Middleware Platform for Distributed Applications Incorporating Robots, Sensors and the Cloud

Elias De Coninck, Steven Bohez, Sam Leroux, Tim Verbelen,
Bert Vankeirsbilck, Bart Dhoedt and Pieter Simoens
Ghent University - iMinds
iGent, Technologiepark Zwijnaarde 15
B-9052 Gent, Belgium
Email: {elias.deconinck, steven.bohez, sam.leroux, tim.verbelen,
bert.vankeirsbilck, bart.dhoedt, pieter.simoens}@intec.ugent.be

***Abstract*—Cyber-physical systems in the factory of the future will consist of cloud-hosted software governing an agile production process executed by autonomous mobile robots and controlled by analyzing the data from a vast number of sensors. CPSs thus operate on a distributed production floor infrastructure and the set-up continuously changes with each new manufacturing task. In this paper, we present our OSGi-based middleware that abstracts the deployment of service-based CPS software components on the underlying distributed platform comprising robots, actuators, sensors and the cloud. Moreover, our middleware provides specific support to develop components based on artificial neural networks, a technique that recently became very popular for sensor data analytics and robot actuation. We demonstrate a system where a robot takes actions based on the input from sensors in its vicinity.**

I. Introduction

The term Industry 4.0 refers to a vision on future manufacturing environments with smart systems and production facilities autonomously exchanging information, triggering actions and controlling each other independently [1]. The integration of the Internet-of-Things (IoT) in the manufacturing process is a key enabler, as it delivers the necessary information for context-aware assistance of people, machines and robots active on the production floor in the execution of their tasks.

With manufacturing moving to high-mix, low-volume production with high cycle rates, factory cyber-physical systems (CPS) must be able to flexibly accommodate changing production floor configurations. Agile manufacturing thus requires a CPS software design that adheres to the principles of modularity, service orientation and decentralization [2]. Sensors, actuators, factory robots as well as cloud-hosted components should be dynamically discoverable as services that can be combined to realize distributed CPS applications.

In this paper, we present the design and implementation of a middleware solution allowing developers to build CPSs comprised of services communicating through well defined service interfaces. While the component-based approach is applicable to many CPSs, we primarily target scenarios in which the information of sensor networks is used to control factory robots. We deploy an optimized component runtime on sensor gateways, robots and the (edge) cloud that abstracts the deployment of and communication between these distributed components. The middleware dynamically discovers attached robots and sensors and exposes these as a service.

One key feature of our middleware is the advanced support for components that make use of Artificial Neural Networks (ANN). ANNs are a family of computational models, loosely inspired by the human brain, that are used to accurately classify and recognize patterns from large amounts of unstructured data [3]. ANNs are able to generalize sytem input, and are very well suited to discover patterns and take similar decisions in similar conditions. This is important in realistic environments, where various factors may impact the fidelity of sensor data, for example, light conditions, noise based on time of the day, etc. Although the technique is known for a few decades, only very recently important breakthroughs were achieved by significantly increasing the number of computational elements (neurons) in the ANN. These *deep* neural networks are very useful at both sensing and actuation endpoints of CPSs, e.g. for image classification [4], speech recognition [5] and visuomotor robotic control [6].

The remainder of this paper is structured as follows. Section II summarizes the related work done in intelligence for factory robotics and robotics in an IoT environment. Section III presents the overall architecture of our middleware solution. In Section IV, AIOLOS is introduced which enables applications to be deployed and distributed on a wide variety of devices. Section V describes the Thing Abstraction Layer (TAL) responsible for providing abstraction interfaces for dynamically discovered sensors and actuators. In Section VI, we introduce the DIANNE framework to manage, build, train and deploy neural networks on compute devices. Section VII showcases the preliminary results and Section VIII concludes this paper.

II. Related Work

Robots, sensors/actuators and server (cloud) systems are the three pillars of any CPS. Most related work has focused on the integration of two of these three pillars.

In [7], the authors propose an IoT architecture for 'things' from industrial environments. The proposed architecture is based on the OPC.NET specification and is built around two components: the data server and the client application. The

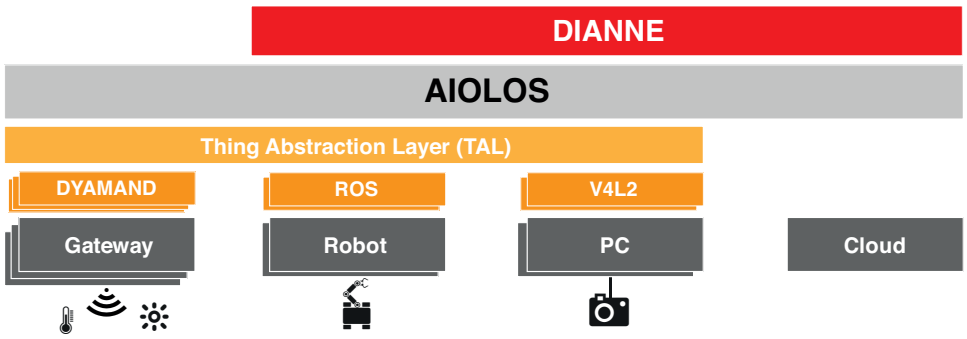

Fig. 1. Architectural layers of our middleware platform. AIOLOS abstracts the distributed deployment of components across nodes with varying hardware architectures and processing power. The Thing Abstraction Layer provides the necessary interfaces for low data rate sensors, robots and high data sensor such as cameras. DIANNE provides additional support for components based on artificial neural networks.

data server collects sensor information and sends commands to actuators while the client application is the front-end application of this data server. The data server can acquire sensor data from fieldbuses (BACnet, LonWork, CANopen, Modbus, EtherCAT), used in smart homes and industrial environments.

The support of the cloud brings various benefits for robots. Riazuelo et al. [8], [9] illustrated the benefit of offloading Simultaneous Localization And Mapping (SLAM) from a mobile robot to the cloud. By allocating the computationally expensive tasks to the cloud they can decrease the costs and power of the robot computer by limiting the on-board processing to simple camera tracking. Bekris et al. [10] propose an architecture to take advantage of splitting computation between the cloud and the robot for motion planning and manipulation, with the additional benefit of environment knowledge sharing. The "Lighting" framework [11] uses the cloud for collective robot learning by distributing planning and trajectory adjustments from the indexed trajectories from many robots. A more detailed survey on cloud robotics and automation can be found in [12].

Chibani et al. [13] discuss the challenges and trends of ubiquitous robots and categorize these into three major topics: 1) Making robots more autonomic; 2) Social awareness and affective interaction; 3) Engineering of ubiquitous robotic platforms. Concerning the last item, the authors point out that an ubiquitous robotic platform should address the issue of connecting robots with smart devices, provide a middleware layer to create plug and play applications and add the ability of providing intelligence to these robots.

The middleware discussed in this paper provides a combined solution of the previous mentioned work, enhancing the developers' options and giving them the power to extend the middleware. Our proposal focuses on the abstraction of things by making available high level control of devices into a middleware.

III. Design Overview

The approach discussed here provides a unifying development platform to connect sensors to robots. Developers can create their application as a set of loosely coupled components. The middleware abstracts the deployment on the underlying set of nodes comprising sensor gateways, robots and cloud infrastructure. Figure 1 shows the different architectural layers of the system.

The previously developed OSGi-based AIOLOS approach [14] forms the foundation of our system, enabling to deploy components of one application over distributed nodes in a way that is transparent to the application developer. The OSGi [15] runtime is supported on various types of hardware architectures. Developers create components either immediately on AIOLOS, or by leveraging on additional functionality provided by the DIANNE layer. DIANNE provides supporting services and abstractions to develop components using artificial neural networks, and is further discussed in section VI. AIOLOS-compatible service interfaces for input and output devices are provided by the Thing Abstraction Layer (TAL), which is responsible for all communication between application components and sensors and robots. We have implemented TAL wrappers for ROS and DYAMAND, which are platforms for controlling robots and sensors respectively. They are further discussed in section V.

The created components are distributed in the form of bundle packages (jar files) that can be started on any device hosting an AIOLOS runtime. Additional bundles can be started or components can be migrated between runtimes to rebalance the computational workload. This creates a very dynamic environment where sensors and actuators can go on- or off-line at any moment. When connection is lost to a remote runtime, the middleware can launch a local version of the remote service.

IV. Distributed Software

AIOLOS [14] is our open-source framework that enables component-based application models to be deployed on multiple devices without the developer having to manage the inter-component communication[1]. The framework is based on an OSGi runtime, compatible with a multitude of heterogeneous devices, ranging from constraitned devices such as Raspberry Pi and Intel Edison up to high-end containers or virtual

[1]Source and documentation available at http://aiolos.intec.ugent.be

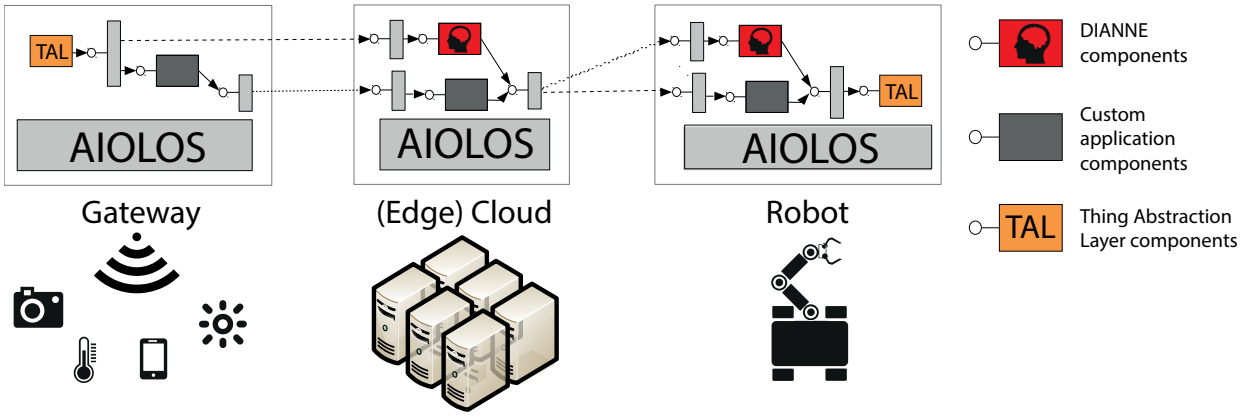

Fig. 2. AIOLOS framework is able to run on client devices (with support for Java vm) and in the (edge) cloud. Every service interface is proxied by AIOLOS, allowing to transparently distribute software components, switch between remote and local execution, or scale out to remote infrastructures.

machines in the cloud. Recent work by other authors indicated that the performance overhead of OSGi on embedded devices, such as sensor gateways, is negligible [16].

The underlying principles of AIOLOS are demonstrated in Figure 2. The AIOLOS runtime instances deployed on various devices in the same IP subnet are able to discover each other automatically. AIOLOS creates proxies for every component service interface running on the same node, as well as for the interfaces imported from services running on other nodes. Method calls are forwarded by the proxy to a component instance implementing the service interface. This component instance can run on the same node (continuous line on the figure) or on a remote node (dashed line).

Proxy policies are used when multiple implementations of the same service interface are available, as these are hidden behind the same proxy. Because each service call is intercepted by the AIOLOS proxies we can gather monitoring information such as link delay, execution time, return value and argument size. Based on this monitoring information a runtime model can be created to help the proxy policies take decisions which implementation candidate to pick. One example scenario is cloud offloading of computationally intensive components. AIOLOS proxy policies support dynamic trade-offs between parameters such as on-board processing and network communication, as for example presented in [17], where the authors compare various robot-cloud configurations for the components involved in robot navigation. Another scenario is a big/little artificial neural network deployment [18], [19], where the big neural network on the cloud is only executed if the confidence of the small neural network is considered too low.

V. Thing Abstraction Layer

The Thing Abstraction Layer (TAL) provides a 'Thing' interface that exposes physical devices as an OSGi service. There are two types of 'things': sensors and actors, and consequently we have created implementations of sensors and robotic actors.

A. General Sensors and Actors

TAL has an abstraction for a wide range of things with sensing and actuation functionality. Things are exposed as services of a given type. Sensing types include temperature, light, contact, camera, and actuator types are e.g. lamp, lock, etc. Each type interface specifies appropriate getters and setters for the properties of that device type. This abstraction enables developers to create their own devices or wrap existing frameworks for ease of development.

To discover sensors and actors with proprietary communication interfaces and command syntax, such as EnOcean, ZigBee, Philips Hue, common USB devices, etc. we use the DYAMAND [20] framework. DYAMAND is an extendable interoperability framework for service discovery and device access protocols. A custom plugin wraps DYAMAND as a TAL provider and exports all discovered devices as TAL services.

Video4Linux 2 (V4L2) is an other standard we included in TAL to support realtime video capture on Linux systems. By implementing a V4L2 TAL provider multimedia sensors, such as webcams, TV tuners, etc., are exposed as OSGi services and discoverable as 'things'.

B. Robotic Actors

The Robot Operating System (ROS) [21] is a set of software libraries, tools and conventions designed to facilitate the creation of complex behaviours for robotic platforms. ROS is designed to be modular so users can pick and choose the combination of modules that works for them and do not waste robot resources. ROS provides a service-based API to control robotics from within an application.

At the basic level the ROS middleware offers a message passing interface that provides inter-process communication for robotics applications. Further, ROS provides a number of core features we use in our system:

1) A message passing system with a clear interface using the anonymous publish/subscribe mechanism defined in the message Interface Description Language (IDL).

2) Recording and playback of publish/subscribe messages.
3) Remote method calls to allow for synchronous request/response interaction between processes.
4) A robot description language to describe and model robots in the Unified Robot Description Format (URDF), which consists of XML document with the physical properties of the robot.

We opted for ROS because it has bindings for C, C++, Python and Java. Next to that it is supported by many robots and sensors, and interfaces with many robot simulators. To integrate ROS with our architecture we needed to create an OSGi service which is discoverable by our Thing Abstraction Layer.

One challenge to integrate robot control through ROS is the inherently asynchronous behavior of the publish/subscribe messaging mechanism. When a message is published the robot itself undertakes this action and updates the required topics. This means that the publisher needs to listen for the topic updates from the robot if the publisher wants to react upon it. OSGi services on the other hand are either blocking or "fire-and-forget". Hence, we needed a mechanism to insert a callback interface when the state of a specified topic changes.

Our solution is the ROS OSGi wrapper with support for asynchronous programming by using the OSGi's Promise API. A Promise is a holder for asynchronous calculations or computations which allows the user to register callbacks to notify the user when it is finished or has failed. Although these Promise objects are lambda friendly, this API itself has no hard dependencies on Java 8.

Listing 1. ROS OSGi service API.

```
public interface Robot {
  Promise<? extends Robot> waitFor(long time);
  Promise<? extends Robot> waitFor(Promise<?>
      condition);
}
public interface Arm extends Robot {
  Promise<Arm> setPosition(int joint, float position
      );
  Promise<Arm> setPositions(float... position);
}
public interface OmniDirectional extends Robot {
  Promise<OmniDirectional> move(float vx, float vy,
      float va);
}
```

Listing 1 shows a (part of the) interface towards a ROS-enabled robot. Developers can chain each action of the robot on a previous action or on a certain event. The Java Future class has similar behavior except that no callbacks can be registered and in the end a synchronous get() method needs to be called, which does not allow for chaining.

Listing 2. ROS component usage.

```
Arm arm = ... ; OmniDirectional base = ...;
arm.setPositions(4f, 2.2f, -1.4f, 2.6f, 1.25f)
   .then(p -> arm.setPosition(0, 2f))
   .then(p -> base.move(0.5f, 0f, 0f))
   .then(p -> base.waitFor(neuralNet.detectObject()))
   .then(p -> base.stop());
```

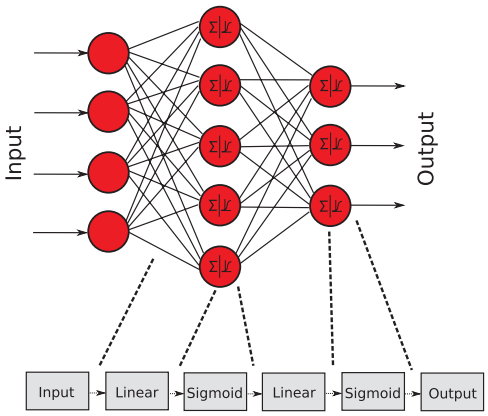

Fig. 3. A feed-forward fully connected Artificial Neural Network split up into a chain of DIANNE modules.

Looking at Listing 2 we can see the benefit of chaining actions and conditions. In this example we move a robot arm to a specific location taking into account the environment of our robot by moving around an object. Later when the arm is in position we instruct the base to drive forward. The base's 'move' method returns immediately because it is given a direction and speed. This mechanism enables us to wait until a predefined condition is achieved such as detecting an object with a neural network using the inputs of the environment.

VI. Distributed Intelligence

As motivated in section I, Artificial Neural Networks (ANN) are a key technique used in many relevant scenarios: pattern recognition of sensor data, robot control, etc. Our DIANNE[2] framework provides support to integrate ANNs in service-ased applications. We refer the reader to [22] for an in-depth discussion of DIANNE and limit the discussion below to the aspects relevant with respect to the scope of this paper.

In DIANNE, neural networks are constructed by defining Modules and their interconnections. We implemented a wide range of modules used as building blocks in state-of-the-art deep neural networks, such as *Convolution*, *MaxPooling*, *Softmax* and *Rectified Linear Units* modules [23]. The modules have two information flows: a forward pass, used during evaluation, and a backward pass, used during training to propagate the errors of the outputs. An Input Module forwards input data, e.g. from a sensor, to various processing modules. Output is collected by an Output Module. Figure 3 shows an example chain of modules for a fully connected neural network.

DIANNE modules are implemented as OSGi services. To account for device heterogeneity, different module implementations are available, including a pure Java based application as well as a CUDA-based implementation for GPU-enabled hosts.

[2]Source and documentation available at http://dianne.intec.ugent.be

Fig. 4. Demo setup of a KuKa Youbot with integrated compute node and a Jetson TK1 as a GPU enabled edge device.

VII. Use cases

To illustrate the features of this system, we use a neural network trained for object recognition to decide what a factory robot should do, e.g. pick and place objects, sorting screws in bins, or even harder tasks such as mounting a side-panel to a car. We created a proof-of-concept (see Figure 4) to sort objects on a conveyor belt. A KuKa Youbot is used as the factory robot in the environment and a camera is installed to monitor the area of the conveyor belt. To demonstrate the benefits of offloading DIANNE modules we used a Jetson TK1 as the GPU enabled edge cloud and the KuKa embedded PC equipped with an Intel Atom D510 as the internal compute device of the factory robot.

DIANNE is deployed with the OverFeat [24] neural network model, which has two pre-trained parameter sets available, accurate and fast. In this test we aim to proof the offloading benefits of our framework so we opted for the accurate model on the edge, which requires more computation, and the fast model locally on the robot. OverFeat is an image classifier built around a convolutional network that is trained on the ImageNet 1K dataset. This dataset has 1000 different classes of which we choose the objects to sort. The camera feed is streamed to the first layer of the neural network which forwards the outputs to the next layers until the output layer is reached. The output classifies the camera feed, emitting a 1000-component vector. Each component indicates the probability that the object belongs to one of the 1000 predefined classes. Based on the object with the highest probability and a minimum threshold, a decision is made to steer the robot with the correct action or do nothing when no object is detected.

Figure 5 shows the achieved frame rate when using OverFeat fast on the embedded pc compared to the frame rate

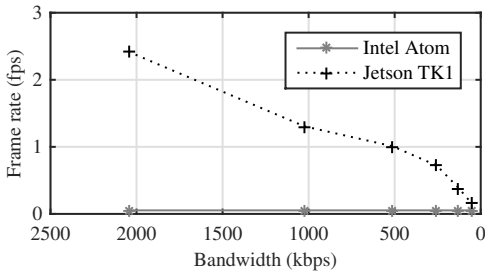

Fig. 5. Frame rate during object classification using OverFeat accurate on Jetson TK1 and OverFeat fast on KuKa embedded PC (Intel Atom CPU D510 4x@1.66Ghz).

while forwarding frames to the edge cloud with a GPU enabled device. We decreased the bandwidth to the edge cloud, which is an acceptable use case for wireless mobile robots, until the link was disconnected. The results show that offloading the neural network to GPU enabled devices is always better than using embedded CPU compute power. The only reason the neural network should be deployed locally is as a back-up case. Keep in mind that during this experiment only the bandwidth was altered while the latency was kept the same. A policy can be created to switch between local and remote neural networks based on given criteria such as latency, bandwidth, required accuracy, etc. If the robot looses connection to the access point it switches the camera feed to the locally deployed neural network so the robot can still operate but with a slower frame rate.

A other possible use case is to share a deployed neural network between multiple robots. Each mobile robot has DIANNE with OverFeat fast deployed locally and when it connects to a wireless network the robot discovers all other DIANNE runtimes in the environment. This enables each robot to pick the best DIANNE runtime in the same subnet, based on a policy, and forward the camera feed to this runtime. Adding a powerful GPU device with a DIANNE runtime in such an environment would benefit all mobile robots.

VIII. Conclusion and future work

This paper proposes a modular middleware platform facilitating the development of applications with components distributed over robots, cloud and sensor systems. The 'Things Abstraction Layer' (TAL) creates an abstraction for common devices, sensors and robots. A ROS OSGi interface was introduced to cope with the asynchronous behavior of ROS's publish/subscribe mechanism. We introduced DIANNE which is able to build, train, evaluate and deploy Artificial Neural Networks (ANN) utilizing specialized hardware if available. Using AIOLOS as the foundation of this middleware we are able to transparently distribute the intelligence between sensors and actuators. This enables us to offload resource intensive parts of ANNs to the cloud or powerful edge devices.

Currently the output of the ANNs control the robots through preprogrammed actions. In the future we will train ANNs with the inputs of the environment to directly control the factory robots with the output of these trained networks. A

ROS simulator can be used to pre-train and evaluate the outputs of ANNs while the actual deployment to a factory robot can be enhanced by on-line training. By adding end-to-end reinforcement learning to DIANNE we can enhance the efficiency or accuracy of the robots while they are deployed into an IoT environment.

ACKNOWLEDGMENT

Part of the work was supported by the iMinds IoT research program. Steven Bohez is funded by Ph.D. grant of the Agency for Innovation by Science and Technology in Flanders (IWT). We would also like to acknowledge NVIDIA for providing us with GPU hardware.

REFERENCES

[1] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster, *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group*. Forschungsunion, 2013.

[2] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios: a literature review," *Technische Universität Dortmund, Dortmund*, 2015.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[5] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39–48, 2015.

[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-522.html

[7] I. Ungurean, N. C. Gaitan, and V. G. Gaitan, "An iot architecture for things from industrial environment," in *Communications (COMM), 2014 10th International Conference on*, May 2014, pp. 1–4.

[8] L. Riazuelo, J. Civera, and J. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401 – 413, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889013002248

[9] L. Riazuelo, M. Tenorth, D. Di Marco, M. Salas, L. Mösenlechner, L. Kunze, M. Beetz, J. Tardos, L. Montano, and J. Montiel, "Roboearth web-enabled and knowledge-based active perception," in *IROS Workshop on AI-based Robotics*, 2013.

[10] K. Bekris, R. Shome, A. Krontiris, and A. Dobson, "Cloud automation: Precomputing roadmaps for flexible manipulation," *IEEE Robotics Automation Magazine*, vol. 22, no. 2, pp. 41–50, June 2015.

[11] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 3671–3678.

[12] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, April 2015.

[13] A. Chibani, Y. Amirat, S. Mohammed, E. Matson, N. Hagita, and M. Barreto, "Ubiquitous robotics: Recent challenges and future trends," *Robotics and Autonomous Systems*, vol. 61, no. 11, pp. 1162 – 1172, 2013, ubiquitous Robotics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889013000572

[14] S. Bohez, E. De Coninck, T. Verbelen, P. Simoens, and B. Dhoedt, "Enabling component-based mobile cloud computing with the aiolos middleware," in *13e Workshop on Adaptive and Reflective Middleware, Proceedings*, 2014, pp. 1–6.

[15] The OSGi Alliance, *OSGi Service Platform, Core Release 5*. aQute, 2012.

[16] M. Stusek, J. Hosek, D. Kovac, P. Masek, P. Cika, J. Masek, and F. Kropfl, "Performance analysis of the osgi-based iot frameworks on restricted devices as enablers for connected-home," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2015 7th International Congress on*. IEEE, 2015, pp. 178–183.

[17] J. Salmeron-Garcia, P. Inigo-Blasco, F. Diaz-del Rio, and D. Cagigas-Muniz, "A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 444–454, 2015.

[18] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2015 International Conference on*. IEEE, 2015, pp. 124–132.

[19] S. Leroux, S. Bohez, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "Resource-constrained classification using a cascade of neural network layers," in *International Joint Conference on Neural Networks, Proceedings*, 2015, pp. 1–7.

[20] J. Nelis, T. Verschueren, D. Verslype, and C. Develder, "Dyamand: dynamic, adaptive management of networks and devices," in *Conference on Local Computer Networks*, T. Pfeifer, A. Jayasumana, and D. Turgut, Eds. IEEE, 2012, pp. 192–195.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[22] E. De Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, S. Leroux, and P. Simoens, "Dianne: Distributed artificial neural networks for the internet of things," in *2e Workshop on Middleware for Context-Aware applications in the IoT, Proceedings*, 2015, pp. 19–24.

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *ArXiv e-prints*, Dec. 2013.