS goals are achieved stably regardless of uncertainty.

In this context, cyber-physical systems-of-systems (CPSoS) require engineering for collaborative adaptations in the uncertain physical world [3]. To promote active research and share common adaptation problems, the Software Engineering for Adaptive and Self-Managing Systems (SEAMS) research community has accumulated several exemplars[1] [4]–[13]. However, there are few exemplars for adaptation engineering of CPSoS. Moreover, while most exemplars have provided simulators, studying CPS only in simulations without physical environments has limitations in reflecting reality. Furthermore, building a physical experimental environment often requires specialized domain knowledge and entails high costs.

To meet the need for a CPSoS exemplar to consider the physical environment realistically for adaptation engineering, we propose an open physical exemplar called *Platooning LEGOs*. As a representative example of CPSoS, we selected a platooning technology for autonomous vehicles [14]. Platooning is an industrial technology that is actively being developed by vehicle manufacturers. Vehicles with the same destination form a platoon through communication, drive in a line to reduce air resistance, thus reducing fuel consumption, and adjust the distances between them to reduce road occupancy. Platooning is self-adaptive to uncertain situations in a driving environment. Our exemplar implements platooning

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2020-2020-0-01795) and (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation).

[1]SEAMS exemplar repository:
https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

2157-2321/21/$31.00 ©2021 IEEE
DOI 10.1109/SEAMS51251.2021.00038

using programmable LEGO robots. Unlike the cases where platooning robots have been implemented and used in experiments privately [15]–[17], we propose *Platooning LEGOs* as a reproducible and expandable exemplar that allows anyone to build the same physical experimental environment for CPSoS engineering. In summary, our *Platooning LEGOs* exemplar contributes to the field as:

- a CPSoS exemplar: an industrial adaptation model problem (platooning) representing both CPS and SoS,
- a physical exemplar: a physical experimental environment producing real data from physical sensors and actuators,
- an open exemplar: an exemplar that allows anyone to build the same physical experimental environment with a limited budget using LEGOs and expand its physical and software elements.

The remainder of this paper is organized as follows. Section II describes related exemplars. Section III presents our *Platooning LEGOs*. Section IV describes how to implement both physical and software artifacts of this exemplar. Section V presents an experiment of a sample scenario. Section VI concludes the paper.

II. Related Exemplars

More than 20 diverse exemplars have thus far been proposed by the SEAMS community for self-adaptive system (SAS) engineering[1]. The target SAS of approximately half of these exemplars is CPS or the Internet of Things [4]–[13]. About half of the CPS exemplars deal with SoS [9]–[13]. In the traffic domain, there are exemplars for adaptive traffic optimization with cooperating vehicles [9], [10]. Other exemplars present collaborative delivery of goods by independent drones [11] and a reconnaissance mission by a team of unmanned aerial vehicles [12]. There is also an exemplar for a self-adaptive ecosystem composing smart agents to control the global food population [13]. Unfortunately, except for DeltaIoT [6] allowing remote access to a real IoT system, existing exemplars only support virtual simulations of the target SAS, which makes it hard to reproduce real physical environments in experiments. Unlike previous studies, our *Platooning LEGOs* exemplar represents an open physical experimental environment of an industrial vehicular CPSoS.

III. Platooning LEGOs

A. SoS-level overview

Platooning is a technology currently being developed by real industries; therefore, levels and functions vary according to the manufacturer. We refered to demonstrations of various platooning techniques and simplified core features that can be implemented using LEGOs. Figure 1 shows an overview of our *Platooning LEGOs*. Each platooning vehicle is a programmable LEGO robot. A vehicle can independently drive in a lane, control its driving speed, change lanes, and detect obstacles ahead. Each vehicle transmits a set of raw data that include its current driving speed, lane, and forward distance. The platoon comprises three vehicles. The first vehicle is the leader, which knows the conditions ahead. The second vehicle is a follower driving along the path of the leader and relaying the leader's state to a third vehicle, another follower, which follows the second vehicle and is outside the leader's communication range. As unknown objects or situations exist in a road environment, each autonomous vehicle makes decisions on its speed and lane so that the platoon adaptively achieves its SoS-level goals in an uncertain environment.

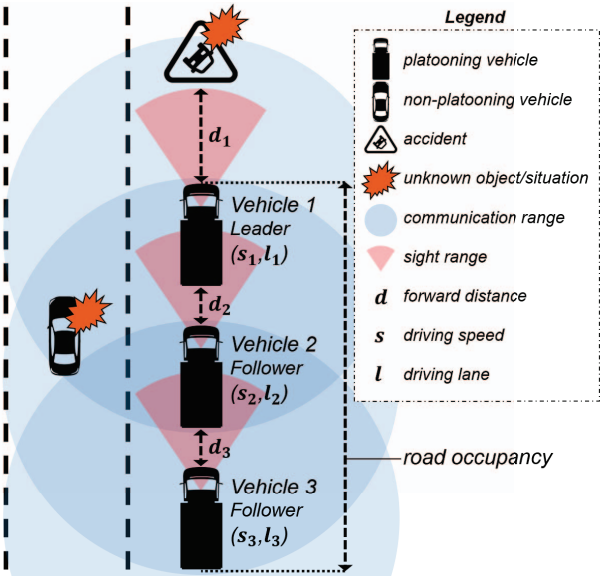

Fig. 1. Overview of *Platooning LEGOs*

TABLE I
Adaptation Goals of the *Platooning LEGOs*

Goal	Description	Evaluation metric
(Hard) Collision prevention	All vehicles of the platoon shall not collide with any vehicle or object.	$(d_1 > 0) \wedge (d_2 > 0) \wedge (d_3 > 0)$
(Hard) Driving in a row	Except when changing lanes, all vehicles of the platoon must drive in a row on the same lane.	$(l_1 = l_2) \wedge (l_2 = l_3) \wedge (l_3 = l_1)$
(Soft) Road occupancy minimization	In order to reduce road occupancy, the distance between vehicles in the platoon should be minimized as much as possible.	$d_2 + d_3 + (3 * vehicleLength)$
(Soft) Travel speed maximization	In order to shorten the travel time, the average driving speed of the platooning vehicles should be maximized within road limits.	$\frac{s_1 + s_2 + s_3}{3}$

The SoS-level adaptation goals of the platoon are summarized in Table I. There are two "hard" goals (with clear satisfaction criteria) and two "soft" goals (without clear-cut criteria). If the two hard goals are achieved, the platoon tries to achieve the soft goals. The first hard goal is to prevent a collision with another vehicle or other object. The second hard goal is to drive in a row in the same lane to minimize air resistance and thus fuel consumption. The first soft goal is to minimize road occupancy for smooth traffic flow. This is

TABLE II
ACTIVITIES OF VEHICLES

Vehicle	Activity
Vehicle 1: Leader	**Monitor & Receive**: • Forward distance (Ultrasonic sensor) • Driver's manual command (Button)
	Analyze & Plan: • Driving speed decision • Driving lane decision
	Execute: • Setting the driving speed • Changing/keeping the driving lane
	Send: • *msg1* (to vehicle 2) – Current driving speed – Current driving lane – Current forward distance
Vehicle 2: Follower	**Monitor & Receive**: • Forward distance (Ultrasonic sensor) • Peers' situation (*msg1*) (Bluetooth)
	Analyze & Plan: • Driving speed decision
	Execute: • Setting the driving speed • Changing/keeping the driving lane
	Send: • *msg2* (to vehicle 3) – Current driving speed – Current driving lane – Current forward distance
Vehicle 3: Follower	**Monitor & Receive**: • Forward distance (Ultrasonic sensor) • Peers' situation (*msg2*) (Bluetooth)
	Analyze & Plan: • Driving speed decision
	Execute: • Setting the driving speed • Changing/keeping the driving lane
	Send: • None

achieved by minimizing the distances between the platooning vehicles by speed adjustments. The second soft goal is to maximize the platoon's driving speed to minimize travel time. The data-based logical or numeric evaluation metrics of the goals are displayed in Table I. In cases in which the experimental data cannot accurately show whether the goals are achieved in a physical environment (e.g., a side collision of a vehicle), users can observe the experiment itself in addition to the data to determine whether the goals are achieved.

B. Self-adaptive constituent vehicles

In this section, we describe the behavior of each constituent vehicle. Each vehicle is autonomous and stand-alone. The vehicles' activities are summarized in Table II. The activities are organized as a MAPE (Monitor, Analyze, Plan, and Execute) loop of SAS [18]. Although in this work we implemented an independent MAPE loop in each vehicle without a coordinator, different types of MAPE patterns for SoS can be used in our exemplar [19]. Our LEGO vehicles are equipped with three kinds of sensors: a color sensor for line following, an ultrasonic sensor for obstacle detection, and a maximum of two buttons for reception of the driver's manual commands. We assume the color sensor for line following to be a basic function for autonomous driving, and it is thus not described in Table II. Only driving lane and speed adaptations are described.

The first vehicle (leader) senses forward distances to detect accidents or other obstacles. It can also sense the driver's commands, such as manual control of speed or lane, through buttons. In some real examples of platooning, a leader vehicle allows manual driving. However, even if there is no manual command, it can automatically adapt the driving speed and lane. Moreover, it sends its current driving speed, lane, and forward distance to a follower vehicle.

The second vehicle (follower) senses forward distances and receives the leader's current state. However, unlike the leader, when it is part of a platoon, its steering wheel and accelerator cannot be used, and only adaptive cruise control is allowed. In real cases, a follower can leave or join the platoon at the driver's command, but the exemplar described herein only covers the joined state of the platooning protocol. If a user wishes to allow manual command of a follower vehicle, it can be expanded using buttons. Based on the monitored and received situation of the platoon, the second vehicle follows the leader's lane and decides the speed.

The third vehicle (follower) follows the first and second vehicles. Because the leader's communication range may be limited, the third vehicle only receives messages from the second vehicle. This communication topology is called "predecessor following" [20]. Although all activities of the third vehicle are subsumed by the second one, it is still an independent constituent system, and it adapts its speed to contribute to the platoon's goals.

C. Environmental uncertainties of the platoon

The environmental uncertainties that can be addressed by *Platooning LEGOs* are summarized in Table III. The platoon has limited knowledge of the peers' situation, the physical road environment, such as other vehicles or accidents, and human drivers' behavior. It is almost impossible to enumerate all possible situations of the platoon at the time of designing. Therefore, the platoon should be adaptive to the uncertainty of the environment. Because *Platooning LEGOs* provide a physical experimental environment, realistic physical events can be simulated in experiments. Moreover, the platoon interacts with the environment through sensors and actuators. Actual interactions may differ from the expected interactions due to sensing/actuating noise or failure. Such incomplete interactions can also produce uncertain platoon operation results. To simulate diverse settings and address various uncertainties, sensor/actuator noise or failure rates can be introduced to experiments.

IV. IMPLEMENTATION

A. Physical implementation

1) Vehicle implementation: Each vehicle is an independent LEGO MINDSTORMS EV3 robot (The LEGO Group, Denmark), as shown in Figure 2. Each robot is equipped with two main wheels connected to motors and one auxiliary wheel. It

TABLE III
ENVIRONMENTAL UNCERTAINTIES ADDRESSED BY THE *Platooning LEGOs*

Environmental uncertainty	Subcategory	Description
Uncertainty due to the limited knowledge of the environment	Unpredictable peers' situations	Each vehicle determines its behavior based on the peers' situation received through a network connection, but the possible peers' situations cannot be fully anticipated in advance.
	Unpredictable physical environment	Unexpected physical conditions, such as an interruption by another vehicle or an accident on the road, may occur, which cannot be predicted.
	Unpredictable human behavior	A human driver can give a direct command to a vehicle through buttons, and the behavior may not be accurately predictable and may interfere with the achievement of the platoon's goals.
Uncertainty due to the incomplete interaction with the environment	Inaccurate sensing	The sensed or received information may not accurately reflect the actual environmental situation because of sensor noise or message transmission/reception delays.
	Sensing failure	The vehicles may fail to obtain information on the environment because of the broken physical sensors or message loss.
	Inaccurate effecting	The planned adaptation behavior may not be ideally applied to the physical world because of the motor's limited precision or physical constraints such as frictional force of the road.
	Effecting failure	With a very low probability, the planned action may not be executed due to a connection error with the motor or other unexpected causes.

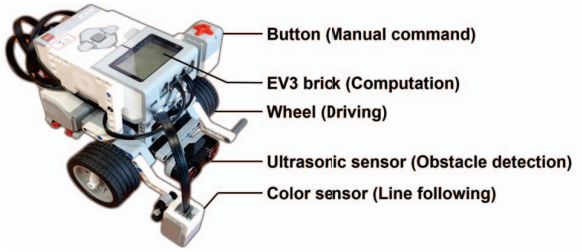

Fig. 2. A LEGO Mindstorms EV3 vehicle

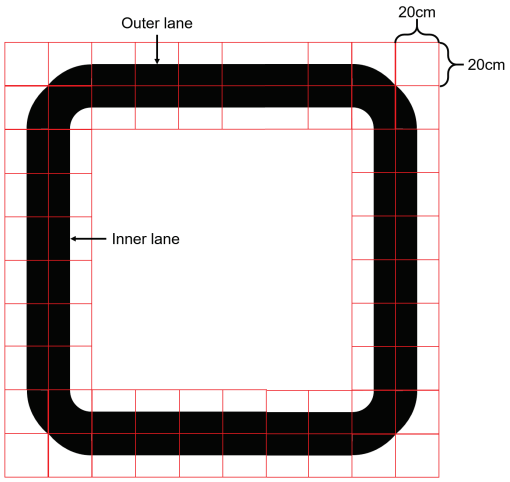

Fig. 3. Road environment

is also equipped with a color sensor and an ultrasonic sensor to sense the road and the situation in front of it. Messages from other vehicles are received via Bluetooth, which is embedded in the EV3. A button can also be attached for commands from a human driver, such as manual lane change or acceleration. All physical implementations are very simple and follow the building instructions provided by the manufacturer. The instructions and manuals of the *Platooning LEGOs* have been

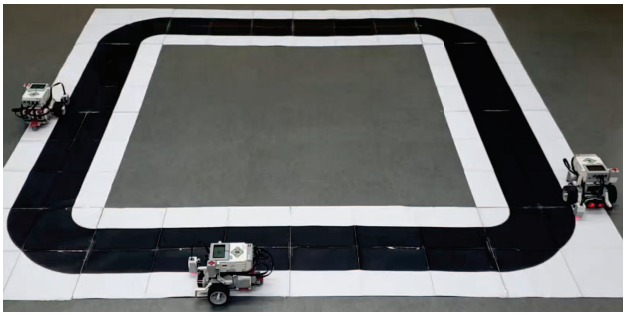

Fig. 4. A physical implementation of *Platooning LEGOs*

uploaded to our GitHub repository[2].

2) Road implementation: To simulate platooning on an endless highway, engineers print circular roads, as shown in Figure 3, and attach them to a floor. There are an outer and an inner lane. Vehicles drive clockwise following the boundaries of the black line. When a vehicle decides to change lanes, it turns in the direction of the new lane and drives until it finds a white area. The road implementation material has also been uploaded to our repository[2]. The simulated road environment can be easily replicated using sheets of paper and a space of only about 2m×2m. Figure 4 shows the vehicles and simulated road environment. The three vehicles are in the outer lane.

As *Platooning LEGOs* provides a physical experimental environment with very simple physical implementations, they allow software engineers to focus on the vehicles' software. The software implementation guide is provided in the next subsection.on.

B. Software implementation

The vehicles' software is implemented using a Python API[3]. Each vehicle iteratively performs monitoring, analysis, planning, and execution. A detailed description of each step is presented in the code skeleton shown in Algorithm 1.

[2]*Platooning LEGOs* repository: https://doi.org/10.5281/zenodo.4604167 (https://github.com/yongjunshin/Platooning-LEGOs)

[3]Mindstorms EV3 API: https://pybricks.github.io/ev3-micropython

Algorithm 1 A vehicle code skeleton

```
1: Configuration of sensors, motors, and vehicle
2: data = DataLog('time','lane','speed')
3: watch = StopWatch()
4: while True do
5:    time = watch.time()
6:    color = colorSensor.reflection()
7:    dist = ultrasonicSensor.distance()
8:    peerLane = laneMailBoxFrontVehicle.read()
9:    Adaptation of lane and speed
10:   vehicle.drive(speed, turnRateFor(lane))
11:   laneMailBoxBackVehicle.send(lane)
12:   data.log(time, color, dist, peerLane, lane, speed)
13: end while
```

1) Monitor & Receive: The lane and speed adaptations are based on recognition of the road environment and the peer vehicles' state. The road environment is monitored by sensors. A color sensor monitors the floor and returns a color value of the road (Algorithm 1, line 6). An ultrasonic sensor measures the distance to objects in front of the vehicle in millimeters (Algorithm 1, line 7). The peer vehicles' state is received via Bluetooth. A vehicle shares a mailbox with another vehicle and can receive data using the $read()$ function (Algorithm 1, line 8). For more complex scenarios, additional sensors, such as touch, gyroscope, and infrared sensors, and mailboxes for information reception from more peers can be used following the API[3] and our manual and sample codes[2].

2) Analyze & Plan: Each vehicle adapts its driving lane and speed by analyzing the monitored environment, the peers' state, and its own state (Algorithm 1, line 9). Engineers can use their own adaptation approaches and analyze their effectiveness. To guide their implementation, we provide a code skeleton and the sample code used in our experiment through our repository[2].

3) Execute: A $vehicle$ is an instance of a *DriveBase* object in the API, and the adaptation decision on speed and lane is executed by the $drive()$ function of the instance (Algorithm 1, line 10). The function receives the speed (mm/s) and rotation angle (deg/s) as inputs. The driving lane is a specific concept in our exemplar, so a lane change decision must be converted to a rotation angle. Changing the lane can be realized by turning clockwise or counterclockwise. A reusable code for following and changing lanes can be found in our repository[2].

4) Send: To achieve SoS goals, the state and adaptation decisions of a vehicle should be known to a follower. The vehicle uses the $send()$ function of a mailbox shared with the follower so that the follower can read the data. This communication allows the vehicles to be integrated into an SoS.

5) Data logging: Logging data is an important feature of an experimental environment. The LEGO API[3] provides a simple logging function implemented in two lines of code (Algorithm 1, lines 2 and 12). The data are saved as a CSV file. A timestamp for each iteration of the adaptation loop can also be extracted (Algorithm 1, lines 3 and 5).

V. Experiment

A. Sample scenario

To demonstrate the feasibility of the *Platooning LEGOs* as a physical experimental environment for CPSoS engineering, we conducted a sample experiment. The platooning vehicles were programmed to achieve the adaptation goals described in Table I. The implementation code can be found in our repository[2]. To check whether our platoon implementation is sufficiently adaptive to environmental uncertainties, we introduced two events that could interfere with the goal achievement while the platoon is driving. The first event was an interruption by a *moving obstacle*, such as a non-platooning vehicle on a highway. The second event was a blockage of a lane due to a *fixed obstacle*, such as a traffic accident.

B. Experiment results

The experimental code and result data have been uploaded to our repository[2]. The experimental results are visualized in Figure 5. A video of the experiment has also been released[4]. Figure 5 (a–d) shows the achievement of the platoon's adaptation goals. Figure 5 (e–f) shows the adaptations of each vehicle. The unexpected events (obstacles) are also shown. The two hard goals (collision prevention and driving in a row) were achieved in all scenarios. On the other hand, the achievement of the two soft goals varied depending on the situation.

When moving obstacles interfered with the platoon's driving (moving obstacles 1 and 2 in Figure 5), road occupancy increased and travel speed decreased. However, after the moving obstacles disappeared, the vehicles adapted their driving speeds to reduce road occupancy and increase the platoon's travel speed. When stationary obstacles blocked a lane (fixed obstacles 1 and 2 in Figure 5), the travel speed decreased. The leader decided to change lanes, and the followers also changed lanes to bypass the obstacles. The vehicles then adapted their speed to maximize the platoon's travel speed. The experiment confirmed that the *Platooning LEGOs* can be used as a case of industrial self-adaptive CPSoS and a physical experimental environment for CPSoS engineering.

VI. Conclusion

Many CPSoS are required to adaptively achieve their goals in uncertain environments. An industrial example of a CPSoS is platooning technology for clustered autonomous driving of independent vehicles. The vehicles adapt their driving lanes and speeds to achieve the platoon's adaptation goals. To facilitate research into self-adaptive CPSoS engineering, we presented an exemplar called *Platooning LEGOs*, which is a model problem of platooning technology implemented using LEGOs. We provided system descriptions, adaptation goals, environmental uncertainties that can be addressed in this exemplar, and physical and software implementation manuals of the experimental environment. We conducted a sample

[4]Experiment demonstration video - https://youtu.be/tRSoTPq5EEI

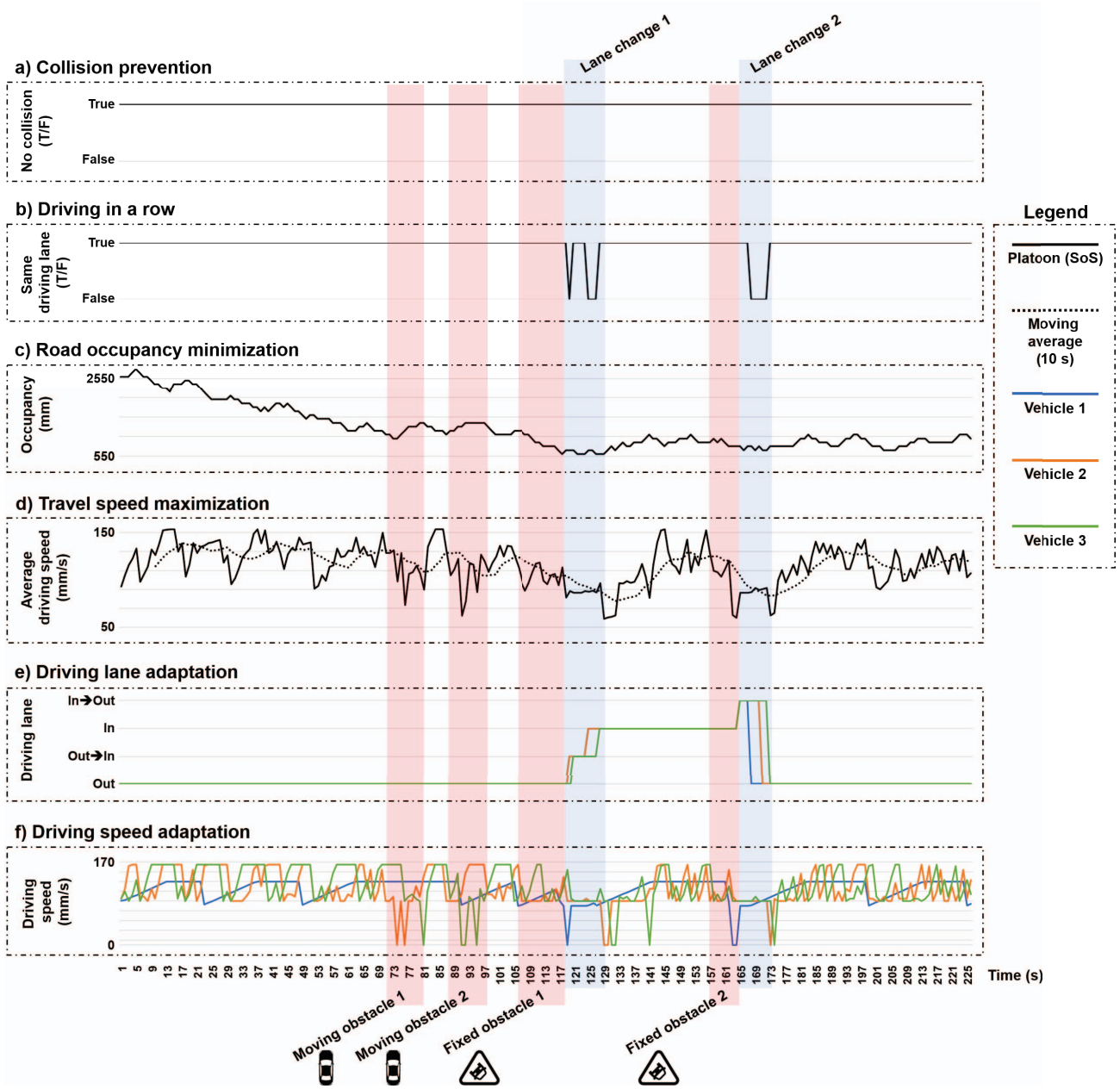

Fig. 5. Sample experiment results

experiment to validate the feasibility of the *Platooning LEGOs* and have made all experimental materials available. We showed that our physical exemplar is conducive to experimenting with adaptation approaches in real CPSoS. Moreover, it is expandable and easy to implement with a limited budget and without considerable mechanical knowledge. *Platooning LEGOs* can be used by the SEAMS community as a common physical experimental environment for self-adaptive CPSoS engineering.

References

[1] T. Bures, D. Weyns, B. Schmer, E. Tovar, E. Boden, T. Gabor, I. Gerostathopoulos, P. Gupta, E. Kang, A. Knauss *et al.*, "Software engineering for smart cyber-physical systems: Challenges and promising solutions," *ACM SIGSOFT Software Engineering Notes*, vol. 42, no. 2, pp. 19–24, 2017.

[2] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, pp. 1–41, 2015.

[3] S. Engell, R. Paulen, M. A. Reniers, C. Sonntag, and H. Thompson,

"Core research and innovation areas in cyber-physical systems of systems," in *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*. Springer, 2015, pp. 40–55.

[4] M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, and F. Plasil, "An architecture framework for experimentations with self-adaptive cyber-physical systems," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2015, pp. 93–96.

[5] S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns, "Undersea: an exemplar for engineering self-adaptive unmanned underwater vehicles," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017, pp. 83–89.

[6] M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns, and D. Hughes, "Deltaiot: A self-adaptive internet of things exemplar," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017, pp. 76–82.

[7] F. Krijt, Z. Jiracek, T. Bures, P. Hnetynka, and I. Gerostathopoulos, "Intelligent ensembles-a declarative group description language and java framework," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017, pp. 116–122.

[8] M. Provoost and D. Weyns, "Dingnet: a self-adaptive internet-of-things exemplar," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 195–201.

[9] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy, "Traffic routing for evaluating self-adaptation," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2012, pp. 27–32.

[10] I. Gerostathopoulos and E. Pournaras, "Trapped in traffic? a self-adaptive framework for decentralized traffic optimization," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 32–38.

[11] P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman, and B. Nuseibeh, "Dragonfly: a tool for simulating self-adaptive drone behaviours," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 107–113.

[12] G. Moreno, C. Kinneer, A. Pandey, and D. Garlan, "Dartsim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2019, pp. 181–187.

[13] A. Bennaceur, C. McCormick, J. García-Galán, C. Perera, A. Smith, A. Zisman, and B. Nuseibeh, "Feed me, feed me: an exemplar for engineering adaptive software," in *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2016, pp. 89–95.

[14] C. Bergenhem, S. Shladover, E. Coelingh, C. Englund, and S. Tsugawa, "Overview of platooning systems," in *Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012)*, 2012.

[15] D. Martinec and Z. Hurák, "Vehicular platooning experiments with lego mindstorms nxt," in *2011 IEEE International Conference on Control Applications (CCA)*. IEEE, 2011, pp. 927–932.

[16] E. Kita, H. Sakamoto, H. Takaue, and M. Yamada, "Robot vehicle platoon experiment based on multi-leader vehicle following model," in *2014 Second International Symposium on Computing and Networking*. IEEE, 2014, pp. 491–494.

[17] E. Kita and M. Yamada, "Vehicle velocity control in case of vehicle platoon merging," in *2019 4th International Conference on Intelligent Transportation Engineering (ICITE)*. IEEE, 2019, pp. 340–344.

[18] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[19] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 76–107.

[20] A. Abunei, C. R. Comşa, C. F. Caruntu, and I. Bogdan, "Redundancy based v2v communication platform for vehicle platooning," in *2019 International Symposium on Signals, Circuits and Systems (ISSCS)*. IEEE, 2019, pp. 1–4.