# THE PATH FROM RESEARCH TO INDUSTRY

**Robotics research groups around the world are using Robot Operating System (ROS) to develop their prototypes quickly. While the first version of ROS was aimed primarily at the R&D community, its successor, ROS 2, has been redesigned completely to be industrial grade and applicable in research, prototyping, deployment and production. This allows ROS 2 prototypes to evolve into products suitable for real-world applications. To explore the state of the art, Saxion University of Applied Sciences and nine companies are developing an industrial mobile robot. This article describes experiences from the development process and presents an outlook on the potential of ROS 2 for industry.**

WILCO BONESTROO

The plan to explore ROS 2 for industrial mobile robots was born during a meeting on open issues in robotics. This discussion resulted in the Next Generation Navigation, or NeNa, research project, a collaboration by Saxion with the companies Demcon, Hencon, Hollander Techniek, Indes, Opteq Mechatronics, Riwo, Romias, Singa and Wewo. The project is funded partly by SIA, the Dutch Taskforce for Applied Research (*Nationaal Regieorgaan Praktijkgericht Onderzoek*).

During that first meeting, several companies explained how they developed their navigation functionality for their robots or AGVs (automatic guided vehicles) such as those shown in Figure 1. Most of the attending companies used commercial, closed-source solutions for navigation and fleet management, such as Navitec Systems and BlueBotics. Other companies had built their software from scratch. Both approaches have their drawbacks: the first creates a dependency on an external party, while the second requires considerable software development effort for maintenance and further development.

The Mechatronics research group at Saxion had been working with smaller robots for both research and education, such as those shown in Figure 2. Those robots use the open-source navigation solution provided by ROS. We wanted to explore whether that solution was also applicable to the larger industrial robots developed by the industrial partners, so we started the NeNa research project in 2019. Our main question was whether we could build a robust and accurate navigation solution based on ROS that would provide a similar functionality and quality as the current approaches.

## Requirements

The companies in the project apply their robots in different fields, ranging from mining and heavy industries to logistics and farming. As it was impossible to select one use case that could address all the application fields, we decided to develop a new use case that could demonstrate the feasibility of our requirements. We needed a prototype that could demonstrate:

**AUTHOR'S NOTE**

Wilco Bonestroo is a researcher in the Mechatronics research group of Saxion University of Applied Sciences, located in Enschede (NL). He focuses on autonomous systems and AI with a special interest in navigation and localisation.

w.j.bonestroo@saxion.nl
www.saxion.nl/onderzoek/smart-industry/mechatronica

1a

1b

*Examples of mobile robots from project partners.*
*(a) Hencon.*
*(b) Wewo.*

2
Lightweight research and education robots.

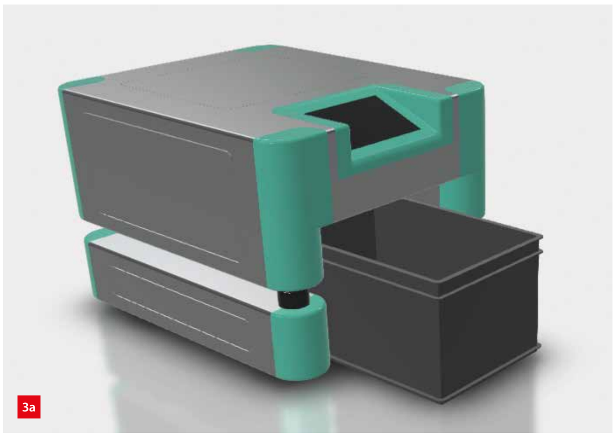
- autonomous navigation, including dynamic obstacle avoidance, no-go areas, speed zones and virtual line following;
- localisation without installed external infrastructure, such as reflectors or induction wires;
- integration with existing fleet manager software;
- precision docking for specific tasks.

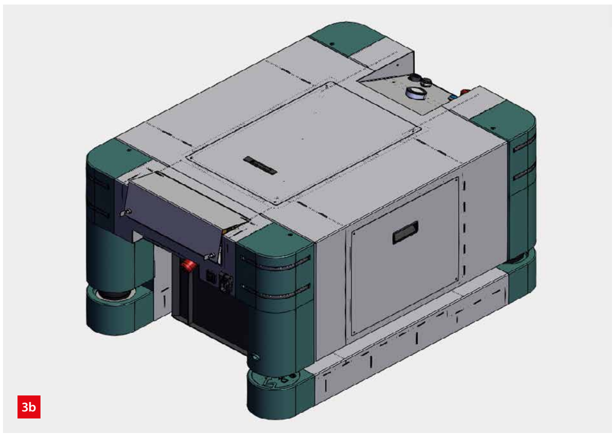
Our use case was inspired by straddle carriers, which carry their load underneath instead of putting it on top. We wanted to demonstrate solutions on industrial equipment, but we also wanted to use the prototype in our research labs and offices. Therefore, a small industrial robot was designed and developed. The robot is a tunnel vehicle that can drive over euro crates, pick them up and move them around. Figure 3 shows impressions of the concept and design. To make our findings relevant for the bigger robots, we provided our NeNa robot with the components that are also used in the larger machines developed by the companies. It has two 2D Sick lidar sensors to provide a 360° view of the environment. The lidars are used for safety, but also for localisation and obstacle avoidance. The robot is also equipped with a 3D Intel® RealSense™ camera to detect the crates. The partners in the project were not only interested in navigation functionality, but also in the ability of a robot (or fleet of robots) running ROS to be integrated into their automation ecosystem, as the robots had to be controlled from a fleet manager. The navigation behaviour also had to be adjustable by the end user. For example, in some areas such as hallways, the robot should follow a virtual line, while in other areas, such as large rooms or production halls, the robot should navigate freely. In addition, some areas have speed limits, or the speed of the robot should be adjusted based on whether there are people around it.

## Parallel development

To manage the lead time of our project, we designed and implemented our hardware and software in parallel. So, while the first CAD drawings were being created, the simulation model of the robot was developed in ROS. This model had all the sensors that we planned to use in our robot. In Gazebo, the default simulator in ROS, the virtual robot could be tested in different environments. For example, we could work on a precision docking manoeuvre based on simulated sensor data while the actual robot was still under development. Figure 4 shows the robot in a simulated office environment as well as a simulation of robot-crate interaction.

To use both the industrial hardware and the advanced navigation algorithms from Navigation 2, we orchestrated computation over two different computers: an industrial PLC and a general-purpose PC. The PC runs Linux with ROS and executes the navigation algorithms that perform complex tasks like localisation and path planning. The PLC provides strict timing and handles the safety functionality of the robot. It takes care of low-level communication with the sensors and motors. Moreover, it checks whether the PC is operating as expected. For the communication between the PLC and the PC we have experimented with OPC UA (Open Platform Communications – Unified Architecture). We did some initial tests and concluded that this provides


3a


3b

The NeNa robot prototype (with crate).
(a) Artist impression.
(b) CAD design.

## ROS, ROS 2 and ROS Industrial

ROS development started around 2007, at Stanford University and in the company Willow Garage [1]. The first official ROS release was in 2010. One of the main ROS goals was to stimulate collaboration between developers by providing standard communication interfaces. Although developed originally for one specific humanoid robot, more and more people started contributing to the project and ROS was applied to many different robots. Using ROS allowed research groups to focus on their own specific research topics and simply use the functionality provided by others.

Up to now, the ROS community has released 13 distributions. The first letter of the distribution name indicates their order. The most recent ones are Kinetic, Lunar, Melodic and Noetic. The distributions alternate between five-year support (long-term support or LTS) and two-year support. Support means that they are actively maintained and updated by the community. Each distribution is supported on exactly one Ubuntu Linux distribution. Canonical, the company behind Ubuntu, is one of the 17 companies in the ROS technical steering committee. Their main focus is on the security of robotic systems running Ubuntu and ROS.

Within a distribution, software is organised and delivered in ROS packages. Today, there are hundreds of them. Some packages provide drivers for common robotics hardware, such as lidars, inertial sensors and cameras. In addition, there are advanced packages for navigation, localisation, path planning and robot manipulation, as well as ROS packages that provide wrappers for common libraries, such as OpenCV for image manipulation, the Point Cloud Library (PCL) for handling 3D point cloud data from cameras and lidars, and YOLO or TensorFlow for object detection and localisation. They are installed easily with the Ubuntu package manager.

Over the past decade, ROS has become the standard framework in the academic world and in research environments. However, because ROS was not designed for production environments, it was quite a challenge to go from prototype to software that could be deployed industrially. Companies would have to review all the software used in their product. Based on requirements from industry combined with insights gained from working with ROS, there was a discussion on whether those requirements could be integrated into the existing framework, or a redesign was needed. Around 2015 it was decided that it would be better to design a new ROS version from the ground up to provide security, reliability and real-time capabilities. It should also run on many platforms, not only on (Ubuntu) Linux, Mac and Windows, but also on embedded and real-time systems. This was to become ROS 2.

A remarkable difference between ROS and ROS 2 is the middleware that is used to communicate between components, or – in ROS terms – between 'nodes'. The middleware in ROS was developed from scratch. In parallel, however, in the past decade a number of middleware solutions have matured, such as ZeroMQ, Protocol Buffers and Data Distribution Service for real-time systems (DDS). After analysing these existing solutions, DDS was selected as the middleware for ROS 2. DDS is a proven standard used typically in mission-critical systems, such as in military, aerospace and industrial automation. In ROS 2 there is no longer a single master node that controls the whole system. Nodes discover each other automatically using DDS and the system is truly distributed.

ROS 2 is also released in distributions. Again, the first letter indicates the order: Dashing, Eloquent, Foxy. As ROS 2 is still under heavy development, the support is shorter than for ROS. Foxy is considered an LTS version and has come with three-year support. ROS 2 is expected to be as stable as ROS within a year.

To complicate things a little more, there is also an initiative called ROS Industrial. Its goal is to bring the advantages of ROS, such as reuse of existing software, to industrial robots. However, ROS Industrial is not another version of ROS. It builds on the existing ROS core and provides packages that are aimed specifically at industry, such as deterministic path planners for manipulators and drivers for robot arms. Manufacturers of robot arms are actively encouraged to develop and support their own drivers in the ROS Industrial initiative. As code quality and reliability are essential for industrial applications, each ROS Industrial package has a status description that indicates whether the package is experimental, developmental or production ready. Although ROS Industrial is open source, it also provides commercial services, such as training, development and support. As ROS 2 targets industry, ROS Industrial is pushing its community towards ROS 2.

Navigation has always been a major part of ROS and is used on many existing service robots. The navigation functionality of ROS was also redesigned for ROS 2, resulting in the Navigation 2 package [2]. Navigation 2 is aimed at dynamic environments and supports a wider range of sensor to be used in mapping, localisation and navigation. It also supports contextual navigation behaviour. This means that the navigation can be adapted based on the area where the robot is driving or what the sensors are seeing.

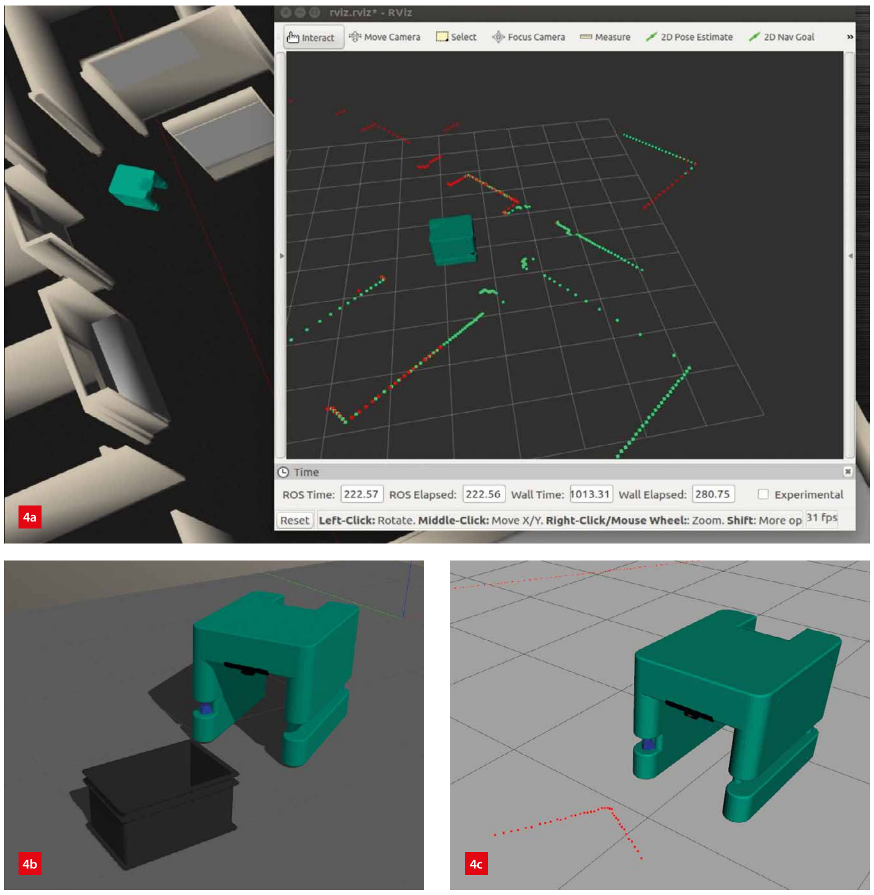
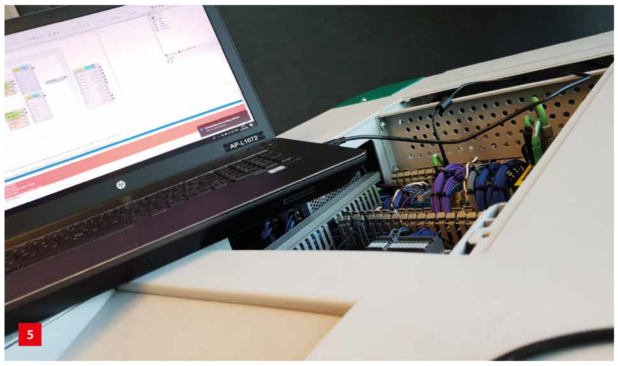


*Figure 4. Simulation of the NeNa robot and a crate in an office environment.*
*(a) Robot model in the office, with visualisation of sensor data in the right screen.*
*(b) Robot-crate interaction.*
*(c) Visualisation of sensor data: how the robot sensors actually 'see' the crate.*

a modular approach, because OPC UA is available on both the PC and PLC. The goal is to be able to replace the PLC or the PC with any other system that also supports OPC UA. We are now in the final phase of the project. The hardware has been delivered, the drivers for the hardware are currently being developed and tested, and the high-level software has been partly demonstrated in separate simulations while new features are still under development. When all the hardware has been completed (Figure 5), we will start integration and system tests to determine whether we can meet all the requirements. In the final phase of the project, everything has to be integrated into two robots performing their tasks based on a fleet manager. We aim to complete the project in April 2021.



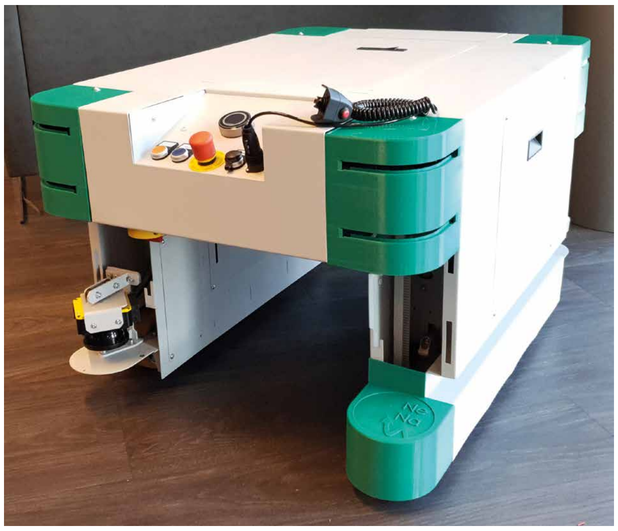
## Which ROS distribution to choose?

We have spent quite some time developing our software for different ROS distributions. When we started in 2019, navigation in ROS 2 was unstable. As the involved companies were still mainly interested in ROS 2 and because ROS and ROS 2 can be combined in one system, we decided to develop our first robot model and simulation of the robot both in ROS and ROS 2. During the project, the recommended distributions in ROS progressed. Moreover, newer distributions of ROS 2 became more stable and provided functionality we needed in the project. Eventually, we have developed our robot models and software in five distributions, and this required a lot of effort.

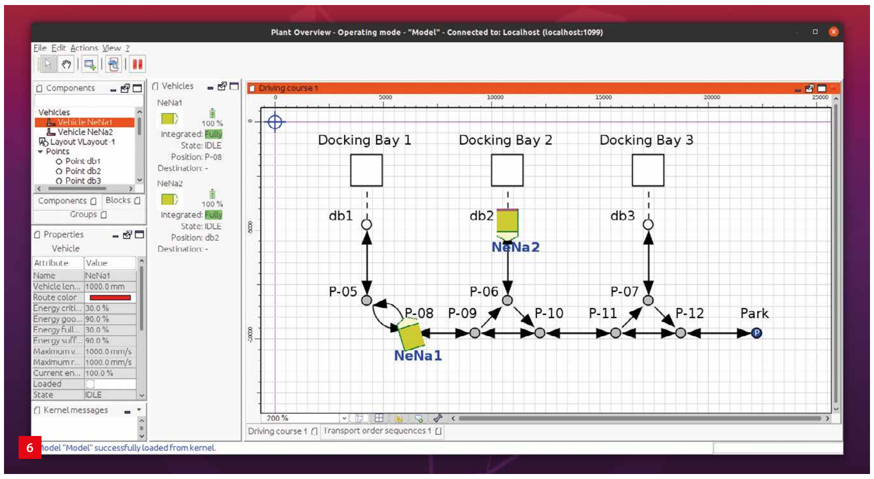
In hindsight, we should have been more careful in selecting the distribution(s). For companies, such choices are even more important. Currently, ROS is more stable than ROS 2 and features more packages; support for ROS, however, will end in 2025. Meanwhile, robot and application developers are already moving their focus towards ROS 2 and the ecosystem is growing. We also experienced that navigation in ROS 2 is quite advanced compared to other parts. For developing a functional prototype quickly, ROS is still the logical choice, because it is stable and there are many packages to build on. When aiming to actually use the software in a product, ROS 2 would be the logical choice. To summarise: "If you want to go fast, go ROS. If you want to go far, go ROS 2."

## Integration in the industrial ecosystem

In a research setting it is acceptable to control a robot manually or even by typing commands from a terminal. However, mobile robots or AGVs in production environments have their own place in the automation ecosystem. One of the specific requirements in our project



*Realisation of the NeNa robot, in two views.*



NeNa1
100 %
Integrated: Fully
State: IDLE
Position: P-08
Destination: -

NeNa2
100 %
Integrated: Fully
State: IDLE
Position: db2
Destination: -

| Attribute | Value |
|---|---|
| Name | NeNa1 |
| Vehicle len... | 1000.0 mm |
| Route color | 🟥 |
| Energy criti... | 30.0 % |
| Energy goo... | 90.0 % |
| Energy full... | 30.0 % |
| Energy suff... | 90.0 % |
| Maximum v... | 1000.0 mm/s |
| Maximum r... | 1000.0 mm/s |
| Current en... | 100.0 % |
| Loaded | ☐ |
| State | IDLE |

Model "Model" successfully loaded from kernel.

6

*Screenshot of a factory model in OpenTCS.*

was that the robots should be controlled from a fleet manager. The fleet manager translates tasks from ERP or MES systems into navigation tasks, then dispatches those tasks to the appropriate robots and keeps track of the whole fleet. To demonstrate interoperability between ROS and fleet managers, we have integrated our robot with OpenTCS, an open-source fleet manager developed by Fraunhofer IML. One of the project partners was already using OpenTCS in their systems. Figure 6 shows a screenshot of a factory model with several docking bays and two NeNa robots.

To provide an interface between the fleet manager and the robot, OpenTCS uses 'vehicle drivers' to send commands to specific robots. We have developed a ROS 2-OpenTCS vehicle driver, which makes all functionality from the fleet manager available in ROS 2. Developing a driver is conceptually straightforward, but in Java this turned out to be a challenge. ROS 2 can be used with any programming language, with support for different languages being provided by client libraries that translate between the specific language and the generic ROS functionality. The ROS client libraries for C++ (rclcpp) and Python (rclpy) are well developed and thoroughly tested. However, OpenTCS is written in Java and the client library for Java was under development at the start of the project. Eventually, developing the driver and integrating it with a client library 'under construction' took more effort than expected.

An interesting aspect of ROS 2 is that it can be deployed in different ways. By default, it uses the standard open-source core, but there are also companies providing commercial and certified versions. For example, Apex.AI has developed a version of ROS 2 for safety-critical applications in the automotive industry. Their software is real-time, reliable and deterministic. Moreover, it is certified according to the automotive functional safety standard ISO 26262. The communication middleware in ROS 2 can also be exchanged easily. The default implementation Fast RTPS is suitable to be used within ROS, but it does not implement the full DDS standard. However, it can be replaced with Eclipse's Cyclone DDS, which is open source and does implement the full standard. Moreover, when support is required, there are also commercial versions such as Adlink's Vortex OpenSplice providing DDS.

## Conclusions

We jumped on the ROS 2 train rather early. In only a year and a half, navigation in ROS 2 has developed from highly unstable to a complete functional navigation system. On the Navigation 2 website [3], the 'Getting Started' steps can now be followed to have the navigation in a simulation up and running within an hour. However, specific features, such as virtual line following, require additional development. ROS 2 and Navigation 2 are still under development and getting the most out of them requires a serious software development team to work on them.

For companies involved in robotics, developments in ROS are interesting to follow. Developers of sensors or other components can reach a large worldwide group of potential customers by providing a well-maintained package with ROS drivers. Companies such as Intel, Xsens, Sick, Universal Robots and ABB robots provide drivers for their products. For system integrators, ROS 2 is interesting because it allows prototypes combining existing components to be built quickly, while also developing these prototypes into production-ready code. In our opinion, ROS 2 is currently not yet stable enough to ship in products, but based on discussions within the community, we expect that this will be realised within a year. ROS 2 is already used by the robotics teams in companies such as Amazon, Bosch and Rover Robotics.

As robotics students around the world are using ROS in their projects, it is interesting for companies to tap into this knowledge. To keep up to date with the general developments, it is advised to follow the discussions and announcements on the ROS forum [4]. There are many ways to learn ROS, with online tutorials, books and videos available. However, we have experienced that learning ROS is challenging, because there are so many different topics (Linux, ROS concepts, packages, tools, tool chains, algorithms, etc.) that one would have to grasp at once to get started. To get up to speed, ROS Industrial training courses can be followed all over Europe. In the Netherlands, these training courses are organised by Fontys University of Applied Sciences, Delft University of Technology and Saxion University of Applied Sciences.

### REFERENCES

[1] Quigley, M., et al. , "ROS: an open-source Robot Operating System", *ICRA Workshop on Open Source Software*, Vol. 3 (2), p. 5, 2009.
[2] Macenski, S., et al., "The Marathon 2: A Navigation System", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
[3] *navigation.ros.org*
[4] *discourse.ros.org*