



A Distributed Framework for the Control and Cooperation of Heterogeneous Mobile Robots in Smart Factories

Nicol Naidoo

Supervisors: Prof. Glen Bright and Prof. Riaan Stopforth

Submitted in fulfilment of the degree of Doctor of Philosophy in Engineering in the

Mechatronics and Robotics Research Group

School of Engineering

September 2017

Dedicated to my wife and daughter

Declaration: Submission

Supervisor

As the Candidate's Supervisor, I agree to the submission of this thesis.

Signed: Date:.....

Prof. Glen Bright

Student

I declare this thesis has not previously been submitted for any degree in this university, or any other university and it is my (Nicol Naidoo) original work.

Signed: Date:

Mr. Nicol Naidoo

Declaration: Plagiarism

I, Nicol Naidoo (203503153), declare that

1. The research reported in this thesis, except where otherwise indicated, is my original research.
2. This thesis has not been submitted for any degree or examination at any other university.
3. This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
 - a. Their words have been re-written but the general information attributed to them has been referenced
 - b. Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.
5. This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Signed: Date:

Mr. Nicol Naidoo

Declaration: Publications

Details of contribution to publications that form part and/or include research presented in this thesis (include publications in preparation, submitted, *in press* and published and give details of the contributions of each author to the experimental work and writing of each publication)

Publications

In the publications listed, Mr. Nicol Naidoo was the developer and lead author of the work:

1. N Naidoo, G Bright, " Support Vector Machine Learning in Multi-Robot Teams," in *Proceedings of the 11th International Conference on Autonomic and Autonomous Systems*, Rome, 2015, pp. 1-6.
2. N Naidoo, G Bright, R Stopforth, ZF Zelasco, J Donayo, "Optimizing Search and Rescue Missions through a Cooperative Mobile Robot Network," in *Proceedings of the 8th IEEE International Conference on Robotics and Mechatronics (RobMech)*, Port Elizabeth, 2015, pp. 7-11.
3. N Naidoo, G Bright, R Stopforth, "A cooperative mobile robot network in ROS for advanced manufacturing applications," in *Proceedings of the 6th International Conference on Competitive Manufacturing (COMA)*, Stellenbosch, 2016, pp. 281-286.
4. N Naidoo, G Bright, R Stopforth, "Navigation and Control of Cooperative Mobile Robots using a Robotic Middleware Platform," in *Proceedings of the 12th IEEE International Conference on Control & Automation (ICCA)*, Kathmandu, 2016, pp. 927-932.
5. N Naidoo, G Bright, R Stopforth, " The Cooperation of Heterogeneous Mobile Robots in Manufacturing Environments using a Robotic Middleware Platform," in *Proceedings of the 8th IFAC Conference on Manufacturing, Modelling, Management and Control*, Troyes, 2016, pp. 1032-1037.

6. N Naidoo, G Bright, R Stopforth, "Towards a Decentralised Mobile Robot Learning System for Indoor Environments," in Proceedings of the 24th IEEE International Conference on Mechatronics and Machine Vision in Practice (M2VIP), Auckland, 2017.

7. N Naidoo, G Bright, R Stopforth, M Bergamini, J Kamlofsky, J Zelasco, F Ansaldo, "Semi-Autonomous Robot Control System with 3D Vision Scheme for Search and Rescue Missions," in Proceedings of the 24th IEEE International Conference on Mechatronics and Machine Vision in Practice (M2VIP), Auckland, 2017.

Submissions

In the submission listed, Mr. Nicol Naidoo was the developer and lead author of the work:

1. N Naidoo, G Bright, R Stopforth, "A Framework for Establishing Cooperation in Multiple Heterogeneous Mobile Robot Systems Configured for Manufacturing Applications," paper submitted to the *R & D Journal of the South African Institution of Mechanical Engineering*, 2017.

Signed: Date:

Mr. Nicol Naidoo

Acknowledgements

I firstly thank the Lord Jesus Christ for blessing me with the opportunity to pursue this degree and for making all of this possible.

I am grateful to my supervisors, Professor Glen Bright and Professor Riaan Stopforth, for guiding me through this journey and for the provision of resources for this research.

I would like to thank the senior staff at Clifford Machines and Technology, for their continuous support throughout the duration of this degree.

Lastly, I thank and appreciate my wife, Aurelle Naidoo, for her love and support during every step of the way. She truly sacrificed as much as I had during the past few years.

Abstract

The present consumer market is driven by the mass customisation of products. Manufacturers are now challenged with the problem of not being able to capture market share and gain higher profits by producing large volumes of the same product to a mass market. Some businesses have implemented mass customisation manufacturing (MCM) techniques as a solution to this problem, where customised products are produced rapidly while keeping the costs at a mass production level. In addition to this, the arrival of the fourth industrial revolution (Industry 4.0) enables the possibility of establishing the decentralised intelligence of embedded devices to detect and respond to real-time variations in the MCM factory.

One of the key pillars in the Industry 4.0, smart factory concept is Advanced Robotics. This includes cooperation and control within multiple heterogeneous robot networks, which increases flexibility in the smart factory and enables the ability to rapidly reconfigure systems to adapt to variations in consumer product demand. Another benefit in these systems is the reduction of production bottleneck conditions where robot services must be coordinated efficiently so that high levels of productivity are maintained.

This study focuses on the research, design and development of a distributed framework that would aid researchers in implementing algorithms for controlling the task goals of heterogeneous mobile robots, to achieve robot cooperation and reduce bottlenecks in a production environment. The framework can be used as a toolkit by the end-user for developing advanced algorithms that can be simulated before being deployed in an actual system, thereby fast prototyping the system integration process.

Keywords: Cooperation, heterogeneity, multiple mobile robots, Industry 4.0, smart factory, manufacturing, middleware, ROS, OPC, framework.

Contents

Contents	xiv
List of Figures	xix
List of Tables	xxi
Abbreviations	xxiii
Chapter 1 Introduction.....	1
1.1 Mass customisation manufacturing and Industry 4.0.....	1
1.2 Robotics in manufacturing	2
1.3 Motivation	4
1.4 Aim and objectives.....	5
1.5 Research methodology	5
1.6 Research contributions	6
1.7 Thesis outline	6
1.8 Chapter summary	7
Chapter 2 Literature survey	9
2.1 Mobile robot systems	9
2.1.1 Single robot systems	10
2.1.2 Multi-robot systems	11
2.1.3 Heterogeneity	12
2.1.4 Cooperative vs. competitive systems	13
2.1.5 Communication.....	14
2.1.6 Control architectures	15
2.2 Robotic middleware	16
2.2.1 Middleware roles	17
2.2.2 Middleware platforms	18
2.3 Smart factories.....	25
2.3.1 An introduction to the Industry 4.0 concept	26
2.3.2 M2M communication and Advanced Robotics	28
2.4 Chapter summary	30
Chapter 3 Design of the framework	31
3.1 Design requirements.....	31
3.2 ROS middleware	33

3.3	Industrial data communication	35
3.3.1	Communication model	36
3.3.2	Data communication standards	37
3.3.3	OPC-UA	39
3.4	IDE and GUI	41
3.5	SCADA	42
3.6	Final design overview	43
3.7	Functional design specifications	44
3.8	Chapter summary	46
Chapter 4	Framework development	47
4.1	Development and verification strategies	47
4.2	Development overview	48
4.3	OPC-UA communication	49
4.4	GUI development in Python Tkinter	52
4.4.1	Main screen and remote robot connection	53
4.4.2	Robot configuration	55
4.4.3	Task management	56
4.4.4	OPC-UA tag assignment	57
4.4.5	Robot location	59
4.4.6	Python OPC-UA Client	59
4.5	Mobile robot system integration	61
4.5.1	ROS implementation	61
4.5.2	Robot OPC-UA Client	62
4.5.3	Decentralised robot control	63
4.5.4	Stage simulation	64
4.6	SCADA development	65
4.7	Python programming features	68
4.7.1	Local variables and OPC tags	68
4.7.2	Heterogeneous mobile robots and cooperation	69
4.7.3	Machine learning support	71
4.8	Framework review	72
4.9	Chapter summary	73
Chapter 5	Case studies	75
5.1	Study I: Single mobile robot control	75
5.1.1	Description of the study	75
5.1.2	Application development	76
5.1.3	Simulation results	80

5.2	Study II: Cooperation in heterogeneous mobile robot teams	84
5.2.1	Description of the study	84
5.2.2	Application development	85
5.2.3	Simulation results.....	86
5.3	Study III: Machine learning robots	89
5.3.1	Description of the study	89
5.3.2	Application development	90
5.3.3	Simulation results.....	91
5.4	Study IV: Integration with a third party system	93
5.4.1	Description of the study	94
5.4.2	Application development	94
5.4.3	Simulation results.....	96
5.5	Chapter summary	99
Chapter 6	Discussion	101
6.1	Benefits of the framework.....	101
6.1.1	Popular software platforms	101
6.1.2	Mobile robot control	102
6.1.3	Distribution features.....	102
6.1.4	Robot heterogeneity and cooperation	103
6.1.5	Machine learning	103
6.1.6	Interoperability benefits	104
6.2	Further scope of the framework	104
6.2.1	Business management systems	105
6.2.2	Resource sharing.....	105
6.2.3	Robotic manipulators	105
6.3	Related research	106
6.4	Limitations and advanced development	108
6.5	Chapter summary	110
Chapter 7	Conclusion	113
7.1	Aim and objectives.....	113
7.2	Research summary	114
7.3	Research contributions	117
7.4	Recommendations for further research	118
References	119
Appendix A:	Python Tkinter GUI code	133
Appendix B:	Remote robot control code	167
Appendix C:	Robot OPC-UA Client code	171

Appendix D: Stage development programs	173
D1. Launch file	173
D2. Yaml file	179
D3. World file	180
D4. Map and turtlebot files	181
Appendix E: Functions for heterogeneous robot programs	183
Appendix F: SVM function code	189
Appendix G: Python application code for Study I.....	191
G1. Algorithm and mobile robot navigation program	191
G2. Simulation program.....	196

List of Figures

Figure 1.1: IFR report on global install base of industrial robots	3
Figure 2.1. Some AGV solutions by JBT: (a) AGV Forklift, (b) AGV unit load carrier	10
Figure 2.2. Robotnik's JR2 mobile manipulator.	11
Figure 2.3. Categorisation of cooperative behaviours in MMRS.	14
Figure 2.4. The middleware layer.	16
Figure 2.5. OROCOS libraries.	22
Figure 2.6. The cyber-physical production system.	26
Figure 2.7. Industry 4.0 environment.	27
Figure 2.8. M2M communication in smart factories.	29
Figure 2.9. Robotics and Industry 4.0.	30
Figure 3.1. Preliminary design overview.	32
Figure 3.2. Topic-based communication in ROS.	34
Figure 3.3. Hierarchical communication and control model in manufacturing plants.	36
Figure 3.4. The 7 layers of the OSI model.	38
Figure 3.5. Ethernet frame with EtherCAT data.	39
Figure 3.6. Hierarchical communication and control model with OPC-UA.	40
Figure 3.7. Final design overview of the framework.	43
Figure 4.1. Overview of the framework development discussed in this chapter.	48
Figure 4.2. Communication between Matrikon OPC-UA Server and Clients.	50
Figure 4.3. Matrikon OPC Server alias (tag) creation.	51
Figure 4.4. Matrikon OPC Server and Explorer with tag value monitoring.	51
Figure 4.5. Flowchart of the GUI functionality.	52
Figure 4.6. The GUI main screen with offline and online connections.	53
Figure 4.7. The robot list pop-up window.	54
Figure 4.8. A simplified setup flow of a SSH connection.	55
Figure 4.9. The robot configuration GUI screen.	56
Figure 4.10. The task management GUI screen.	57
Figure 4.11. Algorithm and robot tag assignment screen.	58
Figure 4.12. Tag assignment screen with OPC tag list pop-up window.	58
Figure 4.13. The task location configuration screen.	59
Figure 4.14. Python OPC Client communication with the Server.	60

Figure 4.15. Overview of the mobile robot system integration.	61
Figure 4.16. A basic format of the ROS core implementation.	62
Figure 4.17. State diagram of the local navigation algorithm.....	63
Figure 4.18. Stage file associations with Appendix references in brackets.	64
Figure 4.19. Stage simulation window with map and robots.....	65
Figure 4.20. Ecava SCADA Project Editor showing the OPC client tag configuration.	66
Figure 4.21. SCADA screen development using Inkscape.....	67
Figure 4.22. A SCADA project execution in the Internet Explorer web browser.	67
Figure 4.23. Functions that map and update local variables.....	69
Figure 4.24. Functions developed for heterogeneous robot applications.	70
Figure 4.25. SVM train application created in Python Tkinter.....	71
Figure 4.26. A review of the framework development.	72
Figure 5.1. Control algorithm to determine the robot task priority.	78
Figure 5.2. Flow diagram for the application program of Case Study I.	79
Figure 5.3. SCADA mimic for the material handling application of Case Study I.	80
Figure 5.4. Buffer levels and task weight outputs of Case Study I, simulation-1.	81
Figure 5.5. x-y robot coordinates during simulation-1 of Case Study I.	82
Figure 5.6. Buffer levels and task weight outputs of Case Study I, simulation-2.	83
Figure 5.7. x-y robot coordinates during simulation-2 of Case Study I.	83
Figure 5.8. Buffer levels and task weight outputs of Case Study II, simulation-1.	86
Figure 5.9. x-y robot coordinates during simulation-1 of Case Study II.	87
Figure 5.10. Buffer levels and task weight outputs of Case Study II, simulation-2.	88
Figure 5.11. x-y robot coordinates during simulation-2 of Case Study II.	88
Figure 5.12. Flow diagram for the application program of Case Study III.	90
Figure 5.13. Buffer levels and SVM outputs of Case Study III, simulation-1.	92
Figure 5.14. Buffer, demand levels and SVM outputs of Case Study III, simulation-2.....	93
Figure 5.15. PLC and Python algorithms for Case Study IV.....	95
Figure 5.16. TwinCAT 3 PLC programming environment.	96
Figure 5.17. Time delay analysis for slow communication intervals.	97
Figure 5.18. Time delay analysis for average communication intervals.	97
Figure 5.19. Time delay analysis for fast communication intervals.	97

List of Tables

Table 1. Summary of the four industrial revolutions.....	2
Table 2. Middleware comparisons (part 1).....	24
Table 3. Middleware comparisons (part 2).....	25
Table 4. Starting line reference to the Python code in Appendix A.	53
Table 5. OPC tag definitions for Case Study I.....	77
Table 6. Robot and task compatibility table for the first simulation of Case Study II.	85
Table 7. Robot and task compatibility table for the second simulation of Case Study II.....	85
Table 8. Time delay analysis for the three simulations in Case Study IV.	98
Table 9. Reference summary of the framework features.	116

Abbreviations

AI	Artificial Intelligence
AIMM	Autonomous Industrial Mobile Manipulator
AMCL	Adaptive Monte Carlo Localization
ARCADE	Architecture for Real-time Control and Autonomous Distributed Execution
CAD	Computer Aided Design
CARMEN	Carnegie Mellon Navigation
CBSE	Component-Based Software Engineering
CIP	Common Industrial Protocol
CLARAty	Coupled Layer Architecture for Robotic Autonomy
CORBA	Common Object Request Broker Architecture
CPPS	Cyber-Physical Production System
CPU	Central Processing Unit
DARPA	Defence Advanced Research Projects Agency
DCS	Distributed Control System
DOF	Degrees of Freedom
EKF	Enhanced Kalman Filter
ERP	Enterprise Resource Planning
FMS	Flexible Manufacturing Systems
GUI	Graphical User Interface

HMI	Human Machine Interface
HTTP	Hyper-Text Transfer Protocol
IEC	International Electro-technical Commission
IFR	International Federation of Robotics
IIoT	Industrial Internet of Things
ICE	Internet Communications Engine
ICT	Information and Communications Technology
IDE	Integrated Development Environment
IRT	Isochronous and Real-Time
ISO	International Standards Organisation
IT	Information Technology
LRF	Laser Range Finder
LQI	Link Quality Indicator
M2M	Machine-to-Machine
MARIE	Mobile and Autonomous Robotics Integration Environment
MCM	Mass Customisation Manufacturing
MES	Manufacturing Execution System
MMRS	Multiple Mobile Robot Systems
MRP	Manufacturing Resource Planning
MSRDS	Microsoft Robotics Developer Studio
MRS	Multi-Robot Systems
MRTM	Multi-Robot Task Module
OPC-UA	OPC Unified Architecture

OROCOS	Open Robot Control Software
OS	Operating System
OSI	Open System Interconnection
P2P	Peer-to-Peer
PDM	Product Data Management
PLC	Programmable Logic Controller
Pyro	Python Robotics
ROS	Robot Operating System
RSSI	Received Signal Strength Indicator
RTDB	Real Time Data Base
RTP	Real-time Transport Protocol
RTT	Real Time Toolkit
SCADA	Supervisory Control and Data Acquisition
SOA	Service-Orientated Architecture
SNRP	Simple Network Robot Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TWR	Two Way Ranging
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
VPN	Virtual Private Network
WS	Web Services
XML	eXtended Markup Language

Chapter 1 Introduction

This introductory chapter discusses the problem of mass customisation and the role of mobile robotics in these types of manufacturing environments. The motivation of the research study, along with the objectives, methodology, and contributions are also addressed.

1.1 Mass customisation manufacturing and Industry 4.0

The advancement of electronics and technology in the past few decades has led to its adoption in the manufacturing industry and businesses have seen tremendous growth in productivity due to the use of advanced technology in its processes [1]. The increase in productivity margins has catapulted competition on a global scale since consumers require fast deliveries and have access to a wide range of courier service options [2]. Another factor that has changed the present consumer market is the mass customisation of products. Manufacturers are now challenged with the problem of not being able to capture market share and gain higher profits by producing large volumes of the same product to a mass market [3]. Some businesses have implemented mass customisation manufacturing (MCM) techniques as a solution to this problem, where customised products are produced rapidly while keeping the costs at a mass production level.

The “mass customisation” concept was first formally introduced in the book *Future Perfect* by Stanley M. Davis in 1983 and thereafter defined as a strategy for product variety and individualisation by Joseph Pine in 1993 [4]. The concept has now become a reality through the use of robotics in manufacturing, advances in computer-aided design (CAD) packages, product data management (PDM), and industrial networking technologies [5]. In order to continuously achieve profitability and maintain the competitive advantage over others, leading manufacturers spend a great deal of time optimising their MCM processes to manage

predicted changes with precision, and to respond to unexpected changes with speed and flexibility [6]. The management of such variations can be achieved through real-time data processing, analysis, and control in the current fourth industrial revolution (Industry 4.0).

Industry 4.0, or the industrial internet of things (IIoT), has been the popular topic of discussion among researchers and manufacturers during the past three years. Table 1 [7] is a summary of the four industrial revolutions in history.

Table 1. Summary of the four industrial revolutions.

	Time periods	Technologies and capabilities
First	1784 – mid 19 th century	Water and steam powered mechanical engineering
Second	Late 19 th century – 1970's	Electric powered mass production based on the division of labour
Third	1970's – Today	Electronics and information technology drives new levels of automation of complex tasks
Fourth	Today–	Sensor technology, interconnectivity and data analysis allow mass customisation, integration of value chains and greater efficiency

The vision of Industry 4.0 is the use of sensor and network technology in ‘smart products’ that define how machines should undertake processes. In this environment, the decentralised intelligence of embedded devices has the ability to detect and respond to real-time variations in the MCM factory; this can be contrasted to the slower response times and single point of failure in traditional centralised factory control systems.

1.2 Robotics in manufacturing

Over the past few decades, extensive research in robotics has resulted in a continuous growth in industrial application sectors such as automotive, electronics, metal, and food amongst others. Figure 1.1 [8] is a report by the International Federation of Robotics (IFR), showing the growth since 1985. According to the data, it took almost 50 years for the install base to

reach the first one million units, contrasted to just eight more years to reach the projected second million units.

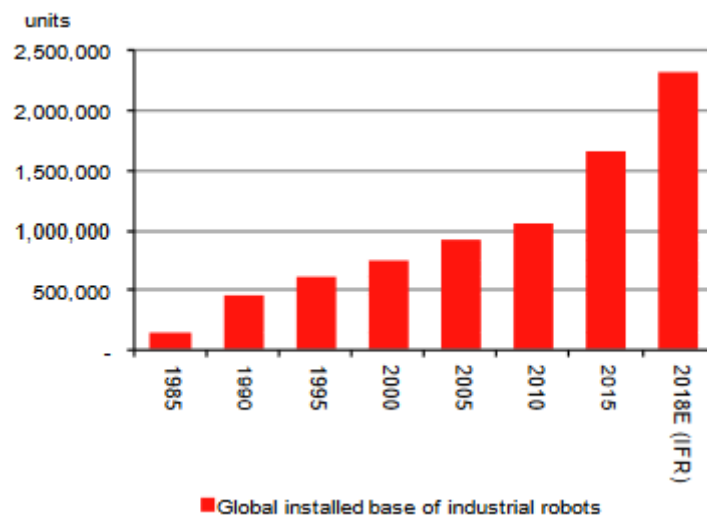


Figure 1.1: IFR report on global install base of industrial robots

From the install base and according to an *Industrial Robotics Marketing Report*, materials handling is currently the leading application of industrial robots, accounting for more than 33% of the global industrial robotics market in 2013 [9]. A contributing factor to this statistic is the technological advancement (and lower cost) of robotic vision systems, navigation and mobility, where companies are beginning to offer sophisticated material handling methods for factories and warehouses [10]. Hence, the use of mobile robots in industry has become popular, particularly in MCM factories where reconfiguration and flexibility are key requirements in the competitive climate with complex consumer demands [11].

The cooperation of multiple mobile robot systems (MMRS) is recently becoming a popular area of interest to both researchers and manufacturers, since a network of mobile robots has the ability to perform tasks much quicker and more efficiently than single robot systems. The management of mobile robot resources, however, is non-trivial due to the existence of heterogeneity in the network of robots [12]. Heterogeneities may exist due to the variety of actuators, sensors, and communication protocols developed by different manufacturers. The next chapter explores literature in the areas of mobile robots and cooperative robotic research,

and investigates the impact of MMRS in an Industry 4.0 environment, together with the associated challenges.

1.3 Motivation

Globally, a great measure of resources have been allocated to the research and development of robotics, autonomous mobile robots, and cooperation among robots [13][14], and forecasts reveal a significant adoption of mobile robots in industry and homes in the near future [15]. Advanced robotics and machine-to-machine (M2M) communication, alongside others, are considered key pillars in the Industry 4.0 concept, hence the globalised interest in these areas of research.

The issue of heterogeneity in multiple mobile robot systems was investigated and the use of robotic middleware is discussed in the literature survey (chapter 2) as a solution to mask the heterogeneities in the network. Various middleware platforms were studied in the survey, revealing that the choice of robotic middleware for a particular application requires careful consideration, since some platforms contain attributes which others lack. In the context of an industrial application, the Robot Operating System (ROS) stands out from the rest, primarily due to its commercial use [16] and its application in industry through the ROS Industrial Consortium [17]. ROS, however, does have its limitation in regard to functionality for cooperation in multiple mobile robot networks.

The motivation behind the research stems from the need of a robotic framework that can be used as a tool for the development of cooperation systems in industrial MMRS. The need exists due to the modern popularity of MMRS and cooperation in these systems to facilitate flexibility in smart factories, thereby managing the problem of mass customisation. To summarise, the following problems and challenges have been identified in the literature survey, thereby driving the need for the design and development of the framework:

- Mass customisation and global competition [3].
- Increase in the use of advanced technology [1] and robotics [8] in manufacturing processes.

-
- The existence of heterogeneity in multiple mobile robot systems [12].
 - The need for robotic middleware to mask heterogeneities and facilitate multi-robot cooperation in industrial environments.
 - The introduction of the fourth industrial revolution (Industry 4.0) and the key role of robotics in this setting [18].
 - The need for the operation of intelligent mobile robots in smart factory environments [7].

1.4 Aim and objectives

The aim of this study was to research, design, and develop a distributed framework for managing multiple, heterogeneous mobile robots in smart factories.

The objectives of this research study were to:

- Investigate the impact of mobile robot systems in industry.
- Investigate the problem of mass customisation together with the introduction of the fourth industrial revolution (Industry 4.0).
- Research the need for robotic middleware and the various platforms available for mobile robot systems.
- Research the design requirements of a distributed framework for use in an industrial MMRS setting.
- Design and develop the framework that facilitates cooperation in heterogeneous multiple mobile robot teams.
- Test and validate the performance of the framework by the use of algorithms and simulation tools.

1.5 Research methodology

The verification of the framework developed in this study was achieved through a case study approach. A series of case studies were performed that involved different scenarios pertaining

to mobile robot material handling, and simulated robot task applications. Since the framework is a tool that will be used by the end-user, the studies demonstrate the application of example user algorithms that can be used in the high-level robot decision making process. Simulation results are also documented for each case study, verifying the functionality of the framework and user algorithms, and the navigation of the mobile robots in a simulated environment.

1.6 Research contributions

The design and development of the framework in this research study was an original effort and has made the following five contributions:

1. The framework has made the remote control of robots more accessible by *industrial engineers* since the framework adheres to an international industrial communication protocol standard.
2. The framework has made it easier for *students or researchers* to advance the field of multi-robot cooperation since solutions can be developed and tested in these scenarios. This is achieved through the use of the simple graphical user interface and the development of algorithms in Python.
3. The *advanced robotics* pillar in Industry 4.0 has been strengthened due to the common interface provided by the framework to achieve intelligent robotic systems.
4. The capability of the popular *ROS middleware* has been enhanced, since ROS has lacked the functionality for cooperative multi-robot systems.
5. The *machine learning* field of research can broaden its application to mobile robot networks, via the use of Python's powerful machine learning library.

1.7 Thesis outline

Chapter 2 is a literature survey that investigates the role of mobile robot systems in industry, the associated challenges, and its impact in smart factories. Chapter 3 presents the methodological approach taken during the design of the framework. The design requirements

are defined, outlining the key components of the framework that enables its functionality in a smart factory environment. Chapter 4 describes the development and verification strategies adopted, and thereafter discusses the development of the main components of the framework. Chapter 5 presents four case studies, each verifying various aspects of the framework in simulation results. Chapter 6 discusses the benefits of the framework, research related to the functionality of the framework, and the limitations as well as advanced development features that should be implemented as improvements to the system. Chapter 7 concludes the research study by summarising the work, revisits the contributions discussed here, and discusses some recommendations for further research.

1.8 Chapter summary

This chapter introduced the *mass customisation* and *Industry 4.0* concepts, and thereafter discussed the growth of robotics in industry. The motivation of the study was discussed and the objectives, methodology, and contributions of the research were defined to establish the scope of work involved in this study. The chapter concluded with an outline of the thesis.

Chapter 2 Literature survey

The purpose of the survey was to investigate the role of mobile robot systems in industry. Associated with this are factors such as heterogeneity, cooperation, communication, and control architectures, all of which are discussed in the survey. The various types of robotic middleware platforms are reviewed as they are essential to the establishment of common interfaces in networks that contain robotic heterogeneity. The final topic of discussion is “smart factories” (another term for Industry 4.0); here the purpose of the concept is introduced, along with the influence of advanced robotic systems and the related challenges.

2.1 Mobile robot systems

Apart from manufacturing, mobile robots can be used in a variety of other applications where it may be dangerous for humans to work in such as mining, search and rescue [19], planetary exploration [20], and toxic waste clean-up [21]. In the manufacturing environment, mobile robots are required to operate autonomously, which involves the use of mobile robot sensors for the robots to: 1) build or update maps of the environment, 2) localise its position on the map, 3) navigate the environment, and 4) avoid obstacles. Some manufacturing applications call for the use of other auxiliary equipment to be installed on mobile robots such as vision sensors for object recognition and grippers for pick and place operations. The following subsections discuss some of the applications with single and multiple mobile robot systems and the pertinent aspects related to these systems, viz. heterogeneity, cooperation, communication, and robot control architectures.

2.1.1 Single robot systems

Automated Guided Vehicles (AGVs) were invented in 1953 and have been used as material transporters in manufacturing facilities or warehouses ever since [22]. Various types of AGVs exist, Figure 2.1 depicts two examples by JBT AGVs [23]. Figure 2.1 (a) is an example of an AGV forklift which automatically moves the load to its programmed destination after being loaded by an operator. Figure 2.1 (b) is an AGV unit load carrier, used to move loads from one station to another; these AGVs can interface with conveyors and are usually equipped with either or combinations of belts, powered rollers, and mechanised lifts to automatically load and unload batches.



Figure 2.1. Some AGV solutions by JBT: (a) AGV Forklift, (b) AGV unit load carrier

The challenges of navigation with AGVs can be simplified by restricting their path to predetermined routes via demarcated floors or underground cables; however, this can be a disadvantage in Flexible Manufacturing Systems (FMS) where some of the flexibility of the system is sacrificed due to fixed structures designed to accommodate the AGV's path. In their research paper [24], Sabbatini et al. describe the process of installing and configuring AGVs in a factory, and thereafter discuss some of the barriers associated with further adoption of the technology:

- expensive cost
- difficulty in achieving the desired efficiency to absorb the cost
- changeovers can be time consuming and expensive due to the lack of flexibility
- safety concerns

Another type of mobile robot that is becoming very popular in FMS environments due to its flexibility features is the Autonomous Industrial Mobile Manipulator (AIMM). An AIMM is an industrial robot that is capable of performing tasks such as part feeding, assembly, and transportation. In general, AIMMs are designed with four elements: the mobile platform, sensors (e.g. vision, laser range finders), a manipulator, and a tool (e.g. grippers, welding gun). Figure 2.2 [25] is the JR2, an AIMM manufactured by Robotnik, designed for logistics and industrial mobile applications. The JR2 arm (manipulator) has six degrees of freedom (DOF) and the mobile platform uses four high powered motor wheels, capable of supporting loads of up to 100kg. The sensors mounted on the robot include two rangefinders for navigation and safety, as well as two RGB-D sensors to detect obstacles located at different heights.



Figure 2.2. Robotnik's JR2 mobile manipulator.

The AIMM category of robots are well suited for FMS environments, although product changeovers can be time consuming on single robot systems if they are not easily reconfigurable.

2.1.2 Multi-robot systems

Multi-robot systems (MRS) potentially have several advantages over single robot systems; Yan et al [26] discuss the following benefits in their research paper:

- A MRS has a better spatial distribution.
- A MRS can achieve a superior system performance, such as the time to complete a task, or energy consumption of the robots.
- Robustness and fault-tolerance can be realised in MRS due to data fusion and information sharing in the network of robots.
- A MRS can potentially cost less since a large number of simpler robots will be cheaper to build and program, contrasted to a single, powerful robot.
- System reliability, flexibility and scalability features exist in MRS.

Some preliminary work in MRS began in the late 1980's with a few projects such as ACTRESS [27], CEBOT [28], ALLIANCE [21], M+ [29], and MURDOCH [30], each yielding successful results. Today, multiple mobile robot systems are being implemented in industry and have proven to be very efficient. One such case is the KIVA Systems automated warehouse, where mobile robots optimise the execution of customer orders by placing frequently required material in locations that are closer to the delivery area, rather than storing them in fixed locations. Kiva, now known as Amazon Robotics [31], was acquired by Amazon.com in 2012 due to its significant success rate in efficiently fulfilling customers' orders.

The implementation of a MMRS in a factory can be a challenging task with many factors to consider, these may include: factory infrastructure, robot heterogeneity, robot path planning and frequent obstacle avoidance actions due to the volume of robots in the environment, safety and interaction with humans, and coordination and cooperation between the robots to achieve specific goals. The following sections discuss some of these challenges inherent in MMRS.

2.1.3 Heterogeneity

A MMRS can be homogeneous or heterogeneous. In homogeneous networks, each robot in the team has an identical capability, whilst in heterogeneous robot teams capabilities differ since robots can be specialised for specific tasks, making task allocation in these networks more difficult to manage [32]. Robot task allocations in heterogeneous systems are

determined by the individual capabilities, whereas there exists a greater flexibility in homogeneous systems and tasks can be assigned dynamically. The advantage of heterogeneous over homogeneous systems becomes clear when task-flexibility is required in a FMS environment. In a FMS factory, task variety is common and having a team of robots with ‘mixed’ resources could execute tasks more quickly if the robots are managed in an efficient manner. Another benefit of having a heterogeneous mobile robot team is redundancy and fault-tolerances: if a particular robot in the team has failed due to a malfunction or requires maintenance, another ‘helper’ robot that is suited to the task can assist and take over the job; in this way, robots can help each other and eliminate potential bottlenecks in a production environment, thereby ensuring a material flow optimisation [33].

Apart from the benefits associated with task-flexibility, Parker [34] explains some reasons why heterogeneity exists in multi-robot systems:

- Heterogeneity can present *engineering benefits* as it can be challenging and impractical to design single, powerful robots with numerous capabilities.
- Following the point above, it makes *economic sense* to distribute the specialised abilities across a team of simple robots, rather than commissioning a smaller group of expensive robots.
- The reality of *diversity* in a homogeneous system can exist over time due to robot wear and tear, differences in robot construction as a result of maintenance, as well as differences in sensor calibration.

The next two sections introduce two types of collective behaviours inherent in multi-robot systems and investigate ways in which heterogeneous robots can communicate to influence team goals.

2.1.4 Cooperative vs. competitive systems

MRS environments can either be cooperative or competitive [35]. In competitive systems, robots try to accomplish their own self-interests and compete with other robots in the network. Examples of such systems are strategy games such as RoboCup [36] and chess [37]. Conversely, cooperative systems are more applicable to manufacturing environments as they

involve robots that interact with each other, either directly or indirectly, to achieve common goals. In these systems the robots are not ‘selfish’, but will act in ways that would benefit the global goal of the system; for example, in a production environment, robots can cooperative to alleviate the problem of bottlenecks in the system [33] and hence, prevent significant decrease in efficiency.

In MMRS networks, cooperative systems can be further categorised into *collective swarm systems* and *intentionally cooperative systems*, as depicted in Figure 2.3 [34]. Collective swarm systems are pertinent to homogeneous mobile robot networks where robots can resemble the characteristics of birds, bees, or ants – an area of research commonly known as swarm robotics [38]. On the other hand, heterogeneous robot teams are intentional cooperative systems since robots are aware of each other and act based upon the abilities of the members in the team. These systems can be further categorised into strongly cooperative or weakly cooperative solutions, based upon the level of communication between the robots. Solutions can form high levels of communication and synchronisation (strongly cooperative), or allow for periods of operational independence among robots (weakly cooperative) [34].

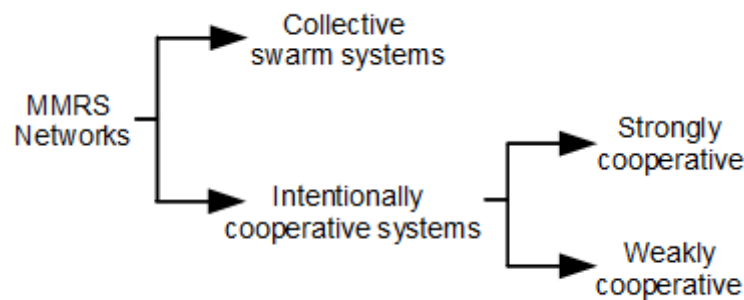


Figure 2.3. Categorisation of cooperative behaviours in MMRS.

2.1.5 Communication

Literature [26] identifies two forms of communication inherent in MMRS networks: *implicit* and *explicit*. Explicit communication involves a direct form of data exchange (either unicast or broadcast) between robots in the network, and will make use of: 1) an on-board communication module, 2) a physical communication medium, and 3) a data communication protocol. Implicit communication is an indirect form of data retrieval, where robots

communicate by either using the changes in the environment through sensors, or collecting information left by other robots.

The advantage that explicit communication systems has over implicit types is the accuracy of the data exchanged between robots, however, the communication bandwidth of the system can increase in proportion to the number of robots in the network, which can lead to a decrease in system performance. Implicit networks are more stable and fault tolerant, but is not an ideal option in a manufacturing system due to the dependency on data integrity to ensure predictable outcomes in these environments.

2.1.6 Control architectures

The control architecture of a MMRS is critical to the design of the network [39] since it influences other decisions that are made later in the design stage, such as cooperation and communication methods [40]. Parker [34] identifies four types of control architectures:

- *Centralised* architectures control the team of robots from a single point. The advantage of this architecture is the global view point of the environment and the robots, thus optimal plans can be produced [26]. Nevertheless, centralised systems are vulnerable due to the single point of failure, and can be practically unrealistic for real-time control scenarios, for example, when mobile robots are required to perform dynamic obstacle avoidance techniques.
- *Decentralised* architectures comprise of individually controlled robots, each having complete control of their actions based upon local knowledge. This system is robust to failure and flexible; however, it may be difficult to ensure the execution of high level goals due to the local, independent behaviour of each robot in the team.
- *Hierarchical* control architectures combine centralised and decentralised schemes. Similar to military operations, there are ‘leader’ robots in the team that assume control over other ‘follower’ robots. Hierarchical architectures are still prone to failure due to the dependency of the ‘follower’ robots on the ‘leaders’.
- *Hybrid* architectures have become a popular control scheme [34]. It is also a combination of the centralised and decentralised architectures, but unlike

hierarchical systems, no particular robot has control over another in the team. Hybrid systems ensure localised robot control for actions that require real-time processing, and support control for higher-level team goals [41].

2.2 Robotic middleware

The robotic middleware is an abstraction layer established between the application layer and operating system in computing systems, as shown in Figure 2.4 [42]. The purpose of the middleware in robotic systems is to mask the heterogeneity of hardware devices in the network, created by differences in sensors and actuators. Another reason is software related, and includes the simplification of software design and project development costs [42]. This is achieved by the provision of standard interfaces to robot sensors and actuators, ensuring software modularity and the deployment of re-usable, high-level code on different hardware architectures [43].

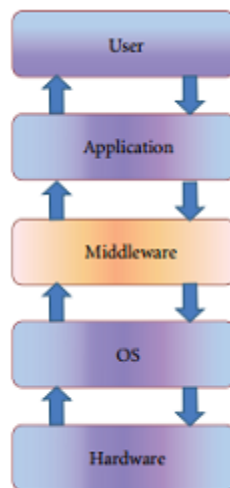


Figure 2.4. The middleware layer.

The middleware layer plays a key role in the cooperation of multiple mobile robots, especially heterogeneous configurations. The abstraction allows engineers to focus on the development and integration of: 1) robot localisation, navigation and obstacle avoidance [44], and 2) high-level cooperation algorithms [45]. Hence the intricacy of the underlying control of hardware devices is avoided since it is managed by the middleware.

2.2.1 Middleware roles

The choice of robotic middleware can be a challenging task at the start of a design process as there is a myriad of platforms available. Investigating the possible use of each platform requires an understanding of the various roles provided by a middleware service; the following middleware roles are discussed in literature:

- *Support communication and interoperability* [46]: Since robots and devices are designed by different manufacturers, communication and simple interoperability mechanisms are required.
- *Automatic discovery and self-configuration* [46]: The mobility of robots makes them dynamic systems and given that they operate in flexible environments (e.g. FMS), automatic discovery and configuration of external devices may be required to enhance efficient operations.
- *Hardware abstraction* [42]: The heterogeneity of low-level robot devices such as sensors and actuators must be masked through the use of software drivers and standardised interfaces.
- *Software modularity* [42]: A library of device drivers and interfaces provided by the middleware platform will make software development more convenient, portable and scalable.
- *Cooperation operations* [46]: The functionality to integrate high-level robot decision making applications is necessary for the development and use of cooperation mechanisms.
- *Self-operable* [44]: The middleware layer is required to operate with the low-level software tiers without the direct interference from the user.
- *Simulation component* [47]: In order for the fast prototyping of solutions, the middleware must provide a simulation component or the functionality to plug-in a simulation service.

The following section investigates the application of these roles in some of the common middleware platforms that are adopted in MRS networks.

2.2.2 Middleware platforms

The Internet Communications Engine (ICE) [48] is a robotic middleware built for distributed systems, using an object-orientated approach. The communication protocol involves the use of ICE objects that reside in local or remote robots, an interface, and clients. Objects respond to client commands by adhering to the interface's definition of behaviours. ICE is characterised for multi-platform and multi-language support, and its application was seen in the development of the Multi-Robot Task Module (MRTM): a module created with the task of robots helping each other by transparently including the behaviours performed by others into its own set of behaviours [49]. In the MRTM system, each robot runs an instance of MRTM which allows for the access of behaviours of other robots in the network, but does not necessarily implement them.

The Physically Embedded Intelligent Systems (PEIS) Kernel [50] provides self-configuration of PEIS devices in the network. Devices are permitted to dynamically join and leave the Kernel since they are all connected to a uniform communication model. The motivation behind the PEIS middleware was to use smart robot technology in simple components, functioning in smart environments, and establish cooperation through these components rather than employing expensive, monolithic robots. A wide range of PEIS experiments were conducted to prove the utilisation of different robotic components in various tasks [51], [52]. Another middleware that allows for automatic discovery and configuration is the Universal Plug and Play (UPnP) Robot Middleware [53], developed by the Korea Institute of Science and Technology. The UPnP middleware enables robots in the network to discover and interact with devices such as cameras and sensors.

The Miro [54] middleware, like ICE, was developed using an object-orientated approach but adhering to the Common Object Request Broker Architecture (CORBA) standard, a framework for developing and maintaining distributed software systems. An application of Miro was seen in the research of heterogeneous mobile robot cooperation in search and rescue missions [43]. The research used the capabilities of different robots in the network to divide the search space by sharing local maps and sensor data, and also involved the development of pyMiro, a Python binding for Miro.

The Cooperative Intelligent Network Management Architecture (CINeMA) middleware was developed to also function in robotic search and rescue disaster missions [55]. CINeMA ensures network connectivity and localisation through robot cooperation in disaster areas of collapsed buildings or underground, where the 1) radio signal strength is low due to the absorption and reflection of signals, and 2) GPS signal is affected. The approach used by CINeMA is to monitor the RSSI (Received Signal Strength Indicator) and LQI (Link Quality Indicator), and cooperatively move robots within a safe region of communication through the application of Enhanced Kalman Filters (EKF) and k-NN algorithms, based on the signal strength. The middleware also supports the re-establishment of a ‘lost’ robot’s localised position by using a two-way ranging (TWR) technique.

ARCADE (Architecture for Real-time Control and Autonomous Distributed Execution) [56] is a platform that can be integrated with ICE to transfer data between real-time databases (RTDB) so that information can be exchanged between robots in the team. The RTDB used in ARCADE is KogMo-RTDB, which was originally developed for autonomous cars [57]. The purpose of the RTDB is to ensure the real-time management and exchange of data objects during communication, and it was shown to handle large, complex datasets while maintaining reliable real-time performance [58].

CoHoN [59] was developed to work with heterogeneous communication hardware using a small-sized data packet in the communication, which benefits communication bandwidth and efficiency. The communication protocol involves a topic based publish/subscribe method, and the message selection process is constructed through a multicast tree. The subscriber broadcasts message requests and will receive responses from nodes with the relevant data; the subscriber then selects the best node to receive data streams from. The advantage of this type of communication is the distributed characteristic to discover all of the available data routes, and the avoidance of a centralised routing option.

A middleware that supports the flexible inter-operability of components to simplify the software development and integration process is the Mobile and Autonomous Robotics Integration Environment (MARIE) [60]. MARIE was developed for integrating both new and

existing software components in robotic networks, and provides flexibility by allowing for the adaptation of different applications and communication protocols.

The Player Project [61] is an open source robotic network server that provides a client with interfaces to communicate with the sensors and actuators of a robot through Transmission Control Protocol (TCP) sockets. The server contains a large software library for device driver implementations and can interface with Stage – a two dimensional simulation package that can be used to prototype the software development process. Player can be installed on a computer that is directly connected to the robot hardware, and since it is a server operating with TCP socket communication, multiple clients (residing on local or remote computers) can connect to Player via different sockets and thus have access to the robot hardware on a particular node. The client application programs can be written in any programming language that supports TCP socket communication; these include C, C++, Java, and Python. Player was used by Naidoo et al. [12] to establish MMRS cooperation in an application that involves the reduction of bottlenecks in manufacturing systems.

In support of its technology programs (Mars and Intelligent Systems), NASA implemented the Coupled Layer Architecture for Robotic Autonomy (CLARAty) [62]. The middleware was developed through collaboration between NASA, the California Institute of Technology's Jet Propulsion Laboratory, Carnegie Mellon University, the Ames Research Centre, and the University of Minnesota. CLARAty's architecture comprises of a functional layer and a decision layer. The functional layer includes components such as digital and analogue I/O, motion control, mapping, navigation, vision, planning, terrain evaluation and simulation [62]. The decision layer is responsible for robot reasoning, it thus plans, schedules, executes activity plans, and dynamically modifies sequences if required [63]. The communication between the functional and decision layers is achieved through a client-server and publisher/subscriber method.

ASEBA is an event-based architecture that was developed to improve the efficiency and scalability of low-level robot control [64]. The middleware is implemented in a robot system that contains many microcontrollers, each being connected through a shared communication bus. Software tasks are distributed to all microcontrollers, and only the relevant data is

communicated to the main processor; in this way, modularity and efficiency is maintained [64]. Each microcontroller can communicate with sensors and actuators, and even process data with some low-level control, due to the powerful capabilities of modern microcontrollers. ASEBA hence improves real-time robot performance but also increases software complexity; although the developers have tried to reduce this drawback by providing data abstraction mechanisms and an Integrated Development Environment (IDE) to develop and debug the robot control software [64].

The Carnegie Mellon Navigation (CARMEN) platform is an open-source toolkit for robot control software [65], that comprises of three layers: 1) the base layer, which masks the low-level details of the hardware, 2) the navigation layer provides localisation, motion planning, and dynamic object tracking functionality, and 3) the user-level layer, reserved for tasks that can be developed by users to implement the functionality from the navigation layer. CARMEN also provides a simulation environment and the programs that were developed in this middleware have proven to be robust to a diversity of failures, as investigated by Montemerlo et al. [65].

Another middleware platform that was created as a toolkit, to be used by students, is Pyro (Python Robotics) [66] – an open-source software that uses a Python-based robotics programming environment. Pyro comprises of multiple modules such as finite state machines, direct control, behaviour-based control, reactive control, and fuzzy logic. It can also be integrated with several robot simulators, viz. Stage, Gazebo, Robocup soccer, and Khepera [67]. The communication protocol in Pyro is structured on a client-server approach, and the software modularity allows for inter-changeability of user programs between small and large robotic systems [67].

Orca [68] is an open-source platform which applies the principles of Component-Based Software Engineering (CBSE) [69] to develop component-based robotic systems. CBSE provides software developers with opportunities to use existing plug-in components, rather than develop modules from first principles. This approach offers tremendous software engineering advantages since it enforces modular systems and thus reduces maintenance costs, and increases flexibility and robustness of the complete system. Orca uses the ICE

middleware as its core for communication between components; this enables the platform to function as a distributed system, for example, two Orca components can communicate between each other, each written in different programming languages, running on two separate operating systems [68]. An application of Orca was seen in the DARPA Grand Challenge III [70], an autonomous ground vehicle competition, where the platform was implemented on one of the teams system.

OROCOS (Open Robot Control Software) [71] is an open-source middleware that adheres to the CORBA standard. The aim of OROCOS is the development of a modular platform for robot and machine control. OROCOS supports four C++ libraries, as shown in Figure 2.5 [72]: 1) the OROCOS Real-Time Toolkit (RTT), provides the infrastructure and functionality to build applications in the C++ programming language, 2) the components library, containing components for management and control, 3) the kinematics and dynamics C++ library, which allows for the calculation of kinematic chains in real-time, and 4) the Bayesian filtering library provides algorithms for the application of Kalman and Particle Filters. An OROCOS application can be built using pre-defined components that are contributed by the robotics community, or developed from the beginning by using the RTT.

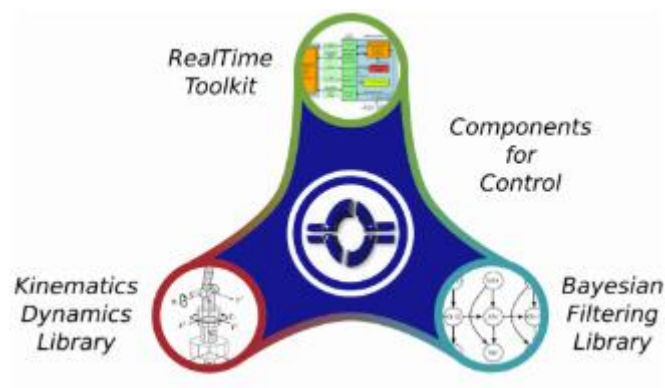


Figure 2.5. OROCOS libraries.

The Microsoft Robotics Developer Studio (MSRDS) was developed by Microsoft to enable users to program robots in the Windows-environment [73]. The motivation behind the development was to implement Service-Orientated Architectures (SOA) into the area of robotics. The SOA programming approach allows users to directly interchange services so

that a desired behaviour can be achieved [74]. The MSRDS consists of four components in its architecture [73]:

- The *concurrency and coordination runtime library* provides classes and methods for assistance with concurrency, coordination and failure handling, and provides the functionality to write code segments that operate independently.
- The *decentralised software services library* allows applications to run multiple independent services in parallel.
- The *visual simulation* component is a 3D simulation environment and includes the functionality to perform physics simulations. The environment supports both indoor and outdoor scenes, and has a variety of simulated robots such as LEGO NXT, iRobot Create, a KUKA robotic arm, and a Mobile Robots Pioneer 3DX, amongst others.
- The *visual programming language* allows users to graphical define the way in which data should be transferred in the application.

In addition to the components mentioned above, MSRDS includes a variety of services and packages that are configured to work on specific robots [73].

A recently popular middleware used in the robotics community is the Robot Operating System (ROS) [75], which uses a client-server method for control flow and publisher/subscriber approach for data flow. ROS nodes communicate with each other in a synchronous form or an asynchronous peer-to-peer (P2P) manner by publishing messages and subscribing to published messages. It has been developed in a modular architecture so that software packages can be easily integrated into the ROS framework. A wide variety of ROS modules are available for various robotic operations such as robot path planning, navigation, obstacle avoidance, and simulation tools, amongst others [76]. ROS has become a popular choice of middleware for the following reasons:

- ROS is an open source software package.
- A large software library exists for device driver implementations.
- Commercial robots are now being powered by ROS [16].
- The ROS industrial consortium has propagated its use in industry [17].
- ROS is actively developed and updated.

- Fault tolerance solutions exist in ROS due to the isolation of individual nodes [42].
- Client applications can be programmed in various programming languages.
- Due to its popularity, a great measure of help is available through community forums and wiki-tutorials.

The comparison between each middleware platform discussed in this section is summarised by Table 2 and Table 3 [42].

Table 2. Middleware comparisons (part 1)

Middleware attributes	ICE	PEIS	Miro	CINeMA	ARCADE	CoHoN	MARIE	Player
Control model	Centralised	Decentralised	Event driven	Hybrid	Centralised	Message orientated	Centralised	Not applicable
Fault tolerance	No explicit fault handling	No explicit fault handling	No explicit fault handling	Yes	Yes	Yes	No	No explicit fault handling
Standards/Technology	CORBA	P2P	CORBA	OPRoS	ICE	Publish/subscribe	Interoperability Technology	3-tier architecture
Open-source	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Real-time	Yes	Yes	No	Yes	Yes	Yes	No	No
Windows	Yes	No	Yes	No	No	No	No	Yes
Linux	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Simulator	No	No	Yes	No	Yes	No	Yes	Stage, Gazebo
Distributed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Security	Yes	No	No	Yes	Indirectly supported	No	No	Yes

Table 3. Middleware comparisons (part 2)

Middleware attributes	CLARAty	ASEBA	CARMEN	Pyro	Orca	OROCOS	MSRDS	ROS
Control model	Centralised, decentralised, event driven	Event driven	Hybrid	Independent	Not applicable	Event driven	Distributed messaging	Message orientated
Fault tolerance	Yes	No	Yes	No	No explicit fault handling	No explicit fault handling	Yes	No explicit fault handling
Standards/Technology	Object Orientated	Event-based	TCP, IPC	TCP, XML, HTTP, OpenGL	ICE	CORBA	.NET/SOA	Message, RPC services
Open-source	Partially	Yes	Yes	Yes	Yes	Yes	Commercial	Yes
Real-time	Most modules	Yes	No	No	No	Yes	No	Yes
Windows	Only cygwin	No	No	Yes	Yes	Yes	Yes	Partial functions
Linux	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Simulator	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Distributed	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Security	Yes	No	Yes	Yes	No	No	Yes	No

2.3 Smart factories

This section begins by introducing the Industry 4.0 concept, and discusses the characteristics and requirements of the model in an industrial setting. Machine-to-Machine (M2M) communication and Advanced Robotics are also discussed, since they are considered as key pillars of Industry 4.0.

2.3.1 An introduction to the Industry 4.0 concept

In the recent past, industrial processes began to implement information technology (IT), but the current global trend goes beyond this due to the technological advancement of electronics and the internet. The term ‘Industry 4.0’ originated in Germany, although its concept is largely adopted in other countries, also known as the *Industrial Internet of Things*, *Smart factories*, or *Advanced manufacturing* [7]. The notion that these terms have in common is that the traditional manufacturing and production methodology is at the cusp of a digital transformation. The present increase in the use of information and communications technology (ICT) in manufacturing, together with the implementation of ‘intelligent’ embedded sensors and devices has realised an integration between the real-world and virtual-world, known as the cyber-physical production system (CPPS), depicted in Figure 2.6 [77].

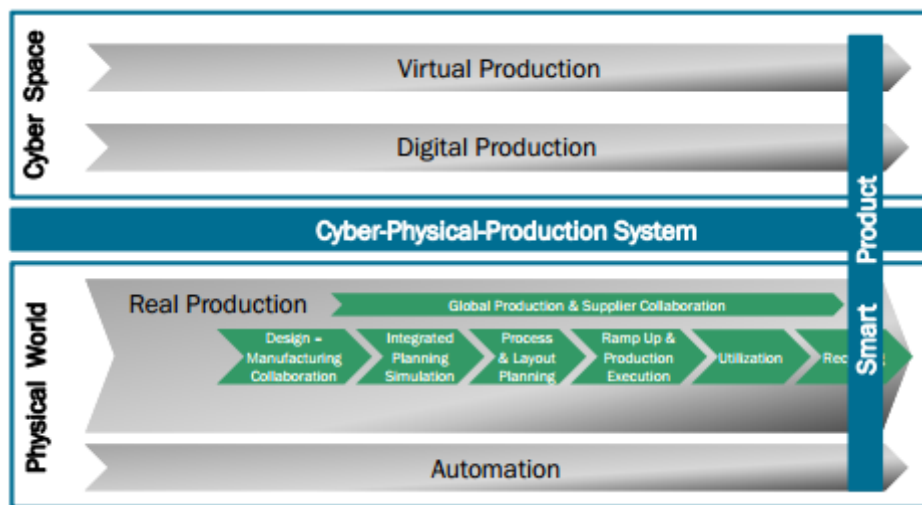


Figure 2.6. The cyber-physical production system.

2.3.1.1 Cyber-physical production systems

CPPS’s are machine networks that are organised in a similar fashion to social networks. They link IT with the mechanical and electronic components of a factory, which enables communication between the components through the network. In this way, machines can share information about faults, stock or demand levels, and changes in customer orders,

which can essentially lead to the coordination of processes and production targets to meet deadlines. Apart from the utilisation of ‘smart’ machines, CPPS’s allow for the networking of these machines and smart products to the entire supply chain, thus building smart factories that are able to produce products based on consumer demand. An illustration of this concept is given by Figure 2.7 [78].

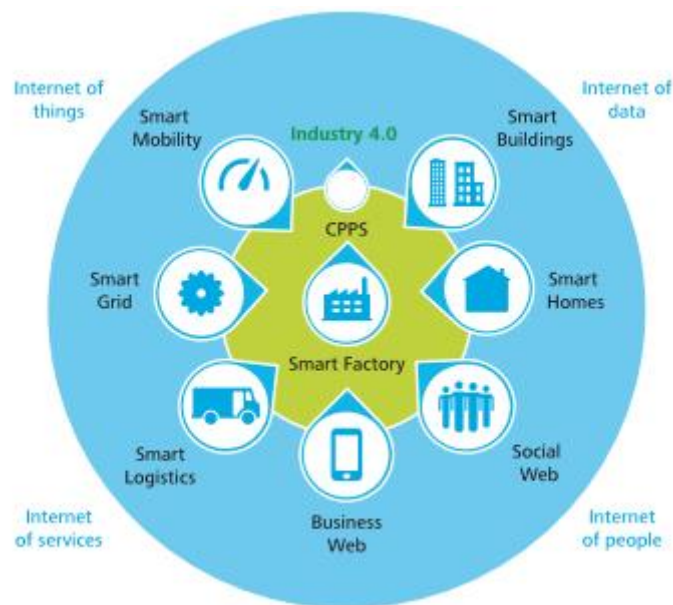


Figure 2.7. Industry 4.0 environment.

2.3.1.2 Characteristics and requirements

In their report [79], Schlaepfer et al. discuss four main characteristics of Industry 4.0, and how these reveal the capacity for change in traditional manufacturing environments:

- *Vertical networking of smart production systems* involves the use of CPPS’s to enable factories to dynamically react to changes in customer demand or stock levels, and establish production systems that are customer-specific and individualised. Another contribution of CPPS’s in vertical networking is maintenance management and waste reduction; since resources and products are networked, variations in quality and

machine breakdowns can be attended to more rapidly, thus optimising the use of resources.

- *Horizontal integration* is the networking of inbound logistics, warehousing, production, marketing, sales, and outbound logistics, all of which is possible through the CPPS, ensuring global optimisation. The continuous traceability of a product (also known as ‘product memory’) can be achieved, allowing for the dynamic mass customisation of products at various points in the production process.
- *Through-engineering* occurs seamlessly during the design, development and manufacturing of new products due to the availability of engineering data at all stages in a product’s life cycle.
- *Exponential technologies* such as sensor technology and nanotechnology accelerate individualised solutions, flexibility, and cost savings in manufacturing processes. An example of this can be seen in the field of mobile robotics, where artificial intelligence (AI) and advanced sensor technology can increase autonomy, leading to more flexibility in manufacturing processes.

The requirements of a smart factory environment involve a variety of technological developments, these include [7]: 1) ICT to digitise information; 2) CPPS’s that use ICT for monitoring and control of physical processes, an example can be intelligent robots that are able to reconfigure due to dynamic product specification changes; 3) distributed network communications to link resources, machines (or robots), products, people, and systems; 4) simulation and modelling tools for product design; 5) data collection, big data analysis and cloud computing.

2.3.2 M2M communication and Advanced Robotics

Research identifies, amongst others, Machine-to-Machine (M2M) communication and Advanced Robotics as key technology enablers, or pillars in the Industry 4.0 concept [18]. Figure 2.8 [80] illustrates the concept of M2M communication in smart factories. The networking between machines and decentralised control at the machine allows for the possibility of dynamic product changes to be made at the source. Figure 2.8 also gives a

holistic view of the discussion in the previous section, relating to the networking of smart materials, machines, products and cloud networks for vertical and horizontal benefits.

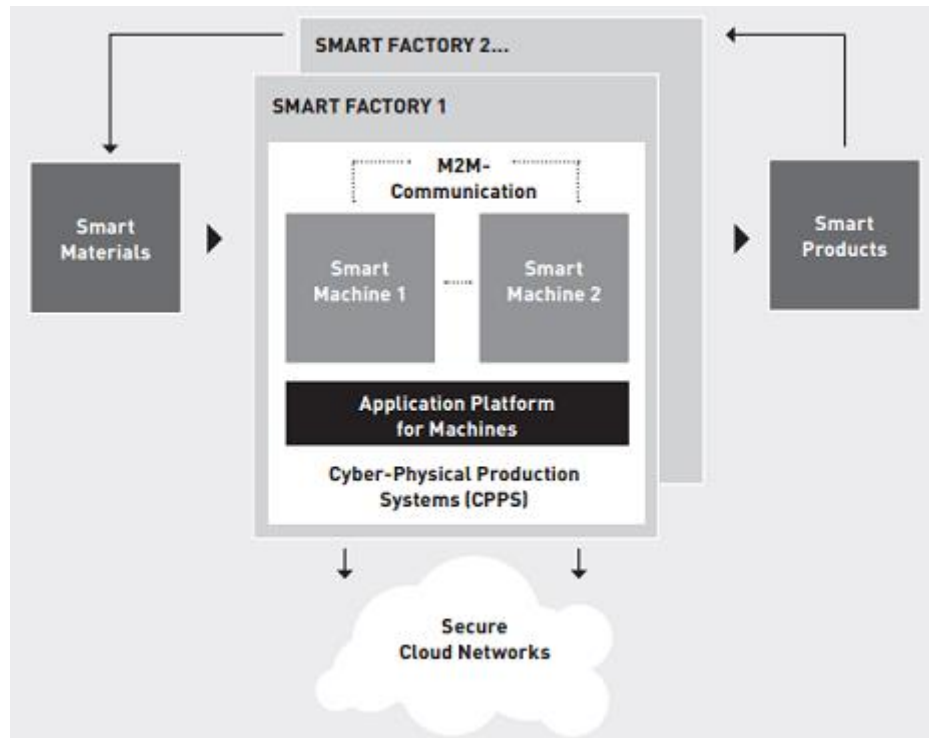


Figure 2.8. M2M communication in smart factories.

The British Prime Minister, Theresa May, announced in January 2017 that Industry 4.0 was one of the five areas of focus in a plan by the British government to boost the economy [13]. Previous to this, in November 2016, the British government announced a £4.7 billion budget for research and development into four areas, two of which are robotics and artificial intelligence [13]. Figure 2.9 [15] is a good representation of the impact of robotics on the industrial revolutions. Robotic mobility (3rd robotic revolution) and intelligent robot systems (4th robotic revolution) allow for decentralised control, which fits in directly with the Industry 4.0 concept. Decentralised control, configurable robots, and robot cooperation will increase flexibility in the smart factory and enable the ability to rapidly reconfigure systems to adapt to variations in consumer product demand [7].

Future trends in mobile robots reveal that in 2020, 26 million mobile robots will facilitate autonomy in smart factories, unmanned transportation, and connected homes [15]. Mobile

robotics and cooperative robotics are also considered as ‘topics of the future’ in industry and institutions of higher learning [14].

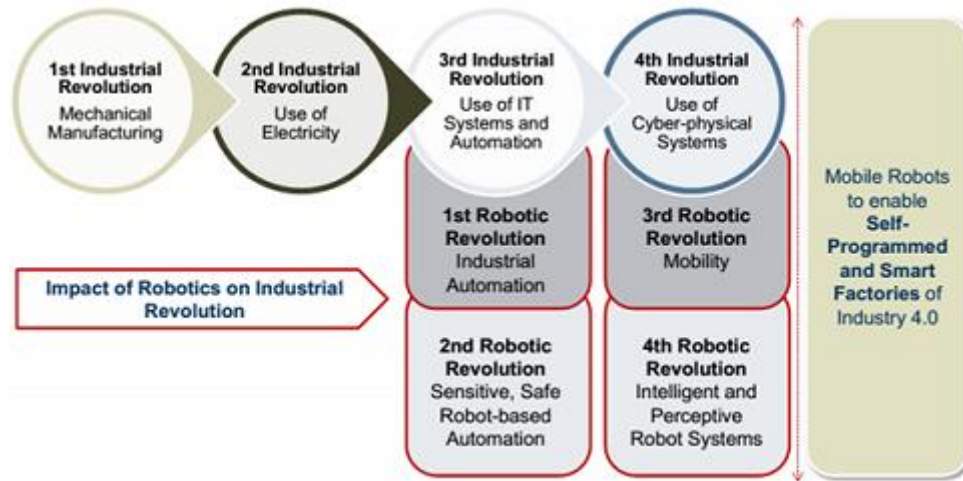


Figure 2.9. Robotics and Industry 4.0.

In order for robots to be easily integrated into manufacturing processes the issue of robot communication protocol standards must be considered [7]. If the standards are proprietary or only adapted nationally, manufacturers will be forced to use equipment supplied by a few companies, which can increase costs and limit the adoption of Industry 4.0 due to the lack of flexibility in the system [7]. Hence, international standard communication protocols and interoperability across robotic interfaces are essential to the implementation of robotic systems in smart factories.

2.4 Chapter summary

The literature survey investigated the role of mobile robot systems in industry; associated topics such as heterogeneity, robot communication, and middleware platforms were also discussed. The survey concluded by discussing the characteristics and requirements of a smart factory environment as well as the influence of advanced robotics in these environments.

Chapter 3 Design of the framework

The aim of this chapter is to present the methodological approach taken during the design of the framework. The first section outlines the requirements of the framework and discusses some components of the preliminary design. The subsequent sections explain the design choices made over other possible solutions and, in some cases, background knowledge about specific topics are given to enhance the reasons for the corresponding design decisions. The chapter concludes by discussing the presentation of a detailed design overview, together with design specifications and assumptions.

3.1 Design requirements

The preliminary design overview is represented by Figure 3.1. The components in the figure are discussed in the following list, which outlines the important requirements of the framework to function in a smart factory environment:

- The *robot middleware* is one of the essential components of the framework since it masks the differences of the heterogeneous robots in the network, thereby establishing common interfaces to seamlessly communicate with each robot.
- The *local control program* is necessary for indirectly communicating with the robot hardware through the interfaces provided by the middleware layer. Local control is required for fast, real-time robot functions such as obstacle avoidance and local navigation.
- The application of an *international standard data communication protocol* will promote flexibility of the framework for integration into a smart factory environment. Interoperability across robot interfaces, as well as the other components, is also required for successful system integration.

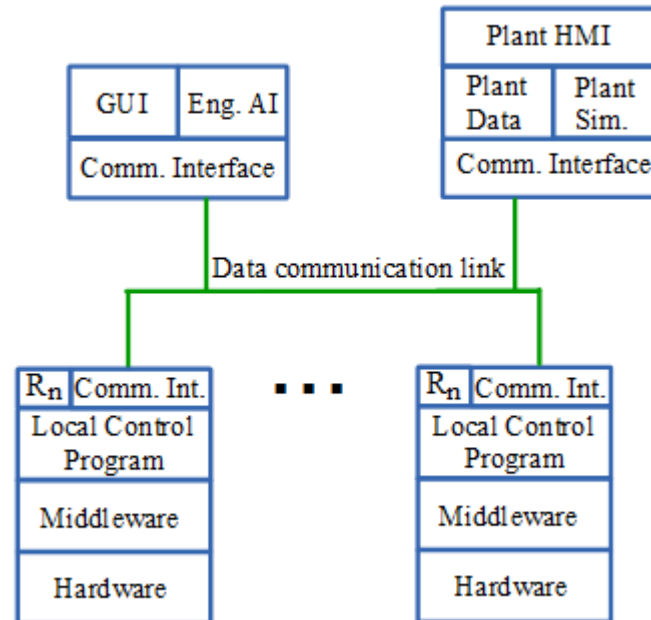


Figure 3.1. Preliminary design overview.

- A *graphical user interface* (GUI) assists the user in configuring robots or tasks in the network. The GUI communication interface will allow for a connection to any remote robot in the network.
- The engineering *artificial intelligence* (AI) component allows for the programming of intelligent user algorithms that will create a network of smart mobile robots, responding to specific states in the manufacturing process.
- The communication architecture is a *distributed system*, thus enabling flexible and fault-tolerant solutions: the GUI and AI components are not limited to fixed locations in the network.
- A *hybrid control architecture* is required for the control of mobile robots in the team. Decentralised control is necessary for the local, real-time robot response to obstacle avoidance and navigation, particularly in a dynamic manufacturing environment that comprises of moving machinery, people, and other mobile robots. The centralised control part of the hybrid scheme involves the high-level, decision making process which does not require real-time responses from the robot. The single point of failure

in the high-level control can be eliminated by the use of a redundant system, located at another point in the distributed network.

- *Plant simulated data* is required for the testing and simulation of the framework's functionality.
- A *human-machine interface* (HMI) is necessary for the display of plant essential data as well as the current status of each robot in the team.

3.2 ROS middleware

Section 2.2 discussed a comprehensive survey on the various robotic middleware platforms. The Robot Operating System (ROS) was chosen to be implemented among them due to the following reasons:

- ROS is a free, open source installation.
- ROS installs on the popular Linux Ubuntu distribution, and is actively developed and updated.
- There is an availability of a large software library for device driver implementations.
- ROS is a distributed system, thus software nodes and user applications can run on different machines and they can communicate with each other.
- The modular design of ROS allows for easy integration of additional functionality [81].
- Commercial robots are powered by ROS [16].
- ROS has been introduced into industry through the ROS industrial consortium [17], and is backed by well known industrial players such as Yaskawa, ABB, BMW, and Siemens, amongst others.
- Applications can be programmed in various programming languages, viz. Python, C++, Lisp, and Java.
- Due to its popularity, a great deal of help is available through wiki-tutorials, community forums and research papers.

The ROS middleware consists of nodes, messages, topics and services. Nodes communicate with each other by passing messages, which are made up of data types, including nested structures and arrays. The ROS nodes can communicate in two ways: through a synchronous *service*, or asynchronous *topic*. Services behave like function calls in conventional programming languages, and they are defined by specific names, provided by a node. A request to a service will result in a response with data of a specific data type (e.g. boolean, string, integer, etc.). Topics involve the streaming of data by a particular node. In this scheme, the communication between nodes occurs in a peer-to-peer (P2P) manner by publishing messages and subscribing to published messages. An initial event called the ‘naming service’ is centralised and relies on a master node, as shown in Figure 3.2 [47].

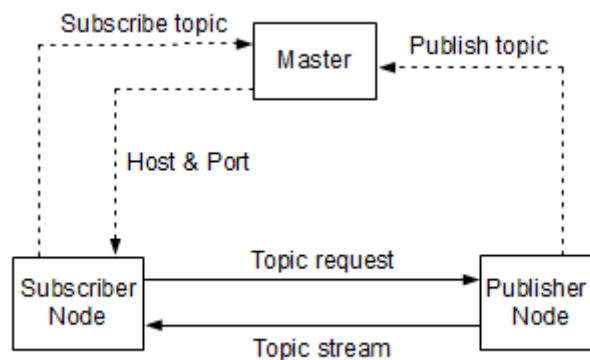


Figure 3.2. Topic-based communication in ROS.

The communication sequence between a publisher and subscriber involves the following steps [82]:

- 1) The publisher node registers (publishes) the topic (e.g. a laser scan) to the master node, also referred to as the naming server, and informs the master about the entry point of topic data.
- 2) The subscriber node queries the master on access to the particular topic.
- 3) The master sends a response to the subscriber with entry point data, such as the host address and port number.

4) The subscriber now directly communicates with the publisher (host) via Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) connections, requesting for topic data.

5) The publisher responds to the subscriber by sending the topic data stream (e.g. laser scan data).

In relation to the design of the framework in this research, each remote robot in the network was powered by the ROS middleware and consists of master and subscriber/publisher nodes. The reason for the use of multiple masters in the multi-robot system was to enhance the reliability of the system and eliminate the single point of failure. Also, individual masters mean that each robot has direct, real-time control over its sensors and actuators, which is a necessity in both single and multiple robot systems.

The ROS robot simulation package implemented in the design and development of the framework was *Stage* – the two dimensional simulator. Stage provides interfaces for the simulation of robot sensors and actuators, without the actual physical hardware. Robot attributes like size, colour, and shape can be created in model files, and a map of the environment can also be loaded as an image into the Stage ‘world’. This functionality allows for the fast prototype testing of robot behaviours in various scenarios without using the actual robot hardware.

3.3 Industrial data communication

The data communication standard applied in an industrial setting is crucial to the interoperability of devices in the network. Common communication interfaces across all devices are necessary in order to maintain flexibility and robustness in the system, and as discussed, it is a requirement for the adoption of a smart factory concept. Before discussing the data communication standards applied in industry and the option chosen in the design of the framework, a review of the basic industrial communication model is required.

3.3.1 Communication model

The traditional hierarchical model of the communication and control system approach adopted by most manufacturing operations is given by Figure 3.3 [83].

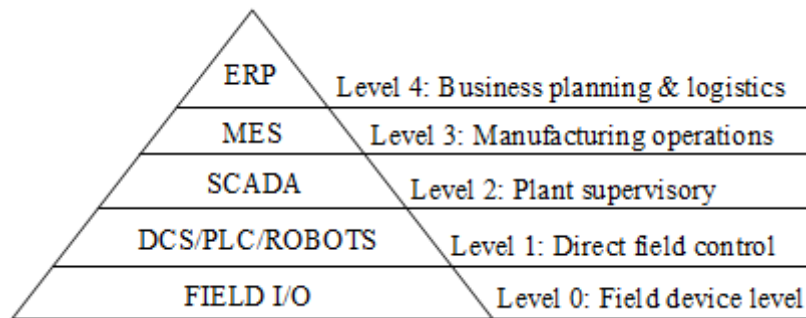


Figure 3.3. Hierarchical communication and control model in manufacturing plants.

The hierarchical structure in Figure 3.3 begins from the low-level, real-time control response and monitoring of Input/ Output (I/O) field devices (e.g. sensors, actuators, digital I/O), to the slow response, higher-level control of the manufacturing processes. The Distributed Control System (DCS) and Programmable Logic Controller (PLC) units have the function of controlling a variety of electrical, mechanical or process equipment through data signal processing and the execution of industrial computer programs. PLC's contain Central Processing Units (CPUs) that operate at fast program cycle times. In high-speed manufacturing plants, there is a greater demand for extremely fast program cycle times, to monitor and control real-time operations. Robots also form part of "Level 1" (in Figure 3.3) due to the control of its own hardware through the use of its on-board microprocessor.

The Supervisory Control and Data Acquisition (SCADA) system retrieves data from level 1 and presents it in the form of status messages, graphics, and trends for the plant operator or engineer to analyze and control equipment on the factory floor. A level higher sees the Manufacturing Execution System (MES), or in some applications the Manufacturing Resource Planning (MRP) system, whose function is the management and scheduling of engineering resources and processes to optimise productivity on the factory floor. The MES also bridges the gap between the manufacturing process and the business management level of the company.

The highest level in the hierarchy in Figure 3.3 is the Enterprise Resource Planning (ERP) tool, which is a business management software package that specialises in: 1) the planning of production, product demand and supply chains, and 2) the management of material, warehouse logistics, maintenance activities, and human resources. ERP systems are value added services to the business as they assist senior managers with meaningful data and tools to efficiently manage areas of the business.

3.3.2 Data communication standards

Data communication technology in industry was originally developed on serial-based interfaces by different companies, which later became standards. Some of these standards include PROFIBUS [84], CANbus [85], Modbus [86], and CC-Link [87], among many others. PROFIBUS has been the most successful serial bus technology [88], and is still widely adopted in industry today. In the modern era, Ethernet-based communication protocols are becoming very popular due to the increased speed offered by the network architecture and the cost effective solutions that exist [88]. These networks integrate well with the Industry 4.0, smart factory concept, primarily attributed to the real-time communication ability and ease of installation of the Ethernet infrastructure. Many Ethernet-based communication protocols exist, the most popular ones are Ethernet/IP [89], PROFINET [84], and EtherCAT [90].

Ethernet/IP is an industrial Ethernet protocol, which uses the first four layers of the Open System Interconnection (OSI) model, as shown in Figure 3.4. The generic OSI model was created by the International Standards Organisation (ISO) in 1984 and defines seven layers that describe guidelines for interoperability between various devices in the network. Ethernet/IP uses the Common Industrial Protocol (CIP) over TCP/IP, which resides in the fourth (Transport) layer of the OSI model. CIP provides a common set of services and messages for industrial control systems, and multiple CIP connections can be made over one TCP connection. An unlimited number of nodes can be connected in an Ethernet/IP system due to the use of the standard Ethernet infrastructure and switches. The protocol has proven to provide very efficient slave peer-to-peer (P2P) communications [88], although one of the drawbacks of Ethernet/IP is its limited real-time capability.

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data link
1	Physical

Figure 3.4. The 7 layers of the OSI model.

PROFINET is a popular industrial Ethernet protocol and is widely used by system integrators and manufacturers such as Siemens. It consists of three classes [88]:

- *Class A* provides a link to the PROFIBUS network through proxies and remote procedure calls on the TCP/IP layer. It is mainly used for parameter data and cyclic I/O since the cycle time is around 100ms.
- *Class B* is also known as PROFINET Real-Time (RT), with cycle times around 10ms, providing a more real-time communication interface. Typical applications using Class B include factory automation and process automation.
- *Class C* is also known as PROFINET IRT (Isochronous and Real-Time) and requires specific hardware to produce cycle times of 1ms, particularly used in high-speed and motion control applications.

EtherCAT is standardised to operate on the second layer of the OSI model, thus since processes are handled in hardware, each node in the network introduces a minimum processing latency, enabling fast Ethernet speeds of up to 100Mbits/sec. The EtherCAT protocol uses standard Ethernet frames, as shown in Figure 3.5 [91], where the protocol data consists of the EtherCAT header (red) and 1 to 15 datagrams (green). The protocol can connect up to 65535 slave nodes in a system. Messages are issued by the EtherCAT Master with data for all nodes; as the message is transmitted from and back to the Master, each slave

node processes its datagram and inserts new data into the frame as it passes through (also termed ‘pass-through reading’). This form of communication allows the EtherCAT network to achieve maximum bandwidth utilisation.

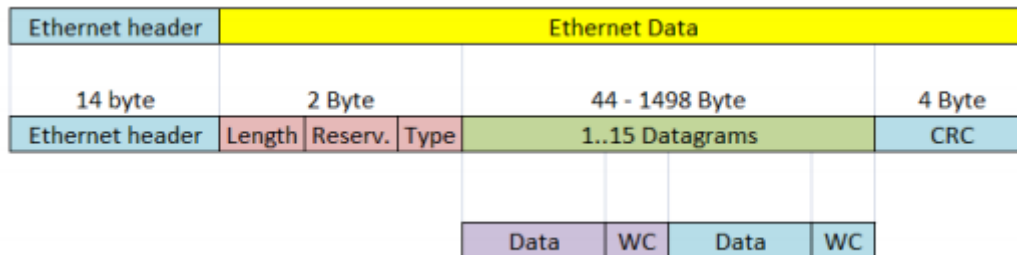


Figure 3.5. Ethernet frame with EtherCAT data.

The following list is a summary of the key benefits of the EtherCAT protocol [92]:

- EtherCAT is a free and open protocol.
- Fast, deterministic, with very short cycle times.
- International Electro-technical Commission (IEC) standard.
- Supports master-slave and P2P communication.
- Operates on standard Ethernet.
- Slave devices use inexpensive components.
- High precision time synchronisation for high-speed and motion control applications.

Another advantage of the EtherCAT system is that together with OPC-UA (discussed in the next section), it supports Industry 4.0 convergence of information and automation technologies [92].

3.3.3 OPC-UA

One of the key requirements of an Industry 4.0 factory is the interoperability of devices and systems in the network. This can be a challenging task, particularly in networks that contain heterogeneous devices and various communication protocols, like the few mentioned in the previous section. A data communication standard that specialises in device and system

interoperability is the OPC Unified Architecture (OPC-UA), developed and maintained by the OPC Foundation [93]. OPC-UA offers a secure method of server-client communication, having the ability to securely connect through firewalls and VPN (Virtual Private Network) connections. Multiple servers (each connected to their own set of homogeneous devices) can be configured in the network and clients can instruct the server(s) to send data at real-time intervals. The power of OPC-UA is realised when data is shared in networks that contain mixed Ethernet protocols such as Ethernet/IP, PROFINET, and EtherCAT, since these architectures provide OPC-UA servers in order to integrate data into larger systems.

OPC-UA has also made it possible to share data from the factory floor through to the higher-level business systems such as the ERP, illustrated in Figure 3.6. This communication model can be compared with the traditional model (Figure 3.3) discussed earlier, where the scalability of OPC-UA is identified and the integration of devices and systems in a smart factory becomes a reality.

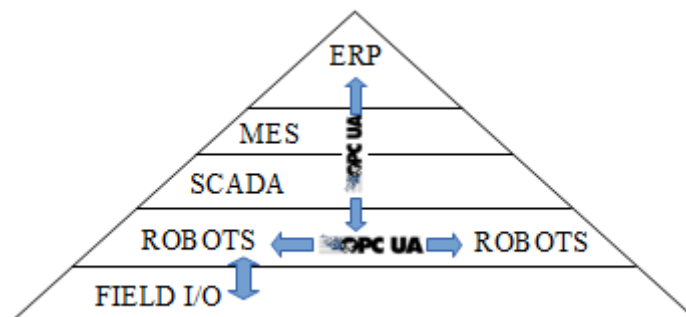


Figure 3.6. Hierarchical communication and control model with OPC-UA.

Tom Burke, the president and executive director of the OPC Foundation, made the following statement about OPC-UA and the Industrial Internet of Things (also known as Industry 4.0) [94]:

“OPC-UA is clearly positioned to be the Industrial Internet of Things infrastructure for moving data from sensors to the cloud. OPC-UA security has been validated and endorsed by many organizations inclusive of BSI in Germany. We are actively working with all of the IoT protocols, adding full support of the IoT protocols directly into the OPC-UA base

architecture... We have an extensible model that will allow us to plug in any IoT protocol of the past and present directly into OPC-UA.”

The interoperability among the layers in the architecture shown in Figure 3.6, and the ease at which the low-level data is made available to the higher level systems in the network, are the reasons why OPC-UA was chosen as the data communication standard in this research. The OPC-UA Server implemented in the framework for data tag configuration and real-time data exchange was Matrikon OPC [95]. The software product was chosen due to its longevity and stability in the industry, as well as its ability to interface with automation and control devices of leading manufacturers [95].

3.4 IDE and GUI

As per the discussion on Figure 3.1, the AI component of the framework is required for the programming of intelligent user algorithms that will create a network of smart mobile robots, responding to specific states in the manufacturing process. This function is made possible through the use of an Integrated Development Environment (IDE), which contains various libraries to assist with the programming of algorithms, interfaces to enable communication with other devices in the network, and a GUI development environment for the design of simple interfaces to assist the user with robot and task configuration. Various IDE platforms were researched; among them were Qt [96], the R-project [97], and Python [98].

Qt is a complete C++ application development environment that provides a rich library of components used for rapid, and aesthetically pleasing, graphic development. It is available as open source or commercial installations on Windows, MacOS, and Linux Operating Systems (OS). Qt can be integrated into an industrial setting through the “QtOpcUa” module, which enables support of the OPC-UA industrial communication standard. The drawback of Qt is the use of the C++ programming language, which can be difficult for the end-user to grasp and seamlessly program algorithms.

The R-project is an excellent tool for statistical computation. The programming environment contains an integrated suite of software functions for data manipulation and storage, and data

analysis. Programs are written in the R-programming language, however, C, C++, and Fortran code can be linked and called at run-time for computationally heavy tasks [97]. The R-project would have been the design choice in this research if it also had the ability to develop GUI's together with industrial communication interfaces (such as OPC-UA) for integration into an industrial robot network.

Python was chosen as the end-user programming language due to:

- its simple object-orientated structure;
- the built-in GUI development platform, named Tkinter, which has a cross-platform ability and contains a stable core (being first released almost three decades ago);
- its powerful mathematical libraries such as Numpy, Scipy, and Scikit-learn [99] each contributing to the depth of the framework in developing advanced algorithms for robot control and cooperation applications;
- the support of the OPC-UA communication protocol through the use of its OPC-UA client library;
- its popularity in the computer programming and data engineering fields, making it a familiar platform for the end-user to work in.

The Python IDE used for the development of the framework was PyCharm (Community Edition), since it is a free, user friendly IDE, and was sufficient for satisfying the objectives of the research study.

3.5 SCADA

The purpose of the HMI, or Supervisory Control and Data Acquisition (SCADA) block in Figure 3.1 is to merely visualize the actual status and values of the plant variables of the application. Several SCADA packages were researched and many proved too costly to be implemented in this research study. From the free-version SCADA packages researched, Ecava IGX [100] was chosen due to it being a web server application, thus there was no need for an additional client program or software plug-in. The only additional requirement was an offline or online internet browser (which is a default installation on most computers) to view

the mimic screen. The free, unregistered version gives the user a two hour period to access the mimic, after which the application shuts down. This limitation was acceptable for the study since in practice, the SCADA was used for research and simulation purposes and can always be restarted after the time period has expired.

Ecava IGX also supports the OPC-UA communication protocol and contains an OPC-UA client; hence it adheres to the data communication design standard of the framework.

3.6 Final design overview

Figure 3.7 gives an overview of the components in the software framework that was designed for the programming of remote mobile robots in a manufacturing environment.

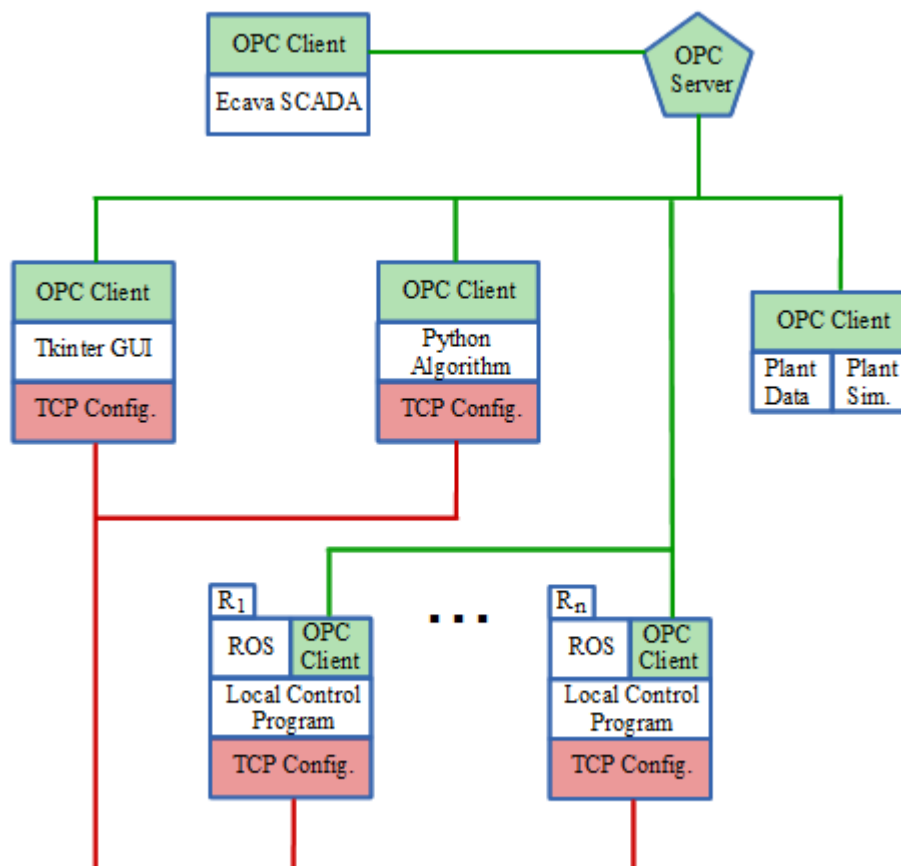


Figure 3.7. Final design overview of the framework.

The green blocks and lines in the figure represent *real-time* OPC-UA data communications, whilst the red structures represent *offline*, configuration related communication. The control architecture of the framework is a hybrid one, where the centralised part of the scheme is seen in the high-level, decision making process involving the Python algorithms and the OPC-UA data communication.

The goal tasks of each mobile robot are achieved through the OPC-UA tag variables, which are configured in the Matrikon OPC-UA Server. The robot motion commands and the mapped environment x and y coordinate points are OPC tag outputs from the Python algorithm – located anywhere in the network. This distributed feature of the framework accommodates for the possibility of having multiple programming stations, each able to do the same task of processing algorithms. The user’s Python algorithm requires plant data, either actual data from a live plant or simulated data, the latter of which was used in the framework development and system tests.

Each robot in the network is powered by the ROS middleware. The decentralised part of the hybrid control architecture is identified by the robot’s local control program, which works in conjunction with the ROS core to perceive data from proximity sensors (such as a laser range finder) and to plot paths to goal locations while avoiding obstacles. The Adaptive Monte Carlo Localization (AMCL) algorithm [101] was used for the path planning and obstacle avoidance routines, since it is one of the ROS packages and produced a good performance in simulation tests.

3.7 Functional design specifications

The following list outlines the functional design specifications, defined in order to fulfil the contributions of the research study:

- Mobile robots operate at linear velocities ≤ 0.2 m/s.
- Mobile robots operate at angular velocities ≤ 0.4 rad/s.
- Range finder sensors are used for distance measurements and are positioned at the front of each robot.

-
- Robots are required to plan and navigate the environment while avoiding obstacles.
 - Required scan cycle time of the robots' local control program is ≤ 100 ms, so that real-time response to navigation and obstacle avoidance routines can be performed.
 - Required scan cycle time for the higher-level decision making algorithms is ≤ 1 sec.
 - OPC-UA tag scan rates are ≤ 500 ms.
 - The complete list of OPC-UA Server tags is accessible by the GUI.
 - The GUI provides an interface for the mapping of OPC-UA tags to local variables in the Python high-level control program.
 - The GUI provides an interface for the configuration of robot parameters (dimensions, hardware, technical limitations), and the configuration of task parameters (robot physical and technical requirements).
 - Remote downloading of robot configuration data is enabled through the GUI.
 - Robots are assigned static IP addresses since they are located on an Ethernet network, and are accessed by the GUI through use of these addresses.
 - No direct inter-robot communication is permitted.
 - The SCADA shows the status of the manufacturing process and the locations of the mobile robots.
 - The use of Python coded machine learning algorithms for high-level robot control is required.

The following design assumptions are made:

- The Stage simulation environment produces neither noise nor dynamic obstacles, except in cases where multiple mobile robots are simulated.
- The static map used for the Stage simulation is a direct representation of the environment.
- Free navigation paths between robot goal locations exist, as per the design of the static map employed.
- Individual robots ignore the existence of other, multiple robots executing the same task.

3.8 Chapter summary

This chapter presented the methodological approach taken during the design of the framework. The design requirements were defined, outlining the key components of the framework that will enable its functionality in a smart factory environment.

The reasons for choosing the ROS middleware as the core platform for each robot were discussed, and a description of the communication mechanism employed by the ROS architecture was described. The section on industrial data communication examined the communication and control models in manufacturing plants, together with the common communication standards employed in industry. The OPC-UA data communication standard that specialises in device and system interoperability was selected as the communication protocol for the framework due to the ‘interoperability’ requirement of Industry 4.0.

The basis for selecting Python as the programming language for creating the GUI and developing user algorithms were discussed, as well as the use of the Ecava IGX SCADA software in the system design. The chapter concluded with a discussion on the final design overview, and the functional design specifications and assumptions.

Chapter 4 Framework development

Following the discussion on the final design overview in the previous chapter, this chapter examines the software development of the framework components. The subsequent two sections are preliminary ones that describe the structure of the approach taken to present the details of the software development.

4.1 Development and verification strategies

The presentation of the software development for the framework involves a structured approach, where the main components of the system are discussed in separate sections. Some discussions will involve the use of flow diagrams to give an overview of the subsequent work, or to describe the sequence of functionality in the development. In the penultimate section, a review of the framework is covered with the purpose of conveying a depiction of the system integration and a summary of the developmental work.

The software development correlates with the verification of the framework, which was done through a case study approach. A series of case studies are performed in the next chapter, and involve different scenarios pertaining to mobile robot material handling applications. Hence, some of the developmental work in this chapter relates to these applications; for example, the SCADA development section will show the creation of screens that contain material storage graphics. Likewise, another section such as *Python programming features*, discuss the development of user functionalities that are applied in the case studies as part of the verification of the framework.

4.2 Development overview

The structural diagram in Figure 4.1 gives an overview of the framework development discussed in this chapter.

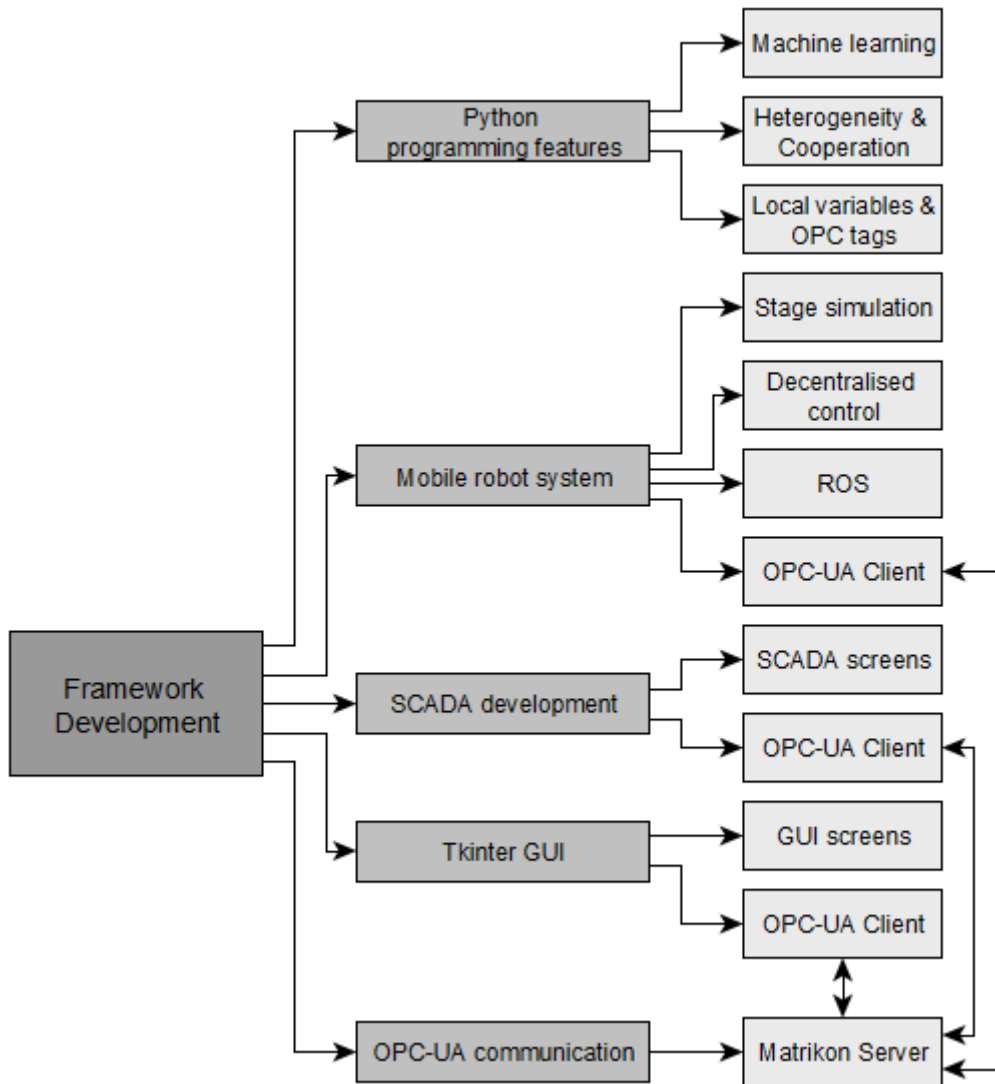


Figure 4.1. Overview of the framework development discussed in this chapter.

The key sections depicted in the figure are summarised below:

-
- *OPC-UA communication* involves the configuration of the Matrikon OPC-UA Server and communication between OPC-UA Clients, viz. Python, SCADA and the mobile robot system.
 - *Tkinter GUI* discusses the development of the graphical screens for user interaction. The OPC-UA client configuration and tag assignments are also discussed here.
 - The *mobile robot system* section details the ROS software implementation, decentralised-local robot control, the Stage simulator integration, and the OPC-UA client configuration.
 - *SCADA development* covers the creation of the mimic screens for the viewing of plant data and locations of the mobile robots. The OPC-UA client configuration for the SCADA node is also discussed.
 - *Python programming features* involves a discussion on the development of three features which all strongly contribute to the contributions of this research study: 1) local variable mapping to OPC Server tags, 2) heterogeneous mobile robot cooperation, and 3) support for machine learning applications in robot control systems.

4.3 OPC-UA communication

The communication arrangement between the Matrikon OPC-UA Server and each Client in the framework is given by Figure 4.2. The alias (or tag) configuration that represent the system variables are executed through the Server, which also holds a real-time database of all tags in the system.

The Matrikon OPC Explorer package is a general purpose OPC Client, provided by Matrikon, for testing the functionality of the system. The value, quality (good or bad), and timestamp data of the Server configured tags can be viewed. This functionality was used to verify the status of the tags before they were actually scanned by the OPC-UA Clients in the framework.

The other remote OPC-UA Clients in the framework, viz. *Ecava IGX SCADA*, *Python Program*, and *ROS Robot*, are each configured to read and/or write values of the OPC tags. The manner in which each client connects to the server to retrieve data is discussed in the following respective sections.

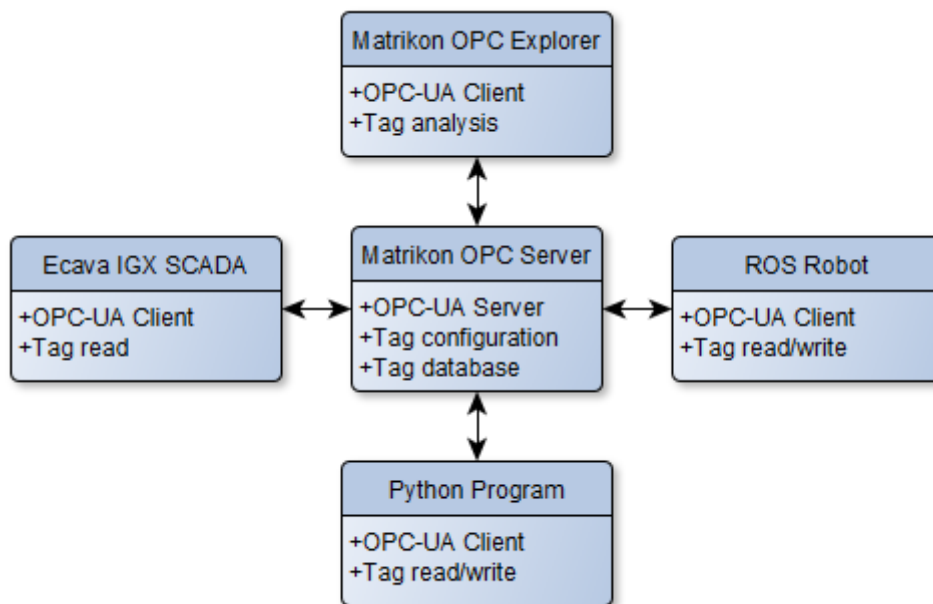


Figure 4.2. Communication between Matrikon OPC-UA Server and Clients.

Figure 4.3 shows a brief process of the way in which OPC tags are created through the Matrikon OPC Server interface. The name of the server was created in the “Alias Configuration” node and thereafter, each tag in the system was configured as a new alias. Tag properties such as *name* and *data type* are mandatory fields. The *update rate* was used in conjunction with the *poll when inactive* property, which if selected, will continue to update the value of the tag even if no OPC clients are accessing the tag at that specific time.

If the *read only* checkbox is selected, OPC clients are restricted from writing values to that specific tag. Tag values can also be scaled through the use of text expressions, calculation, linear, square root, and gain/offset functions, all of which were not used in the development of the framework.

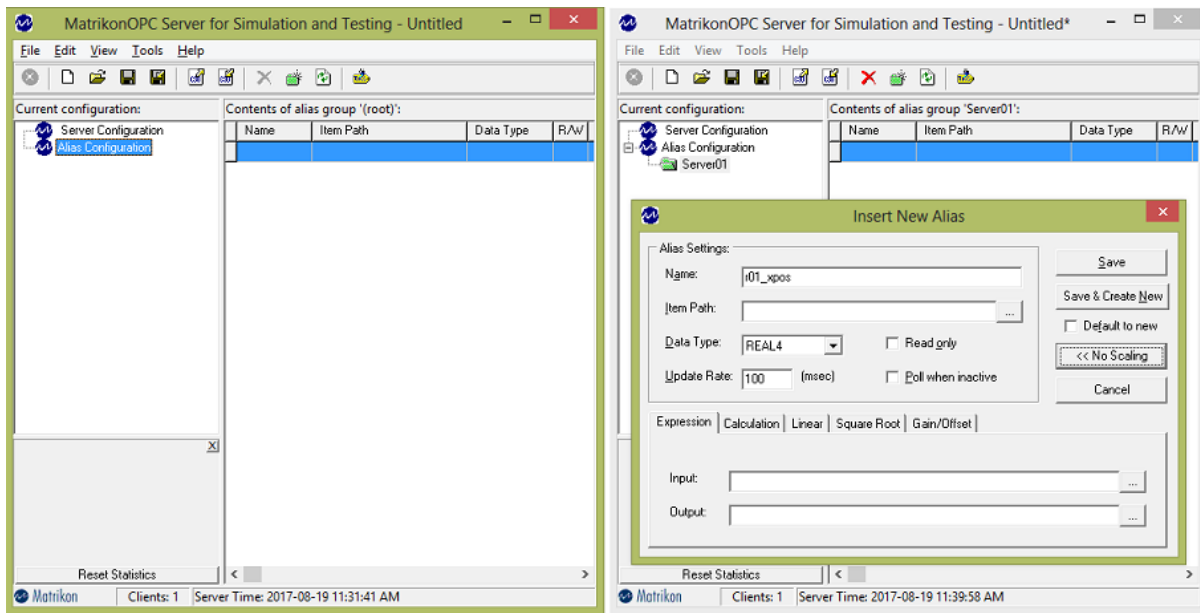


Figure 4.3. Matrikon OPC Server alias (tag) creation.

Figure 4.4 shows two windows positioned side-by-side. The window on the left is the Matrikon OPC Server with tags configured for the material handling application (discussed in the next chapter). The Explorer window on the right was used to monitor and verify the value, quality and timestamp data of the tags in the system.

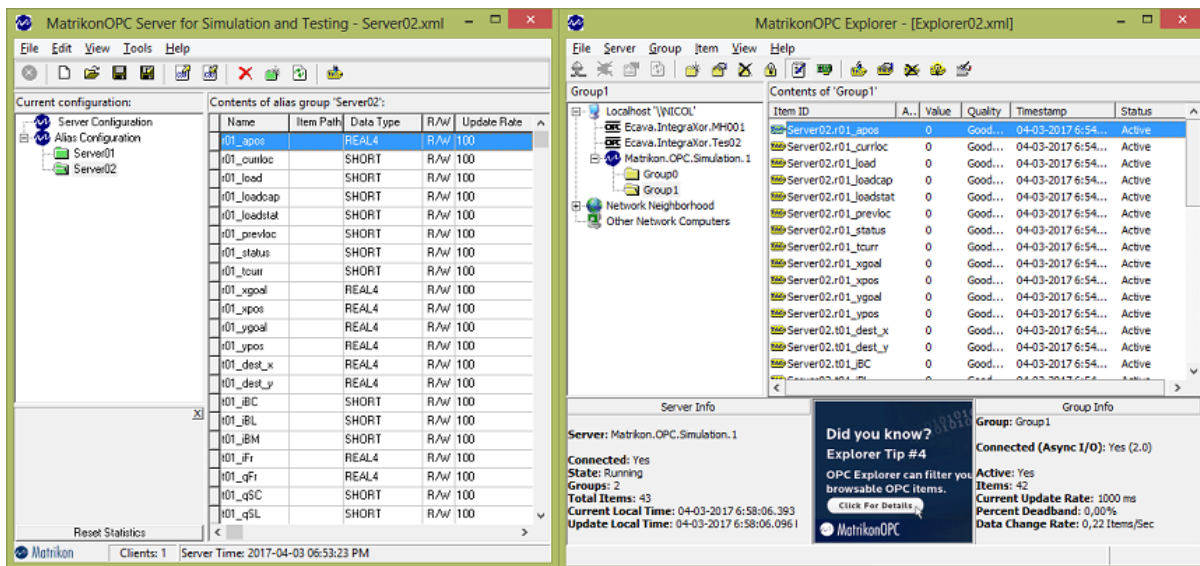


Figure 4.4. Matrikon OPC Server and Explorer with tag value monitoring.

4.4 GUI development in Python Tkinter

A concise visual of the GUI functionality is given by the flowchart layout in Figure 4.5. The *main* screen contains a menu of buttons, and was used to connect to a remote mobile robot through the use of an IP address. Each mobile robot operates on a Linux OS, integrated with the ROS middleware layer. The user is able to ‘ping’ an IP address to check if it exists in the network. Other buttons such as *robot configuration* and *task management* will direct the user to screens where robots and tasks can be configured respectively for specific applications. The *assign tags* button will navigate the user to a screen where OPC-UA tags can be assigned to local Python algorithm variables.

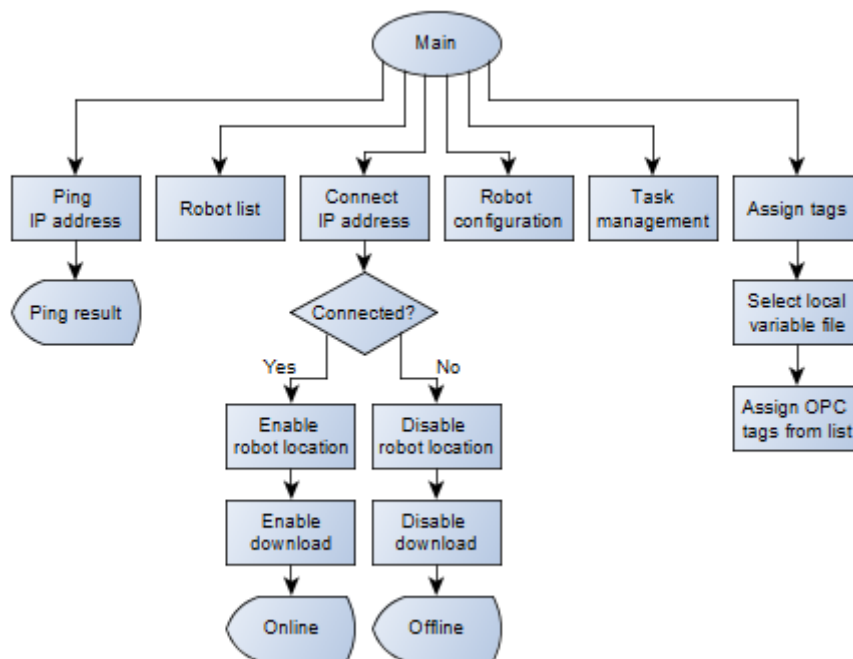


Figure 4.5. Flowchart of the GUI functionality.

The complete Python code for the Tkinter GUI development is included in Appendix A. It contains object orientated code for the graphic development of each screen and accommodates for user interactions that can possibly occur on the screens. The list represented by Table 4 gives a quick reference to starting places in the Python code where the development for the major screens (as outlined by Figure 4.5) occurs. The robot and task related data are stored in text files, located in the directory where the GUI application is

executed. The distributed feature of the application allows for its execution at any location in the same Ethernet network of the mobile robots.

Table 4. Starting line reference to the Python code in Appendix A.

GUI screen/function	Line reference
Main	100
Connect	173
Robot configuration	228
Robot list	433
Task management	479
Assign tags	687
Robot location	954

4.4.1 Main screen and remote robot connection

The main screen, with offline and online connections, is shown in Figure 4.6.

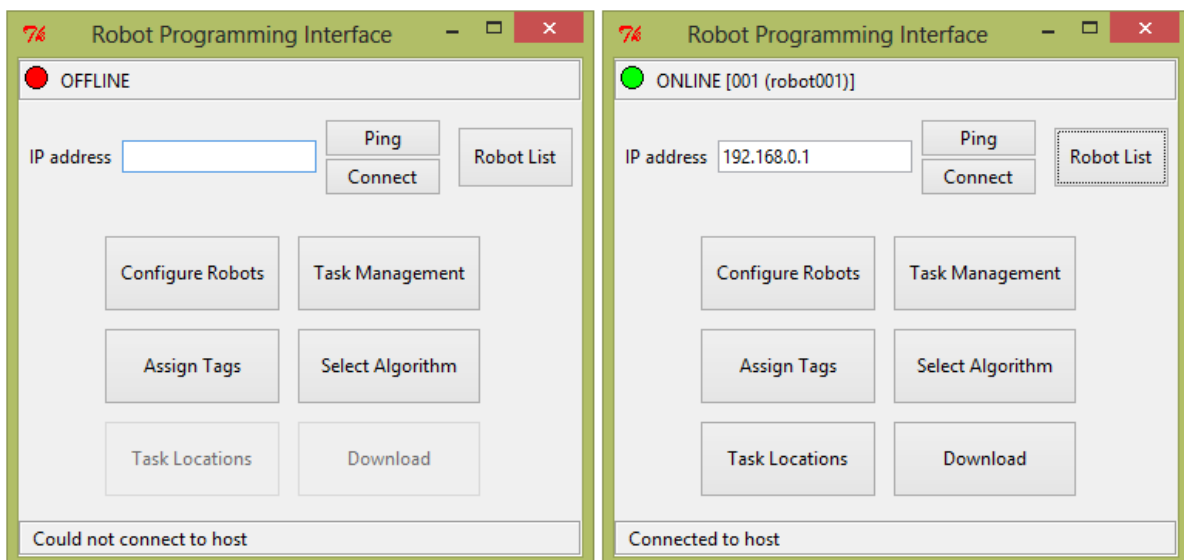
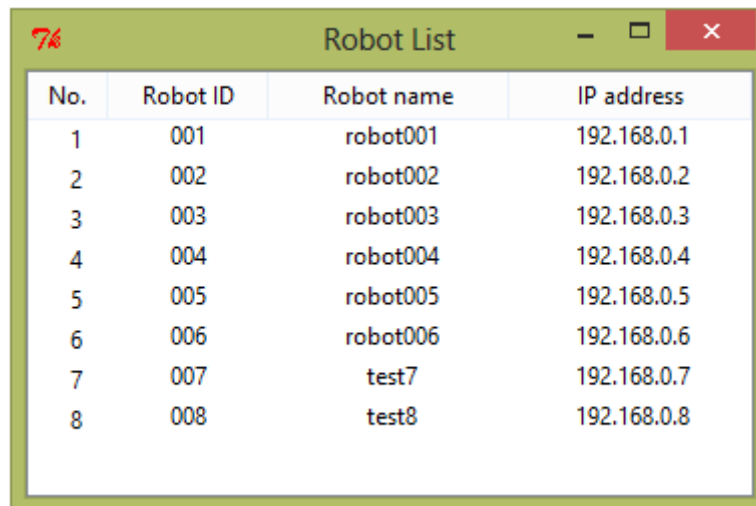


Figure 4.6. The GUI main screen with offline and online connections.

In the offline mode, the *task locations* and *download* buttons are disabled, whereas they are enabled in the online mode since the user would be connected to the remote robot via its IP address. Status messages at the bottom of the screen indicate whether the user “could not connect to host”, or is “connected to host”, and red or green icons at the top of the screen signify whether the user is offline or online to the robot respectively. During an online connection, the ID and name of the robot is displayed (*001 (robot001)*) as shown in Figure 4.6), this was coded to ensure that the user is connected to the correct robot in the network.

There is also functionality to view the list of robots in the network by clicking on the *Robot List* button, which populates a list in a pop-up window as shown in Figure 4.7. The list is itemised with the robot ID’s, names, and IP addresses. Static IP addresses are used for each robot which was sufficient for this research study; however, this can be a cumbersome method of configuration in a large network of robots. Hence, future development will see the use of dynamic IP addresses and smarter, automated ways in which the user can connect to remote robots.



The image shows a screenshot of a software window titled "Robot List". The window has a green title bar with a red "76" icon on the left and standard window control buttons (minimize, maximize, close) on the right. The main content area contains a table with four columns: "No.", "Robot ID", "Robot name", and "IP address". The table lists 8 robots with sequential IDs and names, and IP addresses in the 192.168.0.x range.

No.	Robot ID	Robot name	IP address
1	001	robot001	192.168.0.1
2	002	robot002	192.168.0.2
3	003	robot003	192.168.0.3
4	004	robot004	192.168.0.4
5	005	robot005	192.168.0.5
6	006	robot006	192.168.0.6
7	007	test7	192.168.0.7
8	008	test8	192.168.0.8

Figure 4.7. The robot list pop-up window.

The connection between the GUI and the remote robot was made through the SSH (Secure Shell) protocol [102]. SSH allows for a secure remote login from one computer to another by using strong encryption methods. The connection works on a client-server model, and was established through a sequence of steps illustrated by Figure 4.8 [102], where in this application, the SSH Server is the remote robot's Linux OS and the SSH Client is the Python GUI code.

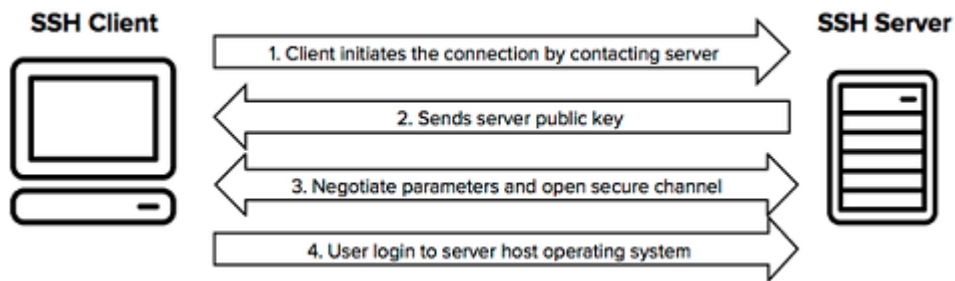


Figure 4.8. A simplified setup flow of a SSH connection.

The Python SSH client implementation was achieved through the *Paramiko* package, a Python execution of the SSH protocol. The code segment that performs the SSH connection is found in Appendix A, lines 175–186. During the connection, the remote robot ID was retrieved from the robot's Linux OS and was downloaded to the local machine that executes the GUI.

4.4.2 Robot configuration

The robot configuration GUI pop-up window is shown in Figure 4.9. Each robot was configured and identified by a *robot ID*, *name*, and *IP address*. The user can also record the length (*L*), width (*W*), and height (*H*) dimensions of the robot, as well as the technical limitations such as the maximum load capacity, and linear/angular speed. Other robot hardware features can be recorded by selecting the appropriate check boxes, this is necessary for specific tasks that require robots installed with these features.

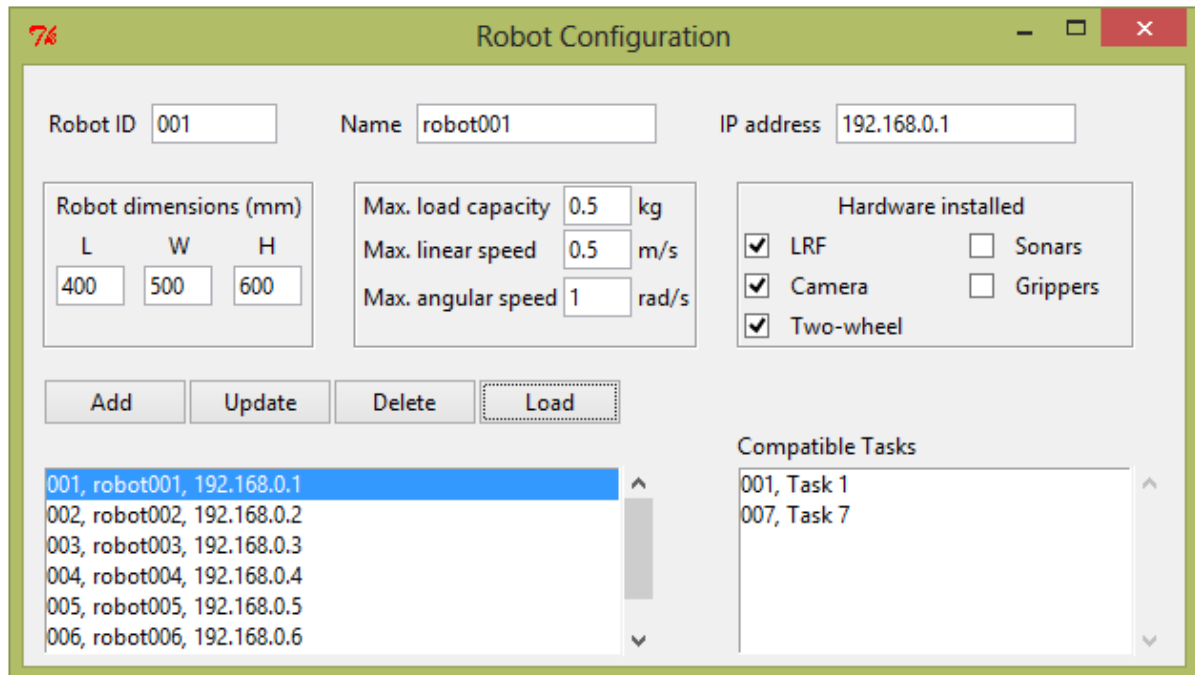


Figure 4.9. The robot configuration GUI screen.

Users are able to *add* new robots to the current list, and *update* existing records. The *delete* button enables the removal of records from the robot list, while the *load* function populates the screen with robot data from the selected field in the list box. Data records for each robot are stored in text files, located in the same directory as the executable file of the GUI application.

The *compatible tasks* list box show the tasks that are able to use the current loaded robot, a feature that is useful for the programmer when working with robot task algorithms. The development involved in determining the compatible tasks is discussed in section 4.7.2, since the feature was used in the cooperation between heterogeneous mobile robots.

4.4.3 Task management

The task management GUI pop-up window is shown in Figure 4.10, which has a similar layout to the robot configuration screen. Every task is configured with an *ID* and *name*, and the robot requirements for the task are entered in the appropriate fields, which include: 1) maximum robot dimensions, 2) technical characteristics such as minimum robot load

capacity and speed, 3) the maximum number of robots that are allowed in a team for the particular task, and 4) the hardware requirements of the robot.

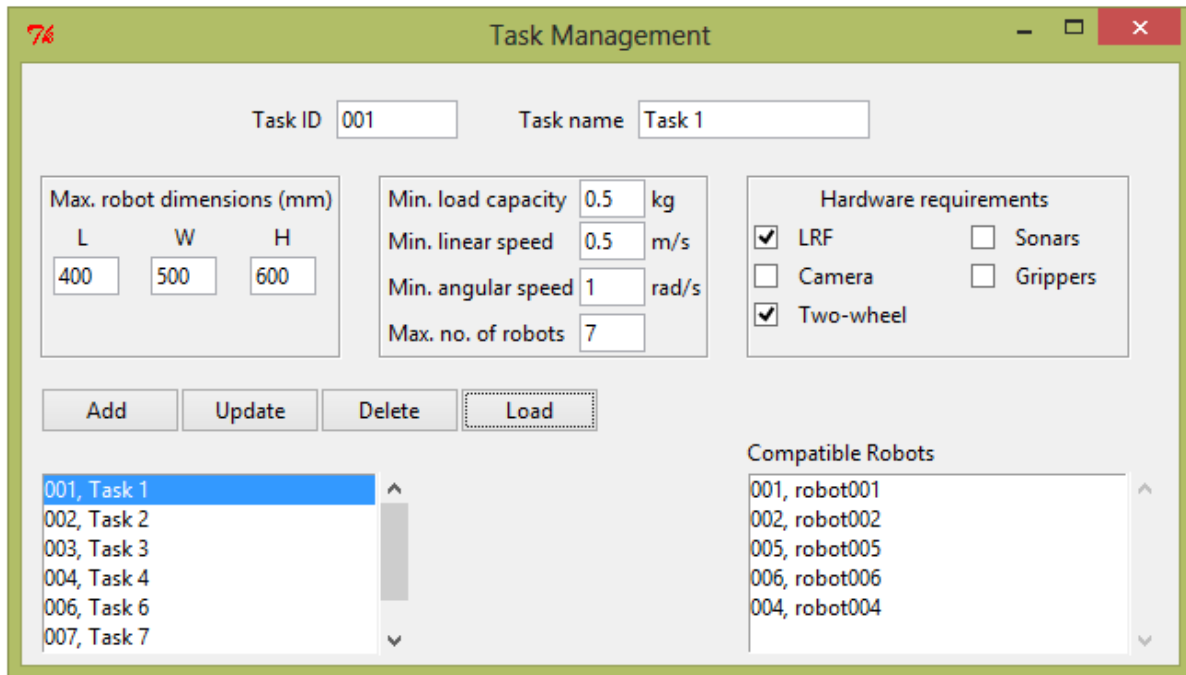


Figure 4.10. The task management GUI screen.

Similar to the robot configuration screen, the task management screen displays a list of compatible robots for the loaded task, based on the robot requirement fields entered by the user. As discussed in the previous section, the development involved in determining the compatible tasks to robots, and vice versa, is discussed further in the chapter.

4.4.4 OPC-UA tag assignment

The variable file selection tool in Figure 4.11 allows the user to choose a variable list from possible algorithm files for a specific application. The tag assignment GUI function is a valuable feature of the framework since the user can define any local variable name in the algorithm and then simply map the variables to the OPC-UA tags, which are automatically scanned in the “OPC Tags” pop-up window, shown in Figure 4.12. The mapping is necessary for the loading of OPC-UA tag values to local variables without the user directly searching and coding tags in the algorithm (or program). This gives the system flexibility and ease of

use. The process by which OPC-UA tags are read from the Matrikon OPC Server is discussed in section 4.4.6.

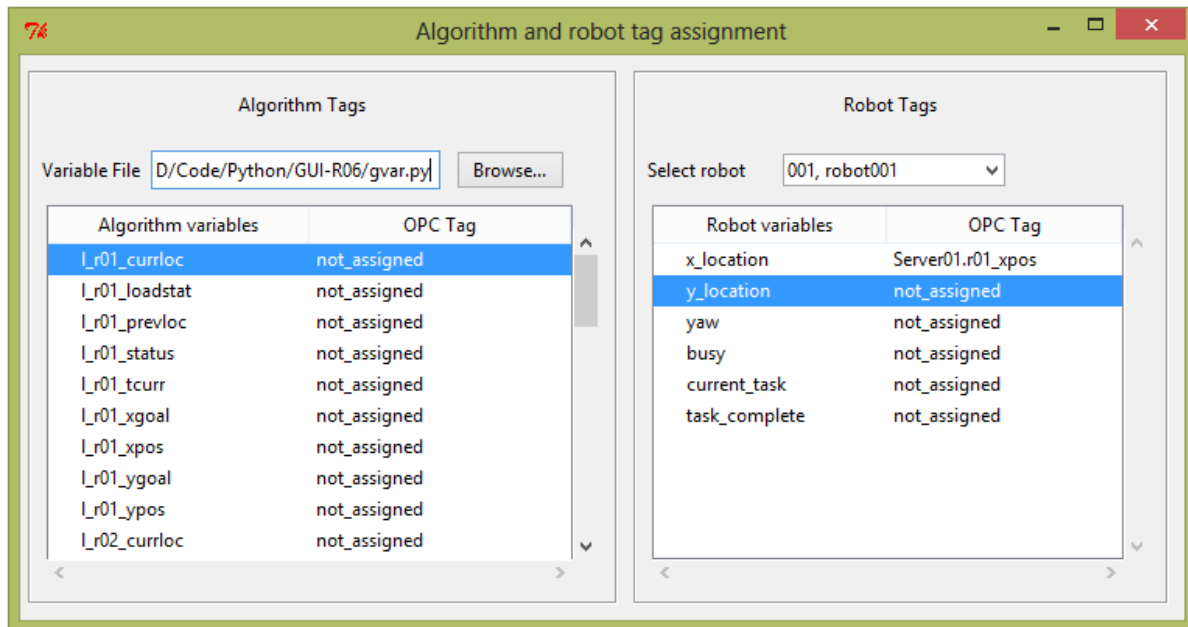


Figure 4.11. Algorithm and robot tag assignment screen.

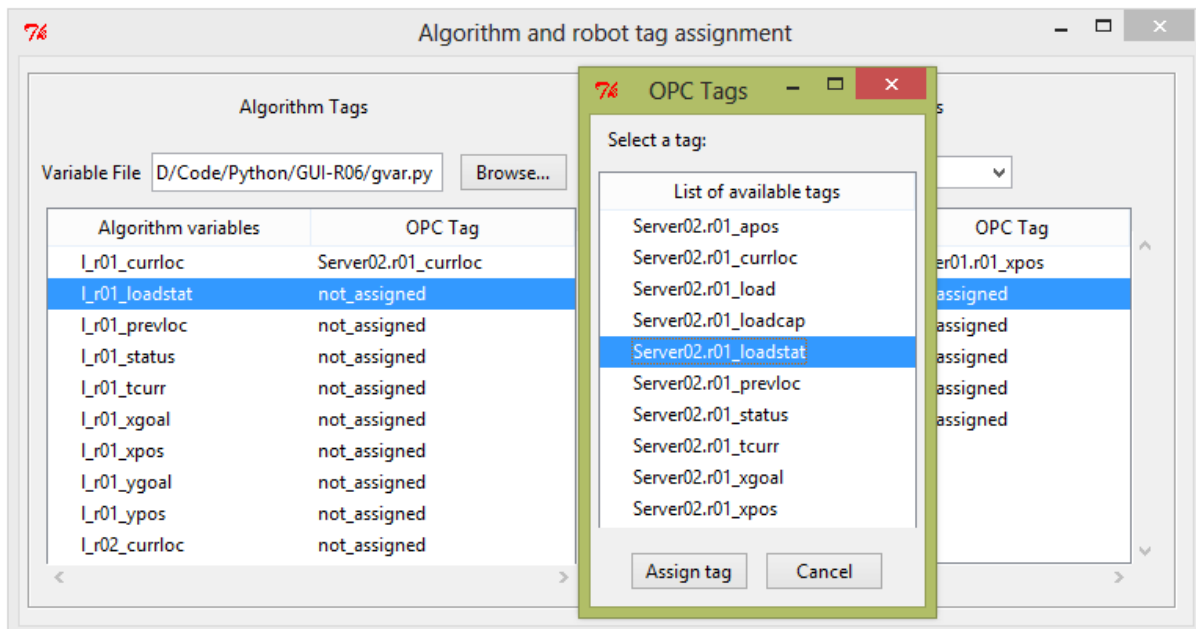


Figure 4.12. Tag assignment screen with OPC tag list pop-up window.

4.4.5 Robot location

The *task location configuration* screen, shown in Figure 4.13, is only accessed when an online connection to the remote robot is made. The purpose of this utility is for the calibration of x-y coordinate points in an environment where the robot's localised coordinates do not correlate with its actual location on a map. Through this screen, the user is able to move the robot in the environment by clicking on the *motion command* buttons, and assign x-y data to specific *task locations*. This data can then be used by the robot to move to an actual task location, and not one that is incorrect due to robot calibration errors.

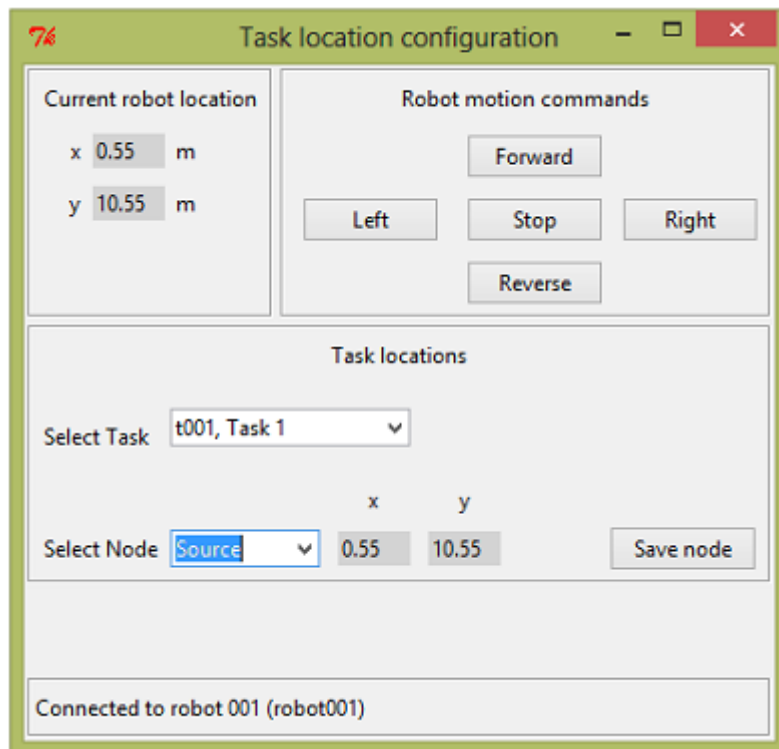


Figure 4.13. The task location configuration screen.

4.4.6 Python OPC-UA Client

The Python OPC-UA Client was established through the *OpenOPC* library, which provides methods for connections to OPC Servers and methods for reading and writing values from/to OPC tags respectively. The flowchart in Figure 4.14 outlines the communication sequence

required to 1) connect to an OPC Server, 2) write or read tags, and 3) disconnect from the Server. The creation of an OPC Client object (*opc* in Figure 4.14) is required to initiate the communication, and the IP address of the Server is specified. In the figure, the *localhost* address is used which implies that the Client and Server reside on the same local machine. Connection to the Server requires the use of a Server name; in this case it is *Matrikon.OPC.Simulation*.

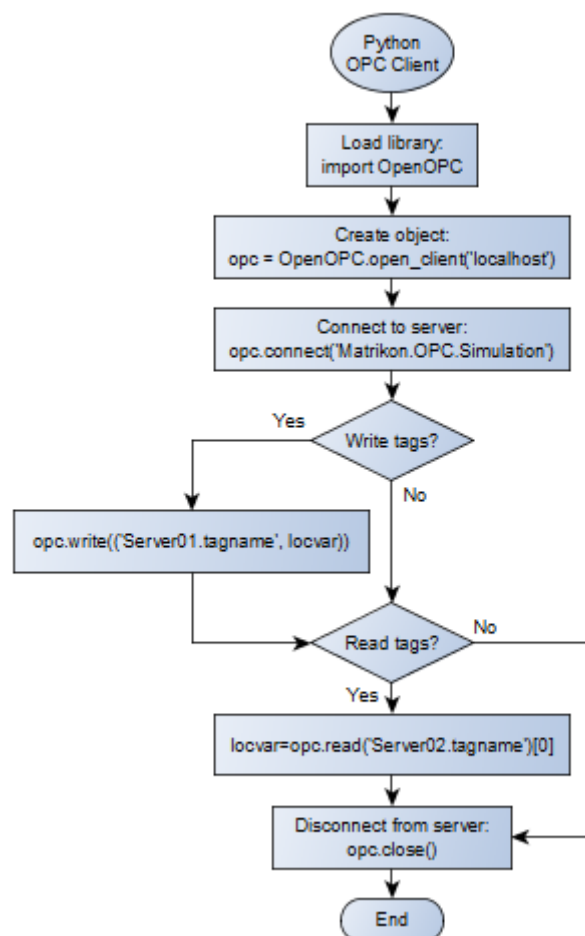


Figure 4.14. Python OPC Client communication with the Server.

The writing of OPC tags involves the specification of the Server alias (*Server01*), followed by the tag name. The *tagname* is then assigned the value retained in the *locvar* variable, as per Figure 4.14. The tag read process is similar, yet requires the specification of the tag index, where “[0]” calls for the value. Other indices can call for the tag quality (good or bad) or timestamp.

4.5 Mobile robot system integration

The *development overview* section in this chapter introduced four topics of discussion in the mobile robot system integration, each of which is covered in the subsequent sections. An overview of the remote mobile robot system is given by Figure 4.15, which shows the communication between each module in the scheme.

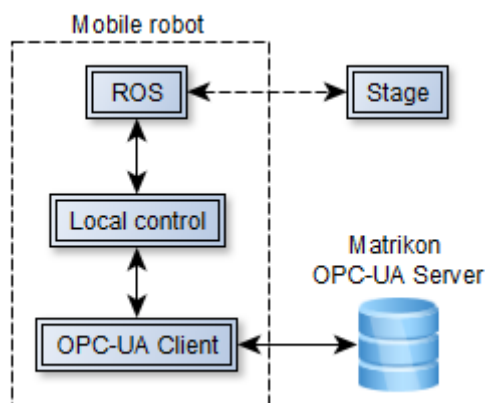


Figure 4.15. Overview of the mobile robot system integration.

4.5.1 ROS implementation

The remote mobile robot code was also implemented using the Python programming language, and the program for each robot's ROS implementation and localised control is documented in Appendix B. The Python client library for ROS is *rospy*, which enables the ability to interface between ROS topics and services. The flow diagram in Figure 4.16 represents a basic implementation of the ROS core, using topics that were subscribed to during the initialisation.

Topic names are also defined by the programmer at initialisation, and it is imperative that the topic name be used in a call-back function where the attributes of the topic are accessed. For example, the *rosAI.py* program in Appendix B subscribes to the *amcl_pose* topic in line 115. The topic name is defined as *amclCb*, which is the name of the call-back function in line 107,

and it is here where the AMCL attributes of the robot's position are retrieved for use in the main loop of the code.

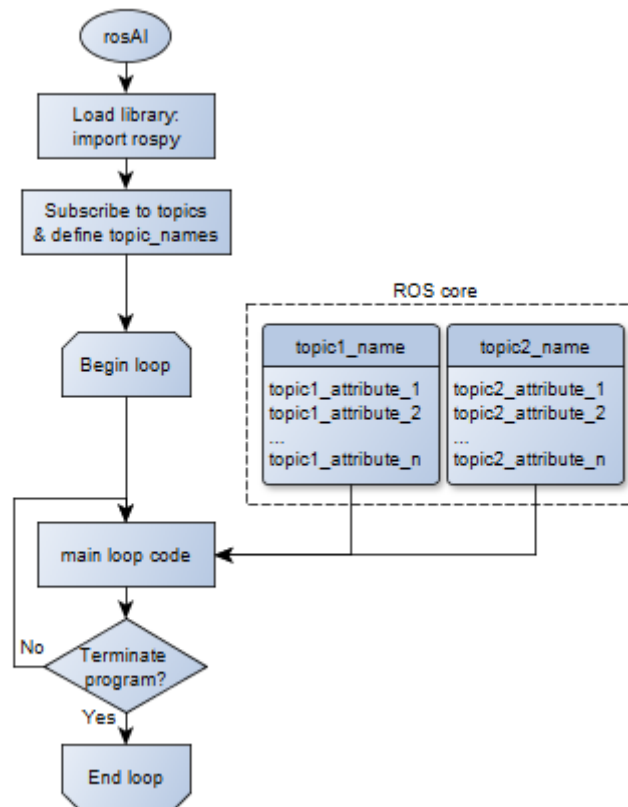


Figure 4.16. A basic format of the ROS core implementation.

4.5.2 Robot OPC-UA Client

The OPC-UA Client code implementation for each mobile robot is documented in Appendix C, under the *OPCdata.py* filename. Since the program was written in Python, the *OpenOPC* library was used here as well, hence the connection to the Matrikon OPC Server and the methods used for accessing OPC tags are direct implementations of the procedure outlined in section 4.4.6. One apparent difference is the change in IP address configuration due to the remote residence of the Client.

The *OPCdata.py* file was imported as a library in the remote robot control code (Appendix B), and the functions used from the file were: 1) *getxyGoal* which reads the robot's x and y

coordinate goal location from the OPC-UA Server, 2) *setStatus* writes the current boolean status of the robot, whether it is busy or available, and 3) *xyPos* writes the current x and y coordinate position of the robot to the OPC-UA Server.

4.5.3 Decentralised robot control

The ROS *move_base* action client was used to move the robot to a specific location in the environment by accepting goal coordinate locations. The state diagram in Figure 4.17 shows the sequence involved in the local navigation algorithm. The sequence begins by waiting for a goal location command from the higher-level control system, via the OPC Server, and executes the *move_base* service after the “busy” status of the mobile robot is set. The *status* variable was used to reserve the robot resource to a particular task so that there are no goal location conflicts.

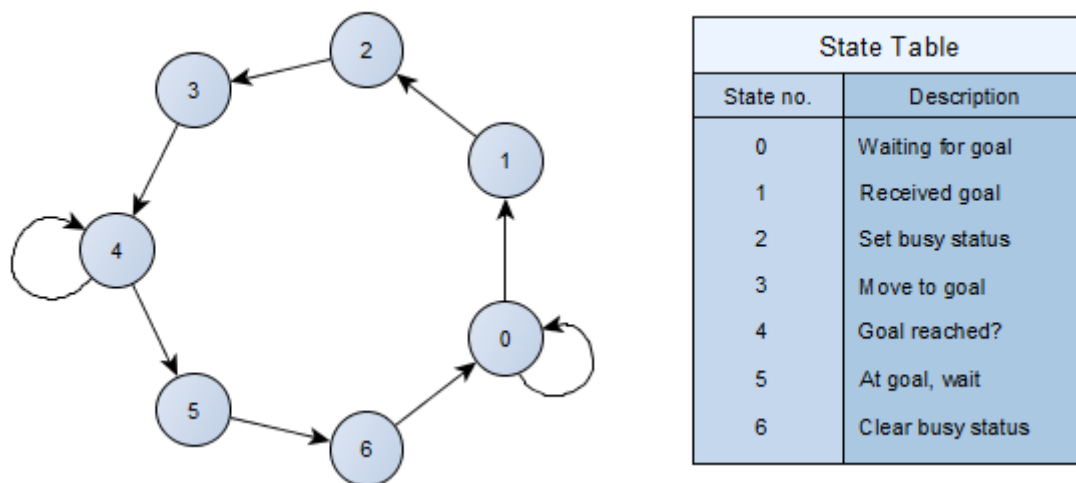


Figure 4.17. State diagram of the local navigation algorithm.

The goal was reached when the robot was within a distance of 0.5m from the actual location, which was calculated by using the standard distance formula between two points, as shown in Equation 4.1.

$$d = \sqrt{(x_g - x_c)^2 + (y_g - y_c)^2} \quad (4.1)$$

4.5.4 Stage simulation

The ROS/Stage simulation of the mobile robots involves the use of specific files, each containing parameters or definitions of the robots and the simulation environment. The diagram in Figure 4.18 represents the association between the different file types configured to achieve the simulation in Stage. References to the code in the Appendix are shown in brackets for each file.

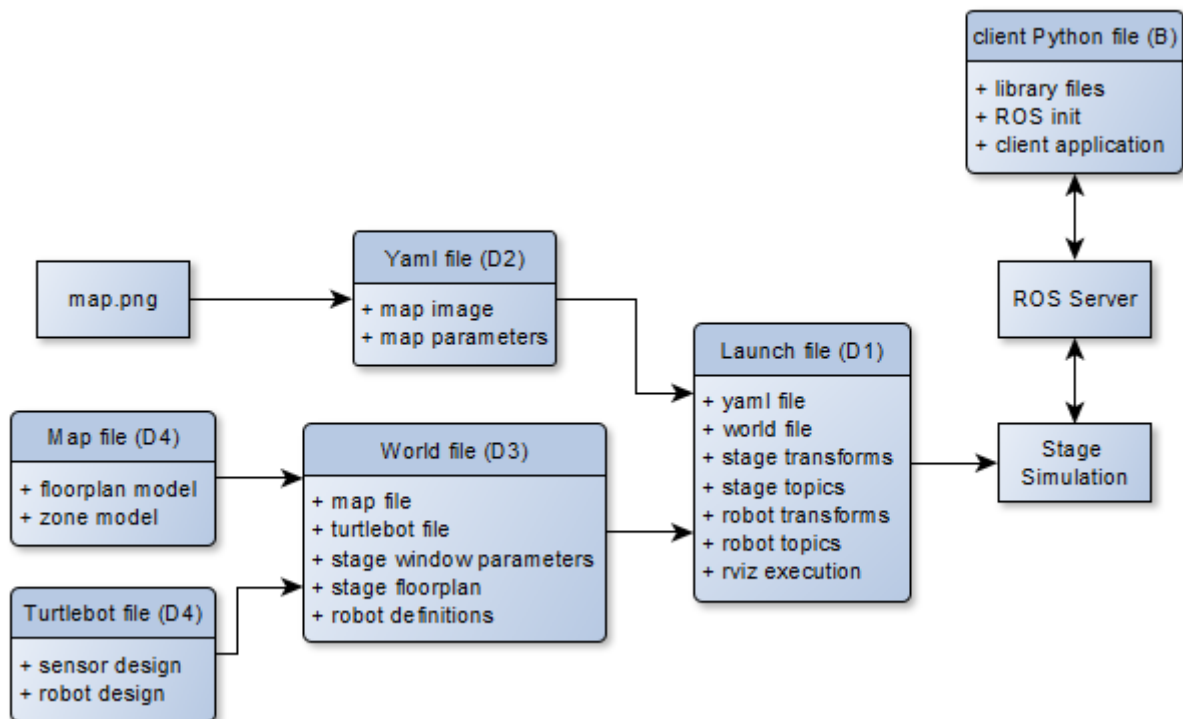


Figure 4.18. Stage file associations with Appendix references in brackets.

The function of each file illustrated in Figure 4.18 is discussed in the list below:

- The *world* file is a programmable description of the simulation environment, which includes the robots and the map. The simulation window size and floor size can be defined here, as well as the attributes of the robots, such as the names, colours, and start-up positions in the simulation.
- The *map* file is included in the *world* file and allows for modular programming through the use of models.

- The *turtlebot* file is similar to the *map* file in that it allows for the definition of models related to the robots. These include robot size and hardware attached to the robot, such as laser range finders (LRF).
- The *yaml* file is required for the specification of the map data, which include the map image name and the resolution of the image.
- The *launch* file is essential since it was used to define the links between the various ROS nodes in the simulation. ROS and Stage topics were configured in this file for each robot in the system, and the parameters for the localisation algorithm, in this case AMCL, were also set. In addition, an optional definition for the ROS 3D robot visualisation tool, *rviz*, can be configured here.

An execution of the *launch* file produces a Stage simulation window with the map, robots, and LRF scanners as shown in Figure 4.19. The map illustrates the locations of two tasks, each containing source and destination points for the robots to transport material between goal locations.

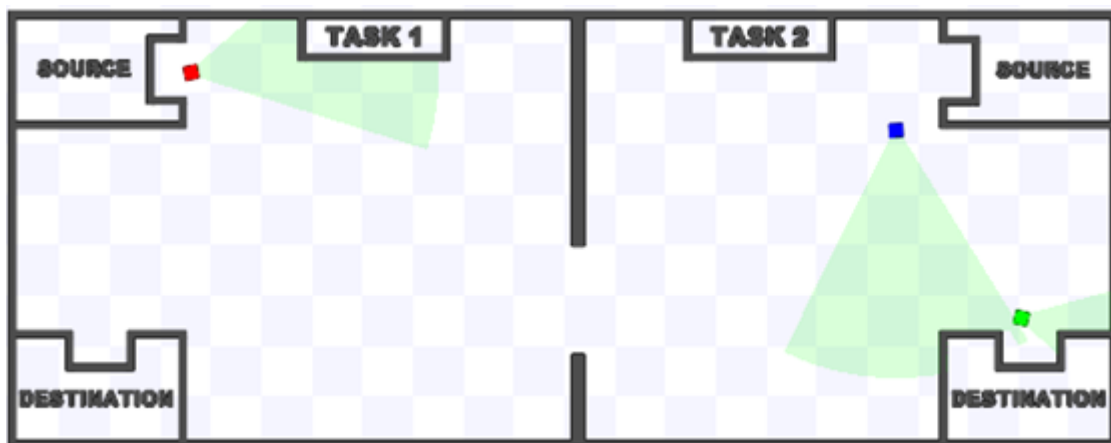


Figure 4.19. Stage simulation window with map and robots.

4.6 SCADA development

The SCADA development was performed using the free, unregistered version of the Ecava IGX software package. The version gives a two hour period to access the mimic, which was

found acceptable during simulation studies. As mentioned at the outset of this chapter, the following chapter involves the application of case studies with mobile robot material handlers, thus the SCADA mimic development caters for this approach. Figure 4.20 shows the Ecava Project Editor and the configuration page of the mapping between local variables and OPC Server tags. The OPC setup between Client and Server was a simple process and once completed, local SCADA variables were used directly as fields in the mimic.

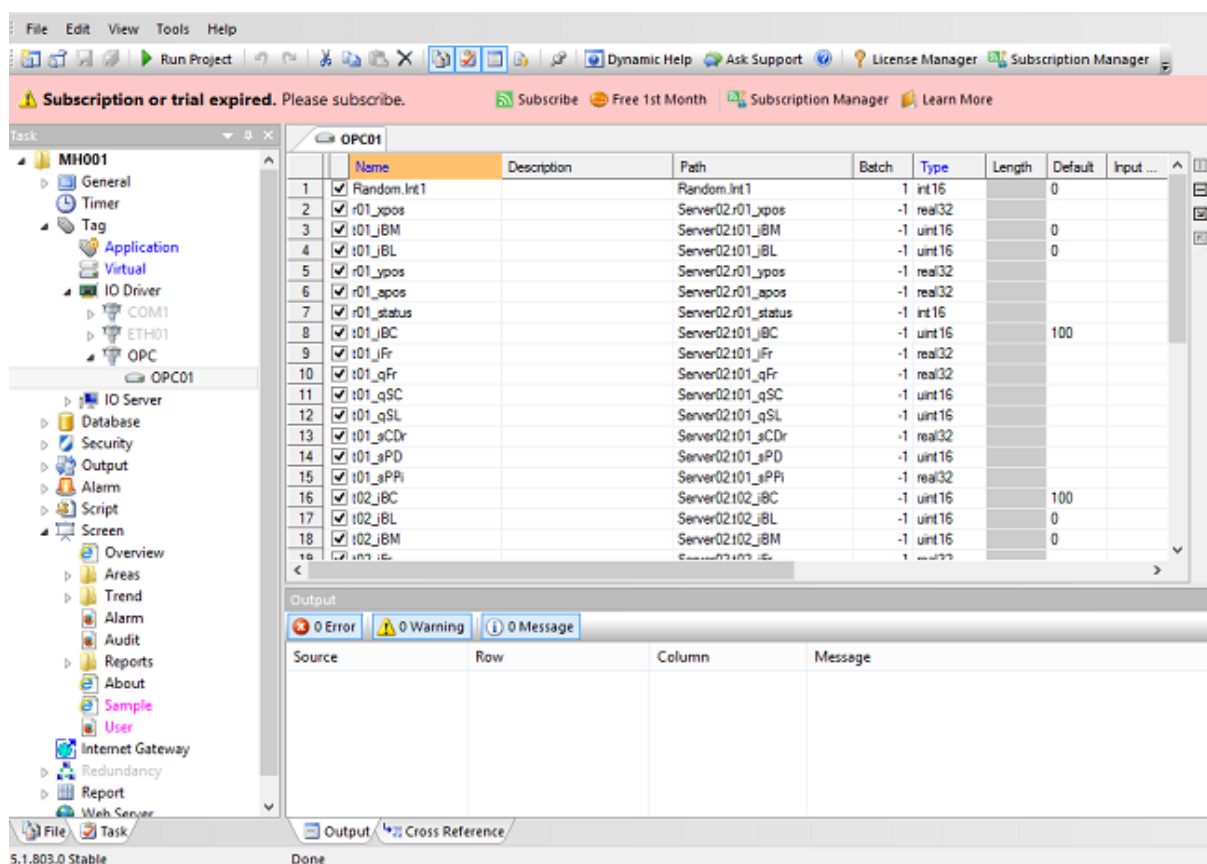


Figure 4.20. Ecava SCADA Project Editor showing the OPC client tag configuration.

The material handling system consists of two tasks, each containing an input material buffer and an output storage buffer. Mobile robots were tasked with transporting material from the input buffer to the output storage area for specific tasks, either task #1 or task #2. The task goal output for the robot was determined by the end user's Python algorithm and the task status together with the robot goal locations are shown on the SCADA mimic. The development of the mimic was done in Inkscape (Figure 4.21), and an example of the SCADA project execution is shown in Figure 4.22.

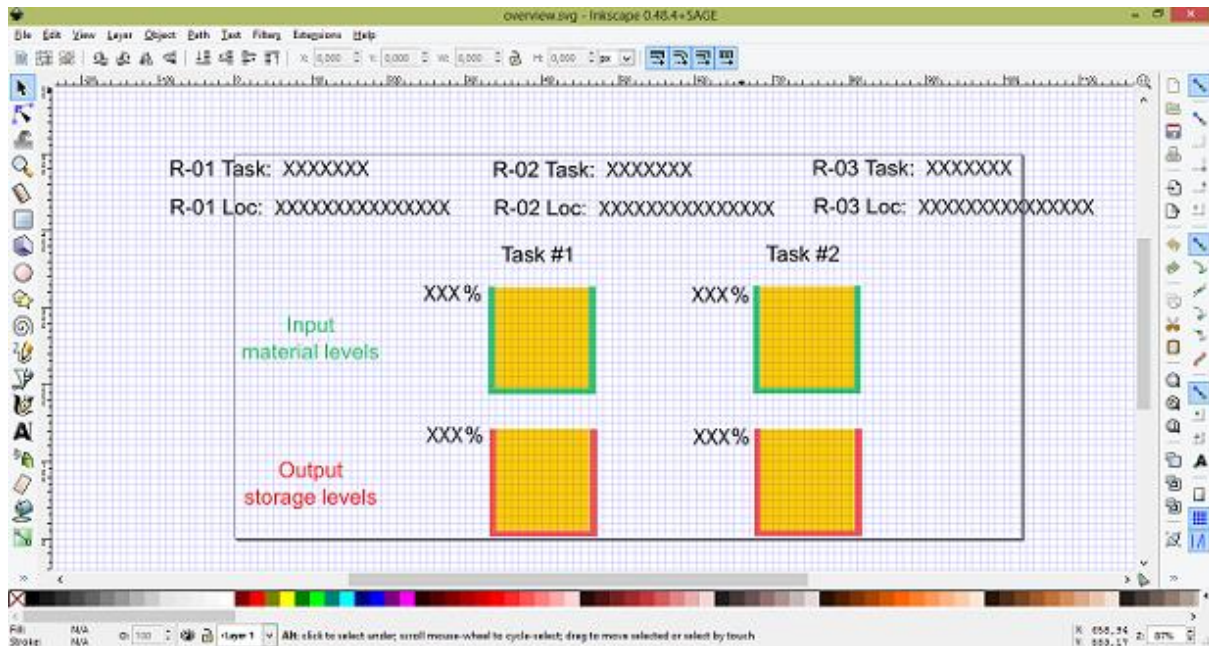


Figure 4.21. SCADA screen development using Inkscape.

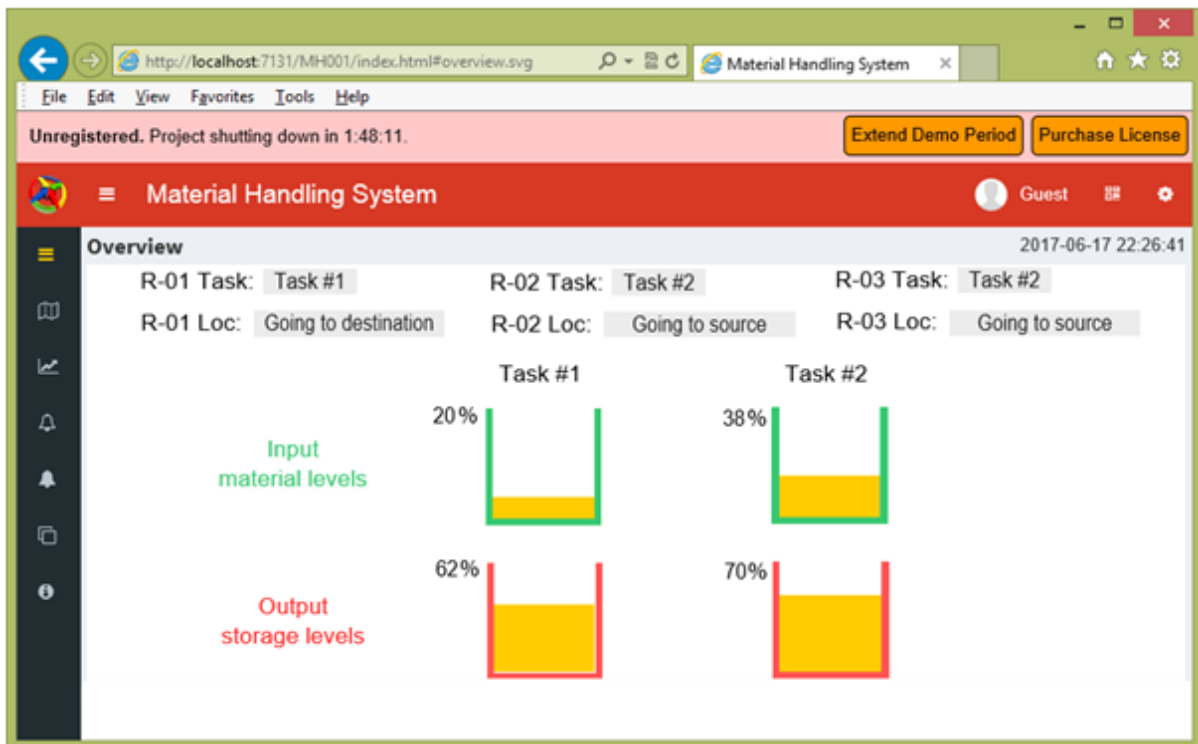


Figure 4.22. A SCADA project execution in the Internet Explorer web browser.

4.7 Python programming features

The use of the Tkinter GUI is the first part of the process required by the end-user to program the high-level intelligence of a mobile robot control system. The second part involves the user development of Python algorithms that access the data created through the GUI. The purpose of this section is to present the development involved in providing support for user programming features that satisfy the contributions of this research study. These features will also be used in the case studies to verify the performance of the framework.

4.7.1 Local variables and OPC tags

Section 4.4.4 discussed the process of using the GUI to map local variables in the user's Python program to OPC Server tags. The reason for the mapping was to enable the user to define any local variable name in the program (or algorithm) and then simply use the variables in place of the OPC Server tags, thus saving the programming time of searching for tags and configuring each of them for value retrieval.

Figure 4.23 is a flowchart representation of the two Python functions that were developed to: 1) obtain the local variables and OPC tags from the mapped file (*avarmap.txt*, created through the GUI), and 2) update the local variables with the actual values of the OPC tags, as per their mapping schedule. The text structure of the mapped file is shown on the left of Figure 4.23.

The *mapVars* function goes through each line in the *avarmap.txt* text file and stores the local variable names and OPC tag names to arrays *locArr* and *opcArr* respectively. Thereafter, the *updateVars* function steps through the indices of *locArr* and reads the corresponding indexed OPC tag value. Each value is assigned to the mapped local variable defined in the *gvar.py* file, which is achieved by using the name attribute of the elements in *locArr*.

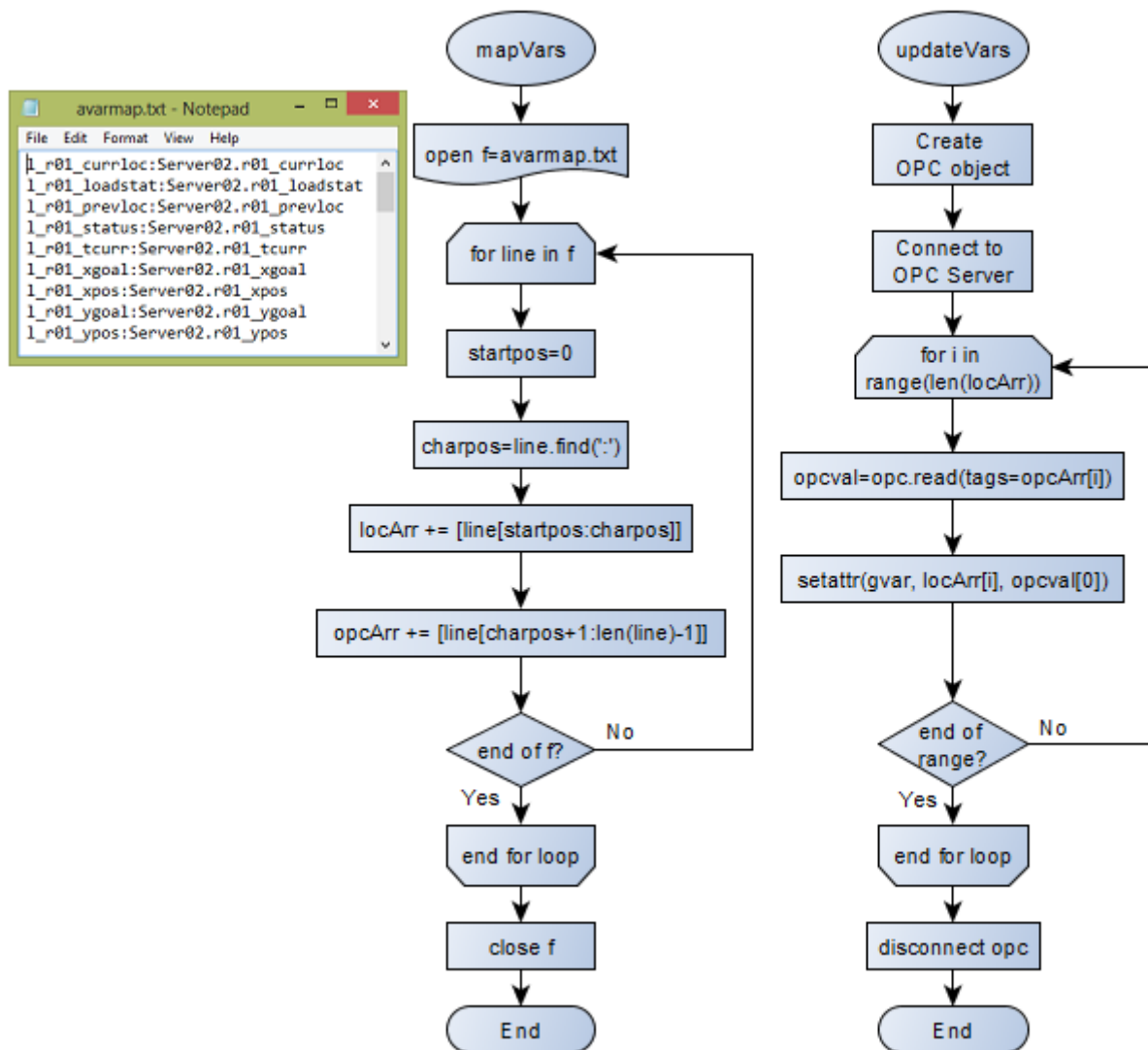


Figure 4.23. Functions that map and update local variables.

4.7.2 Heterogeneous mobile robots and cooperation

The functions that were developed for heterogeneous and cooperative mobile robot applications involve compatibility checks between robots and tasks. Figure 4.24 gives an overview of the development of these functions in Python, and the source code can be found in Appendix E. The *getCompatTasks* and *getCompatRobots* functions return compatibility matrices which are used in the display of compatible tasks or robots respectively in the GUI, as discussed in sections 4.4.2 and 4.4.3.

The configuration data required for the execution of these functions is obtained from the *taskconfig.txt* and *robconfig.txt* files, the string structure of which are shown in the figure. The lines in the files are structured to contain configuration information for the tasks and robots.

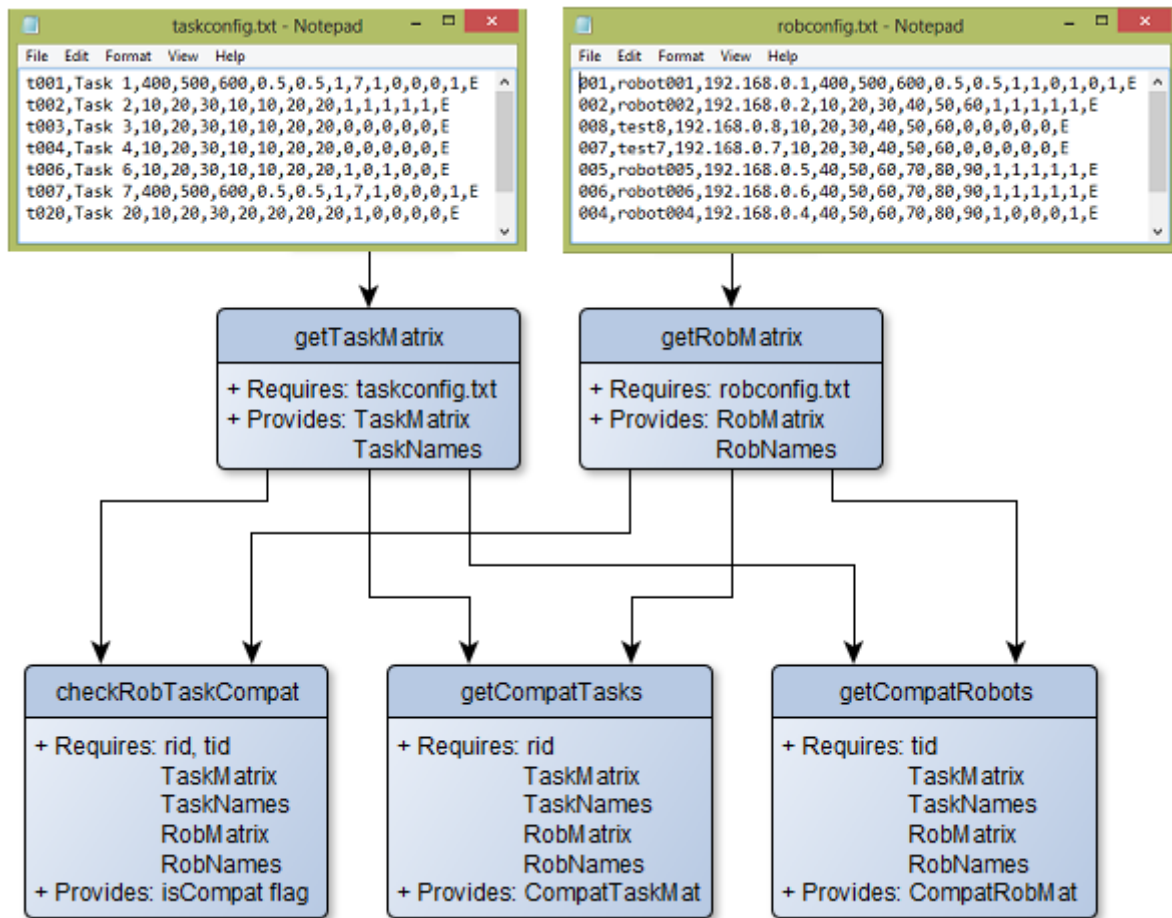


Figure 4.24. Functions developed for heterogeneous robot applications.

The *checkRobTaskCompat* function is a useful tool for the end-user since it requires just the task and robot ID's and returns a Boolean result defining whether a robot is compatible to a particular task. It can be used to create algorithms which automatically determine the robots in the network that are suited for specific tasks. An application of this function is seen in the next chapter.

4.7.3 Machine learning support

One of the contributions of this research study is the application of the field of machine learning in multiple mobile robot networks through the use of Python's powerful machine learning library, *scikit-learn*. An effort was made to implement one of the machine learning algorithms namely, the Support Vector Machine (SVM) algorithm, to the robot material handling scenario. The case study application of the SVM algorithm is discussed in the next chapter, but prior to its use some development work in Python was prepared.

SVM learning is related to statistical theory and is commonly used in the field of bioinformatics due to its ability to accommodate for high dimensional space data. There are various types of SVM classifiers; the basic linear classifier (also known as linear SVM), was implemented in this research. Linear SVM predicts whether an input belongs to one of two classes, and is achieved by building a model from a database of training examples that contains the mapping of input data to class labels. With this in mind, an SVM training application was developed in Python Tkinter. The source code for the application is documented in Appendix F, and its execution is shown in Figure 4.25.

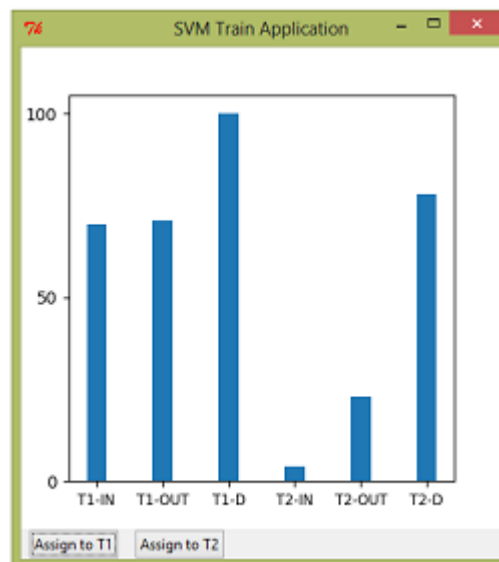


Figure 4.25. SVM train application created in Python Tkinter.

The *SVM Train Application* was used in one of the case studies (section 5.3) to create a database of training examples required in the generation of the SVM model. The y-axis levels for each task (1 or 2) shown in the figure are randomly generated every time the “Assign to Tx” button is pressed. This action will assign a binary class label to the values of the SVM features shown on the x-axis and the results are stored in a text file.

4.8 Framework review

The structured flow diagram in Figure 4.26 is a brief representation of the software components in the framework, each of which was discussed in this chapter. The end-user development involves the use of a choice between various programming features as well as the access to standard Python libraries and packages. The use of Ethernet communication backbone and the OPC-UA data communication standard makes the framework a distributive one; hence the end-user development can take place at any point in the same network.

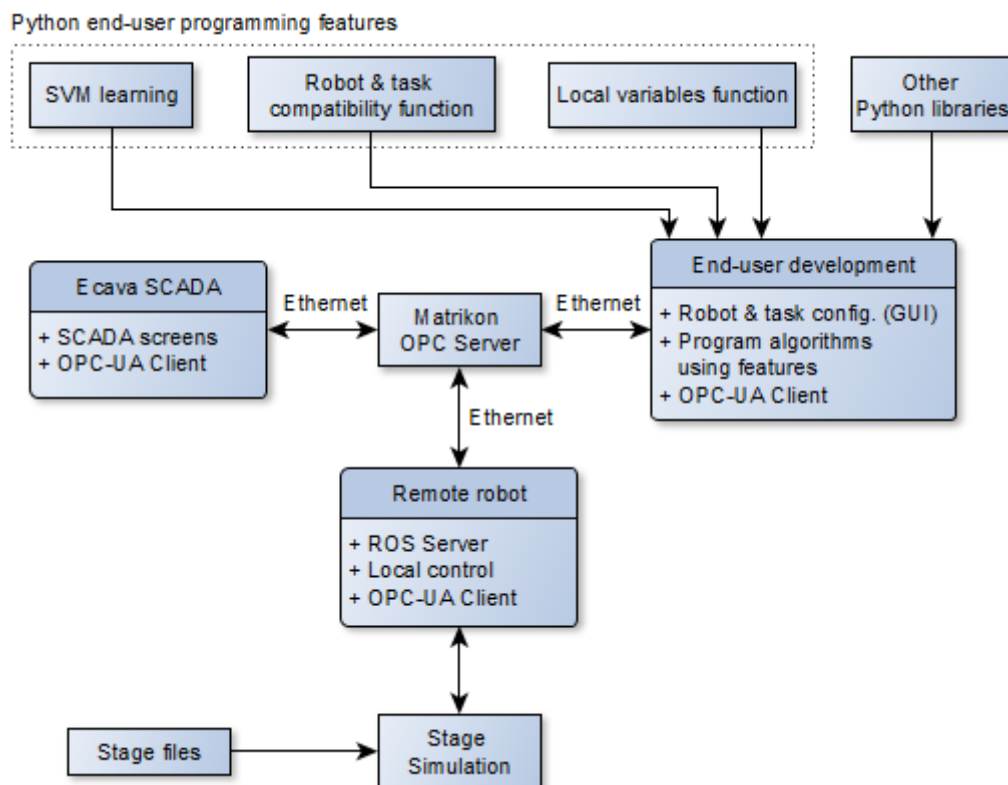


Figure 4.26. A review of the framework development.

4.9 Chapter summary

This chapter began by describing the development and verification strategies adopted, and thereafter produced an overview of the topics of discussion. The development of the main components of the framework were discussed, these include: 1) the Python Tkinter GUI screens where robot, task, and OPC tag configurations are done, 2) the mobile robot system, which detailed the ROS software implementation, decentralised-local robot control, the Stage simulator integration, and the OPC-UA client configuration, 3) the Ecava SCADA development, and 4) the development of Python programming features to provide support for end-user applications that satisfy the contributions of this research study. The chapter concluded by discussing the brief representation of the software components in the framework.

Chapter 5 Case studies

This chapter presents a series of case studies with the purpose of verifying the development work of the framework. The various functionalities discussed in the previous chapter are tested here through the application of different mobile robot scenarios. Since the framework is a tool that will be used by the end-user, the studies also demonstrate the application of example user algorithms that can be used in the high-level robot decision making process.

Four case studies are discussed in this chapter. The first involves a material handling application with a single mobile robot; the following two also pertain to material handling applications, but focus on cooperation in heterogeneous multi-robot teams and robot machine learning. The final study proves the flexibility and robustness of the framework through communication with a commercial third party system. Each case study begins with a description, followed by an application development section, and thereafter discusses the simulation results of the scenarios presented.

5.1 Study I: Single mobile robot control

The purpose of this study was to demonstrate the functionality of the framework in performing the high-level control of single mobile robot tasks. The study is also seen as a building block for the following studies.

5.1.1 Description of the study

The application of the framework in this case study pertains to a robot material handling scenario between two tasks, each containing an input buffer and an output storage location. An assumption made at the outset was that the robot was compatible to both tasks, and it

transported material from the input to output locations. Another assumption was the automated loading and offloading of material.

The robot prioritised work for a particular task based on the state of variables in a user defined algorithm. Some of the variables included the *supply chain profit per product index* and the *supply chain current product demand* which influenced the importance of a task, and hence controlled the current task priority of the robot.

Section 5.1.3 presents the simulation results of the application in two types of scenarios, each biased so that the robot prioritised one task over the other. The details of the simulation scenarios are discussed in the next section, along with the development work for this application.

5.1.2 Application development

The OPC tag definitions for the material handling application and the algorithm development are listed in Table 5. Since a single mobile robot was used in the system, the tags associated with the robot are prefixed by “r01_”, whilst the tags associated with task 1 and 2 are prefixed by “t01_” and “t02_” respectively. The x-y coordinate related tags pertain to 1) the robot’s current position and 2) the goal position of the task, either its source (input buffer) or destination (material storage) location. The *r01_status* tag was used to determine whether the robot is available for operation on a particular task, and the *r01_tcurr* tag informs the user program and SCADA application on the current task that the robot is working on.

The task related tags are split into three categories: input, output, and supply chain. The buffer levels are calculated as a percentage, defined by Equation 5.1, where x is the task number and the definitions of the variables being listed in Table 5.

$$tx_{iBL} = \frac{tx_{iBM}}{tx_{iBC}} * 100 \quad (5.1)$$

The reason for including tags that pertain to the supply chain was to make the application more relevant to an actual scenario, where these variables do play a vital role in determining the type of product (or task) that is more crucial to a manufacturer at any particular time.

Table 5. OPC tag definitions for Case Study I.

OPC tag name	Tag description	Data type	Unit
r01_xpos	robot-01 x position	decimal	m
r01_ypos	robot-01 y position	decimal	m
r01_status	robot-01 idle, busy signal	binary	
r01_tcurr	robot-01 current task number	unsigned	
r01_currloc	robot-01 current location	unsigned	
r01_prevloc	robot-01 previous location	unsigned	
r01_xgoal	robot-01 x goal position	decimal	m
r01_ygoal	robot-01 y goal position	decimal	m
r01_loadcap	robot-01 load capacity	unsigned	
r01_load	robot-01 current load	unsigned	
r01_loadstat	robot-01 loading/offloading status		
t01_iBM	task-01 input buffer number of material	unsigned	mat
t01_iBL	task-01 input buffer level	unsigned	%
t01_iFr	task-01 input fill rate	decimal	pr/hr
t01_iBC	task-01 input buffer capacity	unsigned	mat
t01_qSM	task-01 output buffer number of material	unsigned	mat
t01_qSL	task-01 output storage level	unsigned	pr
t01_qFr	task-01 output fill rate	decimal	pr/hr
t01_qSC	task-01 output storage capacity	unsigned	mat
t01_sCDr	task-01 supply chain consumer demand rate	decimal	pr/hr
t01_sPPi	task-01 supply chain profit per product index	decimal	
t01_sPD	task-01 supply chain current product demand	unsigned	pr
t02_iBM	task-01 input buffer number of material	unsigned	mat
t02_iBL	task-02 input buffer level	unsigned	pr
t02_iFr	task-02 input fill rate	decimal	pr/hr
t02_iBC	task-02 input buffer capacity	unsigned	pr
t02_qSM	task-02 output buffer number of material	unsigned	mat
t02_qSL	task-02 output storage level	unsigned	pr
t02_qFr	task-02 output fill rate	decimal	pr/hr
t02_qSC	task-02 output storage capacity	unsigned	mat
t02_sCDr	task-02 supply chain consumer demand rate	decimal	pr/hr
t02_sPPi	task-02 supply chain profit per product index	decimal	
t02_sPD	task-02 supply chain current product demand	unsigned	pr

The Python application algorithm is represented by the block diagram shown in Figure 5.1, and the source code is documented in Appendix G1. The variables presented in the diagram are the OPC tags defined in Table 5, with the exception of the prefix text. The algorithm was executed at regular intervals for task 1 and task 2, hence the T_{wi} output shown in the figure is the weighted value for each task. A task becomes a higher priority when its result becomes more negative than the other task(s). For example, if $T_{w1} < T_{w2}$ then task 1 will eventually be prioritised by the robot as soon as it has completed its current task. The algorithm was developed to consider the level of the input buffer and its capacity, the level of the output storage, the output fill rate and consumer demand rates, and the profit per product index.

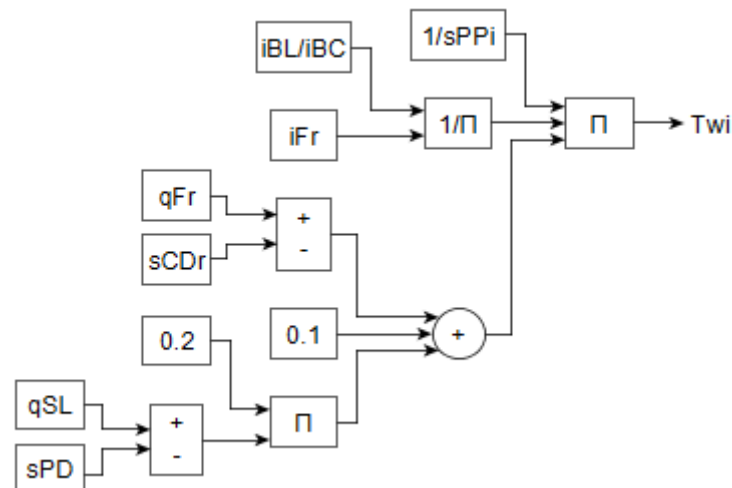


Figure 5.1. Control algorithm to determine the robot task priority.

The control algorithm was executed within the *matHand_algo.py* program (Appendix G1), which also contains function calls for the robot motion commands for each task, the initialising and updating of local variables and tags, and the execution of a simulation program named *matHand_sim.py* (Appendix G2). The structure and interaction of the programs is explained through the flow diagram in Figure 5.2.

The purpose of the simulation program was to establish a plant replica system where the generated data can be used to test the algorithm and essentially, the functionality of the framework. The code functions in the simulation program were used to fill the input buffers at pre-defined fill rates, and empty them whenever the robot was at the source location to

load material. The output storage filled when the robot was at the destination location and material was being offloaded. This data is visualised by the SCADA mimic for the application, shown in Figure 5.3.

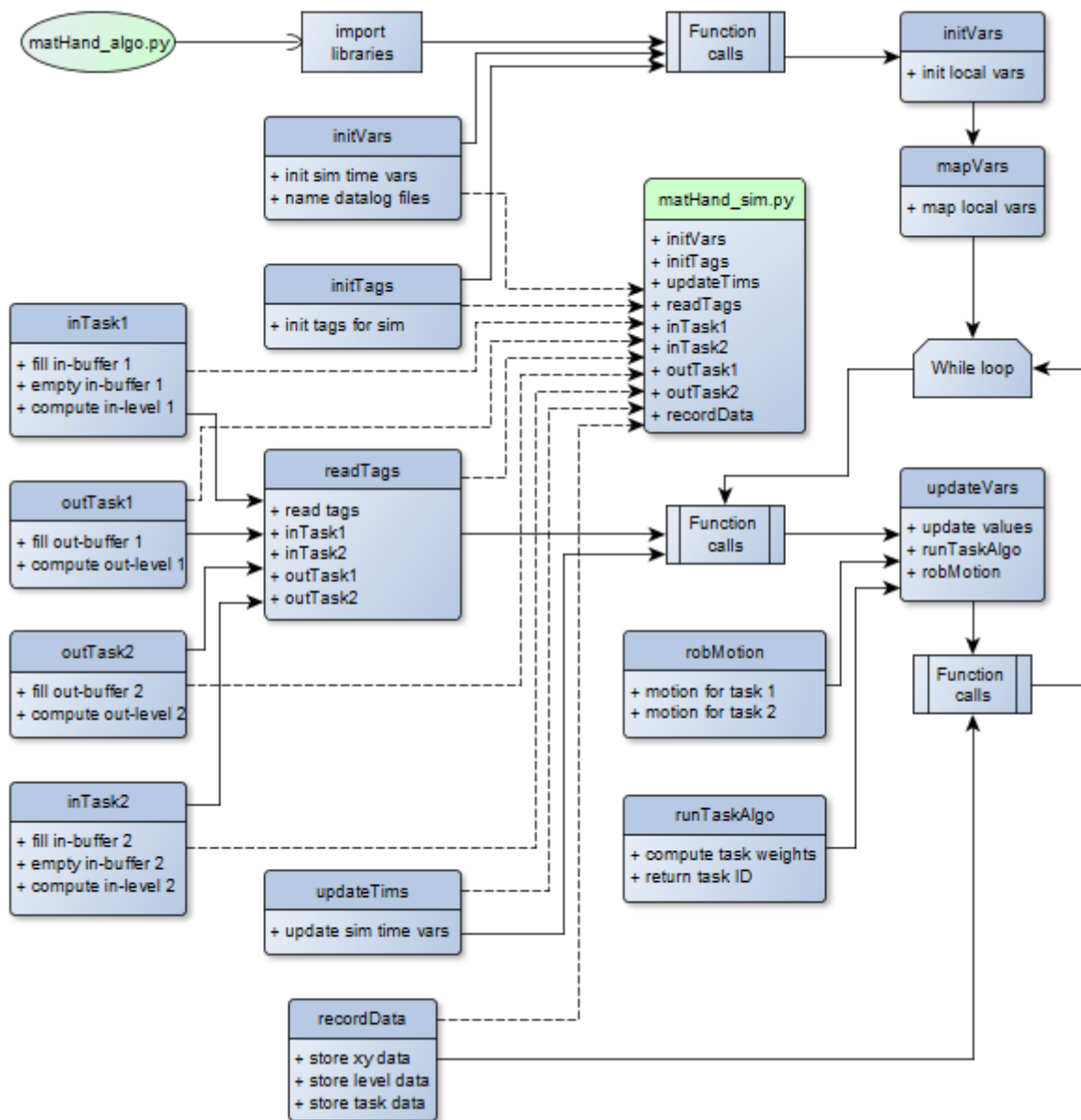


Figure 5.2. Flow diagram for the application program of Case Study I.

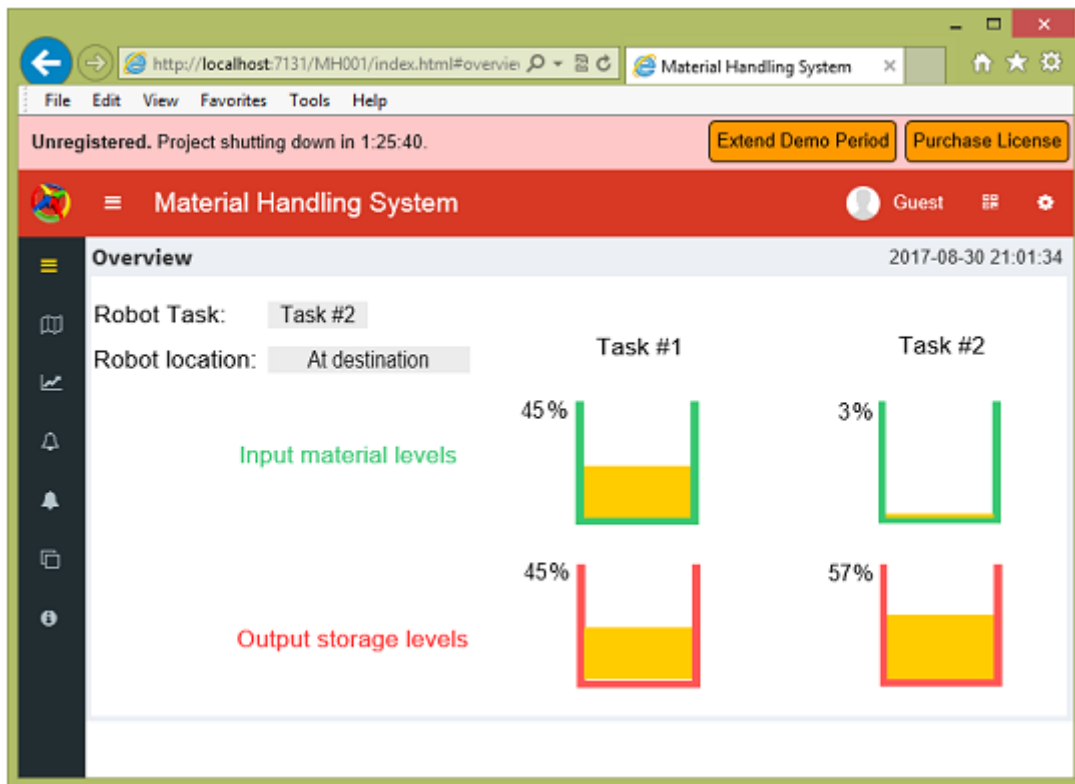


Figure 5.3. SCADA mimic for the material handling application of Case Study I.

The development of the application program involved the use of the mapping of local variables to OPC tags, a Python programming feature of the framework that was discussed in section 4.7.1. The program also includes the development of the *recordData* function, as shown in Figure 5.2, which stores the buffer levels and the robot's x-y coordinate data for use in the presentation of the simulation results.

Finally, the *robMotion* function code sets the robot goal location coordinates, depending on the task number determined by the algorithm. The low-level motion commands are actually handled by ROS, so the end-user's high-level Python program is modular, with the focus being on simulating artificial intelligent algorithms.

5.1.3 Simulation results

As previously mentioned, two simulation scenarios for this application were carried out, each biased so that the robot prioritises one task over the other. During both simulations, the load

carrying capacity, $r01_loadcap$, of the robot was set to 20 materials, while the input and output buffer capacities ($t0x_iBC$, $t0x_qSC$) were assigned 100 materials. The input buffers were simulated to fill at a pre-defined fill rate ($t0x_iFr$) from the beginning to the end of the simulation, which was terminated when either output storage $t01_qSL$ or $t02_qSL$ reached its full capacity.

5.1.3.1 First simulation

The first simulation was developed to influence a greater input fill rate at task 1. The $t01_iFr$ tag was assigned the value 30 pr/hr, whilst $t02_iFr$ was set at 10 pr/hr. The results of the simulation are shown by the plotted graphs in Figure 5.4. The graph on the left of the figure displays the input and output levels for task 1 and 2, and the impact of the load carrying mobile robot can be seen by the rise in output levels, a few seconds after the decline in input levels.

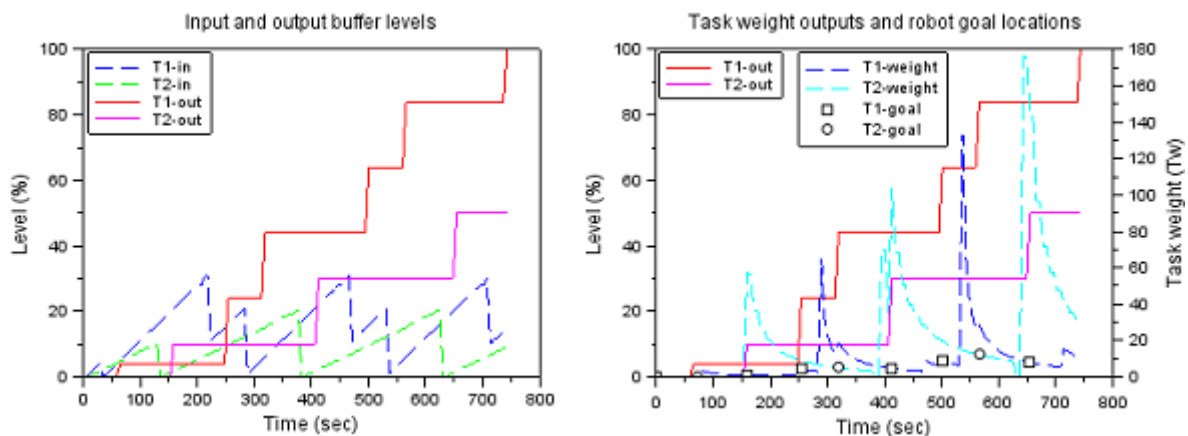


Figure 5.4. Buffer levels and task weight outputs of Case Study I, simulation-1.

The graph on the right of Figure 5.4 is much more interesting as it verifies the influence of the task-weighted algorithm on the results of the simulation. The task 1 robot goal was generated 6 times, against the task 2 goal of 3 times during the simulation. This result is expected since the input fill rate of task 1 was much higher than task 2, causing a lower weight contribution to the final TW_i output of the algorithm as per Figure 5.1. Also shown in Figure 5.4 is the rise in the corresponding output level soon after the task goal decisions were made for the mobile robot. For example, at 500 seconds into the simulation, the robot's goal

was task 1, and at 550 seconds it can be seen that there was a significant rise in the output level of task 1.

Figure 5.5 is an x-y coordinate map position plot of the robot during the simulation, which can be viewed in conjunction with the Stage map in Figure 4.19 to visualise the path required by the robot to travel from one location to another. The figure illustrates the distribution of time spent by the robot between the source (input buffer) and destination (output storage) locations for task 1 and 2, where it is clear that more time was spent on task 1. Once again, this is an expected result since the algorithm computed twice the number of robot goals for task 1 than task 2.

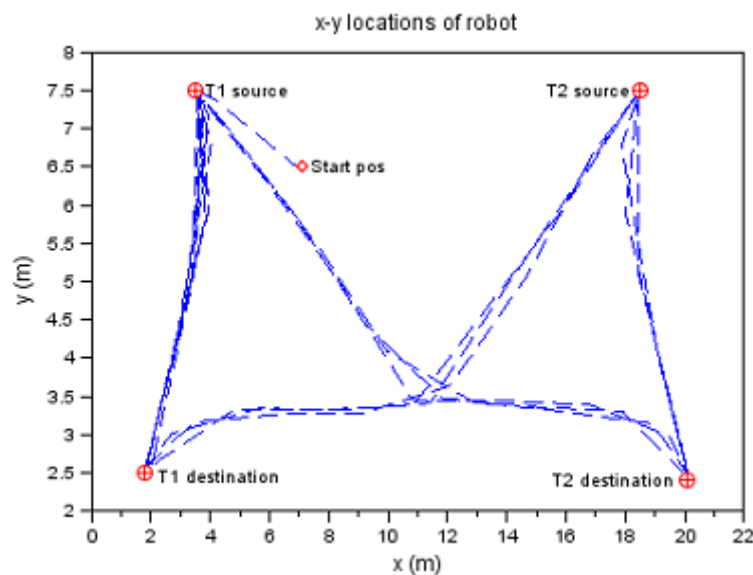


Figure 5.5. x-y robot coordinates during simulation-1 of Case Study I.

5.1.3.2 Second simulation

The results of the second simulation are given by Figure 5.6 and Figure 5.7, arranged in a similar manner to the first simulation. The idea behind the tests performed here was to make the robot more biased towards the execution of task 2. This was achieved by assigning a difference in supply chain profit per product index, $t0x_sPPi$. The $t01_sPPi$ tag was assigned the value 0.5, whilst $t02_sPPi$ was set at 1.5, and the input fill rates for both tasks were made

equal at 30 pr/hr. The results in Figure 5.6 reveal a minor bias towards the task 2 goal, being generated 5 times against the task 1 goal of 4 times during the simulation. Figure 5.7 also shows a greater concentration of time spent by the robot at the task 2 locations.

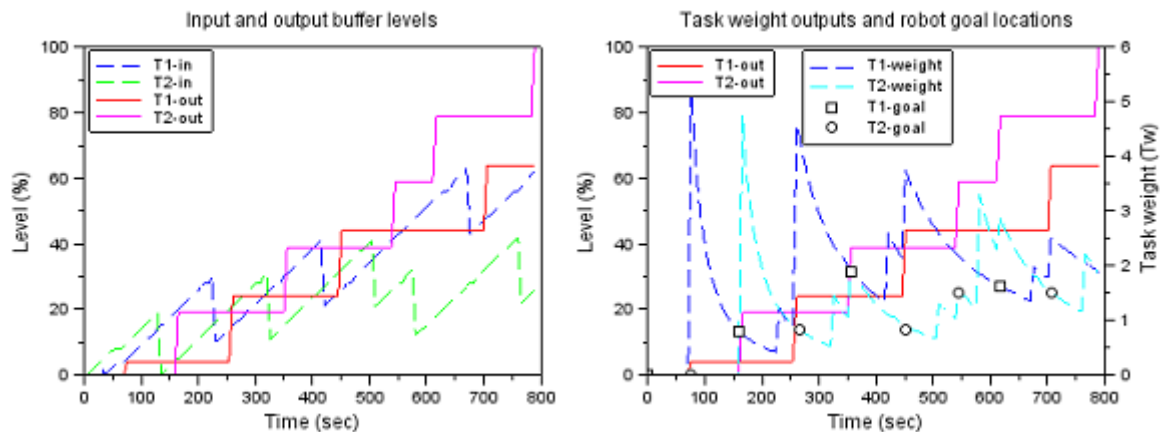


Figure 5.6. Buffer levels and task weight outputs of Case Study I, simulation-2.

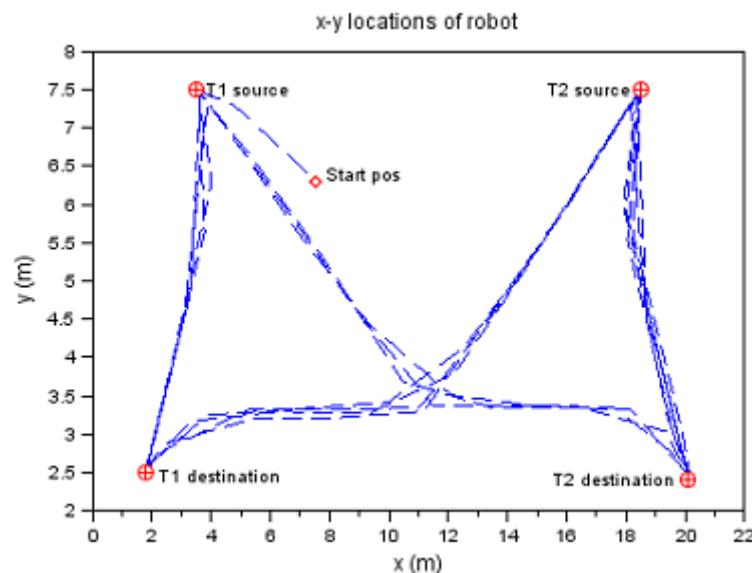


Figure 5.7. x-y robot coordinates during simulation-2 of Case Study I.

Comparing Figure 5.4 to Figure 5.6, one can notice a similar occurrence of data ‘spikes’ in the task weight outputs of $T1-weight$ and $T2-weight$. These results are contributed to the computation of the input buffer level product in the algorithm described by Figure 5.1. A low iBL (input buffer level) value will result in a high T_{wi} output. This result can also be verified

by the times at which the ‘spike’ occurs versus the time in the simulation where a low iBL is seen. For example, in Figure 5.6 at approximately 150 seconds into the simulation, $T2-weight$ has a value of almost 5 and at that particular time, $T2-in$ (shown in the left of Figure 5.6) has a value of almost 0%.

5.2 Study II: Cooperation in heterogeneous mobile robot teams

The purpose of this study was to demonstrate the ability of the framework to control heterogeneous mobile robot teams in manufacturing bottleneck scenarios and cooperation tasks.

5.2.1 Description of the study

The application of the framework in this case study also pertained to the robot material handling scenario between two tasks, discussed in the previous study. The difference with the application discussed here was the introduction of additional mobile robots to create situations where robot cooperation can occur to produce common goal outcomes.

This study also applied the Python programming feature discussed in section 4.7.2, which was developed for heterogeneous mobile robot systems. Two types of multi-robot simulations are introduced in section 5.2.3 to show the ability of the framework to adapt to different scenarios:

- The first simulation involved a manufacturing bottleneck scenario with two robots and allowed robot #1 (R-01) to work on task 1 or task 2, depending on the output from the Python algorithm. Robot #2 (R-02) focused on task 2 only.
- The second simulation demonstrated cooperation among three robots. R-01 worked on task 1 only and R-02 worked on task 2 only. Robot #3 (R-03) did work on task 1 or task 2, depending on the output from the algorithm.

5.2.2 Application development

The OPC tag definitions used in this application were similar to the list defined in Table 5, with the addition of tags for the second and third robots. Hence, the prefixes to the robot related tags are *R02* and *R03* for robot #2 and robot #3 respectively. The development of the SCADA mimic is illustrated in Figure 4.22, which shows the current task and location of each robot in the system.

The Python code development for this application is similar in structure to the documentation in Appendix G, with the addition of the modular motion code for the additional robots, and the application of the *checkRobTaskCompat* function discussed in section 4.7.2. Table 6 and Table 7 outline the robot and task configuration parameters used for the first and second simulations respectively.

Table 6. Robot and task compatibility table for the first simulation of Case Study II.

	<i>L</i>	<i>W</i>	<i>H</i>	Load Cap.	Lin. speed	Ang. speed	LRF	Cam	Two wheel	Sonar	Grip.	Compat. Robot/Task
Robot 1	400	500	600	10	2	2	Yes	Yes	Yes	Yes	Yes	Task 1 & Task 2
Robot 2	500	600	700	10	2	2	Yes	Yes	Yes	Yes	Yes	Task 2
Task 1	400	500	600	0.5	0.5	1	Yes	N/A	N/A	N/A	Yes	R-01
Task 2	500	600	700	10	2	2	Yes	Yes	Yes	Yes	Yes	R-01 & R-02

Table 7. Robot and task compatibility table for the second simulation of Case Study II.

	<i>L</i>	<i>W</i>	<i>H</i>	Load Cap.	Lin. speed	Ang. speed	LRF	Cam	Two wheel	Sonar	Grip.	Compat. Robot/Task
Robot 1	400	500	600	0.5	0.5	1	Yes	No	Yes	No	Yes	Task 1
Robot 2	500	600	700	10	2	2	Yes	Yes	Yes	Yes	Yes	Task 2
Robot 3	400	500	600	10	2	2	Yes	Yes	Yes	Yes	Yes	Task 1 & Task 2
Task 1	400	500	600	0.5	0.5	1	Yes	N/A	N/A	N/A	Yes	R-01 & R-03
Task 2	500	600	700	10	2	2	Yes	Yes	Yes	Yes	Yes	R-02 & R-03

5.2.3 Simulation results

For the first and second simulation, the load carrying capacity ($r0x_loadcap$) of each robot were either set to 10 or 20 materials, while the input and output buffer capacities ($t0x_iBC$, $t0x_qSC$) were 100 materials. The input buffers were simulated to fill at a pre-defined fill rate ($t0x_iFr$) from the beginning to the end of the simulation, which was terminated when either output storage $t01_qSL$ or $t02_qSL$ reached its full capacity.

5.2.3.1 First simulation

The first simulation was created to influence a bottleneck at the task 2 input buffer by: 1) making $r01_loadcap$ set to 20 materials and $r02_loadcap$ set to 10 materials, and 2) keeping the input fill rates of both tasks constant at 30 pr/hr. Since R-02 was only dedicated to task 2, R-01 assisted R-02 whenever the cooperation algorithm outputted the corresponding result. The results of the simulation are shown by the plotted graphs in Figure 5.8.

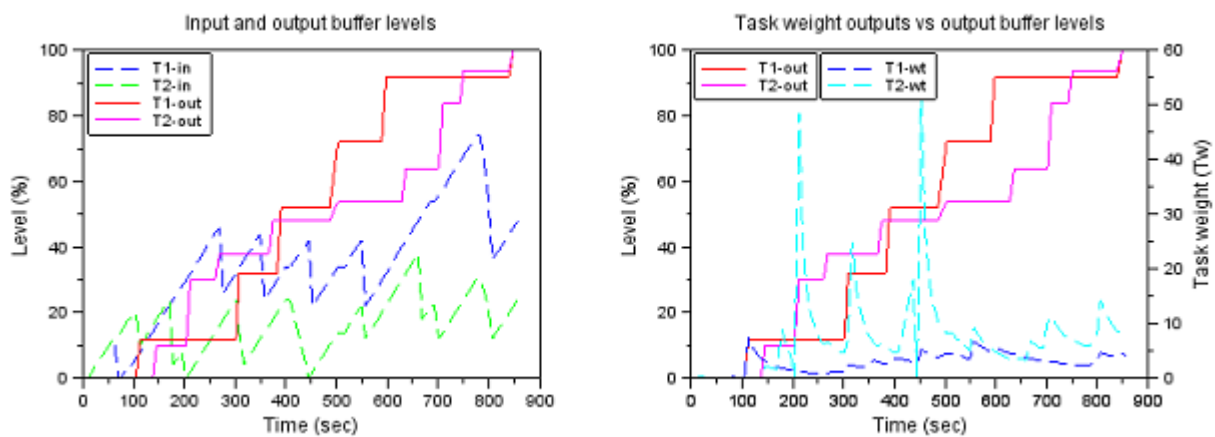


Figure 5.8. Buffer levels and task weight outputs of Case Study II, simulation-1.

As per the results in the previous case study, the graph on the left of the figure displays the input and output levels for task 1 and 2, whilst the graph on the right of shows the influence of the task-weighted algorithm on the results of the simulation. The times at which R-01 assisted R-02 can be seen by the ‘jump’ in the $T2\text{-out}$ buffer level at approximately 200 and 700 seconds into the simulation. This is verified by the result of the task-weighted algorithm

shortly before the ‘jump’ since the $T2-wt$ values are slightly lower than the $T1-wt$ values, causing R-01 to choose task 2 over task 1 at those particular points in the timeline.

Another verification of R-01 cooperating with R-02 in this simulation can be seen by the x-y coordinate map position plot of the two robots during the simulation, given by Figure 5.9. These plots can be viewed in conjunction with the Stage map in Figure 4.19 to visualise the path required by the robots to travel from one location to another. The figure shows the concentration of time spent by each robot between the source (input buffer) and destination (output storage) locations for task 1 and task 2. It is expected that R-02 only travels between the source and destination of task 2, however, R-01 shares its interests between task 1 and task 2, and it can also be seen that it executes the work for task 2 twice in the simulation, thereby verifying the results shown in Figure 5.8.

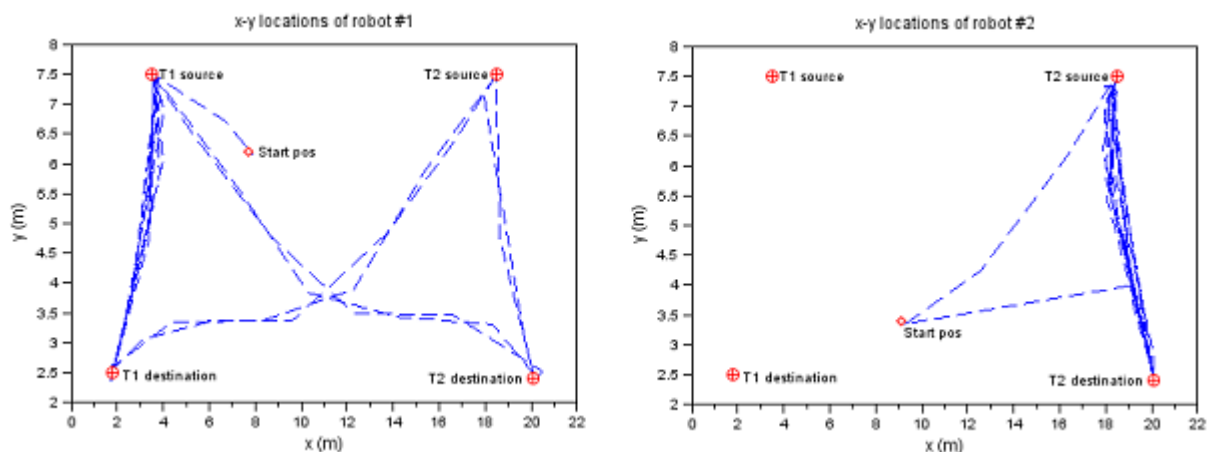


Figure 5.9. x-y robot coordinates during simulation-1 of Case Study II.

5.2.3.2 Second simulation

The second simulation was created to influence an equal output fill rate for both tasks by: 1) making $r01_loadcap$ and $r02_loadcap$ set to 10 materials, and 2) keeping the input fill rates of both tasks constant at 30 pr/hr. Since R-01 was only dedicated to task 1 and R-02 only dedicated to task 2, R-03 was made the cooperating robot agent by assisting at task 1 or 2 whenever the cooperation algorithm outputted the corresponding result. The results of the second simulation are shown by the plotted graphs in Figure 5.10 and Figure 5.11.

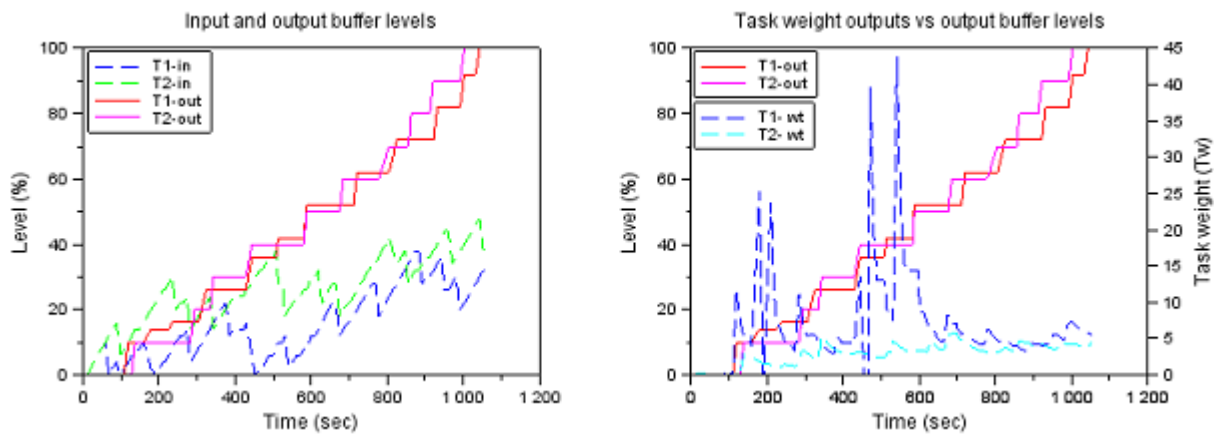


Figure 5.10. Buffer levels and task weight outputs of Case Study II, simulation-2.

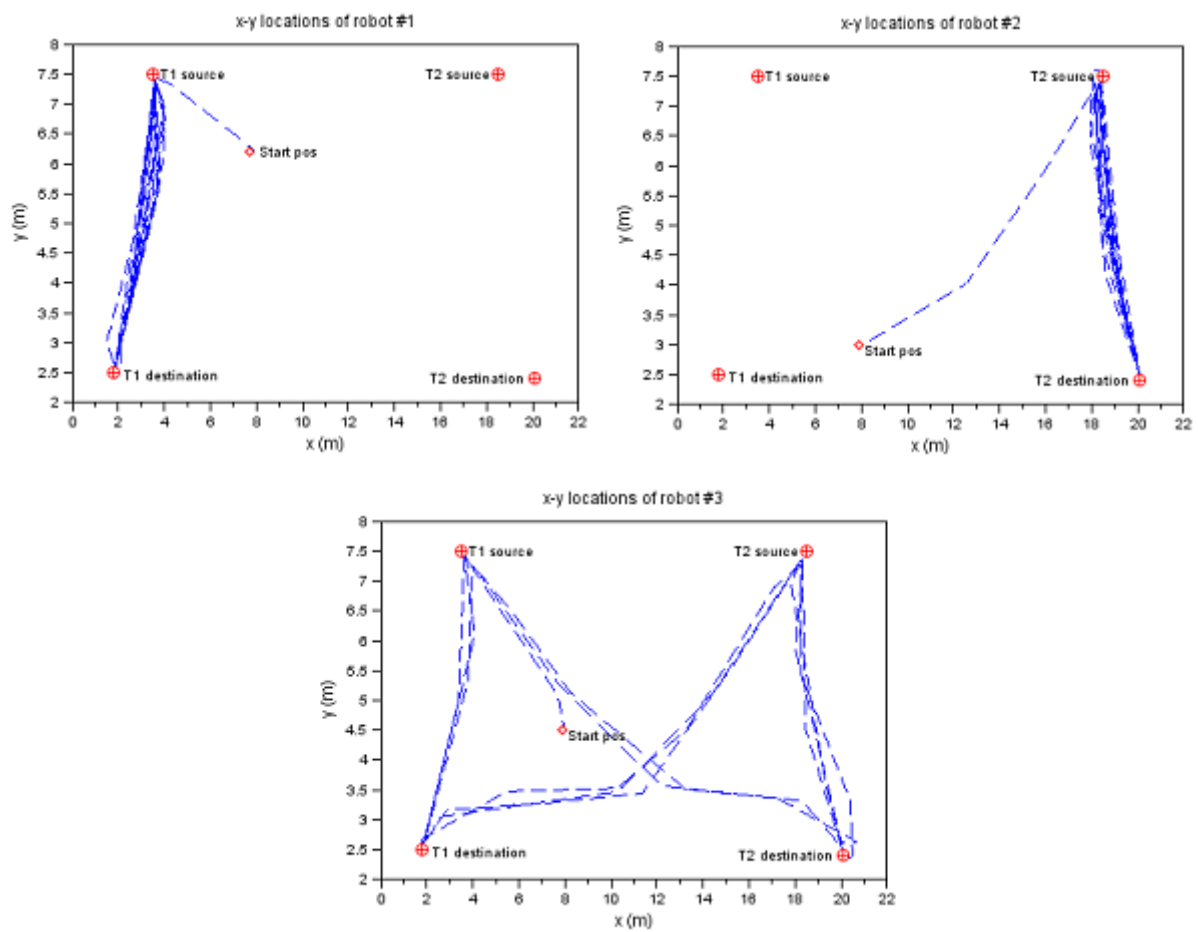


Figure 5.11. x-y robot coordinates during simulation-2 of Case Study II.

The goal of the simulation was achieved as it can be seen from the results of *T1-out* and *T2-out* that each line graph ‘follows’ the other. It is also expected from the x-y position plot of the three robots during the simulation (given by Figure 5.11), that R-03 evenly splits its work during the simulation due to the constant input fill rates of each task as well as the same load capacities of each robot; i.e. the consistency of the input variables in the simulation resulted in a similar output fill rate of each task.

In a similar analysis to the previous case study, the presence of data ‘spikes’ in the task weight outputs of *T2-wt* and *T1-wt* can be seen in Figure 5.8 and Figure 5.10 respectively. Once again, these results are contributed to the computation of the input buffer level product in the algorithm described by Figure 5.1, where a low input buffer level value will result in a high T_{wi} output.

5.3 Study III: Machine learning robots

One of the contributions of this research is to broaden the scope of the machine learning field to mobile robotics. This case study aimed to demonstrate the contribution via the application of a Python machine learning library (*scikit-learn*) in the framework. The vision of this application of the framework is for future use by researchers in applying machine learning strategies in robotic applications.

5.3.1 Description of the study

This application is an adaptation of the first case study which involved a single mobile robot, two tasks, and the transportation of material between a source and destination location. The algorithm discussed in section 5.1.2 was not used here, rather, a Support Vector Machine (SVM) algorithm was applied through the use of the *scikit-learn* Python library. The first part of the process involved a data training exercise which was discussed in section 4.7.3, thereafter, the training data was used to generate a SVM model. The model was then applied in the user algorithm with simulation inputs to determine the task goal of the mobile robot.

This study demonstrates two simulations of the SVM application. The first simulation involves the monitoring of the task input and output buffer levels to determine the goal output, whilst the second simulation includes a supply chain task demand parameter as an additional SVM data feature to influence the outcome of the simulation.

5.3.2 Application development

The flow diagram in Figure 5.12 describes the application program for this case study. Most of the functions and procedures in the figure were omitted to avoid duplication of the process already discussed in Figure 5.2. The main highlight in the figure, however, is the inclusion of the *runSVMAlgo* function.

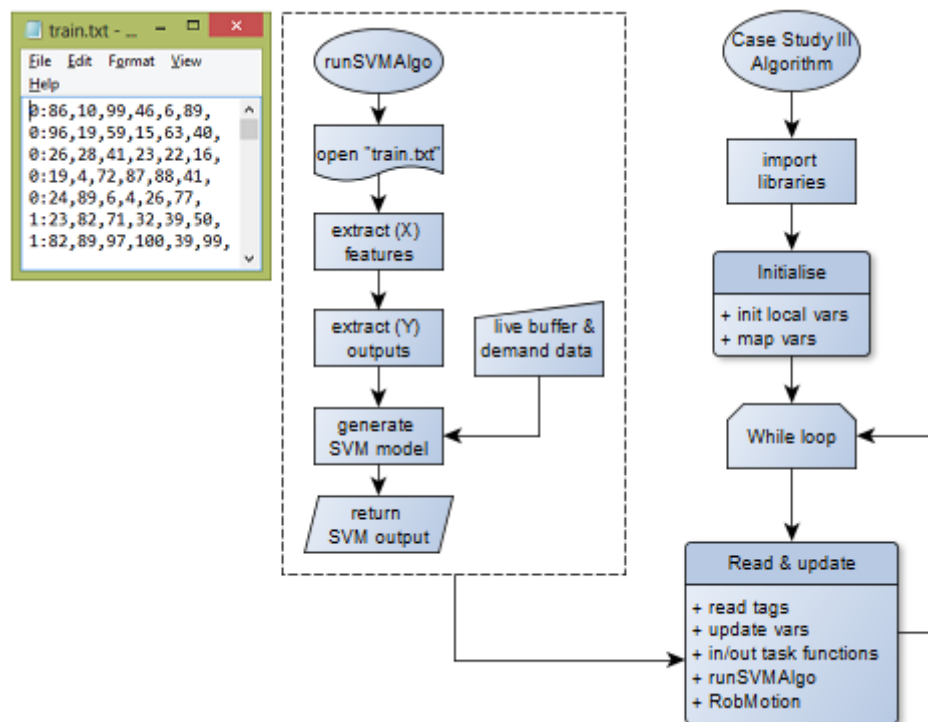


Figure 5.12. Flow diagram for the application program of Case Study III.

The developed SVM GUI application (Figure 4.25) was used to create “train.txt” files by randomly generating percentage levels every instant the data was assigned to a task. The file contains data for input and output buffers, and task demand levels for particular outcomes, i.e. whether the robot should perform task 1 or task 2 (“0” or “1” respectively in Figure 5.12).

The developed algorithm extracted the training features and outputs from the text file to create a SVM model. The data inputted to the model was the real-time simulation data of the input and output buffer levels, as well as (in the case of the second simulation) the supply chain task-demand data. The output from the algorithm was the task number, which was then used to prioritise the robot's work.

5.3.3 Simulation results

As stated in the description of this study, two simulations were performed in the application of the SVM algorithm:

- The *first simulation* trained the SVM model to produce an unbiased outcome. Task numbers were assigned to data training sets to influence the filling of both output buffers. For example, if $t01_qSL$ was 40% and $t02_qSL$ was 20%, then the trained output would have been task 2, since the robot will be required to fill that buffer. Of course, this also depends on the state of the input buffer as it would not make sense to assign the robot to an empty input buffer. The training process involved the creation of a “train.txt” that contained 312 data samples (lines in the file).
- The *second simulation* trained the SVM model to produce a biased outcome towards task 2 for the first half of the simulation by using the supply chain demand data (as a percentage). The bias was created by randomly generating task 1 demand levels between 0 and 50, and task 2 demand levels between 50 and 100. The “train.txt” file for this simulation contained 1414 data samples.

5.3.3.1 First simulation

The results of the first simulation are given by the plotted graph in Figure 5.13. The *SVM-out* line graph in the figure shows the trend of the output from the SVM algorithm for the duration of the simulation. The output is read as a “1” for task 1 and “2” for task 2. Noticeable in the figure is the rise in the output levels shortly after the decision of the algorithm. For example, at approximately 200 seconds into the simulation, the value of *SVM-out* was 1, and thereafter at 250 seconds, a rise in the *T1-out* level is seen.

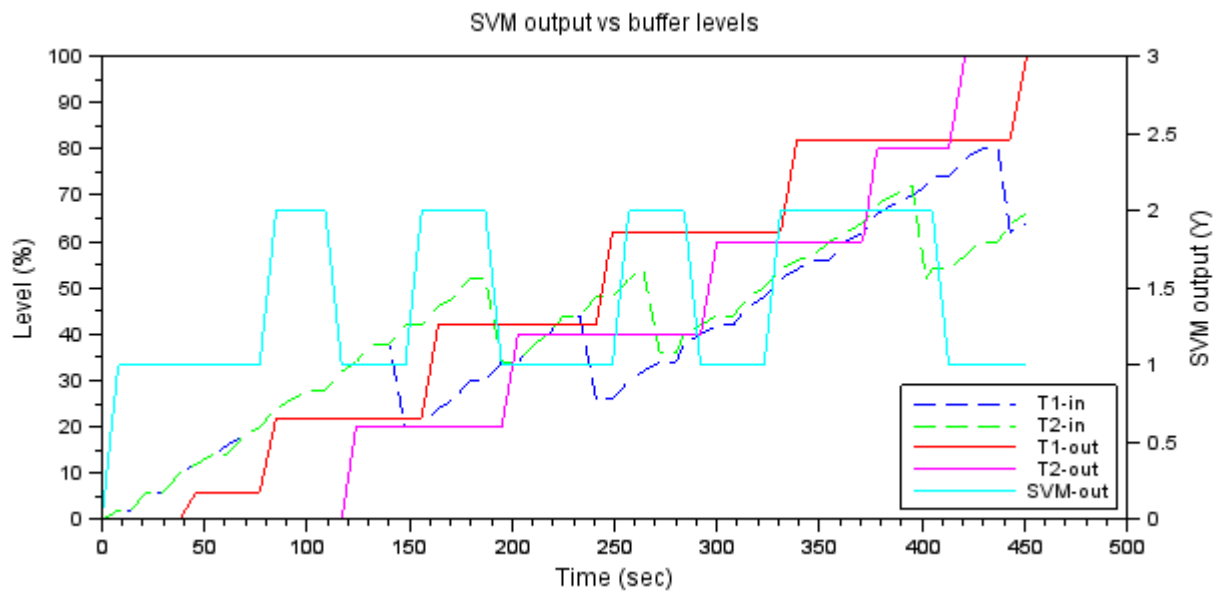


Figure 5.13. Buffer levels and SVM outputs of Case Study III, simulation-1.

The output buffer results illustrated in the figure reveal the ‘follow’ trend in the line graphs. This verifies the application of the unbiased SVM training model, since the robot was commanded to share its work load between the tasks. Another way in which the model can be verified is through an analysis of the *SVM-out* line graph. The periods in the waveform show a near equal distribution of outputs for the tasks. A total number of 11 trips were made by the robot during the simulation, 6 of which were made for task 1 while the remaining 5 were for task 2. The additional trip for task 1 can be explained by the initial one (at the start), where the load carried by the robot was not its capacity of 20 material.

5.3.3.2 Second simulation

The results of the second simulation are given by the plotted graph in Figure 5.14. The task 1 and task 2 demand level trends (*T1-D* and *T2-D* respectively) are shown in the figure, where the randomised levels between 0 and 50 is visualised for *T1-D*, and 50 to 100 for *T2-D*. As mentioned in the description of this simulation, the training process involved a bias towards task 2 for the initial phase of the simulation. This bias is clearly noticed in the result of *T2-out* and *T1-out*, since *T2-out* ramps up to 60% long before *T1-out*. The result also verifies the execution of the SVM model, since the model was trained to prioritise the filling of *T2-out* early in the simulation due to the high demand.

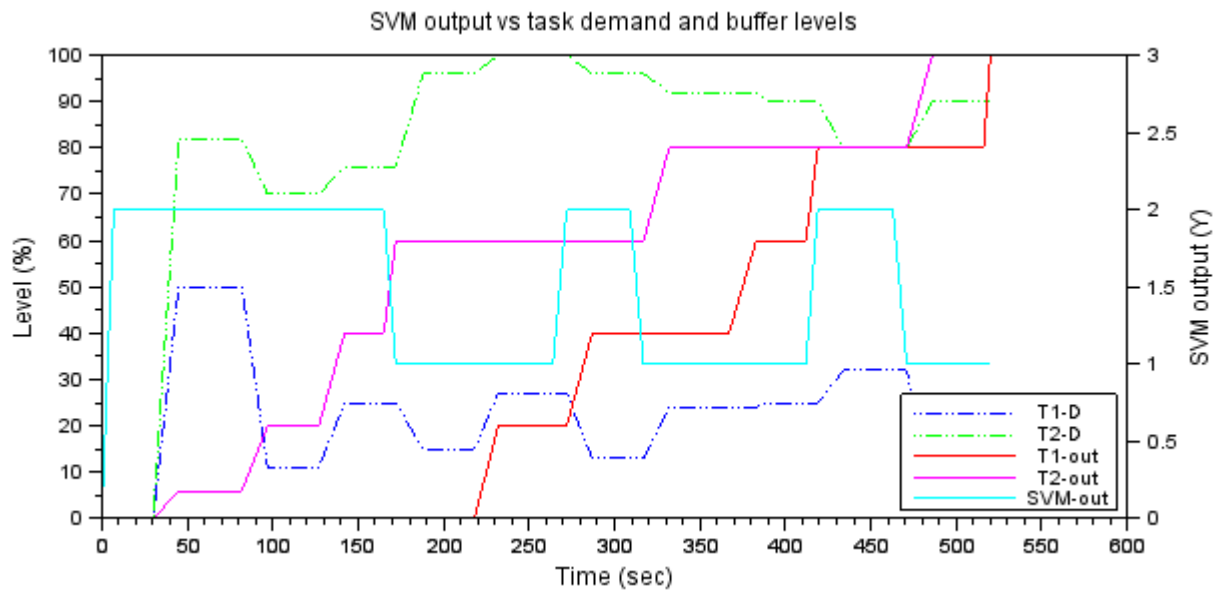


Figure 5.14. Buffer, demand levels and SVM outputs of Case Study III, simulation-2.

The total number of trips made by the robot during the simulation was 12; 7 of which were made for task 2 while the remaining 5 were for task 1, and thus confirms the bias towards task 2. In comparison with the first simulation, this simulation was 84 seconds longer, contributed to the additional trip by the robot. The additional trip is noted during the first 100 seconds into the simulation, where the focus on one task caused the emptying of material quicker than the input buffer fill rate, thus resulting in the non-utilised loading capacity of the robot.

5.4 Study IV: Integration with a third party system

The aim of this case study was two-fold: 1) to prove the integration of the framework with an advanced technological third party system, thereby enhancing its ability to form part of an Industry 4.0 network, and 2) to test the robustness of the data communication link between the framework and the third party system.

5.4.1 Description of the study

Section 3.3.2 explored various industrial data communication standards, among them being EtherCAT, which together with OPC-UA, supports Industry 4.0 convergence of information and automation technologies [92]. The work in this study demonstrates the integration of the framework with a Beckhoff control system, which uses EtherCAT as its data communication standard [91].

The system comprised of the Beckhoff soft-PLC, Beckhoff OPC Server, and the user's Python OPC Client and algorithm. The ROS middleware for robot control was non-existent in this application since it was assumed that the communication will be executed via the EtherCAT network, or the PLC digital I/O hardware. Another reason for excluding the ROS component was to satisfy the aim of the study, which was to test the reliability of communication between the framework and another system (other than ROS).

The application involved the switching between three tasks, each having a fixed execution time, simulated by a PLC timer. A handshake signal (“robot ready”) indicated when a task was complete and determined when the next task was to be switched by the Python application. The robustness of the OPC data communication link was tested through the monitoring of delay times between the switching of tasks. The next section discusses the algorithm development of the sub-systems in this study.

5.4.2 Application development

Figure 5.15 is a flow chart description of the PLC and Python algorithms developed for this application. The PLC program was developed in the Beckhoff TwinCAT 3 programming environment (Figure 5.16), where the *ROI_Ready* and *TaskID* variables were defined, and thereafter implemented as OPC tags by the Python Client and TwinCAT OPC Server. The Python algorithm checks whether the simulated *ROI_Ready* tag is set to begin the assignment of tasks (through the *TaskID*), and clears the tag when a new task has been assigned. On the other end, the PLC program sets *ROI_Ready* when a task is complete, which is determined by the elapsed time for the particular task.

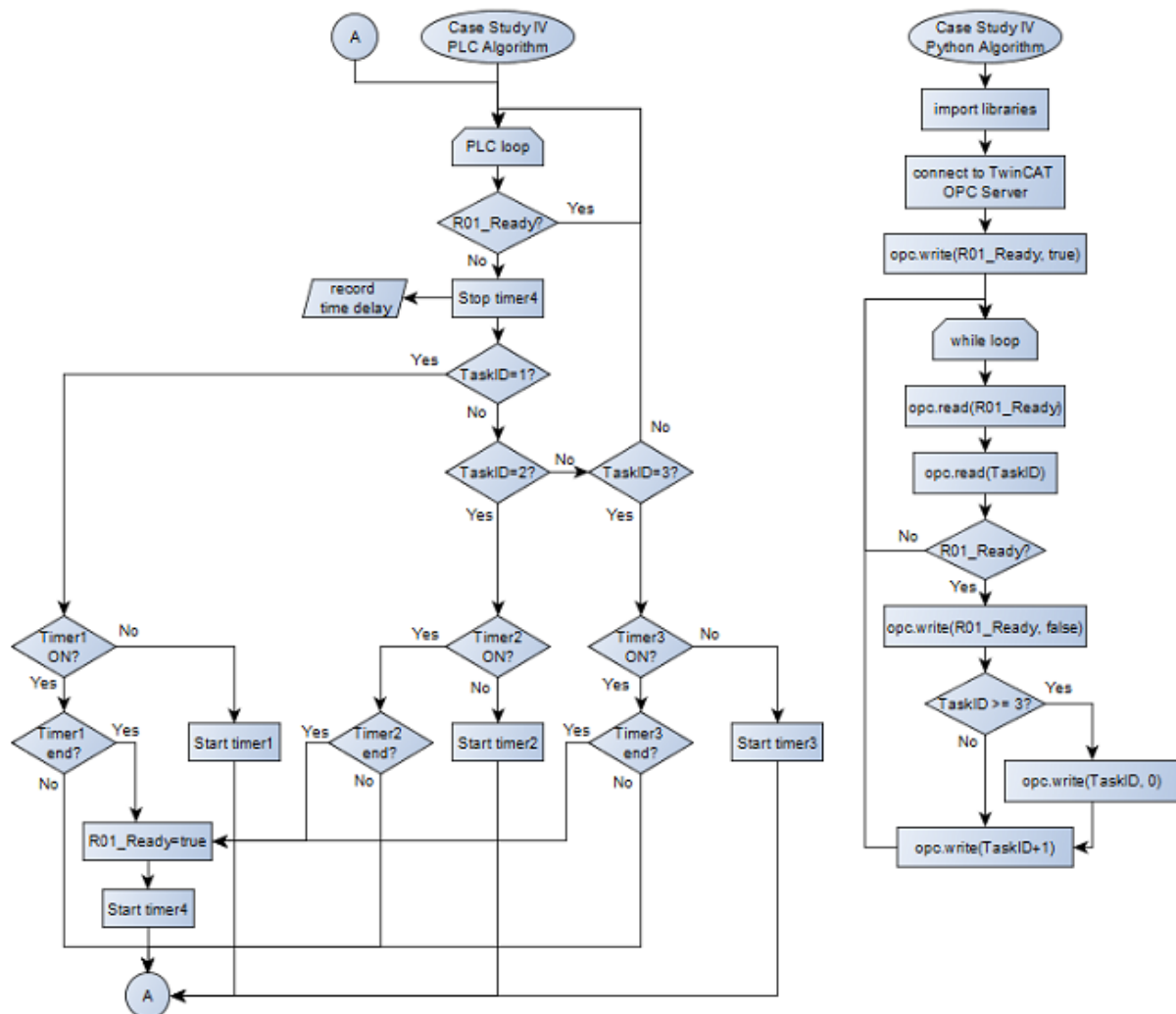


Figure 5.15. PLC and Python algorithms for Case Study IV.

The *TaskID* is incremented by 1 in the Python program after every task is complete. Three tasks were configured, each containing ID's 1, 2, and 3, and the program was developed such that the value of the *TaskID* cycles between the three ID's (as shown at the bottom of the Python algorithm in the figure).

PLC timers were used to simulate the time a robot would take to complete a particular task. Each task was assigned a timer variable, and only a single timer was started at any point in time through the control of the *TaskID* variable. A fourth timer was also used to record the time delay between the switching of tasks. The time delay measurement was utilised to evaluate the quality of the OPC communication exchange between the two sub-systems.

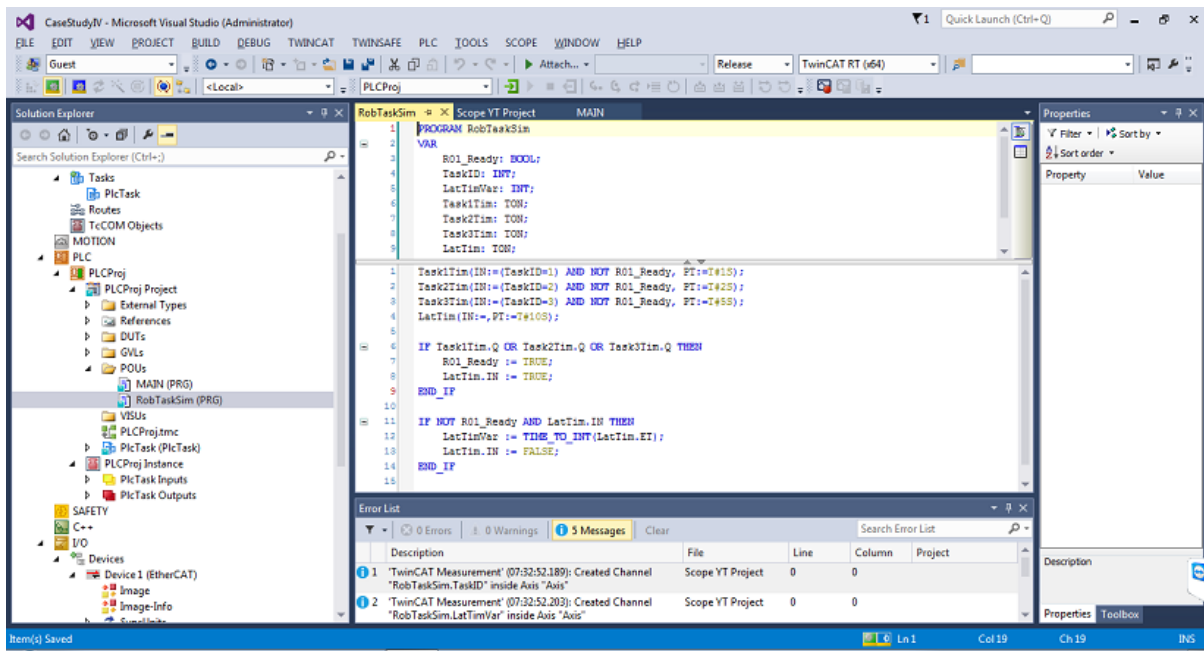


Figure 5.16. TwinCAT 3 PLC programming environment.

5.4.3 Simulation results

The simulation of the system involved the application three scenarios, each differing from the other by the task time configuration. Each PLC task timer simulated the robot task durations in accordance with these configurations:

- The *slow* communication scenario simulated task 1, 2, and 3 time periods to 10, 20, and 30 seconds respectively. The results of this simulation are given by Figure 5.17.
- The *average* communication scenario simulated task 1, 2, and 3 time periods to 1, 2, and 5 seconds respectively. The results of this simulation are given by Figure 5.18.
- The *fast* communication scenario simulated task 1, 2, and 3 time periods to 100, 200, and 500 milliseconds respectively. The results of this simulation are given by Figure 5.19.

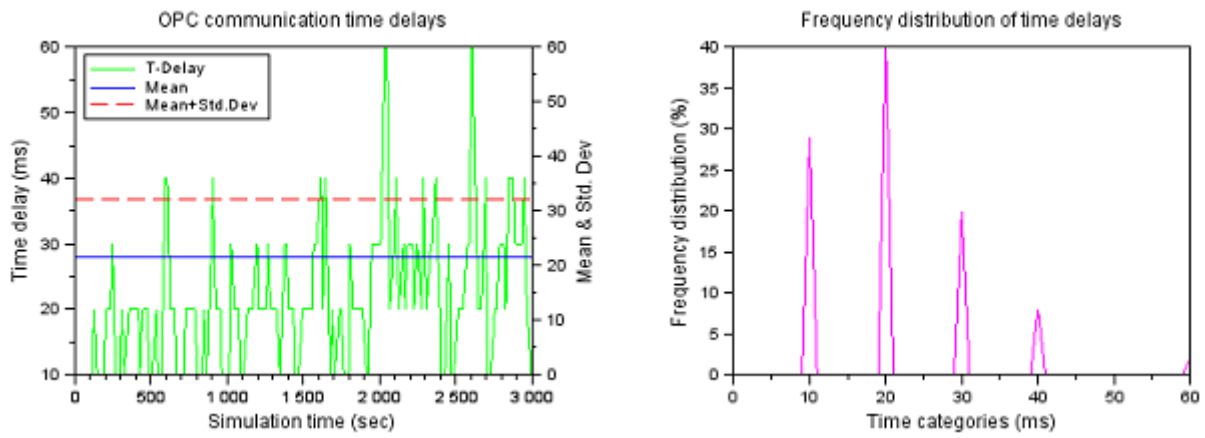


Figure 5.17. Time delay analysis for slow communication intervals.

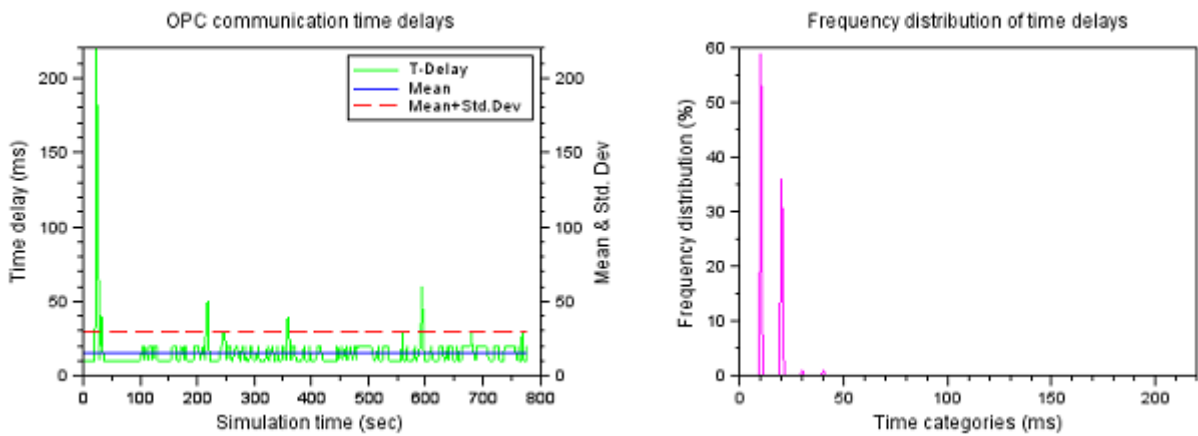


Figure 5.18. Time delay analysis for average communication intervals.

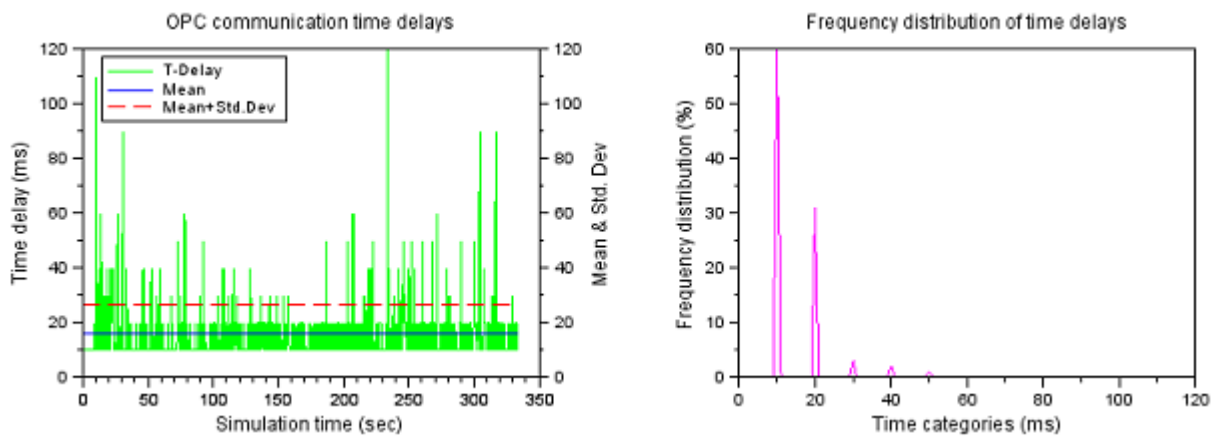


Figure 5.19. Time delay analysis for fast communication intervals.

The three figures show the results of the OPC communication time delay for the duration of the simulations, as well as the mean and standard deviation statistical indicators, given by Equation 5.2 and Equation 5.3 respectively:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (5.2)$$

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (5.3)$$

Where: x_i is the individual time delay sample in the data set.

n is the total number of samples in the data set.

Another statistical feedback illustrated in the figures is frequency distribution of the time delays, arranged in millisecond time categories. The categories are multiples of 10ms due to this being the PLC scan cycle time, thus determining the resolution of the time delay samples. The distribution gives a direct indication of the percentage contribution of each category to the total number of samples in the simulation.

Table 8 lists a comparative breakdown of the time delay analysis for the three simulations. The total number of samples differs across the simulations due to the variations in the task times as well as the total simulation time. The simulation time for the *slow* communication scenario was generated in 50 minutes to increase the number of samples due to the long duration of the tasks.

Table 8. Time delay analysis for the three simulations in Case Study IV.

	Slow rate	Average rate	Fast rate
Mean (ms)	21.5	15.6	15.8
Std. Deviation (ms)	10.6	14.0	10.8
Mean + Std. Dev. (ms)	32.1	29.6	26.6
Total no. of samples	150	294	1117
Worst delay (ms)	60	220	120
Freq. distrib: ≤20 ms (%)	70	95	92
Freq. distrib: >20 ms & ≤40 ms (%)	28	3	5
Freq. distrib. >40 ms (%)	2	2	3

A fundamental field listed in the table is the *worst delay* feedback, which reveals the worse case value of 220ms across all three simulations. This result is well within the required scan cycle time for the higher-level decision making algorithms, which is ≤ 1 second as defined in the functional design specifications in section 3.7. The result also satisfies the specification of the OPC tag scan rate, being ≤ 500 ms.

The mean and standard deviation statistics are located in the lower end of the spectrum for all three simulations, which shows that the time delays are generally short, even in the fast rate simulation. This conclusion is also verified by the frequency distribution analysis, revealing that time delays of ≤ 20 ms contributed to the majority of the distribution. The time delays greater than 40 ms only allot to between 2% and 3% of the total sample set, and even in those cases, the worst delays are well within the specification.

A noticeable difference in the *slow rate* result (Figure 5.17), compared to the other two, was the increase in time delays 30 minutes into the simulation, which determined the 70% (versus 95% and 92%) frequency distribution value listed in Table 8. A possible reason for this result was the loading of the CPU by the OPC Server and the PLC tasks over time, and/or the effect of temperature associated with these factors. The result did not compromise the aim of the study due to the outcome of the worst case time delay, and a definitive conclusion would require further analysis which is beyond the scope of this research.

5.5 Chapter summary

This chapter presented four case studies, each verifying various aspects of the framework. The first study involved the control of a single mobile robot in a material handling scenario between two tasks. The application of the framework was tested through the development of user algorithms, the mapping of local variables and OPC tags, the communication with a ROS simulated robot, and the integration with a SCADA package.

The second study was built from the first and included an application which demonstrated the cooperation of heterogeneous mobile robots. The Python programming features for heterogeneous robots discussed in the previous chapter were implemented here. The

simulation results verified the cooperation of mobile robots in reducing bottleneck scenarios, and working towards common goals.

The third study explored the ability of the framework to support the implementation of machine learning algorithms in mobile robot applications. This was achieved through the use of Python's *scikit-learn* machine learning library, as well as an implementation of the developmental tool for SVM learning discussed in the previous chapter. The simulation results proved the performance of the SVM models, which were created during the SVM training phase of the application.

The final study verified the flexibility of the framework in its integration with a commercial third party system, whose control system uses EtherCAT as its data communication standard. This verification enhances the capability of the framework to be included in an Industry 4.0 system. The reliability of the data communication exchange was also tested here, and the results showed that the time delays are well within the specification for high-level robot decision making algorithms.

Chapter 6 Discussion

The objectives of this chapter are: 1) to discuss the benefits of the framework in relation to its functionality and application as examined in the case studies, 2) to discuss the application of the framework in other areas that are beyond the scope of this research study, 3) to discuss the related research in literature related to the functionality of the framework, and 4) to address the limitations of the framework and introduce a list of advanced features.

6.1 Benefits of the framework

This section discusses the benefits of the framework by examining its structure, functionality, and application in the various areas of robotics.

6.1.1 Popular software platforms

The framework has been developed by employing popular software platforms such as Python and ROS, thereby making it easier for end-users to grasp and utilise.

Python's popularity has grown due to its simple object-orientated structure, cross-platform support, and the existence of powerful mathematical and statistical libraries such as Numpy, Scipy, and Scikit-learn, the last of which was applied in the machine learning case study. Through the Python programming platform, it is envisioned that the end-user will be able to develop advanced, intelligent algorithms for higher-level robot control.

ROS has also grown in popularity as a robot middleware platform, and it was discussed in section 3.2 how commercial robots are now being powered by ROS [16] and its scope in industry through the ROS Industrial Consortium [17]. Through ROS, the end-user will be

able to mask robot heterogeneity by using standard interfaces and thus simplify the low level robot control process.

6.1.2 Mobile robot control

The first case study discussed in the previous chapter introduced mobile robot control in a material handling application, but the framework can be applied in other mobile robot applications, such as mobile manipulators in FMS and reconfigurable environments [103], toxic waste clean-up [21], detection of hazardous gas [104], and operations that involve confined spaces [105].

The framework is merely a tool that leaves the programming and control of advanced robots in the hands of the end-user. The GUI development allows the end-user to configure the parameters of mobile robots and tasks, which as a result, can be used during the development of robot programs. The functionality of mapping local Python variables to OPC tags further eases the development process for the end-user by eliminating the need for directly searching and coding tags in the program.

Another feature provided by the GUI that benefits the user is the task location screen, discussed in section 4.4.5, which can be used to move the robot in the environment and assign x-y data to specific task locations. This allows the robot to move to actual task locations, without the concern of robot calibration errors.

6.1.3 Distribution features

The use of the Ethernet communication infrastructure and the OPC-UA data communication standard gives the framework a distributive feature, thus the end-user development can take place at any point in the same network. The benefit of this feature is appreciated in large networks where it is more convenient and practical to be closer to the sub-system that is being commissioned.

Another benefit of the distributed framework is realised in the addition of OPC, Ethernet components, such as the SCADA system, which can be used to visualise the status of the

manufacturing processes. The verification of this feature was seen in the mobile robot material handling application, discussed in section 4.6 and section 5.1.2.

6.1.4 Robot heterogeneity and cooperation

The introductory chapter discussed the existence of heterogeneity in multiple mobile robot systems (MMRS), together with the benefit of employing a team of robots to execute tasks. The second case study demonstrated an example of such a system, where a ‘multi-skilled’ robot was able to assist other robots in the material handling application, thereby reducing the effect of a production bottleneck. This was made possible by applying the framework functions that were developed for scenarios which involve heterogeneous robots and cooperation applications. The simulation results in the case study verified the performance of the framework functions and the algorithm, since the cooperating agent was able to offer its assistance due to the induced bottleneck scenarios.

The case study also proved the support for the end-user development of algorithms pertaining to robot cooperation applications. A myriad of cooperation algorithms can now be developed and tested in heterogeneous or homogeneous robot teams. This is an attractive feature of the framework, particularly in smart factory environments where robot intelligence and flexibility are key factors in the establishment of advanced, efficient systems.

6.1.5 Machine learning

The machine learning field of research is popular in the areas of bioinformatics, and text and image recognition. The demonstration of an application of the SVM learning algorithm in the third case study proves that machine learning can also be adopted in robotic systems. This was made possible through the use of Python’s scikit-learn library, which contains a multitude of other learning algorithms that are easily available to the end-user. To name a few of just the popular algorithm groups include: neural networks, decision trees, clustering, classification, nearest neighbour, and Gaussian models. Researchers will find this benefit of Python and the framework very useful, since a wide range of algorithms are now made available within a framework developed for robotic applications.

The field of machine learning applications in robotics is also applicable to an Industry 4.0 environment, since Advanced Robotics is one of the key pillars in the Industry 4.0 concept [18]. Machine learning algorithms can be developed and optimised to create smart, intelligent robots and systems.

6.1.6 Interoperability benefits

The fourth case study verified the integration of the framework with a control system that uses EtherCAT as its data communication standard, thus showing the flexibility of the system in a real-world scenario. One could argue that the high-level control could have been programmed in the PLC. This holds true for singular systems, but becomes complex in large, heterogeneous systems with different types of PLC's, data communication protocols, and/or robotic middleware employed in the network. Hence in these systems, the Python framework would be required to establish interoperability and 'speak the common language' through the OPC protocol.

Interoperability across robotic interfaces was established in the literature survey chapter as a key requirement to the implementation of robotic systems in smart factories. The integration (discussed in section 5.4) was achieved through the OPC protocol, described as an industrial standard that specialises in device and system interoperability. Due to this feature of the OPC standard, the scope of the framework goes beyond the EtherCAT system integration and includes other Ethernet protocols such as Ethernet/IP and PROFINET, since these architectures provide OPC servers in order to integrate data into larger systems. Hence, the framework is characterised with the benefit of being included in a network with mixed Ethernet protocols, satisfying the interoperability requirement of Industry 4.0.

6.2 Further scope of the framework

The three sub-sections outlined here discuss the application of the framework in other areas (beyond the scope of this research study), such as business management systems, multi-robot resource sharing, and applications that involve fixed robotic manipulators.

6.2.1 Business management systems

The implementation of the OPC-UA data communication standard in the framework has opened its scope to the integration of business management systems such as the Enterprise Resource Planning (ERP) tool. ERP was described in chapter 3 of this thesis as a business management software package that specialises in: 1) the planning of production, product demand and supply chains, and 2) the management of material, warehouse logistics, maintenance activities, and human resources.

The integration of OPC and ERP has become a modern mechanism in linking the shop (plant) floor data to the high-level enterprise [106]. SAP, a market leader in enterprise application software, has proven this integration with the Beckhoff TwinCAT OPC-UA platform [107]. This is the same platform that was used to integrate the framework in the fourth case study. Hence, the vision of this research study sees the application of the framework in a business management setting where supply chain data, and data from other process that are connected to the ERP, can be modelled to control the low level robot resources.

The continuous data exchange and feedback between the enterprise level and robot resources on the shop floor can aid in the improvement of consumer delivery schedules, particularly in a mass customisation manufacturing environment.

6.2.2 Resource sharing

The networking feature of the framework allows for the possibility of resource sharing in a team of multiple mobile robots. Robots can use the sensory data from other robots in the network before decisions can be made, thereby optimising processes. For example, a mobile robot can choose to perform an alternative task if it discovered congestion at the primary task location through the laser data from a robot at that location.

6.2.3 Robotic manipulators

The scope of this research study covers the application of mobile robot systems, however, the results from the fourth case study shows that the use of the framework can be adapted to suit

high-speed robotic manipulator applications. This is made possible through the short time delay verification in the case study, where the worst case delay was measured at 220 milliseconds. A variety of robotic manipulator applications exist in industry, which include the following:

- Welding
- Injection moulding
- Pick and place
- Packaging
- Dispensing
- Cutting, milling and drilling
- Assembly
- Painting

An adaptation of the framework in these applications can possibly be the management of multiple tasks in locations that surround the fixed, multi-axis robot, through the use of developed algorithms by the end-user.

6.3 Related research

This section discusses research in literature related to the functionality of the framework. Due to the originality of the framework development, there is no direct comparison; however, the related aspects in literature are discussed, such as systems that support remote robot programming in single and multi-robot systems, networked robots, control of heterogeneous robots, and robot cooperation.

The remote programming of networked robots within a robotics tele-laboratory has been a research effort by Marin et al. [108] to enable students to remotely program experiments via the Internet. The Simple Network Robot Protocol (SNRP), which is a web-based protocol, was used to permit the integration of robots and sensor devices in the system. Through the research project, students have access to the following equipment:

-
- A conveyor belt with sensors at each end, allowing objects to be moved in any direction, controlled at various speeds.
 - A Motoman manipulator to move objects from the conveyor belt and classify them on an auxiliary table.
 - A camera mounted on the manipulator, which implements visual servo controls and object tracking algorithms.
 - A network camera mounted on top of the conveyor belt, which video streams objects on the belt in real-time.

Each piece of equipment was connected to a robot server, responsible for the remote control access and programming through the SNRP servers. Devices are accessed through the Internet by accepting TCP and UDP connections, made possible by SNRP. SNRP also provides the possibility of using other transport protocols, such as the Real-time Transport Protocol (RTP), to improve the robustness of the network connection. The research study presented by Marin et al. [108] provided two experiments that demonstrated the successful implementation of the remote programming system.

The programming of autonomous mobile robots in a MRS system was researched by Tucac and Garcia [109]. The study involved a decision-making process undertaken by each robot in the network to achieve cooperation. The process includes: 1) each robot decides on the best action to execute based on its perception; 2) robots interact with each other and can reconsider their decision by taking into account the decisions that are being processed by other robots. The second part of the process involved communication through the use of a set of primitives that were implemented in the Logic Programming language. The framework was applied in a cleaning task application, where each robot in the team can find a box and interact with others to determine which one should pick up the box.

Another research study [110] discussed the development of “Protoswarm”, a language for programming multi-robot systems in a high level. The application pertained to a swarm of robots, where programmers were allowed to develop code without actually considering the details of individual robots. This was made possible through the control of a single virtual spatial computer, and the computations in the computer are executed by local interacting

robots. The computational models that are created are based on manifolds of space that process the code, called the “Amorphous Medium”. Some of the assumptions made in the research are the population of a finite number of robot agents in the medium, and the communication between neighbours within a fixed distance. Experiments were performed on a group of 40 autonomous mobile robots, each equipped with bump and light sensors, infra-red communication sensors, and localisation systems. The results proved to be robust and scalable, and verified the objectives of the Protoswarm programs.

Trifa et al. discussed the control of heterogeneous robotic swarms over the Internet using an SOA, Web Services (WS) architecture [111]. The idea behind this research was to use the benefit of modular software components exchanging data over HTTP, and the distributed characteristic of the system eliminates the need of a centralised controller. The research discussed the creation of a central server, built to act as a gateway between the group of robots and the end-user. The server implemented a WS which allowed for the real-time data retrieval of individual robot status as well as the system as a whole. An experiment with a team of e-puck robots showed the performance of various robot actions such as setting motor speeds, illuminating LEDs, and reading proximity sensors.

The advantage that the framework has over the related research discussed in this section is its support for integration and interoperability in an industrial, smart factory environment. This is made possible through an implementation of the international, industrial standard, OPC data communication protocol. Another benefit is its integration with the ROS middleware, which in itself, supports control for a wide range of industrial robots and is a popular choice for researchers as well.

6.4 Limitations and advanced development

The purpose of this section is to address the limitations of the framework and to introduce a list of advanced features that should be implemented as improvements to the system.

As discussed in chapter 3, a hybrid control architecture was adopted for the control of the mobile robots. The centralised part of the scheme is attributed to the OPC Server and this

makes it a single point of failure in the network. In order to reduce this risk, a redundant OPC Server can be implemented to ensure the availability and robustness of the system. This can be achieved in one of three ways; either cold, warm, or hot standby modes, depending on the time requirement of having the secondary system online. A hot standby would require the simultaneous operation of both primary and secondary OPC Servers, each using separate hardware so that there is no single point of failure. In the event of a primary failure, the switchover is instantaneous with no data loss, contrasted to the cold or warm cases where the recovery process is slower. The two OPC Servers would typically connect to a redundancy management system that is responsible for the switch over, data integrity, and maintenance of the single point of connections between OPC Clients.

In an Industry 4.0 factory, where the ease of access to data is essential, the establishment of security measures is critical to ensure that valuable data is protected against theft or loss. The OPC component of the framework does contain security protocols that ensure: 1) the authentication of clients and servers, 2) the authorisation of users, and 3) the integrity and auditing of communication between clients and servers [112]. In addition to this, OPC security is based on industry standard security algorithms and can be scalable to meet the environment and application requirements [112].

The ROS component of the framework is however, insecure in its communication protocol. The developers of ROS are currently implementing SROS, a set of security enhancements to ROS [113]. The motivation behind this development is the increase trend in cyber threats which poses great risks in industrial and home environments where robots are beginning to be integrated in Industry 4.0 and IIoT systems respectively.

The advanced development features of the framework include the following:

- Organisation of the framework code in a Python package, making it easier for the end-user to utilise, since it would require a simple installation process in a Windows or Linux OS.
- The creation of user access levels in the GUI that permits some users to read or write parameters and others to read only.

-
- The use of dynamic IP addresses (as opposed to the current system of static addresses) or MAC addresses of the robots in the network. The implementation of this feature will make the framework more easy to use, especially in fast paced FMS environments.
 - Support for the development of end-user algorithms in other programming languages, since the present limitation is Python. The design factors to consider are: 1) the ability of integration with the OPC protocol, and 2) the support for machine learning algorithm development through other programming languages. The inclusion of this advanced feature will increase the scope of the framework to a wider audience who may not be familiar with Python.
 - Functionality for the application of robotic swarm systems. This will require the performance test of the framework in these scenarios, where design factors to consider are communication bandwidth limitations and the local robot control methodology with regard to localisation and navigation in the large network of robots.

6.5 Chapter summary

The discussion in this chapter began by describing the benefits of the framework, which included the use of popular software platforms of Python and ROS, and features that enable the control and cooperation of heterogeneous mobile robots in smart factory environments. A further scope of the framework was also discussed, pertaining to areas such as business management systems, multi-robot resource sharing, and applications that involve fixed robotic manipulators.

Section 6.3 surveyed research related to the functionality of the framework. Some of the related aspects that were discussed included systems that support remote robot programming in single and multi-robot systems, networked robots, control of heterogeneous robots, and robot cooperation. The main conclusion from the survey was the industrial interoperability advantage that the framework in this study has over the related research.

The final section in this chapter discussed the two design sources of the framework that contribute to its limitations. The first one is the centralised OPC Server which creates a single

point of failure in the system. The solution to this drawback was discussed as an implementation of a redundant OPC Server that would use separate hardware to eliminate the single point of failure and increase the robustness of the system. The second limitation that was addressed is the issue of security, particularly in smart factory environments where data integrity and data protection are key requirements. The insecure communication protocol in the ROS component of the framework was identified, and it was also mentioned that the developers of ROS are currently implementing SROS, a set of security enhancements to ROS, which should provide a better, secure form of communication. Also discussed in the final section was a list of advanced features that should be implemented as improvements to the system.

Chapter 7 Conclusion

This final chapter restates the aim and objectives of the thesis, and draws a summary of the research study. The research contributions that were defined in the introduction are expounded to include the conclusions established in previous chapters. The last section identifies some recommendations for further research related to this study.

7.1 Aim and objectives

The aim of this study was to research, design, and develop a distributed framework for managing multiple, heterogeneous mobile robots in smart factories.

The objectives of this research study were to:

- Investigate the impact of mobile robot systems in industry.
- Investigate the problem of mass customisation together with the introduction of the fourth industrial revolution (Industry 4.0).
- Research the need for robotic middleware and the various platforms available for mobile robot systems.
- Research the design requirements of a distributed framework for use in an industrial MMRS setting.
- Design and develop the framework that facilitates cooperation in heterogeneous multiple mobile robot teams.
- Test and validate the performance of the framework by the use of algorithms and simulation tools.

7.2 Research summary

The first chapter in this thesis introduced the concepts of mass customisation and Industry 4.0, and thereafter discussed the growth of robotics in industry. The motivation and necessity behind the development of the framework was also addressed in this chapter. The existence of the need of this research study was discussed as being due to the modern popularity of MMRS and cooperation to facilitate flexibility in smart factories, thereby managing the problem of mass customisation.

The literature survey investigated the role of mobile robot systems in industry and the associated topics such as heterogeneity, robot communication, cooperation, and middleware platforms were also discussed. The survey concluded by discussing the characteristics and requirements of a smart factory (or Industry 4.0) environment as well as the influence of advanced robotics in these environments.

Chapter three presented the methodological approach taken during the design of the framework. The design requirements were defined, outlining the key components of the framework that enabled its functionality in a smart factory environment. The reasons for choosing the ROS middleware as the core platform for each robot were discussed, and a description of the communication mechanism employed by the ROS architecture was described. The basis for selecting Python as the programming language for creating the GUI and developing end-user algorithms was also discussed. A key examination on industrial data communication standards revealed the adoption of the OPC-UA protocol in the framework. This was chosen due to its specialisation in device and system interoperability, which are essential requirements in an Industry 4.0 system.

The development of the main components of the framework was discussed in chapter four, which included:

1. The Python Tkinter GUI screens where robot, task, and OPC tag configurations are done.

2. The mobile robot system, which detailed the ROS software implementation, decentralised-local robot control, the Stage simulator integration, and the OPC-UA client configuration,
3. The SCADA development.
4. The development of Python programming features to provide support for end-user applications that satisfy the contributions of this research study.

The fifth chapter presented four case studies, where each verified various aspects of the framework. The first study involved the control of a single mobile robot in a material handling scenario between two tasks. The application of the framework was tested through the development of user algorithms, the mapping of local variables and OPC tags, the communication with a ROS simulated robot, and the integration with a SCADA package. The second study was built from the first and included an application which demonstrated the cooperation of heterogeneous mobile robots. The Python programming features for heterogeneous robots were implemented in this study. The simulation results verified the cooperation of mobile robots in reducing bottleneck scenarios, and working towards common goals. The third study explored the ability of the framework to support the implementation of machine learning algorithms in mobile robot applications. This was achieved through the use of Python's scikit-learn machine learning library, as well as an implementation of the developmental tool for SVM learning. The simulation results proved the performance of the SVM models, which were created during the SVM training phase of the application. The final study verified the flexibility of the framework in its integration with a commercial third party system (Beckhoff, TwinCAT), whose control system uses EtherCAT as its data communication standard. This verification showed the capability of the framework to be included in an Industry 4.0 system. The reliability of the data communication exchange was also tested in this study, and the results showed that the communication time delays are well within the specification for high-level robot decision making algorithms. The application section references for each feature of the framework is summarised in Table 9.

Table 9. Reference summary of the framework features.

	Application reference
Single mobile robot control	Section 5.1
Framework distribution	Section 5.1.2
Multiple mobile robot control	Section 5.2
Heterogeneous robot support	Section 5.2
Robot cooperation	Section 5.2
Machine learning robot control	Section 5.3
Framework interoperability	Section 5.4

The previous chapter described the benefits of the framework, which included the use of popular software platforms of Python and ROS, and features that enable the control and cooperation of heterogeneous mobile robots in smart factory environments. A further scope of the framework was also discussed, pertaining to areas such as business management systems, multi-robot resource sharing, and applications that involve fixed robotic manipulators. Research related to the functionality of the framework was also discussed in this chapter. Some of the related aspects that were discussed included systems that support remote robot programming in single and multi-robot systems, networked robots, control of heterogeneous robots, and robot cooperation. The limitations of the framework was addressed, one of them being the centralised OPC Server which creates a single point of failure in the system. The solution to this drawback was discussed as an implementation of a redundant OPC Server that would use separate hardware to eliminate the single point of failure and increase the robustness of the system. The second limitation that was addressed is the issue of security, particularly in smart factory environments where data integrity and data protection are key requirements. The insecure communication protocol in the ROS component of the framework was identified, and it was also mentioned that the developers of ROS are currently implementing SROS, a set of security enhancements to ROS, which should provide a better, secure form of communication. Also discussed in the sixth chapter was a list of advanced features that should be implemented as improvements to the system.

7.3 Research contributions

The design and development of the framework in this research study was an original effort and has made the following five contributions:

1. The framework can be utilised by *industrial engineers* to remotely control robots since the framework adheres to the international, industrial OPC-UA communication protocol standard. The integration of the framework with the industrial Beckhoff TwinCAT control system in the fourth case study (section 5.4) verifies this contribution.
2. The framework can be utilised by *students* to develop solutions in the field of multi-robot cooperation. This was achieved through the use of the simple graphical user interface and the development of algorithms in Python. Application examples of this utilisation were discussed in the first two case studies in chapter 5.
3. The *advanced robotics* pillar in Industry 4.0 has been strengthened due to the OPC interface provided by the framework to achieve intelligent robotic systems. Advanced algorithms can be developed and simulated before being deployed in an actual system, thereby fast prototyping the system integration process.
4. The capability of the popular *ROS middleware* has been enhanced, since ROS has lacked the functionality for cooperative multi-robot systems. The functionality of this contribution has been verified in the second case study (section 5.2).
5. The *machine learning* field of research can broaden its application to mobile robot networks, via the use of Python's powerful machine learning library. An application example utilising the SVM learning algorithm from the scikit-learn library was demonstrated in the third case study (section 5.3), where the results proved the performance of the SVM models that were created during the SVM training phase of the application.

7.4 Recommendations for further research

The first recommendation involves the topic of decentralised robot control. An algorithm selection tool can be included in the framework GUI that will allow the end-user to remotely program the decentralised intelligence of robots in the industrial network. Thus, each robot would be able to scan data within the factory and make smart, real-time decisions based on the local, decentralised algorithms. A paramount factor that must be considered for this type of control to be a success is the issue of safety to humans and the factory environment, since a ‘loosely’ controlled robot can lead to catastrophic circumstances.

The map of the environment in this research study was static and predefined. Further research will see the application of Simultaneous Localisation and Mapping (SLAM) techniques, where maps of the environment are dynamically built as robots navigate. In addition to this, map data could be shared among robots in the network due to the distributive characteristic of the system.

SLAM algorithms can also be applied to the research area of Search and Rescue (SAR) robotics, and the framework could be adapted to suit this application. The redesign of the data communication part of the framework must be considered, since OPC-UA would not be applicable to non-industrial type environments. In a SAR application, the implementation of ROS together with OpenCV has the potential to yield interesting results. OpenCV is a free, open source, computer vision library that can be used for object, facial recognition, and classification of human actions, all of which are key elements in SAR scenarios.

References

- [1] C Dahlman, *Technology, globalization and international competitiveness: Challenges for developing Countries.*: Department of economic and social affairs of United Nations, 2007.
- [2] M Joerss, F Neuhaus, J Schröder, C Klink, and F Mann, "Parcel delivery: The future of last mile," McKinsey & Company, Travel, Transport and Logistics 2016.
- [3] G Qiao, R Lu, and C McLean, "Flexible Manufacturing System for Mass Customization Manufacturing," *International Journal of Mass Customisation*, vol. 1, pp. 374-393, 2006.
- [4] J Pine, *Mass Customization-the New Frontier in Business Competition.*: McGraw-Hill, 1993.
- [5] J Heikkila, "From Supply to Demand Chain Management: Efficiency and Customer Satisfaction," *Journal of Operations Management*, vol. 20, pp. 747-767, 2002.
- [6] Y Koren and M Shpitalni, "Design of Reconfigurable Manufacturing Systems," *Journal of Manufacturing Systems*, vol. 29, pp. 130-141, 2010.
- [7] R Davies, "Industry 4.0: Digitalisation for productivity and growth," European Parliamentary Research Service, 2015.
- [8] IFR, Macquarie Research, "Robots: Global Growth," Macquarie Research, 2016.
- [9] Global Industry Analysis, "Industrial Robotics Market - Global Industry Analysis, Size, Share, Growth, Trends and Forecast, 2014 - 2020," 2014.

-
- [10] F Tobe. (2017, July) The Robot Report. [Online]. <https://www.therobotreport.com/news/whats-happening-in-robotics-five-trends-to-watch>
- [11] C Stillstrom and M Jackson, "The concept of mobile manufacturing," *Journal of Manufacturing Systems*, vol. 26, pp. 188-193, 2007.
- [12] N Naidoo, G Bright, and R Stopforth, "Cooperative Autonomous Robot Agents in Flexible Manufacturing Systems," in *Proceedings of the 8th International Conference on Intelligent Systems and Agents*, Lisbon, 2014, pp. 190-194.
- [13] C Andrews. (2017, July) Engineering and Technology. [Online]. <https://eandt.theiet.org/content/articles/2017/07/industry-4-challenges-and-opportunities/>
- [14] Grenzebach. (2016, November) Industry 4.0 – Challenges in Automation Technology. [Online]. <https://www.grenzebach.com/press/industry-40-challenges-in-automation-technology/>
- [15] Frost & Sullivan, "The future of mobile robots," Frost & Sullivan, A Study 2015.
- [16] Wikipedia. (2017, July) Robots using ROS. [Online]. <http://wiki.ros.org/Robots>
- [17] ROS-industrial. (2017, July) ROS-industrial. [Online]. <http://rosindustrial.org/ricamericas/current-members/>
- [18] J Bechtold, A Kern, C Lauenstein, and L Bernhofer, "Industry 4.0 - The Capgemini Consulting View," Capgemini Consulting, 2014.
- [19] R R Murphy, "Alliance: Marsupial robots for urban search and rescue," *IEEE Intelligent Systems*, vol. 20, pp. 14-19, 2000.

-
- [20] A Stroupe, A Okon, and M Robinson, "Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance," *Autonomous Robots*, vol. 20, pp. 113-123, 2006.
- [21] L E Parker, "ALLIANCE: An architecture for fault-tolerant multi-robot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 220-240, 1998.
- [22] P R Wurman, R D'Andrea, and M Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," in *Proceedings of AI Magazine*, 2008, pp. 9-20.
- [23] JBT. (2017, July) JBT AGVs. [Online]. <http://www.jbtc-agv.com/en/Solutions/Products/Unit-Load-Automatic-Guided-Vehicles-AGVs>
- [24] L. Sabattini et al., "Technological roadmap to boost the introduction of AGVs in industrial applications," in *IEEE International Conference on Intelligent Computer Communication and Processing*, 2013.
- [25] Robotnik. (2017, July) Robotnik. [Online]. <http://www.robotnik.eu/manipulators/jr2/>
- [26] Z Yan, N Jouandeau, and A A Cherif, "A Survey and Analysis of Multi-Robot Coordination," *International Journal of Advanced Robotic Systems*, vol. 10, pp. 1-18, 2013.
- [27] H Asama, A Matsumoto, and Y Ishida, "Design of an autonomous and distributed robot system: ACTRESS," in *Proceedings of IROS'89*, Tsukuba, 1989, pp. 283-290.
- [28] T Fukuda, T Ueyama, and Y Kawauchi, "Concept of cellular robotic system (CEBOT) and basic strategies for its realisation," *Computers and Electrical Engineering*, vol. 18, no. 1, pp. 11-39, 1992.
- [29] S C Botelho and R Alami, "M+: A scheme for multi-robot cooperation through

- negotiated task allocation and achievement," in *Proceedings of ICRA'99*, Detroit, 1999, pp. 1234-1239.
- [30] B P Gerkey and M J Matarić, "Murdoch: Publish/subscribe task allocation for heterogeneous agents," in *Proceedings of Agents'00*, Barcelona, 2000, pp. 203-204.
- [31] Amazon. (2017, July) Amazon Robotics. [Online]. <https://www.amazonrobotics.com>
- [32] Y Cao, A Fukunaga, and A Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7-27, 1997.
- [33] N Naidoo, G Bright, and R Stopforth, "Material Flow Optimisation in Flexible Manufacturing Systems," in *Proceedings of the 6th IEEE International Conference on Robotics and Mechatronics (RobMech)*, Durban, 2013, pp. 1-5.
- [34] L E Parker, "Multiple Mobile Robot Systems," in *Handbook of Robotics*. Berlin Heidelberg: Springer-Verlag, 2008, ch. 40.
- [35] S J Russell and P Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed.: Prentice Hall, 2002.
- [36] RoboCup. (2017, July) RoboCup 2017. [Online]. <https://www.robocup2017.org/eng/index.html>
- [37] ICGA. (2017, July) ICGA Tournaments. [Online]. <https://www.game-ai-forum.org/icga-tournaments/>
- [38] E Sahin and W Spears, "Swarm robotics, a state of the art survey," Lecture notes in Computer Science 2005.
- [39] P Inigo-Blasco, F Diaz-del-Rio, C Romero-Ternero, and D Cagigas-Muniz, "Robotics software frameworks for multi-agent robotic systems development," *Elsevier Robotics*

-
- and Autonomous Systems*, vol. 60, pp. 803-821, 2012.
- [40] D Drenjanac and S D Tomic, "Middleware Challenges in Robotic Fleets for Precision Agriculture," *Journal of Mechanics Engineering and Automation*, vol. 3, pp. 702-714, 2013.
- [41] R Simmons, S Singh, D Hershberger, J Ramos, and T Smith, "First results in the coordination of heterogeneous robots for large-scale assembly," in *Proceedings of the ISER Seventh International Symposium on Experimental Robotics*, Honolulu, 2000.
- [42] A Elkady and T Sobh, "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography," *Journal of Robotics*, vol. 2012, pp. 1-15, 2012.
- [43] D Kruger, I Van Lil, N Sunderhauf, R Baumgartl, and P Protzel, "Using and extending the Miro middleware for autonomous mobile robots," in *Proceedings of the international conference on Towards Autonomous Robotic Systems (TAROS 06)*, Survey, 2006.
- [44] N Naidoo, G Bright, and R Stopforth, "Navigation and Control of Cooperative Mobile Robots using a Robotic Middleware Platform," in *Proceedings of the 12th IEEE International Conference on Control & Automation (ICCA)*, Kathmandu, 2016, pp. 927-932.
- [45] N Naidoo, G Bright, and R Stopforth, "A Distributed Framework for Programming the Artificial Intelligence of Mobile Robots in Smart Manufacturing Systems," *International Journal of Advanced Robotic Systems*, 2017.
- [46] M Filippi, A Saffiotti, and F Cavallo, "Report on the interoperability aspects of Robot-Era services," Large-scale integrating project (IP) , 2012.
- [47] N Naidoo, G Bright, and R Stopforth, "A cooperative mobile robot network in ROS for

- advanced manufacturing applications," in *Proceedings of the 6th International Conference on Competitive Manufacturing (COMA)*, Stellenbosch, 2016, pp. 281-286.
- [48] M Henning, "A New Approach to Object-Oriented Middleware," *IEEE Internet Computing*, vol. 8, pp. 66-75, 2004.
- [49] C Agüero and M Veloso, "Transparent Multi-Robot Communication Exchange for Executing Robot Behaviors," in *Proceedings of the 10th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, 2012.
- [50] M Broxvall, B S Seo, and W Y Kwon, "The PEIS Kernel: A Middleware for Ubiquitous Robotics," in *Proceedings of IROS-07, Workshop on Ubiquitous Robotic Space Design and Applications*, 2007.
- [51] M Broxvall, A Loutfi, S Coradeschi, and A Saffiotti, "An ecological approach to odour recognition in intelligent environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, 2006.
- [52] M Broxvall, M Gritti, A Saffiotti, B Seo, and Y Cho, "PEIS ecology: Integrating robots into smart environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, 2006.
- [53] S Ahn et al., "UPnP Robot Middleware for Ubiquitous Robot Control," in *Proceedings of the 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI2006)*, 2006.
- [54] S Enderle et al., "Miro: middleware for autonomous mobile robots," in *Proceedings of IFAC Conference on Telematics Applications in Automation and Robotics*, 2001.
- [55] S Chouhan, D Pandey, and Y Chul Ho, "CINeMA: Cooperative Intelligent Network Management Architecture for Multi-Robot Rescue System in Disaster Areas," in

-
- Proceedings of the International Conference on Electrical, Electronics, Computer Science, and Mathematics Physical Education and Management (ICEECMPE)*, New Delhi, 2014, pp. 51-61.
- [56] D Althoff, O Kourakos, and M Lawitzky, "An Architecture for Real-Time Control in Multi-Robot Systems," *Human Centered Robot Systems*, vol. 6, pp. 43-52, 2009.
- [57] M Goebel and G Farber, "A real-time-capable hardware and software architecture for joint image and knowledge processing in cognitive automobiles," in *Proceedings of the 2007 IEEE Intelligent Vehicles Symposium*, 2007, pp. 734-740.
- [58] M Goebel, "A real-time architecture for the integration of cognitive functions," Technische Universitat Munchen, PhD thesis 2009.
- [59] S Planthaber, J Vogelgesang, and E NieBen, "CoHoN: A Fault-Tolerant Publish/Subscribe Tree-Based Middleware for Robots with Heterogeneous Communication Hardware," in *Proceedings of the 6th International Conference on Systems and Networks Communications (ICSNC)*, 2011.
- [60] C Cote, Y Brosseau, D Letourneau, C Raievsky, and F Michaud, "Robotic Software Integration Using MARIE," *The International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 55-60, 2006.
- [61] B Gerkey, R Vaughan, and A Howard, "The Player/Stage Project: tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, 2003, pp. 317-323.
- [62] I A Nesnas, R Simmons, and D Gaines, "Claraty: challenges and steps toward reusable robotic software," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 23-30, 2006.

-
- [63] I A Nesnas et al., "Claraty: an architecture for reusable robotic software," *Space Robots*, vol. 5083, pp. 253-264, 2003.
- [64] S Magnenat, V Longchamp, and F Mondada, "ASEBA, an event-based middleware for distributed robot control," in *IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems*, San Diego, 2007.
- [65] M Montemerlo, N Roy, and S Thrun, "Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, 2003, pp. 2436-2441.
- [66] D S Blank, D Kumar, L Meeden, and H A Yanco, "The pyro toolkit for AI and robotics," *AI Magazine*, vol. 27, no. 1, pp. 39-50, 2006.
- [67] D S Blank, D Kumar, L Meeden, and H A Yanco, "Pyro: an integrated environment for Robotics education," in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI '05)*, 2005, pp. 1718-1719.
- [68] A Makarenko and A Brooks, "Orca: components for robotics," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems Workshop on Robotic Standardization (IROS '06)*, Beijing, 2006.
- [69] C Szyperski, *Component software : beyond object-oriented programming*, 2nd ed. Boston: Addison-Wesley Longman Publishing Co., 2002.
- [70] Wikipedia. (2017, July) DARPA Grand Challenge. [Online]. https://en.wikipedia.org/wiki/DARPA_Grand_Challenge
- [71] OROCOS. (2017, July) The OROCOS Project. [Online]. <http://www.orocos.org/>

-
- [72] OROCOS. (2017, July) The OROCOS Project. [Online]. <https://people.mech.kuleuven.be/~orocos/pub/documentation/rtt/v2.4.x/doc-xml/orocos-overview.pdf>
- [73] K Johns and T Taylor, *Microsoft Robotics Developer Studio*, 1st ed. Indianapolis: Wiley Publishing, Inc., 2008.
- [74] J S Cepeda, L Chaimowicz, and R Soto, "Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics," in *Latin American Robotics Symposium and Intelligent Robotics Meeting*, São Bernardo do Campo, 2010.
- [75] B Gerkey et al., "ROS: an open-source robot operating system ," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- [76] ROS. (2017, July) List of ROS packages for Indigo. [Online]. <http://www.ros.org/browse/list.php>
- [77] A Jäger and F Ranz, "Implications for Learning Factories from Industry 4.0.," Fraunhofer Austria Research and ESB Business School, 2014.
- [78] V Mahidhar and D Schatsky, "The Internet of Things," Deloitte University Press, 2013.
- [79] R C Schlaepfer, M Koch, and P Merkofer, "Industry 4.0: Challenges and solutions for the digital transformation and use of exponential technologies," Deloitte, A Study 2015.
- [80] W MacDougall, "Industrie 4.0: Smart manufacturing for the future," Germany Trade and Invest, Berlin, 2014.
- [81] V A Hax, N L Duarte Filho, S S Da Costa Botelho, and O M Mendizabal, "ROS as a middleware to Internet of Things," *Journal of Applied Computing Research*, vol. 2, no. 2, pp. 91-97, 2012.

-
- [82] N Naidoo, G Bright, R Stopforth, J F Zelasco, and J Donayo, "Optimizing Search and Rescue Missions through a Cooperative Mobile Robot Network," in *Proceedings of the 8th IEEE International Conference on Robotics and Mechatronics (RobMech)*, Port Elizabeth, 2015, pp. 7-11.
- [83] W Herroelen and R Leus, "On the merits and pitfalls of critical chain scheduling," *Journal of Operations Management*, vol. 19, pp. 559-577, 2001.
- [84] PROFIBUS and PROFINET International (PI). (2017, August) PROFIBUS. [Online]. <http://www.profibus.com/pi-organization/about-pi/>
- [85] CANbus. (2017, August) CANbus. [Online]. <https://www.canbus.us/>
- [86] The Modbus Organisation. (2017, August) Modbus. [Online]. <http://www.modbus.org/>
- [87] CC-Link Partner Association. (2017, August) CC-Link. [Online]. <http://am.cc-link.org/en/index.html>
- [88] Z Lin and S Pearson, "An inside look at industrial Ethernet communication protocols," Texas Instruments Incorporated, White paper 2013.
- [89] P Brooks, "EtherNet/IP: Industrial Protocol White Paper," Institute of Electrical and Electronic Engineers, White paper 2001.
- [90] EtherCAT Technology Group. (2017, August) EtherCAT. [Online]. <https://www.ethercat.org/default.htm>
- [91] Beckhoff, "EtherCAT System Documentation," Beckhoff, Manual Version 5.2, 2017.
- [92] Advantech, "Using EtherCAT for Industrial Control Communications," Advantech, Technical white paper 2015.

-
- [93] OPC Foundation. (2017, August) OPC Foundation. [Online]. <https://opcfoundation.org/>
- [94] T Burke, "President's Welcome," OPC Foundation, OPC Connect Newsletter 2017.
- [95] Matrikon. (2017, August) Matrikon OPC. [Online]. <http://www.matrikonopc.com/index.aspx>
- [96] Qt. (2017, August) Qt. [Online]. <https://www.qt.io/>
- [97] R-project. (2017, August) What is R? [Online]. <https://www.r-project.org/about.html>
- [98] Python. (2017, August) Python. [Online]. <https://www.python.org/>
- [99] scikit-learn. (2017, August) scikit-learn: Machine learning in Python. [Online]. <http://scikit-learn.org/stable/>
- [100] Ecava IGX. (2017, August) Ecava Integraxor Web SCADA. [Online]. <https://www.integraxor.com/>
- [101] ROS. (2017, August) AMCL ROS package. [Online]. <http://wiki.ros.org/amcl>
- [102] SSH communications security. (2017, August) SSH Protocol. [Online]. <https://www.ssh.com/ssh/protocol/>
- [103] M Hvilshøj, S Bøgh, O Madsen, and M Kristiansen, "The mobile robot "Little Helper": Concepts, ideas and working principles," in *IEEE Conference on Emerging Technologies & Factory Automation*, 2009.
- [104] M Faisal, M Alsulaiman, K Al-Muteb, and M Emaduddin, "Gas Detection and Mapping Using an Autonomous Mobile Robot," in *28th International Conference on Computer Applications in Industry and Engineering*, 2015.

-
- [105] S Opfer, H Skubch, and K Geihs, "Cooperative Path Planning for Multi-Robot Systems in Dynamic Domains," in *Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training*, J Będkowski, Ed.: Intech, 2011, ch. 11, pp. 237-258.
- [106] B Lydon. (2012, February) Automation.com. [Online].
<https://www.automation.com/automation-news/article/digital-factory-superstructure-emerging-with-opc-ua>
- [107] S Hill. (2015, January) Automation World. [Online].
<https://www.automationworld.com/article/technologies/erp/how-plant-floor-data-drives-enterprise>
- [108] R Marin et al., "Remote Programming of Network Robots Within the UJI Industrial Robotics Telelaboratory: FPGA Vision and SNRP Network Protocol," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 12, pp. 4806-4816, December 2009.
- [109] M Tucac and A J Garcia, "High Level Programming Tools for Robotic Interaction Protocols: a Logic Programming Approach," in *4th International Symposium of Autonomous Minirobots for Research and Edutainment (AMiRE 07)*, Buenos Aires, 2007.
- [110] J Bachrach, J McLurkin, and A Grue, "Protoswarm: A Language for Programming Multi-Robot Systems Using the Amorphous Medium Abstraction," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, 2008, pp. 1175-1178.
- [111] V M Trifa, C M Cianci, and D Guinard, "Dynamic Control of a Robotic Swarm using a Service-Oriented Architecture," in *13th International Symposium on Artificial Life and Robotics*, Beppu, 2008.

-
- [112] OPC Foundation. (2017, September) Security. [Online]. <http://wiki.opcfoundation.org//index.php?title=Security&printable=yes>
- [113] ROS. (2017, September) SROS. [Online]. <http://wiki.ros.org/SROS>

Appendix A: Python Tkinter GUI code

```
1> # Filename: main.py
2> from Tkinter import *
3> import ttk
4> import tkFileDialog
5> import inspect
6> import OpenOPC # the OPC client library for Python
7> import paramiko # for file transfers
8> import gvar
9> import os
10> import time
11> import platform
12> import subprocess
13> import LibHetRobots
14>
15>
16> # Global variables:
17> treestore = ""
18> connFlag = False
19>
20>
21> def checkIP(hostname):
22>     pingFlag = False
23>     try:
24>         if platform.system() == "Windows":
25>             response = subprocess.check_output("ping %s -n 1 -w 1000"
26> % hostname, shell=True) # limits the response to 1000ms
27>         else:
28>             response = subprocess.check_output("ping -c 1 -W 1000" +
29> hostname, shell=True)
30>         if response.find('TTL=') != -1: # check if we get a valid
31> TTL response from host
32>             pingFlag = True
33>     except:
34>         pingFlag = False
35>     return pingFlag
36>
```

```
36> def getRoblist():
37>     rlist = []
38>     f = open('robconfig.txt', 'r')
39>     for line in f:
40>         startpos = 0 # char start position in string
41>         rrec = []
42>         charpos = line.find(',')
43>         while charpos != -1:
44>             rrec.append(line[startpos:charpos])
45>             startpos = charpos + 1
46>             charpos = line.find(',', startpos, len(line))
47>         rlist.append(str(rrec[0] + ", " + rrec[1]))
48>     f.close()
49>     rlist.sort()
50>     return rlist
51>
52>
53> def getTasklist():
54>     tlist = []
55>     f = open('taskconfig.txt', 'r')
56>     for line in f:
57>         startpos = 0 # char start position in string
58>         trec = []
59>         charpos = line.find(',')
60>         while charpos != -1:
61>             trec.append(line[startpos:charpos])
62>             startpos = charpos + 1
63>             charpos = line.find(',', startpos, len(line))
64>         tlist.append(str(trec[0] + ", " + trec[1]))
65>     f.close()
66>     tlist.sort()
67>     return tlist
68>
69>
70> def find_idx(mainstr, substr, n):
71>     start = mainstr.find(substr)
72>     while start >= 0 and n > 1:
73>         start = mainstr.find(substr, start+len(substr))
74>         n -= 1
75>     return start
76>
77>
78> def getLoclist(t_id, n_id):
79>     loclist = []
```



```
80> f = open('robots/%slocations.txt' % ron_id, 'r')
81> for line in f:
82>     charpos = line.find('%s,%s' % (t_id, n_id))
83>     if charpos != -1:
84>         loclist.append(line[find_idx(line, ',', 2) +
1:find_idx(line, ',', 3)])
85>         loclist.append(line[find_idx(line, ',', 3) +
1:find_idx(line, ',', 4)])
86>         loclist.append(line[find_idx(line, ',', 4) +
1:find_idx(line, ',', 5)])
87>     return loclist
88> f.close()
89>
90>
91> def getRobname(r_id):
92>     f = open('robconfig.txt', 'r')
93>     for line in f:
94>         charpos = line.find(r_id)
95>         if charpos != -1:
96>             return line[find_idx(line, ',', 1) + 1:find_idx(line,
',', 2)]
97>     f.close()
98>
99>
100> class App(Frame):
101>     def __init__(self):
102>         # self.MyFont=font.Font(weight='bold')
103>         self.root = Tk()
104>         Frame.__init__(self)
105>         self.root.title("Robot Programming Interface")
106>         self.root.rowconfigure(5, weight=1)
107>         self.master.columnconfigure(5, weight=1)
108>         self.grid(sticky=W+E+N+S)
109>         self.conn_stat = StringVar()
110>         self.conn_stat.set("OFFLINE")
111>         self.conn_led = StringVar()
112>         self.conn_led.set("red")
113>
114>         connFrame = Frame(self.root, width=500, height=50,
borderwidth=2, relief=GROOVE)
115>         connFrame.grid(row=0, column=0, padx=0, pady=1,
columnspan=4, sticky=NSEW)
116>         self.w = Canvas(connFrame, width=19, height=19)
```

```
117>         self.w.grid(row=0, column=0, padx=0, pady=0,
118>         sticky=NSEW)
119>         self.conn_circ = self.w.create_oval(2,2,16,16,
120>         fill="red")
121>         self.w.itemconfig(self.conn_circ,
122>         fill=self.conn_led.get())
123>         self.connlab = ttk.Label(connFrame,
124>         textvariable=self.conn_stat).grid(row=0, column=1, padx=0, pady=0)
125>
126>         labFrame = Frame(self.root, width=200, height=200)
127>         labFrame.grid(row=1, column=0, padx=0, pady=10)
128>         self.IPlabel = ttk.Label(labFrame, text="IP
129>         address").grid(row=0, column=0, padx=5, pady=5)
130>         self.IPinput = StringVar()
131>         self.IPadd = ttk.Entry(labFrame, background='white',
132>         width=20, textvariable=self.IPinput)
133>         self.IPadd.grid(row=0, column=1, padx=0, pady=5)
134>
135>         butFrame = Frame(self.root, width=500, height=500)
136>         butFrame.grid(row=1, column=1, padx=0, pady=10,
137>         sticky=NSEW)
138>         butFrame2 = Frame(self.root, width=500, height=500)
139>         butFrame2.grid(row=2, column=0, padx=50, pady=10,
140>         columnspan=3, sticky=NSEW)
141>         butFrame2.rowconfigure(0, minsize=60)
142>         butFrame2.rowconfigure(1, minsize=60)
143>         butFrame2.rowconfigure(2, minsize=60)
144>
145>         statFrame = Frame(self.root, width=100, height=5,
146>         borderwidth=2, relief=GROOVE)
147>         statFrame.grid(row=3, column=0, padx=0, pady=0,
148>         columnspan=3, sticky=NSEW)
149>         self.status = StringVar()
150>         self.statlab = ttk.Label(statFrame,
151>         textvariable=self.status).grid(row=0, column=0, padx=5, pady=0)
152>
153>         self.test_butn = ttk.Button(butFrame, text="Ping",
154>         command=self.PingTest)
155>         self.test_butn.grid(row=0, column=1, padx=5, pady=0,
156>         sticky=NSEW)
157>         self.connect_butn = ttk.Button(butFrame,
158>         text="Connect", command=self.RobConnect)
159>         self.connect_butn.grid(row=2, column=1, padx=5, pady=0,
160>         sticky=NSEW)
```

```
146>         self.roblist_butn = ttk.Button(butFrame, text="Robot
List", command=self.Win_RobList)
147>         self.roblist_butn.grid(row=0, column=2, padx=5, pady=5,
rowspan=3, sticky=NSEW)
148>
149>         self.configrob_butn = ttk.Button(butFrame2, text="
Configure Robots ", command=self.Win_ConfigRob)
150>         self.configrob_butn.grid(row=0, column=0, padx=5,
pady=5, sticky=NSEW)
151>         self.taskman_butn = ttk.Button(butFrame2, text=" Task
Management ", command=self.Win_ConfigTask)
152>         self.taskman_butn.grid(row=0, column=1, padx=5, pady=5,
sticky=NSEW)
153>         self.asgntags_butn = ttk.Button(butFrame2, text="Assign
Tags", command=self.Win_AssignTags)
154>         self.asgntags_butn.grid(row=1, column=0, padx=5,
pady=5, sticky=NSEW)
155>         self.selalgo_butn = ttk.Button(butFrame2, text="Select
Algorithm", command=self.Win_SelectAlgo)
156>         self.selalgo_butn.grid(row=1, column=1, padx=5, pady=5,
sticky=NSEW)
157>         self.locations_butn = ttk.Button(butFrame2, text="Task
Locations", command=self.Win_LocationConfig)
158>         self.locations_butn.state(["disabled"])
159>         self.locations_butn.grid(row=2, column=0, padx=5,
pady=5, sticky=NSEW)
160>         self.download_butn = ttk.Button(butFrame2,
text="Download", command=self.RobDownload)
161>         self.download_butn.state(["disabled"])
162>         self.download_butn.grid(row=2, column=1, padx=5,
pady=5, sticky=NSEW)
163>         #self.download_butn.state(["!disabled"])
164>         self.root.mainloop()
165>
166>         def PingTest(self):
167>             pingResult = checkIP(self.IPinput.get())
168>             if pingResult == True:
169>                 self.status.set("Ping success!")
170>             else:
171>                 self.status.set("Destination host unreachable")
172>
173>         def RobConnect(self):
174>             self.status.set("Establishing connection...")
175>             try:
176>                 ssh_client = paramiko.SSHClient()
```

```
177>
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy()) #
    prevent security key auth
178>
        ssh_client.connect(hostname=self.IPinput.get(),
        username='nicol', password='nicol')
179>
        ftp = ssh_client.open_sftp()
180>
        ftp.get("/home/nicol/robconfigfiles/rid.txt",
        "robots/rid.txt") # transfer file
181>
        f = open('robots/rid.txt', 'r')
182>
        for line in f:
183>
            ron_id = line.rstrip()
184>
            f.close()
185>
            ftp.close()
186>
            ssh_client.close()
187>
            self.status.set("Connected to host")
188>
            self.conn_led.set("green")
189>
            self.w.itemconfig(self.conn_circ,
            fill=self.conn_led.get())
190>
            self.conn_stat.set("ONLINE [%s (%s)]" % (ron_id,
            getRobname(ron_id)))
191>
            self.locations_butn.state(["!disabled"])
192>
            self.download_butn.state(["!disabled"])
193>
        except:
194>
            self.status.set("Could not connect to host")
195>
196>
        def RobDownload(self):
197>
            try:
198>
                self.status.set("Downloading file...")
199>
                ssh_client = paramiko.SSHClient()
200>
                ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy()) #
                prevent security key auth
201>
                    ssh_client.connect(hostname=self.IPinput.get(),
                    username='nicol', password='nicol')
202>
                    ftp = ssh_client.open_sftp()
203>
                    ftp.put("robots/001locations.txt",
                    "/home/nicol/robconfigfiles/001locations.txt") # transfer file
204>
                    ftp.close()
205>
                    ssh_client.close()
206>
                    self.status.set("Download complete")
207>
            except:
208>
                self.status.set("Could not connect to host")
209>
210>
        def Win_ConfigRob(self):
```

```
211>         ClassRobconf()
212>
213>         def Win_RobList(self):
214>             ClassRoblist()
215>
216>         def Win_ConfigTask(self):
217>             ClassTaskman()
218>
219>         def Win_AssignTags(self):
220>             ClassAssigntags()
221>
222>         def Win_SelectAlgo(self):
223>             ClassSelectAlgo()
224>
225>         def Win_LocationConfig(self):
226>             ClassLocationConfig()
227>
228>     class ClassRobconf(Frame):
229>         def __init__(self):
230>             Frame.__init__(self)
231>             self.Robconf = Toplevel()
232>             self.Robconf.title("Robot Configuration")
233>             global idVar, nameVar, ipVar, r_lenVar, r_wtVar,
234>                 r_htVar, r_ldVar, r_linspdVar, r_angspdVar
235>                 global r_lrfVar, r_sonVar, r_camVar, r_gripVar,
236>                 r_2whlVar
237>                 global select, select2, roblist, robsublist, robrec
238>
239>             frame1 = Frame(self.Robconf, width=100, height=100)
240>             frame1.grid(row=0, column=0, columnspan=3)
241>             Label(frame1, text="Robot ID").grid(row=0, column=0,
242>                 sticky=W)
243>             idVar = StringVar()
244>             ttk.Entry(frame1, width=10,
245>                 textvariable=idVar).grid(row=0, column=1, padx=(5,30), pady=20,
246>                 sticky=W)
247>             Label(frame1, text="Name").grid(row=0, column=2,
248>                 sticky=W)
249>             nameVar = StringVar()
250>             ttk.Entry(frame1, textvariable=nameVar).grid(row=0,
251>                 column=3, padx=(5,30), pady=20, sticky=W)
252>             Label(frame1, text="IP address").grid(row=0, column=4,
253>                 sticky=W)
254>             ipVar = StringVar()
```

```
247>         ttk.Entry(frame1, textvariable=ipVar).grid(row=0,
           column=5, padx=(5,30), pady=20, sticky=W)
248>
249>         frame2 = Frame(self.Robconf, width=5, height=10,
           borderwidth=2, relief=GROOVE)
250>         frame2.grid(row=1, column=0, padx=10, pady=0,
           sticky=NSEW)
251>         Label(frame2, text="Robot dimensions (mm)").grid(row=0,
           column=0, columnspan=3)
252>         ent_width=5
253>         Label(frame2, text="L").grid(row=1, column=0, padx=15,
           pady=0, sticky=W)
254>         r_lenVar = StringVar()
255>         ttk.Entry(frame2, width=ent_width,
           textvariable=r_lenVar).grid(row=2, column=0, padx=5, pady=0,
           sticky=W)
256>         Label(frame2, text="W").grid(row=1, column=1, padx=15,
           pady=0, sticky=W)
257>         r_wtVar = StringVar()
258>         ttk.Entry(frame2, width=ent_width,
           textvariable=r_wtVar).grid(row=2, column=1, padx=5, pady=0, sticky=W)
259>         Label(frame2, text="H").grid(row=1, column=2, padx=15,
           pady=0, sticky=W)
260>         r_htVar = StringVar()
261>         ttk.Entry(frame2, width=ent_width,
           textvariable=r_htVar).grid(row=2, column=2, padx=5, pady=0, sticky=W)
262>
263>         frame3 = Frame(self.Robconf, width=10, height=10,
           borderwidth=2, relief=GROOVE)
264>         frame3.grid(row=1, column=1, padx=10, pady=0,
           sticky=NSEW)
265>         Label(frame3, text="Max. load capacity").grid(row=0,
           column=0, padx=0, pady=0, sticky=W)
266>         r_ldVar = StringVar()
267>         ttk.Entry(frame3, width=ent_width,
           textvariable=r_ldVar).grid(row=0, column=1, padx=0, pady=0,
           sticky=NSEW)
268>         Label(frame3, text="kg").grid(row=0, column=2, padx=0,
           pady=0, sticky=W)
269>         Label(frame3, text="Max. linear speed").grid(row=1,
           column=0, padx=0, pady=2, sticky=W)
270>         r_linspdVar = StringVar()
271>         ttk.Entry(frame3, width=ent_width,
           textvariable=r_linspdVar).grid(row=1, column=1, padx=0, pady=2,
           sticky=NSEW)
```

```
272>         Label(frame3, text="m/s").grid(row=1, column=2, padx=0,
      pady=0, sticky=W)
273>         Label(frame3, text="Max. angular speed").grid(row=2,
      column=0, padx=0, pady=2, sticky=W)
274>         r_angspdVar = StringVar()
275>         ttk.Entry(frame3, width=ent_width,
      textvariable=r_angspdVar).grid(row=2, column=1, padx=0, pady=2,
      sticky=NSEW)
276>         Label(frame3, text="rad/s").grid(row=2, column=2,
      padx=0, pady=0, sticky=W)
277>
278>         frame4 = Frame(self.Robconf, width=5, height=10,
      borderwidth=2, relief=GROOVE)
279>         frame4.grid(row=1, column=2, padx=10, pady=0,
      sticky=NSEW)
280>         Label(frame4, text="Hardware installed").grid(row=0,
      column=0, columnspan=8)
281>         r_lrfVar = IntVar()
282>         ttk.Checkbutton(frame4, width=0,
      variable=r_lrfVar).grid(row=1, column=0, padx=0, pady=0, sticky=E)
283>         Label(frame4, text="LRF").grid(row=1, column=1,
      padx=(0,30), pady=0, sticky=W)
284>         r_sonVar = IntVar()
285>         ttk.Checkbutton(frame4, width=0,
      variable=r_sonVar).grid(row=1, column=2, padx=0, pady=0, sticky=E)
286>         Label(frame4, text="Sonars").grid(row=1, column=3,
      padx=(0,10), pady=0, sticky=W)
287>         r_camVar = IntVar()
288>         ttk.Checkbutton(frame4, width=0,
      variable=r_camVar).grid(row=2, column=0, padx=0, pady=0, sticky=E)
289>         Label(frame4, text="Camera").grid(row=2, column=1,
      padx=(0, 30), pady=0, sticky=W)
290>         r_gripVar = IntVar()
291>         ttk.Checkbutton(frame4, width=0,
      variable=r_gripVar).grid(row=2, column=2, padx=0, pady=0, sticky=E)
292>         Label(frame4, text="Grippers").grid(row=2, column=3,
      padx=(0, 10), pady=0, sticky=W)
293>         r_2whlVar = IntVar()
294>         ttk.Checkbutton(frame4, width=0,
      variable=r_2whlVar).grid(row=3, column=0, padx=0, pady=0, sticky=E)
295>         Label(frame4, text="Two-wheel").grid(row=3, column=1,
      padx=(0, 30), pady=0, sticky=W)
296>
297>         frame5 = Frame(self.Robconf)
```

```
298>         frame5.grid(row=2, column=0, padx=10, pady=(15,0),
    columnspan=4, sticky=NSEW)
299>         ttk.Button(frame5, text=" Add ",
    command=self.addEntry).grid(row=0, column=0, padx=(0, 0), pady=0,
    sticky=W)
300>         ttk.Button(frame5, text="Update",
    command=self.updateEntry).grid(row=0, column=1, padx=(0, 0), pady=0,
    sticky=W)
301>         ttk.Button(frame5, text="Delete",
    command=self.deleteEntry).grid(row=0, column=2, padx=(0, 0), pady=0,
    sticky=W)
302>         ttk.Button(frame5, text=" Load ",
    command=self.loadEntry).grid(row=0, column=3, padx=(0, 0), pady=0,
    sticky=W)
303>
304>         frame6 = Frame(self.Robconf)
305>         frame6.grid(row=3, column=2, padx=7, pady=(0, 0),
    columnspan=1, sticky=NSEW)
306>         Label(frame6, text="Compatible Tasks").grid(row=0,
    column=0, padx=0, pady=0, sticky=NSEW)
307>
308>         frame7 = Frame(self.Robconf)
309>         frame7.grid(row=4, column=0, padx=10, pady=(0,5),
    columnspan=5, sticky=NSEW)
310>         scroll = ttk.Scrollbar(frame7, orient=VERTICAL)
311>         select = Listbox(frame7, yscrollcommand=scroll.set,
    height=6, width=50)
312>         scroll.config(command=select.yview)
313>         scroll.grid(row=0, column=5, padx=(0, 0), pady=0,
    sticky=NS)
314>         select.grid(row=0, column=0, padx=(0, 0), pady=0,
    columnspan=4, sticky=NSEW)
315>
316>         frame8 = Frame(self.Robconf)
317>         frame8.grid(row=4, column=2, padx=10, pady=(0,5),
    columnspan=5, sticky=NSEW)
318>         scroll12 = ttk.Scrollbar(frame8, orient=VERTICAL)
319>         select2 = Listbox(frame8, yscrollcommand=scroll12.set,
    height=6, width=34)
320>         scroll12.config(command=select2.yview)
321>         scroll12.grid(row=0, column=7, padx=(0, 0), pady=0,
    sticky=NS)
322>         select2.grid(row=0, column=0, padx=(0, 0), pady=0,
    columnspan=4, sticky=NSEW)
323>
```

```

324>         self.setSelect() # configure roblast and select
        listBox
325>         self.Robconf.focus_force()
326>
327>     def setSelect(self):
328>         totrec = 0 # total number of records
329>         global roblast, robsublist, robrec
330>         roblast=[]
331>         f = open('robconfig.txt', 'r')
332>         for line in f:
333>             startpos = 0 # char start position in string
334>             robrec=[]
335>             charpos = line.find(',')
336>             while charpos != -1:
337>                 robrec.append(line[startpos:charpos])
338>                 startpos = charpos + 1
339>                 charpos = line.find(',', startpos, len(line))
340>                 roblast.append(str(robrec[0] + ", " + robrec[1] +
        ", " + robrec[2]))
341>                 totrec = totrec + 1
342>         f.close()
343>         roblast.sort()
344>         select.delete(0, END)
345>         for i in range(totrec):
346>             select.insert(END, roblast[i])
347>
348>         # Compatible tasks list:
349>         def setSelect2(self, rid):
350>             totrec = 0 # total number of records
351>             CompatTaskMat = LibHetRobots.getCompatTasks(rid)
352>             select2.delete(0, END)
353>             for i in range(len(CompatTaskMat)):
354>                 select2.insert(END, str(CompatTaskMat[i][0] + ', '
        + CompatTaskMat[i][1]))
355>
356>         def whichSelected(self):
357>             return int(select.curselection()[0])
358>
359>         def addEntry(self):
360>             f = open('robconfig.txt', 'a')
361>             f.write('%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,E\n'
        % (idVar.get(), nameVar.get(), ipVar.get(),
362>         r_lenVar.get(), r_wtVar.get(), r_htVar.get(),

```

```
363>     r_ldVar.get(), r_linspdVar.get(), r_angspdVar.get(),
364>     r_lrfVar.get(), r_sonVar.get(), r_camVar.get(),
365>     r_gripVar.get(), r_2whlVar.get()))
366>         f.close()
367>         self.setSelect()
368>
369>     def updateEntry(self):
370>         str1 = select.get(ACTIVE) # get the text of the
           currently selected item in the listbox
371>         str2 = str1[0:str1.find(',')]
372>         lines = open('robconfig.txt', 'r').readlines()
373>         for i in range(len(lines)):
374>             if lines[i].find(str2) != -1:
375>                 lines[i] =
           '%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,E\n' % (idVar.get(),
           nameVar.get(), ipVar.get(),
376>         r_lenVar.get(), r_wtVar.get(), r_htVar.get(),
377>         r_ldVar.get(), r_linspdVar.get(), r_angspdVar.get(),
378>         r_lrfVar.get(), r_sonVar.get(), r_camVar.get(),
379>         r_gripVar.get(), r_2whlVar.get())
380>             break
381>         f = open('robconfig.txt', 'w')
382>         f.writelines(lines)
383>         f.close()
384>         self.setSelect()
385>
386>     def deleteEntry(self):
387>         str1 = select.get(ACTIVE) # get the text of the
           currently selected item in the listbox
388>         str2 = str1[0:str1.find(',')]
389>         lines = open('robconfig.txt', 'r').readlines()
390>
391>         for i in range(len(lines)):
392>             if lines[i].find(str2) != -1:
393>                 lines.remove(lines[i])
394>                 break
395>         f = open('robconfig.txt', 'w')
```

```
396>         f.writelines(lines)
397>         f.close()
398>         self.setSelect()
399>
400>         def loadEntry(self):
401>             global robrec
402>             str1 = select.get(ACTIVE) # get the text of the
           currently selected item in the listbox
403>             str2 = str1[0:str1.find(',')]
404>             f = open('robconfig.txt', 'r')
405>             for line in f:
406>                 if line.find(str2) != -1:
407>                     startpos = 0 # char start position in string
408>                     robrec = []
409>                     charpos = line.find(',')
410>                     while charpos != -1:
411>                         robrec.append(line[startpos:charpos])
412>                         startpos = charpos + 1
413>                         charpos = line.find(',', startpos,
           len(line))
414>                     break # breaks the for-loop
415>             f.close()
416>             # set the fields on the form:
417>             idVar.set(robrec[0])
418>             nameVar.set(robrec[1])
419>             ipVar.set(robrec[2])
420>             r_lenVar.set(robrec[3])
421>             r_wtVar.set(robrec[4])
422>             r_htVar.set(robrec[5])
423>             r_ldVar.set(robrec[6])
424>             r_linspdVar.set(robrec[7])
425>             r_angspdVar.set(robrec[8])
426>             r_lrfVar.set(robrec[9])
427>             r_sonVar.set(robrec[10])
428>             r_camVar.set(robrec[11])
429>             r_gripVar.set(robrec[12])
430>             r_2whlVar.set(robrec[13])
431>             self.setSelect2(int(robrec[0]))
432>
433>         class ClassRoblist(Frame):
434>             def __init__(self):
435>                 Frame.__init__(self)
436>                 self.Roblistwin = Toplevel()
437>                 self.Roblistwin.title("Robot List")
```

```
438>         self.tree = ttk.Treeview(self.Roblistwin, height=20) #
height param modifies the window height - adjust if lots of entries!
439>         self.tree["columns"] = ("rob_id", "rob_name", "rob_ip")
440>         self.tree.column("#0", width=40, anchor=N)
441>         self.tree.heading("#0", text="No.")
442>         self.tree.column("rob_id", width=80, anchor=N)
443>         self.tree.heading("rob_id", text="Robot ID")
444>         self.tree.column("rob_name", width=120, anchor=N)
445>         self.tree.heading("rob_name", text="Robot name")
446>         self.tree.column("rob_ip", width=120, anchor=N)
447>         self.tree.heading("rob_ip", text="IP address")
448>         self.tree.pack()
449>
450>         totrec = 0 # total number of records
451>         roblist=[]
452>         f = open('robconfig.txt', 'r')
453>         for line in f:
454>             startpos = 0 # char start position in string
455>             robrec=[]
456>             charpos = line.find(',')
457>             while charpos != -1:
458>                 robrec.append(line[startpos:charpos])
459>                 startpos = charpos + 1
460>                 charpos = line.find(',', startpos, len(line))
461>                 roblist.append(str(robrec[0] + ", " + robrec[1] +
", " + robrec[2]))
462>                 totrec = totrec + 1
463>         f.close()
464>         roblist.sort()
465>         for i in range(len(roblist)):
466>             s = str(i+1)
467>             startpos = 0
468>             charpos = roblist[i].find(',')
469>             str1 = roblist[i][startpos:charpos]
470>             startpos = charpos + 1
471>             charpos = roblist[i].find(',', startpos,
len(roblist[i]))
472>             str2 = roblist[i][startpos:charpos]
473>             startpos = charpos + 1
474>             str3 = roblist[i][startpos:len(roblist[i])]
475>             self.tree.insert("", "end", text=s,
values=(str1, str2, str3))
476>
477>         self.Roblistwin.focus_force()
```

```
478>
479>     class ClassTaskman(Frame):
480>         def __init__(self):
481>             Frame.__init__(self)
482>             self.Taskman = Toplevel()
483>             self.Taskman.title("Task Management")
484>             global idVar, nameVar, r_maxlenVar, r_maxwtVar,
r_maxhtVar, \
485>                 r_minldVar, r_minlinspdVar, r_minangspdVar,
r_maxrobVar
486>             global r_lrfVar, r_sonVar, r_camVar, r_gripVar,
r_2whlVar
487>             global select, tasklist, taskrec, select2
488>
489>             frame1 = Frame(self.Taskman, width=100, height=100)
490>             frame1.grid(row=0, column=0, columnspan=3)
491>             Label(frame1, text="Task ID").grid(row=0, column=0,
sticky=W)
492>             idVar = StringVar()
493>             ttk.Entry(frame1, width=10,
textvariable=idVar).grid(row=0, column=1, padx=(5,30), pady=20,
sticky=W)
494>             Label(frame1, text="Task name").grid(row=0, column=2,
sticky=W)
495>             nameVar = StringVar()
496>             ttk.Entry(frame1, textvariable=nameVar).grid(row=0,
column=3, padx=(5,30), pady=20, sticky=W)
497>
498>             frame2 = Frame(self.Taskman, width=5, height=10,
borderwidth=2, relief=GROOVE)
499>             frame2.grid(row=1, column=0, padx=10, pady=0,
sticky=NSEW)
500>             Label(frame2, text="Max. robot dimensions
(mm)").grid(row=0, column=0, columnspan=3)
501>             ent_width=5
502>             Label(frame2, text="L").grid(row=1, column=0, padx=15,
pady=0, sticky=W)
503>             r_maxlenVar = StringVar()
504>             ttk.Entry(frame2, width=ent_width,
textvariable=r_maxlenVar).grid(row=2, column=0, padx=5, pady=0,
sticky=W)
505>             Label(frame2, text="W").grid(row=1, column=1, padx=15,
pady=0, sticky=W)
506>             r_maxwtVar = StringVar()
```

```
507>         ttk.Entry(frame2, width=ent_width,
textvariable=r_maxwtVar).grid(row=2, column=1, padx=5, pady=0,
sticky=W)
508>         Label(frame2, text="H").grid(row=1, column=2, padx=15,
pady=0, sticky=W)
509>         r_maxhtVar = StringVar()
510>         ttk.Entry(frame2, width=ent_width,
textvariable=r_maxhtVar).grid(row=2, column=2, padx=5, pady=0,
sticky=W)
511>
512>         frame3 = Frame(self.Taskman, width=10, height=10,
borderwidth=2, relief=GROOVE)
513>         frame3.grid(row=1, column=1, padx=10, pady=0,
sticky=NSEW)
514>         Label(frame3, text="Min. load capacity").grid(row=0,
column=0, padx=0, pady=0, sticky=W)
515>         r_minldVar = StringVar()
516>         ttk.Entry(frame3, width=ent_width,
textvariable=r_minldVar).grid(row=0, column=1, padx=0, pady=0,
sticky=NSEW)
517>         Label(frame3, text="kg").grid(row=0, column=2, padx=0,
pady=0, sticky=W)
518>         Label(frame3, text="Min. linear speed").grid(row=1,
column=0, padx=0, pady=2, sticky=W)
519>         r_minlinspdVar = StringVar()
520>         ttk.Entry(frame3, width=ent_width,
textvariable=r_minlinspdVar).grid(row=1, column=1, padx=0, pady=2,
sticky=NSEW)
521>         Label(frame3, text="m/s").grid(row=1, column=2, padx=0,
pady=0, sticky=W)
522>         Label(frame3, text="Min. angular speed").grid(row=2,
column=0, padx=0, pady=2, sticky=W)
523>         r_minangspdVar = StringVar()
524>         ttk.Entry(frame3, width=ent_width,
textvariable=r_minangspdVar).grid(row=2, column=1, padx=0, pady=2,
sticky=NSEW)
525>         Label(frame3, text="rad/s").grid(row=2, column=2,
padx=0, pady=0, sticky=W)
526>         Label(frame3, text="Max. no. of robots").grid(row=3,
column=0, padx=0, pady=2, sticky=W)
527>         r_maxrobVar = StringVar()
528>         ttk.Entry(frame3, width=ent_width,
textvariable=r_maxrobVar).grid(row=3, column=1, padx=0, pady=2,
sticky=NSEW)
529>
```

```
530>         frame4 = Frame(self.Taskman, width=5, height=10,
borderwidth=2, relief=GROOVE)
531>         frame4.grid(row=1, column=2, padx=10, pady=0,
sticky=NSEW)
532>         Label(frame4, text="Hardware requirements").grid(row=0,
column=0, columnspan=8)
533>         r_lrfVar = IntVar()
534>         ttk.Checkbutton(frame4, width=0,
variable=r_lrfVar).grid(row=1, column=0, padx=0, pady=0, sticky=E)
535>         Label(frame4, text="LRF").grid(row=1, column=1,
padx=(0,30), pady=0, sticky=W)
536>         r_sonVar = IntVar()
537>         ttk.Checkbutton(frame4, width=0,
variable=r_sonVar).grid(row=1, column=2, padx=0, pady=0, sticky=E)
538>         Label(frame4, text="Sonars").grid(row=1, column=3,
padx=(0,10), pady=0, sticky=W)
539>         r_camVar = IntVar()
540>         ttk.Checkbutton(frame4, width=0,
variable=r_camVar).grid(row=2, column=0, padx=0, pady=0, sticky=E)
541>         Label(frame4, text="Camera").grid(row=2, column=1,
padx=(0, 30), pady=0, sticky=W)
542>         r_gripVar = IntVar()
543>         ttk.Checkbutton(frame4, width=0,
variable=r_gripVar).grid(row=2, column=2, padx=0, pady=0, sticky=E)
544>         Label(frame4, text="Grippers").grid(row=2, column=3,
padx=(0, 10), pady=0, sticky=W)
545>         r_2whlVar = IntVar()
546>         ttk.Checkbutton(frame4, width=0,
variable=r_2whlVar).grid(row=3, column=0, padx=0, pady=0, sticky=E)
547>         Label(frame4, text="Two-wheel").grid(row=3, column=1,
padx=(0, 30), pady=0, sticky=W)
548>
549>         frame5 = Frame(self.Taskman)
550>         frame5.grid(row=2, column=0, padx=10, pady=(15,0),
columnspan=4, sticky=NSEW)
551>         ttk.Button(frame5, text=" Add  ",
command=self.addEntry).grid(row=0, column=0, padx=(0, 0), pady=0,
sticky=W)
552>         ttk.Button(frame5, text="Update",
command=self.updateEntry).grid(row=0, column=1, padx=(0, 0), pady=0,
sticky=W)
553>         ttk.Button(frame5, text="Delete",
command=self.deleteEntry).grid(row=0, column=2, padx=(0, 0), pady=0,
sticky=W)
```

```
554>         ttk.Button(frame5, text=" Load ",
    command=self.loadEntry).grid(row=0, column=3, padx=(0, 0), pady=0,
    sticky=W)
555>
556>         frame6 = Frame(self.Taskman)
557>         frame6.grid(row=3, column=2, padx=7, pady=(0, 0),
    columnspan=1, sticky=NSEW)
558>         Label(frame6, text="Compatible Robots").grid(row=0,
    column=0, padx=0, pady=0, sticky=NSEW)
559>
560>         frame7 = Frame(self.Taskman)
561>         frame7.grid(row=4, column=0, padx=10, pady=(0,5),
    columnspan=5, sticky=NSEW)
562>         scroll1 = ttk.Scrollbar(frame7, orient=VERTICAL)
563>         select = Listbox(frame7, yscrollcommand=scroll1.set,
    height=6, width=30)
564>         scroll1.config(command=select.yview)
565>         scroll1.grid(row=0, column=5, padx=(0, 0), pady=0,
    sticky=NS)
566>         select.grid(row=0, column=0, padx=(0, 0), pady=0,
    columnspan=4, sticky=NSEW)
567>
568>         frame8 = Frame(self.Taskman)
569>         frame8.grid(row=4, column=2, padx=10, pady=(0,5),
    columnspan=5, sticky=NSEW)
570>         scroll12 = ttk.Scrollbar(frame8, orient=VERTICAL)
571>         select2 = Listbox(frame8, yscrollcommand=scroll12.set,
    height=6, width=34)
572>         scroll12.config(command=select2.yview)
573>         scroll12.grid(row=0, column=7, padx=(0, 0), pady=0,
    sticky=NS)
574>         select2.grid(row=0, column=0, padx=(0, 0), pady=0,
    columnspan=4, sticky=NSEW)
575>
576>         self.setSelect() # configure roblast and select
    listbox
577>         self.Taskman.focus_force()
578>
579>         def setSelect(self):
580>             totrec = 0 # total number of records
581>             global tasklist, taskrec
582>             tasklist=[]
583>             f = open('taskconfig.txt', 'r')
584>             for line in f:
585>                 startpos = 1 # char start position in string
```



```
586>         taskrec=[]
587>         charpos = line.find(',')
588>         while charpos != -1:
589>             taskrec.append(line[startpos:charpos])
590>             startpos = charpos + 1
591>             charpos = line.find(',', startpos, len(line))
592>         tasklist.append(str(taskrec[0] + ", " +
    taskrec[1]))
593>         totrec = totrec + 1
594>         f.close()
595>         tasklist.sort()
596>         select.delete(0, END)
597>         for i in range(totrec):
598>             select.insert(END, tasklist[i])
599>
600>         def setSelect2(self, tid):
601>             CompatRobMat = LibHetRobots.getCompatRobots(tid)
602>             select2.delete(0, END)
603>             for i in range(len(CompatRobMat)):
604>                 select2.insert(END, str(CompatRobMat[i][0] + ', ' +
    CompatRobMat[i][1]))
605>
606>         def whichSelected(self):
607>             return int(select.curselection()[0])
608>
609>         def addEntry(self):
610>             f = open('taskconfig.txt', 'a')
611>             f.write('%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,E\n'
    % (idVar.get(), nameVar.get(),
612>         r_maxlenVar.get(), r_maxwtVar.get(),
613>         r_maxhtVar.get(), r_minldVar.get(),
614>         r_minlinspdVar.get(), r_minangspdVar.get(),
615>         r_maxrobVar.get(),
616>         r_lrfVar.get(), r_sonVar.get(), r_camVar.get(),
617>         r_gripVar.get(), r_2whlVar.get()))
618>         f.close()
619>         self.setSelect()
620>
```

```
621>         def updateEntry(self):
622>             str1 = select.get(ACTIVE) # get the text of the
           currently selected item in the listbox
623>             str2 = str1[str1.find(',')]
624>             lines = open('taskconfig.txt', 'r').readlines()
625>             for i in range(len(lines)):
626>                 if lines[i].find(str2) != -1:
627>                     lines[i] =
           '%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,E\n' % (idVar.get(),
           nameVar.get(),
628>             r_maxlenVar.get(), r_maxwtVar.get(),
629>             r_maxhtVar.get(), r_minldVar.get(),
630>             r_minlinspdVar.get(), r_minangspdVar.get(),
631>             r_maxrobVar.get(),
632>             r_lrfVar.get(), r_sonVar.get(), r_camVar.get(),
633>             r_gripVar.get(), r_2whlVar.get())
634>                 break
635>             f = open('taskconfig.txt', 'w')
636>             f.writelines(lines)
637>             f.close()
638>             self.setSelect()
639>
640>         def deleteEntry(self):
641>             str1 = select.get(ACTIVE) # get the text of the
           currently selected item in the listbox
642>             str2 = str1[str1.find(',')]
643>             lines = open('taskconfig.txt', 'r').readlines()
644>
645>             for i in range(len(lines)):
646>                 if lines[i].find(str2) != -1:
647>                     lines.remove(lines[i])
648>                     break
649>             f = open('taskconfig.txt', 'w')
650>             f.writelines(lines)
651>             f.close()
652>             self.setSelect()
653>
654>         def loadEntry(self):
```

```
655>         global taskrec
656>         str1 = select.get(ACTIVE) # get the text of the
        currently selected item in the listbox
657>         str2 = str1[0:str1.find(',')]
658>         f = open('taskconfig.txt', 'r')
659>         for line in f:
660>             if line.find(str2) != -1:
661>                 startpos = 1 # char start position in string
662>                 taskrec = []
663>                 charpos = line.find(',')
664>                 while charpos != -1:
665>                     taskrec.append(line[startpos:charpos])
666>                     startpos = charpos + 1
667>                     charpos = line.find(',', startpos,
        len(line))
668>                 break # breaks the for-loop
669>         f.close()
670>         # set the fields on the form:
671>         idVar.set(taskrec[0])
672>         nameVar.set(taskrec[1])
673>         r_maxlenVar.set(taskrec[2])
674>         r_maxwtVar.set(taskrec[3])
675>         r_maxhtVar.set(taskrec[4])
676>         r_minldVar.set(taskrec[5])
677>         r_minlinspdVar.set(taskrec[6])
678>         r_minangspdVar.set(taskrec[7])
679>         r_maxrobVar.set(taskrec[8])
680>         r_lrfVar.set(taskrec[9])
681>         r_sonVar.set(taskrec[10])
682>         r_camVar.set(taskrec[11])
683>         r_gripVar.set(taskrec[12])
684>         r_2whlVar.set(taskrec[13])
685>         self.setSelect2(int(taskrec[0]))
686>
687>         class ClassAssigntags(Frame):
688>             def __init__(self):
689>                 Frame.__init__(self)
690>                 self.Win_tags = Toplevel()
691>                 self.Win_tags.title("Algorithm and robot tag
        assignment")
692>                 self.grid(sticky=W + E + N + S)
693>                 self.Win_tags.focus_force()
694>
695>                 # Algorithm labels and fields:
```

```

696>         self.FileTextVar = StringVar()
697>         algoFrame = ttk.Frame(self.Win_tags, borderwidth=2,
        relief=GROOVE)
698>         algoFrame.grid(row=0, column=0, padx=5, pady=10,
        sticky=N)
699>         label_a = ttk.Label(algoFrame, text="Algorithm Tags")
700>         label_a.grid(row=0, column=0, padx=5, pady=10,
        columnspan=5, sticky=N)
701>         label1 = ttk.Label(algoFrame, text="Variable File")
702>         label1.grid(row=1, column=0, padx=5, pady=10,
        sticky=NSEW)
703>         label2 = ttk.Entry(algoFrame, background='white',
        width=30, textvariable=self.FileTextVar)
704>         label2.grid(row=1, column=1, padx=(0,5), pady=10,
        sticky=NSEW)
705>         browse_butn = ttk.Button(algoFrame, text="Browse...",
        command=self.load_file, width=10)
706>         browse_butn.grid(row=1, column=2, padx=5, pady=10,
        sticky=NSEW)
707>         self.algotree = ttk.Treeview(algoFrame)
708>         self.algotree.grid(row=2, column=0, padx=(10,0) ,
        pady=0, columnspan=5, sticky=NSEW)
709>
710>         # Robot labels and fields:
711>         robFrame = ttk.Frame(self.Win_tags, borderwidth=2,
        relief=GROOVE)
712>         robFrame.grid(row=0, column=1, padx=(5,10), pady=10,
        sticky=N)
713>         label_r = ttk.Label(robFrame, text="Robot Tags")
714>         label_r.grid(row=0, column=0, padx=(50,0), pady=10,
        columnspan=3, sticky=N)
715>         label_sel = ttk.Label(robFrame, text="Select robot")
716>         label_sel.grid(row=1, column=0, padx=(5,0), pady=10,
        sticky=W)
717>         self.rbox_value = StringVar()
718>         self.rbox = ttk.Combobox(robFrame,
        textvariable=self.rbox_value)
719>         self.rbox.grid(row=1 , column=1, padx=5, pady=12,
        sticky=W)
720>         self.rbox['values'] = getRoblist() # get the list of
        robot id's and names from text file
721>         self.rbox.bind("<<ComboboxSelected>>",
        self.NewSelection)
722>         self.robtree = ttk.Treeview(robFrame)

```

```
723>         self.robtree.grid(row=2, column=0, padx=(10, 0),
724>         pady=0, columnspan=5, sticky=NSEW)
725>         # Algorithm tree:
726>         self.algotree.bind("<Double-1>", self.OnDoubleClick)
727>         self.algotree["columns"] = ("opc_col")
728>         self.algotree.column("opc_col", width=150)
729>         self.algotree.column("#0", width=150)
730>         self.algotree.heading('#0', text='Algorithm variables',
731>         anchor=N)
732>         self.algotree.heading("opc_col", text="OPC Tag",
733>         anchor=N)
734>         ysb = ttk.Scrollbar(algoFrame, orient=VERTICAL,
735>         command=self.algotree.yview)
736>         xsb = ttk.Scrollbar(algoFrame, orient=HORIZONTAL,
737>         command=self.algotree.xview)
738>         # add tree and scrollbars to frame
739>         self.algotree.grid(in_=algoFrame, row=2, column=0,
740>         sticky=NSEW)
741>         ysb.grid(in_=algoFrame, row=2, column=5, padx=(0,10),
742>         pady=(15,0), sticky=NS)
743>         xsb.grid(in_=algoFrame, row=3, column=0, padx=(10,0),
744>         pady=(0,10), columnspan=5, sticky=EW)
745>         self.algotree['yscroll'] = ysb.set
746>         self.algotree['xscroll'] = xsb.set
747>
748>         # Robot tree:
749>         self.robtree.bind("<Double-1>", self.OnDoubleClick2)
750>         self.robtree["columns"] = ("opc_col")
751>         self.robtree.column("opc_col", width=150)
752>         self.robtree.column("#0", width=150)
753>         self.robtree.heading('#0', text='Robot variables',
754>         anchor=N)
755>         self.robtree.heading("opc_col", text="OPC Tag",
756>         anchor=N)
757>         ysbr = ttk.Scrollbar(robFrame, orient=VERTICAL,
758>         command=self.robtree.yview)
759>         xsbr = ttk.Scrollbar(robFrame, orient=HORIZONTAL,
760>         command=self.robtree.xview)
761>         # add tree and scrollbars to frame
762>         self.robtree.grid(in_=robFrame, row=2, column=0,
763>         sticky=NSEW)
764>         ysbr.grid(in_=robFrame, row=2, column=5, padx=(0, 10),
765>         pady=(15, 0), sticky=NS)
```

```

753>         xsbr.grid(in_robFrame, row=3, column=0, padx=(10, 0),
       pady=(0, 10), columnspan=5, sticky=EW)
754>         self.robtree['yscroll'] = ysbr.set
755>         self.robtree['xscroll'] = xsbr.set
756>
757>         global tagstr
758>         tagstr = ''
759>         self.Win_tags.after(0, self.update_clock)
760>
761>     def update_clock(self):
762>         global tagstr
763>         if tagstr != '':
764>             try:
765>                 if treestore == "algotree":
766>                     itemi = self.algotree.selection()[0]
767>                     self.algotree.item(itemi, values=tagstr)
768>                     tagstr = ''
769>                     x = self.algotree.get_children()
770>                     taglist=[]
771>                     for i in range(len(x)):
772>                         str1 = str(self.algotree.item(x[i],
       "value"))
773>                         startpos=str1.find('(',0,len(str1))
774>                         endpos=str1.find(')',startpos+1,len(str1))
775>                         taglist.append(str1[startpos+2:endpos-
       2])
776>                     f = open('taglist.txt', 'w')
777>                     for i in range(len(x)):
778>                         f.write('%s\n' % str(taglist[i]))
779>                     f.close()
780>                     # create the variable-tag map in a text
       file:
781>                     f = open('avarmap.txt', 'w')
782>                     for i in range(len(x)):
783>                         f.write('%s:%s\n' %
       (str(self.algotree.item(x[i], "text")), taglist[i]))
784>                     f.close()
785>                 else:
786>                     itemi = self.robtree.selection()[0]
787>                     self.robtree.item(itemi, values=tagstr)
788>                     tagstr = ''
789>                     x = self.robtree.get_children()
790>                     taglist=[]

```

```

791>         for i in range(len(x)):
792>             str1 = str(self.robtree.item(x[i],
           "value"))
793>             startpos=str1.find('(',0,len(str1))
794>             endpos=str1.find(')',startpos+1,len(str1))
795>             taglist.append(str1[startpos+2:endpos-
           2])
796>             # get the current selected robot ID:
797>             cr_id =
           self.rbox_value.get()[0:self.rbox_value.get().find(',')]
798>             f = open('robots/%staglist.txt' % cr_id,
           'w')
799>             for i in range(len(x)):
800>                 f.write('%s\n' % str(taglist[i]))
801>             f.close()
802>             # create the variable-tag map in a text
           file:
803>             f = open('robots/%svarmap.txt' % cr_id,
           'w')
804>             for i in range(len(x)):
805>                 f.write('%s:%s\n' %
           (str(self.robtree.item(x[i], "text")), taglist[i]))
806>             f.close()
807>             except:
808>                 print "no variable selected"
809>                 self.Win_tags.after(500, self.update_clock)
810>
811>         def load_file(self):
812>             fname =
           tkFileDialog.askopenfilename(filetypes=(("Python files", "*.py"),
813>             ("HTML files", "*.html;*.htm"),
814>             ("All files", "*.*")))
815>             if fname:
816>                 try:
817>                     #print "Filename: ", fname
818>                     self.FileTextVar.set(fname) # set the label2
           textvariable
819>                     algopath = open('algopath.txt', 'r').readline()
           # get the previous algo file path
820>                     avars = []
821>                     for name, data in inspect.getmembers(gvar,
           inspect.isclass):

```

```
822>         avars.append(name)
823>         lines = open('taglist.txt', 'r').readlines()
824>         for i in range(len(avars)):
825>             s = avars[i]
826>             #print s
827>             if (len(lines[i]) > 5) and
            (algotree==fname): # check if missing assignments and if there is a
            new file
828>                 self.algotree.insert("", "end", text=s,
            values=lines[i])
829>             else:
830>                 self.algotree.insert("", "end", text=s,
            values="not_assigned")
831>             f = open('algotree.txt', 'w')
832>             f.write(fname) # store the algo file path
833>             f.close()
834>             self.Win_tags.focus_force()
835>         except:
836>             print "Open Source File", "Failed to read
            file\n'%s'" % fname
837>             return
838>
839>     def OnDoubleClick(self, event):
840>         global treestore
841>         treestore = "algotree"
842>         d = ClassOpctags()
843>         d.opcTags() # call opctag pop-up window
844>
845>     def OnDoubleClick2(self, event):
846>         global treestore
847>         treestore = "robtree"
848>         d = ClassOpctags()
849>         d.opcTags() # call opctag pop-up window
850>
851>     def NewSelection(self, event):
852>         cr_id =
            self.rbox_value.get()[0:self.rbox_value.get().find(',')]
853>         rvars = []
854>         rvars.append("x_location")
855>         rvars.append("y_location")
856>         rvars.append("yaw")
857>         rvars.append("busy")
858>         rvars.append("current_task")
859>         rvars.append("task_complete")
```



```
860>         self.robtree.delete(*self.robtree.get_children())
861>         try:
862>             lines = open('robots/%staglist.txt' % cr_id,
      'r').readlines()
863>         except:
864>             f=open('robots/%staglist.txt' % cr_id, 'w')
865>             lines=[]
866>             for i in range(len(rvars)):
867>                 f.write('\n')
868>                 lines.append('\n')
869>             f.close()
870>         for i in range(len(rvars)):
871>             s = rvars[i]
872>             if len(lines[i]) > 5: # check if missing
      assignments
873>                 self.robtree.insert("", "end", text=s,
      values=lines[i])
874>             else:
875>                 self.robtree.insert("", "end", text=s,
      values="not_assigned")
876>
877>         class ClassOpctags(Frame):
878>             def __init__(self):
879>                 Frame.__init__(self)
880>                 self.Win_opc = Toplevel()
881>                 self.Win_opc.title("OPC Tags")
882>                 self.opctree = ttk.Treeview(self.Win_opc)
883>                 self.opctree.bind("<Double-1>", self.TagDoubleClick)
884>                 self.opc = OpenOPC.open_client('localhost')
885>                 self.opc.connect('Matrikon.OPC.Simulation')
886>
887>             def opcTags(self):
888>                 labell = Label(self.Win_opc, text="Select a tag:",
      height=0, width=10)
889>                 labell.grid(row=0, column=0, padx=5, pady=5, sticky=W)
890>
891>                 self.opctree.heading('#0', text='List of available
      tags', anchor=N)
892>                 self.opctree.grid(row=1, column=0, padx=5, pady=5)
893>                 taglist = self.opc.list('Configured Aliases.Server02')
894>                 for i in range(len(taglist)):
895>                     self.opctree.insert("", "end", text=taglist[i])
896>
897>                 butFrame = Frame(self.Win_opc, width=200, height=100)
```

```
898>         butFrame.grid(row=2, column=0, padx=5, pady=5)
899>         assignButn = ttk.Button(butFrame, text="Assign tag",
    command=self.GetTag)
900>         assignButn.grid(row=0, column=0, padx=5, pady=5,
    sticky=W)
901>         cancelButn = ttk.Button(butFrame, text="Cancel",
    command=self.Win_opc.destroy)
902>         cancelButn.grid(row=0, column=1, padx=5, pady=5,
    sticky=W)
903>
904>         self.Win_opc.focus_force()
905>
906>         def TagDoubleClick(self, event):
907>             itemt = self.opctree.selection()[0]
908>             s = self.opctree.item(itemt, "text")
909>             global tagstr
910>             tagstr = s
911>             self.opc.close()
912>             self.Win_opc.destroy()
913>
914>         def GetTag(self):
915>             itemt = self.opctree.selection()[0]
916>             s = self.opctree.item(itemt, "text")
917>             global tagstr
918>             tagstr = s
919>             self.opc.close()
920>             self.Win_opc.destroy()
921>
922>         class ClassSelectAlgo(Frame):
923>             def __init__(self):
924>                 Frame.__init__(self)
925>                 self.Win_algo = Toplevel()
926>                 self.Win_algo.title("Algorithm selection")
927>                 self.grid(sticky=W + E + N + S)
928>                 self.Win_algo.focus_force()
929>
930>                 # Algorithm labels and fields:
931>                 self.FileTextVar = StringVar()
932>                 labell = ttk.Label(self.Win_algo, text="Algorithm
    File")
933>                 labell.grid(row=1, column=0, padx=5, pady=10,
    sticky=NSEW)
934>                 label2 = ttk.Entry(self.Win_algo, background='white',
    width=30, textvariable=self.FileTextVar)
```

```
935>         label2.grid(row=1, column=1, padx=(0,5), pady=10,
936>         sticky=NSEW)
937>         browse_butn = ttk.Button(self.Win_algo,
938>         text="Browse...", command=self.load_file, width=10)
939>         browse_butn.grid(row=1, column=2, padx=5, pady=10,
940>         sticky=NSEW)
941>         accept_butn = ttk.Button(self.Win_algo, text="Accept",
942>         command=self.Win_algo.destroy, width=10)
943>         accept_butn.grid(row=2, column=2, padx=5, pady=10,
944>         sticky=NSEW)
945>
946>     def load_file(self):
947>         fname =
948>         tkFileDialog.askopenfilename(filetypes=(("Python files", "*.py"),
949>         ("HTML files", "*.html;*.htm"),
950>         ("All files", "*.*")))
951>         if fname:
952>             try:
953>                 #print "Filename: ", fname
954>                 self.FileTextVar.set(fname) # set the label2
955>                 textvariable
956>                 self.Win_algo.focus_force()
957>             except:
958>                 print "Open Source File", "Failed to read
959>                 file\n'%s'" % fname
960>             return
961>
962>     class ClassLocationConfig(Frame):
963>         def __init__(self):
964>             Frame.__init__(self)
965>             self.Win_location = Toplevel()
966>             self.Win_location.title("Task location configuration")
967>             self.grid(sticky=W + E + N + S)
968>             self.opc = OpenOPC.open_client('localhost')
969>             self.opc.connect('Matrikon.OPC.Simulation')
970>             self.Win_location.focus_force()
971>
972>         # Frames:
973>         locFrame = ttk.Frame(self.Win_location, width=200,
974>         height=200, borderwidth=2, relief=GROOVE)
975>         locFrame.grid(row=0, column=0, padx=2, pady=2,
976>         sticky=NS)
```

```
967>         moveFrame = ttk.Frame(self.Win_location, width=200,
    height=200, borderwidth=2, relief=GROOVE)
968>         moveFrame.grid(row=0, column=1, padx=2, pady=2,
    sticky=NS)
969>         taskFrame = ttk.Frame(self.Win_location, width=200,
    height=200, borderwidth=2, relief=GROOVE)
970>         taskFrame.grid(row=1, column=0, padx=2, pady=(2,50),
    colspan=6, sticky=NSEW)
971>         connFrame = ttk.Frame(self.Win_location, width=200,
    height=20, borderwidth=2, relief=GROOVE)
972>         connFrame.grid(row=2, column=0, padx=2, pady=2,
    colspan=6, sticky=NSEW)
973>
974>         # Current robot location widgets:
975>         label1 = ttk.Label(locFrame, text="Current robot
    location")
976>         label1.grid(row=1, column=0, padx=5, pady=5,
    colspan=5, sticky=NSEW)
977>         label2 = ttk.Label(locFrame, text="x")
978>         label2.grid(row=2, column=0, padx=(0,0), pady=5,
    sticky=E)
979>         ttk.Label(locFrame, text="m").grid(row=2, column=2,
    padx=(0, 20), pady=5, sticky=W)
980>         self.r_curr_xlocVar = StringVar()
981>         label2b = ttk.Label(locFrame, background='light grey',
    width=6, textvariable=self.r_curr_xlocVar)
982>         label2b.grid(row=2, column=1, padx=(5,0), pady=5,
    sticky=W)
983>         label3 = ttk.Label(locFrame, text="y")
984>         label3.grid(row=3, column=0, padx=(0,0), pady=5,
    sticky=E)
985>         ttk.Label(locFrame, text="m").grid(row=3, column=2,
    padx=(0, 20), pady=5, sticky=W)
986>         self.r_curr_ylocVar = StringVar()
987>         label3b = ttk.Label(locFrame, background='light grey',
    width=6, textvariable=self.r_curr_ylocVar)
988>         label3b.grid(row=3, column=1, padx=(5,0), pady=5,
    sticky=W)
989>         label4 = ttk.Label(locFrame, text="yaw")
990>         label4.grid(row=4, column=0, padx=(0,0), pady=5,
    sticky=E)
991>         ttk.Label(locFrame, text="rad").grid(row=4, column=2,
    padx=(0, 20), pady=5, sticky=W)
992>         self.r_curr_yawlocVar = StringVar()
```

```
993>         label4b = ttk.Label(locFrame, background='light grey',
width=6 , textvariable=self.r_curr_yawlocVar)
994>         label4b.grid(row=4, column=1, padx=(5,0) , pady=5,
sticky=W)
995>         labelconn = ttk.Label(connFrame, text="Connected to
robot %s (%s)" % (ron_id, getRobname(ron_id)))
996>         labelconn.grid(row=0, column=0, padx=(0, 0), pady=5,
sticky=W)
997>         self.r_curr_xlocVar.set('0.55')
998>         self.r_curr_ylocVar.set('10.55')
999>         self.r_curr_yawlocVar.set('150.55')
1000>
1001>         # Robot move widgets:
1002>         ttk.Label(moveFrame, text="Robot motion
commands").grid(row=0, column=0, padx=5, pady=5, columns=3,
sticky=N)
1003>         fwd_butn = ttk.Button(moveFrame, text="Forward",
command=self.move_fwd).grid(row=1, column=1, padx=5, pady=5)
1004>         stop_butn = ttk.Button(moveFrame, text="Stop",
command=self.stop).grid(row=2, column=1, padx=5, pady=5)
1005>         left_butn = ttk.Button(moveFrame, text="Left",
command=self.move_left).grid(row=2, column=0, padx=10, pady=5)
1006>         right_butn = ttk.Button(moveFrame, text="Right",
command=self.move_right).grid(row=2, column=2, padx=5, pady=5)
1007>         rev_butn = ttk.Button(moveFrame, text="Reverse",
command=self.move_reverse).grid(row=3, column=1, padx=5, pady=5)
1008>
1009>         # Task widgets:
1010>         label_heading = ttk.Label(taskFrame, text="Task
locations")
1011>         label_heading.grid(row=0, column=0 , padx=5,
pady=(5,10), columns=8, sticky=N)
1012>         ttk.Label(taskFrame, text="Select Task").grid(row=1,
column=0, padx=5, pady=5, sticky=NSEW)
1013>         self.tbox_value = StringVar()
1014>         self.tbox = ttk.Combobox(taskFrame, width=10,
textvariable=self.tbox_value)
1015>         self.tbox.grid(row=1, column=1, padx=(0,5),
pady=(10,20), columns=2 , sticky=NSEW)
1016>         self.tbox['values'] = getTasklist() # get the list of
task id's and names from text file
1017>         self.tbox.bind("<<ComboboxSelected>>",
self.TaskSelection)
1018>         ttk.Label(taskFrame, text="x").grid(row=2, column=2,
padx=5, pady=(0,0), sticky=S)
```

```
1019>         ttk.Label(taskFrame, text="y").grid(row=2, column=3,
1020>         padx=5, pady=(0,0), sticky=S)
1021>         ttk.Label(taskFrame, text="yaw").grid(row=2, column=4,
1022>         padx=5, pady=(0,0), sticky=S)
1023>         ttk.Label(taskFrame, text="Select Node").grid(row=3,
1024>         column=0, padx=5, pady=5, sticky=W)
1025>         self.nbox_value = StringVar()
1026>         self.nbox = ttk.Combobox(taskFrame, width=10,
1027>         textvariable=self.nbox_value)
1028>         self.nbox.grid(row=3, column=1, padx=(0,5), pady=5,
1029>         sticky=W)
1030>         self.nbox['values'] = ('Source', 'Destination')
1031>         self.nbox.bind("<<ComboboxSelected>>",
1032>         self.NodeSelection)
1033>         self.r_node_xlocVar = StringVar()
1034>         label_node_x = ttk.Label(taskFrame, background='light
1035>         grey', width=6 , textvariable=self.r_node_xlocVar)
1036>         label_node_x.grid(row=3, column=2, padx=5, pady=2,
1037>         sticky=W)
1038>         self.r_node_ylocVar = StringVar()
1039>         label_node_y = ttk.Label(taskFrame, background='light
1040>         grey', width=6, textvariable=self.r_node_ylocVar)
1041>         label_node_y.grid(row=3, column=3, padx=5, pady=2,
1042>         sticky=W)
1043>         self.r_node_yawlocVar = StringVar()
1044>         label_node_yaw = ttk.Label(taskFrame, background='light
1045>         grey', width=6, textvariable=self.r_node_yawlocVar)
1046>         label_node_yaw.grid(row=3, column=4, padx=5, pady=2,
1047>         sticky=W)
1048>         ttk.Button(taskFrame, text="  Save node  ",
1049>         command=self.save_location).grid(row=3, column=5, padx=(5,5), pady=5,
1050>         sticky=E)
1051>
1052>         def move_fwd(self):
1053>             wkey = 'i'
1054>             self.butn_event(wkey)
1055>
1056>         def move_left(self):
1057>             wkey = 'u'
1058>             self.butn_event(wkey)
1059>
1060>         def move_right(self):
1061>             wkey = 'o'
1062>             self.butn_event(wkey)
1063>
```

```
1050>         def move_reverse(self):
1051>             wkey = ','
1052>             self.butn_event(wkey)
1053>
1054>         def stop(self):
1055>             wkey = 'k'
1056>             self.butn_event(wkey)
1057>
1058>         def butn_event(self, keysend):
1059>             self.opc.write(('Server01.keystroke', keysend))
1060>
1061>         def save_location(self):
1062>             self.r_node_xlocVar.set(self.r_curr_xlocVar.get())
1063>             self.r_node_ylocVar.set(self.r_curr_ylocVar.get())
1064>             self.r_node_yawlocVar.set(self.r_curr_yawlocVar.get())
1065>             t_id =
1066>                 self.tbox_value.get()[0:self.tbox_value.get().find(',')] # get task
1067>                 ID
1068>                 n_id = self.nbox_value.get()
1069>                 f = open('robots/%slocations.txt' % ron_id, 'a')
1070>                 f.write('%s,%s,%s,%s,%s,E\n' % (t_id, n_id,
1071>                 self.r_curr_xlocVar.get(), self.r_curr_ylocVar.get(),
1072>                 self.r_curr_yawlocVar.get()))
1073>                 f.close()
1074>
1075>         def TaskSelection(self, event):
1076>             if self.nbox_value.get() != "":
1077>                 lclist =
1078>                     getLoclist(self.tbox_value.get()[0:self.tbox_value.get().find(',')],
1079>                     self.nbox_value.get())
1080>                 try:
1081>                     if len(lclist) != 0:
1082>                         self.r_node_xlocVar.set(lclist[0])
1083>                         self.r_node_ylocVar.set(lclist[1])
1084>                         self.r_node_yawlocVar.set(lclist[2])
1085>                     except:
1086>                         self.r_node_xlocVar.set('')
1087>                         self.r_node_ylocVar.set('')
1088>                         self.r_node_yawlocVar.set('')
1089>
1090>         def NodeSelection(self, event):
1091>             if self.tbox_value.get() != "":
```

```
1087>         lclist =
           getLoclist(self.tbox_value.get()[0:self.tbox_value.get().find(',')],
           self.nbox_value.get())
1088>         try:
1089>             if len(lclist) != 0:
1090>                 self.r_node_xlocVar.set(lclist[0])
1091>                 self.r_node_ylocVar.set(lclist[1])
1092>                 self.r_node_yawlocVar.set(lclist[2])
1093>         except:
1094>             self.r_node_xlocVar.set('')
1095>             self.r_node_ylocVar.set('')
1096>             self.r_node_yawlocVar.set('')
1097>
1098>     if __name__ == "__main__":
1099>         app = App()
```


Appendix B: Remote robot control code

```
1> # Filename: rosAI.py
2> import sys
3> import rospy
4> import actionlib
5> import OPCdata
6> import math
7> from multiprocessing import Process, Value, Array
8> from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
9> from math import radians, degrees
10> from actionlib_msgs.msg import *
11> from geometry_msgs.msg import Point, PoseWithCovarianceStamped,
    Pose2D
12> from nav_msgs.msg import Odometry
13>
14>
15> class map_navigation():
16>
17>     def __init__(self):
18>         # declare the coordinates of interest
19>         global xg, yg
20>         global xgprev, ygprev, statenum
21>         xg=0.0
22>         yg=0.0
23>         xgprev=0.0
24>         ygprev=0.0
25>         self.goalReached = False
26>         # initialise
27>         rospy.init_node('map_navigation', anonymous=False)
28>         statenum = 0
29>         tripcount = 0
30>
31>         while (1):
32>             global rstatus
33>
34>                 if (statenum==0):
35>                     try:
36>                         xgprev=xg
37>                         ygprev=yg
```

```
38>                 xg, yg = OPCdata.getxyGoal(xg,yg)
39>                 print (xg, yg)
40>                 statenum=1
41>             except OPCdata.OpenOPC.TimeoutError:
42>                 print "TimeoutError occured"
43>
44>             if (statenum==1):
45>                 try:
46>                     xg, yg = OPCdata.getxyGoal(xg,yg)
47>                     self.goalReached = self.moveToGoal(xg,
yg)
48>             except OPCdata.OpenOPC.TimeoutError:
49>                 print "TimeoutError occured"
50>
51>             if ((statenum==1) & (self.goalReached)):
52>                 try:
53>                     rstatus=0
54>                     OPCdata.setStatus(rstatus)
55>                     statenum=2
56>             except OPCdata.OpenOPC.TimeoutError:
57>                 print "TimeoutError occured"
58>
59>             if (statenum==2):
60>                 statenum=0
61>                 rospy.sleep(5)
62>
63>     def shutdown(self):
64>         # stop the robot
65>         rospy.loginfo("Quit program")
66>         rospy.sleep()
67>
68>     def moveToGoal(self,xGoal,yGoal):
69>         global xc, yc
70>         #define a client for to send goal requests to the
move_base server through a SimpleActionClient
71>         ac = actionlib.SimpleActionClient("/robot_0/move_base",
MoveBaseAction)
72>
73>         #wait for the action server to come up
74>         while(not
ac.wait_for_server(rospy.Duration.from_sec(1000.0))):
75>             self.shutdown()
76>             goal = MoveBaseGoal()
77>             pose_xy = PoseWithCovarianceStamped()
```

```
78>         print "xc, yc:", xc, yc
79>     print "xg, yg:", xGoal, yGoal
80>
81>         #set up the frame parameters
82>     goal.target_pose.header.frame_id = "map"
83>     goal.target_pose.header.stamp = rospy.Time.now()
84>
85>         # moving towards the goal*/
86>     goal.target_pose.pose.position = Point(xGoal,yGoal,0)
87>     goal.target_pose.pose.orientation.x = 0.0
88>     goal.target_pose.pose.orientation.y = 0.0
89>     goal.target_pose.pose.orientation.z = 0.0
90>     goal.target_pose.pose.orientation.w = 1.0
91>     rospy.loginfo("Sending goal location...")
92>     ac.send_goal(goal)
93>     ac.wait_for_result(rospy.Duration(60))
94>         dx = xGoal-xc
95>         dy = yGoal-yc
96>         dist = math.sqrt(math.pow(dx,2) + math.pow(dy,2))
97>     print "DISTANCE: ", dist
98>
99>         #if(ac.get_state() == GoalStatus.SUCCEEDED):
100>             if dist<=0.5:
101>                 rospy.loginfo("The robot has reached the
destination")
102>                 return True
103>             else:
104>                 rospy.loginfo("The robot failed to reach the
destination")
105>                 return False
106>
107>     def amclCb(msg):
108>         global xc, yc
109>         xc = msg.pose.pose.position.x
110>         yc = msg.pose.pose.position.y
111>         OPCdata.xyPos(xc,yc)
112>
113>     if __name__ == '__main__':
114>         try:
115>             rospy.Subscriber('/robot_0/amcl_pose',PoseWithCovarianceStamped,amclC
b)
116>                 rstatus = Value('i', 0)
117>                 wait_time = Value('i', 0)
```

```
118>         OPCwait = Process(target=OPCdata.getwtime,
    args=(wait_time,))
119>         OPCwait.start()
120>         Navproc = Process(target = map_navigation())
121>         Navproc.start()
122>         rospy.spin()
123>
124>     except rospy.ROSInterruptException:
125>         rospy.loginfo("map_navigation node terminated.")
```

Appendix C: Robot OPC-UA Client code

```
1> # Filename: OPCdata.py
2> import sys
3> import rospy
4> import gvar
5> import OpenOPC # the OPC client library for Python
6>
7> def getxyGoal(x,y):
8>     opc = OpenOPC.open_client('192.168.201.1')
9>     opc.connect('Matrikon.OPC.Simulation')
10>    x=opc.read('Server02.r01_xgoal')[0]
11>    y=opc.read('Server02.r01_ygoal')[0]
12>    opc.close()
13>    return x, y
14>
15> def setStatus(status):
16>    opc = OpenOPC.open_client('192.168.201.1')
17>    opc.connect('Matrikon.OPC.Simulation')
18>    opc.write(('Server02.r01_status', status))
19>    opc.close()
20>
21> def xyPos(x,y):
22>    opc = OpenOPC.open_client('192.168.201.1')
23>    opc.connect('Matrikon.OPC.Simulation')
24>    opc.write(('Server02.r01_xpos', x))
25>    opc.write(('Server02.r01_ypos', y))
26>    opc.close()
27>
28> def getwtime(w):
29>    w.value=5
```


Appendix D: Stage development programs

D1. Launch file

```
1> <!--Filename: turtlebot_stage_psu.launch-->
2> <launch>
3>   <arg name="base"          default="$ (optenv TURTLEBOT_BASE kobuki)"/>
   <!-- create, rhoomba -->
4>   <arg name="stacks"        default="$ (optenv TURTLEBOT_STACKS
   hexagons)"/> <!-- circles, hexagons -->
5>   <arg name="3d_sensor"     default="$ (optenv TURTLEBOT_3D_SENSOR
   kinect)"/> <!-- kinect, asus_xtion_pro -->
6>   <!-- Name of the map to use (without path nor extension) and
   initial position -->
7>   <arg name="map_file"      default="/home/nicol/catkin-
   ws/src/robsim05/worlds/psu.yaml"/> <!-- psu -->
8>   <arg name="world_file"    default="/home/nicol/catkin-
   ws/src/robsim05/worlds/psu.world"/>
9>   <arg name="initial_pose_x" default="7.925"/>
10>  <arg name="initial_pose_y" default="5.925"/>
11>  <arg name="initial_pose_a" default="0.0"/>
12>  <param name="/use_sim_time" value="true"/>
13>  <!-- ***** Stage ***** -->
14>  <!--
15>     Publishes transforms:
16>     /base_link -> /base_laser
17>     /base_footprint -> /base_link (identity)
18>     /odom -> base_footprint
19>     Publishes topics:
20>     /odom : odometry data from the simulated odometry
21>     /base_scan : laser data from the simulated laser
22>     /base_pose_ground_truth : the ground truth pose
23>     Parameters:
24>     base_watchdog_timeout : time (s) after receiving the last
   command on cmd_vel before stopping the robot
25>     Args:
26>     -g : run in headless mode.
27>     -->
28>  <node pkg="stage_ros" type="stageros" name="stageros" args="$ (arg
   world_file)"/>
```

```

29>   <param name="base_watchdog_timeout" value="0.5"/>
30>   <remap from="odom" to="odom"/>
31>   <remap from="base_pose_ground_truth"
    to="base_pose_ground_truth"/>
32>   <remap from="cmd_vel" to="mobile_base/commands/velocity"/>
33>   <remap from="robot_0/base_scan" to="robot_0/scan_filtered"/>
34>   <remap from="robot_1/base_scan" to="robot_1/scan_filtered"/>
35>   <remap from="robot_2/base_scan" to="robot_2/scan_filtered"/>
36> </node>
37> <!-- ***** Robot Model ***** -->
38> <!-- BEGIN ROBOT 1-->
39> <group ns="robot_0">
40>   <node pkg="nodelet" type="nodelet"
    name="mobile_base_nodelet_manager" args="manager"/>
41>   <node pkg="nodelet" type="nodelet" name="cmd_vel_mux" args="load
    yocs_cmd_vel_mux/CmdVelMuxNodelet mobile_base_nodelet_manager">
42>     <param name="yaml_cfg_file" value="$(find
    turtlebot_bringup)/param/mux.yaml"/>
43>     <remap from="cmd_vel_mux/output"
    to="mobile_base/commands/velocity"/>
44>   </node>
45>   <!-- 4: Localization algorithm: AMCL (http://wiki.ros.org/amcl) -
    ->
46>   <node pkg="amcl" type="amcl" name="amcl" output="screen">>
47>     <param name="transform_tolerance" value="0.2" />
48>     <param name="max_particles" value="1000"/>
49>     <param name="initial_pose_x" value="7.925"/>
50>     <param name="initial_pose_y" value="5.925"/>
51>     <param name="initial_pose_a" value="0.0"/>
52>     <param name="odom_frame_id" value="/robot_0/odom"/>
53>     <param name="base_frame_id" value="/robot_0/base_link"/>
54>
55>     <remap from="map" to="/map"/>
56>     <remap from="scan" to="scan_filtered"/>
57>     <remap from="static_map" to="/static_map"/>
58>   </node>
59>   <node pkg="move_base" type="move_base" respawn="false"
    name="move_base" output="screen">
60>     <rosparam file="$(find
    turtlebot_navigation)/param/costmap_common_params.yaml"
    command="load" ns="global_costmap" />
61>     <rosparam file="$(find
    turtlebot_navigation)/param/costmap_common_params.yaml"
    command="load" ns="local_costmap" />

```



```
62>     <rosparam file="$(find
      turtlebot_navigation)/param/local_costmap_params.yaml" command="load"
      />
63>     <rosparam file="$(find
      turtlebot_navigation)/param/global_costmap_params.yaml"
      command="load" />
64>     <rosparam file="$(find
      turtlebot_navigation)/param/dwa_local_planner_params.yaml"
      command="load" />
65>     <rosparam file="$(find
      turtlebot_navigation)/param/move_base_params.yaml" command="load" />
66>     <rosparam file="$(find
      turtlebot_navigation)/param/global_planner_params.yaml"
      command="load" />
67>     <rosparam file="$(find
      turtlebot_navigation)/param/navfn_global_planner_params.yaml"
      command="load" />
68>     <!-- external params file that could be loaded into the
      move_base namespace -->
69>     <rosparam file="$(find turtlebot_navigation)/param/dummy.yaml"
      command="load" />
70>     <remap from="map" to="/map"/>
71>     <!-- Override MOVE_BASE Frame Params to include "robot_X"
      prefix -->
72>     <param name="global_costmap/laser_scan_sensor/sensor_frame"
      value="/robot_0/base_laser_link"/>
73>     <param name="global_costmap/laser_scan_sensor/topic"
      value="/robot_0/scan_filtered"/>
74>     <param name="global_costmap/robot_base_frame"
      value="/robot_0/base_link"/>
75>
76>     <param name="local_costmap/global_frame"
      value="/robot_0/odom"/>
77>     <param name="local_costmap/laser_scan_sensor/sensor_frame"
      value="/robot_0/base_laser_link"/>
78>     <param name="local_costmap/laser_scan_sensor/topic"
      value="/robot_0/scan_filtered"/>
79>     <param name="local_costmap/robot_base_frame"
      value="/robot_0/base_link"/>
80>     </node>
81> </group>
82> <!-- BEGIN ROBOT 2 -->
83> <group ns="robot_1">
84>     <node pkg="nodelet" type="nodelet"
      name="mobile_base_nodelet_manager" args="manager"/>
```

```
85> <node pkg="nodelet" type="nodelet" name="cmd_vel_mux" args="load
yocs_cmd_vel_mux/CmdVelMuxNodelet mobile_base_nodelet_manager">
86>   <param name="yaml_cfg_file" value="$(find
turtlebot_bringup)/param/mux.yaml"/>
87>   <remap from="cmd_vel_mux/output"
to="mobile_base/commands/velocity"/>
88> </node>
89> <!-- 4: Localization algorithm: AMCL (http://wiki.ros.org/amcl) -
->
90> <node pkg="amcl" type="amcl" name="amcl" output="screen">>
91>   <param name="transform_tolerance" value="0.2" />
92>   <param name="max_particles" value="1000"/>
93>   <param name="initial_pose_x" value="7.925"/>
94>   <param name="initial_pose_y" value="2.925"/>
95>   <param name="initial_pose_a" value="0.0"/>
96>   <param name="odom_frame_id" value="/robot_1/odom"/>
97>   <param name="base_frame_id" value="/robot_1/base_link"/>
98>
99>   <remap from="map" to="/map"/>
100>     <remap from="scan" to="scan_filtered"/>
101>     <remap from="static_map" to="/static_map"/>
102>   </node>
103>   <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
104>     <rosparam file="$(find
turtlebot_navigation)/param/costmap_common_params.yaml"
command="load" ns="global_costmap" />
105>     <rosparam file="$(find
turtlebot_navigation)/param/costmap_common_params.yaml"
command="load" ns="local_costmap" />
106>     <rosparam file="$(find
turtlebot_navigation)/param/local_costmap_params.yaml" command="load"
/>
107>     <rosparam file="$(find
turtlebot_navigation)/param/global_costmap_params.yaml"
command="load" />
108>     <rosparam file="$(find
turtlebot_navigation)/param/dwa_local_planner_params.yaml"
command="load" />
109>     <rosparam file="$(find
turtlebot_navigation)/param/move_base_params.yaml" command="load" />
110>     <rosparam file="$(find
turtlebot_navigation)/param/global_planner_params.yaml"
command="load" />
```

```
111>         <roscparam file="$(find
turtlebot_navigation)/param/navfn_global_planner_params.yaml"
command="load" />
112>         <!-- external params file that could be loaded into the
move_base namespace -->
113>         <roscparam file="$(find
turtlebot_navigation)/param/dummy.yaml" command="load" />
114>         <remap from="map" to="/map"/>
115>         <!-- Override MOVE_BASE Frame Params to include
"robot_x" prefix -->
116>         <param
name="global_costmap/laser_scan_sensor/sensor_frame"
value="/robot_1/base_laser_link"/>
117>         <param name="global_costmap/laser_scan_sensor/topic"
value="/robot_1/scan_filtered"/>
118>         <param name="global_costmap/robot_base_frame"
value="/robot_1/base_link"/>
119>
120>         <param name="local_costmap/global_frame"
value="/robot_1/odom"/>
121>         <param
name="local_costmap/laser_scan_sensor/sensor_frame"
value="/robot_1/base_laser_link"/>
122>         <param name="local_costmap/laser_scan_sensor/topic"
value="/robot_1/scan_filtered"/>
123>         <param name="local_costmap/robot_base_frame"
value="/robot_1/base_link"/>
124>         </node>
125>     </group>
126>     <!-- BEGIN ROBOT 3 -->
127>     <group ns="robot_2">
128>         <node pkg="nodelet" type="nodelet"
name="mobile_base_nodelet_manager" args="manager"/>
129>         <node pkg="nodelet" type="nodelet" name="cmd_vel_mux"
args="load yocs_cmd_vel_mux/CmdVelMuxNodelet
mobile_base_nodelet_manager">
130>             <param name="yaml_cfg_file" value="$(find
turtlebot_bringup)/param/mux.yaml"/>
131>             <remap from="cmd_vel_mux/output"
to="mobile_base/commands/velocity"/>
132>         </node>
133>         <!-- 4: Localization algorithm: AMCL
(http://wiki.ros.org/amcl) -->
134>         <node pkg="amcl" type="amcl" name="amcl" output="screen">>
135>             <param name="transform_tolerance" value="0.2" />
```

```

136>         <param name="max_particles" value="1000"/>
137>         <param name="initial_pose_x" value="7.925"/>
138>         <param name="initial_pose_y" value="4.525"/>
139>         <param name="initial_pose_a" value="0.0"/>
140>         <param name="odom_frame_id" value="/robot_2/odom"/>
141>         <param name="base_frame_id"
    value="/robot_2/base_link"/>
142>
143>         <remap from="map" to="/map"/>
144>         <remap from="scan" to="scan_filtered"/>
145>         <remap from="static_map" to="/static_map"/>
146>     </node>
147>     <node pkg="move_base" type="move_base" respawn="false"
    name="move_base" output="screen">
148>         <rosparam file="$(find
    turtlebot_navigation)/param/costmap_common_params.yaml"
    command="load" ns="global_costmap" />
149>         <rosparam file="$(find
    turtlebot_navigation)/param/costmap_common_params.yaml"
    command="load" ns="local_costmap" />
150>         <rosparam file="$(find
    turtlebot_navigation)/param/local_costmap_params.yaml" command="load"
    />
151>         <rosparam file="$(find
    turtlebot_navigation)/param/global_costmap_params.yaml"
    command="load" />
152>         <rosparam file="$(find
    turtlebot_navigation)/param/dwa_local_planner_params.yaml"
    command="load" />
153>         <rosparam file="$(find
    turtlebot_navigation)/param/move_base_params.yaml" command="load" />
154>         <rosparam file="$(find
    turtlebot_navigation)/param/global_planner_params.yaml"
    command="load" />
155>         <rosparam file="$(find
    turtlebot_navigation)/param/navfn_global_planner_params.yaml"
    command="load" />
156>         <!-- external params file that could be loaded into the
    move_base namespace -->
157>         <rosparam file="$(find
    turtlebot_navigation)/param/dummy.yaml" command="load" />
158>         <remap from="map" to="/map"/>
159>         <!-- Override MOVE_BASE Frame Params to include
    "robot_x" prefix -->

```

```

160>         <param
           name="global_costmap/laser_scan_sensor/sensor_frame"
           value="/robot_2/base_laser_link"/>
161>         <param name="global_costmap/laser_scan_sensor/topic"
           value="/robot_2/scan_filtered"/>
162>         <param name="global_costmap/robot_base_frame"
           value="/robot_2/base_link"/>
163>
164>         <param name="local_costmap/global_frame"
           value="/robot_2/odom"/>
165>         <param
           name="local_costmap/laser_scan_sensor/sensor_frame"
           value="/robot_2/base_laser_link"/>
166>         <param name="local_costmap/laser_scan_sensor/topic"
           value="/robot_2/scan_filtered"/>
167>         <param name="local_costmap/robot_base_frame"
           value="/robot_2/base_link"/>
168>         </node>
169>     </group>
170>     <!-- ***** Maps ***** -->
171>     <node name="map_server" pkg="map_server" type="map_server"
           args="$ (arg map_file) ">
172>         <!-- <param name="frame_id" value="/robot_0/odom"/> -->
173>     </node>
174>     <!-- ***** Visualisation ***** -->
175>     <!--<node name="rviz" pkg="rviz" type="rviz" args="-d $(find
           turtlebot_stage)/rviz/robot_navigation.rviz"/> -->
176> </launch>

```

D2. Yaml file

```

1> # Filename: psu.yaml
2> image: map02.png
3> resolution: 0.025000
4> origin: [0.0000 ,0.0000, 0.000000]
5> negate: 0
6> occupied_thresh: 0.3
7> free_thresh: 0.2
8> obstacle_range: 2.5
9> raytrace_range: 3.0

```

D3. World file

```
1> # Filename: psu.world
2> include "map.inc"
3> include "turtlebot.inc"
4>
5> resolution 0.02
6>
7> window
8> (
9>   size [ 600 700 ] # in pixels
10>  scale 50   # pixels per meter
11>  center [ 0.0 0.0 ]
12>  rotate [ 0 0 ]
13>
14>  show_data 1           # 1=on 0=off
15> )
16>
17> floorplan
18> (
19>  name "psu"
20>  size [ 21.95 8.6 0.8 ] #the real size in meters (from yaml,
    resolution*pixels)
21>  pose [ 10.975 4.3 0.0 0.0 ]
22>  bitmap "map02.png"
23> )
24>
25> # throw in a robot
26> turtlebot
27> (
28>  pose [7.925 5.925 0.0 0.0 ]
29>  #pose [21.95 8.6 0.0 0.0 ]
30>  name "robot0"
31>  color "red"
32> )
33>
34> turtlebot
35> (
36>  pose [7.925 2.925 0.0 0.0 ]
37>  name "robot1"
38>  color "blue"
39> )
40>
```

```
41> turtlebot
42> (
43>  pose [7.925 4.525 0.0 0.0 ]
44>  name "robot2"
45>  color "green"
46> )
```

D4. Map and turtlebot files

```
1> # Filename: map.inc
2>
3> define floorplan model
4> (
5>  color "gray30"
6>
7>  boundary 1
8>
9>  gui_nose 0
10>  gui_grid 1
11>  gui_move 0
12>  gui_outline 0
13>  gripper_return 0
14>  fiducial_return 0
15>  ranger_return 0.5
16>
17>  obstacle_return 0
18> )
19>
20> define zone model
21> (
22>  color      "orange"
23>  size [ 4 4 0.02 ]
24>
25>  gui_nose 0
26>  gui_grid 0
27>  gui_move 1
28>  gui_outline 0
29>
30>  obstacle_return 0
31>  ranger_return -1 # transparent to range sensors
32> )
```

```
1> # Filename: turtlebot.inc
2>
3> define kinect ranger
4> (
5>   sensor
6>   (
7>     range_max 6.5
8>     fov 58.0
9>     samples 640
10>  )
11> # generic model properties
12> color "black"
13> size [ 0.06 0.15 0.03 ]
14> )
15>
16> define turtlebot position
17> (
18>   pose [ 0.0 0.0 0.0 0.0 ]
19>   size [ 0.2552 0.2552 0.40 ]
20>   origin [ 0.0 0.0 0.0 0.0 ]
21>   gui_nose 1
22>   drive "diff"
23>   color "gray"
24>
25>   kinect(pose [ -0.1 0.0 -0.11 0.0 ])
26> )
```


Appendix E: Functions for heterogeneous robot programs

```
1> # Filename: LibHetRobots.py
2> import math
3>
4> def getRobMatrix():
5>     # get no. of columns (robots) first:
6>     c = 0 # set no. of columns to 0
7>     f = open('robconfig.txt', 'r')
8>     for line in f:
9>         c += 1 # inc no. of columns
10>    f.close()
11>    # now get the contents:
12>    i = -1
13>    r = 11 # no. of rows
14>    RobMatrix = [[0 for x in range(r)] for y in range(c)]
15>    r = 2
16>    RobNames = [[0 for x in range(r)] for y in range(c)]
17>    f = open('robconfig.txt', 'r')
18>    for line in f:
19>        i += 1
20>        startpos = 0 # char start position in string
21>        rrec = []
22>        charpos = line.find(',')
23>        while charpos != -1:
24>            rrec.append(line[startpos:charpos])
25>            startpos = charpos + 1
26>            charpos = line.find(',', startpos, len(line))
27>            RobMatrix[i] = [int(rrec[0]), int(rrec[3]), int(rrec[4]),
28>                           int(rrec[5]), float(rrec[6]),
29>                           float(rrec[7]), float(rrec[8]), int(rrec[9]),
30>                           int(rrec[10]), int(rrec[11]),
31>                           int(rrec[12]), int(rrec[13])]
32>            RobNames[i] = [rrec[0], rrec[1], rrec[2]]
33>        f.close()
34>    return RobMatrix, RobNames
```

```

34>
35> def getTaskMatrix():
36>     # get no. of columns (tasks) first:
37>     c = 0 # set no. of columns to 0
38>     f = open('taskconfig.txt', 'r')
39>     for line in f:
40>         c += 1 # inc no. of columns
41>     f.close()
42>     # now get the contents:
43>     i = -1
44>     r = 12 # no. of rows
45>     TaskMatrix = [[0 for x in range(r)] for y in range(c)]
46>     r = 1
47>     TaskNames = [[0 for x in range(r)] for y in range(c)]
48>     f = open('taskconfig.txt', 'r')
49>     for line in f:
50>         i += 1
51>         startpos = 1 # char start position in string
52>         rrec = []
53>         charpos = line.find(',')
54>         while charpos != -1:
55>             rrec.append(line[startpos:charpos])
56>             startpos = charpos + 1
57>             charpos = line.find(',', startpos, len(line))
58>             TaskMatrix[i] = [int(rrec[0]), int(rrec[2]), int(rrec[3]),
59>                             int(rrec[4]), float(rrec[5]),
60>                             float(rrec[6]), float(rrec[7]),
61>                             int(rrec[8]), int(rrec[9]), int(rrec[10]),
62>                             int(rrec[11]), int(rrec[12]), int(rrec[13])]
63>             TaskNames[i] = [rrec[0], rrec[1]]
64>         f.close()
65>     return TaskMatrix, TaskNames
66> # check if the robot (rid) is compatible with a particular task (tid)
67> def checkRobTaskCompat(rid, tid):
68>     RobMatrix, RobNames = getRobMatrix()
69>     TaskMatrix, TaskNames = getTaskMatrix()
70>     isCompat = False
71>     for i in range(len(RobMatrix)):
72>         if rid == RobMatrix[i][0]:
73>             for j in range(len(TaskMatrix)):
74>                 if tid == TaskMatrix[j][0]:
75>                     if ((RobMatrix[i][1] <= TaskMatrix[j][1]) and

```

```

76>             (RobMatrix[i][2] <= TaskMatrix[j][2]) and
77>             (RobMatrix[i][3] <= TaskMatrix[j][3]) and
78>             (RobMatrix[i][4] >= TaskMatrix[j][4]) and
79>             (RobMatrix[i][5] >= TaskMatrix[j][5]) and
80>             (RobMatrix[i][6] >= TaskMatrix[j][6]) and
81>             ((RobMatrix[i][7]==1 and
TaskMatrix[j][8]==1) or (TaskMatrix[j][8]==0)) and
82>             ((RobMatrix[i][8]==1 and
TaskMatrix[j][9]==1) or (TaskMatrix[j][9]==0)) and
83>             ((RobMatrix[i][9]==1 and
TaskMatrix[j][10]==1) or (TaskMatrix[j][10]==0)) and
84>             ((RobMatrix[i][10]==1 and
TaskMatrix[j][11]==1) or (TaskMatrix[j][11]==0)) and
85>             ((RobMatrix[i][11]==1 and
TaskMatrix[j][12]==1) or (TaskMatrix[j][12]==0)):
86>             isCompat = True
87>
88>     print RobMatrix[0]
89>     print TaskMatrix[0]
90>     print isCompat
91>
92> # get the compatible tasks for a particular robot
93> def getCompatTasks(rid):
94>     RobMatrix, RobNames = getRobMatrix()
95>     TaskMatrix, TaskNames = getTaskMatrix()
96>     c = 0
97>     # get range of the CompatTaskMat:
98>     for i in range(len(RobMatrix)):
99>         if rid == RobMatrix[i][0]:
100>             for j in range(len(TaskMatrix)):
101>                 if ((RobMatrix[i][1] <= TaskMatrix[j][1]) and
102>                     (RobMatrix[i][2] <= TaskMatrix[j][2])
and
103>                     (RobMatrix[i][3] <= TaskMatrix[j][3])
and
104>                     (RobMatrix[i][4] >= TaskMatrix[j][4])
and
105>                     (RobMatrix[i][5] >= TaskMatrix[j][5])
and
106>                     (RobMatrix[i][6] >= TaskMatrix[j][6])
and
107>                     ((RobMatrix[i][7]==1 and
TaskMatrix[j][8]==1) or (TaskMatrix[j][8]==0)) and

```

```

108>                                     ((RobMatrix[i][8]==1 and
TaskMatrix[j][9]==1) or (TaskMatrix[j][9]==0)) and
109>                                     ((RobMatrix[i][9]==1 and
TaskMatrix[j][10]==1) or (TaskMatrix[j][10]==0)) and
110>                                     ((RobMatrix[i][10]==1 and
TaskMatrix[j][11]==1) or (TaskMatrix[j][11]==0)) and
111>                                     ((RobMatrix[i][11]==1 and
TaskMatrix[j][12]==1) or (TaskMatrix[j][12]==0)):
112>                                     c += 1
113>
114>     # Now get data for the CompatTaskMat:
115>     r = 2
116>     CompatTaskMat = [[0 for x in range(r)] for y in range(c)]
117>     idx = 0
118>     for i in range(len(RobMatrix)):
119>         if rid == RobMatrix[i][0]:
120>             for j in range(len(TaskMatrix)):
121>                 if ((RobMatrix[i][1] <= TaskMatrix[j][1]) and
122>                     (RobMatrix[i][2] <= TaskMatrix[j][2])
and
123>                     (RobMatrix[i][3] <= TaskMatrix[j][3])
and
124>                     (RobMatrix[i][4] >= TaskMatrix[j][4])
and
125>                     (RobMatrix[i][5] >= TaskMatrix[j][5])
and
126>                     (RobMatrix[i][6] >= TaskMatrix[j][6])
and
127>                     ((RobMatrix[i][7]==1 and
TaskMatrix[j][8]==1) or (TaskMatrix[j][8]==0)) and
128>                     ((RobMatrix[i][8]==1 and
TaskMatrix[j][9]==1) or (TaskMatrix[j][9]==0)) and
129>                     ((RobMatrix[i][9]==1 and
TaskMatrix[j][10]==1) or (TaskMatrix[j][10]==0)) and
130>                     ((RobMatrix[i][10]==1 and
TaskMatrix[j][11]==1) or (TaskMatrix[j][11]==0)) and
131>                     ((RobMatrix[i][11]==1 and
TaskMatrix[j][12]==1) or (TaskMatrix[j][12]==0))):
132>
133>         CompatTaskMat[idx][0] = TaskNames[j][0]
134>         CompatTaskMat[idx][1] = TaskNames[j][1]
135>         idx += 1
136>
137>     #print RobMatrix[0]
138>     #print TaskMatrix[0]

```

```

139>         return CompatTaskMat
140>
141>     # get the compatible robots for a particular task
142>     def getCompatRobots(tid):
143>         RobMatrix, RobNames = getRobMatrix()
144>         TaskMatrix, TaskNames = getTaskMatrix()
145>         c = 0
146>         # get range of the CompatTaskMat:
147>         for i in range(len(RobMatrix)):
148>             if tid == TaskMatrix[i][0]:
149>                 for j in range(len(RobMatrix)):
150>                     if ((RobMatrix[j][1] <= TaskMatrix[i][1]) and
151>                         (RobMatrix[j][2] <= TaskMatrix[i][2])
152>                         and
153>                         (RobMatrix[j][3] <= TaskMatrix[i][3])
154>                         and
155>                         (RobMatrix[j][4] >= TaskMatrix[i][4])
156>                         and
157>                         (RobMatrix[j][5] >= TaskMatrix[i][5])
158>                         and
159>                         (RobMatrix[j][6] >= TaskMatrix[i][6])
160>                         and
161>                         ((RobMatrix[j][7]==1 and
162>                          TaskMatrix[i][8]==1) or (TaskMatrix[i][8]==0)) and
163>                         ((RobMatrix[j][8]==1 and
164>                          TaskMatrix[i][9]==1) or (TaskMatrix[i][9]==0)) and
165>                         ((RobMatrix[j][9]==1 and
166>                          TaskMatrix[i][10]==1) or (TaskMatrix[i][10]==0)) and
167>                         ((RobMatrix[j][10]==1 and
168>                          TaskMatrix[i][11]==1) or (TaskMatrix[i][11]==0)) and
169>                         ((RobMatrix[j][11]==1 and
170>                          TaskMatrix[i][12]==1) or (TaskMatrix[i][12]==0))):
171>                             c += 1
172>
173>         # Now get data for the CompatTaskMat:
174>         r = 3
175>         CompatRobMat = [[0 for x in range(r)] for y in range(c)]
176>         idx = 0
177>         for i in range(len(RobMatrix)):
178>             if tid == TaskMatrix[i][0]:
179>                 for j in range(len(RobMatrix)):
180>                     if ((RobMatrix[j][1] <= TaskMatrix[i][1]) and
181>                         (RobMatrix[j][2] <= TaskMatrix[i][2])

```


Appendix F: SVM function code

```
1> # Filename: SVM_MatHand.py
2> import numpy as np
3> import random
4> import matplotlib.pyplot as plt
5> from sklearn import svm
6> import tkinter as tk
7> import tkinter.ttk as ttk
8> from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
9> from matplotlib.backends.backend_tkagg import NavigationToolbar2TkAgg
10> from matplotlib.figure import Figure
11>
12> # get data
13> f = open('train.txt', 'r')
14> Y=[]
15> X1=[]
16> for line in f:
17>     fnum = 0 # no. of features
18>     startpos = 0 # char start position in string
19>     charpos = 1
20>     print line
21>     Y.append(int(line[startpos:charpos]))
22>     startpos = 2
23>     charpos = line.find(',')
24>     while charpos != -1:
25>         fnum += 1
26>         X1.append(int(line[startpos:charpos]))
27>         startpos = charpos + 1
28>         charpos = line.find(',', startpos, len(line))
29> f.close()
30> X=np.array(X1).reshape(len(Y), fnum)
31>
32> # SVM computation
33> clf = svm.SVC(kernel='linear', C=1)
34> clf.fit(X, Y)
35> print(clf.predict([[20,10,30,5,5,0]]))
36> #print(X)
37>
38> class Application(tk.Frame):
```

```

39>     def __init__(self, master=None):
40>         tk.Frame.__init__(self, master)
41>         self.createWidgets()
42>     def createWidgets(self):
43>         fig=plt.figure(figsize=(4,4))
44>         ax=fig.add_axes([0.1,0.1,0.8,0.8])
45>         canvas=FigureCanvasTkAgg(fig, master=root)
46>         canvas.get_tk_widget().grid(row=0, column=0, columnspan=20)
47>         canvas.show()
48>
49>         self.plotbutton=ttk.Button(master=root, text="Assign to T1",
    command=lambda: self.plot(canvas, ax, 0))
50>         self.plotbutton.grid(row=1, column=0)
51>         self.plotbutton2=ttk.Button(master=root, text="Assign to T2",
    command=lambda: self.plot(canvas, ax, 1))
52>         self.plotbutton2.grid(row=1, column=1)
53>
54>     def plot(self, canvas, ax, y):
55>         data = (random.randint(0, 100), random.randint(0, 100),
    random.randint(0, 100), random.randint(0, 100), random.randint(0,
    100), random.randint(0, 100))
56>         print(data)
57>         ind = np.arange(6) # the x locations for the groups
58>         width = .3
59>         rects1 = ax.bar(ind, data, width)
60>         x = range(0, 6)
61>         ax.set_xticks(x)
62>         ax.set_yticks([0, 50, 100])
63>         labels = ['T1-IN', 'T1-OUT', 'T1-D', 'T2-IN', 'T2-OUT', 'T2-D']
64>         ax.set_xticklabels(labels, fontsize='small')
65>         canvas.draw()
66>         ax.clear()
67>         # write data to svm train file:
68>         f = open('train.txt', 'a')
69>         line = '%s:%s,%s,%s,%s,%s,%s,\n' %
    (y, data[0], data[1], data[2], data[3], data[4], data[5])
70>         f.writelines(line)
71>         f.close()
72>         print line
73>
74> root=tk.Tk()
75> root.title("SVM Train Application")
76> app=Application(master=root)
77> app.mainloop()

```


Appendix G: Python application code for Study I

G1. Algorithm and mobile robot navigation program

```
1> # Filename: matHand_algo.py
2> import OpenOPC # the OPC client library for Python
3> import gvar
4> import matHand_sim
5> import math
6> import time
7>
8>
9> def mapVars():
10>     f = open('avarmap.txt', 'r')
11>     global locArr, opcArr
12>     locArr = []
13>     opcArr = []
14>     for line in f:
15>         startpos = 0 # char start position in string
16>         charpos = line.find(':')
17>         locArr += [line[startpos:charpos]]
18>         opcArr += [line[charpos+1:len(line)-1]]
19>     f.close()
20>     return 0
21>
22>
23> # Update the local variables with values from OPC mapped tags (wrt
    avarmap.txt)
24> # Call this function regularly!
25> def updateVars():
26>     global locArr, opcArr
27>     global opc
28>     opc = OpenOPC.open_client('localhost')
29>     opc.connect('Matrikon.OPC.Simulation')
30>     try:
31>         for i in range(len(locArr)):
32>             opcval = opc.read(tags=opcArr[i], timeout=500) # read
    OPC tag values
33>             setattr(gvar, locArr[i], opcval[0]) # assign OPC tag
    values to mapped local variables
```

```
34>         runTaskAlgo()
35>         robMotion()
36>     except OpenOPC.TimeoutError:
37>         print "ALGO TimeoutError occured"
38>     opc.close()
39>     return 0
40>
41>
42> def initVars():
43>     global T1count, T2count
44>     T1count = 0
45>     T2count = 0
46>
47>     gvar.l_taskID = 0
48>     gvar.l_t01_weight = 0.0
49>     gvar.l_t02_weight = 0.0
50>     gvar.l_t01_x1 = 0.0
51>     gvar.l_t01_x2 = 0.0
52>     gvar.l_t01_x3 = 0.0
53>     gvar.l_t02_x1 = 0.0
54>     gvar.l_t02_x2 = 0.0
55>     gvar.l_t02_x3 = 0.0
56>
57>     gvar.l_t01_iBL = 0
58>     gvar.l_t01_iFr = 0.0
59>     gvar.l_t01_iBC = 100
60>     gvar.l_t01_qSL = 0
61>     gvar.l_t01_qFr = 0.0
62>     gvar.l_t01_sCDr = 0.0
63>     gvar.l_t01_sPPi = 0.0
64>     gvar.l_t01_sPD = 0
65>
66>     gvar.l_t02_iBL = 0
67>     gvar.l_t02_iFr = 0.0
68>     gvar.l_t02_iBC = 100
69>     gvar.l_t02_qSL = 0
70>     gvar.l_t02_qFr = 0.0
71>     gvar.l_t02_sCDr = 0.0
72>     gvar.l_t02_sPPi = 0.0
73>     gvar.l_t02_sPD = 0
74>
75>     gvar.l_t01_src_x = 0.0
76>     gvar.l_t01_dest_x = 0.0
77>     gvar.l_t01_src_y = 0.0
```

```
78>     gvar.l_t01_dest_y = 0.0
79>     gvar.l_t02_src_x = 0.0
80>     gvar.l_t02_dest_x = 0.0
81>     gvar.l_t02_src_y = 0.0
82>     gvar.l_t02_dest_y = 0.0
83>
84>
85> def runTaskAlgo():
86>     global opc
87>     try:
88>         gvar.l_t01_x1 = 1/(gvar.l_t01_iFr * (float(gvar.l_t01_iBL) /
            float(gvar.l_t01_iBC)))
89>     except:
90>         gvar.l_t01_x1 = 0
91>     gvar.l_t01_x2 = 1/gvar.l_t01_sPPi
92>     gvar.l_t01_x3 = (gvar.l_t01_qFr - gvar.l_t01_sCDr) +
            ((gvar.l_t01_qSL - gvar.l_t01_sPD) * 0.2) + 0.1
93>     gvar.l_t01_weight = gvar.l_t01_x1 * gvar.l_t01_x2 * gvar.l_t01_x3
94>     print(gvar.l_t01_x1, gvar.l_t01_x2, gvar.l_t01_x3,
            gvar.l_t01_weight)
95>
96>     try:
97>         gvar.l_t02_x1 = 1/(gvar.l_t02_iFr * (float(gvar.l_t02_iBL) /
            float(gvar.l_t02_iBC)))
98>     except:
99>         gvar.l_t02_x1 = 0
100>         gvar.l_t02_x2 = 1/gvar.l_t02_sPPi
101>         gvar.l_t02_x3 = (gvar.l_t02_qFr - gvar.l_t02_sCDr) +
            ((gvar.l_t02_qSL - gvar.l_t02_sPD) * 0.2) + 0.1
102>         gvar.l_t02_weight = gvar.l_t02_x1 * gvar.l_t02_x2 *
            gvar.l_t02_x3
103>         print(gvar.l_t02_x1, gvar.l_t02_x2, gvar.l_t02_x3,
            gvar.l_t02_weight)
104>
105>         if gvar.l_t01_weight <= gvar.l_t02_weight:
106>             gvar.l_taskID = 1
107>         else:
108>             gvar.l_taskID = 2
109>         return gvar.l_taskID
110>
111>
112>     # Function that handles the robot motion commands
113>     def robMotion():
114>         global opc, T1count, T2count
```

```

115>         # compute distance from goal:
116>         dx = gvar.l_r01_xgoal - gvar.l_r01_xpos
117>         dy = gvar.l_r01_ygoal - gvar.l_r01_ypos
118>         dist = math.sqrt(math.pow(dx, 2) + math.pow(dy, 2))
119>         # move robot:
120>         if (dist <= 0.3) and (gvar.l_r01_status == 0): # robot-01
    is available. Note: robot must clear its own status!
121>             # TASK-1 code:
122>             if (gvar.l_taskID == 1) and (gvar.l_r01_currloc == 0):
    # robot is waiting for command
123>                 resp = opc.write(('Server02.r01_xgoal',
    gvar.l_t01_src_x))
124>                 resp2 = opc.write(('Server02.r01_ygoal',
    gvar.l_t01_src_y))
125>                 if (resp == "Success") and (resp2 == "Success"):
126>                     opc.write(('Server02.r01_currloc', 1)) #
    robot-01 is on its way to the source
127>                     opc.write(('Server02.r01_tcurr', 1)) # robot-
    01 current task goal is #1
128>                     opc.write(('Server02.r01_status', 1)) # robot-
    01 set to busy
129>                     gvar.l_r01_status = 1
130>                     T1count += 1
131>                     print "T1count: ", T1count
132>
133>                 if (gvar.l_r01_tcurr == 1) and ((gvar.l_r01_currloc ==
    1) or (gvar.l_r01_currloc == 2)): # robot is waiting at T#1 source
134>                     opc.write(('Server02.r01_currloc', 2)) # robot-01
    is at the source
135>                 if (gvar.l_r01_loadstat == 1):
136>                     resp = opc.write(('Server02.r01_xgoal',
    gvar.l_t01_dest_x))
137>                     resp2 = opc.write(('Server02.r01_ygoal',
    gvar.l_t01_dest_y))
138>                     if (resp == "Success") and (resp2 ==
    "Success"):
139>                         opc.write(('Server02.r01_currloc', 3)) #
    robot-01 is on its way to the destination
140>                         opc.write(('Server02.r01_status', 1)) #
    robot-01 set to busy
141>                         gvar.l_r01_status = 1
142>                         opc.write(('Server02.r01_loadstat', 0)) #
    clear load status
143>

```

```
144>         if (gvar.l_r01_tcurr == 1) and ((gvar.l_r01_currloc ==
           3) or (gvar.l_r01_currloc == 4)): # robot is waiting at T#1
           destination
145>         opc.write(('Server02.r01_currloc', 4)) # robot-01
           is at the destination
146>         if (gvar.l_r01_loadstat == 1):
147>             opc.write(('Server02.r01_currloc', 0)) # robot
           has no goal location
148>             # opc.write(('Server02.r01_tcurr', 0)) # robot
           is released from task
149>             opc.write(('Server02.r01_loadstat', 0)) #
           clear load status
150>
151>         # TASK-2 code:
152>         if (gvar.l_taskID == 2) and (gvar.l_r01_currloc == 0):
           #robot is waiting for command
153>             resp = opc.write(('Server02.r01_xgoal',
           gvar.l_t02_src_x))
154>             resp2 = opc.write(('Server02.r01_ygoal',
           gvar.l_t02_src_y))
155>             if (resp == "Success") and (resp2 == "Success"):
156>                 opc.write(('Server02.r01_currloc', 1)) #
           robot-01 is on its way to the source
157>                 opc.write(('Server02.r01_tcurr', 2)) # robot-
           01 current task goal is #2
158>                 opc.write(('Server02.r01_status', 1)) # robot-
           01 set to busy
159>                 gvar.l_r01_status = 1
160>                 T2count += 1
161>                 print "T2count: ", T2count
162>
163>         if (gvar.l_r01_tcurr == 2) and ((gvar.l_r01_currloc ==
           1) or (gvar.l_r01_currloc == 2)): # robot is waiting at T#2 source
164>             opc.write(('Server02.r01_currloc', 2)) # robot-01
           is at the source
165>             if (gvar.l_r01_loadstat == 1):
166>                 resp = opc.write(('Server02.r01_xgoal',
           gvar.l_t02_dest_x))
167>                 resp2 = opc.write(('Server02.r01_ygoal',
           gvar.l_t02_dest_y))
168>                 if (resp == "Success") and (resp2 ==
           "Success"):
169>                     opc.write(('Server02.r01_currloc', 3)) #
           robot-01 is on its way to the destination
```

```

170>             opc.write(('Server02.r01_status', 1)) #
            robot-01 set to busy
171>             gvar.l_r01_status = 1
172>             opc.write(('Server02.r01_loadstat', 0)) #
            clear load status
173>
174>             if (gvar.l_r01_tcurr == 2) and ((gvar.l_r01_currloc ==
            3) or (gvar.l_r01_currloc == 4)): # robot is waiting at T#2
            destination
175>             opc.write(('Server02.r01_currloc', 4)) # robot-01
            is at the destination
176>             if (gvar.l_r01_loadstat == 1):
177>                 opc.write(('Server02.r01_currloc', 0)) # robot
            has no goal location
178>                 # opc.write(('Server02.r01_tcurr', 0)) # robot
            is released from task
179>             opc.write(('Server02.r01_loadstat', 0)) #
            clear load status
180>
181>             return 0
182>
183>
184>     if __name__ == "__main__":
185>         matHand_sim.initVars()
186>         matHand_sim.initTags()
187>         initVars()
188>         mapVars()
189>         while 1:
190>             matHand_sim.updateTims()
191>             matHand_sim.readTags()
192>             updateVars()
193>             matHand_sim.recordData(T1count, T2count,
            gvar.l_t01_weight, gvar.l_t02_weight)

```

G2. Simulation program

```

1> # Filename: matHand_sim.py
2> # simulate the buffer levels, consumer demand rates etc.
3> import OpenOPC # the OPC client library for Python
4> import time
5> from datetime import datetime
6>
7>

```

```
8> def initVars():
9>     global start
10>     global t01_iB_time_count, t02_iB_time_count, t01_qB_time_count,
    t02_qB_time_count, sim_time_count
11>     global start_t01_iB_time, start_t02_iB_time, start_t01_qB_time,
    start_t02_qB_time, start_sim_time
12>     global XYdatestr, LEVdatestr, TSKdatestr
13>     start = time.time()
14>     t01_iB_time_count = 0
15>     t02_iB_time_count = 0
16>     t01_qB_time_count = 0
17>     t02_qB_time_count = 0
18>     sim_time_count = 0
19>     start_t01_iB_time = time.time()
20>     start_t02_iB_time = time.time()
21>     start_t01_qB_time = time.time()
22>     start_t02_qB_time = time.time()
23>     start_sim_time = time.time()
24>
25>     XYdatestr = "XY_" + datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
26>     LEVdatestr = "LEV_" + datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
27>     TSKdatestr = "TSK_" + datetime.now().strftime('%Y-%m-%d_%H-%M-
    %S')
28>
29>
30> def initTags():
31>     global opc
32>     opc = OpenOPC.open_client('localhost')
33>     opc.connect('Matrikon.OPC.Simulation')
34>     opc.write(('Server02.t01_iBM', 0))
35>     opc.write(('Server02.t01_iBL', 0))
36>     opc.write(('Server02.t01_iBC', 100))
37>     opc.write(('Server02.t02_iBM', 0))
38>     opc.write(('Server02.t02_iBL', 0))
39>     opc.write(('Server02.t02_iBC', 100))
40>     opc.write(('Server02.t01_iFr', 30))
41>     opc.write(('Server02.t02_iFr', 30))
42>     opc.write(('Server02.t01_qFr', 0))
43>     opc.write(('Server02.t02_qFr', 0))
44>     opc.write(('Server02.t01_sPPi', 0.5))
45>     opc.write(('Server02.t02_sPPi', 0.5))
46>     opc.write(('Server02.t01_qSC', 100))
47>     opc.write(('Server02.t02_qSC', 100))
48>     opc.write(('Server02.t01_qSM', 0))
```

```
49>     opc.write(('Server02.t01_qSL', 0))
50>     opc.write(('Server02.t02_qSM', 0))
51>     opc.write(('Server02.t02_qSL', 0))
52>     opc.write(('Server02.t01_sCDr', 0))
53>     opc.write(('Server02.t02_sCDr', 5))
54>     # x-y source and destination locations:
55>     opc.write(('Server02.t01_src_x', 3.5))
56>     opc.write(('Server02.t01_src_y', 7.5))
57>     opc.write(('Server02.t01_dest_x', 1.8))
58>     opc.write(('Server02.t01_dest_y', 2.5))
59>     opc.write(('Server02.t02_src_x', 18.5))
60>     opc.write(('Server02.t02_src_y', 7.5))
61>     opc.write(('Server02.t02_dest_x', 20.1))
62>     opc.write(('Server02.t02_dest_y', 2.4))
63>     # robot tags:
64>     opc.write(('Server02.r01_loadcap', 20))
65>     opc.write(('Server02.r01_currloc', 0))
66>     opc.write(('Server02.r01_tcurr', 0))
67>     opc.write(('Server02.r01_loadstat', 0))
68>     opc.write(('Server02.r01_status', 0))
69>     opc.close()
70>
71>
72> def updateTims():
73>     global t01_iB_time_count, t02_iB_time_count, t01_qB_time_count,
74>         t02_qB_time_count, sim_time_count
75>     global start_t01_iB_time, start_t02_iB_time, start_t01_qB_time,
76>         start_t02_qB_time, start_sim_time
77>     t01_iB_time_count = int((time.time() - start_t01_iB_time) * 1000)
78>     t02_iB_time_count = int((time.time() - start_t02_iB_time) * 1000)
79>     t01_qB_time_count = int((time.time() - start_t01_qB_time) * 1000)
80>     t02_qB_time_count = int((time.time() - start_t02_qB_time) * 1000)
81>     sim_time_count = int((time.time() - start_sim_time) * 1000)
82>
83> def readTags():
84>     global opc
85>     global l_t01_iBM, l_t01_iBL, l_t01_iBC, l_t01_iFr
86>     global l_t02_iBM, l_t02_iBL, l_t02_iBC, l_t02_iFr
87>     global l_t01_qSM, l_t01_qSL, l_t01_qSC, l_t01_qFr
88>     global l_t02_qSM, l_t02_qSL, l_t02_qSC, l_t02_qFr
89>     global l_r01_tcurr, l_r01_currloc, l_r01_prevloc, l_r01_load,
90>         l_r01_loadcap
91>     tout = 500
```



```
90>     opc = OpenOPC.open_client('localhost')
91>     opc.connect('Matrikon.OPC.Simulation')
92>     try:
93>         l_t01_iBM = opc.read('Server02.t01_iBM', timeout=tout)[0]
94>         l_t01_iBL = opc.read('Server02.t01_iBL', timeout=tout)[0]
95>         l_t01_iBC = opc.read('Server02.t01_iBC', timeout=tout)[0]
96>         l_t01_iFr = opc.read('Server02.t01_iFr', timeout=tout)[0]
97>
98>         l_t02_iBM = opc.read('Server02.t02_iBM', timeout=tout)[0]
99>         l_t02_iBL = opc.read('Server02.t02_iBL', timeout=tout)[0]
100>         l_t02_iBC = opc.read('Server02.t02_iBC',
    timeout=tout)[0]
101>         l_t02_iFr = opc.read('Server02.t02_iFr',
    timeout=tout)[0]
102>
103>         l_t01_qSM = opc.read('Server02.t01_qSM',
    timeout=tout)[0]
104>         l_t01_qSL = opc.read('Server02.t01_qSL',
    timeout=tout)[0]
105>         l_t01_qSC = opc.read('Server02.t01_qSC',
    timeout=tout)[0]
106>         l_t01_qFr = opc.read('Server02.t01_qFr',
    timeout=tout)[0]
107>
108>         l_t02_qSM = opc.read('Server02.t02_qSM',
    timeout=tout)[0]
109>         l_t02_qSL = opc.read('Server02.t02_qSL',
    timeout=tout)[0]
110>         l_t02_qSC = opc.read('Server02.t02_qSC',
    timeout=tout)[0]
111>         l_t02_qFr = opc.read('Server02.t02_qFr',
    timeout=tout)[0]
112>
113>         l_r01_tcurr = opc.read('Server02.r01_tcurr',
    timeout=tout)[0]
114>         l_r01_currloc = opc.read('Server02.r01_currloc',
    timeout=tout)[0]
115>         l_r01_prevloc = opc.read('Server02.r01_prevloc',
    timeout=tout)[0]
116>         l_r01_load = opc.read('Server02.r01_load',
    timeout=tout)[0]
117>         l_r01_loadcap = opc.read('Server02.r01_loadcap',
    timeout=tout)[0]
118>
119>         inTask1()
```

```
120>         inTask2()
121>         outTask1()
122>         outTask2()
123>
124>     except OpenOPC.TimeoutError:
125>         print "SIM TimeoutError occured"
126>
127>     opc.close()
128>
129>
130>     def inTask1():
131>         global opc
132>         global start_t01_iB_time
133>         global l_t01_iBM, l_t01_iBL, l_t01_iBC, l_t01_iFr
134>         global l_r01_tcurr, l_r01_currloc, l_r01_prevloc,
135>         l_r01_load, l_r01_loadcap
136>         # input buffer:
137>         if l_t01_iBL < 100:
138>             if t01_iB_time_count >=
139>                 int(60.0/float(l_t01_iFr))*1000:
140>                 start_t01_iB_time = time.time()
141>                 opc.write(('Server02.t01_iBM', l_t01_iBM + 1)) #
142>                 increment input buffer
143>                 l_t01_iBL = int((float(l_t01_iBM) /
144>                 float(l_t01_iBC)) * 100)
145>                 opc.write(('Server02.t01_iBL', l_t01_iBL))
146>                 if (l_r01_tcurr == 1) and (l_r01_currloc == 2): # robot at
147>                 source
148>                     if l_t01_iBM > l_r01_loadcap:
149>                         opc.write(('Server02.r01_load', l_r01_loadcap))
150>                         opc.write(('Server02.t01_iBM', l_t01_iBM -
151>                         l_r01_loadcap))
152>                     else:
153>                         opc.write(('Server02.r01_load', l_t01_iBM))
154>                         opc.write(('Server02.t01_iBM', 0))
155>                         opc.write(('Server02.r01_loadstat', 1)) # done loading
156>                         # compute percentage:
157>                         l_t01_iBL = int((float(l_t01_iBM) / float(l_t01_iBC)) *
158>                         100)
159>                         opc.write(('Server02.t01_iBL', l_t01_iBL))
160>
161>
162>     def inTask2():
163>         global opc
```

```

157>         global start_t02_iB_time
158>         global l_t02_iBM, l_t02_iBL, l_t02_iBC, l_t02_iFr
159>         global l_r01_tcurr, l_r01_currloc, l_r01_prevloc,
           l_r01_load, l_r01_loadcap
160>
161>         # input buffer:
162>         if l_t02_iBL < 100:
163>             if t02_iB_time_count >=
           int(60.0/float(l_t02_iFr))*1000:
164>                 start_t02_iB_time = time.time()
165>                 opc.write(('Server02.t02_iBM', l_t02_iBM + 1)) #
           increment input buffer
166>                 l_t02_iBL = int((float(l_t02_iBM) /
           float(l_t02_iBC)) * 100)
167>                 opc.write(('Server02.t02_iBL', l_t02_iBL))
168>             if (l_r01_tcurr == 2) and (l_r01_currloc == 2): # robot at
           source
169>                 if l_t02_iBM > l_r01_loadcap:
170>                     opc.write(('Server02.r01_load', l_r01_loadcap))
171>                     opc.write(('Server02.t02_iBM', l_t02_iBM -
           l_r01_loadcap))
172>                 else:
173>                     opc.write(('Server02.r01_load', l_t02_iBM))
174>                     opc.write(('Server02.t02_iBM', 0))
175>                     opc.write(('Server02.r01_loadstat', 1)) # done loading
176>                     # compute percentage:
177>                     l_t02_iBL = int((float(l_t02_iBM) / float(l_t02_iBC)) *
           100)
178>                     opc.write(('Server02.t02_iBL', l_t02_iBL))
179>
180>
181>     def outTask1():
182>         global opc
183>         global sim_time_count
184>         global l_t01_qSM, l_t01_qSL, l_t01_qSC, l_t01_qFr
185>         global l_r01_tcurr, l_r01_currloc, l_r01_prevloc,
           l_r01_load, l_r01_loadcap
186>
187>         # output buffer:
188>         l_t01_qSL = int((float(l_t01_qSM) / float(l_t01_qSC)) *
           100)
189>         opc.write(('Server02.t01_qSL', l_t01_qSL))
190>         l_t01_qFr = (60000.0 / sim_time_count) * float(l_t01_qSL)
191>         opc.write(('Server02.t01_qFr', l_t01_qFr))

```

```

192>         if l_t01_qSL < 100:
193>             if (l_r01_tcurr == 1) and (l_r01_currloc == 4): #
robot at destination
194>                 opc.write(('Server02.t01_qSM', l_t01_qSM +
l_r01_load))
195>                 opc.write(('Server02.r01_load', 0))
196>                 opc.write(('Server02.r01_loadstat', 1)) # done
offloading
197>                 # compute percentage and fill rate:
198>                 l_t01_qSL = int((float(l_t01_qSM) /
float(l_t01_qSC)) * 100)
199>                 opc.write(('Server02.t01_qSL', l_t01_qSL))
200>                 l_t01_qFr =
(60000.0/sim_time_count)*float(l_t01_qSL)
201>                 opc.write(('Server02.t01_qFr', l_t01_qFr))
202>
203>
204>     def outTask2():
205>         global opc
206>         global sim_time_count
207>         global l_t02_qSM, l_t02_qSL, l_t02_qSC, l_t02_qFr
208>         global l_r01_tcurr, l_r01_currloc, l_r01_prevloc,
l_r01_load, l_r01_loadcap
209>
210>         # output buffer:
211>         l_t02_qSL = int((float(l_t02_qSM) / float(l_t02_qSC)) *
100)
212>         opc.write(('Server02.t02_qSL', l_t02_qSL))
213>         l_t02_qFr = (60000.0 / sim_time_count) * float(l_t02_qSL)
214>         opc.write(('Server02.t02_qFr', l_t02_qFr))
215>         if l_t02_qSL < 100:
216>             if (l_r01_tcurr == 2) and (l_r01_currloc == 4): #
robot at destination
217>                 opc.write(('Server02.t02_qSM', l_t02_qSM +
l_r01_load))
218>                 opc.write(('Server02.r01_load', 0))
219>                 opc.write(('Server02.r01_loadstat', 1)) # done
offloading
220>                 # compute percentage and fill rate:
221>                 l_t02_qSL = int((float(l_t02_qSM) /
float(l_t02_qSC)) * 100)
222>                 opc.write(('Server02.t02_qSL', l_t02_qSL))
223>                 l_t02_qFr = (60000.0 / sim_time_count) *
float(l_t02_qSL)

```

```
224>         opc.write(('Server02.t02_qFr', l_t02_qFr))
225>
226>
227>     # Store the XY robot motion data and the input/output levels
228>     def recordData(T1cnt, T2cnt, T1wt, T2wt):
229>         global XYdatestr, LEVdatestr, TSKdatestr
230>         global l_t01_iBL, l_t02_iBL, l_t01_qSL, l_t02_qSL
231>         global opc
232>         global sim_time_count
233>
234>         opc = OpenOPC.open_client('localhost')
235>         opc.connect('Matrikon.OPC.Simulation')
236>
237>         try:
238>             xloc = opc.read('Server02.r01_xpos', timeout=500)[0]
239>             yloc = opc.read('Server02.r01_ypos', timeout=500)[0]
240>             f = open('%s.csv' % XYdatestr, 'a')
241>             f.write('%f;%s;%s\n' % (sim_time_count/1000, xloc,
yloc))
242>             f.close()
243>             f = open('%s.csv' % LEVdatestr, 'a')
244>             f.write('%f;%s;%s;%s;%s;%f;%f\n' %
(sim_time_count/1000, l_t01_iBL, l_t02_iBL, l_t01_qSL, l_t02_qSL,
T1wt, T2wt))
245>             f.close()
246>             f = open('%s.csv' % TSKdatestr, 'a')
247>             f.write('%f;%i;%i;%i\n' % (sim_time_count/1000, T1cnt,
T2cnt, T1cnt+T2cnt))
248>             f.close()
249>         except OpenOPC.TimeoutError:
250>             print "recData TimeoutError occured"
251>         opc.close()
252>
253>
254>     if __name__ == "__main__":
255>         initVars()
256>         initTags()
257>         while 1:
258>             updateTims()
259>             readTags()
260>             time.sleep(1)
```