Berliner Hochschule für Technik

Haptic exploration and modeling of unknown mechanisms

presented by

Yu Jen Chen

EDV.Nr.:902273

Bachelor's thesis submitted to the Department VII – Electrical Engineering – Mechatronics – Optometry of the Berlin
University of Applied Sciences
Berlin for obtaining the academic degree

Bachelor of Engineering (BEng.)

in the course

Humanoid Robotics

Date of submission March 22, 2023

Studiere Zukunft

Supervisor
Prof. Dr. Manfred Hild Berlin University of Technology
Reviewer
Dr. Simon Untergasser Berlin University of Applied Sciences

short version

The aim of this work is to develop a robot based on a kinematically free model, ie a robot without knowledge of its own mechanisms, which explores the environments using sensorless haptic feedback, creates a map and identifies the explored environments. A five-bar linkage two-degree-of-freedom robotic manipulator was developed to explore the labyrinths as unfamiliar environments. Current-based collision detection was used as haptic feedback. To control the robot, a virtual mobile robot is formed in a conguration space and controlled by the minimal recurrent controller, a feedback neural network made up of two neurons. In the absence of a kinematic model, the model of the map created by the robot leads to the conclusion that the robot has fully explored the unknown environment and is able to recognize different environments to a certain extent.

This shows that it is possible to control the behavior of the robot without giving it a kinematic model. This can be achieved without machine learning methods or optimization controls.

Abstracts

The aim of this work is to develop a robot based on a kinematic-free model, ie, a robot wi thout knowledge of its mechanisms, which explores the environments using sensor less haptic feedback, creates a map and identities the explored environments. A ve-bar-linkage-robot manipulator with two degrees of freedom was developed to explore mazes as unknown environments. Current-based collision detection was used as haptic feedback. To control the robot, a virtual mobile robot is formed in Conguration Space and controlled by Minimal Recurrent Controller, a feedback neural network of two neurons. In the absence of a kinematic model, the model of the map created by the robot leads to the result that the robot has fully explored the unknown environment and is able to recognize dierent environments to some extent. This proves that it is possible to control the robot's behavior without giving it a kinematic model. This can be achieved without machine learning methods or optimization control.

Explanation

I certify that I have written this thesis independently without outside help and have only used the sources and aids indicated. Excerpts taken from other works, either literally or in spirit, are identified and the sources are indicated.

Date Signature

Table of contents

List of Figures

List of Tables

Machine Translated by Google

Chapter 1

Introduction

In a dark room, people typically rely on haptic perception to recognize and explore their surroundings, such as touching walls, sensing the location of obstacles, or looking for the light switch. In robotics, environmental exploration is also an important area of research because of the need to build an understanding of the environment before or during task performance. This helps avoid obstacles, reduce damage, and improve task completion efficiency.

In order to develop more intelligent robots that can explore their environment like humans, it is worth investigating how to explore an environment with haptic perception without visual support. This ability can help robots be more flexible and able to work in poorly lit environments, while also helping robots better adapt to different environments.

Therefore, this human ability can serve as a reference and inspiration for robotics. By mimicking human behavior and learning from robots' experiences, intelligent and more efficient robots can be developed to achieve wider application.

1.1 Issue

In the field of robotics, there are many perception methods that work in a similar way to human perception. It is common to use sensors such as LiDAR or optical cameras to perceive the environment of robots. In comparison, however, there are some irreplaceable features of the perception methods based on haptic feedback: For example, unlike LiDAR, haptic feedback is usually not limited by the material of the object to be detected (such as glass) or can a came ra are affected by ambient light and lose their function. The methods of haptic perception can be divided into sensory and sensorless. Compared to sensorless approach, sensory approach can provide more precise and rich information. However, sensorless haptic feedback is more economical and is therefore also being investigated.

As previously mentioned, humans explore their surroundings by stretching out their limbs. When using robots, it can be taken into account how humanoid robots or manipulators move and where they should move in order to explore the environment. In order to control the movement of the robot, it is often necessary to have the kinematic model of the robot, which means that complex mathematical models for forward and

Reverse kinematics must be created. This not only increases the complexity of the algorithm, but also requires more effort from the controller.

After people have explored the environment, they often create a model of the environment and know, for example, where light switches or obstacles are located. Similarly, when exploring an unfamiliar environment, robots need to understand their environment, including the spatial structure of the environment, the location and shape of obstacles
etc.

Taking the above sensory and control-related aspects into account, it would be possible to overcome the need for haptic exploration of the environment without the use of sensors and kinematic models, and develop a certain level of understanding of the environment, resulting in a theoretically and economically leaner approach for exploration leads.

1.2 State of Research

According to [16, 22], strategies for exploring the environment can generally be divided into two types: model-based and reactive. The former are often used to solve exploration problems in SLAM (Simultaneous Localization and Mapping) and also robotic manipulator based environmental exploration[20, 26], where the basic idea is to explore the least explored area using Information Theory methods [25, 23] or Frontier Based Exploration [4, 28]. The latter, on the other hand, couple perception and action directly, e.g. B. Braitenberg Vechicle[3], Minimal Recurrent Controller[19, 11] or the common algorithm for exploring labyrinths Wall follower[15]. While the former is less commonly used in controlling robotic manipulators, it's worth investigating how it can be applied here as it's more responsive and easier to implement.

Various approaches have been developed for sensorless haptic perception and collision detection in robot manipulators, e.g. e.g. [10, 5]. In the work by Bethge [2], an approach to collision detection using artificial neural networks is presented. Furthermore, impedance control is used to control a robot in [13] that explores a maze using sensorless haptic feedback.

Furthermore, model-free control methods have been investigated, but these usually require the use of machine learning, which takes time and a training amount [29].

1.3 Objective of the work

In order to enable an exploration method that is as lean as possible and based on haptic feedback, an exploration algorithm for the robot manipulator is to be developed in this work on a sensorless haptic feedback and in particular on an unknown kinematic model, i.e. "the robot does not know its own mechanism". The algorithm is used implemented on a robotic manipulator consisting of two servomotors.

The robot will be designed to work in two modes. Mode 1 focuses on building a simple telerobot with the goal of extending the control method in Mode 2 without a kinematic model and sensorless collision detection. In Mode 2, a five-bar linkage is used to explore the maze, modeling and comparing maps of the unknown environment.

1.4 Structure of the work

The entire essay is structured as follows: All relevant technical details and the hardware design are presented in Chapter 2. Chapter 3 follows, introducing the basics of using the two modes. In chapters 4 and 5 an overview and a detailed description of the algorithms for modes 1. and 2. are given, together with the results and a summary and an outlook, respectively.

Chapter 2

Description of the used robot

The robot used in this work is presented below. The circuit diagram, the most important components used and the hardware conguration for the two modes are described.

2.1 Overview of the robot

The robot consists of two servo motors, a main board, a microcontroller, four buttons, a monitor and two different levers. Below is a schematic of the entire system. The motor is not marked in the diagram as it is connected directly to the microcontroller.

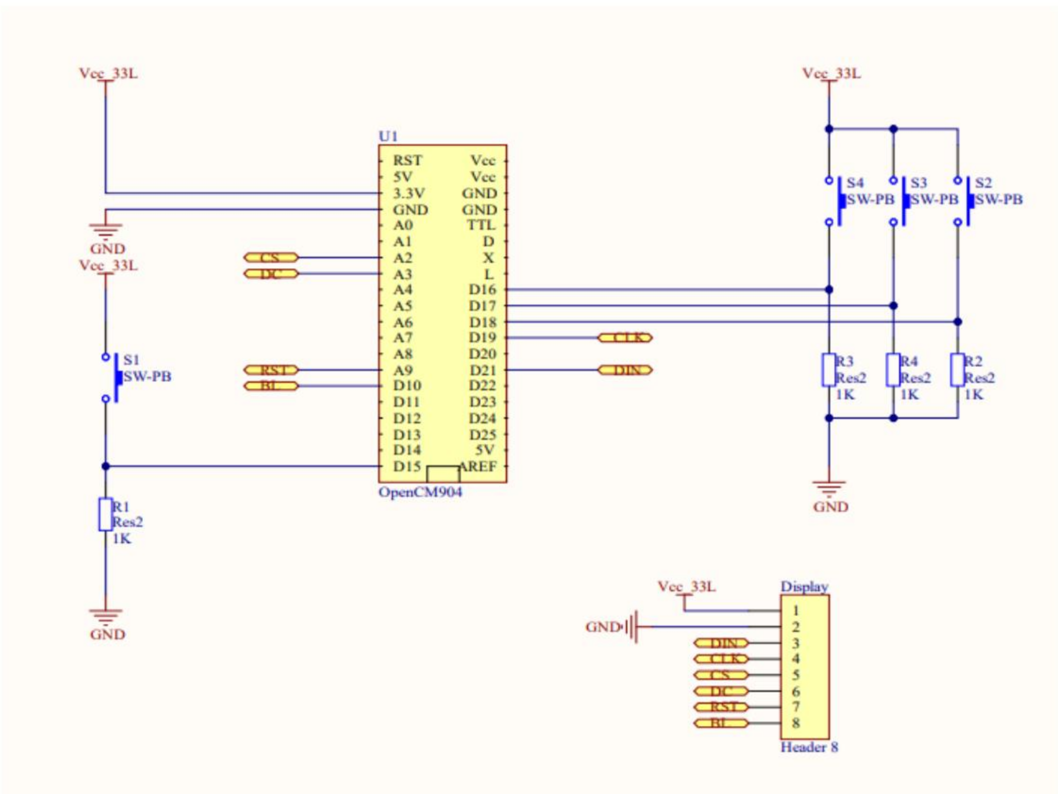


Figure 2.1: Circuit diagram of the robot's main circuit board

2.2 Components of the robot

2.2.1 Engines

The servomotors on the robot are two motors from ROBOTIS type XL330-M077-T.
The motors can be operated in different modes:

1. Current control mode

2.Velocity Control Mode

3.Velocity Control Mode

4. Current-based position control mode

5.PWM Control Mode

The two servomotors are controlled with PWM (Pulse-width Modulation) control mode.
The motors are controlled by the microcontroller via TTL communication. About it

In addition, the current speed, current angle of rotation, current and input voltage of the motor can also be read from it.

Figure 2.2: XL330-M077-T Servo Motor by ROBOTIS Co., Ltd.
Source: XL330-M077-T[product image]. (2023). ROBOTIS Co.,Ltd. Retrieved from https://emanual.robotis.com/docs/en/dxl/x/xl330-m077/

2.2.2 Monitor

The monitor is a 2.4 inch LCD mode 1 from Waveshare Electronics based on the SPI protocol and is mainly used to display the status of the robot.

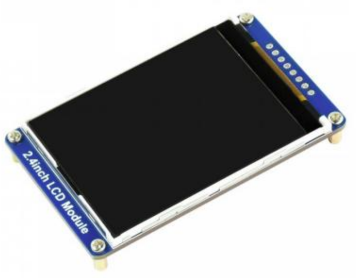


Figure 2.3: 2.4 inch LCD mode 1 Waveshare Electronics Source: 240×320, General 2.4 inch LCD monitor mode 1e, 65K RGB[product image]. (2023). Waveshare Electronics. Retrieved from https://www.waveshare.com/2.4inch-LCD-Modus1e.htm

Machine Translated by Google

2.2.3 Microcontrollers

OpenCM9.04 Type C from ROBOTIS is used as the controller for the robot. It is an ARM Cortex-M3 microprocessor-based controller with a clock speed of 72 MHz, 128 KB Flash memory and 20 KB SRAM memory. It also has several common I/O interfaces such as UART, SPI, I2C, PWM, etc. The main feature of the OpenCM9.04 compared to other controllers is the direct connection to the servo motors. This allows no additional circuitry to be required to drive the motors, reducing the need for hardware design. Details of the software and hardware applications are listed below:

programming

The programming of the robot is based on the Arduino framework [21] and is written in the programming language C and C++. The robot is programmed with two loops in both mode 1 and mode 2, with the sampling frequency of the loops being slightly different.

In Mode 1, Loop 1 reads the battery voltage at a rate of 1Hz, while Loop 2 does most of the work of Mode 1 and updates the monitor at a rate of 25Hz.

In the mode 2 program, loop 1 reads the battery voltage and updates the monitor at 10 Hz, while loop 2 explores mode 2 at 25 Hz. Since Mode 2 is primarily geared towards exploring the environment, it is a long-term task that does not require constant updating of the monitor, so the sample rate of Loop 1 is reduced to reduce overhead.

hardware interrupts

Four hardware interrupts are set up on the robot's four buttons, which enable basic operations such as pausing the run and resuming the run, etc.

Figure 2.4: OpenCM9.04 typeC microcontroller Source:
OpenCM9.04 typeC[product image]. (2023). ROBOTIS Co.,Ltd. Retrieved from
https://emanual.robotis.com/docs/en/parts/controller/opencm904/

2.2.4 Mechanical Conguration for Mode 1 and Mode 2

In mode 1 and mode 2, the mechanical conguration is different. In mode 1, the two servomotors are each connected with just one rod, in mode 2 the two motors are connected to each other via a five-link linkage. The pivot at the center of the mechanism has an isolation column as an endector, used to sense the environment serves. Figures 2.5 and 2.6 show the 3D model and the plan view of the technical drawing of the two modes, where only the rod lengths are indicated.

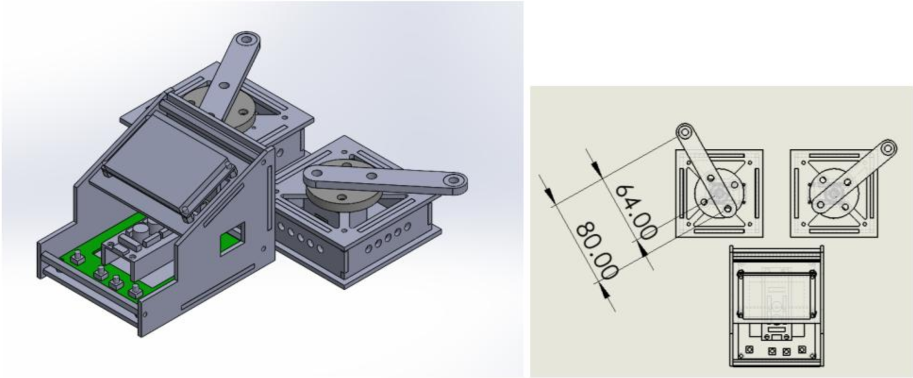


Figure 2.5: 3D model and technical drawing in mode 1

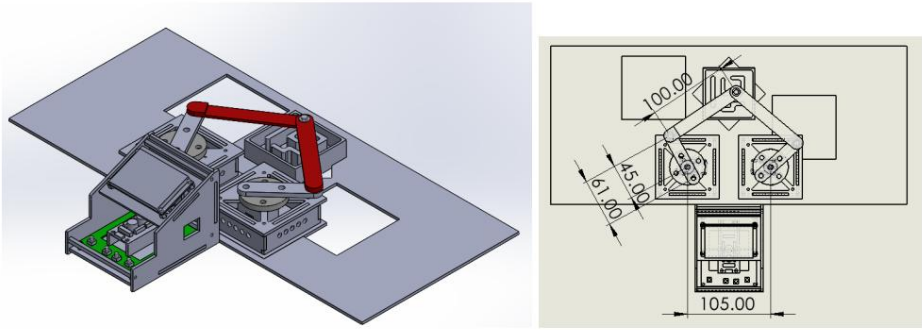


Figure 2.6: 3D model and technical drawing in mode 2

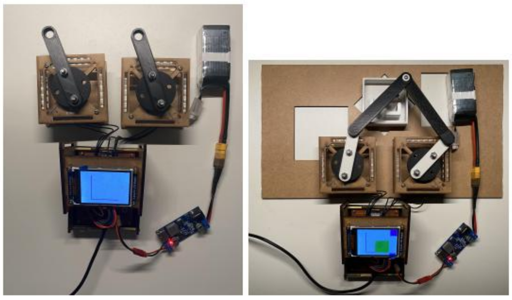


Figure 2.7: Structure of the robot in mode 1 (left) and mode 2 (right)

Chapter 3

Thematic basics and practical implementation

In the following, the basics, which are applied both in these two modes together, as well as in their respective individual applications, are presented. The basics used in this work are described both in general terms and in their practical implementation.

3.1 The theoretical basis of mode 1 and mode 2

3.1.1 Degrees of Freedom

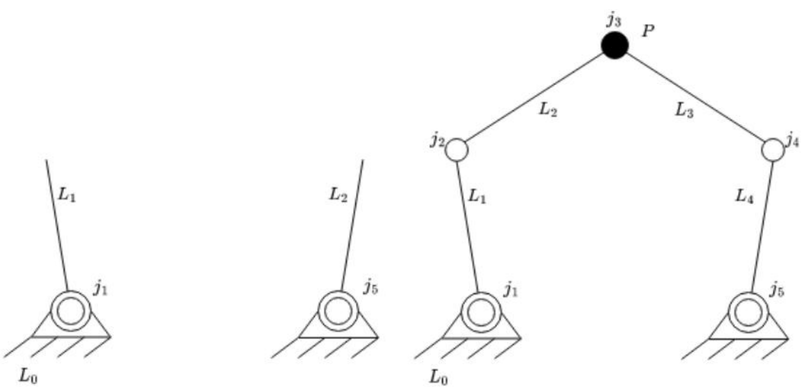


Figure 3.1: Schematic representation of the mechanisms in mode 1 (left) and mode 2 (right)

The degrees of freedom of a mechanism is dened as the minimum independent parameters required to describe the position of each component in the mechanism. To determine the degree of freedom of the robot's mechanism, Grübler's equation can be used as follows [17]:

$$F = m(N - 1) - \sum_{i=1}^{j} c_i \tag{3.1}$$

F : degree of
freedom m : type of gear (m = 6 for spatial, m = 3 for plane gear)
N : number of gear links (Li) j :
number of joints (ji) ci :
inevitability of a single joint

In this work, the robot moves with swivel joints (ci = 2) in a plane (m = 2). The degree of freedom of the mechanism for mode 1 and mode 2 can be calculated as follows: Mode 1:

$$j = 2$$
$$N=3$$
$$F = 3(3 ÿ 1) ÿ \sum_{i=1}^{2} 2 = 2$$

Mode 2:

$$j = 5$$
$$N=5$$
$$F = 3(5 ÿ 1) ÿ \sum_{i=1}^{5} 2 = 2$$

3.1.2 Configuration

In robotics, conguration is a representation used to fully represent a robot's state. For a robot, the conguration can be represented by a set of coordinates (mobile robot) or by a set of joint angles (robot manipulator), depending on the robot's motion and conguration. Apart from that, a good explanation is given in the book “Modern Robotics”[17, p.11]: “The minimum number n of real-valued coordinates needed to represent the conguration is the number of degrees of freedom (dof) of the robot. ”

According to [17], the dimension of conguration is the degree of freedom. It was also already stated in 3.1.1 that the degree of freedom of both modes is 2, i.e. the dimension of the conguration is 2. Here, conguration of the robot can be described by (ÿ1, ÿ2) , where ÿ1[ÿ] and ÿ2[ÿ] are the rotation angles of both at pivot 1 and pivot 5 as shown in Figure 3.1.

3.1.3 Configuration Space

The conguration space (C-space) is used in robotics to describe all possible conguration. It is the **abstract mathematical** space in which all possible congurations are gathered. Each state of the robot is represented by a dot in the C-Sapce.

As is well known, the degree of freedom is the number of independent variables, and the degree of freedom corresponds to the dimension of the conguration. So if the system is controlled by N independent variables, its C space is an N-dimensional space.

So in this work, two motors are mounted on a robot whose rotation angle is an independent variable. The resulting degree of freedom of the two modes is 2, so that the dimension of the conguration and the C space formed by the conguration is also 2. The resulting 2-dimensional space S plays an important role in this work.

$$S = \{(ÿ1, ÿ2) \mid ÿ1\ ÿ\ [0, 360), ÿ2\ ÿ\ [0, 360)\} \quad (3.2)$$

collision detection

The XL-330-M7 servo motor used is a DC motor that can be used to detect a collision with an obstacle using electricity. The supply voltage of the motor is U, the internal resistance R, back EMF E and the current I. The equation for the voltage in the steady state is as follows[9]:

$$U = E + IR \quad (3.3)$$

According to [12] Back EMF, the back EMF constant Kb and engine speed ÿ can be determined.

$$E = Kbÿ \quad (3.4)$$

This means that E is proportional to ÿ, i.e. assuming that the motor is ideally blocked during rotation by an obstacle, this means that at constant U, ÿ = 0, E = 0, which according to the equation results in an increase in I. So here it is possible to detect a collision by reading the current value to see if a certain threshold is exceeded. No additional sensor is required to achieve this.

3.2 The theoretical basis of mode 1

3.2.1 PID controller

A PID controller is a closed-loop controller that generates control signals to adjust the behavior of the system by comparing the difference between the setpoint and the actual value (ie, the error). The PID controller consists of three parts: P part, I part and D part. Below is a brief overview of the three parts:

1. The function of the proportional term is to increase the output by scaling the dev adjustment between the setpoint and the actual value.

2. The I component is used to adapt the output variable using the deviation accumulated over time and can be used to eliminate permanent control deviation.

3. The D term is used to adjust the output according to the rate of change of the difference between the current and past deviation. This is often used to improve system response time.

The mathematical expression for the controller in a digital system is as follows[1]:

$$uk = Kpek + Ki \sum_{j=0}^{k} ej + Kd(ek\ ÿ\ ekÿ1) \quad (3.5)$$

k : sampling
sequence uk : output variable at the k-th sampling
time ek : deviation from the k-th sampling time
ekÿ1 : deviation from the k ÿ 1-th sampling time
Kp : proportional factor
Ki : integration factor
Kd : dirential factor

The P, I and D terms can be adjusted by the parameters Kp, Ki and Kd in equation 3.5. Removing the D component (set Kd to 0) or the I component (set Ki to 0) results in three further variants of P controllers, PI controllers and PD controllers. In this work only the PD controller is used, ie the I component is removed.

3.3 The theoretical basis of mode 2

3.3.1 Work Space

In robotics, the robot's workspace is the area in **real** space where the robot can actually work and perform its intended task. It is dened by the limits of the robot's movement. In general, the work space can be dened as the area that the end ector can reach. An endeector is a component attached to the end of a robot manipulator that is used to interact with the environment. Examples of endectors are grippers, suction cups or samples.

The robot in Mode 2 uses the five-bar linkage to control the end effector to explore the environment. To determine the area that the robot can explore, it is necessary; examine the work space of the robot.
The work space of the five-bar linkage can be determined as shown in [7]:

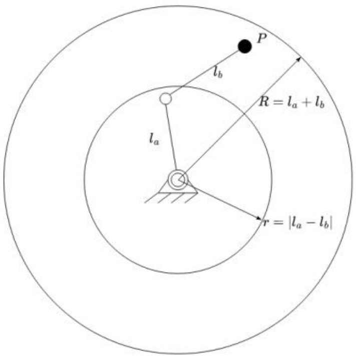


Figure 3.2: Work space of the two-pivots-in-line

1. A five-bar linkage can be divided into two-pivots-in-line.
   The work space of the mechanism is the area that the point P, namely the endeector, can reach (see Figure 3.2).
2. The work space where the two two-pivots-in-row overlap is the
   Robot work space (see Figure 3.3).

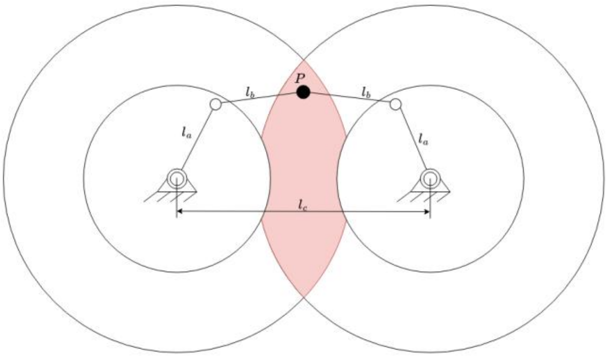
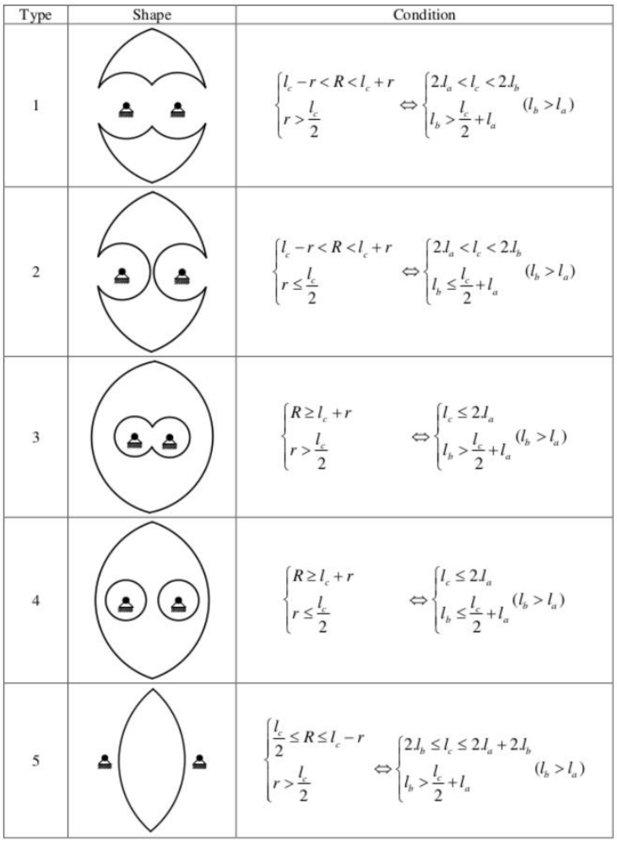


Figure 3.3: Work space of the robot based on a five-bar linkage

According to the relationship between la, lb and lc in Figure 3.3, the robot can be divided into 5 types depending on the work space. A detailed breakdown of all congurations is shown in Figure 3.4.



Type	Shape	Condition
1		$\begin{cases} l_c - r < R < l_c + r \\ r > \frac{l_c}{2} \end{cases} \Leftrightarrow \begin{cases} 2l_a < l_c < 2l_b \\ l_b > \frac{l_c}{2} + l_a \end{cases} (l_b > l_a)$
2		$\begin{cases} l_c - r < R < l_c + r \\ r \leq \frac{l_c}{2} \end{cases} \Leftrightarrow \begin{cases} 2l_a < l_c < 2l_b \\ l_b \leq \frac{l_c}{2} + l_a \end{cases} (l_b > l_a)$
3		$\begin{cases} R \geq l_c + r \\ r > \frac{l_c}{2} \end{cases} \Leftrightarrow \begin{cases} l_c \leq 2l_a \\ l_b > \frac{l_c}{2} + l_a \end{cases} (l_b > l_a)$
4		$\begin{cases} R \geq l_c + r \\ r \leq \frac{l_c}{2} \end{cases} \Leftrightarrow \begin{cases} l_c \leq 2l_a \\ l_b \leq \frac{l_c}{2} + l_a \end{cases} (l_b > l_a)$
5		$\begin{cases} \frac{l_c}{2} \leq R \leq l_c - r \\ r > \frac{l_c}{2} \end{cases} \Leftrightarrow \begin{cases} 2l_b \leq l_c \leq 2l_a + 2l_b \\ l_b > \frac{l_c}{2} + l_a \end{cases} (l_b > l_a)$

Figure 3.4: 5 types of congurations of a robot with a five-bar linkage Source:[7, table. 1]

3.3.2 Singularity

In robotics, a singularity can be intuitively understood as a specific position in the work space at which a robotic manipulator or kinematic system loses all or part of its degree of freedom. From a mathematical point of view, there is a matrix in the kinematic equations of a robot manipulator or robot, called the Jacobian matrix, that describes the mapping of the conguration from each robot joint position to the rate of change of the endector posture in the work sapce. The occurrence of singularities leads to the Jacobian matrix not having full rank, ie the transformed vectors are not linearly independent and the robot's mechanism loses degrees of freedom. For example, a robot that can move arbitrarily in the plane can only move parallel to the one-axis at the singularity.

The four cases of the occurrence of singularities of a robot five-bar linkage in this work are shown in Figure 3.5:

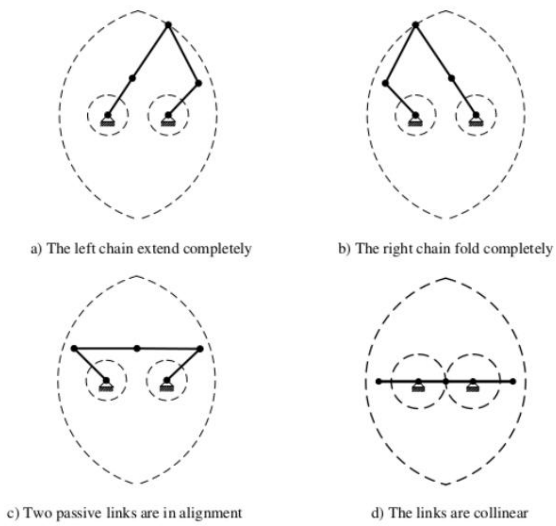


Figure 3.5: Singularity of the five-bar linkage Source:[7, gure. 8th]

By design, Type I singularities (a and b in Figure 3.5) are always present and occur at the boundary of the work space. These can be avoided by restricting the work space, while type II(c in Figure 3.5) and type III(d in Figure 3.5) singularities can be eliminated by appropriate values for the link length. To avoid occurrence of singularities and prevent interference between two active rods, **type 1** conguration (1 in Figure 3.4) on robots is used in this work [7].

3.3.3 Minimal Recurrent Controllers

The Minimal Recurrent Controller (MRC) is a feedback neural network consisting of two neurons (see Figure 3.6). It is proposed in [11, 19] for the control of a dierential robot with two distance sensors for autonomous exploration and obstacle avoidance of the environment. MRC is a reactive approach in which the robot can explore the environment very robustly and avoid obstacles. Compared to the Braintenberg Viechle [3], this avoids the problem of robots equipped with two distance sensors often getting stuck in corners. The input units of this neural network are Sl and Sr, which map the values of the distance sensors between ÿ1 and 1 (where ÿ1 stands for no obstacle and 1 for a detected obstacle); the initial values are for the backwards and forwards movement Ml and Mr, which are also between ÿ1 and 1. The tanh function is used as the activation function for the neurons N1 and N2 in this neural network. According to [19], the relationships between the weights can be arranged as follows to achieve the desired effect of this neural network:

$$w_1, w_3 > \tag{3.6}$$

$$1\ w_r \text{ ÿ ÿ}4w_1, w_l \text{ ÿ ÿ}4w_2 \tag{3.7}$$

$$w_2 \text{ ÿ ÿ}2w_1, w_4 \text{ ÿ ÿ}2w_3 \tag{3.8}$$

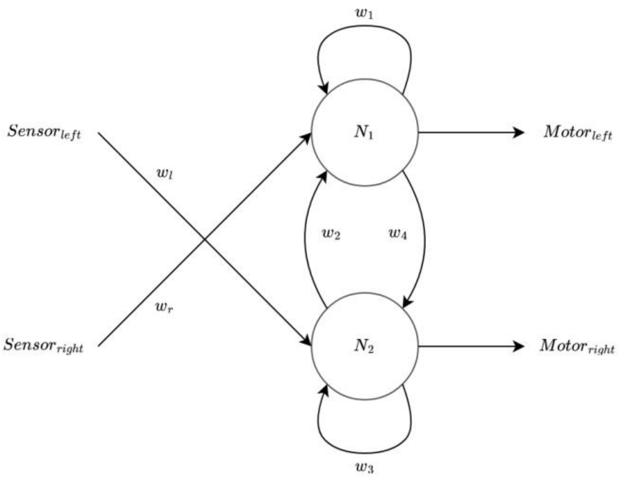


Figure 3.6: MRC neural network

3.3.4 Moment in image processing

Moment is a mathematical way of describing an image. The pixel values and coordinate positions in an image are converted into a series of mathematical values and used in applications such as image processing, image recognition and image analysis.

Moment is often used to describe segmented image objects. A number of properties of an image can be determined from moments, including area, information about the geometric center or the direction of rotation. The moments used in this work are as follows [14, 8]:

- Off-center moments

$$m_{pq} = \sum_{x}\sum_{y} x^p y^q I(x, y) \tag{3.9}$$

- Key moments

$$\mu_{p,q} = \sum_{ij} (x \ddot{y} \bar{x})^p (y \ddot{y} \bar{y})^q I(x, y),\ \bar{x} = \frac{m_{10}}{m_{00}},\ \bar{x} = \frac{m_{01}}{m_{00}} \tag{3.10}$$

- Scale-invariant moments

$$\ddot{y}_{pq} = \frac{\mu_{pq}}{\mu_{00}^{(1+\frac{p+q}{2})}} \tag{3.11}$$

where p + q are the order of the moments and x and y are the pixel coordinates.

- Hu moments

$$\begin{aligned}
Hu_1 &= \ddot{y}_{20} + \ddot{y}_{02} \\
Hu_2 &= (\ddot{y}_{20}\ \ddot{y}\ \ddot{y}_{02})^2 + 4\ddot{y}_{11}^2 \\
Hu_3 &= (\ddot{y}_{30}\ \ddot{y}\ 3\ddot{y}_{12})^2 + (3\ddot{y}_{21}\ \ddot{y}\ \ddot{y}_{03})^2 \\
Hu_4 &= (\ddot{y}_{30} + \ddot{y}_{12})^2 + (\ddot{y}_{21} + \ddot{y}_{03})^2 \\
Hu_5 &= (\ddot{y}_{30}\ \ddot{y}\ 3\ddot{y}_{12})(\ddot{y}_{30} + \ddot{y}_{12})((\ddot{y}_{30} + \ddot{y}_{12})^2\ \ddot{y}\ 3(\ddot{y}_{21} + \ddot{y}_{03})^2) \\
&\quad + (3\ddot{y}_{21}\ \ddot{y}\ \ddot{y}_{03})(\ddot{y}_{21} + \ddot{y}_{03})(3(\ddot{y}_{30} + \ddot{y}_{12})^2\ \ddot{y}\ (\ddot{y}_{21} + \ddot{y}_{03})^2) \\
Hu_6 &= (\ddot{y}_{20}\ \ddot{y}\ \ddot{y}_{02})((\ddot{y}_{30} + \ddot{y}_{12})^2\ \ddot{y}\ (\ddot{y}_{21} + \ddot{y}_{03})^2) + 4\ddot{y}_{11}(\ddot{y}_{30} + \ddot{y}_{12})(\ddot{y}_{21} + \ddot{y}_{03}) \\
Hu_7 &= (3\ddot{y}_{21}\ \ddot{y}\ \ddot{y}_{03})(\ddot{y}_{30} + \ddot{y}_{12})((\ddot{y}_{30} + \ddot{y}_{12})^2\ \ddot{y}\ 3(\ddot{y}_{21} + \ddot{y}_{03})^2) \\
&\quad \ddot{y}\ (\ddot{y}_{30}\ \ddot{y}\ 3\ddot{y}_{12})(\ddot{y}_{21} + \ddot{y}_{03})(3(\ddot{y}_{30} + \ddot{y}_{12})^2\ \ddot{y}\ (\ddot{y}_{21} + \ddot{y}_{03})^2)
\end{aligned}$$

Where I(x,y) is the pixel value at position (x,y) in the image. Binary images are used in this work, so I(x, y) is:

$$I(x,y) = \begin{cases} 1 & \text{object} \\ 0 & \text{background} \end{cases} \tag{3.12}$$

The highest order descriptor used in this work is p + q ÿ 2, where descriptor 0th order p + q = 0 is usually used to describe the area, descriptor 1st order p + q = 1 to describe the centroid and descriptor 2 order p + q = 2 is used to describe the distribution of the pixels [27].

Chapter 4

mode 1

This chapter describes in detail the purpose, the mode of operation , the results and one summary in mode 1.

4.1 Uses of Mode 1

in mode 1, the user can manually turn motor 1 M1 so that motor 2 M2 always stays at the same angle (see figure 4.1). If an obstacle (i.e. a collision) occurs while motor 2 is turning towards M1 , M1 must be as close as possible stay at the position where M2 can no longer rotate due to the collision.In this case, the user can no longer rotate the rod on the M1 , but can sense the virtual collision by touch.In addition, the position where the collision has taken place is visualized on an external monitor (see Figure 4.2).

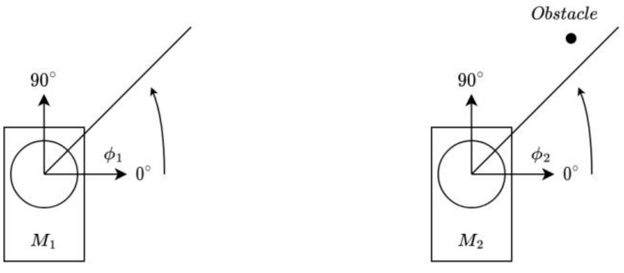


Figure 4.1: Schematic representation of the robot in mode 1

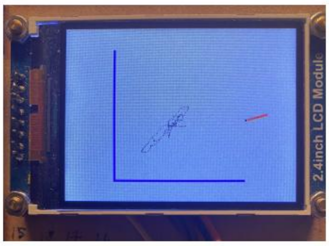


Figure 4.2: The monitor on the robot in mode 1 A collision at around 180[ÿ]

Machine Translated by Google

4.2 Overview of the algorithm

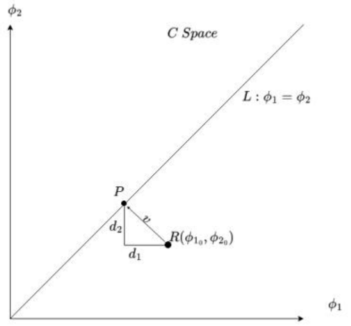


Figure 4.3: Schematic representation of the C Space in mode 1

The problem with mode 1 is that both motors must always be in the same position. In terms of the C space, it is true that R(ÿ10 , ÿ20), the current conguration, must always remain on the straight line L : ÿ1 = ÿ2 in mode 1 (see Figure 4.3). So the main problem is to control the motors in such a way that if there is a deviation, the point R must move to the projection point P, since P is the closest point to R on the line ÿ1 = ÿ2 .

4.3 Detailed description of the algorithm

The algorithm here uses the two PD controllers to control two motors to control the horizontal and vertical position of R, respectively, and the inputs to the two controllers. The deviation, that is, the deviation between the positions on P and R in the horizontal and vertical directions, are d1 and d2 in Figure 4.3, which can be determined as follows:

In order to determine d1 and d2 , the coordinate of P must first be determined, which can be determined by projecting it onto a straight line:

$$P\left(\frac{\text{b 2ÿ10 ÿ abÿ20 ÿ ac}}{\text{2 a + b 2}}, \frac{\text{ÿ abÿ10 + a 2ÿ20 ÿ bc}}{\text{2 a + b 2}}\right) \tag{4.1}$$

Here a,b and c are the coefficients of the linear equation L : aÿ1 +bÿ2 +c = 0, here a = 1, b = ÿ1 and c = 0. Inserting a, b and c into equation 4.1 gives P:

$$P\left(\frac{\text{ÿ10 + ÿ20}}{2}, \frac{\text{ÿ10 + ÿ20}}{2}\right) \tag{4.2}$$

Since in control engineering deviation = setpoint ÿ actual value applies, d1 and d2 can be determined as follows:

$$\text{d1} = \frac{\text{ÿ10 + ÿ20}}{2} \text{ ÿ ÿ10} = \frac{\text{ÿ20 ÿ ÿ10}}{2} \tag{4.3}$$

$$\text{d2} = \frac{\text{ÿ10 + ÿ20}}{2} \text{ ÿ ÿ20} = \frac{\text{ÿ10 ÿ ÿ20}}{2} \tag{4.4}$$

Here, d1 and d2 are used as input variables for the PD controller. A schematic representation of the control loop is shown in Figure 4.4.

Machine Translated by Google

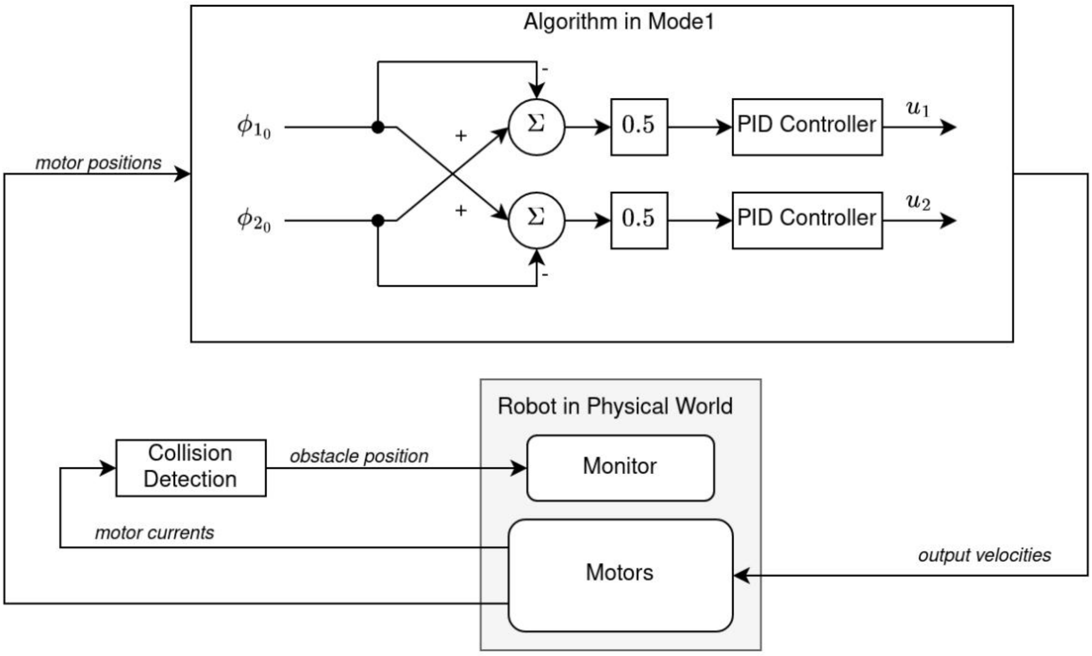


Figure 4.4: Schematic representation of the mode 1 algorithm

4.4 Results

C Space **based position control**

A visual representation of the C Space robot using PD control at Mode 1 is shown in the following figures. The yellow dots refer to conguration. It can be clearly seen that conguration is next to or on the line of the center line ÿ1 = ÿ2 . Some congurations are slightly offset due to the angle difference between the rod of the motor when rotated by the user. When the bar is released by the user, the conguration moves straight towards the center line as shown in Figure 4.3. In addition, it can be seen that the proportional factor Kp inuences the size and behavior of the middle region, which is smallest at Kp = 1.0 but oscillates in the C sapce. At Kp = 0.3 the middle range is largest.

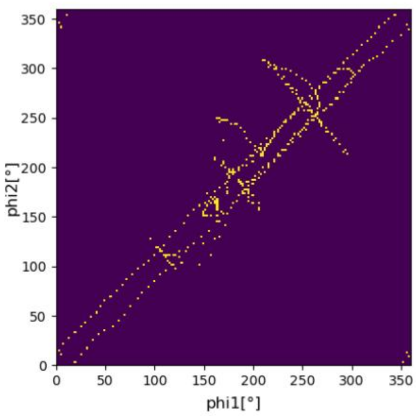


Figure 4.5: Visual representation of the C space in mode 1 PD controller, Kp = 1.0, Kd = 0.25
oscillation at around (250, 250); Collision at circa (160, 250) and (250, 250)

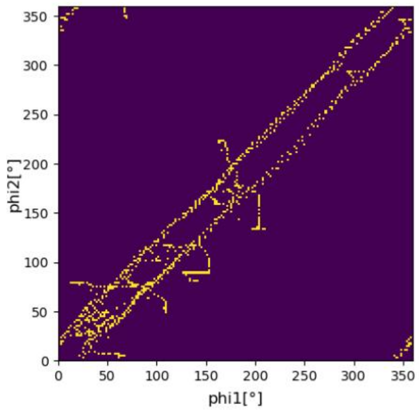


Figure 4.6: Visual representation of the C space with mode 1 PD controller, Kp = 0.7, Kd = 0.25
collision at around (180, 210) and (200, 140)

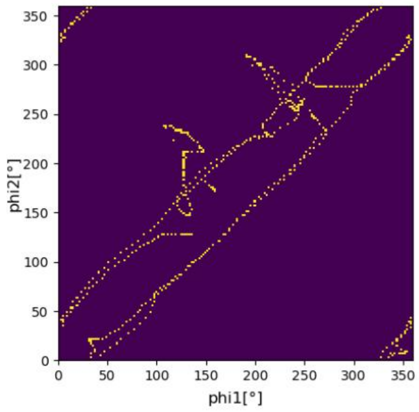


Figure 4.7: Visual representation of the C space with mode 1 PD controller, Kp = 0.3, Kd = 0.25
collision at around (100, 240) and (140, 170)

collision detection

Figures 4.5, 4.6 and 4.7 show points slightly further from the center line. These can be viewed as large angular displacements due to collisions, since ideally a collision results in |ÿ10 ÿ ÿ20 | 0 would result. Here's another data set, which is excellently shown in the lower chart in Figure 4.8. The angles of the two motors are the same most of the time, but the angle deviations occur at around 50s and 70s. The middle diagram shows whether a collision has taken place or not (1 as yes, 0 as no) and the top diagram shows that the currents of the two motors are increasing at these two points in time. The behavior of conguration in C space can be observed off-center according to these two points in C space (see Figure 4.9).

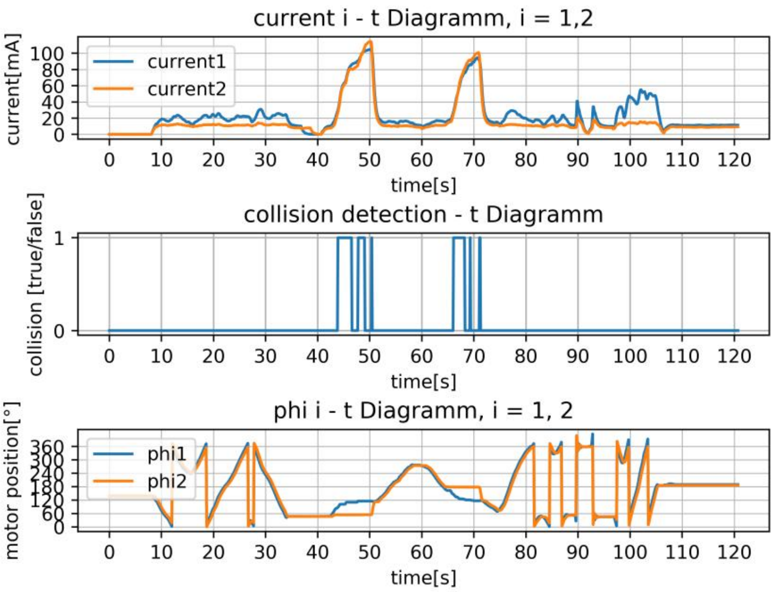


Figure 4.8: Current, collision detection and conguration versus time diagram currenti is the current read by motor i collisions at around 50s and 70s

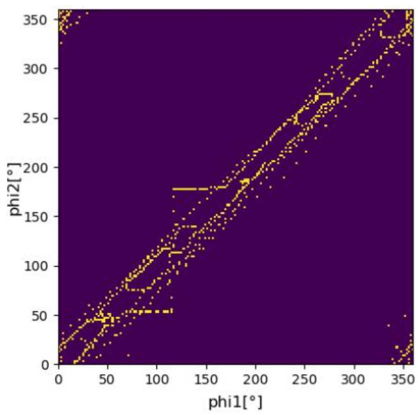


Figure 4.9: Visual representation of the C space with mode 1 PD controller, Kp = 0.7, Kd = 0.25
collision at around (110, 50) and (250, 250)

4.5 Summary and Outlook

The results of mode 1 show that simple haptic feedback or naive current-based collision detection can be implemented. More importantly, the behavior of the robot can be controlled simply by controlling the conguration, ie the position of R in C space. These two methods can be extended to the robot's haptic exploration algorithm in Mode 2.

Machine Translated by Google

Chapter 5

mode 2

This chapter describes in detail the purpose, operation, results and summary of Mode 2.

5.1 Uses of Mode 2

For mode 2, a model-free exploration algorithm with sensorless haptic feedback should be developed. The task in this mode is to enable the robot to explore two 60mm x 60mm 2D mazes (red squares in Figure 5.1) through a five-bar linkage (as shown in Figure 5.1). Maps are created as the environment is explored. Once the two maps are created, they are compared to see if they match and what the difference in rotation angle is. Here, too, it should be noted that the kinematic model of the five-part linkage is unknown to the robot.

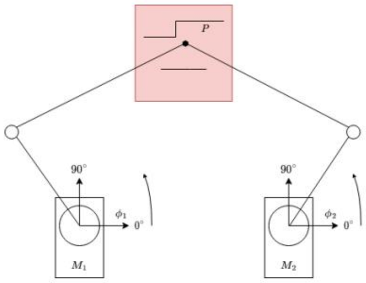


Figure 5.1: Schematic representation of the robot in mode 2

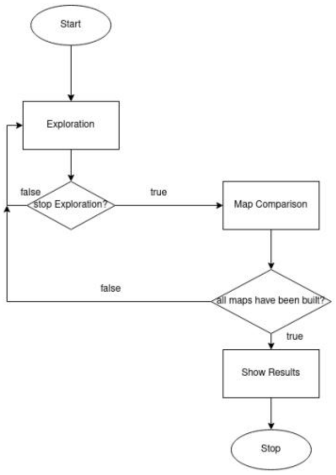


Figure 5.2: Process in mode 2 as a flow slide grams

5.2 Overview of the algorithm

The whole process consists of two parts, exploration and map comparison (see Figure 5.2).

exploration

The exploration is based on a virtual robot controlled by MRC in C Space (see Figure 5.3). The basic principle of the algorithm can be broken down as follows:

1. A two-dimensional C space based on the angles of the two motors he becomes puts.

2. A virtual mobile robot R with two virtual distance sensors in C space is created; the position coordinates of R in C space are dened according to the current angle of the motors.

3. When the robot explores the real world, the current-based collision detection is performed, and in case of a collision, an obstacle is marked in front of the virtual robot in C Space.

4. As R explores the C space through MRC, the horizontal and vertical velocities of R with respect to the C space are calculated using the kinematics of the dierential robot, which correspond to the motor control signals of the real robot.

5. The accessible area in C Space must be limited in advance by M inÿ1 ÿ ÿ1 < M axphi1 and M inÿ2 ÿ ÿ2 < M axÿ2 . The reason for this is the structural properties of the five-bar linkage, where the inverse kinematics can be solved for two solutions given the position of an endector. However, in this work the kinematics are unknown and the multiple solutions increase the complexity of the map and may cause the robot to enter a singularity that prevents it from functioning properly (see Figure 5.4 and compare with Figure 3.5).

Map comparison

After the creation of each map, the obstacles of the current map are extracted during preprocessing using the connected component labeling algorithm[6] and stored in a feature set. After the two maps are created, the saved feature set is used to compare the two maps based on the spatial relationship between the obstacles.

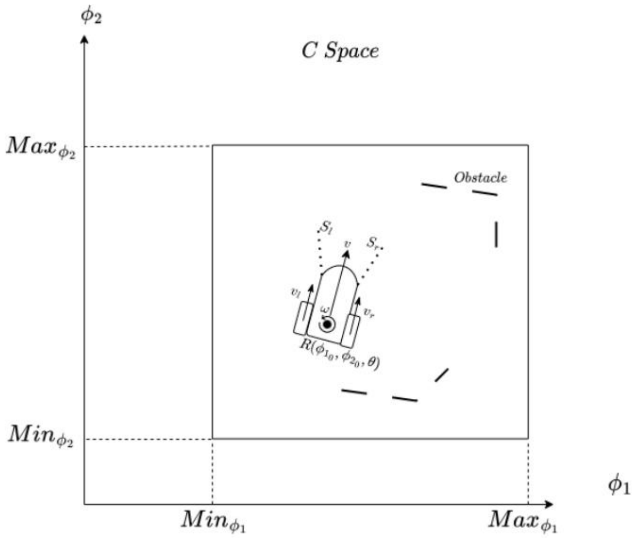


Figure 5.3: Virtual mobile robot R in the constrained C space It should
be mentioned once again that a differential robot R is drawn in the diagram but does not
actually exist. As explained in 3.1.3, this point R represents a conguration and the
parameter ÿ is added to describe the direction of the differential robot.

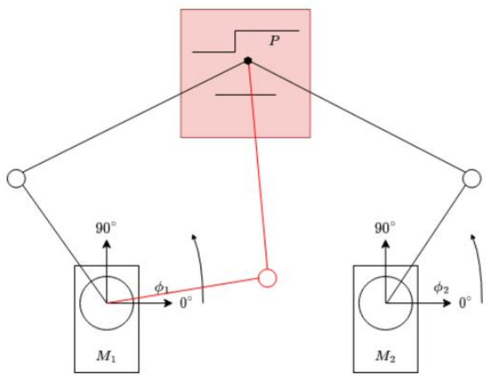


Figure 5.4: The case of multiple solutions of a five-bar linkage

Machine Translated by Google

5.3 Detailed description of the algorithm

An overview of the algorithm is shown in the figure below and the respective functions are shown in detail.

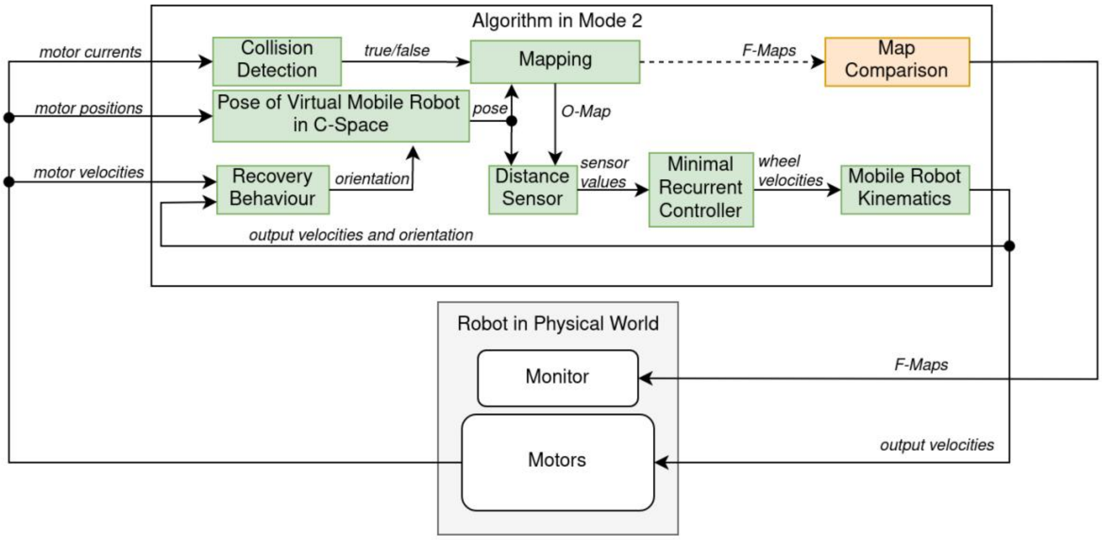


Figure 5.5: Overview of the MRC-based haptic exploration algorithm

5.3.1 Exploration

mapping

In order to get a better understanding of the environment to be explored and to control the behavior of the robot, maps of the environment to be explored are created. Two binary maps, Omap and Fmap, are used here. The position of the obstacle is stored in the first and used to control the behavior of the robot, while in the latter the position reached by the robot is stored and can be used to analyze the environment. The cards are explained in detail as follows:

1. **O-Map: All** obstacle positions marked by collision detection are stored in O Map and provided to the distance sensor to determine the surrounding obstacles (0 is empty, 1 is occupied). As already mentioned, to limit the accessible area in the C space, points on the lines M inÿ1 = ÿ1, M axÿ1 = ÿ1, M inÿ2 = ÿ2 and M axÿ2 = ÿ2 are also marked as obstacles (see Figure 5.3).

2. **F-Map:** All positions visited by R are stored in F Map (0 as unvisited, 1 as visited), these positions are the rotation angles of the motors.

Collision detection **and** obstacle labeling

To detect the occurrence of a collision, as a naive approach, the current can be used as a criterion: ie if the current from motors imotor is greater than the threshold value icrit , it is assumed that a collision has taken place. On collision then a w wide grid obstacle (wall) perpendicular to the Rv is marked in w grid in front of the Rv **(Omap only),** The position of the wall is based on the w grid in front of the current robot position (posx, posy) and the wall is symmetrically marked with w points along ÿ ± pi/2 on the Omap corresponding to the current orientation ÿ. This can be represented as follows:

Machine Translated by Google

if imotor > icrit,

$$xw = posx + w \cos(ÿ)\ yw \quad (5.1)$$

$$= posx + w \sin(ÿ) \quad (5.2)$$

$$xmap = xw + j \cos(ÿ \pm ymap\ \frac{ÿ}{2}),\ ÿ\frac{w}{2} ÿ\ j\ ÿ\ \frac{w}{2} \quad (5.3)$$

$$= yw + j \sin(ÿ \pm \frac{ÿ}{2}),\ ÿ\frac{w}{2} ÿ\ j\ ÿ\ \frac{w}{2} \quad (5.4)$$

$$Omap(xmap, ymap) = 1 \quad (5.5)$$

Where imotor is the current read from motors. icrit is the collision detection threshold.

Distance Sensor

With this function the sensor values of the virtual robot R are calculated. For simplification, the coordinates of the robot are now converted from (ÿ1, ÿ2) to (x, y). Suppose the robot is in a plane coordinate system (x,y), its position is (posx, posy) and its orientation is ÿ. The robot has four sensors scanning at ÿi degrees, i = 1, 2, 3, 4 to the left (1 and 2) and right (3 and 4) sides of the robot, with a maximum scanning distance of dmax . The distance detected by sensors Sl and Sr can be calculated using the following equations:

$$^{i}x_{avail} = posx + dmax \cos(ÿ + ÿi) \quad (5.6)$$

$$^{i}y_{avail} = posy + dmax \sin(ÿ + ÿi) \quad (5.7)$$

$$xsens(k)^{i} = posx(1\ ÿ\ k) + xavail\ k,\ 0\ ÿ\ k\ ÿ\ 1 \quad (5.8)$$

$$ysnes(k)^{i} = posy(1\ ÿ\ k) + yavail\ k,\ 0\ ÿ\ k\ ÿ\ 1 \quad (5.9)$$

for u = $\frac{j}{d\ max}$, 0 ÿ j ÿ dmax

$$you = \begin{cases} \min \sqrt{(posx\ ÿ\ x\ i\ sens(u))2 + (posy\ ÿ\ y\ i\ sens(u))2} & \text{if } Omap(x\ ^{i}sens(u), y\ i\ sens(u)) = 1 \\ max\_dis & \text{otherwise} \end{cases} \quad (5.10)$$

the distance is then mapped between ÿ1 and 1 using the tanh function

$$SL = \tanh(\frac{d\ max}{2\ d} \frac{d1+d2}{2}) \quad (5.11)$$

$$Sr = \tanh(\frac{2\ d}{max\ 2} \frac{2}{d3+d4})2 \quad (5.12)$$

Minimal recurrent controller

MRC is a central part of the haptic exploration algorithm. The input quantities of the controller are the virtual sensor values Sl and Sr from Distance Sensor left and right, and the output unit is the unscaled wheel speed from R left and right. The mathematical representation of this neural network can be expressed by Equations 5.13 and 5.14.

$$Ol = \tanh(wrSr + w1Ol + w2Or) \quad (5.13)$$

$$Or = \tanh(wlSl + w3Or + w4Ol) \quad (5.14)$$

Mobile Robot Kinematics

The kinematics of R are the forward kinematics of a robot with a dual drive. After the unscaled velocities of the wheels of R are given from MRC and adjusted by the scaling factor s, the linear and angular velocities of R in

Machine Translated by Google

be calculated in relation to the C space. Its speed in both axes can be calculated from the trigonometric functions. This is used as a PWM signal to control the servo motors. The overall kinematics look like this:

vl = sOl vr = (5.15)

sOr (vr+vl) (5.16)

$$v = \frac{}{2} \quad (5.17)$$

$$ÿ = \frac{(vrÿvl)}{2B} \quad (5.18)$$

ÿ = ÿtÿ1 + ÿdt vx (5.19)

= vcos(ÿ) vy (5.20)

= vsin(ÿ) (5.21)

v : velocity of the reference point R vr : velocity
of the right wheel vl : velocity of the left
wheel b : distance between the centers
of the two wheels ÿ : angular velocity of the reference point R
ÿ : orientation of the robot in C space

vx : speed of the robot on the horizontal axis in C space vy : speed of the robot
on the vertical axis in C space

Recovery Behaviour

In robotics, recovery behavior refers to a robot's ability to recover from an unpredictable error or loss of control through self-correction, repositioning, restarting, or other means. In many practical applications, in-situ rotation is considered to be a simple recovery behavior for navigators[30, 18]. This is also used here as the recovery behavior of the R. The trigger condition is given when the output speed of the controller is significantly greater than the speed returned by the motor.

5.3.2 Map Comparison

The whole process is divided into two parts: **preprocessing** and **comparing.** The two cards are processed here as images. **Preprocessing** is performed after a map is created. The two maps are then converted into a feature set in **comparisons** and compared to each other.

preprocessing

After creating a map Ki , i = 1, 2 the preprocessing is performed as follows:

- Filtering and background removal
  A threshold filter is applied and the background is removed from the input image. Then the image is downsampled to a lower resolution.

- Image segmentation through Connected Component Labeling (CCL)
  The downsampled image is tagged with CCL, where each pixel is assigned a label based on the connectivity of neighboring pixels. All pixels with the same label are treated as one feature.

Machine Translated by Google

- Creation of the feature set with feature descriptor

  Based on the image marked by CCL, the features are extracted and descriptors D are formed with the following elements: perimeter p, length l, width w, area a, and first and second order Hu moment (Hu1 and Hu2). Then the feature's descriptors are added to the feature set Fi and features with area smaller than a threshold are removed. The descriptors, features, and feature sets can be represented as follows:

$$D_{ij} = \left[p_{ij}, l_{ij}, \ldots Hu1_{ij}, Hu2_{ij}\right]^T, j = 1...n \quad (5.22)$$

$$F_i = \begin{bmatrix} D_{i1} & D_{i2} & \ldots & D_{ij} \\ \bar{x}_{i1} & \bar{x}_{i2} & \ldots & \bar{x}_{ij} \\ \bar{y}_{i1} & \bar{y}_{i2} & \ldots & \bar{y}_{ij} \\ id_{i1} & id_{i2} & \ldots & id_{ij} \end{bmatrix}, j = 1...n \quad (5.23)$$

$$F_i = \left[f_{i1}\ f_{i2} \ldots f_{ij}\right], j = 1...n \quad (5.24)$$

n : maximum for the number of features that can be marked

i : index for i-th card

j : index for j-th feature fij : j-

th feature in i-th map $(\bar{x}_{ij}, \bar{y}_{ij})$:

centroid position of feature fij

idij is the identity of the fij , it is then mentioned in second process **comparisons .**

Compare

After both maps have been created and the feature set has been completed, the following is performed:

- Identity assignment

  Each fij is assigned an identityid based on the Euclidean distance dDeskrip(Dij , Dgh) between it and the descriptor D of all other features fgh . That is, for fij one nds an fgh, h=j, with minimum dDeskrip, and if dDeskrip is less than a threshold one assigns fij and fgh the same id, otherwise a new id.

$$dDeskrip(f_{ij}, f_{gh}) = \sqrt{(p_{ij} - p_{gh})^2 + \ldots (Hu2_{ij} - Hu2_{gh})^2}, g = 1, 2, h = 1..n, h = j \quad (5.25)$$

- Determination of the angular deviation through the sum of the Euclidean distances

  In F1 all f1j in position $\bar{x}_{1j}, \bar{y}_{1j}$ are rotated stepwise by 360 degrees(2π). The minimum Euclidean distance between f1j and the "nearest" feature f2j in f2 with "the same identity" (id1j = id2j) is calculated. These distances from all f1j are summed. The angle with the smallest sum is taken out by rotation R to determine the most probable angular deviation between the two images.

  The process can be represented as follows: for θ ∈ [0, 2π]:

$$\left[\bar{x}'_{1j}, \bar{y}'_{1j}\right]^T = R(\theta)\left[\bar{x}_{1j}, \bar{y}_{1j}\right] \quad (5.26)$$

$$fdis(\bar{x}_{1j}, \bar{y}_{1j}) = \min\left(\sqrt{(\bar{x}'_{1y} - \bar{x}_{2h})^2 + (\bar{y}'_{1y} - \bar{y}_{2h})^2}\right), h = 1...n \quad (5.27)$$

Machine Translated by Google

$$\sum_j f_{dis}(\bar{x}_{1j}, \bar{y}_{1j}) \tag{5.28}$$

The most probable angle deviation ÿmin is the angle that equation 5.28 is minimum.

- MAPE-based differentiation of maps by feature locations and areas

  The MAPE (Mean Absolute Percentage Error) calculated based on the positions and areas of the fij in the respective fi is used to determine the differences between the maps. The MAPE can be derived as follows:

$$ed1 = \sum_j \sqrt{x_{1j}^2 + y_{1j}^2} \tag{5.29}$$

$$ed2 = \sum_j \sqrt{x_{2j}^2 + y_{2j}^2} \tag{5.30}$$

$$MAPE(ed1, ed2) = \frac{|ed1 - ed2|}{\frac{ed1+ed2}{2}} \tag{5.31}$$

$$A1 = \sum_j D_{1j}[4] = \sum_j a_{1j} \tag{5.32}$$

$$A2 = \sum_j D_{2j}[4] = \sum_j a_{2j} \tag{5.33}$$

$$MAPE(A1, A2) = \frac{|A1 - A2|}{\frac{A1+A2}{2}} \tag{5.34}$$

5.4 Results

In this section, the maps generated with the exploration algorithm are shown, again in two parts, **exploration** and **map comparison.** It should be noted that the time required to create each complete map is 15 minutes and the accessible area in C Space is dened as follows:

$$Min\theta_1 = 73[°] \tag{5.35}$$

$$Max\theta_1 = 184[°] \tag{5.36}$$

$$Min\theta_2 = 106[°] \tag{5.37}$$

$$Max\theta_2 = -5(355)[°] \tag{5.38}$$

$$\tag{5.39}$$

5.4.1 Exploration

Figures 5.6 and 5.7 show the two mazes in plan view, the Omap (left) and Fmap (right) as examples of results. The map in both figures is the same maze, but one is rotated. Looking at the real maze and fmap, one can see that the basic features (the purple zone in the yellow area) of the environment are well developed, although the fmap created after exploration is somewhat deformed

is.

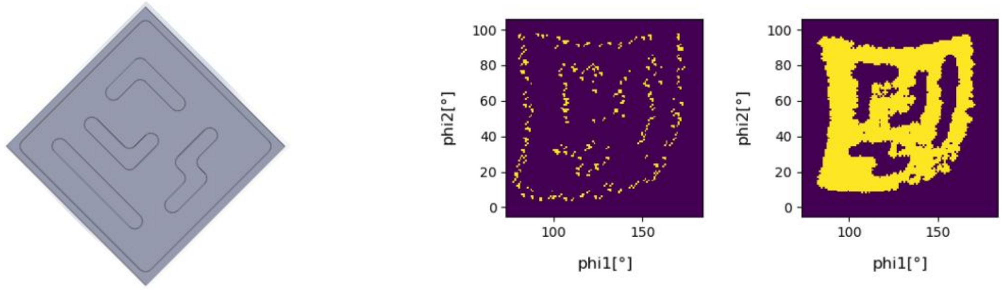


Figure 5.6: Exploration results

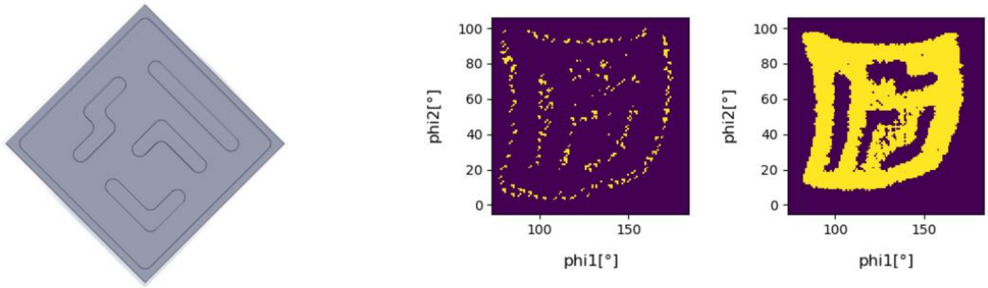


Figure 5.7: Exploration results

Figure between Work Space **frame and** C Space **frame**

It can be clearly seen in Figures 5.6 and 5.7 that the positions of the obstacles/features in the real mazes and in map Fmap do not match exactly, and Fmap is subject to additional rotations and so on. The reason for this is shown in Figure 5.8, where the movement of R along the axis Cÿ1 and Cÿ2 lead to corresponding movements of the endector P in directions Wÿ1 and Wÿ2 , causing: As shown on the left side of Figure 5.8, a movement of Cÿ1 to a by the rotation of the engines right rotation of M1 and Cÿ2 up to a left rotation of M2. By considering the relationship between Cÿi and Wÿi , i = 1, 2, as shown in the bottom right-hand side of Figure 5.8, it can be concluded that Wÿi is obtained by horizontal reflection along to the left, a subsequent rotation of about 45 degrees and a slight stretching is mapped into Cÿi . This mapping relationship becomes clearer when looking at Figures 5.6 and 5.7.

5.4.2 Map Comparison

Shown here are three sets of cards after downsampling and CCL, and the results of the comparison. A brief overview of the three datasets follows:

1. Group 1: The labyrinth to be explored consists of two identical labyrinths, but with different angles. The descriptors used are area, perimeter, length and width.

2. Group 2: The labyrinth to be explored consists of two identical labyrinths, but with different angles. The descriptors used are area, perimeter, length, width, and first and second order Hu moment.

3. Group 3: The labyrinths to be examined are two different labyrinths with angular deviations. The descriptors used are area, perimeter, length, width, and first and second order Hu moment.

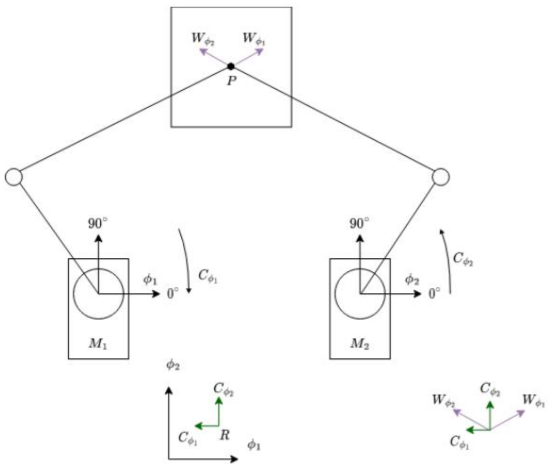


Figure 5.8: Schematic representation of the relationship between the robot's Work Space and C Space

Each group will present the data in the following order:

1. Top view of the 3D model of the environment.
2. The downsampled and CCL marked cards.
3. Tabular representation of the corresponding feature sets F1 and F1.
4. Normalized and non-normalized sum of Euclidean distances with different angles of rotation.
5. MAP E(A1, A2) and AP E(A1, A2) are presented in the form of a table.

Group 1

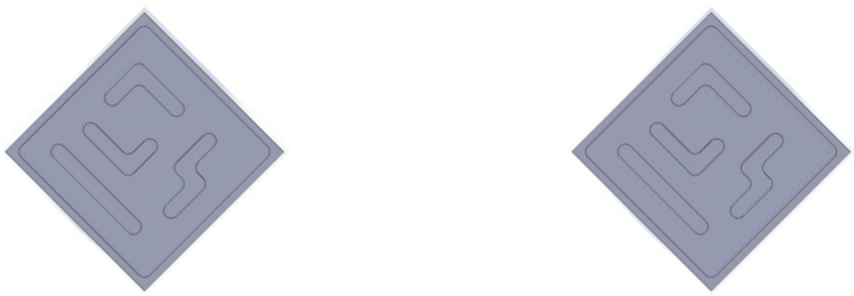


Figure 5.9: The real world mazes in Group 1

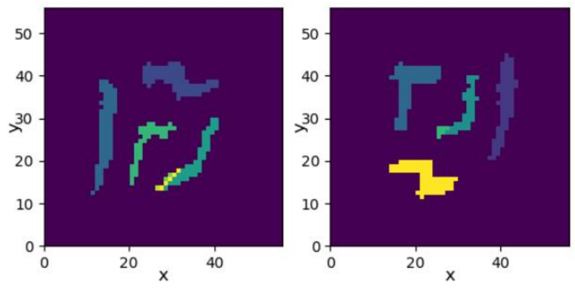


Figure 5.10: The scanned and marked cards in group 1 (left card 1 K1, right card 2 K2)

Angular deviation between two maps is ÿ180[ÿ](180[ÿ])

	Feature 1	Feature 2	Feature 3	Feature 4
perimeter 0.4444444 1.0000000 0.5000000 0.1111111 length 0.2294403 1.0000000 0.4905069 0.2329356 width 0.5096185 0.0000000 0.0971618 0.47898 09 area 0.9722222 0.9722222 0.2222222 0.0833333				
position x	30	13	35	23
position y id	38	26	20	23
	0	1	2	3

Table 5.1: Feature Set of Map 1 in Group 1

	Feature 1	Feature 2	Feature 3	Feature 4 perimeter
0.9444444 0.6666667 0.0000000 0.3333333 length 0.8510334 0.2089231 0.0875532 0.0000000 width 0.0293110 1.0000000 0.163 5907 0.5756669 area 0.8611111 1.0000000 0.0000000 0.6944444				
position x	40	18	31	21
position y id	32	36	31	15
	1	4	3	0

Table 5.2: Feature Set of Map 2 in Group 1

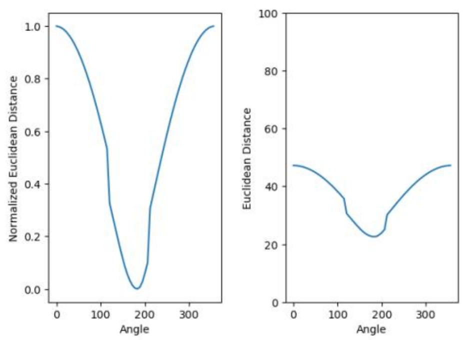


Figure 5.11: The normalized and non-normalized sum of the minimum Euclidean distance over rotation from 0[ÿ] to 360[ÿ] ÿmin is 183[ÿ]

group 2

Figure 5.12: The real world mazes in Group 2

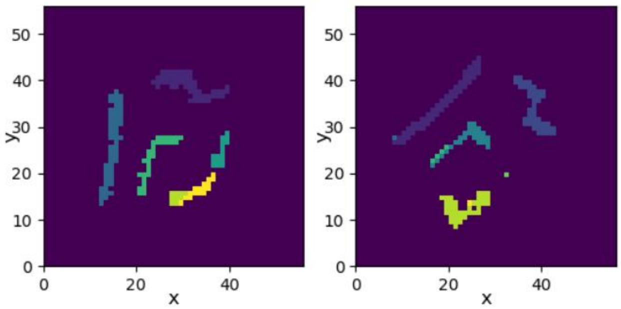


Figure 5.13: The scanned and marked cards in group 2 (left card 1 K1, right card 2 K2)

Angular deviation between two maps is ÿ45[ÿ](315[ÿ])

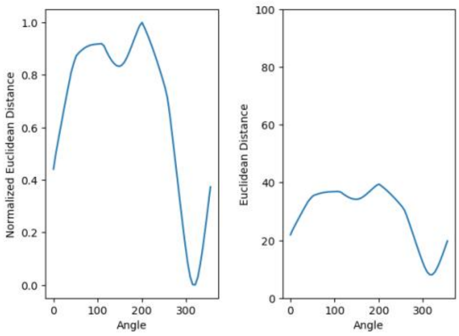


Figure 5.14: The normalized and non-normalized sum of the minimum Euclidean distance over rotation from 0[ÿ] to 360[ÿ] ÿmin is 320[ÿ]

	feature 1	features 2	feature 3	feature 4
Scope 0.5151515 0.7575758 0.4545455				0.0909091
Length 0.3083044 0.8745526 0.3959768 Width 0.8514799				0.1864161
0.0422882 0.7144658				0.0411113
Surface 0.9677419 0.9032258 0.4516129 0.0000000				
Hu2 0.0880229 0.8638422 0.4621422 Hu2 0.0702350				0.4460578
0.8140637 0.2897522				0.3402381
Position x 30.0000000 14.0000000 23.0000000 36.0000000				
Position y 37.0000000 26.0000000 22.0000000 21.0000000				
i.e	0	1	0	2

Table 5.3: Feature Set of Map 1 in Group 2

	feature 1	features 2	feature 3	feature 4
Circumference 1.0000000 0.2727273 0.0000000 Length				0.1515152
1.0000000 0.2340754 0.1243146 Width				0.0000000
0.0000000 0.7913867 0.8649035				1,0000000
Area 1.0000000 0.5806452 0.0322581				0.3548387
Hu1 1.0000000 0.1553764 0.3893801 Hu2 1.0000000				0.0000000
0.0919608 0.1584698				0.0000000
position x 18.0000000 38.0000000 23.0000000 22.0000000				
Position y 34.0000000 33.0000000 26.0000000 11.0000000 0				
i.e	1	0	0	

Table 5.4: Feature set of Map 2 in Group 2

group 3

Figure 5.15: The real world mazes in Group 3

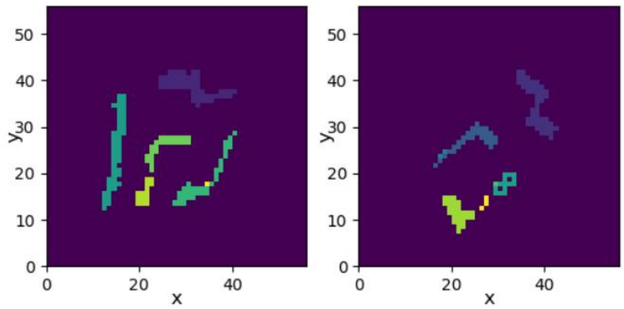


Figure 5.16: The scanned and marked cards in group 3 (left card 1 K1, right card 2 K2)

The angular deviation between two cards is ÿ45[ÿ](315[ÿ])

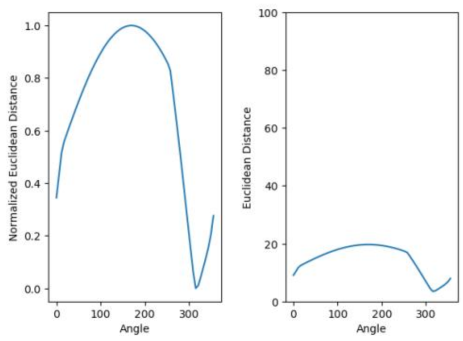


Figure 5.17: The normalized and non-normalized sum of the minimum Euclidean distance over rotation from 0[ÿ] to 360[ÿ] ÿmin is 315[ÿ]

	Feature 1	features 2	feature 3	feature 4
Perimeter	0.6969697	1.0000000	0.5151515	0.2727273
Length	0.4278322	1.0000000	0.7303029	0.2305210
Width	1.0000000	0.0000000	0.7327873	0.6782928
Area	1.0000000	0.8666667	0.3666667	0.0000000
Hu1	0.1903882	1.0000000	0.9588710	0.3450914
Hu2	0.0750356	1.0000000	0.8008420	0.1311741
position x	30.0000000	14.0000000	33.0000000	24.0000000
Position y	38.0000000	24.0000000	18.0000000	25.0000000
i.e	0	1	2	3

Table 5.5: Feature Set of Map 1 in Group 3

	feature 1	features 2	feature 3
Circumference	0.5151515	0.3030303	0.0000000
Length	0.4587603	0.3456033	0.0000000
Width	0.9034140	0.9097295	0.6443887
Surface	0.8333333	0.2000000	0.0666667
Hu1	0.2666084	0.4120514	0.0000000
Hu2	0.1157352	0.1681231	0.0000000
location x	37.0000000	24.0000000	20.0000000
position y	34.0000000	26.0000000	10.0000000
i.e	0	3	3

Table 5.6: Feature set of map 2 in group 3

In group 1, Hu moments were not used as a descriptor, but the results show that the angular deviation determination is correct. However, it was found through experiment that at an angular deviation of 45[ÿ], 135[ÿ], 225[ÿ], or 315[ÿ] between the two mazes, the image exhibited the greatest distortion, leading to the feature matching failed, so the Hu moments were added to Group 2. While Hu1 and Hu2 worked, the error increased when higher order Hu moments were applied.

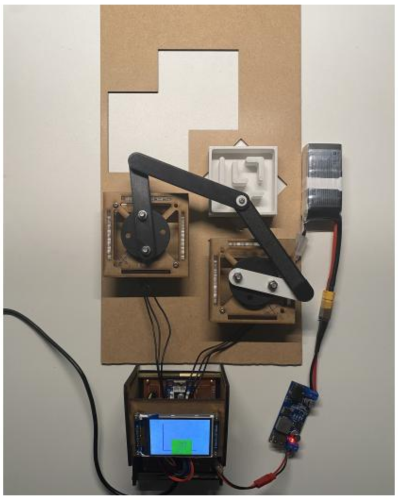
In groups 2 and 3 it can be seen that in group 2 the id assignments in the table do not exactly match, despite the fact that the cards are the same, but as can be seen from 5.14 in the figure, this has no significant impact on the calculation of the angle deviation. For the same angular deviation but for different maps (see Group 2 and Group 3), the standard deviations of the curves in Group 2 differ in the same order of magnitude as in Group 3 (see Table 5.7). In addition, it can also be seen from Table 5.7 that the MAPE of Group 1 and Group 2 differs from Group 3. Therefore, the standard deviation and MAPE can be used to approximate the similarity and angular deviation.

	Group 1	Group 2	Group 3
MAP E(ed1, ed2)	0.32%	0.17%	0.02%
MAP E(A1, A2)	0.68%	0.13%	0.17%
standard deviation	0.6825883932324971	0.7715407073995874	0.41237692495464046

Table 5.7: Resulting MAPE and standard deviation

5.5 Another variant

The robot can also be equipped with other mechanisms. As long as the set accessible area is adjusted accordingly, the robot does not need a kinematic model to support the exploration of the environment. Of course, it should be ensured that singularities do not occur if possible, which can be set using Figure 5.18. The image below shows an example of a robot adapted to another body and the map it created in C Space.



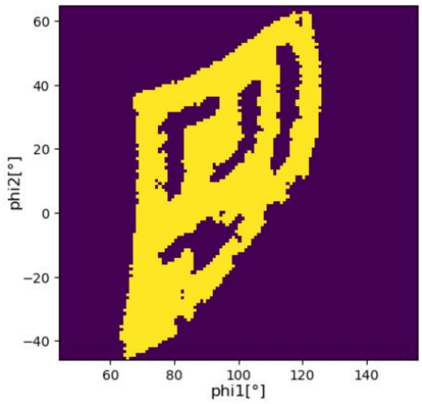


Figure 5.18: One of the possible variants of the robot in mode 2 and the corresponding C Space

5.6 Summary and Outlook

In mode 2, a haptic exploration algorithm is implemented based on the two conditions of the absence of the sensors and the kinematic model of the robot mechanism. In this algorithm, naive current-based collision detection is used to mark the position of obstacles and MRC is used to control the robot actions and the angle returned by the servomotor to position the robot in C space. This approach demonstrates the effectiveness of this exploration algorithm in an unknown environment of 60mm × 60mm and proves that the method is able to explore and model the environment with different positions and unknown kinematic model of the robot after a small adjustment.

In addition, it is found that simple descriptors can be used to distinguish between different obstacles and thus to further distinguish between maps. However, the ability of the method to adapt to different maps generated by different mechanisms of the robot is not yet fully understood and needs further research and validation.

Overall, the haptic exploration approach used in this research, which is based on no sensors and no model of the robot, offers an alternative solution for exploration and modeling in unfamiliar environments.

Since the exploration algorithm is based on the MRC, which is a "reactive" approach, it does not predict the behavior of the robot well, it can cause the robot to regets explored and wastes time, which is relatively inefficient. (Exploration time in space mm60×60mm is 15 minutes)

If the “model-based” approach using the trajectory planning method in navigator robots [24] can be implemented in the control of a virtual robot in mode 2, this can allow the robot to explore its behavior effi ciently. The feasibility of this approach can also be examined.

As for the accessible area in the C Space, if the area can be expanded to an unlimited one, the robot would have a wider range of applications, or if the robot can explore the environment without adjustment with different mechanism. However, avoiding singularities under such conditions is a major challenge. Furthermore, it is also an open question how this approach can be extended to robots with higher degrees of freedom.

When comparing cards, the id assignment can be seen as a key element. During development, choosing a good descriptor has a big impact on the accuracy of the id assignment. In addition, the complexity (e.g. more obstacles) of the environment can also affect the accuracy of the id assignment. This problem can be overcome by choosing a more accurate feature descriptor and implementing it in a more expensive controller.

bibliography

[1] Avr221: Discrete pid controller on tinyavr and megaavr devices. 2016

[2] Stefan Bethge. Detection of robot limb collisions based on motor and Sensor data, 2011. Student work.

[3] Valentino Braitenberg. Vehicles: Experiments in Synthetic Psychology. MIT Press, Cambridge, MA, 1984.

[4] Daniel Butters, Emil T. Jonasson, Robert Stuart-Smith, and Vijay M. Pawar. Ecient environmental guided approach for exploration of complex environments. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 929-934, 2019.

[5] Lin Cheng, Jiayu Liu, Juyi Li, and Jianzhong Tang. Sensorless collision detection for a 6-dof robot manipulator.

[6] Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, and Yuyan Chao. The connected component labeling problem: A review of state-of-the-art algorithms. Pattern Recognition, 70, 04 2017.

[7] Tuong Hoang, Trung Vuong, and Bang Pham. Study and development of parallel robots based on 5-bar linkage. 10 2015.

[8] Zhihu Huang and J. Leng. Analysis of hu's moment invariants on image scaling and rotation. volume 7, pages V7-476, 05 2010.

[9] Austin Hughes. Electric Motors and Drives. Elsevier Ltd., USA, 3rd edition, 2006.

[10] Yoonkyu Hwang, Yuki Minami, and Masato Ishikawa. Virtual torque sensor for low-cost rc servo motors based on dynamic system identification utilizing parametric constraints. Sensors, 18(11), 2018.

[11] Martin Hulse and Frank Pasemann. Dynamic neural schmitt trigger for robot control. pages 783-788, 08 2002.

[12] M ficrete. Kanniah, Jagannathan. Ercan and Acosta Calderon, Carlos A. Practical Robot Design: Game Playing Robots. CRC Press, 1st edition, 2013.

[13] Yasuhiro Kato, Pietro Balatti, Juan Gandarias, Mattia Leonori, Toshiaki Tsuji, and Arash Ajoudani. A self-tuning impedance-based interaction planner for robotic haptic exploration. IEEE Robotics and Automation Letters, 7:1-8, 10 2022.

[14] John Killian. Simple image analysis by moments, 2001.

[15] Guillermo Jesus Laguna Mosqueda. Exploration of an unknown environment with a die rental drive disc robot. Master's thesis, Computer Science Department, CIMAT, Guanajua to, Mexico, 10 2013.

[16] David Charles Lee. The map-building and exploration strategies of a simple, sonar-equipped, mobile robot: An experimental, quantitative evaluation. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 23-30, Minneapolis, MN, USA, April 1996. IEEE.

[17] Kevin M Lynch and Frank C Park. Modern Robotics: Mechanics, Planning, and Control. Cambridge University Press, USA, 1st edition, 2017.

[18] Eitan Marder-Eppstein. rotate_recovery. Available on-line: **http://wiki.ros.org/rotate_recovery,** 2019. Accessed 6 March 2023.

[19] Frank Pasemann, Martin Hulse, and Keyan Ghazi-Zahedi. Evolved neurodynamics for robots control. pages 439-444, 01 2003.

[20] Gavin Paul, Stephen Webb, Dikai Liu, and Gamini Dissanayake. Autonomous robot manipulator-based exploration and mapping system for bridge maintenance. Robotics and Autonomous Systems, 59:543-554, 07 2011.

[21] ROBOTIS. Opencm 9.04 controller, nd

[22] Leonardo Romero, Eduardo F. Morales, and L. Enrique Sucar. Exploration and navigation for mobile robots with perceptual limitations. International Journal of Advanced Robotic Systems, 3(3):36, 2006.

[23] Tim Schneider, Boris Belousov, Georgia Chalvatzaki, Diego Romeres, Devesh K. Jha, and Jan Peters. Active exploration for robotic manipulation, 2022.

[24] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. Introduction to Autonomous Mobile robots. The MIT Press, 2nd edition, 2011.

[25] Lila Torabi, Muslim Kazemi, and Kamal Gupta. Conguration space based efficient view planning and exploration with occupancy grids. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2827-2832, 2007.

[26] Yiming Wang, Stuart James, Ellie Stathopoulou, Carlos Beltran-Gonzalez, Yoshinori Konishi, and Alessio Del Bue. Autonomous 3-d reconstruction, mapping, and exploration of in-door environments with a robotic rod. IEEE Robotics and Automation Letters, PP:1-1, 07 2019.

[27] Michael A Wirth. Shape analysis and measurement. 2004

[28] B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146), volume 4, pages 3715-3720 vol.4, 1998.

[29] Bowei Zhang and Pengcheng Liu. Model-based and model-free robot control: A review. In Proceedings of the 8th International Conference on Robot Intelligence Technology and Applications, Cardi, UK, December 2020. IEEE.

[30] Kaiyu Zheng. ROS navigation tuning guide. CoRR, abs/1706.09068, 2017.

Appendix A

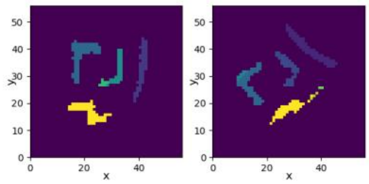
Attachment

A.1 Additional Results of Mode2

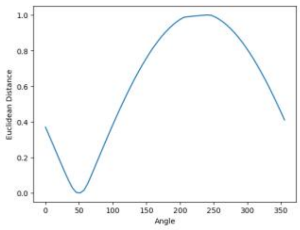
A.1.1 Examples of other map comparisons

example 1

Top view of the mazes



The processed and marked maps with 45 degree angular deviation



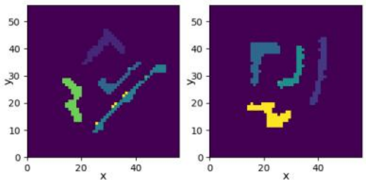
The normalized minimum sum of Euclidean distance over rotation from 0[ÿ] to 360[ÿ] .
ÿmin is at 51[ÿ]

MAP E(ed1, ed2)	0.04%
MAP E(A1, A2)	0.05%

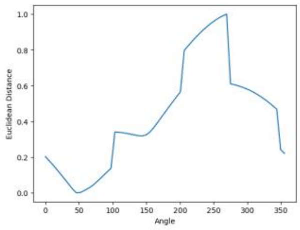
Resulting MAPE

example 2

Top view of the mazes



The processed and marked maps with 45 degree angular deviation



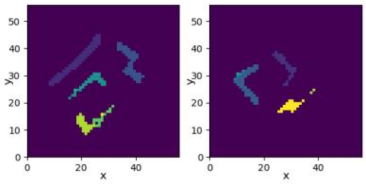
The normalized minimum sum of Euclidean distance over rotation from 0[ÿ] to 360[ÿ] . ÿmin is 46[ÿ]

MAP E(ed1, ed2) 0.01	% MAP
E(A1, A2) 0.37%	

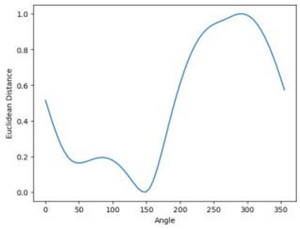
Resulting MAPE

Example 3

Top view of the mazes



The processed and marked maps with -90 degree angular deviation

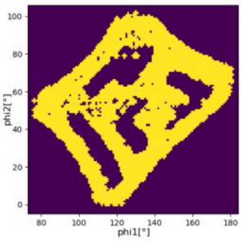
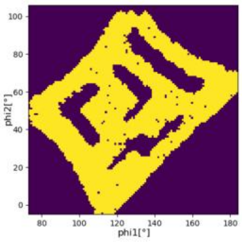
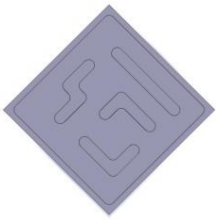
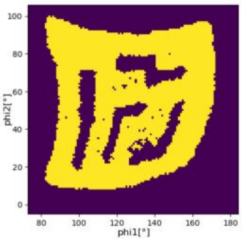
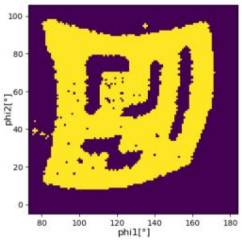
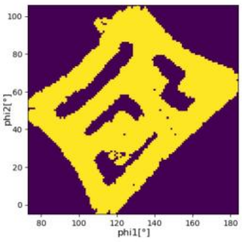
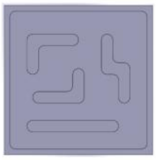
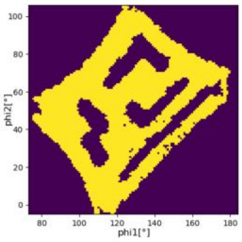
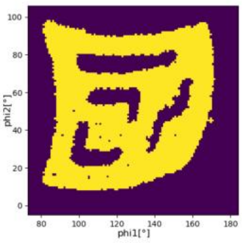
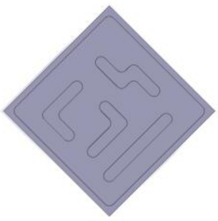
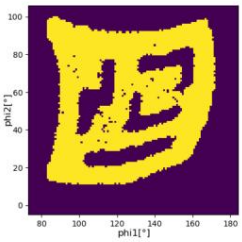


The normalized minimum sum of Euclidean distance over rotation from 0[ÿ] to 360[ÿ] . ÿmin is 149[ÿ]

MAP E(ed1, ed2) 0.32% MAP	
E(A1, A2) 0.95%	

Resulting MAPE

A.1.2 All maps created in 45 degree increments

Oberansicht des Labyrinths	Karte in *C Space*	Oberansicht des Labyrinths	Karte in *C Space*
			
			
			
			

B.2 Source Code for Mode 1

Denition of Constants

```
#define M_PI 3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8     // Constantpi // S
4 6 # define DXL_SERIAL S erial 1 #                      erialportto communica te wi th the Dynamixel mo tor controller // S erialporttooutput debug me s
define DEBUG_SERIAL S erial # define                      sa ge s .
DXL_DIR_PIN 28 # define                                  // D irection pi nforthe mo tor controller // Pi nforthe LED on
LED_PIN 14 # defineperiod                                themicrocontrollerb oa rd .
1 1 0 0 0 0 0 0 # defineperiod 2                         // P eriodofthefirsttaski n mi croseconds / / P eriodofthesecondtaski
40000 #define I nput #define I                          n mi croseconds / / A ddressoftheinputvolta geneinthe Dynamixel
nput #define _voltage _add 144                           protocol / / S izeoftheinputvoltagei nbytesinthe Dynamixel protocol / / Timeou tforc
Timeou t 10 #define_voltage _size 2                      ommunica ti on wi th the Dynamixel mo tor controller / / Number ofmotorsb ei ngcontrolled .
Motor_Num 2 #define
Max_ang_v ( 5 0 . 0 F )
#define Min_ang_v ( ÿ50.0 F )                             // Maximum angular velocityconstr aintforthemotors .
                                                         // Minimum angular velocityconstr aintforthemotors .

#define Kp1 ( 0 . 7 F ) #                                // P roproportional g ai nfor motor 1
define Ki1 ( 0 . 0 F ) #                                 // Integral ai info engine 1
define Kd1 ( 0 .      2 5 F )                            // D erivativeg ai nfor engine 1

#define Kp2(0.7F) #define                                // P roportionalg ai nfor motor 2
Ki2(0.0F) #define Kd2(0.                                  // Integral ai info engine 2
                      2 5 F )                            // D erivativeg ai nfor engine 2
```

loop 1

```
// Check ifitif ' s tim eto run the first task
( current M icros ÿ previous sM icros 1 >= period 1 ) { voltage = 0 ;
       uint 1 6 _input _
       // Read theinputvoltageof Motor 1 and storeitinvoltage 1 dxlread ( Motor_1_ID ,
            · Input _Voltage , ( uint 8 _add , _voltage _size _t ÿ)& input _voltage , sizeof( input _voltage ) , timeout );
       voltage 1 = input _Inputvoltage ;
       // Read theinputvoltageof Motor 2 and storeitinvoltage 2 inputdxl
               _voltage = 0 ; read
            · ( Motor_2_ID , voltage I nput _V _add , input _voltage _size , (uint 8 _t ÿ)& input _voltage , sizeof( input _voltage ) , timeout );
       2 = input _oltagevoltage ;
       // Upda te thepreviou sM icros 1 variableto keep trackof when to run thistaskag ai npreviou sM icros 1 += period 1 ;

}
```

Machine Translated by Google

loop 2

```
,
// Check ifits tim eto run thesecondtaskif ( current M icros ÿ previou sM icros 2
>= period 2 ) {
    // Upda te thepreviou sM icros 2 variablefor timing purposespreviou sM icros 2 += period 2 ;

                                                                                                   ,
    // C all the visual visual    _ robotfunctiontoclearthepreviousvisualizati onofr ob ot            shea di ng
          _ robot ( head ding ,     1 ) ;

    / ÿ Ge tessentialdata ÿ / s / / Ge
    tpresentleftpresentve locity and rightpresentvelocityf loat ang1_v = dxl . get Present Velocity ( Motor_1_ID ,
    UNIT_RPM ) ; float ang2_v = dxl . get Present Velocity ( Motor_2_ID , UNIT_RPM ) ;

    // Get tpresentleftpresentposition and rightpresentpositionfloatpresent_positionfloatpresent_position
                              _ 1 = dxl     . get Present P osition ( Motor_1_ID , UNIT_DEGREE ) ; . get Present P osition
                              _ 2 = dxl     ( Motor_2_ID , UNIT_DEGREE ) ;

    // Get tpresentfloatcurrent   leftpresentcurrent and rightpresentcurrent 1 = ab s ( dxl . get Present Current
                     _ ( Motor_1_ID , UNIT_MILLI_AMPERE ) ) ; 2 = ab s ( dxl . get Present Current ( Motor_2_ID ,
    float current    _ UNIT_MILLI_AMPERE ) ) ;

    / / Apply to IIR filtertothepresentcur rentreadingsf 1 + alpha 1 ÿ f 2 + alpha 1 ÿ f
     _ current    _ 1 = ( 1 ÿ alpha 1 ) 2 = ( 1 ÿ    ÿ current    _            _ current    _ 1 ;
    f _ current   _ alpha 1 )                       ÿ current    _            _ current    _ 2 ;

    / / Apply an IIR filtertothepresentvel ocityreadings f_ ang 1_v = ( 1 ÿ al pha 2 ) f_ ang 2_v = ( 1 ÿ al
    pha 2 )                        ÿ ang1_v + alpha 2 ÿ f_ ang 1_v ; ÿ ang2_v + alpha
                                   2 ÿ f_ ang 2_v ;

    // C o nve rtthepresentpositioner eadings from degreestointegersint 1 6
         _ tphi 1 = round ( fmod ( present _ positiontphi 2 = round ( fmod   _ 1 + 1 3 5    ,   3 6 0 . 0 ) );
    int 1 6  _ ( present _ positionphi 1 = ( phi 1 < 0) ? phi 1 + 360 : phi 1 ; phi   _ 2 + 4 5    ,   3 6 0 . 0 ) );
    2 = ( phi 2 < 0) ?phi 2 + 360 : phi 2 ;

    / ÿ Mo tion Plan ni ng ÿ / p hi 1 =
    ( p hi 2 > 270 && p hi 1 < 9 0) ? phi 1 + 360 : phi 1 ; phi 2 = ( phi 1 > 270 && phi 2 < 9 0) ? phi 2    // adjustp hi 1 // adjustp   if needed to make the motion sm oo the r
    + 360 : phi 2 ; 1 = ( phi 2 ÿ phi 1 ) ÿ 0 . 5 ; // calculateerrorforp hi 1 2 = ( p hi 1 ÿ p hi 2 ) ÿ 0 . 5 ; //    hi 2 if needed to make the motion sm oo the r
    do uble error          _ calculateerrorforp hi 2
    do uble error          _

    // PID C ontrollerforp hi 1 do ubl ep _
    controldo ubl ed            _ 1 = Kp1 ÿ error          _ 1 ; // calculate proportional control forp hi 1 1 ); // calculatederivativeco ntrolforp
                  _ control     _ 1 = Kd1 ÿ ( errorcontrol    _ 1 ÿ previous        _ error    _ hi 1 // calculateintegralcont rolfor phi1 ,
    // i   _ control    _ 1 = i    _         _ 1 + Ki1 ÿ error     _ 1 ;                                    currentlynotu se d / /
    do ubl e d_ p hi1 = p _ controlprevious    _ 1 + d    _ control    _ 1 + i //   _ control    _ 1 ;   calculatefinalcontrol signalforp hi 1
                  _ error    _ 1 = error         _ 1 ;    storepreviouserrorfor p hi 1

    // PID C ontrollerforp hi 2 do ubl ep _
    controldo ubl ed            _ 2 = Kp2 ÿ error          _ 2 ; // calculate proportional control for p hi 2 2 ); // calculatederivativeco ntrolforp
                  _ control     _ 2 = Kd2 ÿ ( error           _ 2 ÿ previous        _ error    _ hi 2
    // i   _ control    _ 2 = i    _ control    _ 2 + Ki2 ÿ error // calculate 2 integral control for phi2 ,       currentlynotu se d / /
    do ubl e d_ p hi2 = p _ controlprevious / ÿ    _ 2 + d    _ control    _ 2 + i    _ control    _ 2 ;   calculatefinalcontrol signalforp hi 2
    Mo tion Pl an  _ error    _ 2 = error         _ 2 ; // store previous error for p hi 2
    ni ng ÿ /

    / ÿ Motor C ontr ol ÿ / d_ p hi1                                                                                          ,
    = ( d_ p hi1 > Min_ang_v ) ? ( ( d_ p hi1 < Max_ang_v ) ? d_ p hi 1 : Max_ang_v ) d_ p hi2 = ( d_ p hi2 > Min_ang_v ) ? ( ( d_ p hi2 <   : Min_ang_v ; // Clamp motor 1   , velocity velocity
    Max_ang_v ) ? d_ p hi 2 : Max_ang_v ) dxl . setGoalPWM ( Motor_1_ID , d_ p hi 1 ÿ ( 5 0 . 0 / voltage 1 )   : Min_ang_v ;   // Clamp motor 2
                                                                       , UNIT_PERCENT ); dxl .   // Set the goal PWM for motor 1
    setGoalPWM(Motor_2_ID, d_phi2*(50.0/voltage2), UNIT_PERCENT); / ÿ Motor C ontr ol ÿ /       // Set the goal PWM for motor 2

    / ÿ V isualization and C ollision D etection ÿ / uint 1 6
             _ t phi1_w = fmod ( phi1 , t phi2_w =    3 6 0) / 2 ; // convertp hi 1 toavalue be tween 0 and 180 for visualization 3 6 0) / 2 ; // convertp hi 2 toavalue be tween 0 and 180
    uint 1 6 _ fmod ( phi2 ,                            for visualization

    // Current based collision detection 1 > 25 && f
    if ( ( f   _ current   _            _ current    _ 2 > 2 5) && from s ( f    _ current    _ 1 ÿ f   _ current    _ 2 ) < 5) { set
          // I fthereisacollisioncollision = 1 ;        ,   the collision flag to 1 and turn on the LED
```

Machine Translated by Google


```
digital W rite Fast ( LED_PIN , LOW) ; // Turn on the LED
// Visualizeobstacle on screen
P ai n t_ D r aw P oi nt ( collisiondo ubl _ x , collision _ y , ( ( 4 5 WHITE , DOT_PIXEL_2X2 , DOT_FILL_AROUND ) ; 1 8 0 . 0 ); cos
eangle = M_PI ÿ collision 0 ÿ p hi 1 ) / x = 120 + ( angle ) ;
_ ( visual _ bar + 5) ÿ bar + sin ( to gl e ) ;
collision _ y = 280 + ( visual _ 5) ÿ
P ai n t_ Dr aw P oi nt ( collision _ x , collision _ y , BLACK , DOT_PIXEL_2X2 , DOT_FILL_AROUND ) ;

} else {
// If there is no collisioncollision = 0 ; digital W , set the collision flag to 0 and turn off the LED
rite Fast ( LED_PIN ,
HIGH ) ; // Turn off the LED

} / ÿ V isualization and C ollision D etection ÿ /
}
```

C.3 Source Code for Mode 2

Denition of Constants

```
#define M_PI 3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 6          // Constantpi // S
#define DXL_SERIAL S erial 1 #define                               erialportto communica te wi th the Dynamixel mo tor controller // S erialporttooutput debug me s sa ge
DEBUG_SERIAL S erial #define                                       s .
DXL_DIR_PIN 28 #define LED_PIN                                     // D irection pi nforthe mo tor controller // Pi nforthe LED on
14 #defineperiod 1 1 0 0 0 0                                       themicrocontrollerb oa rd .
0 0 0 #defineperiod 2 40000 #define I                               // P eriodofthefirsttaski n mi croseconds / / P eriodofthesecondtaski
nput #define I nput #define                                         n mi croseconds / / A ddressoftheinputvolta geneinthe Dynamixel
Timeou t 10 #define _ voltage    _ add 144                          protocol / / S izeoftheinputvoltagei nbytesinthe Dynamixel protocol / / Timeou tforc
Motor_Num 2        _ voltage    _ size 2                            ommunica ti on wi th the Dynamixel mo tor controller / / Number ofmotorsb ei ngcontrolled .
#define Max_ang_v ( 1 0 . 0
F ) #define Min_ang_v ( ÿ10.0
F ) #define E x pl ore #define max_di s                              // Maximum angular velocityconstr aintforthemotors .
21 # definedt ( 0 .                                                 // Minimum angular velocityconstr aintforthemotors .
                    _ v 8                                           // Maximum translational velocity for the virtual robot.
                                                                    // Maximum sensibledistanceforthevirtualrobot.
                  1                                                 // Time stepforvirtualm obil erobot // ID of the first mo tor
F ) # define Mo to r_1_ID 3 #                                       bei ngcontrolled .
define Mo to r_2_ID 4 # define                                      // ID of the second motor at ngcontrolled .
alpha 1 ( 0 . 8 F ) # define alpha                                  // Constant value for the IIR filter for current .
2 ( 0 . 6 F ) # define Range 56                                     // Constant value for the IIR filter for velocity .
                                                                    // Half size of work area ( theareato be explored ; O_map and F_map ) .
%
# definecs % #  _ length 112                                        // Fullsizeofworkarea(theareatobeexplored; O_map and F_map).

define C enter 1 129 # define                                       // Center of work area inhorizontal ÿaxis ( p hi 1 ) .
C enter 2 51 # defineshift                                          // Center of work area in V ertical ÿaxis ( p hi 2 ) .
                _ x ( Center 1 ÿ Range )                            // H orizontal shift from CS pa ce work area ( O_map and F_map ) .
# defineshift _ y ( Center 2 ÿ Range )                              // Vertical shift from CS space work area ( O_map and F_map ) .
# define Cell      _ bit 1                                          // Number of bits percellin the map ( O_map and F_map ) .
#define R eg _     size 8                                           // Size of the register tos of ethe map ( O_map and F_map ) .
#define R eg _ partition 8 #define M                                //Number ofcellsperregister(Reg_size/Cell//Maximum shiftvalueforaregiste r// _ bits ) .
axs hi ft 7       _                                                 Sizeofthe downsampling image forfeatureextraction      ( Reg _ partition ÿ 1 ) .
# define DWNS_img_size 56 % #

define V alid      _ descriptor 6                                   // Number of valid descriptors for feature ma t ching .

% # define M ax #  _ features        7                                // Maximum number of features for features ma t ching .
define Max_maps 2 # define                                          // Maximum number of maps forfeature ma t ching . ( 2 * Range ) )
Ma p_ si ze ( int ) ( ( ( 2 ÿ Range )                                        / Reg _ partition ) // Size of work area ( O_map and F_map ) .

uint 8   _ t O_Map [ Map_size ] = { 0 } ; t                              // O_Map
uint 8   _ F_Map [ Map_size ] = { 0 } ;                                  // F_Map //
featurefeature     _ set [ Max_maps ] [ Max    _ features ] ;            featuresset ( Max_map x M ax_feature ) // smallfeaturessetwillo
do ubl e D ata    _ P oi nts [ Max_maps ] [ 3 ] [ M ax    _ features ] ; t   nl ys av eposition and idoffeaturesbit 0 forpixelv al uebit 1ÿ7 forstoringlabel
uint 8   _ DWNS_img [ DWNS_img_size ÿ DWNS_img_size ] ; // Downsampled image ,

typedefstruct { do ubl
    edescriptor [ Validuint 8         _ descriptors ] ;     / / A do ubl earray wi th size V alid / /     ,    _ descriptors
         _ tpositiontposition x ;                            xpositionofthefeature / / ypositionofthefeature,/ / centroid .
    uint 8   _ _ y ; tid ; } feature ;                       features ID ÿ1 for nonÿexist              scentroid .
    int 8  _
```

C.3.1 Exploration

Read and Write of OMap **and** FMap

```
uint 8 _ t read_map ( intar g1 , g2 , // Clamp uint 8 _ t ÿ map_p tr ) { intar
    the coordinate to wi t hi n the bounda ri es of the map . arg 1 = ( arg 1 >= 0) ? ( arg 1 < csarg 2 =
    ( arg 2 >= 0) ? ( arg 2 < cs _ length ) ? arg 1 : length ) ? cs _ length ÿ 1 : 0 ; length ÿ
                                 _ argument 2 : cs _ 1 : 0 ;
                                 ,
    // Compute the index into the map sarray . int temp = ( int ) ( arg 1 /
    ( Reg _ size / Cell _ bits ) ) + ( ( cs _ length ) / ( Reg_size ) ) ÿ arg 2 ;

    // Compute the shift required to ext ract the value of the pixel .
    uint 8 _ tshift = ( Reg _ partition ÿ 1 ÿ arg 1 % Reg _ partition ) ;

    // Extract and return the value of the pixel return ( map_p tr [ temp ] & ( 1 ·
    << shift ) ) >> shift ;

} v oi dwrite_map ( intar g1 , intar g2 , // uint 8 _ tv al ue , uint 8 _ t ÿ map_p tr ) {
    Clamp thecoordinateto wi thi nthebounda ri esofthe map . arg 1 = ( arg 1 >= 0) ? ( arg 1 < csarg 2 =
    ( arg 2 >= 0) ? ( arg 2 < cs _ length ) ? arg 1 : length ) ? cs _ length ÿ 1 : 0 ; length ÿ
                                 _ argument 2 : cs _ 1 : 0 ;
                                 ,
    // Compute the index into the map sarray . int temp = ( int ) ( arg 1 /
    ( Reg _ size / Cell _ bits ) ) + ( ( cs _ length ) / ( Reg_size ) ) ÿ arg 2 ;

    // Compute the shift required to set the value of the pixel shift ift = ( Reg _ partition ÿ 1 ÿ arg 1 % ·
    uint 8 _ Reg _ partition ) ;

    // S etorclearthevalueofthepixeldependi ng on theinputvalue . if ( value == 1) { map_p tr [ temp ] } else { map_p tr
    [ temp ] &= ~( 1 << shift ) ;
                    | = ( 1 << shift ) ;


    }
}
```

Distance Sensor

```
v oi ddistance _ sensor ( int 1 6 _ t pos_x , t pos_y , int 1 6 // fouranglesin which _ float head ding , do ubl e ÿ l _ value , do ubl e ÿ r _ value ) {
    to measure re distancetoobstaclesfloatan gl e [ 4 ] ={ M_PI ÿ( 1 5 /
                                            1 8 0 . 0 ) , M_PI ÿ( 1 0 / 1 8 0 . 0 ) , M_PI ÿ( ÿ10 / 1 8 0 . 0 ) , M_PI ÿ( ÿ15 / 1 8 0 . 0 ) } ;

    // I initializethedistancev aluesfor each an gl etothe maximum distancedo ubl edistance[4] ={ max_dis , max_dis , max_dis ,
    max_di s } ;

    // Declarevariablestostor ethepositionof each sensorint 1 6
          _ t x_ sens , y_ sens ;

    // I teratethrough each anglefor ( uint 8 i < 4 ; i++)
    { // positionofthe_sensoratthecurrentangl e i = 0 ;

              x _ sens = po s_x + max_dis ÿ cos ( head di ng + an gle [ i ] ) ; y_ sens = po s_y +
              max_dis ÿ sin ( head di ng + an gl e [ i ] ) ;

              // I teratethrough each point along the line be tween therobot and thesensorfor( uint 8 j < max_di s ; j++) { // positionofthecurrentp
              ointfloatu = j/float ( max _di s ) ; tx = round ( po s_x ÿ ( 1 ÿ u ) + j = 0 ;
                        xty = round ( po s_y ÿ ( 1 ÿ u ) + y_ sens ÿ u ) ;

                        int 1 6 _                          _ sense ÿ u ) ;
                        int 1 6 _

                        // Check if the current point is an obstacle if ( read_map ( x , y , O_Map ) ==
                        1 ) { // distancetotheobstacle and storeitdistance[ i ] =
                                  sqrt ( ( po s_x ÿ x ) break ;
                                                      ÿ ( po s_x ÿ x ) +( po s_y ÿ y )        ÿ ( po s_y ÿ y ) ) ;

                             }
                   }
         }

    // averagedistancetoobstacles on theleft and rightsidesoftherobotd o uble temp_l =( distance [ 0 ] + distance [ 1 ] ) / 2 . 0 ; do uble
    temp_ r =( distance [ 2 ] + distance [ 3 ] ) / 2 . 0 ;


    // Add random noisetothedistancev al ues ( ÿ l ( ÿ r        , buto nl yiftheyaregreatert ha n 0 . value ) = ( temp_l > 0 . 2 ) ?  2
       _ ( temp_l + random ( ÿ200 2 0 0) / 1 0 0 . 0 ) : 0 . 0 ; value ) = ( temp_ r > 0 . 2 ) ? ( temp_ r + random ( ÿ200 2 0 0) / 1 0 0 .
       _ 0 ) : 0 . 0 ;                                       ,
}
```

Machine Translated by Google

loop 1

```
// Check ifitif ( current ' s tim eto run the first task
M icros ÿ previous sM icros 1 >= period 1 ) { voltage = 0 ;
    uint 1 6 _ input _
    // Read theinputvoltageof Motor 1 and storeitinvoltage 1 dxlread ( Motor_1_ID ,
      · Input_Voltageadd _ , input _Voltage ( uint 8 size voltage 1 = _ t ÿ)& input _voltage , sizeof( input _voltage ) , timeout );
    input _voltage ;
    // Read theinputvoltageof Motor 2 and storeitinvoltage 2 inputdxl
        _voltage = 0 ; read
      · ( Motor_2_ID , voltage I nput _V _add , input _voltage _size , (uint 8 _ t ÿ)& input _voltage , sizeof( input _voltage ) , timeout );
    2 = input _oltagevoltage ;

    // Loop through the obstacle map and paint any obstacles detected for ( uint 8
            _ti = 0 ; for i < range ÿ 2 ; i++) { tj = 0 ; j <
        (uint 8 _range ÿ 2 ; j++) { if ( read_map ( j , i
                              , O_Map ) ) {
                Paint _S et P ixel ( 3 0 + i + ( C enter 2 ÿ Range ) , 70 + j + ( Center 1 ÿ Range ) , RED );
            }
        }
    }
    // C reatebordersatthe ed ge oftheobstacle map toav oi drobotdrivingoutofthe work areafor ( uint 8 i < Range ÿ 2 ; w ri te_map ( 0
            _ti = 0 ; i++) {
                , i , , O_Map );
        1 write_map ( Range ÿ 2 ÿ 1 , i , 1 , O_Map );
        write_map ( i , Range ÿ 2 ÿ 1 , 1 , O_Map );
        write_map ( i 1 , 0 , , O_Map );
    }
    // Upda te thepreviou sM icros 1 variableto keep trackof when to run thistaskag ai npreviou sM icros 1 += period 1 ;

}
```

loop 2

```
// Check ifits tim eto run thesecondtaskif ( current M icros ÿ previou sM
icros 2 >= period 2 ) {
    // Upda te thepreviou sM icros 2 variablefor timing purposespreviou sM icros 2 += period 2 ;

    // C all the visual visual    _ robotfunctiontocleartheprevioustvisualizati onofr ob ot    shea di ng
        _ robot ( head ding ,    1 ) ;

    / ÿ Ge tessentialdata ÿ / s / / Ge
    tpresentleftpresentve locity and rightpresentvelocityf loat ang1_v = dxl . get Present Velocity
    ( Motor_1_ID , UNIT_RPM ) ; float ang2_v = dxl . get Present Velocity ( Motor_2_ID , UNIT_RPM ) ;

    // Get tpresentleftpresentposition and rightpresentpositionfloatpresent_positionfloatpresent_position
                        _ 1 = dxl    . get Present P osition ( Motor_1_ID , UNIT_DEGREE ) ; . get Present P
                        _ 2 = dxl    osition ( Motor_2_ID , UNIT_DEGREE ) ;

    // Get tpresentfloatcurrentleftpresentcurrent and rightpresentcurrent 1 = ab s ( dxl . get Present
                _ Current ( Motor_1_ID , UNIT_MILLI_AMPERE ) ) ; 2 = ab s ( dxl . get Present Current ( Motor_2_ID ,
    float current    _ UNIT_MILLI_AMPERE ) ) ;

    / / Apply to IIR filtertothepresentcur rentreadingsf 1 + alpha 1 ÿ f 2 + alpha 1 ÿ f
     _ current    _ 1 = ( 1 ÿ alpha 1 ) 2 = ( 1    ÿ current    _    _ current    _ 1 ;
    f _ current    _ ÿ alpha 1 )    ÿ current    _    _ current    _ 2 ;

    / / Apply an IIR filtertothepresentvel ocityreadings f_ ang 1_v = ( 1 ÿ al pha 2 ) f_ ang 2_v
    = ( 1 ÿ al pha 2 )    ÿ ang1_v + alpha 2 ÿ f_ ang 1_v ; ÿ ang2_v +
                          alpha 2 ÿ f_ ang 2_v ;

    // C o nve rtthepresentpositioner eadings from degreestointegersint 1 6
            _ tphi 1 = round ( fmod ( present _ positiontphi2 = round    _ 1 + 1 3 5    ,    3 6 0 . 0 ) );
    int 1 6    _ ( fmod ( present _ positionphi 1 = ( phi 1 < 0) ? phi 1 +    _ 2 + 4 5    ,    3 6 0 . 0 ) );
    360 : phi 1 ; phi 2 = ( phi 2 < 0) ?phi 2 + 360 : phi 2 ;

    / / Apply shift from CS pa ce coordinatestothe work areacoordinatesx : p hi 1 ÿ shift
    int 8    _ tpo s_x = ( p hi 1 >= 360 + shiftt po s_y = ( p hi    _ x ) ? phi 1 ÿ 360 ÿ shift    _    _ x ;
    int 8    _ 2 >= 360 + shift    _ y ) ? phi 2 ÿ 360 ÿ shift    _ y : phi 2 ÿ shift _ y ; lengthÿ 1 :
    po s_x = ( po s_x >= 0) ? ( po s_x < cs pos_y = ( pos_y _ length ) ? po s_x : cslength ) ?    _ 0 ;
    >= 0) ? ( pos_y < cs    _ po s_y : cs    _ length ÿ 1 : 0 ;

    // Perform collisiondetection and set thecollisionvariab leaccordingly
    uint 8    _ tcollision = 0 ; collision =
    ( f / ÿ Ge tessentialdata_ current    _ 1 > 1 7) << 0 |    (f _ current    _ 2 > 1 7) << 4 ;
    ÿ /

    / ÿ L abelization ÿ / / / L
    abelizeobstacleinobst acle map ifacollision ha s been detectedif ( collision ! = 0) {

        // Check forobstacles wi t hi n 5 unitssoftherobot and label them in theobstacle map
        x _ sens = po s_x + 5 ÿ cos ( head di ng ) ; sin ( head
        y_ sens = pos_y + 5 ÿ for ( int 8    di ng ) ; ti = ÿ1; i <
                _ 2 ; i ++) { sens + i ÿ cos ( head di ng
            uint 8    _ tx = int ( x    _ + 1 . 5 7 ) ) ; . 5 7 ) ); ty = int ( y_ sens + i ÿ sin ( hea
            uint 8    _    di ng + 1
            if ( read_map ( x , y , F_Map ) ! = 1) { write_map ( x ,
                O_Map ) ;    , y ,    1
            }
        }
    }

    // C learther ob ots nea rb yareaintheobstacle map to make iteasiertonavigatew ri te_map ( pos_x , po s_y + 1
                                ,    0 , O_Map );
    write_map ( pos_x , pos_y ÿ 1 0    ,    , O_Map );
    write_map ( pos_x , pos_y ,    0 , O_Map );
    write_map ( po s_x ÿ 1 0    pos_y ,    , O_Map ); ,
    write_map ( pos_x + 1    pos_y ,    0 , O_Map ); ,

    // Labelizether ob otw ri te_map    s nea rb yareaasfreeinthefree map
    ( pos_x , pos_y + 1    ,    1 , F_Map );
```

Machine Translated by Google


```
write_map ( pos_x , pos_y ÿ 1          ,      , F_Map ); 1
write_map ( pos_x , pos_y ,            1 , F_Map );
write_map ( po s_x ÿ 1          pos_y ,          , F_Map ); 1 ,
write_map ( pos_x + 1           pos_y , / ÿ     1 , F_Map ); ,
labelization ÿ /

/ ÿ Mo tion Pl an ni ng ÿ /
// Perception                                                                         ,
// Use distancesensorstodete ctobstacles and dete rmi nether ob otdo ubl inputdo ubl inputdistancesensor    ssurroun di ngs
( pos_x , pos_y ,     _ l = 0 ;
                      _ r = 0 ;
   _                                      hea ding       &input    _ l  ,   &input    _ r) ;

// Minimal Recurrent Controller ( 1 2 . 0 ÿ inputr =
      _ l = ta nh ( 0 .     4 ÿ    tanh ( 0 . ( 1 2 . 0 ÿ    _ l ) ) ;     / / ÿ1 free / / ÿ1   ,   1 occupied ; di s_max free 0 occu pi ed 1 occu pi ed ; di s_max
      _ input 4 ÿ l = ÿ7.0 ÿ inputr = ÿ7.0 ÿ inputl ) ; r ) ;   _ r) );      free      ,   free 0 occu pi ed                 ,
inputinputdo ubl eact                     _ r + 3 . 0 ÿ new _l ÿ 4 . 0 ÿ new_ r ;
do ubl eact      _                        _ l + 3 . 0 ÿ new_ r ÿ 4 . 0 ÿ new _l ;
new _l = ta nh ( act new_   _
r = ta nh ( act / ÿ Mo tion    _
Pl an ni ng ÿ /

/ ÿ D ifferential D riv e Robo t Kinema ti cs ÿ / floatv = ( E x pl orev
= ( v > 1) ? v : float omega =   _ v ÿ     ( new_ r + new _l ) ) /          2 . 0 ;         // average velocity
( E x pl orehea di ng +=        1 ;                                                   // L imitates the minimum velocity to 1 // angularvelocity
omega ÿ dt ; hea di ng = fmod ( hea  _ v ÿ ( new_ r ÿ new _l ) ) / 2 . 0 ;
di ng + 2 . 0 ÿ M_PI , 2 hea di ng =                                                 // Updated the hea di ngoftherobot
( hea di ng < 0) ? hea di ng + 2 . 0 ÿ M_PI : hea di ng ; d_ p hi1   . 0 * M_PI ) ;    // E nsure the hea di ngis be tween 0 and 2 PI
= v ÿ cos ( hea di ng ) ; d_ p hi2 = v ÿ sin ( hea di ng ) ; / ÿ D ifferential D riv e Robo t Kinema ti    //Ensure thehea di ngispositive//velocityof mo tor 1//
cs ÿ /                                                                               velocityof mo tor 2


/ ÿ Re c ove rybeha vi our ÿ / / / I
ftherobotisstuck and one ofthe mo tor velocitiesiscloseto ze ro / / add asmall amount ofrotationtothehea di ngtotry and    ,
getitunstuckif ( ( ab s ( d_ p hi1 ) >= 0 . 1 && ab s ( f_ ang 1_v ) <= 0 .1 ) head di ng += 0 .0 5 ;
                                                        ( ab s ( d_ p hi 2 ) >= 0 . 1 && ab s ( f_ ang 2_v ) <= 0 . 1 ) ) {


} / ÿ Re c ove rybeha vi our ÿ /

/ ÿ Motor C ontr ol ÿ / d_ p                                                                                        ,
hi1 = ( d_ p hi1 > Min_ang_v ) ? ( ( d_ p hi1 < Max_ang_v ) ? d_ p hi 1 : Max_ang_v ) d_ p hi2 = ( d_ p hi2 > Min_ang_v ) ?    : Min_ang_v ;    // Clamp motor 1 velocity   ,
( ( d_ p hi2 < Max_ang_v ) ? d_ p hi 2 : Max_ang_v ) dxl . setGoalPWM( Motor_1_ID , d_phi 1 * ( 5 0 . 0 / voltage 1 ) ,    : Min_ang_v ;    // Clamp motor 2 velocity
UNIT_PERCENT ) ; dxl . setGoalPWM(Motor_2_ID, d_phi2*(50.0/voltage2), UNIT_PERCENT); / ÿ Motor C ontr ol ÿ /      // Set goal PWM for motor 1
                                                                                                                    // Set goal PWM for motor 2


/ ÿ V isualization ÿ / phi 1 =
( phi 1 > 1 8 0) ? phi 1 ÿ 360 : phi 1 ; phi 2 = ( phi 2 > 1 8 0) ? phi 2      // En sure thatp hi 1 is between tween ÿ180 and 180
ÿ 360 : phi 2 ; Set P ixels ( 3 0 + phi2 , 70 + phi1 , GRAY ) ; robot          // En sure thatp hi 2 is between tween ÿ180 and 180
Paint    _ ( head ding , 0 ) ; / ÿ V isualization ÿ /                          // S etapixelinthevisualization                ,
visual    _                                                                    // Update the visualization of the r ob ot      scurrenthea di ng

}
```

C.3.2 Map Comparison

perimeter

```
// Computes theperimeterofas ha pe inbinary image . uint 1 6 uint 8
    _ tperimeter ( uint 8 // I      _ t ÿ   input   ,  uint 8  _ t wid th     ,          _ theight ) {
  nitializetheperimeter variabletozero . uint 1 6
      _ perimeter = 0 ;

  // Loop through all pixels of the input image , for ( uint 8 i < wi d th ÿ     excluding theoute rm east row and column .
  1 ;           _ ti = 1 ;                              i++) { tj
    for (uint 8      _ = 1 ; j < height ÿ 1 ; j++) { t kernel_ sum = 0 ;
      uint 8  _

      // I fthecurrentpixelisafo regroundpixel ( value 1 ) , compute the sum of its 4ÿn ei ghbors . if ( read_map ( iinput ) == 1) { , j , ke r nel_ s um
      += read_map ( i + 0 , j ÿ 1 ke r nel_ s um +=
        read_map ( i ÿ 1 , j + 1 ke r nel_ s um += read_map  ,  input ) ;
        ( i + 0 , j + 1 ker nel_ s um += read_map ( i + 1 , j + 0 ,  input ) ;
                                                             ,  input ) ;
                                                             ,  input ) ;

        // I fthe sum ofthe 4ÿn ei ghborsislesst ha n 4 if ( ker nel_ s um < 4)     ,  incr em enttheperimetervariab le .
        { perimeter + +;

        }
      }
    }
  }

  // I terate over the top , and right edge bo t tom, soft the left image .   ,
  // I faforegroundpixelis found incr em enttheperimetervariab le . for ( uint 8 j ++) { perimeter =
  ( read_map ( j , input ) == 1) ? ( perimeter + 1) :    _ t j = 0 ; j < wi d th ; 0
    perimeter ;                             ,

  } for ( uint 8 i < height ; perimeter = ( read_map   i ++)    _ ti = 0 ;
    ( 0                          ,  i  ,  { input ) == 1) ? ( perimeter + 1) : perimeter ;

  } for ( uint 8    _ t j = 0 ; j < wi d th ; j ++) { height ÿ 1
    perimeter = ( read_map ( j ,                       ,  input ) == 1) ? ( perimeter + 1) : perimeter ;

  } for ( uint 8    _ ti = 0 ; i < height ; i ++) { perimeter =
    ( read_map ( wi d th ÿ 1 i                    ,    ,  input ) == 1) ? ( perimeter + 1) : perimeter ;
  }

  // Return the final perimeter value . returnperimeter ;

}
```

width and length

```
/ / compu tes the wi d th and heightofthelargestrec tangulars ha pe presentinthe bi nary image v oi dc om putedo ubl e ÿ wid th
                    _ shape   _ si ze ( uint 8      _ t ÿ    input      ,                  ,    do ubl e ÿ height ) {
      // I initialize moment variables to zero ubl e m00 = 0 uint 8
                          ,   m10 = 0 idx ; ,   m01 = 0 ;
            _ tx , y ,

      // Compute image moments
      // Loop through all pixels of the in pu image for ( y = 0 ; y < DWNS_img_size ;
      y ++) {
            for ( x = 0 ; x < DWNS_img_size ; x ++) { idx = y ÿ DWNS_img_size
                  + x ;
                  // Check ifpixeliswhite and update moment variablesif ( read_map ( xinput ) ) { , y , m00 += 1 ;
                  m10 += x ; m01 += y ;


                  }
            }
      }

      // Compute the centroid of the ha pe do ubl e x_mean = m10 /
      m00 ; do ubl e y_mean = m01 / m00 ;


      // I nitializecentral moment variablestozerodo ubl e mu11 = 0 mu20 = 0
                          ,              ,   mu02 = 0 ;

      // Compute central moments
      // Loop through all pixels of the in pu image for ( y = 0 ; y < DWNS_img_size ;
      y ++) {
            for ( x = 0 ; x < DWNS_img_size ; x ++) { idx = y ÿ DWNS_img_size
                  + x ;
                  // Check ifpixeliswhite and updatecentral moment variablesif ( read_map ( xinput ) ) { , y , diff = x ÿ x_mean ;

                        do ubl ex      _
                        do ubley _ diff = y ÿ y_mean ; diff ÿ y _ diff ;
                        mu11 += x         _
                        mu20 += x         _ diff * x        _ diff ;
                        y _                  diff ÿ y _ diff ; mu02 +=
                  }
            }
      }

      // Compute no rm a zedcentral moments do ubl e u11 = mu11 /
      m00 ; do ubl e u20 = mu20 / m00 ; do ubl
      e u02 = mu02 / m00 ;


      // Compute semiÿaxesoftheellipsethatf itsthes ha pe do ublea = sqrt ( ( u20 + u02 + sqrt ( ( u20 ÿ
      u02 ) ( u20 ÿ u02 ) + 4 ÿ u11 ÿ u11 ) ) / 2 ) ; do ubl eb = sqrt ( ( u20 + u02 ÿ sqrt ( ( u20 ÿ u02 ) ( u20 ÿ u02 ) + 4 ÿ u11 ÿ u11 ) ) / 2 ) ;


      // Update the wi d th and heightpointers wi th the dim en si onsoftheellipse ÿ wi d th = a ÿ 2 ; ÿ height = b ÿ 2 ;


}
```

Machine Translated by Google

Hu moments

```
// compu tes the ( p + q ) th moment of the ha pe inbinary image . do ubl e calcMoment ( do ubl ep , do ubl eq ,
t ÿ inputt wid th // I nitializeavariabletoa cc um ul atethe sum of the moments uint 8 _ do ubl e sum = 0 ; // Loop uint 8 _ , uint 8 _ theight ) {
    throughallpixelsofthe in pu image . for ( uint 8 tx = 0 ; x < wi d th ; x ++) { for ( uint 8 // E xtractthevalueofthepi
    xel and use it toupdatethe
    sum . sum += read_map ( x ÿ pow ( x , p ) ÿ pow ( y , q ) ;
                _
                    _ ty = 0 ; y < height ; y++) {

                                , height ÿ 1 ÿ y , input )
        }
    }
    // Return the final sum of moments. return total ;

}

// calculates the central moment of the ha pe inbinary image do ubl ec al cC entr alM om ent ( do ubl ep ,
uint 8 uint 8 first do ubl eq , t wid th uint 8 _ t * input , _ , _ theight ) {
    // C alculate the zeroth and second order moments of the binary image do ubl e M00 = calcMomen t ( 0 inputheight ); do ubl e M10 =
    calcMoment (1 height) ; inputdo ubl e M01 = calcMoment ( 0 inputheight ) ; 0 , wid th ,
                                , 0 , , wid th ,
                                , 1 , , wid th ,
    // C alculatethex and ycoordinatesoftheorig inofthe image do ubl eor ginO fX = M10 / M00 ; do ubl eor gi nO fY
    = M01 / M00 ; do ubl e sum = 0 ;


    // Loop through all pixels of the in pu image and calculate the ( p+q ) for ( uint 8 thcentral moment
                _ tx = 0 ; x < w id th ; x ++) { for ( uint 8 sum +=
        read_map ( x _ ty = 0 ; y < height ; y ++) { height ÿ 1 ÿ y ,
                                , input ) ÿ pow ( ( x ÿ or ginO fX ) , p ) ÿ pow ( ( y ÿ or ginO fY ) , q ) ;
        }
    }
    // Return the calculated moment return sum ;

}

// calculatestheno rm ali zedcentral moment ofthes ha pe inbinary image do ubl ec al cN o rm ali ze dC enrt alM om ent ( do ubl
ep , do ubl eq , uint 8 // C alculatetheorderofthe moment do ubl eorder = ( ( p + q ) .0 ) + 1 ; / 2 / / C alculatetheno rmalized moment u uint 8 _ t ÿ input _ t wid th , uint 8 _ theight ) {
    si ngtheorigin moments do ublea = c al cC entr alM oment ( p ,
    q , wid th / / R eturnthecalculatedno rm al ized moment
    returna ;
                                        input , , height ) / pow ( c al cC entr alM om ent ( 0 , 0 , input , wid th , height ) , order );


}

// calculatesthefirst two Hu moments ofthes ha pe inbinary image v oi d getHuMoments ( uint 8 uint 8 uint 8
                        _ t * input , _ t wid th , _ the eight , do ubl e ÿ hu1 , do ubl e ÿ hu2 ) {
    // C alculatethesecondorde rno rm ali zedcentral moments do ubl e nu20 = c al cN o rm ali ze dC
    enrt alM om ent ( 2 do ubl e nu02 = c al cN o rm ali ze dC enrt alM o ent ( 0 , 0 , , wid th , height ) ;
    do ubl e nu11 = c al cN o rm ali ze dC enrt alM om ent ( 1 , , , wid th , height ) ;
                                                , 2 1, inputinputinput wid th , height ) ;

    // C alculatethefirst two Hu moments u si ngtheno rm ali zedcentral moments ÿ hu1 = nu20 + nu02 ; ÿ hu2 = pow ( ( nu20 ÿ nu02 ) ,
    2) + 4 ÿ
                                        ( pow ( nu11 , 2) );
}
```

Connected component labeling

```
// extractsconnectedpixe ltconnected
uint 8   _                    _ component      _ label li ng ( uint 8 //   _ t ÿ   input   ,  uint 8   _ t ÿ equ    _ list   ,  uint 8   _ t wid th    ,  uint 8   _ theight ) {
    bit 1ÿ7 of ÿ input will be use se dtostorethelabel // bit 0 isthepixelvalue

    // O_Map will be use the buffer of equivalents                                list

    uint 8   _ text       _ label = 1 ;          // I initialize then ex tavailablelabelto 1

    // F irstpass : labelconnected components ts
    // Loop through all pixels soft the downsampling image for ( uint 8
              _ ty = 0 ; y < height ; y++) { for ( uint 8 uint 1 6
                  _ tx = 0 ; x < w id th ; x++) {
                  _ tindex = height ÿ y + x ;

            // Check ifthecurrentpixelispartofa componen tif ( ( input [ index ] & 0 b 0 0 0 0 0 0 0
            1 ) == 1) { label = 0 ;
                uint 8   _ tn    _                   //    I nitializethen ei ghbo ri nglabelto 0

                // Check n ei ghbo ri ngpixelsforexistingla bels
                // Check leftpixelif ( x > 0)
                { index = height ÿ
                    y + x ÿ 1 ; tdata = input [ index ] ; if ( data &
                    uint 8   _ 0 b 0 0 0 0 0 0 0 1 == 1) { data =
                    data >> 1 ;

                        // Upda te then ei ghbo ri nglabel wi th the minimum valuelabel == 0 | | data < n
                        n _ label = ( n      _                          _ labels ) ? data : next          _ labels ;
                        // Update the equivalent list label l da ta label
                        w ri te_map ( nw   _          ,        ,    ,  equivalent _ lists ) ;
                        ri te_map ( da ta       ,  n _         ,  1 1,  equivalent _ lists ) ;
                    }
                }

                // Check above pixelif ( y > 0)
                { index =
                    DWNS_img_size ÿ ( y ÿ 1) + x ;
                    uint 8   _ tdata = input [ index ] ; if ( data & 0
                    b 0 0 0 0 0 0 0 1 == 1) {
                        data = data >> 1 ; label =
                        n _ ( n             _ labels == 0 | | data < nlist ) ; lists ) ;  _ labels ) ? data : n          _ labels ;
                        w ri te_map ( nw  _ labels  ,  there ta  ,  1 ,  equivalent _
                        ri te_map ( da ta       ,  n _ labels   ,  1 ,  equivalent _
                    }
                }

                // Check above leftpixelif ( x > 0 && y >
                0) {
                    index = DWNS_img_size ÿ ( y ÿ 1) + x ÿ 1 ; uint 8
                        _ tdata = input [ index ] ; if ( data & 0
                    b 0 0 0 0 0 0 0 1 == 1) { data = data >> 1 ;
                        label = ( n
                        n _                 _ labels == 0 | | data < n data list ) ; 1  _ labels ) ? data : n          _ labels ;
                        w ri te_map ( nw  _ labels  ,  1 list );,   ,  equivalent _
                        ri te_map ( da ta       ,  n _ labels   ,      ,  equivalent _
                    }
                }

                // Check above rightpixelif ( x <
                DWNS_img_size ÿ 1 && y > 0) {
                    index = DWNS_img_size ÿ ( y ÿ 1) + x + 1 ; uint 8
                        _ tdata = input [ index ] ; if ( data & 0
                    b 0 0 0 0 0 0 0 1 == 1) { data = data >> 1 ;
                        label = ( n
                        n _                 _ labels == 0 | | data < nlist ) ; lists ) ;  _ labels ) ? data : n          _ labels ;
                        w ri te_map ( nw  _ labels  ,  there ta  ,  1 ,  equivalent _
                        ri te_map ( da ta       ,  n _ labels   ,  1 ,  equivalent _
                    }
                }

                // I f no label found if ( n          ,  assigna new one
                     _ labels == 0) {
```

Machine Translated by Google


```
n _ label = next _ labels + +;
}

// Store the label inside rvedbitsoftheinputpix elindex = height ÿ y + x ; input [ index ] | = (n

_ labels << 1 ) ;
// Update the equivalent list for the current label if ( n
_ label ! = 0)
{ write_map ( n _ labels , n _ labels , 1 , equivalent _ lists ) ;
}
}
}
}
// Second Pass : updateequivalentlist
// Loop through all the pixels in the input image for ( uint 8 ty = 0 ; y < height ; y
++) { for ( uint 8 _
_ tx = 0 ; x < w id th ; x++) {
// Read thelabelvalue ( bit 1ÿ7) from thefirstpassuint 8
_ tlabel = ( input [ height ÿ y + x ] & 0 b 1 1 1 1 1 1 1 0 ) >> 1 ;

// Check if the label is not 0 if ( label ! = 0) { //
Find the minimum index in
the same labelset based on equivalent list
uint 8 _ t min_l = label ; i < csi
for (uint 8 _ ti = 1 ; if _ length ; i ++) { list ) ) {
( read_map ( labelif ( i < , , equivalent _
min_l ) { min_l = i ;

}
}
}
// I flabelisnotthe minimum labelinthesetif(label!=min_l) , clear the label and update the input value
{ write_map(labellabellist); input
[ height * y + x ] = ( min_l,<< 1 | 0 b 0 0 0 0 0 0 0 1 ) ;

}
}
}

} uint 8 _ tcount = 0 ;
// Count the number of labels for ( uint 8
_ tx = 0 ; x < cs _ length ; x ++) { list ) ) {
if ( read_map ( xcount , x , equivalent _
++;
}
}

// return the count of labels return count ;

}
```

Machine Translated by Google

Build feature set

```
// Loop through all possible lab els and build feature set for ( uint 8
            _ tc = 1 ; c < cs             _ length ÿ 1 ; c++) { // forallpossiblelabel // ifthe number offeatures
    ex ceedthelimitoffeature sif ( index > M axbreak ;                                     , breakoutoftheloop
                          _ features ) {


    } // initializethe sums forxcoordinatesint sum_x = 0 ; int sum_y , ycoordinates        , and areas
    = 0 ; int sum = 0 ;


    // Loop throughallpixelsofthe downsampling image for ( inti = 0 ; i <
    DWNS_img_size ; i + +) { for ( intj = 0 ; j < DWNS_img_size ;
        j + +) { / / clearallbuffer map pixelsinpreviousloopw ri te_map
            ( j , DWNS_img_size ÿ 1 ÿ ibuffe r_m ap ) ; // read 1ÿ7 bitsofthe
            downsampling image aslabel number and che ckifthey match thecurrentlabelif
            ( ( DWNS_img [ i ÿ DWNS_img_size + j ] & 0 b 1 1 1 1 1 1 1 0 ) >> 1 == c ) { w ri te_map ( j , DWNS_img_size ÿ 1 ÿ ibuffe r_m ap ) ; , copy data into the
            buffer map / / se gm enta ti on / / calculatearea
                                                                    ,   1 ,
                sum + +;
                // acc um ul atethecoordinateof se gmen ted object and covertthecoordinates fomr R asterto C artesian sum_x += j ; sum_y += ( DWNS_img_size ÿ 1 ÿ i ) ;


            }
        }

    } // if the area is large enough, if ( sum >= 1 5 )   consideritavalidfeature
    { // get pointer to the
        current feature and compute its descriptors feature ÿ f = & featuref ÿ> descriptor [ 0 ] = perimeter ( buf
        fe r_ma p , DWNS_img_size ,   _ set [ temp ] [ index ] ;
        DWNS_img_size ) ; size ( buffer_ma p , &f ÿ> descriptor [ 1 ] , &f ÿ> descriptor [ 2 ] ) ;
        com putef  _ shape  _
        ÿ> descriptor [ 3 ] = sum ;
        getHuMoments ( buffer_ma p , DWNS_img_size , DWNS_img_size , f ÿ> positionf     &f ÿ> descriptor [ 4 ] ,        &f ÿ> descriptor [ 5 ] ) ;
        ÿ> positionindex _ x = ( int ) ( sum_x / sum ) ;
        + +;              _ y = ( int ) ( sum_y / sum ) ;

    }
}
```

ID assignment

```
// N o rm ali zefeaturedescriptorsu si ng minÿmax scalingfor ( uint 8
        _ tk = 0 ; k < Valid                    _ descriptors ; k++) { // for each descriptor
    do uble min_val = 6 5 5 3 6 ; do ubl e
    max_val = 0 ; for ( uint 8 for ( uint
    8 j++) { / / for    _ ti = 0 ;          i < Max_maps ; t j =        i++) { // for each map features ;
        each feature / / find minimum and maximum of each descripterofvalidfeat ures ( positionisnif ( feature
                                                                                         toutofbounda ri es ) set [ i ] [ j ] .
                    _ set [ i ] [ j ] . position          _ x ! = DWNS_img_size && featureset [ i ] [ j ] .     _ position _ y ! = DWNS_img_size ) )
            do ubl eval = featureif ( val <          _ descriptor[k];
            min_val ) { min_val = val ;

            } if ( val > max_val ) { max_val
                = val ;
            }
        }
    }

    } // ap pl y minÿmax scalingto each descriptorvalueforall featuresfor ( uint 8 i++) { // for each map features ;
            _ ti = 0 ;          i < Max_maps ; t j =
        for ( uint 8 j++) { // for each feature // norma li ze each descripterofvalidfeat ures ( positionisnif ( feature
                                                                          toutofbounda ri es ) set [ i ] [ j ] .
                    _ set [ i ] [ j ] . position          _ x ! = DWNS_img_size && featureset [ i ] [ j ] .     _ position _ y ! = DWNS_img_size ) )
            do ubl eval = featuredto ubl      _ descriptor[k];
            enormalized          _ val = ( val ÿ min_val ) / ( max_val ÿ min_val ) ;
            feature    _ set [ i ] [ j ] . descriptor [ k ] = normalized          _ val ;
        }
    }
  }
}

// A s signfeature ID based on similarityid = 0 ;
uint 8   _ text    _
for ( uint 8 for   _ ti = 0 ;        i < Max_maps ; i++) { // for each map features ; j + +) {
    ( uint 8       _ t j = 0 ;        j < M       _                         // for each feature
        axfeature ÿ f = & feature         _ set[i][j]; //
        skipinvalidfeatures ( positionisnotoutofbou nda ri es ) if ( f ÿ> positioncontinue ;
                    _ x == DWNS_img_size | | f ÿ> position _ y == DWNS_img_size ) {

        } do ubl e mi n    _ distance = 6 5 5 3 6 ;
        int 8 // _ nearest      _ id = ÿ1;
        compare each featuretoallothers and assign ID sbased on similarityfor ( uint 8 // for each map for ( uint 8 l = 0 ; feature ÿ other =
        & featureif ( other ÿ> position _ tk = 0 ; k < Max_maps ; k + +) { l < Max
                    _ t                  _ features ;       l + +) { / / for each featureset [ k ] [ l ] ;
                                              _
                              _ x == DWNS_img_size | | other ÿ> position _ y == DWNS_img_size| | other ÿ>id < 0 ) {

            } do ubl ed = distance ( fif ( d < min    ,    other ) ;
            distance = d ; id =    _ distance ) {
                mi n   _ other ÿ>id ;
                nearest      _
            }
        }

        } // assign ID to feature based on similarity if ( mi n
                _ distance < 0 . 7 ) { f ÿ>id =
            nearest          _ ID ;

        } else {
            f ÿ>id = next        _ id + +;
        }
    }
}
```

Calculation of the sum of the area and the Euclidean distance of each map and the angle dierence

```
// Copy allfeaturesinsmalldat asetbased on id and thepositionshiftedto mi d dl eof image for ( uint 8 for ( uint 8 i = 0 ; feature ÿ f = & feature
            _ tm = 0 ; m < 2 ; m++) { / / loopthroughbothfeatur esetsfeatures ; i++) { // loopthrough each
              _ t            i < Max       _ featureinset // getapointertothefeatu re
                                  _ set [m] [ i ] ;
     // calculatex and ycoordinatesoffeature
     Data   _ P oi nts [m ] [ 0 ] [ i ] = ( f ÿ> position ( f ÿ> positionx ÿ ( DWNS_img_size / 2 ) )  ?  _  : 0 ; :
     Data   _ Points [m] [ 1 ] [ i ] = ( f ÿ> position _ y ! = DWNS_img_size ) ? ( f ÿ> position _ y ÿ ( DWNS_img_size / 2 ) ) // iffeature ha s no ID setÿcoordinateto 0 and add    0 ;
     distancetodistancearr ayif ( f ÿ>id == ÿ1){ P oi nts [m ] [ 2 ] [ i ] = 0 ; distance [m] += sqrt ( f ÿ> position

       Data   _
                                              _ x ÿ f ÿ> position          _ x + f ÿ> position _ y ÿ f ÿ> position _ y ) ;

     } // iffeature has an ID else {          ,  setÿcoordinateto ID and incr em entcount

       Data  _ Points [m ] [ 2 ] [ i ] = f ÿ>id ; count[m] =
       count[m] + 1 ;

     } / / add areatoareaarrayarea [m] += f ÿ>
     descriptor [ 3 ] ;
   }

    I initializevariables
} // do uble min_angle = 0 ;
do uble mi n      _ you stance   _ 1 = 6 5 5 3 6 ;
// Loop throughdifferentangle sfor ( do ubl etheta = 0 ;
                          { theta < 2 ÿ 3 . 1 4 ; theta += 0 .          1 )
   / / I nitializetotaldistanc eforthisan gl edo ubl etotal
                 _ distance = 0 ;

   // Loop throughfeaturesindata set 1 for ( inti = 0 ; features ; i++) {
                    i < Max      _
     //Extractcoordinatesforf eatureido ubl ex = Data
                    _ Points[ 0 ] [ 0 ] [ i ] ;
     do ubley = D ata     _ Points[ 0 ] [ 1 ] [ i ] ;
     do ubl ez = D ata    _ Points[ 0 ] [ 2 ] [ i ] ;
     // Calculatesine and cosine of current an gl etheta = cos ( theta ); theta =
     do ubl ecos     _ sin(theta);
     do ubl esin     _
     / / Apply rotationmat ri xtofeatureido ubl e rx = costheta ÿ y ; theta
     ÿ y ;           _ theta ÿ x ÿ sin       _
     do ubl ery = sindo ubl erz _ theta ÿ x + cos    _
     = z ; // zcoordinate is unchanged

     // I nitialize minimum distanceforhisfeatur edo uble min
                 _ you stance   _ 2 = 6 5 5 3 6 ;
     // Loop throughfeaturesindata set 2 for ( intj = 0 ; j < M ax
                               _ features ; j + +) {
       // Check iffeaturej ha sthe same zÿcoordinateasfeatureii f ( z == D ata
                   _ Points [ 1 ] [ 2 ] [ j ] ) {
         // C alculatedistance be tween rotatedfeaturei and featurejdo ubl e dx = rx ÿ D ata
                               _ Points[ 1 ] [ 0 ] [ j ] ;
         do ubl e dy = ry ÿ D atado ubl ed = _ Points[ 1 ] [ 1 ] [ j ] ;
         sqrt ( dx ÿ dx + dy ÿ dy ) ;
         // Update minimum distance if necessary if ( d < min min
                   _ you stance   _ 2 ) {
              _ you stance   _ 2 = d ;
         }
       }
     }
     // Add minimum distanceforfeatureitototaldistancetotal
          _ distance = ( mi n      _ you stance  _ 2 ! = 6 5 5 3 6) ? (totally     _ distance + mi n     _ you stance  _ 2 )  :  total   _ distance ;
   }

   // Update minimum distance and correspondingan gl eif ( total distance
            _ distance < mi n      _ you stance   _ 1 ) {
     mi n  _       _ 1 = total    _ distance ;
     mi n_angle = theta ;
   }
}
```