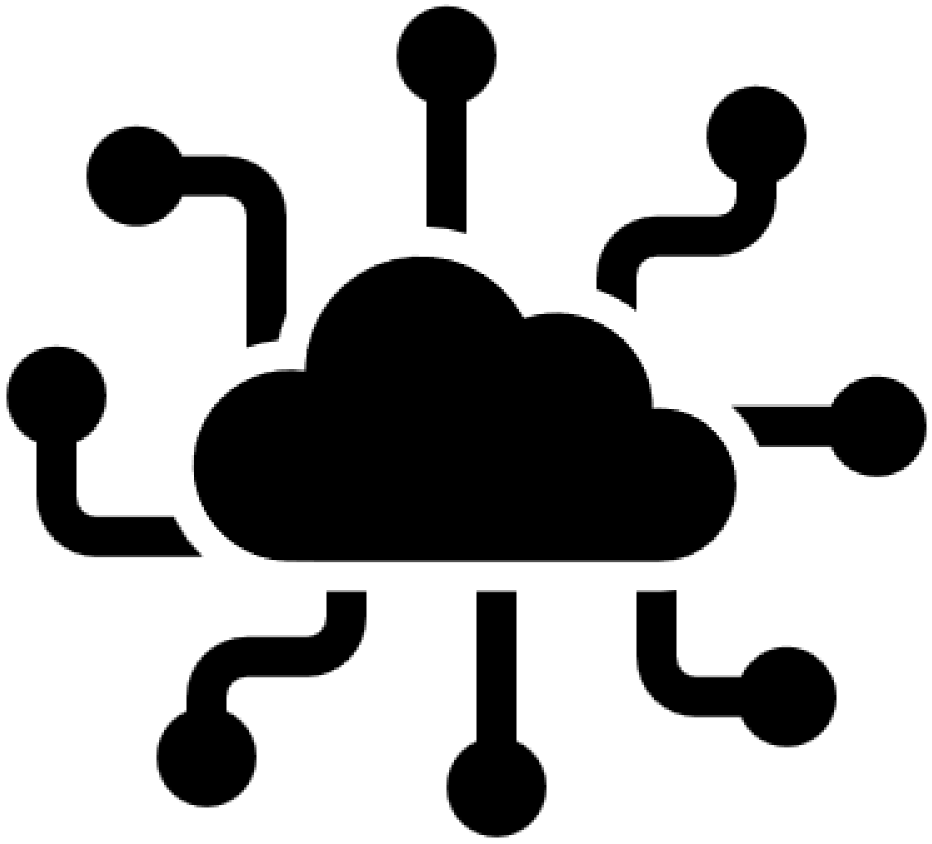
City of Things prototyping kit

HOGESCHOOL ROTTERDAM - PROJECT 7/8

RESEARCH – WHAT IS ROS?



Students

A. Blankwaard	0966307	TI2C

Teachers	**Product Owner**
W.M. Tiest	T. Jaskiewicz
A.M. de Gier	I. Smit

Table of Contents

Contents

Abstract

To operate a food delivery robot there must be a framework applied which can provide: driving, mapping, localization and obstacle avoidance. This framework should also accommodate the communication between the different parts of the robot.

Introduction

The assignment for this project is to create a prototype robot that can be put in to use in the society to get people's opinions about the robot and to see what kind of effect this robot would have on people. As developers we decided to build a robot that can deliver food in restaurants autonomously. This means the robot should be able to drive, map and navigate through a room. To achieve this goal the decision has been made to use a hoverboard which is inexpensive and well available in the second-hand market. This hoverboard will be flashed with a custom firmware to enable the robot to drive this hoverboard. Another decision that has been made is to use a depth camera in combination with SLAM (Simultaneous Localization And Mapping) to determine the position of the robot, to map a room and to avoid any obstacles.

There is also a requirement given that the robot should be scalable meaning it should be possible to expand to multiple robots which can communicate with each other.

To achieve these goals there must be research done to find a framework which can provide the necessary tools. In this research we will look for the best framework to apply on the robot by answering the main question; which framework is the to use on the Hackerboard? Some other question that will need to be answered are:

- Does the framework provide the necessary communication tools needed to provide the Hackerboard with the communication needed between its parts?
- Is the framework able to control the hacked hoverboard?
- Can SLAM data be communicated through the framework?

Theoretical Framework

Methodology

The information gathered in this research will be based on literature found on the internet. This will include scientific articles, websites and news articles. The goal of this research will be to produce a recommendation of which framework will be the best to implement the different parts needed on the delivery robot. This will include driving the hoverboard, localization, mapping and navigation. To make the delivery robot work every part of it needs to be able to access the data which has been gathered by sensors. To facilitate this the framework needs to be able to send a lot of data, send and receive data simultaneously.

Middleware

The Hackerboard is a robot in essence, while there are a lot of robots almost all of them are created to serve a different purpose. To achieve these purposes a lot of different parts will be needed in a robot. Generally you would need sensors, data processing and actuators. All these parts of a robot either consume or produce data through an application made for the specific part. The only question is how to get the data from one application to another?

A solution for this issue could be to create a single application which could be used for all the parts in a robot, but this can get very complicated over time and it would get more difficult to add, modify or remove a part of the robot without breaking the entire program.

This is something that middleware can solve, it facilitates the ability to transfer data from one application to another. To ensure the Hackerboard will stay hackable and scalable we decided we need to use some form of middleware. While there is a lot of middleware available some of it is specifically designed for robots. To find out which middleware would be the best suited for our Hackerboard we will look more into and compare the following in this research: ROS, YARP, Urbi and Rock.

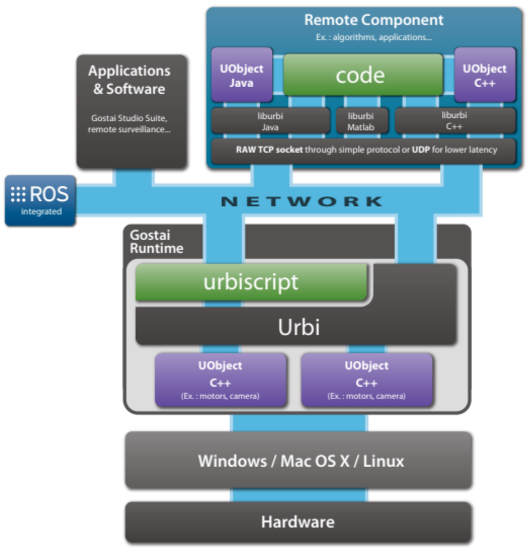
Urbi

Urbi is a bit different from most other middleware options, it consists of Urbi Runtime which is the core of the Urbi system and is used to interface Urbi with the OS. There are UObjects which are used to bind hardware or software components, such as actuators and sensor on the one hand, and voice synthesis or face recognition on the other hand. And lastly it includes urbiscript which can be used to orchestrate all the components.

Urbiscript has been specifically designed to handle concurrent programming. It is object oriented and can be integrated with a lot of other languages such as C++. It also supports and emphasizes parallel and event-based programming, which are very useful in robotics.

Urbi can be used on the major operating systems Microsoft Windows, Linux and Apple macOS. But can also be used on some real-time operating systems like Xenomai, and on specific operating systems such as Aperios.

Overall Urbi seems to be really time focused to facilitate time sensitive applications such as robotics operating at a high speed. This is nice to have but due to the speed our Hackerboard will need to operate at not quite critical. Some other positive aspects of Urbi include the UObjects using C++ which make it easy to interface with a lot of different sensors, actuators and software components. And the comprehensive documentation available for Urbi itself, UObjects and urbiscript. A Downside seems to be the community which doesn’t appear to consist of a great amount of people using Urbi and another downside could be the fact that it hasn’t been updated in a long time although this could also suggest a very stable framework.[9][11]


Remote Component
Ex. : algorithms, applications...
UObject Java
code
UObject C++
liburbi Java
liburbi Matlab
liburbi C++
RAW TCP socket through simple protocol or UDP for lower latency
Applications & Software
Gostai Studio Suite, remote surveillance...
ROS integrated
NETWORK
Gostai Runtime
urbiscript
Urbi
UObject C++ (Ex. : motors, camera)
UObject C++ (Ex. : motors, camera)
Windows / Mac OS X / Linux
Hardware

[9]

ROS

ROS or Robot Operating System is like the other options not an operating system but is an SDK (software development kit). Despite it including middleware functionality it isn't only middleware it includes a lot of other functionality. The middleware part of ROS or as they like to refer to it as plumbing provides the communication needed between all components which are referred to as nodes in the ROS ecosystem. The middleware is based on an anonymous publish/subscribe pattern. This ensures the ability to gather and publish data simultaneously and asynchronous. It also enables nodes to only use the data the need without filtering any unnecessary data traffic.

Apart from the middleware functionality ROS delivers a much bigger package with everything from drivers to algorithms, to user interfaces and a lot of drivers for specific devices which a lot of the other options do not provide. ROS also comes with good developer tools such as launch, introspection, debugging, visualization, plotting, logging and playback. All of this makes ROS a very complete with almost everything needed to create robotic solutions. And everything that isn't included might be found in the ROS community which has a huge following with active members creating nodes for a lot of devices.

ROS is compatible with Microsoft Windows, Linux, Apple macOS and even on some embedded platforms.

Rock

Rock could be another option to serve as middleware for the Hackerboard. Its model is based on the Orocos RTT (Real Time Toolkit). The Orocos RTT has been designed to create framework targeting the implementation of (realtime and non-realtime) control systems, it claims to be highly configurable and interactive component-based which is quite applicable to a robot due to it usually being component based and it's realtime operation. Rock has been built on top of the Orocos RTT so it can be a high-performance and reliable framework to function as middleware in a lot of robotic applications.

Rock like many other middleware frameworks is written in C++ but can also be used with Ruby.

It isn’t supported on all of the operating systems but is known to work on ubuntu LTS releases, other Linux distributions have been tested a long time ago, so they aren’t sure to be compatible. Microsoft Windows isn’t mentioned and MacOS X is known to have problems with running Rock.

A lot of packages and documentation are available with Rock these could provide a great experience in developing with Rock and make it easier to implement Rock with some of the components needed in the Hackerboard.

According to Rock itself it has been specifically developed to address the following issues in existing solutions.

Sustainable systems. The whole architecture and all the tools in Rock are designed with long-living systems in mind. This means there is a lot of functionality on error detection, reporting and handling. Which for the purpose of the Hackerboard wouldn’t especially be necessary at this point but could come in handy in a later stage of the project.

Scalability. Which means providing the tools to be able to manage big systems with a minimum fuss. This could be useful in de development of the Hackerboard due to the flexibility it should provide to keep it hackable.

Reusable codebase. The drivers, localization algorithms and control loops are totally inpendent from Rock’s integration framework. Which provides developers to only use these parts of Rock without needing to integrate a whole Rock framework. We don’t see much use for this for the Hackerboard since we are looking for some type of middleware that will handle almost all the tasks that need to be done, from driving the components to localization, mapping and visualization.[10][12]

YARP

YARP which stands for (Yet Another Robot Platform), describes itself as being “a set of libraries, protocols, and tools to keep modules and devices cleanly decoupled.”. And was initially released in the year of 2002, while that is a long time ago it still gets updated frequently with the latest version update getting released in May of 2022.

It is written by researchers in robotics and made for humanoid robotics particularly, which could result in less optimal use for the Hackerboard which doesn’t quite specify as a humanoid robot. An upside to YARP is the focus on loose coupling which facilitates an easily modifiable environment to add and modify different kind of sensors and actuators without the need to modify the existing system.

Being almost fully written in C++ YARP can operate on a lot of operating systems including Linux, Microsoft Windows, Apple macOS and iOS, Solaris and Android.

Although having a big community using YARP there doesn’t seem to be a lot of existing libraries for specific devices but there are a lot of YARP libraries which cover a lot of general category devices.

YARP is written in C++ but can also be used with other programming languages by using SWIG.

YARP consists of three main components.

YARP_os which interfaces with the operating system.

YARP_dev which can be used to communicate with a lot of different devices like cameras and motor control boards.

YARP_sig which will perform common signal processing tasks. [13]

What is ROS?

ROS is short for Robot Operating System, it is an open source meta-operating system for robots. While having operating system in its name it is not quite an operating system, ROS currently runs on Unix based platforms which would make it possible to run it on a raspberry pi for example. ROS is more a SDK (Software Development Kit) which provides tools for actuating, logging, visualization, introspection and plotting.

Because ROS is open source, there is an active community with it. The community has provided a lot of libraries which makes it easy to extend ROS with numerous devices, data-sources, navigation algorithms and more.

There are some alternatives to ROS which we will look at in short but due to the community driven widely accepted ROS will probably be less compatible for our project.

The core function of ROS is to handle communication between programs to be so called middleware. ROS uses packages and nodes programs and makes sure those nodes are able to communicate.

How can ROS be used in the Hackerboard?

To find out how ROS can be used in the Hackerboard we first need to define what parts are needed in the Hackerboard.

The first part of the Hackerboard is the actuator which in this case is a hoverboard that has been flashed with custom firmware so it can be controlled through UART. After some research on the internet a ROS package can be found that has specifically been developed for use with this hoverboard.

The second part of the Hackerboard is the Intel Realsense D415 depth camera. This camera

Another part of the Hackerboard will be SLAM, which will be done by using an Intel Realsense Depth camera. This camera uses stereoscopic images to produce the produce depth information which can be used in the Hackerboard as a sensor. Another ROS package is available to interface with the Intel Realsense D415 and send its data over the ROS server. [6]

After sending the information from the Intel Realsense depth camera over the ROS server it needs to be processed and used in a SLAM algorithm. SLAM can be done with a variety of sensors and there are a few ROS packages available to perform SLAM. Due to the usage of a depth camera rtabmap_ros will be the best ROS package to perform SLAM with this camera. It has been made for usage with depth camera's including multiple Asus camera's, Xbox Kinect camera's and Intel Realsense camera's including the Intel Realsense D415. [7]

Now that the Hackerboard will be able to perform SLAM, it also must be able to create a map of a room to finally use this map for navigation. Rtabmap_ros also publishes multiple maps to the ROS server though not all the maps can be visualized in Rtabmap_ros. Therefor a tool called Rviz, which is also included with ROS can be used to visualize these maps and other data like: odometry, paths, waypoints, point clouds, navigation targets and camera images. [8]

Lastly ROS has an open-source Android application which is designed for dynamic control and visualization of ROS topics. This application can be used to create a user interface which is easy to understand and can provide the user with all the needed data on his or her android device. [9]

ROS compared to ROS2

The biggest change in the upgrade from ROS1 to ROS2 is the use of DDS in ROS2. DDS or Data Distribution Service is an existing middleware solution. Where ROS1 used ROS its own custom transport protocol TCPROS which uses TCP/IP sockets for transporting message data. ROS2 uses DDS which uses UDP as transport protocol, this enables the developer to configure the level of reliability expected by a node.

One of the upsides of using DDS over ROS's own custom protocol is the ability to assign Quality of Service this enables the developer to configure resource consumption, fault tolerance and communication reliability.

Another big change is ROS2 targeting Python3 instead of Python2 which means more recently developed libraries become available to use with ROS2.

In ROS1 any created codebase could be compiled using catkin. Catkin gets replaced by ament in ROS2 the ROS developers replaced catkin for a few reasons one of them being the devel space which provides a fully working ROS environment without the need to install any packages. This is a great feature, but the complexity of catkin increases significantly and it increased the workload of every ROS developer by the need to handle the devel space in the CMake file of a ROS package.

It is also good to mention the backwards compatibility of ROS2, because ROS1 has been in use for over 10 years it is nice to know that ROS1 servers can still communicate with a ROS2 server by using the bridge communication package. There is also the option of converting ROS1 packages to ROS2 packages, this can be done by changing some libraries in the code, modifying the package.xml file and the CMakeLists.txt file. After these files are modified accordingly the package must be built using ament instead of the deprecated catkin build tool.[14] [15] [16]

Conclusion

In conclusion the ROS framework provides the most complete package to work with, Urbi, Rock and YARP just not having the same driver availability, included algorithms, large community, large package library or the ease of scalability. Especially due to the amount of information and help that can be found on the internet about ROS it is nice to work with and parts of it that might be vague are easily clarified with a quick search on the internet.

Recommendation

After comparing some different middleware options to use, ROS seems to be the best one in this specific case. This is due to the availability and variety of device drivers, algorithms and community support. Thus, the recommendation is to use ROS on the Hackerboard.

Abbreviation’s

Abbreviation	Explanation
	Robot Operating System
SLAM	Simultaneous Localization And Mapping
ROS	Robot Operating System

Glossary

Term	Explanation
Hackerboard	The hacked hoverboard on which our project is based on.
SLAM	SLAM is the computation problem of updating a map of an unknown environment while simultaneously keeping track of an agent’s location within in
ROS	Robot Operating System

Changelog

Version	Date	Change	Author
1.0	18/03/2022	Added introduction	Auke Blankwaard
1.1	23/03/2022	Added ROS description	Auke Blankwaard
1.2	20/04/2022	Wrote methodology	Auke Blankwaard
1.3	05/06/2022	Added YARP, Rock and Urbi	Auke Blankwaard
1.4	06/06/2022	Changed ROS, added middleware description	Auke Blankwaard

1.5	12/06/2022	Added list of references, finishing changes	Auke Blankwaard

References

[1] Fitzpatrick, P., Ceseracciu, E., Domenichelli, D. E., Paikan, A., Metta, G., & Natale, L. (2014). A middle way for robotics middleware. *Journal of Software Engineering for Robotics*. https://lornat75.github.io/papers/2014/fitzpatrick-joser.pdf

[2] Elkady, A., & Sobh, T. (2012). Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *Journal of Robotics*, *2012*, 1–15. https://downloads.hindawi.com/journals/jr/2012/959013.pdf

[3] Gazis, A., & Katsiri, E. (2022, May 15). *Middleware 101 What to know now and for the future*. Acmqueue. Retrieved June 6, 2022, from https://queue.acm.org/detail.cfm?id=3526211&doi=10.1145%2F3526211

[4] *Executive Summary*. (2019, April 18). SWIG. Retrieved June 7, 2022, from https://www.swig.org/exec.html

[5] *realsense2_camera - ROS Wiki*. (2021, February 12). ROS Wiki. Retrieved June 8, 2022, from http://wiki.ros.org/realsense2_camera

[6] *ROS-Mobile - ROS Wiki*. (2020, June 22). ROS Wiki. Retrieved June 8, 2022, from http://wiki.ros.org/ROS-Mobile

[7] *Rtabmap_ros*. (2021, September 13). ROS Wiki. Retrieved June 8, 2022, from http://wiki.ros.org/rtabmap_ros

[8] *rviz - ROS Wiki*. (2018, May 16). ROS Wiki. http://wiki.ros.org/rviz

[9] urbi. (2013). The Urbi Software Development Kit. *The Urbi Software Development Kit*, 1–8.

https://github.com/urbiforge/urbi/blob/master/doc/urbi-sdk.pdf

[10] *The Orocos Real-Time Toolkit | The Orocos Project*. (n.d.). Orocos. Retrieved June 5, 2022, from

https://www.orocos.org/rtt/

[11] Wong, W. G. (2010, August 5). Gostai Urbi. Electronicdesign.Com. Retrieved June 5, 2022, from

https://www.electronicdesign.com/technologies/embedded-revolution/article/21791682/gostai-urbi

[12] What is Rock? (n.d.). Rock-Robotics.Com. Retrieved June 5, 2022, from https://www.rock-robotics.org/documentation/about/index.html

[13] YARP. (n.d.). YARP: What exactly is YARP? Yarp.It. Retrieved June 5, 2022, from

https://www.yarp.it/latest/what_is_yarp.html

[14] Thomas, D. (2015, September). Changes between ROS 1 and ROS 2. Design.Ros2.Org. Retrieved June 6, 2022, from http://design.ros2.org/articles/changes.html

[15] Thomas, D. (2014, August). ROS 2 middleware interface. Design.Ros2.Org. Retrieved June 6, 2022, from http://design.ros2.org/articles/ros_middleware_interface.html

[16] Thomas, D. (2015a, July). The build system "ament_cmake" and the meta build tool "ament_tools." Design.Ros2.Org. https://design.ros2.org/articles/ament.html