Department of Electrical and Computer Engineering

Coordination and Actuator Fault Detection for Autonomous Mobile Robots

Demetris Stavrou

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the University of Cyprus

April, 2016

APPROVAL PAGE

Demetris Stavrou

Coordination and Actuator Fault Detection for Autonomous Mobile Robots

The present Doctorate Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering, and was approved on April 21, 2016 by the members of the Examination Committee.

Committee Chair _____
Dr. George Ellinas

Research Supervisor _____
Dr. Christos Panayiotou

Committee Member _____
Dr. Marios Polycarpou

Committee Member _____
Dr. Loucas Louca

Committee Member _____
Dr. Savvas Loizou

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Περίληψη

Τα κινούμενα ρομπότ, ως η μετέπειτα γενεά των βιομηχανικών ρομπότ που έχουν εκμοντερνίσει την βιομηχανία, έχουν εισαγάγει τεράστιες ευκαιρίες τεχνολογικής ανάπτυξης στον σύγχρονο κόσμο. Με εφαρμογές που εκτείνονται από το απλό νοικοκυριό μέχρι και στην εξερεύνηση του διαστήματος, νέες καινούργιες εφαρμογές έχουν εισαχθεί. Αυτές οι εφαρμογές χρειάζονται αυξημένη ρομποτική αυτονομία, λόγω της αδυναμίας των χειριστών να είναι παρόντες ή απλά λόγω της ανάγκης για βελτιωμένη απόδοση. Όταν τα ρομπότ λειτουργούν πλέον αυτόνομα, ο συντονισμός μεταξύ τους είναι απαραίτητος ώστε να διασφαλιστεί αποδοτική λειτουργία καθώς επίσης και η αύξηση της παραγωγικότητας. Στις περιπτώσεις απρόσμενων γεγονότων, για παράδειγμα όταν ένα από τα ρομπότ δεν ανταποκρίνεται, τα εναπομείναντα ρομπότ πρέπει να είναι ικανά να συντονιστούν κάτω από νέες συνθήκες έτσι ώστε να πετύχουν τους στόχους τους. Τα σφάλματα κινητήρων είναι ακόμα μια πρόκληση που παρουσιάζεται όταν τα ρομπότ λειτουργούν αυτόνομα, για τον λόγο ότι προκαλούν μειωμένη παραγωγικότητα και δημιουργούν θέματα ασφαλείας. Τα σφάλματα κινητήρων επηρεάζουν την ομαλή λειτουργία των ρομπότ και προκαλούν αλλαγή στην πορεία τους, δημιουργώντας συγκρούσεις, ζημίες και ενδεχόμενους τραυματισμούς Στης διατριβή αυτή προσεγγίζονται οι δυσκολίες συντονισμού μεταξύ των ρομπότ και στον εντοπισμό σφαλμάτων στους κινητήρες για συγκεκριμένα ρομποτικά συστήματα. Πρώτα, παρουσιάζεται μια μέθοδος βελτιστοποίησης του συντονισμού μια ομάδας ρομπότ που λειτουργούν σε μια αποθήκη. Προτείνεται ένας αποδοτικός αλγόριθμος ώστε να εξευρεθεί λύση στο πρόβλημα της βελτιστοποίησης του συντονισμού σε πραγματικό χρόνο. Δεύτερο, χρησιμοποιείται μία μέθοδος ανίχνευσης σφαλμάτων η οποία χρησιμοποιεί οδομετρία για να ανιχνεύσει σφάλμα κινητήρων και τα ταυτοποιεί σε σχέση με τρεις αντίστοιχους τύπους σφαλμάτων. Τρίτο, παρουσιάζεται μια νέα μέθοδος η οποία αυξάνει την ευαισθησία εντοπισμού σφάλματος, ενσωματώνοντας πολλαπλά όρια. Εισάγεται η έννοια του προειδοποιητικού σήματος, του οποίου η

συμπεριφορά στον χρόνο χρησιμοποιείται για να ανιχνεύσει σφάλματα μικρής έντασης. Τέταρτο, παρουσιάζουμε μια μονάδα επεξεργασίας η οποία μπορεί να εγκατασταθεί σε ένα κινούμενο ρομότ και να παρέχει διόρθωση πορείας σε περίπτωση ενδεχόμενου σφάλματος, χρησιμοποιώντας ανίχνευση σφάλματος, εντοπισμό και ταυτοποίηση. Η αποδοτικότητα των μεθόδων που προτείνονται αποδεικνύεται δια μέσω προσομοίωσης και επικυρώνεται με τη χρήση σε πραγματικό ρομπότ, με τη χρήση της πειραματικής πλατφόρμας που αναπτύχθηκε και παρουσιάζεται σε αυτή την διατριβή.

Abstract

Mobile robots, as a natural successor of manipulators that revolutionized the industrial world, have introduced vast possibilities for technological advancements in today's modern era. With applications spanning from simple household cleaning to planetary exploration, new challenging and unpredictable environments are introduced. These applications demand for increased robot autonomy, either due to the inability for human operators to be present or simply due to the need for improved efficiency. When robots operate autonomously, coordination among them is essential to ensure efficient operation and increase productivity. In cases of unexpected events, for example when one of the robots becomes unresponsive, the remaining robots must be able to coordinate under the new conditions and achieve their tasks. Actuator faults is another challenge introduced when robots operate autonomously, causing reduced productivity, as well as raising safety issues. An actuator fault affects the normal operation, causing the robot to drift away from its path with the risk of colliding with an obstacle, causing damages to its surroundings and possibly injuring people. This thesis addresses the challenges of coordination and actuator fault detection for certain mobile robot systems. First, a method is presented for optimizing the coordination of a team of robots operating in a warehouse. An efficient algorithm is proposed to provide the solution of the coordination optimization problem in real-time. Second, a model-based fault detection approach is used that utilizes odometry to detect actuator faults and identifies them with respect to three possible fault-types. Third, a new detection method is presented for increasing detection sensitivity by incorporating multiple thresholds. The concept of warning signal is introduced, whose temporal behavior is used to detect faults of small magnitude. Fourth, we present a module that can be installed on a mobile robot and provide path correction in the event of a fault, using fault detection, isolation and identification. The effectiveness of the proposed methods is demonstrated through simulations and validated on a real mobile robot, using the experimentation platform developed and presented in the thesis.

Acknowledgments

I would like to express my gratitude to Christos Panayiotou and Marios Polycarpou for their help, feedback and for giving me the opportunity to study as a PhD student at the prominent KIOS Research Center.

It has been a long and rewarding journey that has given me the chance of meeting several outstanding researchers and colleagues that I now have the honor to call fiends, especially Stelios Neophytou, Michalis Michaelides, Christos Laoudias, Alexandros Kyriakides, Agathoklis Papadopoulos and Lenos Hadjidemetriou. In addition I would like to express my deep friendship to Michalis Skitsas.

I would like to thank Stelios Timotheou for being an excellent collaborator, providing ideas and clever solutions when needed. I would specifically like to express my gratitude to Demetrios Eliades for his valuable guidance, friendship and support he provided, especially during my difficult academic times. I would also like to thank George Milis for his friendship and support; I enjoyed our inspiring conversations and I look forward for our future endeavors.

I would like to thank the faculty of Electrical and Computer Engineering department as well the KIOS administrative assistants. I would also like to acknowledge the University of Cyprus for supporting my research.

I dedicate this thesis to my parents, for their unconditional love throughout my life. I owe everything I accomplish to them. Lastly, I would like to thank my beautiful wife Penelope, for her love and support.

Publications

Published journal publications

- **D. Stavrou**, D. G. Eliades, C. G. Panayiotou, and M. M. Polycarpou, "Fault detection for service mobile robots using model-based method," *Autonomous Robots*, vol. 40, no. 2, pp. 383–394, 2015.

Published conference proceedings

- **D. Stavrou** and C. G. Panayiotou, "Task assignment and agent coordination in a warehouse environment," in *IEEE Mediterranean Conference on Control & Automation*, 2012, pp. 1341–1346.
- **D. Stavrou**, D. G. Eliades, C. G. Panayiotou, and M. Polycarpou, "A path correction module for two-wheeled service robots under actuator faults," in *IEEE Mediterranean Conference on Control & Automation*, 2013, pp. 1119–1126.
- **D. Stavrou** and C. G. Panayiotou, "Localization of a simple robot with low computational-power using a single short range sensor," in *IEEE International Conference on Robotics and Biomimetics*, 2012, pp. 729–734.
- D. G. Eliades, **D. Stavrou**, S. G. Vrachimis, C. G. Panayiotou, and M. M. Polycarpou, "Contamination event detection using multi-level thresholds," *Procedia Engineering*, vol. 119, pp. 1429–1438, 2015.
- **D. Stavrou**, L. Hadjidemetriou, E. Kyriakides, and C. G. Panayiotou, "Low-cost interface platform for controlling a real-time system," in *IEEE Proceedings of Hellenic Student Branch Congress*, 2013.
- C. Ttofis, **D. Stavrou**, D. Koukounis, T. Theocharides, and C. G. Panayiotou, "A laboratory course on 3D vision for robotic applications," in *IEEE International Conference on Microelectronic Systems Education*, 2013, pp. 21–24.

- L. Hadjidemetriou, G. Nicolaou, **D. Stavrou**, and E. Kyriakides, "Low-cost real-time monitoring of a laboratory scale power system," in *IEEE Mediterranean Electrotechnical Conference*, 2016, pp. 1-6.

Contents

Nomenclature

Chapter 3

$\mathcal{C}$	Set of conflicting robots
$\mathbf{C}$	Cost matrix of assignment
f_T^h	Total cost of assignment and coordination (heuristic algorithm)
f_A^h	Assignment cost (heuristic algorithm)
f_C^h	Coordination cost (heuristic algorithm)
f_T^*	Optimal total cost of assignment and coordination
f_A^*	Optimal assignment cost
f_C^*	Optimal coordination cost
L	Set of lanes
N	Number of robots
$\mathcal{R}$	Set of mobile robots in the facility
$\mathcal{R}^{\overrightarrow{s}}$	Sorted set of robots (ascending)
$\mathcal{R}^{\overleftarrow{s}}$	Sorted set of robots (descending)
$\mathcal{S}$	Set of tasks
T	Time to transport a container
T^e	Entrance travel time
T^v	Travel time to container
T^l	Loading time
T^d	Travel time to loading area
T^g	Guard time
t^e	ATW entrance time
$\hat{t}^e$	ATW entrance time including entrance waiting time
t^x	ATW exit time
$\hat{t}^x$	ATW exit time including entrance and exit waiting time

w^e Entrance waiting time

w^x Exit waiting time

$\mathbf{X}$ Assignment matrix

Λ Completion time of all tasks

$\wedge$ Logical function AND

$\vee$ Logical function OR

$\oplus$ Logical function XOR

Chapter 4

b^θ Estimated parameter variation

$\overline{b}^\theta$ Estimated parameter variation upper bound

c Fault identification characteristic

e Estimation error

$\overline{e}$ Estimation error upper bound

h Estimation error bias

K^f Fault occurrence time

n Noise

$\overline{n}$ Noise upper bound

p Small decision margin

t^{det} *PreAlarm* active duration during training

$\overline{t}^{det}$ *PreAlarm* active duration upper bound

u Motor input

u^f Motor input under fault

u^δ Large-change-threshold for motor input

u_{min} Minimum available motor input

u_{max} Maximum available motor input

x Wheel speed

$\hat{x}$ Wheel speed estimation

w Moving average window

x^{ol} Open loop wheel speed

y Wheel speed sensor reading

z Fault structure

α Fault magnitude

$\hat{\alpha}$ Estimated fault magnitude

β Fault time profile

γ Adaptive gain

η^{θ} Parametric uncertainty

η^{x} Modeling uncertainty

$\overline{\eta}$ Uncertainty upper bound

θ^{*} Optimal wheel speed model parameters

$\hat{\theta}$ Wheel speed estimated model parameters

Λ Estimation error filter coefficient

τ *PreAlarm* active duration

ϕ Fault term

Chapter 5

a Uncertainty level

b Moving average warning signal

$\overline{b}$ Moving average warning signal detection threshold

b^{θ} Estimated parameter variation

$\overline{b}^{\theta}$ Estimated parameter variation upper bound

b^{S} Uncertainty bin

D Fault detection alarm signal

d Fault detection signal

e Estimation error

$\overline{e}$ Estimation error upper bound

F^{-1} Inverse cumulative distribution function

$\mathcal{H}$ Uncertainty range

h Uncertainty level values

K^{f} Fault occurrence time

N Number of levels

N^{S} Region divisions

S Set of regions

$\mathcal{U}$ Input range

u_{min}	Minimum available motor input
u_{max}	Maximum available motor input
w	Warning signal
x	Wheel speed
$\hat{x}$	Wheel speed estimation
y	Wheel speed sensor reading
$\mathcal{Y}$	Sensor measurement range
z	Fault structure
α	Fault magnitude
β	Fault time profile
η	Uncertainty
$\overline{\eta}$	Uncertainty upper bound
θ^*	Optimal wheel speed model parameters
$\hat{\theta}$	Wheel speed estimated model parameters
Λ	Estimation error filter coefficient
$\boldsymbol{\mu}$	Multiple thresholds
τ	Moving average processing window
ϕ	Fault term
Ψ	Training data sample set

Chapter 6.1

d	Distance of robot wheels
g	Robot goal
$\overline{M}$	General actuator uncertainty
S	Obstacle-free region
T^f	Fault occurrence time
u	Motor input signal
u_{min}	Minimum available motor input
u_{max}	Maximum available motor input
v	Motor speed
w	Controller input before FDI-PC

$\mathbf{x}$ Robot pose (position and orientation)

$\hat{\mathbf{x}}$ Robot pose (position and orientation) estimation

y Robot pose (position and orientation) measurement

z Fault structure

α Fault magnitude

β Fault time profile

$\overline{\epsilon}^0$ Detection threshold

$\overline{\epsilon}$ Isolation threshold

η Modeling uncertainty

$\overline{\eta}$ Uncertainty upper bound

Λ Estimation error filter coefficient

ϕ Fault term

Φ Set of faults

Chapter 6.2

a Robot angular travel distance

$\hat{a}$ Robot angular travel distance including errors

l Wheel distance from the center

M Number of particles

m Particle

p_{hit} Correct sensor reading distribution

p_{rand} Random sensor error distribution

s Robot straight travel distance

$\hat{s}$ Robot straight travel distance including errors

Y Odometry measurement

w Particle's importance weight

w_{fast} Short term value of localization performance

w_{slow} Long term value of localization performance

w_{avg} Average value of localization performance

$\mathbf{x}$ Robot pose (position and orientation)

z Distance sensor measurement

z_{rand} Distance sensor random measurement weight

z_{hit} Distance sensor correct measurement weight

β Particle regeneration parameter

List of Figures

List of Tables

Chapter 1

Introduction

1.1 From Industrial to Service Robots

Just few decades ago, industrial robots have revolutionized the manufacturing processes with their phenomenal performance in repetitive tasks that require extreme accuracy. In the pursuit for improving the quality of life, there is great demand for robots to become mainstream and assist humans in their everyday life. This requires technological breakthroughs beyond the industrial robots. Despite their capabilities, industrial robots require to operate in highly controlled and predictable environments, and in the majority of cases no humans are allowed in their close vicinity to prevent any accidents. Their lack of mobility means that industrial robots have to operate in a fixed location with limited range of motion. In order to assist humans in their everyday life, robots needed to expand their reach and be able to travel where needed. With the emergence of mobile robots, new applications were introduced spanning from household cleaning robots to planetary exploration rovers. This expansion meant that mobile robots were required to operate in new and unfamiliar environments, many of them being of dynamic nature. Mobility and autonomy became essential in robots, due to the inability for human operators to be present, for example in remote or hazardous environments, and due to the need for improved efficiency.

A *service robot* [56] is a robot that performs useful tasks for humans or equipment excluding industrial automation application. Service robots were created for the purpose of relieving humans from difficult, hazardous and repetitive tasks, both in working and household environments. Furthermore, service robots are aimed towards assisting disabled or elderly people in everyday tasks which are difficult to complete due to the per-

son's condition [45, 54]. Service robots can be semi or fully autonomous robots as well as mobile, depending on the application. Already, there is a large number of service robots deployed in the world; the cleaning robot iRobot Roomba is a particularly successful case study with both domestic [38] and commercial applications. The robot is equipped with brushes and vacuum chamber, autonomously moving around a building in an intelligent or even random patterns cleaning the floors. There are service robots operating in hospitals, office buildings, warehouses, museums and other environments fulfilling services such as deliveries, transportation, education, entertainment and cleaning. Service robots have progressed over the years and are able to work extremely close to humans in applications like rehabilitation where the robot is strapped on person's body as an exoskeleton, helping the person regain motor skills which were lost due to an injury.

A service robot is a complex system with various interconnected subsystems; some of it's core blocks being the controller, actuator and sensor. A robot uses actuators to move around and interact with the environment, and sensors provide feedback on these actions. At the heart of the robot there is a controller which receives information from sensors and computes the signals sent to the actuators. Given an objective, the controller decides how the robot will behave given its current state. For example, the controller of a mobile robot constantly reads the front-facing range sensors and if an obstacle is detected, it steers the robot away in order to avoid collisions. Multiple robots can be deployed in the same environment and work as a team to complete a common task. In order to coordinate, robots exchange information wirelessly and decide how each robot should act, in either centralized or decentralized architecture. Technological advancements driving the market of service robots, are pushing the boundaries of hardware and software, leading to bigger, faster and more complex robots that are more capable than ever.

1.2 Research Challenges and Motivation

The economical benefits associated with increased productivity led to the use of multiple robots co-operating in the same environment [43], thus introducing the challenge of mobile robot coordination. In a multi-robot application, each robot has to carry out its own task contributing to the fulfillment of a general objective. When operating in the same area, robots face the risk of conflict i.e. colliding with each other as well as causing deadlocks. A deadlock occurs when two robots are positioned in such way that each one obstructs the movement of the other, halting their operation indefinitely. Another robot

coordination problem arises when robots try to maximize their own productivity rather than working as a team. Robots are needed to coordinate their actions in order to maximize global productivity. Computing a coordination plan fast, is an additional problem introduced by the dynamic nature of the environments. Since service robots operate in such environments, an unexpected event may disrupt the normal operation, requiring the robots to re-coordinate in order to continue.

Another important challenge is fault diagnosis, which is associated with the reliability and safety of robots around people. A fault is an unexpected change in the system causing unacceptable deviation from the normal operation [50]. A robot faces the risk of having one or more of its components malfunction and thus behave abnormally. The fault changes the behavior of the robot and as a result the controller may lose the control of the robot, putting in danger nearby people or causing damages to its surroundings. With fault diagnosis, a robot will become aware of the fault and can take action in order to prevent any adverse consequences. Therefore, instead of losing control of the robot, the controller will be able to assess the situation and act accordingly. An extreme measure would be to stop any movement to prevent any harmful collisions and raise an alarm to notify the robot owner about the fault. In other situations, the controller can adapt to accommodate the fault so that the robot will continue its function.

Motivation of this thesis is to minimize the costs and risks associated with lack of coordination and actuator faults in service robots. We investigate the problem of multiple robots competing over a set of resources in order to achieve a common objective. These robots need to coordinate in order to efficiently share the common resources and maximize their collective productivity. In specific, we study the problem of transporting a set of containers from the storage to the loading area of a warehouse using multiple robots. In addition to deciding the assignment of containers to robots, efficient coordination is required based on the limitations introduced by the topology of the warehouse. There is limited space for the robots to move which they have to share, therefore coordination is essential to avoid any deadlocks. A way to prevent deadlocks is to make all paths unidirectional, allowing robots to travel in a single way. This requires for the correct design of the path network, but even in this case, such solution is inefficient in terms of traveling time because robots have to travel longer distances, delaying the delivery of containers and use more energy. This motivates us to research for a more efficient solution, where robots maximize the utilization of available resources by coordinating. Furthermore, to remain reliable as a team, the coordination needs to be computed fast in order to allow

fast reconfiguration when an unexpected event occurs.

In the pursuit to minimize the risks associated with faults, we investigate actuator fault detection on differential-drive mobile robots. An approach to detect faults is through model-based residual method. A model is used to estimate the state of the robot while sensors provide information about the actual state of the robot. By comparing the estimated and measured state, a residual is generated. Uncertainty is a key component in this process and dictates when the residual is significant in order to indicate the occurrence of a fault. We investigate actuator fault detection under different fault conditions and using different sensors. To achieve this, we need to take into account the different sources of uncertainty, in order to determine a detection threshold, which is adaptive with respect to the states of the robot. Sometimes, actuator faults of small magnitude are masked by the uncertainty which makes them difficult to be detected; this motivates us to develop a different approach to improve detection sensitivity.

Once a fault is detected, a robot can react in order to mitigate the effects of the fault. We present a path correction method that uses fault detection, isolation and identification in order to alter actuator signals and allow the robot to continue its operation. In another effort to react to faults, we consider the scenario when a robot experiences multiple faults on its sensors, and only one sensor remains operational. In this scenario, we present a localization approach that is suitable for single sensor robots.

1.3 Contributions

The contributions of this work are:

Coordination of mobile service robots in a warehouse

- Mixed Integer Linear Programming formulation that allows optimal solution of the problem using appropriate solvers.

- A near-optimal, low-complexity algorithm that allows real-time coordination of a team of mobile service robots, designed for container transportation.

Actuator fault diagnosis on a differential drive service robot

- A model-based approach for detecting faults using only odometry sensors and identification of those faults with respect to three possible fault-types. The proposed method is designed to be practical, using training to obtain the required parameters.
- A method that uses the temporal behavior of a warning signal in order to detect faults. The method offers significant improvement in fault detection sensitivity.
- The 'Fault Detection Isolation and Path Correction' module which provides actuator fault detection using full-state measurement provided by a camera. The module is designed to work as a plug-in device on the robot, and after detecting, isolating and identifying the fault, takes corrective measures to allow the robot to continue its operation if possible.
- An implementation of the Monte Carlo Localization algorithm, suitable for mobile robots using a single infra-red range sensor.
- Design and implementation of a hardware and software platform that allows experimental work to be conducted wireless on the iRobot Roomba robot.

1.4 Thesis Outline

The outline of this thesis is illustrated in Fig. 1.1. Chapter 2 presents the related work. In Chapter 3 we investigate a problem associated with transporting a set of containers from the storage to the loading area of a warehouse using autonomous robots. In addition to assigning robots to containers, the special topology considered in this work requires coordinated planning of the robots' movement to avoid conflicts. We formulate the joint problem of robot assignment and movement coordination with the objective of minimizing the time required for all robots to carry their assigned containers to the destination, subject to conflict-free movement of all robots. In Chapter 4, we address the problem of fault detection on a differential-drive mobile robot using odometry sensors. Inspired by theoretical work from the field of Fault Diagnosis, we take a model-based approach for detecting and identifying faults. After detection, the fault is identified against three possible fault-types. Finally, the chapter also introduces the problem of small faults, i.e. faults that cannot be detected because the uncertainty masks the effects of the fault. In the same

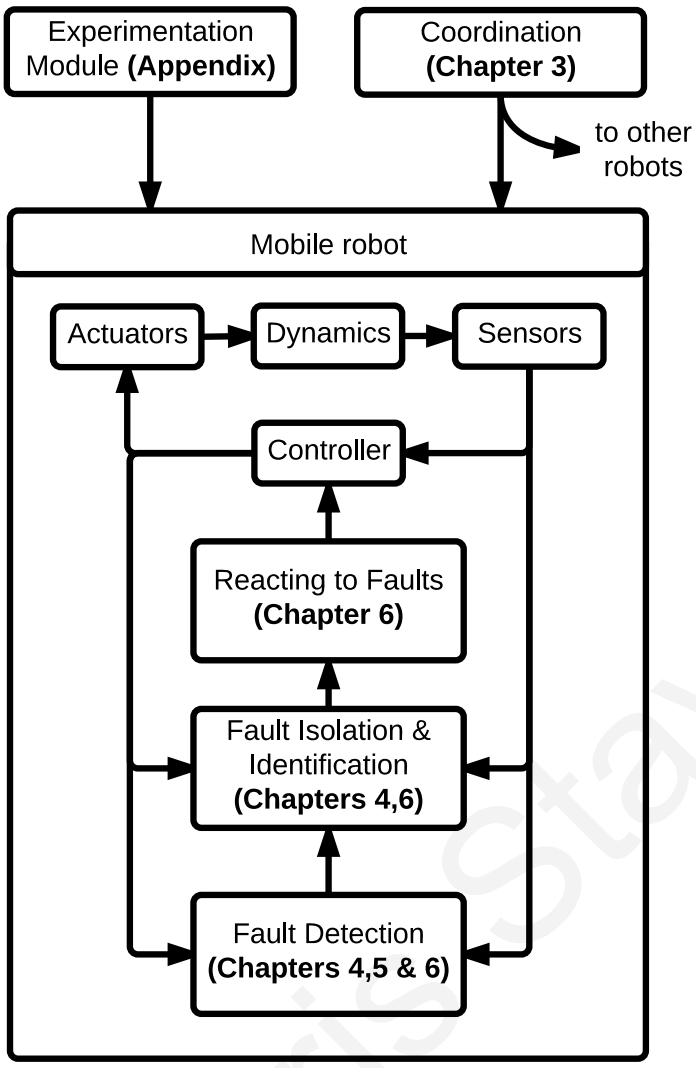


Figure 1.1: Thesis overview.

work we propose an alternative method that is able to improve the detection under small faults. Chapter 5 is motivated from the small faults problem identified in Chapter 4. In this work, we propose a new model-based approach for detecting faults in service robots that aims towards improving detection sensitivity by incorporating multiple thresholds for evaluating the residual. Previously, the upper bound of the uncertainty distribution was used to ultimately determine the detection threshold but in some cases the threshold becomes conservative since it is determined on a worst-case basis. Chapter 6 presents a module that when installed on a mobile robot provides path correction by using fault detection, isolation and identification. In the same chapter, we present a Monte Carlo Localization based method that is suitable for single sensor robots. Finally, in Chapter 7 we present the concluding remarks and future work.

Chapter 2

Related work

2.1 Coordination of Mobile Service Robots in a Warehouse

In the constant pursue for improved efficiency, manufacturing facilities are equipped with robotic systems for automating procedures [90, 111]. Recently they are extensively used in other areas, such as warehouses, container terminals and transportation systems [15, 35]. A facility as such, consists of multiple autonomous mobile robots used as carriers for transporting containers from storage areas to handling stations by following predetermined paths, physically or virtually defined. A centralized system is responsible for designating the containers that have to be transferred and scheduling those operations. A mobile robot is being assigned to transfer a specific container and then proceeds to execute its task.

To limit the storage cost, operators have an incentive to increase the number of containers that are stored in the facility, by trading off the available space left for the robots to travel into. Also, the number of robots operating simultaneously, increases the throughput of the facility. However, as the number of robots is increased and the available moving space is decreased, several challenges are introduced. Robots can run into deadlocks and their aversion or resolution costs valuable resources. For example, a deadlock situation can occur when two robots travel towards each other in opposite directions, in a path wide enough to fit only one of them. In this context, we examine a problem requiring the assignment and movement coordination of multiple robots in a container warehouse to minimize the maximum time needed to complete individual tasks.

Due to similarities, one may relate this problem to the *Vehicle Routing Problem* (VRP)

[24] which considers a fleet of vehicles, all with the same capabilities, used to deliver goods to a set of customers located at different locations and then return back to the depot. This problem has received great attention by many researchers and many solutions have been proposed for the VRP [64] as well as its variants, Vehicle Routing Problem with Time Windows, Capacitated Vehicle Routing Problem and Vehicle Routing Problem with Pick-Up and Delivery. However, there are significant differences with the problem considered in this work. The spatial scale of the VRP is much broader than the considered problem, since nodes in the network usually represent locations within a city. This draws the attention of the problem away from deadlocks or congestion that may occur when vehicles are operating. In a facility with limited free space such as a warehouse, deadlocks and congestion have to be explicitly taken into account. Furthermore, in the VRP case the shortest path normally generates the fastest time but in a facility with multiple vehicles this is not the case. These significant characteristics motivate us to treat this problem differently.

Automating a facility with mobile robots involves solving the scheduling and routing problems [84]. The purpose of scheduling is to coordinate the available vehicles by designating which container should be handled by each vehicle, usually under the constraint of priority and with the objective of minimizing the time. Given successful scheduling, the purpose of routing is to discover an efficient path with respect to time, between each vehicle and it's destination. The algorithms must ensure that the vehicles reach their destination without conflicting with other vehicles.

One way to approach this problem is through the design of the path network on which the autonomous robots move between pick-up and delivery stations prior to the setup of the facility. By focusing on the path network design, more efficient routing solutions can be obtained at the expense of less flexibility in the configuration of the facility [30, 41]. A second approach decomposes the facility into non-overlapping, single vehicle loops that operate in tandem, in a way that the exchange of containers is achieved through transfer stations positioned between adjacent loops [12, 33, 62]. This implies that each container is handled by more than one vehicles before reaching its destination, introducing significant overhead due to the multiple loading/unloading procedures. A third approach is to segment the path network into logical zones and then impose rules on each zone to predict and avoid deadlocks, e.g. restricting the number of vehicles allowed within a specific zone [75, 86] or employing a Petri-net formalism to model and control deadlocks [110]. The main drawback of zone segmenting is the reduced utilization of the available re-

sources, as zones assigned to specific vehicles cannot be utilized by other vehicles. A fourth approach jointly considers the design of the facility and the path network, as well as the routing of the robots to maximize the performance of the overall system [27].

In cases that the movement of the vehicles is not restricted, i.e. they are able to move anywhere within the facility, vehicle paths can be calculated a-priory or on-line. Conflicts can be avoided by extensive planning and utilizing the spatial resources [23] or both spatial and temporal resources of the path network [60, 74, 95]. Such methods generate efficient solutions but the solution space expands rapidly as the number of network paths and the number of vehicles increases, requiring significant computational power, making them unsuitable for real-time execution. To deal with real-time constraints, a common practice is to calculate these paths a-priory. However, unexpected events such as delayed movement, running ahead of schedule or temporary hardware malfunction, may lead to conflicts which invalidate the precomputed paths, hence requiring a new solution. In an effort to lower the complexity, researchers describe a conservative myopic strategy i.e. vehicles are routed one-by-one while all the previous route decisions are respected [61].

Assignment and coordination of vehicles in the context of logistics is a well researched problem; the most relevant work with our problem was the work that addressed the general topology. The solutions proposed in that set of work, are aimed towards environments with no special or dedicated infrastructure where vehicles can utilize to assist their coordination. This is also the case in our problem, but the facility under study has important differences that requires different approach. The most critical being the extremely limited moving space for the robots. This constraint restricts the application of existing methods because there are no alternative paths between two locations in the majority of the facility space. Most of the existing methods rely on alternative paths in order to efficiently solve this problem using incremental approach. Therefore, to address this problem a different approach was needed.

2.2 Actuator Fault Diagnosis on a Differential-drive Service Robot

Many of the service robots applications require increased robot autonomy, either due to the inability for human operators to be present (remote or hazardous environments) or simply due to the need for improved efficiency. When a robot operates autonomously,

an actuator fault can cause significant downtime, reduced productivity as well as raising safety issues. An actuator fault affects the normal operation and causes the robot to drift away from its intended path with the risk of colliding with an obstacle, causing damages to its surroundings and possibly injuring nearby people. Safety is identified as one of the critical issues that needs to be addressed before the wide acceptance of service robots in domestic environments [103]. Despite advancements in technology, experience has shown that a failure will occur at some point [16]. Furthermore, a fault could occur due to external factors, which where not anticipated during the design of the robot, for example people's interactions with the robot could cause damage leading to a fault [13]. The need for a more dependable robot has generated interest for applying fault detection methodologies in robotics.

A fault can occur on any of the robot's subsystems, and in the field of wheeled mobile robotics, the locomotion subsystem has attracted significant research attention. A fault is defined as an unexpected change in the system that causes unacceptable deviation from the normal operation [50]. When a fault is present, the actual robot's movement deviates from the expected and if left untreated, the robot will collide as some point to an element of the environment. A fault tolerant mobile robot is able to continue its operation in the presence of a fault, although it can experience reduced performance. In this thesis focus on fault detection i.e. the ability of the robot to become aware of a fault occurrence. We also apply fault identification in order to learn the magnitude of the fault. This capability, which can be viewed in the context of self-awareness, improves the autonomy of the mobile robot and helps to prevent any consequences the fault may cause since the robot can stop its operation and raise an alarm, preventing any potential collisions. Furthermore, in some cases the robot could reconfigure and accommodate the fault effects on its movement, allowing it to continue operating.

In the last decades various aspects relevant to mobile robot movement have been investigated. Mobile robots usually operate in cluttered areas, such as houses, office buildings or warehouses, and there has been an ongoing research on path planning and navigation in such environments [65, 66, 81, 87, 104, 108]. Navigating in dynamic or even static environments requires obstacle avoidance, which may be stationary or moving, and various methods have been proposed for this purpose [6, 9, 58, 112]. Due to modeling errors, disturbances and other factors, the robot could deviate from its planned path. For this reason, mobile robot control under uncertainty has been investigated in [7, 9, 20, 82], along with the trajectory tracking problem [77]. Some applications, for example in ware-

houses, require the deployment and control of multiple robots [52] in formations [26,78] or with limited communication [6,91].

In Systems and Control research, various theoretical results and methodologies on fault diagnosis have been proposed during the last two decades. These results aim to improve system reliability and fault tolerance, specifically by detecting, isolating, identifying and accommodating faults in linear and non-linear centralized and distributed systems, utilizing model-free and model-based approaches [10,21,46,49]. Fault diagnosis methodologies with learning capabilities have been proposed in the past years, which combine model-based analytical redundancy and adaptive approximation models, such as neural networks, to detect system and sensor faults and to learn the unknown fault dynamics [36,37,83,85,113]. By learning the unknown fault dynamics, and by isolating the type of fault and identifying its magnitude, it is possible to change the control input to accommodate the fault, during operation [113].

Fault detection in the context of mobile robots has attracted significant attention, and both model-free and model-based methods have also been proposed [71]. An early approach called gyrodometry, which even though does not directly detect faults, used the concept of sensor redundancy to maintain correct yaw estimate despite the presence of anomalies on the floor surface [11]. In another work, sensor redundancy is used to detect hardware faults as well as modeling errors [92]. Redundancy in the measurements is also used to detect faults, and along with localization, allow a robot to navigate through a forest area [76]. The method of multiple-model adaptive estimation was used by researchers to detect and identify sensor failures [89]. It contains a bank of Kalman filters and each one is based on a different failure model. Additionally, the bank contains a nominal model estimator. Processing the residual of each estimator, produces a probabilistic interpretation of the system's state. The lowest residual gives an indication which estimator best represents the true state, but this is not always the case because of noise. For this reason, a follow-up work describes a method using an off-line trained neural network to choose which fault has occurred [42]. Kinematic and dynamic models were developed to detect faults that change the wheel radius (flat tire, or wheel deformation) [28]. Their method allows fault detection despite parametric uncertainties in the model. Researchers have also utilized localization algorithms to detect faults [102]. Information from different sensors is fused to provide two or more position estimations, tracked with a Kalman filter. A statistical change detection test (CUSUM) is applied to detect deviations which indicate a fault in the system. Also, there are studies for the effects of faults in a formation of robots

and propose a fault tolerant control design to minimize them [105].

To overcome inaccuracies imposed by mathematical models, some methods take Bayesian approach for tracking the robot and environment states [106, 114]. They approximate the relevant probability distributions using particle filters. Implementations of such filters require computational power well within the limits of a robot. Their accuracy can be adjusted on demand through the increase/decrease of the number of particles, subject of course, to the available computational power. Even though more computationally demanding, neural networks were also utilized for detecting actuator and sensor faults [53]. Neural networks are also used to avoid the limitation imposed by uncertainties in analytical models [94]. In another work, researchers presents an experimental approach on a six degrees-of-freedom robot manipulator, which uses neural networks for analyzing the vibration condition on joints [32]. In the context of mobile robots, faults are detected with a time-delay neural network that is trained to classify faults based on the control input and sensor measurements [22].

There is also a survey about faults occurring on autonomous robots, collected from the teams participating in RoboCup competition [63,100]. The paper presents an adapted fault taxonomy suitable for autonomous robots and provides detailed information about the faults. From the perspective of anomaly detection [19,57], an online and data-driven approach uses Mahalanobis distance to detect abnormal operation and was tested, among other domains, on a mobile robot [70].

A closely related problem is the detection of wheel slip, occurring in rovers moving on off-road terrain [44,79]. Slip is the difference between the velocity measured by the wheel and the actual velocity [5]. It is important that the robot can detect slip reliably, otherwise it will deviate from its intended path. When available, GPS can be utilized to detect slip [4] and when the GPS signal becomes unrealiable slip-detection process switches to a model-based method with measurements from IMU and wheel encoders [109]. In this context, sensor signals are processed by a support vector machine trained a-priory, to distinguish between “normal” and “immobilized” status [47].

2.3 Localization in Mobile Robots

Localization is a fundamental problem in robotics, actively researched since it was first defined. It is a problem occurring in many practical robotic applications, where the robot is required to navigate within a certain area. While the robot navigates, knowledge of its

location is essential for executing accurate and efficient movements towards its destination.

From a theoretical point of view, two localization methods stand out which both were shown to work in practice as well and form the foundations of many localization methods to this date [93]. An Extended Kalman Filter (EKF) approach is the first method, which tracks the robot using Gaussian probability density function for both the pose and sensor measurement uncertainty. Because it is a parametric approach, it works fast and was shown to be accurate.

Furthermore, EKF can be used with non-linear transitional and measurement systems by linearizing around the working point with Taylor expansion. However, in a situation where high uncertainty exists, the pose distribution is affected by the non-linearities of the system and the filter could diverge. Moreover, EKF is a single-hypothesis belief method. Because of this, using it on a limited sensing robot could lead to divergence without recovery. EKF works best and more commonly used when distinguishable landmarks are present in the environment.

On the other hand, a multi-hypothesis belief method tracks multiple potential poses at the same time. This is useful when there is not enough information to allow the convergence on a single hypothesis. An extension of the EKF localization approach in this direction is the multi-hypothesis EKF [55]. Monte Carlo Localization (MCL) [25] is also a multi-hypothesis localization method where the pose belief is approximated using particle filters. MCL works with raw measurements instead of landmarks making it suitable for use with range sensors. Because the belief is represented by a multi modal distribution, the robot is aware of all the possible poses and their uncertainties. MCL works with non-linear models and it is relatively straight forward to implement. However, it is computationally more intensive than EKF.

There are successful implementations of both the MCL and EKF using either simulators or real robots. A method uses MCL on a mobile robot, which able to localize and track its position even after long trajectories (700m) [25]. The authors compare MCL against other approaches to localization and show how the robot is able to both globally and locally localize. In another publication [39], the concept of adaptive sampling was introduced, which improved the performance during global localization and extreme failure scenarios. The robots are equipped with laser finders or large arrays of sonar sensors.

Another approach uses beacons, whose precise locations are stored in a map, in order to localize [68]. The robot tracks its position by using EKF with beacon measurements

and the kinematic model. It should be noted that acquiring, installing and calibrating the extra hardware (beacons) requires additional resources.

The performance of two localization implementations was tested in two different environments, a corridor and a complex office setup [51]. The methods use EKF and Unscented Kalman Filter (UKF), which is an extension of EKF. UKF exhibited better results than EKF at the expense of higher computational complexity. They also showed how localization can be improved even further by calibrating the odometry of the robot. A high-profile robot was used, carrying an array of 8 sonar sensors.

Researchers attempted to replace the dense coverage of an expensive laser range finder by grouping together sequential sparse scans which allows the extraction of useful features [2, 8]. Particle filters are used for filtering the information and estimating the pose. The robot, equipped with few short range infrared sensors, is able to achieve accurate Simultaneous Localization And Mapping (SLAM). Similar sensors were used in another work to study localization abilities in a small static environment, decomposed as a grid map [3]. Because of the short range of the sensors, the robot has to travel close to the obstacles to get any useful information for localization. This could lead to a collision even with a small error. The experiments showed correct localization even though sometimes a large number of particles is used. Similar method was also used on a service robot which navigates in 20m by 20m area, equipped with an array of sonar sensors for obstacle avoidance and a laser range finder for scanning the environment [18].

Through a more theoretic perspective and with the assumption of perfect map and sensing, it was shown that the global localization problem can be solved with tactile sensors [80]. In a follow-up publication [69] the authors relax the assumptions of perfect control and sensing, something also considered probabilistically [31].

All of the aforementioned approaches solve the robot localization problem using more than one (usually large arrays) of sensors. The problem we consider involves a service robot which is left with only one operational infra-red range sensor. To the best of our knowledge this was not addressed so far, which encourages us to investigate and explore this problem.

Chapter 3

Optimizing Container Loading with Autonomous Robots

3.1 Introduction

In this work, we consider a problem associated with autonomous robots loading and delivering containers located in a warehouse facility. In terms of scheduling, an *assignment problem* needs to be solved indicating which robot should undertake each task. In terms of routing, a *coordination problem* needs to be addressed describing how the robots should coordinate their movements in order to avoid conflicts and reach their destination. The topology considered in this work has specific characteristics in terms of container arrangement, loading/unloading procedures and movement constraints. These characteristics introduce a number of challenges that distinguish the considered problem from problems addressing assignment and coordination of autonomous robots in relevant applications. First, the facility topology does not allow alternative paths for the loading and delivery of each container. In fact, a large percentage of containers may have conflicting path segments which makes conflict resolution highly complex. This requires very careful a-priori coordinated planning of the robot paths to avoid conflicts, contrary to most methods that attempt to resolve conflicts on-the-fly with the risk of ultimately not reaching a conflict-free solution. Second, achieving an efficient solution in this confined environment poses a significant challenge; online rerouting [59, 67, 110] in such a highly confined environment is expected to introduce further rerouting causing serious inefficiencies to the facility. The same issue extends to the assignment of containers to robots, as many approaches employ simple algorithms like first-come-first-served [84]. Third,

solutions provided by high-complexity a-priory approaches are not desirable in our case, because the dynamic nature of such facilities requires real-time decision making.

The contributions of this work are the following:

- Formulation and optimal solution of the assignment and coordination problem using Mixed Integer Linear Programming (MILP) tools
- Development of a low time-complexity polynomial algorithm for solving the assignment and coordination problem that provides close to optimal results
- Theoretical analysis of the computational performance of the heuristic algorithm with respect to the optimal solution and other conflict resolution strategies
- Reduced time-complexity algorithm for a special container arrangement

3.2 Problem Statement

The motivation of this work emanates from a common problem encountered in container terminals where containers are stacked into lanes: straddle carriers that handle the containers cannot move next to each other on adjacent lanes, as the space between lanes is enough only for one straddle carrier.This space limitation creates the challenge of simultaneously loading multiple containers without any movement conflicts between straddle carriers.

We consider a topology inspired from the one used in container terminals as illustrated in Fig. 3.1. The area of the facility is divided in two distinct regions, the storage region where all containers are stored and the free-moving region which is the shaded region in the figure. Containers need to be transported from the storage area to the loading area. A robot, similar to the straddle carrier, moves on two sets of wheels, located along its two sides. In order to load a container the robot needs to be positioned above it. This is achieved by placing its wheels to the two sides of the container and then moving into place. After lifting up the container, the robot starts moving towards its destination. Note that the robot is tall enough such that when carrying a container there is enough clearance underneath to move over other containers without hitting them. To save space, containers are stacked into long lanes, and the limited space between two consecutive lanes forms the transportation aisles of the warehouse. Initially the robots are located randomly in the area and after being assigned a task, they move through the free-moving

region to reach the entrance of their corresponding container lane. The free-moving region has no constrained paths therefore a robot can move freely in any direction. In this region, robots can maneuver and avoid collisions [29, 73].

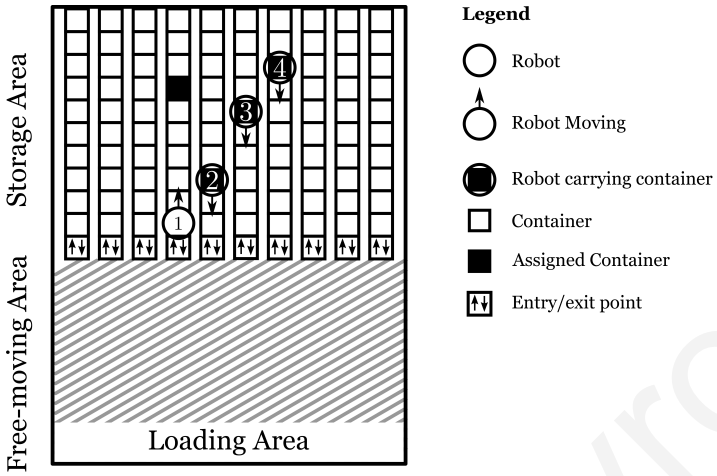


Figure 3.1: The specific facility topology considered in this work. The limited space between the container lanes can cause conflict between two opposite moving robots, as illustrated in this figure with robots 1 and 2.

The facility is equipped with a set $\mathcal{R}$ with $|\mathcal{R}| = N$ identical robots for transporting the containers. A task $s \in \mathcal{S}$ defines a specific container in the facility that needs to be transported from the storage to the loading area, with $\mathcal{S},\ |\mathcal{S}| = N$, being the set of tasks that the operator of the facility requests to be transported and loaded. Notice that for simplicity we have assumed that the number of robots is equal to the number of containers. In case the number of containers is larger than the number of robots, a sequence of problems can be solved by considering subsets of N containers.

In terms of storage space, this topology is efficient. However, due to the extremely limited space in the container storage region, special constraints exist that distinguish this problem from other general topology problems. The long container lanes have a single entry/exit point at the lower end. This means there is only one way to reach any given container and also it is not possible to switch lanes once a robot enters a lane. In order for a robot to switch a lane, it has to travel all the way to the exit of the current lane, move to the entrance of the desired lane and then move up to the desired container. Therefore, even two spatially close containers in different lanes are actually very far in terms of robot path.

Another constraint that arises due to the limited resources, is the movement of two or more robots in adjacent lanes. The aisles are wide enough for only one set of robot wheels,

so it is not possible for two robots to be on adjacent lanes, side-by-side. An example of this situation is shown in Fig. 3.1. Robot 1 is moving upwards while robot 2 is moving downwards. When they meet they enter into a conflict because each one opposes the progress of the other. Later on, robots 3 and 4 which also travel downwards will join the conflict, and this shows how fast it can escalate in the facility.

3.2.1 Problem Decomposition

Addressing our problem, requires the consideration of two related problems: assignment and coordination. Even though the two problems could be treated separately, joint consideration of these provides better results.

Regarding the assignment problem, the binary matrix $\mathbf{X} \in \{0,1\}^{n \times n}$ denotes the assignments of robots to tasks with element $x_{i,s}$ being equal to 1 if task s has been assigned to robot i and 0 otherwise. In addition, each robot should be assigned only one task and no two robots should be assigned to the same task, and every task should be assigned to only one robot i.e.:

$$\sum_{i=1}^{N} x_{i,s} = 1,\ s \in \mathcal{S} \text{ and } \sum_{s=1}^{N} x_{i,s} = 1, i \in \mathcal{R}.$$

Due to the constraints of the problem discussed earlier in this Section, a robot assigned to task s in lane $l_s \in \mathcal{L}$ may conflict with robots assigned to tasks on the same or directly adjacent lanes, i.e. $l_s - 1$, l_s and $l_s + 1$. We define set $\mathcal{C}_i$ as the set of robots which may conflict with robot i, i.e. those located on directly adjacent on the same lane with robot i. Because it is possible to have many robots moving on each lane, conflicts may propagate across the entire facility, so that conflict resolution has to be considered simultaneously for all lanes. The robots have to coordinate with respect to their entrance/exit order in different lanes as well as their movement strategy (when to move and when to stand still). Initially, each robot has to wait for a certain period of time before moving towards its defined lane in order to respect the decided entrance/exit order. During this period, the robot will remain within the free-moving area so that other robots are able to maneuver around it, avoiding a possible collision. It is also possible for a robot to wait at the location of its designated task for another robot to exit before exiting the specific lane.

3.2.2 Objective Function

After being assigned a task, each robot starts to move towards the entrance of the lane the container is positioned into. It proceeds to move and pickup the container, then exits the lane and finally delivers the container at the loading area. The time required for robot i to finish its complete movement is defined as:

$$T_i(\mathbf{x}_i, w_i^e, w_i^x) = \sum_{s \in \mathcal{S}} T_{i,s}^e x_{i,s} + w_i^e + T^l + 2 \sum_{s \in \mathcal{S}} T_s^v x_{i,s} + w_i^x + T^d, \tag{3.1}$$

where $\mathbf{x}_i$ is the assignment vector for robot i, i.e. $x_{i,s} = 1$ if robot i is assigned to task s and 0 otherwise, $T_{i,s}^e$ is the travel time to the lane entrance where task s is located, w_i^e is the *entrance waiting time*, i.e., the time before robot i begins its movement from its initial position in the free-moving area, T_s^v is the travel time from the lane entrance to the s-th container's position, and w_i^x is the *exit waiting time*, i.e., the waiting time at the location of the container before the robot travels towards the lane exit. The objective function is subject to the conflict-free condition which is described in detail in Section 3.3. All robots require the same time to load a container which is defined as T^l. In addition, T^d is the travel time from each lane exit to the loading area. Note that when robots enter and exit the lanes, they should keep a time-distance apart defined as guard time T^g for safety reasons.

There are different metrics for performance like *maximum throughput*, *minimum travel* and *even distribution of workload* [107]. In this work we consider maximum throughput, equivalent to the minimum completion time of all tasks. The completion time of all tasks can be otherwise stated as the maximum time any of the robots requires to complete its task, i.e.:

$$\Lambda(\mathbf{X}, \mathbf{w}^e, \mathbf{w}^x) = \max_{i \in \mathcal{R}} T_i(\mathbf{x}_i, w_i^e, w_i^x) \tag{3.2}$$

where $\mathbf{w}^e \in \mathbb{R}^N, \mathbf{w}^x \in \mathbb{R}^N$ are vectors representing the entrance and exit waiting times for each robot respectively. This equation is subject to having no conflicts between any of the robots. Under certain conditions, robot i may conflict with one or more robots defined as the set $\mathcal{C}^i$. The objective is to find a set of values $\{\mathbf{X}, \mathbf{w}^e, \mathbf{w}^x\}$ such that

$$\{\mathbf{X}^*, \mathbf{w}^{e*}, \mathbf{w}^{x*}\} = \underset{\{\mathbf{X}, \mathbf{w}^e, \mathbf{w}^x\}}{\operatorname{argmin}} \Lambda(\mathbf{X}, \mathbf{w}^e, \mathbf{w}^x) \tag{3.3}$$

In order to derive the optimal values for $\mathbf{X}, \mathbf{w}^e$ and $\mathbf{w}^x$, an optimization problem has to be solved. Note that times T^l and T^d are constant for all robots and all tasks, so that their presence does to affect the objective function; for this reason there are both hereafter ignored.

3.2.3 Illustrative Example

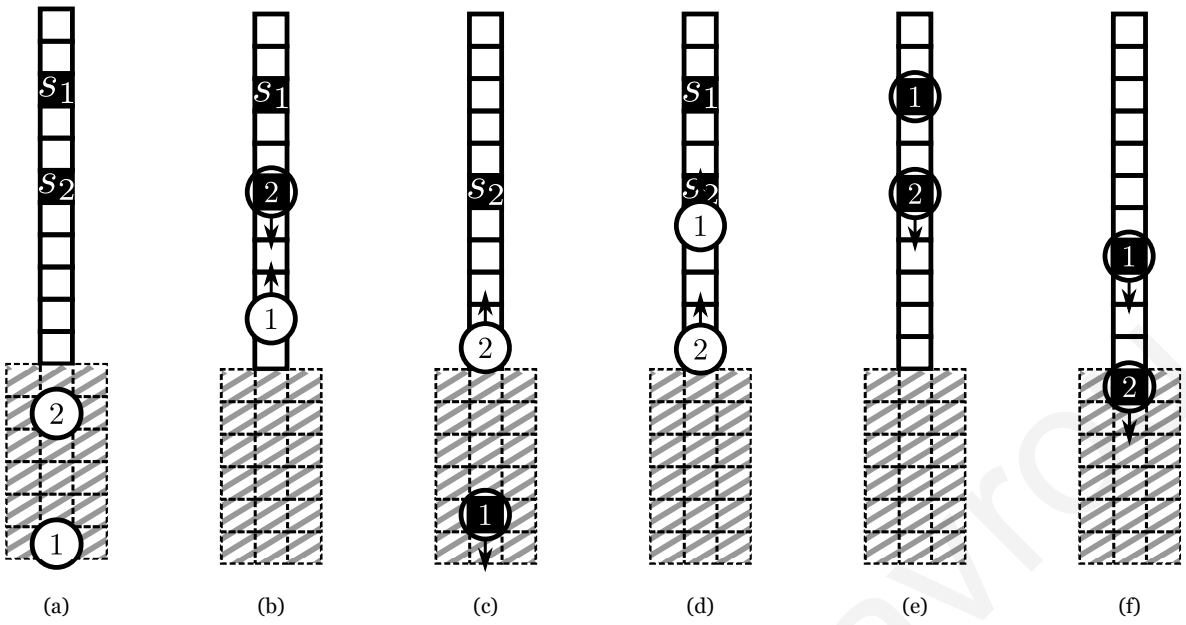


Figure 3.2: Illustrative example of a scenario with 2 robots, 2 containers and 1 container lane. (a) The initial configuration. (b) Scenario 1: Without any waiting times, the robots will reach a conflict. (c) Scenario 2: Robot 2 waits for robot 1 to carry its container out of the lane before entering. The time between robot 1 exiting and robot 2 entering respects the constraint of T^g. The scenario finishes successfully without conflicts (d) Scenario 3: Robot 2 waits for robot 1 to enter plus additional guard time T^g. (e) Both robots reach their tasks but robot 1 needs to wait for robot 2 to move further so the T^g constraint holds. (f) The scenario finishes successfully without any conflicts

To gain a better insight into the problem, in this paragraph we investigate three different cases that can arise in a simple configuration. We assume a small facility composed of only one lane of containers, two robots and two tasks s_1, s_2, and assume that each robot moves 1 grid cell per iteration. The initial configuration at iteration $k = 0$ is shown in Fig. 3.2(a). In this example, s_1 is assigned to robot 1 and s_2 to robot 2. For illustration purposes, we assume that robots movements are discrete, on grid cell at every iteration.

- First we consider the case where both robots start moving towards their assigned tasks without any waiting times, i.e. $\mathbf{w}^e = [0, 0]^\top$ and $\mathbf{w}^x = [0, 0]^\top$. At $k = 7$, robot 2 reaches its task, loads the container and is ready to start moving towards the exit. Robot 1 on the other hand is still traveling towards its container, as shown in Figure 3.2(b). Inevitably, the robots will conflict at $k = 9$.

- Let us consider a second case where the waiting times are adjusted as $\mathbf{w}^e = [0, 25]^\top$ in order to prevent conflicts and have the appropriate guard time $T^g = 4$. Robot 1 starts moving first towards its task while robot 2 waits. Robot 1 reaches its task at $k = 14$ and exits the lane at $k = 23$. Robot 2 will only start moving at $k = 25$ and enters the lane at $k = 27$ as shown in Fig. 3.2(c). The time between robot 1 exiting and robot 2 entering is enough to satisfy the guard time constraint. Finally, robot 2 will exit at $k = 38$.

- In the third case, we solve the same problem differently, using $\mathbf{w}^e = [0, 8]^\top$ and $\mathbf{w}^x = [2, 0]^\top$. Robot 1 starts moving first, while robot 2 waits. At $k = 6$ robot 1 enters the lane and at $k = 9$ robot 2 starts moving as well so it enters the lane at $k = 10$ so they respect T^g. This is shown in Fig. 3.2(d). After robot 1 loads its container, it has to wait until robot 2 reaches its task at $k = 16$ (shown in Fig. 3.2(e)) and start moving at $k = 17$. This way robot 2 exits at $k = 21$, shown in Fig. 3.2(f), and robot 1 at $k = 25$ and therefore respect the guard time.

This example demonstrates how the parameters of eq. (3.1) affect the solution of the problem. Two ways to solve the problem were presented, with $\Lambda_1 = 38$ and $\Lambda_2 = 25$, which means that the second solution is better than the first.

3.3 Abstract Time-Windows

In this section we describe how the time-line of a robot's movement can be graphically represented, inspired by the concept of *Time Windows* used in Operational Research. This representation is called *Abstract Time Windows* (ATW) and provides a method to study the movement of all robots collectively, detect conflicts and resolve them. At this point, for notational clarity we define $t_i^e = \sum_{s\in\mathcal{S}} T_{i,s}^e x_{i,s}$ and $\hat{t}_i^e = t_i^e + w_i^e$ as the time required for robot i to reach the storage region without/with initial waiting time, respectively. Also $t_i^v = \sum_{s\in\mathcal{S}} T_s^v x_{i,s}$ is the travel time that robot i requires to move from the entrance of the storage region to its assigned container. Finally, we define $t_i^x = 2t_i^v + t_i^e$ and $\hat{t}_i^x = t_i^x + w_i^e + w_i^x$, which denote the time from the start until the exit from the storage section without/with waiting.

An ATW represents the time-line of the movement of a specific robot and indicates all traveling and waiting times. An ATW is schematically represented by a line with distinct start and end parts as depicted in Fig. 3.3. Initially, an ATW describes the best-case

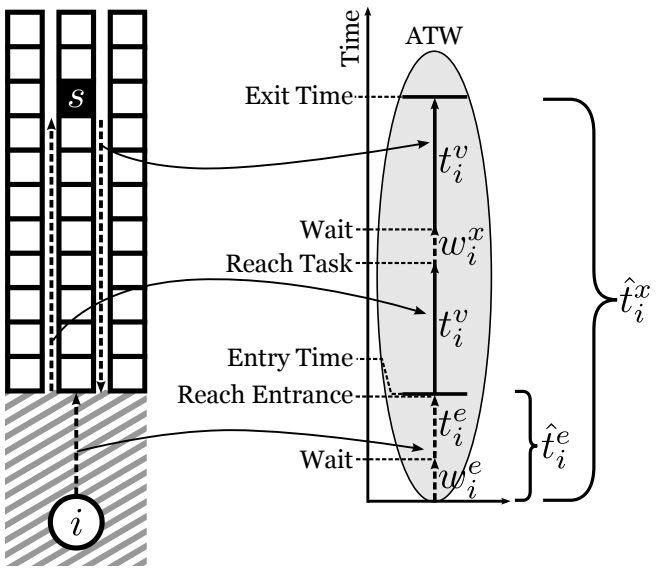


Figure 3.3: Schematic representation of robot movement in the container lane topology. The Abstract Time-Windows (ATW) transformation provides a tool for studying the problem, detecting conflicts between robots and for resolving conflicts with the allowed ATW operations.

scenario of a robot moving from its starting position to the container location and then exiting the lane. Therefore, given an assignment, an ATW starts at time t^e then extends by $2t^v$ depending on the location of the container and finally ends. If required, an ATW can be modified to include waiting times w^e and w^x. The ATW closes at $\hat{t}^x$.

In a facility where $N > 1$, ATWs are used to detect conflicts using the following condition.

Conflict-Free Condition (CFC): Consider any pair of conflicting robots i and k such that $t_i^e \leq t_k^e$ and $k \in \mathcal{C}_i$. Then, these robots can navigate with no conflicts with respect to each other, if exactly one of the following two cases is true:

1. $t_i^x + T^g < t_k^e$

2. $(t_i^e + T^g < t_k^e) \wedge (t_i^x - T^g > t_k^x)$

A solution satisfies the Conflict-Free Condition if all pairwise conflicting robot combinations satisfy the above condition.

This holds because if robot i enters the lane before robot k then there are only two possible outcomes in order to progress without conflicts. The first is when robot i exits safely before robot k enters and the second is when robot k safely enters after robot i and exits before robot i exits. In general, when having ATWs i and k then there are only 3

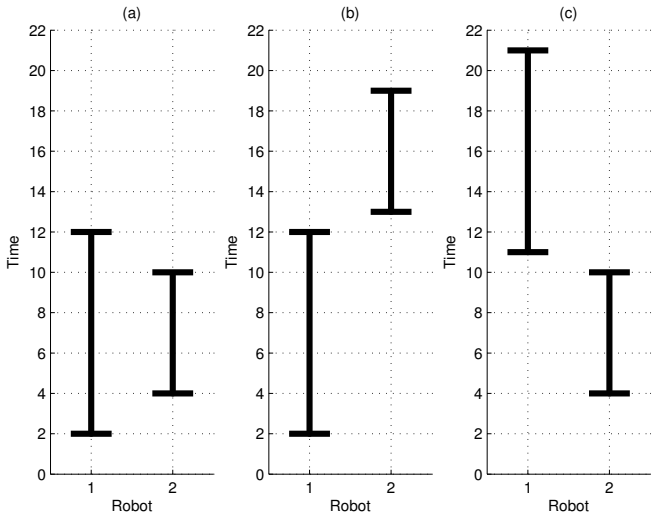


Figure 3.4: There are only 3 possible configurations between two ATWs that satisfy the CFC. (a) The two robots enter in parallel i.e. robot 1 enters followed by 2. After picking up their corresponding containers, robot 2 exits followed by robot 1. T^g constraint is respected in both entering and exiting. (b) In this configuration, robot 1 enters and exits, then robot 2 follows. The time between robot 1 exiting and robot 2 entering should respect the T^g constraint. (c) The same configuration as (b) only this time robot 2 is the first robot entering and exiting.

possible configurations between them that satisfy the CFC which are shown schematically in Fig. 3.4.

If CFC is violated then one can use the following two operations in order to resolve the conflict and make the solution CFC viable:

Shift: Shift ATW by increasing its w^e value.

Extend: Extend ATW by increasing its w^x value.

In other words, the ATW can be shifted upwards or it can be extended but it cannot be compressed. A shift upwards implies that the robot needs to wait before starting to move for w^e time units, while extension implies that the robot needs to wait at the task location for w^x time units. Fig. 3.5(a) shows an example with three ATWs where the CFC is violated in two cases. ATW 2 conflicts with ATW 3 and also ATW 1 conflicts with both ATW 2 and 3. To resolve the first conflict we can use operation Shift and shift ATW 3 as shown in Fig. 3.5(b), i.e. robot 3 will need to wait before starting its movement so that it enters the storage section after robot 2 has exited. To resolve the second conflict, ATW 1 is extended using operation Extend as shown in Fig. 3.5(c). The solution satisfies CFC.

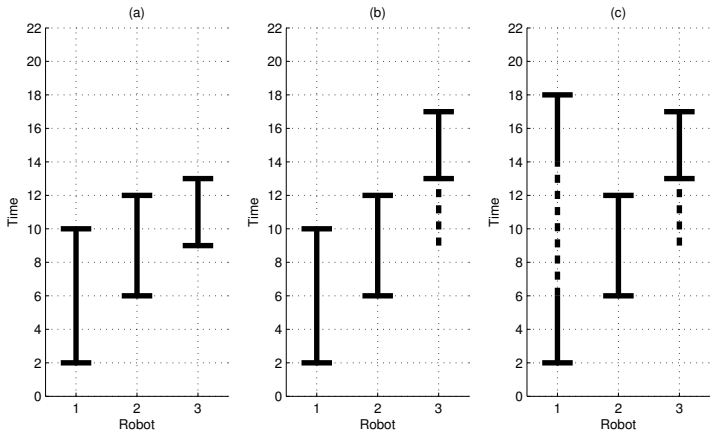


Figure 3.5: An example illustrating conflict resolution using ATWs. (a) The three initial ATWs are generated on the initial robot locations and their corresponding task locations. ATW 2 conflicts with ATW 3 and also ATW 1 conflicts with both ATW 2 and 3. (b) Using ATW operation Shift, ATW 3 is shifted to resolve the conflict with ATW 2. At this point, ATW 1 still conflicts with the other two ATWs. (c) Using ATW operation Extend, ATW 1 is extended to resolve both conflicts. The solution satisfies CFC.

3.4 MILP Formulation

The optimal solution of the considered problem is achieved by incorporating the derived CFCs into a Mixed-Integer Linear Programming (MILP) formulation. Towards this direction we first present some Lemmas necessary for the derivation of the problem formulation.

Lemma 1. *Let $z \in \{0, 1\}$. The logical condition "if $z = 1$ then $\sum_i x_i a_i \leq b$" can be expressed with the MILP constraint:*

$$\sum_i x_i a_i - b \leq M^u(1 - z) \tag{3.4}$$

where M^u is an upper bound on $\sum_i x_i a_i - b$.

Proof: The proof can be found in [72]. ■

Note that in case $z = 0$, the constraint is always satisfied and thus $\sum_i x_i a_i \lesseqgtr b$.

Lemma 2. *The logical condition "if $\sum_i x_i a_i \leq b$ then $z = 1$" can be expressed with the MILP constraint:*

$$\sum_i x_i a_i - b \geq (M^l - \epsilon)z + \epsilon, \tag{3.5}$$

where M^l is a lower bound on $\sum_i x_i a_i - b$ and ϵ is a small tolerance beyond which we regard the constraint not true.

Proof: The proof can be found in [72]. ■

In case $\sum_i x_i a_i > b$, then z can be either 0 or 1.

Lemma 3. *The **exclusive disjunction** condition "if $z = 1$ then $(\sum_i x_i a_{i,1} \leq b_1) \oplus (\sum_i x_i a_{i,2} \leq b_2)$" can be expressed with the following MILP constraints:*

$$\sum_i x_i a_{i,1} - b_1 \leq M_1^u (1 - z) + M_1^u (1 - \delta), \tag{3.6}$$

$$\sum_i x_i a_{i,2} - b_2 \leq M_2^u (1 - z) + M_2^u \delta, \tag{3.7}$$

where M_k^u is an upper bound on $\sum_i x_i a_{i,k} - b_k$ and $\delta \in \{0, 1\}$ indicates whether the first ($\delta = 1$) or the second constraint ($\delta = 0$) holds true.

Proof: When $z = 0$, constraints (3.6) and (3.7) become $\sum_i x_i a_{i,1} - b_1 \leq M_1^u + M_1^u (1 - \delta)$ and $\sum_i x_i a_{i,2} - b_2 \leq M_2^u + M_2^u \delta$ which always hold true by the definition of M_k^u. When $z = 1$ it is true that $\sum_i x_i a_{i,1} - b_1 \leq M_k^u (1 - \delta)$ and $\sum_i x_i a_{i,2} - b_2 \leq M_2^u \delta$. In case $\delta = 1$ then $\sum_i x_i a_{i,1} - b_1 \leq 0$ and $\sum_i x_i a_{i,2} - b_2 \leq M_2^u$, implying that constraint $\sum_i x_i a_{i,1} \leq b_1$ should be satisfied, while similar reasoning yields that the second constraint must hold true when $\delta = 0$.

The exclusive disjunction condition ensures that when the indicator variable z is turned on ($z = 1$) exactly one of the two constraints is satisfied. ■

Lemma 4. *The **conjunction** condition "if $z = 1$ then $(\sum_i x_i a_{i,1} \leq b_1) \wedge ... \wedge (\sum_i x_i a_{i,k} \leq b_k)$, $k = 1, ..., K$" can be expressed with the following MILP constraints:*

$$\sum_i x_i a_{i,k} - b_k \leq M_k^u (1 - z), k = 1, ..., K. \tag{3.8}$$

Proof: The proof can easily be obtained following similar steps to the proof of Lemma 3. ■

Note that Lemma 4 holds for an arbitrary number of constraints K.

Lemma 5. *The **conjunction** condition "if $(\sum_i x_i a_{i,1} \leq b_1) \wedge (\sum_i x_i a_{i,2} \leq b_2)$ then $z = 1$" can be expressed with the following MILP constraints:*

$$\sum_i x_i a_{i,k} - b_k \geq (M_k^l - \epsilon)\delta_k + \epsilon, k = 1, 2. \tag{3.9}$$

$$z \geq \delta_1 + \delta_2 - 1, \quad z,\ \delta_1,\ \delta_2 \in \{0, 1\}. \tag{3.10}$$

Proof: Notice that the condition is equivalent to "(if $\sum_i x_i a_{i,1} \leq b_1$ then $\delta_1 = 1$) $\wedge$ (if $\sum_i x_i a_{i,2} \leq b_2$ then $\delta_2 = 1$) $\wedge$ (if $(\delta_1 = 1 \wedge \delta_2 = 1)$ then $z = 1$)". Following Lemma 2, the first two conditions can be written as (3.9). Because $\delta_k \in \{0, 1\}$, $k = \{1, 2\}$, the third condition can be expressed as (3.10); this is true because only for $\delta_1 = 1$ and $\delta_2 = 1$, variable is forced to take a value $z = 1$, otherwise $z \geq 0$. ■

The MILP formulation of the considered problem is based on the fact that the assignment matrix $\mathbf{X}$, and the waiting vectors $\mathbf{w}^e$ and $\mathbf{w}^x$ must be optimally selected in order to minimize the total cost, defined as $f_T = \max_{i \in \mathcal{R}} \hat{t}_i^x$, and at the same time ensure that there is no conflict between any pair of robots. Note that f_T is equivalent to the cost defined in (3.2) with T^l and T^d ignored, as previously discussed in Sec. 3.2.2. Towards this direction, the approach taken is to define appropriate MILP constraints based on Lemmas 1 - 5 ensuring that whenever two robots are potentially conflicting, i.e. they have been assigned tasks in the same or neighboring lanes, exactly one of the two CFC cases holds true. The developed MILP formulation for the optimal solution of the considered problem is given below:

$$\min_{\{\mathbf{X},\, \mathbf{w}^e,\, \mathbf{w}^x,\, \zeta,\, \hat{\mathbf{t}}^x,\, \hat{\mathbf{t}}^e,\, \boldsymbol{\delta},\, \hat{\boldsymbol{\delta}},\, \tilde{\boldsymbol{\delta}},\, \boldsymbol{\psi},\, \boldsymbol{\xi}\}} \quad \zeta \tag{3.11a}$$

$$\text{s.t. } \hat{t}_i^x \leq \zeta,\ i \in \mathcal{R}, \tag{3.11b}$$

$$\hat{t}_i^x = 2 \sum_{s \in \mathcal{S}} T_s^v x_{i,s} + \hat{t}_i^e + w_i^x,\ i \in \mathcal{R}, \tag{3.11c}$$

$$\hat{t}_i^e = \sum_{s \in \mathcal{S}} T_{i,s}^e x_{i,s} + w_i^e,\ i \in \mathcal{R}, \tag{3.11d}$$

$$\sum_{s \in \mathcal{S}} x_{i,s} = 1,\ i \in \mathcal{R}, \tag{3.11e}$$

$$\sum_{i \in \mathcal{R}} x_{i,s} = 1,\ s \in \mathcal{S}, \tag{3.11f}$$

$$\sum_{s \in \mathcal{S}} x_{i,s} l_s - \sum_{s \in \mathcal{S}} x_{k,s} l_s - (M_2^l - \epsilon)\hat{\delta}_{i,k} \geq 1 + \epsilon,$$
$$i, k \in \mathcal{R},\ k \neq i, \tag{3.11g}$$

$$-\sum_{s \in \mathcal{S}} x_{i,s} l_s + \sum_{s \in \mathcal{S}} x_{k,s} l_s - (M_2^l - \epsilon)\tilde{\delta}_{i,k} \geq 1 + \epsilon,$$
$$i, k \in \mathcal{R},\ k \neq i, \tag{3.11h}$$

$$\delta_{i,k} \geq \hat{\delta}_{i,k} + \tilde{\delta}_{i,k} - 1,\ i, k \in \mathcal{R},\ k \neq i, \tag{3.11i}$$

$$\hat{t}_i^e - \hat{t}_k^e \leq M_1^u(1 - \psi_{i,k}) - T^g,\ i, k \in \mathcal{R},\ k \neq i, \tag{3.11j}$$

$$\psi_{i,k} + \psi_{k,i} = \delta_{i,k},\ i,k \in \mathcal{R},\ k < i, \tag{3.11k}$$

$$\hat{t}_i^x - \hat{t}_k^e \leq -T^g + M_1^u(1-\psi_{i,k}) + M_1^u(1-\xi_{i,k}),$$
$$i,k \in \mathcal{R},\ k \neq i, \tag{3.11l}$$

$$\sum_{s\in\mathcal{S}} T_s^v x_{k,s} - \sum_{s\in\mathcal{S}} T_s^v x_{i,s} \leq M_1^u(1-\psi_{i,k}) + M_1^u \xi_{i,k},$$
$$i,k \in \mathcal{R},\ k \neq i, \tag{3.11m}$$

$$\hat{t}_k^x - \hat{t}_i^x \leq -T^g + M_1^u(1-\psi_{i,k}) + M_1^u \xi_{i,k},$$
$$i,k \in \mathcal{R},\ k \neq i, \tag{3.11n}$$

$$w_i^e \geq 0,\ w_i^x \geq 0,\ i \in \mathcal{R}, \tag{3.11o}$$

$$\delta_{i,k}, \hat{\delta}_{i,k}, \tilde{\delta}_{i,k}, \psi_{i,k}, \xi_{i,k} \in \{0,1\},\ i,k \in \mathcal{R}, k \neq i, \tag{3.11p}$$

$$x_{i,s} \in \{0,1\},\ i \in \mathcal{R},\ s \in \mathcal{S}. \tag{3.11q}$$

In formulation (3.11) constants M_1^u and M_2^l denote an upper and lower bound of quantities $\hat{t}_k^x - \hat{t}_i^x + T^g$ and $x_{i,s}l_s - \sum_{s\in\mathcal{S}} x_{k,s}l_s - 1$, respectively. In particular, M_1^u is set equal to the objective value derived from the heuristic algorithm presented in Section 3.5, while $M_2^l = -\max_{s\in\mathcal{S}}\{l_s\} - 2$. Expressions (3.11a)-(3.11b) are equivalent to the desired objective $\min \max_{i\in\mathcal{R}} \hat{t}_i^x$, while equalities (3.11c) and (3.11d) define the values for $\hat{t}_i^x$ and $\hat{t}_i^e$, respectively. Assignment constraints (3.11e) and (3.11f) indicate that only one task is assigned to each robot and that each task is assigned to exactly one robot, respectively. Constraints (3.11g) - (3.11i) indicate whether robots i and k are in consecutive lanes which is expressed with the logical constraint, "if $|\sum_{s\in\mathcal{S}} x_{i,s}l_s - \sum_{s\in\mathcal{S}} x_{k,s}l_s| \leq 1$ then $\delta_{i,k} = 1$". This condition is equivalent to condition "if $(\sum_{s\in\mathcal{S}} x_{i,s}l_s - \sum_{s\in\mathcal{S}} x_{k,s}l_s \leq 1) \wedge (-\sum_{s\in\mathcal{S}} x_{i,s}l_s + \sum_{s\in\mathcal{S}} x_{k,s}l_s \leq 1)$ then $\delta_{i,k} = 1$ which can be expressed using (3.11g) - (3.11i) following Lemma 5. Constraint (3.11j) ensures that if $\psi_{i,k} = 1$ then $\hat{t}_k^e \geq \hat{t}_i^e + T^g$ according to Lemma 1. Constraint (3.11k) establishes the precedence between robots i and k; if the particular pair of robots is potentially conflicting ($\delta_{i,k} = 1$) then either $\hat{t}_k^e \geq \hat{t}_i^e + T^g$ or $\hat{t}_i^e \geq \hat{t}_k^e + T^g$ indicating that robot i precedes robot k with the necessary guard time and vice-versa. Constraints (3.11l) - (3.11n) ensure that if $\psi_{i,k} = 1$ then exactly one of the non-conflict conditions holds (Lemmas 3, 4); CFC case 1 holds when $\xi_{i,k} = 1$ and CFC case 2 when $\xi_{i,k} = 0$. Finally, constraints (3.11o) - (3.11q) ensure the non-negativity of the waiting times and the binary nature of the indicator and assignment variables. The fact that the resulting formulation belongs in the class of MILP problems indicates that the problem is NP-hard to solve. Hence, the MILP solution approach is not

suitable for real-time execution; nonetheless, it serves as the baseline for the performance evaluation of the heuristic approach developed in the next section.

This problem was solved using the Gurobi solver, using the default solver parameters, with maximum execution time set to half an hour. The value ϵ was chosen to be 10^{-6} which is the feasibility tolerance of the Gurobi solver. Choosing a smaller ϵ parameter, the produced solution will have smaller constraint violations. However, this could lead to larger number of iterations in order to reach a solution, hence larger execution times.

3.5 Low Time-complexity Solution

In this section, we develop a low time-complexity heuristic approach suitable for real-time solution of the considered problem. The approach taken is to decouple the assignment and coordination problems and solve them sequentially.

To deal with the *assignment problem*, the objective is to find the best allocation of robots to tasks that minimizes the assignment cost, defined as $f_A = \max_{i \in \mathcal{R}} t_i^x$, which denotes the time at which all tasks have been completed ignoring potential conflicts; this issue will be addressed in the coordination problem. A task defines a specific container that needs to be transported from the storage area to the loading area, hence the objective of the assignment problem is to minimize the travel time needed to transport all designated containers to the loading area, with the assumption that all robots have the same performance abilities, i.e. they can reach the same speeds. Since no conflicts are considered at this point, w^e and w^x are always zero, therefore the time robot i takes to complete its task is now reduced to $t_i^x = t_i^e + 2t_i^v$ i.e. the time it takes robot i to reach the lane entrance of its assigned container, and the time to reach the container and back. The cost matrix **C** has elements

$$c_{is} = T_{i,s}^e + 2T_s^v$$

which represent the time required by robot i to complete a candidate task s. The assign-

ment problem is defined as

$$\begin{aligned}
&\min_{\mathbf{x}} \max_{1\le i,s\le n} c_{is}x_{is} \\
\text{s.t.} &\sum_{i=1}^{N} x_{i,s} = 1,\ s \in \mathcal{S}, \\
&\sum_{s=1}^{N} x_{i,s} = 1,\ i \in \mathcal{R}, \\
&x_{i,s} \in \{0,1\},\ i \in \mathcal{R},\ s \in \mathcal{S}.
\end{aligned}$$

This problem is equivalent to the *Linear Bottleneck Assignment Problem* (LBAP) [14, 40] which describes the assignment of N jobs to N machines such that the latest completion time is as early as possible. This problem can be solved using a threshold algorithm in $O(n^{2.5}/\sqrt{\log n})$ time [14, Theorem 6.4].

To deal with the *coordination problem*, it is important to find an efficient strategy that shifts and/or extends ATWs in order to complete all tasks with no conflicts, in order to minimize the coordination cost defined as $f_C = \max_{i\in\mathcal{R}}(w_i^e + w_i^x - \gamma_i)$ where $\gamma_i = \max_{j\in\mathcal{R}} t_j^x - t_i^x$, $\gamma_i \ge 0$, is the waiting time that can be added without increasing the total cost. Next, we examine the two-robot problem in order to select an appropriate coordination strategy for the multi-robot problem. To solve the two-robot problem, four cases need to be considered depending on the relative arrangement of the corresponding ATWs: (i) $(t_i^e < t_k^e) \wedge (t_i^v > t_k^v)$ (ii) $(t_i^e > t_k^e) \wedge (t_i^v > t_k^v)$ (iii) $(t_i^e > t_k^e) \wedge (t_i^v < t_k^v)$ (iv) $(t_i^e < t_k^e) \wedge (t_i^v < t_k^v)$. Note that we only need to examine cases (i) and (ii), as cases (iii) and (iv) are their mirror cases which can be addressed by swapping indices i and k. Any initial configuration of two ATWs has 3 possible solutions with different performance (solution 1, 2 and 3) presented in Fig. 3.4. Solutions 1-3 are obtained by applying strategies 1-3, respectively:

Strategy 1 shifts the ATW with the smallest t^v and extends the ATW with the largest t^v

Strategy 2 only shifts the ATW with the smallest t^v

Strategy 3 only shifts the ATW with the largest t^v

An illustration of the strategies is shown in Fig.3.6. Strategies will shift and/or extend the ATWs only when necessary, in order to achieve the best performance in terms of the objective value. The following lemmas are used to choose the best strategy for coordination.

Lemma 6. *The best case scenario performance of Strategy 2 is better than the worst case scenario performance of Strategy 1 by T^g for case (i) and $T^g - 2t_i^v$ for case (ii).*

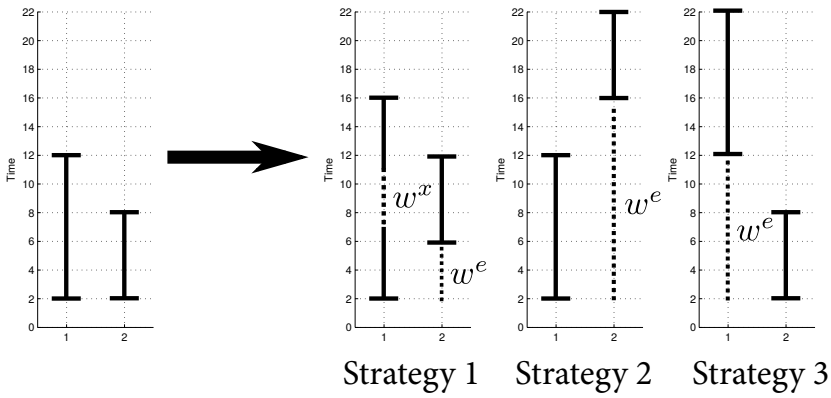


Figure 3.6: Illustration of the three strategies used for arranging ATWs

Proof: The proof can be found in Appendix B.1. ■

Lemma 7. *The best case scenario performance of Strategy 3 is better than the worst case scenario performance of Strategy 1 by less than $T^g - 2t_i^v$ for case (i) and smaller than $2T^g$ for case (ii).*

Proof: The proof can be found in Appendix B.2. ■

Lemma 8. *The best case scenario performance of Strategy 1 is better than the worst case scenario performance of Strategy 2 by $2t_k^v + T^g$ for case (i) and $2t_k^v + T^g$ for case (ii).*

Proof: The proof can be found in Appendix B.3. ■

Lemma 9. *The best case scenario performance of Strategy 1 is better than the worst case scenario performance of Strategy 3 by $t_k^e - t_i^e + t_k^v + T^g$ for case (i) and $t_k^e + 2t_k^v + T^g$ for case (ii).*

Proof: The proof can be found in Appendix B.4. ■

Based on the above analysis, we have chosen Strategy 1 for the solution of the coordination problem. That is because, Lemmas 6 and 7 indicate that when Strategies 2 and 3 are better than Strategy 1, the additional time imposed if Strategy 1 is used instead, is bounded by $2T^g$ which is very small compared to the other parameters. Lemmas 8 and 9 further indicate that the benefit of using Strategy 1 instead of the other strategies is more substantial because it depends on the values of $t^v >> T^g$. In sum, Strategy 1 provides the best performance from the three strategies irrespective of the ATW configuration. Furthermore, by solely using Strategy 1 in the algorithm, the time-complexity is simplified when the problem scales up as $|\mathcal{R}|$ increases. Strategy 1 involves the sequential execution of the Shift and Extend operations; hence, our coordination algorithm also involves

the sequential execution of these operations as described in Algorithm 1. Note that Algorithm 1 is not expected to always be optimal, as the optimal solution may involve a mixture of strategies regarding the relative arrangement of different ATWs.

In the ATW context, the first part of the algorithm resolves conflicts at the entering part of ATWs. The algorithm starts by sorting the robot ATWs with respect to the t^v time, in descending order i.e. the ATW with the larger t^v value is placed first. Starting from the first ATW in the set, the conflicting ATWs are determined and stored in the $\mathcal{C}_r$ set. Depending on which conditions hold, the appropriate waiting time w^e is calculated for robot ATW k. Sorting the ATWs in the $\mathcal{R}^{\overleftarrow{s}}$ reduces the complexity because when Shift is applied starting from the ATW with the largest t^v, ensures that a specific ATW will not have a conflict later and therefore needs to be checked only once. This applies because according to lemmas 8 and 9, when two ATWs are conflicting, it is more efficient to apply Shift (add waiting time w^e) to the ATW with the smallest t^v in order to resolve the conflict. Therefore, starting from the ATW with the largest t^v, any arising conflicts will affect ATWs that have yet to be examined; hence the first part of the algorithm will terminate in one iteration.

The second part of the algorithm resolves the conflicts on the exiting part of ATWs. This time, the t^v values are sorted in ascending order to obtain set $\mathcal{R}^{\overrightarrow{s}}$. The Extend operation is applied starting from the ATW with the smallest t^v, ensures that an examined ATW need not be re-examined. This applies because according to lemmas 8 and 9, when two ATWs are conflicting, it is more efficient to apply Extend (add waiting time w^x) to the ATW with the largest t^v in order to resolve the conflict. Depending on what conditions hold between the two ATWs, waiting time w^x is calculated for robot ATW i. When this operation finishes, all ATWs are conflict free and the solution of the problem is reached.

The algorithm involves the Shift and Extend operations which are of the same time-complexity. Each operation requires sorting the elements which is of time-complexity $O(n \log n)$ and two nested for-loops each of time-complexity $O(n^2)$. Hence, the total time-complexity of Algorithm 1 is dominated by the nested for-loops resulting in $O(n^2)$.

To theoretically examine the performance of Algorithm 1 with respect to the optimal, we define f_A^*, f_C^* and $f_T^* = f_A^* + f_C^*$ as the assignment, coordination and total cost resulting from the optimal solution, respectively. Also, let f_A^h, f_C^h and $f_T^h = f_A^h + f_C^h$ denote the assignment, coordination and total cost resulting from LBAP and Algorithm 1, respectively, such that $f_T^* \leq f_T^h$. Next, we derive theoretical bounds for the assignment, coordination and total cost of the heuristic algorithm.

Algorithm 1 Coordination Algorithm

/* *Operation Shift* */
$\mathcal{R}^{\overleftarrow{s}} = \text{sort}(\mathcal{R}, \mathbf{t}^v, \text{'descending'})$
for $r \in \mathcal{R}^{\overleftarrow{s}}$ **do**
  for $c \in \mathcal{C}_r$ **do**
    $i = \text{argmax}\,\{t_z^v | z \in \{r, c\}\}$;
    $k = \text{argmin}\,\{t_z^v | z \in \{r, c\}\}$;
    if $t_i^e + T^g > t_k^e$ **then**
      $w_k^e = t_i^e - t_k^e + T^g$;
    else
      $w_k^e = 0$;
    end if
  end for
end for
/* *Operation Extend* */
$\mathcal{R}^{\overrightarrow{s}} = \text{sort}(\mathcal{R}, \mathbf{t}^v, \text{'ascending'})$
for $r \in \mathcal{R}^{\overrightarrow{s}}$ **do**
  for $c \in C_r$ **do**
    $i = \text{argmax}_i\,\{t_z^v | z \in \{r, c\}\}$;
    $k = \text{argmin}_i\,\{t_z^v | z \in \{r, c\}\}$;
    if $t_k^x + w_k^e + T^g > t_i^x$ **then**
      $w_i^x = t_k^x + w_k^e - t_i^x + T^g$;
    else
      $w_i^x = 0$;
    end if
  end for
end for

Lemma 10. *For the assignment cost, it is true that* $f_A^h \leq f_A^*$.

Proof: The heuristic algorithm solves the assignment problem using LBAP which by definition provides the minimum assignment cost. In comparison, the MILP algorithm yields the optimal solution that provides the minimum total cost but not necessarily minimum assignment cost. ■

Lemma 11. *The solution of Algorithm 1 yields a coordination cost* $f_C^h \leq 2NT^g$.

Proof: The proof can be found in Appendix B.5. ■

Theorem 1. *The solution of the heuristic algorithm yields an additional total cost of at most* $2NT^g$ *when compared with the optimal i.e.* $f_T^h \leq f_T^* + 2NT^g$.

Proof: From Lemma 10 it is true that $f_A^h \leq f_A^*$; also, from Lemma 11 it is true that $f_C^h \leq 2NT^g \leq 2NT^g + f_C^*$, as $f_C^* \geq 0$. Adding the two inequalities yields $f_A^h + f_C^h \leq f_A^* + f_C^* + 2NT^g$ which completes the proof. ■

Corollary 1. *It is true that* $f_T^h \to f_T^*$ *as* $T^g \to 0$.

The above results provide bounds for the performance of the heuristic algorithm with respect to the optimal and indicate that as the guard time tends to zero the performance tends towards optimality.

Results from the strategy analysis provide useful insight on when the heuristic approach performs best. From Strategy 1 analysis of two ATWs i and k in conflicting lanes, we know that waiting times become zero when $(t_i^e + T^g < t_k^e) \wedge (t_k^e - t_i^e + T^g < 2t_i^v - 2t_k^v)$. In order to satisfy those inequalities, task i which is the farthest in the storage area should be assigned to the robot nearest to the entrance, while task k which is the nearest in the storage area should be assigned to the robot farthest from the entrance. At the same time, tasks i, k should be chosen such that the second inequality is satisfied. From Lemma 10 we know that $f_A^h \leq f_A^*$ and if both above inequalities are satisfied there will not be any added cost due to coordination i.e. $f_C^h = 0$, hence the performance would be optimal. This reverse approach provides guidelines on how to configure containers in the storage area, such that the heuristic approach will provide close-to or equal-to optimal results. One could use the above inequalities to design a selection algorithm that would return the specific containers to be transported based on the robot locations in the warehouse.

3.5.1 Special Case: One-lane Approach

A special case of the problem is when all tasks are located in a single lane. In this case the assignment part of the problem is simplified and leads to faster computation [99]. To solve this problem efficiently we exploit the fact that every robot has to reach the same lane entrance. Therefore, we can derive the relative closeness of robots from the tasks based only on their distance from the entrance e.g. if i is closer to the entrance than k then i is closer to any task compared to k.

In this case the robot set is sorted with respect to the distance from the entrance such that $t_i^e < t_{i+1}^e$ defined as $\mathcal{R}^e$. The task set is sorted as well such that $t_s^v > t_{s+1}^v$ defined as $\mathcal{S}^e$. When these two vectors have this specific arrangement, the cost matrix $c_i = \sum_{s\in\mathcal{S}} T_{i,s}^e x_{i,s} + T^l + 2\sum_{s\in\mathcal{S}} T_s^v x_{i,s}$ fulfills the *bottleneck Monge property*. Therefore the assignment matrix $\mathbf{X} = \mathbf{I}_n$ provides the optimal solution [14] by assigning the farthest task to the closest robot, the second farthest task to the second closest robot and so on. The complexity of the assignment reduces to $O(n \log n)$ dominated by sorting.

The coordination algorithm is similar to the algorithm 1 and the two operations 1 and 2 are executed in sequence: As with assignment, resolving conflicts is also simplified in the one-lane special case because all robots conflict with each other. Because of this it is only required to iterate through the robot list only once for each operation. This reduces the complexity of the coordination algorithm to $O(n)$.

3.6 Simulation Results

In this section we present results for evaluating the performance of the proposed method. Simulations were executed on a Intel Core i7-4790K CPU at 4.0GHz with 16GB of RAM. The simulated warehouse was designed based on the topology of Fig. 3.1. The length of the warehouse is 300 m, with the lower 100 m being the free moving space. The width of each lane is 3 m and they are spaced 5 m apart leaving 1 m for the aisles. The warehouse width depends on the number of lanes used and because that number varies in our experiments so does the warehouse width. To compare the performance of the heuristic algorithm relative to the MILP one we employ the relative optimality gap metric which is defined as:

$$\text{Relative Optimality Gap} = \left(\frac{\text{Heur. Obj. Value}}{\text{MILP Obj. Value}} - 1\right) \times 100\%$$

Algorithm 2 Assignment and Coordination Algorithm for the One-lane Case

$\mathcal{R}^{\overrightarrow{e}} = \text{sort}(\mathcal{R}, \mathbf{t}^e, \text{'ascending'})$
$\mathcal{S}^e = \text{sort}(\mathcal{S}, \mathbf{t}^v, \text{'descending'})$
for $i = 1 \dots n$ **do**
  $x_{\mathcal{R}^{\overrightarrow{e}}(i), \mathcal{S}^e(i)} = 1$
end for
for $r \in \mathcal{R}^{\overrightarrow{e}}$ **do**
  if $t^e_r + T^g > t^e_{r+1}$ **then**
    $w^e_{r+1} = t^e_r - t^e_{r+1} + T^g$
  else
    $w^e_{r+1} = 0$
  end if
end for
$\mathcal{R}^{\overleftarrow{e}} = \text{sort}(\mathcal{R}, \mathbf{t}^e, \text{'descending'})$
for $r \in \mathcal{R}^{\overleftarrow{e}}$ **do**
  if $t^x_{r+1} + w^e_{r+1} + T^g > t^x_r$ **then**
    $w^x_r = t^x_{r+1} + w^e_{r+1} - t^x_r + T^g$
  else
    $w^x_r = 0$
  end if
end for

Figure 3.7 is the cumulative distribution function of the relative optimality gap between the optimal and heuristic solution, which is the result of a set of 1200 problems with $N = 20$ and 10 lanes. The heuristics algorithm solved 40% of the problems optimally and 95% of them with at most 6% optimality gap.

Figure 3.8(a) demonstrates the relative optimality gap with respect to the number of robots in the form of a box-plot[1]. To obtain the results, we simulated 200 problems for each $N = \{5, 10, 15, 20, 25\}$. The initial conditions of each simulation i.e. the positions of robots and tasks, were chosen randomly. The number of lanes is 4 and is kept constant for

[1]The bottom and top of each box indicate the first and third quartiles (25% and 75%) of a ranked data set, while the horizontal line inside the box indicates the median value (second quartile). The horizontal lines outside the box indicate the lowest/highest datum still within 1.5 inter-quartile range of the lower/upper quartile; for normally distributed data this corresponds to approximately 0.35%/99.65%. Red crosses indicate the outliers.

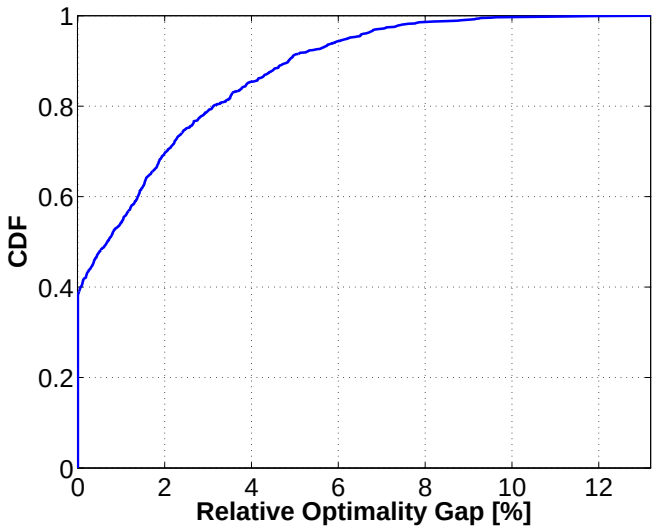
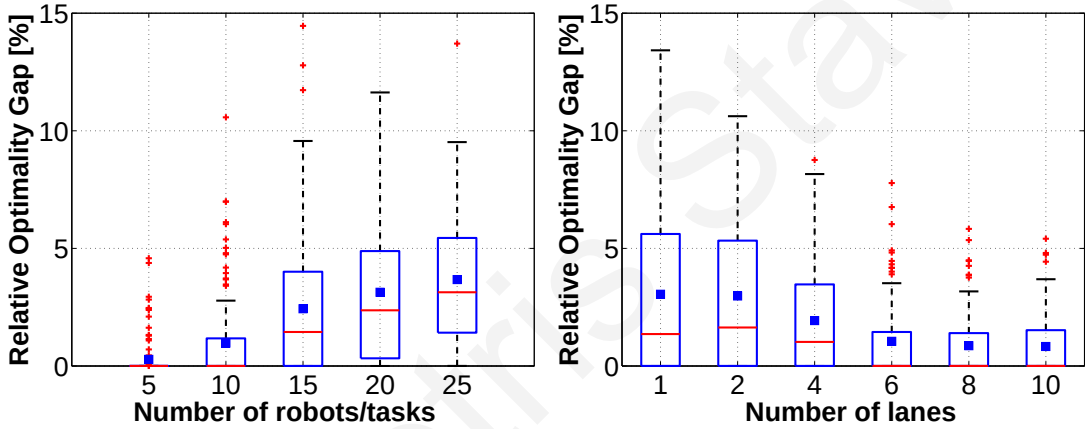


Figure 3.7: The cumulative distribution function of the relative optimality gap between the optimal and heuristic solution.



(a) (b)

Figure 3.8: Relative optimality gap with respect to the number of (a) robots/tasks, and (b) lanes.

all simulations. In all considered cases the mean optimality gap is less than 4%, while 75% and 100% of the problems in each case have relative optimality gap within 6% and 15%, respectively. As the number of robots increases the relative optimality gap also increases. This is because as the number of robot increases, more robots have to move in conflicting lanes resulting in more conflicts, making the problem harder to solve.

The next experiment demonstrates the relative optimality gap with respect to the number of lanes while the number of robots is kept constant at $N = 15$. The results presented in Fig. 3.8(b), show that 75% of all problems in each case have at most 6% gap, while the mean value of each group is below 3%. As the number of lanes decreases i.e. the average number of robots per lane increases (similarly to Fig 3.8(a)), it causes more

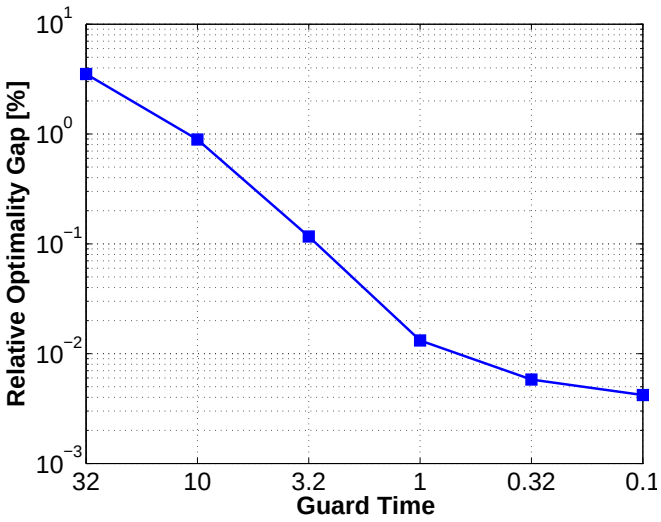
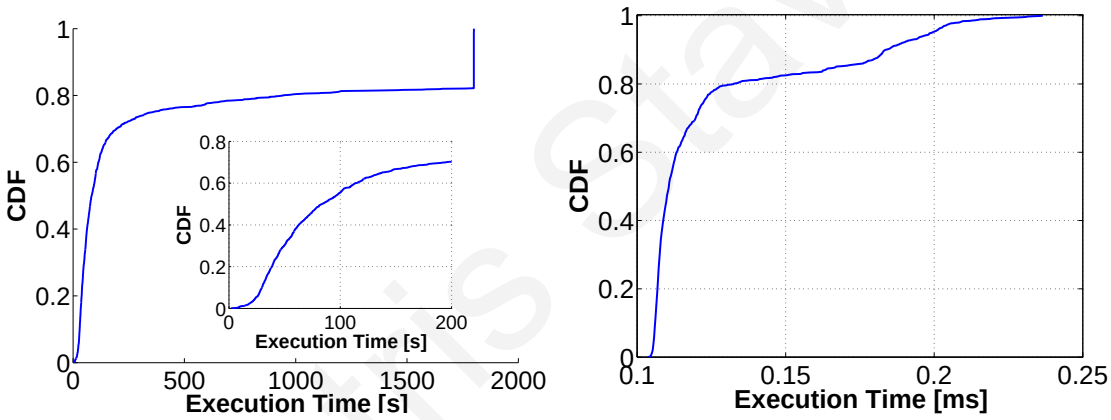


Figure 3.9: Relative optimality gap for varying guard time; the results are averaged over 200 random problems with $N = 15$ and 10 lanes.



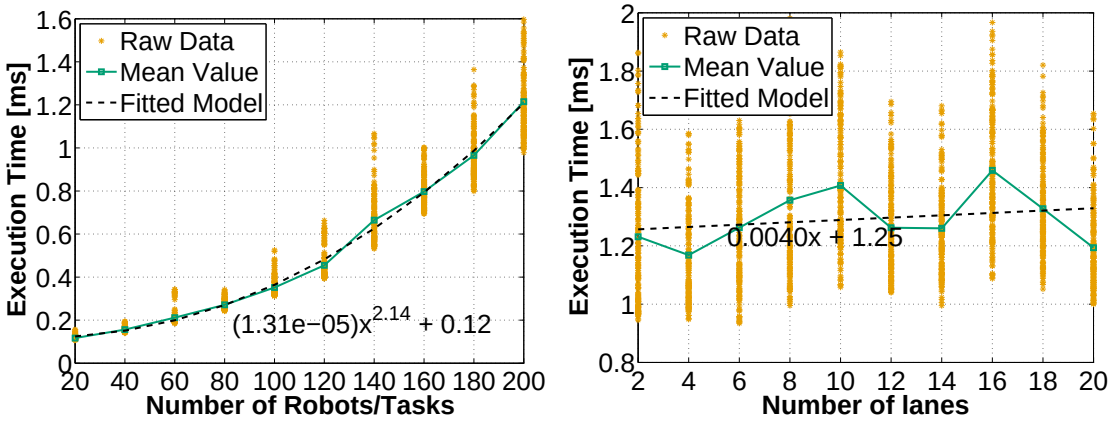
(a) (b)

Figure 3.10: Cumulative distribution function of the execution time of the (a) MILP solver, and (b) proposed heuristic solution.

conflicts and therefore more gap between the optimal and heuristic solution.

Figure 3.9 depicts relative optimality gap for varying guard time. It can be seen that the relative optimality gap drops from 5% at $T^g = 32$ to 0.004% at $T^g = 0.1$, about three orders of magnitude. This behavior is in agreement with the theoretical result that as the guard time decreases the performance of our heuristic algorithm tends to optimality (Corollary 1).

Execution time is an important criterion for the ability of an algorithm to perform in real time. The results presented in Fig. 3.10(a), show that in 70% of problems the MILP solver required more that 50 s to reach a solution. Also about 10% of problems took between 200 and 1800 s while 20% of problems did not finish within the time limit



(a) (b)

Figure 3.11: Heuristic algorithm execution time with respect to the number of (a) robots/tasks, and (b) lanes. Each point on the graph represents a different simulation.

of 30 minutes. From these results we conclude that the MILP solver is not suitable for real-time application due to the large execution times and unpredictable performance, evident from the large variability of the execution time of all problems. Figure 3.10(b) shows the heuristic performance on the same problem set of Fig. 3.10(a). All problems were solved by the algorithm within 0.25 ms, with fastest solution reached within 0.1 ms. Hence, the developed algorithm is very fast and with small execution time variability making it suitable for real-time applications. Compared to the MILP solver, the proposed algorithm is over one million times faster (six-orders of magnitude).

To demonstrate that the implemented algorithm complexity follows the theoretical complexity analysis, execution times were recorded for 200 simulations of varying N as shown in Fig 3.11(a). Each point in the figure denotes the execution time for each simulation, while the solid line depicts the mean values which illustrates the increase trend. The dashed line shows the nonlinear least-squares fit of the polynomial model $a_1 x^{a_2} + a_3$. The fitted value of the exponent $a_2 = 2.14$ indicates that the execution time scales almost quadratically to the number of robots/tasks, as expected from the theoretical time-complexity analysis in Section 3.5. Figure 3.11(b) is the result of the same method but with varying lanes number. As shown from the characteristics of the fitted model, the execution time is almost independent from the number of lanes. Finally, Figs. 3.11(a) and 3.11(b) show that the developed algorithm can solve problems with a large number of robots/tasks or lanes very fast (in less than 1.5 ms for 200 robots), illustrating scalability and potential for real-time deployment.

3.7 Concluding Remarks

In this work we investigate a problem associated with transporting a set of containers from the storage to the loading area of a warehouse using autonomous robots. We define the considered problem incorporating the constraints imposed by the topology. The problem is formulated and solved optimally using Mixed Integer Linear Programming tools. Furthermore, a low time-complexity heuristic algorithm is developed and theoretically investigated. It is shown that its performance relative to optimality is upper bounded by the product of the number of robots and the guard time, implying that the heuristic tends to optimality when the guard time tends to zero. Simulation results indicate 40% of the problems were solved optimally using the heuristic approach, while in general a problem is solved within 5% relative optimality gap. The added cost of the non-optimal solutions is small compared with the improvements in execution time. Our results show that the heuristic approach executes in the order of milliseconds and is six orders of magnitude faster than a state-of-the-art Mixed Integer Linear Programming solver making it suitable for real-time applications.

Chapter 4

Fault Diagnosis on a Differential-drive Mobile Robot

4.1 Introduction

In this chapter, we demonstrate actuator fault detection using an analytical redundancy approach on an iRobot Roomba-based robotic platform but we point out that our approach can be applied to any other two-wheeled mobile robot. In the implemented approach, an observer tracks the velocity error between an estimate made based on sensor readings, and the nominal robot dynamics. Identifying the uncertainties is a critical task towards the practical implementation. Realistically, there are several uncertainty sources which need to be taken into account, so that their bounds are used to generate an adaptive threshold, which sets the limit of the estimation error. Faults are detected when the estimation error exceeds this threshold. Additionally the magnitude of the fault is identified online, providing important information about the severity of the fault. Furthermore we propose an extension to the method that addresses the problem of false negatives (failure to detect the fault) when fault magnitude is low. The method was designed to work with simple odometry sensors. Such sensors, have been in use for many years now [11], because of their low cost and energy requirements and are already installed in many service robots in use today. This implies that the method can be utilized by existing or future robots even if they are not specifically designed with fault detection and accommodation functionality or are not equipped with special hardware.

4.2 General Formulation

4.2.1 Robot Hardware

Mobile two-wheeled differential-drive robots, such as the iRobot Roomba that was used in this work, represent a significant portion of the already deployed household service robots in the world. Such robots are equipped with two main wheels with motors and one passive wheel for balance. Each of the main wheels carries an encoder sensor for odometry.

A central microcontroller is responsible for driving the motors and reading the sensors. Signals $\mathbf{u} = [u_1, u_2]^\top$ control the left and right motors' torque respectively. The signals **u** comply with the specifications provided by the robot manufacturer, with valid value range (u_{min}, u_{max}) and carry no units. These signals are internally converted by the motor controller to Pulse Width Modulation (PWM) signal. We chose to use PWM signal to keep the methodology generic and controller-independent. The encoders installed on the motors provide the revolution count of the motor, which given the diameter of the wheel, is directly transformed to the distance traveled by the wheel. Furthermore, assuming fast enough sampling, the difference in the encoder count provides an estimate of the speed of the motor. In practice, the embedded system loop is designed to meet this criterion, therefore the speed approximation error is insignificant and not included in our calculations directly. The speed of the wheels is measured in mm/s by

$$\mathbf{y}(k) = \mathbf{x}(k) + \mathbf{n}(k) \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^2$ is the speed of the left and right wheel respectively, $\mathbf{n} \in \mathbb{R}^2$ is the noise of the sensors and **y** is the measurement vector. We assume that encoder noise **n** is a random value and each element of the vector is bounded by $\overline{n}$, independent of the input or state of the robot

$$|n_i(k)| \leq \overline{n} \, \forall i \in \{1, 2\} \tag{4.2}$$

and the value of $\overline{n}$ is known from the sensor manufacturer specifications. Otherwise this value can be estimated experimentally by analyzing a collection of sensor data during normal operation. This can take place during the learning period described in Section 4.3.5. Even though sensor faults are possible and can be detected and isolated [85], it is important to note that in this thesis, we focus on actuator faults and therefore we assume that sensors are not susceptible to faults.

4.2.2 Robot Model

The sensors are located on the motor level measuring its speed, therefore the states are the individual speeds of the left and right motors:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \begin{bmatrix} \hat{\theta}_1 x_1(k) + \hat{\theta}_2 u_1(k) \\ \hat{\theta}_3 x_2(k) + \hat{\theta}_4 u_2(k) \end{bmatrix} + \boldsymbol{\eta}(\mathbf{x}(k), \mathbf{u}(k), \hat{\boldsymbol{\theta}}) + \boldsymbol{\phi}(k) \tag{4.3}$$

where $\boldsymbol{\eta}$ is uncertainty, $\hat{\theta} = [\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3, \hat{\theta}_4]^\top$, $\hat{\theta}_i \in \mathbb{R}$ are the estimated model parameters and ϕ is the fault. The value of the model parameters $\hat{\theta}$ may depend on various physical characteristics of the specific robot, e.g. mass, moment of inertia, wheel span and diameter and motor torque, however, in this work we assume that the functional dependence is not known. The four parameters are estimated during the learning period described in Section 4.3.5.

The robot is driven using the inputs u_1 and u_2, which generate motor torque spinning the wheel creating a force that accelerates the robot. As the robot is moving, rolling friction affects the acceleration, slowing down the robot. The linear model estimation requires relatively low processing which is preferable for a small embedded device and experimental data shows that it is accurate when input signals are smooth.

4.2.3 Uncertainties

There are two main sources of uncertainty

$$\eta_i(\mathbf{x}(k), \mathbf{u}(k), \hat{\boldsymbol{\theta}}) = \eta_i^\theta(\mathbf{x}(k), \mathbf{u}(k), \hat{\boldsymbol{\theta}}) + \eta_i^x(\mathbf{u}(k)) \tag{4.4}$$

The first, η^θ, is due to inaccuracies in the model parameters, $\hat{\theta}_1$, $\hat{\theta}_2$, $\hat{\theta}_3$ and $\hat{\theta}_4$. Factors like the temperature, battery charge and others, slightly change the performance of the motor and therefore affect the parameters. During a parameter estimation step, the values are estimated within a range of the true value $\theta_i^* = \hat{\theta}_i(1 + b_i^\theta(k))$, where $b_i^\theta(k)$ is the variation percentage from the true value. We assume that this variation is bounded by

$$|b_i^\theta(k)| \leq \overline{b}^\theta, \ \forall i \in \{1 \dots 4\} \tag{4.5}$$

When this variation is considered in the dynamics model (4.3), the uncertainty is calculated by

$$\begin{bmatrix} \eta_1^\theta(k) \\ \eta_2^\theta(k) \end{bmatrix} = \begin{bmatrix} b_1^\theta(k)\hat{\theta}_1 x_1(k) + b_2^\theta(k)\hat{\theta}_2 u_1(k) \\ b_3^\theta(k)\hat{\theta}_3 x_2(k) + b_4^\theta(k)\hat{\theta}_4 u_2(k) \end{bmatrix} \tag{4.6}$$

However exact value of $b^\theta(k)$ is unknown, therefore using the known value of $\overline{b}^\theta$ the upper bound of the uncertainty is defined as

$$\begin{bmatrix} \overline{\eta}_1^\theta(k) \\ \overline{\eta}_2^\theta(k) \end{bmatrix} = \begin{bmatrix} |\overline{b}^\theta \hat{\theta}_1 x_1(k)| + |\overline{b}^\theta \hat{\theta}_2 u_1(k)| \\ |\overline{b}^\theta \hat{\theta}_3 x_2(k)| + |\overline{b}^\theta \hat{\theta}_4 u_2(k)| \end{bmatrix} \tag{4.7}$$

The second uncertainty, η^x is due to the non-linear dynamics of the robot when there are large changes on the input signal. A service robot controller keeps the changes of the input signal small between iterations because this produces a smooth, continuous movement which is the desirable behaviour most of the time. However, when a sudden change in robot movement is required, the control signal may generate a spike of large magnitude in the estimation error. To accommodate this issue, which is caused due to the discretization of the continuous motor model, we use large-change-threshold u^δ for increasing the uncertainty upper bound. When there are large changes in the input $|u_i(k) - u_i(k-1)| > u^\delta$ of the i-th motor, then $\eta_i^x(k) = \overline{\eta}_i^x$ or else $\eta_i^x(k) = 0$. The value of u^δ is selected during the learning period, see Section 4.3.5.

4.2.4 Faults

It is possible that one or more faults may occur on the mobile robot unexpectedly. A fault may occur on the actuator attached to a wheel and its corresponding controller. This work's focus is on three types of actuator faults as depicted in Fig. 4.1, each one with different characteristics which depend on the cause of failure. The *Performance Degradation*(PD) fault is possible to occur due to a malfunction in the electronic driver of the motor, for example one of the multiple H-bridges providing power, or even the motor itself, for example debris caught in the axle. As a result the motor's performance is a fraction of the nominal one. It is also possible for the electronic driver to become unresponsive to any new input signal, a fault termed as *Unresponsive*(UR). When this happens the motor operates at a constant value, randomly defined at the fault occurrence time. Also a mechanical failure could completely immobilize a wheel, a fault termed as *Stuck-at-Zero*(SZ).

The fault is modeled as an extra term on the robot dynamics, $\phi_i = \beta(k - K_i^f)\alpha_i z_i$ where $\beta(k - K_i^f)$ is the time profile of the fault of the i-th wheel, $i \in \{1, 2\}$ occurring at time K_i^f, which in the case of abrupt faults, $\beta(k - K_i^f) = 1$ if $k \geq K_i^f$ and $\beta(k - K_i^f) = 0$ otherwise, α is the percentage of the performance loss and z_i is the structure of the fault which remains constant. The faults considered in this work affect the performance of the

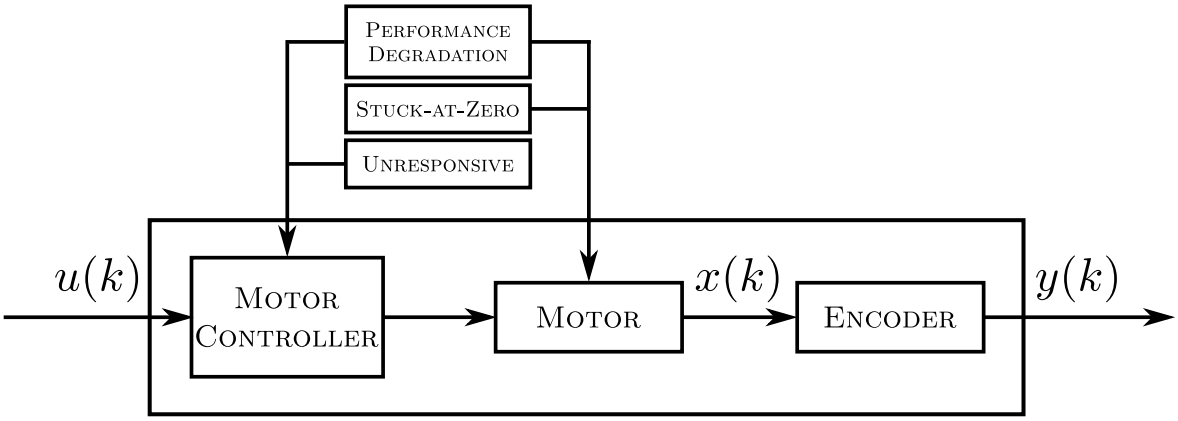


Figure 4.1: A diagram of the internal operation of the robot and sensor feedback. Depicted are the three faults considered in this work. *Performance Degradation* fault can occur directly on the motor itself for example due to debris caught on the axle, or on the electronic motor controller. *Stuck-at-Zero* fault occurs due to mechanical malfunction on the motor or the wheel, which forces the motor to complete stop. Finally the *Unresponsive* fault occurs when the motor controller becomes unresponsive to any new input

motor, for example a degradation fault will cause the motor performance to degrade depending on the magnitude of the fault. Since we don't have direct access on the electronics board that drives the motor or the drive shaft, we can emulate degradation by altering the input signal. In other words, the controller assumes that the input is the correct one, but in practice the input is altered to emulate the faulty behavior. For this reason, the input signal entering the motor under fault is

$$u_i^f(k) = u_i(k)(1 + \beta(k - K_i^f)\alpha_i(k)) \tag{4.8}$$

Under normal operation i.e. $k < K^f$, the input signal is the same as the controller signal. Therefore, based on (4.8) and (4.3) the faults are modelled as:

$$\phi_1(k) = \beta(k - K_1^f)\alpha_1(k)\hat{\theta}_2 u_1(k), \tag{4.9}$$

$$\phi_2(k) = \beta(k - K_2^f)\alpha_2(k)\hat{\theta}_4 u_2(k). \tag{4.10}$$

These equations are the same for all three fault types.

4.3 Methodology

4.3.1 Fault Detection and Isolation

In the following section the fault detection methodology proposed in [37] is used. A Luenberger observer is implemented to estimate the states $\hat{\mathbf{x}} \in \mathbb{R}^2$ of the robot based on the nominal dynamics and the input signal

$$\hat{\mathbf{x}}(k+1) = \hat{\mathbf{x}}(k) + \begin{bmatrix} \hat{\theta}_1 y_1(k) \\ \hat{\theta}_3 y_2(k) \end{bmatrix} + \begin{bmatrix} \hat{\theta}_2 u_1(k) \\ \hat{\theta}_4 u_2(k) \end{bmatrix} - \Lambda(\hat{\mathbf{x}}(k) - \mathbf{y}(\mathbf{k})) \tag{4.11}$$

assuming that $\hat{\mathbf{x}}(0) = \mathbf{y}(0)$. The estimation error filter coefficient matrix Λ is a diagonal matrix of size (2×2) where $0 < \Lambda_{(j,j)} < 1$ for the j-th state, such that it ensures stability of the estimation. Estimation error has an upper bound defined by:

$$\begin{aligned} \overline{\mathbf{e}}(k+1) = \overline{\mathbf{e}}(k)(1-\Lambda) + \begin{bmatrix} |\overline{b}^{\theta}\hat{\theta}_1 y_1(k)| \\ |\overline{b}^{\theta}\hat{\theta}_3 y_2(k)| \end{bmatrix} + \begin{bmatrix} |\overline{b}^{\theta}\hat{\theta}_2 u_1(k)| \\ |\overline{b}^{\theta}\hat{\theta}_4 u_2(k)| \end{bmatrix} \\ + \begin{bmatrix} |\overline{b}^{\theta}\hat{\theta}_1 n(k)| \\ |\overline{b}^{\theta}\hat{\theta}_3 n(k)| \end{bmatrix} + \begin{bmatrix} |\overline{n}(k)(1+\hat{\theta}_1)| \\ |\overline{n}(k)(1+\hat{\theta}_3)| \end{bmatrix} + \overline{\boldsymbol{\eta}}^{x}(k) \end{aligned} \tag{4.12}$$

Under normal operation (absence of faults), the estimation error should always be lower than or equal to the detection threshold i.e. $|e_i(k)| \leq \overline{e}_i(k),\ \forall i$. Therefore, based on this, a detection alarm is raised when this condition is violated, as follows

$$\text{Alarm1:} \begin{cases} \text{Fault} & \text{if } |e_i(k)| - \overline{e}_i(k) > 0 \\ \text{No Fault} & \text{if } |e_i(k)| - \overline{e}_i(k) \leq 0 \end{cases}, \forall i$$

Two observers are used, one for each wheel. Because of this, isolation of the fault location is straightforward. The fault is isolated on the wheel whose observer has raised the alarm. This also suggests that when both wheels fail, faults are detected on each wheel separately, in other words the faults do not mask each other.

4.3.2 Learning the Fault Magnitude

After detection and isolation, the robot can learn the magnitude of the fault. By learning we refer to the estimation of the fault function parameters by using an adaptive law [34]. The magnitude of the fault is also linked to the three types of fault considered in this work (PD, SZ and UR) and depending on the wheel, they all share the same fault structure

which is added to the observer becoming

$$\hat{\mathbf{x}}(k+1) = \hat{\mathbf{x}}(k) + \begin{bmatrix} \hat{\theta}_1 y_1(k) \\ \hat{\theta}_3 y_2(k) \end{bmatrix} + \begin{bmatrix} \hat{\theta}_2 u_1(k) \\ \hat{\theta}_4 u_2(k) \end{bmatrix} - \Lambda(\hat{\mathbf{x}}(k) - \mathbf{y}(k)) + \begin{bmatrix} \hat{\theta}_2 \beta(k - K^f)\hat{\alpha}_1(k) u_1(k) \\ \hat{\theta}_4 \beta(k - K^f)\hat{\alpha}_2(k) u_2(k) \end{bmatrix} \tag{4.13}$$

Let $\hat{\alpha}(k)$ be the estimated fault magnitude, which is computed through the adaptive law

$$\hat{\alpha}_i(k+1) = \hat{\alpha}_i(k) + \gamma z_i(k) \left(e_i(k+1) - (1-\Lambda)e_i(k)\right) \tag{4.14}$$

for which γ is the adaptive gain which corresponds to the rate of change in the gradient descent algorithm and $\mathbf{z}(k) = [\hat{\theta}_2 u_1(k), \hat{\theta}_4 u_2(k)]^\top$. For stability it is shown [1] that $0 < \gamma < 2$. Because the fault magnitude represents the percentage of the motor performance loss, it takes the value -1 when there is complete performance loss and the value of 0 when the motor operates normally. Therefore, to guarantee that $\hat{\alpha}(k)$ remains within the expected fault magnitude is bounded within $\hat{\alpha}(k) \in [-1, 0]$. In specific, the following rule is considered: if $\hat{\alpha}(k) > 0$ or $\hat{\alpha}(k) \leq -1$, then $\hat{\alpha}(k) = \hat{\alpha}(k-1)$.

4.3.3 Small Faults Detection

Faults with smaller magnitude generate smaller estimation error and as a result, it is possible that the error remains below the threshold. Therefore, faults with smaller magnitude may remain undetected. This happens because in terms of estimation error magnitude a small fault is indistinguishable from uncertainty. It is however, possible to detect such faults by exploiting the temporal profile of the estimation error.

We propose a modified alarm condition, defined as:

$$\text{PreAlarm: } \begin{cases} \text{Yes} & \text{if } |e_i(k)| - \overline{e}_i(k) - h > 0 \\ \text{No} & \text{if } |e_i(k)| - \overline{e}_i(k) - h \leq 0 \end{cases}, \forall i$$

where $h \in \mathbb{R}^+$ is a predetermined bias. This modification essentially lowers the threshold, making the detection more sensitive. It is however, inevitably low enough for the *PreAlarm* to be triggered by the noise. *PreAlarm* is an internal alarm and does not directly imply that a fault has occurred. Instead it initiates an algorithm which counts the duration t^{det} that *PreAlarm* is active i.e. At the i-th occurrence that *PreAlarm* changed state from No to Yes at time k_i^a and from Yes to No at time k_i^d, duration is given by $t_i^{det} = k_i^d - k_i^a$. The t^{det} value depends on the sensor noise and h and has an upper bound $\overline{t}^{det}$ which can be determined experimentally. The proposed method for determining $\overline{t}^{det}$ is to first collect experimental data of the robot moving under fault-free state. Then construct a set

of all the t^{det} values recorded for the q times the *PreAlarm* triggered and determine the highest value

$$\overline{t}^{det} = \max\{t_1^{det}, \ldots, t_q^{det}\} \tag{4.15}$$

During the operation of the robot, the duration of the *PreAlarm* is determined by $\tau(k) = k - k^a$. Based on this we define the *Alarm2* as

$$\text{Alarm2:} \begin{cases} \text{Fault} & \text{if } \tau(k) - \overline{t}^{det} > 0 \\ \text{No Fault} & \text{if } \tau(k) - \overline{t}^{det} \leq 0 \end{cases}$$

The trade-off of this modification however, is the added delay to the fault detection. The values of h and $\overline{t}^{det}$ are calculated in the learning period.

4.3.4 Fault Type Identification

Tracking the fault magnitude can provide insight about the fault type. Essentially, this improves the isolation accuracy since the robot is aware not only about which wheel is faulty but also about the type a fault. Considering that effects of some fault types may be treated, knowledge of the fault type may allow a faulty robot to continue its operation without affecting its objectives.

The faults considered in this work, are profiled based on three characteristics which are based on the change of fault magnitude (c_1), the change of the input signal (c_2) and the value of the fault magnitude (c_3):

$$c_1(k) = \frac{1}{w} | \sum_{j=k-w}^{k-1} \hat{\alpha}(j+1) - \hat{\alpha}(j)|, \tag{4.16}$$

$$c_2(k) = \frac{1}{w} | \sum_{j=k-w}^{k-1} u(j+1) - u(j)|, \tag{4.17}$$

$$c_3(k) = 1 - |\hat{\alpha}(k)|. \tag{4.18}$$

The design parameter $w > 0$ defines a temporal window of the moving average filter. In context, the window size is a way to adjust the response of the filter to the changes of the signals. In our experiments we used $w = 15$ which corresponds to 1.5 seconds. Fault type profiling based on the c_1, c_2 and c_3 characteristics, provides the decision tree depicted in Fig. 4.2.

Parameters $p_i \; \forall i \in \{1, 2, 3\}$ in Fig. 4.2 set a small margin around the value of the decision. This is required as due to noise, c_1 and c_3 have some fluctuation and for c_2 to discard small changes in the input. Their values are calculated based on the decision-tree

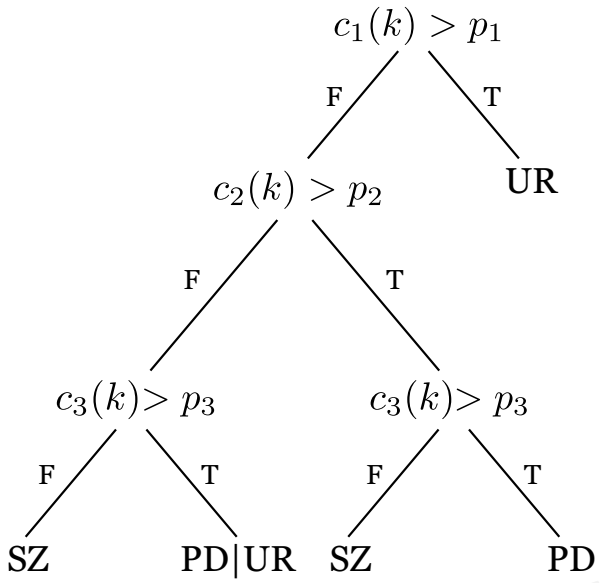


Figure 4.2: Decision tree for identifying the fault type based on the change of fault magnitude (c_1), the change of the input signal (c_2) and the value of the fault magnitude (c_3). The three fault-types *Performance Degradation*, *Unresponsive* and *Stuck-at-Zero* are denoted as *PD*, *UR* and *SZ* respectively. Symbol "|" represents a logical OR operator

analysis, as the values which best classify the fault types. It is possible that the tree reaches the inconclusive decision "PD|UR". In such case, if the condition $c_1 > p_1$ changes from *False* to *True* i.e., the input signal changes, the decision becomes conclusive.

4.3.5 Learning Period

During the learning period, a short interval at the beginning of the robot's deployment, the robot is allowed to estimate different parameters that are used in our methodology. This enhances the generalization of the proposed method and allows readjustment of parameters of the same robot during its lifetime. During this period, the system is monitored to ensure fault-free operation. Even though it may be possible to compute these parameters and program them into the robot prior its deployment, it is not preferable as the target environment could also affect these parameters (temperature, floor material etc) increasing the modeling error. The proposed approach is to identify these parameters after the robot is deployed. A short parameter estimation period allows the robot to learn these parameters at the target environment. The input signals are of sinusoidal form with different frequency for each wheel $u(k) = u_{max}[\sin(\omega_1 k), \sin(\omega_2 k)]^\top, \omega_1 \neq \omega_2$ to ensure persistency of excitation which is required for learning [34]. During the learning period the parameters converge to their final value.

The learning period is also used to calculate the noise upper bound $\overline{n}$. This is done by driving the motor at different known speeds and the fluctuation of the residual between the sensor measured speed and the expected value is regarded as noise. The upper bound is defined as the maximum absolute value of noise.

The value of large-change-threshold u^{δ} which is used to determine uncertainty η^{x} is also calculated during the learning period. The robot enters an input sequence where the input signals are changed abruptly. The robot tries different values and monitors the estimation error. It determines the value of u^{δ} above which the estimation error is large enough to trigger a false alarm.

Finally, two parameters h and $\overline{t}^{det}$ are required for detecting smaller faults. During this phase the robot randomly moves around the environment monitoring the estimation error. Next, it lowers the detection threshold by h and monitors whether the estimation error crosses the detection threshold which leads to *PreAlarm* and records both h and the *PreAlarm* duration t^{det}. At the end of this step it determines $\overline{t}^{det}$ as described in Section 4.3.3. The detection threshold is lowered again and the procedure repeats. In the next step the robot injects different small faults for a short period and determines whether the fault is detected by triggering *Alarm2*. At the end of this step it determines and records the smallest fault magnitude that it was able to detect. The results of this phase consists of a list of h, $\overline{t}^{det}$ and the smallest fault magnitude detectable. The parameters h and $\overline{t}^{det}$ are purely design parameters and the appropriate choice is depended on the application.

4.4 Experimental Results

This section presents the experimental results performed on an iRobot Roomba. The results represent the data recorded from the physical robot.

4.4.1 Parameter Estimation

The robot is loaded with a predetermined input instructions, generated based on two sinusoidal signals of different frequency to achieve persistency of excitation. The robot is left to run for 5 minutes to collect enough data. When the data-collection is completed, the parameter estimation algorithm provides the estimation of the model parameters, $\hat{\theta}_1 = -0.4777$, $\hat{\theta}_2 = 0.09586$, $\hat{\theta}_3 = -0.4352$ and $\hat{\theta}_4 = 0.08546$. It is worth noting that the difference between $\{\hat{\theta}_1, \hat{\theta}_3\}$ and $\{\hat{\theta}_2, \hat{\theta}_4\}$ is due to differences between the motors. The

parameter estimation is evaluated by comparing the open-loop speed estimation

$$\begin{bmatrix} x_1^{ol}(k+1) \\ x_2^{ol}(k+1) \end{bmatrix} = \begin{bmatrix} x_1^{ol}(k) \\ x_2^{ol}(k) \end{bmatrix} + \begin{bmatrix} \hat{\theta}_1 x_1^{ol}(k) + \hat{\theta}_2 u_1(k) \\ \hat{\theta}_3 x_2^{ol}(k) + \hat{\theta}_4 u_2(k) \end{bmatrix} \tag{4.19}$$

against the measured speed $y(k)$. Fig. 4.3(a) shows that the estimation tracks the mea-

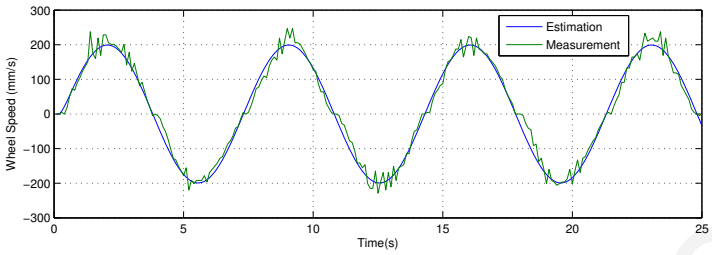


Figure 4.3: Open-loop response of the wheel model with a sinusoidal signal as input.

sured speed closely. The root mean square error is 2.0053 mm/s. Some inaccuracies occur due to the non-linearity of the motor response, particularly in the near zero region.

4.4.2 Fault-free case

This experiment demonstrates the operation of the fault detection observer when there are no faults present in the system. The robot is instructed to follow a square path while the data is collected. The observer, uses the parameters which were previously learned and an arbitrary chosen value for Λ. Figure 4.4(a) shows the observer estimation. For reference, the input signal is depicted in Fig. 4.4(b). During periods that the input signal value is constant (for example the period between 1 and just before 5 seconds), we observe the effects that the sensor noise and parameter estimation error have on the estimation error. The detection threshold, however, is not crossed at any point.

The figure also depicts the estimation error caused by large changes in the input experienced when the robot starts and ends turning, previously defined as η^x. Because the observer monitors the input signal to the motors, it is able to anticipate the error increase, indicated by the sudden raise of the threshold.

4.4.3 Fault Detection

The following results show the overall ability of the observer to detect a fault occurrence while the robot is programmed to follow a predetermined path. In this example, the robot wheel was physically obstructed causing it to lose performance, essentially causing

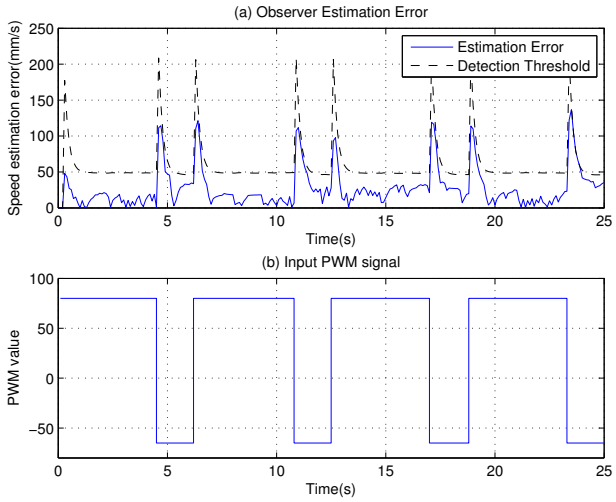


Figure 4.4: (a) Estimation error of the observer when the robot is instructed to follow a square shape. Estimation error remains below the detection threshold throughout the experiment. Also illustrated, is the adaptive level of the threshold. Because of the modeling uncertainty during hard changes of the input, the threshold raises rapidly. (b) is the input signal

a PD fault. This fault was repeated 6 times during the experiment. Intentionally, the obstruction was different each time, causing different fault magnitude and location. Figure 4.5(a)(b) depicts the estimation error of the observer for the left and right wheels respectively. Figure 4.5(c) shows the path followed by the robot as it was recored by its own sensors. The path information is not used for any calculations and it is shown here just for reference. The robot initial position is $(1, 1)$. The arrows indicate the approximate robot positions where the faults occurred. The number over the arrow identifies the fault so it can be traced back on the estimation error graphs. Faults 1,2,3,5 affect either the left or right wheel and their effect on the path is visible on the map, depending on their magnitude. In fault 4, both wheels were obstructed at the same time affecting robot's forward speed illustrated by the change of density of the markers on the plot. Finally, in fault 6 the right wheel was obstructed while the robot was performing a left turn. In the period when the robot operates normally, the estimation error is well below the detection threshold. When a fault occurs however, the estimation error crosses the threshold and the observer raises an alarm. At this point the robot is fully aware that a fault has occurred as well as which is the faulty wheel, being the left or right or both. The estimation error remains above the threshold for the whole duration of the fault. As shown in Fig. 4.5 all faults were correctly detected on both wheels.

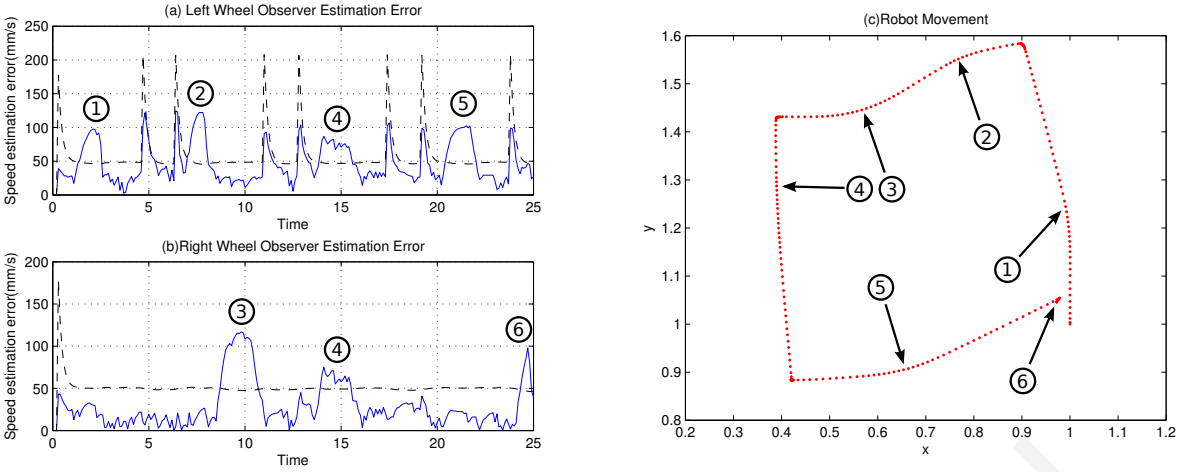


Figure 4.5: The figure presents the data collected when the robot was programmed to move in a square like path while its wheels were intentionally obstructed at various points, essentially generating faults. (a) and (b) show the estimation error of the left and right wheel respectively while (c) illustrates the actual path as it was recorded by the robot's sensors. The numbers represent the fault number and the arrows indicate the position where the faults occurred

4.4.4 Fault Injection

To evaluate the detection accuracy, delay and sensitivity we use software emulated faults, which provide accurate information about fault magnitude and time of occurrence. A software-emulated fault is a fault which affects the robot inputs, in order to cause the real robot to behave as if it was experiencing a real fault event. This approach allows us to have full control over magnitude and time profile of the faults, in order to accurately evaluate the performance of our algorithm with respect to different fault characteristics. Faults are programmed a priory to occur at specific time instants and can have specific duration i.e. the robot returns to normal operation after the fault ends. When a fault occurs, it modifies the input signal of the controller which is used to drive the physical robot. With respect to (4.8), u is the controller signal and u^f is the injected signal which is what drives the physical robot. None of the fault emulation information is known by the detection algorithm.

In experimental results presented next, the robot experiences six consecutive fault injections of type PD. The faults have different magnitudes, starting small and increase with time. All faults have duration of 2 seconds before being removed. The time between two consecutive faults is 1 second.

Figure 4.6(a) is the estimation error of the observer and the detection threshold. After

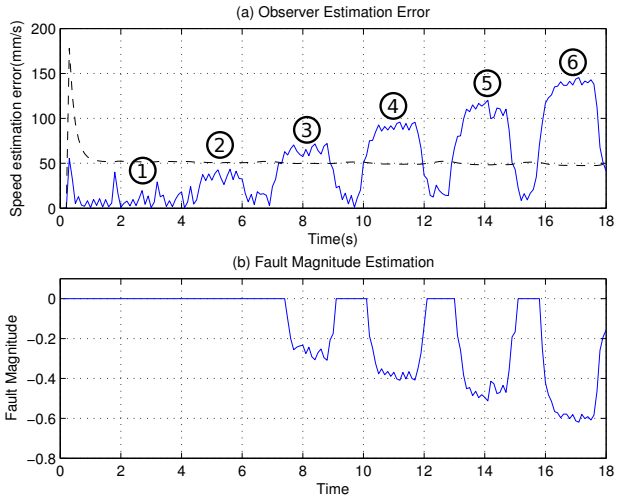


Figure 4.6: Figure presents data collected when the robot was subjected to consecutive faults of increasing magnitude. (a) shows the estimation error of the left wheel observer and the detection threshold. (b) is the fault magnitude as it is estimated by the algorithm on-line. Small magnitude faults may not be detected, as their estimation error is comparable to the estimation error caused by noise and uncertainties.

a fault is detected, the algorithm of learning the fault magnitude is also initiated, demonstrated in Fig. 4.6(b). Table 4.1 displays information about the injected fault magnitude, detection delay and the estimated fault magnitude. The term *ND* indicates that the fault was not detected. The value listed in the magnitude estimation column is the average of the magnitude estimation values collected during the corresponding fault. Results show that faults 1 and 2 were not detected by the observer since the estimation error remained below the detection threshold. Because of their low fault magnitude, the estimation error is comparable to the error caused by uncertainty and therefore it cannot be distinguished by the observer. Fault 3 has higher magnitude and the estimation error is large enough to cross the threshold and be detected within 0.6 seconds. Fault magnitude estimation error is 3.7%. The performance for fault 4 and 5 is similar, with detection delay 0.4 seconds and fault magnitudes estimation error of 3.9% and 0.5% respectively. Fault 6 is detected in 0.3 seconds with increased fault magnitude estimation error of 4.2% compared to the previous faults. This increase can be attributed to the fact that the $u^f(k)$ becomes low, closing in the non-linear behavior of the motor.

Table 4.1: Detection and identification results of 6 faults with increasing magnitude. The detection delay column lists the time between the fault occurring and being detected by the system. The last column lists the approximated fault magnitude. *ND* indicates that the fault was not detected

Fault #	a	Det. Delay	$\hat{a}$
1	-5%	ND	-
2	-15%	ND	-
3	-25%	0.6 s	-28.68%
4	-35%	0.4 s	-38.91%
5	-45%	0.4 s	-45.54%
6	-55%	0.3 s	-59.20%

4.4.5 Small Faults

To demonstrate the improved sensitivity of *Alarm2*, the same experiment presented in Fig. 4.6 and Table 4.1 is used with the modified alarm algorithm with $h = 3$ and $\bar{t}^{det} = 0.3$ s. Figure 4.7(a) depicts the detection threshold at a lower level. In the results, there are two instances at $t = 1.8$ s and $t = 3.2$ s, where the estimation error crosses the threshold but the alarm is not raised because it does not meet the *Alarm2* condition. This exhibits the tolerance of this detection method against the noisy estimation error. As with the *Alarm1* condition, the first fault is not detected as its effects are completely masked by the noise as it can be seen in the figure. Estimation error caused by Fault 2, which previously remained undetected, is enough to cross the threshold and raises the detection alarm. As results in Table 4.2 show, Fault 2 took 1.8 s to be detected after it occurred. Detection of Fault 3 is also successful but with an additional delay inherited by the *Alarm2* condition. Faults 4, 5 and 6 took an additional 0.3 s to be detected. The detection delay does not however affect the approximation of the fault magnitude. This modified version of the algorithm is capable of detecting smaller faults, becoming more sensitive at the expense of detection delay. The delay is caused due to the alarm counter which is necessary to prevent false alarms. The added delay is expected to be close to $\bar{t}^{det}$, also confirmed by the experimental results. The desired trade-off between detection sensitivity and delay is application-dependent. In a case-study example, a supervisor of a robotic warehouse based on the proposed methodology, is able to choose the aforementioned trade-off point. Another possibility is to lower $\bar{t}^{det}$, essentially reducing the delay. Such system however,

is susceptible to false alarms. This gives the supervisor the option to balance between a fast and sensitive detection and handling of false alarms.

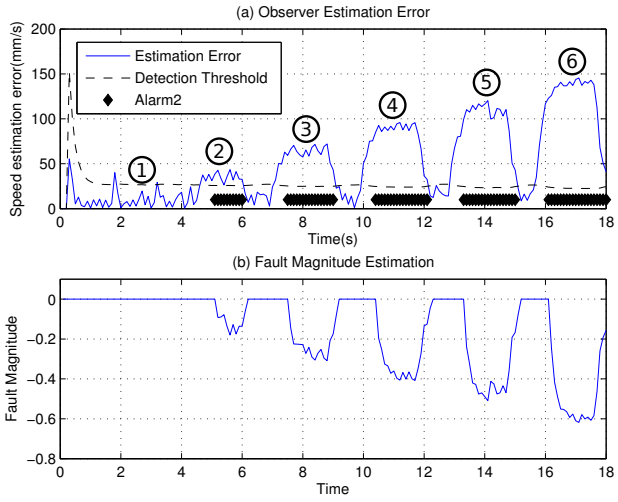


Figure 4.7: Improved fault detection sensitivity using the *Alarm2*. (a) shows that detection threshold is low enough to be triggered by low magnitude fault as well as noise. However, due to the *PreAlarm*, false alarms are prevented. Fault detection is indicated by the diamond marker. (b) is the fault magnitude as it is estimated by the algorithm on-line

Table 4.2: Results of detection and identification using the *Alarm2*. Fault 2 is now detected successfully even though the estimation error it produces is low and comparable with that produced by uncertainty

Fault #	a	Det. Delay	$\hat{a}$
1	-5%	ND	-
2	-15%	1.8 s	-15.10%
3	-25%	0.7 s	-28.66%
4	-35%	0.7 s	-38.84%
5	-45%	0.7 s	-45.49%
6	-55%	0.6 s	-59.26%

Next we present a technique that uses in parallel the two thresholds described in sections 4.3.1 and 4.3.3. The fault is detected with *FuseAlarm* defined as

$$\text{FuseAlarm:} \begin{cases} \text{Fault} & \text{if } \mathit{Alarm1} = 1 \text{ OR } \mathit{Alarm2} = 1 \\ \text{No Fault} & \text{if } \mathit{Alarm1} = 0 \text{ AND } \mathit{Alarm2} = 0 \end{cases}$$

The extended *FuseAlarm* provides fast fault detection and is also able to detect smaller

faults. Again, we use the same experiment for comparison and the results are shown in Fig. 4.8.

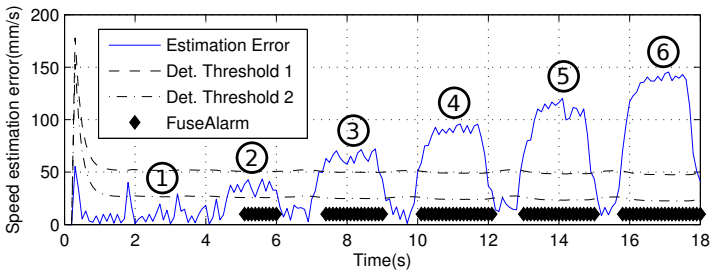


Figure 4.8: A fusion of the previous two techniques provides *FuseAlarm* with fast fault detection as well as sensitivity to small faults. Fault 2 which is a small fault is detected with some delay while the rest of the faults are detected without any delay

4.4.6 Fault Identification

Fault injection is used to emulate the three different types of faults. By tracking the fault magnitude and the three characteristics c_1, c_2 and c_3 the fault is identified by the algorithm based on the decision tree presented in Section 4.3.4. The fault is injected at $t = 2$ s. Figure 4.9 depicts the results of the experiments. After a fault is detected, a short period of 4 seconds is required in order for the filters to stabilize before the decision algorithm starts at $t = 6$ s, indicated on the figure by the vertical dotted line. In this experiment we use $p_1 = p_2 = p_3 = 0.1$, derived from the decision tree analysis. Note that c_1, c_2 and c_3 are normalized so that they fall within the region $[0, 1]$. For the PD fault, Fig. 4.9(a) shows that immediately after the stabilization period ($t = 6$ s) the three decision conditions are False-False-True (F-F-T) giving the inconclusive decision of "PD|UR". Shortly after, at time $t = 7.8$ s, the controller changes the input signal to the wheel, in order to follow the predetermined path, indicated by the change of the c_2 value. In a PD type fault, the fault magnitude is unaffected by the input signal changes and c_1 demonstrates the correct behavior by staying below p_2. This changes the decision conditions to F-T-T which correctly identifies the fault as PD. In Fig. 4.9(b) the UR fault initially (at $t = 6$ s) exhibits the decision conditions F-F-T as with the PD fault. However, when the input changes, the c_1 condition also changes to T and therefore the fault is correctly identified as UR. The SZ fault shown in Fig. 4.9(c), is immediately identified correctly, by giving the decision conditions F-F-F.

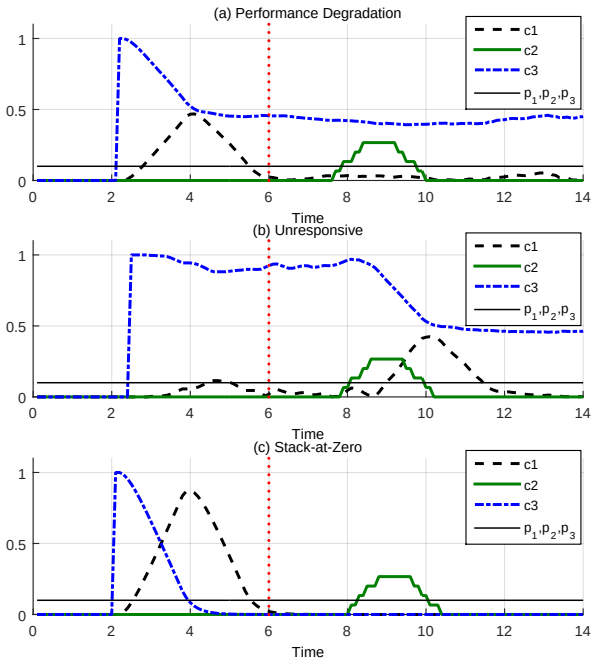


Figure 4.9: Results of the fault identification. The plots show the normalized values of the three fault characteristics c_1, c_2 and c_3 and their corresponding threshold parameters p_1, p_2 and p_3. (a) The PD fault is initially identified as "PD|UR" at $t = 6$ s and shortly after at $t = 8$ s is identified as PD. (b) This fault is initially identified as "PD|UR" at $t = 6$ s and the successfully identified as UR at $t = 9$ s. (c) The SZ is identified correctly immediately after the stabilization period

4.5 Concluding Remarks

The work presented in this chapter is focused on achieving fault detection and identification, with particular emphasis on reducing false alarms, using a model-based approach. Detection and identification of faults is a topic of high importance because it leads to improved robot safety, a requirement for wide acceptance of service robots in domestic environments. The method is evaluated on a real robot with noisy sensors. In an effort to make this method applicable to other robots, a learning period is proposed during which the parameters of the fault detection and identification algorithm are calculated. Furthermore, the method does not require any additional sensors except from the standard odometry sensors found on most service robots.

Experimental results show that the linear model produces accurate and fast estimation of the state with an exception of some special cases. The robot is able to detect faults occurring on either wheel or both wheels simultaneously. Above a certain magnitude the faults are detected without any false alarms. Faults were injected in the system for

evaluating the performance against various fault magnitudes, and they were successfully detected and their magnitudes accurately estimated. We also demonstrated how faults can be classified between three fault-types.

It is possible that faults of smaller magnitude are not detected due to the estimation error magnitude generated by those faults being indistinguishable from that of uncertainty. Experimental results of the proposed extension for addressing this issue demonstrate increased detection sensitivity, with the trade-off of a small delay in the detection time. A new alarm condition fuses the two proposed thresholds to eliminate the detection delay whenever is not necessary, therefore providing fast detection as well as sensitivity for small faults.

The work presented in this chapter has been peer-reviewed and published as a journal paper [96].

Chapter 5

Fault Detection for Mobile Robots Using Multiple Thresholds

5.1 Introduction

In a model-based approach, the residual between an observer using the nominal dynamics, and sensor readings, is compared against an adaptive threshold to determine whether a fault has occurred. An important part of this analytical redundancy approach, are the uncertainties present in the system and their type. Through learning and hardware specifications, the upper bound of each uncertainty type is identified, and their state dependent values are calculated at each iteration, which are used to adapt the detection threshold. Consequently, the detection sensitivity (the ratio of True Positives over the sum of True Positives and False Negatives) of the method is limited by the uncertainty upper bound. In this chapter, we present a new detection scheme for increasing detection sensitivity by incorporating multiple thresholds for evaluating the estimation error, yielding improved results when compared to a conservative worst-case threshold method. It differentiates from other residual-based methods because it combines the estimation error magnitude with the temporal behavior of a warning signal in order to detect faults. The method gains a better insight on the estimation error using multiple thresholds derived from the distribution of the uncertainty. Comparing the estimation error using these thresholds generates the warning signal, based on which faults are detected. We show that fault detection using the proposed warning signal yields improved performance in detection sensitivity in comparison to a conservative worst-case threshold. It is important to note that we do not explicitly use analytical models about the structure of uncertainty, as these

are not typically precisely known in practice. In contrast, this method is data-driven, relying on training data to derive the distribution of uncertainty with respect to the robot states.

5.2 Background on Worst-Case Threshold Fault Detection

This section provides the necessary background on model-based fault detection in discrete times, as presented in [34, 37]. The overall idea is to compare the absolute state-estimation error with an adaptive threshold computed, given an *a-priori* known upper-bound of the uncertainty parameters.

5.2.1 Robot Model and Uncertainties

The robot considered in this work is differential-drive, a very common configuration especially in service robots. Two motorized main wheels are used for motion, while a castor wheel is used for balance. Each of the two main wheels, is equipped with an encoder sensor for odometry. An on-board controller runs a navigation algorithm that allows the robot to move around a flat surface in order to complete its task, by driving the motors and reading values of the sensors. The signal $u(k)$ controls the motor speed, complying with the specifications provided by the robot manufacturer with valid range $u(k) \in \mathcal{U} = [u_{min}, ..., u_{max}]$. The resulting wheel speed is estimated by the model:

$$x(k+1) = x(k) + \theta_1^* x(k) + \theta_2^* u(k) + \eta^x(x(k), u(k)) + \phi(k) \tag{5.1}$$

where $x \in \mathbb{R}$ is the speed of the wheel measured in mm/s, η^x is the modeling uncertainty, ϕ is the fault and $\theta_1^*, \theta_2^* \in \mathbb{R}$ are the model parameters which depend on various physical characteristics of the specific robot. The robot is equipped with two wheels therefore each wheel should be described by its own model (5.1) where the parameters θ match the behavior of each wheel. However, for notational clarity we describe the method for one wheel, even though in practice the same method applies for each wheel of the robot.

Encoders located on the wheels provide the number of revolutions, which is converted to traveled distance given the diameter of the wheel. Because the encoders are sampled at high frequency, the difference between two consecutive readings, estimates the speed

of the wheel which measured in mm/s by:

$$y(k) = x(k) + n(k) \tag{5.2}$$

where $n \in \mathbb{R}$ is the noise of the encoder and $y(k) \in \mathcal{Y} = [y_{min}, ..., y_{max}]$ is the sensor measurement.

A Luenberger observer is implemented to estimate state $x(k)$ based on the nominal dynamics and the input signal:

$$\hat{x}(k+1) = \hat{x}(k) + \hat{\theta}_1 y(k) + \hat{\theta}_2 u(k) - \Lambda(\hat{x}(k) - y(k)) \tag{5.3}$$

assuming $\hat{x}(0) = y(0)$. The value of the estimation error filter coefficient $0 < \Lambda < 1$ is chosen such that it ensures stability of the estimation. Model parameters θ^* are estimated by $\hat{\theta}_1, \hat{\theta}_2 \in \mathbb{R}$ using parameter estimation on training data.

We assume that the i-th parameter $\hat{\theta}_i$ estimates the unknown parameter θ_i^* within a percentage b_i^θ, such that $\theta_i^* = \hat{\theta}_i(1 + b_i^\theta)$. We further assume that b_i^θ is bounded, such that

$$|b_i^\theta| \leq \overline{b}^\theta, \; i \in \{1, 2\} \tag{5.4}$$

Below we provide intuition on some of the sources of uncertainty in robots considered in this work: odometry encoder noise causes uncertainty in the system through the measurement noise. Uncertainty is also introduced from inaccuracies in the model parameters represented by the terms b_1^θ and b_2^θ. Imperfect estimation, as well as other factors like battery charge variation, change the motor performance and ultimately the model parameters. Another source of uncertainty is is due to the non-linear regions in the motor performance. Non-linear behavior is exhibited when large changes in the input signal are observed, usually when the robot is required to do sharp immediate turns. Depending on the robot controller algorithm, such behavior is not frequent because a smooth motion path is typically preferred. Nevertheless, such state is possible and therefore is considered in our method.

5.2.2 Faults

At any given discrete time k, the robot may experience a failure affecting its motorized wheels. The type of faults considered in this work are *persistent performance degradation* i.e. the fault degrades the performance of the motor with respect to the input signal; once the fault has occurred it persists. The fault term is modeled as $\phi(k) = \beta(k - K^f)\alpha(k)z(k)$,

where $\beta(k - K^f)$ is the time profile of the fault occurring at time K^f, $\beta(k - K^f) = 1$ if $k \geq K^f$ and $\beta(k - K^f) = 0$ otherwise, $\alpha(k)$ is the performance loss (fault magnitude) expressed in percentage and $z(k)$ is the structure of the fault. Such faults can occur due to malfunction of the motor's electronic driver or even the motor itself, for example debris caught in the wheel axle.

5.2.3 Estimation Error

The estimation error $e(k)$, i.e. the difference of the measured state and the estimated state is:

$$e(k) = y(k) - \hat{x}(k), \tag{5.5}$$

and after substituting the values in the fault-free case ($\phi(k) = 0$), we compute the estimation error dynamics as:

$$e(k+1) = e(k)(1 - \Lambda) + \eta(y(k), u(k)) \tag{5.6}$$

$$\begin{aligned}\eta(y(k), u(k)) =& \hat{\theta}_1 b_1^{\theta} y(k) - \hat{\theta}_1 b_1^{\theta} n(k) \\ &+ \hat{\theta}_2 b_2^{\theta} u(k) - n(k)(1 + \hat{\theta}_1) \\ &+ n(k+1) + \eta^x(y(k) - n(k), u(k)).\end{aligned} \tag{5.7}$$

The function $\eta(y(k), u(k)) \in \mathcal{H}$ is comprised of all the unknown system uncertainties, including sensor noise, and it cannot be determined analytically. In the case of model-based fault detection, the parameter upper bounds could be used if they were known, to determine the total uncertainty upper bound; this was discussed in Chapter 4. In this work, we do not consider known uncertainty parameter bounds; instead, we assume that the bounds of $\mathcal{H}$ are acquired through training. The upper bound of the uncertainty is computed based on the training data, such that $|\eta(y(k), u(k))| \leq \overline{\eta}(y(k), u(k))$.

5.2.4 Worst-Case Threshold Detection

The Worst-Case method is a single threshold (ST) detection, that relies on the uncertainty upper bound in order to compute the maximum estimation error that can be observed in the fault-free system i.e. when $\phi(k) = 0$. Using uncertainty upper upper bound in (5.6), the estimation error upper bound is:

$$\overline{e}(k+1) = (1 - \Lambda)\overline{e}(k) + \overline{\eta}(y(k), u(k)) \tag{5.8}$$

During normal operation (i.e. in the absence of faults), the absolute value of the estimation error is always smaller or equal to the estimation error upper bound (5.8). The estimation error upper bound is the detection threshold used in the Worst-Case method, which leads to the definition of the following fault detection condition:

$$d(k) = \begin{cases} 1 & \text{if } |e(k)| - \overline{e}(k) > 0 \text{ — FaultST} \\ 0 & \text{if } |e(k)| - \overline{e}(k) \leq 0 \text{ — No-Fault} \end{cases} \tag{5.9}$$

where $d(k)$ is the fault detection signal which can be used to alert an operator or to activate other fault-diagnosis modules. A fault is detected when the fault detection signal switches from 0 to 1 but is possible to return back to 0, depending on the robot input. Despite this, the fault detection alarm remains raised if a fault is detected at any point in time, i.e. $D(k) = 0,\ \forall k < k_0$ and $D(k) = 1,\ \forall k \geq k_0$, where $D(k)$ is the fault detection alarm and k_0 is the time instant when d(k)=1 for the first time.

5.2.5 Challenges Arising

Because the detection threshold is based on the upper bound of the uncertainty, the precision (the ratio of True Positives over the sum of True Positives and False Positives) of the Worst-Case method is theoretically perfect, since no-false-alarms are possible. However, in cases where a fault occurs but the estimation error is not large enough to exceed the detection threshold, it will be classified as *No-Fault* which is a False Negative. This issue becomes significant when the uncertainty distribution includes rare events of high uncertainty. Thus, determining the detection threshold solely based on the uncertainty upper bound is a conservative approach, which leads to the reduction of sensitivity in favor of the no-false-alarm feature of the Worst-Case method.

5.3 Multi-Threshold Detection

The Worst-Case method produces a binary *fault indication* signal; when below the threshold, a high estimation error is as likely as a low estimation error and does not indicate a higher chance abnormal operation. To address the challenge of the limited sensitivity of the conservative Worst-Case method, one approach is to increase the *fault indication* resolution. In this section, we introduce the Multi-Threshold (MT) method; we propose the use of additional thresholds (μ-thresholds) determined from multiple levels of the uncertainty distribution, which are used to generate a warning signal.

5.3.1 Methodology

The multiple levels are computed with respect to the distribution of uncertainty in each state. In a similar way that the Worst-Case method's detection threshold corresponds to the upper bound value of uncertainty $\overline{\eta}(x(k), u(k))$ (5.8), the MT method's μ-thresholds correspond to specific levels of the uncertainty distribution. In specific, the distribution of uncertainty is divided into N levels, and then each level is used to generate a threshold $\mu_j,\ j \in \{1, \ldots, N\}$. At each time step, the estimation error magnitude is compared against μ_j to generate a *warning signal* $w(k) \in \{0, ..., N-1\}$. A fault is detected when the warning signal deviates from its distribution observed during fault-free operation; we implement this by applying a change-detection method.

The proposed methodology has the following steps:

- Collect training data under normal operation
- Construct uncertainty bins based on the training data
- Select number of levels and determine the corresponding uncertainty of each bin
- Compute the multiple μ-thresholds
- Generate warning signal
- Compute fault-detection signal

The overview of the method is depicted in Fig. 5.1.

5.3.2 Levels From Training Data

The method depends on the collection and processing of a finite number of training data collected while the robot is operating normally for the time period K. The values of $y(k)$ and $u(k)$ are readily available in the controller for $k \in K$ while the value of uncertainty needs to be calculated using:

$$\eta(k) = e(k+1) - (1-\Lambda)e(k) \tag{5.10}$$

and estimation error is calculated with (5.6). The training data sample set Ψ is represented as:

$$\Psi = \{(\psi_1, \psi_2, \psi_3) |\ \psi_1 \in \mathcal{Y}, \psi_2 \in \mathcal{U}, \psi_3 \in \mathcal{H}, k \in K\}. \tag{5.11}$$

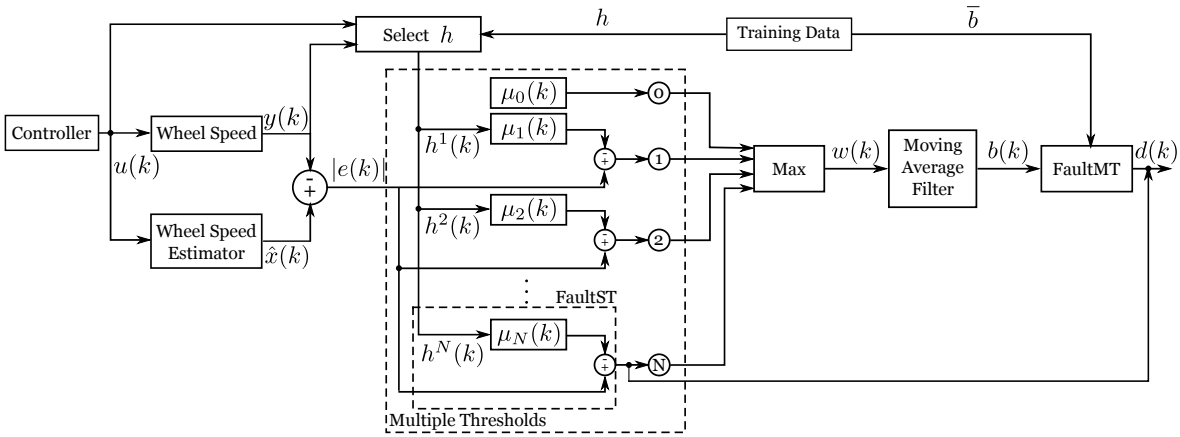


Figure 5.1: Diagram of the multi-threshold fault detection method. Training data is used to select the appropriate h values which are used to determine the μ-thresholds. Each threshold is compared against the absolute estimation error and their output ultimately defines the warning signal $w(k)$. FaultMT condition detects faults by comparing the moving average value against a threshold determined during training. Additionally, faults are detected with the FaultST condition, using the last threshold $\mu^N(k)$.

A two-dimensional plane of $\mathcal{Y}$ and $\mathcal{U}$ is defined based on the properties of the robot i.e. $S \in \{y_{\min} \dots y_{\max}\} \times \{u_{\min} \dots u_{\max}\}$. This plane is divided into m regions, such that $S = \{S_1, \dots, S_m\}$. Each region i encloses a set of uncertainty values which is defined as bin b_i^S. In other words, samples which are close together with respect to y and u are grouped together into a bin:

$$b_i^S = \{\psi_3 \mid (\psi_1, \psi_2) \in S_i,\ (\psi_1, \psi_2, \psi_3) \in \Psi\} \tag{5.12}$$

In general, the shape of bin regions S_i can be arbitrary; in this work we divide each dimension of the plane S into N^S divisions, forming $m = (N^S)^2$ rectangular regions of size $\frac{y_{\max}-y_{\min}}{N^S}$ and $\frac{u_{\max}-u_{\min}}{N_s}$. Uncertainty values are located within bins that represent normal operation states while the rest of the bins are empty because they represent states that are impossible to reach during normal operation.

The distribution of uncertainty values is used to decide the uncertainty value that corresponds to each level, for each bin i. The uncertainty values are computed based on the cumulative distribution function:

$$h_i^j = F_i^{-1}(a_j),\ j \in \{1, ..., N\}, \forall i \tag{5.13}$$

where F_i^{-1} is the inverse cumulative distribution function which returns the uncertainty value h_i^j that corresponds to cumulative function probability a_j for the bin b_i^S. The levels

$a_j, j \in \{1, ..., N\}$ are chosen arbitrary and in this work as defined as:

$$a_j = \frac{j}{N},\ j \in \{1, ..., N\}. \tag{5.14}$$

Therefore, at each time step k, the following signal is computed:

$$h^j(k) = \{h_i^j \mid (y(k), u(k)) \in S_i \subset S\}. \tag{5.15}$$

5.3.3 Fault Detection

With the N uncertainty level signals $h^j(k)$ determined at each iteration, we can proceed and compute the μ-thresholds as follows:

$$\mu(k+1) = (1-\Lambda)\mu(k) + \begin{bmatrix} h^1(k) \\ h^2(k) \\ \vdots \\ h^N(k) \end{bmatrix}. \tag{5.16}$$

following the way that Worst-Case method computes the estimation upper bound (5.8). We define $\mu_0 = 0$. The estimation error is compared against every μ-threshold to generate the warning signal $w(k) \in \{0, ..., N-1\}$ as follows:

$$w(k) = \max\{i \mid |e(k)| - \mu_i(k) > 0, i \in \{1, ..., N-1\}\} \tag{5.17}$$

Under normal operation, the robot passes through various states with different uncertainty. The warning value is tracked with $w(k)$, forming a time-series signal. By definition, $w(k)$ could fluctuate between values 0 and $N-1$ under normal operation. Irrespective of the robot state, the value $w(k)$ indicates how critical the estimation error is, providing a multiple level *fault indication*, with high values being more critical but more rare. A fault is detected when the $w(k)$ signal's distribution deviates, and this can be detected using a change-detection method. In this work, change-detection is achieved using a moving average filter with window τ:

$$b(k) = \frac{1}{\tau} \sum_{i=k-\tau}^{k} w(i) \tag{5.18}$$

Training datasets which include cases with and without faults, can be used to evaluate an upper bound $\bar{b}$ of the $b(k)$ signal for the time period K, using the above methodology, which maximizes certain evaluation metrics; this is further discussed in the Section 5.4.

This upper bound, applied on the moving average signal defines one of the two detection thresholds for the MT method:

$$d(k) = \begin{cases} 1 & \text{if } |e(k)| - \mu_N(k) > 0 \text{ — FaultST} \\ 1 & \text{if } b(k) - \bar{b} > 0 \text{ — FaultMT} \\ 0 & \text{if } b(k) - \bar{b} \leq 0 \text{ — No-Fault} \end{cases} \tag{5.19}$$

The MT method uses two conditions to detect a fault, FaultST which works on the same principle as in the Worst-Case method i.e. when the absolute estimation error exceeds the upper μ_N-threshold, and FaultMT i.e. when the $b(k)$ value exceeds the detection threshold $\bar{b}$. The fault detection alarm is raised when either of the two conditions is met and the fault detection signal switches from 0 to 1 but is possible to return back to 0. Despite this, the fault detection alarm remains raised if a fault is detected at any point in time, i.e. $D(k) = 0, \forall k < k_0$ and $D(k) = 1, \forall k \geq k_0$, where $D(k)$ is the fault detection alarm and k_0 is the time instant when d(k)=1 for the first time.

5.3.4 Implementation

In practice, the fault detection algorithm can be implemented on-board the robot's microcontroller, in order to have access to the input signal of the motors and sensor readings. As mentioned earlier, a fault may occur on any of the robot's motors in the form of performance degradation i.e. under-perform by a certain percentage. It is also possible to have the controller notified as well, and this has the advantage of immediately acting upon a fault, to prevent any consequences or even to correct the behavior by compensating the effects of the faults. Furthermore, we assume that the robot is equipped with a basic navigation controller that steers the robot towards a pre-planned path using navigation-points. "Normal operation" is the act of assigning random navigation-points, forming a path, and letting the robot follow this path. Training data is defined as data covering as many system states as possible, in other words, a large number of random paths followed by the robot. Possible states are the states that the controller can lead the robot in, rather than the states the individual components of the robot can reach. Training data can be generated by issuing random paths and supervise the execution in order to ensure fault-free operation.

Faults are emulated in software and are injected through the control signals of the robot. This is useful because it allows a controlled way to cause faults to the motors with

a specific magnitude. This way, we can evaluate the proposed method's detection performance under various fault magnitude.

5.4 Simulations

5.4.1 Experimental Setup

Evaluation of this method is performed with simulations using the V-REP simulator [88]. The algorithms for fault detection, controlling the robot and collecting data, are implemented in MATLAB which runs synchronously with the simulator, at a rate of 10Hz. The detection algorithm runs at the same rate i.e. the iteration of the algorithm is 100ms. Uncertainty drawn from a Gaussian distribution is added to the data. During a simulated experiment, the robot follows a path which is formed by a collection of navigational points, which act as Bezier points forming a series of curves. Figure 5.2 depicts the simulation environment, including the robot and its corresponding path.

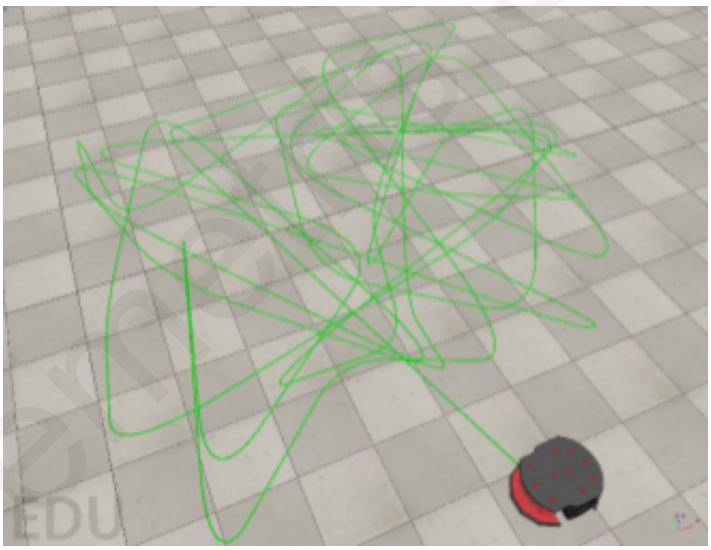


Figure 5.2: The simulation environment includes a differential drive robot moving on a random path driven by a navigation controller.

Faults of predetermined magnitude are injected on the robot through the simulator at specific times requested by the MATALB script. The fault causes the simulated robot to behave as if it was experiencing a performance degradation on its motor. This approach allows us to evaluate the performance of the detection algorithms with respect to different fault characteristics.

5.4.2 Learning from Training Data

Section 5.3.2 describes the procedure for learning the warning level from training data. In this paragraph we describe how this applies on the training dataset that was acquired through simulations. Figure 5.3 depicts the uncertainty values calculated from the measurements and projected on the $\mathcal{Y}, \mathcal{U}$ plane which is divided into regions. The lines in the plane define the boundaries of the regions and the values within each region are grouped into the corresponding bin.

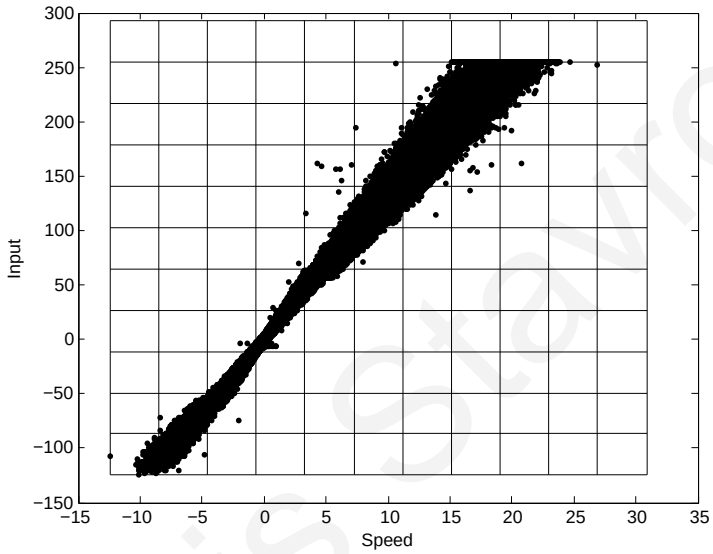


Figure 5.3: Uncertainty readings are grouped together into bins. The bins' boundaries are defined by their corresponding region.

The distribution formed by the set of uncertainty values in each bin determines the h values of the corresponding bin given the set of levels. Figure 5.4 depicts the uncertainty cumulative distribution function and the distribution of a single bin, with dashed lines representing the levels a^{10}, a^9, a^8 and a^7 which determine the values h^{10}, h^9, h^8 and h^7. The same procedure is repeated for every bin. The experimental results were conducted with $N = 10$. Having determined the h values for each bin, the MT μ-thresholds can now be calculated.

The MT method's detection threshold $\bar{b}$ is the next important detection parameter that needs to be determined from training data. We use two metrics for evaluating the detection performance, being the F1 score and the detection delay. The F1 score is the first metric of choice because it considers both the precision and sensitivity of the detection.

$$F_1 = 2\frac{\text{precision} \times \text{sensitivity}}{\text{precision} + \text{sensitivity}} = \frac{2TP}{2TP + FP + FN} \tag{5.20}$$

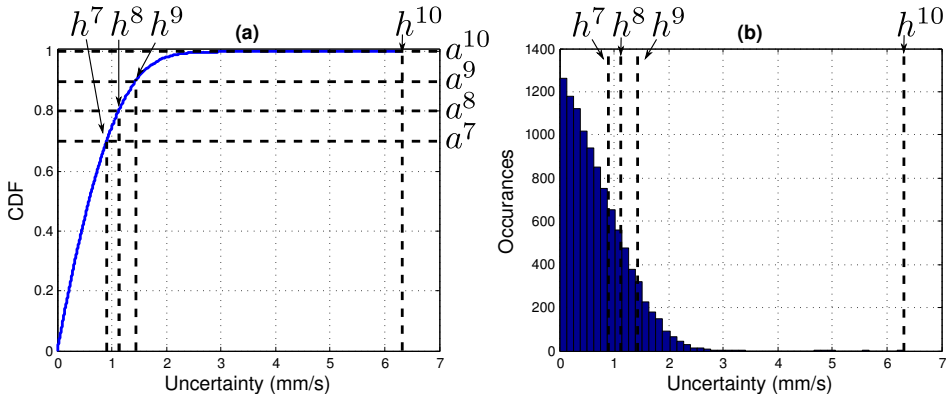


Figure 5.4: The (a) cumulative distribution function and (b) distribution of a single bin's uncertainty values. Also illustrated are the levels a^{10}, a^9, a^8 and a^7 which determine the values h^{10}, h^9, h^8 and h^7.

The second metric *detection delay* which is measured in algorithm iterations, is defined as the delay between the fault occurrence and the time the fault is detected by the algorithm. Depending on whether a fault was detected or not, an experiment is classified by the detection algorithm as "Fault Occurred" or "No Fault". Each decision is then evaluated with respect to the true conditions of the experiment i.e. a "Fault Occurred" decision is classified as True Positive or False Positive if a fault occurred during the robot operation or not, respectively. Similarly, a "No Fault" decision is classified as False Negative or True Negative.

We chose detection delay as a metric because raising the alarm as early as possible is important, especially in cases where a fault has high cost on the system performance and could escalate. Even though the lowest possible detection delay is desirable, the maximum acceptable delay is application dependent. In the same context, we define *acceptable detection delay* as the detection delay under which a detection is considered as True Positive, which is also measured in algorithm iterations. In other words, when a fault occurs and the algorithm detects it with detection delay lower than or equal to the acceptable detection delay, then the detection is classified as True Positive, otherwise is classified as False Negative.

We proceed with evaluating the detection over a range of thresholds and acceptable detection delay as an overview of how the proposed algorithm performs and demonstrate how to select the detection threshold for the FaultMT condition. Figure 5.5(a) illustrates this result of F1 score over a bounded spectrum of the threshold and acceptable detection delay plane.

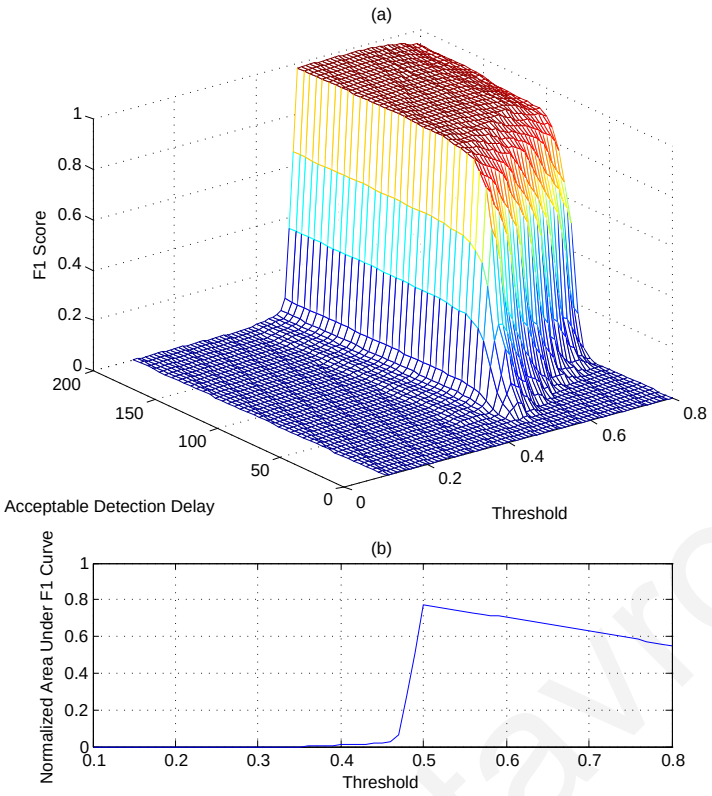


Figure 5.5: (a) F1 score over a wide range of threshold in $[0.1, 0.8]$ and acceptable detection delay in $[0, 200]$ values. (b) Normalized area under the F1 curve to determine which is the best performing overall threshold.

For the window value $\tau = 100$ chosen for the moving average filter, the figure shows that low thresholds ($\bar{b} < 0.46$)) have low F1 score irrespective of the acceptable detection delay because of the high number of False Positives. Higher thresholds ($\bar{b} > 0.55$) have low F1 score when acceptable detection delay is low, due to the high number of False Negatives. This is caused because the $b(k)$ signal requires more time to reach the threshold therefore on low acceptable detection delays, all cases are classified as 'Not Faulty' which causes False Negatives. To determine which threshold performs best overall the acceptable detection delay, we use the area under the F1 curve as the metric and the result is shown in Fig. 5.5(b). The point $\bar{b} = 0.5$ provides the maximum normalized area under F1 curve of 0.771 and therefore is chosen as the threshold.

5.4.3 Fault Detection

The following results demonstrate fault detection using both FaultST and FaultMT fault detection conditions for comparison. The robot is given a set of navigation points to follow and in the middle of the experiment $K^f = 500$, a fault with 35% magnitude is

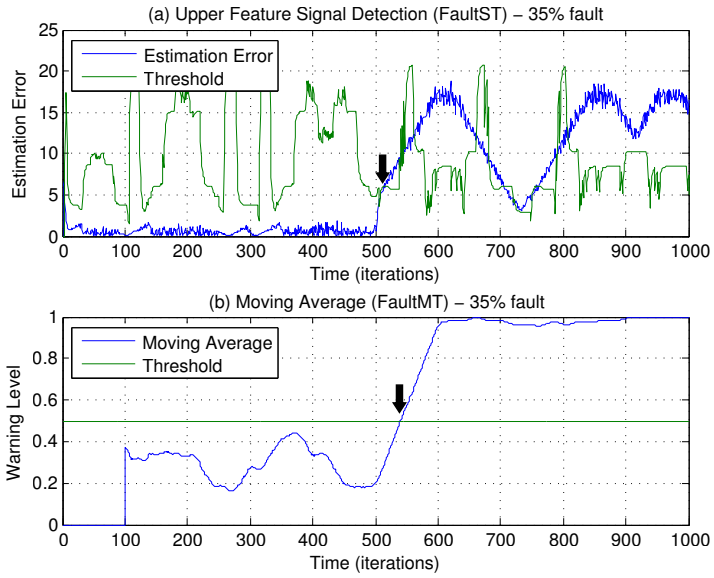


Figure 5.6: Fault injected at $K^f = 500$. (a) Estimation error and upper μ_N-threshold used for FaultST. Fault is detected at $k = 506$, as indicated by the arrow. (b) Moving average signal and the corresponding threshold used for FaultMT. Fault detected at $k = 541$, as indicated by the arrow.

injected on its left wheel. Injecting the fault at this time, allows enough normal operating time before the fault occurs, in order to determine whether an alarm corresponds to the fault or not; if the alarm is raised before K^f this indicates a False Positive. Figure 5.6(a) shows the estimation error and threshold used by the FaultST condition. During the healthy period of the experiment ($k < 500$) the estimation error correctly stays below the threshold indicating normal operation. However, at $k = 506$ soon after the fault occurs, estimation error exceeds the threshold and the fault is detected.

Next we illustrate the warning-based detection of the MT method i.e. using the FaultMT condition. Figure 5.6(b) shows the moving average signal against the detection threshold at $\bar{b} = 0.5$. The moving average filter requires an initialization period which depends on the τ value. During the normal operation, the signal correctly stays below the threshold. After the fault occurs, the signal rises steadily and exceeds the threshold at $k = 541$, detecting the fault.

The next experiment demonstrates the detection results for a small fault of 6.9% magnitude. Figure 5.7(a) shows that due to the small fault magnitude, the estimation error never grows enough, in order to exceed the threshold and therefore the condition FaultST never detects the fault in this example. On the other hand, Fig. 5.7(b) shows that the moving average signal exceeds the threshold and therefore FaultMT detects the fault at

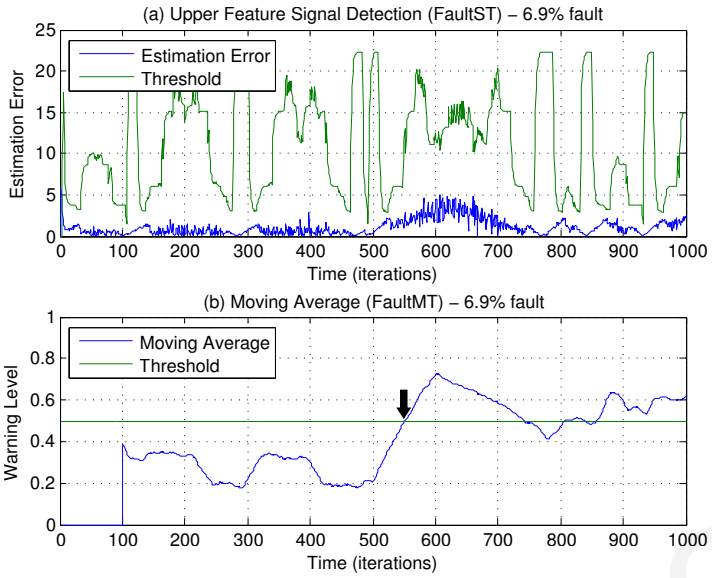


Figure 5.7: Fault injected at $K^f = 500$. (a) Estimation error and upper μ_N-threshold used for FaultST. Fault is not detected. (b) Moving average signal and the corresponding threshold used for FaultMT. Fault detected at $k = 551$, as indicated by the arrow.

$k = 551$.

Due to the fact that the fault is small, the generated estimation error is limited and as the results show, it is not enough to be significant with respect to uncertainty in order to trigger the FaultST condition and thus the Worst-Case method fails to detect it. The FaultMT requires more time but eventually it does detect the fault successfully. This example demonstrates the conservative nature of the Worst-Case method that uses FaultST only and the improved sensitivity of the MT method that uses both FaultST and FaultMT simultaneously.

5.4.4 Performance Evaluation

The data for the evaluation is collected by running 200 experiments of 1,000 iterations each, where the robot operates normally, without any faults. Paths for these experiments are generated randomly. Additionally, the exact same paths are used to execute a set of 200 experiments during which a fault is injected at $K^f = 500$. In each experiment, the performance degradation fault has random magnitude. Using these experiments we compare the FaultST and FaultMT conditions, and demonstrate the differences in detection performance. The two evaluation metrics used are the F1 score and the detection delay.

The detection threshold for the FaultMT method used is $\bar{b} = 0.5$ which was deter-

mined from training data, based on the moving average window $\tau = 100$. Figure 5.8 shows the F1 score of each condition over different accepted detection delay values. In the same figure, we also include the F1 score profile for two additional moving average windows $\tau = 10$ and $\tau = 200$ for comparison; the corresponding thresholds for those windows are $\bar{b} = 0.89$ and $\bar{b} = 0.45$. The FaultST condition exhibits high F1 score even in low values of accepted detection delay. This is because the FaultST requires only few iterations before the estimation error crosses the detection threshold. The FaultMT condition has low F1 score on lower values because of the moving average filter which causes slow rising of the warning signal, taking more time to reach the threshold.

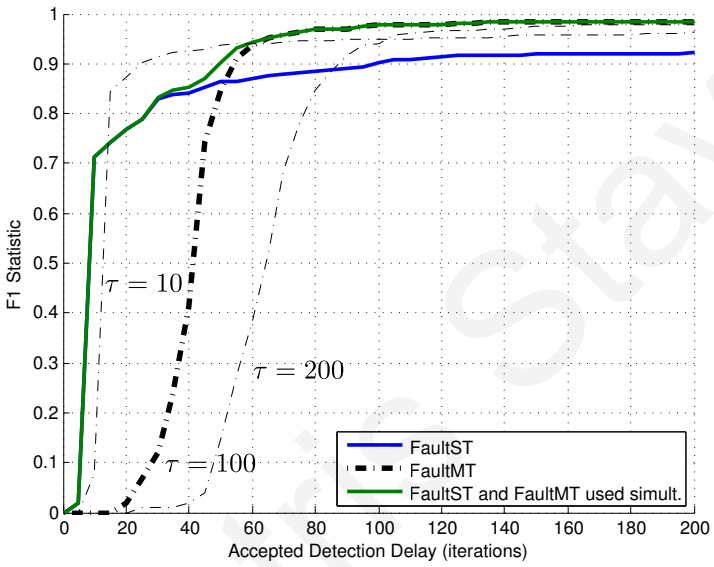


Figure 5.8: F1 score with respect to the Accepted Detection Delay using the upper μ_N-threshold and Moving Average. The Worst-Case method uses FaultST only, while the MT method utilizes both conditions to achieve an overall better detection performance.

As the acceptable detection delay increases, the FaultST F1 score improvement is limited. The FaultMT catches up and performs equally well to the FaultST when acceptable detection delay is 51; at this point, the signal from the moving average is given enough time to exceed the threshold in cases where a fault occurs. For higher values of acceptable detection delay, the FaultMT outperforms the FaultST, with best F1 score of 0.985 when acceptable detection delay is 200. This happens because when the FaultMT is given enough time, it is able to detect faults of smaller magnitude which are not detected by the FaultST condition. This can be seen in Table 5.1 that lists the detection results for a number of acceptable detection delays. Also listed in the table, is the smallest fault magnitude that was detected. As the acceptable detection delay is increased, the FaultMT is

able to detect more faults that where previously undetected, and the decreasing value of the detected fault magnitude, indicates the increasing sensitivity. For 200, the FaultMT returned only 6 False Negative cases with the smallest fault detected at 0.0137 while the FaultST, returned 29 and 0.1414 respectively. The MT method proposed in this chapter, uses both FaultST and FaultMT simultaneously, making it fast to detect faults (FaultST) but also sensitive enough to detect faults of small magnitude (FaultMT).

Table 5.1: Comparison of detection conditions FaultST and FaultMT with respect to fault detection metrics and the Acceptable detection delay.

	FaultST detection condition					FaultMT detection condition				
Acc.	TP	FP	TN	FN	minimum α	TP	FP	TN	FN	minimum α
0	0	0	200	200	-	0	0	200	200	-
20	125	0	200	75	0.2480	2	0	200	198	0.3828
40	145	0	200	55	0.1750	52	0	200	148	0.1108
60	154	0	200	46	0.1662	176	0	200	24	0.0937
80	159	0	200	41	0.1662	188	0	200	12	0.0487
100	164	0	200	36	0.1662	191	0	200	9	0.0481
120	168	0	200	32	0.1662	192	0	200	8	0.0481
140	169	0	200	31	0.1662	194	0	200	6	0.0137
160	170	0	200	30	0.1485	194	0	200	6	0.0137
180	170	0	200	30	0.1485	194	0	200	6	0.0137
200	171	0	200	29	0.1414	194	0	200	6	0.0137

The moving average window can be changed to alter the behavior of the FaultMT condition. A smaller window value, for example $\tau = 10$, will cause the detection to be faster, as shown by the corresponding line in Fig. 5.8. The threshold for $\tau = 10$ filter is $\bar{b} = 0.89$ which is determined through the training procedure. Because of the higher threshold when compared to the $\tau = 100$ filter, some of the faults are not detected thus increasing the False Negatives making the detection less sensitive. On the other hand, a large window will smooth the warning signal too much, requiring more time to detect a fault as shown by the corresponding line in Fig 5.8. In this case, the filter requires more time to detect a fault which sometimes exceeds the acceptable detection delay and registers as False Negative making the detection again less sensitive.

5.5 Concluding Remarks

The chapter presents a new detection condition FaultMT that incorporates multiple μ-thresholds derived from the distribution of the uncertainty in order to produce a warning signal that if given enough time, detects small magnitude faults. The simulation evaluation shows that the FaultST condition is able to detect faults fast, but fails to detect faults of small magnitude. The FaultMT on the other hand uses the temporal behavior of the warning signal and it able to detect faults of small magnitude even though it needs more time to raise the alarm. In fact, FaultMT detection condition detects faults down to 1.37% compared to 14.14% of the FaultST condition and if given enough time, it reaches the score of 0.985 based on the F1 metric. The presented Multiple Thresholds method utilizes both the FaultST and FaultMT conditions making it fast to detect faults and also able to detect smaller magnitude faults with some delay that would not be detected otherwise. To make the method generally applicable to mobile robots, the required detection parameters are determined from training data.

Chapter 6

Reacting to Faults

6.1 Introduction

With fault detection, the controller of the robot becomes aware of the occurrence of a fault. Depending on the fault magnitude, the robot could be able to continue its operation by reacting to the fault. Through fault isolation and identification, the location, type and magnitude of the fault are learned by the robot and this information is used to mitigate the effects of the fault. Since this thesis considers service robots which are mobile, we investigate faults that affect the navigation of the robot within the environment. In the first part, we present a module that can be installed on a service robot that provides path correction through fault detection, isolation and identification. The module uses signals sent to/from the controller to provide this functionality. Information about the fault is used by the module to generate new actuator signals that allow the controller to continue controlling the robot. In the second part, we investigate the localization capabilities of the robot under sensor faults. Localization is an important part for robot navigation within an environment since it provides information about the location of the robot within a frame of reference. A fault on the sensors affects localization resulting in wrong location estimation. In this case, it is important to continue localization using only the healthy sensors. In the presented work, we explore the performance of Monte Carlo Localization when only one of the sensors remains operational.

6.2 Fault Detection, Isolation and Path Correction Module

In this work, a model-based *Fault Detection, Isolation and Path Correction Module* (FDI-PC) is presented, suitable for differential drive service robots. The first step of the FDI-PC is to determine whether an actuator fault has occurred, based on the measured states and the control inputs. The second step is to determine the type of fault that has occurred, i.e. whether the fault affects the left or right wheel, and to identify its magnitude using adaptive estimation. The third step is to modify the control input using a path-correction algorithm so that the robot maintains its ability to reach its goal.

6.2.1 Robot Model and Fault Dynamics

The floor region is defined as $W \subseteq \mathbb{R}^2$. Let $\mathcal{O}$ be the union of all regions within W which are inaccessible for robot movement, due to static objects (e.g. desks and chairs) that occupy that space. The feasible robot movement region is defined as $S = W - \mathcal{O}$. The robot's ability to reach any point in S depends on the connectivity of the areas and its starting position. Besides static obstacles, robots need to avoid moving obstacles, including other robots, and this is handled by a path-planning supervisory system.

The configuration state $\mathbf{x}$ of the robot is defined by its position in the Cartesian plane (coordinates x_1 and x_2) and its orientation (angle x_3), denoted by $\mathbf{x} = [x_1, x_2, x_3]^\top$, $\mathbf{x} \in S \times [-\pi, \pi]$ as shown in Fig. 6.1.

The dynamic equations describing the state of the robot are given by

$$\dot{\mathbf{x}} = \begin{bmatrix} \frac{1}{2}\cos(x_3) & \frac{1}{2}\cos(x_3) \\ \frac{1}{2}\sin(x_3) & \frac{1}{2}\sin(x_3) \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \eta(\mathbf{x}, v) \tag{6.1}$$

where d is the distance between the two wheels and $v = [v_1, v_2]^\top$ is the linear velocity of two motors controlling the left and right wheel respectively. The first term of the state-space describes the known nominal dynamics. The second term, $\eta(x, v)$, corresponds to the unknown modeling uncertainty vector caused by the differences between the nominal and the actual movement and rotation response of the robot (e.g. caused by inertial effects). This is assumed unknown to the controller.

We consider that the indoor environment has a robot localization system, e.g. using cameras. The output vector of the robot states is given by $y = x$. Each robot has to

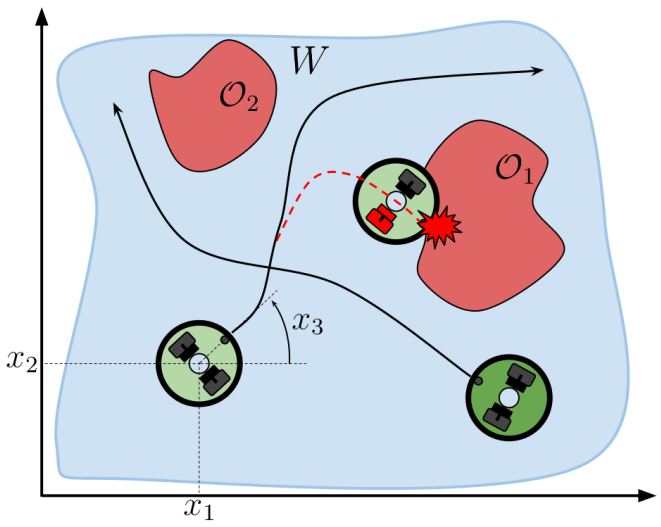


Figure 6.1: A simplified robot movement environment with two robots and two inaccessible regions $\mathcal{O}_1$ and $\mathcal{O}_2$. Each robot has a trajectory to follow within the region $S = W - (\mathcal{O}_1 \cup \mathcal{O}_2)$ to achieve its goal. A fault on the right-actuator of the first robot causes the robot to detour and collide on $\mathcal{O}_1$.

complete certain tasks (goals) within the environment. Let $g \in S$ be the goal of the robot within a certain period. Given the goal g, the current location of the robot y and the feasible region for movement S, a higher-level Supervisory System, as in Fig. 6.2, implements a *Motion Planning Module* to compute the path r that will guide the robot to its destination, comprised of all the intermediate points the robot must navigate.

The robot is equipped with a *Controller Module* which computes the motor control signal w, and u represents the actual input of the motor. We clarify that u and w are the same signal during normal operation, but under certain circumstances that will be presented next, u becomes a modified version of w. For simplicity, we assume here that the input value represents the required motor velocity, therefore under normal operation (no faults present) $v = u$. However, in the special case when a fault has occurred on the j-th actuator of the robot, at time T_j^f, the j-th motor's performance may degrade partially or completely by some unknown factor $\alpha_j \in [-1, 0]$, and the faulty robot input is given by

$$v_j(t) = \Big(1 + \beta(t - T_j^f)\alpha_j\Big)\, u_j(t), \tag{6.2}$$

where $\beta(t - T_j^f)$ is the time profile of the fault. In the case of abrupt faults, $\beta(t - T_j^f) = 1$ if $t \geq T_j^f$ and $\beta(t - T_j^f) = 0$ otherwise.

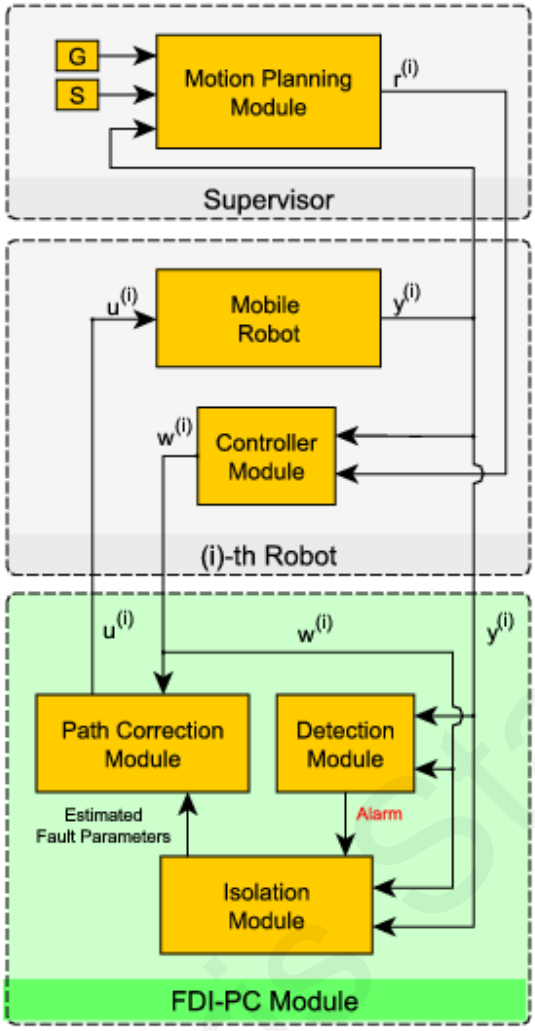


Figure 6.2: System architecture, comprised of: a) the supervisory system for the robots motion path planning, b) the (i)-th mobile robot system, including the controller and c) the proposed FDI-PC Module.

By substituting (6.2) in (6.1), the additional fault dynamics term appears, such that

$$\dot{\mathbf{x}} = \begin{bmatrix} \frac{1}{2}\cos(x_3) & \frac{1}{2}\cos(x_3) \\ \frac{1}{2}\sin(x_3) & \frac{1}{2}\sin(x_3) \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \eta(x,u) + \begin{bmatrix} \frac{1}{2}\cos(x_3) & \frac{1}{2}\cos(x_3) \\ \frac{1}{2}\sin(x_3) & \frac{1}{2}\sin(x_3) \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix} \begin{bmatrix} \beta(t-T_1^f)\alpha_1 u_1 \\ \beta(t-T_2^f)\alpha_2 u_2 \end{bmatrix}, \tag{6.3}$$

where T_1^f and T_2^f are the actuator fault occurrence times on the left and right wheel respectively.

This work considers single-actuator faults that may occur on the robot, thus the set of

faults Φ is given by

$$\Phi = \left\{ \alpha_1 \begin{bmatrix} \frac{1}{2}\cos(x_3)u_1 \\ \frac{1}{2}\sin(x_3)u_1 \\ -\frac{u_1}{d} \end{bmatrix}, \alpha_2 \begin{bmatrix} \frac{1}{2}\cos(x_3)u_2 \\ \frac{1}{2}\sin(x_3)u_2 \\ \frac{u_2}{d} \end{bmatrix} \right\}.$$

6.2.2 FDI-PC Design Architecture

In this work, the *Fault Detection, Isolation and Path Correction Module* (FDI-PC) for two-wheeled non-holonomic mobile robots is proposed, based on the model-based fault detection and isolation scheme presented in [36, 37, 113].

The FDI-PC module will be implemented as an embedded system, thus, the following discrete-time definitions are required: let k be the discrete time instance such that $k\Delta t = t$, where Δt is the sampling time. Let $x(k)$ be the discrete state vector, and let the state matrix be denoted as

$$A(x(k)) = \Delta t \begin{bmatrix} \frac{1}{2}\cos(x_3(k)) & \frac{1}{2}\cos(x_3(k)) \\ \frac{1}{2}\sin(x_3(k)) & \frac{1}{2}\sin(x_3(k)) \\ -\frac{1}{d} & \frac{1}{d} \end{bmatrix}.$$

Considering the forward Euler discretization, the discrete-time state space mobile robot dynamics can be given by

$$\begin{aligned} x(k+1) &= x(k) + A\left(x(k)\right)u(k) + \eta(k) + \beta(k-K)\phi(x(k), u(k)) \\ y(k) &= x(k) \end{aligned}$$

where $u(k) \in [u_{min}, u_{max}]$ is the robot input vector bounded within the feasible control inputs u_{min} and u_{max}, $y(k)$ is the measurable state vector, $\eta(k)$ is the unknown modeling uncertainty term, $\phi(x(k), u(k))$ is the actuator fault dynamics.

Both actuators introduce uncertainty $\eta(k)$ to the model. In general, its magnitude is unknown during operation. However, through experimentation, an upper bound of the uncertainty $\overline{\eta}(k)$ can be selected, such that

$$\overline{\eta}(k) = |A(x(k))| \cdot \overline{M} \cdot |u(k)| \geq \eta(k)$$

where the diagonal matrix $\overline{M}$ is the maximum actuator uncertainty. Uncertainty can also be caused due to the dynamics of the robot. As with the rest of the uncertainties in this work, unmodeled dynamics uncertainty has an upper bound which can be incorporated into the estimation error upper bound which ultimately shapes the detection

threshold. In this work we assume that actuator accelerations are small, well below the robot's actuator performance making the unmodeled dynamics uncertainty small relative to the rest of the uncertainties [48]. Therefore, we do not explicitly include the dynamics uncertainty as it is completely covered by other included uncertainties. In cases where the accelerations assumption is not valid, the dynamics uncertainty can be incorporated into the detection threshold with an additional uncertainty term on eq. (6.1). One would need to identify the upper bound and time delay of this uncertainty either through the specifications of the actuator or through training.

The FDI-PC Module is comprised of three estimators, $\hat{\mathbf{x}}^0$ for the states used in the Detection Module and $\hat{\mathbf{x}}^1$, $\hat{\mathbf{x}}^2$ for the states used in the Isolation Module.

Fault Detection Module

The fault detection state estimator $\hat{\mathbf{x}}^0$ takes into account the mobile robot state at time k, the nominal dynamics and the filtered estimation error to estimate the state at time $k+1$, such that

$$\hat{\mathbf{x}}^0(k+1) = \hat{\mathbf{x}}^0(k) + A(y(k))u(k) - \Delta t \cdot \Lambda(\hat{\mathbf{x}}^0(k) - y(k)),$$

where $\hat{\mathbf{x}}^0(0) = y(0)$, and the filter coefficient matrix Λ is a diagonal matrix of size (3×3) and $\Lambda_{(j,j)} > 0$ for the j-th state, which can be determined experimentally.

At every iteration the estimation error of the j-th state is calculated using the state measurement vector $y_j(k)$ and the estimated state $\hat{\mathbf{x}}_j^0(k)$, such that $e_j^0(k) = y_j(k) - \hat{\mathbf{x}}_j^0(k)$. The absolute value of the estimation error $e_j^0(k)$ for the j-th state has a theoretical upper bound $\bar{\epsilon}_j^0(k)$ (detection threshold) which is calculated based on the uncertainty upper bound $\bar{\eta}(k)$, thus

$$\bar{\epsilon}_j^0(k+1) = (1 - \Delta t \cdot \Lambda_{(j,j)})\bar{\epsilon}_j^0(k) + \bar{\eta}_j(k) \tag{6.4}$$

where $\bar{\epsilon}_j^0(0) = 0$. For further details regarding the derivation of the fault detection algorithm and its properties, see [37].

To detect if a fault has occurred, the *Fault Detection Module* compares the absolute estimation error of the j-th state, $|e_j^0(k)|$ whether it is greater than the detection threshold $\bar{\epsilon}_j^0(k)$. In practice, due to quantization errors caused due to analog-to-digital conversion as well as due to continuous-to-discrete time model transformation, an additional error appears which may cause false positive alarms. Through simulations or analytically it is possible to calculate the maximum discretization/quantization error δ_j affecting the j-th

estimation error. The fault detection logic is therefore

$$\text{Alarm:} \begin{cases} \text{Yes} & \text{if } |e_j^0(k)| - \overline{e}_j^0(k) - \delta_j \geq 0 \\ \text{No} & \text{if } |e_j^0(k)| - \overline{e}_j^0(k) - \delta_j < 0 \end{cases}$$

Fault Isolation Module

The *Fault Isolation Module* objective is to determine which type of actuator fault has occurred and to estimate its magnitude, after it has been activated by the alarm from the *Fault Detection Module*. This module is comprised by a set of two estimators and an isolation logic, each estimator corresponding to an actuator fault type. After isolation is activated, the module continuously checks whether an isolator has crossed the threshold.

The intuition behind fault isolation, is that, in the case of a certain actuator fault type, the absolute estimation error should be below the isolation threshold computed using the same fault type and maximum uncertainty.

When the isolation threshold is crossed, the fault represented by that isolator is no longer considered as a valid candidate. When one fault type remains as valid candidate, the fault is considered isolated. Specifically in the case of actuator faults in mobile robots, the following two fault structures are considered,

$$\Phi = \left\{ \alpha^1 z^1(y(k), u(k)), \alpha^2 z^2(y(k), u(k)) \right\}$$

where $u_1(k)$ and $u_2(k)$ are the actuator inputs, and for the l-th fault structure α^l is the fault magnitude and $z^l(k)$ the regressor vector, which are given by

$$z^1(y(k), u(k)) = \Delta t \begin{bmatrix} \frac{1}{2}\cos(y_3(k)) \\ \frac{1}{2}\sin(y_3(k)) \\ -1/d \end{bmatrix} u_1(k),$$

$$z^2(y(k), u(k)) = \Delta t \begin{bmatrix} \frac{1}{2}\cos(y_3(k)) \\ \frac{1}{2}\sin(y_3(k)) \\ 1/d \end{bmatrix} u_2(k),$$

for left and right-wheel faults, respectively. For notational simplicity, in the following we consider that $z^l(k) = z^l(y(k), u(k)))$.

For the l-th isolator, let $\hat{\mathbf{x}}^l(k)$ be the estimated state and $e^l(k)$ be the estimation error, such that $e^l(k) = y(k) - \hat{\mathbf{x}}^l(k)$. Let $\hat{\alpha}^l(k)$ be the estimated fault magnitude, which is computed through the adaptive law

$$\hat{\alpha}^l(k) = \hat{\alpha}^l(k-1) + \gamma^l z^l(k-1) \left(e^l(k) - (1 - \Delta t \cdot \Lambda) e^l(k-1) \right)$$

for which γ^l is the adaptive gain [1]. To guarantee that $\hat{\alpha}^l(k)$ remains within the expected fault magnitude is bounded within $\hat{\alpha}^l(k) \in [-1, 0]$. In specific, the following rule is considered: if $\hat{\alpha}^l(k) > 0$ or $\hat{\alpha}^l(k) \leq -1$, then $\hat{\alpha}^l(k) = \hat{\alpha}^l(k-1)$.

Finally, the estimated state dynamics of the l-th isolator are given by

$$\hat{\mathbf{x}}^l(k+1) = \hat{\mathbf{x}}^l(k) + A(y(k))u(k) - \Delta t \cdot \Lambda(\hat{\mathbf{x}}^l(k) - y(k)) + \hat{\alpha}^l(k)z^l(k). \tag{6.5}$$

Similar to the *Fault Detection Module*, the absolute estimation error of the isolator has an upper bound that defines the isolation threshold, given by

$$\overline{\epsilon}_j^l(k+1) = (1 - \Delta t \cdot \Lambda)\overline{\epsilon}_j^l(k) + \overline{\eta}_j(k) + (|\alpha_j^l(k)| + 1)|z_j^l(k)|. \tag{6.6}$$

For further details regarding the derivation of the fault isolation algorithm and its properties, see [37].

Path Correction Module

After isolation, the module is able to make corrections to compensate the effect of the actuator fault, if possible, by adjusting the robot input vector. At this point is should be pointed out that the controller of the robot does not require any altering, which is an important objective of this work, since the FDI-PC Module suggested, is plugged onto the existing robot hardware.

The *Path Correction Module* is used to compute the robot input vector $u(k)$, based on the *Controller Module* output vector $w(k)$, the isolated fault structure and the approximated fault magnitude. The objective is to maintain a course as close as possible to the desired path, taking into account the most updated fault magnitude estimation $\hat{\alpha}^l(k)$ from the *Fault Isolation Module* for the l-th fault type. In the case that the corrected control signal for one actuator is greater than the upper bound u_{max}, then the Module makes an additional correction by modifying the control signals for both wheels so that the velocity ratio is maintained. This would allow the robot to follow its desired path in a slower speed.

The robot input vector $u(k)$ can be computed as follows

$$u(k) = \begin{cases} [u_{max},\ u_2'(k) - (u_1'(k) - u_{max})]^\top & \text{if } \frac{w_1(k)}{1+\hat{\alpha}^1(k)} \geq u_{max} \\ [u_1'(k) - (u_2'(k) - u_{max}),\ u_{max}]^\top & \text{if } \frac{w_2(k)}{1+\hat{\alpha}^2(k)} \geq u_{max} \\ u'(k) & \text{otherwise} \end{cases}$$

where

$$u'_j(k) = \begin{cases} \frac{w_j(k)}{1+\hat{\alpha}^j(k)} & \text{if Fault Type } j \in \{1,2\} \text{ isolated} \\ w_j(k) & \text{otherwise} \end{cases}$$

6.2.3 Simulation Results

To evaluate the performance of the proposed algorithm, a simulator was developed to operate a virtual robot and provide the necessary data for fault detection and isolation. An illustrative example is used to demonstrate the fault effects on the robot and to depict the operation of the detection and isolation modules. Afterwards, 10,000 random path scenarios are simulated to evaluate the effectiveness of the fault detection and isolation modules. Finally, the operation of the FDI-PC module is demonstrated in a case study where a mobile service robot is moving towards a specific goal in a small office environment, under an actuator fault.

Illustrative Example

A mobile robot moves within a plane without obstacles, i.e. $S = \mathbb{R}^2$, to achieve 3 consecutive goals, and is provided with a path $r(k)$ to follow. At time $K^f = 1900$ a fault occurs. The fault affects the right wheel, causing a 60% performance loss, i.e. $\alpha_2 = -0.6$, causing it to slow down and drift to the right. Figure 6.3 shows the absolute estimation error on the three states. The dashed line represents the detection threshold for each state. The detection threshold changes in each iteration of the algorithm, based on the robot state, the input and the uncertainty as it was described in (6.4). When the absolute estimation error crosses this threshold, the fault is detected. As shown in the figure, during the fault-free operation, the absolute estimation error is always below the dashed line. However, soon after the fault occurs, the absolute estimation error in all three states crosses the dashed line and therefore is detected.

Next, the operation of the two isolator modules is demonstrated. For this simulation the robot performs a random movement in the environment. A fault occurs at $K^f = 1000$ on the left wheel with magnitude $\alpha_1 = -0.6$. Figures 6.4 and 6.5 show the absolute estimation error of the left and right wheel isolators respectively. The dashed line shows the isolation threshold as it was described in (6.6). As shown in the figures, each isolator attempts to approximate the fault after it receives the detection signal from the the detection module just after the fault occurs. With time, the fault magnitude estimation of the

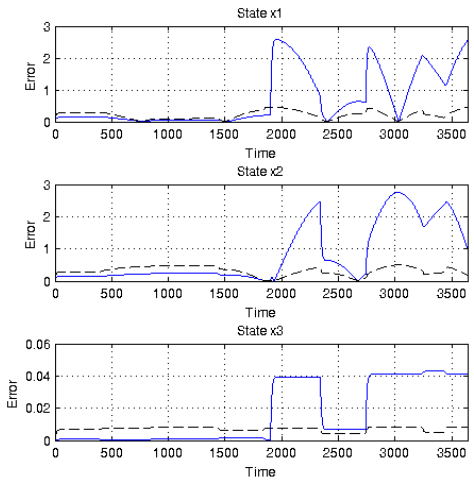


Figure 6.3: The absolute estimation error (solid blue line) and the detection threshold (dashed black line) of the three states of the robot. The fault occurs at time $K^f = 1900$. The fault is detected the first time the absolute estimation error is greater than the threshold, in any of the three states (in this case x_1 at time $k = 1903$.

left wheel isolator converges to $\hat{\alpha}^1 = -0.63$ while the estimation of the fault magnitude for the right wheel isolator fluctuates. At $k = 1450$, the right wheel absolute estimation error of the isolator crosses the dashed line and therefore the fault is isolated on the left wheel.

Random Paths Example

Table I shows the fault detection and isolation results analysis after running 10,000 simulations, where each simulation generates a random path with a random fault magnitude, random occurrence time, on one of the two wheels.

The results show that detection time drops with respect to the fault magnitude. This happens because as the fault magnitude increases, the estimation error due to the fault increases compared to the estimation error due to uncertainty therefore making the fault easier to be detected. On the other hand, low fault magnitude generates lower estimation error due to the fault, which sometimes is hidden by the uncertainty, therefore, the fault is more difficult to be detected.

Small Office Case Study

So far, the performance of the FDI-PC module of the robot was demonstrated in the detection and isolation of motor faults. A common task in mobile robotics, is reaching the

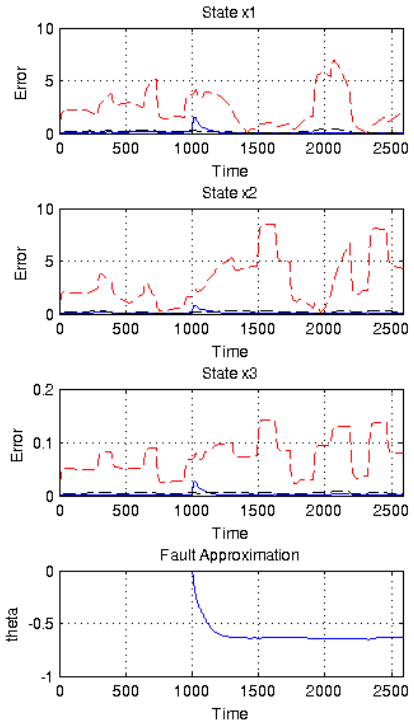


Figure 6.4: Left-wheel-fault isolator

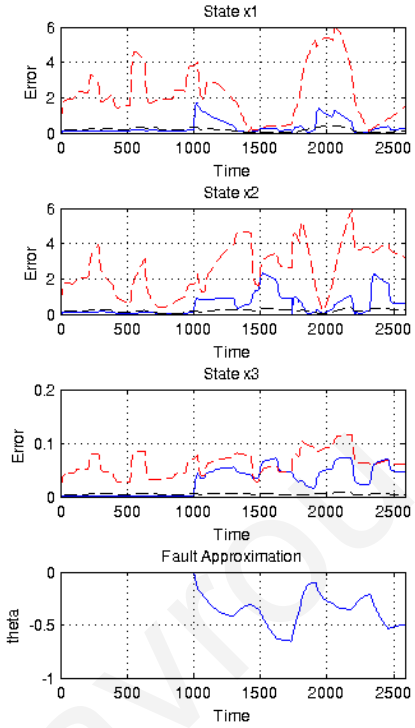


Figure 6.5: Right-wheel-fault isolator

specific point in the environment. The robot equipped with a motion planning, is able to calculate the intermediate points required to reach its goal. The robot starts moving towards the next intermediate point. The controller steers the robot in the required direction to ensure that the robot follows the path. In this work, the controller minimizes the orientation error (bearing) between the robot and the next intermediate point. To achieve this, the robot is steered in the inverse direction of the error. The next simulation illustrates this procedure.

In this simulation, one mobile robot is considered, $m = 1$, moving in a small office environment $W = [0, 6000] \times [0, 3000]$ (mm). There are several objects in the office such as desks, chairs and shelves, which are modeled as obstacles in $\mathcal{O}$. The feasible robot movement area is $S = W - \mathcal{O}$. The robot has to travel from its current position (lower left corner), i.e. the state $x(0) = [100, 100, \pi/2]^\top$, to the charging station (lower right corner), i.e the goal $g = [5900, 200]^\top$. The office is equipped with a small network of cameras that allow full state measurement of the robot.

Table 6.1: Fault detection and isolation results analysis after running 10,000 random simulations.

Description	Small Faults	Medium Faults	Large Faults
	0.05 – 0.35	0.35 – 0.65	0.65 – 0.95
Total Faults	3216	3752	3032
Detected	3216	3752	3032
Average Det. Time (ms)	7.14	2.53	1.85
Correct Isolation	1328 (41.3%)	3553 (94.7%)	3016 (99.5%)
False Isolation	3 (∼0.0%)	16 (0.4%)	14 (0.5%)
No Isolation	1885 (58.7%)	183 (4.9%)	2 (∼0.0%)
Average Iso. Time (ms)	761	434	202
RMSE Fault Est. Error	0.0383	0.0382	0.0383

The Supervisory system calculates the robot path, comprised of five intermediate points within S, shown as black squares in Fig. 6.6 and this information, $r(k)$ is passed to the robot. The mobile robot's wheel-distance is $d = 119.5$ (mm). The upper and lower motor velocities are $u_{max} = 100$ (mm/s) and $u_{min} = 0$ (mm/s) respectively. The maximum motor uncertainty is given as

$$\overline{M} = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}.$$

A motor is subject to partial or total performance degradation, of magnitude $\alpha_1, \alpha_2 \in [-1, 0]$. The robot controller, the camera system and the FDI-PC module operate at a rate of 100Hz i.e. $\Delta t = 10$ (ms).

In Fig. 6.6, the dashed red line shows the movement of the robot without the proposed FDI-PC Module. Between the first and second intermediate points, a 60% performance degradation fault occurs on the right wheel, i.e. $\alpha = -0.6$. This fault causes the robot to drift to the right. The controller is constantly trying to correct the bearing between the robot and the destination but fails to do so since the controller does not take into account the new dynamics. The solid blue line, demonstrates the same scenario when the robot is equipped with the FDI-PC Module. As illustrated, the fault is detected and isolated shortly after occurring. When the fault is isolated, the estimated magnitude of the fault is utilized by the *Path Correction Module*. The controller tries to minimize the bearing error between the robot and the second intermediate point, however, because of the FDI-PC, the controller regains control over the robot, and makes the necessary corrections to

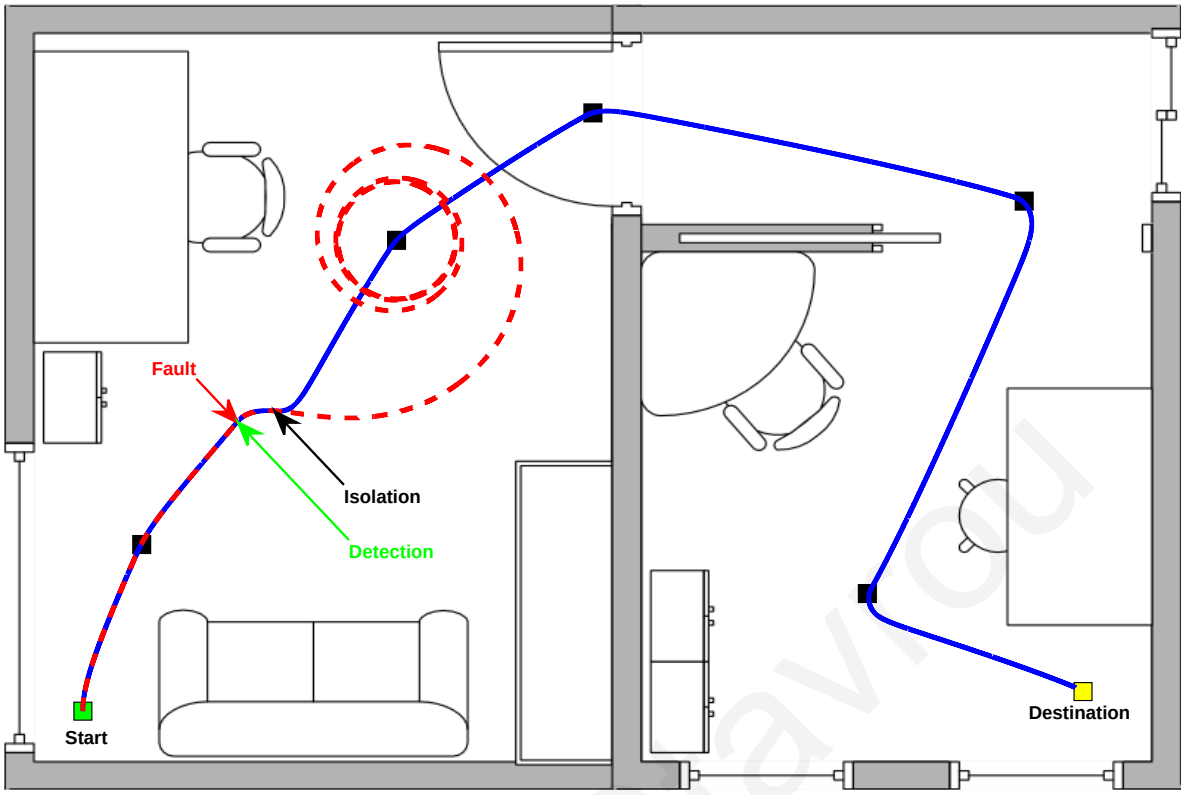


Figure 6.6: A small office environment with obstacles. The robot moves from the 'Start' location (green square) to the 'Destination' location (yellow square), passing by the five intermediate locations (black squares). The red arrow indicates the location where a right-wheel fault has occurred. The dashed red line corresponds to the robot movement without the FDI-PC Module. The blue line corresponds to the robot movement with the FDI-PC Module activated. The green arrow indicates the location where the fault is detected and the black arrow the location where the fault is correctly isolated and path correction is activated.

reach the next intermediate point. Even though the actuator fault is persistent, and affects the right motor performance for the rest of the operation, the robot is able to navigate as planned and reach its destination.

6.3 Monte Carlo localization with a single range sensor

6.3.1 Introduction

Mobile robots use localization to establish their own position and orientation (pose) within the frame of reference. A commonly used approach for localization is to use on-board sensors to gather information about the environment. Using this information along with kinematic and dynamic models, the pose of the robot can be estimated. A fault causes a sensor to return false information to the controller, having negative effects on localization causing the pose estimation to deviate. In this event, the is high risk of collision since the controller navigates the robot using the wrong pose. Having an array of sensors provides redundancy between the sensors, therefore a good pose estimate can be maintained if few of the sensors become faulty. In order to improve the pose estimate in the event of a fault, we can apply fault diagnosis to detect and isolate the faulty sensors and disable them. This will make the estimate more accurate since localization is performed using information only from healthy sensors. In this work, we consider the scenario where only one sensor remains healthy. In order to react to this event, we assume that faults were detected and all the faulty sensors were disabled. Then we switch to a localization algorithm appropriate for use with a single sensor.

We use the Monte Carlo Localization algorithm and extend it in order to perform well using only one sensor. The single sensor limitation is a key difference between the presented method and others. To the best of our knowledge, other robots use arrays of such sensors positioned at various angles around the robot. Because the environment information received by the robot is extremely limited, existing localization methods cannot be applied directly. The reason is because any robot carrying one sensor receives one-dimensional information even though the environment is two-dimensional. In contrast, a robot carrying more than one sensor is able to receive two-dimensional information. That is why important challenges are introduced when moving from a multiple sensor robot to a single sensor robot.

6.3.2 Modeling

Map

The environment representation is created using fixed decomposition. The map is a discrete transformation of the real environment which forms a grid. The transformation

is necessary for computational and memory purposes but decreases the accuracy of the world representation which ultimately affects the precision of localization. The choice of the grid resolution is chosen considering the required precision as well as the size of the map in terms of memory. The map we used was manually prepared a-priory, using an top-down design of the environment.

Kinematic Model

Roomba is a differential drive robot supported by a castor wheel on the front. The robot is circular and the wheel axle is in the middle. The robot's **pose** at time t is denoted by $\mathbf{x}_t = (x, y, \theta)^\top$, which represents the global reference frame where x and y are the coordinates, θ is the heading.

The control vector which describes the actions of the robot, is strictly derived from the odometry which is a widely accepted method. Odometry returns the distance traveled by each wheel since last time retrieved i.e. between $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ and is denoted by $Y_t = [Y^L, Y^R]^\top$. Y^L and Y^R represent odometry of the left and right wheel respectively.

Odometry is used to calculate the movement of the robot between consecutive poses. To demonstrate, let $\mathbf{x}_t$ and $\mathbf{x}_{t+k}$ be the pose of the robot at time t and $t+k$ respectively. Assuming that $\mathbf{x}_t$ and Y_t are known, there is not enough information to accurately calculate the final pose $\mathbf{x}_{t+k}$. However, if k is small, $\mathbf{x}_{t+k}$ is accurately approximated. Therefore, assuming odometry information is sampled fast enough, the control vector u is derived using

$$u_t = \begin{pmatrix} s \\ a \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} Y_t^R \\ Y_t^L \end{pmatrix}$$

where s and a are the distances that cause the robot to move forward/backward and rotate right/left respectively, both measured in mm. Positive values of s indicate forward movement while negative indicate backward movement. For a, positive means that the robot rotated to the left while negative means to the right. Because odometry is not perfect, it is affected by errors which cause it to drift over time. These are modeled by zero-mean random variables with finite variance [104]. Therefore the actual control vector is

$$\hat{u}_t = \begin{pmatrix} \hat{s} \\ \hat{a} \end{pmatrix} = \begin{pmatrix} s \\ a \end{pmatrix} + \begin{pmatrix} \epsilon_{\alpha_1|s|+\alpha_2|a|} \\ \epsilon_{\alpha_3|s|+\alpha_4|a|} \end{pmatrix}$$

where ϵ is a zero-mean random error, and the subscript defines its variance. Subscripts $\alpha_{\{1,2,3,4\}}$ are used to describe the errors affecting the translation and rotation of the robot generated from its movement.

The noisy control vector $\hat{u}_t$ is used to define the next position of the robot

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \begin{pmatrix} \hat{s}_t \cos\theta_t \\ \hat{s}_t \sin\theta_t \\ \frac{\hat{\alpha}_t}{l} \end{pmatrix} \tag{6.7}$$

where l is the distance of the wheel from the center pivot point of the robot. x and y are in mm while θ is in radians.

Perception Model

To sense the environment, our robot uses an infrared light range sensor. The sensor emits infrared light and measures the distance of the obstacle using the angle of the arrival of the reflected light. The response of the sensor with respect to the distance is not linear. To derive the transformation function, the robot was placed at a known distance from an obstacle and a large set of measurements were taken. Using the sample set, the real response of the sensor was approximated using a power function. The sensor can detect an obstacle in the range of 100mm up to 800mm.

Even though the measurements are reliable, they are also subject to noise and interference from external light sources. To test this, 10,000 measurement samples were taken with the sensor placed 400mm in front of an obstacle. The results show two kinds of errors. The first is the measurement error around the true distance. This is modeled using a narrow, zero-mean Gaussian. The results also show a random error occurring at various moments. This random error is modeled as a uniform distribution over the entire sensor measurement range. Including the random errors in the perception model is important as it injects some dynamic behavior to the algorithm. Even though we assume a static map, the modest dynamic elements of the model increase the tolerance of the algorithm to unexpected events.

Therefore, the perception model is made out of two probability distributions. p_{hit} represents the probability distribution of reading the correct distance with some added measurement noise and p_{rand} which represents the probability distribution of a random measurement error occurring. To create the two distributions, information from the map is required. The map used here, is the same one as in kinematic model. Using the raycasting method, the distance between the robot and the obstacle is measured. The measured distance is used to form the distributions, which are mixed together by weighted average

parameters

$$p(z_t|\mathbf{x}_t, m) = \begin{pmatrix} z_{hit} & z_{rand} \end{pmatrix} \begin{pmatrix} p_{hit}(z_t|x_t, m) \\ p_{rand}(z_t|x_t, m) \end{pmatrix} \tag{6.8}$$

where z_t is the measurement at time t, m is the map, and z_{hit}, z_{rand} are the weight parameters of each distribution. To identify these parameters we used the *expectation maximization* algorithm to calculate the maximum likelihood estimates (similar to [104]).

6.3.3 Localization Algorithm

Markov Localization

Our robotic platform localization algorithm is based on MCL algorithm which uses *Markov Localization*. The robot's pose is tracked with a belief state which is represented with an arbitrary probability density function. Markov Localization is an iterative procedure that uses the *Total Probability Theorem* to predict the belief and *Bayes rule* to update this prediction. Prediction is made using the current state (*Markov Assumption*) of the robot and the control vector u.

$$\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_{t-1}) bel(\mathbf{x}_{t-1}) d\mathbf{x}_{t-1}$$

where $\overline{bel}(\mathbf{x}_t)$ is the belief prediction for time t and $bel(\mathbf{x}_{t-1})$ is the belief update from the previous iteration. The prediction is updated using the sensor measurements

$$bel(\mathbf{x}_t) = p(z_t|\mathbf{x}_t)\overline{bel}(\mathbf{x}_t)$$

To simplify the computational complexity of representing the belief, MCL applies the *Sampling Importance Re-sampling* method using *Particle Filters*. The particles represent a specific robot pose hypothesis which is altered according to the kinematic model (6.7) and their importance weight is derived from the perception model (6.8). Based on that, the belief is then re-sampled and the filter converges to the (one or more) possible poses of the robot. The re-sampling step is particularly important for the performance of the localization. The accuracy can be improved by increasing the number of the particles by trading off computational power i.e. more particles give better filter accuracy but require more computational power. Robots that carry fast processors are able to process several thousand particles in real time. However, since we are investigating the implementation of this algorithm on low computational-power robots, we aim to operate the filter with less than 500 particles.

Limited Information Localization

As already mentioned, the robot uses a single range sensor. The sensor returns information for one dimension only, even though the robot is moving into a two-dimensional environment. Due to this fact, it becomes increasingly difficult to avoid localization divergence. One option is to employ control methods in order to maximize the information gathered from the environment with the purpose of achieving better estimation [101]. Another option is that, in the event of divergence, the estimation distribution can be broadened in an effort to re-approximate the true pose. Divergence can be detected by examining the history of the localization performance of the robot. As suggested in [104], localization performance can be approximated by the following probability of sensor measurements $p(.)$ which in particle filters is approximated by the mean value of the importance weight

$$p(z_t|z_{1:t-1}, u_{1:t}) = w_{avg} \approx \frac{1}{M}\sum_{m=1}^{M} w_t^{[m]} \tag{6.9}$$

where M is the total number of particles and $w_t^{[m]}$ is the importance weight of particle m at time t. Using two different smoothing factors $a_{fast} \gg a_{slow} > 0$, the short ($w_{fast}$) and long term ($w_{slow}$) localization performance is tracked.

$$w_{fast} = w_{fast} + a_{fast} w_{avg}$$

$$w_{slow} = w_{slow} + a_{slow} w_{avg}$$

The two values describe the current situation of the localization. Others, for example [104], use this factors (high w_{slow} value and lower w_{fast}) to detect reallocation of the robot by an external factor (the *kidnapped robot problem*). In such a case, random particles are injected to the set in an effort to approximate the new and unknown pose.

In our case however having $w_{slow} > w_{fast}$ occurs often and *does not* denote reallocation of the robot and therefore cannot be dealt with random particles injection. The limited perception information combined with the small number of particles is the reason this happens. While the robot tries to localize it will sometimes end up with a complex belief distribution that cannot be represented with the small number of particles being used. In other words, sometimes there are many possible poses for the robot and the limited sensing fails to reduce them. From our observations, this is likely to happen when the robot is traveling in a straight corridor, where not obstacles exist. During this time, the long term confidence of localization performance (w_{slow}) would rise but in reality the

robot could be positioned nearby. As soon as it reaches an obstacle and takes a measurement, the short term confidence will drop much faster than the long term and therefore reaching the $w_{slow} > w_{fast}$ situation.

As soon as the localization performance drops, the robot will initiate the new particle injection. The new particles are drawn from a Gaussian probability distribution function, whose variance is inverse proportional to w_{slow}

$$\mathbf{x}^{new} \sim \mathcal{N}(\mathbf{x}_t, \begin{pmatrix} \frac{\beta_1}{w_{slow}} & \frac{\beta_2}{w_{slow}} & \frac{\beta_3}{w_{slow}} \end{pmatrix}^\top)$$

where β_1, β_2, β_3 are the constants set a-priory. We used $\beta_1 = 400$, $\beta_2 = 400$, $\beta_3 = 10$ for our experiments. The intuition of parameterizing the variance of the distribution, is to adjust the distribution according to the localization performance. As explained, it is likely that the true pose is close by, therefore the first newly created particles should be placed near. If the localization continues to fail, it will cause the w_{slow} to keep falling and therefore push the next particles farther away. Injecting the particles this way, also increases the tolerance of the algorithm in unexpected events. Using this method, the robot is able recover from a temporary interference using the method described as shown later in the results.

6.3.4 Experimental Results and Discussion

The robot's environment covers an area of 2700mm×1700mm with obstacles forming the layout. The grid cell size is 100mm×100mm. The robot is tracked using a camera through a special marker placed on top of the chassis. This system provides the ground truth data required for performance evaluation of the localization method under testing. It is important to note that the camera data is used for post-processing the results and performance evaluation. Data from the camera is never transmitted to the robot.

Next, a series of two experiments is used to evaluate the performance of the described localization method. The experiments were conducted using the experimentation platform presented in this thesis.

Point to point Navigation

During this experiment, the robot starts its course from point A. It is then driven to point B and finally to point C. The initial position at point A is not known exactly, therefore the

robot is placed at $\mathbf{x}^{initial}$ given by

$$\mathbf{x}^{initial} \sim \mathcal{N}(\mathbf{x}^{A}, \begin{pmatrix} 100 & 100 & 0.1 \end{pmatrix}^{\top})$$

where $\mathbf{x}^{A}$ is the pose of the robot at when placed at point A with $\theta = 0$.

The initial uncertainty causes the particles to scatter in the state space nearby point A. Because of the low number of particles used in our approach, the initial particle scattering could prolong the divergence of the filter if not treated early on. One way to accelerate the estimation of the initial approximation, is to greatly increase the number of particles thus the accuracy of the filter, before reducing it again to normal levels. An implementation of this is shown in [3] and [17] where several thousands particles were used to successfully approximate the initial pose. This however, is not a suitable solution in our approach, since the number of particles has to be kept at minimum levels. Alternatively, the robot is programmed to pivot two complete circles before moving forward in an effort to use the sensor readings from the obstacles to improve the pose approximation.

To observe the effect of particle number on the localization performance, the same route was repeated, each time with different number of particles. Figure 6.7 shows the results using 200 particles. The blue line is the true path the robot followed, while the red line is the route given by the localization algorithm. The bottom graph shows the error which is the distance between the true and estimated pose. Figure 6.8 show the results using 400 particles.

Navigation with interference

Even though we are assuming a static environment we are interested to observe the tolerance of our localization method to a short random interference. We introduce this interference to the sensor measurements by altering the values. In section 6.3.2 we described that the perception model covers a small dynamic behavior of the environment. Furthermore, we improved the dynamic tolerance through adjustment of a_{fast} and a_{slow} as described in section 6.3.3.

Figures 6.9 and 6.10 show the robot driven at the same route as before. At the point D, the intentional interference is introduced. In the first case (figure 6.9) this interference is short, lasting about 2 seconds. Such interference can occur when a human walks in front of the robot unexpectedly. This gives an idea how the robot's perception of its pose is affected when it is operating in a static environment, like an office where the furniture

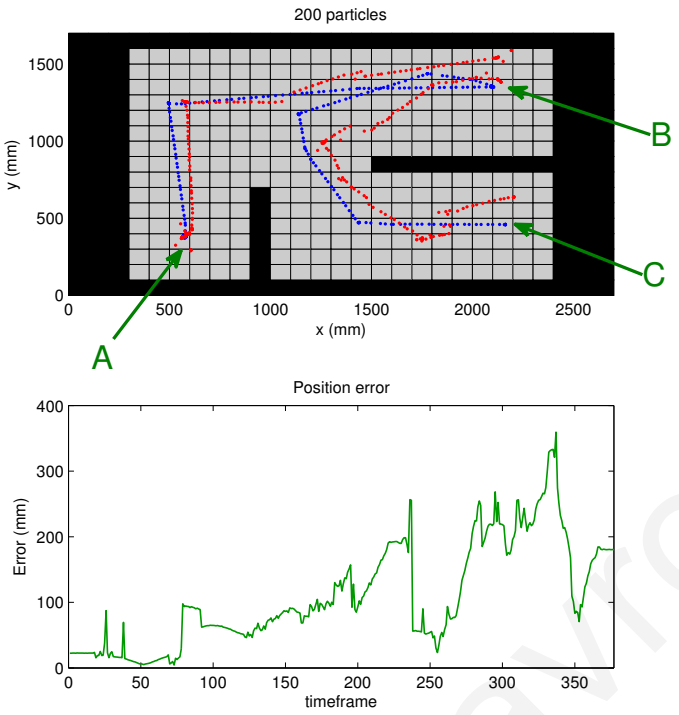


Figure 6.7: Localization results using 200 particles

layout remains the same, including however, some dynamic elements like the humans working there and walking around.

In the second case shown in figure 6.10, the interference is introduced at the same point but it lasts much longer. The purpose of this experiment is to observe the localization performance when the robot's sensor interference is more persistent. In such scenario, its likely that the pose distribution will diverge. As a result, one or more estimated poses exist which do not correspond to the true pose. Therefore, the robot was intentionally forced to lose track of its position, in an effort to observe the recovery capabilities of our algorithm and whether is able to return to its original pose estimation.

Discussion

Comparison of the estimated and true trajectories, shows that localization is accurate within 15cm in most cases. Results show that our robot works well with as low as 400 particles. The level of accuracy required is, of course, application dependent.

Equally important to the good performance, the results also reveal the weaknesses of the method. The fact that a conservative sensory equipment is used, makes the robot prone to localization errors. One major issue is the lack of feedback measurements. This

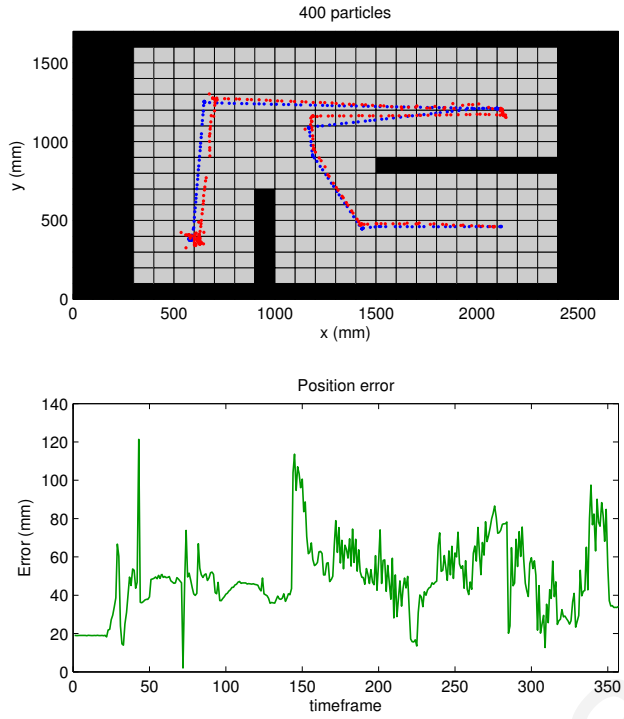


Figure 6.8: Localization results using 400 particles

happens when the robot travels along a path, that does not contain any objects within the range of the sensor. Even though a measurement indicating the absence of an obstacle **is** useful information, sometimes it is not enough to decrease the pose uncertainty to desired levels. To describe the same event in terms of MCL terminology, the weight distribution of particles is uniform therefore the variance of the resulting pose distribution is not decreased. An even worse scenario is when the particles' uniform weight distribution causes the pose distribution variance to increase even more and cause filter divergence.

The described scenario can be witnessed from the exhibited results. The left part (when the robot leaves the charging station area and travels upwards) of figure 6.7, shows the true and estimated trajectories being side by side. This was caused by a small heading error at point A. While the robot is traveling upwards, the top wall is too far to be seen by the sensor and therefore the reading is *absence of obstacles*. Even though this information is correct, is not enough to increase the weight of the correct particles and therefore shift the estimated pose closer to the true pose. However, we have shown that despite this issue, the robot is able correct the error and converge on the true pose later on.

Another important outcome, is the level of robustness of the algorithm. Even though theoretically the robot should operate in a totally static environment, figures 6.9 and 6.10

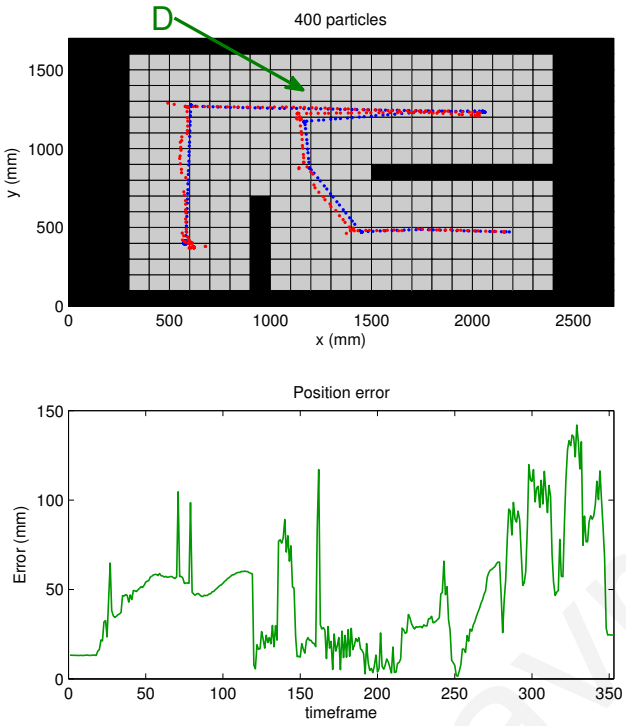


Figure 6.9: Short Interference

show that the algorithm can sustain some dynamic interference. Results show that the short-term interference did not cause any major divergence to the filter. This did not come as a surprise, as the robot was designed to dampen the significance of sensor measurements have on particles' weights. This dampening was exceeded in the long-term interference experiment. The robot was intentionally forced to converge to a false pose, in an effort to study its ability to re-approximate the correct pose. As shown in figure 6.10 the interference causes the robot to shift the approximation to a pose located around 280mm away from the true pose. Despite this, the robot does not diverge any further neither jumps to a completely irrelevant pose, something that could have happened if global localization was initiated. Instead it gradually minimizes the error and manages to re-approximate the correct pose. It is important to note that the true pose information from the camera was **completely unknown** by the robot in all experiments.

6.4 Concluding Remarks

In this chapter we presented two methods that can be applied after a fault is detected, in order to allow a robot to remain operational. In the first part we presented the *Fault*

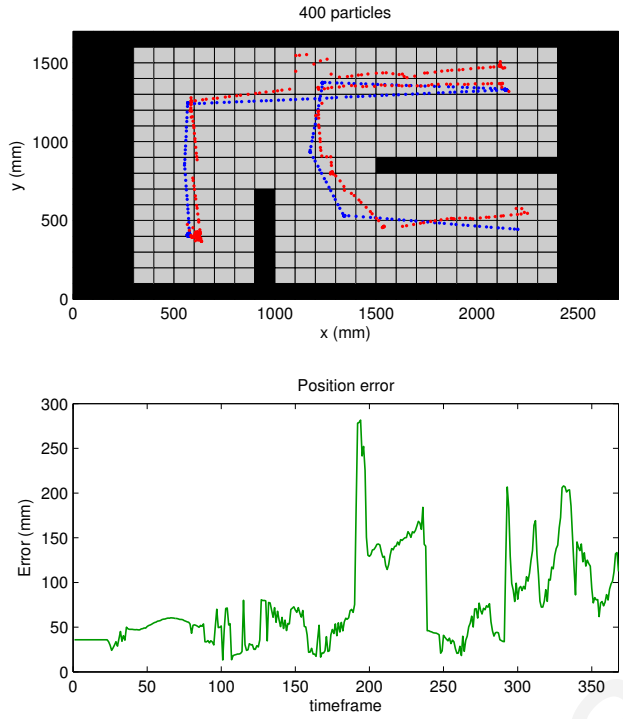


Figure 6.10: Long Interference

Detection, Isolation and Path Correction Module for a differential-drive mobile service robots. The results demonstrate that, in the event of an actuator fault, a robot equipped with the FDI-PC module is able to continue its operation and even reach its goals before being physically repaired. The module can be an external device that connects to the existing robot hardware and does not require any reprogramming of the robot controller. The simulation results show that the isolation success is lower when the fault magnitude is small, since the error introduced by the fault is hidden in the uncertainty error. In addition, analysis of the simulation results has shown that the symmetry of the left and right-wheel fault structure reduces the correct fault isolation rate. Information from isolation and identification is used to achieve path correction. Based on the controller signals, the module is able to generate new actuator signals that allow the controller to remain in control of the robot. This was demonstrated in simulation where a mobile robot equipped with the FDI-PC module, suffered a fault and it was able to reach its destination. This work was published and presented in a peer-reviewed conference [97].

In the second part, we present a Monte Carlo Localization approach that uses a single infra-red range sensor. We considered the case when due to failure only one range sensor remains healthy. In order to react to this event, we assumed that faults were detected and

all the faulty sensors were disabled. Then we switch to the presented localization approach which is appropriate for a single sensor. The results show that the robot can maintain an estimate of its pose within 15cm on average using 400 particles. The estimation is maintained even with short interferences on the sensor. Longer interferences cause the estimation to deviate but after some iterations, the estimation converges back to the true pose. The work was published and presented in a peer-reviewed conference [98].

Chapter 7

Conclusions

The vision for service robots is to improve human quality of life by replacing humans in dangerous, difficult and repetitive tasks. Early generations of service robots have proved that humans and robots can coexist in the same environment. Technological advancements are pushing the boundaries of hardware and software leading to bigger, faster and more complex robots that are more capable than ever. This raises challenges that need to be addressed before the seamless integration of service robots into diverse environments.

With regards to the challenge of coordination, we have presented a method that allows a group of robots to transport a set of containers from the storage to the loading area. Coordination is essential in this application to prevent any deadlocks and collisions between the robots in order to minimize the cost and associated risks. Related methods in the literature are not applicable for the specific topology of this problem because of limited space, lack of alternative paths, and unconventional structure of the robots. We formulated the problem as such to reflect the topology of the warehouse and the problem was solved optimally. Since service robots are expected to operate in dynamic environments, an unexpected event may occur that will require re-computation of the solution. Therefore, to ensure the reliability of the robot group, we developed a low-complexity algorithm that performs in real-time and provides near optimal solution.

With regards to the Fault Diagnosis challenge, we presented fault detection methods for actuator faults on differential-drive mobile robots. The first method uses odometry sensors to maintain an estimate of the speed, based on which faults are detected. We consider this method to be more general, because odometry sensors are a commonly installed sensor on mobile robots. By considering several uncertainty sources, the adaptive thresholds are used to detect faults occurring on the left and right wheels of the robot.

By tracking the control and sensor signals as well as the fault magnitude, the algorithm classifies the fault between three different fault types. This method was tested on a real service robot, using the experimentation platform we developed for the specific robot.

Sometimes, faults of small magnitude are not detected because they are indistinguishable from the uncertainty. A new detection scheme was introduced, and addresses this issue by increasing detection sensitivity using multiple thresholds. It differentiates from other residual-based methods because it combines the estimation error magnitude with the temporal behavior of a warning signal for detecting faults. It yields improved results when compared to a conservative worst-case threshold method.

When the robot becomes aware of the occurrence of a fault, it can react in order to mitigate the effects of the fault and continue its operation if possible. We developed the 'Fault Detection, Isolation and Path Correction Module', which is able to detect actuator faults using an external camera. After detection, the module isolates and identifies the magnitude of the fault. Based on this information and the controller signals, the module generates new actuator signals that allow the robot to maintain its path. We also considered the scenario in which a robot detects and disables faulty sensors, in an effort to maintain localization. Under this scenario, the robot is left with only one healthy sensor and switches to a localization algorithm suitable for this case. We presented a Monte Carlo Localization based approach, that works with a single infra-red range sensor. Experimental results on a real system showed that the robot is able to maintain a good pose estimate given enough particles, even under short sensor interferences.

An experimentation platform was designed and implemented for validating algorithms on a real robot. Accompanied software was also developed for controlling of the robot's actuators and reading the sensors. The platform offers direct, remote or hybrid control; in direct control the algorithms run directly on the hardware, while in remote control the robot acts as hardware in the loop, with the main algorithm running on the workstation.

7.1 Remarks for Applications

The methods presented in this thesis could be applied on real robots assisting towards the realization of the service robot vision. In the context of coordination, the method presented could act as mid-level planning tool, in a robot-equipped warehouse that uses the relevant topology. The operator of the warehouse would request a set of specific con-

tainers that need to be transported on the loading station. At this point, the algorithm will use the current positions of the robots to assign containers and issue a coordination plan, which will be sent to the robots. This requires a navigation controller that will be able to transform the plan into actuator movement. The plan defines the location and time the robot needs to be; the navigation controller should be able to move the robot to that location on time, of course within some margins. There should also be a monitoring algorithm that tracks the location of every robot. If the state of the system deviates from the plan, then the coordination algorithm should issue and transmit an updated plan.

With respect to actuator fault detection, the presented algorithms could serve as a starting point towards providing fault tolerant service robots. First, the robot manufacturer should have a good model for the behavior of the robot. For the methods that require uncertainty bounds, the manufacturer should identify possible sources of uncertainty and their bounds, and define their values in the algorithm. For the methods that require training, the manufacturer should include the necessary interface to allow the user to initiate the training procedure. To ensure long-term operation of the algorithm, changes in the robot dynamics should be taken into account because with time, robot components wear out. **We identify this as the next important challenge** that is mostly overlooked in literature and needs to be addressed for realizing fault detection on mainstream service robots. One approach to address this with the presented methods, is through training, even though it would require running the training procedure often to ensure that any changes in the dynamics are included in the model.

7.2 Future Work

With respect to the container loading problem, the heuristic approach was shown to perform optimally 40% of the time in random initial conditions. By studying the initial conditions of the problem scenarios that led to the optimal solution, it could shed light on container arrangement patterns under which the heuristic algorithm performs best. Identification of such patterns may ultimately improve the production of the facility, since these patterns could be exploited in order to arrange the containers in the first place. One possible implementation of this, is an algorithm that indicates the storage location of the containers during the initial arrangement of containers, with respect to their loading priority; containers with higher priorities would need to be loaded first, while lower priority containers could be loaded last. Such algorithm could lead to a facility that steadily op-

erates optimally through the sole use of the the heuristic algorithm.

Part of our work was dedicated to provide robots with fault detection capabilities in realistic conditions with respect to uncertainties. To accomplish this we used sensor feedback to measure the robot state, and we assumed that those measurements were true (with added uncertainty of course). It is possible however, that sensors could also experience faults and this critically affects the fault detection performance with increased false alarms and/or missed alarms. To successfully detect sensor faults, it requires to have redundancy between the measurements. However, redundant information is not always available. In this case, it would be interesting to generate information redundancy through the use of additional robots. Should a robot suffer a failure, it can send a distress signal to other robots and they could provide their measurements with respect to the robot under question and reach a consensus whether a failure has occurred or not. This collaborative method of fault detection would benefit future applications that involve multiple robots.

In order to detect a fault, the robot requires the knowledge of the nominal behavior to compare against. In the context of robotics, this nominal behavior is usually described by a model that was designed based on specifications or was estimated through learning or a combination of the two. In the majority of existing work, experiments are performed within a well controlled environment over a short period of time, while in reality a robot could potentially operate for years. Throughout its lifetime, robot's hardware degrades and therefore the nominal behavior deviates from the initial one. Due to this, at some point the robot will detect faults which are actually caused by normal degradation of its hardware. Even with adaptive models, it will be challenging to distinguish between a subtle fault and hardware degradation and in such case it would be possible to adapt to a faulty behavior rather than report it. To the best of our knowledge, such problem has not been addressed yet.

Appendix A

Experimentation Platform

A.1 Introduction

When possible, experimental validation is an important part of research because it allows the researcher to evaluate the performance of the developed methods in realistic conditions. A wide range of robotic research equipment is available and comes in many forms. For our work, we were particularly interested in investigating algorithms around service robots and for this reason, the iRobot Roomba was an ideal choice. Roomba is a low cost differential-drive robot equipped with floor cleaning hardware. It is able to move and rotate at place, it has an array of short-range distance sensors at the front in order to detect obstacles and avoid crashing on them. The developers provide a communication port which is located at the top of the robot which allows full control of the robot. The communication protocol is well documented and provides the necessary information to send and receive data to/from the robot.

There are several reasons why this platform was valuable for our work:

- **Experimental validation of the developed methods.** Running algorithms on a real robot, gives a deep insight on the challenges introduced by real life conditions providing valuable feedback. An algorithm working correctly on the experimental robot is likely to be suitable for use on a commercial robot.

- **Hardware in the loop** This feature allows fast testing and debugging of algorithms on the experimental robot. Developing algorithms in high level languages supported by a workstation is commonly more fast and efficient that developing for hardware directly. Therefore, algorithms concerning the feedback and control of the robot are developed and run on a workstation but the control and sensor sig-

nals are sent and received from the experimental robot. Furthermore, the debugging features offered by the software environment make it efficient to monitor the whole operation.

- **Telemetry** All data available from the communication protocol, can be retrieved and sent back to the workstation. This data can be processed and visualized, providing important information about the behavior of the robot and how well the testing algorithm performs. Furthermore, valuable datasets can be acquired.
- **Fault injection** Having the controller of the robot running on the workstation, allows the real-time software manipulation of signals. This is useful when we need to inject a fault in the system because the signals are altered based on fault model.
- **Hardware extension** This feature allow the extension of the robot capabilities by providing a way to interface additional hardware which becomes part of the robot. By implementing the required drivers, a wide range of sensors and actuators can be added to the robot.

A.2 Platform Description

The platform is composed of the hardware and software parts. A hardware module installed on the robot provides a remote processing unit with a wireless link to a workstation. The module was designed to use off-the-shelf electronic components of low cost, in order to make the platform easy to reproduce by other researchers. At the heart of the hardware module is a dsPIC30F4011 microcontroller by Microchip. The specific microcontoller offers up to 30 MIPs, a fast ADC module which is useful for interfacing analog sensors and a double serial module which is used for communications, among other features. The microcontroller is interfaced to the robot through the interface port which is a 7-pin mini-DIN socket, located on the top of the robot on the right side. The port provides unregulated 12V power as well as serial input and output. Communication is established using the serial module of the microcontroller. The platform communicates wirelessly with a workstation through the second serial port of the microcontroller. Wireless communication is established through a link between two XBee modules programmed to work in serial mode. In this mode, the XBee pair acts as a bridge between the computer and the microcontroller; they communicate using the standard serial protocol and the wireless part (handshake, packet acknowledgement etc.) are handled by the

XBee modules. The power provided by the robot is passed through a switching regulator to step it down 3.3V which is required for the microcontroller and the XBee module. Figures A.1 and A.2 show the platform installed on the robot. This configuration is used to provide two modes of operation:

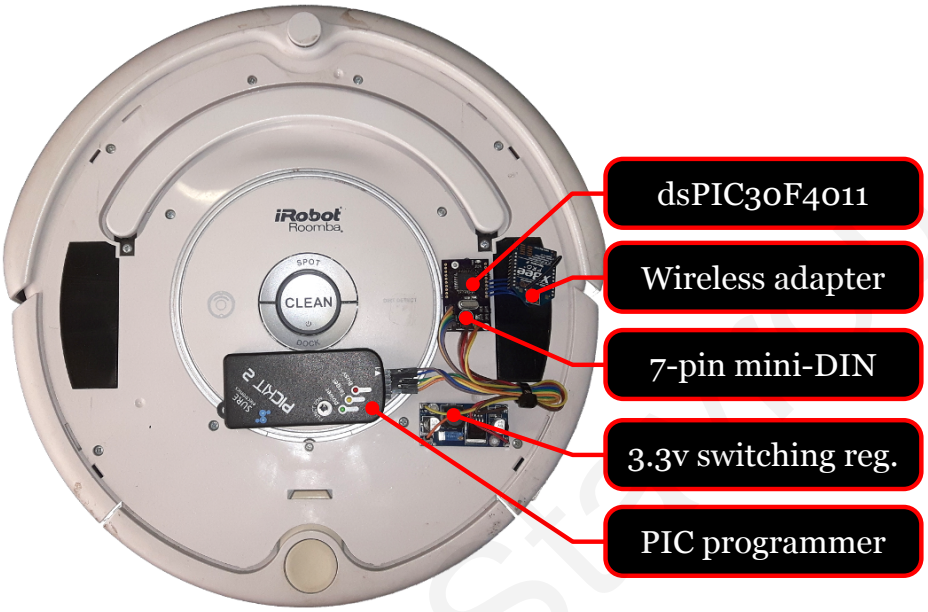


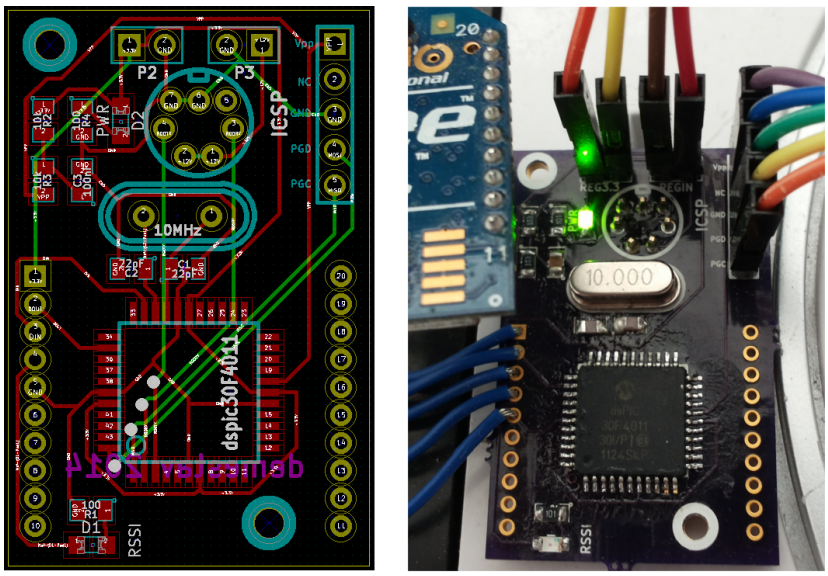
Figure A.1: The main components of the hardware module.

Direct Control: In this mode, the robot is fully controlled by the microcontroller. Necessary firmware is implemented in order to allow driving the actuators of the robot and read measurements off sensors. Important data can be transmitted back to the computer for monitoring and further processing.

Remote Control: The control algorithm is developed and executed on a workstation. Control signals are composed on the computer and sent to the platform wirelessly. After receiving the data, the microcontrller passes it unaltered to the robot, working as a repeater. Similarly, data coming from the robot sensors are transmitted from the robot to the microcontroller which sends them unaltered to the computer. In this mode, the robot just becomes an extension of the computer programming environment.

In some cases a **hybrid mode** is also useful, where different algorithms run on the computer and the microcontroller. More specific, we developed an abstraction layer on the microcontroller to handle tasks (for example, driving the motors) which can be called from the computer software. This is particularly useful when time-critical tasks (fast polling of a sensor input) is required.

In order to control and monitor the robot, it was necessary to develop libraries both



(a) (b)

Figure A.2: (a) The hardware board layout. (b) The populated board installed on the robot.

on the workstation and on the hardware module. The main scripting language used to produce the results presented in this Thesis was MATLAB by MathWorks. We developed a MATLAB library that provides full control of the robot within the MATLAB environment based on the 'iRobot Roomba Open Interface Specification' document provided by the manufacturers of the robot. The document defines the byte structure necessary to run specific instructions on the robot. Apart from the low-level commands, the library offers the ability to implement high-level commands that make more efficient the algorithm development. We should clarify that the software part of the platform is not necessary to exist within MATLAB. We have already ported some of the libraries to Python. In fact, a Python application with graphical interface was developed that assisted in collecting large datasets used in this work. Furthermore, the application is used to control and monitor the robot through graphs.

An overview of the platform's components and modes of operation are illustrated in Fig.A.3.

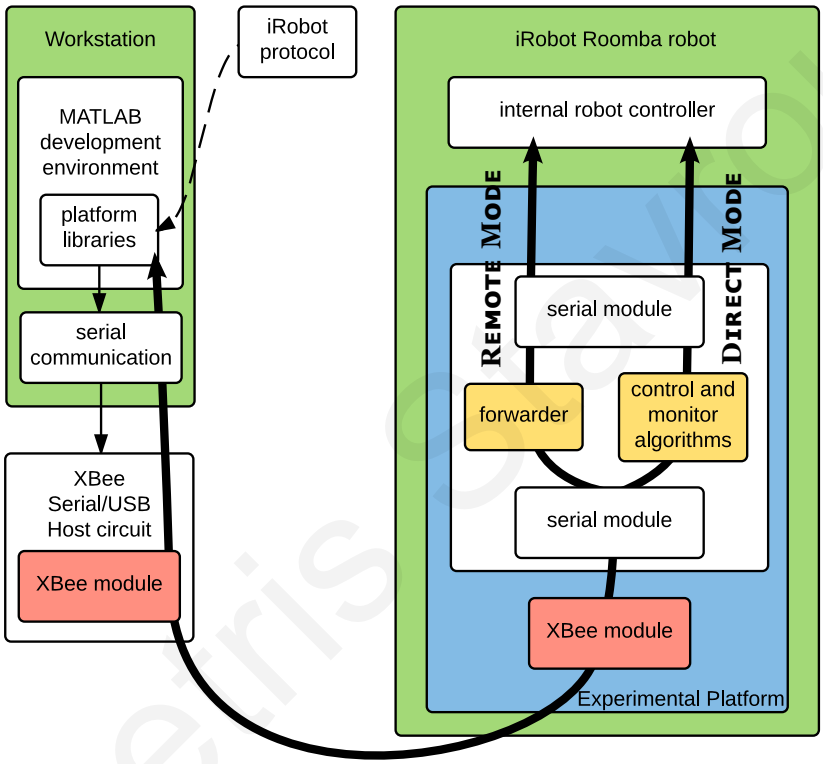


Figure A.3: High-level diagram of the platform. The robot can be controlled and monitored wirelessly using the developed MATLAB libraries within the development environment. In direct mode, algorithms can be loaded and executed directly on the microcontroller while in remote mode, the robot is fully controlled using the libraries, acting as hardware in the loop.

Appendix B

Mathematical Proofs

B.1 Proof of Lemma 6

For case (i), the best case scenario for Strategy 2 occurs when $t_k^e > t_i^x + T^g$ requiring $\mathbf{w}^e = [0,0]^\top$ and $\mathbf{w}^x = [0,0]^\top$ leading to the solution for Strategy 2, $\Lambda_2 = t_k^e + 2t_k^v$. For the same conditions, the worst case scenario for Strategy 1 occurs when $(t_i^e + T^g < t_k^e) \wedge (t_k^x + T^g > t_i^x)$ which requires $\mathbf{w}^e = [0,0]^\top$ and $\mathbf{w}^x = [t_k^x - t_i^x + T^g, 0]^\top$, giving the solution $\Lambda_1 = t_k^e + 2t_k^v + T^g$. Strategy 2 performs better by $\Lambda_1 - \Lambda_2 = T^g$.

For case (ii), Strategy 2 solution is non-conditional therefore it always requires $\mathbf{w}^e = [0, t_i^x - t_k^e + T^g]^\top$ and $\mathbf{w}^x = [0,0]^\top$, leading to the solution $\Lambda_2 = t_i^e + 2t_i^v + 2t_k^v + T^g$. Similarly to lemma 6, the worst case scenario for Strategy 1 in case (ii) gives the solution $\Lambda_1 = t_i^e + 2t_k^v + 2T^g$. Strategy 2 performs better by $\Lambda_1 - \Lambda_2 = T^g - 2t_i^v$.

B.2 Proof of Lemma 7

For case (i), Strategy 3 solution is non-conditional therefore it always requires $\mathbf{w}^e = [t_k^x - t_i^e + T^g, 0]^\top$ and $\mathbf{w}^x = [0,0]^\top$, leading to the solution $\Lambda_3 = t_k^e + 2t_k^v + 2t_i^v + T^g$. The worst case scenario for Strategy 1 occurs when $(t_i^e + T^g > t_k^e) \wedge (t_k^x + T^g > t_i^x)$ which requires $\mathbf{w}^e = [0, t_i^e - t_k^e + T^g]^\top$ and $\mathbf{w}^x = [t_k^x - t_i^x + T^g, 0]^\top$ giving the solution $\Lambda_1 = t_i^e + 2t_k^v + T^g$. Strategy 3 performs better by $\Lambda_1 - \Lambda_3 = t_i^e - t_k^e - 2t_i^v + T^g$. From the definition of case (i) we know that $t_k^e > t_i^e$ therefore $\Lambda_1 - \Lambda_3 < T^g - 2t_i^v$.

For case (ii), best case scenario for Strategy 3 occurs when $t_i^e > t_k^x + T^g$ requiring $\mathbf{w}^e = [0,0]^\top$ and $\mathbf{w}^x = [0,0]^\top$ leading to the solution $\Lambda_3 = t_i^e + 2t_i^v$. For the same conditions, worst case scenario for Strategy 1 occurs when $t_k^x + w_k^e + T^g > t_i^x$ which

requires $\mathbf{w}^e = [0, t_i^e - t_k^e + T^g]^\top$ and $\mathbf{w}^x = [2t_k^x - 2t_i^v + 2T^g]^\top$ giving the solution $\Lambda_1 = t_i^e + 2t_k^v + 2T^g$. Strategy 3 performs better by $\Lambda_1 - \Lambda_3 = 2t_k^v - 2t_i^v + 2T^g$. From the definition of case (ii) $t_i^v > t_k^v$ therefore $\Lambda_1 - \Lambda_3 < 2T^g$.

B.3 Proof of Lemma 8

For case (i), best case scenario for Strategy 1 occurs when $(t_i^e + T^g < t_k^e) \wedge (t_k^x + T^g < t_i^x)$ requiring $\mathbf{w}^e = [0, 0]^\top$ and $\mathbf{w}^x = [0, 0]^\top$ leading to the solution $\Lambda_1 = t_i^e + 2t_i^v$. For the same conditions, worst case scenario for Strategy 2 occurs when $t_k^e < t_i^x + T^g$ which requires $\mathbf{w}^e = [0, t_i^x - t_k^e + T^g]^\top$, giving the solution $\Lambda_2 = t_i^e + 2t_i^v + 2t_k^v + T^g$. Strategy 1 performs better by $\Lambda_2 - \Lambda_1 = 2t_k^v + T^g$.

For case (ii), best case scenario for Strategy 1 occurs when $t_k^x + w_k^e + T^g < t_i^x$ requiring $\mathbf{w}^e = [0, t_i^e - t_k^e + T^g]^\top$ and $\mathbf{w}^x = [0, 0]^\top$ leading to the solution $\Lambda_1 = t_i^e + 2t_i^v$. Also, from lemma 6 we know that the solution for Strategy 2 is non-conditional and is $\Lambda_2 = t_i^e + 2t_i^v + 2t_k^v + T^g$. Therefore, Strategy 1 solution is better by $\Lambda_2 - \Lambda_1 = 2t_k^v + T^g$.

B.4 Proof of Lemma 9

For case (i), we know from lemma 8 that best case scenario solution for Strategy 1 is $\Lambda_1 = t_i^e + 2t_i^v$. Also, from lemma 7 we know that the solution for Strategy 3 is non-conditional and is $\Lambda_3 = t_k^e + 2t_k^v + 2t_i^v + T^g$. Therefore, Strategy 1 performs better by $\Lambda_3 - \Lambda_1 = t_k^e - t_i^e + 2t_k^v + T^g$ which is always positive because of the condition $t_i^e < t_k^e$.

For case (ii) from lemma 8 we know that best case scenario solution for Strategy 1 is $\Lambda_1 = t_i^e + 2t_i^v$. For the same conditions, worst case scenario for Strategy 3 occurs when $t_i^e < t_k^x + T^g$ which requires $\mathbf{w}^e = [t_k^x - t_i^e + T^g, 0]^\top$ and $\mathbf{w}^x = [0, 0]^\top$ giving the solution $\Lambda_3 = t_k^e + 2t_k^v + 2t_i^v + T^g$. Strategy 1 performs better by $\Lambda_3 - \Lambda_1 = t_k^e - t_i^e + 2t_k^v + T^g$ which is a always positive because of the condition $t_i^e < t_k^x + T^g$.

B.5 Proof of Lemma 11

Let us define $\hat{t}_{i,m}^e = t_i^e + \sum_{\tau=1}^m w_{i,\tau}^e$ and $\hat{t}_{i,m}^x = \hat{t}_{i,m}^e + 2t_i^v + \sum_{\tau=1}^m w_{i,\tau}^x$, where $w_{i,\tau}^e$ and $w_{i,\tau}^x$ are the entrance and exit waiting time of ATW i in step τ of the algorithm, respectively. Note that the m-th external for-loop iteration of the Shift and Extend operations are the m-th and $(N + m)$-th step of the algorithm, respectively.

Algorithm 1 is comprised of the Shift and Extend operations. The Shift operation examines ATWs in descending t_i^v value order; assuming that ATW i is examined during the m-th iteration of the Shift operation, other ATWs belonging to the set $j \in \mathcal{C}_i$ are appropriately shifted ($w_{j,m}^e \geq 0$ and $w_{j,m}^x = 0$) to ensure that $\hat{t}_{i,m}^e \leq \hat{t}_{j,m}^e - T^g$, $j \in \mathcal{C}_i$ and $t_i^v > t_j^v$. This condition is ensured by appropriately shifting ATW j by

$$w_{j,m}^e = \max(0, \hat{t}_{i,m-1}^e - \hat{t}_{j,m-1}^e + T^g). \tag{B.1}$$

To show that the total cost increase for this ATW pair is bounded by T^g, it suffices to prove that

$$\hat{t}_{j,m}^x - \max(\hat{t}_{i,m-1}^x, \hat{t}_{j,m-1}^x) \leq T^g. \tag{B.2}$$

In the case that $\hat{t}_{i,m-1}^x > \hat{t}_{j,m-1}^x$, condition (B.2) can be written as $\hat{t}_{j,m}^e + 2t_j^v - \hat{t}_{i,m-1}^e - 2t_i^v \leq T^g$. After substituting $\hat{t}_{j,m}^e = \hat{t}_{j,m-1}^e + w_{j,m}^e$ the equation is $\hat{t}_{j,m-1}^e + 2t_j^v + \max(0, \hat{t}_{i,m-1}^e - \hat{t}_{j,m-1}^e + T^g) - \hat{t}_{i,m-1}^e - 2t_i^v \leq T^g$ which is always true because $t_j^v - t_i^v < 0$. In the case that $\hat{t}_{i,m-1}^x < \hat{t}_{j,m-1}^x$, condition (B.2) can be written as $\hat{t}_{j,m}^x - \hat{t}_{j,m-1}^x = w_{j,m}^e \leq T^g$. By substituting $\hat{t}_{\kappa,m-1}^e = \hat{t}_{\kappa,m-1}^x - 2t_\kappa^v$, $\kappa = \{i, j\}$, into (B.1) yields $\max\left(0, \hat{t}_{i,m-1}^x - \hat{t}_{j,m-1}^x + 2(t_j^v - t_i^v) + T^g\right) \leq T^g$, which is true because $\hat{t}_{i,m-1}^x - \hat{t}_{j,m-1}^x < 0$ and $2(t_j^v - t_i^v) < 0$.

The Extend operation examines ATWs in ascending t_i^v value order; assuming that ATW i is examined during the m-th iteration of the Extend operation, i.e. step $n = N+m$ of the algorithm, other ATWs belonging to the set $j \in \mathcal{C}_i$ are appropriately extended ($w_{j,n}^x > 0$ and $w_{j,n}^e = 0$) to ensure that $\hat{t}_{i,n}^x \geq \hat{t}_{j,n}^x + T^g$, $j \in \mathcal{C}_i$ and $t_i^v > t_j^v$. This condition is ensured by appropriately shifting ATW j by

$$w_{i,n}^x = \max(0, \hat{t}_{i,n-1}^x - \hat{t}_{j,n-1}^x + T^g). \tag{B.3}$$

To show that the total cost increase for this ATW pair is bounded by T^g, it suffices to prove that

$$\hat{t}_{i,n}^x - \max(\hat{t}_{i,n-1}^x, \hat{t}_{j,n-1}^x) \leq T^g. \tag{B.4}$$

In the case that $\hat{t}_{i,n-1}^x > \hat{t}_{j,n-1}^x$, condition (B.4) can be written as $\hat{t}_{i,n}^x - \hat{t}_{i,n-1}^x = w_{i,n}^x \leq T^g$; this is equivalent to $\max(0, \hat{t}_{j,n-1}^x - \hat{t}_{i,n-1}^x + T^g) \leq T^g$ which is true because $\hat{t}_{j,n-1}^x - \hat{t}_{i,n-1}^x < 0$. In the case that $\hat{t}_{i,n-1}^x < \hat{t}_{j,n-1}^x$, condition (B.4) can be written as $\hat{t}_{i,n-1}^x - \hat{t}_{j,n}^x + \max(0, \hat{t}_{j,n-1}^x - \hat{t}_{i,n-1}^x + T^g) \leq T^g$ which is true because $\hat{t}_{i,n-1}^x < \hat{t}_{j,n}^x = \hat{t}_{j,n-1}^x$.

The algorithm needs 2N steps in order to complete. Therefore, we proved that $f_C^h \leq 2NT^g$.

Bibliography

[1] Åström, Karl J and Wittenmark, Björn, *Adaptive control*. MA, USA: Addison-Wesley, 1994.

[2] F. Abrate, B. Bona, and M. Indri, "Experimental EKF-based SLAM for mini-rovers with IR sensors only," in *Proc. European Conference on Mobile Robots*, 2007.

[3] ——, "Monte Carlo Localization of mini-rovers with low-cost IR sensors," in *Proc. International Conference on Advanced Intelligent Mechatronics*, 2007, pp. 1–6.

[4] R. Anderson and D. M. Bevly, "Estimation of slip angles using a model based estimator and GPS," in *Proc. American Control Conference*, vol. 3, 2004, pp. 2122–2127.

[5] A. Angelova, L. Matthies, D. Helmick, and P. Perona, "Learning and prediction of slip from visual information," *Journal of Field Robotics*, vol. 24, no. 3, pp. 205–232, 2007.

[6] N. Ayanian and V. Kumar, "Decentralized feedback controllers for multiagent teams in environments with obstacles," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 878–887, 2010.

[7] A. Becker and T. Bretl, "Approximate steering of a unicycle under bounded model perturbation using ensemble control," *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 580–591, 2012.

[8] K. R. Beevers and W. H. Huang, "SLAM with sparse sensing," in *Proc. IEEE International Conference on Robotics and Automation*, 2006, pp. 2285–2290.

[9] L. Blackmore, M. Ono, A. Bektassov, and B. C. Williams, "A probabilistic particle-control approximation of chance-constrained stochastic predictive control," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 502–517, 2010.

[10] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault-Tolerant Control*, 2nd ed. Berlin, Germany ; New York, USA: Springer, 2006.

[11] J. Borenstein and L. Feng, "Gyrodometry: A new method for combining data from gyros and odometry in mobile robots," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 423–428.

[12] Y. A. Bozer and M. M. Srinivasan, "Tandem configurations for automated guided vehicle systems and the analysis of single vehicle loops," *IIE Transactions*, vol. 23, no. 1, pp. 72–82, 1991.

[13] D. Brscić, H. Kidokoro, Y. Suehiro, and T. Kanda, "Escaping from children's abuse of social robots," in *Proc. ACM/IEEE International Conference on Human-Robot Interaction*, 2015, pp. 59–66.

[14] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009.

[15] H. J. Carlo, I. F. Vis, and K. J. Roodbergen, "Transport operations in container terminals: Literature overview, trends, research directions and classification scheme," *European Journal of Operational Research*, vol. 236, no. 1, pp. 1–13, 2014.

[16] J. Carlson and R. R. Murphy, "Reliability analysis of mobile robots," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 1, 2003, pp. 274–281.

[17] G. Cen, N. Matsuhira, J. Hirokawa, H. Ogawa, and I. Hagiwara, "Mobile robot global localization using particle filters," in *Proc. International Conference on Control, Automation and Systems*, 2008, pp. 710–713.

[18] G. Cen, H. Nakamoto, N. Matsuhira, and I. Hagiwara, "Effective application of monte carlo localization for service robot," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 1914–1919.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, p. 15, 2009.

[20] H. Chen, M.-M. Ma, H. Wang, Z.-Y. Liu, and Z.-X. Cai, "Moving horizon tracking control of wheeled mobile robots with actuator saturation," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 2, pp. 449–457, 2009.

[21] J. Chen and R. J. Patton, *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Boston, USA: Kluwer Academic Publishers, 1999.

[22] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Fault detection in autonomous robots based on fault injection and learning," *Autonomous Robots*, vol. 24, no. 1, pp. 49–67, 2008.

[23] S. C. Daniels, "Real time conflict resolution in automated guided vehicle scheduling," Ph.D. dissertation, Dept. of Industrial Eng., Penn. State University, PA, USA, 1988.

[24] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[25] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 2, 1999, pp. 1322–1328.

[26] J. P. Desai, J. P. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 905–908, 2001.

[27] V. Digani, L. Sabattini, C. Secchi, and C. Fantuzzi, "Ensemble coordination approach in multi-agv systems applied to industrial warehouses," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 922–934, 2015.

[28] W. Dixon, I. Walker, and D. Dawson, "Fault detection for wheeled mobile robots with parametric uncertainty," in *Proc. IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, vol. 2, 2001, pp. 1245–1250.

[29] H. Durrant-Whyte, D. Pagac, B. Rogers, M. Stevens, and G. Nelmes, "Field and service applications-an autonomous straddle carrier for movement of shipping containers-from research to operational autonomous systems," *IEEE Robotics and Automation Magazine*, vol. 14, no. 3, pp. 14–23, 2007.

[30] P. J. Egbelu and J. M. Tanchoco, "Potentials for bi-directional guide-path for automated guided vehicle based systems," *International Journal of Production Research*, vol. 24, no. 5, pp. 1075–1097, 1986.

[31] L. H. Erickson, J. Knuth, J. M. Kane, and S. M. LaValle, "Probabilistic localization with a blind robot," in *Proc. IEEE International Conference on Robotics and Automation*, 2008, pp. 1821–1827.

[32] I. Eski, S. Erkaya, S. Savas, and S. Yildirim, "Fault detection on robot manipulators using artificial neural networks," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 115–123, 2011.

[33] R. Z. Farahani, G. Laporte, E. Miandoabchi, and S. Bina, "Designing efficient methods for the tandem AGV network design problem using tabu search and genetic algorithm," *International Journal of Advanced Manufacturing Technology*, vol. 36, no. 9-10, pp. 996–1009, 2008.

[34] J. A. Farrell and M. M. Polycarpou, *Adaptive approximation based control: unifying neural, fuzzy and traditional adaptive approximation approaches*. New Jersey, USA: John Wiley & Sons, 2006.

[35] H. Fazlollahtabar and M. Saidi-Mehrabad, "Methodologies to optimize automated guided vehicle scheduling and routing problems: a review study," *Journal of Intelligent and Robotic Systems*, vol. 77, no. 3-4, pp. 525–545, 2015.

[36] R. M. Ferrari, T. Parisini, and M. M. Polycarpou, "Distributed fault detection and isolation of large-scale discrete-time nonlinear systems: An adaptive approximation approach," *IEEE Transactions on Automatic Control*, vol. 57, no. 2, pp. 275–290, 2012.

[37] R. Ferrari, T. Parisini, and M. Polycarpou, "A fault detection and isolation scheme for nonlinear uncertain discrete-time sytems," in *Proc. IEEE Conference on Decision and Control*, 2007, pp. 1009 –1014.

[38] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: a study of the roomba vacuum in the home," in *Proc. ACM SIGCHI/SIGART Conference on Human-robot interaction*, 2006, pp. 258–265.

[39] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *Proc. AAAI National Conference on Artificial Intelligence*, vol. 1999, 1999, pp. 343–349.

[40] D. R. Fulkerson, I. Glicksberg, and O. Gross, *A production line assignment problem*. Sta. Monica, CA: The Rand Corporation, 1953.

[41] R. Gaskins and J. M. Tanchoco, "Flow path design for automated guided vehicle systems," *International Journal of Production Research*, vol. 25, no. 5, pp. 667–676, 1987.

[42] P. Goel, G. Dedeoglu, S. Roumeliotis, G. S. Sukhatme *et al.*, "Fault detection and identification in a mobile robot using multiple model estimation and neural network," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 3, 2000, pp. 2302–2309.

[43] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE Spectrum*, vol. 45, no. 7, pp. 26–34, 2008.

[44] F. Gustafsson, "Slip-based tire-road friction estimation," *Automatica*, vol. 33, no. 6, pp. 1087–1099, 1997.

[45] P. Harmo, T. Taipalus, J. Knuuttila, J. Vallet, and A. Halme, "Needs and solutions-home automation and service robots for the elderly and disabled," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 3201–3206.

[46] I. Hwang, S. Kim, Y. Kim, and C. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 3, pp. 636–653, May 2010.

[47] K. Iagnemma and C. C. Ward, "Classification-based wheel slip detection and detector fusion for mobile robots on outdoor terrain," *Autonomous Robots*, vol. 26, no. 1, pp. 33–46, 2009.

[48] G. Indiveri, A. Nüchter, and K. Lingemann, "High speed differential drive mobile robot path following control with bounded wheel speed commands," in *Proc. IEEE International Conference on Robotics and Automation*, 2007, pp. 2202–2207.

[49] R. Isermann, "Supervision, fault-detection and fault-diagnosis methods—an introduction," *Control Engineering Practice*, vol. 5, no. 5, pp. 639–652, May 1997.

[50] ——, *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer, 2006.

[51] E. Ivanjko and I. Petrović, "Extended kalman filter based mobile robot pose tracking using occupancy grid maps," in *Proc. IEEE Mediterranean Electrotechnical Conference*, 2004.

[52] A. Jadbabaie, J. Lin *et al.*, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003.

[53] G. Jager, S. Zug, T. Brade, A. Dietrich, C. Steup, C. Moewes, and A.-M. Cretu, "Assessing neural networks for sensor fault detection," in *International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications*, 2014, pp. 70–75.

[54] C. Jayawardena, I. H. Kuo, U. Unger, A. Igic, R. Wong, C. I. Watson, R. Stafford, E. Broadbent, P. Tiwari, J. Warren *et al.*, "Deployment of a service robot to help older people," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 5990–5995.

[55] P. Jensfelt and S. Kristensen, "Active global localization for a mobile robot using multiple hypothesis tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, 2001.

[56] K. Kawamura, R. T. Pack, M. Bishay, and M. Iskarous, "Design philosophy for service robots," *Robotics and Autonomous Systems*, vol. 18, no. 1, pp. 109–116, 1996.

[57] E. Khalastchi, M. Kalech, G. A. Kaminka, and R. Lin, "Online data-driven anomaly detection in autonomous robots," *Knowledge and Information Systems*, vol. 43, no. 3, pp. 657–688, 2015.

[58] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[59] C.-O. Kim and S. Kim, "An efficient real-time deadlock-free control algorithm for automated manufacturing systems," *International Journal of Production Research*, vol. 35, no. 6, pp. 1545–1560, 1997.

[60] C. W. Kim and J. M. Tanchoco, "Conflict-free shortest-time bidirectional AGV routeing," *International Journal of Production Research*, vol. 29, no. 12, pp. 2377–2391, 1991.

[61] ——, "Operational control of a bidirectional automated guided vehicle system," *International Journal of Production Research*, vol. 31, no. 9, pp. 2123–2138, 1993.

[62] K. S. Kim and B. Do Chung, "Design for a tandem AGV system with two-load AGVs," *Computers and Industrial Engineering*, vol. 53, no. 2, pp. 247–251, 2007.

[63] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proc. International Conference on Autonomous Agents*, 1997, pp. 340–347.

[64] G. Laporte, "Fifty years of vehicle routing," *Transportation Science*, vol. 43, no. 4, pp. 408–416, 2009.

[65] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[66] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[67] C.-C. Lee and J. T. Lin, "Deadlock prediction and avoidance based on petri nets for zone-control automated guided vehicle systems," *International Journal of Production Research*, vol. 33, no. 12, pp. 3249–3265, 1995.

[68] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.

[69] J. S. Lewis and J. M. O'Kane, "Guaranteed navigation with an unreliable blind robot," in *Proc. IEEE International Conference on Robotics and Automation*, 2010, pp. 5519–5524.

[70] R. Lin, E. Khalastchi, G. Kaminka *et al.*, "Detecting anomalies in unmanned vehicles using the mahalanobis distance," in *Proc. IEEE International Conference on Robotics and Automation*, 2010, pp. 3038–3044.

[71] M. Luo, D. Wang, M. Pham, C. Low, J. Zhang, D. Zhang, and Y. Zhao, "Model-based fault diagnosis/prognosis for wheeled mobile robots: a review," in *Proc. IEEE Conference on Industrial Electronics Society*, 2005, pp. 6–pp.

[72] L. Magatao, "Mixed integer linear programming and constraint logic programming: towards a unified modeling framework," Ph.D. dissertation, University of Curitiba, May 2005.

[73] H. Martínez-Barberá and D. Herrero-Pérez, "Autonomous navigation of an automated guided vehicle in industrial environments," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 4, pp. 296–311, 2010.

[74] R. H. Möhring, E. Köhler, E. Gawrilow, and B. Stenzel, "Conflict-free real-time AGV routing," in *Proc. of Operations Research*, 2005, pp. 18–24.

[75] R. L. Moorthy, W. Hock-Guan, N. Wing-Cheong, and T. Chung-Piaw, "Cyclic deadlock prediction and avoidance for zone-controlled AGV system," *International Journal of Production Economics*, vol. 83, no. 3, pp. 309–324, 2003.

[76] Y. Morales, E. Takeuchi, and T. Tsubouchi, "Vehicle localization in outdoor woodland environments with sensor fault detection," in *Proc. IEEE International Conference on Robotics and Automation*, 2008, pp. 449–454.

[77] P. Morin and C. Samson, "Control of nonholonomic mobile robots based on the transverse function approach," *IEEE Transactions on Robotics*, vol. 25, no. 5, pp. 1058–1073, 2009.

[78] N. Moshtagh, N. Michael, A. Jadbabaie, and K. Daniilidis, "Vision-based, distributed control laws for motion coordination of nonholonomic robots," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 851–860, 2009.

[79] L. Ojeda, D. Cruz, G. Reina, and J. Borenstein, "Current-based slippage detection and odometry correction for mobile robots and planetary rovers," *IEEE Transactions on Robotics*, vol. 22, no. 2, pp. 366–378, 2006.

[80] J. M. O'Kane and S. M. LaValle, "Localization with limited sensing," *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 704–716, 2007.

[81] R. Olfati-Saber and R. M. Murray, "Distributed cooperative control of multiple vehicle formations using structural potential functions," in *IFAC World Congress*, 2002, pp. 346–352.

[82] B. S. Park, S. J. Yoo, J. B. Park, and Y. H. Choi, "A simple adaptive control approach for trajectory tracking of electrically driven nonholonomic mobile robots," *IEEE Transactions on Control Systems Technology,*, vol. 18, no. 5, pp. 1199–1206, 2010.

[83] M. M. Polycarpou and A. J. Helmicki, "Automated fault detection and accommodation: a learning systems approach," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 11, pp. 1447–1458, 1995.

[84] L. Qiu, W.-J. Hsu, S.-Y. Huang, and H. Wang, "Scheduling and routing algorithms for AGVs: a survey," *International Journal of Production Research*, vol. 40, no. 3, pp. 745–760, 2002.

[85] V. Reppa, M. M. Polycarpou, and C. G. Panayiotou, "Adaptive approximation for multiple sensor fault detection and isolation of nonlinear uncertain systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 137–153, 2014.

[86] S. A. Reveliotis, "Conflict resolution in AGV systems," *IIE Transactions*, vol. 32, no. 7, pp. 647–659, 2000.

[87] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[88] E. Rohmer, S. P. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[89] S. Roumeliotis, G. S. Sukhatme, G. A. Bekey *et al.*, "Sensor fault detection and identification in a mobile robot," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 1998, pp. 1383–1388.

[90] L. Sabattini, V. Digani, C. Secchi, G. Cotena, D. Ronzoni, M. Foppoli, and F. Oleari, "Technological roadmap to boost the introduction of AGVs in industrial applications," in *Proc. IEEE International Conference on Intelligent Computer Communication and Processing*, 2013.

[91] A. V. Savkin and H. Teimoori, "Decentralized navigation of groups of wheeled mobile robots with limited communication," *IEEE Transactions on Robotics*, vol. 26, no. 6, pp. 1099–1104, 2010.

[92] S. Scheding, E. Nebot, and H. Durrant-Whyte, "The detection of faults in navigation systems: A frequency domain approach," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 3, 1998, pp. 2217–2222.

[93] R. Siegwart and I. R. Nourbakhsh, *Introduction to autonomous mobile robots*. The MIT Press, 2004.

[94] E. N. Skoundrianos and S. G. Tzafestas, "Finding fault-fault diagnosis on the wheels of a mobile robot using local model neural networks," *IEEE Robotics and Automation Magazine*, vol. 11, no. 3, pp. 83–90, 2004.

[95] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic, "Time windows based dynamic routing in multi-AGV systems," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 1, pp. 151–155, 2010.

[96] D. Stavrou, D. G. Eliades, C. G. Panayiotou, and M. M. Polycarpou, "Fault detection for service mobile robots using model-based method," *Autonomous Robots*, vol. 40, no. 2, pp. 383–394, 2015.

[97] D. Stavrou, D. Eliades, C. Panayiotou, and M. Polycarpou, "A path correction module for two-wheeled service robots under actuator faults," in *Proc. Mediterranean Conference on Control and Automation*, 2013.

[98] D. Stavrou and C. Panayiotou, "Localization of a simple robot with low computational-power using a single short range sensor," in *Proc. IEEE International Conference on Robotics and Biomimetics*, 2012, pp. 729–734.

[99] ——, "Task assignment and agent coordination in a warehouse environment," in *Proc. Mediterranean Conference on Control and Automation*, 2012, pp. 1341–1346.

[100] G. Steinbauer, "A survey about faults of robots used in robocup," in *RoboCup 2012: Robot Soccer World Cup XVI*. Springer, 2013, pp. 344–355.

[101] E. Stump, V. Kumar, B. Grocholsky, and P. M. Shiroma, "Control for localization of targets using range-only sensors," *The International Journal of Robotics Research*, vol. 28, no. 6, pp. 743–757, 2009.

[102] P. Sundvall and P. Jensfelt, "Fault detection for mobile robots using redundant positioning systems," in *Proc. IEEE International Conference on Robotics and Automation*, 2006, pp. 3781–3786.

[103] T. S. Tadele, T. de Vries, and S. Stramigioli, "The safety of domestic robotics: A survey of various safety-related publications," *IEEE Robotics and Automation Magazine*, vol. 21, no. 3, pp. 134–142, 2014.

[104] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[105] B. T. Thumati, T. Dierks, and J. Sarangapani, "A model-based fault tolerant control design for nonholonomic mobile robots in formation," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 9, no. 1, pp. 17–31, 2012.

[106] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis [robot fault diagnosis]," *IEEE Robotics and Automation Magazine*, vol. 11, no. 2, pp. 56–66, 2004.

[107] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677–709, 2006.

[108] H. Wang, Y. Chen, and P. Souères, "A geometric algorithm to compute time-optimal trajectories for a bidirectional steered robot," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 399–413, 2009.

[109] C. C. Ward and K. Iagnemma, "Model-based wheel slip detection for outdoor mobile robots," in *Proc. IEEE International Conference on Robotics and Automation*, 2007, pp. 2724–2729.

[110] N. Wu and M. Zhou, "Shortest routing of bidirectional automated guided vehicles avoiding deadlock and blocking," *IEEE/ASME Transactions on Mechatronics*, vol. 12, no. 1, pp. 63–72, 2007.

[111] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, p. 9, 2008.

[112] J. Yang, Z. Qu, J. Wang, and K. Conrad, "Comparison of optimal solutions to real-time path planning for a mobile vehicle," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,*, vol. 40, no. 4, pp. 721–731, 2010.

[113] X. Zhang, M. M. Polycarpou, and T. Parisini, "A robust detection and isolation scheme for abrupt and incipient faults in nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 4, pp. 576–593, Apr. 2002.

[114] D. Zhuo-hua, C. Zi-xing, and Y. Jin-xia, "Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey," in *Proc. IEEE International Conference on Robotics and Automation*, 2005, pp. 3428–3433.