DEGREE PROJECT FOR MASTER OF SCIENCE WITH SPECIALIZATION IN ROBOTICS
DEPARTMENT OF ENGINEERING SCIENCE
UNIVERSITY WEST

ROS based communication system for AGVs

- A service oriented architecture (SOA) approach

Nithin Ramesh

A THESIS SUBMITTED TO THE DEPARTMENT OF ENGINEERING SCIENCE

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE WITH SPECIALIZATION IN ROBOTICS

AT UNIVERSITY WEST

2016

Date:	March 23, 2016
Author:	Nithin Ramesh
Examiner:	Fredrik Danielsson
Advisor:	Mattias Bennulf, Hogskolan Vast
Programme:	Master Programme in Robotics
Main field of study:	Automation with a specialization in industrial robotics
Credits:	60 Higher Education credits (see the course syllabus)
Keywords:	Industrial communication, AGV communication, multi-agent control
Template:	University West, IV-Master 2.7
Publisher:	University West, Department of Engineering Science S-461 86 Trollhättan, SWEDEN Phone: + 46 520 22 30 00 Fax: + 46 520 22 32 99 Web: www.hv.se

Summary

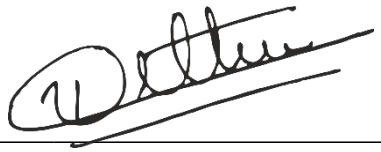
This project first explored various methods of designing a communication and control system for an AGV. It then implemented a SOA based communication system in ROS on the selected AGV. The ROS package created in the project implemented functions of the Aria and ArNetworking libraries from Adept. The next step of the project implemented the functions of teleoperation, mapping and transfer of maps and navigation from Aria into ROS. The packages built implemented these functions in different ways to test the best method to transfer Aria functions into ROS. A generic set of rules were then formulated that aided the conversion of these functions for users unfamiliar with either of the two packages (ROS and Aria).

Preface

The author would like to express gratitude to his supervisor, Mattias Bennulf, for the guidance he received over the course of this work. Also, to XiaoXioa Zhang for helping with the finer nuances of the work. The author also expresses gratitude to Kewal Shaholia and Daniel Nilsson for the time spent discussing aspects of the work and to Ulrica Agell for the help rendered with documenting this work.

Affirmation

This master degree report, *ROS based communication system for AGVs*, was written as part of the master degree work needed to obtain a Master of Science with specialization in Robotics degree at University West. All material in this report, that is not my own, is clearly identified and used in an appropriate and correct way. The main part of the work included in this degree project has not previously been published or used for obtaining another degree.



Signature by the author

28-06-2016
Date

Nithin Ramesh

Contents

Preface

Main Chapters

1 Introduction

A typical production line consists of many blocks of processes working together to manufacture various products. The movement of materials across the production floor as well as from and to storage areas is essential for such a production process to function smoothly and efficiently. An AGV (Automated Guided Vehicle) is one such solution used to move material. Typically, an AGV must have certain basic functions like being able to traverse the production floor either autonomously or with the help of guides, communicating with a central system for scheduling and order allocation and be capable of detecting obstacles and circumventing them successfully. There exist many models of control and communication of an AGV system which have been tested and successfully implemented. One such model is the multi-agent system.

In a multi-agent system, every AGV is afforded some level of autonomy in terms of decision making while the central control unit takes part only in the critical decisions of the system like scheduling, routing and order processing. The AGV itself in such a system is capable of obstacle detection, has knowledge about the fetching and delivery process and has full communication with the control unit. This type of distributed architecture allows for lighter AGV systems with much higher versatility.

This project aims at implementing such a multi-agent system with the aid of ROS (Robot Operating System) using a service oriented architecture (SOA) approach. ROS is a meta-operating system with high functionality that allows for versatile communications in a network. Aided by the vast library and an active development community, it is possible to build a modular system of control for mobile robot applications.

This project works with Adept's patrolbot and uses the base Aria and ArNetworking libraries to build functions into ROS.

1.1 Aim

- To create a ROS meta-server for the purpose of communication with an AGV
- To create a modular system based on SOA that can be scaled up or expanded with functionalities
- To test if there exists a generic method of implementing Aria functions in ROS

2 Background

2.1 Key issues for design of an AGV control system

Communication systems in AGVs are always wireless in nature and range from radio to Bluetooth to Wi-Fi and ZigBee amongst others [1] . In order to have an effective system of AGVs in any industrial environment, it is necessary to design a control system for them. The criteria to evaluate the performance of various methods or models include travel time, vehicle utilization, queue length and cost of handling [1]. According to Le-Anh Et al. [1] and Vis [2], there exist several key issues that need to be addressed while designing a control system for AGVs. They are briefly discussed below.

2.1.1 Guide-path design

Designing guide-paths can be the most basic and crucial step in the design of an AGV system. Most approaches assume that the factory layout as well as the points of contact (i.e. pickup, drop, and charging points) are predefined and will remain constant after the implementation of the AGV system [1]. Once all the paths that the AGV will travel along (guide-paths) have been determined, a flowpath layout can be designed. A flowpath not only consists of an interconnected network of guide-paths but also the direction of travel allowed along each of the paths [2]. The flowpath design directly influences the speed of the AGV system since it decides the path to be taken for pickup and delivery of goods within the factory floor. The direction of travel between nodes (meeting points of two or more guide-paths) can be unidirectional or bidirectional. While bidirectional guide-paths are more efficient, control on these paths is more difficult due to traffic moving in opposite directions. Wherever space allows, a multilane guide-path can be implemented wherein a part of it along its length is designated for traffic flow along a particular direction. Le-Anh Et al. [1] have discussed in their paper the various models of flowpaths and their advantages and disadvantages.

2.1.2 Number of vehicles

In order to determine the optimum number of AGVs that a system should consist of, several factors must be considered [2]:

- Number and type of units to be transported
- Capacity of the AGV
- Speed of the AGV
- Flowpath layout
- Number and location of contact points
- Traffic congestion
- Cost of the system

AGVs can either be single-load or multi-load capacity vehicles. A single-load capacity vehicle can pick up and transport only one unit at a time, while the multi-load capacity vehicle can pick-up several units along its path before reaching the end destination. While multi-load capacity vehicles can increase the throughput of the system, they have a downside of requiring much more complex scheduling methods than with single-load vehicles [1]. The optimum

number of AGVs in a system can be estimated either using an analytical model or through simulation

2.1.3 Management of AGVs

Management of AGVs can be looked at in three forms namely, vehicle dispatch, vehicle scheduling and routing and the positioning of idle vehicles.

Vehicle dispatch is an ever existing function that has its roots from three instances occurring on the factory floor i.e. when a new load has been initialised, when an AGV has completed its drop-off and when an AGV is idle in its parking spot. The dispatching function of a vehicle management system utilises dispatch rules to prioritise loads and choose appropriate vehicles to pick up new loads. The control could be centralised (Entire system is governed from a central location) or decentralised (system has several autonomous control centres that govern a set of AGVs and a zone of the factory floor) [1]. Decentralised systems can have heuristic dispatch rules like random vehicle rule, farthest vehicle rule, longest idle vehicle rule and least utilised vehicle rule which focus on vehicle utilisation or rules like random work centre rule, shortest travel time/distance rule, maximum outgoing queue size rule or a first come first served rule which are based on requests for load pickups [2].

Once an AGV has been decided and assigned for dispatch, it needs to be scheduled a route to follow in order to reach its target pickup and drop off points. The primary objective of the scheduling and routing function is to minimise travel time so that a higher throughput can be achieved. Such a routing function can be devised with the help of algorithms [2]. Algorithms can either be of the static or dynamic type. A static algorithm assigns predefined paths between two nodes, say *i* and *j*. However, this kind of algorithm is highly inflexible to change in load and traffic conditions. A dynamic routing algorithm is based on real time information fed from the system and is thus capable of varying routes based on traffic patterns and load conditions.

Once a vehicle has dropped off its load at the designated area, unless it is immediately assigned another pickup, it turns idle. In such a case, the AGV needs to be positioned so that it can effectively reach the next order whenever assigned to it. Several approaches exist to positioning the AGV once idle [1] [2], prominent of which are:

- Central-zone positioning : Once idle, the AGV is routed to a central location in the factory
- Distributed positioning: Instead of a central zone, several "parking areas" are defined across the factory and the AGV is routed to the nearest one. This is often effective when the size of the factory floor is rather large and it takes a long time for the AGV to traverse across it.
- Drop-off point positioning: The AGV stays at the location of its previous drop-off until assigned a new pickup.
- Circulatory Looping: A circulatory loop is defined and the AGV is made to run along the loop until reassigned.

2.1.4 Battery management

Although battery management is usually not considered important when designing an AGV system, in cases of high utilisation and low off-times, it becomes crucial to include it as a part of the system. It isn't necessary for systems that contain AGVs with an on-line charging system or when utilisation of the system is below 50% [2]. The effect of incorporating battery management could lead to higher number of AGVs or replacement batteries for existing AGVs thereby increasing the cost of the overall system.

2.1.5 Failure management

Failure in a system of AGVs could occur from individual equipment failure [2] or from over-crowding of AGVs causing a deadlock in traffic [1]. Though there isn't much research on the effects of failure of AGVs on the performance of the system, it is easily possible to design a system so that a deadlock can be prevented. Some techniques to avoid deadlocks are queue utilisation, balancing workload, controlling traffic at intersections and with zone planning.

2.2 Models of communication and control systems for AGVs

There exist many methods of wireless communication that can be implemented with an AGV for communication and control. While the method may vary, the system in itself will have to take into consideration all the factors previously mentioned while deploying an AGV. Below are discussed several example works that utilise different methods of communication.

2.2.1 AGV Communication system using wireless IEEE 802.11g network

Liu et al. [3] developed a server based communication for a multi-AGV system used for transportation in a life science laboratory. The authors chose to use an IEEE 802.11g wireless network over other communication methods like Bluetooth and GPRS/GSM due to its large bandwidth and fast data transfer abilities which is ideal for real-time monitoring systems. They state that having several robots connected over the same network may cause congestion and delay in data transmission. In order to avoid this and to stabilise the network, they utilised a 2.4 GHz wireless marine bridge to act as a bridge between the router and the robot.

The system architecture is modelled such that all the robot boarded control centres (PCs) are connected to the same external network via a router. Also connected to the same network is a remote server control centre (PC) that acts as a bridge to higher operational commands from a process management system (PMS) and an optional robot database. The robot boarded control centres were mounted onto the mobile robots and connected via a robot switch that had access to motion control, sensor data and the monitoring cameras.

In order to optimize navigation within the laboratory environment, they focussed on two aspects, namely localization and obstacle avoidance. To tackle localization, the authors used a hybrid system consisting of an IR radio module called StarGazer coupled with passive IR markers that had independent IDs mounted onto the ceiling of the laboratory. Their research suggested that the AGV could be localized very accurately using this system. The maximum number of IDs that could be assigned via this method was found to be limited at 4095. Obstacle avoidance was achieved via an on-board sonar (DUR5200), a human sensor module (DHM5150) and IR sensors at the front of the robot base. Utilising the data from all these sensors, an optimal path planning algorithm was developed and implemented.

To complement the hardware, the authors developed a GUI (Graphic user interface) that allowed the user to view the location of the robot, plan its path, control each robot individually and get an overview of the system. Data integrity was assured in the system by assigning individual IP addresses to each device and establishing individual connections from each of them to the central server.

2.2.2 AGV Communication system using Bluetooth and radio signals

Boje and Kotze [4] implemented an interesting system for AGV communication. Motion control of the AGV was done with a control PC mounted on it that communicated via radio

signals. Whereas, the actual control of the AGVs i.e. task assignment, routing and scheduling was done via a base control station using Bluetooth communication. The authors opine that Bluetooth is effective in small work areas and are easy to integrate with systems all over the world.

They localized the AGV position using an indoor GPS system consisting of a set of infrared (IR) and ultrasonic pulses sent from fixed locations at fixed times. Since ultrasonic waves travel much slower (at the speed of sound) than IR rays (speed of light), the difference in time between receiving the IR and ultrasonic pulses could be used to determine the distance from the sensor. The AGV used a method of triangulation to determine its exact location in the workspace. If the AGV received less than three sets of pulses, it was automatically registered as having travelled to its maximum boundary and was to find its way back its last known location in the workspace. Obstacle avoidance was achieved using a set of Sharp GP2D12 IR sensors. Since the output of the sensor was logarithmic between the ranges of 10 cm to 80 cm, it was ideal for detection. In order to avoid the problem of sensing objects lesser than 10 cm from the sensor, the sensors were mounted in such a way that the lower limit (10 cm) would fall within the outer boundary of the AGV.

The communication with the AGV took place via a Bluetooth link with the AGV control PC. The base control station sent control signals and received positional and motional data (speed, direction, angle of wheels etc.) from the AGV. The authors also developed a user interface (UI) to allow for human interaction with the AGV. The AGV could also be controlled via a joystick that was mounted alongside the control PC.

2.2.3 AGV Communication system using ZigBee

Chen et.al. [5] designed a humanoid AGV system meant for catering applications. Meant to be used in restaurants to deliver food and interact with customers, the service robot can be considered an AGV for the purpose of study. The authors designed the robot to have human-like features to increase the comfort level of interaction with customers. Each individual catering robot has a decentralized control system on board which has access to various functions including motion control, path sensing and guidance, obstacle sensing, audio and touch screen input from users and display and speaker for output functions.

The robot is equipped with 8 ultrasonic sensors to aid obstacle detection as well as high accuracy wheel encoders and a payload capacity of up to 23 Kgs. The system has multi-point input for the robot to get orders from. Customers are provided with pagers to place orders, while the kitchen has a central server that coordinates these orders and signals to the robot when food is ready to be picked up and served to the customers. The overall communication is handled using a ZigBee network, where each input point (i.e. pagers on tables, server at the kitchen and robot control unit) are all nodes connected to the network. In order to generate a map of the working area, the authors manually draw a map into Mapper3, a mapping software which can be used in tandem with MobileSim, a simulation software designed for the robot. With the aid of the simulation software, the robot is able to navigate paths along the mapped area with high accuracy.

The authors tested the robot in an office scenario with a complex grid of cubicles to judge its ability to communicate with the server and navigate across various paths. The ZigBee communication system proved fruitful and the robot was able to complete a path across the office environment effectively while avoiding collisions.

2.2.4 Decentralized AGV system

As discussed before in section 2.1.3, control of an AGV system can either be centralised or decentralised. When control is decentralised, the system consists of several agents (decision making entities in the system) that are connected and work together to solve problems that are either beyond the capabilities or knowledge of individual agents [6]. Agents can be low-level i.e. allowed to take decisions within a limited domain or high-level i.e. control wider actions of a group of agents for a given set of functions.

Farahvash and Boucher [6] as well as Weyns et al. [7] describe a multi-agent approach to designing a complex AGV system. The system designed by Farahvash et al. [6] consists of 4 primary agents namely cell agent, scheduling agent, material manager agent and traffic controller agent. Based on a set of inputs from the system or environment, each agent takes a decision which is then fed into a relational database. Every agent also takes input from the relational database, thus making a bridge of communication between the various agents. In order to aid the decision making process, each agent is provided with a set of methods it can choose from based on the scenario. The methods described in section 2.1.1 for traffic control and management can be considered as examples for the same.

Weyns et al. [7] have a different approach wherein they define two types of agents namely, transport agent and AGV agent. As the name suggests, each AGV is handled by an individual AGV agent. All the AGV agents work in tandem with the transport agent to receive an order and report once the order has been completed. The transport agent decides which AGV is to be scheduled based on the location and status from each AGV agent. It is located centrally at the transport base on a dedicated terminal. The transport agent is shielded from low-level messages and communication like AGVs interacting with sensors and such through the use of a virtual environment. The virtual environment is segmented between nodes and allows the AGV autonomous control till it receives the next command from the transport agent. This also allows for the transport agent to communicate with a large number of agents without any delay in the communication.

2.3 Service Oriented Architecture (SOA)

The two approaches in section 2.2.4 above showcase a new trend in control architecture called service oriented architecture (SOA). With multi-agent systems, it is possible to request or subscribe to a service from an agent on the same network when required, thus, allowing for a streamlined system with agents that are capable of specialised tasks [8]. There are various software solutions that handle integration of multi-agent systems into service oriented systems. SOA is an approach used to create an "architecture" using service-client interactions as a base. A service is usually a software component that performs a small function, which in the case of an AGV could be reading battery level, enabling/disabling motors, reading odometry information or sensor data. One of the advantages of an SOA is that the components or services that are a part of it don't need to be bound together as strictly as in a traditional software program. This is especially useful when a part of a system needs to be modified or changed. When such an instance occurs, the functioning of other parts of the SOA system don't get affected. Another advantage is that SOA allows for interoperability between different applications running on different platforms. [9]

An SOA could be considered to have 3 elements, namely, service providers that publish services, service consumers or clients that utilise the services and a central service registry that maintains a list of clients and services for mutual discovery.

2.4 ROS

Robot Operating System (ROS) is an open-source meta-operating system for robots. It can in some ways even be considered a middleware or a framework. It currently runs only on Unix-based platforms and tested primarily on Ubuntu and MacOS X systems, while development support is availaible to other Unix platforms like Debian, Fedora, Gentoo and other Linux platforms. Amongst its various release versions or "distros", the Indigo distro is stable and is provided with Long Term Support (LTS). ROS allows for the implementation of programs written in a variety of languages including C++, Java, python and Lisp. ROS packages (explained in detail in the next chapter) consist of a variety of individual programs compiled together [10] [11].

2.5 ROS concepts

ROS has 3 levels of concepts namely the ROS filesystem level, ROS computational graph level and the ROS community level. This paper shall provide some introduction to various aspects of ROS at each of these levels. [10]

2.5.1 ROS nodes and the master node

A robot control system in ROS consists of various processes each running in a node. Every node opened by a process is streamlined to perform one specific task. As an example, a node could enable/disable motors or publish sensor information or perform path planning or localization. Nodes are usually written in one of the languages compatible to ROS like C++ or python using a client library like roscpp (for C++) or rospy (for python). Every node must be assigned a unique name on the ROS network. Having two nodes of the same name will cause the first node to shut down when the second is initialised [10].

ROS requires a network to have a Master node. The ROS master acts as a central registry and provides services like name registration and lookup to the peer-to-peer network of ROS processes. The Master allows various nodes in the ROS network to find each other, data and discover/utilise services. It also runs a parameter server which maintains setup data for all the nodes. This data can then be accessed by any node running on the system [10].

2.5.2 ROS messages, topics and ROS bags

Nodes on the ROS network communicate with each other by passing messages. There are various message types which allow anything from a simple bool or int to large sensor data to be transmitted. A message is basically a data structure consisting of various typed fields and can even hold nested structure or arrays. It is even possible to define custom message types to hold messages that don't fit into any of the predefined types.

These messages are accessible to all nodes using a transport system that allows them to publish/subscribe to the message broadcast. The node sending out a message publishes it to a topic. The node that looks to receive messages subscribes to the same topic and is able to read the messages being transmitted. It is possible to have more than one node publish to the same topic as well as multiple subscribers to the same topic. It is also possible for a single node to publish or subscribe to many topics at once. By allowing incorporating this functionality, information production and consumption is essentially decoupled.

ROS bags are used to save information gathered during a process and also to playback the data. It could be used to store sensor data, key inputs etc.

2.5.3 ROS Services and ActionServers

While the functionality of the publish/subscribe model is very robust and effective for data transfer, it isn't suitable for a request-reply type of interaction. ROS provides the functionality of services to allow for request/reply interaction between the various components of a distributed SOA architecture. Nodes can offer services under a particular name and other nodes can avail the service by sending a request message to the service node and awaiting a reply.

While traditional servers in ROS allow for a request/reply interaction, it is not possible to pre-empt a request with another one. When such a situation is required, it is possible to make use of the ActionLib stack in ROS to implement an ActionServer. The ActionServer and its ActionClient communicate on the "ROS Action Protocol" built on ROS messages. The action itself consists of three parts, namely, a Goal, feedback and Result. The goal could be the final value of a parameter being used to control a process. For example, in navigation, a goal could be the pose of the location the AGV should reach. Feedback tells the client the current progress of the action, while the result is a message sent at the end of the action exactly once. The result and feedback messages might be completely different types or of the same type.

2.5.4 ROS packages

A ROS package can be considered to be the most basic component that can be built and released in ROS. A package can consists of a variety of nodes, services, actions, libraries, configuration files and any other dependencies it may need. Once built, the package can then be run on ROS.

2.6 ROS communication

Error! Reference source not found. shows how the various components describe in the previous section interact with each other. As seen, each package can consist of a combination of nodes, services, clients and bags. When a node is initialised on the ROS network, it registers itself with the master node. When nodes need information from each other, they route their queries through the master node. Once communication is established, data transfer takes place.

2.7 Some common ROS commands

ROS is usually initialised from the command-line. Following are some common commands used to run various aspects of ros:

- rosrun <package_name> <node_name>
  This command runs a node (node_name) from the specified package (package_name). It is not necessary for the user to change to the directory of the package to run it. ROS finds the package automatically and runs it
- rostopic <bw> <delay> <echo> <find> <hz> <info> <list> <pub> <type>
  This command does various operations with respect to a particular topic. General syntax is rostopic <command_name> <package_name>/<topic_name>. The various commands do, respectively, display bandwidth, display delay on the topic, display messages published to the topic, find topics by message type, display publishing rate, publishes information about the topic, publish to a topic and displays the topic type

- roslaunch <package_name>
  Some packages are initialised with a launch file. This command finds the .launch file and runs it.

2.8 Aria and ArNetworking

Aria or Advanced Robot Interface for Applications is a C++ library developed for Adept's MobileRobots AGV platforms. The library consists of functions that enable the user to control robot motion parameters, receive odometry and sensor data and contains tools that allow users to integrate all these functions with other custom hardware added to the AGV. The Aria library also contains functions for laser mapping, localization and navigation.

ArNetworking is a part of the Aria library that contains functions for remote controlling of a robot over a network. The ArNetworking server and client functions are used to set up a centralised server and communicate with the AGVs on the network. These functions can be used to connect remotely, access sensor data and send commands to the AGV [12].

2.9 Mapper3 and MobileEyes

Mapper3 is a software application that allows users to edit and convert laser log scans (.2d) from an Adept robot into maps (.map). It also allows the user to add goals, forbidden lines and areas, home points and areas, docking stations amongst other mapping objects. The map files generated by Mapper3 can then be used by the AGV to navigate the mapped area [13].

MobileEyes is another software application developed by Adept meant to be a graphical visualisation tool to remotely monitor and control an AGV. It works on the ArNetworking protocol and requires the AGV to be connected on a server. The application allows the user to load maps, command the AGV to go to a goal, home or dock positions or to teleoperate the AGV

2.10 Previous work done on the PatrolBot

The Patrolbot at University West has been used as the platform to conduct this project. The PatrolBot previously had been programmed for tasks like navigation and mapping using the Aria and ArNetworking libraries. The previous works on the robot also included implementing these functions through a web-based HMI for remote operation as well as to enable the AGV to communicate with other industrial devices like a PLC. The system and its functions had been tested indoors for factors like precision, communication and robustness. Most of the original functions were written in C++ in accordance with the Aria library requirements. [14]

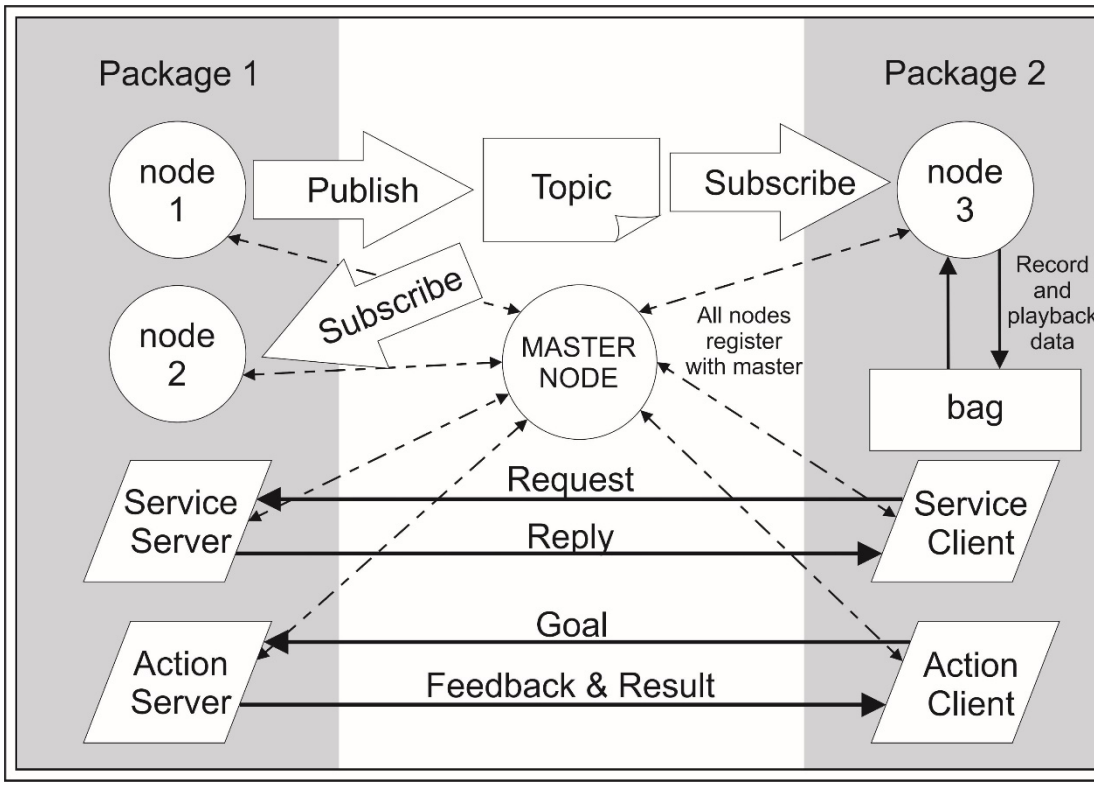


Figure 1 ROS communication model

3 Method

In order to create a multi-agent system on ROS, this project was aimed at creating a set of ROS packages which could be fully portable to any control unit or AGV that would be added to the network at any later stage. The ROS packages have incorporated various aspects of the native library provided by the selected AGVs manufacturer to make them fully functional within a ROS environment while utilising all advantages that ROS has to offer. The packages have been designed such that any outside system, for example a remote user, need not have the native library installed in order to control the actions of the AGV. The project looked to handle some preliminary functionalities for the AGV including communication, mapping and navigation.

In order to solve the above requirements, the project was handled in 3 steps.

3.1 Literature study

In order to get an idea about AGV control systems, the project was started with research into pre-existing solutions for control. The study took a preliminary look at the various methods of communication to gain a better understanding of the different forms that existed in today's industrial scenario. The literature study also looked into the requirements of an AGV control system, while determining that a decentralised service oriented approach was the best way to solve the problem.

3.2 Studying software libraries related to the selected AGV

The libraries provided by the manufacturer for the selected AGV were studied in order to implement predefined functions and functionalities. Once the functions necessary for the project had been identified, all of its individual components were also explored so as to get a better understanding of their working. At this stage, if there existed any previous work done on the same libraries with ROS, the same could be studied as well to gain perspective about a method of implementing selected functions in ROS.

3.3 Design and implementation of the ROS system

The ROS package was planned to include scripts that could start a server on the robot, start the laser scanners, allow for teleoperation control of the AGV, log scans to a file, generate maps, load/transfer maps and allow for the user to command the AGV to go to specific positions on the map i.e. navigation.

In the duration of the project, the possibility of creating a generic method to convert functions in the libraries provided by the manufacturer so that they could run as ROS functions was also explored. This was so that more functionality could be added with ease even by programmers not familiar with the project, thus affording it with some level of modularity.

4 Implementation

As mentioned in the previous section, the project was implemented in three stages. The literature study showed that using a multi-agent system built on the principles of service oriented architecture would be a highly effective method of setting up a control system for AGVs. ROS allows for such service oriented interactions between different components of a network.

4.1 Project set-up

The robot used to test the various functions being implemented during the project was a patrolbot from Adept. The patrolbot has an on board computer, laser and sonar scanners, gyroscopic correction with a payload of 40 Kg. The on-board computer is running on the Debian Wheezy 7.1 platform. ROS indigo was chosen as the distro since it is currently the most stable version released as LTS (Long term support). The latest versions of Aria and ArNetworking libraries were also installed on the patrolbot. The same patrolbot has been used at the research centre previously for implementation of various Aria functions through a web-based interface. This project was meant to test portability of those functions into ROS while effectively utilising all features of ROS.

All the packages were built and made in a catkin workspace. The scripts were all written in C++. The catkin build replaces the traditional .makefile with a .Cmakelist. Hence, all packages had to have custom built .Cmakelist files. The CmakeList file lists all dependencies of a package, links target libraries, install files and has a list of all executable nodes, services and messages in the package.

The network used for testing was a WLAN (wireless local area network). A laptop was used as another system on the network running on Ubuntu 12.04 with ROS hydro. For ease of testing, the master node was created on the patrolbot, but ideally, a control terminal on the floor should be made the master in order to communicate with multiple AGVs.

All code part of this project can be found here: https://github.com/nithin2302/robot_interface

4.2 The RosAria package

As mentioned before, there have been a few attempts to incorporate functions from the Aria library into ROS, the most prominent being the RosAria package. While this addressed the most preliminary aspects of Aria, it did not however breach the subject of ArNetworking. As a result, any functionality requiring the use of ArNetworking's libraries could not be directly used in ROS. To understand better how Aria was initialised and incorporated with ROS, the RosAria package was studied in some detail.

The RosAria package [15] incorporates functions of Aria as a class. The RosAria class initiates a ROS node with the name "RosAria". It publishes data on the ROS network on the following topics:

- pose_pub – Current position of the AGV with respect to its initial starting position
- bumpers_pub – State of bumpers i.e. whether collision has occurred or not

- sonar_pub, sonar_pointcloud2_pub – Readings from the ultrasonic sensors published as a 2d plot
- voltage-pub, recharge_pub, state_of_charge_pub – Data on battery state of the AGV
- motors_state_pub – Whether motors are published or not

It also subscribes to the cmdvel_sub i.e. the linear and angular velocities of the AGV. Alongside, it also provides a service to enable and disable the motors. It also provides the function to enable and get mapping data from the lasers on the AGV. This function has not been explored since this project explored an individual function for mapping with the lasers.

4.3 Program functionalities

The program implemented several functionalities of the Aria library utilising ArNetworking. It endeavoured to create an example program incorporating each functionality of ROS to test compatibility. The project has also tried initiating Aria at different levels in the program. The RosAria package initiates RosAria and its objects through a class object. This project has initiated the same through the main function and through a function in the program. It implements all functionalities through 2 packages. The teleop_action package contains functions only for teleoperation of the AGV and it depends on the RosAria node. The robot_interface package consists of mapping, map transfer and navigation nodes and are independent of other packages while having dependencies only on the Aria and ArNetworking libraries.

4.3.1 Teleoperation using the RosAria package (teleop_action package)

The RosAria package connects to the motors of the patrolbot but does not provide any functions for teleoperation using a device connected on a network like a joystick controller or keyboard. The TeleOp_action package implements teleoperation in two ways. The first is through a simple subscriber. The second is using an action server. The RosAria node is to be run while using this package as it has dependencies on it.

In the first approach, a node running on the laptop on the network logged keys from the keyboard as they were pressed and published them to a topic. The idea behind this was that any device like a PLC, an Arduino board, a game controller or keyboard could send simple int values when buttons are pressed or joystick is moved. The node running on the AGV would read values from the subscribed topic and use the values to move the AGV around. For easy usability, the numbers 1 through 6 were used respectively for move forward, backward, left and right, stop and quit the teleoperation. The node on the AGV is started using "$ rosrun /teleop_action/TeleOp_SubnDo".

While using an action server, it is possible to pre-empt actions. Hence, using this, only one client can be connected at a time. When another client requests for teleoperation, the first client is automatically disconnected due to pre-empting. On the user side is the same program that logs keystrokes and subscribes as well as a node that behaves as a client for the action server running on the patrolbot. The client sends a goal of "6" which is the int assigned to quit the teleoperation. As long as the goal is not equal to the values arriving on the subscribed topic, teleoperation continues. The server can be started with the command "$ rosrun /teleop_action/TeleOp_Server" and the client using "$ rosrun /teleop_action/TeleOp_Client".

4.3.2 Mapping with Lasers using ArNetworking (ROS_sickLogger node)

The Aria library has a predefined function called sickLogger. This package was built on the basis of that function. Here, Aria has been initiated in the main function and remains running

until the node is closed. The logging function was then initiated under a ROS node. The logging function begins after the robot is moved and continues to map until the mapping function is closed. During mapping, it is also possible to provide gals to the robot at locations that might be required. Adding goals can also be done at the later stage when editing the map. It then generates a .2d file containing the logged details. This file can then be exported into other systems either using the send_map node (discussed below) or using an external flash drive. The .2d file is then edited using mapper3 to define goals, home positions and the docking point. It also allows the user to define prohibited areas and prohibited lines. The saved file then has the extension .map for the robot to recognise that it is a map file. It is not possible to publish the sensor data directly through this function to a topic since the laser logging function in Aria closes off the data.

4.3.3 Transmit map between different parts of the network (send_map node)

This is a simple set of scripts that are run on the patrolbot and the client terminal. If we are to transfer a map from the robot to the terminal, the command "$ rosrun /robot_interface/send_map" is used. This opens up a publisher on the robot which waits for a subscriber on the client side. Once the subscriber to the map is opened, the script publishes the contents of the chosen file and then stops publishing at the end of the file. The subscriber reads the values off the topic and saves is as a .2d file on the terminal it was run in. The same can also be implemented as a service based interaction wherein the client requests for a map to be sent or requests for the AGV to load a map from the client.

4.3.4 Navigation of the AGV using ArNetworking (PBGo_server, PBGo_client nodes)

In order to run these nodes, an ArNetworking server is required to be run on the robot. The server takes arguments to define the host and the map the robot is to use. This is a functionality implemented purely using the Aria functionality. The code from the previous work has been used unaltered. Since this needs to be run only on the robot, it was concluded that it is not required to implement this into ROS. The server also has the functionality to allow for maps to be switched in the go. This is especially useful when the robot enters a different area, for example when it enters a new level of the building.
The PBGo_server node implements the Aria initialisation in a function within the main function. Adapted from the navigation code from the previous work done on the same patrolbot, the script contains two classes, one to handle inputs and the other to handle outputs. The InputHandler class contains member functions initialised with pointers to the robot object and the goal requested for the robot to go to specific goals or a set of coordinates or the home and dock positions. The OutputHandler contains member functions initialised with a pointer to the robot object and publishes the current status of the robot and the goal it is moving towards. Once the robot reaches its goal, it publishes success of navigation or prints error messages in case the robot gets lost or is unable to reach the goal or if the robot disconnects for some reason. Once the client requests a goal, the function to go to a goal is called. The server object in ROS can be used to signal a bool type callback function and passes two pointer arguments namely the request and reply to the function. At the end of the function, i.e. when the AGV has reached its goal, the function returns the value of the reply as the goal having succeeded. The PBGo_server has dependencies on the Aria and Ar:Networking libraries and can only be run on an adept robot.

The PBGo_client can be run from any terminal with ROS. The client requests the AGV to go to a particular goal or to the home or dock positions. This a pure request-reply interaction and hence, it has no dependence on any libraries outside of ROS. The client waits till it gets a confirmation that the robot has reached its goal and then exits.

The robot uses the ArNetworking and Aria libraries to navigate along the loaded map. It utilises the lasers to perform localisation and updates its position accordingly. The AGV can also be controlled using MobileEyes with a loaded map.

5 Results and discussion

5.1 Generic method of implementing Aria functions in ROS

The project implemented several functions from the Aria and ArNetworking library as shown above. The initialisation of Aria and the creation of the robot and robot connector objects are crucial to any Aria function implemented on the patrolbot. This was tested in three scenarios. Through the RosAria node, the initialisation was done through a class, wherein a new robot and connector objects were initialised every time another instance of the class was created. These objects were used by the teleop_action pakage. In the robot_interface package, the Aria objects were initialised once through the main(ROS_sickLogger node) and through a function within the main(PBGo nodes). In all three cases, it was possible to access the Aria robot objects and perform operations on the robot.

In order to have catkin find the Aria and ArNetworking libraries, it was necessary to make some changes to the Cmakelist file. The libraries had to be linked physically as installation of the Aria libraries provided by Adept don't follow Debian policies.

It was also found that the Aria robot connector could be parsed with arguments in two ways i.e. using the argc and argv arguments from the main or it can be manually built using an argument builder. Considering all these facts, the implementation of various other functions of Aria in ROS can be considered to be straightforward. The steps to do the same would be:

1. Edit Cmakelist file to include dependencies on Aria and ArNetworking components
2. Also physically link the libraries and their directories in the file so that catkin is able to find the libraries during make
3. Include Aria/Aria.h and Aria/ArNetworking.h header files in the script
4. Implement functions as required
5. Make sure to initiate Aria (Aria::init) before performing any operations with Aria functions and objects

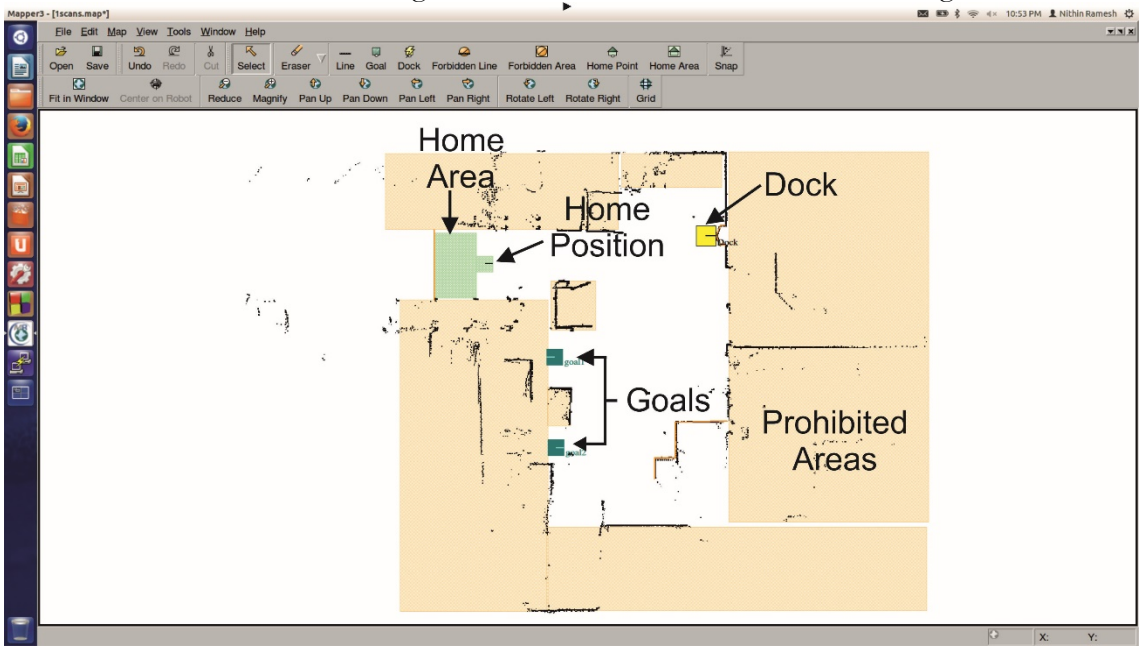
5.2 Security protocols in ROS

ROS does not have any inherent protocols that allow for security of data. Since ROS runs on a network, the primary level of security would be the network security itself. A network with a higher level of encryption would possibly improve security to a large extent. Aria allows for a login option wherein entered username and password are compared against a central database on the server. When implementing Aria functions, it is possible to use this feature as an added level of security.

Figure 2 2d point-cloud data from the laser sensor visualised using Mapper3

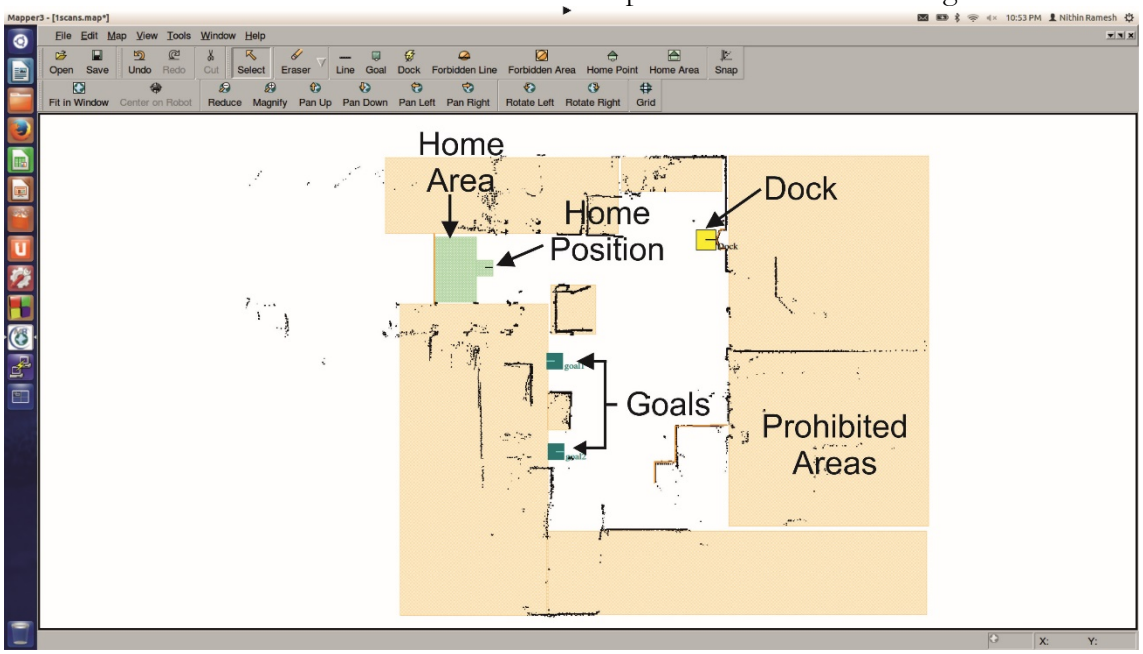
5.3 Mapping and editing the map

The below Figure 2 and Figure 3 show the map generated by the ROS_sickLogger node before and after editing the map. Editing of the map is done using Mapper3. As seen, the definition of various components of the map like goals, dock, home regions and prohibited areas are very straightforward.



5.4 Result of navigation

The below shown map in Figure 3



was used to start a server on the patrolbot. When the PBGo_client node is started, it prompted the user to enter the name of the goal they wish to go to, i.e. home, dock or goal#. Once entered, the patrolbot moves to the entered location and the server replies with a success message.

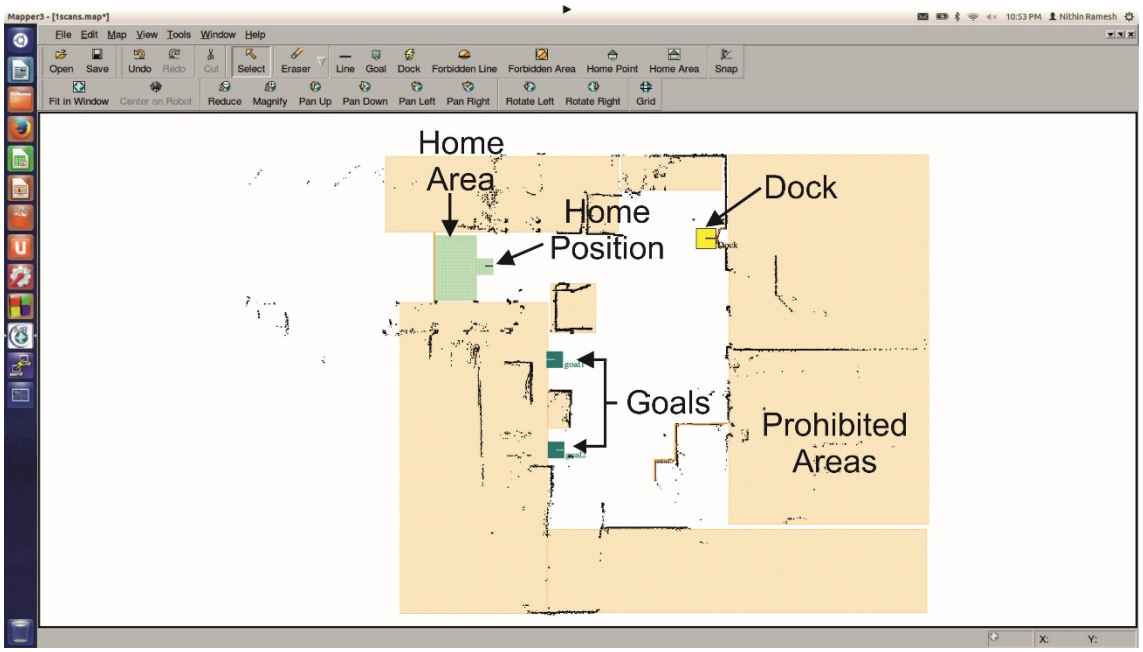


Figure 3 Map after editing and defining various positions, goals and areas

6 Conclusion

This project explored the possibility of using ROS as a communication system to control an AGV. From the results, it can be seen that it is possible to implement various functions and functionalities provided by manufacturers in their native libraries using ROS. The various aspects of ROS, like publish-subscribe, services and bags can be used optimally to control and communicate with an AGV on a network. The publish-subscribe functionality is very efficient at broadcasting and receiving information pertaining to sensor data, location data and scheduling orders. On the other hand, services are well suited to handle request-reply interactions like navigation and transfer of maps. It is also possible to use ROS' ActionServers to perform time-sensitive operations like teleoperation, where pre-empting of orders might be necessary.

The packages designed during the project all followed the basic principles of service oriented architecture, i.e. a service provider, a client and a central registry, since ROS in itself is designed along these principles.

The project also found that it is possible to streamline the implementation of Aria and ArNetworking libraries in ROS, so as to create a common set of packages. By continuing development along the guidelines suggested, it is possible to create a package with modularity. Since ROS does not limit the number of nodes that can exist on the same network, it is possible to scale up such a network to include as many AGVs as required.

6.1 Future Work and Research

The development of the project is only limited by the number of functions in Aria. As more development occurs in the Aria and ArNetworking libraries, the same developments can be implemented in ROS. Some additions that can be considered would be path-planning, scheduling, visualisation of the AGVs on the map using ROS and creation of an ArNetworking server using the ROS framework.

It may also be a necessity to have AGVs from different manufacturers running different libraries on the same network. In such a case, a common ROS master server could be implemented to handle all the AGVs and their respective functions.

6.2 Critical Discussion

The project especially highlighted the need for a method of controlling and implementing functions in AGVs. While each manufacturer provides specific functionality coupled with its own unique libraries, the provision of having a common platform to implement all functions in any AGV would definitely change the face of the mobile robotics field, in the author's view.

6.3 Generalization of the result

The versatility of ROS was witnessed during the project. Since ROS allows for implementation of external libraries in a variety of languages, it isn't limited by the make or model of an AGV. The implementation of parts of the Aria library in the duration of the project could be seen as proof of this.

7 References

[1] T. Le-Anh and M. D. Koster, "A review of design and control of automated guided vehicle systems," *European Journal of Operational Research,* vol. 171, pp. 1-26, 2006.

[2] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research,* vol. 170, pp. 677-709, 2006.

[3] H. Liu, N. Stoll, S. Junginger and K. Thurow, "A Common Wireless Remote Control System for mobile robots in laboratory," Crown, 2012.

[4] E.P.Boje and B.J.Kotze, "An integrated control system for Automatic Guided Vehicles (AGV)".

[5] C. Chen, Q. Gao, Z. Song, O. Liping and X. Wu, "Catering Service Robot," in *8th World Congress on Intelligent Control and Automation*, Jinan, China, 2010.

[6] P. Farahvash and T. O. Boucher, "A multi-agent architecture for control of AGV systems," *Robotics and Computer-Integrated Manufacturing,* vol. 20, pp. 473-483, 2004.

[7] D. Weyns, T. Holvoet, K. Schelfthout and J. Wielemans, "Decentralized Control of Automatic Guided Vehicles - Applying Multi-Agent Systems in Practice," in *OOPSLA*, Nashville, Tennessee, USA, 2008.

[8] C. Wang, H. Ghenniwa and W. Shen, "Real time distributed shop floor scheduling using," *International Jurnal of Production Research,* vol. 46, no. 9, pp. 2434-2452, May 2008.

[9] J. Sacha, B. Biskupski, D. Dahlem, R. Cunningham, R. Meier, J. Dowling and M. Haahr, "Decentralising a service-oriented architecture," Springerlink.com, 2009.

[10] "ROS Documentation," Open Source RObotics Foundation, [Online]. Available: wiki.ros.org. [Accessed 22 05 2016].

[11] A. Dietrich, S. Zug and J. Kaiser, "The R in Robotics," *The R Journal,* vol. 5, no. 2, pp. 117-128, 2013.

[12] A. MobileRobots, "ARIA - MobileRobots research and academic customer support," 10 02 2016. [Online]. Available: http://robots.mobilerobots.com/wiki/ARIA. [Accessed 14 06 2016].

[13] A. MobileRobots, "Mapper3 - MobileRobots research and academic customer supprt," 3 Augusst 2015. [Online]. Available: http://robots.mobilerobots.com/wiki/Mapper3. [Accessed 14 06 2016].

[14] M. Bennulf, "Mobile robot navigation and control : Applied on Patrolbot," Hogskolan Vast - DIVA, Trollhattan, Sweden, 2015.

[15] S. Jurić-Kavelj, "ROSARIA - ROS wiki," 10 04 2011. [Online]. Available: http://wiki.ros.org/ROSARIA. [Accessed 10 06 2016].