Proceedings of the
1st International Conference and Exhibition on Future RFID Technologies
Eszterhazy Karoly University of Applied Sciences and
Bay Zoltán Nonprofit Ltd. for Applied Research
Eger, Hungary, November 5–7, 2014. pp. 93–100

doi: 10.17048/FutureRFID.1.2014.93

Simulation Framework of Autonomous Robots as ROS basis

Zoltán Krizsán, Szilveszter Kovács

University of Miskolc, Institute of Information Science
krizsan@iit.uni-miskolc.hu
szkovacs@iit.uni-mikolc.hu

Abstract

The Autonomous Robotics is a novel paradigm for handling Human-Robot interaction. In this area the simulation without hardware is very important. The goal of this paper is the introduction of possible software framework for general robot simulation and real-time monitoring. From architectural point of view the Autonomous Robot software system can be divided into three main parts, the physical robot platform, the behavior logic and the software framework supporting and operating the behavior logic on the physical robot. Due to the flexible ROS architecture the components are interchangeable hence it is easy to create a new framework for an autonomous Robot. This paper suggests the Robot Operating System (ROS) based implementation of the software framework for Ethologically Inspired Autonomous Robots and presenting the details of components and the cooperation of them.

Keywords: ROS simulation

1. Introduction

Within the robotics area, the task of robot systems can change quickly. If the job and environmental circumstances change frequently, reusable and reconfigurable components are needed, in addition to a framework which can handle these changes. The effort needed to develop such components depends on the programing language, the development environment and other frameworks used. Many programming languages can be applied for this process, but developing a brand new framework is a difficult task. Applying an existing framework as a base system, such as the ROS in the case of this paper, the associated development tasks can be dramatically simplified.

Frameworks and middleware are gaining popularity through their rich set of features which support the development of complex systems. Joining together any robot framework and robot drivers can form a complex and efficient system with relatively small effort. In many cases, the task of the system designer is reduced to the configuration of an already existing framework.

On the other hand, when an existing framework requires only a set of completely new features, it can simply be extended with them. Improving an existing framework is an easier job than developing a new one because the design and implementation of such a system requires specialized knowledge and skills (design and implementation patterns). In most cases, the missing functionalities can be simply embedded into the existing framework, but there are some cases when this is hard to achieve. An existing framework can be improved simply only in case it is well designed and implemented. In spite of this, building a brand new system from is a much longer process that requires much effort. In the area of robotics, there are many robot parts that share similar features, so the concept of robot middleware as a common framework for complex robot systems is obvious. Probably there is no framework which can fulfill all of the above requirements entirely.

The primary goal of the researcher is to find a robot framework in this environment that is easy to use, reliable and also easy to extend. If it is impossible to find a perfect one, the secondary goal could be the improvement of an existing one. In the final case, if no acceptable framework exists, a brand new one would have to be implemented from the ground up.

There are some main requirements of robot application middleware. First of all it has to support as loosely as possible coupling of components. The weak build and running dependency are important because of easy component interchangeability, parallel development and testing. If the user's robot system is decoupled then the members of the team can develop and test the separated part of the project easily. Second mandatory feature of robot middleware is the hardware and sensor simulation existence. Third feature should be the existence of command tools helping the system control and usage. In this paper we explore these requirements and suggest solutions.

The paper is organized as follows. The main requirements of robot framework is explored and an overview of robot middleware are given is section 2. In the section 3 our recommendations are detailed. Application of suggestion is presented in the section 4. Finally the conclusion emphasizes the advantage of solution.

2. Related work

In this section, some robot frameworks are compared to give a starting impression YARP, OpenRDK, OpenRTM-aist and ROS systems. Frameworks and robot middleware are gaining popularity through their rich set of features, helping the development of complex robot systems.

Definition 2.1. Robot middleware is a software middleware that extends communication middleware such as CORBA, ICE or XML RPC. It provides tools,

libraries, APIs and guidelines to support the creation and operation of both robot components and robot systems. Robot middleware also acts as a glue that establishes a connection among robot parts in transparent way.

One distributed environment for robot cooperation is OpenRDK. The user's robot system can be developed using a set of Agents, through a simple process. A Module is a single thread inside an agent process. Every module has a repository in which a set of internal properties are published. Inter-agent (i.e., inter-process) communication is accomplished by two methods: through property sharing and message sending. RConsole is a graphical tool for remote inspection and management of modules. It can be used as both the main control interface of the robot and for debugging while developing software. It is just an agent that happens to have some module that displays a GUI. The framework can be downloaded from [2].

Another important robot middleware platform is Yet Another Robot Platform (YARP). Communication in YARP generally follows the Observer design pattern (for more details see [1]). Every YARP connection has a specific type of carrier associated with it (e.g., TCP, UDP, MCAST (multi-cast), shared memory, within-process). Ports can be protected by SHA256 based authentication. Each port is assigned a unique name and it is registered into a name server. The YARP name server, which is a special YARP port, maintains a set of records, the keys of which are text strings (the names of the ports). The remainder of each record contains whatever information is needed to make an initial connection to a given port.

A third important robot middleware technology is OpenRTM-aist, which is a convenient modular system also built on the Common Object Request Broker Architecture (CORBA). The online component search is supported by the CORBA naming service, which is a simple process that acts as a servant object that can be accessed remotely by components as well as the graphical system editor. In OpenRTM-aist, originally introduced by Noriaki ANDO in [3] (more details about it can be found in [4]), the software is modularized into components of RT functional elements (called RT-components). Each RT-Component has an interface (a "port") for communication with other components. The RT system is constructed by connecting the ports of multiple components to each other in to aggregate RT-Component functions. The advantage of OpenRTM-aist is that it provides an simple way to create and co-operate various robot parts. In OpenRTM-aist, many programming languages can be used for component development, including C++, Java and Python. Some tools exist to support the development process by automatically generating the skeleton of components. The developer, then, only has to fill out the generated skeleton source, by concentrating only on the business logic to be implemented.

The forth robot middleware which has similar approach as OpenRTM-aist is the ROS (Robot Operating System) detailed in [8]. This system supports the components (as previous ones) but it is called nodes. The communication among nodes can be establish by topics (asynchronous strongly typed message queue) or service call (synchronous). These two mechanisms are the implementation of lightweight

dependency. Instead of creating one application containing more objects, several applications run parallel and sending the operation request as message. One component is very small part, easy to understand and test so the development of component can be delegated to individual member of team. During the implementation and integration test the messages can be investigated or sent by rostopic command line utility.

Additionally the system has a graphical monitoring and debugging system which called rviz and a complete 3D environment for simulation (Gazebo). These two tools helping visualization the realtime data.

We have chosen the ROS middleware because of the existence of graphical simulation environment, command line tools, and huge number of available components.

3. Suggested topology and Simulation Framework

In this section the design concept of our robot independent Behaviour Engine and the simulation environment is detailed.

The most important intention of design process is the minimization of dependency among system parts because of the code reusability and testing. If the application is built from classes depending each other then the result is a big monolithic executable file with no chance to replace any part of it. Application of interface leads to a flexible system but the pieces can be changed in the compile time only. The next step to increase interchangeability is the dependency injection what uses any descriptor (mostly xml) for relationship definition. Unfortunately, in case of C++ language this dependency injection is not available due to the language limits. The next possible solution for lazy system can be the utilization of design patterns (bridge, strategy, adapter) what solve more structural end behaviour problem in gently way. All of previously mentioned solution requires more programming knowledge and experience and leads to larger source code base inside one application.

Our suggested solution for the minimal dependency problem is the utilization of ROS robot middleware and Nodelet ROS package. If the cooperative algorithms are outsourced to separated nodes only then the communication among them is expensive because the ROS serializing the message content into XML in the publisher node and deserializing it in the subscriber node. Without nodelet package the system works correctly but the information flow becomes slow as a result of message packaging. That is the problem of strong dependency is solved but a new one is appeared the performance issue.

Writing our node as nodelet requires minimal code modification. Without nodelet the initialization of application can be inside the main function or in the constructor of class. Instead of these standards the user's classes have to be the subclass of Nodelet and should override the onInit method.

The second requirement of efficient system in ROS is the communication optimization using shared pointer and nodelet. It is very important in case of large amount of data and frequent calculation such as image processing, people detec-

tion and recognition. In normal case the messages are sent as webXML content. A new possibility is opened with the introduction of nodelet because the nodelet manager (shipped by the system) loads the participant algorithms into one address space. Moreover the pieces of application can be implemented as plugin establishing the runtime loading according to user settings. One additional step is required to exploit this improvement. The advertiser nodelet has to publish the message as shared pointer i.e. the data is wrapped into a specified class.

Our recommendation is the careful design of layered application into well separated nodelets and topic based asynchronous communication via shared pointer messages. The engine of application is implemented in a layer that is a collection of nodes. In the case of real system, what is running in the robot, the lower layer providing the real sensor information and the robot behaves according to commands. On the other hand during the simulation or the integration test the lower layer publishes the sensor values and consumes the commands in virtual space or mocked entities. In point of view of business logic there is no differences among these situations and it does not require any modification in source code.

Fortunately the sensor simulation and 3D environment are also exist in ROS system. There is the Gazebo application introduced in [7] and ROS interface for Gazebo package to provide complex indoor and outdoor environment for robot simulation. It contains a lot of robot model yet and it can be extended custom ones in easy way. It supports multiple physics engine such as ODE and Bullet and based on OGRE providing realistic rendering (shadow, textures).

The simulation is provided to several type of sensor such as camera, depth image, RFID, GPS, sonar, laser ray and can be extended by new plugins. Some of them can be visualized as colour circle or translucent sphere what is very useful to debug and check for example in case of intelligent space detailed (f.e. in [6]). Gazebo provides models of many common sensors. In the real world, sensors exhibit noise, in that they do not observe the world perfectly. For this reason the Gazebo can add noise to Ray, Camera and IMU sensor.

In order to use custom model (robot or building) we have to define the parts of tho robot as links and connections of links as joint. Two main feature of link have to be defined the visual part and the collision. Additionally sensors can be adhere to the link. This model can be defined in URDF (ROS XML standard) or SDF (specification of Gazebo). The Gazebo uses the SDF format but the converter tool exists converting the URDF to SDF.

Because of performance issue we have to minimize the number of links so the visual and collision property of link should be a complex mesh made by any outer application. Fortunately the Gazebo supports more mostly used model description formats f.e. the Collada dae file what can be exported from the Google Sketchup and Blender.

4. Robot Independent Behaviour Example as Application Example

In the following part of paper deals with the structure of Behaviour Engine controlled robot and the 3D simulation environment construction. A porter robot is implemented in the basis of Robot independent behaviour engine. This porter exploring the corridor mainly. If the battery level is decreased to predefined lower limit then it goes to the recharging station. If any person appears in the corridor (stranger or staff) then the robot goes close to human and explore it. These behaviours (goto_charger, goto_human, explore) use the official ROS Navigation component collection i.e. our behaviour system send a new navigation goal if the appropriate circumstances are meet.

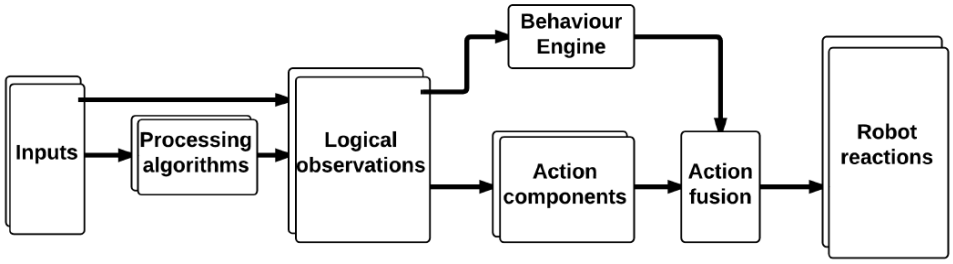


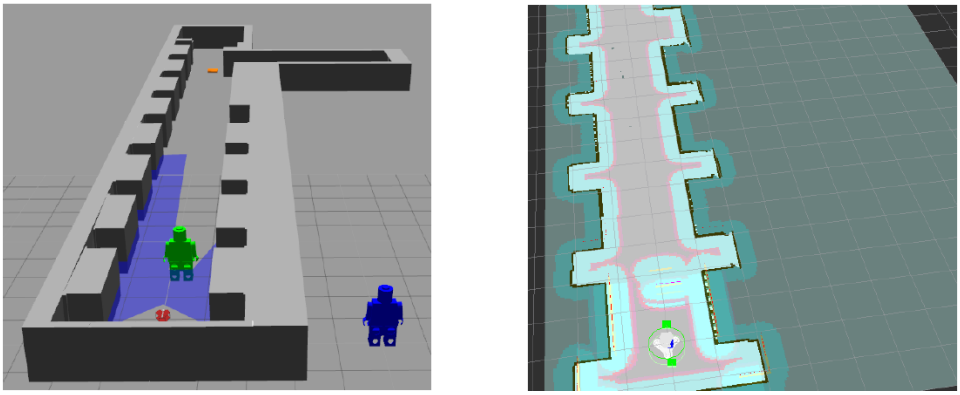
Figure 1: The tiers of behaviour engine system

In the Figure 1 we can see the tiers of the robot independent behaviour engine. The information is flow from left to right side i.e. the information is occurred in any component in the inputs tier and the control command is sent to the robot from the robot reactions tier. The information is converted in every internal tier which activating the next component. The central logic calculates the robot reaction base on sparse fuzzy rule base and the observations providing the Logical observations tier. This central logic is implemented inside the Behaviour engine node. Every node before this node in the flow provides the information to the decision, and later nodes perform the appropriate reaction according to outcome. Every node connects each other via message queue so in the development period we have to send dummy messages instead of source node and examine the result message. Moreover during the simulation the information to Inputs are generated by Gazebo and the control commands are get a Gazebo plugin also.

The plan of corridor of ELTE Etology is shipped as 2D picture. This 2D pixel based graphics is used as the base of 2D map in ROS navigation and expanded to 3D object in one easy step what is exported to dae file format (Google Sketchup).

The robot model is constructed in URDF file format because the ROS Navigation package requires the existence of it. The model of other simulation participants (the stranger and staff human) what is required by the features of behaviour engine are found as free resource on the internet. The Figure 2a shows the Gazebo

GUI in which the position and orientation of robot, stranger, staff can be changed, moreover the laser range is displayed as well.



(a) Simulated environment in Gazebo (b) The visual interface of Rviz

Figure 2: Graphical interfaces of ROS-Gazebo system.

The system can be observed and circumstances can be changed via dedicated graphical client named gzclient from any Ubuntu based workstation (Figure 2a) or by the web based client. Futhermore a new web based interface is developed in which the parameters and additional inputs (f.e. bell, battery level) of our system are provided. Our solution wraps the official web extension of Gazebo named gzweb.

In the Figure 2b the environment is presented in point of view of robot. We can see the footprint of the robot, the piece of map and the lethal zone close to walls and obstacle which is restricted area to the centre of the robot. In the presented case in the Figure 2a the stranger is human depicted by Lego man however it is an lethal area to the robot in the Figure 2b. In fact this obstacle generated cloud is a goal of navigation in case of goto_human behaviour or unusable part of the map in the others.

5. Conclusion

The system that is based on ROS robot framework can work efficiently and can be developed quickly. It is easy to use same logic in case of simulation and production environment due to ROS flexible concept. The weak dependency amount system components can be established via ROS strongly typed topics. The simulation of real robot can be reached using the graphical native and web based client too. Additionally, the sensors of the robot are simulated such as camera, depth image, laser in the 3D environment of Gazebo system. The application of ROS and gazebo is useful for every research team who want to implement the robot system under Ubuntu operation system.

Acknowledgements. This research was supported by the Hungarian National Scientific Research Fund grant no: OTKA K100951.

References

[1] YARP robot middleware website, *http://eris.liralab.it/yarpdoc/index.html*

[2] OpenRDK robot middleware website, *http://openrdk.sourceforge.net/"*

[3] Ando, Noriaki and Suehiro, Takashi and Kitagaki, Kosei and Kotoku, Tetsuo and Yoon, Woo-Keun RT-middleware: distributed component middleware for RT (robot technology), *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, (2005), IEEE, 3933-3938.

[4] Ando, Noriaki and Kurihara, Shinji and Biggs, Geoffrey and Sakamoto, Takeshi and Nakamoto, Hiroyuki and Kotoku, Tetsuo, Software deployment infrastructure for component based rt-systems, *Journal of Robotics and Mechatronics*, (2011), 23: (3), 350-359.

[5] Nathan Koenig and Andrew Howard Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, (2004), 2149-2154.

[6] Szemes PT, Hashimoto H, Korondi P Pedestrian-behavior-based mobile agent control in intelligent space, *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT* 54:(6), (2005) 2250-2257.

[7] Koenig, Nathan, and Andrew Howard Design and use paradigms for gazebo, an open-source multi-robot simulator, *Intelligent Robots and Systems, Proceedings. 2004 IEEE/RSJ International Conference* (2004) 2149-2154

[8] Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng ROS: an open-source Robot Operating System, *ICRA workshop on open source software* Vol. 3. No. 3.2. (2009)