Cloud Computing Support for the Multi-Agent Robot Navigation System

Submitted: 2nd January 2017; accepted: 11th April 2017

Wojciech Dudek, Wojciech Szynkiewicz, Tomasz Winiarski

DOI: 10.14313/JAMRIS_2-2017/18

Abstract:

This paper presents a navigation system structure for mobile service robots in the agent-based distributed architecture. The proposed navigation system is a part of the RAPP framework. The RAPP framework is an open-source software platform to support the creation and delivery of robotic applications, which are expected to increase the versatility and utility of robots. All key navigation tasks are defined and divided into separate components. The proper allocation of navigation components, in the four-agent structure of the RAPP infrastructure, enables high-demanding computations offloading and was the main goal of this work. Navigation system components were implemented using ROS framework. Experimental results for the NAO robot executing risks detection task proved the validity of the proposed approach.

Keywords: *robot navigation system, cloud computing, service robot*

1. Introduction

Service and social robots operate in human environment and support us in our everyday tasks. One of the core abilities of service mobile robots is autonomous navigation in unstructured environments. Such an area of operation requires from mobile robots additional effort during self-localization, environment map building, planning and motion execution tasks. In case of the environment change, robot controller needs to replan pre-planned motion path or in some situations even whole task. To satisfy above constrains, robot systems are aimed to gather great amount of data, that describe current state of the unstructured environment. In turn, processing such amount of data requires high computational power and massive storage capacity to compute and collect essential knowledge [16]. Navigation system is the core unit of every mobile robot controller, thus almost any decision it makes in complex control systems, is a result of a big data processing algorithm work.

To overcome above issues, robotic researchers develop multiple platform control systems. Authors of the survey [6] present current state of robot-cloud cooperation. They distinguish four potential profits of the Cloud in robotic systems:

- Big data – access to large packages of images, trajectories or descriptive data,
- Cloud computing – parallel grid computing, learning, planning,
- Collective Robot Learning – sharing trajectories and outcomes in multiple robot systems,
- Human Computation – image and video analysis, classification, learning and error recovery.

Among other robot behaviours, the Cloud is also used in robot navigation. The work [12] presents the C^2TAM system that is used to process visual SLAM – vSLAM. In this solution the environment map building task, using heavy computational power algorithms, process RGB-D data sent from multiple robots to the cloud. In work [13] authors focus on different approaches to processing distribution among build-in robot computer and the cloud. They consider stereo pair cameras image processing in the mobile robot teleoperation task. They present processing and transmission time of data in different image resolution and different wireless communication cases. The work shows that the proper distribution of processing has a crucial influence to stability and robustness of the robot control system. DAvinCi [1] project developers based the cloud system on ROS (Robot Operating System [4,11]) communication mechanisms and the Hadoop cluster [15]. It was used to process the FastSLAM (Fast Simultaneous Localization And Mapping) – algorithm in parallel. Rapyuta platform [10] allows dynamic allocation of robot safe processing environments (that are based on ROS). These environments are strictly conjugated one to another to allow information and services exchange between robots. In this solution bidirectional communication is based on WebSockets.

Above systems, apart from using the robot computer, benefit from the second platform – the Cloud. This solution has many advantages. It not only supplies robot controllers with additional computation power and storage capacity, but also reduces robot platform cost. Among the advantages of the Cloud computing, researchers note also: the reliability, large memory capacity, energy-saving, stable power supply, better use of resources and easier modernization of the Cloud then the on-board robot platform.

Most of known robot systems that are supported by the Cloud, are dedicated to a robot type, an application, an algorithm or perform a single service. We assume, that the proposed system is distributed among hardware platforms and modular to be able to extend its capabilities. In addition it enables robots to:

- store large packages of data,
- process data and draw conclusions,
- request computationally heavy services,
- learn objects,

- share robotic applications even if the robot types differ,
- share abilities and data with other robots.

The Cloud part of the RAPP system can handle many robots at once – it is composed of multi-threaded services. Furthermore, we assume that the navigation system will allow different type robots share the same task applications. Such applications (RApps) can be implemented by non-experienced programmer using the RAPP API and are stored in the Cloud. When a robot requires a certain ability, the robot downloads the RApp and interprets the RAPP API methods depending on the specific robot platform controller implementation.

The RAPP navigation system, as a part of the RAPP system that is oriented on the robot navigation tasks, is distributed across whole RAPP system structure. Our approach benefits from the theory of the embodied agent [18] and utilizes paradigms of this theory in the distributed navigation system structure synthesis. The system structure synthesis is one of the main issues of the presented work. In [2] the robot on-board part of the navigation system is presented. In this article we present the design, development and execution process of the cloud part of the RAPP navigation system. As a specification and modelling description standard, the System Modelling Language (SysML) [5] is used. Firstly, in Section 2, the general structure of the RAPP system is presented. In Section 3.1 we show constrains and requested behaviours (abilities) of our navigation system. Section 3.2 describes system components, that are responsible for the system abilities execution. The communication between the robot controller and the Cloud is described in Section 3.3. Succeeding Section 4 reveals the navigation system implementation and verification in the example task. Summary of the work is given in the Section 5.

2. RAPP System Structure

Development of a variable structure robot controller, that operates both in the robot and in the cloud platform, is the main goal of the RAPP project [9, 17]. We merge cloud services with on-board computing into one robot controller structure making the cloud essential for proper robot operation. In this article we present a navigation system that is a part of the distributed multi-agent robot control system. The Core Agent (a_{core}) operates in the onboard robot computer and is responsible for the direct control of the robot effectors and receptors. Depending on the user command, the a_{core} is able to launch additional agents or use services of Repository Agent (a_{rep}) that located in the cloud. Life cycle of a typical application – Dynamic Agent (a_{dyn}) is presented in Fig. 1. Implemented and built applications for all supported robots are stored in the RAPP Store in a_{rep}(step = 0). While a user requests the robot to realize a desired task (step = 1), the a_{core} downloads appropriate a_{dyn} from the RAPP store (step = 2). If the robotic application – RApp, is composed of two agents – Cloud Agent (a_{cloud}) and Dynamic Agent, the a_{cloud} is launched in the cloud (RAPP platform) and the a_{dyn} in the Robot platform (step = 3). Next the a_{dyn} as a master agent takes control over the Robot platform and using a_{rep} and a_{cloud} services is responsible for the user task completion (step = 4). Finally when the task is finished, the a_{dyn} and the corresponding a_{cloud} are terminated (step = 5). Detailed specification of the general structure of the RAPP system was presented in [17].

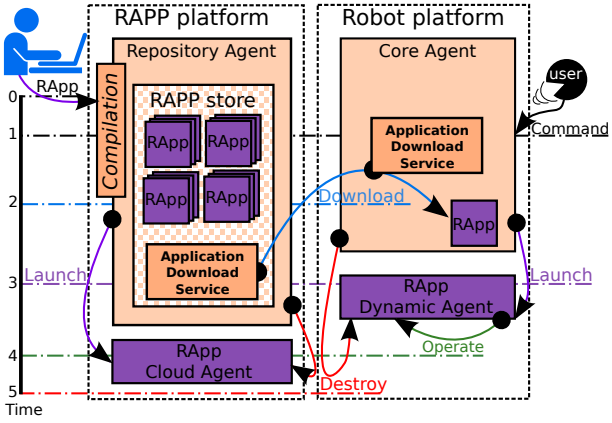


Fig. 1. Life cycle of a RApp – Dynamic Agent

3. Navigation System

3.1. System Requirements and Services

Distributed systems have to be designed under well defined general structure, otherwise the complex structure will be fuzzy or even the system can behave in nondeterministic way. The most common approach in distributed modular systems design is to separate it into modules and distribute them among hierarchical structure elements. Software of the presented navigation system is divided into three levels (Fig. 2). At the top in the hierarchy (level 3) is the navigation system, which is composed of agents (level 2). Each type of an agent is designed according to the specification presented in section 2. We use system components as the lowest hierarchical element (level 1), whereas design of the latter heavily depends on the implementation. However, a component implementation have to obey its higher level structure element constrains and execute one or more elementary system services.

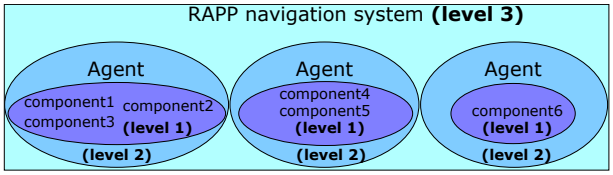


Fig. 2. Layers in the RAPP navigation system structure

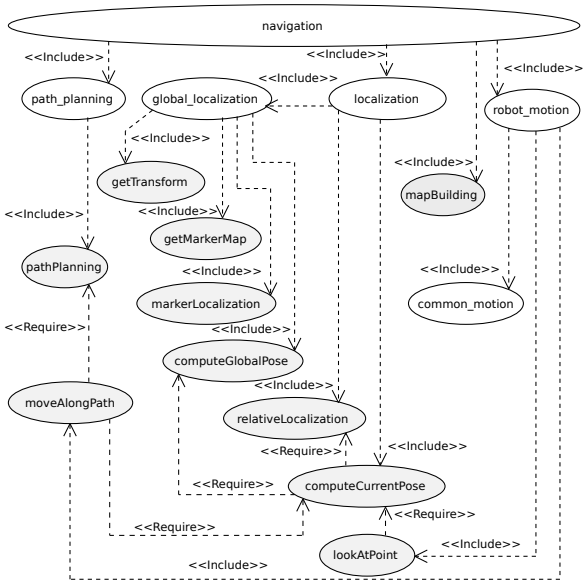
According to the RAPP system specification, our navigation system should respond to every possible navigation related request of every possible task in the RAPP store. As our work focuses on the distributed navigation system design, we propose a limited number of services, that handle core navigation tasks:

- `pathPlanning2d` – plan collision-free path through the environment,
- `getTransform` – return state of the given robot joint or pose of the given receptor in the requested coordinate frame,
- `getMarkerMap` – return localization markers map,
- `markerLocalization` – return transformation from the localization marker to the robot camera coordinate frame,
- `relativeLocalization` – using sensor data, update transformation from the current robot pose to the robot initialization pose,
- `computeGlobalPose` – return robot global pose, considering localization marker map, transformation from the robot camera to the robot coordinate frame and the `markerLocalization` service transformation,
- `computeCurrentPose` – return current robot pose in the global coordinate frame using poses from the `computeGlobalPose` service and the `relativeLocalization` service,
- `lookAtPoint` – orient the robot camera to the point given in the global coordinate frame,
- `moveAlongPath` – move robot base along the given path,
- `common_motion` – group of the core robot movements (such as: move robot using relative or absolute velocity values, robot joints control and set robot predefined posture).

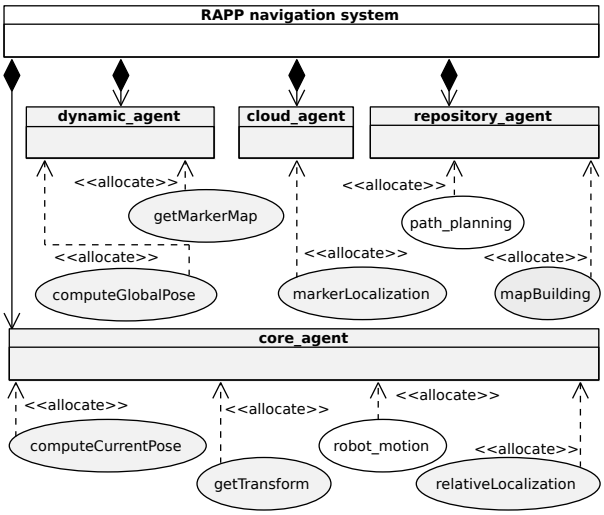
The navigation system abilities are divided hierarchically (Fig. 3(a)). We distinguish elementary services that are executed by the system components (gray ellipses) and a complex services that are composed of a specific set of elementary services (white ellipses). To take advantage of the distributed system structure, proper services allocation to the system agents is required. We distinguish:

- `Core services` – depend on the robot type, or require fast feedback from the robot platform,
- `Repository services` – do not belong to above group, are common in robot applications and require high computational resources,
- `Cloud services` – do not belong to above groups, are task specific and require high computational resources,
- `Dynamic services` – do not belong to above groups.

Each of the above groups should be realized in the corresponding agent, thus we distributed our navigation system services as shown in the Fig. 3(b). Localization of a mobile robot is an algorithm specific service. We focused our system specification on landmark (QR-code) based global localization and Extended Kalman Filter relative localization. The complex service is composed of some elementary services that are robot type dependent (`relativeLocalization`, `getTransform`, `computeCurrentPose`) and some that are global localization algorithm specific. One of the latter requires high computational power (`markerLocalization`) and two of them don't (`computeGlobalPose`, `getMarkerMap`).



(a) Hierarchical structure of the services



(b) Distribution of the services in the general structure of RAPP system

Fig. 3. Core navigation system services

3.2. System Components

Components are the executive parts of the navigation system, thus they have to satisfy restrictions given by the system specification and remain feasible in the sense of software implementation. We assume that one component can execute part of one or more elementary services within one agent subsystem only. The latter implies that a single component comprise a single subsystem of an agent. Real receptors and real effectors components are robot hardware drivers. Virtual receptors and effectors components gather data using real receptors and effectors respectively. Control subsystem components implement services of the agent and enable other agents components to request them.

Taking into consideration above rules, we determined several system components and distributed them among agents as shown in Fig. 4. As the a_{core} specification and implementation depends heavily on the considered robot platform, we present the a_{core} of the NAO robot as an example.

The `pathPlanning` complex service is executed using four components:

- *rapp_map_server* – loads the environment map and provide its data to the other a_{rep} components,
- *rapp_costmap2d* – computes two dimension cost map of the given robot type motion in the given two dimension occupancy grid map. Component allows to configure the map parameters,
- *global_planner* – calculates collision-free path using the data downloaded from *rapp_costmap2d* and the requested start and goal robot poses. As the component is based on the known ROS package – global_planner [8], it is able to use two different algorithms: A* and Dijkstra's. It computes the path along the given grid edges, or with use of the Simple Potential Calculation algorithm,
- *rapp_path_planning* – receives the *pathPlanning* service requests and responds to them using services of the *rapp_costmap2d*, *rapp_map_server*, *global_planner* components.

The `global_localization` service, as a distributed service, has to be composed of several components located in adequate agents:

- *qrCodeDetection* – implements `markerLocalization` service in the a_{cloud} agent,
- *camera_server* – implements `captureImage` service in the a_{core} agent; delivers images from the robot camera,
- *estimator_server* – implements `getTransform` service in the a_{core} agent,
- *localization* – implements `computeGlobalPose` and `getMarkerMap` services as a dynamic-link library included to a_{dyn} agent (RApp).

The `robot_motion` complex service is executed by the following components located in the a_{core} agent:

- *move_server* – implements interface between API calls and a_{core} agent subsystems; it is located in the a_{core} control subsystem,
- *execution_server* – implements motion requests interface from the a_{core} control subsystem to the robot real effectors; it is located in the a_{core} virtual effector subsystem,
- *obstacle_server* – gathers data from a_{core} real receptors and delivers information about detected obstacles to the a_{core} control subsystem.

The *computeCurrentPose* service is integrated with the *getTransform* service in the *estimator_server* component. The last service – the `relativeLocalization` – is executed by cooperation of the following components allocated in the a_{core} agent subsystems:

- *state_server* – gathers data from the robot interoreceptors (real_receptors), computes the robot current state and delivers it to the control_subsystem,
- *robot_localization* – using data from the *state_server* estimates current pose of the robot (relative to the robot initial pose) and delivers it to the control_subsystem.

The components layer of the complex navigation system architecture is presented in the Fig. 4.

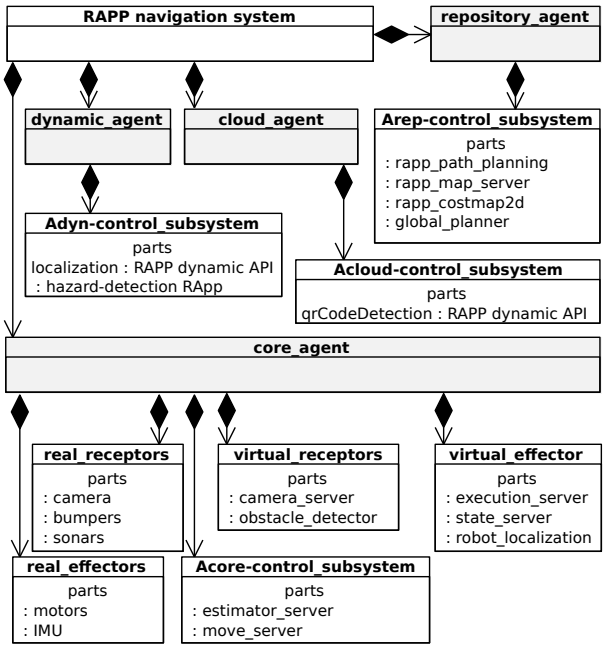


Fig. 4. Distribution of the system components in the RAPP general structure [2]

3.3. Cross Component Communication

Both the embodied agent theory and RAPP system structure [17], restrict connections between components. Communication among agent subsystems is based on the specified buffers, thus one subsystem can communicate only with specific subsystems within the same agent. Furthermore, the RAPP system specification restricts communication between agent, in order to keep the system stable after particular agent launch or destroy action. The rules above were taken into account in the components distribution procedure to avoid forbidden connections.We present the general communication structure of distributed system components, that are required to perform robotic tasks. Distributed systems, especially that involve robots, require secured communication between their components. As one of the main topics of our ongoing research, we consider to implement well established network communications security techniques [7, 14], to solve the problem of the secure communication between distributed system components. In this Section we present used cross component communication in the example services operation. Operation of a complex service allocated in the cloud requires specific communication between components that execute the elementary operations of that service. There are two complex navigation services that use the cloud processing – `path_planning` and `global_localization`.

The first one – the `path_planning` service – operates in the a_{rep} only. Therefore, the service does not

need to follow the inter-agent communication rules. However, the a_{rep}, as an agent that serves for many different robots, sets additional communication regulations:

- internal communication of the a_{rep} is realized using RPC requests/reply interactions,
- there must be the possibility to request the service multiple times at once.

The activity diagram that describes the internal communication between components during the `path_planning` service execution is presented in the Fig. 5. As the Fig. 4 and Fig. 5 show, the a_{rep} is composed of the *rapp_map_server*, the *rapp_costmap2d*, the *global_planner*, and the *rapp_path_planning* components. During the a_{rep} initialization process, the given number of each component instances are launched. The a_{dyn} starts the service by sending the request to the a_{rep}. The request includes several parameters:

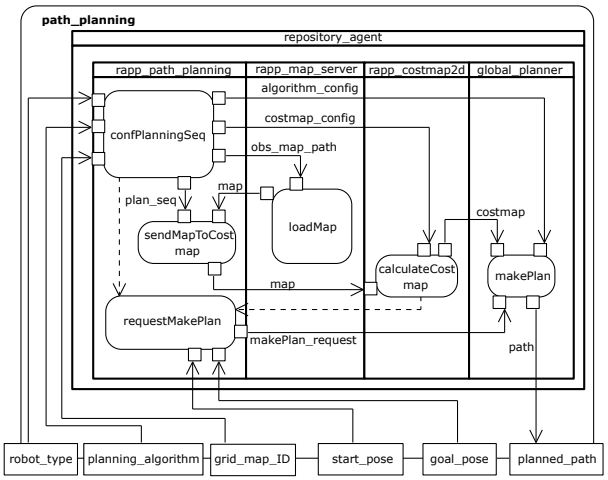


Fig. 5. Internal communication of the a_{rep} during the path_planning service execution

- robot_type – ID of the robot type, defines robot size and kinematics,
- planning_algorithm – ID of one of the available planning algorithms,
- grid_map_ID – ID of the occupancy map of the environment pre-stored in the cloud,
- start_pose – the initial pose of the required path,
- goal_pose – the end pose of the required path.

Next, the *rapp_path_planning* component interprets the request and configures the appropriate (currently unused) instances of the other components. Afterwards, it loads the occupancy grid map using the elementary service (`load specified map`) of the `rapp_map_server` component. The map is sent to the `calculate costmap` elementary service of the proper instance of the *rapp_costmap2d* component. Next, the `makePlan` elementary service of the proper instance of the *global_planner* component gets the costmap from the *rapp_costmap2d* component instance. After this sequence, the *rapp_path_planning* component requests the `makePlan` elementary service to plan the path. Finally, the `makePlan` service calculates the path and sends it to the *rapp_path_planning* component to translate it to the RAPP object.

The second complex navigation service that takes advantage of the cloud processing is the `global_localization` service. Apart from the other services this one is realized by three cooperating agents – a_{cloud}, a_{dyn} and a_{core}. The component that is responsible for the service execution is the RApp (a_{dyn}). It uses other agents elementary services and integrates obtained responses. The integration process of the elementary services is presented in the Fig. 6. As the `global_localization` service requires interactions with the robot type dependent a_{core} agent, the example was presented using the NAO robot instance of the a_{core}.

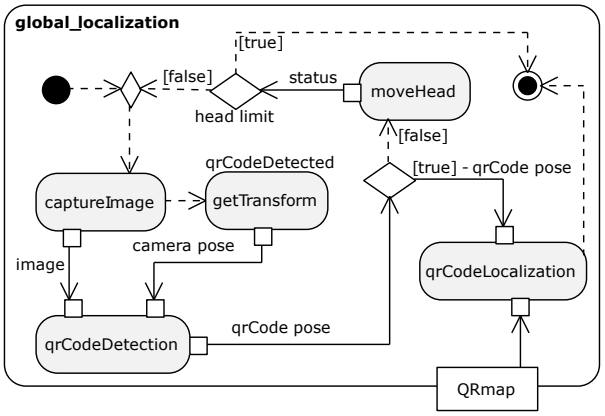


Fig. 6. Activity diagram of the global_localization service – NAO robot case

The a_{dyn} agent starts the service execution by requesting the a_{core} for the picture (calls the `captureImage` elementary service), and for the current transformation from the camera to the robot coordinate frame (calls `getTransform` elementary service). Next, the a_{dyn} sends the obtained data to the a_{cloud}, to the `qrCodeDetection` service. This service is responsible for finding a QR-code in the picture and if one was found, it calculates the pose of the QR-code in the robot coordinate frame and sends it as a response to the a_{dyn}. Next, the transformation is being converted by the a_{dyn} elementary service (the `qrCodeLocalization`). This service gets the pre-composed *QRmap*, that defines the pose of each QR-code in the global coordinate frame. Finally, the pose of the robot in the global frame is computed using the map and the transformation from the QR-code to the robot coordinate frame. However, if the `qrCodeDetection` service returns lack of QR-code in the obtained picture, the a_{dyn} requests the a_{core} to change the robot head orientation (requests the `moveHead` service). Then, the next picture is captured and checked. The described loop ends in two situations, either the QR-code is found in the picture or the robot head reached its extreme orientation (*head limit* = true).

4. Navigation System Implementation and Verification

The system implemented observing the rules and diagrams presented in the preceding sections. As the a_{core} of the system depends on the robot type, the a_{core} of the NAO robot was implemented. Virtual receptors of the system interact with the robot hardware using the robot producer NAOqi framework. We used some well-known libraries and modules of the *Robot Operating System* (ROS) [11]:

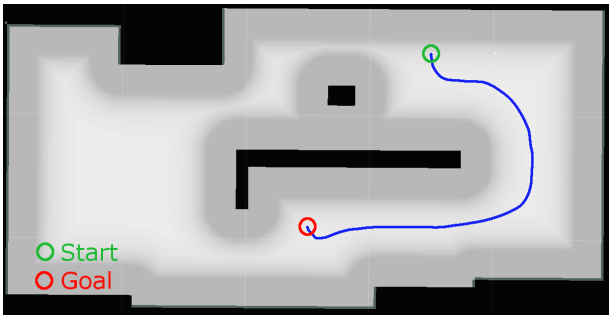
- `global_planner` – as the `global_planner` component of our system,
- `robot_localization` – as the `robot_localization` component of our system,
- `costmap_2d` – equipped with the RAPP specific plug-in as the `rapp_costmap_2d` component of our system,
- `map_server` – enhanced with on-the-fly change of the published map function as the `rapp_map_server` component of our system.

As the communication between the robot and the cloud platform is based on request/response pattern, we implemented the communication using HTTP protocol. In both platforms, ROS service requests/responses are transformed to the HTTP requests/responses.

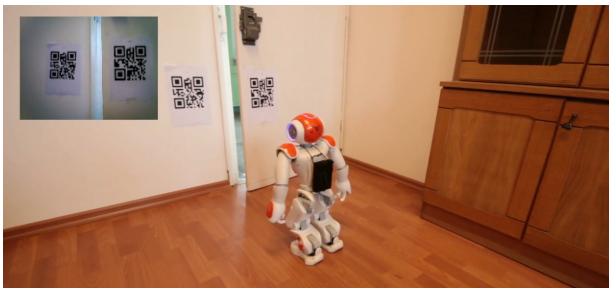
Verification of the system was conducted by testing each service separately during example RApp execution. The hazard detection task [2,3] was chosen as the verification task and was implemented as the RApp. The hazard detection task is the sequence of some services requests:

1) *standUp* – sets the predefined posture of the robot (one of the elementary services of the `common_motion` complex service). The robot prepares to the localization and the motion execution.
2) *global_localization* – execution of the `global_localization` service. The system defines the robot current pose.
3) *hazard detection loop* – executed for every hazard in the environment:
   a) *path_planning* – execution of the `path_planning` service. The a_{dyn} requests the cloud for the collision-free path and forwards it to the a_{core} as the request for motion (Fig. 7(a)).
   b) *moveAlongPath* – execution of the `moveAlongPath` service (`robot_motion` complex service). The robot moves to the next hazard detection pose.
   c) *lookAtPoint* – execution of the `lookAtPoint` service (`robot_motion` complex service). The robot points its camera to the next hazard detection point.
   d) *checkHazard* – execution of the `checkHazard` service (it's one of the a_{rep} services). The cloud returns status of the next object to the RApp (Fig. 7(b)).
   e) *global_localization* – execution of the `global_localization` service.

The RApp is implemented using NAO robot specific a_{core}. In the environment there were two hazards – the door status (opened/closed) and the lamp status (on/off). As the RApp governs the robot platform and is aware of the current state of the system, it is responsible for handling any unexpected situation e.g. communication error, module crash or hardware failure. In our verification task we use the robot voice to communicate any encountered unexpected situation. If the voice is disabled, we send instructions to the user in an email and light up red lights around the robot eyes.



(a) Path planned using the *path_planning* service and sent to the a_{core} for execution. Black areas represent restricted areas, gray areas inflate black areas to increase distance between robot and obstacles, white areas represent area that is available for robot motion



(b) NAO robot during the hazard detection task (video: `https://vimeo.com/141078577`) – checking the door status

Fig. 7. Verification task execution

The conducted experiments showed that the robot localized itself with proper accuracy and the inter-platform (robot ↔ cloud) communication was correct. The paths planned by the a_{rep} were returned to the robot platform and their execution in the environment was collision-free. The latter is the result of:

- sufficiently accurate global localization,
- proper relative localization during each path execution,
- correct motion execution between points of each path.

The largest distance between the actual robot path and the desired path was 19 cm and is considered as relatively small in the NAO robot case.

Implemented code is stored in the repositories (`https://github.com/rapp-project/`). The code is compatible with ROS indigo version and contains the core_agent of the NAO robot

5. Summary

In this paper the distributed robot navigation system was described. It is based on the embodied agent theory and enables distribution of the robot control software between two separate processing platforms – any mobile robot and a cloud platform. Furthermore, a proposed software structure allows robots to exchange knowledge and abilities between themselves. Presented navigation system cloud services support multiple requests at once, thus one cloud platform is able to satisfy many robots needs for additional memory and processing power. The system was tested using the NAO robot to verify that the cloud processing enable robots with low computation capabilities to realize quite complex service task. Thanks to four agent cooperation, the system is able, among others, to localize robot properly, build an occupancy map, plan a collision-free path, execute robot motion along the desired path. In this paper we mainly focused on detail description of two in-cloud services – robot path planning and global localization. The first of these two operates in the cloud only and the second is realized by cooperation of both platforms (the cloud and robot platform). The navigation system allows to configure in-cloud services in such a way that, diverse robot types are supported. The proposed system was implemented and verified with use of the NAO robot. The conducted experiments showed that every service of the system acts properly and a quite complex task can be fulfilled by a robot platform equipped with low on-board computational power. In addition the system allows to speed up and simplify robotic application development. The presented system requires from a developer familiarity with the available services only. Robotic application programmers do not need to know, neither specific communication rules, nor complex robot control frameworks like ROS.

Acknowledgements

This work is funded by the FP7 Collaborative Project RAPP (Grant Agreement No. 610947), funded by the European Commission. The work is supported by the Polish Ministry for Science and Higher Education scientific research funds for the years 2014–2016 granted for the realization of co-financed international project.

AUTHORS

Wojciech Dudek* – Institute of Control and Computation Engineering, Warsaw University of Technology, 00–665 Warszawa, ul. Nowowiejska 15/19, e-mail: wojciech.dudek.mail@gmail.com, www: https://www.robotyka.ia.pw.edu.pl/.

Wojciech Szynkiewicz – Institute of Control and Computation Engineering, Warsaw University of Technology, 00–665 Warsaw, Nowowiejska 15/19, e-mail: W.Szynkiewicz@elka.pw.edu.pl .

Tomasz Winiarski – Institute of Control and Computation Engineering, Warsaw University of Technology, 00–665 Warsaw, Nowowiejska 15/19, e-mail: tmwiniarski@gmail.com.

*Corresponding author

REFERENCES

[1] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "DAvinCi: A cloud computing framework for service robots". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, 3084–3089.

[2] W. Dudek, K. Banachowicz, W. Szynkiewicz, and T. Winiarski, "Distributed NAO robot navigation system in the hazard detection application". In: *21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR'2016*, 2016, 942–947, 10.1109/MMAR.2016.7575264.

[3] W. Dudek, W. Szynkiewicz, and T. Winiarski, "Nao Robot Navigation System Structure Development in an Agent-Based Architecture of the RAPP Platform". In: R. Szewczyk, C. Zieliński, and M. Kaliczyńska, eds., *Recent Advances in Automation, Robotics and Measuring Techniques*, vol. 440, 2016, 623–633, 10.1007/978-3-319-29357-8_54.

[4] O. S. R. Foundatioin. "Robot Operating System". http://ros.org/. [Online; accessed 10-April-2016].

[5] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: The systems modeling language. 3rd ed.*, Elsevier, Morgan Kaufmann, 2015.

[6] 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, 2015, 398–409, 10.1109/TASE.2014.2376492.

[7] R. L. Krutz and R. D. Vines, *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*, Wiley Publishing, 2010.

[8] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment". In: *International Conference on Robotics and Automation*, 2010.

[9] P. Mitkas, "Assistive robots as future caregivers: The rapp approach". In: R. Szewczyk, C. Zieliński, and M. Kaliczyńska, eds., *Progress in Automation, Robotics and Measuring Techniques. Vol. 2 Robotics.*, vol. 351, 2015, 171–179.

[10] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform", *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, 2015, 481–493, 10.1109/TASE.2014.2329556.

[11] 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.

[12] J. Riazuelo, J. Civera, and J. M. M. Montiel, "C2TAM: A cloud framework for cooperative tracking and mapping", *Robotics and Autonomous Systems*, vol. 62, no. 4, 2014, 401–413.

[13] J. Salmerón-Garcıa, P. Íñigo Blasco, F. D. del Rıo, and D. Cagigas-Muñiz, "A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing", *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, 2015, 444–454, 10.1109/TASE.2015.2403593.

[14] J. Sen, "A survey on wireless sensor network security", *CoRR*, vol. abs/1011.1529, 2010.

[15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system". In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, 2010, 1–10.

[16] M. Tenorth and M. Beetz, "Knowrob—knowledge processing for autonomous personal robots". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, 4261–4266.

[17] C. Zieliński, W. Szynkiewicz, M. Figat, M. Szlenk, T. Kornuta, W. Kasprzak, M. Stefańczyk, T. Zielińska, and J. Figat, "Reconfigurable control architecture for exploratory robots". In: K. Kozłowski, ed., *10th International Workshop on Robot Motion and Control (RoMoCo)*, 2015, 130–135, 10.1109/RoMoCo.2015.7219724.

[18] C. Zieliński, T. Kornuta, and T. Winiarski, "A systematic method of designing control systems for service and field robots". In: *19-th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR*, 2014, 1–14, 10.1109/MMAR.2014.6957317.