Simulation Modelling Practice and Theory 87 (2018) 155–169

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Modelling and simulation of a non-holonomic omnidirectional mobile robot for offline programming and system performance analysis

Mostafa Sharifi*[a], XiaoQi Chen[a], Christopher Pretty[a], Don Clucas[a], Erwan Cabon-Lunel[b]

[a] Department of Mechanical Engineering, University of Canterbury, Christchurch 8041, New Zealand
[b] École Nationale Supérieure de Techniques, Paris 91120, France

ARTICLE INFO

Keywords:
Gazebo simulator
ROS
Wheeled mobile robot
3D simulation
Kinematic modelling
State estimation

ABSTRACT

This paper presents 3D modelling and simulation of a non-holonomic omnidirectional mobile robot, MARIO – Mobile Autonomous Rover for Intelligent Operations, using the Gazebo simulator and Robot Operating System (ROS), aiming for offline programming and system performance analysis. For this purpose, MARIO as a four wheel active driving/steering (4WD4S) platform has been modelled and simulated based on the physical developed model. Gazebo enables simulation of the world environment, physical model, sensors and control system through the Unified Robot Description Format (URDF) file. ROS is interfaced with Gazebo which allows utilization and implementation of different robotic software and tools on the simulated robot. This presented approach allows development, testing and validation of MARIO and required software before implementation on the real system. The presented approach also provides the essential theory and practice for robotic system specialists in modelling and simulation of ground mobile robotic systems using Gazebo simulator and ROS.

1. Introduction and background

Nowadays modelling and simulation are major parts of the scientific and engineering processes, especially of robotic systems. Modelling and simulation play an important role in offline programming, performance analysis, and development of advanced control algorithms for robotic systems [1]. Design, testing, and validation of robotic systems ranging from indoor mobile robots [2] to outdoor mobile robots [3], articulated industrial manipulators [4], underwater robotic systems [5], and humanoid robots [6] would be impossible without proper modelling and simulation tools. Furthermore, simulation can be used as a tool to develop virtual environments for training operators [7] as well as an educational tool for teaching and learning basic concepts of robotic systems [8]. Additionally, simulation provides low cost means of testing and experimentation, and makes controlling disturbances much easier compared with using real robotic systems [9].

Along with the development and progress in powerful and affordable computing technologies in last two decades, a number of proprietary and open source robotic modelling and simulation software have been developed. Examples of open source robotic simulator software which have achieved popularity among users and are available freely for personal or academic use are Open

* Corresponding author.
E-mail addresses: mostafa.sharifi@pg.canterbury.ac.nz (M. Sharifi), xiaoqi.chen@canterbury.ac.nz (X. Chen), chris.pretty@canterbury.ac.nz (C. Pretty), don.clucas@canterbury.ac.nz (D. Clucas), erwan.cabon–lunel@ensta-paristech.fr (E. Cabon-Lunel).

https://doi.org/10.1016/j.simpat.2018.06.005
Received 6 May 2017; Received in revised form 21 May 2018; Accepted 26 June 2018
Available online 28 June 2018
1569-190X/

Dynamics Engine (ODE) [10], Robotic Toolbox for MATLAB [11], Microsoft Robotics Developer Studio (MRDS) [12], Webots [13], Virtual Robot Experimentation Platform V-Rep [14], Modular Open Robots Simulation Engine MORSE [15], and Gazebo [16]. MRDS has not been updated since version 4.0 which was released in 2012. Although replacement tools such as VIPLE (Visual IoT/Robotics Programming Language Environment [17]) has been developed by MRDS users which provides multiple simulation environments integration, including Unity simulator and 2D/3D Web simulators [18]. Selecting the most suitable simulation tool for a specific purpose like research, development or education can be difficult. The variety of simulation tools, features provided by each tool, user-friendliness and dependency on external packages are some of the main considerations which can make it challenging to choose the most suitable robotic simulation tool [1].

The Gazebo Project was a part of the Player/Stage/Gazebo projects, which have been developed since 2001. The Player Project provides a network interface to different types of robots and sensors. Stage is a simulator for a large population of mobile robots in a 2D environment [19]. Gazebo is a multi-robot simulation tool which has the capability of accurate and efficient simulation of a population of robots, sensors and objects in a 3 dimensional world. Gazebo generates realistic sensor feedback and has a robust physics engine to generate interactions between objects, robots and environment. Furthermore, Gazebo provides high-quality graphics, and suitable programmatic and graphical user interfaces [16]. Gazebo is offered freely as a stand-alone software, but has also been packaged along with Robot Operating System (ROS) as the simulation tool. ROS was originally developed by the Stanford Artificial intelligence Laboratory in Support of the Stanford AI Robot (STAIR) project. ROS is an open source robotic middleware that provides libraries and tools to help software developers produce robot programs. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more [20].

There have been many robotics research projects where Gazebo/ROS has been used widely. BoniRob is an autonomous field robot platform for individual plant phenotyping and Gazebo has been used to model and simulate the farm environment and the robotic model [21]. Linner and Shrikathiresan presented modelling and operating robotic environments using Gazebo/ROS with some common examples to prove the performance and effectiveness of using ROS and Gazebo for this aim [22]. A comprehensive simulation of quadrotor UAVs has been carried out using ROS and Gazebo [23] and is able to simultaneously simulate different features such as flight dynamics, on-board sensors like IMUs, external imaging sensors and complex environments. A set of simulations for manipulation tasks using ROS and Gazebo have been completed to illustrate the techniques of implementing robot control in a short time [24].

MARIO - Mobile Autonomous Rover for Intelligent Operation, is a four wheels active driving/steering (4WD4S) research prototype that has been designed to be used in the agricultural environment for automation of the agricultural tasks such as data collection, monitoring and inspections. The integrated mechatronic design and development of MARIO prototype has been presented in [25]. MARIO is equipped with on-board sensors for localization including GPS, stereo camera, and IMU. This paper presents the approach for offline programming and performance analysis of MARIO through modelling and simulation using Gazebo simulator and ROS.

Section 2 presents the parallel process of both a simulation model of the robot and the environment using Gazebo and ROS. MARIO simulation also includes simulation of the sensors and hardware interfaces to interact with the actuators. Section 3 describes the developed model based control system, and a brief explanation of the ROS third part packages used for the state estimation, and navigation system. Section 4 reports the experimental results and validation of the developed software by testing on both simulated and real model. Section 5 sums up findings and conclusion of the work presented in this paper.

2. Model description and simulation

In this section, all the physical links and joints, as well as kinematics and dynamics, sensing and basic control interfaces of MARIO will be modelled and described individually. The Gazebo simulator provides dynamics simulation by considering gravity, friction, and contact forces provided by the included physics engines such as Open Dynamics Engine - ODE, Simbody, Dynamic Animation and Robotics Toolkit - DART or Bullet. Different types of Gazebo plugins enable the development of control interfaces and sensing systems for the simulated robots. The main parts required to model a controllable robotic system in general are:

1. World environment model description
2. Physical model description
   - Kinematic and dynamic modelling of the robot links
   - Kinematic and dynamic modelling of the robot joints
3. ROS/Gazebo Plugins to model the sensors and control/hardware interfaces

Fig. 1 shows the general structure and the required components in Gazebo to model a robotic system.

2.1. World description

World is a general term to describe objects, global parameters and physics properties. By default, a world is defined by Gazebo with default required parameters. The objects in the world can be static or dynamic. Static objects such as building, lights or walls are defined by their visual and collision geometry. Dynamic objects such as robots are defined not only by visual and collision geometry but also by their inertia information. Objects can be created using standard geometric shapes, or inserted from the model database, or created and by any 3D modelling tools and imported to the Gazebo simulator environment.

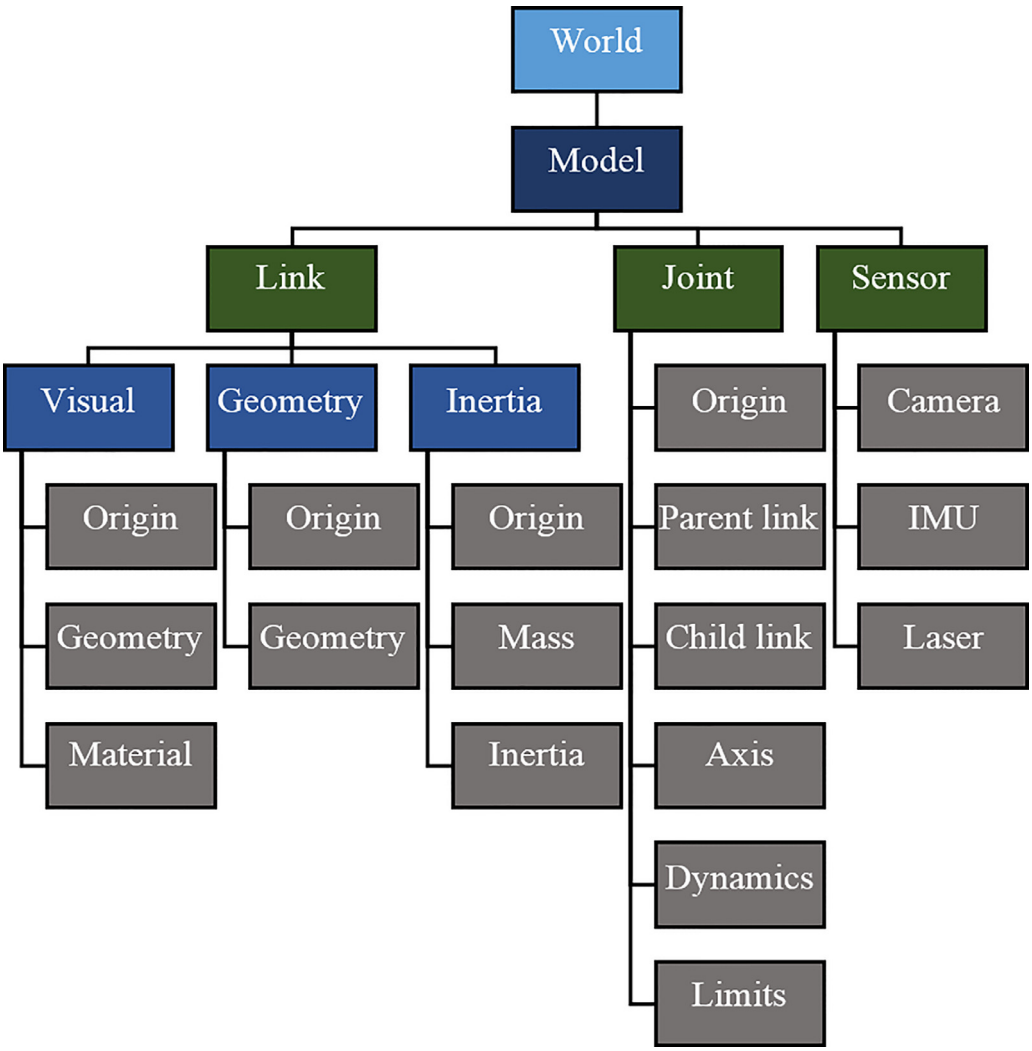

Fig. 1. General structure of required components to model a robotic system in Gazebo.

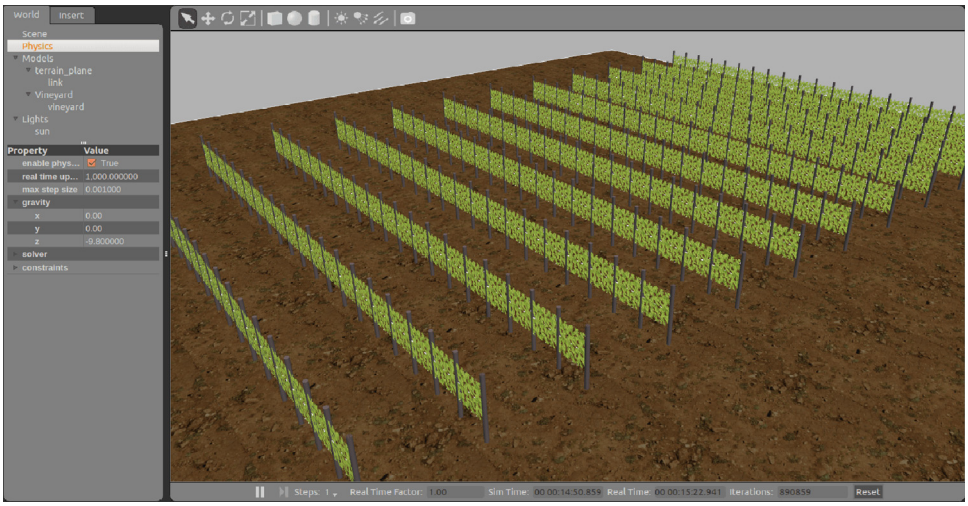

Fig. 2. Gazebo simulator environment including objects in the world and world parameters; vineyard development, including the terrain and the vine trees with parameters.

Fig. 2 shows a simulated environment in Gazebo with the set physics parameters such as gravity. Two static objects as the modelled terrain and agricultural environment (vineyard) are also included in the simulated world environment. Both objects were created using the SketchUp software.

2.2. Physical model description

2.2.1. MARIO physical platform description

MARIO as a 4WD4S platform consists of 4 wheel modules attached to a chassis body that holds all the electronics. Each of the modules is equipped with a servo to provide steering and a DC motor to drive the wheels. It is also equipped with equipped with the ZED [v.1.0, Stereolabs] stereo camera, a 9 DOFs Razor IMU and a Swiftnav Piksi RTK GPS unit as well. The platform base dimension is 0.52 m × 0.52m × 0.33 m. Figure shows the physical developed model with the on-board sensors and electronics on field and the 3D CAD model of the MARIO base developed by SolidWorks.

2.2.2. MARIO simulated model

A robot, as a dynamic object in the world, consists of links that are connected to each other by joints. A link in Gazebo describes the kinematic and dynamic properties of a physical link in the form of visual/collision geometry and inertia information. A joint models kinematic and dynamic properties of a joint such as joint type, motion axes, and joint safety limits. All these information are described in the Universal Robotic Description Format - URDF file format [26]. URDF is an XML file format used by Gazebo and ROS to model all the components of a robot. To model MARIO as a 4WD4S platform, 9 links and 8 joints need to be defined. The base_link describes the chassis and four links (link1 to link4) are connected to it by four revolute joints (servo1 to servo4) as steering links. Each of the steering links are connected to a driving link (link11 to link44) as wheel by a continuous joint (motor1 to motor4). Each sensor also should be defined as a physical link attached with a fixed type joint to the base_link. with no attached sensor. A virtual link called base_footprint is also needed by some of the of the ROS third party software such as navigation which is placed on the ground and it is sized as the footprint of the platform. Fig. 4 presents the kinematic schematics of the base_link and a wheel module with its links and joints. Fig. 5 shows the kinematic diagram of MARIO platform, showing the kinematic chain of all the physical and virtual links and joints, each joint with the 6 DOFs information of its placement with respect to the previous link.

2.2.3. Physical geometry and inertia

The physical geometry of each link needs to be defined in the form of visual and collision geometry. There are two ways to model the geometry of each link of a robot, inserting standard 3D shapes, or importing as mesh files. Physical geometry for a typical four wheel WMR can be modelled using a cubic box as the chassis and 4 cylindrical shapes as the wheels. Using continuous joints, wheels can be attached to the chassis. Inertial information of each link is essential, if a proper simulation is required. Inertial parameters define the mass, centre of the mass, and the moment of inertia tensor matrix for each modelled link. These information are obtained from SolidWorks. Below is the example of MARIO URDF code to model the imu_link.

```
<link name="imu_link">
  <visual>
    <geometry>
      <box size="0.08 0.05 0.03"/>
    </geometry>
    <material name="blue">
      <color rgba="0 0 .9 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <box size="0.08 0.05 0.03"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="0.02"/>
    <inertia ixx="0.0001" ixy="0" ixz="0" iyy="
0.000001" iyz="0" izz="0.0001"/>
  </inertial>
</link>
```

The alternative approach to define visual and collision geometry is to use mesh files in the format of COLLADA or STL files. These files can be generated by CAD tools such as Blender, SolidWorks or SketchUp. The MARIO 3D CAD model, Fig. 3(b), was developed using SolidWorks as described in the previous section. To provide colour information, COLLADA file format can be used to present visual geometry of a link. Collision geometry does not need material or colour information, therefore it can be represented as STL file

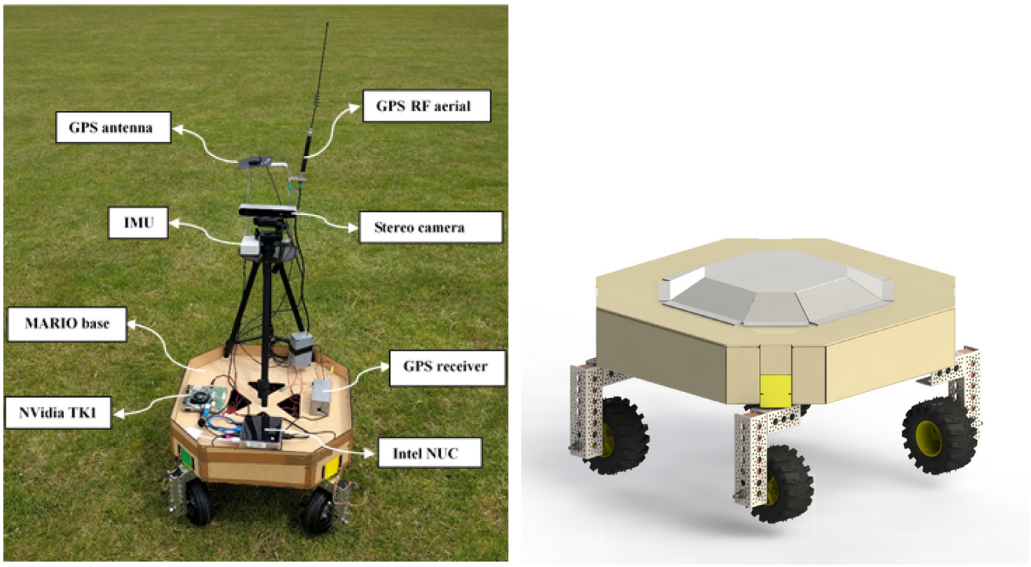

Fig. 3. MARIO platform: a) MARIO developed prototype and the on-board electronics and sensors, b) 3D CAD model of MARIO base developed by SolidWorks.

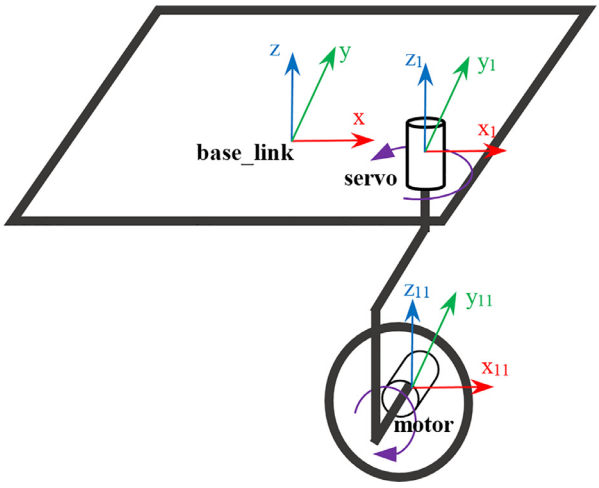

Fig. 4. Kinematic diagram of MARIO: kinematic schematic and coordinates of the base_link and one wheel module.

format.

Joint Representation Joints are needed to connect the links to each other and form the kinematic and dynamic relationships between them. Joint element in the URDF file defines the kinematics and dynamics of the joint as well as joint type and safety limits. Fig. 6 shows the tree structure of a two-link robot for better understanding of the different terms in joint element. Below is the modelled continuous joint element that connects the steering link (link1) to the driving link (link11) in MARIO.

```
<joint name="motor1" type="continuous">
    xyz="0.0 -0.10536 -0.030705"    rpy="0.0 0.0 0.0" />
    <parent link="link1"/>
    <child link="link11"/>
    <axis xyz="0 0 1" />
    <dynamics damping="0.0" friction="0.0"/>
    <limit effort="30" velocity="2.0"/>
</joint>
```

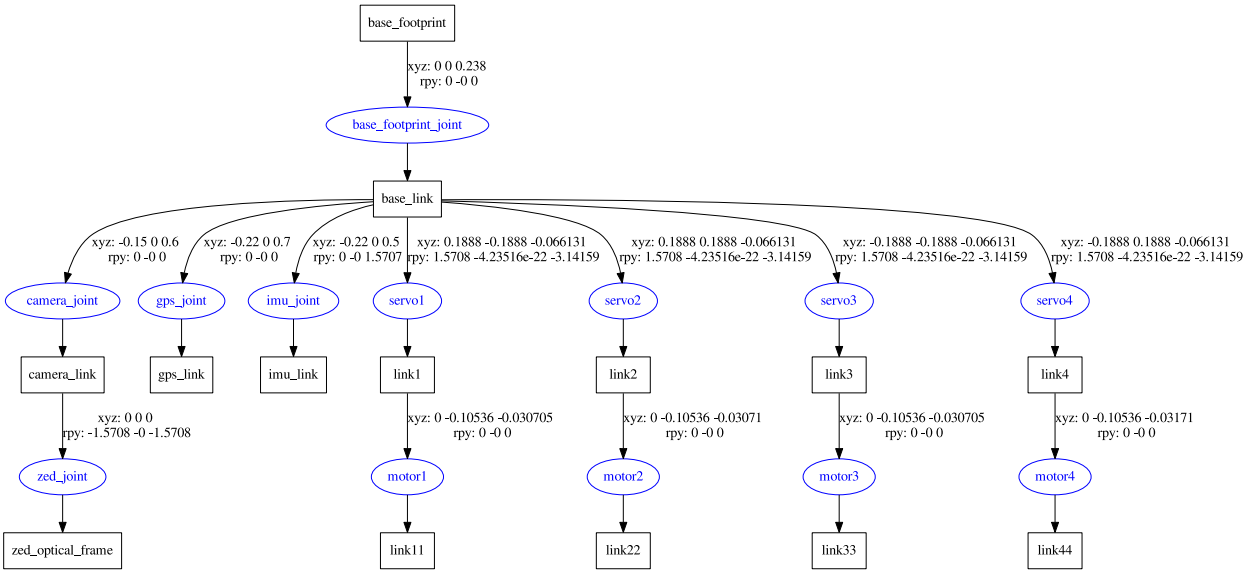

Fig. 5. Kinematic diagram of MARIO: kinematic chain generated from URDF file.

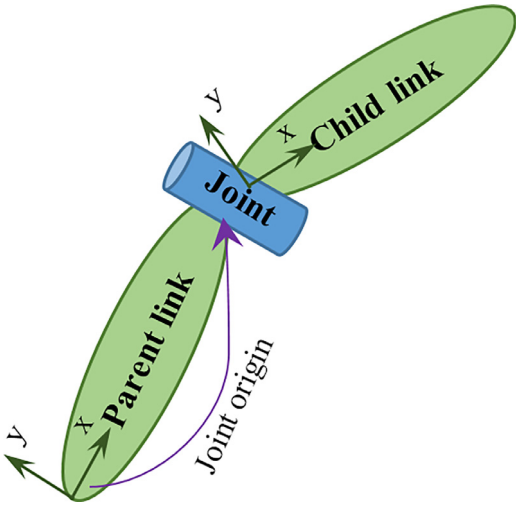

Fig. 6. Tree structure of a two-link robot.

2.3. Sensor modelling

With many different sensors on the robot, each sensor can be simulated independently as a Gazebo plugin and must be physically attached to the robot model as a link. These plugins are included in the URDF file. These plugins are typically C++ libraries loaded by Gazebo that have access to Gazebo' s API. In general, plugins are permitted by Gazebo to perform different kinds of task such as motion control or getting sensor data. These plugins output sensor information in form of standard ROS messages and services. Common parameters, such as error characteristics and transformation frame of each sensor, can be defined as well as other parameters related to each sensor. By default, sensors in Gazebo have no noise and they sense the simulated environment perfectly. To make them more realistic, noise can be added. A first order Gaussian error model is typically used for all the sensors to add noise to the measurement taken from each sensor. This is modelled by setting the mean and the standard deviation of the Gaussian distribution. Each measurement of $Y(t)$ at time t is given by:

$$Y = \hat{Y} + B + N_Y \tag{1}$$

$$\dot{B} = -\frac{1}{\tau}B + N_B \tag{2}$$

where in Eqs. (1) and (2), $\hat{Y}$ is the raw measured value, B is the bias, N_Y is additive noise that affects the measurement, and N_B defines

the characteristics of random drift with time constant τ [23].

2.3.1. Inertial measurement unit (IMU)

The inertial measurement unit (IMU) reports the robot body' s acceleration from a three axis accelerometer, angular rates from a three axis gyroscope and absolute orientation around the Z axis by a magnetometer. Integration of these measurements with other sensors provides a good reference for a localization system. Noise and bias are the two types of disturbance that are applied to the angular rates and acceleration measurements of IMU. Four groups of parameters need to be set for the IMU model: angular rate noise and bias, acceleration noise and bias. Also, no noise or bias is added to the orientation measurement as it is considered a perfect value in the world frame. Noise is sampled and added from a Gaussian distribution by setting the mean and standard deviation of the Gaussian distribution. Bias is sampled once and will be added at the start of simulation. The simulated IMU on MARIO is modelled with no associated noise in Gazebo, the code below shows the required URDF code to model the IMU.

```
<joint name="imu_joint" type="fixed">
    <axis xyz="1 0 0"/>
    <origin xyz="-0.16 -0.05 0.51"/>
    <parent link="base_link"/>
    <child link="imu_link"/>
 </joint>
  <gazebo>
      <plugin name="imu_plugin" filename="
libgazebo_ros_imu.so">
          <alwaysOn>true</alwaysOn>
           <bodyName>imu_link</bodyName>
           <topicName>imu_data</topicName>
           <serviceName>imu_service</serviceName>
           <updateRate>10.0</updateRate>
      </plugin>
  </gazebo>
```

2.3.2. Stereo vision camera

Stereo camera is a vision system using at least two cameras to simulate the way human vision works. Stereo camera gives the ability to generate 3D images. Typically, two cameras that are placed horizontally from each other are used to capture images from different angles. Through several pre-processing steps, depth information is achieved as a disparity map. The disparity map includes the information of the differences in horizontal coordinates of the two input images. The Multi-camera plugin used by Gazebo simulates a stereo camera and synchronises their output. This plugin is similar to the Camera plugin that simulates a monocular camera. All the required parameters such as frame rate, image width and height, output format, base line distance and noise parameters can be defined within Gazebo. A Gaussian disturbance is sampled for each pixel individually and it is added to each colour channel of that pixel. In our system, a stereo vision system is used as a visual odometry source for the state estimation system to enhance the localization of the robot. Fig. 7 shows the visualization of simulated stereo camera by showing the left and right images of an example scene in Gazebo. The parameters from the physical ZED stereo camera were used to model the simulated stereo camera.

2.4. Control plugin

To be able to control the motion of each joint in one degree of freedom, a Gazebo control plugin is required. Gazebo also needs to be interfaced with a robot middleware such as ROS to control each joint. A meta package called *gazebo_ros_pkgs* provides a set of ROS packages to interface with Gazebo. A set of packages called *ros_control* provide controllers, hardware interfaces and toolboxes to control joint actuators. Common controllers such as *effort, position* and *velocity* are provided by *ros_control*. These controllers (typically PID type) take joint states and set points as inputs and output effort, position or velocity. Each joint is interfaced with the relevant controller by the relevant hardware interface. The same developed software control system is used to control the physical robot, so that the control messages are written to the physical speed/position controllers to actuate control the actuation of the joints. The speed and position feedbacks are read from the encoders back to the control system.

An overview of the low level control system of simulated MARIO is shown in Fig. 8. By launching the ROS package containing the URDF file of MARIO, the simulated model is opened in the virtual world of Gazebo. This also loads the control interface and waits for all the controllers to be loaded. The transmission tag in the MARIO URDF file defines the type of command interface and the relationship between the joint and actuator. Controller manager provides the infrastructure to load, start, stop and unload the

Fig. 7. Visualization of the simulated stereo camera as left and right images.

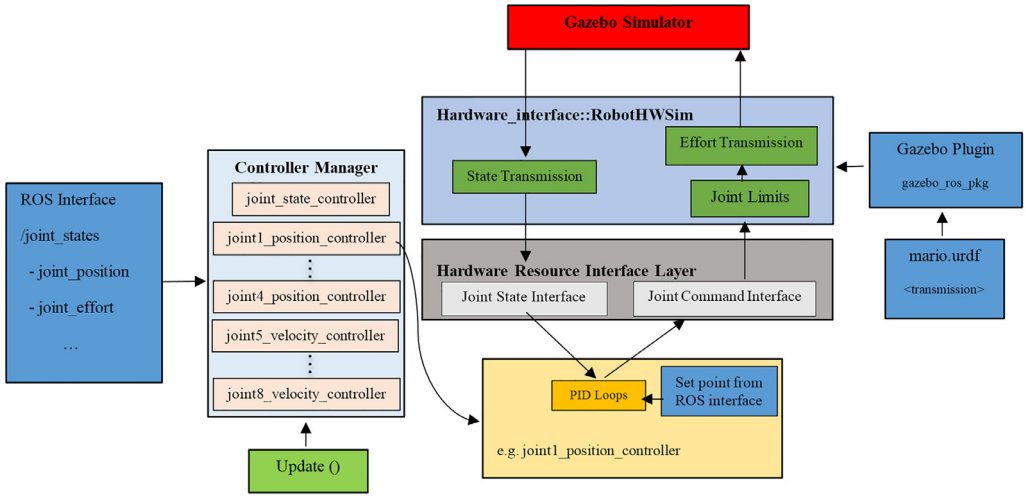

Fig. 8. Overview chart of low level control system of simulated MARIO in Gazebo and ROS.

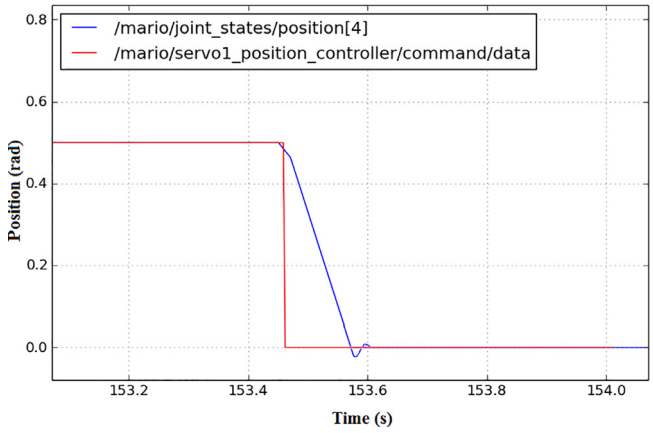

Fig. 9. Joint position controller performance.

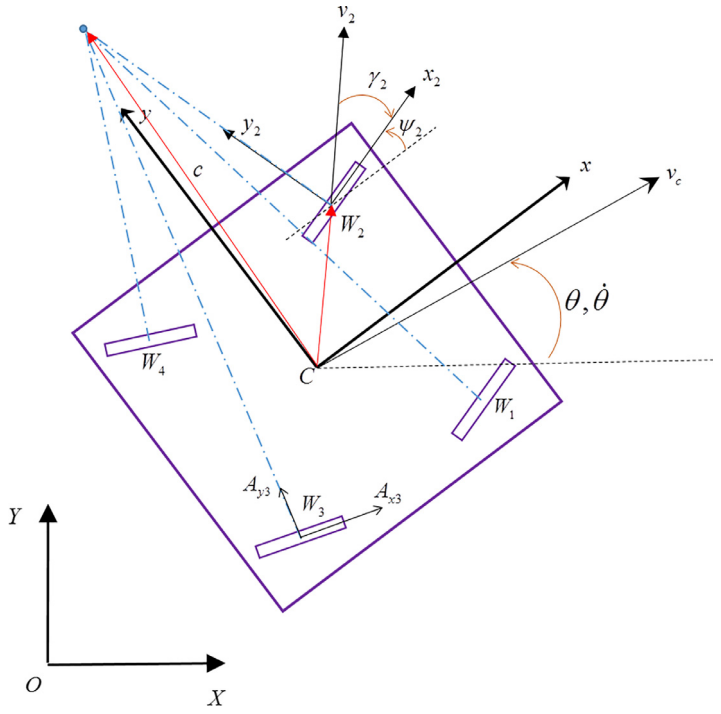

Fig. 10. Kinematic notation of MARIO.

controllers in a real-time manner. A YAML configuration file is also needed which includes information of controllers such as joint controller type and parameters. Hardware_interface provides position, velocity and effort interfaces between ROS and Gazebo. Our system has 9 controllers of which one provides joint states, four are position controllers to control steering joints and remaining four are velocity controllers to control each of the driving joints. Fig. 9 shows the performance of the PID position controller of joint1 and the response of the controller to the step input from 0.5 radians to 0.

2.5. Standard kinematics model

In general, ros_control provides a closed loop control system for each individual joint. To model a robotic system in Gazebo, kinematic formulation of the system is not required. However, kinematic formulation is essential to develop the control system. The kinematic diagram of MARIO is shown in Fig. 10. The robot frame is assumed as a rigid body that moves in a planar motion.

Velocity of each of the four wheels can be formulated as:

$$\overrightarrow{v_i} = \overrightarrow{v_c} + \dot{\theta} * k * \overrightarrow{W_i} \tag{3}$$

where in Eq. (3), $\overrightarrow{v_i}$ is the speed vector of each wheels, $\overrightarrow{v_c}$ is the linear speed vector and $\dot{\theta}$ is the angular rate of the robot' s frame in $\{C, x, y\}$coordinate with respect to the fixed global coordinate $\{O,\ X,\ Y\}$, $\overrightarrow{W_i}$ is the distance vector of each wheel from the centre of robot' s frame, and k is a unit vertical vector.

In the pure rolling condition, the wheel rotation rate, $\dot{\phi}_i$ and steering angle, ψ_i for a wheel with the radius of r with respect to the robot' s frame coordinate system are given by:

$$\dot{\phi}_i = \frac{|v_i|}{r}, \quad tan(\psi_i) = \frac{v_{iy}}{v_{ix}}, \quad i = 1, \ ...,4, \tag{4}$$

where in Eq. (4), v_{ix} and v_{iy} are the vector components of $\overrightarrow{v_i}$ in the robot' s local frame$\{C, x, y\}$ as:

$$v_{ix} = v_{cx} - W_{iy}\dot{\theta}, \quad v_{iy} = v_{cy} - W_{ix}\dot{\theta}, \tag{5}$$

The two terms in Eq. (4) define the kinematics constraints of driving speeds and steering angles of wheels with respect to the linear and angular velocities at the robot' s frame. Based on Eq. (4), the slip angle of each wheel γ_i can be formulated as:

$$\gamma_i = \cos^{-1}\left(\frac{v_{ix}}{|v_i|}\right), \tag{6}$$

Non-zero input linear and angular velocities to the base will results the whole platform to rotate around a centre of rotation based on the concept of rotational motion of rigid body. The centre of rotation based on the input velocity is formulated as:

$$\vec{c} = \frac{\vec{v_c}}{\dot{\theta}}, \tag{7}$$

Acceleration of each wheel a_i at point w_i and the acceleration of robot' s frame a_c at point C based on the kinematics are given by:

$$a_i = a_c + \ddot{\theta}k\overrightarrow{W_i} - \dot{\theta}^2\overrightarrow{W_i}, \tag{8}$$

where

$$a_c + a_{cx} + a_{cy}, \quad a_{cx} = (\dot{v_x} - \dot{\theta}v_y), \quad a_{cy} = (\dot{v_y} + \dot{\theta}v_x), \tag{9}$$

and by knowing $a_i = a_{cx} + a_{cy}$, therefore:

$$a_{ix} = \dot{v_x} - \dot{\theta}v_y - W_{ix}\dot{\theta}^2 - W_{iy}\ddot{\theta}, \tag{10}$$

$$a_{iy} = \dot{v_y} + \dot{\theta}v_x - W_{iy}\dot{\theta}^2 + W_{ix}\ddot{\theta}, \tag{11}$$

3. Control, state estimation, and navigation by ROS

3.1. Model base controller (MBC)

A model based controller (MBC) based on the kinematics of the system is responsible to drive the robot with a desired input speed. Through inverse kinematics, desired speed information in the local frame in the form of linear and angular velocities is received and the controller outputs driving speed and steering angle for each wheel. These outputs are set points for the relevant ROS controllers to control the position, velocity, or effort on each joint. The formulation provided in (Eqs. (3)–(7)) used in the process of inverse kinematics. The MBC is designed to switch between 4 modes of steering based on the received input velocity command as follows:

- angular and linear velocity is zero results in a stop command for the wheels;
- angular velocity is zero and the linear velocity is non-zero. In this case the wheels are aligned with the linear velocity vector of the base;
- angular velocity is non-zero and the linear velocity is zero. In this case the robot executes a zero turn, where the wheels are angled tangential to the robots centroid.
- Rotational and linear velocity is non-zero. In this case the center of rotation is calculated (Eq. (7)) and the wheels are angled tangential to that center.

The forward kinematics in MBC is achieved using the Kabsch algorithm [27]. This algorithm computes the best fit translation between two related sets of points ($\{P, Q\}$) based on the minimization of the Root Mean Square Deviations (RMSD). The algorithm works by calculating the covariance matrix of the related sets of point $A = P^TQ$. Using this covariance matrix the rotation matrix is calculated using a Singular Value Decomposition (SVD) depicted in Eqs. (12) and (13).

$$A = VSW^T, \tag{12}$$

$$R = W\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix}V^T, \tag{13}$$

where R is the resulting rotation matrix and $d = sign(det(WV^T))$ in order to preserve a right-handed coordinate system. This rotation matrix is then used to calculate the transformation between the sets of points. In terms of the odometry information each point in the set is the old position of each wheel and the new position of each wheel. The calculated transformation then approximates the relative motion of the base.

3.2. State estimation and navigation

State estimation, control, and navigation systems are key components of a mobile robotic system to enable the robot to perform required tasks. Fig. 11 shows an overview chart of the control system, state estimation and navigation system.

The state estimation system includes an EKF, using *robot_localization* package to estimate the robot pose by fusing all the measurements from multiple sensors. For our system, relative displacement in form of *X, Y*, and *Z* positions in the world frame is provided by wheel odometry and visual odometry. Orientation information is provided by IMU in the form of absolute orientations (roll, pith, and yaw) and angular velocities. In a 2D navigation scenario, yaw and *Z* axis angular velocity are needed to provide orientation information for EKF.

To drive the robot in the world frame, a navigation system is needed. This is achieved by the 2D navigation stack provided by ROS. Navigation stack takes pose information of the robot from the state estimation system and a goal pose from user input. It will

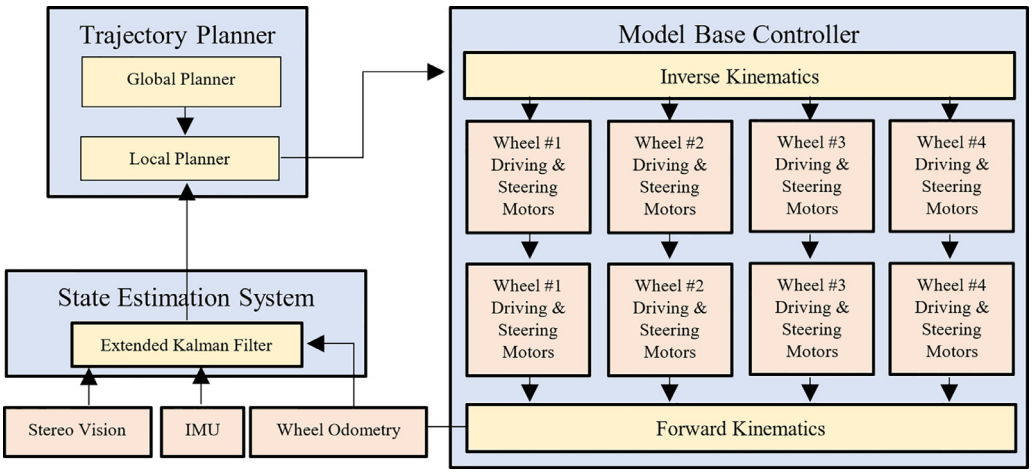

Fig. 11. Overview chart of control, state estimation and navigation system.

generate a path from the current location of the robot to the goal location by its global planner, it then generates the safe velocity commands sent to the robot base. This system will drive the robot close to the generated path toward the goal point while avoiding obstacles.

4. Experimental results and discussions

The section provides the experimental results and discussions aiming for the validation of MBC and the performance of ROS third party software used for the state estimation system. For each scenario, the input velocity command from a joystick has been recorded and then played back for the both simulated and physical MARIO. The results from each experiment were recorded for post processing and further analysis. Record/playback of the information is provided by *rosbag* tools in ROS.

4.1. Validation of the kinematic model and MBC

To validate the kinematic model and MBC, both real and simulated mobile robots perform a test trajectory by receiving the recorded velocity command from a joystick. Fig. 12 shows the graphs of linear and angular velocity feedbacks for both systems received from the MBC in form of odometry (through forward kinematics).

The measured velocities from both simulated and real models present the similar behaviours in phase and magnitude, however the lag can be observed in the measured data from the real model. This lag is due to the physical on-board speed controller response and the latency in the serial communication for transmitting the data from the main on-board computer to the physical DC motor controllers. The average percentage error for each axis velocity compared to the input velocity has been quantified and presented in Table 1. Although the error from the real model is almost the double of the simulation mainly due to the phase lag, but the experimental results is satisfactory for the performance validation of MBC.

Fig. 13 demonstrates the travelled trajectory of the real and simulated robots by receiving the same velocity command visualized by Rviz tool (ROS visualization tool). It was aimed to have all the possible combinations of v_x, v_y, and $\dot{\theta}$generated by the joystick for this experiment.

To present the performance of MBC inverse kinematics, a set of results are presented in Fig. 14 showing the performed steering control for each of the 4 servos during the experiment. Fig. 14(a) depicts the results from the real model, while the simulation results are provided in Fig. 14(b). The comparison between the real and simulation model is shown in Fig. 14(c) for the performed steering angle of servo1, as the phase lag is seen for the same reason described earlier. A cross correlation between the two signals quantifies 190 *ms* phase lag, where the simulation leads the real. The presented results validates the performance of the MBC inverse kinematics.

4.2. State estimation system test

In this test scenario, simulated MARIO has been commanded using a joystick. A ROS teleoperation package has been used for this purpose to generate the velocity command for MARIO using a PS3 joystick. The purpose of this experiment is to test the performance of the state estimation system with the information from modelled sensors including the stereo camera and IMU in the simulation environment. Visual odometry is achieved through feeding the left and right images from the simulated stereo camera into *viso2_ros* package. *Viso2_ros* package from *libviso* library [28], is a feature based visual odometry program that outputs odometry measurements from a monocular or stereo camera. Visual odometry, wheel odometry, and IMU data have been integrated by the *EKF* from *robot_localization* package.

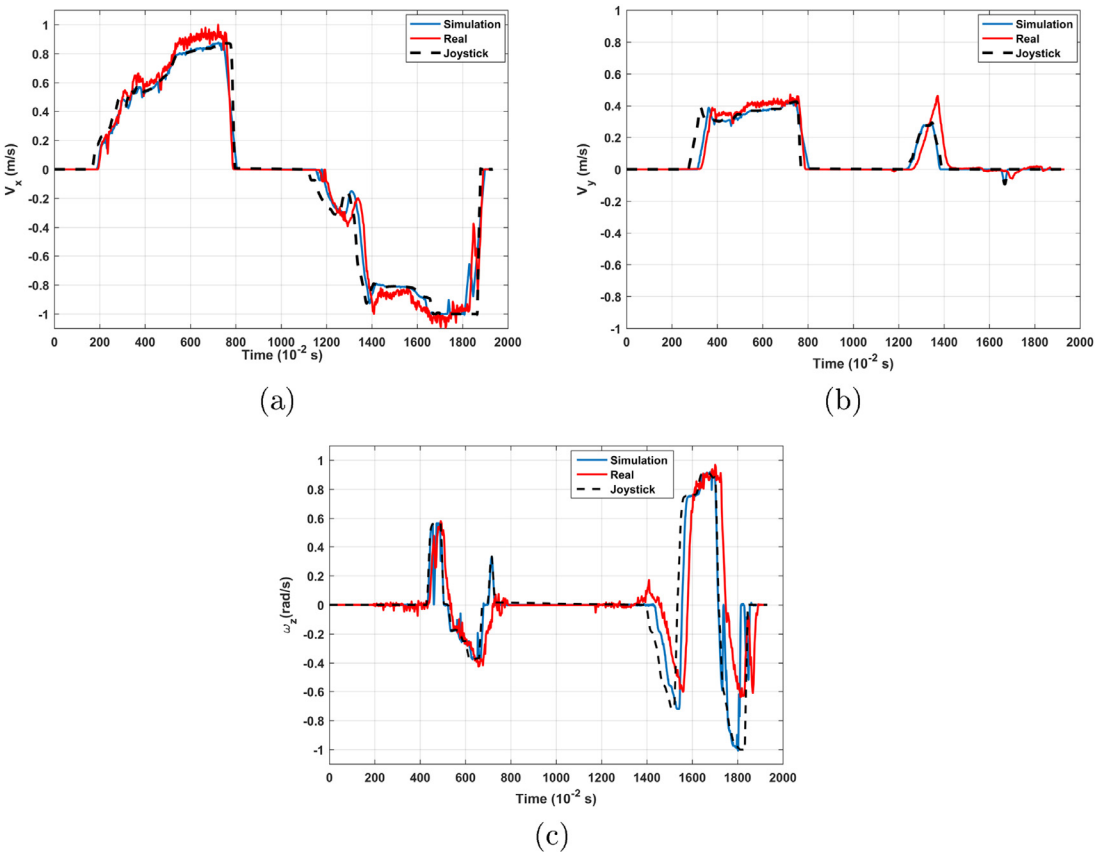

Fig. 12. Graphs of measured linear and angular velocity values from the simulation, real system and velocity command: (a) Robot frame linear velocity v_x, Robot frame linear velocity v_y, Robot frame angular velocity $\dot{\theta}$.

Table 1
Comparison of the errors of both simulation and real model for the input velocity commands.

	Simulation error %	Real error %
v_x	4.36	7.38
v_y	1.39	3.34
$\dot{\theta}$	5.99	13.29

Fig. 15 shows MARIO in the simulated vineyard environment in Gazebo (Fig. 15(b)), and the travelled trajectory visualized in RViz (Fig. 15(a)), where the blue, green, and red plots represent the wheel odometry, visual odometry, and filtered odometry respectively. The filtered odometry is obtained from the *EKF*, where it fuses the IMU rotational information with both wheel and visual odometry displacement information. Both Wheel and visual odometry suffer from the accumulated drift over time. This drift is more presented in turns and it can be seen in Fig. 15(b). The results from the state estimation system show a better estimation of the travelled trajectory by the fusion of wheel odometry, visual odometry, and IMU data so that the robot is back to the home position (starting point) with the least drift (0.3 m) compared to the wheel and visual odometry. Further analysis for comparison was not achievable, as the simulation of a GPS to provide the ground truth positioning information was not possible in Gazebo.

Fig. 16 shows all the ROS nodes and topics involved in this experiment generated by the ROS command *rqt_graph*.

5. Conclusion

This paper has presented the modelling and simulation of a 4WD4S mobile robotic platform, MARIO, using Gazebo simulator and ROS for the purpose of offline programming and system design validation. Gazebo simulator allowed modelling and simulation of different components of MARIO including physical model, sensing and control system. It also enabled the simulation of the world environment for robot operation. Interfacing Gazebo and ROS allowed access to a wide range of different robotic tools and software to be utilized in the simulated model. ROS provides the framework to develop the control, state estimation and navigation system.

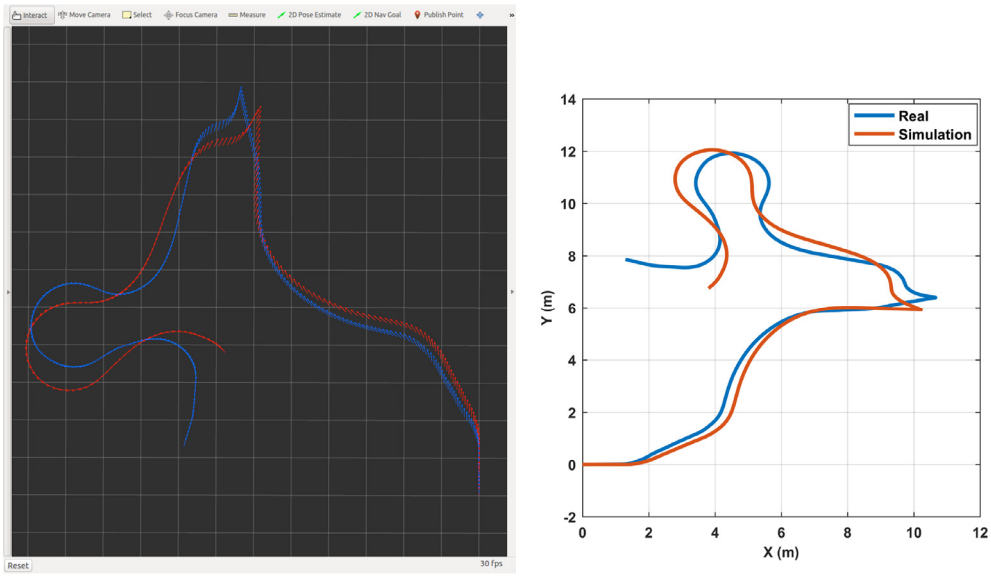

Fig. 13. Travelled trajectory of both simulated and real MARIO on the same input velocity command: a) Rviz visualization, b) Labeled plot with axis-units.

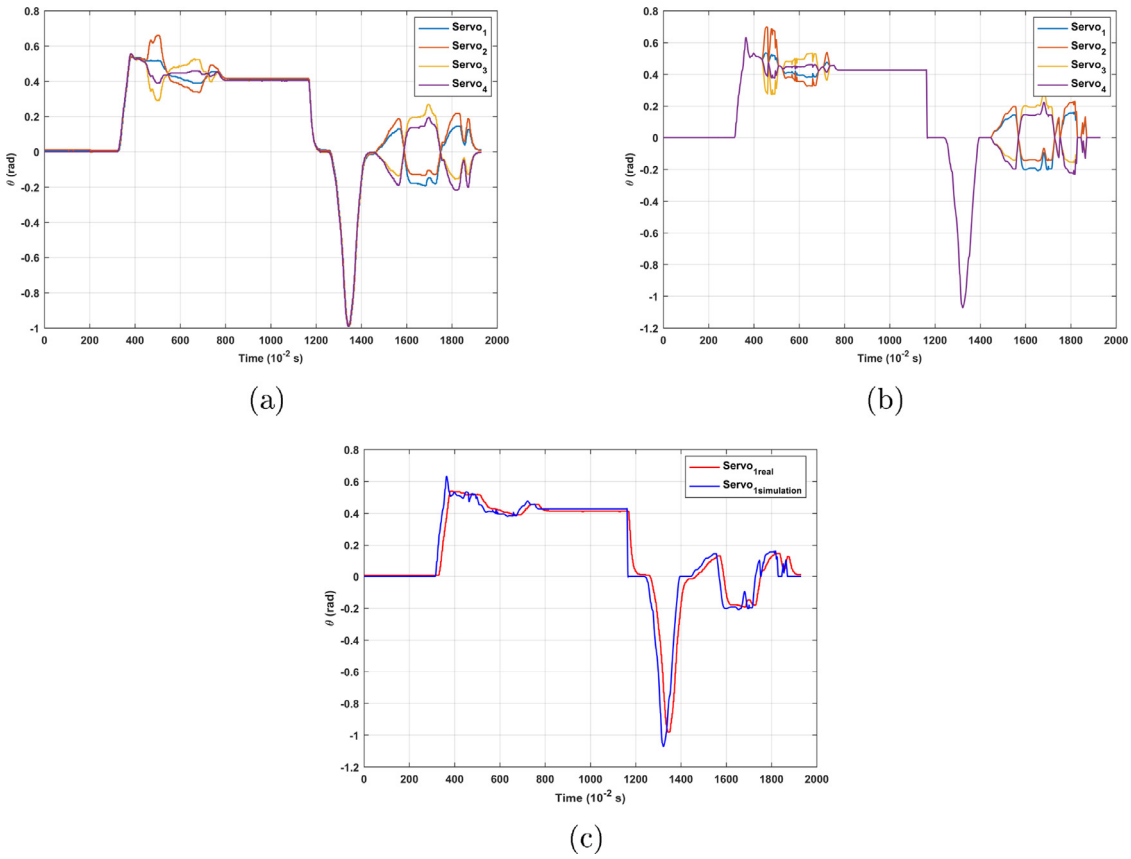

Fig. 14. MBC inverse kinematics performance analysis: (a) servos steering angles in real, (b) servos steering angles in simulation, (c) servo1 steering angles in both real and simulation.

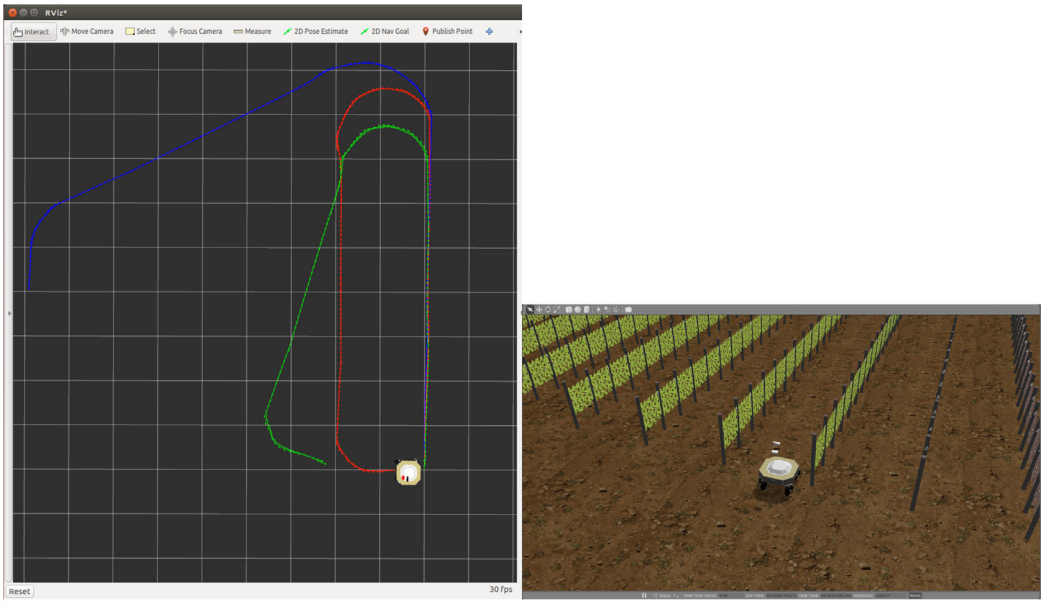

Fig. 15. Teleoperated navigation: (a) RViz visualization of odometry information for the travelled trajectory by MARIO, (b) The simulated vineyard environment for MARIO teleoperation in Gazebo. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

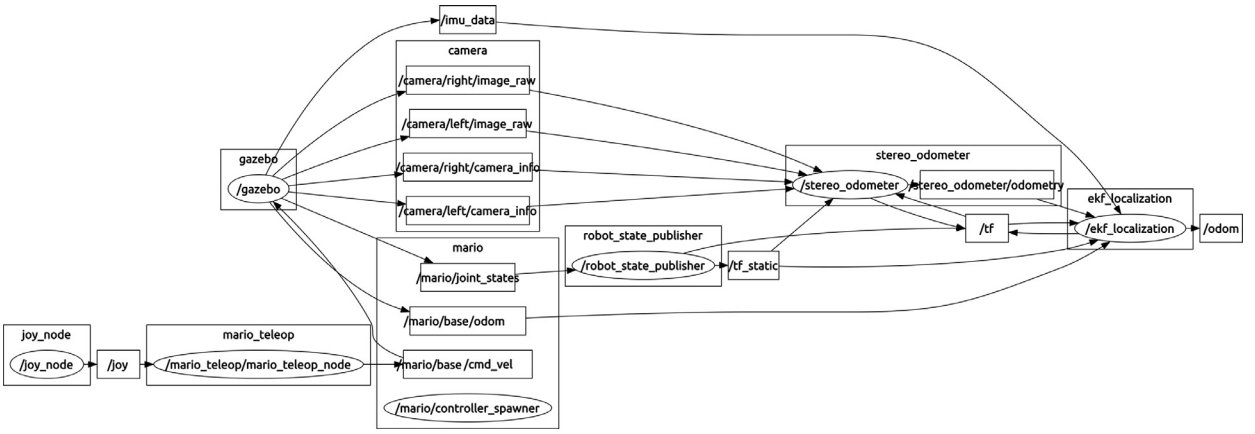

Fig. 16. All the active ROS nodes (ellipses) and message topics (rectangles) flowing between them obtained from rqt_graph.

The kinematic model base controller was tested and validated successfully using the experimental results achieved in the simulation and real environment. Furthermore, an example scenarios of Gazebo/ROS for teleoperation control of MARIO have been presented. Through this example, the performance of the simulated sensing system alongside the ROS state estimation system were tested and analysed. The approach provided in this paper allows successful development and test of MARIO and different implemented robotic software in a simulation environment before implementation on the real system. The presented approach can also be considered as a framework for other researchers for modelling and simulation of wheeled mobile robotic systems using Gazebo and ROS.

Acknowledgments

The authors acknowledge Department of Mechanical Engineering, University of Canterbury for funding and supporting the project; departmental technical staffs for their contribution and technical support to the project.

References

[1] M. Torres-Torriti, T. Arredondo, P. Castillo-Pizarro, Survey and comparative study of free simulation software for mobile robots, Robotica 34 (04) (2016) 791–822.
[2] S.A. Mehdi, K. Berns, Behavior-based search of human by an autonomous indoor mobile robot in simulation, Universal Access Inf. Soc. 13 (1) (2014) 45–58.
[3] A. Matta-Gómez, J. Del Cerro, A. Barrientos, Multi-robot data mapping simulation by using microsoft robotics developer studio, Simul. Modell. Pract. Theory 49 (2014) 305–319.
[4] S. Yamacli, H. Canbolat, Simulation of a scara robot with pd and learning controllers, Simul. Modell. Pract. Theory 16 (9) (2008) 1477–1487.
[5] M. Sarkar, S. Nandy, S. Vadali, S. Roy, S. Shome, Modelling and simulation of a robust energy efficient auv controller, Math. Comput. Simul. 121 (2016) 34–47.
[6] Z. Lu, C. Liu, J. Peng, H. Zhao, H. Wang, Motion simulation platform for a humanoid robot base on matlab, Intelligent Control and Automation (WCICA), 2014 11th World Congress on, IEEE, 2014, pp. 13–18.
[7] S. Rehman, S.J. Raza, A.P. Stegemann, K. Zeeck, R. Din, A. Llewellyn, L. Dio, M. Trznadel, Y.W. Seo, A.J. Chowriappa, et al., Simulation-based robot-assisted surgical training: a health economic evaluation, Int. J. Surg. 11 (9) (2013) 841–846.
[8] F. Caglar, S. Shekhar, A. Gokhale, S. Basu, T. Rafi, J. Kinnebrew, G. Biswas, Cloud-hosted simulation-as-a-service for high school stem education, Simul. Modell. Pract. Theory 58 (2015) 255–273.
[9] X. Tian, F. Gao, C. Qi, X. Chen, D. Zhang, External disturbance identification of a quadruped robot with parallel–serial leg structure, Int. J. Mech. Mater. Des. 12 (1) (2016) 109–120.
[10] E. Drumwright, J. Hsu, N. Koenig, D. Shell, Extending open dynamics engine for robotics simulation, International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Springer, 2010, pp. 38–50.
[11] P. Corke, Matlab toolboxes: robotics and vision for students and teachers, IEEE Rob. Autom. Mag. 14 (4) (2007).
[12] J. Jackson, Microsoft robotics studio: a technical introduction, IEEE Rob. Autom. Mag. 14 (4) (2007).
[13] O. Michel, Webots: symbiosis between virtual and real mobile robots, International Conference on Virtual Worlds, Springer, 1998, pp. 254–263.
[14] E. Rohmer, S.P. Singh, M. Freese, V-rep: a versatile and scalable robot simulation framework, Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE, 2013, pp. 1321–1326.
[15] G. Echeverria, N. Lassabe, A. Degroote, S. Lemaignan, Modular open robots simulation engine: morse, Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 46–51.
[16] N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, 3 IEEE, 2004, pp. 2149–2154.
[17] Y. Chen, G. De Luca, Viple: visual iot/robotics programming language environment for computer science education, Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International, IEEE, 2016, pp. 963–971.
[18] Y. Chen, Analyzing and visual programming internet of things and autonomous decentralized systems, Simul. Modell. Pract. Theory 65 (2016) 1–10.
[19] D.S. Michal, L. Etzkorn, A comparison of player/stage/gazebo and microsoft robotics developer studio, Proceedings of the 49th Annual Southeast Regional Conference, ACM, 2011, pp. 60–66.
[20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, Ros: an open-source robot operating system, ICRA Workshop on Open Source Software, 3 Kobe, 2009, p. 5.
[21] A. Ruckelshausen, P. Biber, M. Dorna, H. Gremmes, R. Klose, A. Linz, F. Rahe, R. Resch, M. Thiel, D. Trautz, et al., Bonirob–an autonomous field robot platform for individual plant phenotyping, Precis. Agric. 9 (841) (2009) 1.
[22] T. Linner, A. Shrikathiresan, M. Vetrenko, B. Ellmann, T. Bock, Modeling and operating robotic environent using gazebo/ros, in: Proceedings of the 28th International Symposium on Automation and Robotics in Construction (ISARC2011), pp. 957–962.
[23] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, O. Von Stryk, Comprehensive simulation of quadrotor uavs using ros and gazebo, International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Springer, 2012, pp. 400–411.
[24] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, J. Zhang, Manipulation task simulation using ros and gazebo, Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on, IEEE, 2014, pp. 2594–2598.
[25] M. Sharifi, M.S. Young, X. Chen, D. Clucas, C. Pretty, Mechatronic design and development of a non-holonomic omnidirectional mobile robot for automation of primary production, Cogent Eng. 3 (1) (2016) 1250431.
[26] W. Garage, Universal robot description format (urdf), http://www.ros.org/urdf/(2009).
[27] W. Kabsch, A solution for the best rotation to relate two sets of vectors, Acta Crystallogr. Sect. A 32 (5) (1976) 922–923.
[28] A. Geiger, J. Ziegler, C. Stiller, Stereoscan: dense 3d reconstruction in real-time, Intelligent Vehicles Symposium (IV), 2011 IEEE, IEEE, 2011, pp. 963–968.