RoCoSys: A Framework for Coordination of Mobile IoT Devices

Christian Krupitzer*, Martin Breitbach*, Johannes Saal*, Christian Becker*, Michele Segata†, and Renato Lo Cigno†
*Chair of Information Systems II, University of Mannheim, Germany
Email: {christian.krupitzer,martin.breitbach,johannes.saal,christian.becker}@uni-mannheim.de
†Dept. of Information Engineering and Computer Science, University of Trento, Italy
Email: {msegata,locigno}@disi.unitn.it

***Abstract*—Mobile IoT devices enable new classes of systems, such as cyber-physical systems. These systems pose challenges as they should seamlessly interact with users and other systems. In this paper, we address the problem of interaction between mobile pervasive IoT devices. Our contributions are threefold. First, we present a concept for a framework for coordination of mobile IoT devices. Second, we implement a reusable robot platform using the Mindstorms toolkit and a customizable adaptation logic for their coordination based on our framework. Third, we show its usability with two applications: an intelligent vehicle highway system as well as a smart vacuum cleaner.**

I. Introduction

Pervasive computing technology blurs the line of interaction between humans and machines. On the one hand, this technology makes everyday objects intelligent and connects them for additional benefits. On the other hand, pervasive technology offers new styles of human-machine interaction based on mobile, pervasive IoT. One example are cyber-physical systems (CPSs) that are defined as *engineered systems whose operations are monitored, coordinated, controlled, and integrated by computing and communication cores interacting with the physical environment* [1]. Examples for such systems are intelligent vehicles, robotics in production, or unmanned vehicles in warehouse logistics that interact with employees.

These systems pose new challenges. First, the users should not need to learn new paradigms for control, hence, the interfaces for controlling the system must be easy understandable. Further, CPSs have to interact with the environment. Last, they communicate with other systems and share a common pervasive environment. Hence, they need coordination. This is a non trivial issue as common standards for IoT are often missing.

Focusing on the last mentioned challenge, this paper presents a flexible approach for coordination of mobile pervasive agents. We combine this with reusable, flexible robots that are based on the Mindstorms tool kit. Both can be used by developers out of the box with minor customization in applications with self-driving robots. Further, the approach can be extended to any other mobile IoT application domain. Whereas most approaches focus on specific domains, we offer a reusable and extendable system based on the MAPE control principle. MAPE stands for monitoring the system and environment (M), analyzing the monitored values (A), planning for adaptation of the system resources (P), and controlling the execution of these resources (E). This enables self-adaptation of the resources to changes in the environment.

Our contributions are threefold. First, we introduce *RoCoSys*, a system for the coordination of mobile IoT devices. The concept is generically usable in many different application domains. This paper presents an implementation in Java focusing on coordination of Mindstorms robots. By modularization of the algorithms within the MAPE components, the implementation of RoCoSys is reusable in other mobile IoT applications. Second, we implement self-driving Mindstorms robots which are coordinated by RoCoSys. Developers may use our implementation of RoCoSys and the robots as development toolkit for developing own applications, such as coordinating unmanned vehicles for logistics, management of traffic with self-driving cars, or coordination of robots in production facilities. The robots can easily be customized and extended, as they are implemented with the flexible Mindstorms set. However, our functionality for autonomous driving and communicating may be reused without the need for changes. Additionally, developers can reuse the RoCoSys implementation and only need to customize the algorithms for analyzing and planning. Third, for showing the flexibility and reusability of RoCoSys, we used our system for implementing two applications: an autonomic smart vacuum cleaner (SVC) as well as a system that coordinates self-driving vehicles.

The remainder of the paper is structured as follows. In Section II, we present the design of our generic framework, RoCoSys, for coordinating mobile IoT devices. Section III presents the implementation of self-driving robots that can be used in different settings as well as a specific implementation of RoCoSys for coordination of our robots. We evaluate RoCoSys in the two aforementioned applications. Section V discusses related work. Last, Section VI concludes the paper and suggests future work.

II. System Model

Our system model for *RoCoSys* follows the division of an adaptive system into the managed resources that should be adapted and the adaptation logic that controls the adaptation of the managed resources. This model is known from the self-adaptive systems domain (e.g., [2], [3], or [4]). The managed resources contain hardware and software that offer functionality to users or other systems. As we focus on CPSs, more specifically robots, the managed resources are robotic systems and infrastructure that robots may use, e.g., charging stations or signaling systems. The robots monitor and affect the environment as well as perform specific tasks autonomously.

RoCoSys coordinates them, hence, it acts as the adaptation logic. It monitors both, environment and managed resources, and adapts the parameters of the robots if necessary. Figure 1 shows the basic structure of RoCoSys and the connection to robots. Many self-adaptive systems share the concept of a feedback loop in the adaptation logic which constantly monitors the system state and the environment as well as plans and controls adaptation of the resources as reaction to changes in the resources or the environment. The most well-known approach for this feedback loop is the MAPE-K loop [4]. First, the adaptation logic gathers information about the managed resources via *sensors*. This knowledge is used by the following four MAPE components [2]: monitor, analyzer, planner, and executor.

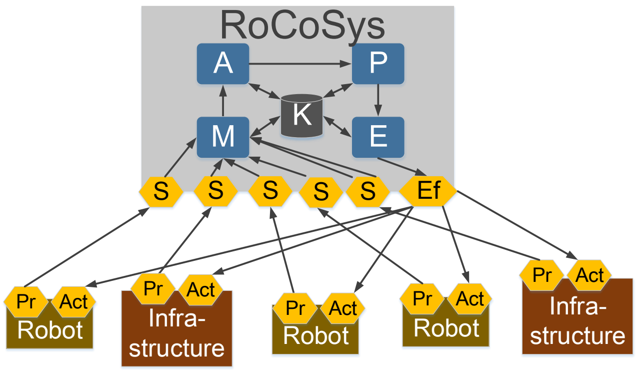


Figure 1: The architecture of RoCoSys and the connection to robots. The amount of robots as well as the amount of RoCoSys subsystems are chosen randomly (S=Sensor, M=Monitor, A=Analyzer, P=Planner, E=Executor, Ef=Effector, Pr=Probe, Act=Actuator).

The *monitor* collects, aggregates, correlates, and filters sensor data. In RoCoSys, first, it registers new robots in the system. It assigns IDs and stores the robots' data. The monitor prepares the data for the subsequent reasoning process, e.g., by excluding irrelevant data or by building models representing the system and the environment.

The *analyzer* receives data from the monitor and conducts data analysis to determine a need for adaptation. Therefore, the component uses an algorithm that reasons on the monitored data. This algorithm makes decisions based on models, rules, goals, or utility functions. It needs to be easily interchangeable for different applications and also within an application, e.g., for providing different analysis algorithms for different monitored situations. Here, it is important to allow a variable extension of possible analyzing results instead of confining the choice on certain predefined values.

If the algorithm determines a need for change, the analyzer forwards relevant data to the planner. The *planner* defines parameters of the robots that should be changed, e.g., the speed or direction of driving or the items a robot arm should grab. Depending on the characteristics of the use case, the optimal procedure to determine proper parameters may vary. Therefore, similar to the analyzing algorithm, the planning algorithm needs to be interchangeable. The design enables to implement different planning algorithms and the planner chooses the appropriate algorithm(s) based on the analyzing result. In addition to the high-level planning algorithms, which calculate a suitable set of actions for one analyzing result, the planner also contains basic planning modules. These generate the parameters for specific actions, e.g., determines the velocity needed for achieving a location at a defined point in time. The planning modules are used by multiple high-level planning algorithms. After all planning algorithms have finished their calculations, the component composes them to an overall plan which is forwarded to the executor.

The *executor* transforms the received plan into low-level instructions and schedules the execution of the adaptation actions. The executing function of the AL uses *effector* interfaces to interact with the managed resources. We have a 1:1 mapping from sensor objects to robots, hence, we use various sensor objects simultaneously. For sending instructions, the architecture contains only one effector component since instructions are sent in sequential order.

A *knowledge* component contains a repository or a database which stores relevant information such as log files, executed plans, or analyzed symptoms [2]. All MAPE components are able to access the data and to store new information.

The robots need to implement interfaces to gather data and to adapt their state (e.g., velocity of self-driving robots or movement of robot arms). These interfaces are called *probes* and *actuators*, respectively. So far, we offer a set of instructions for collecting information from the robots' probes and controlling them through the actuators. Developers may extend and customize the instruction set for their specific applications thanks to the modular structure.

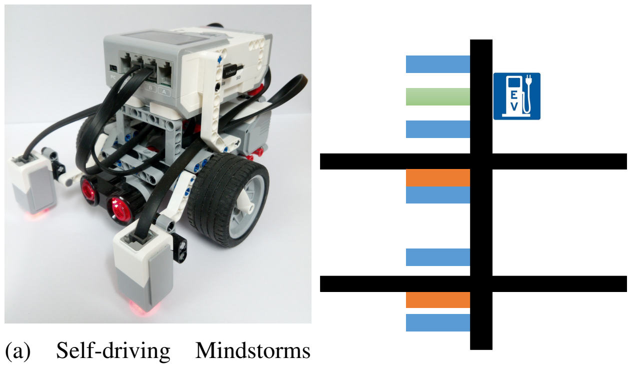
The system itself is reusable. Whereas monitor and executor can be reused out of the box, the analyzer and planner may require customization. However, through a modular approach, all parts are encapsulated. This makes algorithms for analyzing and planning reusable and exchangeable. Additionally, decentralization of the adaptation logic is supported. In the next section, we sketch the implementation of the coordination system for a specific robot platform.

III. Implementation

The design presented in Section II is generically usable in systems that handle coordination of mobile IoT devices, such as robots. In this paper, we focus on self-driving robots that we built using the Mindstorms kit. Still, the domain is quite large and includes a variety of applications, e.g., self-driving vehicles, smart vacuum cleaners, rescue robots, or unmanned vehicles in warehouses logistics. In this section, we first present the implementation of the self-driving robots. Then, we present the implementation of RoCoSys for coordinating these robots.

A. *Implementation of Self-driving Robots*

Mindstorms is a popular robot kit widely used for educational purposes [5]. It contains a programmable central unit running a variant of Linux, several sensors, and various parts to customize functionality and appearance. Our robots consist of the central unit, two engines for driving, and three sensors to monitor the environment. Figure 2a shows such a robot.



(a) Self-driving Mindstorms robot.

(b) Schema of the track.

Figure 2: Setup of the robot and schema of the track with positioning information.

In order to follow the track, the robot uses a color sensor. Furthermore, the robot maintains gaps to other robots or obstacles using an ultrasonic sensor. A second color sensor recognizes markings on the floor for positioning and specific information, e.g., marking intersections. The markings act as an indoor positioning system. Figure 2b shows a schema for a track segment with markings. The blue markings are equally distributed for positioning. The orange markings indicate a crossing, the green one a parking spot for charging a robot.

Additionally, the LeJOS EV3 virtual machine [6] installed on the robot enables the robot to run Java software. The Mindstorms robot communicates with RoCoSys through an 802.11 (Wi-Fi) network. Due to their simple design, the robots can be used as self-driving robots in many applications. Only the markings for positioning are needed. Furthermore, these robots are easily extendable and – through the flexibility of the Mindstorms kit – the robots can be customized (e.g., by an arm or a cart) for many scenarios.

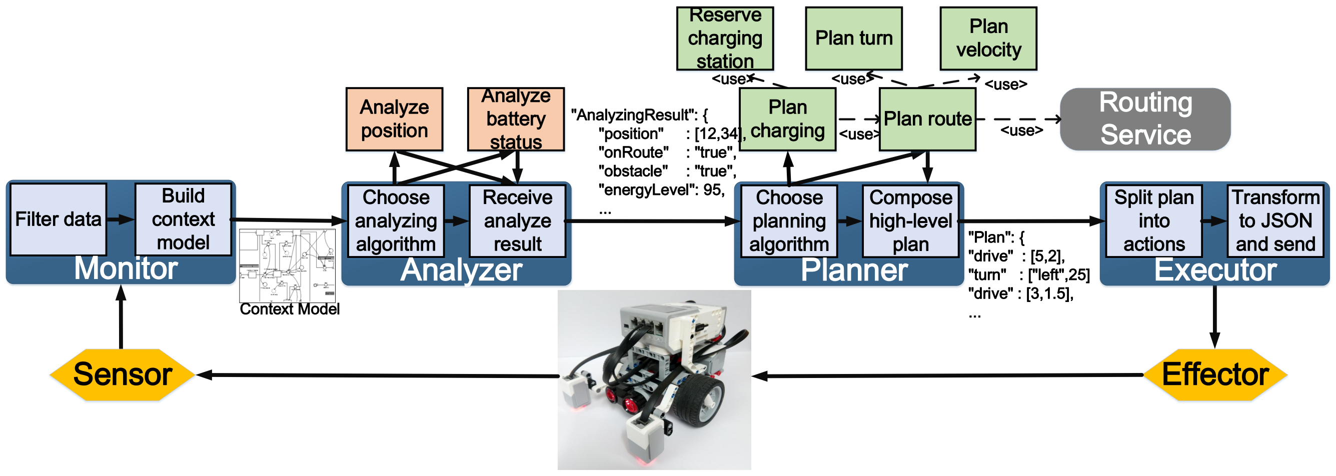
B. Implementation of RoCoSys for Robot Coordination

Our coordination system is designed as a self-adaptive system which can adapt the system's resources – here: the robots – autonomously without user interaction [7]. In this paper, we used the Java-based version of the FESAS framework [8] for the implementation of RoCoSys. The framework simplifies the creation and deployment of MAPE components. Furthermore, FESAS abstracts from common issues, e.g., the communication between components in the adaptation logic or the connection between managed resources and the adaptation logic. Therefore, developers are able to concentrate on algorithms within the MAPE components. FESAS clearly separates the workload for implementing an adaptation logic into two domain responsibilities. Designers specify the general architecture of the system including the components and their connections. Developers implement the functionality and algorithms for the MAPE components. FESAS offers the FESAS IDE for both roles which allows to design and implement the adaptation logic using plug-ins for the Eclipse IDE [8]. As RoCoSys uses a centralized adaptation logic, we use the `Central Adaptation Logic` design pattern that is offered by the FESAS IDE. Furthermore, we used the FESAS IDE for implementing functionality and algorithms of the MAPE components. Figure 3 shows RoCoSys and the integration of specific algorithms for analyzing and planning. In the following, we explain the implementation of the components.

The `sensors` and `effectors` receive input data from the robots as JSON strings or send JSON strings with instructions to robots, respectively. Sensors and effectors use the socket module provided by FESAS. The `monitor` aggregates all sensor data and forwards it to the `analyzer` every time the sensor receives messages with updated data. Therefore, the monitor creates a system model that captures the robots' states and a context model that represents the environment. Modularization of the components enables reusing the existing data exchange protocol and the monitoring algorithm, however, developers can customize them for specific purposes using the FESAS IDE.

The `analyzer` determines whether the current system state requires adaptation (e.g., updating a robot's route). Therefore, it calls the analyzing algorithms to evaluate the current system state. These algorithms are interchangeable and need to implement the interface `IAnalyzingAlgorithm`. Figure 3 shows two example algorithms. One compares the current position of a robot with its route/ destination and controls its correctness. The second algorithm analyzes the battery status. Both may trigger planning an adaptation, here, either adjusting the robot's route or charging the battery. However, in the example, only the algorithm for routing is triggered as the analyzer identified an unknown obstacle. As a result, the algorithm returns a list containing robot IDs and the corresponding `AnalyzingResult`. The analyzing algorithms may be application-specific, but existing algorithms can be reused easily through the modular design of the system.

After receiving a result from the analyzer, the `planner` calls the planning algorithms. Developers using RoCoSys have to implement these planning algorithms for controlling the adaptation. Different algorithms may be implemented that react to different analyzing results. These algorithms use *basic planning modules* (e.g., turn, adapt velocity, change lane) that achieve specific adaptation actions. As mentioned, in the example shown in Figure 3, we assume the analyzer triggered an adjustment of the robot's route, only. Therefore, RoCoSys takes various factors into account – such as the position of robots and obstacles as well as the destination and structure of the environment – to calculate an optimal route for a robot. In order to achieve this, RoCoSys offers a *routing service*. This global service provides the functionality needed for navigation, such as calculating the best route for a robot with a certain destination or requesting information about the current distance between two robots. In case of planning a route, the planner requests a route from the routing service. Then, it uses the basic planning modules to configure a set of actions – e.g., drive x meters with velocity α, turn left with angle β, drive y meters with velocity γ – to reach the destination. Figure 3 shows the flow for planning a route. As these actions change parameters of the robots, robots have to offer corresponding methods. Using our self-driving robots, developers can reuse the existing basic planning modules. An adaptation plan is further processed by the `executor`. The executor splits the plan into instructions for changing parameters of the robots. Following, the `effector` marshals the instructions to the JSON format and sends them to the robots.



```
"AnalyzingResult": {
  "position"    : [12,34],
  "onRoute"     : "true",
  "obstacle"    : "true",
  "energyLevel": 95,
  ...
```

```
"Plan": {
"drive"  : [5,2],
"turn"   : ["left",25]
"drive" : [3,1.5],
...
```

Figure 3: Architecture of RoCoSys with example algorithms (represented by green and orange boxes) for analysis and planning.

Developers can reuse all components and algorithms we implemented, especially the monitoring and executing component as well as the sensor/ effector components for the interaction with robots. Using the FESAS IDE, developers can customize existing or implement new algorithms within the analyzer and planner components. Furthermore, the robots can be reused and, by doing so, developers can use our set of basic planning modules. Hence, developers only have to implement the application-specific analyzing and planning algorithms for their application. Extensibility and reusability are supported by the modular design. Furthermore, FESAS offers decentralization by design. This makes it possible to build a decentralized system for coordination, e.g., for large systems with many robots or systems controlling a large area. FESAS handles decentralization issues, such as (un)marshalling of data within the adaptation logic. Developers only have to change the algorithms used for analyzing and planning of adaptations. In the next section, we evaluate the usability of RoCoSys within two applications for coordinating robot systems.

IV. Evaluation

This section presents the implementation of two systems: a demonstrator for a platooning coordination system (PCS) as well as a smart vacuum cleaner (SVC). We evaluate both systems quantitatively and qualitatively.

A. *Application Domains*

This section describes the two applications: (i) the PCS and (ii) the SVC. In the following we describe the implementation of these systems using RoCoSys, however, due to space constraints we do not present the details of the algorithms for analyzing and planning.

1) Platooning Coordination System (PCS): Platooning is driving in convoys of semi-automated vehicles with a distance of only a few meters between them [9]. Vehicles need to drive autonomously or at least support the driver in holding the distance and keeping the vehicle within the lane boundaries. In our scenario, the Platoon Coordination System (PCS) coordinates the formation of platoons. It receives information from drivers, e.g., their destination, searches a suitable platoon, and navigates the vehicle to the platoon. After reaching the platoon, a vehicle uses vehicle-to-vehicle communication as well as sensors (e.g., distance sensors) for controlling the joining process. Once in a platoon, the vehicle autonomously keeps the lane and the distance to preceding vehicles. Further, if a platoon meets another platoon, the PCS decides whether they should merge or overtake. The PCS receives constantly updates from vehicles about their positions and, hence, can determine when a vehicle should leave a platoon or a platoon should be dissolved. Figure 4 shows the platooning process on a highway: A vehicle leaves `platoon (1)`, `platoon (2)` overtakes `platoon (3)` and, furthermore, an additional vehicle joins `platoon (2)`.

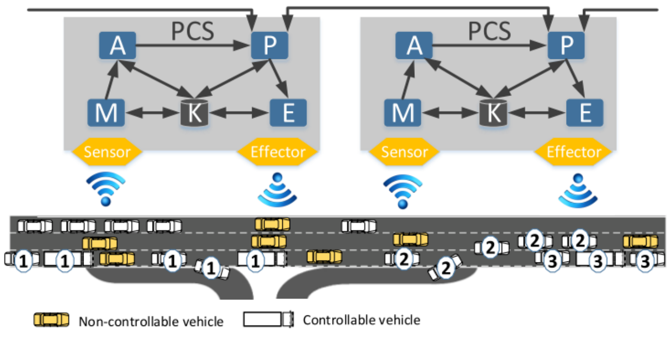


Figure 4: The platooning process on a highway.

Using RoCoSys, we built a PCS that coordinates the vehicles and platoons. In our demonstrator, our self-driving robots act as self-driving vehicles. We implemented analyzing and planning algorithms for platooning coordination[1]. However, RoCoSys's flexible design enables to use the system in other applications where self-driving robots should be coordinated, e.g., warehouse logistics. Therefore, developers have to customize the analyzing and planning algorithms.

2) Smart Vacuum Cleaner (SVC): As second use case, we used RoCoSys for implementing a SVC system. There are already commercial products available, however, they often miss coordination between multiple SVC agents as well as smartness in the path finding. One of our self-driving robots acts as SVC. The distribution of dirt in the environment is unknown to the robot. The robot can determine its current

[1]A movie showing the platooning approach can be found at: https://www.youtube.com/watch?v=Nnrbq-4Dn24

location. Without coordination, the robot decides a random direction and moves left, right, or forwards.

RoCoSys adds smartness to the robot by acting as smart path finding system. It analyzes the sensor data of the robot and adjusts the robot's next actions. One of the most important advantages of the system is the self-adaptation: During the cleaning process, the self-adaptive system collects data about the environment, e.g., obstacles. All collected data is stored in a map of the environment, so that RoCoSys can use it as input for path finding algorithms. Further, RoCoSys can switch between different algorithms for different set-ups of rooms, e.g., depending on the presence of obstacles. In total, we implemented three different algorithms. During cleaning, the adaptation logic (implemented with RoCoSys) monitors and analyzes the sensor data received from the robot. When the system detects an obstacle in the current path of the robot, the adaptation logic is responsible to calculate a new path for cleaning and to send new instructions to the robot. Obstacles are stored in the map, in order to improve future navigation. Additionally, we implemented a simulator for analyzing the best algorithm for different room set-ups.

B. Discussion

We evaluated our systems with different parameters to show the usability of RoCoSys. Within the platooning scenario, we measured the time for communication between robots and the adaptation logic. More specifically, we measured the time for check-in at the system, hence, the establishment of the first connection from a robot to RoCoSys. In order to exclude defective measurements, 4 of the 50 runs (8 %) were omitted prior to data analysis. Table I shows the results.

Table I: Time (in milliseconds) for the registration of vehicles at RoCoSys (SD = Standard deviation).

	PCS	**Robot**		**Comm.**	**Total**
		Decode JSON	**Connection**		
Minimum	297ms	558ms	1068ms	562ms	2670ms
Maximum	563ms	799ms	1311ms	831ms	3253ms
Average	409ms	667ms	1194ms	655ms	2925ms
Median	407ms	662ms	1199ms	659ms	2933ms
SD	59.6ms	46.4ms	62.7ms	54.1ms	118.7ms

The check-in process takes up to 3 seconds. In detail, the robot is accountable for around 60% of the time and, here, most of the time (around 40%) is needed for setting up the connection. The reason may be that Mindstorms do not support Wifi natively. The LeJOS OS integrates the drivers for Wifi and, hence, they are not optimized for the robots' hardware. Bluetooth is supported natively by Mindstorms and might reduce the check-in time. However, this does not influence the driving behavior of the robot as this is done autonomously and separated from communication. Hence, the time is acceptable. Furthermore, it might be possible to set-up the communication first and subsequently start driving. After initially setting up the connection, subsequent communication is faster. Additionally, the evaluation shows that only a small amount of time accounts for the actual coordination at the PCS. This shows that the coordination with RoCoSys is rather fast.

The second measurement shows the flexibility of RoCoSys through exchangeable analyzing and planning algorithms. We evaluated the cleaning time (in simulation steps) needed by the SVC robot for different room settings (we used the simulator for simplification reasons). Table II shows the steps needed on average (50 runs per algorithm) for the exploration of the room and the cleaning, respectively. Without further touching the system, the performance can be increased by just replacing the analyzing and planning algorithms. It is possible to save up to 93.3% time for the exploration of the room and up to 90.2% for cleaning. Through the flexibility introduced by RoCoSys new algorithms can be loaded at runtime.

Table II: Steps needed on average by the algorithms for exploration of the room and cleaning, respectively. `RandomExplore` and `RandomClean` are used without RoCoSys.

	1 Room, no obstacles	**1 Room, static obstacles**	**3 Rooms, static obstacles**	**3 Rooms, moving obstacles**
RandomExplore	2247	2580	3885	3969
SmartExplore	191	258	260	374
RandomClean	1379	1117	2375	2835
CircleClean	1500	1842	2205	2507
SmartClean	168	181	261	307

Both systems share a common robot platform based on LEGO Mindstorms. These robots can be used in many more scenarios where self-driving robots are needed and, furthermore, they can be extended easily. The evaluation shows the flexibility of the system. On the one hand, the robots can be used in different applications that integrate self-driving robots. The flexibility of the Mindstorms toolkit offers the possibility to extend the functionality, e.g., add an arm. On the other hand, the coordination system, RoCoSys, is reusable. We showed its usage in two applications, but due to its modular design, the application domain is unlimited.

V. Related Work

In this section, we present related work in coordinating mobile IoT devices and robots in the domains of self-adaptive systems, platooning coordination, and SVC.

First, we focus on approaches in the field of self-adaptive systems that target the coordination of mobile IoT devices or robots. Kramer and Magee present a framework for implementing self-adaptive systems [3]. One of their use cases is the domain of robotics. However, they rather focus on using the framework for one robot than on coordinating many robots. In [10], the authors present an approach for using models in the MAPE-K cycle for guaranteeing the validity of adaptation actions. They prove their concept in a system with transportation robots in a warehouse. However, they do not target coordination. Other frameworks in the field of self-adaptive systems, e.g., *Rainbow* [11], *SASSY* [12], or *ArchStudio* [7], offer approaches for building an adaptation logic of a self-adaptive system, however, neither do they focus on the coordination of entities nor on robotics. In [13], we presented a framework for coordinating pervasive applications. However, the implementation is based on a specific middleware that do not target robots.

Different approaches have proven the feasibility of platooning, e.g., PATH [14], SARTRE [9], KONVOI [15], or

COMPANION [16]. In the COMPANION project, heuristics for calculating the optimal route for platoons are used [16]. The SARTRE approach mentions that a back office will do platoon coordination [9], but the authors do not specify how. PATH offers a concept for an automated highway [14]. In [15], the authors present a central system for coordinating platoons using data mining algorithms. The approach is comparable to the PCS, however, the PCS integrates individual factors for matching platoons rather than similar routes only. However, to the best of our knowledge, none of these approaches compare different strategies that try to optimize different objectives. Our modular system simplifies the evaluation of strategies through the possibility of exchanging algorithms. This is out of scope of this work.

Besides of commercial cleaning robots, several research prototypes exist. Luo *et al.* propose an approach for area-covering operations with obstacle avoidance inspired by neural networks [17]. Doh *et al.* focus on path generation and low computational load [18]. Okazaki *et al.* present a SVC that uses an arm-like mechanism in order to improve cleaning results [19]. A recent work from 2014 concentrates on the path planning for autonomous vacuum cleaner robots [20]. Kang *et al.* present a SVC with focus on robust obstacle detection mechanisms by using an infrared line in front of the robot [21]. To the best of our knowledge, none of the research concentrates on coordination of SVC robots.

Further areas of related work are rescue robots, agricultural robots, or military robots. We exclude them due to space constraints. However, to the best of our knowledge, there is no approach in literature that integrates a reusable and expandable robots platform with a flexible and easily customizable coordination framework for simplifying development of self-driving robots. This is the focus in this paper.

VI. Conclusion

In this paper, we presented RoCoSys, a flexible framework for coordination of mobile IoT devices. Additionally, we implemented self-driving Mindstroms robots. Following our design, we present a reference implementation of RoCoSys in Java for coordinating robots. Developers can use our implementations of RoCoSys and the robots as a development toolkit for their own applications. The toolkit offers capabilities for (i) autonomously driving of robots, (ii) communication between robots and RoCoSys, (iii) pre-processing of data in RoCoSys, and (iv) sending of instructions to the robots. Developers may customize the existing procedures for analysis and planning of adaptation. Using this toolkit, we built systems in two application domains for showing its flexibility.

As future work, we plan to implement further applications in other domains of mobile IoT, e.g., an autonomous production facility or coordination of applications in pervasive environments. Additionally, we plan to extend the existing systems. For the SVC, we will integrate the existing simulation in the adaptation logic. This enables the adaptation logic to find the best algorithm for a specific setting and to learn new algorithms at run time. In this paper, we presented our demonstrator of the PCS coordinating Mindstorms robots. In order to examine the approach on a realistic level, it would be suitable to use a platooning simulator. Currently, we integrate the PCS and the platooning simulator *PLEXE* [22] into a testbed for infrastructure-aided platooning coordination that offers a flexible approach for evaluating various platooning strategies.

References

[1] K. Johansson, G. Pappas, P. Tabuada, and C. Tomlin, "Special issue on control of cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3120–3121, 2014.

[2] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," in *Proc. FOSE*, 2007, pp. 259–268.

[4] C. Krupitzer, F. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, no. Part B, pp. 184–206, 2015.

[5] S. Kim and J. Jeon, "Introduction for Freshmen to Embedded Systems Using LEGO Mindstorms," *IEEE Transactions on Education*, vol. 52, no. 1, pp. 99–108, 2009.

[6] LeJOS, "leJOS EV3," 2016, URL: http://www.lejos.org/ev3.php.

[7] P. Oreizy et al., "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intellgent Systems*, vol. 14, no. 3, pp. 54–62, 1999.

[8] C. Krupitzer, F. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr, "FESAS IDE: An Integrated Development Environment for Autonomic Computing," in *Proc. ICAC*, 2016, pp. 15–24.

[9] T. Robinson, E. Chan, and E. Coelingh, "Operating Platoons On Public Motorways: An Introduction To The SARTRE Platooning Programme," in *Proc. ITSWC*, 2010.

[10] M. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," in *Proc. SEAMS*, 2014, pp. 125–134.

[11] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[12] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, "SASSY: A Framework for Self-Architecting Service-Oriented Systems," *IEEE Software*, vol. 28, no. 6, pp. 78–85, 2011.

[13] V. Majuntke, S. VanSyckel, D. Schäfer, C. Krupitzer, G. Schiele, and C. Becker, "COMITY: Coordinated application adaptation in multi-platform pervasive systems," in *Proc. PerCom*, 2013, pp. 11–19.

[14] R. Horowitz and P. Varaiya, "Control design of an automated highway system," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 913–925, 2000.

[15] R. Kunze, R. Ramakers, K. Henning, and S. Jeschke, "Organization and operation of electronically coupled truck platoons on german motorways," in *LNCS 5928*, 2009, pp. 135–146.

[16] J. Larson, K. Liang, and K. Johansson, "A distributed framework for coordinated heavy-duty vehicle platooning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 419–429, 2015.

[17] C. Luo, S. X. Yang, and X. Yuan, "Real-time Area-Covering Operations with Obstacle Avoidance for Cleaning Robots," in *Proc. IROS*, 2002, pp. 2359–2364.

[18] N. Doh, Kim, and W. Chung, "A practical path planner for the robotic vacuum cleaner in rectilinear environments," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 2, pp. 519–527, 2007.

[19] A. Okazaki, T. Senoo, J. Imae, T. Kobayashi, and G. Zhai, "Real-Time Optimization for Cleaner-Robot with Multi-Joint Arm," in *Proc. ICNSC*, 2009, pp. 885–890.

[20] K. Hasan, A.-N. Abdullah, and K. Reza, "Path Planning Algorithm Development for Autonomous Vacuum Cleaner Robots," in *Proc. ICIEV*, 2014.

[21] M.Kang, K.Kim, D. Noh, J. Han, and S. Ko, "A Robust Obstacle Detection Method for Robotic Vacuum Cleaners," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 587–595, 2014.

[22] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, and R. Lo Cigno, "PLEXE: A Platooning Extension for Veins," in *Proc. VNC*, 2014, pp. 53–60.