NTNU
Norwegian University of
Science and Technology

Digital Twins of Mobile Manipulators

Focused on KUKA KMR iiwa

Jørgen Usterud Myrvold

2022-05-12

Preface

This specialization project is part of my master's degree in Engineering & ICT at NTNU and this paper concludes the project at the Department for Mechanical and Industrial Engineering at NTNU in the spring of 2022. The project revolves around digital twins of mobile manipulators with the goal of gathering as much knowledge as possible before the master's thesis will be written in the fall of 2022.

The focus of this project has been to explore possibilities for digital twins of mobile manipulators and see what has previously been done in the field. The project also lays a theoretical foundation within the control of manipulators and mobile platforms.

I want to thank my supervisor Lars Tingelstad for guidance and support throughout this project.

Summary

Mobile manipulators are an essential part of Industry 4.0 where increased automation and customization is in focus. One major issue of such robots is the development cost. Digital twins are replicas of the actual robot in a virtual simulation environment which allows for testing of robotic applications without the need for a physical robot, thus lowering the development cost significantly.

Recent developments in computer hardware have enabled highly realistic simulators that narrow the sim-to-real gap significantly compared to earlier simulators. This provides more realistic development environments in addition to enabling possibilities for machine learning and synthetic data generation that can be used for training such models.

In addition to the digital twins, the project also presents fundamental kinematics for controlling manipulators and holonomic wheeled robots using mecanum wheels such as the KUKA KMR iiwa. For controlling mobile robots, navigation and mapping are also essential, and this project presents the theory behind this as well.

Contents

Preface i

Summary iii

1 Introduction 1
- 1.1 Background and motivation . . . 1
- 1.2 Problem description . . . 2
- 1.3 Outline . . . 2

2 Robot kinematics 3
- 2.1 Poses and frames . . . 3
- 2.2 Manipulator kinematics . . . 4
  - 2.2.1 Kinematics of a 7-dof manipulator . . . 6
- 2.3 Wheeled robots . . . 9
  - 2.3.1 Kinematic model for a four mecanum wheeled mobile robot 10

3 SLAM and Navigation 15
- 3.1 Maps . . . 15
- 3.2 SLAM – Simultaneous Localization and Mapping . . . 17
  - 3.2.1 Kalman Filter . . . 18
  - 3.2.2 Particle Filter . . . 19
  - 3.2.3 Graph-based Optimization . . . 20
- 3.3 Navigation . . . 21
  - 3.3.1 Costmap . . . 22
  - 3.3.2 Adaptive Monte Carlo localization . . . 23
  - 3.3.3 Dynamic Window Approach . . . 24

4 Digital twins 25
- 4.1 Robotic simulators . . . 26
  - 4.1.1 Simulator comparison . . . 28
- 4.2 Sim-to-Real . . . 29
  - 4.2.1 How to narrow the sim-to-real gap . . . 29

5 Conclusion and Further Work **33**
5.1 Conclusion 33
5.2 Further work 34

List of Figures

2.1 6 examples of common robot joints [18, p. 16] 5
2.2 Example of an anthropomorphic manipulator 6
2.3 Horizontally mounted LBR iiwa with corresponding DH frames illustrated [7] 7
2.4 Example of omni wheel (left) and mecanum wheel (right) [18, p. 514] 9
2.5 Common configurations of omni wheels (left) and mecanum wheels (right) to achieve holonomic drive [18, p. 516] 10
2.6 Here the variables used to derive the inverse kinematics for the robot are shown [10] 11

3.1 Example of an OGM and associated `.yaml` file created by Turtlebot 3 [27, p. 325] 16
3.2 Graphical representation of the SLAM problem [30, p. 1154] . . . 17
3.3 Illustration of the concept of the Kalman filter [27, p. 334] 19
3.4 Example of a costmap corresponding to the map in Figure 3.1a [27, p. 338] 22
3.5 Relationship between the distance to the object and the costmap value [27, p. 356] 23

4.1 Intuition behind domain randomization [35] 30
4.2 Example of domain randomization using Isaac sim [24]. Shows different lighting, textures, and object positions while the camera position remains the same. 31

List of Tables

2.1 A possible set of DH parameters for LBR [7] 7

4.1 Comparison between popular robotics simulators [3]. (NVIDIA Isaac was not part of the original table from [3]) 26
4.2 Comparison between different robotics simulators used for mobile ground robotics [3]. (NVIDIA Isaac was not part of the original table from [3]) 27
4.3 Comparison between different robotics simulators used for manipulators [3] 27

Chapter 1

Introduction

1.1 Background and motivation

Industry 4.0 is revolutionizing production and manufacturing by utilizing automation, machine learning, and real-time data to monitor production within a factory, as well as across different production plants. Every part of the supply chain is connected through the internet, and this provides immense opportunities for more efficient production and customized products.

A significant part of this transition is the widespread use of autonomous robots. We have already seen adoption where robots have taken over tasks that are labor-intensive and tedious so that human personnel can do other tasks. Still, one of the significant advances in Industry 4.0 is more versatile robots that can adapt to new situations. This will enable robots to take on tasks that only humans could previously accomplish, and free up employees to do other tasks.

One large category of such robots is mobile manipulators, such as KUKA KMR iiwa. These are robots that can move around on a factory floor, or in other areas, in addition to manipulating objects. This enables the robot to do pick and place tasks where the robot is tasked with picking up an object from one location and dropping it off at another. Such robots will become important tools for future factories, but they are not yet at that stage.

One major challenge in developing such robots is the cost of the equipment and development. One of the proposed solutions for lowering development costs is using digital twins in virtual simulation environments. However for a digital twin to be useful it has to resemble the real world to a high degree so that the behavior observed in the simulation represents the behavior in the real world. Recent development in computer hardware have enabled highly realistic simulators both concerning physics and being photo-realistic.

This paper will explore which possibilities a digital twin of the KUKA KMR iiwa provides and how it can affect the development of robotic applications. The advantages of not having to test multiple iterations of the application on a physical robot are immense and it also enables applications including machine learning as the simulation is not limited to running in real-time.

1.2 Problem description

The end goal of the master thesis is to create a digital twin for the KUKA KMR iiwa in a suitable simulation environment. The digital twin should be tested and compared to the real-world robot, and the goal is that the digital twin can be used as a development platform for future robotics applications using the KUKA KMR iiwa. This paper does not focus on the implementation of such a digital twin, but rather it explores the possibilities and discusses how such a solution can be implemented.

1.3 Outline

This paper is a literature study and will discuss topics relevant to a digital twin of a mobile manipulator such as the KUKA KMR iiwa. Chapter 2 presents the fundamentals of robotic kinematics. Section 2.1 explains some terms and basic concepts for understanding the following sections which present manipulator kinematics (Section 2.2) and kinematics for wheeled robots (Section 2.3).

Chapter 3 presents the theory behind SLAM and navigation, which are essential parts of mobile robots. Section 3.2 presents different approaches to SLAM while Section 3.3 presents some commonly used approaches for autonomous navigation.

Chapter 4 presents the term Digital Twin and discusses its use cases. Section 4.1 presents a comparison of different simulators, and the chapter ends with a discussion on the relation between the simulator and the real world. Finally, Chapter 5 concludes the paper and discusses further work.

Chapter 2

Robot kinematics

This chapter presents fundamental theory relevant for understanding robotic control and methods described in later chapters. Usually previously developed libraries and packages will be used for controlling the robot, but some theoretical background is necessary to use them efficiently.

Section 2.1 gives an introduction of poses and frames which are essential for controlling robots. Section 2.2 describes the kinematics of manipulators with an example of a 7-dof manipulator, and lastly, Section 2.3 describes the kinematics of a wheeled robot using mecanum wheels.

2.1 Poses and frames

A pose is used to describe a position and orientation for an object in 2D or 3D space. Relative to a reference frame S, the pose of an object in 2D can be described as

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.1}$$

In the case of a 2D plane, an object has 3 degrees of freedom (dof). It can move in the x and y direction or it can rotate θ degrees around the z-axis, hence the vector $\mathbf{X}$ is sufficient to describe any pose in 2D space. Even though $\mathbf{X}$ is sufficient to describe the pose of an object in 2D it can be somewhat cumbersome to work with, and a more common description of poses is using homogeneous transformation matrices. A homogeneous transformation matrix, T, consists of a rotation matrix, R, and a column-vector, p, describing translation, as shown

in Equation 2.2. The transformation matrix describes rotation and translation relative to a reference frame S.

$$T = \begin{bmatrix} R & p \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.2}$$

In the case of 2D $p = [x, y]^T$ and R as described in Equation 2.3 [18, p. 64]. All rotation matrices $R \in SO(2)$ in 2D or $R \in SO(3)$ in 3D. This means that for all R it holds that $R^T R = I$ and $\det(R) = 1$ [18]. These are properties that imply that there is no scaling involved in the rotation and that the rotation is uniquely defined.

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.3}$$

There are two main benefits of using this notation. The first is that it makes transformations between different frames to a matter of simple matrix multiplication, and the second one is that the procedure for transformations is the same in 3D. The only difference being that $R \in SO(3)$ and not $SO(2)$, and $p \in \mathbb{R}^{3\times 1}$.

The reason for the increased dimensions in R and p in 3D is that a rigid body in 3D has 6 dof, translation in x, y, and z-direction, and rotation in roll, yaw, and pitch direction. To accommodate for this increase in degrees of freedom the homogeneous transformation matrix T has to be in $\mathbb{R}^{4\times 4}$, but it still retains the shape described in Equation 2.2.

There are multiple ways of describing poses and frames such as exponential coordinates, Denavit-Hartenberg parameters (DH-parameters), and quaternions which each have their use cases, but for a basic understanding of robotic control, this is sufficient. Section 2.2 describes the kinematics of a manipulator using DH-parameters.

2.2 Manipulator kinematics

A robotic manipulator consists of joints and links. A link is a rigid body that is part of the robot while a joint connects exactly two links. There are different types of joints, and they provide the robot with varying degrees of freedom. Figure 2.1 shows 6 examples of commonly used joints.

- **Revolute**: One rotational dof. Rotates around a given axis.

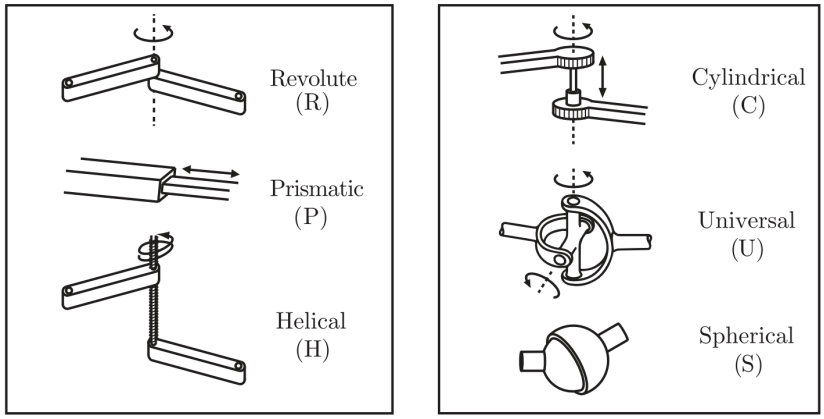


Figure 2.1: 6 examples of common robot joints [18, p. 16]

- **Prismatic**: One translational degree of freedom. Translates along a given axis.
- **Helical**: Also called a screw joint with one dof. It allows for a combined translational and rotational motion along a screw axis.
- **Cylindrical**: Allows independent translation and rotation around one axis, with two dof.
- **Universal**: A combination of two revolute joints arranged orthogonally which provides two dof.
- **Spherical**: Provides 3 dof similar to a shoulder. Works like a universal joint in addition to providing rotation around the axis of the outer link.

Most modern robotic manipulators use only revolute joints and are often referred to as n-dof manipulators, where n is the number of joints. In general, a manipulator operating in 2D needs 3 dof for the end-effector to reach all poses within its workspace, while a manipulator in 3D space needs 6 dof. The workspace of a robot is the space that the end-effector can reach. This is primarily determined by the robot's structure and is independent of the task [18, p. 33]

As a 6-dof robot can reach any position with its end-effector 6 dof should be sufficient. However, only having 6 degrees of freedom may cause issues with singularities and obstacle avoidance. A singularity occurs when the end-effector loses the ability to move instantaneously in one or more directions [18, p.191]. Examples of common singularities in 6-dof robots are two collinear revolute joint axes, three coplanar and parallel revolute joint axis and four revolute joint axes intersecting at a common point [18, p.201]. Having an extra degree of freedom simplifies this problem and allows the manipulator to move more freely which in

many cases avoids such singularities.

More generally a 7-dof manipulator is kinematically redundant. This means that while a 6-dof has a single configuration that takes the end-effector to the desired position, a kinematically redundant manipulator has infinitely many configurations to achieve that. This also means that a 7-dof manipulator has the ability to avoid obstacles and it can create solutions to the inverse kinematic problem which is optimal with respect to some criterion [18, p. 226].

2.2.1 Kinematics of a 7-dof manipulator

There exist multiple kinds of 7-dof manipulators, but this section will focus on the kinematics of KUKA LBR iiwa (hereby referred to as LBR), which is the manipulator on the KUKA KMR iiwa. This section will describe the kinematics of the robot using Denavit-Hartenberg parameters (DH-parameters). The choice of parameters and the calculations are based on [7].

There are multiple ways of describing the kinematics of a manipulator and the reason for choosing DH parameters is that it uses the minimum number of parameters for describing transformations between frames. This saves computational power which improves real-time performance. One drawback of DH-parameters is that they are prone to ill-conditioned parameters. This can happen if two adjacent joints are nearly parallel. There exist other kinematic models such as the product of exponentials (PoE), which has the benefit of treating rotational and prismatic joints equally. In addition, PoE has an explicit definition of frames, whereas DH-parameters have multiple conventions for assigning link frames. The main drawback of PoE is that it requires 6 parameters to define a twist which is more computationally expensive.

The LBR is a 7-dof manipulator with zero link offset with an anthropomorphic arm structure. This means that the manipulator bears similarities to a human arm and the joints are often described as *shoulder*, *elbow*, and *wrist* as shown in Figure 2.2

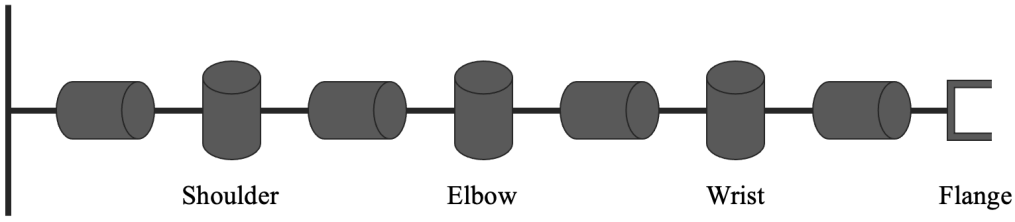


Figure 2.2: Example of an anthropomorphic manipulator

Before computing the forward kinematics of the LBR the DH parameters has to be defined. When choosing these parameters there is some freedom of choice but for simplicity, the same parameters as in [7] are used. This is shown in Figure 2.3.

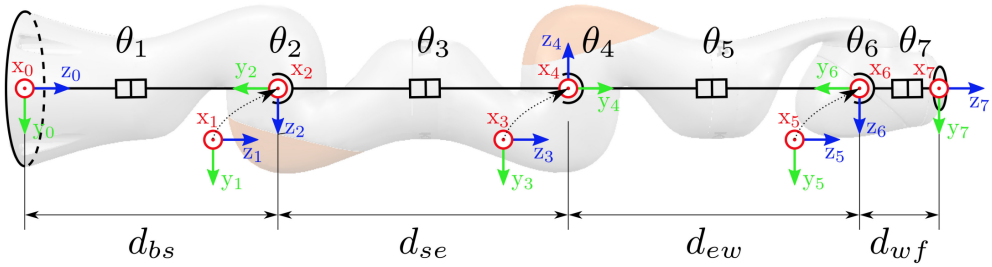


Figure 2.3: Horizontally mounted LBR iiwa with corresponding DH frames illustrated [7]

In Figure 2.3 θ_i describes the angle of joint i and d_{ij} is the distance between the base, wrist, elbow, and flange. To calculate the forward kinematics a set of DH-parameters has to be chosen. Based on Figure 2.3 one possible set of such parameters is shown in Table 2.1.

Table 2.1: A possible set of DH parameters for LBR [7]

i	a_i [mm]	α_i [rad]	d_i [mm]	θ_i [rad]
1	0	$-\frac{\pi}{2}$	d_{bs}	θ_1
2	0	$\frac{\pi}{2}$	0	θ_2
3	0	$\frac{\pi}{2}$	d_{se}	θ_3
4	0	$-\frac{\pi}{2}$	0	θ_4
5	0	$-\frac{\pi}{2}$	d_{ew}	θ_5
6	0	$\frac{\pi}{2}$	0	θ_6
7	0	0	d_{wf}	θ_7

Using DH parameters, the parameters for joint i are always relative to joint $i-1$. The list below explains each parameter in Table 2.1

- a_i – The distance between z_{i-1} and z_i along x_i.
- α_i – The angle between z_{i-1} and z_i with the rotational axis along x_i.
- d_i – The distance between x_{i-1} and x_i along z_{i-1}.
- θ_i – The angle between x_{i-1} and x_i with the rotational axis along z_{i-1}.

Table 2.1 defines all the necessary parameters for computing the transformation between each adjacent link. Using all the transformations, ${}^{i-1}T_i$, the forward

kinematics of the manipulator can then be calculated by multiplying all transformations sequentially. Each transformation can be described as Equation 2.4 [18, p. 590]

$$^{i-1}T_i = \mathrm{Rot}_{\hat{x}}(\alpha_{i-1})\mathrm{Trans}_{\hat{x}}(a_{i-1})\mathrm{Trans}_{\hat{z}}(d_i)\mathrm{Rot}_{\hat{z}}(\theta_i) \tag{2.4}$$

where

$$\begin{aligned}
\mathrm{Trans}_{\hat{z}}(d_i) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathrm{Rot}_{\hat{x}}(\alpha_{i-1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
\mathrm{Trans}_{\hat{x}}(a_{i-1}) &= \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathrm{Rot}_{\hat{z}}(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{2.5}$$

Combining all matrices from Equation 2.5 the resulting transformation is shown in Equation 2.6

$$^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i\sin\alpha_{i-1} \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i\cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Now that each transformation is known by Equation 2.6 the final pose of the end-effector in the base frame can be calculated as the product of all frames

$$^0\mathbf{T}_n = ^0\mathbf{T}_1 \ldots ^{n-1}\mathbf{T}_n \tag{2.7}$$

where n=7 for the LBR.

Inverse kinematics is the other important problem to solve for controlling the robot. This is a much more complicated problem and will not be solved here. There exist other papers that have comprehensive explanations of this such as [7, 32, 13]

2.3 Wheeled robots

In many modern robotics applications, manipulators are placed atop mobile bases. This makes the robot more versatile, but it also comes with some challenges when modeling its behavior. This section will discuss the kinematics of wheeled robots.

In this section, we assume that the robot moves on a plane surface with no skidding. This is to create a consistent model of how the wheels' rotation affects the robot's pose.

There are two main categories of wheeled robots; holonomic drive and non-holonomic drive. A formal definition is that a robot with holonomic drive has no constraints on its velocity $\dot{q} = (\dot{\phi}, \dot{x}, \dot{y})$, while a robot with non-holonomic drive has a single Pfaffian velocity constraint. An example of a non-holonomic robot can be a car where the constraint prevents the car from moving directly sideways, despite that it can reach any configuration (ϕ, x, y) in an obstacle-free environment [18, p. 514].

Even though a non-holonomic robot can reach any configuration in an obstacle-free environment this is not always true in real life where obstacles exist. The main advantage of holonomic robots is that they are more maneuverable. The ability to move directly sideways allows them to access spaces that non-holonomic robots can't access, and in many cases maneuvering a holonomic robot is simpler than maneuvering a non-holonomic robot. There are, however, advantages to using non-holonomic robots as well. To achieve non-holonomic drive the robot needs special wheels that can move in multiple directions which makes them more fragile and not suited for higher speeds. Usually, holonomic robots use *omni wheels* or *mecanum wheels* shown in Figure 2.4. Both types have rollers with a small diameter which works best on flat, hard surfaces [18, p. 515].

Figure 2.4: Example of omni wheel (left) and mecanum wheel (right) [18, p. 514]

The rest of this section will focus on kinematics for holonomic robots as the KMR

has holonomic drive. There are two main types of wheels used in holonomic robots, namely omni wheels and mecanum wheels, shown in Figure 2.4. Both types allow the wheels to move sideways and in line with the wheel's direction. The omni wheels consist of multiple rollers around the circumference of the wheel which are oriented in line with the plane of the wheel, whereas the rollers on the mecanum wheels usually are mounted at about 45° to the plane of the wheel [18]. There are multiple ways of configuring a robot with omni wheels or mecanum wheels, but two common ones are shown in Figure 2.5

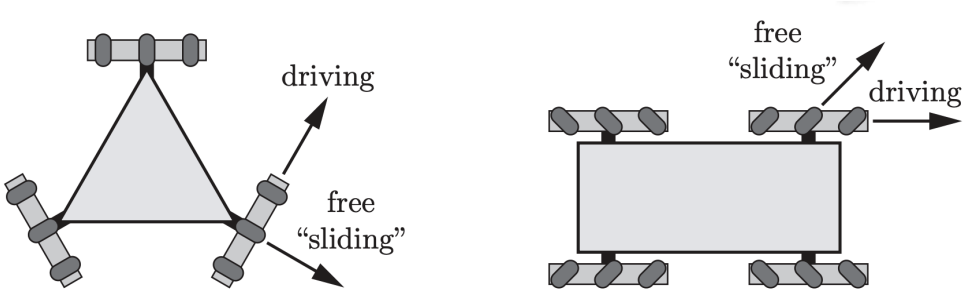


Figure 2.5: Common configurations of omni wheels (left) and mecanum wheels (right) to achieve holonomic drive [18, p. 516]

In Figure 2.5 the *driving* direction of the wheel is the direction in which the motor drives the wheel and the *free sliding* direction is the direction in which the rollers can roll freely without any power from the motor. In this illustration, the rollers on the mecanum wheels are mounted at $\gamma = 45°$.

2.3.1 Kinematic model for a four mecanum wheeled mobile robot

Kinematics is used to describe the motion of a robot based on some input. The forward kinematics describes the robot's motion based on input from each joint's position, and the inverse kinematics describes what the input to each joint or motor has to be to achieve a certain movement. This chapter will discuss the kinematics for the mobile base in the 2D plane.

The KMR iiwa is a robot with holonomic drive using mecanum wheels. That means that the forward kinematics describes where the robot moves given some input to the four wheels, while the inverse kinematics outputs the way each wheel has to move in order to reach some target velocity $[v_x, v_y, \omega]^T$. The derivation of the kinematic model is based on the paper of Taheri, Qiao, and Ghaeminezhad [31].

In the plane, the robot's motion can be described by a vector $[v_x, v_y, \omega]^T$ which corresponds to the forward and backward velocity in the x-direction, the left and

right velocity in the y-direction, and the rotational velocity around the z-axis. This is relative to the robot's X-Y frame. In addition, each wheel of the robot has two velocities, the angular velocity of the wheel, ω_i, and the velocity of the rollers, v_{ir}. Equation 2.8 shows how these velocities relate.

$$\begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = \mathbf{T} \begin{bmatrix} v_{xR} \\ v_{yR} \\ \omega_R \end{bmatrix} \tag{2.8}$$

The task at hand is to calculate $\mathbf{T}$ to find an inverse kinematic. Figure 2.6 shows a schematic illustration of one mecanum wheel and the relevant constants needed to derive the inverse kinematics.

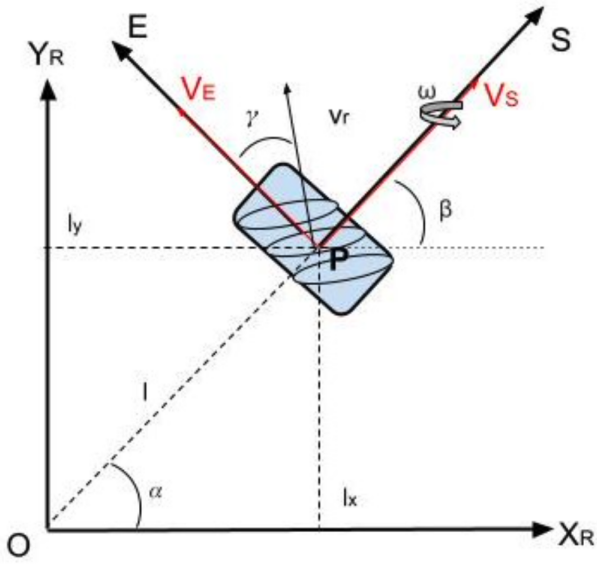


Figure 2.6: Here the variables used to derive the inverse kinematics for the robot are shown [10]

The constants in Figure 2.6 are described in the list below [10]

- X_R-Y_R, hereby references as X-Y, is the robot frame
- E-S is the wheel frame
- v_{ir} and ω_i is the angular velocity of the rollers and the wheel in the wheel frame (E-S).
- β is the angle between the S-axis and the X-axis.
- γ is the angle between the v_r and the E-axis.

- α is the angle between the X-axis and the line from O to the center of the wheel P.
- l_y and l_x are the horizontal and vertical distances from the center of the robot.
- r is the radius of the wheel.

In Equation 2.8 $\mathbf{T}$ is the transformation matrix between the robot frame and the wheel frame. This transformation matrix consists of three main transformations

1. Transform the wheel's velocity ω_i and v_{ir} to the center of the wheel, ${}^{w_i}\mathbf{T}_P$.
2. Transform the velocities from the wheel's frame to the frame of the robot, ${}^{P}\mathbf{T}_R$
3. And lastly transforming the velocity of the robot to the wheel's, T'

The first point is to transform the velocities ω_i and v_{ir} into the E-S frame. v_E is directly proportional to the roration ω_i and the radius of the wheel r, and is described in Equation 2.9

$$v_E = \omega_i r \tag{2.9}$$

Including the velocity of the rollers the velocity of the wheel in the E-S frame can be described as

$$\begin{bmatrix} v_S \\ v_E \end{bmatrix} = \begin{bmatrix} 0 & \sin\gamma \\ r & \cos\gamma \end{bmatrix} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = {}^{w_i}\mathbf{T}_P \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} \tag{2.10}$$

Further the transformation from the E-S frame to the robot's frame X-Y. The only transformation between the two frames is a pure rotation of β around the z-axis. To calculate this Equation 2.3 can be used, which results in Equation 2.11

$$\begin{bmatrix} v_{xR} \\ v_{yR} \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} v_S \\ v_E \end{bmatrix} = {}^{P}\mathbf{T}_R \begin{bmatrix} v_S \\ v_E \end{bmatrix} \tag{2.11}$$

The last transformation describes the transformation between the velocity of the wheels in the robot frame $[v_{xR}, v_{yR}]^T$ and the robot's velocity $[v_{xR}, v_{yR}, \omega_R]^T$. As the mecanum wheels have planar movement, the angular velocity of the robot affects both the velocity of the wheels and the rollers. This transformation is shown in Equation 2.12

$$\begin{bmatrix} v_{xR} \\ v_{yR} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -l_y \\ 0 & 1 & l_x \end{bmatrix} \begin{bmatrix} v_{xR} \\ v_{yR} \\ \omega_R \end{bmatrix} = \mathbf{T'} \begin{bmatrix} v_{xR} \\ v_{yR} \\ \omega_R \end{bmatrix} \tag{2.12}$$

Combining all the transformations from Equation 2.10, 2.11, and 2.12, the resulting transformation matrix $\mathbf{T}$ can be expressed as shown in Equation 2.13

$$\mathbf{T} = {}^{w_i}\mathbf{T}_P^{-1}\,{}^{P}\mathbf{T}_R^{-1}\,\mathbf{T'} = \begin{bmatrix} 0 & \sin\gamma \\ r & \cos\gamma \end{bmatrix}^{-1} \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & -l_y \\ 0 & 1 & l_x \end{bmatrix} \tag{2.13}$$

Using the fact that $l_x = l\cos\alpha$ and $l_y = l\sin\alpha$ with trigonometric functions the expression can be simplified to Equation 2.14. The detailed calculations can be found in [31].

$$\mathbf{T} = \frac{1}{-r\sin\gamma} \begin{bmatrix} \cos(\beta-\gamma) & \sin(\beta-\gamma) & l\sin(-\alpha+\beta-\gamma) \\ -r\cos\beta & -r\sin\beta & lr\sin(-\alpha+\beta) \end{bmatrix} \tag{2.14}$$

As the velocity of the rollers, v_{ir} cannot be controlled by the motor, it is desired to express the inverse kinematics as a function of only the angular velocity of the wheels. The correlation between the velocity of the roller v_{ir} and the angular velocity ω_i is given by Equation 2.15

$$v_{ir} = \frac{1}{\cos\gamma} r_r \omega_i \tag{2.15}$$

Using Equation 2.15 the inverse kinematics can be expressed as Equation 2.16

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = -\frac{1}{r} \begin{bmatrix} \frac{\cos(\beta_1-\gamma_1)}{\sin\gamma_1} & \frac{\sin(\beta_1-\gamma_1)}{\sin\gamma_1} & \frac{l_1\sin(\beta_1-\gamma_1-\alpha_1)}{\sin\gamma_1} \\ \frac{\cos(\beta_2-\gamma_2)}{\sin\gamma_2} & \frac{\sin(\beta_2-\gamma_2)}{\sin\gamma_2} & \frac{l_2\sin(\beta_2-\gamma_2-\alpha_2)}{\sin\gamma_2} \\ \frac{\cos(\beta_3-\gamma_3)}{\sin\gamma_3} & \frac{\sin(\beta_3-\gamma_3)}{\sin\gamma_3} & \frac{l_3\sin(\beta_3-\gamma_3-\alpha_3)}{\sin\gamma_3} \\ \frac{\cos(\beta_4-\gamma_4)}{\sin\gamma_4} & \frac{\sin(\beta_4-\gamma_4)}{\sin\gamma_4} & \frac{l_4\sin(\beta_4-\gamma_4-\alpha_4)}{\sin\gamma_4} \end{bmatrix} \tag{2.16}$$

As mentioned at the beginning of this section mecanum wheels are usually mounted parallel to the forward direction of the robot, as shown in Figure 2.5. In other words this means that $\beta = \frac{\pi}{2} \vee -\frac{\pi}{2}$. In addition, the rollers of mecanum wheels are mounted at a 45° angle. This leaves γ with only four possible values, $\gamma = \frac{\pi}{4} \vee -\frac{\pi}{4} \vee \frac{3\pi}{4} \vee -\frac{3\pi}{4}$. Using this simplification Equation 2.16 can be rewritten as Equation 2.17

$$\begin{aligned}
\omega_1 &= \frac{1}{r}(v_x - v_y - (l_x + l_y)\omega) \\
\omega_2 &= \frac{1}{r}(v_x + v_y + (l_x + l_y)\omega) \\
\omega_3 &= \frac{1}{r}(v_x + v_y - (l_x + l_y)\omega) \\
\omega_4 &= \frac{1}{r}(v_x + v_y + (l_x + l_y)\omega)
\end{aligned} \tag{2.17}$$

Equation 2.17 concludes the inverse kinematics problem for a robot with mecanum wheels mounted parallel to the direction of travel which is the case with KMR.

Chapter 3

SLAM and Navigation

There are two major challenges using mobile manipulators, or more specifically autonomous mobile robots (AMRs), which are the problems of navigation and mapping. For a robot to move efficiently in an area it needs a map, however, the creation of such a map can be challenging. Navigation of AMRs involves many problems such as localization, path planning, and object avoidance. [27] mentions four core requirements to achieve such a task.

1. Map
2. Pose of the robot
3. Sensing
4. Path calculation and driving

If the robot knows its pose and can sense its surroundings it is possible to create a map using SLAM. If the robot also has a map available it enables it to accomplish navigation tasks which takes the robot from point A to point B. This chapter will consider the mobile manipulator as an AMR where navigation of the mobile platform is in focus.

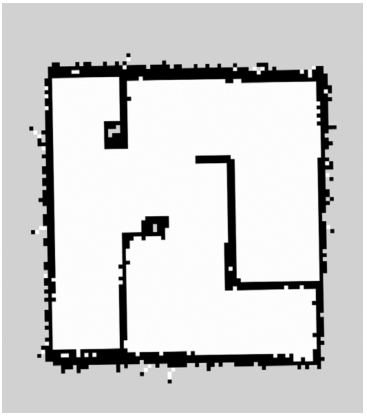
The following sections will discuss these topics more in detail. Section 3.1 describes maps and how to represent them, then Section 3.2 discusses how maps are created using SLAM, and lastly, Section 3.3 describes the process of navigation and path planning. The map and navigation section is mainly based on [27] whereas the SLAM section is based on [30].

3.1 Maps

Maps are fundamental to navigation for AMRs. They can contain different kinds of information and come in different formats. Traditionally a 2D occupancy grid map (OGM) has been used, but in recent years 3D maps have been used and

sometimes in combination with object segmentation [27, p. 324]. As the KMR is a mobile manipulator moving on a plane factory floor, a 2D map is sufficient for navigation purposes and will therefore be the focus of this section.

2D occupancy grid maps are commonly used within the ROS community and will also be sufficient for this application. An OGM is represented as a greyscale image as shown in Figure 3.1a. Here the white area is where the robot is free to move, the black area is occupied and the gray area is unknown.



(a) Example of an occupancy grid map

```
image: map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

(b) Example of `.yaml` file associated with the map

Figure 3.1: Example of an OGM and associated `.yaml` file created by Turtlebot 3 [27, p. 325]

When computing the map each pixel is assigned a value based on the probability that it is occupied, denoted as *occ*. This probability is calculated using the posterior probability of Bayes' theorem. If the value of *occ* is close to 1 indicates a higher probability that the area is occupied and if *occ* is closer to 0 indicates a free area. These values are published as ROS messages and converted to integers $[0, 100]$ including -1 which indicates that the area is unknown [27].

Accompanying the map is a configuration `.yaml` file which specifies properties of the map. Resolution specifies the area of a pixel in meters and the origin is the 2D pose of the lower-left pixel on the form $[x, y, \text{yaw}]$ (yaw is often ignored) [9, 34].

An important addition to the OGM is the costmap which is used for navigation. This will be further discussed in Subsection 3.3.1

3.2 SLAM – Simultaneous Localization and Mapping

As discussed in the previous section maps are essential to AMRs, but they are not always available. Simultaneous localization and mapping, or SLAM, is the problem of creating a map using a mobile robot with different sensors.

A formal definition of the SLAM problem is presented in [30].

> A mobile robot roams an unknown environment, starting at an initial location x_0. Its motion is uncertain, making it gradually more difficult to determine its current pose in global coordinates. As it roams, the robot can sense its environment with a noisy sensor. The SLAM problem is the problem of building a map of the environment while simultaneously determining the robot's position relative to this map given noisy data.

The challenge of SLAM is to do both localization and mapping at the same time, whereas each task separately can be done easily.

A graphical representation of the SLAM problem is shown in Figure 3.2.

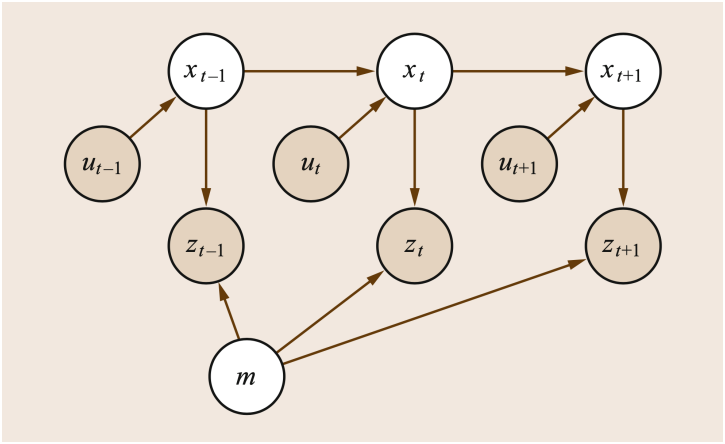


Figure 3.2: Graphical representation of the SLAM problem [30, p. 1154]

In Figure 3.2 $t \in [0, T]$ is the time interval and T is the terminal time. x_t is the robot's pose at time t and m is the map which is static and therefore time irrelevant. u_t is the relative motion of the robot, often derived from odometry, and z_t is the observations of the map gained from sensors. Together the SLAM problem can be defined probabilistically as

$$p(x_t, m | Z_t, U_t) \tag{3.1}$$

Equation 3.1 describes the *online* SLAM problem which is when the robot calculates its pose and the map in real-time, as it discovers the world. The counterpart of online SLAM is known as *full* SLAM which can be described as $p(X_T, m|Z_T, U_T)$. This is usually done in batches such that the robot explores the world for a while before it terminates and then generates the estimated map and path. The rest of this section will mainly focus on online SLAM described in Equation 3.1.

To calculate this probability the robot needs a model that relates the odometry, u_t, to the location, x_t, and the observations, z_t, to the map, m. The two main models that are used are shown in Equation 3.2.

$$\begin{aligned} \text{Motion model:} \quad & p(x_t|x_{t-1}, u_t) \\ \text{Measurement model:} \quad & p(m|x_t, z_t) \end{aligned} \tag{3.2}$$

The motion model describes the probability that the robot is located at x_t given the last position, x_{t-1}, and the odometry in the last interval u_t, whereas the measurement model describes the probability of doing the observation z_t given it pose x_t and the current map [30].

There are three main paradigms in SLAM used to calculate these probabilities and finally to create a map. The first SLAM applications used extended Kalman filters (EKF SLAM), but in most cases, EKF has been replaced by particle filters for online SLAM. The last paradigm is graph-optimized approaches which were primarily used for full SLAM but have recently seen adoption in online SLAM applications as well [30, 15]. The following sections will present a brief overview of the different paradigms.

3.2.1 Kalman Filter

Kalman filter, or extended Kalman filter (EKF), has historically been the most influential method used for online SLAM [30]. The filter uses a model to estimate the current state from the previous one. Then the error between the prediction of the previous state and the measured values of the current state is used to update the estimation to be more accurate [27]. A schematic illustration of this is shown in Figure 3.3.

A drawback of the regular Kalman filter is that it is limited to linear models. Extended Kalman filters use Taylor series expansion to linearize the models so that they can work with non-linear systems as well.

The EKF algorithm uses a single state vector, $\boldsymbol{\mu_t}$, where the estimated pose of the robot is stored in addition to the estimated locations of a set of features detected

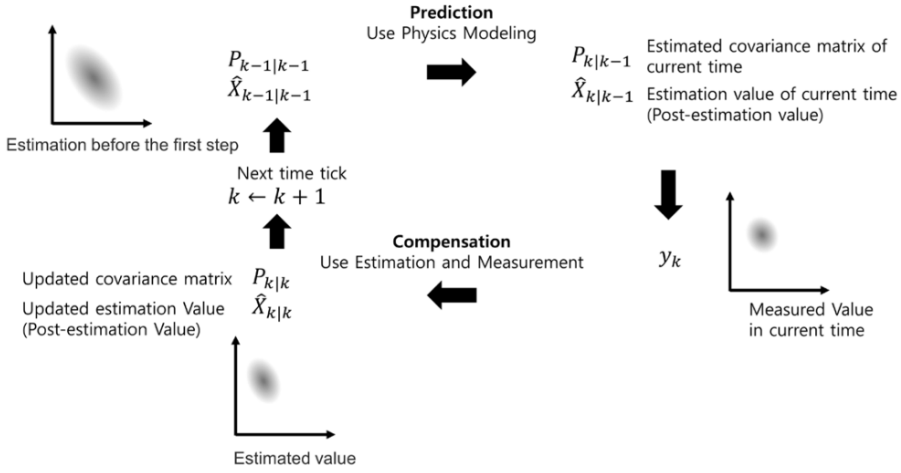


Figure 3.3: Illustration of the concept of the Kalman filter [27, p. 334]

by the sensors. The algorithm presents the estimate by a multivariate Gaussian as shown in Equation 3.3 [30]

$$p(x_t, m|Z_t, U_t) = \mathcal{N}(\boldsymbol{\mu_t}, \boldsymbol{\Sigma_t}) \tag{3.3}$$

$\boldsymbol{\mu_t}$ describes the pose of the robot and location of N features and its dimension is $3 + 2N$ as three variables are needed to describe the robot's pose and 2 coordinates are needed for each feature. The matrix $\boldsymbol{\Sigma_t}$ is the covariance of the robot's assessment of its expected error in the guess $\boldsymbol{\mu_t}$, which is of dimension $(3 + 2N) \times (3 + 2N)$ [30].

The dimension of $\boldsymbol{\Sigma_t}$ highlights one of the main drawbacks of this approach as its size increases quadratically. After some time this will cause an issue with memory and performance will significantly decrease. Different alternative approaches have been suggested focusing on reducing the problem of the dimension of the covariance matrix [30].

3.2.2 Particle Filter

In recent years particle filters have become the most popular algorithm in object tracking [27]. Contrary to EKF which uses an approximated linear model to search for parameters to describe the state, particle filters predict these parameters based on a probabilistic trial-and-error method [27]. A particle can be thought of as a guess of the true current state of the system. In addition to the state, each particle

also contains a weight that indicates the importance of the particle, based on the new sensor measurement.

When initializing the particle filter, N particles are distributed near the robot's initial pose. Each particle is assigned a weight of $1/N$ which indicates that they are all equally important. N is a parameter that is empirically determined, usually in the hundreds [27]. After initialization 4 main steps are repeated [27].

1. **Prediction:** The particles are moved based on the motion model (Equation 3.2)
2. **Update:** The probability and weight of each particle are calculated based on the new sensor information
3. **Pose estimation:** The pose of the robot is estimated using the position, orientation, and weight of all the particles
4. **Resampling:** New particles replace less weighted particles and inherit information from the highly weighted particles. There still have to exist N particles after this step.

There exist different variants of particle filters, but the main idea behind them is roughly as described above. Examples of implementations are Monte Carlo localization (MCL) which is commonly used in pose estimation algorithms [27], and Rao-Blackwelled particle filters which utilize a combination of Kalman filter and particle filter [27].

Particle filters are commonly used in SLAM systems, but they suffer from two main problems. The first is that the number of particles, N, is set empirically. This greatly affects the performance of the system and is tightly coupled with the amount of uncertainty the filter needs to represent. The other problem is that the system is prone to particle depletion in the case of nested loops and re-visits. Particle depletion means that there is a lack of particles representing the true state [30]. There exist approaches attempting to improve the situation, but they do not eliminate the problem in general.

3.2.3 Graph-based Optimization

Graph-based optimization techniques for SLAM were mainly used for full SLAM problems, but in recent years the technique has also been adopted in online SLAM applications. The idea is that the pose of the robot and the location of landmarks are represented as nodes in a graph. Each pair of points, x_{t-1}, x_t, along the robots path is connected with an edge containing information from the odometry u_t. The edges between the robot's location x_t and a landmark m_i describe the

soft constraint between them [30]. Finally, this results in a sparse graph used to generate the map.

When the graph is created the optimization problem is to minimize the error caused by the soft constraints. There exist different approaches for this, such as [14, 2].

One major advantage of this approach is that the sparse graph allows much higher dimensional maps compared to EKF SLAM. Due to the limited space used to store, and update the graph, graphical SLAM can create significantly larger maps compared to other methods.

One example of a framework implementing graph-based optimization for SLAM is Google's Cartographer [11], however, their system works for online SLAM. The system combines both local and global approaches to 2D SLAM where each scan of the environment is matched against a submap M. Matching each scan to a submap will, over time, accumulate errors. When mapping larger areas multiple submaps are created and their poses are stored in memory for the loop closing optimization. Once a submap no longer changes each pair of submaps are considered for loop closing, and if a good match is found the relative poses of the submaps are added to the optimization problem [11].

3.3 Navigation

Navigation is the task of moving the robot from an initial pose to a goal pose using a map created by for example SLAM as discussed in Section 3.2. The robot should be able to plan a path and avoid obstacles and possible moving objects autonomously. The task consists of the following four subtasks [27]

1. **Sensing:** The robot gathers information from the wheel encodes and other sensors that describe the pose of obstacles such as walls, furniture, and other objects relative to the robot.

2. **Localization / Pose estimation:** Based on the sensory input, the pose of the robot is estimated on the map. Multiple such methods exist, but one commonly used is AMCL which will be discussed in Subsection 3.3.2.

3. **Motion planning / Path planning:** Creates a path from the current pose to the goal pose on the map. Usually, this task is separated into a global path which creates a path throughout the whole map, and a local path which handles path planning for a small area around the robot.

4. **Move / Obstacle avoidance:** When the robot follows the planned path obstacles and moving objects can occur. Obstacle avoidance is the task

of avoiding collision with these objects and planning an alternative path. DWA is one algorithm used for this task, and it will be explained in Subsection 3.3.3.

The following subsections will present three main components of navigation; costmaps, Adaptive Monte Carlo Localization (AMCL), and Dynamic Window Approach (DWA)

3.3.1 Costmap

As a result of SLAM, a static map for navigation is generated. It defines occupied areas, free areas, and unknown areas. A costmaps is an extension of this map describing the cost for the robot to move in different parts of the map. A higher cost indicates a higher probability of colliding. An example of a costmap is shown in Figure 3.4

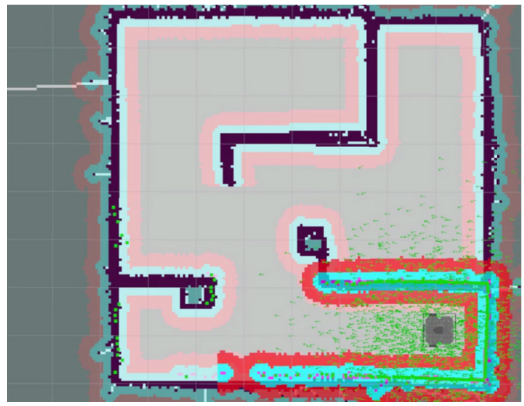


Figure 3.4: Example of a costmap corresponding to the map in Figure 3.1a [27, p. 338]

The costmap is generated based on the OGM in Figure 3.1a and the brighter colors are the parts of the map that is recognized by the robot from its current pose.

Costmaps can be separated into two types with the first being a global costmap which is used for navigation over the entire fixed map, and the second is a local costmap used for obstacle avoidance and path planning in the area close to the robots. Both maps are represented the same way with a grid containing values between 0 and 255 which indicates the cost. Figure 3.5 shows the relationship between the distance to the object and the costmap value.

Using this map it is possible to use different pathfinding algorithms such as A* and others. There exist multiple ROS packages and other implementations which

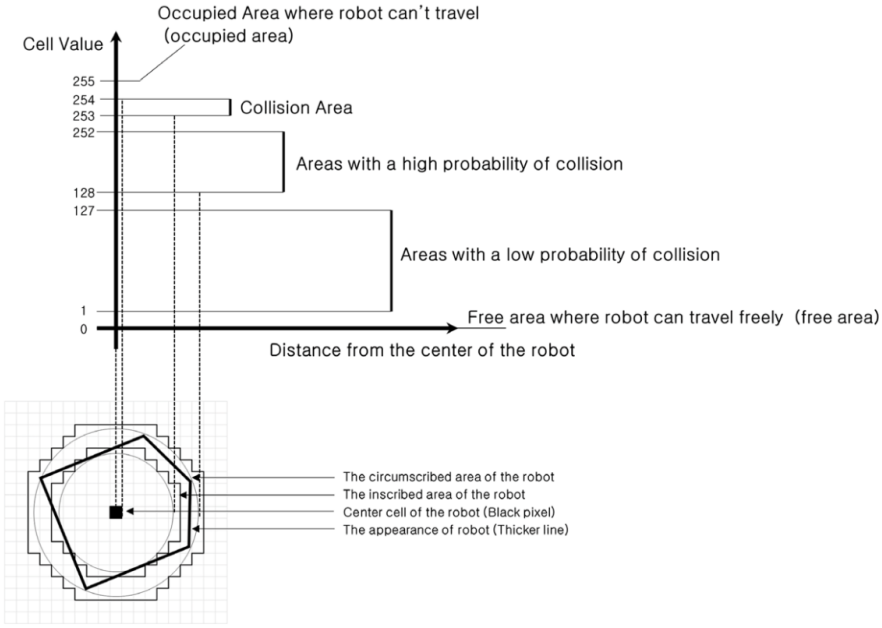


Figure 3.5: Relationship between the distance to the object and the costmap value [27, p. 356]

automate the process of path planning. One such example is the ROS package Navigation 2 [19].

3.3.2 Adaptive Monte Carlo localization

Adaptive Monte Carlo localization (AMCL) is based on the Monte Carlo localization (MCL) pose estimation algorithm using particle filters discussed in Subsection 3.2.2. The main benefit of AMCL is that it has better real-time performance which it achieves by using fewer samples (equivalent to particles) than MCL.

The procedure of localization is similar to the general algorithm for particle filters which consists of predicting possible samples of the pose, updating the samples with sensor information, estimating the pose, and resampling which creates new likely samples. The details of particle filters are discussed in Subsection 3.2.2.

This algorithm is commonly used in packages for navigation using ROS. One example of this is the Navigation 2 [19] package which provides functionality for autonomous navigation in ROS applications.

3.3.3 Dynamic Window Approach

Dynamic Window Approach (DWA) is a commonly used collision avoidance strategy in ROS. The robot initially follows a global path, but when obstacles occur that are not present on the map, the robot needs to plan a local path and DWA is one strategy for planning this path.

The robot has a limited number of available movements based on the dynamics of the robot. A holonomic robot can move in x and y direction in addition to rotation θ around the z-axis. The task of DWA is to continuously choose the optimal collision-free velocity. DWA does not primarily consider the robot in the 2D xy-plane, but rather in the velocity space consisting of (v_x, v_y, ω), where ω is the rotational velocity around the z-axis. The algorithm searches the velocity space for possible velocities and chooses the best one.

To choose the best, an objective function $G(v_x, v_y, \omega)$ considers different metrics and decides what combination of (v_x, v_y, ω) yields the best outcome. To choose what is the best outcome the objective function may consider cost on the costmap, distance to global path, distance to goal, orientation to goal, and orientation to path [17]. When the best velocities are determined so that the objective function is maximized the robot executes this.

Chapter 4

Digital twins

Autonomous mobile robots (AMRs) and mobile manipulators will be an essential part of manufacturing and industrial work, and it is one of the primary building blocks of Smart Factories and self-organization *Industry 4.0* [16]. One of the main challenges of developing such robots is that testing is time-consuming and that it requires expensive resources. An essential criterion for the widespread adaptation of such robots is that the development cost is lowered and that testing becomes less resource-demanding.

In general, a *Digital Twin* (DT) is a virtual model of a physical entity with similar properties. This allows us to expose the DT to a variety of situations and observe how the physical entity is likely to be affected or behave. The characteristics of a digital twin are as follows [1]

1. To integrate various types of data from the physical entity and to map the physical entity faithfully.
2. To exist along the lifecycle of the physical entity and evolve with the physical one, and to accumulate knowledge from the physical one
3. Not only to describe the physical entity but also to optimize the physical one based on the cyber model.

In the field of robotics, the DT allows developers to efficiently test new algorithms and check that the robot behaves as expected. This significantly reduces the cost of development and also provides a safe environment where humans nor robots are exposed to hazards.

Section 4.1 discusses common features of robotics simulators and compares some of the most used simulators. Section 4.2 discusses the main problem of simulators which is the gap between the real world and the simulation.

4.1 Robotic simulators

In order to develop a DT and be able to test it, a simulator is needed to simulate the real world. It needs a physics engine to replicate collisions between objects and to provide the robot with the possibility to affect the environment.

[3] defines a robotics simulator as an end-user software application that includes at least the following functionality:

1. Physics engine for realistic modeling of physical phenomenon
2. Collision detection and friction models
3. Graphical User Interface (GUI)
4. Import capability for scenes and meshes
5. API especially for programming language used by the robotics community (C++/Python)
6. Models for an array of joints, actuators, and sensors readily available

Table 4.1 shows an overview of some common robotics simulators with their features.

Table 4.1: Comparison between popular robotics simulators [3]. (NVIDIA Isaac was not part of the original table from [3])

Simulator	RGBD + LiDAR	Force sensor	Linear + Cable actuator	Multi-Body Import	Soft-Body Contacts	DEM Simulation	Fluid Mechanics	Headless Mode	ROS Support	HITL	Teleoperation	Realistic Rendering	Inverse Kinematics
Airsim	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓, Unreal	✗
CARLA	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓, Unreal	✗
CoppeliaSim	✓	✓	Linear only	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓
Gazebo	✓	✓	Linear only	✓	✗	Through Fluidix	Through Fluidix	✓	✓	✓	✓	✗	✓
MuJoCo	✓	✓	✓	✓	✓	✓	Limited	✓	✗	HAPTIX only	HAPTIX only	✗	✗
PyBullet	✓	✓	Linear only	✓	✓	✓	✗	✓	✗	✗	✓	✗	✓
SOFA	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓, Unity	✗
UWSIM	RGBD only	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓	✓, custom	✗
Chrono	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓, offline	✓
Webots	✓	✓	Linear	✓	✗	✗	Limited	✓	✓	✗	✓	✗	✗
NVIDIA Isaac	✓[24]	✓[3]	✓[24]	✓[24]	✓[3]	Through Flex [25]	Through Flex [20]	✓[24]	✓[24]	✓[22]	✓[21]	✓, Omniverse	✓[3]

There exist multiple simulators that are suitable for different robotics applications. [3] divides robotics applications into 7 main categories; (i) Mobile Ground Robotics, (ii) Manipulation, (iii) Medical Robotics, (iv) Marine Robotics, (v) Aerial Robotics, (vi) Soft Robotics, and (vii) Learning for Robotics. As the main focus of this paper is the mobile manipulator KMR the rest of this chapter will focus on simulators developed for (i) mobile ground robotics and (ii) manipulation

Mobile Ground Robotics

Mobile ground robotics is a wide sub-domain in the field of robotics. It consists of many other domains such as locomotion, perception, cognition, and navigation [29]. A simulator for mobile ground robotics, therefore, needs to provide an interface that makes simulations in all these domains possible. One of the main enablers for this is that the simulator provides a comprehensive sensor suite. In Table 4.2 a list of simulators is shown with some of the essential features.

Table 4.2: Comparison between different robotics simulators used for mobile ground robotics [3]. (NVIDIA Isaac was not part of the original table from [3])

Simulator	GPS	Tracks	Wheels	Legs	Mecanum/Omni Wheels	Heightmap Import	OpenDrive/ OpenStreetMap	Pathplanning	ROS Support	RGBD	LiDAR	Realistic Rendering
Gazebo	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
CoppeliaSim	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Raisim	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓, Unity
Webots	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
PyBullet	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗
CARLA	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓, Unreal
Chrono	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓, POV-Ray
NVIDIA Isaac	✗[21]	?	✓[24]	✓[24]	✓[24]	✗[26]	✗	✓[21]	✓[24]	✓[24]	✓[24]	✓[24]

Table 4.2 is a general comparison with a large scope. The features are selected to cover the entire space of mobile ground robots, however, the use case for the KMR is narrower. The KMR is designed for indoor use, on a flat surface where it moves over relatively small distances compared to e.g. an autonomous car. This has to be considered when choosing the appropriate simulator.

Manipulators

Table 4.3 shows a list of popular simulators mainly used for manipulators.

Table 4.3: Comparison between different robotics simulators used for manipulators [3]

Simulator	Pathplanning	Inverse Dynamics	Inverse Kinematics	Suction	Deformable Objects	Force/Torque Sensor	Realistic Rendering
SimGrasp	✓	✓	✓	✗	✗	✓	✗
Gazebo	✓	✓	✓	✗	✗	✓	✗
CoppeliaSim	✓	✗	✓	✓	✗	✓	✗
PyBullet	✓	✓	✓	✗	✓	✓	✗
MuJoCo	✗	✓	✗	✗	✓	✓	✗
NVIDIA Isaac	✓	✗	✓	✓	✓	✓	✓

For manipulators, a different set of features are important. One major difference between the mobile ground robots is that manipulators interact with smaller objects in pick-and-place tasks. This requires a finer control and stable physics that robustly handles contacts. The simulator also needs position, velocity, and torque

control for joints to control the arm, in addition to visual sensors such as RGB and RGB-D. Specific to manipulators are features such as inverse and forward kinematic solvers as well [3].

4.1.1 Simulator comparison

The section above gives an overview of a range of different simulators and their pros and cons. This section will discuss three promising options for creating a DT of a mobile manipulator.

Gazebo

Gazebo was the most widely used robotics simulator counted by citations in the period 2016-2020 [3]. It is an open-source, general-purpose simulator and it supports the most common features for robotics simulators as shown in Table 4.1, 4.2, and 4.3. The simulator supports multiple physics engines such as Bullet, Dynamic Animation, Robotics Toolkit, Open Dynamics Engine, and Simbody [8]. It also offers many common sensors such as cameras, RGB-D and GPS, however Gazebo does not support realistic rendering which means that what the cameras in the simulator sees does not transfer very well into real world application.

Gazebo has two main benefits over other simulators, namely, its tight integration with ROS and its large open-source community. The tight integration with ROS provides efficient testing of control software, and the large community means that there exist lots of resources if issues occur during development.

PyBullet

PyBullet is a python module used for physics simulations, robotics, and deep reinforcement learning (RL) based on the Bullet Physics SDK which is written in C++ [5, 4]. PyBullet is easy to use, but it has rather minimal support for relevant sensors compared to other simulators even though it has features such as inverse dynamics, inverse kinematics, deformable objects, and force and torque sensors [3].

NVIDIA Isaac Sim

NVIDIA Isaac Sim was first released in 2019 and has later been updated with the latest version being 2021.2 and version 2022.1 announced at GTC 2022 [6, 24, 23]. Due to its young age, there have not been as many publications using Isaac sim compared to Gazebo or other popular simulators, however, this does not mean that the simulator is not suited for the use case.

NVIDIA Isaac Sim is a robotics simulator based on NVIDIA Omniverse. Isaac sim is suitable for both navigation and manipulation tasks. It simulates sensor data such as RGB-D, LiDAR, and IMU data in addition to supporting ROS and ROS 2 [24]. Isaac SDK is a tightly coupled toolkit that provides building blocks and tools to efficiently develop robotics applications which can easily be tested using Isaac sim [21]

Isaac Sim also supports realistic rendering using real-time ray and path tracing [23]. This is tightly integrated with NVIDIA's GPUs which provides Isaac Sim with one of the most photorealistic environments, while at the same time being efficient. Photorealism is one of Isaac Sim's major benefits. The simulator also features a synthetic data generation tool for generating photorealistic datasets which makes training and deploying AI models more efficient.

One of the drawbacks of Isaac sim is that it is still in an early stage, and the resources available are somewhat limited. The documentation is comprehensive, but the lack of an established community might be a disadvantage. The simulator is proprietary which for some might also be a drawback.

Despite some drawbacks, Isaac sim looks to be a promising robotics simulator with the necessary features for developing and testing the mobile manipulator, KMR.

4.2 Sim-to-Real

Simulators create a replication of the real world, however, they are never perfect. This gap between these two worlds is called the *sim-to-real gap*, or sometimes the *reality gap*. For the development of robotics applications, it is essential that this gap is small so that the testing in the simulator actually represents how the robot will behave in the real world. This is also an important factor when developing robots using RL transferred onto physical robots.

The sim-to-real problem is mainly related to RL applications, however, this section will briefly discuss some challenges and possible solutions. This is also relevant for mobile manipulators as they may also be used in robotic applications utilizing RL. This has been done with some degree of success so far [33], however, researchers have had larger success with RL using stationary manipulators [28, 12].

4.2.1 How to narrow the sim-to-real gap

There exist multiple methods for narrowing the sim-to-real gap as presented in [35].

Domain Randomization

As the domain of the real world and the domain of the simulator differs according to the sim-to-real gap, domain randomization attempts to bridge this gap by extending the simulation domain using randomness such that at least some part of the simulation domain overlaps with the domain of the real world as shown in Figure 4.1

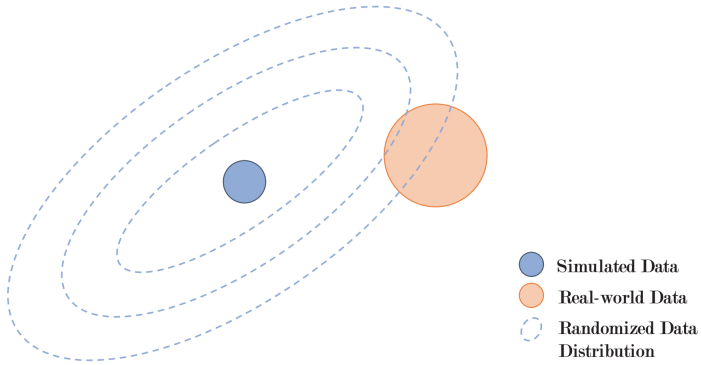


Figure 4.1: Intuition behind domain randomization [35]

[35] divides domain randomization into *visual randomization* and *dynamics randomization* depending on which parts of the simulation are being randomized. Dynamics randomization targets the physical attributes of the simulator such as object dimensions, object and robot link masses, surface friction coefficients, and robot joint damping coefficients [35].

Most RL robotics applications rely heavily on computer vision for tasks such as object detection, pose estimation, and semantic segmentation. A simulation environment will provide objects with different textures and lighting that probably will not work as intended when transferred into the real world. Visual randomization targets this by randomizing different visual attributes in the simulation, e.g. different textures on objects, different camera positions, and different light sources. An example of visual randomization using NVIDIA Isaac Sim is shown in Figure 4.2

Figure 4.2: Example of domain randomization using Isaac sim [24]. Shows different lighting, textures, and object positions while the camera position remains the same.

Simulation environment

Different simulators have different features, as discussed in Section 4.1, and the choice of simulators is key to closing the sim-to-real-gap. In general, more realistic simulators will naturally yield better results, however, one might measure *more realistic* slightly differently based on the application. If the application uses an autonomous vehicle mainly relying on cameras as sensory input, photorealism will be an important feature. However, a simulator used for training an AI for robotic grasping may place more importance on physically accurate simulations.

A mobile manipulator is a hybrid of different robots and thus it requires a general-purpose simulation environment. It needs to be physically accurate to enable realistic testing of robotic grasping with the manipulator while at the same time providing the relevant sensors for AMRs. Cameras would be a natural sensor to utilize in a mobile manipulator application, and for this a photo-realistic environment would be a major benefit.

NVIDIA Isaac sim has not been tested during this project, however, it appears to provide most, if not all, of the necessary features for a digital twin of a mobile manipulator. Many simulators have functionality for different parts of such a robot, but Isaac sim appears to provide support for the entire mobile manipulator and will likely be suited for a digital twin of the KMR iiwa.

Chapter 5

Conclusion and Further Work

5.1 Conclusion

This paper has presented fundamental kinematics and control of both manipulators and wheeled robots focusing on holonomic robots using mecanum wheels, such as the KUKA KMR iiwa. In addition, theory behind navigation and mapping has been presented as this is an essential part of autonomous mobile robots.

The paper also discussed the use of digital twins and how this may be implemented using a robotic simulator. This is a particularly interesting topic as it enables cheaper and more efficient development of robotic applications. In recent years simulators have become more physically accurate which allows for more realistic interactions between the robot and its environment. These advancements have significantly improved simulations of robotic grasping where fine control of the manipulator is essential, and the physics has to be accurate. Previously most robotic simulators used for manipulators have focused on physics whereas photorealistic environments have been a missing feature. However recently simulators such as NVIDIA Isaac sim have introduced photorealistic real-time rendering while at the same time providing physically accurate environments. This allows for more realistic testing of camera-reliant robotic applications.

These advancements contribute to narrowing the sim-to-real gap which has been one of the main challenges facing robotic simulation. This is especially true of machine learning models that are trained in simulation and later deployed in the real world. The goal is that no further training is needed before transferring the models to the real world, but this has usually not been the case because of the sim-to-real gap.

5.2 Further work

Further work on this project may include testing how robots developed using a simulator, such as Isaac sim, behaves in the real world. As this has not been done with the KMR iiwa the first challenge would be implementing a digital twin of the robot in the simulation environment with the necessary controls and sensors. In addition, a simulation environment of the MANULAB at NTNU has to be created to test the robot's behavior.

For the digital twin to be of any value a similar control interface has to be implemented for the digital twin and the robot. ROS 2 is the most commonly used interface in robotics applications and it is the obvious choice for this task as well. Isaac sim supports ROS out of the box which is an advantage, but the KMR iiwa does not. This will require some extra configurations, but some work has been done on this in [10]. If time allows testing a simple pick-and-place task would be interesting as it will test multiple parts of the system.

References

[1] Stefan Boschert and Roland Rosen. *Digital twin-the simulation aspect Mechatronic Futures*. Berlin, Germany: Springer Verlag, 2016.

[2] Luca Carlone, Rosario Aragues, José A. Castellanos, and Basilio Bona. "A fast and accurate approximation for planar pose graph optimization". In: *The International Journal of Robotics Research* 33.7 (2014), pp. 965–987. DOI: 10.1177/0278364914523689.

[3] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. "A Review of Physics Simulators for Robotic Applications". In: *IEEE Access* 9 (2021), pp. 51416–51431. DOI: 10.1109/ACCESS.2021.3068769.

[4] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. 2021. URL: http://pybullet.org (visited on 04/27/2022).

[5] Erwin Coumans, Yunfei Bai, and Jasmine Hsu. *pybullet 3.2.4*. 2022. URL: https://pypi.org/project/pybullet/ (visited on 04/27/2022).

[6] Steve Crowe. "NVIDIA Isaac SDK now available for robotics developers". In: *The Robot Report* (Apr. 19, 2019). URL: https://www.therobotreport.com/nvidia-isaac-sdk-now-available-for-robotics-developers/ (visited on 04/26/2022).

[7] Carlos Faria, Flora Ferreira, Wolfram Erlhagen, Sérgio Monteiro, and Estela Bicho. "Position-based kinematics for 7-DoF serial manipulators with global configuration control, joint limit and singularity avoidance". In: *Mechanism and Machine Theory* 121 (Mar. 2018), pp. 317–334. DOI: 10.1016/j.mechmachtheory.2017.10.025.

[8] Gazebo. *Physics Parameters*. URL: https://classic.gazebosim.org/tutorials?tut=physics_params&cat=physics (visited on 04/26/2022).

[9] Brian Gerkey and Tony Pratkanis. *map_server*. 2020. URL: http://wiki.ros.org/map_server (visited on 05/02/2022).

[10] Charlotte Heggem and Nina Marie Wahl. *Configuration and Control of KMR iiwa with ROS2*. NTNU, Dec. 2019.

[11] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. "Real-time loop closure in 2D LIDAR SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1271–1278. DOI: 10.1109/ICRA.2016.7487258.

[12] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. "Sim-To-Real via Sim-To-Sim: Data-Efficient Robotic Grasping via Randomized-To-Canonical Adaptation Networks". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 12619–12629. DOI: 10.1109/CVPR.2019.01291.

[13] I. Kuhlemann, A. Schweikard, P. Jauer, and F. Ernst. "Robust inverse kinematics by configuration control for redundant manipulators with seven DoF". In: *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*. 2016, pp. 49–55. DOI: 10.1109/ICCAR.2016.7486697.

[14] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. "G2o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949.

[15] Mathieu Labbé and François Michaud. "Long-term online multi-session graph-based SPLAM with memory management". In: *Autonomous Robots* 42.6 (Aug. 2018), pp. 1133–1150. ISSN: 1573-7527. DOI: 10.1007/s10514-017-9682-5.

[16] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. "Industry 4.0". In: *Business & Information Systems Engineering* 6.4 (Aug. 2014), pp. 239–242. ISSN: 1867-0202. DOI: 10.1007/s12599-014-0334-4.

[17] David V. Lu. *Fundamentals of Local Planning*. Unpublished manuscript, Conference notes. 2017. URL: https://roscon.ros.org/2017/presentations/ROSCon%5C%202017%5C%20Fundamentals%5C%20of%5C%20Local%5C%20Planning.pdf.

[18] Kevin M. Lynch and Frank C. Park. *Modern Robotics*. Cambridge, England: Cambridge University Press, 2017. ISBN: 978-1-10-715630-2.

[19] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 2718–2725. DOI: 10.1109/IROS45743.2020.9341207.

[20] NVIDIA. *Fluid Dynamics UAV, blimp, subsea*. 2021. URL: https://forums.developer.nvidia.com/t/fluid-dynamics-uav-blimp-subsea/187145/2?u=user164 (visited on 04/20/2022).

[21] NVIDIA. *Isaac SDK*. 2022. URL: https://docs.nvidia.com/isaac/isaac/doc/index.html (visited on 04/20/2022).

[22] NVIDIA. *Multi robot control support*. 2021. URL: https://forums.developer.nvidia.com/t/multi-robot-control-support/184937/2?u=user164 (visited on 04/20/2022).

[23] NVIDIA. *NVIDIA Isaac Sim*. 2022. URL: https://developer.nvidia.com/isaac-sim (visited on 04/26/2022).

[24] NVIDIA. *Omniverse: Isaac Sim*. 2022. URL: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim.html (visited on 04/20/2022).

[25] NVIDIA. *Physics Core - Omniverse Create*. 2022. URL: https://docs.omniverse.nvidia.com/app_create/prod_extensions/ext_physics.html (visited on 04/20/2022).

[26] NVIDIA. *Questions regarding USD scenes and Isaac Sim*. 2021. URL: https://forums.developer.nvidia.com/t/questions-regarding-usd-scenes-and-isaac-sim/186793/5?u=user164 (visited on 04/20/2022).

[27] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, and TaeHoon Lim. *ROS Robot Programming: A Handbook is written by TurtleBot3 Developers*. ROBOTIS Co.,Ltd., 2017. ISBN: 979-11-962307-1-5.

[28] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. "RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[29] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. "A review of mobile robots: Concepts, methods, theoretical framework, and applications". In: *International Journal of Advanced Robotic Systems* 16.2 (2019). DOI: 10.1177/1729881419839596.

[30] Cyrill Stachniss, John J. Leonard, and Sebastian Thrun. "Springer Handbook of Robotics". In: Berlin, Heidelberg: Springer, 2008. Chap. 46. Simultaneous Localization and Mapping.

[31] Hamid Taheri, Bing Qiao, and Nurallah Ghaeminezhad. "Kinematic Model of a Four Mecanum Wheeled Mobile Robot". In: *International Journal of Computer Applications* 113 (Mar. 2015), pp. 6–9. DOI: 10.5120/19804-1586.

[32] Pavel Trutman, Mohab Safey El Din, Didier Henrion, and Tomas Pajdla. "Globally Optimal Solution to Inverse Kinematics of 7DOF Serial Manipulator". In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 6012–6019. DOI: 10.1109/LRA.2022.3163444.

[33] Cong Wang, Qifeng Zhang, Qiyan Tian, Shuo Li, Xiaohui Wang, David Lane, Yvan Petillot, and Sen Wang. “Learning Mobile Manipulation through Deep Reinforcement Learning”. In: *Sensors* 20.3 (2020). ISSN: 1424-8220. URL: https://www.mdpi.com/1424-8220/20/3/939.

[34] Brian Wilcox. *ROS2 Index: nav2_map_server*. 2022. URL: https://index.ros.org/p/nav2_map_server/ (visited on 05/02/2022).

[35] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.