Modeling, Identification and Control, Vol. 38, No. 3, 2018, pp. 157–165, ISSN 1890–1328

Software Components of the Thorvald II Modular Robot

Lars Grimstad Pål Johan From

Faculty of Science and Technology, Norwegian University of Life Sciences, N-1432 Ås, Norway. *E-mail:* lars.grimstad@nmbu.no

Abstract

In this paper, we present the key software components of the Thorvald II mobile robotic platform. Thorvald II is a modular system developed by the authors for creating robots of arbitrary shapes and sizes, primarily for the agricultural domain. Several robots have been built and are currently operating on farms and universities at various locations in Europe. Robots may take many different forms, and may be configured for differential drive, Ackermann steering, all-wheel drive, all-wheel steering with any number of wheels etc. The software therefore needs to be configuration agnostic. In this paper we present an architecture that allows for simple setup of never-seen-before robot configurations. The presented software is organized in a collection of ROS packages, made available to the reader. These packages allow a user to create her or his own robot configurations and simulate these robots in Gazebo using a provided plugin. Although the presented packages were created to be used with Thorvald robots, they may also be useful for people who are looking to develop their own robot and are interested in testing various robot configurations in simulation before settling on a specific design. To create a robot, the user lists modules with key parameters in one single configuration file and gives this as an input to the robot at startup. Example configuration files are provided within the packages. In this paper, we discuss key aspects of the ROS packages and provide directions on where to find updated information on how to install and use these.

Keywords: modular robots, agricultural robots, mobile robots

1 Introduction

As robots are moving from structured to non-structured environments, new challenges emerge. This is especially true for mobile robots that need to be adapted to preexisting infrastructure originally intended for human workers. Humans are agile creatures highly capable of assessing and adapting to new environments. This is less true for robots. In many cases, even slight changes to a robot's environment may render the robot useless. Here, flexibility in the robot's mechanical and kinematic design becomes useful, as it allows the robot to be modified to accommodate variation in working environments. Agriculture is a domain where one-size-fits-all is not adequate. Farms tend to be different from one another, and it will therefore require robots with a wide variety of shapes and sizes to serve all production types found in agriculture. In this paper, we present software packages for a class of modular robots that can be assembled in environment-specific configurations. The main motivation of this work has been the agricultural domain, but the presented material can be used in a wide range of applications.

It is well established that food production needs to increase in efficiency and reduce its environmental impact to meet future needs brought around by increased population and increased demand for environmental

doi:10.4173/mic.2018.3.2

friendly production. Getting high-precision, energy-efficient robots into the fields and greenhouses may be part of the solution.

There is increasing academic and commercial effort aimed at getting robots into farms, and several new and interesting robots appear every year. One early robot by van Henten et al. (2002) was designed to harvest cucumber in greenhouses. This robot uses heating pipes found between plant rows as rails for locomotion. Two other early robots were the API robot (Bak and Jakobsen (2004)) and the BoniRob (Ruckelshausen and et.al (2009)), which both are robots with four-wheel drive and four-wheel steering for the open fields. Some agricultural robots are designed to be used for solving several tasks, like the AgBot II (Bawden et al. (2014)), which is a differential drive robotic carrier that can be equipped with various implements. Other robots are specialized in solving one type of task, like the Robotanist (Mueller-Sim et al. (2017)), a robot for high-throughput crop phenotyping. Various robots have also been developed for harvesting (Lehnert et al. (2017) Feng et al. (2018)) and pruning (Botterill et al. (2016)), while yet other robots have been developed for in-farm transportation, such as the Bin-Dog (Ye et al. (2017)), a robotic platform for bin management in orchards.

Common for most agricultural robots is that they are designed to work in one specific environment. Many robots are also intended to solve only one specific task in that environment. Mobile robots may vary greatly in design depending on where they are to operate and normally offer little to no options for customization.

One of several challenges faced by those who are looking to commercialize their robotic systems, is the great variation found in farm infrastructure. Although variation may be expected between farms that grow different crops, also farmers who grow the same crop may use different systems. One example is strawberry production. A farmer may grow strawberries in the open field, in polytunnels or in greenhouses. This represents three widely different environments for a robot. To add to the challenge, there may also be great variation within each of these three environments. One farmer growing strawberries in polytunnels may have crop growing in beds in the ground, another may have crop growing on tabletops mounted on poles, and yet another may have crop in trays suspended from the ceiling. Then there is also great variation between different varieties of strawberries, variation in row spacing, variation in tunnel design and so on.

Looking at the example above, one can argue that there should be added some flexibility in the design of robots to accommodate variation found in agricultural environments. This was the thinking behind the Thorvald II robotic system created by the authors.

Thorvald II robots are assembled from modules. These modules can be combined in various ways to create a wide range of robots. By using a modular design, robots can quickly be created or rebuilt for new tasks in new environments. Several robots have been assembled from these modules and are operating in a wide range of agricultural environments.

In this paper we present key components of the robot's software and an early release of packages for Robot Operating System (ROS) (Quigley and et.al (2009)) that can be used for simulating arbitrary Thorvald II robots in Gazebo (Koenig and Howard (2004)). This includes a package containing a plugin for Gazebo, a package for generating robot descriptions in Unified Robot Description Format (URDF), including tags for mass and inertia properties, packages for simulating the robot base, a package for teleoperation, a package with example launch files for starting simulations and more. When publishing velocity commands to a real or simulated robot, the robot will calculate the correct joint commands for the current robot configuration, and output velocity estimates based on real or simulated joint states.

Robots are generated from one single configuration file, and the user may copy one of the provided robot configurations or combine modules freely to create a robot to his or her own specifications. The ROS packages are therefore useful for people who are planning to create their own robot, even if this robot is not based on Thorvald modules, and are looking for a tool for testing various robot designs in simulation.

The paper is organized as follows: Section 2 gives an overview of the Thorvald II robotic system hardware and describes the presented ROS packages. Section 3 deals with velocity commands and odometry, and Section 4 describes our approach to simulating arbitrary Thorvald robots. Finally, we conclude the paper and provide information on how to install the presented ROS packages in Section 5.

2 The Thorvald II Modular Robot

Thorvald II is a system for creating custom robots, primarily for the agricultural domain. The system is based on a handful of modules that can be assembled in various ways to create a wide array of robots with different shapes and properties. Working in several projects on different farms, the authors saw the need for a robot that could easily be reconfigured to fit more than one farm environment. The Thorvald II system was therefore created. This modular approach to creating robots helps save both time and costs when creating new environment-specific robots. E.g. when a

Figure 1: A handful robots created from Thorvald II modules. The tall robot in the background on the right has a custom frame made from welded steel pipes.

(a) **(b)**

Figure 2: Small robots created with Thorvald II modules. (**a**) A differential drive robot for greenhouses. This robot has a simple custom frame made from sheet metal. (**b**) A one-wheel drive, one-wheel steering robot.

research project finishes, a robot that has been used in the project is easily rebuilt for new tasks, in a different crop or on a different farm.

Several robots have been constructed from Thorvald II modules, a handful of which can be seen in Figure 1 and 2. These robots have been operating in open fields, polytunnels and greenhouses, in various types of crop at different locations in Europe. Many of the robots go through frequent rebuilds, as they move from environment to environment.

2.1 Robot Hardware

When creating or rebuilding a Thorvald robot, modules are connected through simple mechanical and electrical interfaces. The robot's structural frame is in most cases made using aluminum pipes and clamps, with special clamps providing simple mechanical connections for attaching modules to the frame. As complexity is contained within modules it is also possible to create custom frames, for example by welding pipes or sheet metal together. This is done when creating robots with special frame requirements, like the tall robot in the background in Figure 1 and the low greenhouse robot depicted in Figure 2a.

Figure 3: The most important robot modules. (**A**) battery enclosure, (**B**) drive module, (**C**) steering module, (**D**) suspension module.

Mechanical and electrical aspects of the Thorvald II system are described in more detail in Grimstad and From (2017a) and Grimstad and From (2017b). Modules relevant for this paper are described in brief below. Some of the modules are depicted in Figure 3.

2.1.1 Battery Enclosure

The battery enclosure module holds a battery and electronics, and is used as a connection point for power and communication to other modules. Several battery enclosures can be connected in parallel to increase the robots range.

2.1.2 Drive Module

The drive module is used for propulsion. The module holds a motor and gear assembly with a flange on the output. A wheel connects to this flange.

2.1.3 Steering Module

The steering module holds a motor and gear assembly with a flange on the output shaft. This flange connects to a drive module. The steering module acts as a servo motor and turns the drive module about a vertical axis, pointing the drive wheel in whichever direction is desirable.

2.1.4 Suspension Module

The suspension module is fitted with a shock absorber and allows for vertical travel. This module can be connected between a steering module and the robots frame to increase the robot's traction on uneven terrain, as it helps with keeping wheels in the ground.

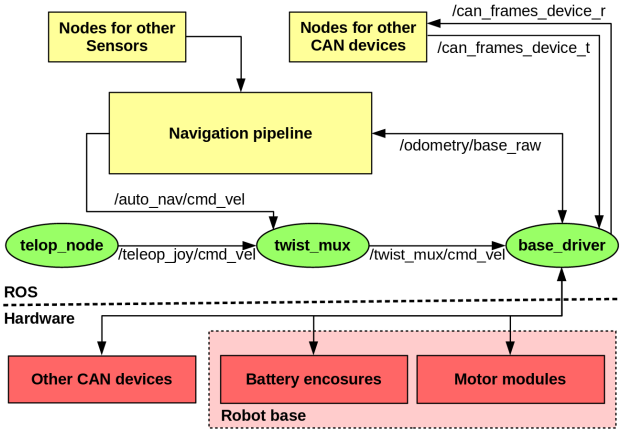


Figure 4: Simplified diagram of ROS nodes and robot base hardware

```
# Base calculator is used for calculating joint commands from velocity commands goven in the
base_link frame, and for calculating odometry from actual joint states
base_calculator: "pltf_clc_std::PltfClcStd"


# Motor drives
# type 0: propulsion/steering (default module)
# type 1: propulsion/propulsion (used for some diff. drive robots)
motor_drives:
  (
    drive0: {node: 1,
             x: -0.547, y: -0.30,
             type: 0,
             leg_mesh: 0,
             r_wheel: 0.2, prp_gr_rt: 42, prp_enc_ppr: 1024, prp_max_rpm: 3000, prp_sim_a: 1.5,
             str_gr_rt: 42.8571429, str_enc_ppr: 1024, str_lim: 3.078027759, str_sim_v: 4},

    drive1: {node: 2,
             x:  0.547, y: -0.30,
             type: 0,
             leg_mesh: 1,
             r_wheel: 0.2, prp_gr_rt: 42, prp_enc_ppr: 1024, prp_max_rpm: 3000, prp_sim_a: 1.5,
             str_gr_rt: 42.8571429, str_enc_ppr: 1024, str_lim: 3.078027759, str_sim_v: 4},

    drive2: {node: 3,
             x:  0.547, y:  0.30,
             type: 0,
             leg_mesh: 2,
             r_wheel: 0.2, prp_gr_rt: 42, prp_enc_ppr: 1024, prp_max_rpm: 3000, prp_sim_a: 1.5,
             str_gr_rt: 42.8571429, str_enc_ppr: 1024, str_lim: 3.078027759, str_sim_v: 4},

    drive3: {node: 4,
             x: -0.547, y:  0.30,
             type: 0,
             leg_mesh: 3,
             r_wheel: 0.2, prp_gr_rt: 42, prp_enc_ppr: 1024, prp_max_rpm: 3000, prp_sim_a: 1.5,
             str_gr_rt: 42.8571429, str_enc_ppr: 1024, str_lim: 3.078027759, str_sim_v: 4},
  )
```

Figure 5: Excerpt of robot configuration file. Modules used to create a robot are listed here with key parameters.

2.1.5 Passive Wheel Modules

Various passive wheel modules can be used to support the robot. One example is the rear caster wheels on the tall differential drive robot seen in the background in Figure 1, another example is the rear support wheels seen on the one-wheel drive robot in Figure 2b.

2.2 ROS Packages

Robot Operating System (ROS) is used as the software framework for the robot. The robot's software is designed to support the modular hardware. This means that all Thorvald II robots run the same software for driving the robot base, calculating odometry from joint states and so on. The only aspect separating one robot from another, is one single configuration file.

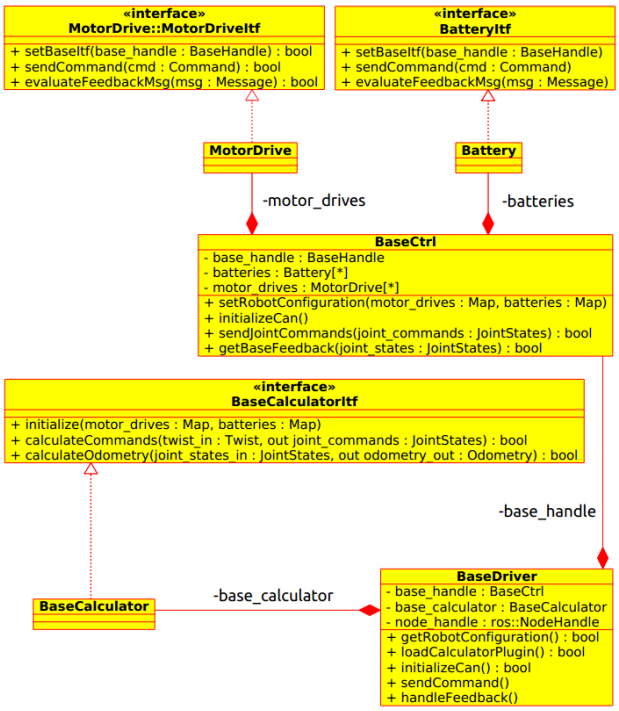


Figure 6: Simplified diagram of the BaseDriver class

Several ROS packages have been made available to the community. Packages that are relevant to this paper include:

- *thorvald_base* simulates or communicates with a real robot.
- *thorvald_teleop* provides a hardware-independent node for teleoperation.
- *thorvald_gazebo_plugins* provides a plugin for gazebo for simulating robots.
- *thorvald_model* provides a script for generating robot descriptions from robot configuration files. It also provides example robot configuration files.
- *thorvald_can_devices* provides interfaces for various types of CAN devises, such as motor drives and batteries. It also provides libraries that implement these interfaces.
- *thorvald_twist_mux* provides configuration parameters for the twist multiplexer *twist_mux* from the external package with the same name.
- *thorvald_bringup* provides some useful example launch files for launching some common and some not so common robot configurations.

Figure 4 shows a simplified setup for a real robot. Here we have two topics for velocity

commands: *telop_joy/cmd_vel* and *auto_nav/cmd_vel*. *telop_joy/cmd_vel* is used for teleoperation, and is published to by *telop_node* from the package *thorvald_teleop*, while *auto_nav/cmd_vel* is a placeholder for one or more topics used for commands during autonomous navigation. A multiplexer node subscribes to the velocity command topics and publishes messages from whichever topic that has the highest priority (and to which messages are currently being published) to *twis_mux/cmd_vel*. The *base_driver* from the *thorvald_base* package subscribes to this topic. The node calculates joint commands, which it sends to the robot base. The node also receives feedback from the base, which it uses to estimate robot velocities. Velocity estimates are published to *odometry/base_raw*. In the case of a simulated robot, *base_driver* will simulate joint states internally, and estimate robot velocities from these.

Modules containing motor controllers or batteries are connected to a Controller Area Network (CAN). The addresses of these modules are specified in the robot configuration file. If other devices are connected to the same CAN, the *base_driver* can be used to relay communication to and from these devices as well, on the *can_frames_device_t* and *can_frames_device_r* topics, respectively. This is useful in the case where the robot is fitted with an implement that communicates through CAN.

2.3 Robot Configuration

For each robot configuration, there is one configuration-specific file. This file lists the modules used on a given robot with relevant parameters, like position in the robot's coordinate frame, communication ID, simulation parameters and so on. The configuration file also specifies which kinematic model plugin *base_driver* — the node responsible for driving the base — should use for calculating joint commands and odometry. The parameters found in the configuration file selected by the user are loaded to the ROS parameter server at startup, where they are available for all nodes in the ROS network.

The user can easily create a new configuration file or modify existing files, and will normally have one file for each of her or his preferred hardware configurations, and then switch between these as the robot goes through rebuilds. An excerpt of a configuration file can be seen in Figure 5.

3 Driving the Robot Base

As our robots may take many forms, our software must be able to handle robots with different kinematic properties. Our robot must be able to calculate joint commands from configuration-independent velocity commands given in the robots coordinate frame, and it must be able to estimate the robot's actual velocity in the same coordinate frame based on joint feedback. Here we look at how our robot handles this.

3.1 Base Driver Node

The ROS node *base_driver* from the package *thorvald_base* is responsible for communicating with a real robot or simulating the robot base. The node uses an instance of the BaseDriver class from the same package to drive the robot base. Figure 6 depicts a simplified diagram of this class.

The node subscribes to robot velocity commands of type *geometry_msgs/Twist*, and outputs robot velocity estimates of type *nav_msgs/Odometry*. The node looks up the current robot configuration at startup. This information is passed to the members *base_calculator* and *base_handle*.

The base driver's *base_handle* is an instance of BaseCtrl and implements base related CAN hardware on the current robot configuration. Its members include motor controllers and batteries. Various motor controllers and batteries can be used, as long as they adhere to the motor controller interface, or the battery interface respectively, provided by the *thorvald_base* package.

The base driver's *base_calculator* is a plugin representing the kinematic model. This calculates joint commands from target robot velocities given in the robot's coordinate frame, *base_link*, and outputs odometry estimated from real or simulated motor feedback. The type of plugin to use is specified in the above mentioned configuration file. Most robots use the default plugin as this is designed to handle a wide variety of robots. Currently one-wheel drive robots have their own plugin. Two-wheel differential drive robots are supported by the default plugin, but a specialized and more efficient plugin is also available. In this paper, only the default plugin is considered.

Parameters used for calculating commands and velocity estimates can be changed at run-time. One example where this is useful can be found in a greenhouse environment. The robot shown in Figure 2a, assembled for greenhouse applications, is fitted with special double wheels for driving on flat floors and rails, respectively. These wheels have different diameters, and the robot therefore updates the diameter to be used in calculations whenever it transitions between flat surfaces and rails. This robot is described in more detail in Grimstad et al. (2018).

3.2 Joint Commands

The robot's velocity commands are given in the *base_link* coordinate frame. This coordinate frame is rigidly attached to the robot, with *x-axis* in the forward direction and *z-axis* pointing directly upward. This means that the robot can translate in the x and y and rotate about z. For velocity commands with non-zero rotation, the center of the arc that the robot is moving on is calculated, and steering angles and wheel speeds are calculated accordingly. For pure translation, steering angles and wheel speeds are the same for all wheels, and its just a matter of pointing all wheels in the desired direction.

Joint commands are passed to each motor controller instance, and sent to the robot base. For steering modules, rotation is constrained by the power and signal cable going to the drive module. The motor controllers are therefore electrically limited to 180 degrees in either direction. The steering module allows for fast rotation of the drive module, but we do not want to turn more than necessary. In many cases smoother operation can be achieved by offsetting the desired steering angle by 180 degrees and reversing the wheel speed command. A simple example is a robot altering between driving straight forward and straight backwards. According to the calculated joint commands, steering position should alter between 0 and 180 degrees, and wheel speed should remain positive. In this case it is obvious that it is better to leave the steering in the straight ahead position and reverse wheel speed when backing. For this reason, steering position commands will be offset and wheel speed commands reversed if this means reaching the target faster, or better avoiding the rotation limits. This is done before commands are sent to the base. Each motor is connected to a motor controller with a control loop for reaching target wheel speeds and steering positions.

3.3 Odometry

Calculating joint commands for the robot is somewhat trivial, calculating robot odometry from joint states is less so. First, we may encounter robots with very different kinematic properties, and second, the various wheels do not necessarily fully agree on which direction the robot is moving at any given time.

3.3.1 Estimating Robot Velocity

All the robot's steering and propulsion motors are fitted with encoders, and wheel speeds and steering positions are therefore available. Motor positions in the *base_link* frame are known from the robot configuration file. For a number of iterations, we pair wheels at random and calculate the point of intersection between the lines normal to each wheel in the ground plane. The average intersection point gives us a decent estimate of the robot's center of rotation, that is, the center of the arc on which the robot is driving. We use this together with wheel speeds to estimate the robot's movement.

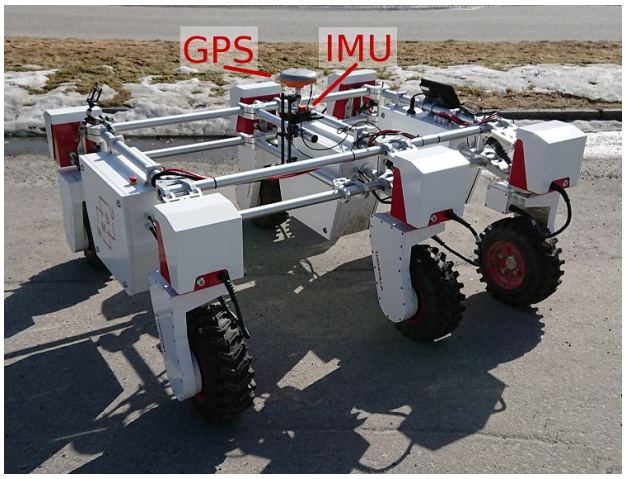


Figure 7: One of the robots that were used for testing the robot's ability to estimate velocity from joint states. This robot has six-wheel drive and six-wheel steering.

3.3.2 Verification of Estimated Robot Velocity

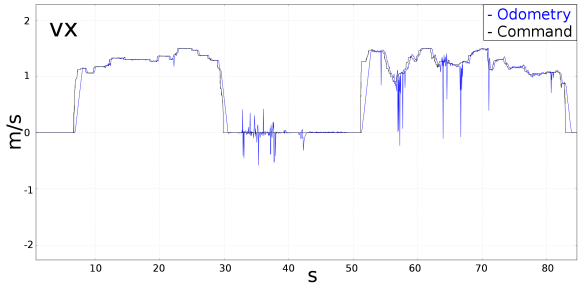
Wheel-slip varies across different ground surfaces, tires may be of slightly different diameters, be inflated to slightly different pressures, and so on. As such, encoder-based odometry is, on its own, not sufficient for keeping track of the robot's absolute pose in the world. Encoder-based odometry is, however, still very useful e.g as input for SLAM algorithms, or for keeping track of the robot between low-frequency GPS measurements. It is therefore important that the robot is capable of providing accurate velocity estimates from encoder feedback.

The simple approach for estimating robot velocity from joint states, described above, has shown to work well with all configurations it has encountered. Tests with two different robot configurations can be found in in Grimstad and From (2018). Here the robot's path calculated from odometry was plotted against ground truth. Ground truth was recorded using RTK-GNSS and IMU. The result of one of these test are summarized below.

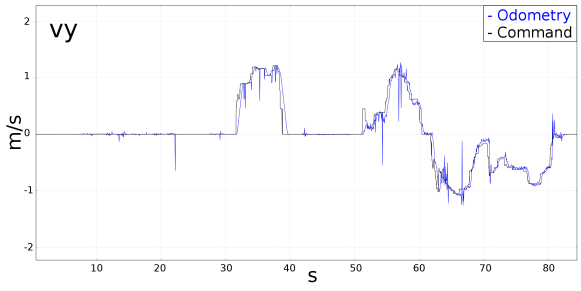
Figure 7 depicts the six-wheel drive, six-wheel steering, asymmetric robot used in the test. The results are shown in Figure 8 and 9. In this particular test, the robot was driving mostly on tarmac with some gravel patches. Commands given in the robot's *base_link*

Figure 8: Path calculated from estimated robot velocity based on motor feedback together with the path recorded using RTK-GNSS. The robot model is to scale and the grid is 1 m x 1 m.

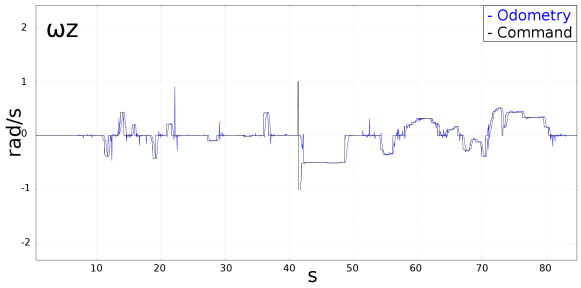
frame consisted of translation in x and y, as well as rotation about z, and the robot drove for 1 minute and 17 seconds before stopping in roughly the same position as it started. During this time the robot traveled 80.2 m according to ground truth and 81.0 m according to odometry. When the robot stopped, the difference between the robot's position in the world frame according to ground truth and odometry was 2.2 m. There is in other words little slip, and we conclude that the robot provides accurate encoder-based velocity estimates.



(a) Linear velocity, x component



(b) Linear velocity, y component



(c) Angular velocity, z component

Figure 9: Velocity commands and raw odometry from a six-wheeled robot

4 Simulation

When tasked with creating a new robotic system, being able to simulate that system may be vital to reduce costs and risk to human life or property. However, setting up a realistic simulated environment with realistic robots driving about may be a time consuming and tedious endeavor. If custom robots are used, the developer must spend time on creating a good robot model and all the surrounding aspects related to getting it to move in the simulated environment.

Our modular system enables us to create a wide range of robots. We do not wish to reinvent the wheel every time a new robot design is to be run in simulation. As described above, our robot's software is designed to support a great deal of variation in hardware. It is therefore important that our simulation too is able to cope with the robots we wish to build. To achieve this, a Xacro script was created for generating robot descriptions from the above mentioned configuration file. A Gazebo model plugin was also created.

4.1 Robot Description

The Xacro script for generating robot descriptions is found in the *thorvald_model* package and takes the parameters found in the above mentioned robot configuration file as input. The script outputs the robot description in URDF. This can for example be used to visualize the robot in RViz (3D visualization tool for ROS).

The user may specify additional Xacro files that

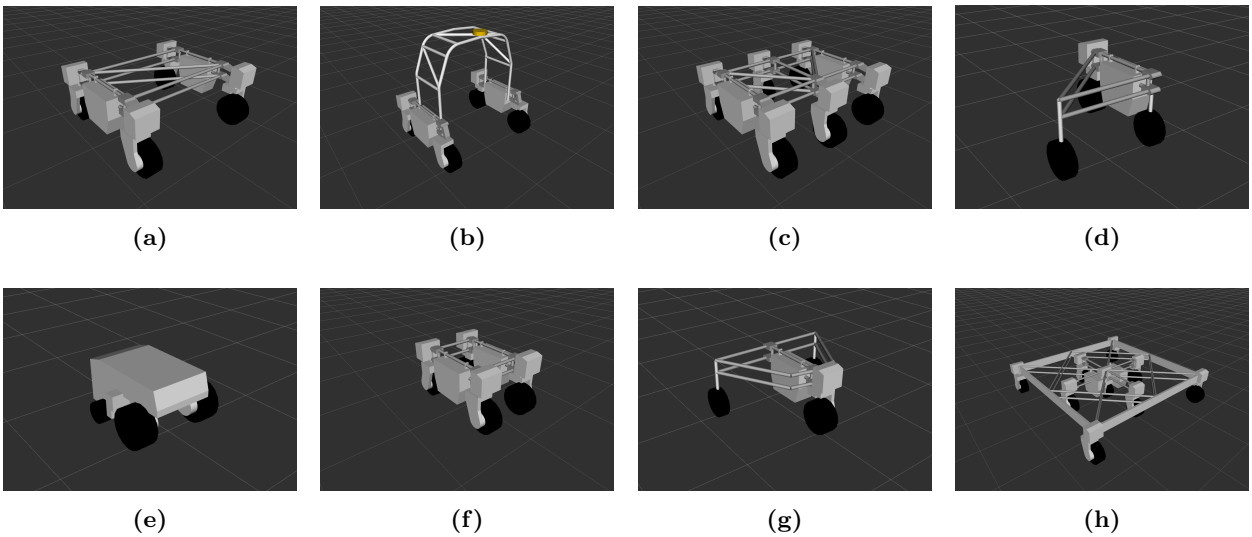


Figure 10: Robots generated by the presented system. (**a**) 4WD, 4WS, wide, with suspension, (**b**) 4WD, 2WS, custom tall frame, (**c**) 6WD, 6WS, with suspension, (**d**) 1WD, 1WS, rear-wheel drive, (**e**) 2WD, 0WS, custom frame, (**f**) 4WD, 4WS, slim, (**g**) 1WD, 1WS, front-wheel drive, (**h**) 9WD, 9WS.

should be included by the script. Here the user can add sensors or custom frame members that are not included in the default modules.

The output URDF robot description also includes mass and inertia properties for each module on the robot. This means that the robot can be simulated in various robot simulation frameworks. Here we will focus on simulation in Gazebo.

Figure 10 shows a handful robots created by the script. These are just some of the endless configurations that are fully supported in Gazebo simulation and in the real world. The robots vary in size and drive type, but they are all created using the same modules. Most of the depicted robots have also been assembled and tested in the real world.

4.2 Gazebo Simulation

The package *thorvald_gazebo_plugin* provides a Gazebo plugin for use with ROS. This plugin reads the robot description from the robot's namespace on the ROS parameter server at startup, and moves the robot's joint according to the joint states simulated by the *base_driver*.

The robots in Figure 10b and 10e are fitted with custom frames, and therefore make use of the optional additional Xacro for adding custom frame meshes to the model. The script also supports TF prefixes and namespaces for individual robots, thus enabling simulation of multiple robots at the same time, as seen in Figure 11.

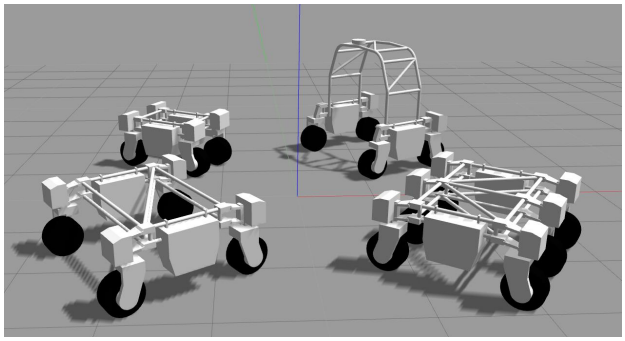


Figure 11: Four different robot configurations in the same Gazebo simulation

5 Conclusion

In this paper we have described the workings of ROS packages for running and simulating arbitrary robots assembled from Thorvald II modules. These packages have been made available for the community in the hope that they may be of use.

We have discussed key elements of the software framework, and described how several robot configurations can be achieved without making alterations to the robot's code. We have discussed how the robot calculates commands and odometry, and we revisited results from earlier experiments quantifying the robot's performance.

The provided software packages are designed to be used with robots created from Thorvald II modules, both real and simulated. The high level of customability means that they can also be used for emulating a

wide range of robots that are not created from Thorvald modules, and run these in simulation. As many ROS-based mobile robots adhere to the same conventions, e.g. which message types to use for commands or odometry, integrating the provided packages into a users own system will in many cases require little or no alterations to the users setup.

The presented packages enable simulation of a wide array of robots. Tweaking, or completely redesigning a robot is done by modifying only one single configuration file, and several different robot configurations may run in the same simulation. The presented packages thus provide the user with a powerful tool for testing various robot designs for swarm applications in simulation. People who are looking to buy or make their own robot may benefit greatly from the presented packages, as experimenting with dimensions, drive types, motor parameters and so on, helps them to identify and specify their requirements.

The reader should note that the presented software is subject to continuous development. Certain aspects of the packages may therefore change without warning. Updated information on the packages and how to install and use them is available on the NMBU Robotics web pages:
www.nmbu.no/en/faculty/realtek/research/groups/roboticsandcontrol/thorvaldinstall

References

Bak, T. and Jakobsen, H. Agricultural robotic platform with four wheel steering for weed detection. *Biosystems Engineering*, 2004. 87(2):125 – 136. doi:10.1016/j.biosystemseng.2003.10.009.

Bawden, O., Ball, D., Kulk, J., Perez, T., and Russell, R. A lightweight, modular robotic vehicle for the sustainable intensification of agriculture. In *Australian Conf. on Robotics and Automation*. Univ. Melbourne, 2014. URL http://eprints.qut.edu.au/82219/.

Botterill, T., Paulin, S., Green, R., Williams, S., Lin, J., Saxton, V., Mills, S., Chen, X., and Corbett-Davies, S. A robot system for pruning grape vines. *Journal of Field Robotics*, 2016. 34(6):1100–1122. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21680, doi:10.1002/rob.21680.

Feng, Q., Zou, W., Fan, P., Zhang, C., and Wang, X. Design and test of robotic harvesting system for cherry tomato. *International Journal of Agricultural and Biological Engineering*, 2018. 11(1):96–100. doi:10.25165/j.ijabe.20181101.2853.

Grimstad, L. and From, P. J. Thorvald II - a Modular and Re-configurable Agricultural Robot. In *IFAC 2017 World Congress*. 2017a. doi:10.1016/j.ifacol.2017.08.1005.

Grimstad, L. and From, P. J. The thorvald ii agricultural robotic system. *Robotics*, 2017b. 6(4). URL http://www.mdpi.com/2218-6581/6/4/24, doi:10.3390/robotics6040024.

Grimstad, L. and From, P. J. A configuration-independent software architecture for modular robots. In *4th IEEE/IFToMM Intl. Conf. on Reconfigurable Mechanisms and Robots*. 2018.

Grimstad, L., Zakaria, R., Le, T. D., and From, P. J. A novel autonomous robot for greenhouse applications. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

van Henten, E., Hemming, J., van Tuijl, B., Kornet, J., Meuleman, J., Bontsema, J., and van Os, E. An autonomous robot for harvesting cucumbers in greenhouses. *Autonomous Robots*, 2002. 13(3):241–258. doi:10.1023/A:1020568125418.

Koenig, N. and Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*, volume 3. pages 2149–2154 vol.3, 2004. doi:10.1109/IROS.2004.1389727.

Lehnert, C., English, A., McCool, C., Tow, A. W., and Perez, T. Autonomous sweet pepper harvesting for protected cropping systems. *IEEE Robotics and Automation Letters*, 2017. 2(2):872–879. doi:10.1109/LRA.2017.2655622.

Mueller-Sim, T., Jenkins, M., Abel, J., and Kantor, G. The robotanist: A ground-based agricultural robot for high-throughput crop phenotyping. *2017 IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2017. pages 3634–3639. doi:10.1109/ICRA.2017.7989418.

Quigley, M. and et.al. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, 2009.

Ruckelshausen, A. and et.al. Boniroban autonomous field robot platform for individual plant phenotyping. In *European Conf. Precision Agriculture*. pages 841–847, 2009.

Ye, Y., Wang, Z., Jones, D., He, L., Taylor, M. E., Hollinger, G. A., and Zhang, Q. Bin-dog: A robotic platform for bin management in orchards. *Robotics*, 2017. 6(2). doi:10.3390/robotics6020012.