Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2016

MSc THESIS

Hybrid Navigation System for Lely Mixing and Feeding Robot

He Cheng

CE-MS-2016-14

Abstract

To date, autonomous robots have been widely used and appear huge advantages in dairy farming that is a labor-intensive industry. Vector is an automatic feeding system developed by Lely to feed cows accurately and flexibly with minimum labor requirements in dairy farms. As the autonomous mobile robot of the Vector, the Mixing and Feeding Robot (MFR) is able to automatically distribute feed for cows based on a reactive navigation system. However, since the reactive approach only responds to stimuli at the moment, MFR is susceptible to errors in the barn environment like the temporary loss of valid ultrasonic or inductive sensing; without following a fence or a metal stripe, MFR is quite limited in its ability to navigate.
In this project, a hybrid navigation system is proposed combining the reactive navigation currently used on MFR and the map-based navigation consisting of mapping, localization and path planning. Due to MFR's incapability to build a map of the environment with current sensors, a laser scanner is added for map building. Based on Adaptive Monte Carlo Localization (AMCL), the Striped-based Adaptive Monte Carlo Localization algorithm (SAMCL) is proposed for MFR to localize itself based on the map. Moreover, a control strategy is developed to switch between the reactive navigation and the map-based navigation accordingly. To evaluate the performance, the hybrid navigation system is tested in the simulation model built by the robotics simulator V-REP and it is concluded that the proposed navigation system improves MFR's error-tolerant capacity and navigation performance.

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Hybrid Navigation System for Lely Mixing and Feeding Robot

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

He Cheng
born in Ruian, China

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Hybrid Navigation System for Lely Mixing and Feeding Robot

by He Cheng

Abstract

To date, autonomous robots have been widely used and appear huge advantages in dairy farming that is a labor-intensive industry. Vector is an automatic feeding system developed by Lely to feed cows accurately and flexibly with minimum labor requirements in dairy farms. As the autonomous mobile robot of the Vector, the Mixing and Feeding Robot (MFR) is able to automatically distribute feed for cows based on a reactive navigation system. However, since the reactive approach only responds to stimuli at the moment, MFR is susceptible to errors in the barn environment like the temporary loss of valid ultrasonic or inductive sensing; without following a fence or a metal stripe, MFR is quite limited in its ability to navigate.

In this project, a hybrid navigation system is proposed combining the reactive navigation currently used on MFR and the map-based navigation consisting of mapping, localization and path planning. Due to MFR's incapability to build a map of the environment with current sensors, a laser scanner is added for map building. Based on Adaptive Monte Carlo Localization (AMCL), the Striped-based Adaptive Monte Carlo Localization algorithm (SAMCL) is proposed for MFR to localize itself based on the map. Moreover, a control strategy is developed to switch between the reactive navigation and the map-based navigation accordingly.

To evaluate the performance, the hybrid navigation system is tested in the simulation model built by the robotics simulator V-REP and it is concluded that the proposed navigation system improves MFR's error-tolerant capacity and navigation performance.

Laboratory : Computer Engineering
Codenumber : CE-MS-2016-14

Committee Members :

Advisor:	dr. ir. Arjan van Genderen, CE, TU Delft
Advisor:	Rene Beltman, Product Development, Lely
Chairperson:	dr. Msc. Sorin Cotofana, CE, TU Delft
Member:	dr. ir. Andre Bossche, EI, TU Delft

To the memories of my father and grandparents.

To my mother who has shaped me into the man I am today with all that she has.

Contents

List of Figures

List of Tables

List of Acronyms

AMCL Adaptive Monte Carlo Localization

CE Computer Engineering

EI Electronic Instrumentation

EKF Extended Kalman Filter

EEMCS Electrical Engineering, Mathematical and Computer Science

ICC Instantaneous Center of Curvature

IMU Inertial Measurement Unit

SAMCL Stripe-based Adaptive Monte Carlo Localization

MCL Monte Carlo Localization

MFR Mixing and Feeding Robot

ROS Robot Operating System

SLAM Simultaneous Localization and Mapping

Acknowledgements

It is a great pleasure to acknowledge my deepest gratitude to my two supervisors at TU Delft and at Lely, respectively. Thank dr. Arjan van Genderen for his supervision on my process as well as his guide on my thesis report. Thank Rene Beltman for hosting me at Lely and guiding me along the way. Further, thank dr. Sorin Cotofana for being the chairperson and thank dr. Andre Bossche for being a member of my committee. Besides, I am also very grateful to Lely Vector team for all their help.

Without all my friends, I cannot reach my goals. Thank Bella, Yefu, Kang, Chenyun and Laura for their unfailing support. Thank Zheng, Vincent, Nancy, Liang and Zhimian for backing me up during the course of this project. Thanks to all who are not mentioned here but did light up my life.

From the bottom of my heart, a special gratitude goes out to my family, especially my dearest mother, who nurture me and inspire me with unconditional love.

He Cheng
Delft, The Netherlands
October 25, 2016

1 Introduction

Recent years, great success has been made in the field of autonomous robotics, a branch of robotics that studies and designs robots able to perform behaviors or tasks with a high degree of autonomy. To date, autonomous robots have already been widely used in dairy farming that is a labor-intensive industry involving many tasks such as feeding, milking, livestock management, etc. Gradually, these robots appear huge advantages especially in aspects of increasing productivity, improving accuracy, and decreasing labor cost.

Due to the complexity of the farming environment, high intelligence and robustness are required for farming robots to perform tasks in a noisy environment. However, farming robotics is also born with a high requirement for a low cost so that it is affordable to farmers usually with the limited budget. Therefore, instead of purely to develop an advanced robot, the main concern in this field is to develop a good robot with a reasonable cost.

Figure 1.1: Lely Vector

As a leading innovator in agriculture, Lely manufactures machinery for dairy farms, including Lely Astronaut for automatic milking, Lely Vector for automatic feeding, etc. To be more detailed, Lely Vector is able to feed cows accurately and flexibly with minimum labor requirements. As shown in Figure 1.1, this feeding system consists of four parts:

1. **Feed Kitchen:** the area where feed for cows is stored.

2. **Feed Grabber:** the robotic claw that picks different portions of feed and put it into the container of the MFR.
3. **MFR:** the autonomous mobile robot able to automatically mix, distribute and push feed for cows.
4. **Charging Station:** the place where MFR charges its battery with the charging pole and where MFR waits for feed grabber to put feed.

1.1 Context

1.1.1 Lely MFR

This project centers on how MFR navigates in barn environment for distributing or pushing feed, as shown in Figure 1.2. Since cows stay in areas enclosed by fences, MFR should distribute feed along fences so that cows could reach feed just by stretching their heads out of the fences.

Figure 1.2: MFR navigation

For navigation, MFR is equipped with the following sensors (more details in Chapter 3):

- **Two ultrasonic sensors** allows MFR to follow feeding fences at pre-determined distances.
- **Two inductive sensors** are used in two ways. One is to enable MFR to follow metal stripes to a target place. The other is to inform MFR that it has already reached its target and it is time to execute next action when a metal stripe, as a marker, is detected.
- **A gyroscope** is used to measure angular velocity, which mainly enables MFR to turn with a certain angle. Meanwhile, the angular velocity is also a signal to check if wheel slip happens.
- **Two wheel encoders** are used to measure the velocity of two motored wheels, from which MFR's velocity as well as the distance MFR has driven could be estimated.

1.1.2 Sample Barn Environment

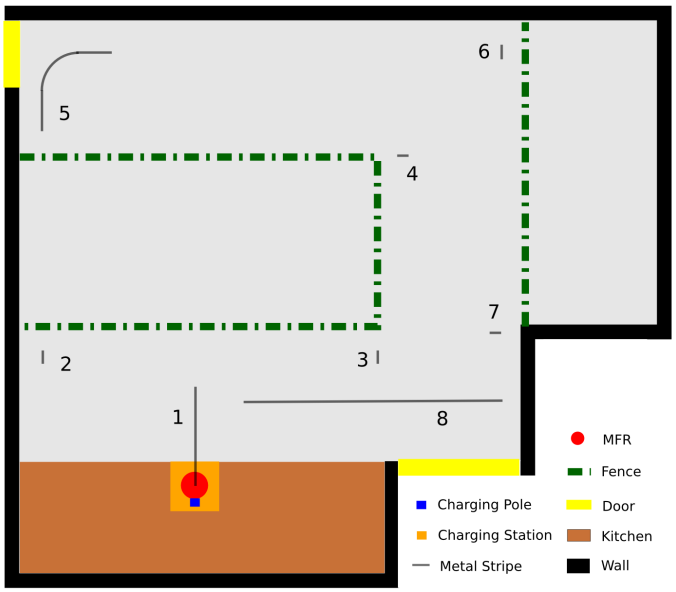


Figure 1.3: Sample barn environment

Since barn environments vary in different farms, it is impossible to discuss them all in this thesis. Instead, a $26 \times 26m^2$ sample barn environment (see Figure 1.3) is set up covering basic features of barn environment. The light gray part is floor; the black parts are walls; the green lines are fences enclosing areas where cows stay; the brown part is feed kitchen; the red dot is MFR; the blue dot is the charging pole; the dark gray lines numbered are metal stripes; the yellow parts are doors.

1.1.3 Feed Distribution Process

After feed is loaded and mixed, MFR autonomously drives through the barn distributing feed for cows. Based on the barn environment in Figure 1.3, the process of feed distribution is summarized as follows:

- Start from the charging station, rotate 180 degrees counterclockwise, and drive forward following the No.1 metal stripe till its end;
- Drive straight forward 1 meter and rotate 90 degrees counterclockwise;
- Follow the fence till No.2 metal stripe is detected by the inductive sensors, and rotate 180 degrees counterclockwise;
- Distribute feed along the fence till No.3 metal stripe is detected, drive straight forward 2 meters, and rotate 90 degrees counterclockwise;
- Distribute feed till No.4 metal stripe is detected, drive forward 2 meters, and rotate 90 degrees counterclockwise;

- Distribute feed along the fence till No.5 metal stripe is detected;
- Follow No.5 metal stripe till its end;
- Follow the wall till No. 6 metal stripe is detected, rotate 90 degrees clockwise;
- Distribute feed along the fence till No.7 metal stripe is detected;
- Follow the wall till No.8 metal stripe is detected;
- Follow No.8 metal stripe till its end;
- Drive straight forward till No.1 metal stripe is detected;
- Follow No.1 metal stripe back to the charging station, connecting to the charging pole.

1.2 Problem Description

Before introducing MFR navigation system, it is necessary to explain the concept of the reactive approach. This approach embeds the agent's control strategy into a collection of pre-programmed condition-action pairs with minimal internal states [3] [4] [5]. Typically they apply a simple functional mapping between stimuli and appropriate responses. In MFR navigation system, these mappings rely on the direct coupling between sensing and action. For example, ultrasonic sensing triggers fence following or wall following action; inductive sensing triggers stripe following or next action. Currently, MFR utilizes a two-level navigation system. In the high level, a route with action sequences is manually planned in advance; in the low level, the reactive approach is employed to take care of every single action. Since barn environment usually does not change much after being built up, such a navigation system has proven effective. However, it still suffers from several problems:

- MFR temporarily loses track of the feeding fence during fence following due to part of the fence is moved by farmers or other reasons. In this case, MFR would stop navigating if it cannot find the fence again.
- MFR loses track of the metal stripe. In this case, MFR would stop and then look for the stripe by turning left and right for a certain degree. If no stripe is found by this way, MFR would just stop there and raise an alarm. In barns, it is common that the floor is dirty. If the error is due to the fact that the stripe is covered by dirty stuff, hardly could MFR recover from that.
- MFR heavily relies on metal stripes. For example, in Figure 1.3, MFR is not able to navigate back without No.8 metal stripe.

1.3 Objectives

Since reactive navigation only responds to the sensing at the moment, hardly could the above problems be handled by the current navigation system. Therefore, here comes map-based navigation that is the combination of three processes [6] [7].

- Mapping, the process of memorizing the data about the environment in a suitable way of representation.
- Localization, the process of deriving the current pose of the robot within the map.
- Path planning, the process of choosing a course of action to reach a goal, given the current pose.

In map-based navigation, the robot is able to know its pose in the environment, which compensates the drawbacks of reactive navigation. Since the peculiarity of barn environment, reactive navigation has its obvious advantage in fence following and strip following, map-based navigation could not fully replace it.
The primary objective of this work is to develop a hybrid navigation system merging reactive navigation and map-based navigation that tolerates the errors mentioned above and improve MFR's navigation performance. The focus is placed on a low-cost solution with minimal changes in hardware, specified in two requirements:

- Make the full of sensors that are currently installed on MFR.
- Avoid dramatically increasing the cost of Lely Vector if new sensors are needed. For example, it is not feasible to install on every MFR an industrial laser scanner that generally costs €5,000 on average.

The research questions that this thesis aims to answer are:

- How could MFR build up a map of the barn environment? Are current sensors enough? If not, what kind of sensors are needed?
- How could MFR localize itself based on the map it keeps? Are current sensors enough? If not, what kind of sensors are needed?
- Could the proposed hybrid navigation system address the above problems?

1.4 Overview

The rest of this thesis is structured as follows. **Chapter 2** presents the background of robot navigation including map building, localization and path planning. **Chapter 3** introduces the configuration of MFR. **Chapter 4** mainly describes how MFR is simulated in the simulator V-REP. **Chapter 5** analyzes the insufficiency of the AMCL algorithm and proposes the SAMCL algorithm for localization. **Chapter 6** describes the design of the hybrid navigation system combining the reactive navigation currently used on MFR and the map-based navigation. **Chapter 7** introduces the setup of simulated experiments and the corresponding results. **Chapter 8** reaches conclusions of this thesis project, clarifies the main contributions, and discusses future work.

2 Background

This chapter presents the related topics about this thesis project, covering recursive state estimation, map-based navigation, the Robot Operating System (ROS) and robotics simulators. Section 2.1 introduces recursive state environment, which is the foundation of probabilistic robotics. Section 2.2 states the concepts and the state of the art of map-based navigation including localization, mapping and path planning. Section 2.3 introduces the structure of ROS, the Navigation Stack, and frames and transforms. Section 2.4 mainly introduces the robotics simulator called V-REP, which is used for simulation in this project.

2.1 Recursive State Estimation

This section presents the recursive state estimation, including basic concepts, Bayes filter, and particle filter, all of which are the elements of probabilistic robotics described in [8].

2.1.1 Robot Environment Interaction

Equipped with sensors, a robot can obtain information about its environment. However, sensors are noisy and usually give relative information. Therefore, the robot maintains a belief of its state. More details are presented below.

Environments are characterized by **state**, which in this project can be considered as the collection of all aspects of the robot that can impact the future. For example, the state variables can be:

- The robot pose, which consists of position and orientation relative to a global coordinate frame. A rigid mobile robot represents its pose by six variables, three Cartesian coordinates (x, y, and z) for its position and Euler angles (pitch, roll, and yaw) for its orientation.
- The robot velocity, which consists of a velocity for each pose variable.

Definition 2.1 *A state is denoted by x and the state at time t is denoted by x_t.*

A robot interacts with its environment in two ways:

- **Sensor measurement** is the process that the robot retrieves information about the state of its environment through its sensors.
- **Control action** is the process that the robot influences the state of its environment through its actuators.

Correspondingly, there are two types of data:

- **Measurement data** carries information about a momentary state of the environment, such as range data, image data, etc.
- **Control data** carries information about the change of state in the environment, such as the velocity command for a robot.

Definition 2.2 *The measurement data at time t is denoted by* z_t. *The notion* $z_{t_1:t_2} = z_{t_1}, z_{t_1+1}, z_{t_1+2}, ..., z_{t_2}$ *denotes the sequence of all measurement data acquired from time* t_1 *to time* t_2, *for* $t_1 \leq t_2$.

Definition 2.3 *The control data at time t is denoted by* u_t. *The notion* $u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, ..., u_{t_2}$ *denotes the sequence of all control data from time* t_1 *to time* t_2, *for* $t_1 \leq t_2$.

The evolution of state is characterized by probabilistic laws. In probabilistic robotics, it is assumed that the robot performs a control action u_t first and then a measurement z_t (see Figure 2.1). The evolution of state in the process of control action can be expressed as a probability distribution:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) \tag{2.1}$$

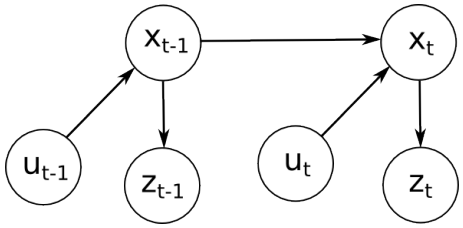


Figure 2.1: Evolution of Bayesian states

Theorem 2.1 *The* ***Markov Assumption*** *postulates that the conditional probability distribution of future states only depends on the present state.*

According to the Markov Assumption, if the state x_t is complete, it is a sufficient summary of all previous control actions $u_{1:t-1}$ and sensor measurements $z_{1:t-1}$. Hence, the expression 2.1 can be simplified as

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.2}$$

where the probability $p(x_t|x_{t-1}, u_t)$ is the **state transition probability** that specifies how the environment state evolves over time as a function of the control action u_t.

Similarly, if x_t is complete, the measurement process can be modeled as

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.3}$$

where the probability $p(z_t|x_t)$ is the **measurement probability** that specifies how the sensor measurement z_t is generated from the environment state x_t.

Another fundamental concept in probabilistic robotics is a **belief** that reflects the robot's knowledge about the environment state.

Definition 2.4 *A belief over a state variable x_t is denoted by $bel(x_t)$.*

A belief distribution is a posterior distribution over the state x_t conditioned on all the previous measurements $z_{1:t}$ and controls $u_{1:t}$. The distribution can be expressed as

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.4}$$

The belief $bel(x_t)$ is calculated after taking in the measurement z_t. It is also very important to calculate the posterior just after performing the control u_t and before z_t, which is expressed as

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.5}$$

This process is called **prediction** since $\overline{bel}(x_t)$ predicts the state x_t based on the previous posterior and the control u_t before taking in the measurement u_t. The process that calculates $bel(x_t)$ from $\overline{bel}(x_t)$ is called **correction**.

2.1.2 Bayes Filter

Bayes filter is the most general algorithm to calculate the belief distribution *bel* from measurements and controls. The pseudo algorithm is depicted in Algorithm 1. First, the predicted belief $\overline{bel}(x_t)$ is calculated by the integral of the product of the previous belief $bel(x_{t-1})$ and the probability that the state x_t is achieved from x_{t-1} by taking in the control u_t. This step is called **prediction**, implemented in Line 3. Second, the final belief $bel(x_t)$ is calculated by the product of normalization constant η, the probability that the measurement z_t is observed at each hypothetical state x_t, and the predicted belief $\overline{bel}(x_t)$. The normalization constant η ensures $bel(x_t)$ to sum up to 1. This step is called **correction**, implemented in Line 4.

Algorithm 1 Bayes Filter, adopted from [8]

1: **Inputs:** $bel(x_{t-1}), u_t, z_t$
2: **for** all x_t **do**
3: $\quad \overline{bel}(x_t) = \int\ p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ \mathrm{d}x$
4: $\quad bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5: **end for**
6: **Output:** $bel(x_t)$

2.2 Map-based Navigation

This section introduces the three processes of the map-based navigation: mapping, localization and path planning [6] [7].

2.2.1 Localization

As the cornerstone of successful navigation, localization is to estimate the robot's pose in the environment. With the initial pose known, the robot can track its pose by a motion model (e.g. odometry model [1]) that accommodates small pose error approximated by a unimodal distribution (e.g Gaussian distribution). Due to the error, the pose estimate is not fully correct and the robot is absolutely uncertain about its pose. Over time, the uncertainty of the robot's absolute pose grows without bound. In order to reduce the uncertainty, a map of the environment is needed. With the map known, the robot can use its sensors to observe the environment around it and match these observations to the map. By this way, the robot can determine its absolute pose in the environment.

To date, the most popular approach for localization is the **Monte Carlo localization** (MCL) [9], which is an algorithm for robots to localize by using a particle filter. The key idea underlying MCL is to represent the posterior belief $bel(x_t)$ by a set of N weighted, random samples or particles $X_t = \{< x_t^{[n]}, \omega_t^{[n]} > \mid n = 1, ..., N\}$. $\omega_t^{[n]}$ is a non-negative numerical weighting factor called **importance factor**, that is, the weight of a particle. The total weight of all particles is 1. In 2D environment, x_t is expressed as

$$< x, y, \theta > \tag{2.6}$$

where $< x, y, \theta >$ denotes a robot pose, x and y are the robot's coordinates in a world-centered Cartesian reference frame, and θ is the robot's orientation, that is, yaw in Euler angles.

The pseudocode of the MCL algorithm is depicted in Algorithm 2, which calculates the sample set X_t recursively from the set X_{t-1}. It inputs a sample set X_{t-1} along with the most recent control u_t, the most recent measurement z_t and the map m. It outputs the sample set X_t, which represents the belief $bel(x_t)$ called the **target distribution**. $\overline{X}_t$ is a temporary sample set, which represents the belief $\overline{bel}(x_t)$ called the **proposal distribution**. The algorithm is realized in four steps:

1. **Prediction** (Line 4). When the robot moves, MCL recursively generates a particle $x_t^{[n]}$ based on the particle x_{t-1}^n and the control u_t.

2. **Correction** (Line 7). In order to do resampling, the importance factor $\omega_t^{[n]}$ is calculated for each particle $x_t^{[n]}$ based on the measurement z_t and the map m (implemented in Line 5). The weighted sample set $\overline{X}_t$ can approximately represent the posterior belief $bel(x_t)$, but is does not distribute according to it yet.

3. **Normalization** (Line 9 $\sim$ 11). The total weight of all particles are normalized to 1.

4. **Resampling** (Line 12 $\sim$ 15). N new particles are drawn randomly from the weighted sample set $\overline{X}_t$ with probability proportional to ${\omega_t}^{[n]}$ $(n = 1, ..., N)$. That

[1] odometry model, available at `http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/06-motion-models.pdf`

is, the more weighted particle are more likely to be drawn (possibly more than once) while the less weighted ones are rarely chosen, which makes particles converge towards a better estimate of the robot's pose. By this way, the temporary particle set $\overline{X}_t$ is transformed into a corrected particle set X_t of the same size.

Algorithm 2 Monte Carlo Localization (adopted from [8])

1: **Inputs:** X_{t-1}, u_t, z_t, m
2: $\overline{X}_t = X_t = \emptyset, \alpha = 0$
3: **for** n = 1 to N **do**
4: $\quad$ *Sample* ${x_t}^{[n]} \sim p(x_t \mid u_t, {x_{t-1}}^{[n]})$
5: $\quad$ ${\omega_t}^{[n]} = p(z_t \mid {x_t}^{[n]}, m)$
6: $\quad$ $\alpha = \alpha + {\omega_t}^{[n]}$
7: $\quad$ $\overline{X}_t = \overline{X}_t + < {x_t}^{[n]}, {\omega_t}^{[n]} >$
8: **end for**
9: **for** n = 1 to N **do**
10: $\quad$ ${\omega_t}^{[n]} = {\omega_t}^{[n]} \ / \ \alpha$
11: **end for**
12: **for** n = 1 to N **do**
13: $\quad$ *draw* $x_t^{[i]}$, $i \in 1, ..., N$ *with probability* $\propto {\omega_t}^{[i]}$
14: $\quad$ $X_t = X_t + < {x_t}^{[i]}, {\omega_t}^{[i]} >$
15: **end for**
16: **Output:** X_t

The AMCL [10] is implemented to increase the efficiency of MCL by using KLD-sampling to dynamically adjust the size of the particle set. That is to say, when the robot is highly uncertain about its pose, AMCL will increase the number of particles; on the contrary, the number of particles will be decreased. In this way, the AMCL makes a trade-off between localization accuracy and computation efficiency. The pseudocode of AMCL algorithm is depicted in Algorithm 3, where the KLD-sampling is mainly implemented in Line 8 $\sim$ 13. The key idea of the KLD-sampling method is to bound the approximation error introduced by the sample-based representation of the particle filter. The number of samples needed to avoid exceeding the bound ε is approximated by [10]

$$n = \frac{1}{2\varepsilon}\chi^2_{k-1,1-\delta} \tag{2.7}$$

where ε is the pre-specified bound; $1 - \delta$ is the possibility that guarantees the bound is not exceeded; k is the number of possible bins, limited by a given bin size Δ; χ^2 is chi-squared distribution [2].

2.2.2 Mapping

Mapping is the process of building a map of a mobile robot's environment based on sensory information. Actually, a map is a list of objects in the environment and their

[2]chi-squared distribution, available at `https://en.wikipedia.org/wiki/Chi-squared_distribution`

Algorithm 3 Adaptive Monte Carlo Localization (adopted from [8])

Inputs: $X_{t-1}, u_t, z_t, m, \varepsilon, \delta, \Delta$
$\overline{X}_t = X_t = \emptyset, n = 0, k = 0, \alpha = 0$
do
  Sample ${x_t}^{[n]} \sim\ p(x_t \mid u_t, {x_{t-1}}^{[n]})$
  ${\omega_t}^{[n]} = p(z_t|\ {x_t}^{[n]}, m)$
  $\alpha = \alpha\ + {\omega_t}^{[n]}$
  $\overline{X}_t = \overline{X}_t + < {x_t}^{[n]}, {\omega_t}^{[n]} >$
  if ${x_t}^{[n]}$ falls into an empty bin b **then**
    $k = k\ + 1$
    $b = \textit{non-empty}$
  end if
  $n = n\ + 1$
while $n < \frac{1}{2\varepsilon}\chi^2_{k-1,1-\delta}$
for n = 1 to N **do**
  ${\omega_t}^{[n]} = {\omega_t}^{[n]}\ /\ \alpha$
end for
for n = 1 to N **do**
  draw $x_t^{[i]},\ i\ \in\ 1, ..., N$ *with probability* $\propto\ {\omega_t}^{[i]}$
  $X_t = X_t + < {x_t}^{[i]}, {\omega_t}^{[i]} >$
end for
Output: X_t

locations. An accurate map facilitates robots to complete complex tasks successfully.

In the field of robotics, the dominant way of map representation is **metric map**, in which the environment is represented as a set of objects with precise geometric positions within an absolute coordinate frame. A metric map is easy to construct, gives precise representation, and is easy for different robots to reuse [1]. However, for a large-scale environment, the metric approach suffers from serious problems caused by memory and time complexity. Recent progress in metric mapping has made it possible to build useful and accurate metric maps of reasonably large-scale environments. Successful metric mapping examples can be found in [11][12], whose environment sizes are larger than $200m \times 200m$. In comparison, the scale of a barn environment is much smaller, for example, $100m \times 100m$. Therefore, the metric approach is fully suitable for representing barn environments.

Within metric representation, **occupancy grid mapping** algorithm [13][14][15] enjoys huge popularity. In a grid map (see Figure 2.2), the environment is represented by a block of cells and each cell is assigned a probability of being occupied.

In the process of mapping an unknown environment, good localization is essential for constructing an accurate map while a good map is a precondition for accurate localization [16][17]. This interdependence makes map building difficult because errors

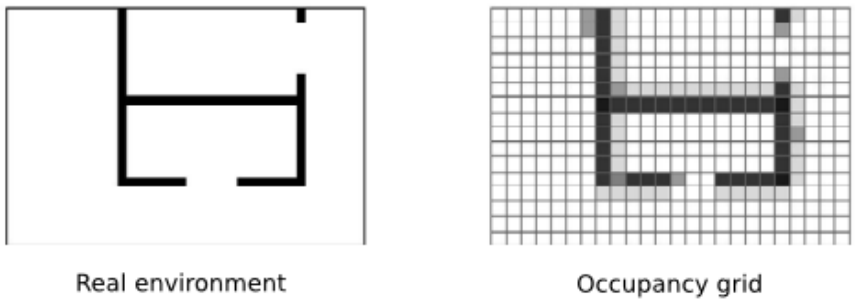


Figure 2.2: Example of occupancy grid map [1]

in localization arising during mapping are incorporated into the map and subsequently need to be detected and corrected. Therefore, the mapping process is often referred to **Simultaneous Localization and Mapping** (SLAM) that is intensively researched in the field of robotics. A survey of SLAM algorithms can be found in [18]. The OpenSLAM community [19] also provides an access to mainstream SLAM algorithms like CEKF-SLAM, DP-SLAM, GMapping, etc.

Rao-Blackwellized particle filters [20] have been introduced as an effective method to solve SLAM problem. Its key idea is to estimate a posterior $p(x_{1:t}|z_{1:t}, u_{0:t})$ about the potential poses $x_{1:t}$ of the robot given its control $u_{0:t}$ (e.g. odometry information) and measurement $z_{1:t}$, which is then used to compute a posterior over maps and poses:

$$p(x_{1:t}, m|z_{1:t}, u_{0:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{0:t}) \tag{2.8}$$

As a highly efficient Rao-Blackwellized particle filter, the Gmapping [11][12] substantially improves the performance of Rao-Blackwellized particle filters by applying two approaches described in [21]:

- The proposal distribution is computed highly accurately by evaluating the likelihood around the most likely pose obtained by a scan-matching procedure combined with odometry information. The scan-matching procedure is to match the current observation against the map built so far with the purpose of finding the most likely pose.
- The adaptive resampling strategy is applied that a resampling step is only taken when needed, which enables the algorithm to keep a reasonable particle diversity and therefore significantly reduces the risk of particle depletion [22].

Based on the above considerations, GMapping will be used to build maps for barn environments in this project.

2.2.3 Path planning

Path planning is a natural extension of localization that enables a robot to find the optimal and collision-free route from the start position to the goal position

through an environment with obstacles. Path planning requires a map of the environment and the robot's awareness of its location on the map that both are discussed above.

Classically, the environment of the robot is represented as a graph $S = (S, E)$, where S is the set of possible robot localizations and E is the set of edges representing paths between these locations. The cost of each edge represents the cost of taking the path. Hence, planning a path can be modeled as searching least-cost paths on this kind of weighted graph. Two most popular graph search algorithms are Dijkstra's algorithm [23] and A* algorithm [24]. Dijkstra's algorithm is able to more accurately produce the shortest path to the goal while consumes more computing power. A* algorithm typically has a better performance than Dijkstra's algorithm, which, however, is very likely to result in less optimal paths.

The move_base module (see Section 2.3.2) has already implemented popular path planning algorithms and will be directly applied in this project for path planning. Hence, details will not be touched in this thesis report. If interested, more information can be found in [25].

2.3 ROS

ROS, developed by Stanford Artificial Intelligence Laboratory and Willow Garage, is an open-source meta-operating system for robots since it provides a structured communication layer based on a host operating system that can only be a Unix-based system currently. It provides services such as low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides rich tools and libraries for software development across multiple computers. Its aim is to provide a flexible and general framework to simplify the development of robot software.

2.3.1 General Concepts

An ROS system is a peer-to-peer network of small, mostly independent processes called **nodes** that run at the same time. Usually, each node is responsible for a specific computation or task. For example, one node performs map building, one node performs localization, one node performs path planning, one node controls velocity, etc. To make the system work, these nodes should be able to communicate with each other. Therefore, a manager of the system is needed enabling nodes to find and communicate with one another. Such a manager is called the **ROS Master** that provides registration services and lookup information. Nodes register their information with the Master, and then receive information about other registered nodes and establish connections with the nodes they are interested in.

The communication among nodes is realized by passing messages. A **message** is a simple data structure. ROS provides standard message data types such as integer, float, twist, imu, odometry, etc. Meanwhile, ROS developers can also define your own custom

message data types.

ROS provides two communication mechanisms for nodes. The more common one is the **publish/subscribe model** shown in Figure 2.3. In this model, messages are passed via ROS transport system called topics. A topic is a name used to identify the content of the message. If a node wants to send a certain kind of data, it will publish messages to the proper topic or topics; if a node wants to receive a certain kind of data, it will subscribe the data to the topic or topics. Generally, multiple nodes can publish or subscribe to one topic; one node can also publish or subscribe to multiple topics.

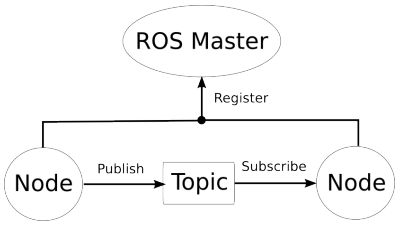


Figure 2.3: Publish/Subscribe model

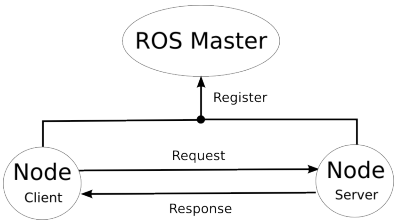


Figure 2.4: Request/Response model

The publish/subscribe model is very flexible and can handle most cases. However, as a many-to-many transport, it is not suitable for a distributed system that often requires **request/response** interaction. To address this problem, ROS provides the other communication mechanism named services shown in Figure 2.4. A service is defined by a pair of messages, one for the request and the other for the response. The node that sends a request is called **client**, and the one that response the request is called **server**.

The above introduction is mainly about the ROS Computation Graph level, which, basically, is enough to understand this project. More information about the ROS Filesystem Level and the ROS Community Level can be found in [26][27][28][29][30].

2.3.2 Navigation Stack

The **Navigation Stack**[3] is a collection of packages for autonomous navigation of robot including mapping, localization and path planning. Considering it as a whole, it inputs data from odometry, sensors and map, and outputs velocity commands for a mobile robot base.

The map of the environment will be built with the package **gmapping**[4], which implements the mapping algorithm in [11][12]. If a map is available, it can be provided for the Navigation Stack via **map_server** [5], which offers map data as a ROS Service.

[3]Navigation Stack, available at `http://wiki.ros.org/navigation`
[4]gmapping, available at `http://wiki.ros.org/gmapping`
[5]map_server, available at `http://wiki.ros.org/map_server`

The Navigation Stack utilizes the package **amcl** that implements AMCL algorithm described in [10]. This package can localize a robot in 2D environment against a known map by taking in laser scans, and transform messages from the odometry frame to the robot base frame.

The path planning module is called **move_base** [6], which mainly consists of a global planner and a local planner. The global planner has the task of building a path from the staring pose to the goal pose over the entire, static map of the environment. On the other hand, the local planner, based on the environment surrounding the robot at the moment, decides which action to choose from a set of possible actions in order to fulfill the global plan. Besides, the environment is represented as a grid map, where each cell has a cost that is higher in correspondence of obstacles and lower in free space. As mentioned in Section 2.2.3, cost is used for planning paths and the planner tends to move in least-cost cells, which enables the robot to avoid obstacles.

2.3.3 Frames and Transforms

In the field of robotics, the location of objects in three-dimensional space is constantly concerned. In order to describe the pose of an object in space, a coordinate frame is attached rigidly to the object. Then, the pose of this frame is described relative to some reference frames. Any frame can serve as a reference frame. Respect to the reference frame, the pose of an object is actually the transform from the frame attached rigidly to the object itself to the reference frame. For example, Figure 2.5 shows typical frames for mobile robots following the convention of **REP105 - Coordinate Frames for Mobile Platforms** [7]. The frame called **map** is a world fixed frame that works as a long-term global reference since a robot's pose in this frame will not drift over time. By default, the pose of a mobile robot is relative to the map frame. The frame called **odom** is a world-fixed frame that is useless as a long-term reference but useful as an accurate, short-term local reference, since a robot's pose relative to this frame is guaranteed to be continuous but will drift over time. The odom frame is typically used for an odometry source. The frame rigidly attached to the robot base is called **base_link**. Referring to the map frame, the robot's pose can be obtained by calculating the transform from the map frame to the base_link frame, which actually is the map→odom transform plus the odom→base_link transform. In ROS, a package named **tf** [8] is provided for keeping track of multiple coordinate frames over time. More information can also be found in [31]. Besides REP 105, **REP 103 - Standard Units of Measure and Coordinate Conventions** [9] is also an important guide to coordinate frame conventions. To better integrate and reuse open resources, this project follows REP 105 and REP 103.

[6] move_base, available `http://wiki.ros.org/move_base`
[7] REP 105, available at `http://www.ros.org/reps/rep-0105.html`
[8] tf, available at `http://wiki.ros.org/tf`
[9] REP 103, available at `http://www.ros.org/reps/rep-0103.html`

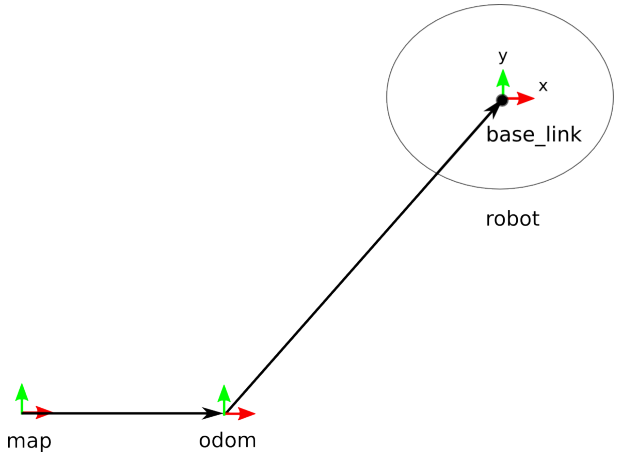


Figure 2.5: Illustration of frames and transforms

2.4 Robotics Simulators

Over the last few years, robotics simulators have grown with the boom of the field of robotics. Even though the scenarios a robot may encounter in the real world is more than that can be simulated, current simulators provide many and increasingly more features that largely facilitates simulation and make simulation closer to the real world. Creating a virtual model of a robot by simulators can significantly shorten the development life cycle of a project and also reduce cost.

Currently, there are many powerful robotics simulators, such as V-REP[10], Gazebo[11], Webots[12], etc. More information about simulators can also be found in [32]. However, not all of them are suitable for simulating MFR. The suitable simulator should fulfill the following requirements:

- The simulator is compatible with ROS
- The simulator can simulate variety of hardware, like sensors and actuators
- The simulator has strong physics engines

Developed by Coppelia Robotics, **V-REP** is an open-source 3D robot simulator and free for academic use. V-REP is becoming more and more popular in the robotics community due to its obvious advantages. It supports physics engines like ODE, Bullet, Vortex and Newton [13]. Meanwhile, it has a rich built-in library of sensors and actuators. In addition, it offers many possibilities to interface with the external world. The plugin **vrep_ros_bridge**[14] provides a communication interface between V-REP and ROS, which

[10] V-REP, available at `http://www.coppeliarobotics.com/`
[11] Gazebo, available at `http://gazebosim.org/`
[12] Webots, available at `https://www.cyberbotics.com/index`
[13] V-REP's dynamics module, available at `http://www.coppeliarobotics.com/helpFiles/en/dynamicsModule.htm`
[14] vrep_ros_bridge, available at `http://wiki.ros.org/vrep_ros_bridge`

enables ROS to control V-REP simulation externally. Therefore, V-REP is chosen as the simulator for this project.

3 Robot Configuration

This chapter mainly presents the configuration of MFR as a reference basis for the simulation model described in Chapter 4. Section 3.1 lists MFR's basic specifications. In Section 3.2 MFR's drive system is introduced. After that, Section 3.3 introduces the sensors that are currently used on MFR or should be added for the proposed navigation system.

3.1 Basic Specifications

Table 3.1 lists MFR's basic specifications, including size, weight and speed.

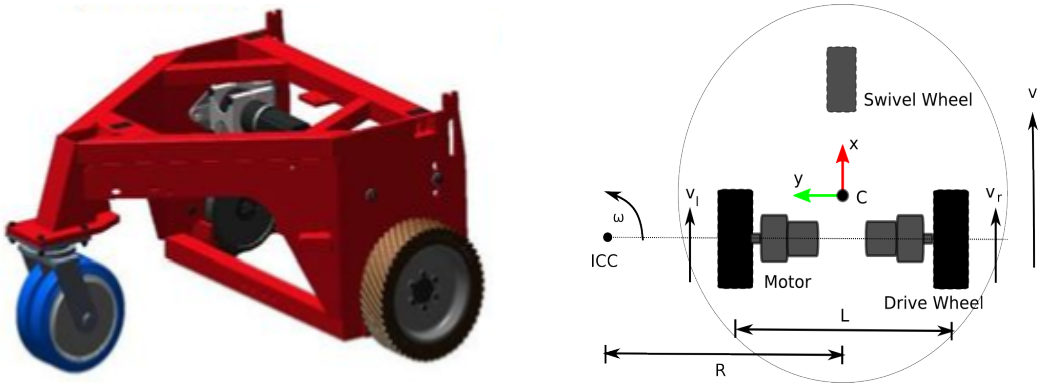
Table 3.1: Basic specifications

Dimensions	
Length	227 cm
Width	160 cm
Height	190 cm
Weight	1485 kg
Requirements	
Max Speed	0.5 m/s
Average Speed	0.25 m/s

3.2 Drive System

MFR utilizes the differential drive mechanism (see Figure 3.1a) that consist of two rear drive wheels mounted on a common axis with independent motors. Two drive wheels can independently rotate forward or backward. To keep MFR's balance, a front swivel wheel is added. The swivel wheel is free to turn and not controllable. To be noted, if the robot reverses its direction, the swivel wheel can cause an undesired motion since the swivel must turn 180 degrees. Therefore, this is not allowed in this project.

As shown in Figure 3.1b, v_l denotes the velocity of the left drive wheel; v_r denotes the velocity of the right drive wheel; v denotes MFR's linear velocity; ω denotes MFR's angular velocity; L is the distance between the two drive wheels; Instantaneous Center of Curvature (ICC) is the point that the robot rotates about when two drive wheels are rotating at different speeds; R is the signed distance from the ICC to the midpoint



(a) Configuration (b) Kinematics

Figure 3.1: Drive system

between the two drive wheels. v, ω and R can be calculated by:

$$v = \frac{v_l + v_r}{2} \tag{3.1}$$

$$\omega = \frac{v_r - v_l}{L} \tag{3.2}$$

$$R = \frac{L}{2}\frac{v_r + v_l}{v_r - v_l} \tag{3.3}$$

Table 3.2 lists the specifications of the drive wheels and Table 3.3 presents those of the swivel wheel. To make it clear, a coordinate frame is attached to point C that is the center of MFR on the bottom, marked as $(0, 0, 0)$. The z-axis is pointing upward. The positions of all components listed in this chapter are relative to this frame.

Table 3.2: Specifications of the drive wheels

Size (mm)	
Diameter	300
Width	90
Positions (mm)	
x-axis	± 460
y-axis	-260
z-axis	150

3.3 Sensors

Sensors are the key components for every robot to observe the environment and estimate its state. Currently, MFR is equipped with two ultrasonic sensors, two inductive sensors, a gyroscope and two wheel encoders for reactive navigation. In this project, another two kinds of sensors are needed, an accelerometer and a laser scanner.

Table 3.3: Specifications of the swivel wheel

Size (mm)	
Diameter	250
Width	120
Position (mm)	
x-axis	0
y-axis	450
z-axis	125

3.3.1 Ultrasonic Sensors

An ultrasonic sensor (see Figure 3.2) emits a cone-shaped ultrasonic beam that reflects off of objects within the wave field. The echo is then received by the sensor. An ultrasonic sensor has a blind zone within which it cannot receive the echo accurately. The minimum distance is the outer edge of the blind zone. The range between the minimum and the maximum sensing distances is the valid sensing range. To be noted, the opening angle of beam cone varies with different distances at which objects are detected.

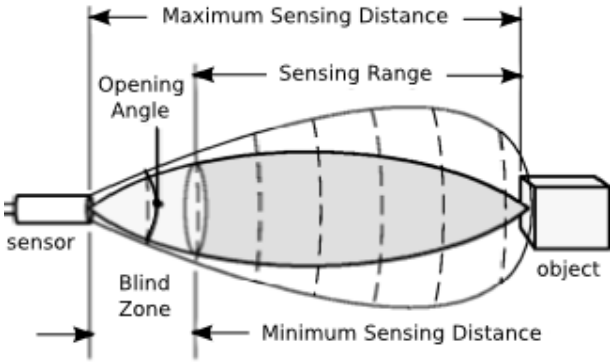


Figure 3.2: Illustration of ultrasonic sensing (modified from [2])

PIL P42 ultrasonic sensor is currently used on MFR. When tested with a horizontal iron pole ∅40 millimeters, the minimum sensing distance is about 0.27 centimeters and the maximum is 319 centimeters. Table 3.4 lists the opening angles of the cone-shaped beam with different distances at which the iron pole is detected.

Table 3.4: PIL P42 performance

Distance (cm)	50	100	150	200	250	300	350	400
Opening angle (°)	18	12	8	6.5	4.5	1.5	-	-

Two such ultrasonic sensors are installed on the red iron shelf in the front of MFR, as shown in Figure 3.3. One is oriented to the right side, and the other is oriented to the left,

which ensures MFR to follow the feeding fence at the pre-determined distances.Table 3.5 lists the positions of two ultrasonic sensors. Since the positions of ultrasonic sensors can be adjusted accordingly along the iron shelf vertically, the values along the z-axis are not specified here.

Figure 3.3: Positions of ultrasonic sensors

Table 3.5: Ultrasonic sensor positions

axis	value (mm)
x	± 250
y	1310

3.3.2 Inductive Sensors

Figure 3.4: Inductive sensor

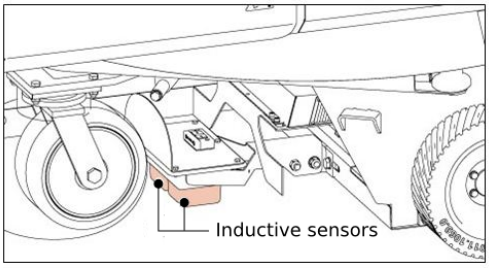


Figure 3.5: Positions of inductive sensors

An inductive sensor (see Figure 3.4) is a type of non-contact sensor used to detect metal objects. An inductive sensor has an oscillator generating a fluctuating magnetic field around the sensing face, the blue part in Figure 3.4. When a metallic object moves into the inductive sensor's field of detection, the inductive sensor's own magnetic field will

be weakened, which causes a change in the output voltage of the sensor.

As shown in Figure 3.5, two inductive sensors are installed on the bottom of MFR for following metal stripes or detecting metal stripe markers. Table 3.6 lists the positions of the inductive sensors.

Table 3.6: Inductive Sensor Positions

axis	value(mm)
x	± 50
y	225
z	40

The relation between the output voltage and the distance between the metal stripe and inductive sensors is shown in Figure 3.6, where the blue part is the output of the left inductive sensor and the red is that of the right.

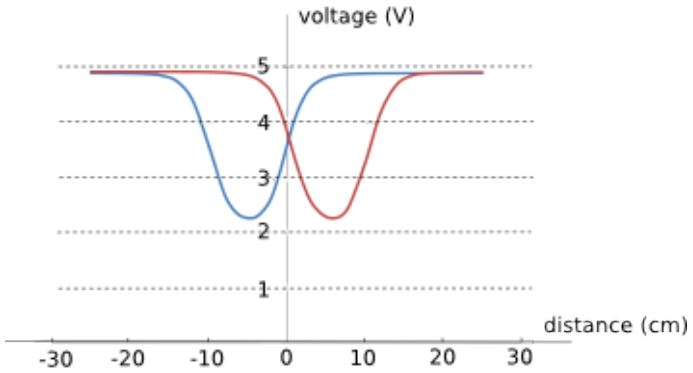


Figure 3.6: Relation between output voltage and distance

3.3.3 IMU

As the necessary device for map-based navigation, an inertial measurement unit (IMU) is the combination of gyroscopes and accelerometers. A gyroscope is a device used to measure angular velocity. An accelerometer is used to measure accelerations.

MFR has a gyroscope with the rate noise density of 0.015 $^{\circ}/sec/\sqrt{Hz}$ but lacks an accelerometer. For this project, an accelerometer is added in the simulation model described in Chapter 4.

3.3.4 Wheel Encoders

A wheel encoder is used to measure the amount of rotation completed by the wheel in the form of ticks, from which odometry information is retrieved. MFR is equipped with

two wheel encoders that are integrated with motors RECM377/4, one for each drive wheel. Between two timestamps, the average velocity is given by

$$v_{avg} = \frac{\Delta_{ticks}}{\Delta_t \cdot res} \cdot circ \tag{3.4}$$

where Δ_{ticks} is the difference between the total number of ticks revealed by two subsequent readings, Δ_t is the difference between the timestamps of the two detections, *res* is resolution that is the number of ticks per turn, and *circ* is the wheel circumference.

3.3.5 Laser Scanner

Currently, MFR does not have a laser scanner but it is necessary for mapping, the reason of which is explained in Section 6.4. A laser scanner offers a long-range and high-accuracy range measurement with a wide scan angle. The scanning is achieved by an embedded rotating mirror that changes the direction of the laser beam with a certain angle increment and a certain frequency. Since a laser scanner provides a large amount of range data, it is widely used in navigation despite its high price.

4 Simulation Model

This chapter describes the simulation model of MFR. Section 4.1 introduces how to import MFR's body part into V-REP. In Section 4.2, the simulation of MFR's drive system is presented. Then the simulation of sensors is described in Section 4.3. After that, Section 4.4 presents the simulation of the charger detector and the charging pole.

4.1 Body Part Importation

V-REP is only able to import some file formats that describe objects as triangular meshes, such as OBJ, DXF, STL, etc. However, the available CAD file format is STEP that is not a triangular mesh format. Therefore, the file should be converted to an appropriate triangular mesh format first. In addition, the complexity of the converted CAD model is still so high that it consumes a large amount of computing power, which would make the simulation run extremely slow. Therefore, before importing the model into V-REP, its complexity is reduced to less than 10,000 faces by the mesh decimation approach in MeshLab [1]. The finished body part of MFR is shown in Figure 4.1.

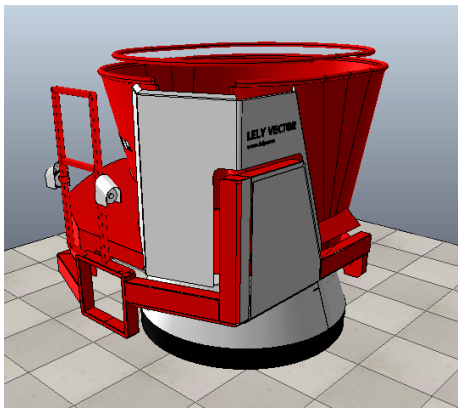


Figure 4.1: Simulated MFR body

4.2 Drive System Simulation

The simulated drive system is shown in Figure 4.2. V-REP provides joints [2] that can be used to build mechanisms. The two motors that control the drive wheels are simulated by

[1] MeshLab, available at `http://meshlab.sourceforge.net/`

[2] Joint types and operation, available at `http://www.coppeliarobotics.com/helpFiles/en/jointDescription.htm`

two revolute joints ($joint_3$ and $joint_4$) under *Torque or force mode* with the joint motors enabled. The joints can be controlled to rotate about the $axis_3$ with given velocities that can be set in the script file [3]. In this project, control algorithms are implemented in ROS nodes that publish velocity commands for the motors. Correspondingly, V-REP subscribes these commands in the form of twist messages and actuates the motors.

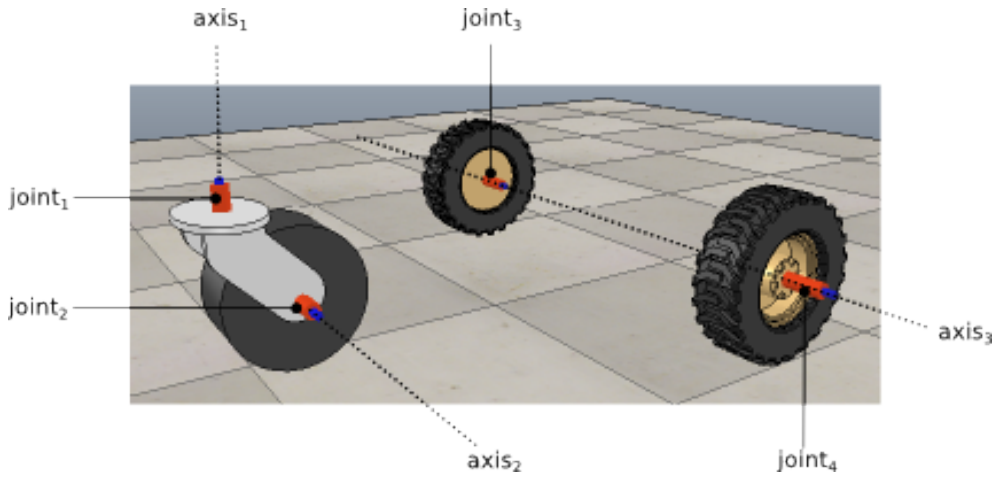


Figure 4.2: Simulated drive system

The mechanism of the swivel wheel is mainly simulated by two revolute joints ($joint_1$ and $joint_2$) also under *Torque or force mode* but with the joint motors disabled. By this way, $joint_1$ is free to rotate about the $axis_1$ and $joint_2$ is free to rotate about the $axis_2$.

4.3 Sensor Simulation

4.3.1 Ultrasonic Sensor Simulation

Proximity sensors are the sensors that detect the presence of objects without physical contact. V-REP provides a built-in proximity sensor model that is able to simulate five subtypes: ultrasonic, infrared, laser, inductive and capacitive [4].

An ultrasonic sensor can be simulated the cone-shaped ultrasonic proximity sensor. According to the performance of PIL P42 sensor mentioned in Section 3.3.1, the minimum sensing distance is set to 0.27 m and the maximum is 2.5 m; the opening angle is set to 10 degrees. The ultrasonic data is published on topics "/ultrasonic_distance/left" and "/ultrasonic_distance/right" accordingly.

[3] V-REP embedded scripts, available at `http://www.coppeliarobotics.com/helpFiles/en/scripts.htm`

[4] V-REP proximity sensors, available at `http://www.coppeliarobotics.com/helpFiles/en/proximitySensors.htm`

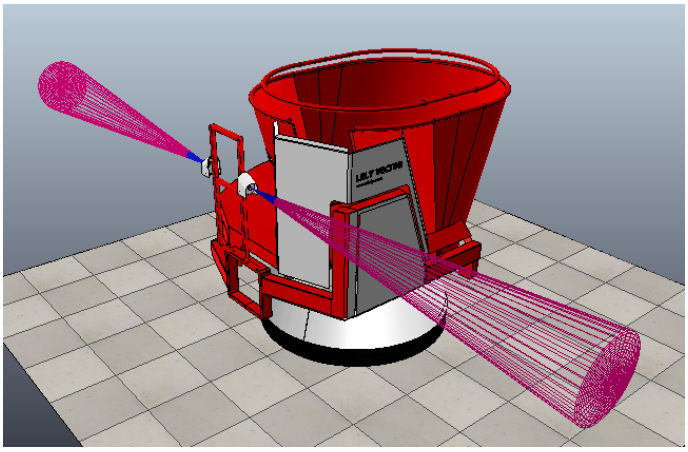


Figure 4.3: Simulated ultrasonic sensors

4.3.2 Inductive Sensor Simulation

Similarly, an inductive sensor could be simulated by the cylinder-shaped inductive proximity sensor. The radius is set to 0.045 m, and the radius far is set to 0.08 m (see Figure 4.4). The finished model is shown in Figure 4.5.

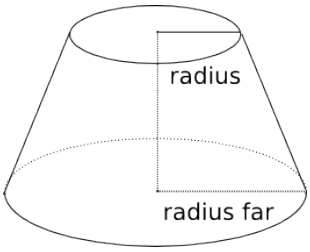


Figure 4.4: Illustration of radius and radius far

Since the output of proximity sensors in V-REP is distance, the simulated inductive sensors will also output distance for convenience. The inductive data is published on topics "/inductive_distance/left" and "/inductive_distance/right" accordingly. If the mathematical expression of the voltage-distance relation in Figure 3.6 is known, it is also easy to do the conversion from distance to voltage.

4.3.3 IMU Simulation

V-REP provides a gyroscope model and an accelerometer model, which can be combined to simulate an IMU. In the simulation, the IMU frame is placed to coincide with the robot base frame.

In ROS, IMU data is published or subscribed in the from of IMU message [5]. However,

[5]IMU message, available at `http://docs.ros.org/api/sensor_msgs/html/msg/Imu.html`

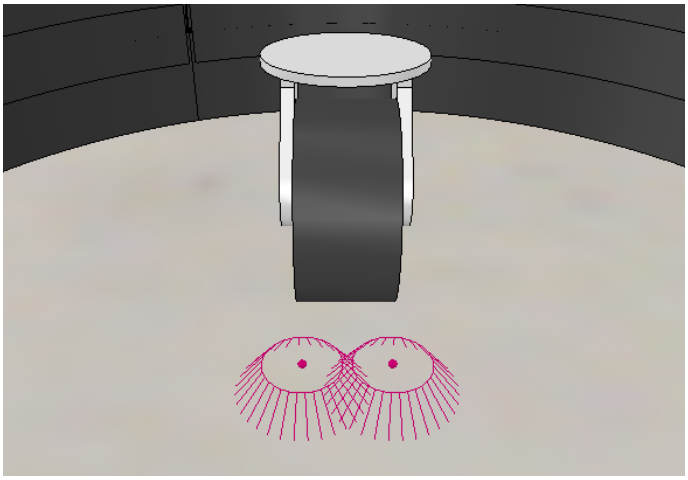


Figure 4.5: Simulated inductive Sensors

there is no publisher type available in V-REP for publishing IMU message. Hence, a new publisher type called **simros_strmcmd_get_imu_state** is created and the IMU data is published on topic "/imu/data_raw".

4.3.4 Wheel Encoder Simulation

V-REP does not provide an wheel encoder model, but we can generates encoder ticks based on joint position values by

$$ticks = \frac{cur\_pos - prev\_pos}{resolution} \tag{4.1}$$

where *cur_pos* means the current joint position, *prev_pos* means the joint position of the previous time step, *resolution* means the amount of radian per tick. The encoder data is published on topics "/encoder_ticks/left" and "/encoder_ticks/right" accordingly.

4.3.5 Laser Scanner Simulation

V-REP provides a laser scanner model that can be directly used. To be noted, the laser scan data is published on topic "/scan".

4.4 Charger Detector and Charging Pole Simulation

When MFR drives back to the charging pole, it should stop immediately if connected to the charger. In the simulation, an inductive sensor (see Figure 4.6) is used as the detector that tells if MFR is connected to the charger. Meanwhile, the interface of the charger (see Figure 4.7) should be made detectable only for inductive sensors. When the distance returned by the detector is smaller than a predetermined value, it means MFR is connected to the charger.

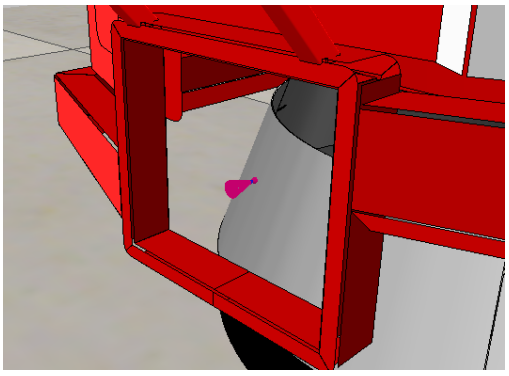


Figure 4.6: Simulated charger detector

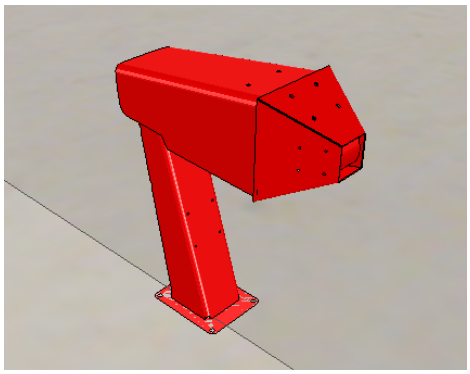


Figure 4.7: Simulated charging Pole

5 SAMCL Algorithm

This chapter proposes the SAMCL algorithm. Section 5.1 first clarifies the default reference frame used in this chapter. Section 5.2 introduces how to fake a laser scanner using the two ultrasonic sensors already installed on MFR. Then Section 5.3 explains the reason why the AMCL algorithm is not sufficient for MFR's localization. After that, Section 5.3 presents the description of the SAMCL algorithm.

5.1 Reference Frame

The robot frame named base_link is attached to the center of MFR base, with the z-axis pointing upward, as shown in Figure 5.1. Unless otherwise stated, the x-axis and the y-axis mentioned in this chapter are silently referred to the base_link frame. Hence, the positive direction of the x-axis represents MFR's forward direction and the positive direction of the y-axis represents MFR's leftward direction.

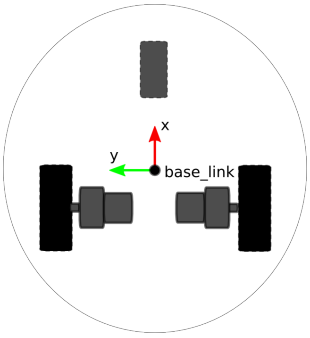


Figure 5.1: Reference frame

5.2 Laser Scanner Faking

As mentioned in Section 2.3.2, the amcl package takes in n pieces of range data from a laser scanner with a given angle increment Δ within a certain scan angle θ, for example, 180 degrees (see Figure 5.2). The number n can be given as

$$n = \frac{\theta}{\Delta} + 1 \tag{5.1}$$

However, MFR does not have a laser scanner. To solve this problem, the range data from the two ultrasonic sensors can be used to fake a laser scanner despite the relatively low accuracy and the relatively short detection range (see Figure 5.3). One ultrasonic sensor

can be modeled as one laser beam at a certain direction. To be noted, the two ultrasonic sensors are not installed at the same point but with a certain distance d. The coordinate frame for the faked laser scanner, named **base_scan**, is attached on the center point O. The faked laser scan data can be given as

$$L_1 = \begin{cases} D_r + \frac{d}{2}, & D_r \ is \ valid \\ 0, & D_r \ is \ invalid \end{cases} \tag{5.2}$$

$$L_i = \ 0, \qquad i \in 2, ..., n-1 \tag{5.3}$$

$$L_n = \begin{cases} D_l + \frac{d}{2}, & D_l \ is \ valid \\ 0, & D_l \ is \ invalid \end{cases} \tag{5.4}$$

where D_r and D_l represents the range data of the right ultrasonic sensor and the left respectively; $L_i, i \in 1, ..., n$ represents laser scan data.

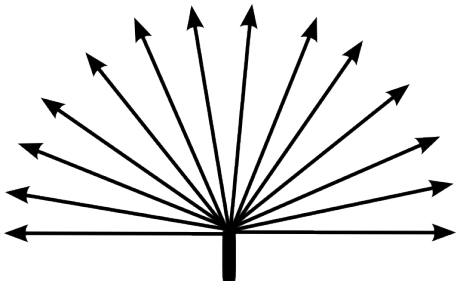


Figure 5.2: Illustration of a laser scanner

By this way, the faked laser scanner has two pieces of range data at most (the two solid arrowed lines in Figure 5.3); other range data is always zero (the dotted arrowed lines in Figure 5.3), which represents invalid measurements.

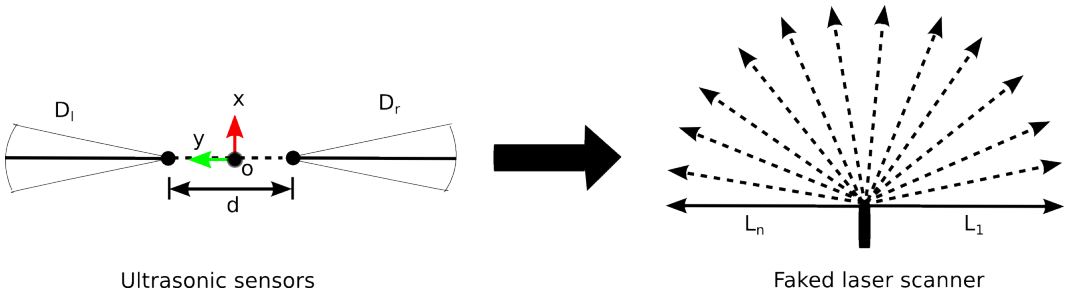


Figure 5.3: Illustration of faking a laser scanner from two ultrasonic sensors

5.3 Insufficiency of the AMCL Algorithm

As mentioned in Chapter 2, the AMCL algorithm consists of prediction step, correction step, normalization step, and resampling step. In the prediction step, the robot estimates

its pose by odometry; in the correction step, the laser scan data is used to reduce the uncertainty of pose estimate. Since the real laser scanner provides range data from all directions within its wide scan angle, the robot is able to correct the predicted pose along both the x-axis and the y-axis. However, the faked laser scanner only has two pieces of valid data that can only reduce the uncertainty of MFR's pose along the y-axis if fences or walls are detected. Therefore, the AMCL using the faked laser scanner cannot help correct the pose error along the x-axis. That is to say, MFR's pose estimate along the x-axis only relies on odometry-based prediction. However, the error in odometry increases over time without bound. For example, when MFR drives along the fence, the uncertainty grows, as shown in Figure 5.4. For a short distance, the uncertainty might stay within a reasonable range; however, for a long distance, the uncertainty might grow to several meters till the navigation fails. Therefore, the AMCL algorithm is not sufficient for MFR to localize itself in the environment.

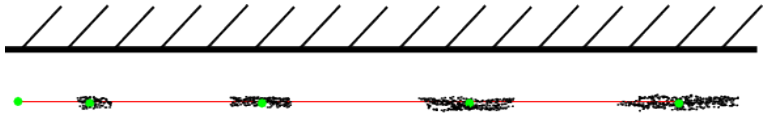


Figure 5.4: Illustration of AMCL using the faked laser scanner

5.4 The SAMCL Algorithm

5.4.1 Set-up of Calibration Metal Stripes

According to the analysis in Section 5.3, there should be sensor readings to reduce the uncertainty along the x-axis. Among the sensors MFR already has, besides the ultrasonic sensors, only the inductive sensors on MFR's bottom are capable of observing the environment in the way of detecting metal stripes on the floor. Hence, the natural idea is to set up metal stripes for pose calibration, named **stripe markers** to avoid confusion. The pose of every calibration marker in the environment is pre-stored in a YAML file in the form of

$$<< x, y, z >, < a, b, c, d >> \tag{5.5}$$

where $< x, y, z >$ represents the position of a random point on the marker by Cartesian coordinates, $< a, b, c, d >$ represents the orientation of the marker by a quaternion. Since a marker is always placed on the floor, the position value z is always zero and so are the orientation values a and b.

To make it easier to explain, the pose of a marker is typed as

$$<< x, y >, \theta >, \ \theta \in [-\frac{\pi}{2}, \frac{\pi}{2}) \tag{5.6}$$

where θ is the Euler angle Yaw. The conversion between Euler angles and Quaternion

can be easily done, by hand [1] or by online calculator [2].

A stripe marker can be modeled as a straight line in the global coordinate, as shown in Figure 5.5. If $\theta_0 \neq -\frac{\pi}{2}$ or 0, the equation of the marker $<< x_0, y_0 >, \theta_0 >$ can be given as

$$y = kx + b \tag{5.7}$$

where the slope $k = \tan\theta_0$, the y-intercept $b = y_0 - kx_0$. If $\theta_0 = -\frac{\pi}{2}$, the equation is given as

$$x = x_0 \tag{5.8}$$

If $\theta_0 = 0$, the equation is given as

$$y = y_0 \tag{5.9}$$

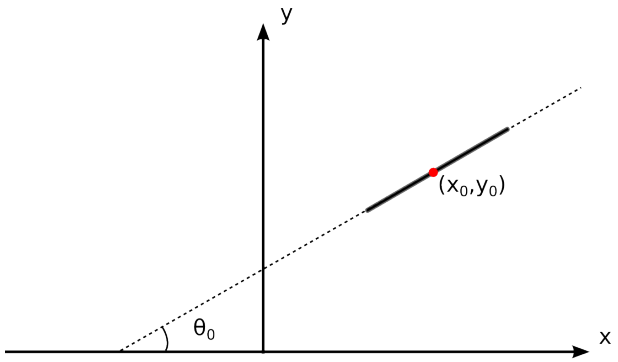


Figure 5.5: Illustration of a stripe marker in the global frame

5.4.2 Description of the SAMCL Algorithm

Now the SAMCL algorithm is proposed for localizing MFR in the barn environment, depicted in Algorithm 4. $S = \{s_j \mid j = 1, ..., M\}$ is the set of stripe markers. *robot_pose* is the estimated pose in the latest update. h is the pre-specified distance threshold to determine if the robot is close enough to a stripe marker. In this algorithm, Line $4 \sim 23$ is the same to the AMCL algorithm; Line $24 \sim 33$ is added for resampling particles if a stripe marker is detected, which is implemented in three steps: **determining the stripe marker**, **calculating the calibration line**, and **resampling**.

1. Determining the Stripe Marker

When at least one inductive sensor detects a metal stripe, the algorithm will check which stripe marker it is by calculating the distance d between *robot_pose* and each marker pose $s_j (j \in 1, ..., M)$; if d is smaller than the threshold h, then s_j is the detected marker.

[1] Conversion equations, available at http://www.chrobotics.com/library/understanding-quaternions
[2] Online calculator, available at http://quaternions.online/

Algorithm 4 SAMCL

Inputs: $X_{t-1}, u_t, z_t, m, \varepsilon, \delta, \Delta, S, h, robot\_pose$
$\overline{X}_t = X_t = \emptyset, n = 0, k = 0, \alpha = 0$
do
  Sample ${x_t}^{[n]} \sim p(x_t \mid u_t, {x_{t-1}}^{[n]})$
  ${\omega_t}^{[n]} = p(z_t \mid {x_t}^{[n]}, m)$
  $\alpha = \alpha + {\omega_t}^{[n]}$
  $\overline{X}_t = \overline{X}_t + < {x_t}^{[n]}, {\omega_t}^{[n]} >$
  if ${x_t}^{[n]}$ falls into an empty bin b **then**
    $k = k + 1$
    $b = non\text{-}empty$
  end if
  $n = n + 1$
while $n < \frac{1}{2\varepsilon}\chi^2_{k-1,1-\delta}$
for n = 1 to N **do**
  ${\omega_t}^{[n]} = {\omega_t}^{[n]} \ / \ \alpha$
end for
for n = 1 to N **do**
  draw $x_t^{[i]}$, $i \in 1, ..., N$ *with probability* $\propto {\omega_t}^{[i]}$
  $X_t = X_t + < {x_t}^{[i]}, {\omega_t}^{[i]} >$
end for
if at least one inductive sensor gives valid data **then**
  for j = 1 to M **do**
    $d = distance(robot\_pose, s_j)$
    if d < h **then**
      calculate the equation of the calibration line
      for n = 1 to N **do**
        relocate $x_t^{[n]}$ *to its corresponding foot point on the calibration line*
      end for
      break
    end if
  end for
end if
Output: X_t

2. Calculating the Calibration Line

Before discussing the calibration line, it is necessary to simplify the model of the two inductive sensors. As shown in Figure 5.6, point O_1 and point O_2 are respectively the center of the left inductive sensor and the right; point C is the center of point O_1 and point O_2. To simplify the calculation, the two inductive sensors together are modeled as one point, that is, point C. Since the size of the inductive sensor is very small (e.g. $radius = 4cm$) relative to that of the environment, the error caused by this model is ignorable. Hence, the coordinate frame **base_ind** is attached on point C.

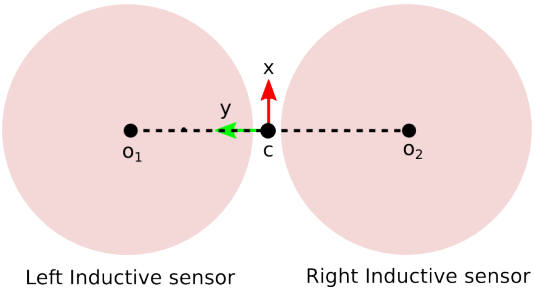


Figure 5.6: Simplified model of inductive sensors

When a stripe marker is detected, the position of the inductive sensors are determined that is exactly above the marker. Since the transform between the base_ind frame and the base_link frame is known, the point A' can be determined. Meanwhile, the calibration line is parallel to the stripe marker. Hence, the equation of the calibration line can be calculated based on the point A' and the slope of the stripe marker.

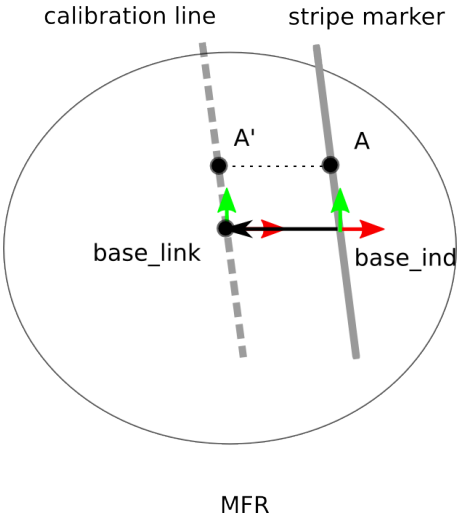


Figure 5.7: Illustration of calibration line calculation

3. Calibrating

Since the equation of the calibration line is determined, particles should be relocated to their foot points on the calibration line and their importance weights are still kept the same.The process of calibration is illustrated in Figure 5.8.

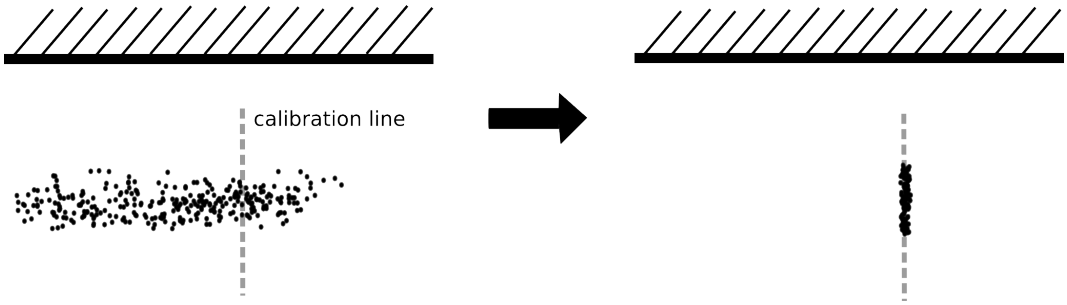


Figure 5.8: Illustration of calibration

6 Hybrid Navigation System

This chapter describes the design of the novel navigation system for MFR. Above all, the overview of the system is described in Section 6.1. Section 6.4 introduces how to build an occupancy grid map for the barn environment. Section 6.5 states how the localization module is set up. After that, Section 6.6 presents the hybrid navigation strategy combining the reactive navigation and the map-based navigation.

6.1 Architecture

The navigation system mainly consists of five modules, as shown in Figure 6.1. The time source module offers the simulation time as the system clock. The V-REP module includes MFR simulation model and the barn environment. This module provides sensor data for other modules and receives velocity commands used to control MFR's movement. The map module offers map data to the localization module and the hybrid control module. By taking in map data and sensor data, the localization module enables MFR to estimate its pose in the environment. With sensor data, odometry and transforms, the hybrid control module decides to follow the map-based control strategy or the reactive control strategy and outputs the corresponding velocity commands. More details will be presented in the rest of this chapter.

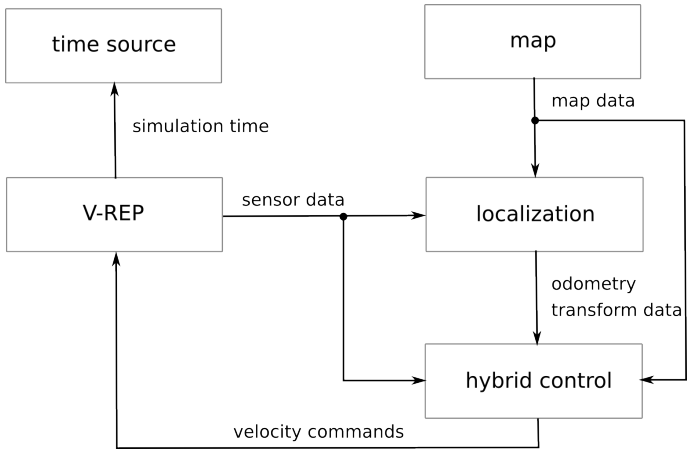


Figure 6.1: Architecture of hybrid navigation system

6.2 V-REP Module

The V-REP module (see Figure 6.2) corresponds to the simulation model described in Chapter 4. The simulation time in V-REP is published and will be used as ROS system clock. More details are presented in Section 6.3. The sensors publish measurement data on the corresponding topics shown in the illustration figure. After receiving the velocity commands on topic "twist_mux/cmd_vel", the motors will be set to rotate accordingly.

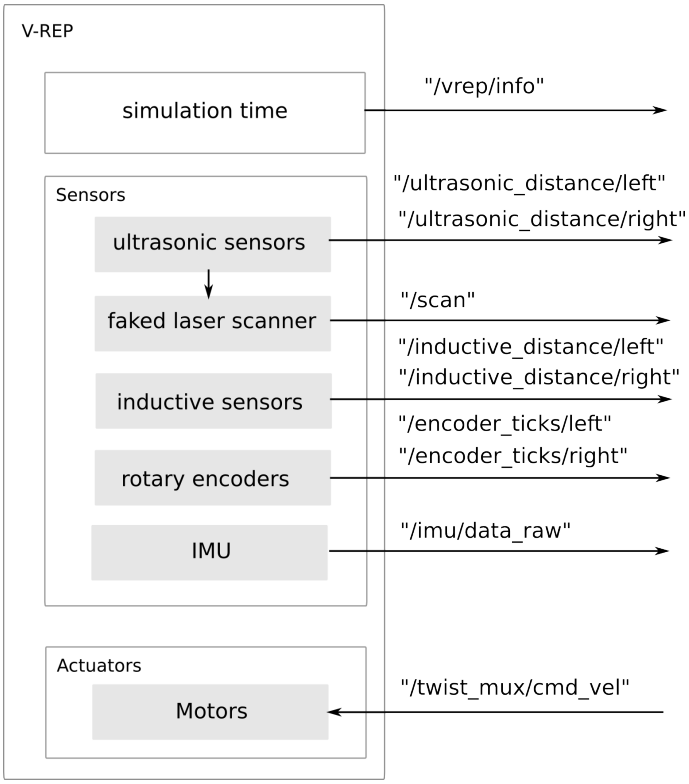


Figure 6.2: Illustration of V-REP module

6.3 Time Source Module

Normally, ROS system uses the system clock of the computer that it runs on as a time source. Since this navigation system is developed and tested with the simulation model, the simulation time generated by V-REP should be used as the time source, which allows ROS system has a consistent time measurement with V-REP. Hence, the package **sim_time_to_clock** is developed to subscribe the simulation time and then publish it on topic "/clock", as shown in Figure 6.3.

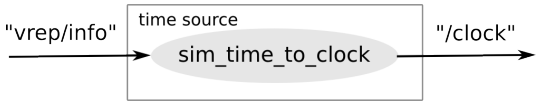


Figure 6.3: Illustration of time source module

6.4 Map Module

For navigation in this project, an occupancy grid map of the barn environment should be provided. Assuming the map has already been constructed, a package named **map_server** [1] is used to read the map from file and to offer the map data to other nodes, as shown in Figure 6.4.

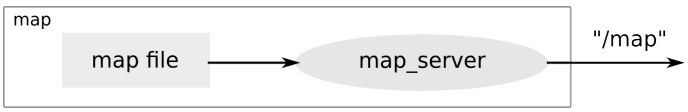


Figure 6.4: Illustration of Map module

If the map of the environment has not been built for the map module yet, the gmapping package can be used to build a map of the barn environment. However, here comes the same problem arising in the application of the AMCL algorithm: MFR is not able to provide the real laser scan data that the Gmapping algorithm needs. Therefore, as described in Section 5.2, the two ultrasonic sensors are faked as a laser scanner. To build the map, MFR is manually controlled to drive around in the test barn environment (see Figure 6.6a). The map building system is shown in Figure 6.5, which, to be noted, is independent of the navigation system. For manual control, a keyboard control node **turtle_teleop_key** is used to publish twist messages on topic "/turtle2/cmd_vel" that is subscribed by V-REP to control motors. The package **mfr_wheel_odom** is developed that takes in encoder ticks and outputs wheel encoder odometry on topic "/odometry/raw" and the odom → base_link transform. By subscribing laser scan data and transforms, the node **slam_gmapping** of the gmapping package is able to construct an occupancy grid map that will be saved to file via map_server.

Applying the map building system, MFR is manually driven around in the test barn environment (see Figure 6.6a) and the occupancy grid map is constructed shown in Figure 6.6b. The black part of the map corresponds to the occupied area, the white part corresponds to the unoccupied area, and the gray part corresponds to the unknown area. Since the map is totally a mess, map building is stopped before it is finished. The key reason why the map building fails is that mapping, as a SLAM process, requires high-quality localization. However, in our case, MFR's localization is only based on odometry since the scan-matching (see Section 2.2.2) fails with the extremely limited range data from the two ultrasonic sensors. In this case, the pose estimation error will grow without bounds with the increasing distance that MFR drives. Unfortunately, the pose error will be incorporated into the map, since the

[1] map_server, available at `http://wiki.ros.org/map_server`

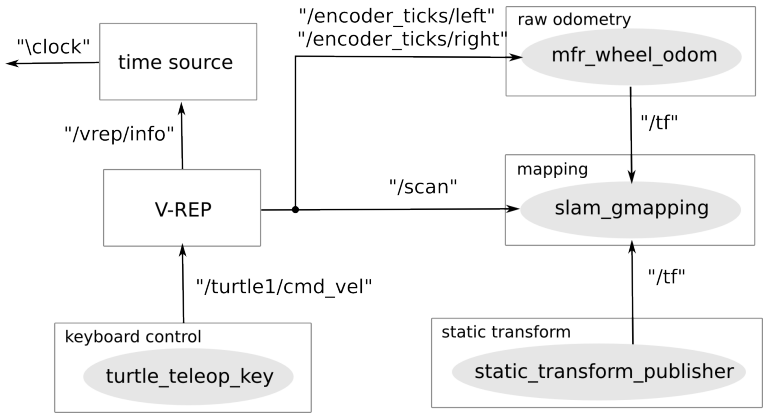
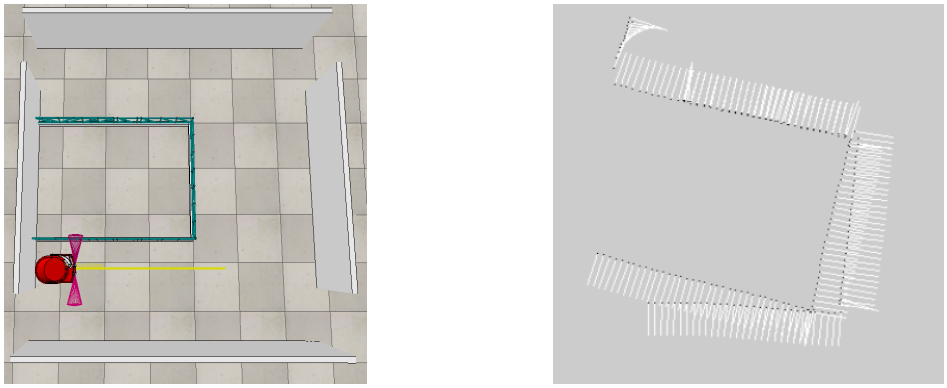


Figure 6.5: Map building system

observed data about the environment cannot be constructed on the right spot on the map. On the contrary, a bad-quality map will further increase the error in localization. Therefore, it is insufficient to build a map by just using the current sensors on MFR.



(a) Test environment (b) Occupancy grid map

Figure 6.6: Map building with the faked laser scanner

Considering the large cost of an industrial laser scanner, it is not financially feasible to install a laser scanner on every MFR. Hence, here comes an alternative: the laser scanner is just used to build a map and will be removed after it is done.

6.5 Localization Module

As the foundation of the navigation system, the localization module is shown in Figure 6.7. The package **mfr_wheel_odom** is implemented in this project to input wheel encoder ticks and to output odometry data, known as wheel encoder odometry. Meanwhile, an IMU is used to provide accurate measurements of angular velocities and linear accelerations that are then fused into an orientation by the IMU filter by Madgwick [33],

implemented in **imu_filter_madgwick**[2]. To obtain a more accurate odometry, the node **ekf_localization_node** [3] is used to fuse the filtered IMU data and the wheel encoder odometry into the IMU odometry. The ekf_localization_node also provides the odom → base_link transform. By inputting necessary messages and the stripe file, the node **samcl** publishes the estimated pose on topic "amcl_pose" and the map → odom transform.

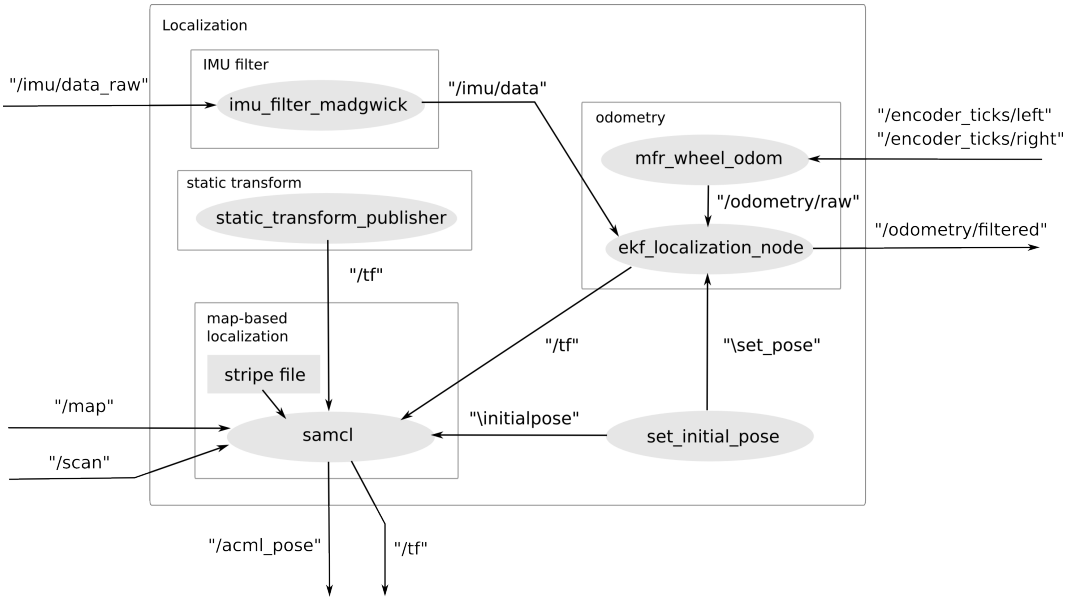


Figure 6.7: Illustration of localization module

6.6 Hybrid Control Module

By inputting transforms, sensor data and map data, this hybrid control module (see Figure 6.8) outputs velocity commands. The route file is actually the set of navigation actions which are previously planned according to the specific barn environment. By reading the actions, MFR is able to execute them one by one, such as, drive straightforward, follow the feeding fence, etc. As the core of this module, the package **mfr_navigation_controller** implements the combination of the reactive navigation strategy and the map-based navigation strategy. The hybrid control strategy works like this: for the reactive control, the controller generates velocity commands on topic "/base_vel" based on the momentary sensor readings; for the map-based control, the controller sends a goal to the move_base (see Section 2.3.2) that plans an obstacle-free path to the goal and publishes velocity commands on topic "/cmd_vel". It is notable that the velocity on "/base_vel" is assigned a higher priority than that on "/cmd_vel". The package **twist_mux** [4] is applied to output the higher prioritized velocity command

[2]imu_filter_madgwick, available at `http://wiki.ros.org/imu_filter_madgwick`
[3]ekf_localization_node, available at `http://wiki.ros.org/robot_localization`
[4]twist_mux, available at `http://wiki.ros.org/twist_mux`

if there is more than one velocity commands at the same time.

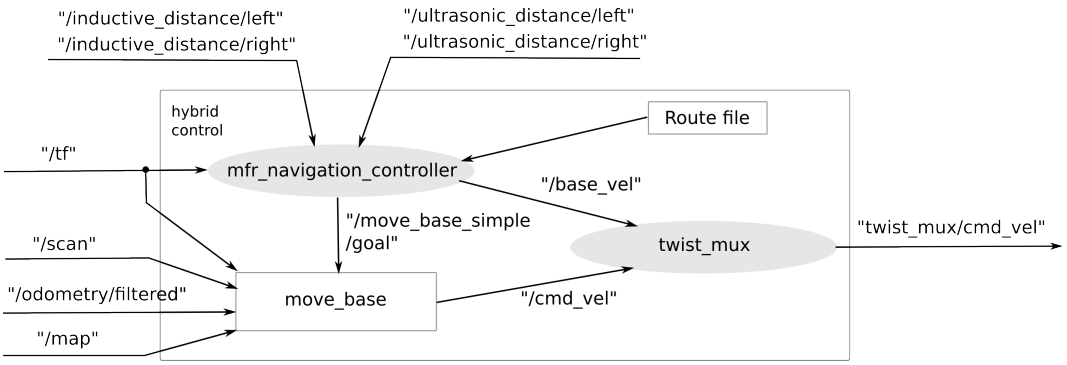


Figure 6.8: Illustration of hybrid control module

To be more detailed, five modes of navigation actions are implemented in the navigation controller, summarized as below.

A. Turn

This mode utilizes the reactive approach. In this mode, MFR is able to turn clockwise or counterclockwise with a given angle. The velocity commands for turning will be published until the given angle is reached. The angle turned is calculated by the difference between MFR's orientation at the moment and its initial orientation.

B. Straight Driving

This mode also utilizes the reactive approach. In this mode, MFR is able to drive straight forward or backward until a given distance is reached or a metal stripe is detected. MFR's orientation is kept by applying a PID controller [5], a simple but widespread control algorithm that continuously calculates the error value as the difference between the desired setpoint and a measured process variable, and applies a correction of the error. For straight movement, the desired setpoint is MFR's initial orientation; the measured process variable is MFR's orientation at the moment; the correction is achieved by the adjustment of velocities (see Figure 6.9).

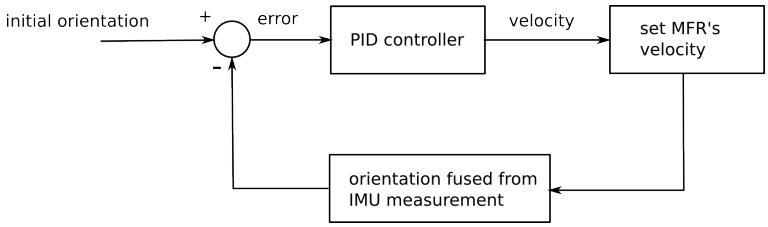


Figure 6.9: Illustration of straight driving control

[5]PID controller, available at `https://en.wikipedia.org/wiki/PID_controller`

C. Free Navigation

This mode utilizes the map-based navigation. In this mode, MFR navigates along an obstacle-free route (see Figure 6.10) that is usually defined as a sequence of waypoints plus a goal. A waypoint is a point on the route that the robot should pass by before the destination is reached; the goal is actually the destination. It is easy to understand that the number of waypoints can be zero, but there must be a goal as the end of a path. As shown in Figure 6.8, the mfr_navigation_controller sends the waypoints and the goal one after another to the move_base that will plan obstacle-free paths and publish velocity commands on topic "/cmd_vel" for MFR to follow the planned paths. With this mode, MFR is able to navigate to a point without the need of following a metal strip as mentioned in Chapter 1, which brings more flexibility to MFR's navigation.

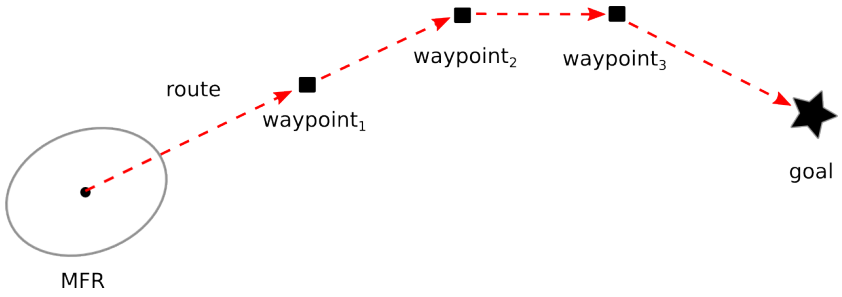


Figure 6.10: Illustration of a route

D. Fence Navigation

This mode combines the reactive navigation and the map-based navigation. In terms of the reactive navigation, a PID controller is applied to make MFR follow the feeding fence with a pre-determined distance by correcting the error between the measured ultrasonic distance and the desired distance, as shown in Figure 6.11. This approach can accurately and responsively control the distance between MFR and the feeding fence, but it is susceptible to the temporary loss of the range data from the two ultrasonic sensors. On the other hand, the map-based navigation cannot control the distance as well as the former, but it is able to suffer from the temporary loss of ultrasonic data since MFR knows where it is with the localization module and where to go with the move_base. Therefore, the reactive navigation is preferred to the map-based navigation in this mode; if the former fails, the latter will be taken until the former is available again. As shown in Figure 6.12, MFR follows the fence to the point A by the reactive navigation; from the point A to the point B, MFR cannot detect the fence and follows the planned path by the map-base navigation; after reaching the point B, MFR continues fence following till the end of the fence and then navigates to the goal.

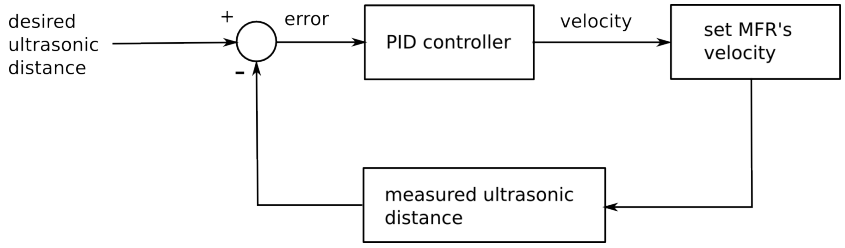


Figure 6.11: Illustration of fence following control

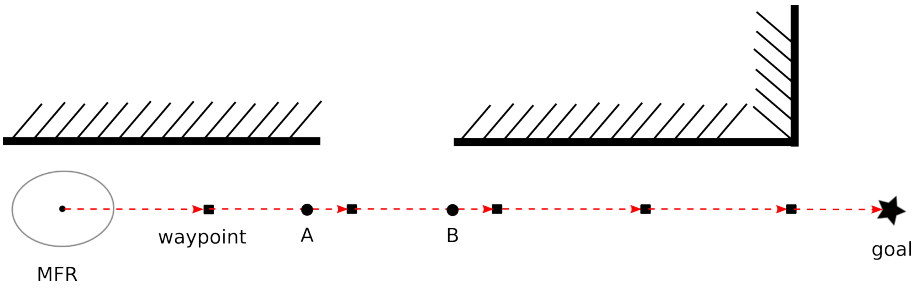


Figure 6.12: Illustration of fence navigation mode

E. Stripe Navigation

Principally, the proposed navigation system does not need to follow a metal stripe as much as the old navigation system does. However, for some cases, it is still needed. For example, when MFR drives back to the charging area, MFR's pose should be controlled so accurately that MFR can be connected to the charger that is small in size. Obviously, this case cannot be handled by map-based navigation. In addition, barn environments are quite complex and it cannot be denied that stripe following is not useful anymore.

Similar to the last one, this mode also applies the combined strategy. Similarly, a PID controller is used enabling MFR to follow a metal stripe with measurement data from the two inductive sensors, as shown in Figure 6.13. When the reactive navigation fails, the map-based navigation will be taken. As shown in Figure 6.14, MFR follows the stripe to the point A by the reactive approach; from the point A to the point B, the stripe cannot be detected, so MFR navigates to the point B by the map-based approach; after that, MFR continues stripe following till the end and navigate to the goal.

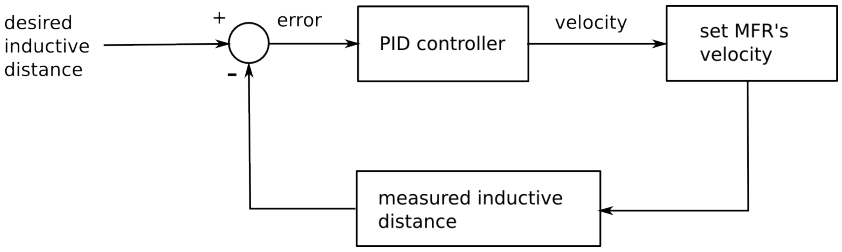


Figure 6.13: Illustration of stripe following control

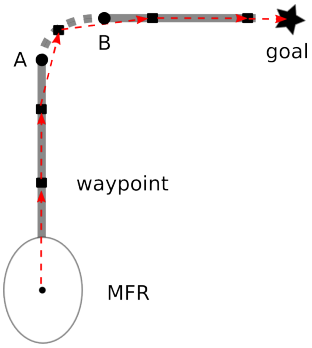


Figure 6.14: Illustration of stripe navigation mode

7 Simulated Experiments and Results

This chapter presents the set-up of experiments and the evaluation of the proposed navigation system. Section 7.1 describes how environments are set up in V-REP for the experiments on map building, localization, and hybrid navigation. Section 7.2 shows the result of map building with a laser scanner. In Section 7.3, the SAMCL pose estimate is compared with the AMCL pose estimate. Then the hybrid navigation system is evaluated in Section 7.4. In the end, a short summary is presented in Section 7.5.

7.1 Environment Set-up

7.1.1 Barn Environment

Based on the sample barn environment described in Section 1.1.2, a barn environment (see Figure 7.1) is set up in V-REP for testing the proposed navigation system. The green objects are feeding fences enclosing the area for cows to stay; the long metal stripes are set for MFR to follow; the short metal stripes are stripe markers for localization; the white parts around are walls.

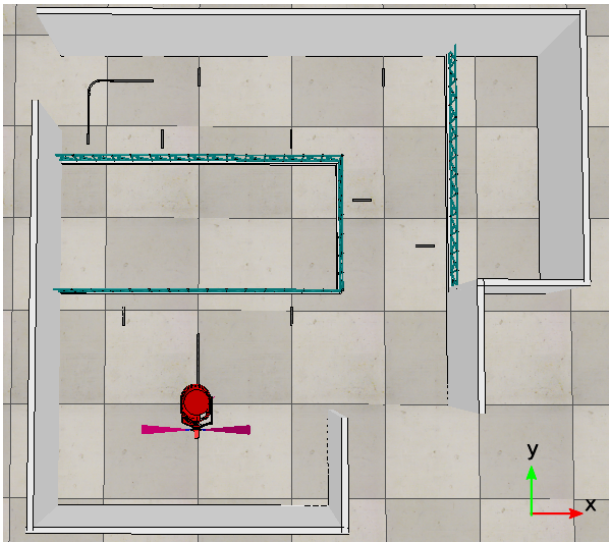


Figure 7.1: Sample barn environment in V-REP

7.1.2 Barn Environment with Errors

As shown in Figure 7.2, the error cases (marked with letters) mentioned in Section 1.2 are simulated in the barn environment:

A The ultrasonic signal is temporarily lost in this 90-degree corner with the corner radius of 1 meters (the red part), which is simulated by making this part undetectable by ultrasonic sensors.

B This part of the fence (2 meters) is removed and therefore the ultrasonic sensors cannot detect it in this area.

C This part of the metal stripe (the quarter-circle arc shape with the radius of 0.5 meters) cannot be detected, which is simulated by the removal of this part.

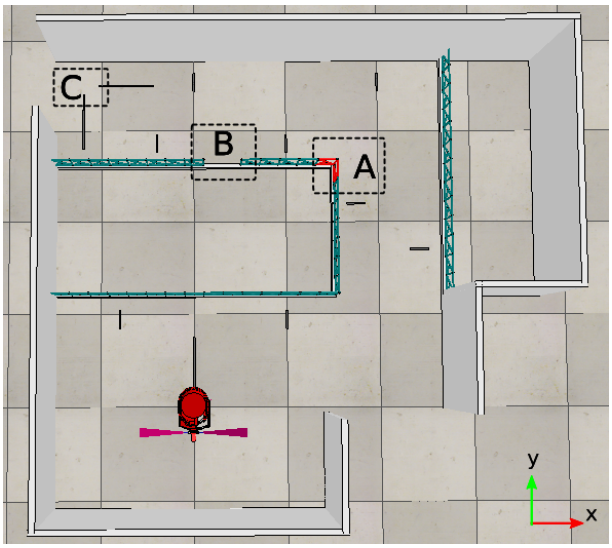


Figure 7.2: Sample barn environment with errors in V-REP

7.1.3 Test Environment for Localization

A typical test environment is set up for comparing the SAMCL pose estimate with the AMCL pose estimate. As shown in Figure 7.3, MFR starts at $(-5.5, 4.7, 0.0)$ with the yaw orientation of 0 radian. Two calibration markers are placed at $(0.0, 4.7, 0.0)$ and $(5.0, 4.7, 0.0)$, respectively.

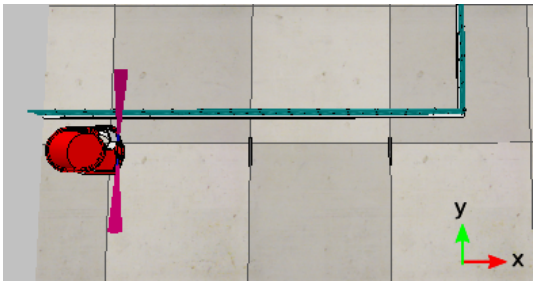


Figure 7.3: Test environment for localization

7.2 Experiment 1: Map Building

Equipped with a laser scanner, MFR is manually controlled to drive around in the environment shown in Figure 7.1. A good grid map (see Figure 7.4) is built with the resolution of 0.05 meters, which serves as the map data for the experiments in Section 7.3 and Section 7.4.

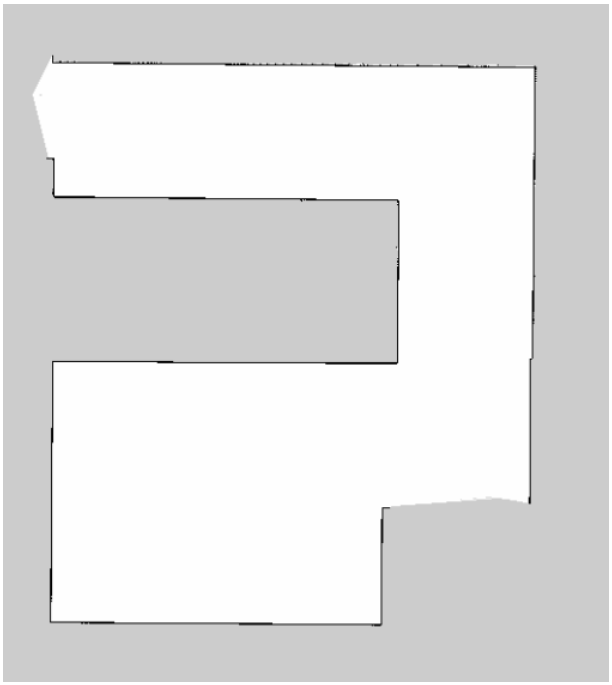


Figure 7.4: Occupancy grid map of the sample barn environment

7.3 Experiment 2: Localization

In this experiment, MFR will be manually controlled to drive forward along the feeding fence twice in the environment shown in Figure 7.3 for two scenarios, respectively.

1. **AMCL pose estimate:** the AMCL algorithm is applied to estimate MFR's pose. For comparison, V-REP publishes MFR's real pose in the simulated environment. The results are shown in Figure 7.5 and Figure 7.7.

2. **SAMCL pose estimate:** the SAMCL algorithm is used for MFR's pose estimate and its real pose is published by V-REP as a reference. The results are shown in Figure 7.6 and Figure 7.8.

Figure 7.5 shows the evolution of the AMCL particle set at five time points (the red parts). As shown in the figure, MFR starts at time t_0 and then the AMCL particle set grows longer and longer over time, which means the uncertainty of the AMCL pose estimate increases. As explained in Section 5.3, the AMCL algorithm can control the uncertainty along the y-axis by taking in the ultrasonic range data but is not able to calibrate the estimated pose along the x-axis by using metal markers. Therefore, the uncertainty along the y-axis is well controlled but not the x-axis. Moreover, Figure 7.7 presents the evolution of the position values on the x-axis and the y-axis over time. We can see that the estimated pose along the y-axis is quite close to the real one; however, along the x-axis, the pose error keeps growing without bounds.

Similarly, Figure 7.6 depicts the evolution of the SAMCL particle set at five time points. As shown in the figure, the uncertainty of pose estimate along the y-axis is small and meanwhile, the uncertainty along the x-axis is reduced twice by the calibration with the stripe markers. Furthermore, it is clearly seen from Figure 7.7 that the pose error along the x-axis is corrected at the point A and the point B (marked in the figure) that correspond to the first stripe marker and the second stripe marker, respectively. By this way, the pose error is controlled within a reasonable range.

Hence, this experiment proves that the SAMCL algorithm outperforms the AMCL algorithm in this project with stripe markers properly set.

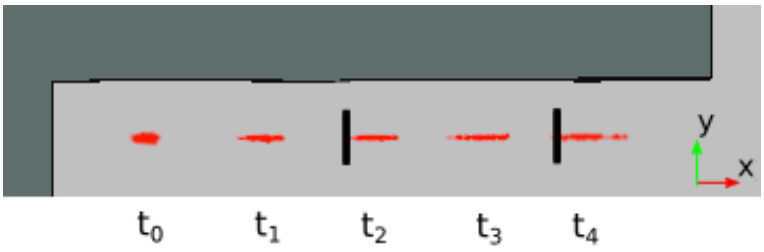


Figure 7.5: The uncertainties of AMCL pose estimate over time

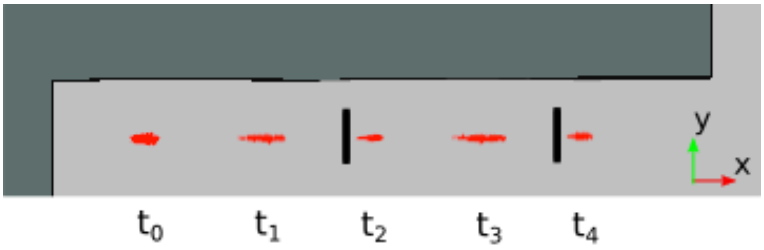


Figure 7.6: The uncertainties of SAMCL pose estimate over time

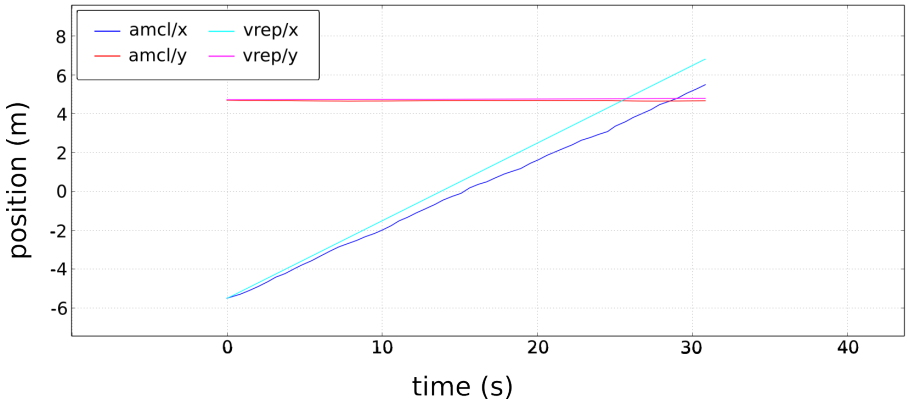


Figure 7.7: Comparison between the pose estimated by AMCL and the real pose provided by V-REP

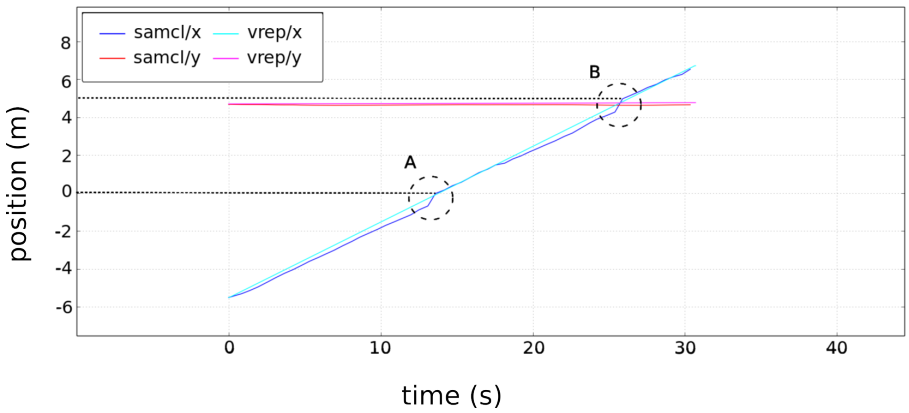


Figure 7.8: Comparison between the pose estimated by SAMCL and the real pose provided by V-REP

7.4 Experiment 3: Hybrid Navigation

In this experiment, the proposed navigation system is tested in both the error-free barn environment (see Figure 7.1) and the barn environment with errors (see Figure 7.2). Starting from the charging pole, MFR automatically navigates for feed distribution and

drives back after it is done.

7.4.1 Navigation in the Error-free Environment

As shown in Figure 7.9, the yellow route is the trajectory MFR has finished in the error-free environment. To be noted, MFR's job is to distribute feed along the feeding fence for cows. The trajectory shows that MFR follows the fences well with the predetermined distance, so it means this navigation system is able to finish feed distribution task accurately. To navigate to the fences, MFR should also be able to drive in a straight line, turn for a certain degree, and follow metal stripes, which is proved by the successful navigation as the trajectory shows. By now, the hybrid navigation system has performed all the navigation tasks that the current navigation system can do, which is mentioned in Section 1.1.3. More importantly, MFR is able to drive from the point A to the point B directly, unlike the process mentioned in Section 1.1.3 that MFR should follow the wall till No.8 metal stripe is detected and then follow the stripe till its end. Obviously, this kind of free navigation provides a more efficient route. Moreover, with the proposed navigation system, MFR is able to take different routes to the same goal with more flexibility via different sets of waypoints (see the free navigation mode in Section 6.6). To conclude, the hybrid navigation system can not only fulfill all the tasks that the current navigation system can perform, but also reduce MFR's dependence on long metal stripes in navigation that is one of the problems mentioned in Section 1.2.

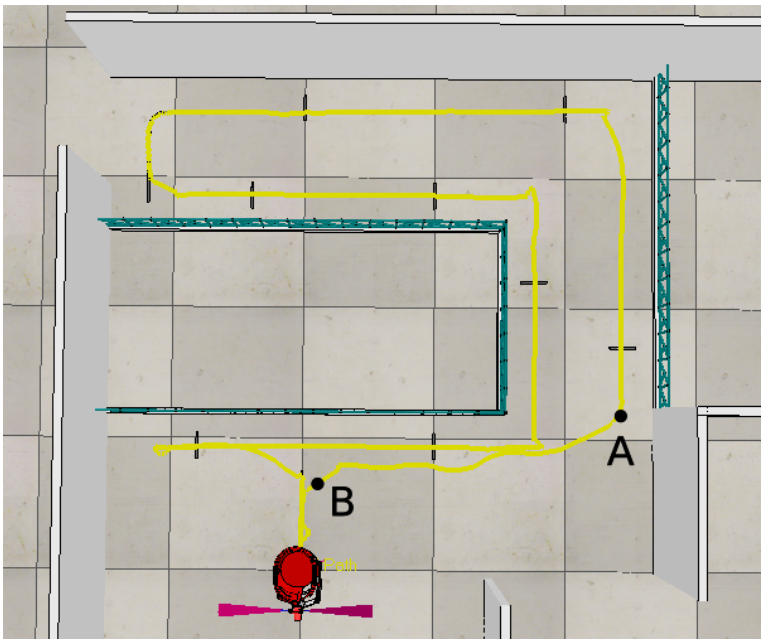


Figure 7.9: Trajectory in error-free environment

7.4.2 Navigation in the Environment with Errors

In this section, MFR automatically navigates in the environment with errors (see Figure 7.2) and the finished trajectory is shown in Figure 7.10. It is clear that the trajectory is complete, that is to say, MFR's navigation in this environment is as successful as that in the last environment (see Figure 7.1). Considering the existence of three errors described in Section 7.1.2, it is sufficient to prove that the proposed navigation system is capable of tolerating these three errors.

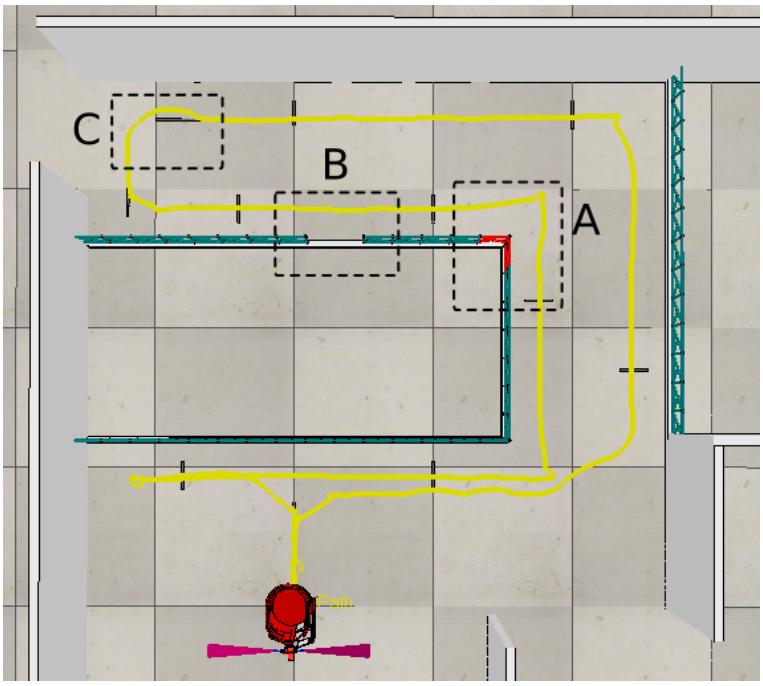


Figure 7.10: Trajectory in the environment with errors

- Around the corner A, the valid ultrasonic signal is lost temporarily. According to the analysis in Section 1.2, with the current system, MFR will just stop if no fence can be found again by in-place rotation. However, with the proposed navigation system, MFR tolerate this error. Even though the distance between MFR and the fence is not kept as well as that during fence following, the more important thing is that MFR does not fail in this area and can continue to distribute feed along the next fence.

- In the area B, part of the fence is removed and the ultrasonic sensors cannot detect anything. Unlike the current navigation described in Section 1.2, the hybrid navigation tolerates this error, which is similar to the error in the area A.

- In the area C, MFR loses track of the metal stripe, so MFR switches from the stripe following to the map-based navigation and continues driving. Then MFR detects the metal stripe again and follows it to the end. However, for the current

version of MFR, it will stop if the stripe cannot be found by in-space rotation, as described in Section 1.2.

7.5 Summary of Experiments

In the first experiment, an accurate grid map is constructed by the gmapping package based on laser range data. The second experiment proves that the SAMCL algorithm is capable of controlling the pose estimate error within a reasonable range depending on the density of stripe markers that are set. Further, the third experiment shows that the proposed hybrid navigation system has two obvious advantages over the current navigation system:

- The errors mentioned in Section 7.1.2 can be well tolerated.
- MFR is given more intelligence and flexibility in the aspect of navigation.

8 Conclusions and Future Work

This chapter finalizes this thesis project. In Section 8.1, the conclusions of this project are presented. Then Section 8.2 lists the main contributions. In Section 8.3, the proposals for future work are suggested.

8.1 Conclusions

Utilizing a reactive approach, the current navigation system running on MFR is susceptible to errors in the environment and has a limited navigation capacity. To improve MFR's error-tolerant capacity and navigation performance, a hybrid navigation system is proposed in this project combining the reactive navigation and the map-based navigation.

First, with the robotics simulator V-REP, a simulation model (see Chapter 4) is built for MFR based on the robot's configuration (see Chapter 3). Since the real robot is not ready for running ROS system, the simulation model is used as the platform for developing and verifying the proposed navigation system.

As the key to the map-based navigation, the localization algorithm is explored in Chapter 5. The AMCL algorithm targets at mobile robots equipped with a laser scanner, so it is not fully suitable for MFR with limited range data from the two ultrasonic sensors. Hence, the SAMCL algorithm is put forward for localization by fusing ultrasonic information and inductive information. To make the localization work, an accelerometer is added in the simulation model.

Moreover, Chapter 6 describes the proposed hybrid navigation system mainly including map module, localization module, and hybrid control module. The map module is responsible for offering map data to other modules. If the map is not ready, it can be constructed by the map building system. Since it fails to build a good map with the current sensors, a laser scanner is added for map building. Further, based on the SAMCL algorithm, the localization module is set up to estimate MFR's pose in the environment. Besides, the hybrid control module chooses to follow the map-based control strategy or the reactive control strategy and publishes the corresponding velocity commands for the motors.

Besides, the performance of the proposed navigation system is evaluated in the simulation model. As a result, the SAMCL algorithm could provide a reasonably good pose estimate if stripe markers are properly set. Further, compared with the current navigation system running on MFR, the proposed navigation system is able to tolerate

the errors like the temporary loss of valid ultrasonic or inductive signal; meanwhile, it enables MFR to navigate without following a long metal stripe for general cases. Hence, it can be concluded that the proposed navigation system improves MFR's error-tolerant capacity and navigation performance.

However, the proposed navigation system additional requirements on the barn environment it could work. Due to the limited detection range of ultrasonic sensors mentioned in Section 3.3.1, MFR is not able to free navigate for a long distance (eg. more than 5 meters) in a relatively large empty area without objects around. Additionally, stripe markers should be set properly according to how fast the pose error grows. Therefore, a barn environment should be well set in advance.

Obviously, the proposed navigation system cannot compete with the state-of-the-art navigation systems using laser scanners, 3D cameras, or other advanced sensors. However, it does achieve a good navigation performance with relatively low-cost sensors. More importantly, this project opens the door of map-based navigation and further improvement in navigation to MFR.

8.2 Main Contributions

The first contribution of this thesis is the simulation model of MFR and barn environments, which makes it possible to develop and test algorithms and navigation systems without a real MFR. In the model, sensors and actuators are simulated according to MFR' configuration. Moreover, the model is made controllable by ROS nodes, therefore, the simulation model is just used to obtain sensor information from the simulated environment and actuate the motors, and the development of the navigation system can be all done in ROS.

The second contribution lies in localization. Due to the lack of a laser scanner, the AMCL fails to control the growth of pose error. Hence, the SAMCL algorithm is put forward calibrating MFR's pose estimate with stripe markers on the floor. As shown in Experiment 2 in Chapter 7, the SAMCL algorithm can control the pose error within a reasonable range according to the density of stripe markers. To make the SAMCL work, mfr_wheel_odom package is developed to estimate wheel encoder odometry from MFR's two wheel encoders.

The third contribution is the exploration of map building. Due to the limited range data from the two ultrasonic sensors, MFR is not able to do scan matching, which is a key reason for the failure of mapping. Instead, a financially feasible alternative is proposed that a laser scanner is just installed for map building process and will be removed after that.

The fourth contribution is the hybrid control module combining the reactive navigation and the map-based navigation. By the hybrid control strategy, MFR is able to tolerate errors mentioned in Section 1.2; MFR is also able to navigate without following long

metal stripes if the environment is well constructed.

The fifth contribution is that different modules are connected working as a navigation system, including V-REP module, map module, localization module and hybrid control module, etc. The proposed navigation system achieves a better navigation performance than the current one.

8.3 Future Work

This project can be extended in the following aspects.

First, the proposed navigation system should be further tested on the real MFR. Even though V-REP is a powerful simulator, it still cannot simulate all the scenarios in the real barn environments. Meanwhile, V-REP does not add any noise, so noises from the real environment and within the real robot should be fully studied to make the navigation work on the real MFR.

Theoretically, the SAMCL algorithm has a capacity to recover from wheel slip to some extent. However, it is not an easy job to simulate and test it in V-REP. Considering the dirty and wet floor in barns, this is an important aspect to explore for a more robust navigation system.

Also, the navigation system should be developed with the ability to avoid dynamical obstacles. In barns, it happens that people or objects unexpectedly stay on the route MFR drives. However, MFR is not able to avoid obstacles dynamically and instead will stop after bumps. For dynamical obstacle avoidance, there should be extra sensors to observe the environment around the robot. For example, several ultrasonic sensors can be installed on the front part of MFR.

In addition, learning algorithms could be introduced to provide a more intelligent navigation. It would be interesting to investigate how to learn from the previous routes and generate a better path. For example, if MFR bumps into obstacles frequently in a certain region, the robot should learn that it is a dangerous area and take another route to avoid that area. Moreover, the waypoints and goals for navigation are manually set in this project, so it is worth exploring how to obtain waypoints and goals automatically by driving around under the manual control.

Besides, it is also very useful if the navigation system could be extended to a multi-robot system. In some barns, there are two MFRs. Currently, they cannot work together in the same barn, so when one MFR is working, the other must wait in the charging area to avoid bumping into each other. However, it would be a nice feature that two MFRs can work together, providing more flexibility for feed distribution. Hence, multi-robot navigation system could be another exciting plan.

Bibliography

[1] D. Filliat and J.-A. Meyer, "Map-based Navigation in Mobile Robots:: I. A Review of Localization Strategies," *Cognitive Systems Research*, vol. 4, pp. 243–282, December 2003.

[2] Ultrasonic Sensing. [Online]. Available: http://www.ab.com/en/epub/catalogs/12772/6543185/12041221/12041229/print.html

[3] R. A. Brooks and J. H. Connell, "Asynchronous Distributed Control System for a Mobile Robot," in *Proc. SPIE*, vol. 0727, February 1987, pp. 77–84.

[4] P. E. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity," in *Proc. AAAI-87*, vol. 1, July 1987, pp. 268–272.

[5] J. H. Connell, "Minimalist Mobile Robotics: A Colony-Style Architecture for an Artificial Creature," *SIAM Rev.*, vol. 34, no. 2, pp. 329–330, September 1992.

[6] T. S. Levitt and D. T. Lawton, "Qualitative Navigation for Mobile Robots," *Artificial Intelligence*, vol. 44, no. 3, pp. 305–360, August 1990.

[7] K. Balakrishnan, O. Bousquet, and V. Honavar, "Spatial Learning and Localization in Rodents : A Computation Model of the Hippocampus and Its Implications for Mobile Robots," *Adaptive Behavior*, vol. 7, no. 2, pp. 173–216, March 1999.

[8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. the MIT Press, September 2005.

[9] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," in *IN PROC. OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI)*, 1999, pp. 343–349.

[10] D. Fox, "KLD-Sampling: Adaptive Particle Filters," in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[11] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, pp. 34–46, 2007.

[12] ——, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, April 2005, pp. 2443–2448.

[13] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 3, pp. 249–265, June 1987.

[14] ——, "Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation," Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, 1989.

[15] H. P. Moravec, "Sensor Fusion in Certainty Grids for Mobile Robots," *AI Magazine*, vol. 9, no. 2, pp. 61–74, July/August 1988.

[16] S. Se, D. G. Lowe, and J. J. Little, "Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks," *I. J. Robotic Res.*, vol. 21, no. 8, pp. 735–760, 2002.

[17] H. Choset and K. Nagatani, "Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization Without Explicit Localization," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 125–137, 2001.

[18] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The SLAM Problem: A Survey," in *Proceedings of the 2008 Conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2008, pp. 363–371.

[19] OpenSLAM. [Online]. Available: http://openslam.org/

[20] K. Murphy, "Bayesian Map Learning in Dynamic Environments," in *In Neural Info. Proc. Systems (NIPS)*. MIT Press, 1999, pp. 1015–1021.

[21] A. Doucet, "On Sequential Simulation-based Methods for Bayesian Filtering," University of Cambridge, Tech. Rep., 1998.

[22] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan, "The Unscented Particle Filter," Cambridge University Engineering Department, Tech. Rep. CUED/F-INENG/TR 380, August 2000.

[23] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *NUMERISCHE MATHEMATIK*, vol. 1, no. 1, pp. 269–271, 1959.

[24] N. J. N. P. E. Hart and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems, Science, and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[25] J.-A. Meyer and D. Filliat, "Map-based Navigation in Mobile Robots:: II. A Review of Map-learning and Path-planning Strategies," *Cognitive Systems Research*, vol. 4, pp. 283–317, December 2003.

[26] ROS Documentation. [Online]. Available: http://wiki.ros.org/

[27] J. M. O'Kane, *A Gentle Introduction to ROS*. Independently published, October 2013, available at http://www.cse.sc.edu/~jokane/agitr/.

[28] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS*. O'Reilly Media Inc., July 2015.

[29] L. Joseph, *Learning Robotics Using Python*. Packt Publishing Ltd., May 2015.

[30] E. Fernández, L. S. Crespo, A. Mahtani, and A. Martinez, *Learning ROS for Robotics Programming.* Packt Publishing Ltd., August 2015.

[31] T. Foote, "tf: The Transform Library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

[32] H. Adam and J. M. Conard, "Survey of Popular Robotics Simulators, Frameworks, and Toolkits," in *Southeastcon, 2011 Proceedings of IEEE*, March 2011, pp. 243–249.

[33] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," in *Proceedings of the IEEE International Conference on Rehabilitation Robotics (ICORR)*, June 2011, pp. 1–7.

[34] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," in *Intelligent Autonomous Systems 13*, vol. 302, September 2015, pp. 335–348.