J Intell Robot Syst
DOI 10.1007/s10846-013-9991-7

Modular Design: A Plug and Play Approach to Sensory Modules, Actuation Platforms, and Task Descriptions for Robotics and Automation Applications

Ayssam Elkady · Jovin Joy · Tarek Sobh · Kimon Valavanis

Received: 22 April 2013 / Accepted: 7 October 2013

Abstract In this paper, the *RISCWare* framework is proposed as a robotic middleware for the modular design of sensory modules, actuation platforms, and task descriptions. This framework will be used to customize robotic platforms by simply defining the available sensing devices, actuation platforms and required tasks. In addition, this framework will significantly increase the capability of robotic industries in the analysis, design, and development of autonomous mobile platforms. *RISCWare* is comprised of three modules. The first module encapsulates the sensors, which gather information about the remote or local environment. The second module defines the platforms, manipulators, and actuation methods. The last module describes the tasks that the robotic platforms will perform such as teleoperation, navigation, obstacle avoidance, manipulation, 3-D reconstruction, and map building. The objective is to design a middleware framework to allow a user to plug in new sensors, tasks or actuation hardware, resulting in a fully functional operational system. Furthermore, the user is able to install and uninstall hardware/software components through system lifetime with ease and modularity. In addition, when hardware devices are plugged into the framework, they are automatically detected by the middleware layer, which loads the appropriate software and avails the device for applications usage. This automatic detection and configuration of devices make it efficient and seamless for end users to add and use new devices and software applications. Several experiments, performed on the *RISCbot II* mobile robot, are implemented to evaluate the *RISCWare* framework with respect to applicability and resource utilization.

Keywords *RISCWare* · Middleware · Plug and play · Teleoperation · Face detection · Face recognition · Modular design · Architecture · *RISCbot II*

A. Elkady (✉) · J. Joy · T. Sobh
School of Engineering, University of Bridgeport,
221 University Avenue, Bridgeport, CT, 06604, USA
e-mail: ayssam.elkady@gmail.com

J. Joy
e-mail: jovinj@gmail.com

T. Sobh
e-mail: sobh@bridgeport.edu

K. Valavanis
Electrical and Computer Engineering,
University of Denver, Denver, CO 80208, USA
e-mail: kimon.valavanis@du.edu

1 Introduction

Robotic middleware is designed to manage and hide the complexity and heterogeneity of components running on possible multiple platforms

Published online: 30 October 2013

in different locations, promote the integration of new technologies, simplify software design, and reduce the time and complexity of the development of robotic software and maintenance costs.

Most existing robot control software, which is installed in the robot to control it, is mainly designed for a specific hardware platform, sensory devices, applications and programming language. Based on these considerations, we propose a robotic middleware, called *RISCWare* (developed at the Robotics, Intelligent Sensing, and Control (RISC) lab). *RISCWare* is a generic framework for robot communication infrastructures that explicitly targets autonomous mobile manipulation platforms. *RISCWare* is developed to support different sensory devices, actuation platforms, and software applications. In addition, it is available for different programming languages. It is designed to be a modular, efficient, flexible, platform-independent middleware platform. Furthermore, the components (software and hardware) can be replaced before or during runtime.

The flexible property of *RISCWare* enables a third party to design self-configuring modules, and the Plug and Play (PnP) property of the *RISCWare* gives the robot's user the ability to plugin a component, which the *RISCWare* then recognizes. The plug-and-play allows auto-detection and auto-reconfiguration of the attached standardized components (hardware and software), according to current system configurations. This automatic detection and reconfiguration of devices and driver software makes it easier and more efficient for end users to install and use or uninstall hardware/software components. When the hardware devices are plugged into the framework, they are automatically detected by the middleware layer, which loads the appropriate software and takes advantage of the device for applications usage. One of the most desirable characteristics of the *RISCWare* is the resource-efficient transmission schemes that achieve the best utilization of available networking resources. The utilization of runtime communication should be efficient because the sensory devices generate many small messages on a regular basis.

RISCWare is implemented as a messaging system (queue-based and component-based) used to communicate between any component (sensory device, actuation platform and software application) at any desired time, as illustrated in Fig. 1. We chose to implement *RISCWare* as a messaging system because messaging provides a high degree of decoupling between components, so it plays a key role in the integration of heterogeneous systems. *RISCWare* facilitates and manages the runtime communication between framework components through the use of messages as the method of integration; *RISCWare* provides the ability to create, define the syntax, manipulate, store, and communicate these messages. The components can be abstracted and decoupled in such a way to be replaced with little or no knowledge by the other components. In addition, messaging offers the ability to process requests asynchronously to increase the performance of the system and reduce system bottlenecks. Most communication between the components is asynchronous and therefore does not require immediate response from the receivers. Once a message is sent, the sender can move on to other tasks; it does not have to wait for a response. Furthermore, parallel processing in *RISCWare* is achieved by introducing multiple message receivers that can process different messages simultaneously.

RISCWare is comprised of three modules. The first module encapsulates the sensors that gather information about the remote or local environment. The second module defines the platforms, manipulators and actuation methods. The third module describes the tasks that the robotic platforms will perform, such as: teleoperation, navigation, obstacle avoidance, and manipulation.

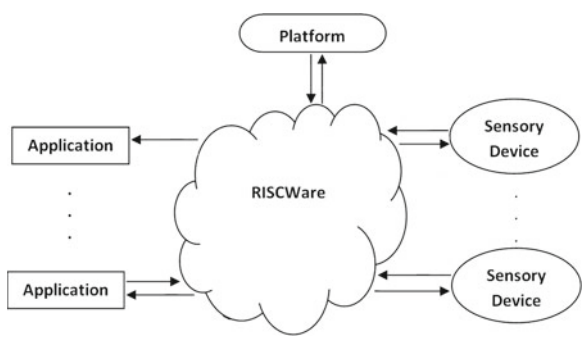


Fig. 1 Overview of the the *RISCWare* framework

Furthermore, several experiments, performed on the *RISCbot II* mobile manipulation platform, are described and implemented to evaluate the *RISCWare* framework with respect to applicability and resource utilization.

2 Related Work

There are several other robotics software platforms available such as Robot Operating System (ROS) [1], CLARAty [2], Player [3], Miro [4], Webots [5], RT-Middleware (RTM) [6] SmartSoft [7], Orca [8], OPRoS [9], and Microsoft Robotic Studio [10]. A survey of robot development environments (RDEs) by Kramer and Scheutz [11] described and evaluated nine open sources, freely available RDEs for mobile robots. In [12] and [13], an overview study of robot middleware is provided. Finally in [14], some freely available middleware frameworks for robotics were addressed, including their technologies within the field of multi-robot systems.

In [15], we outlined the architecture and some important attributes, with the comprehensive set of appropriate bibliographic references that are classified based on middleware attributes for most of the existing robotic middleware, such as Player, CLARAty, ORCA, MIRO, etc. Furthermore in [15], we presented a literature survey and attribute-based bibliography of the current state-of-the art in robotic middleware design.

3 *RISCWare* Features

The goal of this research is to design a framework (software and hardware) to be equipped with the mobile manipulators and enable them to act robustly and autonomously for various tasks and in different environments. This framework supports software modularity and abstraction, does not have any restrictions on the architecture of the control software, and also hides the low-level details of the device by providing hardware APIs. The details of *RISCWare* features are described in [16].

In *RISCWare*, the sensors are allowed to be reconfigured for use in different tasks like a stereo camera that can produce 3D range points and raw images. By using an appropriate multi-level device abstraction hierarchy, these sensor devices could be made integrable. The sensors and actuators of a robot are treated as shared resources to be used by several software applications.

The *RISCWare* components (software and hardware) can be downloaded, installed and configured at or before run-time.

As shown in Fig. 2, *RISCWare* provides some security mechanisms for authentication, authorization, and secure communication. Authentication is the mechanism to provide a way of identifying the user of the systems (such as by

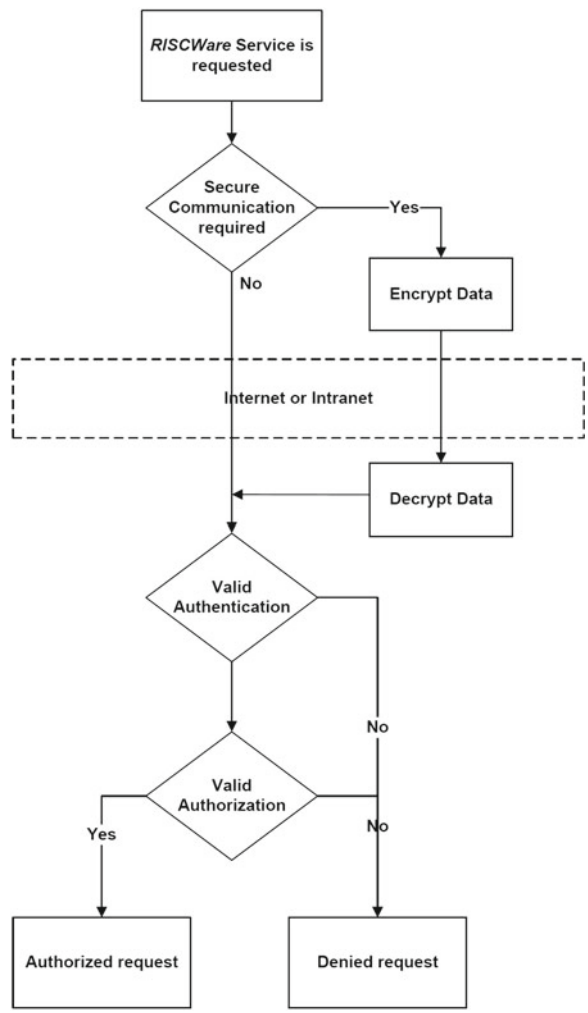


Fig. 2 Authentication, authorization and secure communication

having the user enter a valid user name and password). After the authentication process, authorization is required to determine what types or resources, services and access level a particular authenticated user is permitted. The secure communication is performed by encrypting the messages to prevent eavesdroppers from spying on the messages.

RISCWare supports parallelism by performing a number of processes simultaneously. Data and control flow are passed asynchronously from one component to others over the middleware. Asynchronous communication means that the sender is not required to wait for the message to be received or handled by the recipient. The sender is free to send the message and continue processing.

The framework can be re-scaled as its components grow in order to take full advantage of it. Furthermore, any part of the framework should be able to extend its capabilities without affecting the other parts. The components should be open for extension and closed for modification (i.e., the components should be easily extended without modifying the existing codes).

RISCWare provides reliable communication for higher priority messages in order to guarantee the delivery of a high priority message, even if a partial failure occurs. Guaranteed Delivery uses a store-and-forward mechanism, which means that incoming messages will be written out to a persistent store if the intended consumers are not currently available. Persistence increases reliability, but at the expense of performance. Furthermore, Guaranteed Delivery can consume a large amount of disk space scenarios. *RISCWare* allows configuring of a retry timeout (Time-To-Live) parameter that specifies how long messages are buffered inside the messaging system.

RISCWare provides some fault detection and recovery capabilities to be used in real, critical situations. A failure in one module should not damage the whole system; there is always the possibility of a fault at runtime. The faults in a robot framework should be detected and localized and also the robot should be able to complete its mission or at least to proceed to a safe mode. Even if the hardware and software of an autonomous mobile robot are carefully designed, implemented and tested, there is always the possibility of a fault at runtime. Some level of redundancy and robustness against such fault is crucial for a truly autonomous robot. In order to improve the robustness of the control system against faults at runtime, *RISCWare* is enriched with fault detection capabilities. Partial failure in the system is a fact of life. One of the components may need to be shut down at some time during its continuous operation or have an unpredictable failure. As shown in Fig. 3, the self-reconfigurable fault tolerance component has the ability to detect its faults or anomalies and then repair them autonomously.

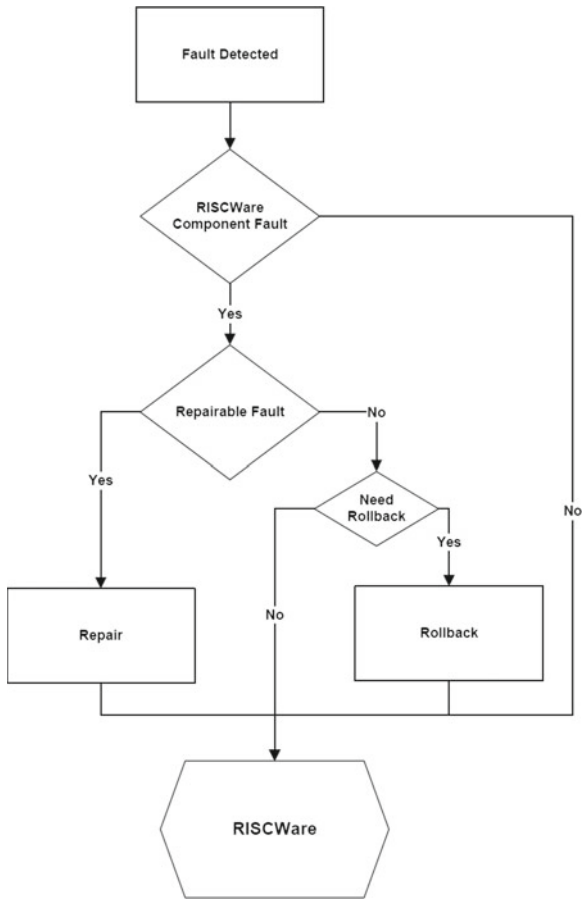


Fig. 3 Self-reconfigurable fault tolerance

4 *RISCWare* Middleware

As shown in Fig. 4, *RISCWare* is installed on top of the Linux operating system and LINX. LINX

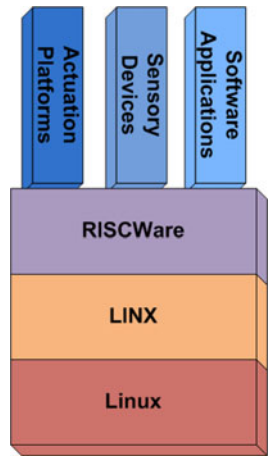


Fig. 4 *RISCWare* components

[17] is a distributed communication protocol stack for transparent interprocess communication. LINX provides the same services to the application regardless of hardware, operating system, physical interconnection, or network topology, and acts as a transport for bearer protocols such as UDP and TCP, as described in [17]. Furthermore, LINX supports failure reporting for both physical CPU interconnects and logical connections between endpoints.

The RISCWare is a middleware which consists of two main parts: the core logic, which contains the main functionalities, and the wrapper, which provides the interface of the logic core to the external components. SWIG, which is a free software development tool, is used to connect between the RISCWare core (written in C++) and the other software components (written in other languages such as Python and Perl). SWIG supports different types of target languages, including scripting languages such as Perl, PHP, Python, Tcl and Ruby, and non-scripting languages such as C#, Common Lisp (CLISP, Allegro CL, CFFI, UFFI), D, Go language, Java, Lua, Modula-3, OCAML, Octave, and R [18].

RISCWare consists of six layers: *OS & Hardware Layer*, *Hardware Abstraction Layer*, *Messaging Layer*, *Actions Abstraction Layer*, *Actions Layer* and *Applications Layer*. Each layer is described in detail in [19]. The *RISCWare* component consists of two main parts: the core logic, which defines the main functionality of the component, and the wrapper, which provides the interface of the logic core to the external components. The functionality, operability and architecture of the *RISCWare* modules are described in more details in [16].

A *RISCWare* user can configure the desired behavior by choosing a set of selective modules at (or before) runtime out of all available modules by simply performing a drag-drop from the available modules panel to the selected modules panel on the GUI, as shown in Fig. 5. In order to start the *RISCWare* (installed in *RISCbot II* as shown in Fig. 6), a small boot-loader program reads and interprets the system configuration profile from an XML file (including the number and type of the components defined in the *RISCWare*), proceeds to load the appropriate components onto memory in accordance with each component profile, and establishes their connections. The *RISCWare* scheduler decides which component to run first and then allocates a process to execute it. The scheduler is responsible for the starting, stopping, suspending, and resuming of the component. The user can select the configuration to load from a GUI form (shown in Fig. 5), or its name could be hard-coded into the boot loader. In this way, different permutations (choices of sensory devices, actuation platforms and software applications) can be loaded during (or before) runtime. The experiments were carried out as follows:

5 *RISCWare* Communications

The *RISCWare* component uses a client/server mechanism for control flow and the publisher/subscriber for data/event flow and defines it as Input/Output Ports. Two modes can be used: Pull or push the data (as shown in Fig. 7). As shown in Fig. 8, the Input-Push port subscribes to the Output-Push port by calling Subscribe(); also it can unsubscribe from the Output port by calling Unsubscribe(). The methods Start() and Push() are called by the server-component but Subscribe() and Unsubscribe() are called by the client-component. The Publisher pushes the messages to all the subscribers and the subscribers receive the messages without requesting them. GetData() is called by the client-component in order to get the last update of the data. On the other

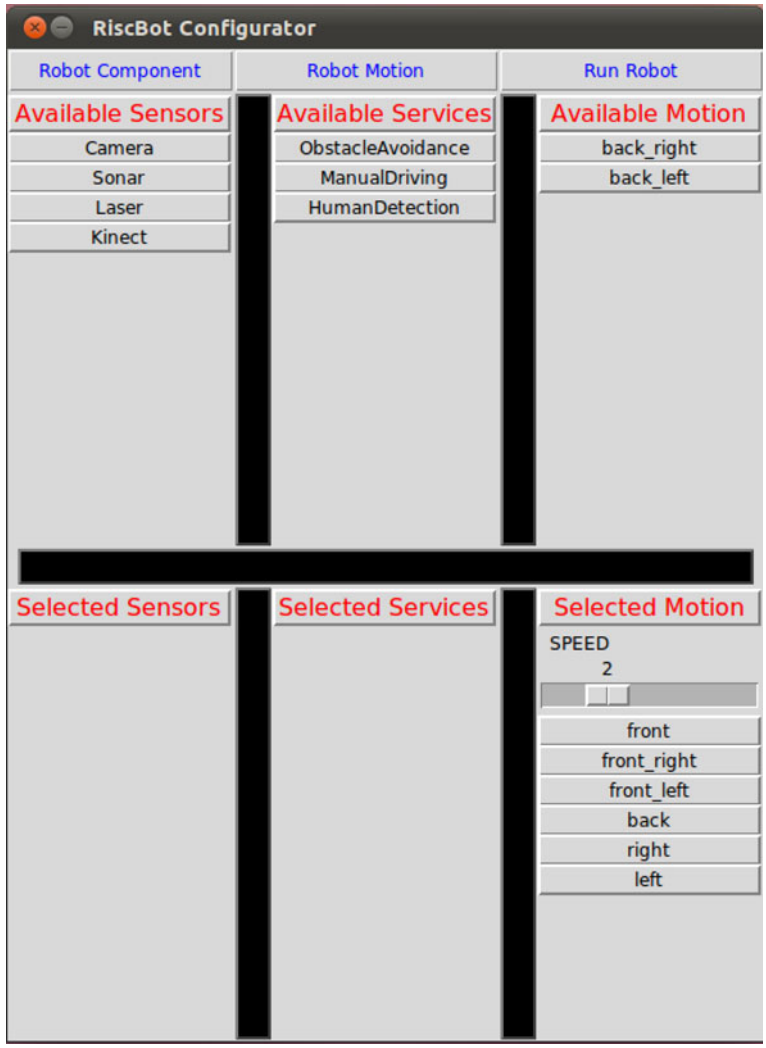


Fig. 5 *RISCWare*'s main menu

hand, the sequence diagram of the Pull Mode is shown in Fig. 9. The subscriber sends a request message to the publisher in order to get a data message. The UML diagrams of the *RISCWare*-component, Input and Output Ports (Push and Pull Modes) are shown in Figs. 10 and 11.

5.1 Configuration Management

Configuration files are a set of XML descriptors describing configurations and properties of the hardware and software in a domain. For example, the *Hardware Configuration File* (HCF) describes hardware configurations and hardware properties, the *Software Configuration File* (SCF) describes a software configuration and the connections among components. The *Hardware/Software Manifest File* (HSMF) describes a software component or a hardware device. The *Properties Descriptor* (PD) describes optional reconfigurable properties, initial values, and executable parameters that are referenced by other domain profiles. The *Configuration Manager Descriptor* (CMD) describes the Configuration Manager components and services used. The Robot Configuration File (RCF) is used to specify the different parameters related to a given robot, such as sensor number, types, position and orientation, platform kinematics model and its parameters such as length, height, width, weight, payload, translation speed, and rotation speed. *Hardware Driver Writing tool*

Fig. 6 *RISCbot II* mobile robot

provides facilities for auto-generating code by using a Driver Template provided by *RISCWare* and the XML Driver file, which is a configuration file provided by the user that contains the required information about the new hardware device.

6 Namespace Directory Service

Services are shared functions and tools that aid architects and developers to implement a solution, and are not a part of the core. A service should be a well-defined function and also should always be available to respond to requests. To be able to use the service, first the new service should add itself in the available services *Namespace Directory* of the middleware. The *Namespace Directory* provides a map between a logical name of sensor (used by the developer such as SONAR1) and the physical sensor. Then, each service should declare its interface to allow any application to communicate and use it. The *Namespace Directory* tracks all the loaded components and key information about the system. It is used to automate the action of locating any *RISCWare* component (in the same way for remote and local components). The publisher component registers its service in the *Namespace Directory* and then the subscriber dynamically locates the publisher via this Directory, as shown in Fig. 12.

Upon initialization, the sensors register their sensory information, defined in an XML descriptor, on the Namespace Directory. The XML descriptor contains meta-information about a hardware device such as the properties and parameters for the device, for example, for the laser range finder's profile, which contains Baud rate, Field of view (the angular scan range in degrees such as 180 or 270 degrees), and angular resolution. The sensory XML descriptor contains:

- ID: Sensors should be identified by a unique ID so that the Namespace Directory can use them. The ID numbers are assigned to all the sensors.
- Sensor Type: The sensor type can be sonar, infrared, laser range-finder, etc.

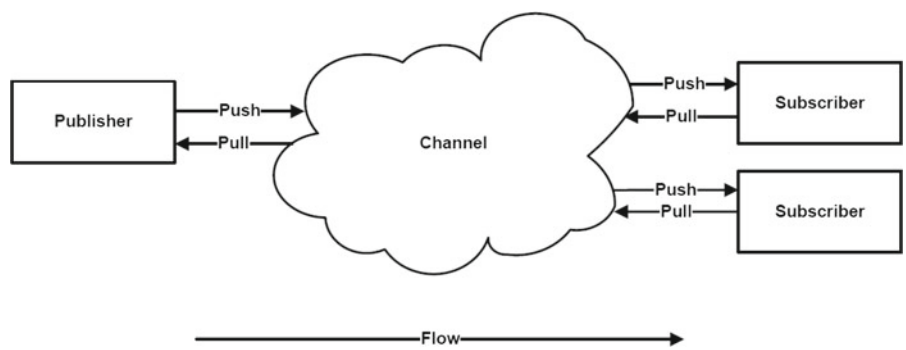


Fig. 7 Data flow of the Push/Pull Modes between the publisher and subscribers

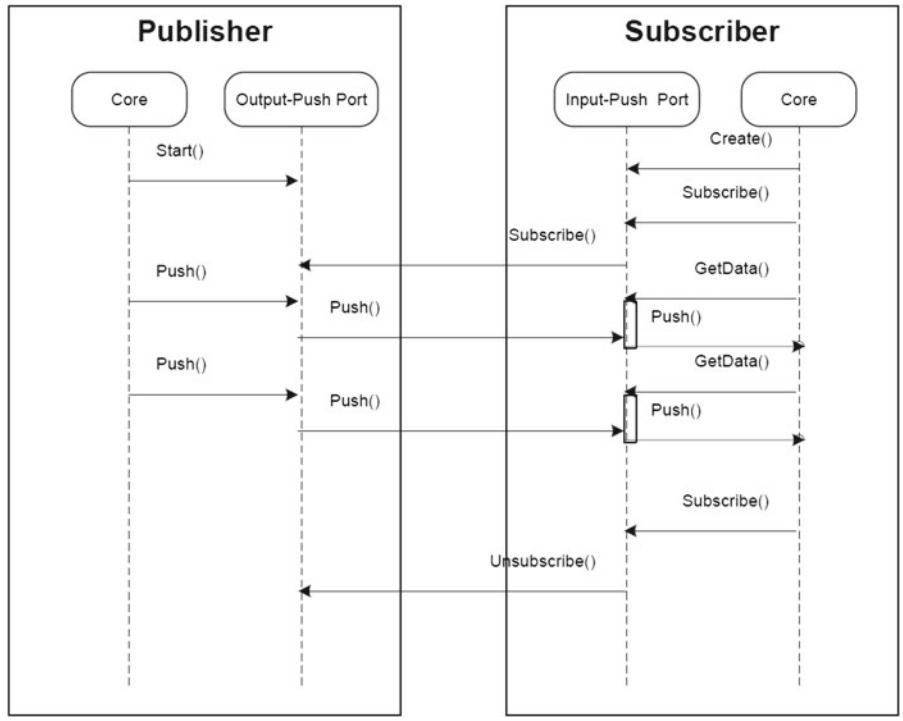


Fig. 8 Sequence diagram of the Push Mode between two components

- Position and Orientation: To describe the location and orientation of each sensor. Some sensors (such as sonars and infrared sensors) depend on their position and orientation but some sensors (such as temperature and humidity sensors) are not seriously dependent on the location and orientation.
- Name: Such as *Front_Sonar_1*, this is used by the *RISCWare* programmers to refer to the sensor. Then the Namespace Directory maps this name to the physical sensor.

7 Experiments

In this section, nine experiments are briefly summarized in order to illustrate and prove the adaptive, modular and robust capabilities of *RISCWare*. Samples of the nine experiments are uploaded on YouTube at (http://www.youtube.com/watch?v=8uYlMB1eMwc). *RISCWare* utilizes its currently available Plug-and-Play, auto-configurable and independent modules (sensory devices, mobile manipulation actuation platforms,

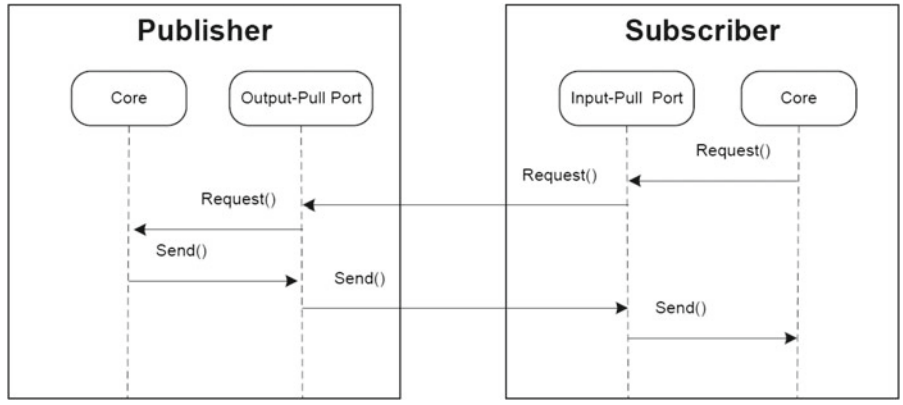


Fig. 9 Sequence diagram of the Pull Mode between two components

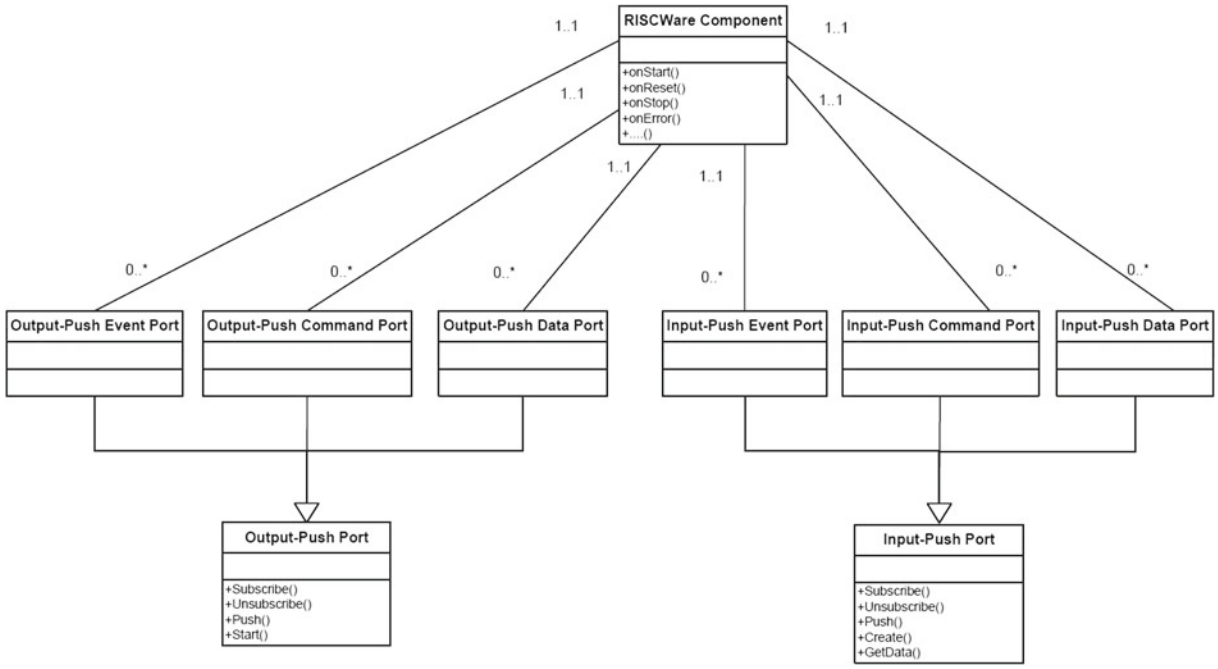


Fig. 10 UML of the *RISCWare*-component, Input and Output Ports (Push Mode)

and goal-directed applications) to create an optimal and ultimate behavior using existing resources. *RISCbot II* (shown in Fig. 6) [20] contains a variety of sensors such as an array of 13 ultrasonic sensors, an array of 11 infrared proximity sensors above the sonar ring, a Hokuyo Scanning

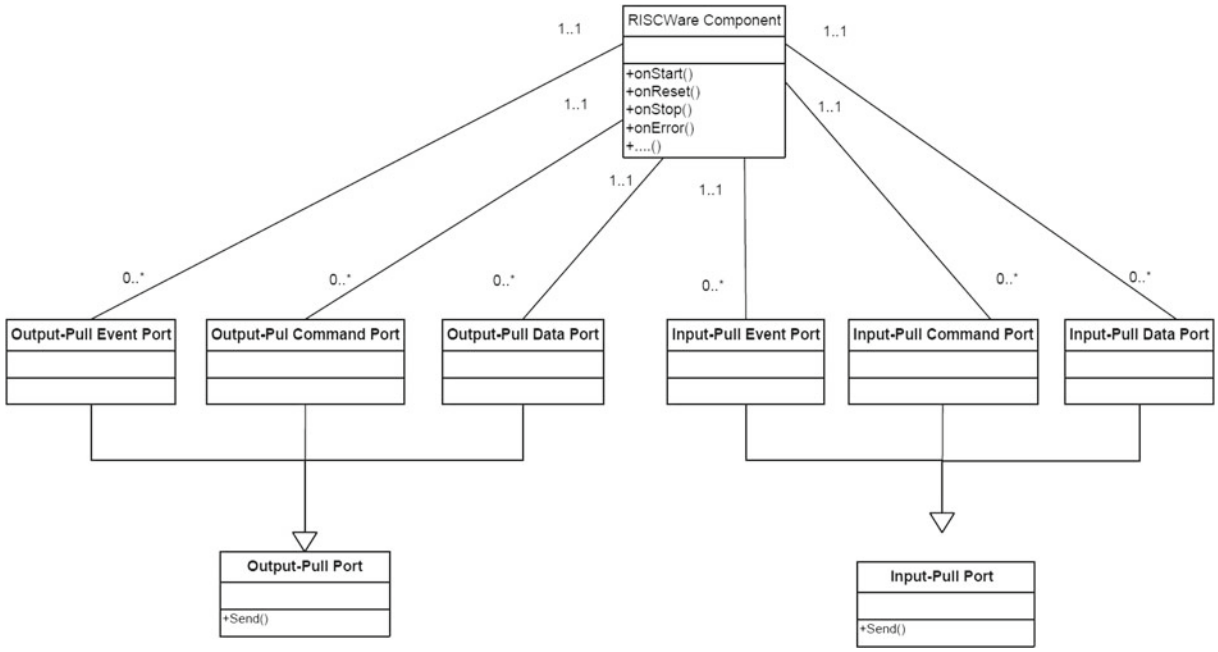


Fig. 11 UML of the *RISCWare*-component, Input and Output Ports (Pull Mode)

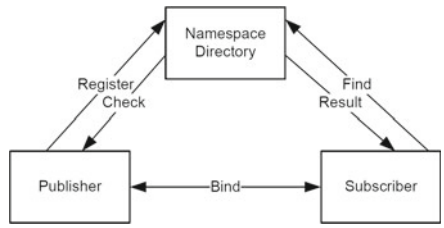


Fig. 12 Collaboration between publisher, subscriber and the Namespace Directory

Laser Rangefinder, wireless network camera and Xbox Kinect. Moreover, the currently developed software applications are: manual driving, obstacle avoidance, and human detection. In order to illustrate that *RISCWare* is a portable framework that can be installed on any mobile manipulation platform, *RISCbot II* is used as a configurable platform to simulate different types of mobile robots via configuring different combinations of directions such as (North, South, East, West, etc.) and the speed limit.

7.1 Experiment 1: Manual Driving Mode, Using a Wireless Camera and *RISCbot II* (Initial Speed = 2)

- Modules:
  - Sensory devices: Wireless camera
  - Software application: Manual driving
  - Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West, North-East, North-West, South-East, and South-West), and initial speed set to 2)
- Description/Results: This experiment illustrates manual-mode driving being performed with fully human intervention operating on the *RISCbot II* to move in all directions at an initial speed 2 (the speed limit is also configurable), using either the GUI controller form, shown in Fig. 13, or the web-based GUI controller form, shown in Fig. 14, which can run on any portable device (i.e. iPhone, Android, etc.). A sample of *RISCbot II* behavior and camera captured images are shown in Fig. 15.

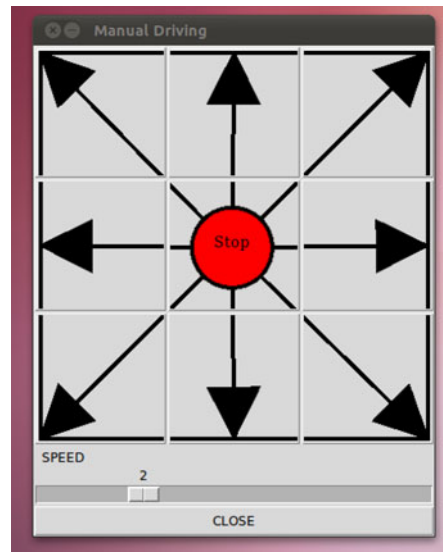
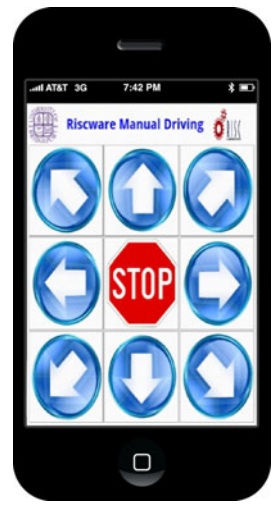


Fig. 13 Manual driving module

7.2 Experiment 2: Manual Driving Mode, Using a Wireless Camera, a Laser Range-Finder and *RISCbot II* (Initial Speed = 2)

- Modules:
  - Sensory devices: Wireless camera and laser range-finder
  - Software Application: Manual driving
  - Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West, North-East, North-West, South-East, and South-West), initial speed set to 2)

Fig. 14 Web-based manual driving module running on iPhone



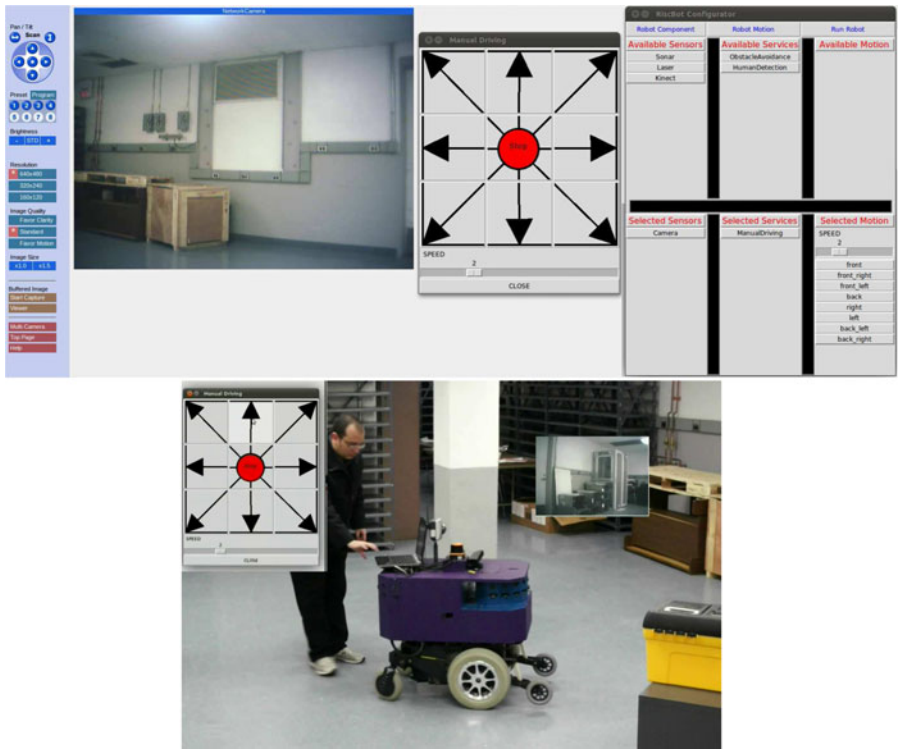


Fig. 15 Experiment 1, manual driving mode using wireless camera

– Description/Results: This experiment illustrates manual-mode driving being performed with fully human intervention, as described in Experiment 1, with the addition of using the laser range-finder as a selected sensory module. As expected, the result of this experiment is the same as what was observed in Experiment 1, since adding the laser range-finder does not have any additional value to the *RISCbot II* behavior.

7.3 Experiment 3: Semi-Autonomous Driving Mode (Manual Driving and Obstacle Avoidance), Using a Wireless Camera, A Laser Range-Finder and *RISCbot II* (Initial Speed = 2)

– Modules:
  – Sensory devices: Wireless camera and laser range-finder
  – Software application: Manual driving and obstacle avoidance
  – Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West, North-East, North-West, South-East, and South-West), initial speed set to 2

– Description/Results: This experiment illustrates semi-autonomous driving (manual driving and obstacle avoidance) being performed with some human intervention and using the laser range-finder sensor and wireless camera as selected sensory modules. Laser range-finder is capable of detecting obstacles at the height of 0.9 meter; *RISCbot II* is able to navigate through obstacles using the Potential Field algorithm for obstacle avoidance in addition to the human-intervention manual driving.

7.4 Experiment 4: Fully Autonomous Driving Mode (Obstacle Avoidance), Using a Laser Range-Finder and *RISCbot II* (Speed Limit = 2)

– Modules:
  – Sensory devices: Laser range-finder
  – Software application: Obstacle avoidance
  – Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West,

North-East, North-West, South-East, and South-West), speed limit set to 2

- Description/Results: This experiment illustrates fully autonomous driving (obstacle avoidance) being performed without any human intervention by using the laser range-finder as a selected sensory module. Laser range-finder is capable of detecting obstacles at the height of 0.9 meter; *RISCbot II* is able to autonomously navigate through obstacles using obstacles using the Potential Field algorithm for obstacle avoidance.

7.5 Experiment 5: Fully Autonomous Driving Mode (Obstacle Avoidance), Using a Laser Range-Finder and Limited *RISCbot II* (North-East, and North-West Directions are Prohibited, Speed Limit Set to 3)

- Modules:
  - Sensory devices: Laser range-finder
  - Software application: Obstacle avoidance
  - Platform: *RISCbot II* (selective directions are prohibited (i.e. North-East, and North-West), speed limit set to 3
- Description/Results: This experiment illustrates fully autonomous driving (obstacle avoidance) being performed without any human intervention by using the laser range-finder as a selected sensory module. Laser range-finder is capable of detecting obstacles at the height of 0.9 meter; the *RISCbot II* is able to autonomously navigate through obstacles using only the permitted directions with a higher speed considering pervious experiments using obstacles using the Potential Field algorithm for obstacle avoidance.

7.6 Experiment 6: Fully Autonomous Driving Mode (Obstacle Avoidance), Using a Laser Range-Finder and Limited *RISCbot II* (North, North-East, and North-West Directions are Prohibited, Speed Limit Set to 3)

- Modules:
  - Sensory devices: Laser range-finder
  - Software application: Obstacle avoidance
  - Platform: *RISCbot II* (selective directions are prohibited (i.e. North, North-East, and North-West), speed limit set to 3)
- Description/Results: This experiment illustrates fully autonomous driving (obstacle avoidance) being performed without any human intervention by using the laser range-finder as a selected sensory module. Laser range-finder is capable of detecting obstacles at the height of 0.9 meter. Since few of the directions of *RISCbot II* were prohibited (i.e. North, North-East, and North-West), the *RISCbot II* available directions to choose from accordingly were East, West, and South as a result, the *RISCbot II* behavior was moving in orbital motion.

Fig. 16 Experiment 9, fully autonomous driving mode with human detection using Xbox Kinect, laser range-finder, and sonar

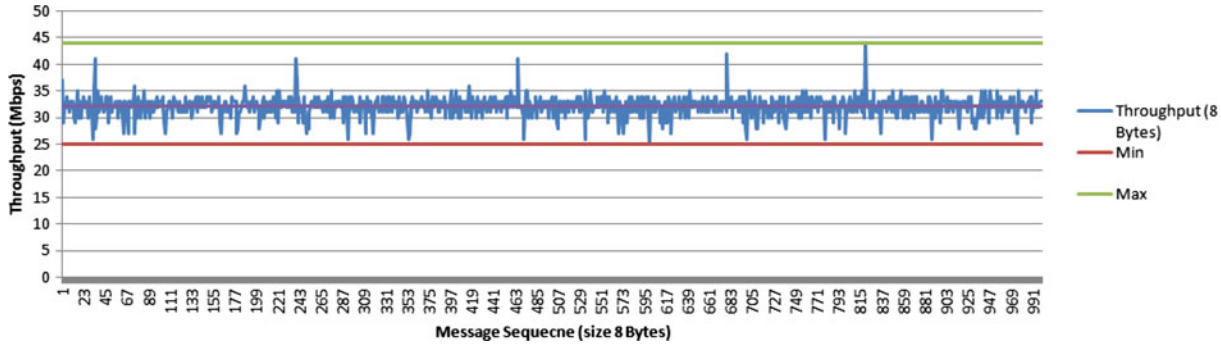


Fig. 17 Throughput of the *RISCWare* tested using a message size of 8 bytes, running on PavilionDV6T Laptop

7.7 Experiment 7: Fully Autonomous Driving Mode (Obstacle Avoidance), Using Sonar Sensors and *RISCbot II* (Speed Limit Set to 2)

- Modules:
  - Sensory devices: Sonar
  - Software application: Obstacle avoidance
  - Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West, North-East, North-West, South-East, and South-West), speed limit set to 2
- Description/Results: This experiment illustrates fully autonomous driving (obstacle avoidance) being performed without any human intervention by using the sonar sensor as a selected sensory module. The sonar is capable of detecting obstacles at the height of 0.7 meter; the *RISCbot II* is able to autonomously navigate through obstacles using the Potential Field algorithm for obstacle avoidance. It was also observed that *RISCbot II* behavior was more optimal using the laser range-finder than using sonar sensors, as the output readings from the laser range-finder are more accurate with a higher sample rate comparing to the sonar.

7.8 Experiment 8: Fully Autonomous Driving Mode (Obstacle Avoidance), Using a Laser Range-Finder, Sonar Sensors and *RISCbot II* (Speed Limit Set to 2)

- Modules:
  - Sensory devices: Sonar and laser range-finder
  - Software application: Obstacle avoidance
  - Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West,

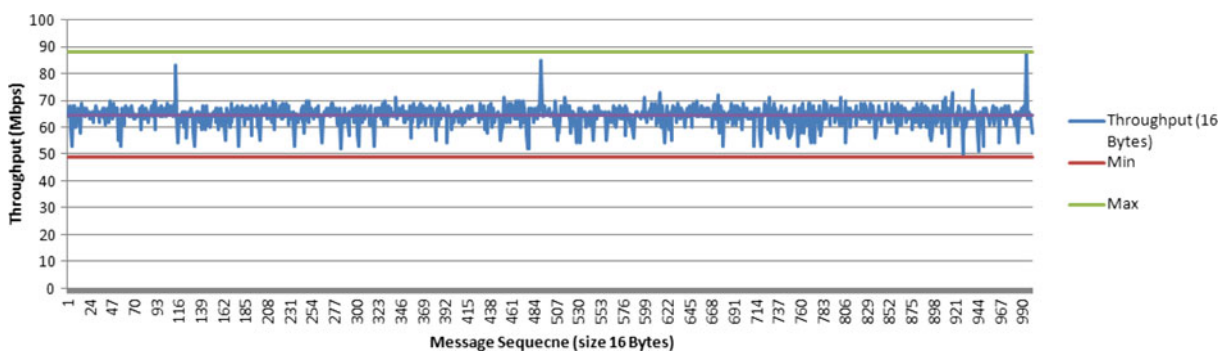


Fig. 18 Throughput of the *RISCWare* tested using a message size of 16 bytes, running on PavilionDV6T Laptop

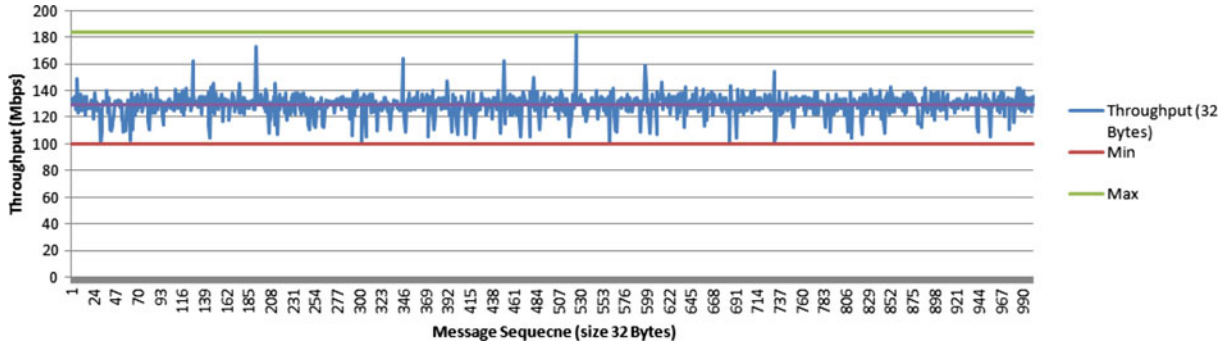


Fig. 19 Throughput of the *RISCWare* tested using a message size of 32 bytes, running on PavilionDV6T Laptop

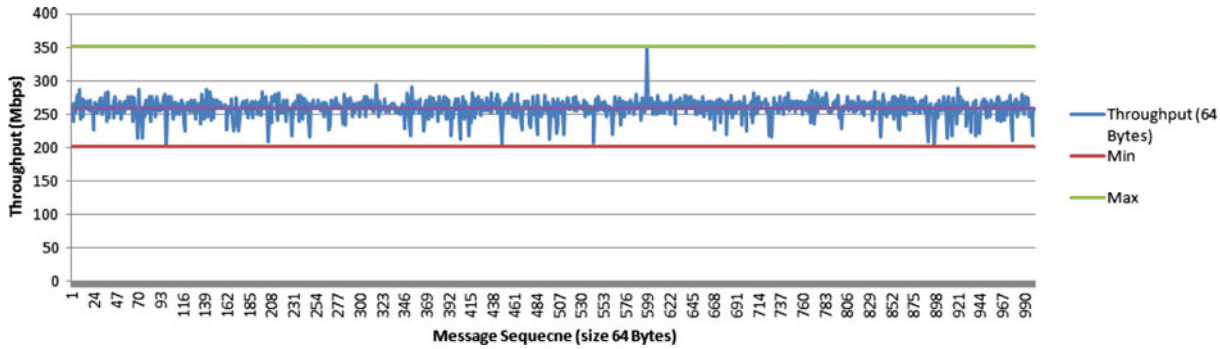


Fig. 20 Throughput of the *RISCWare* tested using a message size of 64 bytes, running on PavilionDV6T Laptop

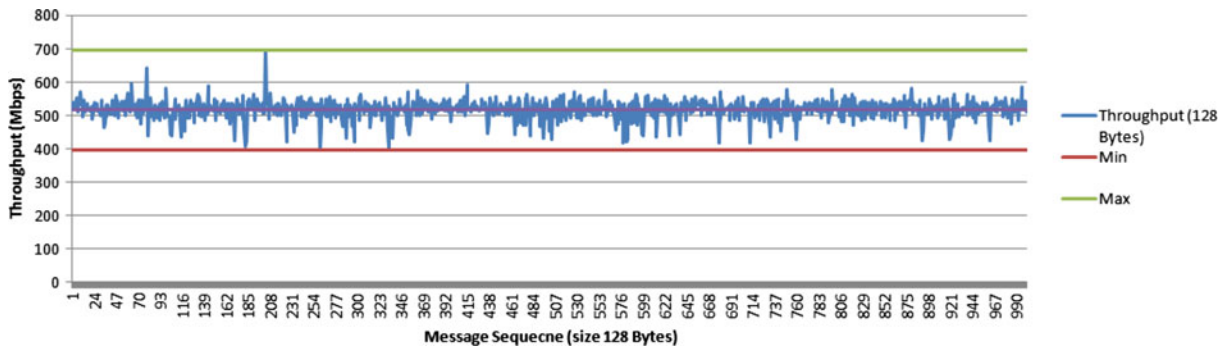


Fig. 21 Throughput of the *RISCWare* tested using a message size of 128 bytes, running on PavilionDV6T Laptop

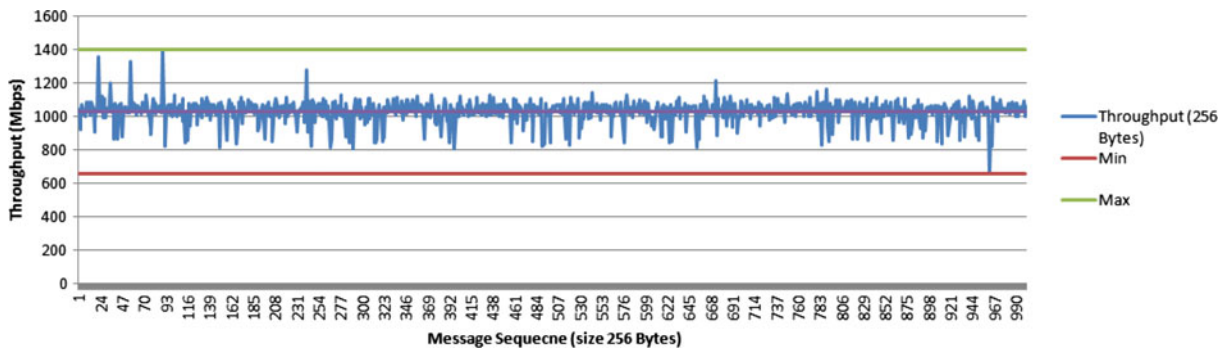


Fig. 22 Throughput of the *RISCWare* tested using a message size of 256 bytes, running on PavilionDV6T Laptop

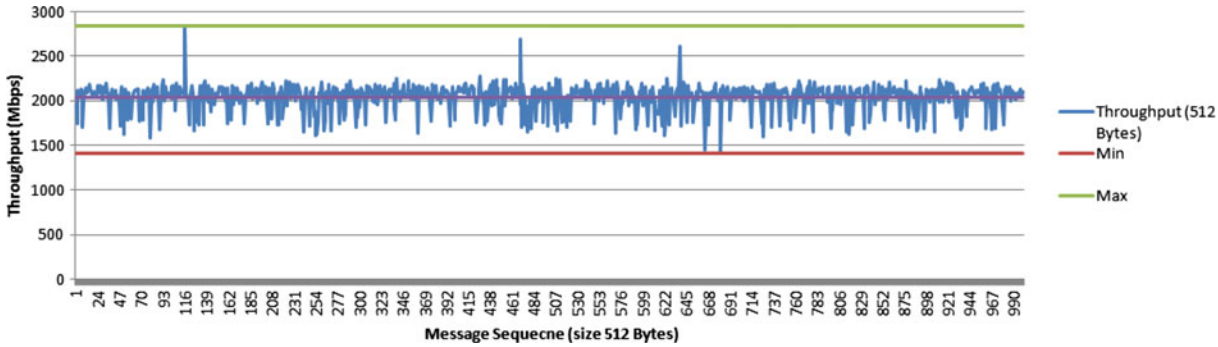


Fig. 23 Throughput of the *RISCWare* tested using a message size of 512 bytes, running on PavilionDV6T Laptop

North-East, North-West, South-East, and South-West), speed limit set to 2

- Description/Results: This experiment illustrates fully autonomous driving (obstacle avoidance) being performed without any human intervention by using the laser range-finder and sonar sensors as selected sensory modules. The laser range-finder and sonar sensors are capable of detecting obstacles at the heights of 0.9 and 0.7 meter; *RISCbot II* is able to autonomously navigate through obstacles using the Potential Field algorithm for obstacle avoidance. In *RISCWare*, sensor fusion is utilized to take the advantage of both sensors' outputs (i.e. laser range-finder and sonar) and generates a faster, more optimal, accurate and robust behavior compared to pervious experiments.

7.9 Experiment 9: Fully Autonomous Driving Mode (Obstacle Avoidance) with Human Detection, Using a Microsoft Xbox Kinect, A Laser Range-Finder, Sonar Sensors, and *RISCbot II* (Speed Limit Set to 2)

- Modules:
  - Sensory devices: Sonar, laser range-finder and Xbox Kinect
  - Software application: Obstacle avoidance and Human Detection
  - Platform: *RISCbot II* (all directions are permitted (i.e. North, South, East, West, North-East, North-West, South-East, and South-West), speed limit set to 2
- Description/Results: This experiment illustrates fully autonomous driving (obstacle avoidance) being performed without any human intervention by using the laser range-finder and sonar sensors as selected sensory modules. Furthermore, *RISCbot II* is able to detect human faces with human detection software module using Xbox Kinect. The results matched those of Experiment 8, with the addition of the human-detection capability via the Xbox Kinect and human detection software module, which is able to detect up to four human faces in the same time. A sample of *RISCbot II* behavior, an output of the face detection module, and the configuration of this experiment, are shown in Fig. 16.

8 Performance Results

Two metrics are used to evaluate the performance of *RISCWare*: latency and throughput. A series of stress tests have been performed, testing different message sizes (16, 32, 64, 128, 256, and

Table 1 Latency and throughput of the *RISCWare*

	PavilionDV6T	Compaqnw9440
Latency	4.59	25.3
Throughput (8 bytes)	32.121	13.53
Throughput (16 bytes	64.365	28.206
Throughput (32 bytes)	129.189	55.376
Throughput (64 bytes)	259.485	113.305
Throughput (128 bytes)	518.183	226.536
Throughput (256 bytes)	1027.752	439.216
Throughput (512 bytes)	2036.389	877.325

Table 2 Comparison of the attributes of the main robotics middleware [15]

Name	System model	Control model	Fault tolerance	Simulator	Linux	Windows	Standards/ technologies	Open source	Behavior coordination	Real time	Distributed environment	Dynamic wiring	Security
RISCWare	6 layers, platform independent abstractions	Event-based mechanism for control flow and the publisher/ subscriber mechanism for data/ event flow	Yes	No	Yes	No	Layered-architecture, Plug Play	No	No	Most modules are real time	Partially	Yes	Partially
CLARAty	2 layer AM; decentralized data; client server; platform independent abstractions and interfaces for various robotic components for motion control, coordination, mobility, manipulation, perception, estimation, navigation and planning.	Supports adaptation for centralized and distributed control; event driven	Yes	Yes	Yes	Only cygwin	OO design patterns, generic programming (C++ STL), ACE/TAO, CPPUnit, Qt, TCP, UDP, Doxygen	Partially	Yes	Most modules are real-time	Yes	Partially	Yes
Player	No particular architectural constraint, Client - Server, Decentralized data	Not applicable; but on module level can be considered centralized, since it relies on polling model	No explicit fault handling capabilities	Stage is a 2D simulator, Gazebo is a 3D simulator	Yes	Yes	3-Tier architecture proxy objects	Yes	No	No	Yes	Yes	Yes

Table 2 (Continued)

Name	System model	Control model	Fault tolerance	Simulator	Linux	Windows	Standards/ technologies	Open source	Behavior coordination	Real time	Distributed environment	Dynamic wiring	Security
OpenRTMaist	Component-based framework, Model Driven Architecture, Platform Independent Model (PIM), Platform specific Model (PSM)	Component-based	Supported by RT-Component model	OpenHRP 3 is a dynamics simulator	Yes	Yes	CORBA	Yes	No	Yes	Yes	Yes	No
OPRoS	Component based, frameworks, validation/ test tools	Client/server mechanism for control flow and the publisher/ subscriber mechanism for data/ event flow	Being developed	Yes	Yes	Yes	Event-driven	Yes	No	Being developed	Yes	Yes	No
ROS	Component based framework, publisher/ subscriber	Message oriented frameworks	Not explicit	Yes	Yes	Partial functions	Message oriented, RPC services	Yes	Yes	Yes	Yes	Yes	No

512 Bytes); the system was run for about an hour and measured the end-to-end data packet latency and the throughput. Two laptops are used for the experiments: HP Pavilion dv6t ($Pavilion_dv6t$) and HP Compaqnw9440 (Compaqnw9440). The specifications of the PavilionDV6T are as follows:

- Processor: 2nd generation Intel(R) Dual Core(TM) i7-2620M (2.7 GHz, 4 MB L3 Cache), with Turbo Boost up to 3.4 GHz
- Graphics: Intel(R) HD Graphics 3000
- RAM: 8 GB DDR3

The specifications of the Compaqnw9440 are as follows:

- Processor: Intel Core 2 Duo T7200 (2.0 GHz, 4 MB L2 cache, 667 MHz FSB)
- Graphics: NVIDIA Quadro FX 1500M, 256 MB
- RAM: 2 GB (667 MHz) DDR2 SDRAM

Figures 17, 18, 19, 20, 21, 22, and 23 show the throughput of different message sizes (16, 32, 64, 128, 256, and 512 Bytes), running on PavilionDV6T. As shown in these Figures, the throughput is doubled as the message size is doubled. The summaries of the average latency (measured in $\mu Sec.$) and throughput (measured in *MBPS*) metrics carried on PavilionDV6T and Compaqnw9440 are shown in Table 1.

9 Technical Contributions

In [15], we described the architecture and some important attributes for most of the existing robotic middleware, such as Player, CLARAty, ORCA, MIRO, etc. As described in [15]. Table 2 summaries and points out the main technical contributions and application significance of *RISCWare*, compared to the existing middleware frameworks.

10 Conclusions

In this work, the *RISCWare* framework is proposed as a modular framework whose hardware and software work together to automatically use available sensing devices and assign resources to perform the required task. Furthermore, the goal is to be able to plug in new sensing devices or new tasks and immediately be able to act on the task, without complicated setup maneuvers. *RISCWare* supports the PnP (Plug-and-Play) property and allows changing the configuration of the control flow and the data flow at runtime. Finally, some experiments, performed on the *RISCbot II* mobile manipulator, were described to evaluate the *RISCWare* middleware.

The primary advantages of *RISCWare* platform are: Software Modularity, Hardware Architecture Abstraction to hide the low-level device-specific details of the device in order to provide the developers with more convenient, standardized hardware APIs; Platform Independence and Portability to be able to run on any platform via changing in the system's configuration. Furthermore, the core of the middleware does not depend on the specific device or software algorithm. In addition, new hardware devices with the same functionality can reuse the existing interfaces and base classes of the already-defined hardware devices; the user needs only to define the hardware-specific functions for them. In *RISCWare*, hardware devices and software work together to automatically use available sensing devices and assign resources to perform the required task. Furthermore, *RISCWare* allows the reconfiguration of the sensors to be used in different tasks such as the stereo camera, which can be used to produce 3D range points and also raw images.

References

1. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
2. Nesnas, I.: The claraty project: coping with hardware and software heterogeneity. In: Brugali, D. (ed.) Software Engineering for Experimental Robotics. Ser. Springer Tracts in Advanced Robotics, vol. 30, chapter 3, pp. 31–70. Springer, Berlin, Heidelberg (2007)
3. Collett, T.H., MacDonald, B.A., Gerkey, B.P.: Player 2.0: toward a practical robot programming framework. In: Proc. of the Australasian Conf. on Robotics and Automation (ACRA). Sydney, Australia (2005)
4. Utz, H., Sablatnog, S., Enderle, S., Kraetzschmar, G.: Miro - middleware for mobile robot applications. IEEE Trans. Robot. Autom. **18**(4), 493–497 (2002)

5. Michel, O.: Cyberbotics ltd. webots tm: professional mobile robot simulation. Int. J. Adv. Robot. Syst. **1**, 39–42 (2004)
6. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.-K.: RT-middleware: distributed component middleware for RT (robot technology). In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2–6 2005, (IROS 2005), pp. 3933–3938 (2005)
7. Schlegel, C., Hassler, T., Lotz, A., Steck, A.: Robotic software systems: from code-driven to model-driven designs. In: International Conference on Advanced Robotics, 22–26 2009, ICAR 2009, pp. 1–8 (2009)
8. Alexei Makarenko, A.B., Kaupp, T.: On the benefits of making robotic software frameworks thin. In: POn the Benefits of Making Robotic Software Frameworks Thin IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'07), 29 Oct.–02 Nov. 2007, San Diego CA (2007)
9. Jang, C., Lee, S.-I., Jung, S.-W., Song, B., Kim, R., Kim, S., Lee, C.-H.: Opros: a new component-based robot software platform. ETRI J. **32**, 646–656 (2010)
10. Jackson, J.: Microsoft robotics studio: a technical introduction. IEEE Robot. Autom. Mag. **14**(4), 82–87 (2007)
11. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: a survey. Auton. Robot. **22**(2), 101–132 (2007)
12. Mohamed, N., Al-Jaroodi, J., Jawhar, I.: Middleware for robotics: a survey. In: 2008 IEEE Conference on Robotics, Automation and Mechatronics, pp. 736–742, 21–24 Sept 2008
13. Mohamed, N., Al-Jaroodi, J., Jawhar, I.: A review of middleware for networked robots. Int. J. Comput. Sci. Netw. Secur. **9**(5), 139–148 (2009)
14. Namoshe, M., Tlale, N., Kumile, C., Bright, G.: Open middleware for robotics. In: 15th International Conference on Mechatronics and Machine Vision in Practice, 2008, M2VIP 2008, pp. 189–194, 2–4 Dec 2008
15. Elkady, A., Sobh, T.: Robotics middleware: a comprehensive literature survey and attribute-based bibliography. J. Robot. **2012**, Article ID 959013, 15 pp. (2012)
16. Elkady, A., Joy, J., Sobh, T., Valavanis, K.: A structured approach for modular design in robotics and automation environments. J. Intell. Robot. Syst. **72**(1), 5–19 (2013)
17. Enea LINX Interprocess Communication (IPC): Online: http://www.enea.com/linx (2011). Accessed 1 Apr 2013
18. Swig: Website: http://www.swig.org/ (2011). Accessed 1 Apr 2013
19. Elkady, A., Joy, J., Sobh, T.: A plug and play middleware for sensory modules, actuation platforms and task descriptions in robotic manipulation platforms. In: Submitted to Proc. 2011 ASME International Design Engineering Technical Conf. and Computers and Information in Engineering Conf. (IDETC/CIE '11) (2011)
20. Elkady, A., Babariya, V., Joy, J., Sobh, T.: Modular design and implementation for a sensory-driven mobile manipulation framework. J. Intell. Robot. Syst. 1–27 (2010). doi:10.1007/s10846-010-9454-3