

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

DIPLOMOVÁ PRÁCE

HIL simulace mobilního robotického systému

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Bc. Michael BOSÁK**
Osobní číslo: **A21N0110P**
Studijní program: **N3918 Aplikované vědy a informatika**
Studijní obor: **Kybernetika a řídicí technika**
Téma práce: **Hardware-in-the-loop simulace mobilního robotického systému**
Zadávací katedra: **Katedra kybernetiky**

Zásady pro vypracování

1. Nastudujte problematiku a význam vytváření Hardware-in-the-loop simulací, zaměřte se na simulace mobilních robotických systémů.
 2. Vyberte si konkrétního mobilního robota, sestavte jeho model a vytvořte jeho 3D vizualizaci. Obojí implementujte ve vhodných nástrojích a ověřte funkčnost.
 3. Vyberte si vhodné strategie řízení daného mobilního robota, implementujte je na vhodném HW a proveďte jejich testování v nastavení Hardware-in-the-loop spolu s vytvořeným modelem a vizualizací.
- Práci konzultujte se zástupci společnosti Siemens s.r.o: Tomáš Svoboda (svoboda.tomas@siemens.com), Florian Gramss (florian.gramss@siemens.com)

Rozsah diplomové práce: **40-50 stránek A4**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

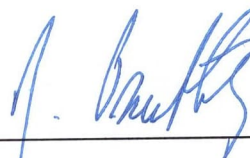
1. Franc Mihalič, Mitja Truntič and Alenka Hren. Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges. University of Maribor, 2022.
2. SIMATIC/SIMOTION virtual commissioning with „Hardware in the Loop“. Siemens AG, 2018.
URL https://cache.industry.siemens.com/dl/files/739/109758739/att_959577/v2/109758739_Documentation_HiL_en.pdf
3. Introduction to Hardware-in-the-Loop Simulation. Controllab.
URL <https://hil-simulation.com/images/stories/Documents/Introduction%20to%20Hardware-in-the-Loop%20Simulation.pdf>

Vedoucí diplomové práce: **Ing. Martin Goubaj, Ph.D.**
Katedra kybernetiky

Datum zadání diplomové práce: **1. října 2022**
Termín odevzdání diplomové práce: **22. května 2023**



Doc. Ing. Miloš Železný, Ph.D.
děkan



Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 22. 05. 2023

.....

Michael Bosák

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce panu Ing. Martinu Goubejovi, Ph.D. za odborné vedení, cenné rady a čas, který mi věnoval při konzultacích. Dále bych chtěl poděkovat panu Ing. Tomáši Svobodovi a dalším vedoucím výzkumného týmu Siemens SUP@ART za pomoc při řešení odborných problémů. Na závěr bych chtěl poděkovat své partnerce za poskytnutí skvělého zázemí při psaní této diplomové práce.

Abstrakt

Tato práce se zabývá vývojem HIL simulace AGV platformy IOT Bot založené na ROS2. Nejprve je sestaven kinematický model robotického systému počítající aktuální pozici a orientaci robota ve 2D prostoru. Tento model je implementován v simulační platformě SIMIT spolu s dynamickým modelem otáček kol robota. V prostředí NX MCD je implementován mechatronický model robota a algoritmus pro určení jeho polohy a orientace. Tento model slouží k porovnání výsledků kinematického modelu a pro 3D vizualizaci. Výstupy obou modelů jsou porovnány s pozicí a orientací reálného robota měřenou navigačním systémem. Tím je ověřena funkčnost obou modelů. Oba modely jsou použity ve dvou simulačních konceptech. Prvním konceptem je SIL simulace pro jejíž účely je vyvinut ROS2-SIMIT konektor a robot je v této simulaci ovládán pomocí ROS2 aplikací implementovaných na stejném zařízení jako simulace. Druhým konceptem je HIL simulace, kdy je simulace robota ovládána stejným HW a SW jako reálný robot. Komunikace mezi zařízením ovládajícím robota a zařízením, kde běží simulace, je implementována pomocí rozhraní OPC UA.

Klíčová slova

HIL simulace, SIL simulace, Virtuální uvádění do provozu, Matematické modelování, Digitální dvojče, Mobilní robot, ROS2, OPC UA

Abstract

This thesis is focused on the development of ROS2-based AGV platform IOT Bot's HIL simulation. In the beginning, the kinematic model of robotic system computing its current position and orientation in 2D space is built. This model and the dynamic model of robot's wheels' revolutions are implemented in SIMIT simulation platform. The robot's mechatronic model and the script determining its position and orientation are implemented in NX MCD. This model is used for the comparison of kinematic model's results and for 3D visualization. The outputs of both models are compared with the positions and orientations measured by navigational system and the models' functionality is verified. Both models are used in two simulation concepts. The first concept is the SIL simulation. The ROS2-SIMIT Bridge is developed for the purposes of this concept. The robot is controlled by the ROS2 nodes implemented in the same device as the robot's simulation. The second concept is the HIL simulation. The simulated robot is controlled by the same HW and SW as the real one. The OPC UA interface is used for the communication between the device controlling robot and the device where the robot's simulation runs.

Key words

HIL simulation, SIL simulation, Virtual commissioning, Mathematical modelling, Digital twin, Mobile robot, ROS2, OPC UA

Contents

1	Introduction	10
1.1	Mobile Robots	11
1.2	SIL Simulation Concept	13
1.3	HIL Simulation Concept	14
2	Theoretical Background	16
2.1	Position and Orientation in Robotics	16
2.2	Kinematic Models	18
2.2.1	Forward Kinematic Model	18
2.2.2	Inverse Kinematic Model	21
2.2.3	Redundant Robots	22
2.3	Dynamic Models of Linear Systems	22
2.4	Control Circuits and PID Controller	26
2.5	Linear Regression	29
3	Technologies Used	31
3.1	SIMIT	31
3.2	NX MCD	33
3.3	ROS2	34
4	AGV Platform IOT Bot	37
4.1	SIMATIC IOT2050	37
4.2	Motors and Extension-Shield	38
4.3	Power Supply	39
4.4	ROS2 Nodes	39
5	IOT Bot's models	41
5.1	Mathematical Kinematic Model	41
5.2	Kinematic model's parameter K_ω Identification	45
5.2.1	Identification using Unsteady RPM values	45
5.2.2	Identification using Steady RPM values	53
5.3	Wheel's RPM Control Loop's Dynamic Model	60
5.4	Models' Implementation using SIMIT	61
5.5	NX MCD Mechatronic Model	66
5.6	Kinematic Models' Functionality Verification	68

5.6.1	IOT Bot's Single Particular Movements	71
5.6.2	IOT Bot's Combination of Movements	76
6	SIL Simulation	80
6.1	SW Setup	80
6.2	ROS2-SIMIT Bridge	83
6.3	SIL Simulation Testing	87
7	HIL Simulation	89
7.1	HW and SW Setup	89
7.2	OPC UA-SIMIT Bridge	91
7.3	HIL Simulation Testing	91
8	Conclusion and Outlook	93
	Bibliography	95

1 Introduction

Nowadays, in the middle of the new industrial revolution known as Industry 4.0, the digitalization of the industry plays an important role. The nine pillars of the Industry 4.0 are (1) Big Data and Analytics, (2) Autonomous Robots, (3) Simulation and Digital Twin, (4) Industrial Internet of Things, (5) Augmented Reality, (6) Additive Manufacturing, (7) Cybersecurity, (8) Cloud Computing and (9) Horizontal and Vertical System Integration [1].

There are two main aims of the digitalization in the industry. The first aim is to make the production more efficient. The reduction of final product's manufacturing cost and human workers amount needed is required. The second aim is more sustainable and environmental friendly manufacturing. Since the number of people on our planet is growing, more products need to be manufactured. If emissions related to that wouldn't be lowered, it could have negative environmental consequences.

The two aims mentioned above could be summarized into term Smart Manufacturing. Another important term is Flexible Manufacturing. Apart from the manufacturing efficiency and sustainability is also important its ability to adapt to changes in terms of the quantity of manufactured products and its type.

This thesis goes deeper in two pillars of the Industry 4.0 - autonomous robots and simulation. In the following section, the mobile robots and their autonomous driving setup options are introduced. These robots play significant role in the flexible manufacturing process because their equipment could be easily changed if the interaction with different product is needed during the manufacturing. Moreover, with the fleet management doesn't matter how many mobile robots are supposed to be controlled. The quantity of manufactured products can be very smoothly increased or decreased while the manufacturing process is still identical.

Two simulation methods, which are currently widely used in the industry to perform a testing of proposed control solutions for different typed of systems, are introduced below. These two simulation concepts are developed in the scope of this thesis for the simulation of certain mobile robotic plat-

form IoT Bot. The Siemens simulation tools are described and used for the mentioned simulations' development. Furthermore, a basic overview of theoretical background related to mathematical modeling of the mobile robots is given in the second chapter.

1.1 Mobile Robots

A mobile robot is an automatic machine capable of transporting itself from one place to another. The robot is equipped with wheels or legs which are used for its transportation. The wheeled robots are convenient for flat surfaces such as industrial manufacturing halls. Mobile robots equipped with legs can operate in rough terrain with the necessity of overcoming difficult obstacles and in places where the ordinary wheeled robots would fail. See the examples of these two kinds of robots in the figure 1.1.

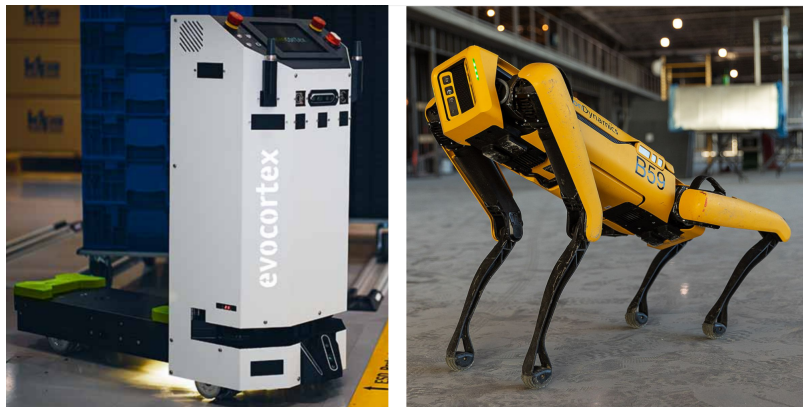


Figure 1.1: Example of the wheeled robot (Evocortex Evo Carrier) [2] and the robot equipped with legs (Boston Dynamics Spot) [3]

The mobile robots are mainly used in the industrial field. Their main task is to carry the material between particular manufacturing places and support the human operators. The path of the robot's movement can be controlled by human. One can still see this procedure in the industry. However, the mobile robots are nowadays mostly programmed to be able to move autonomously. This kind of machine is known as Automated Guided Vehicle (AGV) or Autonomous Mobile Robot (AMR).

There is a significant difference between AGV and AMR. The AGV is capable of following only the path which is marked on the ground by magnetic

tape or by colored line e.g. Certain sensors for the line detection such as light sensor or magnetic sensor are mounted to the AGV in order to detect the line [4].

In comparison with AGV, AMR does not need to have any marked path to deliver the material. It is equipped with tools for scanning the working area such as lidar. The AMR is aware of the obstacles present in the field. It is autonomously able to plan the path to avoid the collision [4]. The principle of this difference is shown in the figure 1.2.

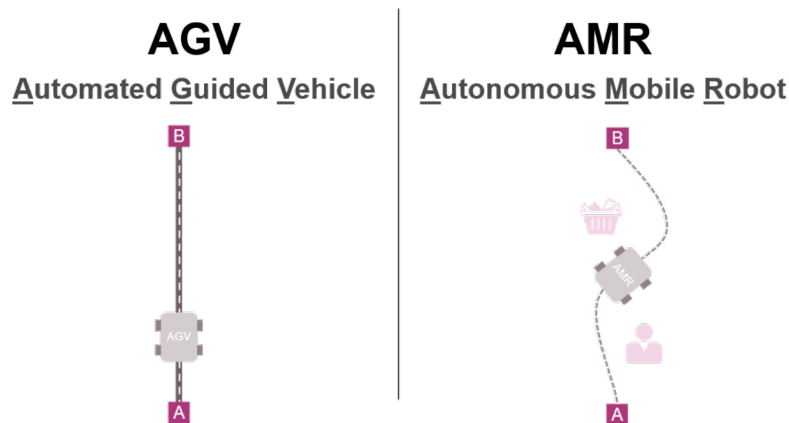


Figure 1.2: The difference between AMR and AGV [4]

Apart from industry, the mobile robots are also used in space exploration. The role of mobile robots there is significant because they can substitute people in certain places. For instance, NASA has sent several rovers to Mars since Mars is too dangerous to be visited by humans.

The mobile robots can save human lives twice in that case. Firstly, they allow to get more information about the planet without the risk of losing human lives. Secondly, due to obtaining information about the planet, it will be safer to settle there for people in the future.

Some people have mobile robot directly in the house or in the garden. The automatic vacuum cleaners or automatic grass cutters are nowadays a common part of the home equipment. Finally, the mobile robot could be used in a healthcare. An automatic wheelchair with robotic arm could be very helpful for older people to support their movement and objects grabbing.

For specific use cases, the mobile robot can be equipped with some serial or parallel manipulator. The serial manipulator is typically the robotic arm such as UR5. One of examples of the mobile robot with parallel manipulator is an Omnid mocobot shown in the Figure 1.3. The use case with several of these robots is also visualized in that figure. It is also the case of a human-robot collaboration. Several robots can collaborate with humans and are helpful with placing the payload to specific position [5].

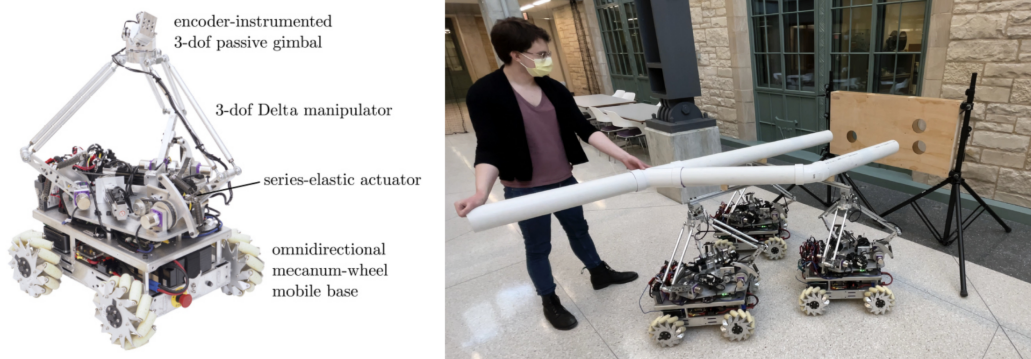


Figure 1.3: Omnid mocobot description and use case of human-robot collaboration during the manipulation with payload [5]

1.2 SIL Simulation Concept

Software in the Loop (SIL) simulation is a concept used for development and testing of control programs for various types of systems. Firstly, a model of the system is created and implemented in an appropriate simulation software such as Matlab/Simulink or SIMIT. Then, the control strategies are proposed and implemented in the loop with the created model. The whole process of the control program development and testing before its final deployment on real system is generally known as virtual commissioning [6].

In the SIL concept, the virtual commissioning is fully software based. It means that both controller and system's model are closed in the loop where the data exchange is not covered by physical connectors and communication protocols but by coupling of signals or using shared memory. Usually the whole loop is implemented in the same computer or processor.

The advantage of this concept is simplicity since only one device is needed in most cases. It is a good concept for initial evaluation of proposed control solution. However, before the solution is deployed on real system it is suitable to test it in a more complex way using a concept described in the following chapter e.g.

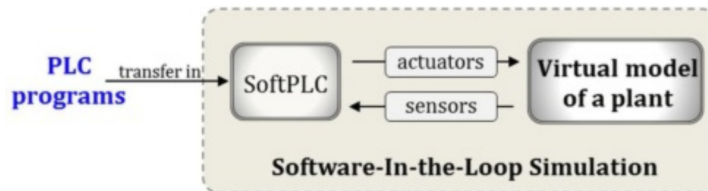


Figure 1.4: Software in the Loop (SIL) simulation concept [6]

1.3 HIL Simulation Concept

Hardware in the Loop (HIL) simulation is more sophisticated concept used for the virtual commissioning. The main idea of HIL is replacing the real system (plant) by a mathematical model implemented in the hardware which allows communication with the device where the control algorithm is implemented. This is the biggest advantage of HIL concept in comparison with the SIL concept.

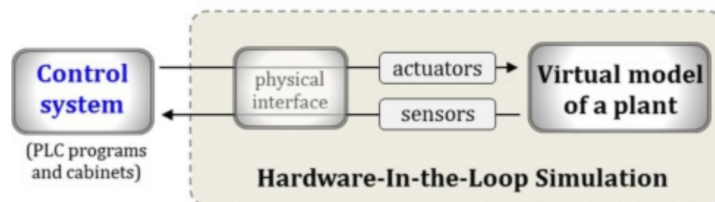


Figure 1.5: Hardware in the Loop (HIL) simulation concept [6]

As mentioned above, with the SIL concept the real communication between modeled system and developing controller cannot be simulated. That could lead to a failure of the control algorithm after its implementation on real control hardware. This failure could be caused by a noise on the wires or by different communication rate. Ideally, when the development of the control algorithm is finished, the real system or machine can be connected to the control loop instead of the HIL simulator [7]. This procedure is demonstrated in the figure 1.6.

The HIL concept brings plenty of advantages during the commissioning of the real device or machine. In that case, the virtual commissioning is as similar to the real one as possible. Moreover, the risks of the real commissioning such as a machine parts breakage and the efforts like energy or traveling costs are reduced.

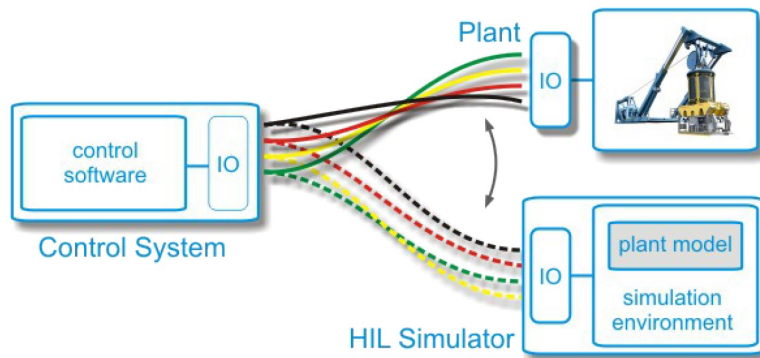


Figure 1.6: Analogy of two connections: control system - plant simulator and control system - plant [7]

The HIL concept is used in both process and machine automation. In other words, the HIL simulator could be developed for robotic arm as well as for chemical process e.g. This concept is widely used for instance in automotive industry. The virtual model of the car communicates in that case with the real components such as gearbox or peripherals like radars or cameras.

2 Theoretical Background

2.1 Position and Orientation in Robotics

Let's assume a three-dimensional coordinate system $F_1 = \{\mathbf{O}_1, \mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1\}$ and $F_2 = \{\mathbf{O}_2, \mathbf{x}_2, \mathbf{y}_2, \mathbf{z}_2\}$. \mathbf{O}_1 and \mathbf{O}_2 is the center of the coordinate system F_1 and F_2 , respectively. See it in the Figure 2.1.

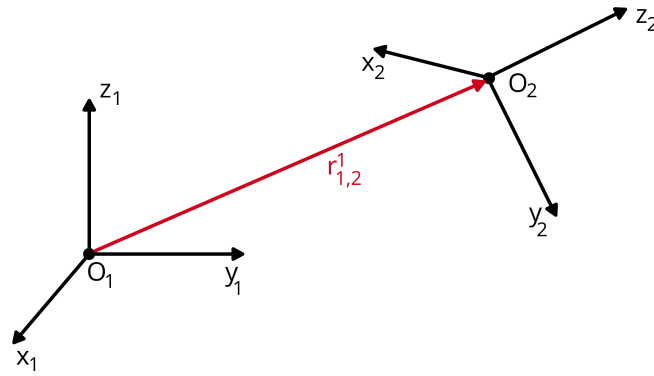


Figure 2.1: Two coordinate systems and their mutual translation

Based on the Figure 2.1 the mutual translation vector between coordinate system F_1 and F_2 expressed in F_1 could be obtained as $\mathbf{r}_{1,2}^1 = \mathbf{O}_2^1 - \mathbf{O}_1^1$ [8]. The center of the coordinate system expressed in its own coordinates always equals to zero. Thus, this formula can be rewritten as $\mathbf{r}_{1,2}^1 = \mathbf{O}_2^1$. The reverse translation is defined similarly as $\mathbf{r}_{2,1}^2 = \mathbf{O}_1^2$.

As a definition of mutual rotation of these two coordinate systems, the rotational matrix $\mathbf{R}_2^1 = [\mathbf{x}_2^1, \mathbf{y}_2^1, \mathbf{z}_2^1]$ is used. Each column of this matrix represents a unit direction vector of corresponding F_2 axis expressed in F_1 [8].

Due to the fact that the columns of this matrix represent the vectors which are mutually perpendicular, the matrix has following important mathematical feature [8]:

$$(\mathbf{R}_2^1)^T \cdot \mathbf{R}_2^1 = \mathbf{I}, \quad (2.1)$$

where \mathbf{I} is a unit three-dimensional matrix.

This mathematical feature is really useful for the inverse mutual rotational matrix definition. Then, it could be obtained as follows [8]:

$$(\mathbf{R}_2^1)^{-1} = (\mathbf{R}_2^1)^T = \mathbf{R}_1^2 \quad (2.2)$$

In practice, three basic types of rotation are defined. Each rotation is related to one axis. The most intuitive is the rotation around z-axis by an angle γ . It is obtained as [8]:

$$\mathbf{R}_x(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

Similarly, the rotation around x-axis by an angle α and the rotation around y-axis by an angle β could be obtained as [8]:

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \quad \mathbf{R}_x(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad (2.4)$$

To get a compact description of the F_2 position and orientation against F_1 both translation vector and rotational matrix could be written in homogeneous transformation matrix which generally looks as follows [8]:

$$\mathbf{T}_2^1 = \begin{bmatrix} & & & \\ & \mathbf{R}_2^1 & & \mathbf{r}_{1,2}^1 \\ \hline 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

If third three-dimensional coordinate system F_3 is assumed and the transformation from F_1 to F_3 determination is needed, then following formula could be used [8]:

$$\mathbf{T}_3^1 = \mathbf{T}_2^1 \cdot \mathbf{T}_3^2 \quad (2.6)$$

In other words, it's about particular homogenous matrices folding in order to get a final one.

2.2 Kinematic Models

2.2.1 Forward Kinematic Model

A forward kinematic model calculates a position and orientation of a single geometric point on the robot in the three-dimensional space. The values of position and orientation are calculated based on the current setup of the robot actuators and its geometric parameters [8].

For better understanding, an example of the forward kinematic model of a planar robotic arm operates in two-dimensional space is given. Since only two-dimensional space is considered, the coordinates related to the third dimension will always equal to zero. See drawing of the considered planar robot in the Figure 2.2.

The inputs of the forward kinematic model are in this case the angles of rotations of the joints Θ_1 , Θ_2 , Θ_3 (current setup of the robot actuators) and the lengths of the arms a_1 , a_2 , a_3 (geometric parameters). Then, the outputs are the coordinates of the point O_3 (end-effector) expressed against the point of the robot base O_0 (x_3^0 and y_3^0) and the orientation of the final effector represented by angle Φ .

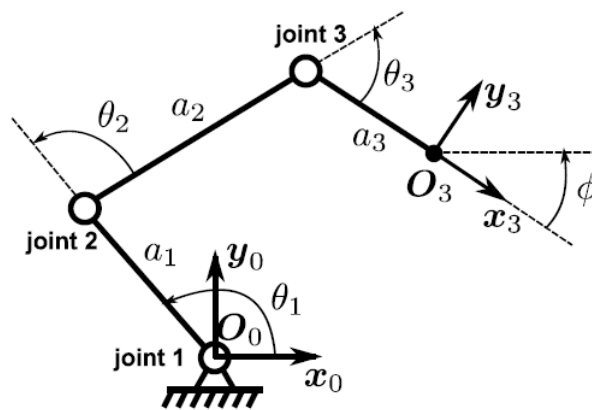


Figure 2.2: Planar robotic arm drawing [8]

The mathematical expression of the forward kinematic model described above includes three following equations [8]:

$$\begin{aligned}
 (1) \quad x_3^0 &= a_3 \cos(\Theta_1 + \Theta_2 + \Theta_3) + a_2 \cos(\Theta_1 + \Theta_2) + a_1 \cos(\Theta_1) \\
 (2) \quad y_3^0 &= a_3 \sin(\Theta_1 + \Theta_2 + \Theta_3) + a_2 \sin(\Theta_1 + \Theta_2) + a_1 \sin(\Theta_1) \\
 (3) \quad \Phi &= \Theta_1 + \Theta_2 + \Theta_3
 \end{aligned} \tag{2.7}$$

If the description of the relation between actuators angular velocity and the velocities of the output coordinates is needed, it is possible to derivate the three equations written above with respect to time. Then a forward instantaneous kinematic model is obtained [8]:

$$\begin{aligned}
 (1) \quad \dot{x}_3^0 &= \frac{d}{dt} \{a_3 \cos(\Theta_1 + \Theta_2 + \Theta_3) + a_2 \cos(\Theta_1 + \Theta_2) + a_1 \cos(\Theta_1)\} \\
 (2) \quad \dot{y}_3^0 &= \frac{d}{dt} \{a_3 \sin(\Theta_1 + \Theta_2 + \Theta_3) + a_2 \sin(\Theta_1 + \Theta_2) + a_1 \sin(\Theta_1)\} \\
 (3) \quad \dot{\Phi} &= \frac{d}{dt} \{\Theta_1 + \Theta_2 + \Theta_3\}
 \end{aligned} \tag{2.8}$$

For the description of the mobile robot's kinematics the same mathematical methods as are described above could be used. The difference is that the mobile robot moves in the two-dimensional space which is considered as infinite from a mathematical point of view. In this case it makes sense to deal with the description of the model which calculates the relation between its actuators velocities and the speed of the mobile robot itself. It means that the forward instantaneous kinematic model is usually used.

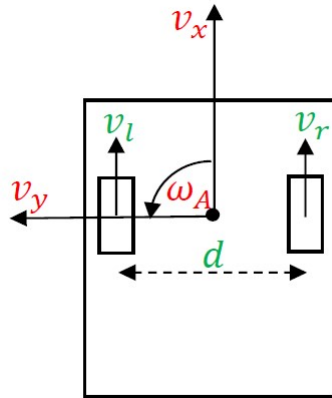


Figure 2.3: Differential drive mobile robot drawing

Let's give an example of the simplest mobile robot - differential drive. See the drawing of this kind of mobile robot in the Figure 2.3. The forward

instantaneous kinematic model of the kind of robot drawn in the Figure 2.3 is very simple. It describes the relation between wheels' velocities (v_l for left wheel and v_r for right wheel) and the robot velocities (linear velocity v_x and angular velocity ω_a):

$$\begin{aligned} v_x &= \frac{1}{2} (v_r + v_l), \\ \omega_A &= \frac{1}{d} (v_r - v_l) \end{aligned} \quad (2.9)$$

The value d is the distance between the wheels' centers. It should be mentioned that for simplicity something like linear velocity of the wheel (value v_l and v_r) is considered in the given model (equation 2.9). This value does not make sense from the physical point of view. The simple rolling of the wheel, after the initial surface friction overcoming, is considered and the following expressions are valid:

$$\begin{aligned} v_r &= r \omega_r, \\ v_l &= r \omega_l, \end{aligned} \quad (2.10)$$

where r is the radius of the wheel,

and ω_r and ω_l is the angular velocity of the right and left wheel, respectively.

In case of mobile robots basically two kinds of velocities are considered. The first is a linear velocity and the second is an angular one. For the position and orientation of the mobile robot in two-dimensional space determination, its forward instantaneous kinematic model should necessarily contain the equations describing the straightforward (x-directional), lateral (y-directional), rotational (around z-axis) motion and the relations between them. In other words, all three degrees of freedom of the robot in two-dimensional space should be considered in overall motion description. If this condition is met, the forward instantaneous kinematic model's equations could be simply integrated with respect to time to get a position and orientation of the robot in the two-dimensional space.

In the model described in equation 2.9 the mentioned condition is not met. Let's integrate the equations contained in this model with respect to time:

$$\begin{aligned} s_x(t) &= \int_0^t v_x(\tau) d\tau = \frac{1}{2} r \int_0^t [\omega_r(\tau) + \omega_l(\tau)] d\tau, \\ \varphi_A &= \int_0^t \omega_A(\tau) d\tau = \frac{1}{d} r \int_0^t [\omega_r(\tau) - \omega_l(\tau)] d\tau, \end{aligned} \quad (2.11)$$

where t symbolize a current time moment and τ is a temporary integration value.

Although, the value φ_A accumulates the rotational movements during the time and at the time moment it provides a current orientation of the robot in two-dimensional space, the position of the robot cannot be determined using this model. The value $s_x(t)$ accumulates the forward movements without considering the robot's orientation. In other words, the lateral motion cannot be distinguished from the straightforward motion. In the scope of this thesis, the forward kinematic model describing fully the position and orientation of the mobile robot is derived and implemented.

2.2.2 Inverse Kinematic Model

An inverse kinematic model solves the opposite issue than the forward one. It is used for robot control, because based on the required position and orientation of robot end-effector (model's inputs), the setup of the robot's actuators can be calculated (model's outputs) [8].

For example, the inverse kinematic model of the planar robotic arm shown in the Figure 2.2 can be derived analytic way considering its forward kinematic model (equation 2.7) and some geometrical dependencies. Similarly, based on the forward instantaneous kinematic model of differential drive mobile robot (equation 2.9) drawn in the Figure 2.3, its inverse instantaneous kinematic model could be obtained. It looks as follows:

$$\begin{aligned} v_l &= v_x - \omega_A \frac{d}{2}, \\ v_r &= v_x + \omega_A \frac{d}{2} \end{aligned} \quad (2.12)$$

In these two cases it is not tough to derive the inverse (instantaneous) kinematic model. However, in comparison with the forward kinematic model derivation which is always derivable analytic way, it could be a very complicated issue. In some cases, there is no analytical solution. It can also happen that the inverse kinematic model cannot be found at all.

As mentioned, the inverse kinematic model is used for the robot control. For this reason, it is one of the most important mathematical knowledge for the industrial purposes. Since it is always tricky to derive this model by hand, the software tool used for the virtual commissioning of the robots came with implemented functions for inverse kinematics' automatic calculation. NX MCD as a one of these tools is introduced below.

2.2.3 Redundant Robots

A redundant robot is a robot which has fewer degrees of freedom number of its end-effector than the number of actuators (joints). The consequence of robot's redundancy is that the task of the robot could be accomplished more than one way [8]. In other words, if the request for the robotic arm would be to reach certain coordinates, than more setups of the robot's actuators could be found.

The planar robotic arm drawn in the figure 2.2 could be redundant if only the position of the end-effector is considered. In this case the end-effector of considered robot has two degrees of freedom (x_3^0 and y_3^0) while the robot always has three joints (Θ_1 , Θ_2 and Θ_3) controlled by actuators. Otherwise, if the orientation Φ is also considered as a degree of freedom, both degrees of freedom number and the number of joints controlled by actuators are equal to three, and the robot is considered as non-redundant.

Similarly, it could be thought about the mobile robots in general. They move in the two-dimensional space and there are infinite options how to get from one place to another. However, each option causes a different orientation angle of the mobile robot in the final destination.

2.3 Dynamic Models of Linear Systems

There are several options how to model a dynamic behavior of a linear system. Let's demonstrate some of these options on a simple dynamic system example - RLC electric circuit. This circuit contains the three passive elementary electronic components - resistor, coil and capacitor. See it in the figure 2.4.

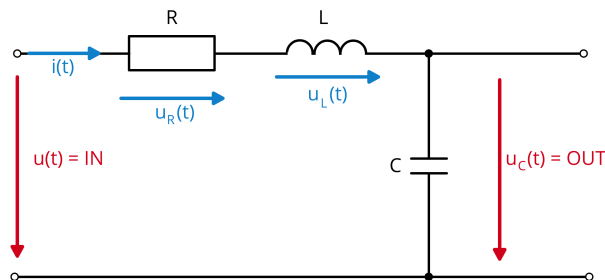


Figure 2.4: RLC electric circuit scheme

Using a Kirchhoff law and the knowledge of the elementary electronic components' behavior, a differential equation (second-order and linear) as a first option for the general dynamic model's mathematical description, could be derived as follows [9]:

$$\begin{aligned} u_R(t) + u_L(t) + u_C(t) &= u(t) \\ R i(t) + L \frac{di(t)}{dt} + \frac{1}{C} \int i(t) dt &= u(t), \end{aligned} \quad (2.13)$$

where R (Ohm), L (Henry) and C (Farad) are the constant parameters of passive electronic components, $u(t)$ is input voltage (also system input), $u_R(t)$ is a voltage on the resistor, $u_L(t)$ is a voltage on the coil, $u_C(t)$ is a voltage on the capacitor (required system output) and $i(t)$ is a circuit's current. Moreover, zero initial conditions are assumed.

From the differential equation the state-space model, as a second introduced option for the dynamic system behavior's modeling, could be obtained. The main idea of the state-space model is the state variables of the system determination. Moreover, the value supposed to be system output is determined. Then, if it is needed, the differential equation is adjusted, and the determined state variables are installed inside. That leads to obtaining of multiple lower order differential equations [9].

There is no exact procedure how to choose the state variables. However, it makes sense to choose some physical interpretable and measurable values such as position or electric voltage in this case [9].

Let's determine the state variables $x_1(t)$, $x_2(t)$ and the required system output $y(t)$ following way:

$$\begin{aligned} x_1(t) &= i(t) \\ x_2(t) &= u_C(t) = \frac{1}{C} \int i(t) dt \\ y(t) &= x_2(t) \end{aligned} \quad (2.14)$$

After substitution of the state variables into equation 2.13 the state-space model is obtained. It includes

- state equations

$$\begin{aligned}\frac{dx_1(t)}{dt} &= -\frac{R}{L}x_1(t) - \frac{1}{L}x_2(t) + \frac{1}{L}u(t) \\ \frac{dx_2(t)}{dt} &= \frac{1}{L}x_1(t)\end{aligned}\quad (2.15)$$

- output equation

$$y(t) = x_2(t) \quad (2.16)$$

For further work with the obtained state-space model is convenient to rewrite it to the following matrix form:

$$\begin{aligned}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} -\frac{R}{L} & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\end{aligned}\quad (2.17)$$

Then, the state-space model can be defined by three matrices:

$$\mathbf{A} = \begin{bmatrix} -\frac{R}{L} & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad (2.18)$$

These three matrices can be used for the derivation of a transfer function as a last introduced option how to describe the dynamic system. General equation for the transfer function's calculation from the state-space matrices is defined as follows [9]:

$$F(s) = \mathbf{C} (s\mathbf{I} - \mathbf{A})^{-1} \mathbf{B}, \quad (2.19)$$

where \mathbf{I} is a unit two-dimensional matrix.

Let's use the equation 2.18 and obtain the transfer function of the electric circuit described in the Figure 2.4:

$$F(s) = \frac{1}{CLs^2 + RCs + 1} = \frac{\frac{1}{CL}}{s^2 + \frac{R}{L}s + \frac{1}{CL}} \quad (2.20)$$

The obtained transfer function's shape corresponds with the shape of the second-order oscillating system's transfer function [9]

$$F(s) = \frac{\omega^2}{s^2 + 2 \xi \omega s + \omega^2}, \quad (2.21)$$

where $\omega = 2\pi f$ describes the oscillations' frequency and ξ is the oscillations' dumping constant.

The denominator's roots of the transfer function given in equation 2.21 are generally complex numbers. In special case if the roots are real numbers, could be this transfer function dividable into two transfer functions of aperiodic first-order system. This procedure is used in this thesis for the mobile robot's motors dynamic behavior modeling. The second-order transfer function is divided into two first-order transfer functions due to implementation reasons.

The typical example of the aperiodic first-order system is RC electric circuit shown in the figure 2.5. The transfer function of this kind of system generally looks like follows [9]:

$$F(s) = \frac{1}{\tau s + 1}, \quad (2.22)$$

where τ is a time constant and in case of RC electronic circuit is obtained as $\tau = RC$.

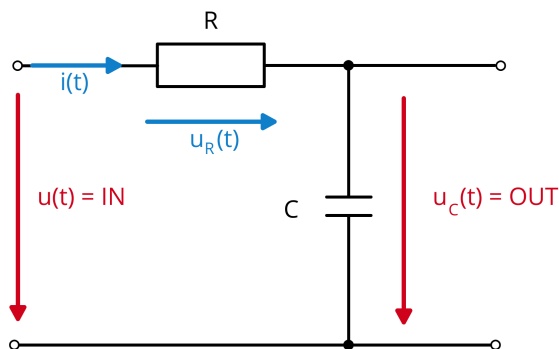


Figure 2.5: RC electric circuit scheme

The transfer function $F(s)$ is generally defined like follows [9]:

$$F(s) = \frac{L\{y(t)\}}{L\{u(t)\}} = \frac{Y(s)}{U(s)}, \quad (2.23)$$

where $u(t)$ is the system input, $y(t)$ is the system output, and $U(s)$ and $Y(s)$ is the Laplace image of the system input and output, respectively. Moreover, the zero initial conditions are anticipated.

Operator L generally symbolized a Laplace transform of a function of time $f(t)$ which is defined as [9]

$$L\{f(t)\} = \int_0^{\infty} f(t) e^{-st} dt, \quad (2.24)$$

where $s = \sigma + j\omega$ is a complex value and the function $f(t)$ should meet the following conditions:

- $f(t)$ is locally integrable on $[0, \infty)$
- $f(t) = 0$ for $t < 0$
- $f(t)$ is of exponential order ($\int_0^{\infty} f(t) e^{-\sigma t} dt$ for $\sigma > 0$)

Each mentioned system above has a different step response - the response for a unit step input signal. See their typical shapes in the figure 2.6.

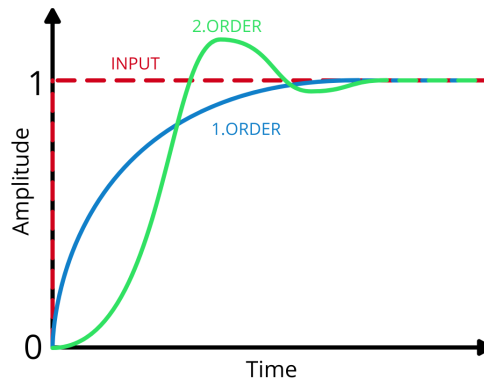


Figure 2.6: Typical shapes of the first-order and second-order systems' step responses

2.4 Control Circuits and PID Controller

There are two main concepts to control any input-output process (system) in the control theory. These two methods are related to the two elementary control circuits - feedforward and feedback control circuit. See their schemes in the figure 2.7.

The control circuit in general includes following values:

- reference value $r(t)$ - intended control circuit output

- process value $y(t)$ - actual control circuit output
- action value $u(t)$ - controller output
- control error $e(t) = r(t) - y(t)$ (only in case of feedback control circuit)

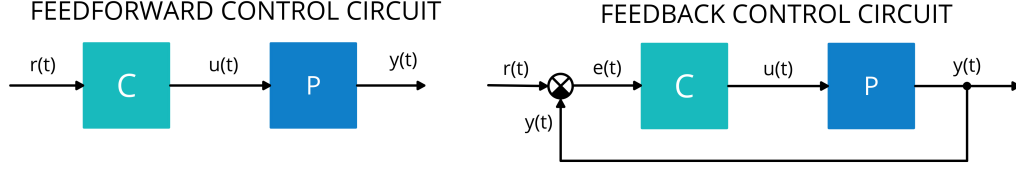


Figure 2.7: feedforward and feedback control circuit both including Controller (C) and Process (P)

Both mentioned concepts are used in this thesis for the SIL and HIL simulation purposes. The first concept is implemented for the simulation of ROS2 controller, which controls the robot in feedforward circuit using the inverse instantaneous kinematic model. The second concept is used for the simulation of robot's inner motor control loops, where the feedback control of the motor's revolutions is implemented. PID controller as the most common industrial controller is used in these loops.

The action value of PID controller includes three elements: Proportional (P), Integrative (I) and Derivative (D). It is expressed by following formula [10]:

$$u(t) = K \left[e(t) + \frac{1}{T_I} \int_0^t e(\tau) d\tau + T_D \frac{de(t)}{dt} \right], \quad (2.25)$$

where $K = K_P$ is a proportional gain, T_I is an international time constant and T_D is a derivative time constant. Moreover, the integrative and derivative gain can be obtained as follows: $K_I = \frac{K}{T_I}$, $K_D = K T_D$. N is derivative filter's time constant's coefficient and out of the general engineers' experiences it can be obtained as $N = 3 \div 10$.

Each of the three elements influences the control value different way [10]:

- Element P: represents a static part of the controller, higher K_P causes higher regulation precision, low-frequency errors' suppression and faster controller's response, but higher over-regulation
- Element I: essential part for regulation's precision - only with that a zero control error can be reached, slows down the controller's re-

sponse, lower T_I causes higher over-regulation

- Element D: speeds up the controller's response, higher T_D causes lower over-regulation

In practice, the PID controller is mostly implemented in discrete computing device such as Programmable Logic Controller (PLC) or any other kind of discrete device like microprocessor. For this implementation, the derivation is supplied by difference calculated from previous and current control error and instead of integral a simple sum is used.

In this thesis the PID controller is simulated in SIMIT where a block for integration is used, and the derivation is supplied using mentioned difference. There is also a low-pass filter implemented for the derivative noise lowering. The low-pass filter is generally obtained by following transfer function [10]:

$$F(s) = \frac{K}{\tau s + 1}, \quad (2.26)$$

where K is a filter's gain and τ is a time constant determining the highest passed frequency. Out of the general engineers' experiences it can be obtained as $\tau \approx \frac{T_D}{N} = \frac{K_D}{N \cdot K_P}$.

In this thesis, the discrete version of low-pass filter is implemented using its difference equation:

$$y(k+1) = \left(1 - \frac{T_s}{\tau}\right) y(k) + K \frac{T_s}{\tau} u(k), \quad (2.27)$$

where T_s is a sampling period, $u(t)$ is a filter's input and $y(t)$ is a filter's output.

Moreover, the accumulation of integrative error is covered using PID controller with anti-windup compensation circuit presented in the figure 2.8. Furthermore, the saturation to bound the controller's output (its action value) is placed in there.

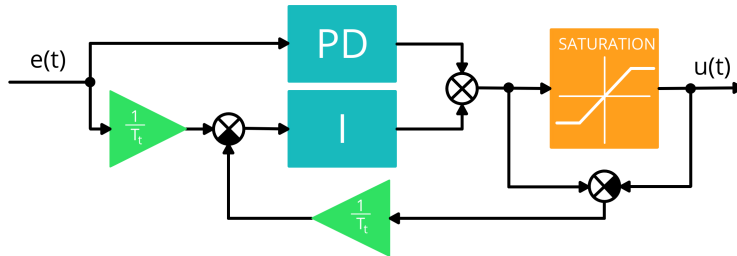


Figure 2.8: PID controller with anti-windup compensation circuit [10]

The parameter T_t can be out of the general engineers' experiences obtained as $T_t \approx \sqrt{T_I T_D} = \sqrt{\frac{K_D}{K_I}}$ [10].

2.5 Linear Regression

In the system modeling there are basically three approaches how to obtain the mathematical model of system. The first approach is called White Box Model. In this case the physics of the system is fully known. Thus, the structure of mathematical model and also its parameters related to physics could be obtained. Completely opposite approach is the Black Box Model. In this case there is not any knowledge of the system's physics. Whole model is determined using input and output data from an experiment. An approach which stands in the middle of these two mentioned approaches is the Grey Box Model. In this approach the structure of the mathematical model is known but its parameters not.

One of the Gray Box Model methods for the parameters' identification is a linear regression. The linear relation between input and output data is supposed. Input and output data are measured and based on them the coefficients describing their relation are estimated. In this thesis this method is used for one of the forward instantaneous kinematic model's parameters identification.

For the principle of this method demonstration, following simple static discrete input-output model is assumed:

$$y(k) = c u(k), \quad k = \{0, 1, 2, \dots, N\}, \quad N \in \mathbb{Z}, \quad (2.28)$$

where c is an estimated constant, $y(k)$ is the input and $u(k)$ is the output at discrete time moment.

The regression model is obtained from the equation 2.28 if the error ε is also supposed [10]:

$$y(k) = c u(k) + \varepsilon(k), \quad k = \{0, 1, 2, \dots, N\}, \quad N \in \mathbb{Z} \quad (2.29)$$

During the experiment, N measurements are done. Then, the measured input and output data are written in following column's vectors:

$$\begin{aligned} \mathbf{U} &= [y(0), y(1), y(2), \dots, y(N)]^T, \\ \mathbf{Y} &= [u(0), u(1), u(2), \dots, u(N)]^T \end{aligned} \quad (2.30)$$

Based on the measured data, the regression model can be rewritten following way [10]:

$$\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N) \end{bmatrix} = c \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(N) \end{bmatrix} + \begin{bmatrix} \varepsilon(0) \\ \varepsilon(1) \\ \vdots \\ \varepsilon(N) \end{bmatrix}$$

$$\mathbf{Y} = c \mathbf{U} + \varepsilon \quad (2.31)$$

If $N > 1$, then more measurements than unknown parameters count are done. In this case the parameter's identification is translated to the optimization task. The aim of this task is to minimize the mean square error sum criterion, which can be expressed as follows [10]:

$$J(c) = \sum_{k=0}^N \varepsilon^2(k) = \varepsilon^T \varepsilon \quad (2.32)$$

The condition of the criterion's minimum is expressed following way [10]:

$$c^* = \operatorname{argmin}(J(c)) \quad (2.33)$$

The value c^* is an optimal estimation of the parameter c and can be calculated using following expression [10]:

$$c^* = (\mathbf{U}^T \mathbf{U})^{-1} \mathbf{U}^T \mathbf{Y}, \quad (2.34)$$

where $(\mathbf{U}^T \mathbf{U})^{-1} \mathbf{U}^T$ is a generalized matrix inverse for the rectangular matrices [10].

3 Technologies Used

3.1 SIMIT

SIMIT is a simulation platform which is mainly intended to use for a complex tests of any automation applications and providing the trainings for the operators before the real system is commissioned. The major purpose of the SIMIT usage in practice is to simulate behavior of the real components which could be found in the real machine or process. In process automation these components could be for example pipes, valves or tanks. Moreover, a lot of components used in machine automation are possible to be simulated in there such as motor drivers sensors etc.

Programming in SIMIT is block-based which means that the instructions are covered by blocks and the connections between them are for data exchange purposes. Plenty of certain components used in the real automation applications are findable as blocks in the libraries and can be simply dragged and dropped into a project chart. In order to organize the structure of the SIMIT project, the logic can be distributed in more than one charts. In a Standard library, the blocks for basic (addition, multiplication etc.) and advanced (integrators, PT blocks for transfer function's definition etc.) mathematical calculations are available.

If some custom block which is not present in the standard libraries is needed than it is possible to develop own block with required functionality. A stand-alone application Component Type Editor (CTE) can be used for these purposes. It is possible to define the connectors (inputs and outputs), block parameters adjustable later in project chart, cyclic calculation executed each simulation cycle, block visualization etc.

For the simulation cycle control is possible to define more time slices. Then, these time slices can be assigned to particular simulation blocks. It is a really useful functionality. For example, in case if a part of the logic does not need to be run each fastest simulation cycle, it can simply be connected to slower time slice and the overall simulation performance is increased.

The SIMIT is used in a concept called SIMATIC Machine Simulator as a part for machine behavior calculation. See a scheme of this concept in the figure

3.1. This concept fully supports the virtual commissioning of a machine in the SIL setup (described in chapter 1.2).

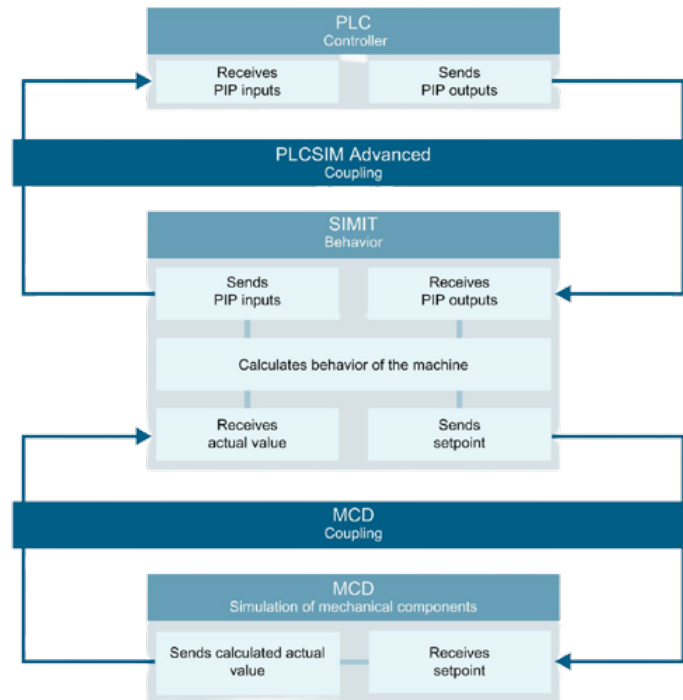


Figure 3.1: SIMATIC Machine Simulator scheme

It is possible to manage a coupling for a full-duplex data exchange with other applications for the SIL simulation purposes in SIMIT. The coupling with NX MCD is doable very easily because only the right running MCD simulation should be selected and then the MCD signals are visible in SIMIT. Similarly, the coupling with TIA portal where virtual PLC controller is programmed can be managed. When the right project is selected, new instance of virtual PLC in PLCSIM Advanced application is automatically created. Then, the PLC program is uploaded to virtual PLC and its signals are visible in SIMIT.

There is also solution for the virtual commissioning in HIL setup (described in chapter 1.3) provided by Siemens. A SIMIT Unit is used for these purposes. It is a hardware interface between SIMIT simulation platform and the real controller.

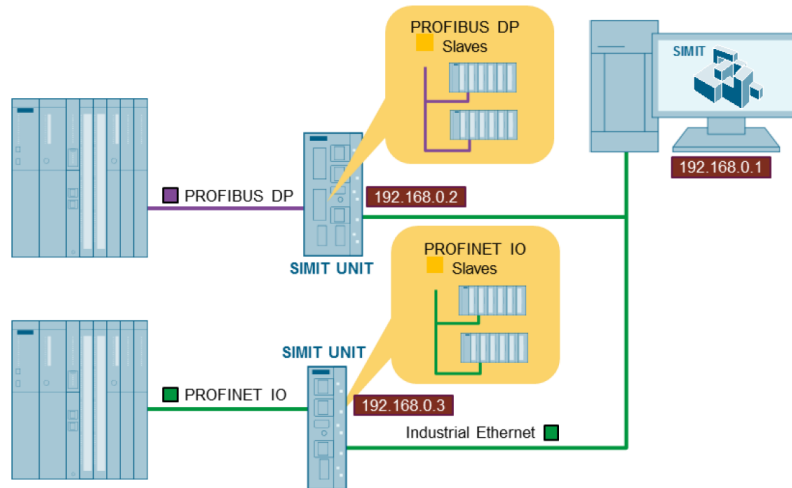


Figure 3.2: HIL simulation setup using SIMIT Unit

Using this device, the behavior of the whole machine or plant can be simulated in real time. The real controllers communicate with the SIMIT Unit via Profibus or Profinet. Then, it is connected to the station via Industrial Ethernet. In this station runs the simulation in SIMIT. See the HIL setup with the SIMIT Unit in the figure 3.2.

3.2 NX MCD

NX is a Computer-Aided Design (CAD) software used for 3D solid and assembly modelling. Any mechanical part of a machine can be designed there. Individual parts can be there merge into one assembly which can be further used in another created assemblies.

NX Mechatronics Concept Designer (NX MCD) is a NX plugin used for the definition of the mechatronic behavior of the created 3D solid. The rigid body of the 3D object can be defined in order to let the object behave as a real body on Earth. It means that when the simulation starts, the 3D object with rigid body defined starts falling down due to the gravity. Moreover, if the collision body is defined on the object, and another collision body is defined on the surface above the solid, the object falls on this surface.

Different kind of joints such as hinge or sliding joint can be also defined between two different rigid bodies. For the movement of the rigid body control, the position or speed controller can be defined. Moreover, in NX MCD the whole robotic arm use-case can be proposed using an Inverse kinematics function. The poses are simply defined, and the kinematics calculations are done automatically.

If a custom behavior of some object is needed, it can be implemented using a C# runtime behavior script. In this script, the NX Open library is used. The objects such as rigid body are simply imported to the script, where it can be worked with their properties. For instance, in case of rigid body, its position and orientation can be read and changed.

In the mentioned concept of SIMATIC Machine Simulator described in the figure 3.1, the NX MCD is used as a simulator of mechanical components. From SIMIT, the setpoints from simulated components are received. For example, in SIMIT a motor driver can be simulated and the simulation of its movement is implemented in NX MCD. The coupling for data exchange can be managed with SIMIT as well as with TIA portal. Moreover, the handshake between two NX MCD simulations can be done.

3.3 ROS2

Robot Operating system (ROS) is a pack of the tools and software libraries for the robot applications' development. ROS2 is an open-source and can be considered really as operating system because it has its own hardware abstraction, low-level control of devices, data exchange between particular processes using messages etc. ROS2 is runnable in a plenty of operating systems such as Windows or any type of Linux (Debian, Ubuntu etc.). Moreover, it can be also hosted in Docker. ROS2 is provided in several distributions such as Humble or Foxy. Each distribution is runnable in different operating systems [11].

The bases of the ROS are nodes. Each node is supposed to be responsible for a single module. For example one node can control wheel motors, another cover the reading data from lidar sensor etc. Particular nodes communicate continuously with each other using services and topics. The communication using the topics is one-directional. A publisher publishes the data and the subscriber receive them. The full-duplex communication is covered by services [11]. This is a service-client based communication. See the diagram

describing these two kinds of communication between nodes in the figure 3.3.

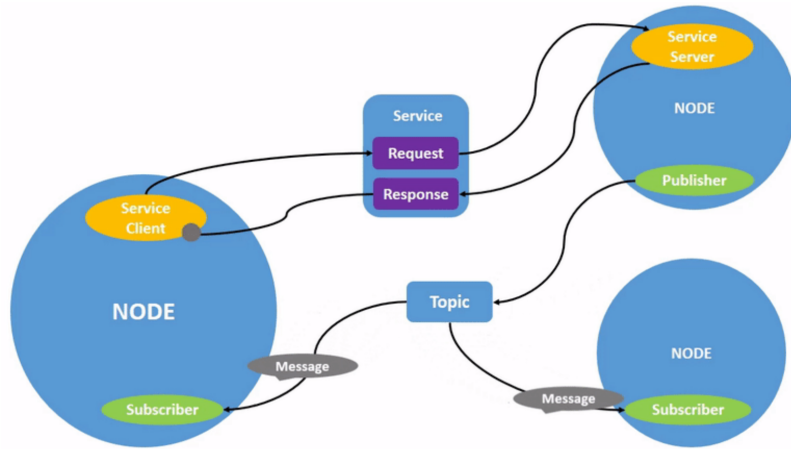


Figure 3.3: Continuous communication between ROS2 nodes using topic and service [11]

If the one-shot communication between nodes is needed, then action can be used. It includes a request message from the client to server, then the response as acknowledgement is sent opposite direction. The requested data are sent using feedback topic [11]. See this procedure in the figure 3.4.

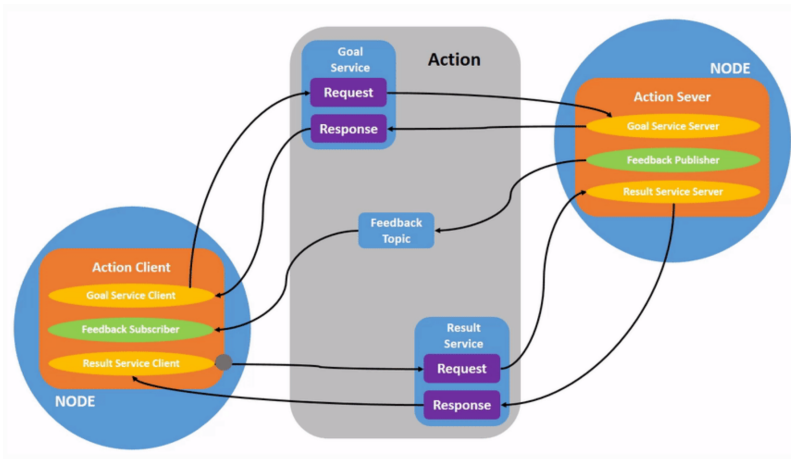


Figure 3.4: One-shot communication between ROS2 nodes using action [11]

The ROS2 nodes are mainly code in C++ and Python. Moreover, the Node-RED can be also used for dealing with ROS2 services, topics and actions. The Node-RED is a tool for flow-based programming (FBD) which is a

programming approach based on the black boxes' networks communicate with each other through messages. It is a user-friendly development tool where the logic is implemented by dragging and dropping the blocks using the connections between them. In these connections the messages flow.

Furthermore, there are some client libraries which can be used for interfacing with ROS2 such rclcs for the interfaces written C#. If there is a request for recording and playing the recorded the data from ROS2 topics, the ROS2 feature called bag recording can be used.

4 AGV Platform IOT Bot

IoT Bot is a ROS2-based AGV platform developed by Eduart Robotic in cooperation with Siemens and Technical College in Nuremberg. It is four wheels' AGV provided in two variants. The first variant is differential drive while standard wheels are used. The second variant of this AGV is the variant with mecanum wheels - mecanum drive [12].



Figure 4.1: IoT Bot - differential drive variant (on the left) and mecanum drive variant (on the right) [12]

In the scope of this thesis, the first variant is worked with. Since the intention of this thesis is the SIL and HIL simulator of this IoT Bot variant development, reverse engineering of the robot should be done. Individual parts of the given robot are investigated and described in this chapter.

4.1 SIMATIC IOT2050

SIMATIC IOT2050 is an industrial Internet Of Things (IOT) Gateway. In practice, it is used for the connection of the factory low-level devices such as SIMATIC PLCs to the internet world. Using this gateway, the data can be easily transferred from the factory to the cloud where their further processing is done [13].

The IOT2050 is equipped with the following hardware components [14]:

- TI SOC AM6528 GP Dual Core processor
- 1 GB RAM (DDR4)
- 2 Ethernet interfaces (100/1000 Mbps)

- 2 USB Type A
- 1 COM interface (RS232/422/485)
- 1 DisplayPort 1.1 A

Moreover, there is a slot for SD card where required operating system can be installed in. In case of IOT Bot, the Linux Debian is used. Two Docker images run there - ROS2 (it manages the robot control) and NODE-Red image. Using the Ethernet and Secure Shell Protocol (SSH) connection, the command prompt of the Debian is smoothly reachable from any operating station. The NODE-Red environment is accessible using the operating station's web browser and available for the user-friendly dealing with the robot data.

As mentioned, the IOT2050 manages the connection between low-level components and the outer world. The outer world is the mentioned operating station. The low-level component is the Extension-Shield described below. The communication between IOT2050 and the Extension-Shield is covered by Universal Asynchronous Receiver-Transmitter (UART). The robot can be remotely controlled by PlayStation4 (PS4) Controller which is connected with the IOT2050 via Bluetooth.

4.2 Motors and Extension-Shield

In the extension-shield, the electronics for the wheels' motors control is installed. Each motor has its own inner control loop with PID controller (related theory described in the chapter 2.4), Pulse Wide Modulation (PWM) drive and the encoder with process unit for the actual revolutions' measurement. The motors are stepper motors controlled by voltage generated by PWM pulses whose amplitude depends on battery's voltage level (range from 17.5 to 23.5 Volts).

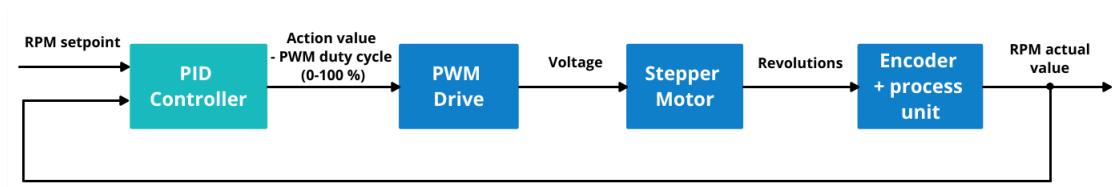


Figure 4.2: Block scheme of the motor's inner control loop

There is also an encoder for the revolution measurement in the inner control loop. The units of setpoint and process value are the Revolutions Per Minute (RPM). See the block scheme of the inner motor's control loop in the figure 4.2.

Apart from the motors' control, the Extension-Shield reads the data from another robot's sensors:

- 4 Time-of-Flight (TOF) distance sensors
- 1 Inertial Measurement Unit (IMU) - orientation, angular velocity and acceleration measurement
- 1 sensor measuring the battery voltage level

4.3 Power Supply

The robot's electronics is supplied by Nickel–Metal Hydride (NiMH) battery providing nominally 19.2 Volts and having 3.0 Ampere-hours capacity. The battery is charged by power source having 100-240 Volts (\sim , 1.2 Amperes, 50/60 Hertz) input and 30 Volts ($=$, 2 Amperes) output.

4.4 ROS2 Nodes

As mentioned, the control of the robot's movement is implemented in ROS2 nodes. The `robot_motion_control_node` contains inverse instantaneous kinematics model which theory is described in the chapter 2.2.2. Based on the target velocity vector (Twist - intended linear and angular velocity) it calculates target motor speed values (motors' setpoints). It is also possible to overcome the kinematics calculations and assign the mentioned target motor speed values directly. See the input topics published by this node in the table 4.2.

Description	Topic	Message type
Taget Motorspeed values	/iotbot/rpm	iotbot_interface/msg/RotationSpeed
Taget Velocity vector	/cmd_vel	geometry_msgs/msg/Twist

Table 4.1: ROS2 node for robot's motion calculation (its inverse kinematics): topics [12]

Another node, `iotbot_shield_node`, reads the data from sensors mentioned above. The topics and services published by this node are listed in the table

4.2 and 4.3, respectively.

Description	Topic	Message type
Controller Values	/joy	sensor_msgs/msg/joy
Target Motorspeed values	/iotbot/rpm	iotbot_interface/msg/RotationSpeed
Current Motorspeed values	/iotbot/rpm/return	iotbot_interface/msg/RotationSpeed
Inertial Measurement Unit	/iotbot/imu	sensor_msgs/msg/Imu
Distance measurements	/iotbot/tof	std_msgs/msg/Float32MultiArray
Battery Voltage	/iotbot/battery	iotbot_interface/msg/Battery

Table 4.2: ROS2 node for communication between the microprocessor and IOT2050: topics [12]

Description	Service	Message type
Determination of light pattern	/iotbot/srv/send_lighting	iotbot_interface/src/SendLighting
Enable signal for driving	iotbot/srv/send_enable	iotbot_interface/src/SendEnable

Table 4.3: ROS2 node for communication between the microprocessor and IOT2050: services [12]

5 IOT Bot's models

5.1 Mathematical Kinematic Model

The IOT Bot is a vehicle moving in two-dimensional space described with global coordinate system $[x, y]$. The robot's local coordinate system is also defined in order to describe its motion. Its origin is located in the robot's Center of Mass (COM). The position of the robot is expressed by coordinates X and Y of this origin in the global coordinate system. In order to get a full description of the robot's coordinates, its orientation Θ expressed against global x-axis should be taken in consideration as well [15]. See the robot in global coordinate system in the figure 5.1. For further work, the robot's coordinates can be written into the following vector:

$$\mathbf{q} = [X \ Y \ \Theta]^T \quad (5.1)$$

As it is described in the figure 5.1, the IOT Bot's COM moves in the global coordinate system at velocity $\mathbf{v} = [v_x \ v_y]^T$ which is expressed in the local coordinate system. The element v_x and v_y determines a longitudinal and lateral IOT Bot's velocity, respectively [15].

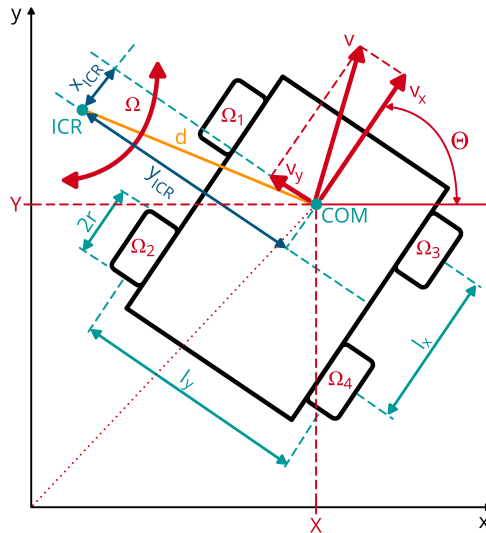


Figure 5.1: IOT Bot kinematics

The basic forward instantaneous kinematic model can be determined if the rotation of the IOT Bot around z-axis (equation 2.3) by orientation angle Θ is considered [15]:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}, \quad (5.2)$$

where ω is the IOT Bot's angular velocity.

The IOT Bot's movement is based on differential drive principle. It means that both driving and steering are controlled with the particular wheels' angular velocities' values. As it is visualized in the figure 5.1, the four wheels are used for controlling the IOT Bot's movement. However, as was proved by the measurement (described in the chapter 5.2) only the two angular velocities (related to IOT Bot's left and right side) can be considered as the forward instantaneous kinematic model's inputs:

$$\begin{aligned} \omega_L &= \omega_1 = \omega_2, \\ \omega_R &= \omega_3 = \omega_4 \end{aligned} \quad (5.3)$$

Using these angular velocities the lateral and angular velocity of the robot can be expressed [15]:

$$v_x = \frac{r}{2} (\omega_L - \omega_R), \quad (5.4)$$

$$\omega = \frac{r}{l_y} (\omega_L + \omega_R), \quad (5.5)$$

where $r = 0.085 \text{ m}$ is the radius of the IOT Bot's wheel.

The other IOT Bot's dimensions according to the figure 5.1 are:

- distance between front and rear set of the wheels $l_x = 0.25 \text{ m}$
- distance between left and right set of the wheels $l_y = 0.31 \text{ m}$

The sign ($-$ in the equation 5.4 and $+$ in the equation 5.5) expressing the relation between ω_L and ω_R can differ model by model. It depends on the direction of wheels' revolutions' setup. The relation of these two velocities in the equations 5.4 and 5.5 was found out by measurement described in the chapter 5.2.

Both equations 5.4 and 5.5 are valid only if the longitudinal slippage is neglected [15]. In case of pure robot's straightforward motion when $\omega = 0$ and $v_y = 0$ the slippage neglect is not a big issue. In the beginning of the

movement the wheels should overcome small initial lateral rolling friction. It could cause a small slippage but then a simple rolling without friction can be considered.

In case of $v_y \neq 0$ the lateral skid which causes the change of the robot's orientation Θ should be taken into account. During the skid steering, the longitudinal velocity is lost as slip and only the tangential velocity v contributes to rotation [16]. It can be described by the following expression [16]:

$$\omega = r \frac{l_y}{l_x^2 + l_y^2} (\omega_L + \omega_R), \quad (5.6)$$

where $l_x^2 + l_y^2$ is a distance between two diagonally opposite wheels.

As mentioned, the steering of the IOT Bot is done by lateral skid and due to that it highly depends on the friction between the wheels and surface. Thus, the relation between IOT Bot's angular velocity ω and the wheels' velocities described in the equation 5.5 or 5.6 is only a pure mathematical representation without any physical knowledge. However, it is useful as a priori information for the experiments described below. Its target is to find out a physical relation between the wheel's revolutions and IOT Bot's angular velocity. Considering the mentioned facts, let's determine the equation for IOT Bot's angular velocity ω calculation as follows:

$$\omega = r K_\omega (\omega_L + \omega_R), \quad (5.7)$$

where K_ω is a parameter to be identified using a real data from the experiment and the two following a priori information can be used for comparison of the experiment's result:

$$K_\omega^{api1} = \frac{1}{l_y}, \quad (5.8)$$

$$K_\omega^{api2} = \frac{l_y}{l_x^2 + l_y^2}, \quad (5.9)$$

where K_ω^{api1} and K_ω^{api2} is based on the equation 5.5 and 5.6, respectively.

In order to determine the position and orientation of the IOT Bot in the two-dimensional global coordinate system, it is necessary to derive the expression of IOT Bot's lateral velocity v_y as well. As it is drawn in the figure 5.1, the IOT Bot rotates around its Instantaneous Center of Rotation (ICR) during the steering. From the figure 5.1, it can be also deduced [15]:

$$|\omega| = \frac{\|\mathbf{v}\|}{\|\mathbf{d}\|}, \quad (5.10)$$

where $\|\mathbf{d}\|$ is a radius of the IOT Bot's rotation around ICR, vector $\mathbf{d} = [X \ Y \ \Theta]^T$, perpendicular to \mathbf{v} , expresses the coordinates of the ICR in the IOT Bot's local coordinate system and symbol $\|*\|$ means the Euclidean vector's norm.

Based on the equation 5.10 it can be also written [17]:

$$\omega = \frac{v_x}{y_{ICR}} = -\frac{v_y}{x_{ICR}} \quad (5.11)$$

By modification of the equation 5.11, the following expression of the IOT Bot's lateral velocity can be obtained:

$$v_y = -x_{ICR}\omega, \quad (5.12)$$

where the parameter X_{ICR} is the x-projection of ICR. During the pure rotation of IOT Bot at spot this constant expresses the radius of the rotation. In other words, it is the distance between the IOT Bot's COM and geometric center of rotation. The IOT Bot's COM has been estimated manually Then, the distance between estimated COM and geometric center of rotation has been measured. The result is $X_{ICR} = 0.02 \text{ m}$.

This result corresponds with the statement about this constant. This statement says that $X_{ICR} < \frac{l_x}{2}$ otherwise the vehicle should not move properly [17]. In the case of IOT Bot it is $X_{ICR} = 0.02 \text{ m} < \frac{l_x}{2} = 0.125 \text{ m}$.

The equation 5.12 can be installed into the equation 5.2 in order to get the following forward instantaneous kinematic model:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ -x_{ICR}\omega \\ \omega \end{bmatrix} \quad (5.13)$$

Let's also install the equations 5.4 and 5.7 into the equation 5.13 as follows:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) & 0 \\ \sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r}{2} (\omega_L - \omega_R) \\ -x_{ICR} r K_\omega (\omega_L + \omega_R) \\ r K_\omega (\omega_L + \omega_R) \end{bmatrix} \quad (5.14)$$

Further, it is necessary to extract the wheels' angular velocities ω_L and ω_R into a single vector on the right side of the equation. Then, the final version of the IOT Bot's instantaneous forward kinematic model is obtained:

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} \cos(\Theta) + r X_{ICR} K_\omega \sin(\Theta) & -\frac{r}{2} \cos(\Theta) + r X_{ICR} K_\omega \sin(\Theta) \\ \frac{r}{2} \sin(\Theta) - r X_{ICR} K_\omega \cos(\Theta) & -\frac{r}{2} \sin(\Theta) - r X_{ICR} K_\omega \cos(\Theta) \\ r K_\omega & r K_\omega \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix}, \quad (5.15)$$

where the values $\omega_L = \omega_1 = \omega_2$ and $\omega_R = \omega_3 = \omega_4$ are the forward instantaneous kinematic model's inputs (IOT Bot's actuators) and the values $\dot{\mathbf{q}} = [\dot{X} \ \dot{Y} \ \dot{\Theta}]^T$ are the forward instantaneous kinematic model's outputs (time differences of IOT Bot's coordinates in the global coordinate system).

In order to obtain the IOT Bot's coordinates q (equation 5.1) at the time t , the forward instantaneous kinematic model's equations should be integrated with respect to time:

$$\begin{aligned}
X(t) &= \int_0^t \left\{ \frac{r}{2} \cos[\Theta(t)] + r X_{ICR} K_\omega \sin[\Theta(t)] \right\} \omega_L(t) + \\
&\quad + \left\{ -\frac{r}{2} \cos[\Theta(t)] + r X_{ICR} K_\omega \sin[\Theta(t)] \right\} \omega_R(t) d\tau, \\
Y(t) &= \int_0^t \left\{ \frac{r}{2} \sin[\Theta(t)] - r X_{ICR} K_\omega \cos[\Theta(t)] \right\} \omega_L(t) + \\
&\quad + \left\{ -\frac{r}{2} \sin[\Theta(t)] - r X_{ICR} K_\omega \cos[\Theta(t)] \right\} \omega_R(t) d\tau, \\
\Theta(t) &= \int_0^t r K_\omega \omega_L(t) + r K_\omega \omega_R(t) d\tau, \tag{5.16}
\end{aligned}$$

where τ is an integration time value.

5.2 Kinematic model's parameter K_ω Identification

5.2.1 Identification using Unsteady RPM values

During the first experiment the IOT Bot rotates at single spot. It means that only the angular velocity ω caused by lateral velocity v_y is considered and longitudinal velocity v_x equals to 0. The rotation was done at different angular velocities ω in both directions. This experiment had three main aims:

1. evaluation that only two forward instantaneous kinematic model's inputs (see equation 5.3) can be considered
2. evaluation of the direction of wheels' revolutions' setup (sign + between ω_L and ω_R in the equation 5.7)
3. identification of the parameter K_ω (see equation 5.7)

During the experiment following data were measured from corresponding ROS2 topics (see the 4.2) in Node-RED:

- $\omega_1, \omega_2, \omega_3$ and ω_4 - wheels' actual angular velocity values (revolutions) in Revolutions Per Minute (RPM)
- ω - IOT Bot's angular velocity - data from Inertial Measurement Unit (IMU) in RPM, but then recalculated to radians per second (rad/s)

The general expression for calculation of angular velocity in rad/s from RMP is obtained as follows:

$$\omega_{rad/s} = \frac{2\pi}{60} \omega_{RPM} \quad (5.17)$$

In the figure 5.2 and 5.3 are the measured left wheels' revolutions in RPM plotted.

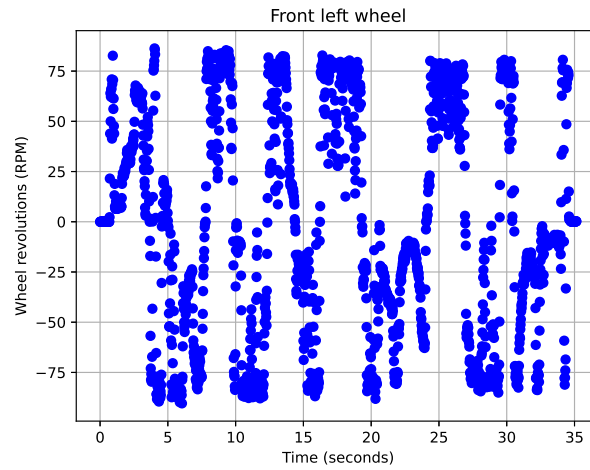


Figure 5.2: IOT Bot's front left wheel's revolutions - ω_1

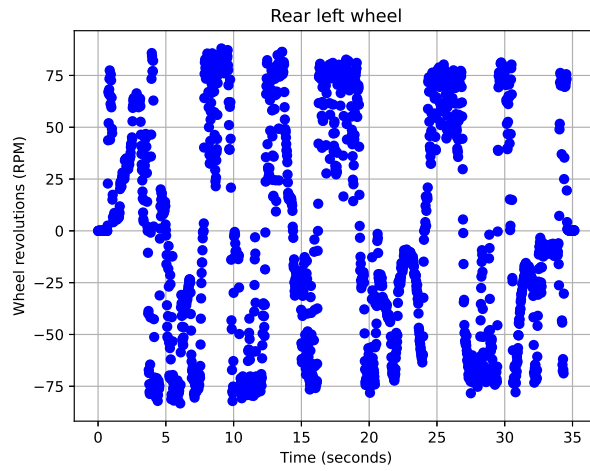


Figure 5.3: IOT Bot's rear left wheel's revolutions - ω_2

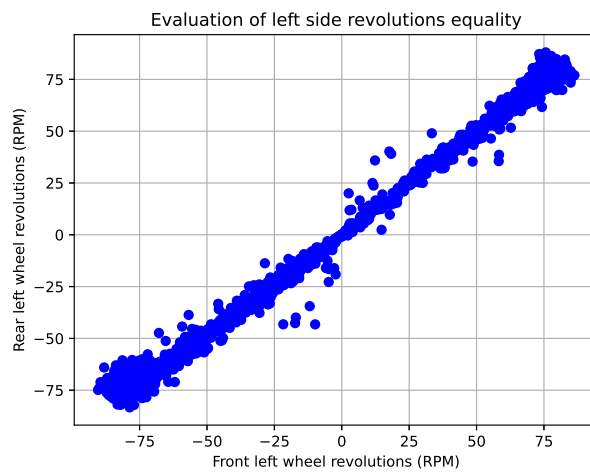


Figure 5.4: Evaluation of $\omega_1 = \omega_2$ (see equation 5.3)

As it can be seen in the figure above, the values ω_1 and ω_2 are spread out around the line $\omega_1 = \omega_2$. The wider area around this line and some outliers are caused by the friction between the wheels and surface. Meaning, the equality $\omega_L = \omega_1 = \omega_2$ is evaluated as useful for the IOT Bot's forward instantaneous kinematic model's purposes.

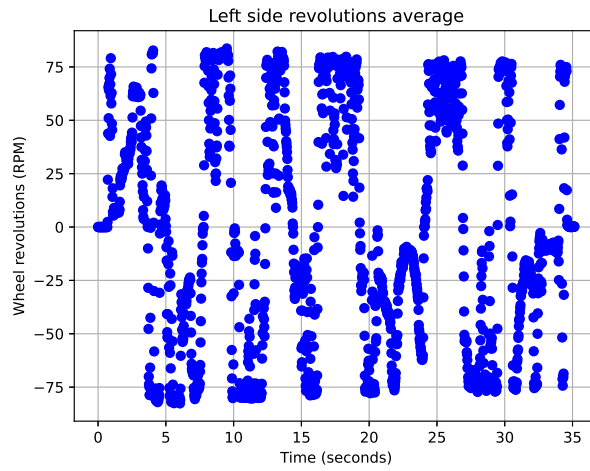


Figure 5.5: Average of ω_1 and ω_2 : $\omega_L = \frac{\omega_1 + \omega_2}{2}$

For further work with the measured data the ω_L is obtained as the average of ω_1 and ω_2 . See almost unrecognizable differences between the figures 5.2, 5.3 and 5.5. In the figure 5.6 and 5.7 are the measured right wheels' revolutions in RPM plotted.



Figure 5.6: IOT Bot's front right wheel's revolutions - ω_3

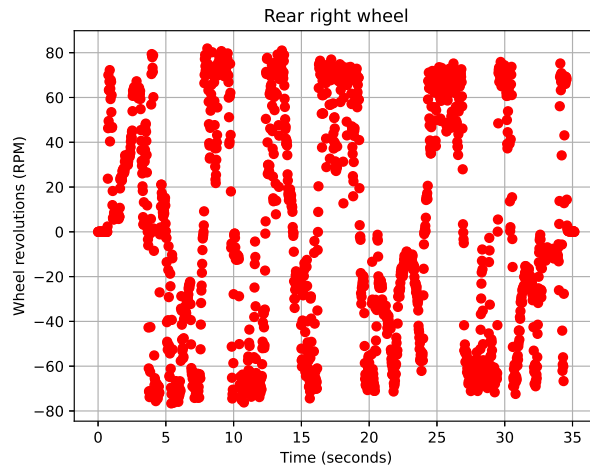


Figure 5.7: IOT Bot's rear right wheel's revolutions - ω_4

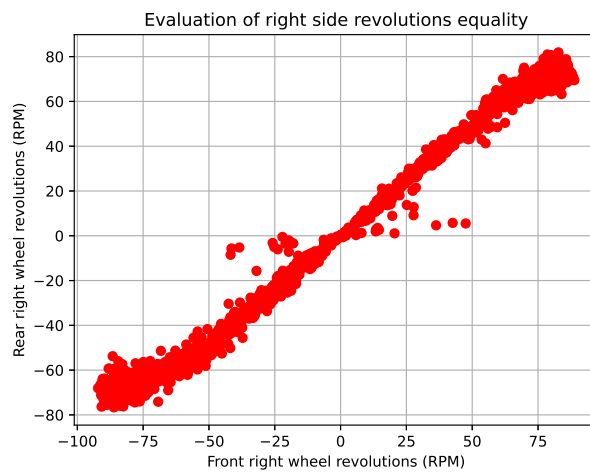


Figure 5.8: Evaluation of $\omega_3 = \omega_4$ (see equation 5.3)

Similarly, as the equality $\omega_L = \omega_1 = \omega_2$, the equality $\omega_R = \omega_3 = \omega_4$ can be evaluated as useful for the IOT Bot's forward instantaneous kinematic model's purposes.

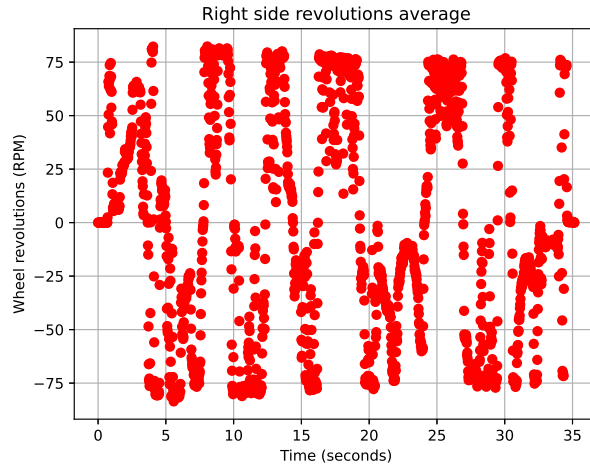


Figure 5.9: Average of ω_3 and ω_4 : $\omega_L = \frac{\omega_3 + \omega_4}{2}$

Similarly, as the ω_L the ω_R is obtained as the average of ω_3 and ω_4 . The plots in the figures 5.6, 5.7 and 5.9 are also almost unrecognizable from each other. In the figure 5.4 the IOT Bot's angular velocity measured in RPM using IMU and recalculated to rad/s is plotted.

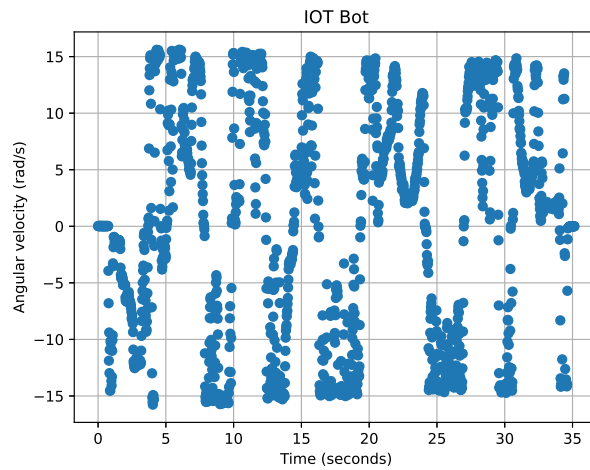


Figure 5.10: IOT Bot's angular velocity

The plotted values in the figures 5.5 and 5.9 are similar. Thus, it can be deduced that the sign + between ω_L and ω_R can be evaluated as correct because a consideration of sign - would null the IOT Bot's angular velocity in the equation 5.7.

In order to identify the parameter K_ω used in the equation 5.7 let's define a regression model according the equation 2.31:

$$\begin{bmatrix} \omega(0) \\ \omega(1) \\ \vdots \\ \omega(N) \end{bmatrix} = K_\omega \begin{bmatrix} r [\omega_L(0) + \omega_R(0)] \\ r [\omega_L(1) + \omega_R(1)] \\ \vdots \\ r [\omega_L(N) + \omega_R(N)] \end{bmatrix} + \begin{bmatrix} \varepsilon(0) \\ \varepsilon(1) \\ \vdots \\ \varepsilon(N) \end{bmatrix}$$

$$\mathbf{\Omega} = K_\omega \mathbf{W} + \varepsilon, \quad (5.18)$$

where $\omega(k)$ ($k = \{0, 1, 2, \dots, N\}$) is measured IOT Bot's angular velocity in rad/s, r is the IOT Bot's wheel radius in meters, $\omega_L(k)$ and $\omega_R(k)$ are the averages of IOT Bot's side revolutions in RPM, $\varepsilon(k)$ is the error's vector, k is a discrete time and N is the number of measured data.

Using the equation 2.34 the parameter K_ω can be identified the following way:

$$K_\omega = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \mathbf{\Omega} \quad (5.19)$$

The time between two measurements steps k and $k+1$ is 0.02 s. It means that the measurement's frequency is 50 Hz. The identification of the parameter K_ω using the equation 5.19 and other data processing was done in a Python script. Considering the units of measured values, the unit of the identified parameter K_ω should be m^{-1} :

$$K_\omega = \frac{\omega}{r (\omega_L + \omega_R)}$$

$$m^{-1} = \frac{rad/s}{m RPM}$$

$$m^{-1} = \frac{\frac{2\pi}{60} RPM}{m RPM} \quad (5.20)$$

In the figure 5.11 the identification is visualized. There are $r (\omega_L(k) + \omega_R(k))$ values in the x-axis and $\omega(k)$ values in the y-axis. The value of the black line's slope is the identified parameter $K_\omega = -1.058 m^{-1}$. The sign $-$ of the identified parameter means that the IOT Bot's rotation's direction is defined oppositely than it was assumed.

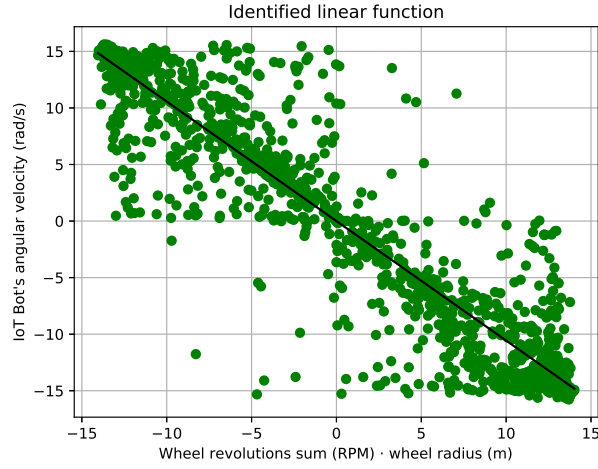


Figure 5.11: Identified K_ω (see the equation 5.7, 5.8 and 5.9)

The plotted data in the figures above come from an experiment done on tile floor. Similar experiments on different surfaces were done as well. Their results (values of identified parameter K_ω) are written in the 5.1.

	Measurement 1	Measurement 2	Measurement 3	Average
Tile floor	-1.058	-1.042	-1.057	-1.052
Floating floor	-1.082	-0.991	-0.985	-1.019
Carper (harder)	-1.061	-1.048	-0.998	-1.036
Carpet (softer)	-1.022	-1.032	-1.035	-1.030

Table 5.1: Identified K_ω on different surfaces

Generally, the higher K_ω the more wheels' motors' energy is translated to rotational movement. As seen in the table 5.1 the higher average K_ω was identified for tile floor. This surface seems to be the biggest advantage for the IOT Bot's rotational movement or for skid steering. The rotational movement on the carpet is generally very tough because the friction between the wheels and surface is very high. Then, the wheels should overcome this friction and some part of its energy is lost. In case of floating floor, there is an opposite problem. The friction is too low and due to the slip a part of the wheels' energy is lost.

If the corresponding IOT Bot's dimensions are substituted into the equation 5.8 and 5.9, the numerical values of the two a priori information about the parameter K_ω are obtained:

$$K_\omega^{api1} = 3.229 \text{ m}^{-1}, \quad (5.21)$$

$$K_\omega^{api2} = 1.995 \text{ m}^{-1} \quad (5.22)$$

If the absolute values of the identified parameter K_ω are compared with the two a priori information values, it can be observed that the identified values are much lower. It means that the influence of the friction between IOT Bot's wheels and the surface is assumed to be much lower in the equation 5.6 than it actually is.

5.2.2 Identification using Steady RPM values

As it can be seen in the figure 5.11 there are many outliers far away from the identified line with slope K_ω . It is caused by the actual values of the wheel's revolutions ω_1 , ω_2 , ω_3 and ω_4 which are not steady. In other words, the data of transient response from initial value to setpoint's value of particular wheel's revolutions are used.

In order to avoid this problem, the second experiment was done. It is quite a similar experiment as the first one having the same aims. Moreover, similar data are measured. The first difference is that wheel's angular velocities' setpoints ω_1^* , ω_2^* , ω_3^* and ω_4^* are measured as well. The second difference is that all data are measured with their timestamps.

The wheel's velocities' actual values should be later compared with the measured setpoints. However, the problem is that a lower sampling frequency is used for the setpoints' measuring. So, the missing setpoints' samples should be calculated afterwards using a linear interpolation. It means that the linear function intersecting two neighboring samples is found and the value in the middle of them is calculated.

In the four figures below the measured particular wheel's revolutions' setpoints and actual values are plotted.

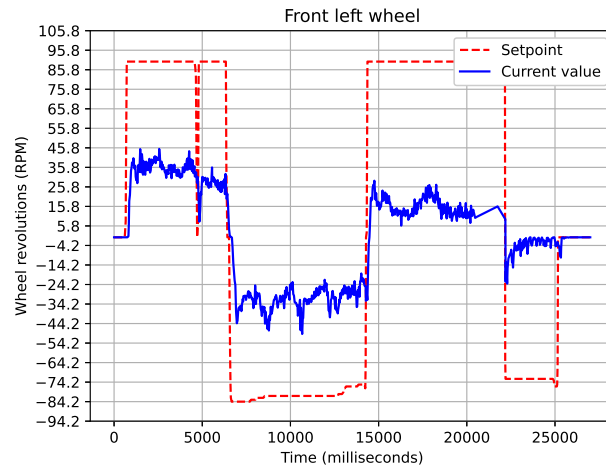


Figure 5.12: IOT Bot's front left wheel's revolutions (setpoint ω_1^* and actual value ω_1)

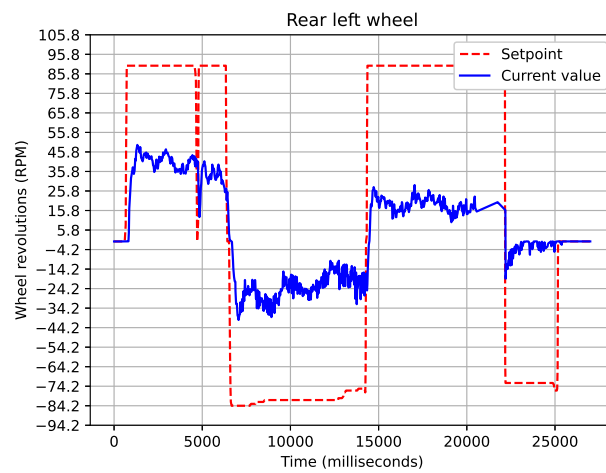


Figure 5.13: IOT Bot's rear left wheel's revolutions (setpoint ω_2^* and actual value ω_2)

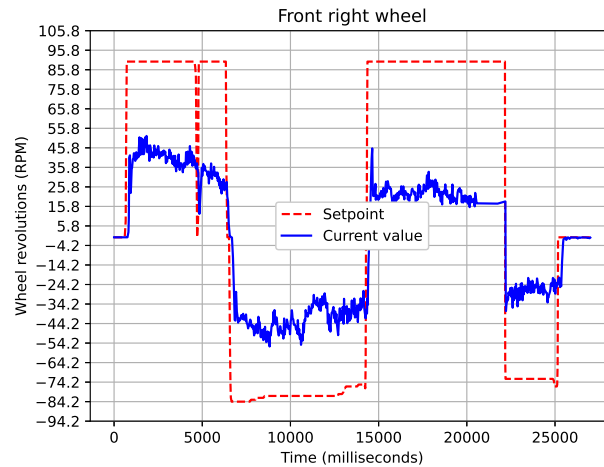


Figure 5.14: IOT Bot's front right wheel's revolutions (setpoint ω_3^* and actual value ω_3)



Figure 5.15: IOT Bot's rear right wheel's revolutions (setpoint ω_4^* and actual value ω_4)

As seen in the four figures above, the actual values of wheels' revolutions never reach the setpoints. The reason is the surface where this experiment was performed. The influence of hard carpet's friction is significant. Thus, it is impossible that the motors satisfy the RPM setpoints' request. In the figure 5.16 the IOT Bot's angular velocity in rad/s measured during this experiment is plotted.

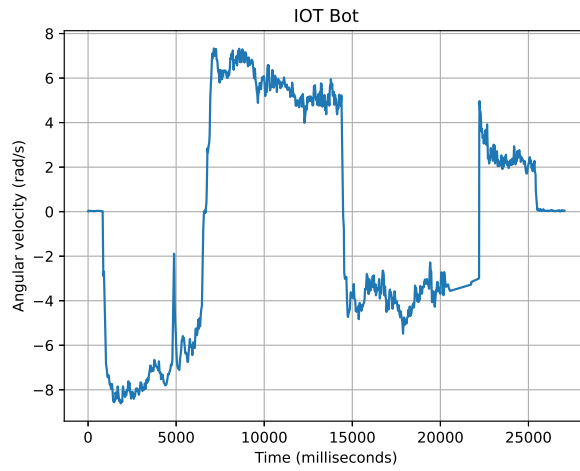


Figure 5.16: IOT Bot’s angular velocity

As can be seen in the figure 5.17 and 5.18, the setpoints of the wheel’s angular velocities are equaled. It is a definitive proof that ω_L and ω_R (side wheels’ revolutions) can be considered as only two inputs to IOT Bot’s forward instantaneous kinematic model.

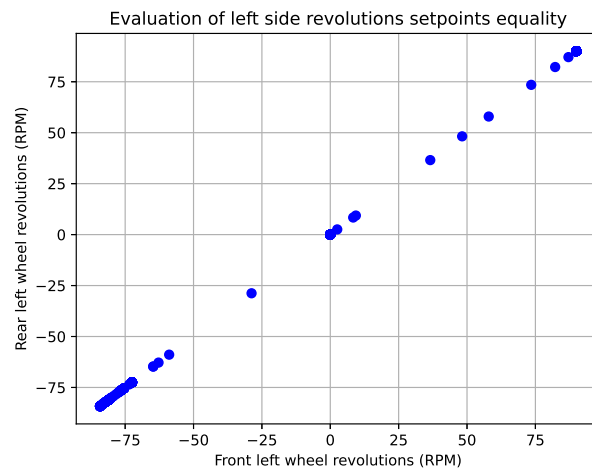


Figure 5.17: Confirmation of $\omega_1 = \omega_2$ (see equation 5.3) - using the setpoints

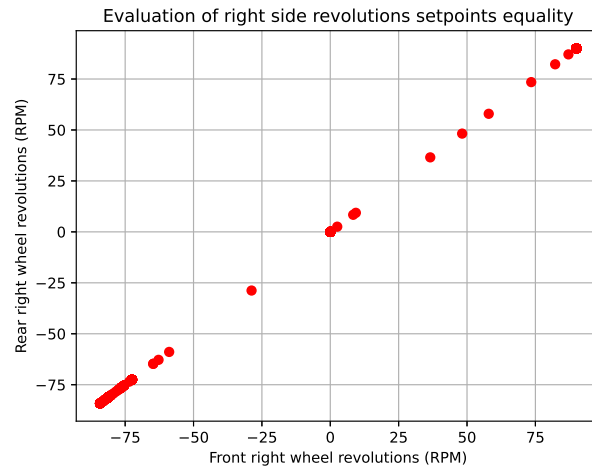


Figure 5.18: Confirmation of $\omega_3 = \omega_4$ (see equation 5.3) - using the setpoints

In the two figures below the actual values of wheels' revolutions are plotted same way as their setpoints in the two figures above.



Figure 5.19: Evaluation of $\omega_1 = \omega_2$ (see equation 5.3) - using the actual values

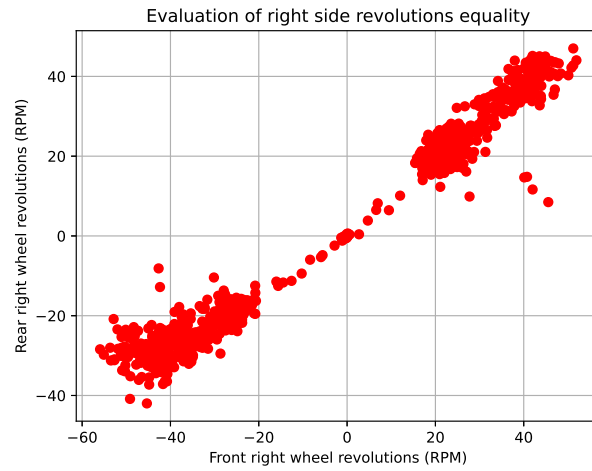


Figure 5.20: Evaluation of $\omega_3 = \omega_4$ (see equation 5.3) - using the actual values

Similarly, as it was in the previous experiment, the average of front and rear wheels' revolutions is used for the parameter K_ω identification.



Figure 5.21: Average of ω_1 and ω_2 : $\omega_L = \frac{\omega_1 + \omega_2}{2}$

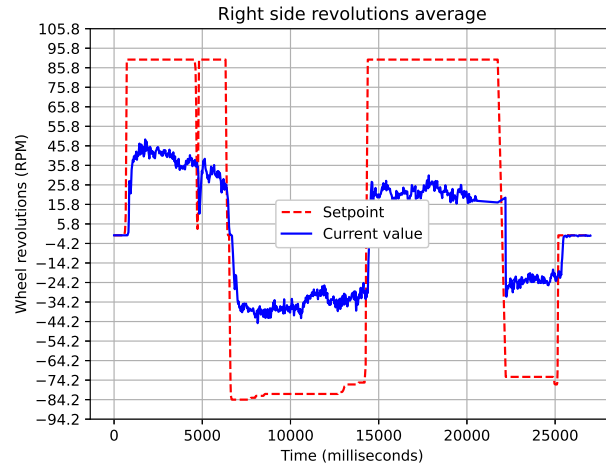


Figure 5.22: Average of ω_3 and ω_4 : $\omega_L = \frac{\omega_3 + \omega_4}{2}$

The principle of K_ω identification's improvement using data from this experiment is that only the steady values of the actual wheel's revolutions are taken into account. The steady values are determined using a defined steady zone around the setpoints.

As it can be seen in the figure 5.23, the number of outliers is much lower than in the plotted data in the figure 5.11. However, similar result was reached. This experiment's data plotted in the figures above was measured when IOT Bot rotated on harder carpet (see the table 5.1). The value of identified parameter is $K_\omega = -1.12$.

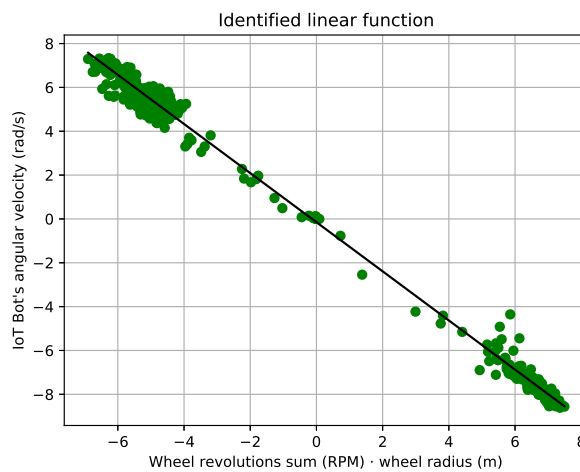


Figure 5.23: Identified K_ω (see the equation 5.7, 5.8 and 5.9)

5.3 Wheel's RPM Control Loop's Dynamic Model

The dynamic part of the PWM drive, motor and encoder with its process unit (see description in the chapter 4.2) is modeled by following transfer function (related theory described in the chapter 2.3) [18]:

$$S(s) = \frac{1.11}{0.069s + 1} \frac{1}{0.0023s + 1} = \frac{1.11}{0.0001587s^2 + 0.0713s + 1} \quad (5.23)$$

The data measured directly from Extension-Shield was used for the transfer function identification. The Strejc method was used for the calculation of its coefficients [18]. See the visualization which parts of inner IOT Bot's control loop are modeled by the transfer function in the figure 5.24.

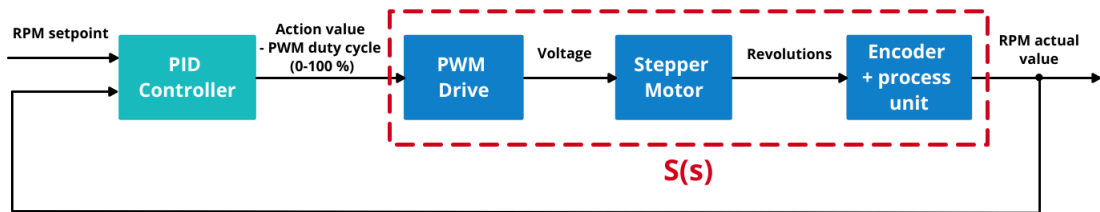


Figure 5.24: Block scheme of the motor's inner control loop - modeled parts

The PID controller is modeled based on the theory described in the chapter 2.4. The derivation of the error value in the derivative part of the controller is supplied by the time difference. The low-pass filter is implemented (using the equation 2.27) for the derivative part in order to lower the noise. Moreover, the anti-windup compensation circuit is included in the output of the circuit. The controller has the following parameters to set:

- proportional gain K_P
- integration gain K_I
- derivative gain K_D
- derivative filter's time constant's coefficient N
- minimum amplitude of action value A_{min}
- maximum amplitude of action value A_{max}
- sampling period T_s

5.4 Models' Implementation using SIMIT

The wheels' RPM control loops' dynamic model and the IOT Bot's kinematic model is implemented in SIMIT (see the chapter 3.1) using five charts. In the first chart, the inner control loops are implemented using two macros. Macro is a part of the program which can be implemented once and then reused multiple times. In this case it is quite convenient because all four wheels' RPM inner control loops are modeled the same way. In comparison with a custom component created using CTE, the advantage is that it is not necessary to use any stand-alone application. See the first chart in the figure 5.25.



Figure 5.25: Model's implementation using SIMIT - first chart

The real IOT Bot's wheel's RPM PID controller has the following parameters: $K_P = 13.513$, $K_I = 195.84$ and $K_D = 0$. It means that the PI controller is used in practice. The same parameters are used in the simulation. The PID controller's macro has the controller's parameters described in chapter 5.3 as the inputs (see the figure 5.26).

The value $K_D = 0$ causes problems during computation of derivative filter's

time constant $\tau \approx \frac{K_D}{N \cdot K_P}$ and integrative anti-windup circuit's parameter $T_t \approx \sqrt{\frac{K_D}{K_I}}$ (see the description of these parameters in the chapter 2.4). It is covered by setting the τ as $\frac{1}{N \cdot K_P}$ and the T_t as $\sqrt{\frac{1}{K_I}}$.

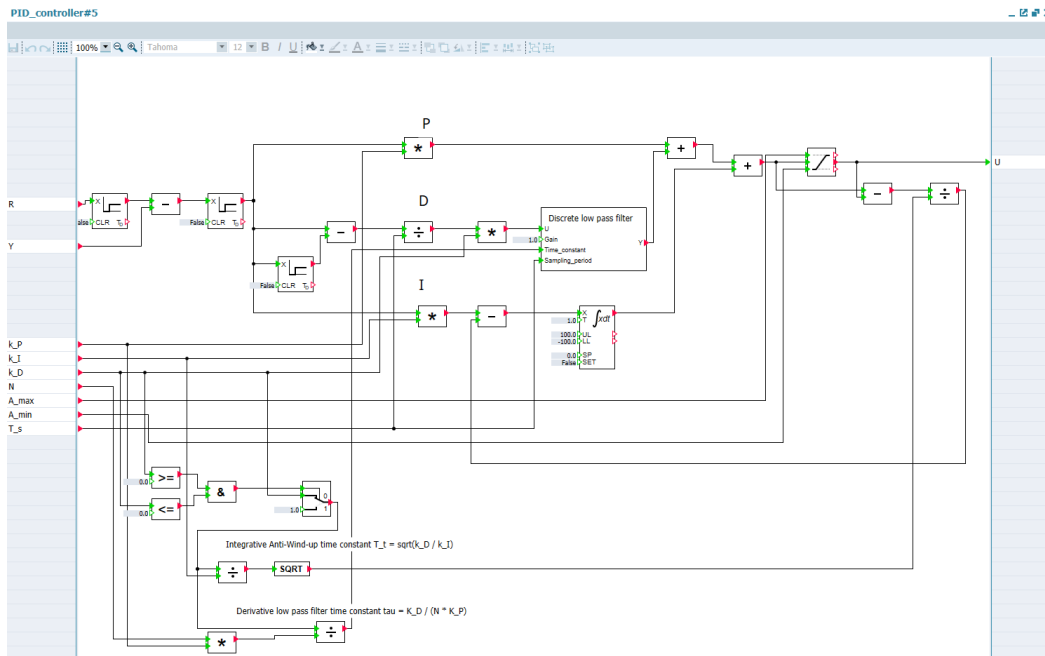


Figure 5.26: Model's implementation using SIMIT - PID controller's macro

The transfer function according to the equation 5.23 is implemented in the process's macro using two first order transfer function blocks (PT1). See it in the figure 5.27.

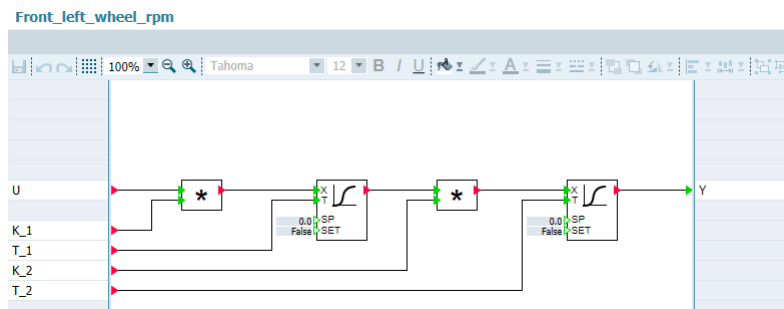


Figure 5.27: Model's implementation using SIMIT - process's macro

The calculations of the IOT Bot's linear (see equation 5.4) and angular velocity (see equation 5.7) are implemented in the second chart (see figure

5.28). In order to have a possibility for changing between the K_ω calculated using the equation 5.9 and the identified values according the table 5.1, the switch is implemented.

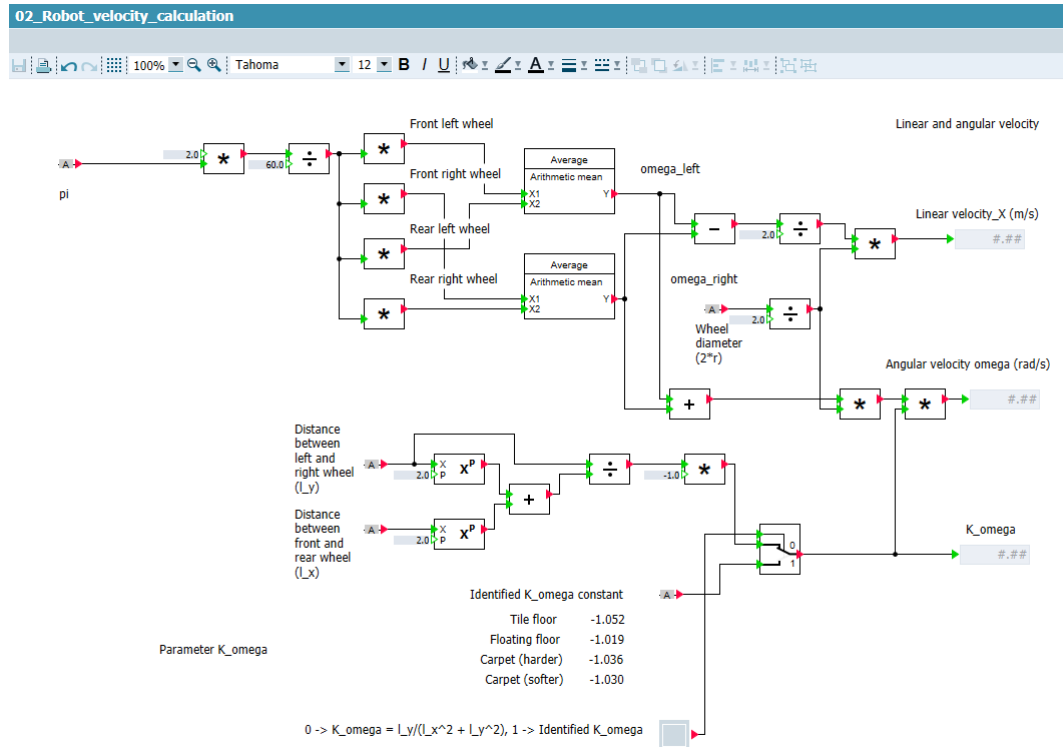


Figure 5.28: Model's implementation using SIMIT - second chart

The forward instantaneous kinematic model according to the equation 5.14 is implemented in the third chart. The position and orientation is obtained from the velocities using integrators. The IOT Bot's orientation angle Θ is scaled to range $(-\pi, \pi]$ using the following equation:

$$\Theta_{(-\pi, \pi]} = -\text{mod}(-\Theta + \pi, 2\pi) + \pi, \quad (5.24)$$

where the operator *mod* means modulo. The modulo is calculated in the custom block created using SIMIT CTE.

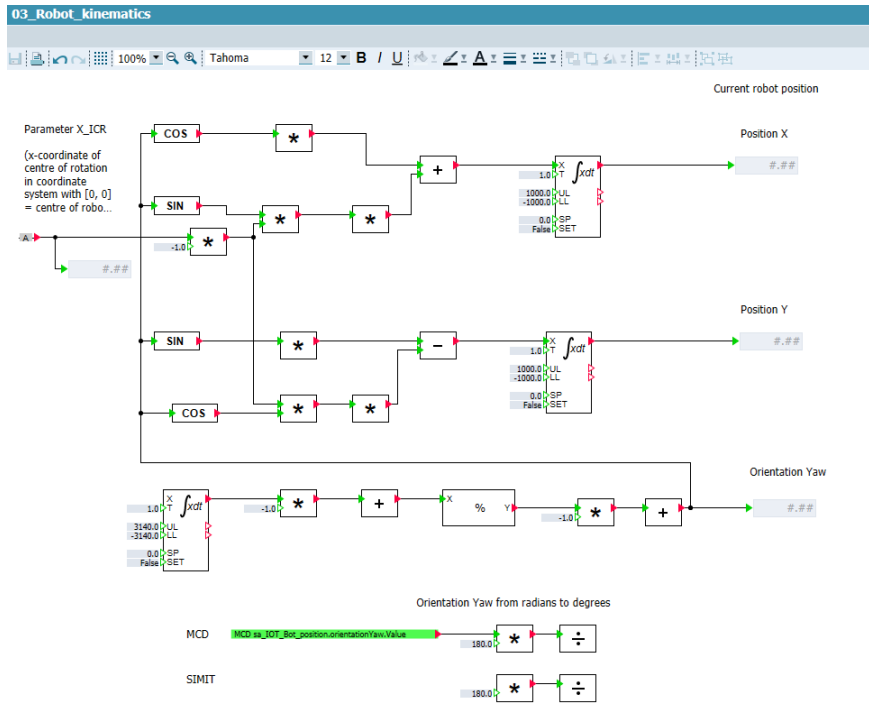


Figure 5.29: Model's implementation using SIMIT - third chart

The instantaneous inverse kinematic model is implemented in the fourth chart (see the figure 5.30) for the purposes of the required IOT Bot's side wheels' RPM calculation. The following equations are derived using the equation 5.4 and 5.7:

$$\begin{aligned} \omega_L &= \frac{1}{r} \left[v_x + \frac{1}{2 K_\omega} \omega \right], \\ \omega_R &= \frac{1}{r} \left[-v_x + \frac{1}{2 K_\omega} \omega \right] \end{aligned} \quad (5.25)$$

Moreover, it is possible to switch on a treatment that IOT Bot turns only at spot ($v_x = 0$ and $\omega \neq 0$) or drive linearly ($v_x \neq 0$ and $\omega = 0$). It can also be switched off and both requests can be fulfilled ($v_x \neq 0$ and $\omega \neq 0$).

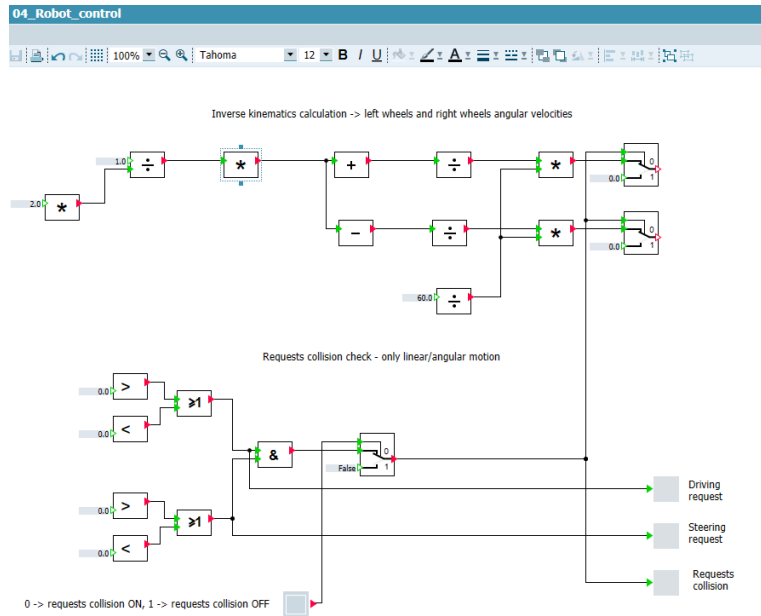


Figure 5.30: Model's implementation using SIMIT - fourth chart

The five chart is used as a user interface. It is possible to switch between the IOT Bot's control using the target velocity vector ($[v_x, \omega]$) and direct control of particular wheels ($[\omega_1, \omega_2, \omega_3, \omega_4]$). The required and actual wheels' RPM values are monitored. The values of IOT Bot's position and orientation are visualized and compared with the similar values calculated using NX MCD mechatronic model (described in the chapter 5.5).

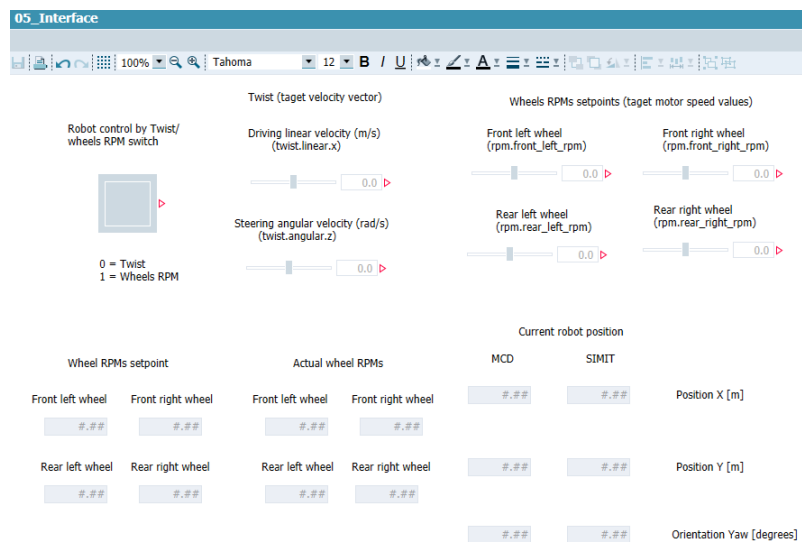


Figure 5.31: Model's implementation using SIMIT - fifth chart

5.5 NX MCD Mechatronic Model

The IOT Bot's mechatronic model is created in NX MCD (see the application's description in the chapter 3.2). The 3D graphics is handed over from the developer of IOT Bot and imported to NX from .stp file.

Using the NX MCD the physics is defined. The IOT Bot's assembly includes one main rigid body (IOT Bot's body) and four wheels' rigid bodies. The collision body is defined for each wheel. The collision material with possibility of static friction change is defined for each wheel's collision body. The four hinge joints are attached to the particular wheels. Their base is the IOT Bot's main rigid body. The angular velocity of the wheels is controlled by defined speed controllers. Moreover, the fourth IOT Bot's distance sensors are defined to simulate the distance measurements. See The IOT Bot's assembly in the figure 5.32.

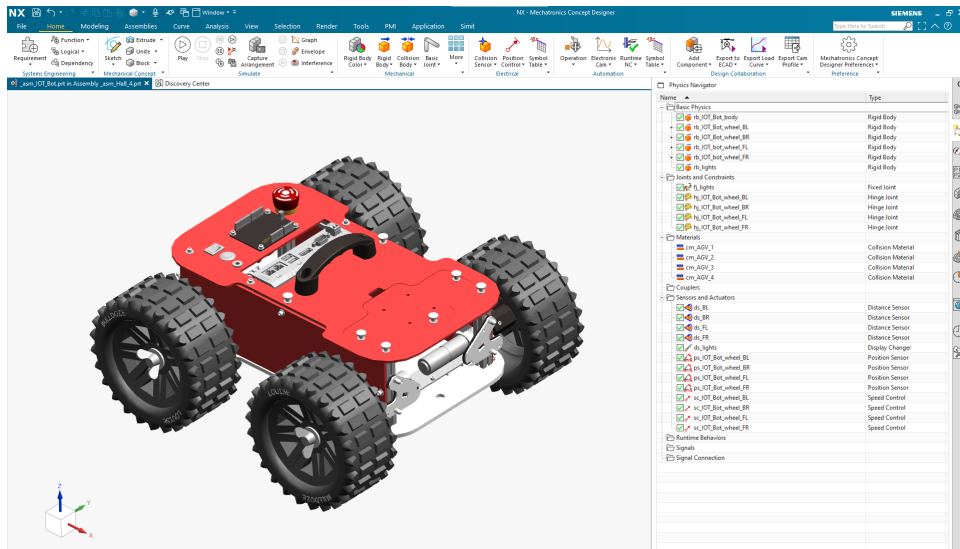


Figure 5.32: IOT Bot's assembly in NX MCD

The IOT Bot's assembly is placed in the assembly of Siemens Nuremberg's machine hall's digital twin (see figure 5.33). The collision bodies and collision materials are defined for the floor and the obstacles. The IOT Bot moves in this virtual hall. The position and orientation of the IOT Bot is determined using the C# runtime behavior which uses the functionalities of NX Open library (see the script in the figure 5.34).

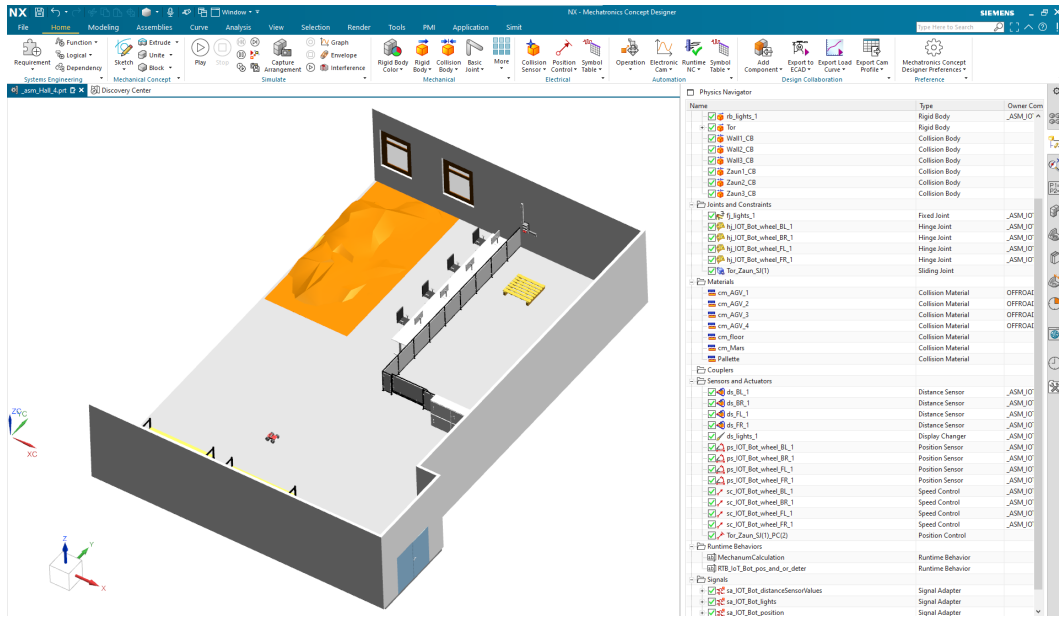


Figure 5.33: Machine hall's digital twin assembly in NX MCD

The position and orientation of the IOT Bot's main rigid body is calculated against the NX MCD's coordinate system origin. For this reason the initial offsets should be taken into account in order to determine these values correctly. The position is obtained directly as the rigid body's feature. The orientation is calculated from general orientation matrix's which is defined like follows:

$$R = \begin{bmatrix} \cos(\alpha) \cos(\beta) & \cos(\alpha) \sin(\beta) \sin(\gamma) - \sin(\alpha) \cos(\gamma) & \cos(\alpha) \sin(\beta) \cos(\gamma) + \sin(\alpha) \sin(\gamma) \\ \sin(\alpha) \cos(\beta) & \sin(\alpha) \sin(\beta) \sin(\gamma) + \cos(\alpha) \cos(\gamma) & \sin(\alpha) \sin(\beta) \cos(\gamma) - \cos(\alpha) \sin(\gamma) \\ -\sin(\beta) & \cos(\beta) \sin(\gamma) & \cos(\beta) \cos(\gamma) \end{bmatrix}, \quad (5.26)$$

where the angle

- α is called yaw and expresses the rotation around z-axis (see equation 2.3)
- β is called pitch and expresses the rotation around y-axis
- γ is called roll and expresses the rotation around x-axis (see equation 2.4)

The angles β and γ are always equal to zero in this case. The IOT Bot's orientation's angle α (yaw) is calculated from the general orientation matrix using the two-input's arcus tangens function:

$$\alpha = \text{atan2}(R(2, 1), R(1, 1)) = \text{atan2}(\sin(\alpha), \cos(\alpha)) \quad (5.27)$$

```

1 using System;
2 using NXOpen;
3 using IxOpen.VectorArithmetic;
4
5 [Definition]
6 public class UserBehavior : BehaviorDef
7 {
8     Session theSession;
9     Part workPart;
10    Part displayPart;
11    RigidBody iotBotRigidBody;
12    Signal positionX;
13    Signal positionY;
14    Signal offsetX;
15    Signal offsetY;
16    Signal angularVelocity;
17    double prevOrientation;
18
19
20 [References]
21 public override void Define(IDefinitionContext access)
22 {
23     access.Connect("Iot Bot main rigid body", out iotBotRigidBody);
24     access.Connect("Iot Bot position x-axis", out positionX);
25     access.Connect("Iot Bot position y-axis", out positionY);
26     access.Connect("Iot Bot orientation angle", out orientationYaw);
27     access.Connect("Iot Bot offset x-axis", out offsetX);
28     access.Connect("Iot Bot offset y-axis", out offsetY);
29
30     access.Connect("Iot Bot angular velocity", out angularVelocity);
31     prevOrientation = 0.0;
32
33     theSession = Session.GetSession();
34     workPart = theSession.Parts.Work;
35     displayPart = theSession.Parts.Display;
36 }
37
38 [Runtime]
39 public override void Start(IRuntimeContext context)
40 {
41 }
42
43 [Runtime]
44 public override void Stop(IRuntimeContext context)
45 {
46     // Insert stopping code here
47 }
48
49 [Runtime]
50 public override void Step(IRuntimeContext context, double dt)
51 {
52     positionX.FloatValue = iotBotRigidBody.Position.x - offsetX.FloatValue;
53     positionY.FloatValue = iotBotRigidBody.Position.y - offsetY.FloatValue;
54     orientationYaw.FloatValue = Math.Atan2(iotBotRigidBody.Orientation.n[1], iotBotRigidBody.Orientation.n[0]) - offsetYaw.FloatValue;
55     angularVelocity.FloatValue = (orientationYaw.FloatValue - prevOrientation) / dt;
56     prevOrientation = orientationYaw.FloatValue;
57 }
58
59 [Runtime]
60 public override void Refresh(IRuntimeContext context)
61 {
62 }
63
64 [Runtime]
65 public override void Repaint()
66 {
67 }
68 }

```

Figure 5.34: NX MCD C# runtime behavior script for the determination of IOT Bot’s position and orientation

The NX MCD simulation is connected with SIMIT simulation using the coupling. It receives the wheel’s RPM actual values from SIMIT and send the position and orientation of the IOT Bot back to compare it with the similar values calculated using the IOT Bot’s kinematic model’s equations implemented in SIMIT.

5.6 Kinematic Models’ Functionality Verification

In order to verify the IOT Bot’s kinematic models’ functionality, the real values corresponding with the model’s inputs and outputs should be measured. Then, the model should be fed by the measured input values and its outputs are measured. In the end, these outputs are compared with their corresponding real values.

The actual IOT Bot’s wheels’ revolutions (RPM) are measured directly on the robot. The data are recorded from /iotbot/rpm topic (see the table 4.2) with their timestamps using ROS2 bag recording feature. These values

correspond with the forward kinematic model's inputs (see the equation 5.15) and the values assigned to wheels' speed controllers in NX MCD. The Pozyx platform providing the real-time localization data [19] is used for the measurement of IOT Bot's 2D position and orientation. The platform functionalities are implemented in ROS2 node and the data are recorded from /poseStamped topic similar way as robot's data. The values of the IOT Bot's x-position, y-position and orientation yaw correspond with the forward kinematic model's outputs (see the equation 5.16) and the values measured in NX MCD.

The Pozyx navigational system uses the static anchors which localizes the object moving in the 2D or 3D space using Ultra-Wideband (UWB) radio technology. The Pozyx company offers a creator kit. It is a complete UWB kit including HW and SW for real-time localization [20]. The HW core of this creator kit is a unified developer tag which can be used as the static anchor as well as the device mounted to localized dynamic object.



Figure 5.35: Pozyx creator kit (development tags on the left, boxes with static anchors on the right and wiring on top) [20]

The eight static anchors are mounted in different heights in machine hall (see its digital twin in the figure 5.33). In order to set up the real-time navigational system using the Pozy creator kit, it is necessary to assign the identification numbers (IDs) and the positions of mounted static anchors to

script for position and orientation determination (SW part of Pozy creator kit). The 3D scan (see the figure 5.36) of machine hall has been made for the determination of anchors' positions.

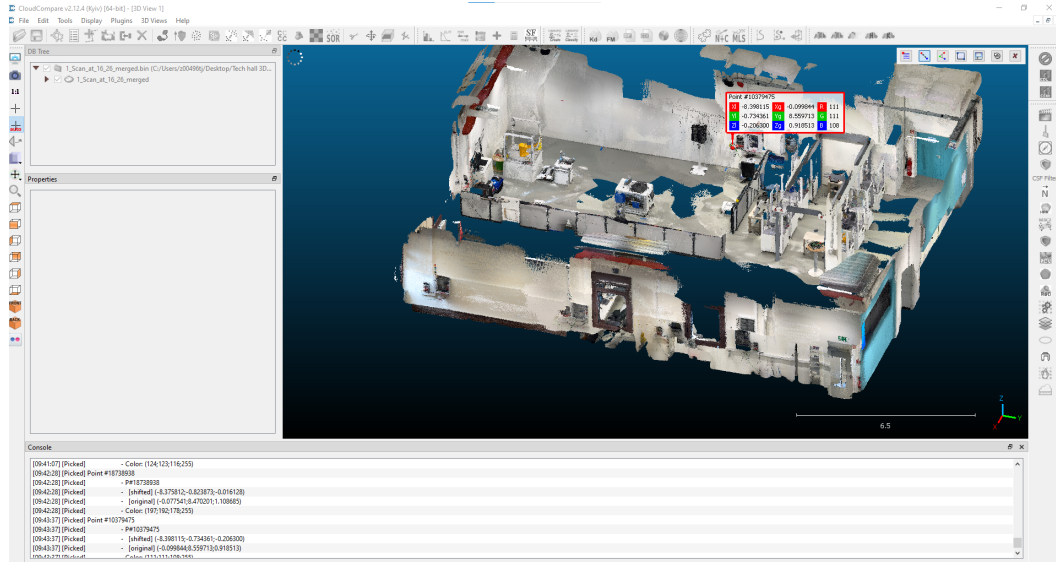


Figure 5.36: 3D scan of machine hall opened in CloudCompare open-source software: one of the anchors is marked and its position is determined

During the experiment, the functionality of IOT Bot's SIMIT forward kinematic model and IOT Bot's NX MCD mechatronic model (with implemented position and orientation determination) has been verified. For the purposes of the verification, two experiments has been done.

5.6.1 IOT Bot's Single Particular Movements

During the first experiment the IOT Bot does the separated movements. It means that only straight forward motion or rotation at spot is performed. The following four plots visualize the IOT Bot's actual wheels' revolution values during this experiment:



Figure 5.37: IOT Bot's front left wheel's revolutions (actual value ω_1)

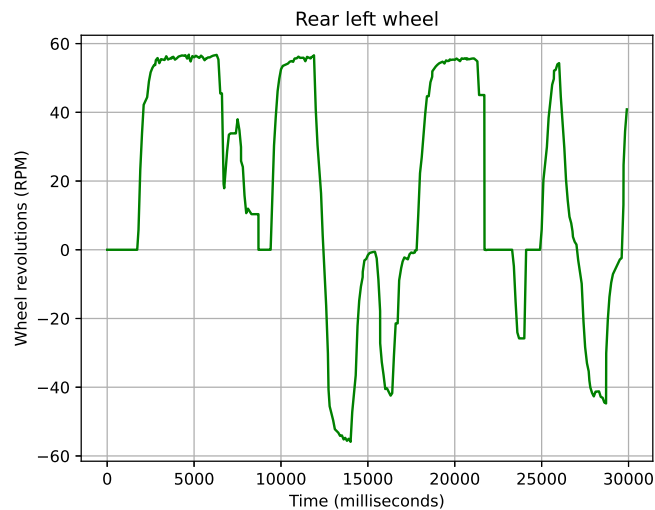


Figure 5.38: IOT Bot's rear left wheel's revolutions (actual value ω_2)



Figure 5.39: IOT Bot's front right wheel's revolutions (actual value ω_3)



Figure 5.40: IOT Bot's rear right wheel's revolutions (actual value ω_4)

In the figure 5.41 and 5.42, the IOT Bot's position in particular axes are plotted over time. The plot in the figure 5.43 visualized its position in 2D space.

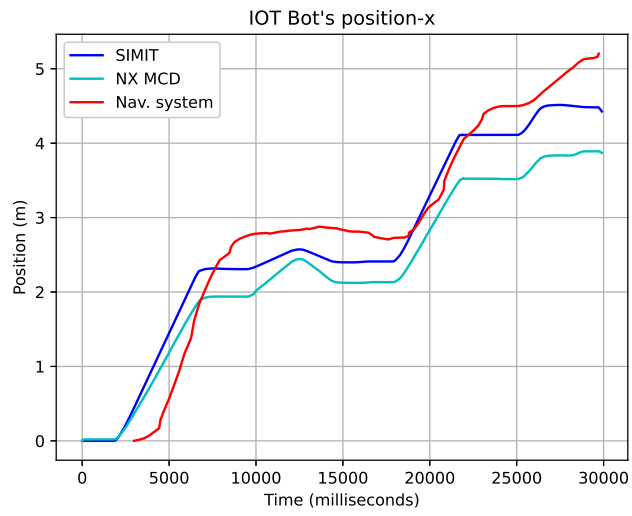


Figure 5.41: IOT Bot's position-x

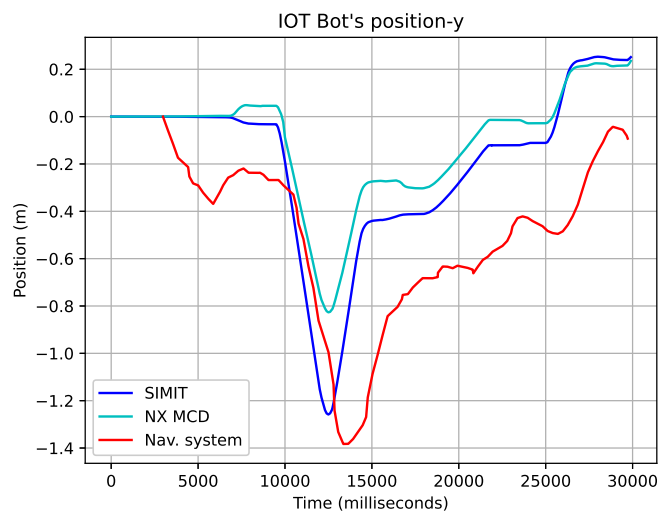


Figure 5.42: IOT Bot's position-y

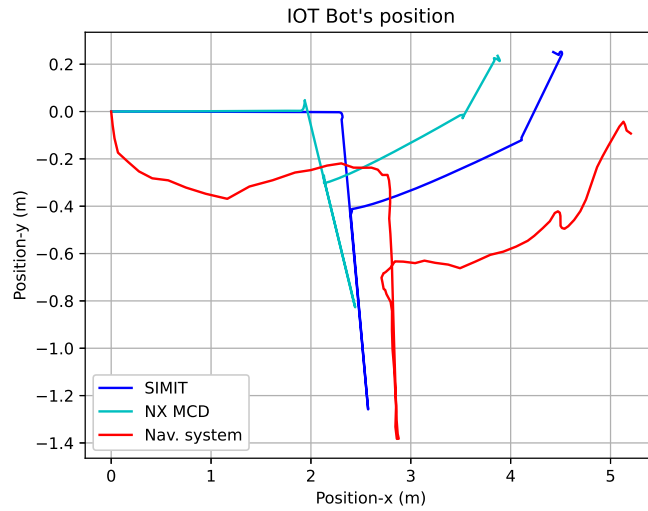


Figure 5.43: IOT Bot's position

As seen in the figures above, the values of IOT Bot's position over time from models are comparable with the real position values measured by navigational system. The difference is caused by the friction which is not part of the forward kinematic model implemented in SIMIT. It can also be observed that in reality the IOT Bot starts slightly turning to side during long linear movement. This behavior cannot be part of the idealized mathematical model since all four wheels are supposed to have the same behavior. In the NX MCD mechatronic model, the friction is modeled. However, using the dynamic friction coefficient of collision material the reality can never be fully described. Generally speaking, the position is determined by integration of velocity which means that even small initial constant deviation rises significantly during the time. The cause can be also the unluckily chosen input signal for this experiment which changes its values very quickly during the time. Then, the model built for ideal input signal is not able to respond quickly enough. Moreover, there can be some uncertainty in navigational system's measurements of position caused by interference of the radio signal.

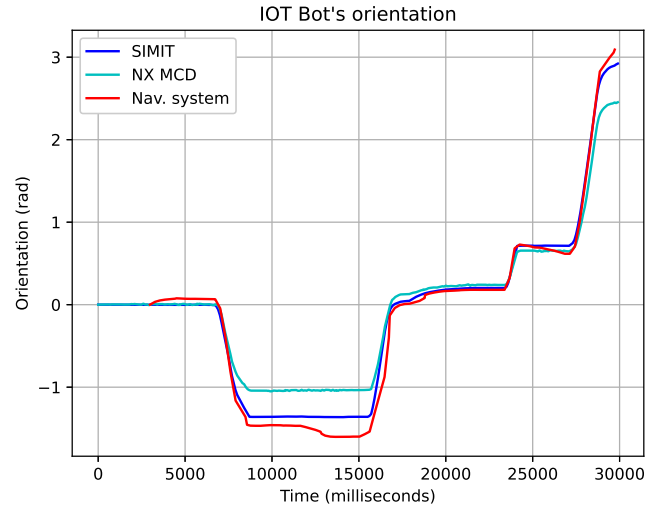


Figure 5.44: IOT Bot's orientation

The IOT Bot's orientation angle yaw is plotted in the figure 5.44. As it can be observed, the deviation of the orientation between reality and model is lower than in case of position. It can be caused by the way how the navigational system computes the orientation. It is computed by combining the data from three sensors available on development tag - accelerometers, magnetometer and Inertial Measurement Unit (IMU). Then, the measurement's error is lowered. The quaternions over time are got from the measurements and the yaw is calculated in each step. There is the rotation constant K_ω according to the equation 5.9 set in the model for this experiment. The lower deviation from reality has been reached than if the constants from 5.1 were set. It is probably caused by the testing on concrete surface which has a better friction properties than the surfaces where this constant has been identified on.

5.6.2 IOT Bot's Combination of Movements

During the second experiment, the combination of IOT Bot's movements was performed. The IOT Bot slightly drove out from its initial position and then started turning in circle. It means that both linear velocity and angular velocity were set and the absolute value of wheels' revolutions was different on each side of robot. See the plotted wheels' revolutions in the following four plots:

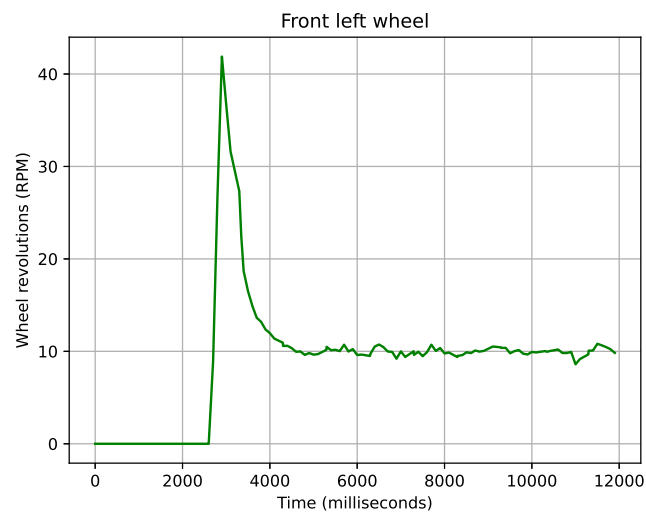


Figure 5.45: IOT Bot's front left wheel's revolutions (actual value ω_1)

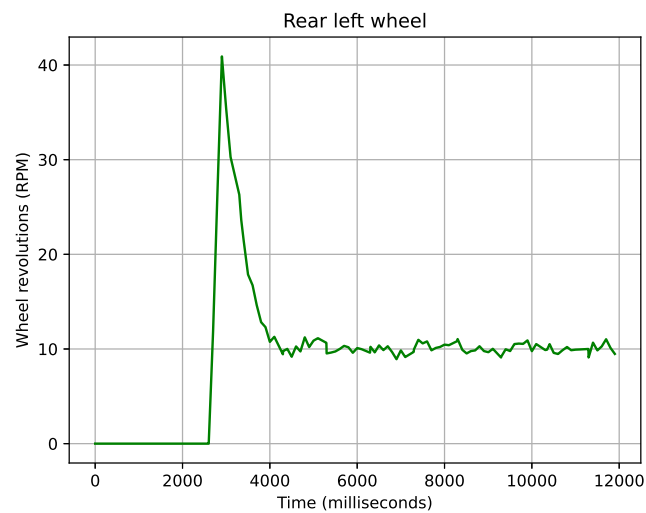


Figure 5.46: IOT Bot's rear left wheel's revolutions (actual value ω_2)

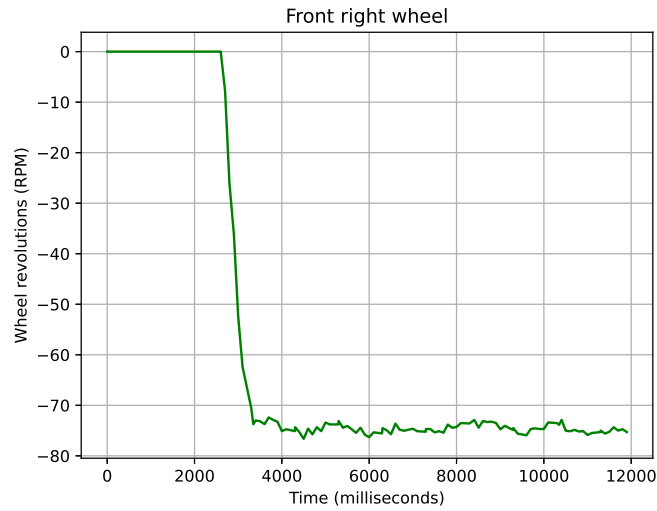


Figure 5.47: IOT Bot's front right wheel's revolutions (actual value ω_3)

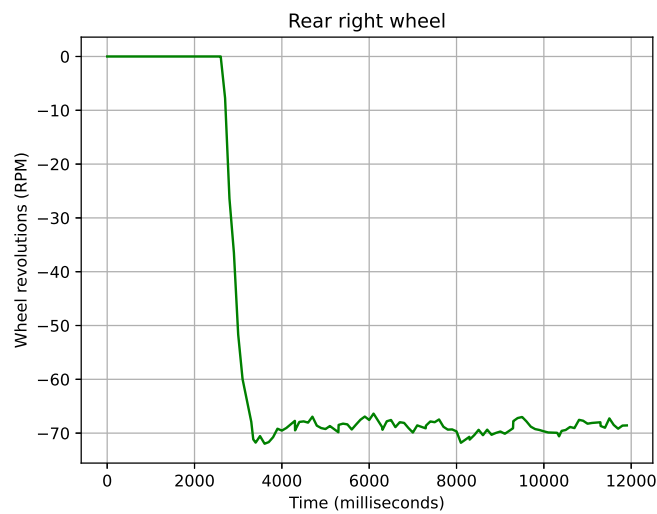


Figure 5.48: IOT Bot's rear right wheel's revolutions (actual value ω_4)

The following three plots visualized the comparison of IOT Bot's position calculated by models and measured by navigational system. Similarly, as in the previous experiment the values are comparable. In this case, the deviations are mostly caused by the friction. The rotation of the IOT Bot in circle is caused mainly by the outer wheels. The inner wheels almost don't rotate and only slide. Thus, the influence of the friction is significant

and causes the differences of the rotation circle radius and in the IOT Bot's position itself.

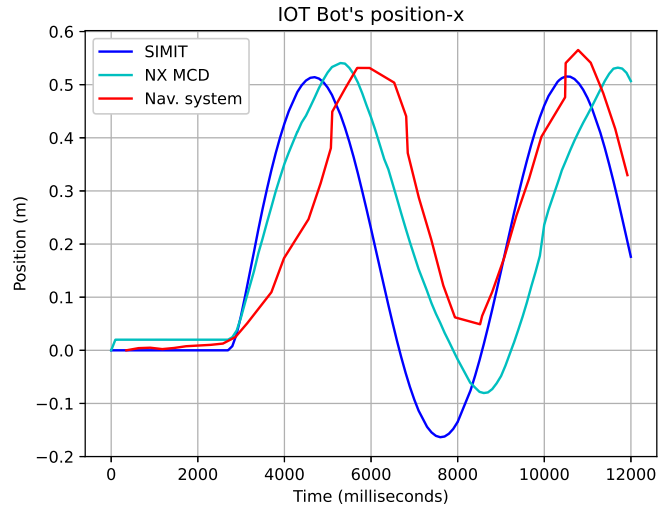


Figure 5.49: IOT Bot's position-x

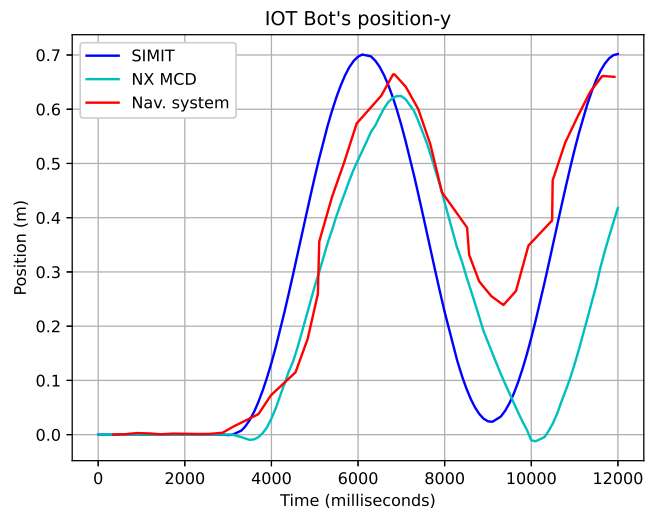


Figure 5.50: IOT Bot's position-y

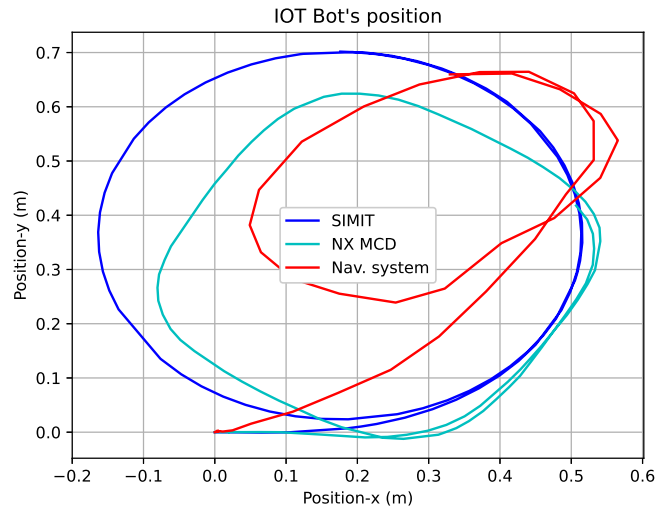


Figure 5.51: IOT Bot's position

Similarly, as in the first experiment, there are lower deviations in the orientation than in position. It is again thanks to the more precise measurement of orientation. The rotational constant K_ω is also set according to the equation 5.9 because this setup causes better results than the values from table 5.1.

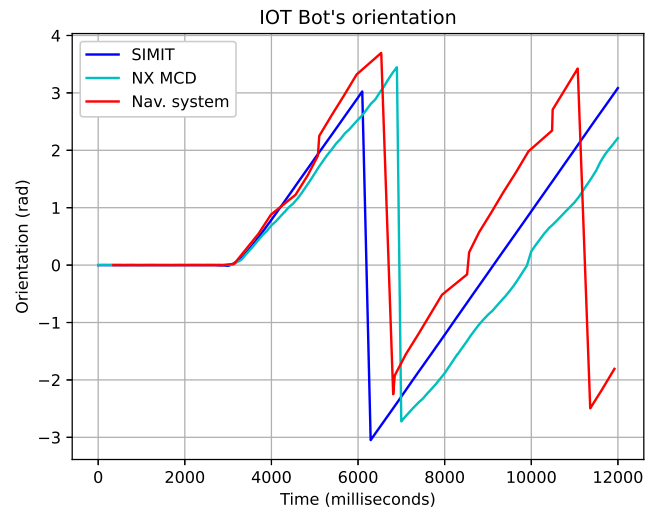


Figure 5.52: IOT Bot's orientation

6 SIL Simulation

6.1 SW Setup

As described in the chapter 1.2, the idea of SIL simulation is to have both controller and system simulated in one device and manage the data exchange between them using the signals' coupling or shared memory. The SIL simulation of IOT Bot runs in 64-bit Windows 10 operating system which runs in HP Zbook Fury 15 G7 (Intel(R) Core(TM) i7-10850H CPU @ 2.70GHz 2.71GHz, 48 GB RAM).

The IOT Bot's model is implemented in SIMIT (wheels' RPM dynamic models and forward kinematic model) and NX MCD (mechatronic model with calculation of IOT Bot's position and orientation). See the description of these models in the chapter 5.4 and 5.5. The model of controller is implemented in ROS2 Foxy distribution (see the ROS2 description in the chapter 3.3). The reverse engineering of the inverse instantaneous kinematic model's calculations controlling the real IOT Bot was done. Similar calculations are implemented in ROS2 node for the simulated IOT Bot's motion control.

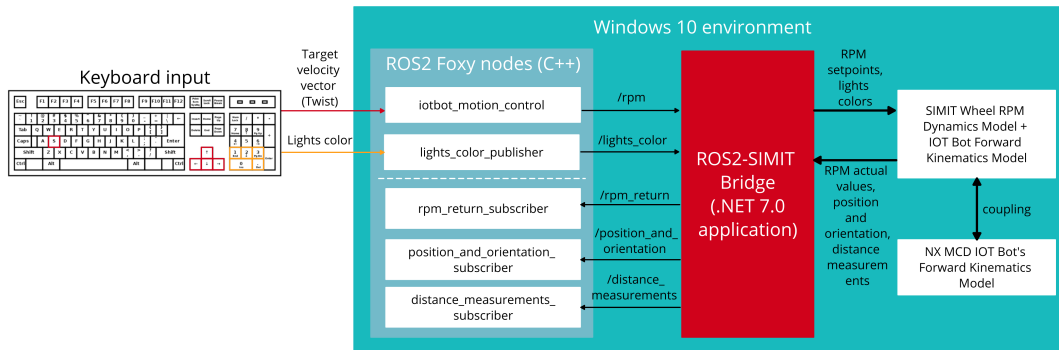


Figure 6.1: SIL simulation's SW setup

ROS2-SIMIT Bridge is implemented for the data exchange between ROS2 and SIMIT purposes. The details of this part are described in the chapter 6.2. See the diagram describing the SW setup of SIL simulation in the figure 6.1.

The IOT Bot's motion control is implemented in `iotbot_motion_control` node. The values of target velocity vector (Twist) are controlled by keyboard. The horizontal arrows control the IOT Bot's linear velocity (v_x) and the vertical arrows control its angular velocity (ω). If the key S is pressed, these velocities are set to zero and the IOT Bot stops. When the keyboard input is detected, the wheels' RPM setpoints are calculated based on these two velocities. Then, they are published using the `/rpm` topic.

Apart from the IOT Bot's motion the simulation of the IOT Bot's lights' control is implemented. There is a display changer implemented for the change of four lights' rigid bodies' (extracted from main IOT Bot's rigid body) colors in NX MCD. The control itself is covered by `lights_color_publisher` similar way as in motion control node. If the input from keyboard is detected, the number is published to NX MCD where the color of lights is changed.

There are also three subscribers implemented in ROS2 Foxy. The first receives the wheels' RPM actual values. The second subscribes two sets of IOT Bot's position and orientation values (x-position, y-position and orientation angle yaw). One set of values is calculated in SIMIT (using the forward kinematic model) and the other set comes from the NX MCD where these values are calculated in C# runtime behavior script (see description in the chapter 5.5). The third subscriber is used for the distance measurements' data reception. These data come from simulated distance sensors in NX MCD. See the list of the topics used for the communication between ROS and SIMIT in the table 6.1.

Description	Topic	Message type
Taget Motorspeed values	<code>/rpm</code>	<code>std_msgs/msg/Float32MultiArray</code>
Current Motorspeed values	<code>/rpm/return</code>	<code>std_msgs/msg/Float32MultiArray</code>
Light's color	<code>/lights_color</code>	<code>std_msgs/msg/Int16</code>
Position and orientation	<code>/position_and_orientation</code>	<code>std_msgs/msg/Float32MultiArray</code>
Distance measurements	<code>/distance_measurements</code>	<code>std_msgs/msg/Float32MultiArray</code>

Table 6.1: ROS2 topics used for the communication with SIMIT project

The IOT Bot's model implementation in SIMIT is similar as the one described in chapter 5.4. There are only small differences. The particular

wheels' RPM setpoints are received as outputs from ROS2 in the first chart (see figure 6.2). Since the IOT Bot's motion control is implemented in ROS2 the fourth chart is used as the user interface (see figure 6.3).

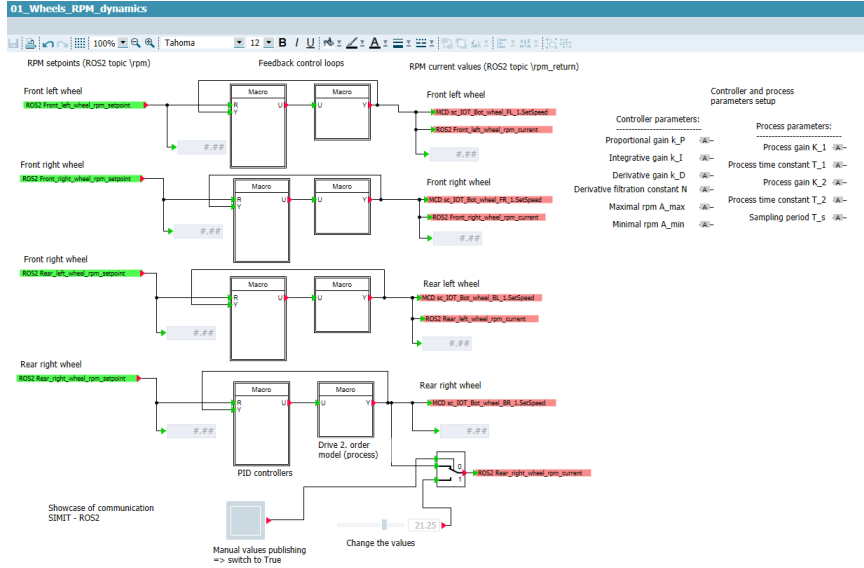


Figure 6.2: Model's implementation using SIMIT - first chart modified for SIL simulation purposes

Moreover, the fifth chart is used for the forwarding of signals from NX MCD to ROS2 and oppositely (see figure 6.4).

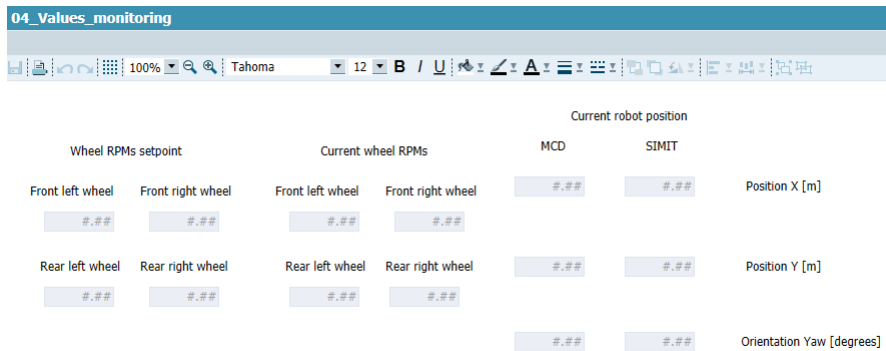


Figure 6.3: Model's implementation using SIMIT - fourth chart modified for SIL simulation purposes



Figure 6.4: Model’s implementation using SIMIT - fifth chart modified for SIL simulation purposes

6.2 ROS2-SIMIT Bridge

ROS2-SIMIT Bridge is a C# .NET 7.0 application developed in scope of this master thesis for the purposes of the full-duplex data flow between ROS2 nodes and SIMIT project. The environment used for the development of this application is Microsoft Visual Studio 2022. The flowchart of the application is drawn in the figure 6.5.

The ROS2-SIMIT Bridge application is built from two main parts - SIMIT and ROS2 part. The SIMIT part is built using SIMIT external embedded coupling application which is an extension of SIMIT and it uses Siemens.Simit.API.coupling library. This application contains two separated applications. The first separated application is used for the definition of inputs and outputs of external application which is the SIMIT project coupled with. This part is used for the definition of variables which are supposed to be exchanged between ROS2 and SIMIT during the IOT Bot’s SIL simulation. The second separated part is for the purposes of data exchange between SIMIT and its coupling extension. This part is used in ROS2-SIMIT Bridge application. The SIMIT part’s flowchart can be seen in the figure 6.6.

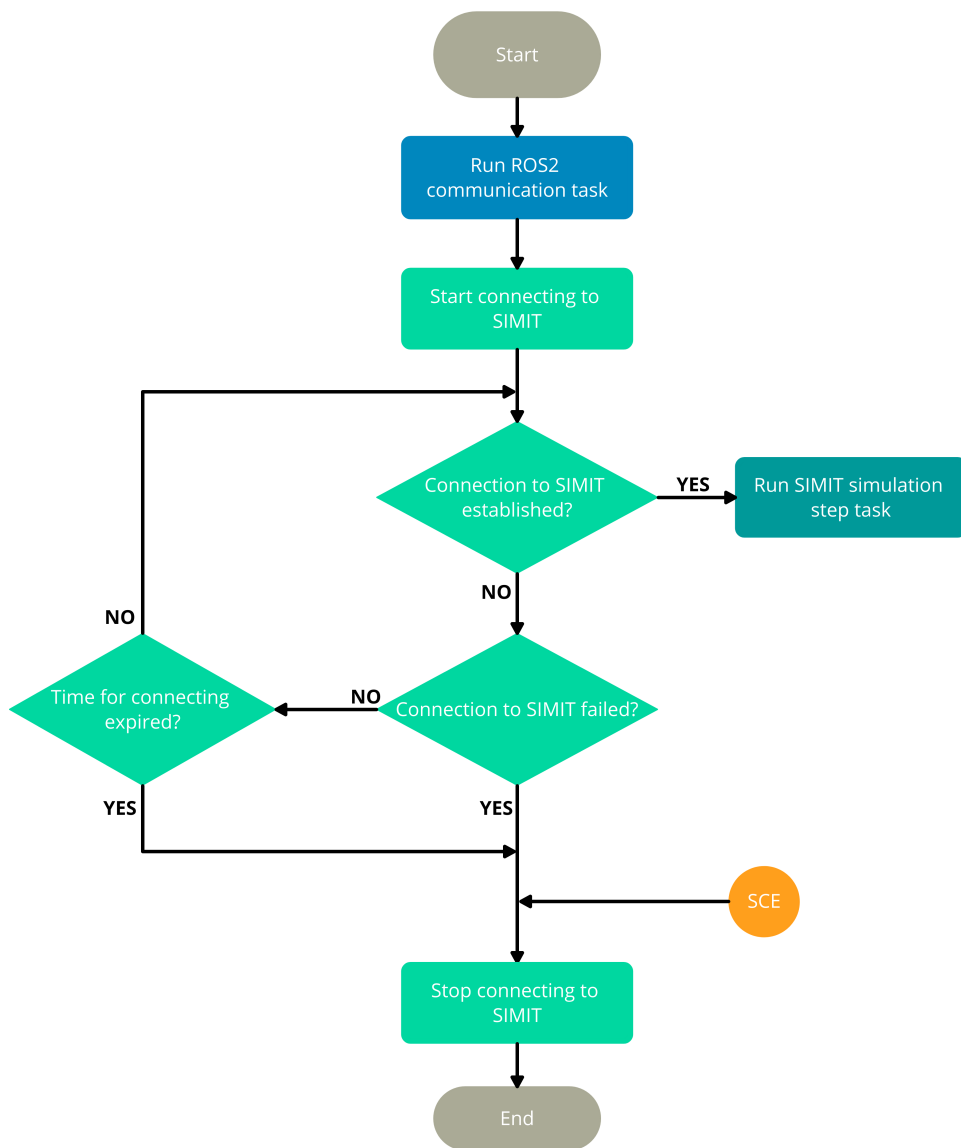


Figure 6.5: ROS2-SIMIT Bridge flowchart

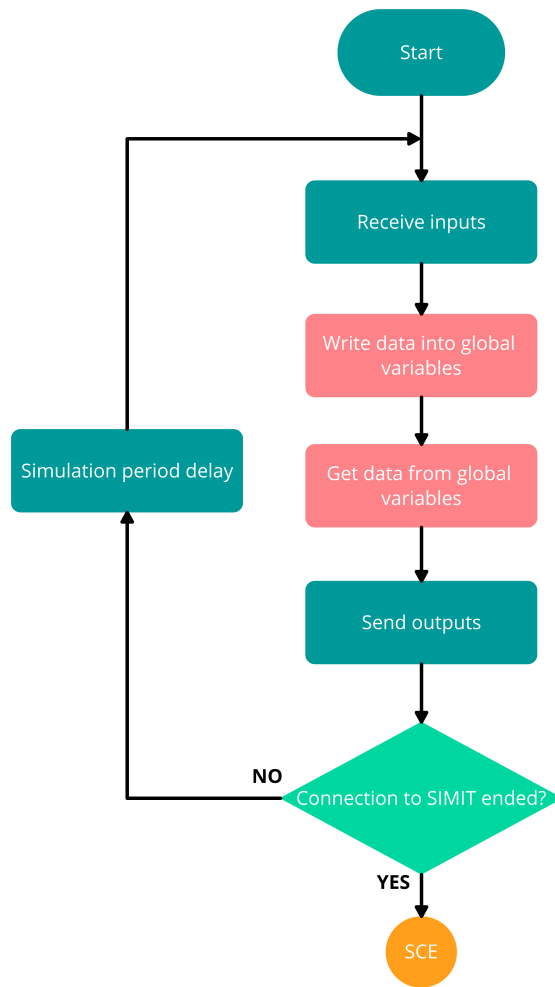


Figure 6.6: ROS2-SIMIT Bridge flowchart: SIMIT part

The second part of the ROS2-SIMIT Bridge application (ROS2 part) is built using `rclnet` which is a .NET wrapper allowing the interaction of .NET applications with ROS2 applications. This library has been released in 2023. The supported distributions of ROS2 are Foxy and Humble. It supports Windows and Linux Ubuntu operating system. In order to run it successfully, the .NET 7.0 and higher is needed [21]. See the flowchart of ROS2 part in the figure 6.7.

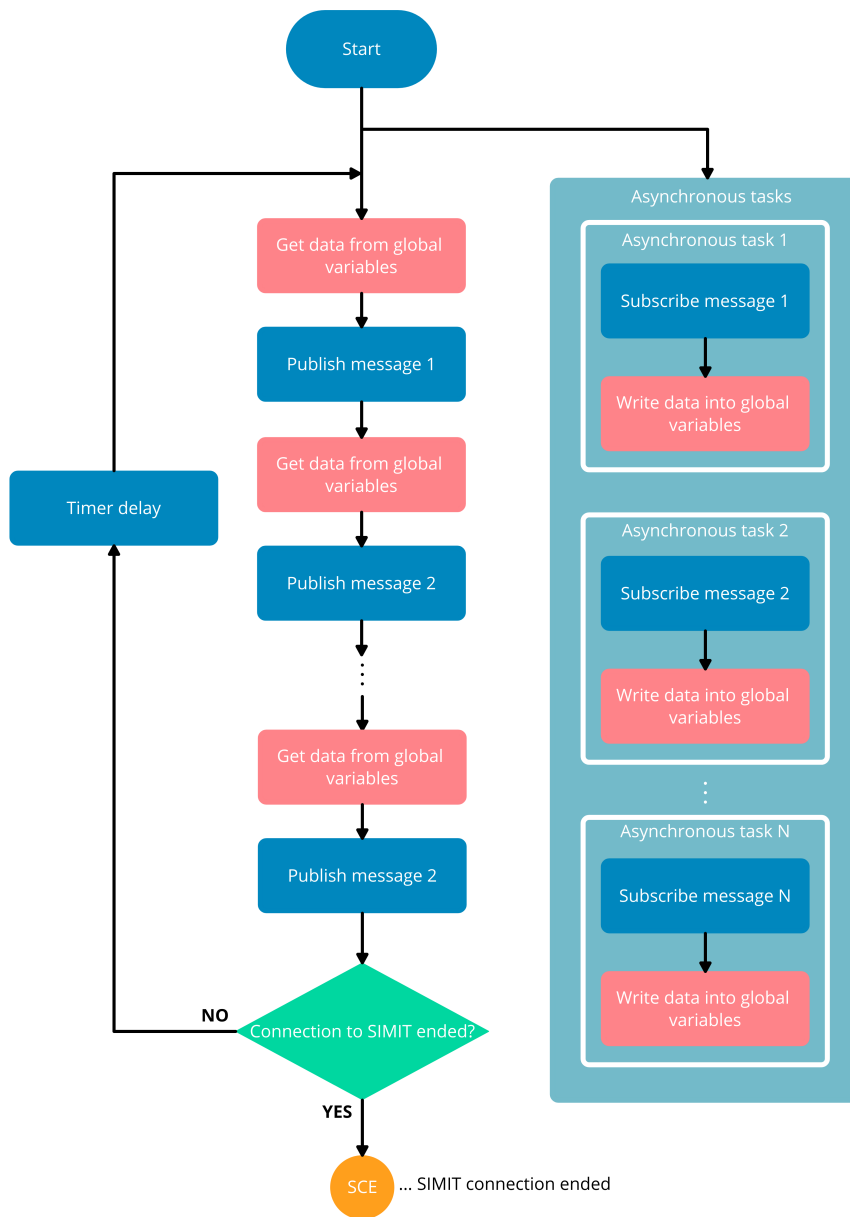


Figure 6.7: ROS2-SIMIT Bridge flowchart: ROS2 part

6.3 SIL Simulation Testing

In order to manage the SIL simulation setup, at least two applications should be opened - ROS2-SIMIT connector in Visual Studio and IOT Bot's model in SIMIT. Additionally, the mechatronic model in NX MCD can be opened. In this case, the coupling with NX MCD in SIMIT should be activated and the connection with SIMIT in NX MCD should be allowed.

To run the SIL simulation, the ROS2-SIMIT Bridge should be turned on first. Then, the SIMIT simulation should be started and the communication between the SIMIT and ROS2-SIMIT Bridge is established. In case of the NX MCD mechatronic model being opened and everything being set according to the previous paragraph, both NX MCD simulation and communication between SIMIT and NX MCD is established automatically. Then, the corresponding ROS2 publishers and subscribers can be run in order to manage the data exchange according to the diagram visualized in the figure 6.2 using the topics described in table 6.1.

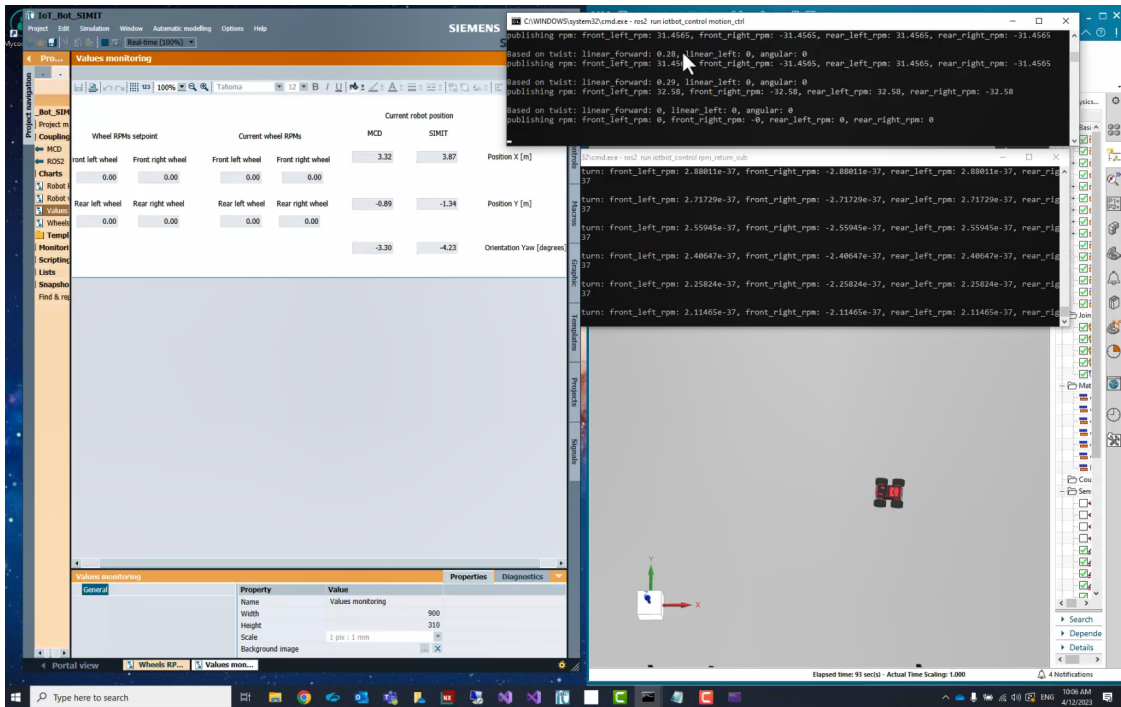


Figure 6.8: SIL simulation: screenshot from testing

The screenshot from the running SIL simulation can be seen in the figure 6.8. There are SIMIT and NX MCD model running, the Twist is published from ROS2 node to control the IOT Bot's motion and the actual wheels' revolutions (RPM) are continuously received by the ROS2 subscriber. To test the SIL simulation functionality several IOT Bot's drives including separated and combined movements were done.

7 HIL Simulation

7.1 HW and SW Setup

As described in chapter 1.3, the main idea of HIL simulation is controlling the simulated system by real controller. The simulation of IOT Bot (system) has similar setup as in case of SIL simulation - SIMIT wheels' RPM dynamic models with robot's forward kinematic model and the NX MCD mechatronic model. It runs in the same laptop as the one described in the chapter 6.1. The simulated system is controlled by the same controller as the real IOT Bot. The control software runs in Linux Debian OS which runs in SIMATIC IOT2050 (described in the chapter 4.1). Each application is implemented in its own docker container. One docker container (iotbot_basis) includes the ROS2 nodes for the IOT Bot's motion control and reading data from the sensors mounted on the extension-shield (see the chapter 4.2). Another docker container contains Node-RED environment where it is possible to deal with the data from ROS2 topics (see the table 4.1, 4.2 and 4.3).

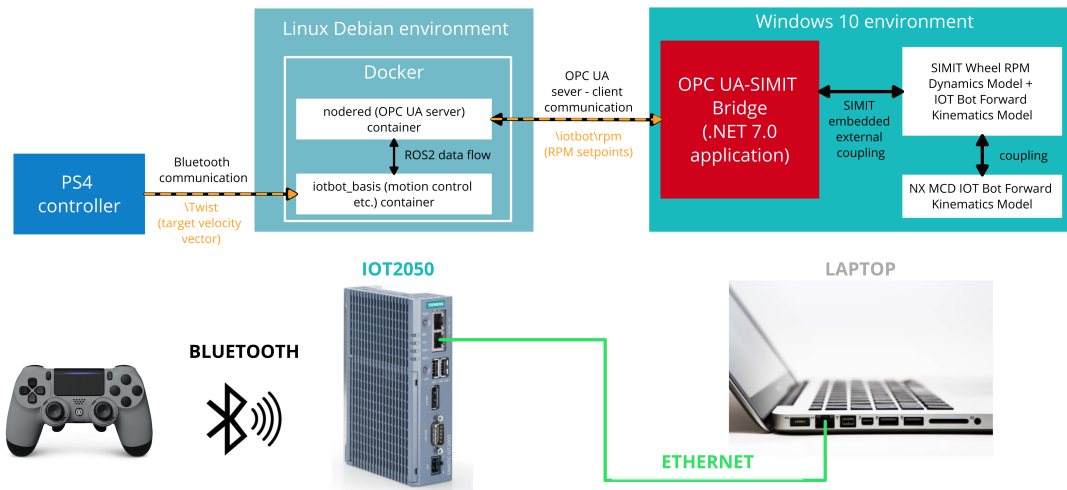


Figure 7.1: HIL simulation's HW and SW setup

As mentioned in the chapter 4.1, the real robot can be controlled by the PS4 controller. The provided Twist (target velocity vector) is sent from

PS4 controller via Bluetooth to IOT2050 where the inverse instantaneous kinematic model implemented in ROS2 node calculates the wheels' RPM setpoints. Then, these setpoints are forwarded to extension-shield via UART where they are used as the inputs to motors' inner control loops and manage the robot's motion control.

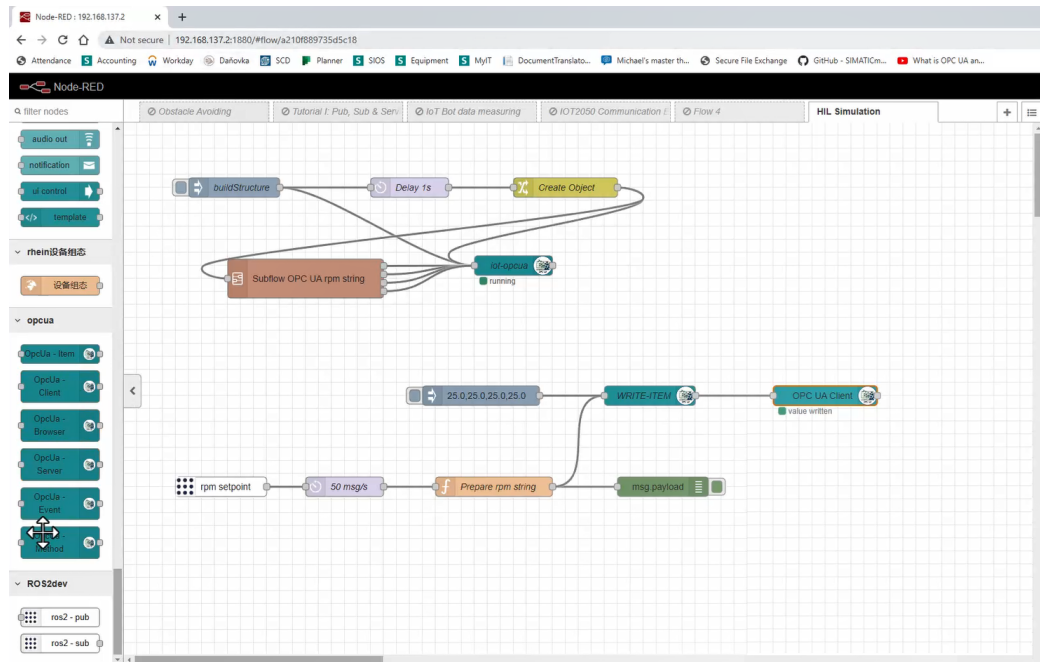


Figure 7.2: NODE-Red: reading the RPM setpoints from ROS2 topic and their continuous publishing via configured OPC UA server

Similarly, it is set up in case of HIL simulation. The difference is that the communication between the real controller and simulated system is managed by OPC Unified Architecture (OPC UA). It is standard for data exchange which uses the server-client model of communication. There is the OPC UA server configured in Node-RED using the node-red-contrib-opcua node which allows to read, write, subscribe or browse OPC UA server [22]. The required IOT Bot's wheels' revolutions (RPM setpoints) are read from corresponding ROS2 topic and then published via configured OPC UA server. The data from OPC UA server are subscribed by OPC UA client defined in OPC UA-ROS2 Bridge which description is given in the chapter 7.2. The OPC UA-SIMIT Bridge forwards the required motors' RPM to the input of simulated IOT Bot's inner motors' control loops. See the diagram describing the IOT Bot's HIL simulation's HW and SW setup in the figure 7.1.

7.2 OPC UA-SIMIT Bridge

OPC UA-SIMIT Bridge is a C# .NET 7.0 application having similar structure as ROS2-SIMIT Bridge described in the chapter 6.2. The difference is that instead of task for the communication with ROS2, the task which defines the OPC UA client and starts the continuous reading of required wheels' RPM is run. The .NET library `Opc.UaFx.Client` [23] is used for communication with OPC UA server. In order to reach the OPC UA server using this library is necessary to provide the right IP address of SIMATIC IOT2050 and the same port number as the OPC UA server defined in Node-RED has.

7.3 HIL Simulation Testing

In order to manage the HIL simulation setup is necessary to have the laptop and IOT2050 in the same network. The OPC UA Bridge in Visual Studio should be opened as well as the SIMIT simulation. The NX MCD mechatronic model can be opened optionally because its presence is not necessary for overall simulation functionality. Dealing with the NX MCD model and its connection with SIMIT is same as in case of SIL simulation (described in the chapter 6.3).

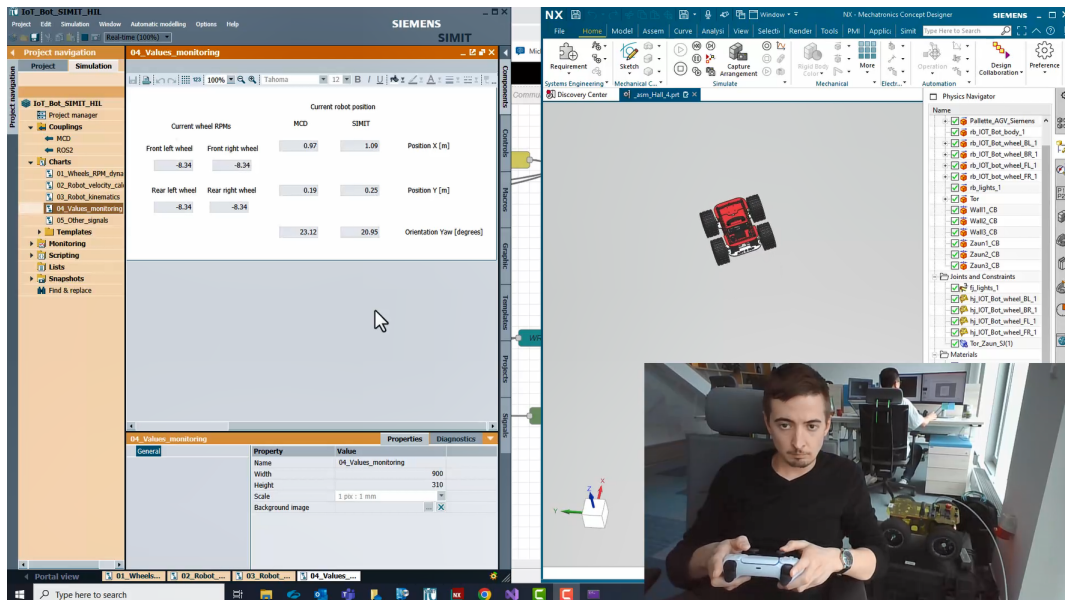


Figure 7.3: HIL simulation: screenshot and webcam shot from testing

To run the HIL simulation, the IOT2050 should be turned on first. Then, the OPC UA-SIMIT Bridge should be run. In case if the IP address and port are set correctly on both server and client side of OPC UA, the communication with OPC UA server is established. As soon as the SIMIT simulation is started, the communication with SIMIT is established as well.

If the robot is placed on some frame and the NX MCD simulation is connected as well, then it is possible to observe that the simulated wheels rotate similarly as the wheels in simulation. This setup was used during the testing. In case of wireless communication between IOT2050 and laptop via router, it would be even possible to drive the real and simulated robot at the same time. If the virtual environment in NX MCD would correspond with the real one, it would be nice to observe the similarity of IOT Bot's motion in virtual reality and real world.

To test the HIL simulation several drives with IOT Bot in virtual machine hall were made. As mentioned, the real IOT Bot stand at frame, the wheels rotated and the virtual IOT Bot was moving in the digital twin of machine hall. During this motion, the current position and orientation were calculated by implemented forward kinematic model in SIMIT and also determined by C# runtime behavior script in NX MCD. See the screenshot and photo from webcam during testing in the figure 7.2.

8 Conclusion and Outlook

The aim of this thesis was the development of HIL simulation of certain mobile robotic system AGV platform IOT Bot. The mathematical model describing the robot's position and orientation in 2D space was built and implemented using SIMIT. In order to model the dynamic part of the robot, the models of its wheels' revolutions inner control loops were implemented in SIMIT as well. For the purposes of comparison of the calculated position and orientation by implemented mathematical model, the mechatronic model of robot in NX MCD with the position and orientation determination functionality was created. This model is also used for the 3D visualization. The results of both models were compared with the data from real navigational system. The calculated values of position and orientation by both models are found as comparable with the real data from navigational system. The deviations are caused by the friction, improperly chosen input signal for testing and by uncertainty of navigational system's radio signal.

Both models were used for the development of two robots' simulation concepts. The first concept is SIL simulation. The ROS2 nodes controlling the robot were implemented in Windows ROS2 Foxy distribution and the whole simulation was performed in the same machine. The ROS2-SIMIT Bridge was developed for the purposes of full-duplex communication between ROS2 nodes and SIMIT simulation. The second concept is HIL simulation. In this case, the interface allowing the controlling of the simulated robot same way as the real one was configured. The OPC UA protocol is used for these purposes. The real robot control setup is kept and the control signals are simply transferred to simulation. Thus, it is possible to perform the motion of real and simulated robot at the same time. Thanks to the 3D visualization in NX MCD it is possible to observe similarity between simulated and real robot's movements in case of the virtual environment being similar to the real one.

The developed SIL simulation represents an opportunity to perform several tests of the IOT Bot's applications without the need to have a physical robot available and without the risk of any damages of the real robot. The same opportunity is represented by HIL setup in case if only robot's control HW (SIMATIC IOT2050) without the robot itself being connected with the simulation. The ROS2-SIMIT Bridge application has a potential to be

used in many other robotic and AGV simulations because the SIMIT itself can be connected with each Siemens simulation software such as PLCSIM Advanced, Plant Simulation or Simcenter Amesim. The kinematic mathematical model itself can be implemented in another platform. It can be used in some more sophisticated navigational algorithm such as Kalman filter where both data from sensors and predictive data are used to estimate the position and orientation of an object. Moreover, it can be used in virtual sensor application running in Industrial Edge. This application is used in cases of some value cannot be measured directly and should be estimated using some mathematical model. If the navigational system is missing, the position and orientation of the mobile robot can be determined using this application.

Bibliography

- [1] A. Shadravan and H. Parsaei, “Impacts of industry 4.0 on smart manufacturing,” 2023.
- [2] (2022) News. [Online]. Available: <https://evocortex.org/category/news/>
- [3] (2020) Boston dynamics’ spot isn’t quite the terrifying robot hunter you think it is. [Online]. Available: <https://www.theverge.com/2020/2/19/21144648/boston-dynamics-spot-robot-mass-state-police-trial-issues>
- [4] (2020) Difference between amr and agv. [Online]. Available: <https://www.geeksforgeeks.org/difference-between-amr-and-agv/>
- [5] M. L. Elwin, B. Strong, R. A. Freeman, and K. M. Lynch, “Human-multirobot collaborative mobile manipulation: The omnid mocobots,” *IEEE Robotics and Automation Letters*, 2022.
- [6] M. Niang, B. Riera, A. Philippot, J. Zaytoon, F. Gellot, and R. Coupât, “A methodology for automatic generation, formal verification and implementation of safe plc programs for power supply equipment of the electric lines of railway control systems,” *Computers in Industry*, 2020.
- [7] C. Kleijn, “Introduction to hardware-in-the-loop simulation,” *Controllab*, 2021.
- [8] M. Goubej, M. Švejda, and M. Schlegel, “Uvod do mechatroniky, robotiky a systémů řízení pohybu,” *University of West Bohemia in Pilsen*, 2012.
- [9] J. Melichar and M. Goubej, “Lineární systémy 1,” *University of West Bohemia in Pilsen, Czech Republic*, 2017.
- [10] J. Melichar and M. Goubej, “Lineární systémy 2,” *University of West Bohemia in Pilsen, Czech Republic*, 2019.
- [11] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, 2022.
- [12] S. May. (2022) Setting up the iot2050 with example image. [Online]. Available: <http://wiki.ros.org/iotbot>

- [13] (2022) Iot edge for insights. [Online]. Available: <https://iot2050.com/>
- [14] (2022) Setting up the iot2050 with example image. [Online]. Available: <https://www.gothightech.com/resources/pdf/Manual%20-%20IOT2050%20Setting%20up%20with%20Example%20Image.pdf>
- [15] K. Kozłowski and D. Pazderski, “Modeling and control of a 4-wheel skid-steering mobile robot,” *University of Technology in Poznań, Poland*, 2004.
- [16] S. May and A. Pastore, “Hierarchical multi robot fleet architecture with a kinematics adaptive drive system,” *Technical College in Nuremberg, Germany*, 2023.
- [17] L. Caracciolo, A. D. Luca, and S. Iannitti, “Trajectory tracking control of a four-wheel differentially driven mobile robot,” *International Conference on Robotics and Automation*, 1999.
- [18] M. Hamcke, “Scalable architecture for identifying and optimizing the control parameters of mobile robot systems at runtime,” *Technical College in Nuremberg, Germany*, 2022.
- [19] (2023) Pozyx. [Online]. Available: <https://www.pozyx.io/>
- [20] (2023) Pozyx creator kit. [Online]. Available: <https://www.pozyx.io/creator-one-kit>
- [21] (2023) rclnet. [Online]. Available: <https://github.com/noelx/rclnet>
- [22] (2023) node-red-contrib-opcua. [Online]. Available: <https://flows.nodered.org/node/node-red-contrib-opcua>
- [23] (2023) Opc.uaFx.client. [Online]. Available: <https://www.nuget.org/packages/Opc.UaFx.Client>

List of Figures

1.1	Example of the wheeled robot (Evocortex Evo Carrier) [2] and the robot equipped with legs (Boston Dynamics Spot) [3]	11
1.2	The difference between AMR and AGV [4]	12
1.3	Omnid macobot description and use case of human-robot collaboration during the manipulation with payload [5]	13
1.4	Software in the Loop (SIL) simulation concept [6]	14
1.5	Hardware in the Loop (HIL) simulation concept [6]	14
1.6	Analogy of two connections: control system - plant simulator and control system - plant [7]	15
2.1	Two coordinate systems and their mutual translation	16
2.2	Planar robotic arm drawing [8]	18
2.3	Differential drive mobile robot drawing	19
2.4	RLC electric circuit scheme	22
2.5	RC electric circuit scheme	25
2.6	Typical shapes of the first-order and second-order systems' step responses	26
2.7	feedforward and feedback control circuit both including Controller (C) and Process (P)	27
2.8	PID controller with anti-windup compensation circuit [10]	28
3.1	SIMATIC Machine Simulator scheme	32
3.2	HIL simulation setup using SIMIT Unit	33
3.3	Continuous communication between ROS2 nodes using topic and service [11]	35
3.4	One-shot communication between ROS2 nodes using action [11]	35
4.1	IOT Bot - differential drive variant (on the left) and mecanum drive variant (on the right) [12]	37
4.2	Block scheme of the motor's inner control loop	38
5.1	IOT Bot kinematics	41
5.2	IOT Bot's front left wheel's revolutions - ω_1	46
5.3	IOT Bot's rear left wheel's revolutions - ω_2	47
5.4	Evaluation of $\omega_1 = \omega_2$ (see equation 5.3)	47
5.5	Average of ω_1 and ω_2 : $\omega_L = \frac{\omega_1 + \omega_2}{2}$	48
5.6	IOT Bot's front right wheel's revolutions - ω_3	48

5.7	IOT Bot's rear right wheel's revolutions - ω_4	49
5.8	Evaluation of $\omega_3 = \omega_4$ (see equation 5.3)	49
5.9	Average of ω_3 and ω_4 : $\omega_L = \frac{\omega_3 + \omega_4}{2}$	50
5.10	IOT Bot's angular velocity	50
5.11	Identified K_ω (see the equation 5.7, 5.8 and 5.9)	52
5.12	IOT Bot's front left wheel's revolutions (setpoint ω_1^* and actual value ω_1)	54
5.13	IOT Bot's rear left wheel's revolutions (setpoint ω_2^* and actual value ω_2)	54
5.14	IOT Bot's front right wheel's revolutions (setpoint ω_3^* and actual value ω_3)	55
5.15	IOT Bot's rear right wheel's revolutions (setpoint ω_4^* and actual value ω_4)	55
5.16	IOT Bot's angular velocity	56
5.17	Confirmation of $\omega_1 = \omega_2$ (see equation 5.3) - using the setpoints	56
5.18	Confirmation of $\omega_3 = \omega_4$ (see equation 5.3) - using the setpoints	57
5.19	Evaluation of $\omega_1 = \omega_2$ (see equation 5.3) - using the actual values	57
5.20	Evaluation of $\omega_3 = \omega_4$ (see equation 5.3) - using the actual values	58
5.21	Average of ω_1 and ω_2 : $\omega_L = \frac{\omega_1 + \omega_2}{2}$	58
5.22	Average of ω_3 and ω_4 : $\omega_L = \frac{\omega_3 + \omega_4}{2}$	59
5.23	Identified K_ω (see the equation 5.7, 5.8 and 5.9)	59
5.24	Block scheme of the motor's inner control loop - modeled parts	60
5.25	Model's implementation using SIMIT - first chart	61
5.26	Model's implementation using SIMIT - PID controller's macro	62
5.27	Model's implementation using SIMIT - process's macro	62
5.28	Model's implementation using SIMIT - second chart	63
5.29	Model's implementation using SIMIT - third chart	64
5.30	Model's implementation using SIMIT - fourth chart	65
5.31	Model's implementation using SIMIT - fifth chart	65
5.32	IOT Bot's assembly in NX MCD	66
5.33	Machine hall's digital twin assembly in NX MCD	67
5.34	NX MCD C# runtime behavior script for the determination of IOT Bot's position and orientation	68
5.35	Pozyx creator kit (development tags on the left, boxes with static anchors on the right and wiring on top) [20]	69
5.36	3D scan of machine hall opened in CloudCompare open-source software: one of the anchors is marked and its position is determined	70

5.37	IOT Bot's front left wheel's revolutions (actual value ω_1) . . .	71
5.38	IOT Bot's rear left wheel's revolutions (actual value ω_2) . . .	71
5.39	IOT Bot's front right wheel's revolutions (actual value ω_3) . . .	72
5.40	IOT Bot's rear right wheel's revolutions (actual value ω_4) . . .	72
5.41	IOT Bot's position-x	73
5.42	IOT Bot's position-y	73
5.43	IOT Bot's position	74
5.44	IOT Bot's orientation	75
5.45	IOT Bot's front left wheel's revolutions (actual value ω_1) . . .	76
5.46	IOT Bot's rear left wheel's revolutions (actual value ω_2) . . .	76
5.47	IOT Bot's front right wheel's revolutions (actual value ω_3) . . .	77
5.48	IOT Bot's rear right wheel's revolutions (actual value ω_4) . . .	77
5.49	IOT Bot's position-x	78
5.50	IOT Bot's position-y	78
5.51	IOT Bot's position	79
5.52	IOT Bot's orientation	79
6.1	SIL simulation's SW setup	80
6.2	Model's implementation using SIMIT - first chart modified for SIL simulation purposes	82
6.3	Model's implementation using SIMIT - fourth chart modified for SIL simulation purposes	82
6.4	Model's implementation using SIMIT - fifth chart modified for SIL simulation purposes	83
6.5	ROS2-SIMIT Bridge flowchart	84
6.6	ROS2-SIMIT Bridge flowchart: SIMIT part	85
6.7	ROS2-SIMIT Bridge flowchart: ROS2 part	86
6.8	SIL simulation: screenshot from testing	87
7.1	HIL simulation's HW and SW setup	89
7.2	NODE-Red: reading the RPM setpoints from ROS2 topic and their continuous publishing via configured OPC UA server	90
7.3	HIL simulation: screenshot and webcam shot from testing	91

List of Tables

4.1	ROS2 node for robot's motion calculation (its inverse kinematics): topics [12]	39
4.2	ROS2 node for communication between the microprocessor and IOT2050: topics [12]	40
4.3	ROS2 node for communication between the microprocessor and IOT2050: services [12]	40
5.1	Identified K_ω on different surfaces	52
6.1	ROS2 topics used for the communication with SIMIT project	81