On Decentralizing Intelligence in Cyber-Physical Systems

Thesis submitted in partial fulfilment of the requirements
for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Tushar Semwal

Under the supervision of

Prof. Shivashankar B. Nair

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati - 781039 Assam India

November, 2018

Dedicated to

My beloved Parents,

My adorable Sister

and

My caring Elder Brothers

Acknowledgements

Foremost, I would like to express my unfeigned gratitude to my supervisor *Prof. Shivashankar B. Nair* for his unwavering support, inexhaustible patience and positive guidance during my doctoral research. I am thankful for his ethical beliefs and philosophy which made me mature not only as a scientific researcher but also as a human.

I am immensely obliged to *Prof. Pradip K. Das* for his constant support and motivation. I would also like to thank the rest of my thesis doctoral committee members - *Dr. Pinaki Mitra* and *Prof. Chitralekha Mahanta* for their insightful comments and suggestions which made me improve the quality and clarity of my work.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IITG - Prof. Diganta Goswami and Prof. S. V. Rao for allowing me to use the facilities and the available resources including the travel support for the conferences. I acknowledge the Technical staff of the Department of Computer Science and Engineering - Mr. Nanu Alan Kachari, Mr. Hemanta Kumar Nath, Mr. Bhriguraj Borah, Mr. Pranjitt Talukdar, Mr. Raktajit Pathak and Mr. Nava Kumar Boro for solving any engineering related issues. I am deeply thankful to - Mr. Monojit Bhattacharjee, Ms. Gauri Khuttiya Deori and Mr. Prabin Bharali for efficiently handling the administrative work. I am obliged to all the faculty members, the staff and security personnel for their constant help and support.

I am grateful to the Tata Consultancy Services (TCS), MHRD, Govt. of India, Science and Engineering Research Board and Microsoft Research Lab India for providing me the financial support and resources during the course of my Ph.D. I am thankful to Mr. Sachin Parkhi, the then Program Manager, TCS Research Scholar Program for his prompt help and quick solutions. I would also like to mention Mr. Rijumoni Dutta from academic affairs and Mrs. Kajal Lata Brahma, Mr. Sunil Sarma and Mr. Raj Kamal Sarmah from F&A section, IITG who have always

welcomed me regarding help for all my claims and related issues.

I would like to specially mention Dr. Shashi Shekhar Jha for his continuous guidance from the very beginning of my Ph.D. life. I am thankful to him to set a standard for me which became my source of motivation. I would also like to mention my seniors - Dr. Satish, Dr. Sibaji and Awnish for being my mentor-cum-friends and creating indelible moments at IITG. I am indeed thankful to my fellow lab mates at the Robotics Lab. - Mohit, Sonia, Abhay, Manoj, Nikhil, Rahul, and Divya for creating a wonderful experience at my workplace. The stimulating discussions, brainstorming and sleepless night working together contribute a significant portion towards my development as an independent researcher.

I am fortunate to have good friends - Ashish, Rakesh, Vishal, Sunil, Prateek and Mayank with whom I have shared some ineffaceable moments of my life at IITG. I want to thank Sonia for being my constant companion in the ups and downs of my life. I am thankful to all my colleagues and friends during my journey as a Ph.D. scholar.

I want to thank the friends from my undergraduate and school days - Sanyam, Suresh, Vishal, Manish, Aneesh, Aditya, Devesh, Madhur and Rohit for the beautiful memories and cherished moments spent with them. I am happy to mention my childhood friends and cousins - Naveen, Praveen, Alka, Archana, Arpana, Shobhna, Nidhi, Sachin, Vipin, Mayank, Rahul, Shubham and Shahzad for their encouragement and support.

Finally yet importantly, I would like to thank Almighty God and my family - Mom, Dad, my elder brothers Hemant and Naveen, my sisters-in-law Ekta and Priya, my little sister Taniya and my dearest nephews and niece Nonu, Miku and Aanu for their boundless love, support, caring, warmth and encouragement all these years. I am truly indebted to them.

November 29, 2018 Tushar Semwal

Declaration

I certify that

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor(s).
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor(s) are not in a position to check for any possible instance of plagiarism within this submitted work.

November 29, 2018 Tushar Semwal

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Dr. Shivashankar B. Nair
Professor
Email : sbnair@iitg.ac.in
Phone : +91-361-258-2356

Certificate

This is to certify that this thesis entitled "**On Decentralizing Intelligence in Cyber-Physical Systems**" submitted by **Tushar Semwal**, in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy, to the Indian Institute of Technology Guwahati, Assam, India, is a record of the bonafide research work carried out by him under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam, India. To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date: November 29, 2018
Place: Guwahati

Prof. Shivashankar B. Nair
(Thesis Supervisor)

Abstract

Advancements in the low-cost computing and communication technologies have led to the mass proliferation of devices connected over a network. The networked devices have engendered a new era wherein they sense, compute and share information thereby forming loosely coupled Cyber-Physical Systems (CPS). Managing data and making intelligent decisions form the major domain of research in a CPS. Cloud-based centralized computation has always been the mainstream architecture due to its ease of implementation and enhanced control. However, data explosion, scalability and privacy issues, are certainly pointing toward the limits of such centralized systems. Decentralizing control and distributing the computing among the devices could be a better alternative for sharing intelligence. Investigating new decentralization mechanisms, thus, forms the major crux of this thesis. Realizing such *decentralized* Cyber-Physical Systems (*d*CPS) is fraught with challenges such as choosing the appropriate communication method, incorporating the right learning and knowledge sharing schemes, ensuring robustness and adaptivity, and the need for a proper middleware to cater to its functioning.

This thesis takes a bottom-up approach and presents its first contribution on extending the functionalities of the *Tartarus*, a multi-agent platform, in order to realize complex *d*CPSs. This section begins with the motivation behind *Tartarus* and discusses features which makes it a disparate environment for developing and deploying *d*CPS. A real-world CPS application comprising robots, a Raspberry Pi with a camera and a human administrator in-the-loop, described herein validates the feasibility of *Tartarus* for developing mechanisms to embed decentralized intelligence in CPSs.

The second contribution discusses an approach to transform a centralized system into a fairly distributed system with partial or no centralized control. The technique uses mobile agents to provide local sharing and in-network processing of data. A Location-Aware and Tracking Service (LATS) as a real-world *d*CPS

based application is used to portray the viability of the proposed approach. The technique allows users to embed queries, such as *where* and *when* about a specific person, into a mobile agent and get the results back to the user. A comparison of the proposed approach with its centralized version proved the former to be more efficient in terms of bandwidth utilization and energy efficiency thus, proving its applicability in practical scenarios.

Cooperative tasks involving autonomous entities such as robots are prone to the classical problem of Mutual Exclusion of shared Resources (MER). The next contribution presents a novel mechanism for ordering the task execution to mitigate the issue of MER in a *d*CPS of multiple robots. While the mobile agents achieve the computation, communication, and control, the physical execution of the tasks is performed by the robots in an asynchronous and pipelined manner without the use of a global clock. Unlike atomic commands which are bound to finish in a fixed period, the pipeline formed using robots as processing units is adaptive to the varying execution times experienced in real world robotic tasks. This makes the system flexible and versatile to the dynamics of the environment of the *d*CPS. The proposed mechanism provides unique features such as addition and deletion of both tasks and robots, on-the-fly. Mobile agents carrying the code or solution deliver the same to a robot to make it execute a task. Experiments conducted in an emulation environment validate the presented characteristics of the proposed mechanism. In addition, an application comprising ordering of jobs in a Warehouse Management System using real robots substantiates the feasibility of the approach in real-world decentralized and distributed systems.

The previous contribution assumed that the solutions carried by the mobile agents are optimal. However, in real systems, there could be several solutions to the same problem. The problem of selecting the best solutions, in a decentralized manner, for a given set of problems distributed across a network of nodes, motivates the next contribution. The proposed mechanism takes a non-conventional route and is inspired by the computational models derived from three immunological theories viz. clonal selection, danger theory, and the immune network theory. Whenever a problem occurs at a node, it releases danger signals to attract the concerned mobile agents carrying mappings for potential solutions. The agents that meet at the

distressed node stimulate and suppress each other thereby forming an immune network. The mobile agents carrying mappings of solutions with superior performance are rewarded and thus clone and grow in population while the remaining ones are penalized and thus diminish. The challenge is to find a generic mapping which can cater to the maximum number of similar problems. Extensive experiments conducted in the emulation environment created using the *Tartarus* platform validates the efficacy of the proposed approach. Besides, the results obtained by embodying the mechanisms in robots that discover the best path-following algorithm, substantiate their working in real-world scenarios.

With a selection mechanism in place, the next obvious step is to evolve the solutions to meet the demands of a varying stream of problems. The previous contribution assumed an infinite supply of solutions which is not entirely realistic when it comes to practical scenarios. In this contribution, the solutions, in the form of robot controllers, are thus evolved in an online and continuous fashion. Based on a single parameter, viz. the cross-reactivity threshold, the proposed approach divides the main task into subtasks and evolves separate sub-controllers for each of them. The better evolved solutions are rewarded and shared with other robots using mobile agents. A comparison of mobile agent-based communication with traditional broadcasting method portrays the former to be more energy efficient than the latter.

All the emulated experiments were conducted using the *Tartarus* platform. As an addendum, a brief review of the immunological terms and concepts employed in this thesis is also presented. Finally, the thesis concludes by summing up the contributions and providing avenues for future research and possible applications.

Contents

List of Figures

List of Algorithms

List of Tables

List of Symbols

Symbols	Description
ϵ	Cross Reactivity Threshold
Ab	Antibody
Ag	Antigen
Ep	Epitope
Pt	Paratope
$\mathcal{L}$	Length of vector
ψ	Affinity Measure
ψ_{th}	Affinity threshold
ψ_{max}	Maximum affinity
ψ_{norm}	Normalize affinity value
AR	Active Region
S	Shape Space
Ctr	Controller/subcontroller
C	Concentration
W	Network of nodes
N	Set of nodes
ID	Unique identifier number
$IRep$	Internal Repertoire
$XRep$	External Repertoire
$\emptyset$	Empty set
$P_{sharing}$	Probability of sharing data
$NewAb$	New antibody
$CandAb$	List of candidate antibodies
$BestAb$	Best antibody

$AgSti$	Antigenic Stimulation
σ	Standard deviation
f_{obs}	A function which rewards obstacle avoidance behavior
f_{light}	A function which rewards movements towards the light source
f_{puck}	A function which rewards puck pushing behavior
$f_{antipuck}$	A function which rewards puck repelling behavior
b_{puck}	Binary variable which is 1 if puck is within gripper else 0
b_{light}	Binary variable which is 1 if light intensity is above a threshold
F_{T_i}	Objective function for task T_i
v_{trans}	Translational speed
v_{rot}	Rotational speed
T	Set of tasks
S_i	Scenario i
γ_{avg}	Average number of inter-robotic node communications per node
Ω	Convergence count
γ	Discounted penalty factor
β	Barrage size
η	Influx rate
$\mathcal{P}$	Set of problem instances
$\mathcal{S}$	Set of solutions
$\mathcal{M}$	Mapping function between problem instance x and solution y
$\mathcal{M}^*$	Mapping function between problem instance x and the best solution
Sti	Stimulations
Sup	Suppressions
τ	Activation value
μ	Set of mobile agents
$\mathcal{F}_x$	Feature vector of x
ζ	Performance score
DGS	Danger Signals
Sg	Strength of DGS
T_{life}	Lifetime of DGS
l_{span}	DGS spanning length

ΔSg	Strength gradient
ΔT_{life}	Lifetime gradient
R	Set of heterogeneous robots
H	Number of hops of mobile agent
J	Job
n	Number of tasks per job
α	Scaling factor for stimulations
Q_{max}	Maximum queue length
NC	Number of clones
$Q_{current}$	Current length of queue
τ_{min}	Minimum activation value
τ_{max}	Maximum activation value
λ	Affinity proportional mutation
Γ	Lifetime of mobile agent
δ	Scaling factor for suppressions
Ψ	Resources
J_{Dist}	Job Distributor
$\overrightarrow{MV_F}$	Motion Vector Forward
$\overrightarrow{MV_B}$	Motion Vector Backward
Z_i	i^{th} Zone

List of Abbreviations

Terms	Abbreviations
MRS	Multi-Robot System
IoT	Internet of Things
CPS	Cyber-Physical System
*d*CPS	Decentralized Cyber-Physical System
MAS	Multi-Agent System
RPC	Remote Procedure Call
FIPA	Foundation for Intelligent Physical Agents
LAN	Local Area Network
CAN	Campus Area Network
TCP/IP	TCP/IP protocol suite
BIS	Biological Immune System
AIS	Aritificial Immune System
PC	Personal Computer
CNP	Contract Net Protocol
M2M	Machine-to-Machine
OTFP	On-The-Fly Programming
WSN	Wireless Sensor Network
REST	Representational State Transfer
Pi	Raspberry Pi
VANETS	Vehicular Networks
IEEE	Institute of Electrical and Electronics Engineers
NIST	National Institute of Standards and Technology
WiFi	Wireless Fidelity
LTE	Long Term Evolution

RFID	Radio-Frequency Identification
P2P	Peer-to-Peer
CEO	Chief Executive Officer
WMS	Warehouse Management System
LATS	Location-Aware and Tracking System
EA	Evolutionary Algorithm
US	Ultrasonic Sensor
LS	Light Sensor
CS	Color Sensor
UHF	Ultra High Frequency
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
GPIO	General Purpose Input Output
GPS	Global Positioning System

Glossary of Terms

Terms	Description
Agent	Usually software systems or programs which are autonomous, distributed and goal directed.
Artificial Immune System (AIS)	Computational Model inspired from immunological metaphors.
Antigens	Foreign substances which invades a living body and induces an immune response.
Antibodies	Y-shaped proteins used by immune system to neutralize foreign antigens.
Immune cell	A cell which is part of the Immune System.
B-cells	The immune cells generated in the bone-marrow of the body responsible for secondary immune response.
T-cells	The immune cells generated in the thymus of the body responsible of recognizing the antigens.

Internet of Things	Devices including sensors connected to the Internet.
Mobile Agent	A special class of agents which have an added ability to migrate and carry its code from one node to another.
Node	An electronic device which is connected to a network and is able to communicate.
Epitope	An antibody binding part of an antigen.
Paratope	A portion of antibody which binds with the matching epitope of an antigen.
Cross Reactivity Threshold (ϵ)	A parameter which defines a small regions around the antibody within which all antigens can be recognized.
Active Region	Small region around an antibody characterized by ϵ.
Hypermutation	An extremely high rate of point mutation.
Cloning	After an immune cell binds with an antigen, it proliferates by forming multiple copies or clones of it having the same antibody.
Solution	A set of rules, instructions, an algorithm or a heuristic, which can solve a problem.
Middleware	A software that provides an interface between an Operating System and applications.

"You have to dream before your dreams can come true."

A.P.J. Abdul Kalam (1931 - 2015)

Indian scientist and leader

1

Introduction

When the world's first toaster, 'El Tosto', found its way into the common households in early 1900's [2], it was designed to do just its assigned job of toasting bread slices, which it did perfectly. Many years later in 1982, a group of graduate students from the Carnegie Mellon University, connected a coke machine to the Internet [5] to usher in an era of connected devices. This networked coke machine performed tasks which included reporting temperature and stock market trends. Though naïve by today's standards, the data collected from the coke machine showed promises in research on product marketing. Finally in 1990 [3], a toaster was made to switch on and off via the Internet which marked the onset of devices connected to the network.

In the not-too-distant days to come, one might live in a scenario wherein a smart refrigerator connects to a patient's dietician, decides what items need to be bought and places the necessary orders at a grocery store. On delivery, a robot stacks them within the refrigerator and also places the desired foodstuff into a smart oven. After the meal is cooked the oven signals the robot to serve the same at the dining table, which it does before ringing the dinner bell. The human being, of course, remains oblivious of most of these Machine to Machine (M2M) interactions. According to the Nobel laureate, David Kahneman, human beings exhibit a cognitive bias amounting to an irrational decision-making behavior due to their self-interest, thus making them sway away from the wise choices [95]. Unless hacked upon illegally, data collected continuously from devices and appliances portray the bare truth of what

goes on within them. The pioneering work by Kahneman and Tversky [96] thus provides some basis for leaving certain decisions to such smart devices which have already infiltrated our living spaces, rather than completely bank on human beings.

Networked devices have brought a new era of intelligence dissemination creating a pervasive and ubiquitous ecosystem. Moyer et al. [133] define a networked appliance as a "dedicated function consumer device with an embedded processor and a network connection". The devices which were once deployed to mind their own business, now share their information with other entities connected to the same network. The exchange of knowledge has opened up a plethora of application scenarios including smart homes [36], vehicular networks [101], elderly care [130], smart grids [172], localization [143], defence [184], water management [155], animal care [110], etc. The end devices forming the network generate a huge amount of data which is collected and processed to extract valuable information. This surge in data has pushed the research in other domains such as Machine Learning [126], Data Science [197], Deep Learning [72] and Optimization [57], to name a few. The number of connected things will reach 20.4 billion by 2020 [60] which is equivalent to at least 2.5 devices per person on this planet. Rapid advancements in Very Large Scale Integration (VLSI) technologies have also led to the proliferation of cheap sensing and embedded devices giving rise to new paradigms such as Cyber-Physical Systems (CPS) and the Internet of Things (IoT). In addition, robots have added another dimension in terms of mobility and actuation in networked devices. The movement of the robots controlled by mutual consensus from the networked devices envisages interesting applications and research areas.

The term *network* defines the way these devices communicate. One widely used approach is Cloud Computing [154] wherein the data is offloaded and analyzed based on which the devices are controlled. These devices form the leaf nodes while the decision-making is left to the cloud. Though using a cloud for networking is a simple and controlled solution, it suffers from issues such as scalability and privacy [21, 169]. Decentralizing control and distributing the computing among the devices could be an alternate architecture for better intelligence sharing. A recent dissertation work [185] from the University of Cambridge that describes a decentralized email architecture

tested on low-cost devices such as Raspberry Pi[1] (Pi[2]), backs the feasibility of decentralized and distributed approaches. In modern days, the term networked devices have evolved into broader areas such as Cyber-Physical Systems and the Internet of Things both of which are often used interchangeably. Such systems could also be realized in a decentralized and distributed manner.

1.1 Cyber-Physical Systems

The term Cyber-Physical System (CPS), coined by Helen Gill in 2006 [11], refers to the confluence of the *cyber* and the *physical* worlds. A CPS is a theoretical framework which treats its components as models [105]. It encompasses a broad group of devices over a network. The *cyber* component is the software which controls and actuates its *physical* counterpart. The interconnection of various heterogeneous devices forms a CPS. The devices within a CPS are generally resource-limited in terms of their computing capability and network bandwidth.

The initial road-map for research in CPS recognized computing and networking, security and resource-scheduling, as some of the primary research domains [105, 168]. Shi et al. [168] state energy-efficiency to be another significant issue in CPS research. Due to the constraint on resources, a CPS needs to ensure that the participating devices are also energy-efficient. Parolini et al. [142] proposed an approach to keep the energy consumption in data centers at a minimum while also satisfying user requests. With the explosion in the population of networked devices, middlewares or platforms, that are scalable to the current needs of a CPS, are also deemed necessary. In the words of Kim and Kumar [105],

> "...due to the scale, structure, and behavioral complexities of today's and tomorrow's CPS, it is an important challenge to develop extensible, scalable, and adaptable software platforms that can operate in distributed, heterogeneous, time-critical, and safety-critical environment."

A CPS has a strong relation with currently popular paradigms such as the Internet of Things (IoT), Industry 4.0, Networked Robotics, Machine-to-Machine (M2M), Edge

[1] https://raspberrypi.org
[2] Henceforth, in this thesis, Pi strictly refers to the Raspberry Pi.

and Fog computing, Vehicular Networks (VANETS), Network of Things and Web of Things [114]. According to Lee [114], a CPS is a theory which is more durable and flexible in comparison to its implementations (e.g., an IoT) and applications (M2M) counterparts. Though a CPS is a more fundamental term, industrial marketing standards have made IoT as the new face of networked devices. However, an IoT has different perspectives and thus many a time overlaps with other paradigms creating confusion. From the various definitions of an IoT released by IEEE [129], the one by NIST, United States, states -

> "Cyber-physical systems (CPS) - sometimes referred to as the Internet of Things (IoT) - involves connecting smart devices and systems in diverse sectors like transportation, energy, manufacturing and health care in fundamentally new ways."

Following the same notion, this thesis considers all modern paradigms under the umbrella of a CPS and thus uses them interchangeably. For pedagogical reasons, an IoT and its other flavors are discussed in the following subsection. A brief introduction on networked robotics is also presented.

1.1.1 Internet of Things

The Internet of Things (IoT) is an interconnection of physical devices over the Internet. These devices referred to as *things* can sense, store, compute and actuate under certain resource-constraints. A mobile smartphone and a sensor mote are some typical examples of things. Al-Fuqaha et al. [7] discuss six main elements needed to facilitate an IoT. An embedded device equipped with sensors, i.e., the *thing*, first needs to be identified by the use of any addressing techniques such as IPv6. Since data is the main driving force, the second element is the capability to sense and collect data. The third element is communication, which could be through WiFi, Bluetooth and LTE-Advanced [63]. Depending upon the application requirement, the range of communication could be either far (LTE) or near (RFID). Computation can be either onboard the *thing* or on a cloud platform [40]. Cloud platforms process big data and provide real-time analytics. Services and knowledge understanding are the final elements which enable the functionality of an

IoT. Identification and sensing, are the basic components of any CPS. Atzori et al. [10] recognize communication and computation as the main research blocks in IoT enabled applications.

Conventional IoT architectures are centralized wherein all the data is offloaded to a cloud platform which provides virtually unlimited storage and computation facilities [74, 22]. The *things* at the leaf end, sample the data and upload it to the cloud platform. Though relatively simpler to implement, this centralization is prone to privacy issues. In addition, since the cloud is a far-off remote server, latencies are introduced in the execution of associated applications [169]. Further, this centralized notion is difficult to scale for a large number of devices [166].

To ease on the difficulties posed by centralization, paradigms such as *Fog* and *Edge* computing evolved. A Boeing 787 generates 500 Gigabytes of data per flight it takes [1] which if allowed to send to a cloud will introduce critical latencies in the response time. Since the data is collected at the edge nodes, it would be sensible to place the storage, computation, and communication close to the *edge*. This forms the basis of Fog [21] and Edge [169] computing wherein virtualized platforms are deployed between the *things* and the cloud. The simplest example could be a Pi acting as a gateway between the *things* and the cloud. Some of the use cases of computing on edge include connected vehicles, smart grids, and location-awareness [169]. Though edge computing may have mitigated issues in centralization to a certain level, the aspect of a central entity at the top is still prevalent in such systems.

Industries play an essential role in the economy of a country. The CPS and IoT have given rise to a fourth industrial revolution defined as *Industry 4.0* [115]. Smart factories under Industry 4.0 are meant to be sustainable making energy-efficiency an important goal. Such factories include smart machines, devices, logistics and product designs which are now user-centric. Mass Customization [55], flexibility, energy-management, remote monitoring, and automation are some of the characteristics of a CPS/IoT that enabled the concept of Industry 4.0 [171].

Interoperability among different IoT setups is a vital requirement. Organizations adopt different protocols within their IoTs often making them incompatible with those used by other firms thus hindering networking. This led to the pro-

posal of *Web of Things* where the *things* are connected using the client-server based web connections. Though this provides a simple cross-communication architecture, client-server based communication is challenging to scale [100].

1.1.2 Networked Robotics

Growth in automation and smart factories has accelerated the deployment of robots in the industries. Actuation and in some cases, mobility, make robots disparate from the concept of *things* in a typical IoT scenario and thus, demands an explicit treatment. The IEEE Robotics and Automation Society (RAS) [6] defines a networked robot as a robotic device connected to communicate through a wired or wireless medium. According to RAS, a networked robot could be either *tele-operated* by a human user from a remote position through a communication channel or *autonomous* wherein the robots and sensors connected to the same network share the information and perform a task. Kumar et al. [111] extend the coverage of networked robotics to multiple robots connected through generally a wireless medium. According to them, such multi-robot systems can perform tasks which require coordination and collaboration which are otherwise impossible or arduous for a single-robot or uncoordinated multi-robot systems. Jha et al. [87, 90, 91] have realized sharing, learning and distributed task-allocation in a set of networked robots. They have also proposed mechanisms to deliver services and provide synchronization, to a set of connected robots using mobile software agents [146]. Kamei et al. [99] have proposed the concept of networked robots connected to a cloud. The robot could be an embedded device, sensor or a smartphone, all of which share the same cloud platform and are controlled by it. This is similar to an IoT wherein the things include the sensors and robots.

A large amount of work has been done in the area of networked robotics. However, the use of non-conventional methods in this field is still in its infancy. Godfrey et al. [66] have formulated an immunology inspired architecture to service robots connected to a network. They have used mobile agents [35] to deliver solutions to robots that encounter a problem. The use of pheromones [69] and cloning has further extended their approach for large-scale networked systems. The current generation of a CPS of robots requires not just solution-providing services but also a means to

adapt and evolve with the changing nature of problems encountered.

1.2 Decentralized CPS

Kuwahara [112] defines autonomous decentralized systems as an integration of multiple subsystems wherein each of which is autonomous and can perform its tasks independently. The subsystems can collaborate to avoid complete failures and also provide maintenance. Though the definition from Kuwahara dates back to more than 20 years, it encompasses the principles of decentralized and distributed computing in the modern days. A new year resolution by Mark Zuckerberg, Facebook founder and current CEO, provides a democratic view [4],

> "...of the most interesting questions in technology right now is about centralization vs. decentralization....with government using technology to watch their citizens many people now believe technology only centralizes power rather than decentralizes it."

Centralized networks have always been a mainstream technology from an implementation point of view owing to their simpler architecture and comparatively cheap initial cost of investment. Centralization provides more control and authoritative powers to the owner and thus aids in rapid updates in rules and policies. However, as pointed out by Zuckerberg, centralized systems are, prone to single point failures, have privacy risks and scaling them poses a great challenge.

A *decentralized* CPS (*d*CPS) is an amalgam of physical subsystems connected and controlled by cyber units in a decentralized manner. The devices forming such a CPS share their information across the network to which they are connected and make decisions based on collective knowledge. Semwal et al. [166] describe some of the advantages of peer-to-peer (P2P) local sharing of knowledge forming such CPS:

1. Trust: Instead of a central server with no control, the data in a decentralized CPS is distributed across the nodes closer to the user and is governed by his/her own rules. The users thus can have more trust since there is limited or no third party involvement.

2. Reduced network latency: The in-network processing on the nodes which is closer to the user reduces the delays which may occur when the same happens at the remote server side.

3. No single-point of failure: Failure of the central server can cause the whole system to go down. For instance, if Google's email server goes down, productivity (at least to some parts of the world) can come to a stand-still. This may not be the case in a decentralized system where the application could be running at multiple nodes.

4. Running cost: Conventional cloud services are based on the "pay-as-you-go" model which increases the cost as the storage and communication increases [21]. Decentralized CPS favors data processing within the nodes and thus reduces the data traffic.

5. Easy to scale: To be precise, centralized systems should be favored for small or controlled scales. However, decentralized systems only need to connect a node to any of the nodes in the network which make them scalable without affecting the performance.

Thus, decentralization provides a set of merits which can cater to better and smart CPS and IoT based applications. The author in [185] has proposed a decentralized architecture for IoT devices to store and provide email services to the users. The same has been evaluated on a system comprising Pi. Jha et al. [90] have proposed mechanisms to provide sharing and learning in multi-robot systems connected in a decentralized manner. The authors have also presented a stigmergy based approach [87] to control the population of autonomous mobile agents moving across a network of nodes.

Nature is an exemplar of decentralized decision making. For example, in insect colonies, information gathered by an individual is shared through local interactions with its peers. *Trophallaxis* exhibited by bees [109] and pheromone communication in ants [83] are typical examples. Such local interactions lead to interesting global behaviors resulting in mound-building by termites [41] and social wasps forming complex nests [182]. The social behavior in insects and animals have inspired researchers to develop algorithms allowing robots to mimic such decentralized skills.

Termite-inspired robots building structures by Werfel et al. [191] and shape formation by thousand of Kilobots [157] by Rubenstein et al. [158], are some of the remarkable decentralized multi-robots systems. With the increase in the number of devices and robots, research in decentralized CPS throws up promising avenues for novel applications. However, implementing such systems opens up a formidable list of challenges and issues which are required to be addressed for their deployment in the real world.

1.3 Research Challenges in Decentralized CPSs

Though decentralized CPSs have advantages over their centralized counterparts they pose their own set of research challenges, some of which have been described below:

1. Choice of Communication method: Since there is no concept of a central node to which all other nodes are connected, methods that ensure efficient message communication become mandatory. Broadcasting is one such widely used method.

2. Middleware: As pointed in [105], the development of middlewares that can meet the current needs of a CPS poses to be a significant challenge.

3. Dynamic nature of the network: Unlike conventional IoTs and sensor networks, a decentralized CPS includes mobile entities such as robots and vehicles which can at any point of time connect to or disconnect from the network. This calls for mechanisms that cope up with such dynamism, during run-time.

4. Scalability: One of the prime reasons behind using a decentralized approach is scalability. Hence, mechanisms designed for such a CPS should provide the appropriate performance while also facilitating scalability.

5. On-the-fly (run-time) Alteration: Upgradation in centralized servers necessarily mean a downtime during which most services may need to be halted. A decentralized CPS is favored in such scenarios since new nodes can be added or existing ones deleted, in run-time. Providing seamless on-the-fly modifications is a challenging task.

6. In-network Information Processing: In order to reduce data size and thus minimize communication cost, in-network processing is carried out wherein data is processed within the nodes itself and only the relevant information is forwarded. However, execution of remote procedures which can carry and process the data is non-trivial.

7. Distributed knowledge sharing: A single node which is part of the *d*CPS may not have the global perception about a task. Thus, knowledge distributed among the nodes needs to be shared. However, what and how to share form some of the crucial questions that need to be answered in such scenarios.

8. Task Ordering and Mutual Exclusion: Tasks delegated to a *d*CPS of robots may require resources which are shared by them. Thus, a mechanism which can ensure mutual exclusion of resources by ordering the execution of tasks is crucial for such decentralized systems. Formulating such methods remains a great challenge.

9. Selection of the best solutions (algorithms, heuristics, mechanism, etc.): Finding the best solution for a given problem is the essence of any intelligent CPS. However, with different problems distributed amongst the nodes, finding the best solution without any central or globally accessible data on their performance scores, becomes a challenging affair. Mechanisms which can search the mappings to select the best solution for a given problem need to be devised to cater to *d*CPSs.

10. Self-adaptation and Organization: The dynamics of the environment and the nature of the problem can change over time. This demands that the algorithms be self-adaptive to such changes and organize or align themselves towards the course that leads to better performances.

11. Security and Privacy: In a *d*CPS, the data is stored and processed across the nodes. Algorithms that can ensure authorization and authentication and cater to such distributed environments are thus needed. A typical example includes the adaptation of blockchain technology in decentralized scenarios.

12. Generation of new solutions and their evolution: A *d*CPS should be self-sustainable in the sense that it not only selects the best solution but also generates and evolves a new set of solutions as and when required. Thus, for a stream of new problems, the system should be capable of adapting and learning.

13. Life-long Learning: Unlike conventional systems which have a separate training and testing phase, life-long learning facilitates the continuous learning of patterns of incoming problems and devises the best possible solution. Embedding such life-long learning into a decentralized CPS will boost its learning ability.

This thesis aims to lay a foundation for catering to the challenges discussed above. The initial half of the thesis involves the development of a middleware suitable for a decentralized CPS followed by a real-world comparison against a centralized system. The latter half of thesis propounds mechanisms inspired by the Biological Immune System, a classical decentralized system found in vertebrates. The thesis makes use of mobile agents to provide task ordering, mutual exclusion, solution selection and evolution and finally tries to wrap up the whole as a *d*CPS capable of decentralized and distributed learning. A brief review of mobile agents is presented in the next section. A short introduction to the Biological Immune System is also presented in the following section.

1.4 Mobile Agents

Agents are software entities that are capable of performing task(s) on behalf of a user [56]. They are autonomous and possess the ability to make their own decisions and drive themselves towards a goal. Maes et al. [121] refer to agents as computational systems that can sense and act autonomously in an environment to realize a set of goals. Just as human beings and robots form entities in the *Physical* world, these agents can be considered to be their counterparts in the *Cyber* world. The agents are typically static, i.e., they perform their task(s) while residing on top of a computing unit and exchange messages with other static agents to collectively collaborate towards a common goal.

However, there is a particular class of agents which has an added capability of being *mobile* known as *mobile agents*. A mobile agent is a piece of software code which can suspend its current task execution, migrate to a remote node while carrying its code and data and resume the code execution at the remote node side. On completing its task on the remote side, it can also bring the results back to the source node. Outtagarts [139] have listed various applications of mobile agents including e-commerce, smart grids, energy efficiency and metering, network management, e-learning, and web-services. The author also emphasizes the efficacy of mobile agents over other conventional message communication techniques such as client-server architecture, Remote Procedure-Call (RPC) [17], CORBA [159], etc.

In addition to exhibiting mobility, a mobile agent can also clone and multiply itself, carry a payload (data or a program), make local decisions, execute a program on a remote site or node, etc. Mobile agents can also be used to churn out and carry intelligence along with them as they migrate within a network [78]. They have been used in a wide range of applications which include wireless sensor networks [33], robot control [97, 179], e-commerce [122], security [24, 120], e-learning [102], networked robotics [66, 67], IoT [64, 163], etc. Some of the major advantages of using mobile agents are:

(a) Bandwidth and latency reduction: A mobile agent has the innate ability to carry the computation in the form of code to a remote site. Instead of fetching the whole raw or unprocessed data from a remote site, *mobility* allows for the computing program or logic to migrate to this site and process the data therein. This results in reducing network traffic and latency.

(b) Discontinuous operation: In a dynamic network where the devices are mobile, it is rare that a continuous connection is maintained between two nodes for a long time. In a conventional client-server system, a sudden disconnection may cause the server to resend the whole data, making it an expensive affair. On the contrary, in a mobile agent-based scenario, migration occurs only when a connection is established. The mobile agent then resides in the new node till the connection to the next node is available. Unlike large amounts of data that need to be processed, a mobile agent is comparatively lightweight. Thus,

a failure in migration does not compound into significant losses in bandwidth and time.

(c) Cloning: The ability of a mobile agent to clone aids in creating multiple copies of the program in the nodes forming a decentralized CPS. This facilitates a certain amount of parallelism and hence rapid execution within the CPS. Godfrey et al. [68] show how cloning can hasten the process of servicing such nodes.

(d) Adaptivity and flexibility: In a traditional centralized system, any upgrade would require the system to be brought down, changes made and then restarted. In a mobile agent-based system, upgrades could be packaged within the mobile agent and released into the network. This *On-The-Fly Programming* (OTFP) [163] support facilitates a higher amount of flexibility. Agents can sense and perceive their environment and change their behaviors accordingly. A mobile agent can add new behaviors in the form of a payload and can also adapt to different situations.

Mobile agents thus have the potential to provide a viable distributed solution to challenges related to a *d*CPS [160].

Mobile agents have been a prominent tool in Wireless Sensor Networks (WSN) where continuous network connectivity is difficult to achieve [33]. Godfrey et al. [64] have proposed a framework for IoT applications where mobile agents are used for providing emergency services to the connected devices. A recent dissertation by Lappänen [117] presents mobile agents for IoT in a resource-oriented manner. The framework proposed provides a REST-based interface to developed opportunistic agent-based applications. Kambayashi et al. [98] have used mobile agents in multi-robot scenarios to implement an intelligent cart system. They have used an evolutionary ant clustering algorithm to design the system. Jha et al. have proposed a mobile agent based mechanism to provide synchronization [87], learning [90] and distributed task allocation in a milieu of networked robots. Semwal et al. [164] have presented ordering of tasks to provide mutual exclusion in a *d*CPS. Mobility in agents has also been exercised in a railway system to provide onboard services to the passengers [59].

These features and applications, thus, justify the use of mobile agents as a tool for realizing *d*CPSs. The following sections describe the significant contributions and provide a brief chapter-wise summary of the thesis.

1.5 Contributions of the Thesis

The major contributions of the thesis are described below:

1. **A middleware for a decentralized CPS:** A brief introduction to the middleware, nicknamed, *Tartarus*, used as a platform to bridge the gap between the cyber and physical systems forms the initial portion of Chapter 2. Though there are several multi-agent platforms, one that can enable the deployment of decentralized and autonomous software entities which in turn can control physical components, is grossly missing. This contribution extends the functionalities of the existing *Tartarus* platform so as to allow it to access different hardware such as Pi and LEGO MINDSTORM NXT robots. A modular architecture for multi-agent platforms which can be used for implementing CPSs is also proposed. A CPS comprising robots, sensors and a human-in-the-loop is also implemented to show the feasibility of using *Tartarus* in real-world scenarios.

2. **Towards decentralization of Cyber-Physical Systems using Mobile Agents:** This contribution presents a mobile agent based approach to transform a centralized CPS into a decentralized one. The proposed approach makes use of local information sharing to achieve a global goal. With a Location-Aware and Tracking System (LATS) as a real-world CPS application, the efficacy of the proposed approach has been portrayed. Mobile agents are responsible for monitoring and tracking the movements of human beings without the need for a central logging server. Fetching the output of queries such as *when* and *where* is or was a person present in an area, is fairly simple from a central database stored in a remote node. However, in large-scale systems where decentralization becomes a necessity, catering to LATS becomes a challenging task. This work uncovers the complexities of implementing a *d*CPS and propounds a simple approach to deal with the problem.

3. **A Mechanism for Mutual Exclusion in a *d*CPS of Multiple Robots:** This contribution mitigates the problem of mutual exclusion which can arise in a *d*CPS of robots. Robots may require resources which are shared, to complete a task. However, if more than one robot tries to access this shared resource without any central commanding authority, it may lead to a catastrophic situation. To address the problem of mutual exclusion in a *d*CPS comprising mobile robots, an agent-based approach is proposed. The approach orders the sequential, interdependent and independent tasks in the form of a pipeline wherein the robots form *mobile computing nodes*. A conventional pipelined computing architecture uses a global clock to synchronize the execution of instructions. In contrast to the fixed times consumed to execute instructions, the execution of tasks by robots in the real world could vary with time due to various factors such as path taken, remaining power in the battery, etc. In order to cater to such dynamic changes in execution times for the same task, the pipeline formed by a set of mobile robots need to possess an inherent adaptive synchronizing mechanism. The proposed approach uses mobile agents that knit through real mobile robotic nodes and manage the execution of tasks in a pipelined manner. The mechanism also facilitates on-the-fly addition and deletion of both robots and tasks. The main highlights of this contribution are:

   (a) A mobile agent based decentralized and distributed mechanism for ordering multi-robot task executions.

   (b) A solution for the mutual exclusion problem among multiple robots within a CPS.

   (c) Validation of the proposed mechanism through emulation.

   (d) Real world implementation of the proposed mechanism using mobile robots in a Warehouse Management System (WMS) type scenario.

4. **Selecting the best solution in a *d*CPS:** The previous contribution assumed that the solutions available within the network are distinct and the best. However, in real systems, there could be several solutions for a given task/problem. Choosing the best among these based on their performances

especially in decentralized and distributed systems remains a challenge. This contribution proposes an immunology inspired mechanism for finding the best solutions for problems distributed across the networked system. Though several hyper- and meta-heuristic approaches have been used to find the best solution, scenarios wherein the problem may change or new ones may arrive, systems employing such methods may need to be halted and retuned offline. This becomes cumbersome if the problems are spread across the nodes in a network. The problem could be one in the soft world, for instance, sorting a set of numbers or it could be something in the real world such as the task of pushing objects in a robotic scenario. Realizing this using conventional client-server based methods will not be robust. Further, it is difficult to scale. The problems can occur at any of the nodes. The goal is to evolve and select those mappings that select the best solution. The mechanism uses mobile agents to carry the evolved mappings as their payloads and share them across the network. The system continuously evolves and learns the mappings life-long.

5. **A Mechanism for the distributed evolution of solutions:** This forms the last contribution of the thesis and discusses a novel immuno-inspired behavior evolution cum selection mechanism for a *d*CPS of robots. Traditional evolutionary methods use a single monolithic robotic controller to perform a task. However, due to complexity in the task, the evolutionary process suffers from bootstrapping and deception issues. The proposed approach divides the main task into several subtasks/problems and evolves separate sub-controllers/solutions for each of them. Each sub-controller forms a new solution for the given problem (subtask) at hand. The approach is distributed in the sense that each node evolves its own set of controllers and shares them with other nodes via mobile agents. Experiments conducted on three different real scenarios portrays the feasibility of the proposed mechanism. In addition, a comparison of results obtained through emulation reveals that the mobile agent-based approach for sharing of solutions is far more energy efficient than broadcasting based methods. This work enhances the previous contribution by not only selecting the best but also generating and evolving the new solutions thereby delivering a life-long learning system.

It may be noted that mechanisms proposed in this thesis do not address some of the issues faced in implementing large scale CPSs such as security of agents and CPS, energy consumption, communication protocols and time-critical executions. These may be left as the future scope of this thesis.

1.6 Outline of the Thesis

The thesis comprises eight chapters. The chapter wise organization of the thesis is given below:

1. **Chapter 1:** This chapter provides the survey and motivation behind the research. For pedagogical reasons, a few terminologies and their origins are discussed.

2. **Chapter 2:** This chapter presents *Tartarus* as a middleware that can be used to realize *d*CPSs. The chapter is based on the work published in [163, 167].

3. **Chapter 3:** A mobile agent based approach to realize a *d*CPS is presented herein. The chapter uses *Tartarus* as a tool to develop cyber entities which in-turn can control the physical devices. The contents in this chapter have been published in [166].

4. **Chapter 4:** This chapter presents a mobile agent based mechanism to solve the problem of providing mutual exclusion in distributed systems devoid of central commanding authorities. The mechanism proposed was tested on a real-world experimental warehouse developed in the Robotics lab. at the Department of Computer Science & Engg., IIT Guwahati. The contents of this chapter are based on the published work reported in [164].

5. **Chapter 5:** This chapter discusses the biological concepts and computational models derived from immunology. The chapter forms a prequel to the subsequent ones which use the principles and theories, propounded by immunologists, to solve a set of problems in a *d*CPS.

6. **Chapter 6:** A mechanism to evolve and share mappings which select the best solution for problems distributed across a network of nodes, is presented in

this chapter. An extensive analysis in the emulated world and a real-world implementation is also presented.

7. **Chapter 7:** The work in the previous chapter assumed a continuous supply of new algorithms from a source. However, in the real-world, encountering new problems implies the need to create and evolve new solutions that can cater to the dynamic nature of the environment. This chapter presents an embodied behavior evolution cum selection algorithm for a *d*CPS of robots. The chapter is based on the published work reported in [165].

8. **Chapter 8:** This final chapter highlights the conclusions arrived at. A summary of the contributions made is presented and new avenues for future research have also been described.

"Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world."

Albert Einstein (1879 – 1955)
Theoretical physicist

2

Bridging the Gap between Cyber and Physical Systems

The pervasive Internet has lured a variety of embedded devices including sensors nodes, controller boards, smartphones, etc., to form their respective niches within its fold. With a heterogeneous set of devices forming the nodes within, using the Internet as a mere passive entity amounts to a gross under-utilization of such a massively networked resource. By tethering robots as mobile nodes onto this network, it is possible to realize both the flow of information as also on-demand *physical* actuation. Managing the communication and control of such a *Cyber-Physical Systems* (CPS) is a complex task and calls for an integrated platform that can cater to scalability, heterogeneity, and ability to be programmed *on-the-fly* and controlled in a distributed and decentralized manner. A Multi-Agent System (MAS) is a paradigm that can cater to the requirements of managing various processes within a CPS. Agents herein, form the cyber components and sit on top of the physical devices (such as embedded hardware, robots, etc.) and cause the execution of a designated set of tasks to achieve the desired goal. Such systems require a platform or middleware for the creation and management of agents. Most MASs use static agents that remain resident on the devices. However, the mobility of such agents can become a vital means for learning and information dissemination across a network of CPSs [90]. Mobile soft agents are capable of migrating autonomously across the network

they inhabit and also execute their inherent code at desired nodes. While there are numerous multi-agent platforms, a system that can cater to the creation of a cyber system comprising static and mobile soft agents which in turn can control physical systems, is grossly missing. *Tartarus* [163], an open-source multi-agent platform has distinctive features which include heterogeneous OS support, interface to embedded hardware, *on-the-fly* programming and security which can satisfy the needs for implementing a CPS. However, its capabilities are limited to emulation of CPS/IoT scenarios [18]. Along with its emulation abilities, *Tartarus* can be an ideal candidate to be augmented with real-world CPS *control* facilities.

This chapter extends the functionalities of *Tartarus* as a multi-agent platform in order to network and integrate robots with other heterogeneous devices so as to form a CPS. Its inherent agent-based technology provides a range of features which include autonomy, intelligence, distributed and decentralized control, among others. The chapter describes the manner in which these functionalities were extended to provide vital features that can make the cyber agents access and control real physical devices with ease. The chapter describes the development and integration of software interfaces which allow access to different hardware such as Pi, Intel® Galileo, Lego® Mindstorm® NXT, etc., all of which make *Tartarus* a tool that can be used not merely for study and analysis of such systems, but also to implement and deploy MAS based CPSs in the real-world. This chapter thus portrays *Tartarus* as a platform to bridge the gap between the *cyber* and *physical* worlds. After an initial motivation followed by a description of the architecture and different existing and augmented features of *Tartarus*, this chapter presents an implementation of a CPS of robots.

2.1 Motivation

The current cyberspace is growing at a fast pace with millions of new devices being added every year. Such a growth has propelled research towards bringing computations closer to the devices in their physical space. Cyber-Physical Systems (CPS) form a new area of research wherein researchers attempt to bridge the gap between *Computational* and *Physical* entities. CPS involves an interconnection of various

devices such as sensor nodes, mobile phones, embedded computers, PCs, routers, etc. and focuses on the interactions of the computational and physical processes with real-time feedback. This makes a CPS both heterogeneous and distributed. Further, a CPS may require intelligence to be embedded within the devices so as to make it smart. In order that devices make their own localized decisions without human intervention, autonomy for the devices within the CPS becomes mandatory. It is naturally desirable that a CPS be able to accommodate heterogeneous devices, provide autonomy to all devices within, facilitate distributed computing while also be scalable. Addressing these challenges calls for the formulation of a platform that can allow rapid prototyping and programming of a CPS.

A *CPS of robots* could be one wherein robots, which are active entities interfaced to the physical world through sensors and actuators, form the major component. Robots are considered to be autonomous or semi-autonomous beings and hence must be capable of decision making. The terms scalability and heterogeneity in a *CPS of robots* would thus allow multiple and different classes of robots to form a complex CPS. Realizing such a CPS which has multiple robots interacting with a host of other devices, all of which aid in solving problems in a distributed or decentralized manner, is a challenging task.

Multi-Agents System (MAS) is a paradigm that can address such challenges from a computational perspective [153]. It involves the coordination of processes springing through the local interactions of individual agents which results in an overall intelligent behavior exhibited by the system. Agents [202] in a MAS can be defined as software entities that carry out a task on behalf of a user, organization, or any client-side system. An agent has various distinguishing features such as autonomy, reactivity, pro-activity, social interaction, adaptability, rationality, specialization, etc. that separate it from any other conventional piece of software. Agents are mostly known to be static entities. However, their mobility adds another dimension to their utility, especially in distributed and decentralized environments. Agents equipped with the capability to migrate from one node to another over a network are called as *Mobile Agents* (MA) [78]. Researchers have used mobile agents in a variety of applications such as robotics [149], network management [16], electronic commerce [122, 140], energy efficiency and metering [209], wireless sensors [33], grid

computing [148], distributed data mining [107], security [24], and e-learning [207], to name a few .

The intrinsic capability of a mobile agent to migrate and make local decisions autonomously makes it a good candidate for a middleware solution for a CPS. Such a mobile agent based middleware for a CPS could help in alleviating the challenges of distributed processing, scalability, flexibility and localized decisions. It is important to note that within a CPS, the end devices should be provided with autonomy facilitated through their computational core. Mobile agents provide a disruptive technology to bring together the computational and physical processes within a CPS. *Tartarus* and its predecessor *Typhon* [123] have been used in a variety of applications ranging from an agent based Internet of Things (IoT) [64], synchronizing cooperative tasks among robots [87], creating partial green corridors for emergency services in a traffic [18], sharing and learning on-the-fly by a set of distributed nodes in a network [90], etc. However, most of these work were limited to emulated scenarios and do not cover the heterogeneity of CPS and IoT systems which are desired in this thesis.

This chapter reforms *Tartarus* into a software tool to render distinctive features that can make the cyber agents access and control the real physical devices with ease. *Tartarus* is completely coded in open-source SWI-Prolog [196] environment. This facilitates *Tartarus* to utilize all Prolog based features [156] such as assertion and retraction of rules, dynamic database, etc. *Tartarus* provides all agent related functionalities including mobility, cloning, and payload carrying capacity, among others. Further, it supports multithreading and heterogeneity in terms of devices. Multithreading allows the platform to host more agents and also facilitates concurrency of all operations within, thereby increasing throughput while also making better use of system resources. This chapter is towards the integration of additional interfaces to allow *Tartarus* to seamlessly bind with the robots and embedded devices such as a Pi.

The next section provides a brief background on the available multi-agent platforms and comments on their viability in CPS and IoT scenarios. In the succeeding sections, the architecture of *Tartarus*, its features followed by a real-world implementation, are presented.

2.2 Bridging Frameworks

In order to control an agent based system, a proper and well-defined host or platform is prime. Currently, agent systems have not been embedded on to any OS. Several middleware environments have however been developed and made available for deployment of agents. An agent platform should be based on an interpreted language as most of the code contained within, for instance, a mobile agent, may have references to functions or predicates that may not be available with the host [78]. This is so because mobile agents are deployed at the remote server. Researchers have developed *mobile* agent platforms in various programming environments. JADE [15], Concordia [200], Agent Tcl [73], Ara [144], TACOMA [93], AgentSpace [174] and Aglets [178] are some of the Java-based mobile agent platforms. Concordia, JADE, and Aglets are purely Java based while Agent Tcl supports three languages viz. Tcl and Schema other than Java. Further Ara supports Tcl, C/C++ and Java with a customized compiler for agents written in C/C++. Prolog-based agent platforms include Jinni [181] which is based on the principles of both Java and Prolog. It supports multithreading, control mobility and inference processing and works in conjunction with BinProlog [180].

Except few, most of the platforms discussed have been reportedly used for realizing purely software systems and their use in the realm of embedded devices is seldom discussed. TACOMA LITE, a low footprint version of TACOMA can run on PDAs. Mobile-C [132] a purely C/C++ based platform with an embeddable C/C++ interpreter that can be used in conjunction with small embedded systems. With its small footprint this platform has been used to develop an agent based real-time traffic detection and management system [132]. AgentSpace, a Java-based mobile agent platform has been used to realize an Internet of Things [64]. *Typhon* [123], is a WinProlog based mobile agent platform that runs on the Chimera (static) Agent System (www.lpa.co.uk). It has been used in conjunction with LEGO® MINDSTORMS® NXT based robots and other third-party sensors in applications that bear a resemblance to a CPS [87].

While most of the popular agent systems are yet to find their way into full-fledged applications which involve control of embedded systems, a few like Mobile-

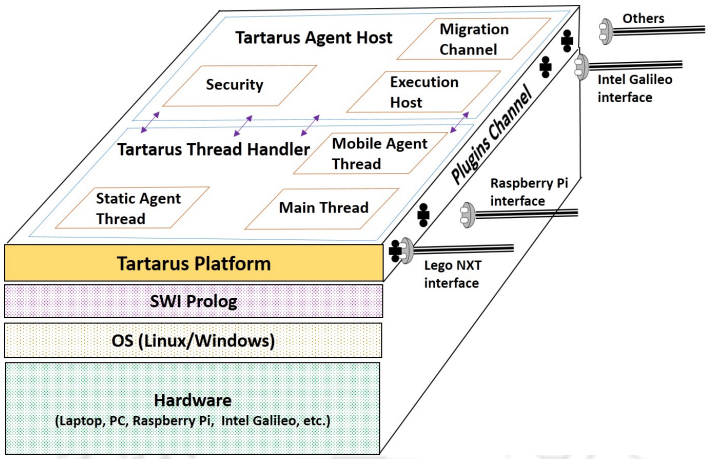


Figure 2.1: Architecture of *Tartarus*

C and *Typhon* provide these features but only to a limited extent. In order to add intelligence and program for the same, Prolog stands out due to its inherent search engine, dynamic code manipulation, and rapid prototyping features. A CPS framework which could facilitate programming in Prolog coupled with both static and mobile agent paradigms could definitely aid in achieving scalability, distributed and decentralized control, code evolution and even *on-the-fly* programming. A smart CPS not only involves computation but also intelligence in making autonomous decisions. As a system, it can comprise homogeneous as well as heterogeneous devices like sensors, embedded controllers, computers, PDAs, etc. The next section presents a description of the architecture of *Tartarus*.

2.3 Tartarus: A Multi-Agent Platform

This section explains the architecture of *Tartarus* along with its significant features and components. *Tartarus* is a platform that can cater to both static and mobile agents, both of which are coded using Prolog.

2.3.1 Architecture

Figure 2.1 depicts the basic architecture of the *Tartarus* platform. As can be seen in the figure, the platform runs atop the SWI-Prolog environment which forms the core execution engine. Further, *Tartarus* comprises three major components whose functionalities are as follows:

1. *Tartarus* Agent Host (*TAH*): The *TAH* forms the core of the *Tartarus* platform. It provides all the services required to create and manage a suite of agent related functionalities within the platform. The *TAH* is responsible for the creation of static and mobile agents, migration of mobile agents from one platform to another, management of agent execution, agent security and the facilitation of all inter-agent interactions. The *TAH* also performs various bookkeeping activities which include keeping a record of the current agents inhabiting the platform, agent hop-times and agent parameters such as process id, agent name, active agent port numbers, payload information, agent handlers, etc.

2. *Tartarus* Thread Handler (*TTH*): The *TTH* acts as a controller for managing multiple threads of executions within the *Tartarus* platform. All the agents whether static or mobile run as a separate thread within *Tartarus*. The *TTH* is responsible for the synchronization of these threads and the preservation of their state and data. Hence, the *TTH* provides a concurrent execution environment within the *Tartarus* platform.

3. *Tartarus* Plugin Channel (*TPC*): The *TPC* forms the crucial component that extends the CPS based functionalities of the *Tartarus* platform. It facilitates the interfacing of the platform with the hardware entities or controllers such as the Pi and NXT robots. Hence, new devices can easily be interfaced with the *Tartarus* platform by integrating their respective plugins via the *TPC*. Such plugins, however, are specific to controllers and need to be written by the programmer or supplied by the third-party vendor who provides the hardware.

2.3.2 Features

Table 2.1 depicts some of the significant predicates within *Tartarus*. Predicates arguments, syntax and their description are mentioned therein. Following is a list of features provided by the *Tartarus* platform:

1. Multi-threading: *Tartarus* provides concurrent creation and execution of mobile agents using multi-threading. Its engine takes care of the management and

Table 2.1: Some of the significant predicates within the *Tartarus* platform

Function of the Predicate	Description	Syntax
Agent Creation	Creates a new agent with the specified name or generates a name out of random alphabet.	*create_static_agent*(AgentName,(IP,Port), HandlerName) *create_mobile_agent*(AgentName,(IP,Port), HandlerName)
Agent Migration	Allows movement of an agent from a source *Tartarus* host to the *Tartarus* destination host, thus empowering mobility to the agents.	*move_agent*(AgentName,(IP, Port)).
Agent Payload	Allows addition of user defined predicates and data.	*add_payload*(AgentName, ListOfPredicates).
Agent Communication	Exchange of messages among static agents stationed at different *Tartarus* hosts.	*agent_post*(AgentName,(DestinationIP, DestinationPort),ListOfPredicates).
Agent Clone	Creates a clone of the parent agent at a given IP and Port number.	*clone_agent*(AgentName,(DestinationIP, DestinationPort),CloneName).
Agent Kill	Kills the specified agent.	*kill_agent*(AgentName).

synchronization of the threads created thus ridding the users the complexities of managing parallel or concurrent computing.

2. Heterogeneity: *Tartarus* is independent of the underlying OS on which it runs. *Tartarus* running on Linux can seamlessly communicate with *Tartarus* running on Windows, thus providing for cross-platform communication.

3. Hardware-in-the-loop: Unlike most of the available agent platforms, *Tartarus* can be ported on embedded boards that currently include the Pi and Intel® Galileo development board. It also comes with an interface for the LEGO® MINDSTORMS® NXT robots. These special interfaces allow developers to access the control systems of the underlying hardware, thus making the job of developers easier.

4. Distributed Control: Agents within *Tartarus* have the capability to share data between nodes, migrate from one node to another and execute the code on a destination node. This localized nature of agents makes it possible to realize distributed and even decentralized control systems. For instance, a remote node N_i can easily retrieve current raw data from a sensor node connected to

N_j and trigger a robot at N_k.

5. Scalability and *on-the-fly* programming: If the number of entities comprising a system can be increased or decreased with ease without compromising the performance, then the system is said to be scalable. However, this notion of scalability is weak and may require that the entire system be brought down or powered off during the process of scaling. With *Tartarus* a system can achieve strong scalability through *on-the-fly* programming. This feature allows addition or deletion of entities or platforms to or from the system seamlessly even during its execution. By introducing mobile agents with appropriate programs as their payload, it is possible to rewrite or retract code at various nodes which will ensure that the overall system works with a new set of hardware or nodes attached, deleted or modified.

6. Payloads: Mobile agents can carry a set of programs or data as a payload to be either executed on demand at the destination node or be downloaded for use by that node. *Tartarus* provides the necessary execution environment to the agents in order to execute a program based on their defined logic. Agents can also be programmed to shed (offload) or upload payloads during runtime. By modifying payloads, one may even attempt to alter the behavior of the agent in runtime.

7. Security: Security in *Tartarus* is at the moment naïve and is implemented using a lock and key mechanism. Each *Tartarus* platform has a lock which could be an encrypted code or password. A mobile agent is allowed to enter a *Tartarus* node only if it has the associated key for the same. Better security aids could be developed to make the system secure. An investigation into adopting Blockchain Technology [147] for securing mobile agents is currently being carried out.

8. Ease of Installation: Installing *Tartarus* is simple and does not require any intricate adjustments such as setting path variables as in the case of Java-based platforms.

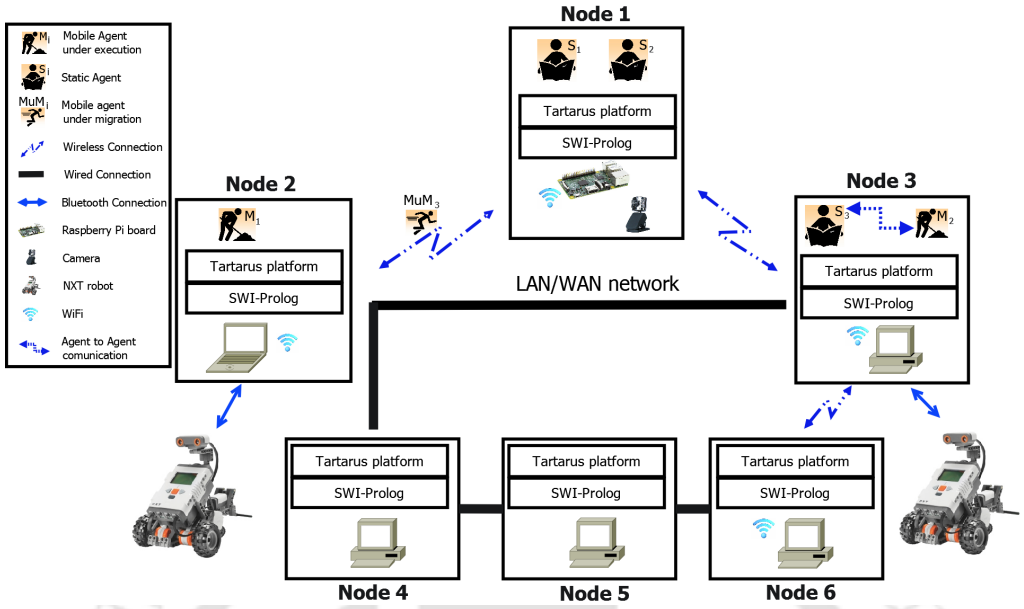


Figure 2.2: A depiction of the *Tartarus* platforms configured as a CPS

2.3.3 Tartarus for CPS

The *Tartarus* agent platform uses the TCP/IP protocol stack for communication over the wired or wireless channel in a LAN/WAN. Each *Tartarus* platform creates its own TCP/IP socket for initializing the communication interface with other *Tartarus* platforms available within the network. *Tartarus* platform allows the creation of both static and mobile agents. Mobile agents can communicate amongst themselves or with static agents and vice versa. Each static agent creates a separate TCP/IP socket and thread for communication and execution respectively. Mobile agents use the default *Tartarus* platform socket for any form of communication. Since *Tartarus* is created using the SWI-Prolog environment, it provides the flexibility of generating interpreted code. *Tartarus* can thus support *on-the-fly* programming (discussed later). Further, the code of *Tartarus* mobile agents could evolve based on the feedbacks collected from the different nodes in the CPS network.

Figure 2.2 shows an approximate visualization of several *Tartarus* platforms configured as a CPS comprising a Pi controller board (Node 1), a laptop (Node 2) and desktop computers (Nodes 3, 4, 5 and 6). In addition, each of the nodes 2 and 3 are connected to a LEGO ® MINDSTORMS ® NXT robot via Bluetooth. The communication among the nodes can be wired or wireless as shown in the figure.

Tartarus platforms are shown hosting mobile agents M_i (M_1 and M_2) and static agents S_i (S_1, S_2 and S_3). The mobile agents under migration are depicted as MuM_i. In Figure 2.2, S_1 and S_2 are hosted at Node 1 while S_3 at Node 3. Further, M_1 and M_2 are under execution at Nodes 2 and 3 respectively while MuM_3 is a mobile agent 3 migrating from Node 2 to Node 1 on a wireless link. The intra-node communication between a static agent S_3 and a mobile agent M_2 is shown on Node 3.

2.4 Tartarus: Real-World Application

To validate the applicability of this extended version of *Tartarus*, a CPS comprising robots, sensors and a human-in-the-loop is implemented. A total of three NXT robots equipped with light and ultrasonic sensors were used along with a Pi Model B+ board coupled with a webcam. Since the NXT robots are not Wi-Fi enabled, they were connected using Bluetooth to the PCs running Windows. The *Lego interface plugin* for the *Tartarus* platform, also developed in-house at the Robotics Lab. at the Department of Computer Science & Engineering IIT Guwahati, was used to control the robots. The Pi hosted the *Tartarus* platform running on Linux. The camera was connected to the Pi board as shown in the Figure 2.4 and interfaced via the *Pi plugin* (also developed in-house) attached to the *TCP* of the *Tartarus* platform. The PCs and the Pi were networked through the wired Ethernet backbone. A total of 13 *Tartarus* platforms formed the nodes which were connected to form a CPS as shown in Figure 2.3.

As an example, CPS scenario, a situation wherein a vehicle encounters a roadblock (for instance, a tree fallen across the road), is considered. The vehicle then signals to the nearest node for help, which in turn takes a snapshot of the condition and sends it over a human administrator. Based on the situation, the latter sends an agent in search of an appropriate rescue vehicle that can remove the block. The agent migrates across the network, finds the concerned vehicle and triggers it to move towards the roadblock and eventually removes it.

Figure 2.3 shows the aerial view of the implementation of the above scenario of the CPS with robots. The CPS comprises a network of 13 nodes - 12 PCs and

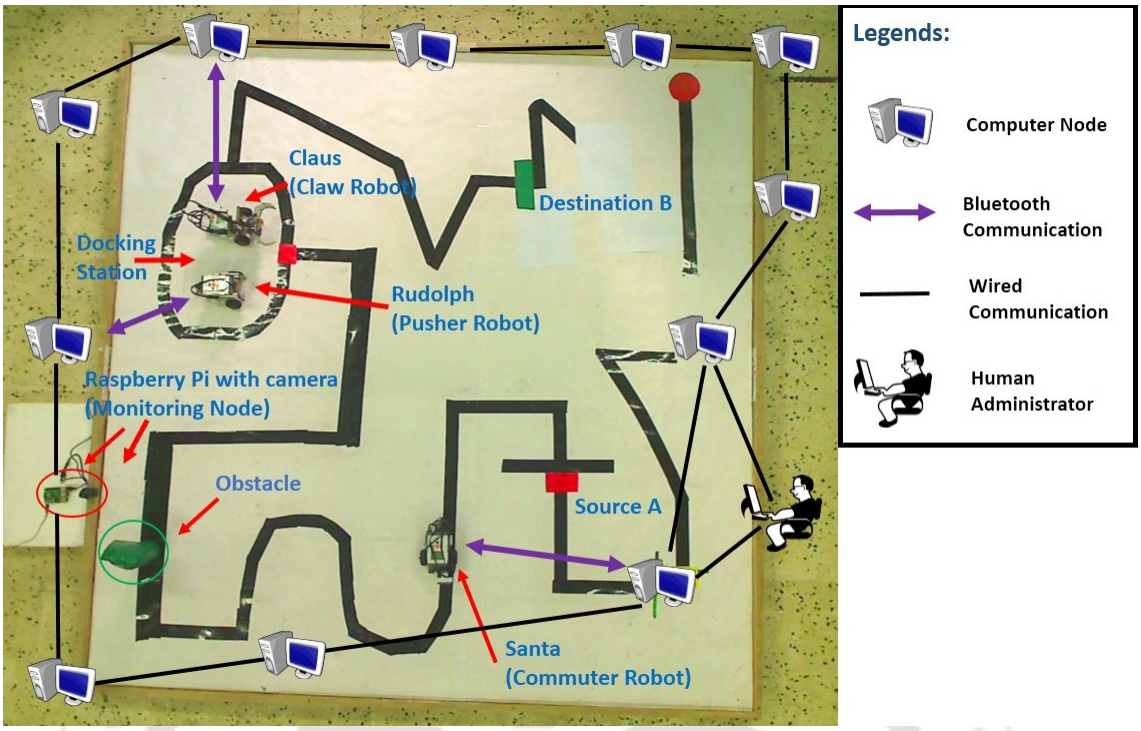


Figure 2.3: The top-view of the implementation setup of the Cyber-Physical System comprising computer nodes (not shown physically) and Pi board with Webcam all configured as *Tartarus* hosts. (The robots are linked to the computer based hosts via Bluetooth)

a Pi controller board, configured as a *Tartarus platform*. Thus, each of these nodes has the *Tartarus* software installed to make them act as *TAH*. The figure also shows three robots - one commuter and two others which are stationed at a docking station and are specially designed for specific purposes which include rescue and salvage operations. The commuter, robot nicknamed *Santa*, is the one that travels along the black-lines path and is analogous to a vehicle on a conventional road. All robots maintain a wireless link (Bluetooth) with a node in the network. Heterogeneity in hardware and software environments is emphasized by the use of the PC based nodes running on Windows and the Pi on Linux, all of which act as *TAH*. A Webcam, interfaced to the Pi board placed near the path, serves as a *Monitoring node* in the CPS network. More of such monitoring nodes could be envisioned to be populated along the length of the path with motors to control their pan and tilt. The 12 PCs used were the maximum available at the time of experiment. However, more number of nodes can be added to increase the scale of the network.

Figure 2.4: The Pi based *Tartarus* host interfaced to the Webcam

Figure 2.5: The snapshot captured by the Webcam mounted on the Pi based *Tartarus host*

As the commuter *Santa* travels along the black path from Source A to Destination B, it encounters a roadblock, a green object symbolizing the tree. As soon as, this robot detects the obstacle using its ultrasonic sensor, it communicates to the nearest *Monitoring node*, which in this case is the Pi. The static agent within the monitoring Pi based *TAH* host, in turn, initiates an action to assess the problem by activating its webcam and taking a snapshot of the current situation. The snapshot is then forwarded to the human administrator available at another remote *TAH* in the network. Figure 2.5 shows the image captured by the webcam in our implementation. Based on the snapshot, the human administrator ascertains the gravity of

the situation and spawns a *Tartarus* mobile agent into the CPS network with the instruction to find the appropriate kind of *rescue* robot that is able to remove the obstruction from the location of the problem. The program to be executed by the *rescue* robot to overcome the problem at hand is also bundled in as its payload. The mobile agent migrates into the CPS network and forages for the specified *Rescue* robot. Since there are two different types of *Rescue* robots, named *Rudolph* with a capability of pushing objects and *Claus* which has claws to pick and place objects, stationed at the docking station, the mobile agent compares their respective configurations against its requirements and chooses the *Claus*. It then initializes *Claus* to start following the black path towards the location of the roadblock. On reaching the specified location *Claus*, using the program in the agent's payload performs the grabbing of the obstacle using its claws and removes it from the path. The commuter robot *Santa*, now finds the path free and continues its sojourn along its path to eventually reach its destination. Having accomplished its task, the *Tartarus* mobile agent guides *Claus* back to its docking station[▶].

2.4.1 Discussions

While the scenario of the CPS with robots implemented seems naïve, it throws more light on the feasibility of using *Tartarus* as a platform for realizing a full-fledged CPS. With a proper (wired or wireless) network of *Tartarus* platform in place, it is possible to make mobile agents move to sense and actuate the connected physical devices including robots and sensors across the network. This potentially opens up a plethora of applications in the real world where robots, sensors, gadgets, and mobile devices need to communicate to perform tasks in a distributed, decentralized and asynchronous manner. In the above implementation, it could happen that *Claus*, while at work on the problem, needs the assistance of *Rudolph* for pushing an object. *Claus* (or the human administrator) can thus request its nearest *TAH* to spawn mobile agents to usher in *Rudolph* to the site of action. Further, if the tasks involved need to be performed in a collaborative and synchronized manner the same could be achieved using a set of mobile agents [87]. The human administrator

[▶] A video of the implementation can be accessed at the following link: https://www.youtube.com/watch?v=qcLgqFjAtik

could pitch in to change the course of actions of the robots and other devices that populate the net by spawning new mobile agents carrying the new programs as their payloads *on-the-fly*. Such agents would seek and search the targeted robots in the CPS network and replace the older programs within the robot/devices, thereby changing their respective behaviors when the overall system is still in operation *(on-the-fly)*.

2.5 Chapter Summary

This chapter detailed the manner in which a multi-agent platform, *Tartarus*[Ω], can aid in the realization of a CPS that includes mobile robots, vehicles, and other heterogeneous devices. The significance of using agents, both static and mobile, has also been stressed. Mobile agents that are inherently autonomous allow the flow of robotic or device-specific programs *on-the-fly*. These agents can facilitate the change of behaviors and actions on the part of the entities (robots and devices) within the network, even while the CPS is active and running. New systems or devices capable of hosting *Tartarus* can be added to an already running CPS network *on-the-fly* thereby making them scalable even during run-time. The features of *Tartarus* together with the implementations portrayed in this chapter, validate its feasibility as a deployment tool for real-world CPS based applications. The fact that a live demonstration of *Tartarus* was allowed at the top-tier International Conference on Autonomous Agents and MultiAgent Systems (AAMAS) held in Singapore [167], authenticates its usefulness in realizing real-world CPSs. A video on the same can be found here[▶].

Connecting the cyber world with its physical counterpart is always fraught with hurdles. The use of *Tartarus* alleviates this to quite an extent by empowering agents to actually sense, think and act in the real world. It is envisaged that by 2020, the number of connected devices across the world will reach 20.8 billion [60]. This will result in the creation of an ecosystem comprising different IoT/CPSs working either sequentially or concurrently. While one IoT may be used for weather forecasting and reporting, another could be working to ameliorate agriculture. Empowered with

[Ω] https://github.com/tushar-semwal/ProjectTartarus
[▶] https://www.youtube.com/watch?v=VeryfhtT5Tk

the *hardware-in-the-loop* feature, *Tartarus* will not only be able to handle issues in forecasting and agriculture but also use the derived information to monitor and drive physical devices such as sensors and actuators, deployed in such scenarios. Subsequent chapters uses *Tartarus* as a tool for the implementation of proposed systems and mechanisms.

One significant issue that needs to be addressed is whether a CPS should be controlled centrally or otherwise. For instance, the CPS presented in this chapter could have been controlled centrally or in a decentralized and distributed manner. Most of the currently available implementations of CPS and IoT, inherently use a centralized cloud-based server that contributes to churning out the necessary intelligent decisions, leaving the associated leaf-devices (including robots) fairly underutilized. Though this centralized methodology, on first sight, seems to provide for an easy, manageable and rapid solution, they suffer from adverse limitations such as low scalability, robustness, and reliability, and at times even prove to be expensive. In contrast, with the number of connected devices on the rise, a decentralized and distributed solution could prove to be a more viable solution. In such a scenario each node generates and stores its own data within itself. Information is thus distributed in the various nodes forming the CPS. This calls for a paradigm that can generate pertinent decision and control information from the data distributed across a network. The decision on whether the components in a CPS should be controlled centrally or otherwise rests on the wisdom of the applications programmer. It is thus mandatory that a platform used to implement a CPS should provide such flexibility so as to facilitate the judicious use of centralized or decentralized control. The following chapter thus endeavors to present a mobile agent based approach to realize decentralized and distributed CPS/IoT systems.

"It is true that contemporary technology permits decentralization, it also permits centralization. It depends on how you use the technology."

Noam Chomsky (1928)
American linguist

3

Decentralizing a Cyber-Physical System

Research in VLSI technologies has been making progress in leaps and bounds resulting in a drastic fall in embedded hardware prices. Embedded devices including mobile phones are now capable of performing much more complex computations than their earlier counterparts thereby paving a way for the realization of computing ecosystems which are both ubiquitous and pervasive. Networking capabilities have added another dimension to the application of these devices and given birth to new paradigms and perspectives while realizing IoTs and CPSs. These physical devices generate huge amounts of data which are traditionally offloaded to a remote server which in turn performs all the relevant computations and decision making. Though such a centralized solution is straightforward and relatively easy to implement, it poses a threat to the privacy and integrity of personal data and also adds to the usage cost due to unnecessary upload and download of data. The use of a centralized system becomes highly sensitive in medical and military applications [173]. A framework that can facilitate data processing and decision making in a decentralized and distributed manner among the devices connected to a network, will greatly reduce the above practical issues and is thus the need of the day.

This chapter presents a mobile agent based approach for realizing Decentralized and Distributed CPS (*d*CPS) including IoT. The approach couples the local peer-to-peer communication with concepts from multi-agent abstraction to transform a centralized system into a fairly distributed system with no or partial centralization.

An indoor Location-Aware and Tracking Service (LATS) has been used as a CPS application to portray the use of mobile agent based mechanisms that share and process data locally within the participating nodes. These mechanisms thus form the crux of the work reported in this chapter. Agents populating the network monitor and track human beings in an indoor environment. Providing LATS is a challenging task in a dynamic environment [199]. Such scenarios call for queries that relate to *where* and *when* a person was or is in the area being monitored, *what* is the direction of the person's movement, etc. Firing queries to a database of related information stored centrally is fairly simple. However, if the person being tracked is in continuous motion, the database becomes dynamic which makes the task of querying, a complex one. The complexity increases if many are tracking multiple human beings. Things worsen further when the devices that track and store the data are numerous and have limited computational and storage resources. Data, in such scenarios, is both dynamic and distributed across a network. The problem compounds as new queries for information are fired asynchronously and concurrently. This chapter stresses the importance and significance of *d*CPS and introduces a mobile agent based approach to realize it. The approach also addresses the issues mentioned above and endeavors to overcome them.

3.1 Motivation

The advent of the Internet of Things (IoT) [10] has facilitated devices to be connected with ease and enhanced to communicate and share data. Gartner Inc. [60] has predicted that by 2020, the IoT will form the basis for most business processes and systems. It has also conjectured that by this year, more than 6.4 billion such devices will become connected. This drastic increase in connected devices is bound to revolutionize and greatly enhance Information and Communication Technologies (ICT) [161]. The Internet serves as an easy, reliable and accessible means for communication but is not without issues. Two of the major issues that crop up in the implementation of a typical IoT are security and the cost incurred in cellular communication. For applications such as a cab inquiry and booking system, which involves devices spread across an enormous geographic area, the use of the Internet can be

traded off with some aspects in security. This may not be true for critical areas such as in military applications, hospitals, industries, smart buildings, etc. where security could be a major concern. Current IoT architectures [74, 104] make use of cloud-based solutions for imparting services to the users. The integrity, safety, and insecurity of data stored in a cloud along with the associated services for sensitive domains like medical and industrial ones, remain matters of concern. The other issue is that in the conventional cloud-based IoT architecture, a device communicates through a central server supporting the cloud platform. This increases the cellular communication costs. A set of devices within a networked infrastructure can communicate locally and also perform computations thus, preventing a very large number of interactions with the cloud [186]. For scenarios such as an IoT for military or health care application, an *Intranet* based solution could perform effectively. Issues like security and communication expenses in an Intranet can be greatly contained. Another important issue is data privacy which is crucial in the case of medical hospitals, government and also for a consumer. Leakage of personal information and data ownership are at risk in a cloud-based centralized architecture.

In cloud-based systems, most of the data and intelligence churning activities are performed by a server hosted elsewhere in a centralized manner. For an Intranet-based solution, a framework that can facilitate this in a decentralized manner needs to be evolved. The devices participating in such an *Intranet of Things*, could include a range of connected embedded devices with their associated interfaces that connect them to the real *physical* world through sensors and actuators. The word "things" in an IoT refers to passive devices which seldom inherit any form of smartness within them. This is due to the fact that it is the cloud which is responsible for the intelligence and not the actual device. What is thus required to make an intelligent Intranet of Things, is a *cyber* counterpart which can induce and embed intelligence into these devices. Multi-Agent Systems (MAS) [201] can act and provide as a fitting solution for realizing embedded intelligence. If such agents are made to operate on top of each embedded device, they can make decisions autonomously at the lower levels, thus transforming a network of such devices into a smart Cyber-Physical System (CPS). Figure 3.1 depicts such a CPS wherein the core comprises the real *physical* world being sensed and controlled via the sensors and actuators.

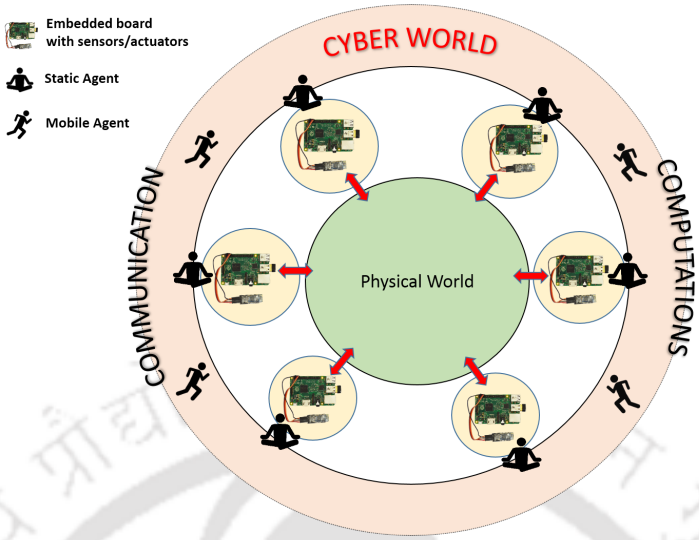


Figure 3.1: An Agent based Cyber-Physical System

The actual decision making and intelligence churning process is carried out by the agents (static and mobile) within the *cyber* world. These agents are programs that run on the connected embedded devices.

The concept of using agents in an Intranet of Things is very similar to the implementation of a *Fog Computing* environment [21]. The cloud is extended to the user side and constitutes a set of distributed and decentralized computing nodes which form the edge of the network. Such a concept has several advantages which include -

1. *Privacy*: Most of the cloud servers are owned by multinational corporations such as Amazon, Google, Microsoft, Cisco, etc. which continuously receive data from the user side. Leakage of personal information and data ownership becomes a critical issue when all of the user's data is collected for analytics purpose at the cloud [186]. A safer solution would be to have a local infrastructure on which the user has more control than the cloud server. This would allow local data filtering and computation before sending it over to the cloud. An agent based system could be a better solution for ensuring privacy.

2. *Cost*: Cloud services follow a "Pay-as-you-go" model which adds to the cost as the storage and network communication increases [21]. In a local computational infrastructure model, these costs can be reduced if the data collected

is filtered locally and the only pertinent information is sent to the cloud.

3. *Network Latency*: A cloud has inherent latency issues and thus may not be a viable solution for applications such as live video streaming in connected vehicles, real-time data analytics in smart grids [21], etc., all of which require a rapid response. An Intranet of Things that uses agents, on the contrary, can provide fast local computations thereby decreasing latency.

4. *Energy*: As already mentioned, agents in an Intranet of Things can filter the acquired data prior to sending it over to the cloud. Since this reduces communication overheads, it also reduces the energy consumed and consequently increases the battery life of the devices constituting the network [29].

The succeeding sections provide a background on earlier realized LATS applications followed by the description of the proposed approach.

3.2 Background

Since the work described herein uses LATS as an agent based application embedded on a Pi, a brief survey on the same is presented below.

Location-dependent services are part of a dynamic model where either the object or the observer or both can be mobile with respect to their geo-location [106]. Some of the classical approaches for tracking of a moving object include the use of Global Positioning System (GPS), Radio Frequency Identification (RFID), camera, etc. The most popular method of positioning is by using a GPS on board a mobile phone. This method is, however, effective mainly outdoors where the device can reach out to the satellites. Indoor localization using such GPS is unreliable due to the topology of the rooms and the erratic and low-intensity satellite signals received within. This calls for an efficient yet cost-effective solution to provide for a reliable indoor positioning and tracking system.

Catarinucci et al. [31] have proposed an IoT-aware architecture for smart healthcare. They have leveraged the use of combining UHF RFIDs [47] and WSNs [204] for deploying a healthcare system. Each patient has an RFID tag which transmits its data to an RFID receiver which in turn transmits the data to the associated

doctor. Since RFID tags are passive devices, the system uses minimum power and is thus quite efficient in terms of energy consumption. The major drawback is that for proper data transfer, the patient has to be in very close proximity to the RFID receiver.

Bluetooth Low Energy (BLE) technologies [70] can offer a far more superior solution than RFIDs. Yoshimura et al. [206] portray a system for analyzing the visitors' length of stay in an art museum through the use of non-invasive Bluetooth based monitoring. In their work, eight Bluetooth sensors were installed in the busiest locations at the Denon wing of the Louvre museum. The data on the number of visitors visiting these places was collected for a period of 5 months and then analyzed to get meaningful results. They have claimed that the use of non-invasive technologies (such as Bluetooth) allows them to gather honest results. This is so since visitors change their behaviors if they are aware of the fact that they are being tracked.

Early work in location-aware services by Wolfson et al. [199] describe a mechanism for tracking moving objects through the use of a database. They present a Database for Moving Objects (DOMINO) on top of an existing database which allows the database management system to predict the future location of the moving object. Every time the object in motion updates its location, its future location is also predicted.

Wolfson et al. [198] has also proposed a trajectory location management system to model the moving object. They highlight the critical issues associated with the point-location management model [198]. A point-location model does not provide facilities for interpolation or extrapolation of location data of the moving object and is not accurate.

In a trajectory location model, an estimate of the source and destination of a moving object is determined. This information is coupled with an electronic map and a trajectory is constructed based on the travel time information. In the real world, the relevant data is not always available at a centralized location. Wolfson et al. [198] conclude that their model needs to be improved to suit scenarios where data is available in a distributed form. The latter part of this chapter shows how LATS can be implemented when the data is distributed across a network of devices.

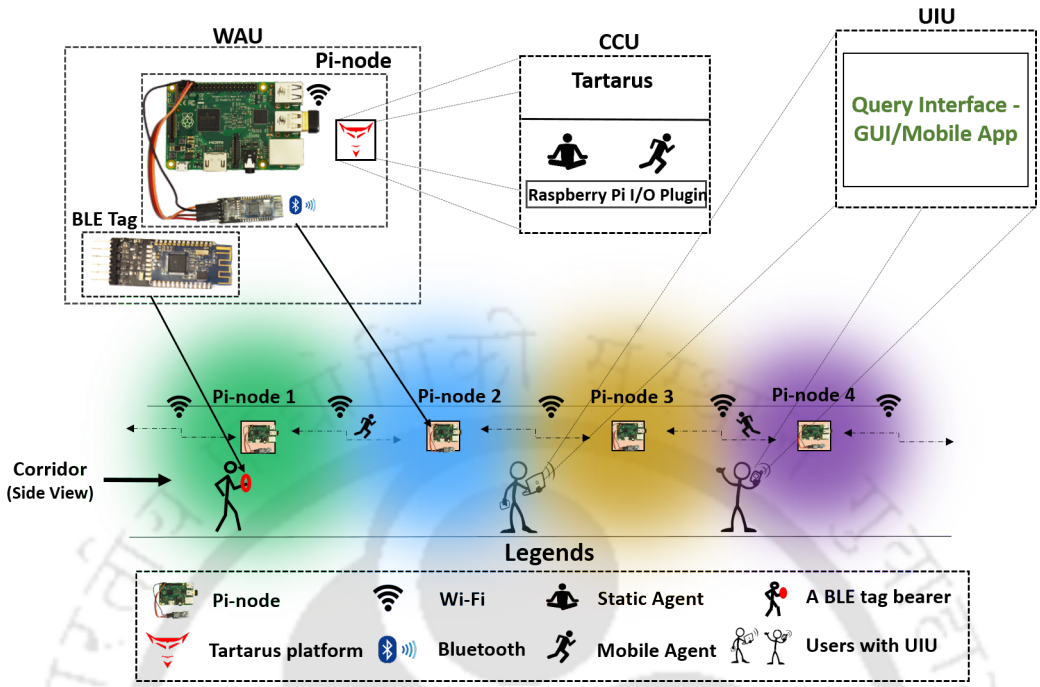


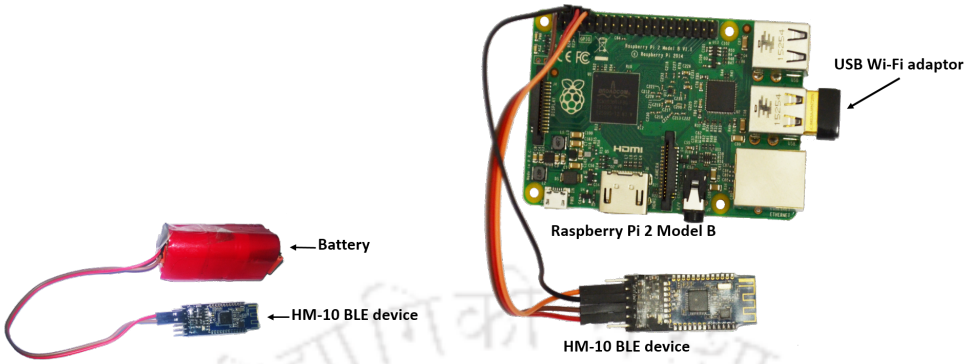
Figure 3.2: Decentralized and Distributed LATS

3.3 Decentralization using Mobile Agents

A complex system can be divided into subsystems, each controlled by an agent. This form of abstraction eases the designing and realization of systems. This section presents Location Aware and Tracking Service (LATS) as a use case and describes the detection mechanism and the main units that comprise the application. Finally, a mobile agent based algorithm to process the queries from the users is discussed.

3.3.1 Detection Mechanism

The lower portion of Figure 3.2 portrays the manner in which Pi-nodes have been deployed along the corridor. A Pi-node consists of a Pi interfaced to a BLE receiver and Wi-Fi adaptor. A Cyber Computing Unit comprising *Tartarus* and its associated plugins run on the Pi. Each Pi-node within the corridor is connected to its neighbor (s) through Wi-Fi. A person who is to be tracked (depicted as a stick figure with a red band on the wrist in the figure) needs to wear a BLE tag that emits beacons at a certain rate. This BLE tag along with the Pi-node forms a Wearable and Acquisition Unit (WAU).



(a) (b)

Figure 3.3: (a) A BLE tag (b) A Pi-node

Users who need to track a person(s) are provided with a User Interaction Unit (UIU) running on their respective computing machine. The functioning of the WAU, CCU, and UIU shown in the upper portion of Figure 3.2, has been detailed in the subsequent subsections. As can be seen in the lower portion of the figure, the corridor is divided into virtual zones (indicated by different colors) whose areas are preset based on the RSS values from the BLE tag received by the associated Pi-node.

When a person enters a zone within the corridor, the BLE receiver of the Pi-node within that zone detects his/her presence in that zone. As the person moves away from this zone and enters the neighboring one, the RSS in the new zone increases while in the former's decreases. This indicates the transition of the user from one zone to another. Eventually, when the RSS detected at the previous zone becomes minimum and that at the next zone becomes maximum, the system detects the presence of the person in the latter zone.

3.3.2 Wearable and Acquisition Unit (WAU)

This unit includes a wearable Bluetooth Low Energy (BLE) device (HM-10) which emits data packets in the form of beacons at preset intervals. These packets are received by a BLE receiver interfaced to a *Pi* via its on-board UART module. Fig 3.3a and 3.3b show a BLE tag (comprising a BLE device and a battery) as a wearable unit (configured as a beacon transmitter) and a Pi-node comprising a *Pi* interfaced

to a BLE receiver as the acquisition unit. The *Pi* also has a USB Wi-Fi adaptor. Each data packet transmitted by the wearable BLE device is 30 bytes long and contains five fields of information as given below:

1. Preamble: This read-only field is 9 bytes wide and contains the manufacturer's data.

2. Universally Unique Identifier (UUID): This field which is 16 bytes wide can be preset to contain the identity of the BLE device.

3. Major: This is a user writable field which helps in identifying a subset of such devices within a large group.

4. Minor: It is also a writable field which is used for specifying a subset of the Major field.

5. Tx Power: This field is a calibrated 2's complement value denoting the signal strength at 1 m from the device. This field is compared with the measured signal strength at the receiving end in order to ascertain the distance between the transmitter and receiver.

The BLE receiver extracts the information within these five fields and forwards it to the CCU.

3.3.3 Cyber Computing Unit (CCU)

A CPS is a tight coupling between the *physical* and the *cyber* worlds. The *Tartarus* platform serves the purpose of a *cyber* unit which runs on top of the *physical* unit (*Pi* in the present case). *Tartarus* comes with a plugin to access the peripherals on board the *Pi*.

A static agent named Database agent within a *Tartarus* instantiation running on a *Pi* fetches the beacon data from the buffer register within the BLE receiver via the UART [125] interface. The Database agent then stores the data in an SQL database along with the time-stamp on the memory card in the *Pi*. If a user remains within a zone for a long period, there will be a large accumulation of data, most of which could be redundant. To avoid this, beacon data is read always but stored only under some conditions.

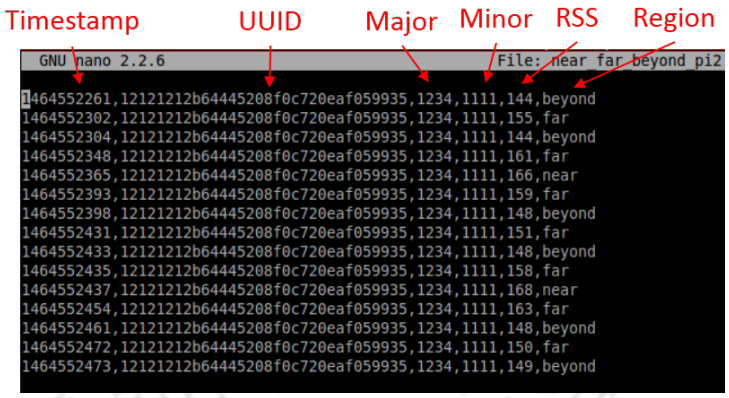


Figure 3.4: A sample snapshot of the part of the database maintained at a Pi-node

Thus, data is logged only when there is considerable change in the Received Signal Strength (RSS) of the beacon. Further, instead of making a decision based on the normally noisy RSS values, three regions — *Beyond*, *Far* and *Near* have been used to describe the position of a user within a zone. The three regions can be defined as follows:

(i) *Beyond*: When the RSS value is zero it means that the person is not detected and is beyond the concerned zone.

(ii) *Far*: This is a case when the person being tracked is far from the Pi-node. This is detected by a weak RSS value at the Pi-node of the concerned zone and would mean that the person wearing the BLE tag is in between 2m to 5m of the radial distance from the associated Pi-node.

(iii) *Near*: A strong RSS value indicates the person to be well within the range i.e. less than 2m in the present case.

Each SQL entry comprises a total of six fields of information — the Timestamp, UUID, Major, Minor, RSS, and Region. A sample snapshot of the data entered at a Pi-node is shown in Figure 3.4.

An entry is made to the SQL database only when the value of the sixth field changes in terms of the Region. For instance, if the sixth field changes from *beyond* to *near*, an entry is logged with the new Region. If the next consecutive entry is also *near*, then no entry to the database is made. Similarly, if the sixth field changes to

either *far* or *beyond*, an entry is made. It may be observed that from the database the information about the period of stay of a user in a particular region or zone can be easily computed. Further, a person may also be tracked as s/he moves from one zone to another. One may also easily infer as to exactly when s/he entered a zone, the amount of time spent within that zone and when s/he exited the same. Thus, as a person passes through a corridor comprising several such zones, the respective Pi-nodes keep track of the next zone to which the person has moved. This is done through the concept of a Motion Vector which has been described below.

3.3.4 Motion Vector

Let $Z = Z_{P_1}, Z_{P_2}, Z_{P_3}, ..., Z_{P_n}$ be a set of zones, where P_j represents the j^{th} Pi-node and n is the total number of Pi-nodes in the network (one per zone). A Motion Vector ($\overrightarrow{MV}$), describes the movement of a person wearing the BLE tag, from one zone to another and is given by,

$$\overrightarrow{MV} = Z_{P_a} \quad \rightarrow \quad Z_{P_b} \ ; \ a, b \ \epsilon \{1, 2, ..., n\}$$

Each Pi-node in a CCU, stores and updates two types of Motion Vectors — Motion Vector Forward ($\overrightarrow{MV_F}$) and Motion Vector Backward ($\overrightarrow{MV_B}$). When a person wearing the BLE tag moves from the *far* region to the *beyond* region of a certain zone, say Z_{P_x}, the corresponding Pi-node, P_x within that zone, sends a message to all its neighbouring Pi-nodes informing that the person bearing the specific UUID is now in the process of leaving its zone Z_{P_x}. If any of the neighbouring Pi-nodes, say P_y, detects this UUID within its zone, Z_{P_y}, it will acknowledge the presence of the person to the Pi-node, P_x. This causes the Pi-node, P_x to update its Motion Vector Forward, $\overrightarrow{MV}_F = Z_{P_x} \rightarrow Z_{P_y}$, against the associated person. Similarly, the Pi-node P_y updates its $\overrightarrow{MV}_B = Z_{P_x} \rightarrow Z_{P_y}$ and $\overrightarrow{MV}_F = Z_{P_y} \rightarrow Z_{P_y}$. The $\overrightarrow{MV}_F = Z_{P_y} \rightarrow Z_{P_y}$ represents a transition from Z_{P_y} to itself. This indicates that the user is currently in that zone and acts as a presence indicator. Table 3.1 shows the Motion Vectors at Pi-nodes P_x and P_y after a user transits from zone Z_{P_x} to zone Z_{P_y} (zone Z_{P_x} is assumed to be the very first entry zone). Here INFINITY represents that a user is not traceable at any of the zone and thus can be considered to be outside of the infrastructure where agent based LATS is deployed.

Table 3.1: Motion Vectors at Pi-nodes P_x and P_y after an inter-zonal transition

INFINITY →	**Zone** Z_{P_x} →	**Zone** Z_{P_y}
MVF	$Z_{P_x} \rightarrow Z_{P_y}$	$Z_{P_y} \rightarrow Z_{P_y}$
MVB	INFINITY $\rightarrow Z_{P_x}$	$Z_{P_x} \rightarrow Z_{P_y}$

The UUID and Major-Minor values allow for classifying a particular BLE device wearer. For example, one can track the faculty members and students in an academic department using the content within these fields. This makes the database contain finer details and thus allow a range of queries to be satisfied. As can be seen, the Database agent thus manages the database and the Motion Vectors within the associated Pi-node.

3.3.5 User Interaction Unit (UIU)

This unit provides an interface for the users to access the tracking service of the agent based LATS. The interface could be in the form of a mobile app or a Graphical User Interface (GUI) running on a *Pi*, a laptop or a PC, all connected to the same network as that of WAU. A *Tartarus* instantiation running on a *Pi* and a laptop has been used to fire queries to the system. To fire a query, a user can release an agent from the same *Tartarus* instantiation. The UIU was populated with mobile agent programs for a set of queries. Since *Tartarus* facilitates agent programming [163], users and developers could write custom mobile agent programs for a range of queries and add them to the UIU to improve its functionality. The code for the agent of the associated query shall be already available with the *Tartarus* as part of the UIU.

Querying

A mobile agent serves the purpose of query processing. Since the databases are distributed over the various Pi-nodes, these mobile agents move from one such node to another and search and retrieve the information that can satisfy the user's query. The mobile agent then aggregates the relevant data concerning the person being tracked and delivers it to the UIU for processing and rendering. A user wearing the BLE device or a third party may wish to query this LATS to gather a range of

information which include -

1. *Where am I?*: Such a query invariably emanates from a person who is lost within the building or does not know how to move around or needs to convey his/her bearings to someone else. Under such conditions, the user can fire an SQL query packaged in a mobile agent to the nearest one-hop neighboring Pi-node. Once the mobile agent enters this Pi-node, it executes its code and eventually lands up in the Pi-node of the zone in which the person is currently present. The agent then retrieves the location information stored *a priori* within this Pi-node and provides it to the user. A segment of the relevant mobile agent code is presented in Figure 3.5.

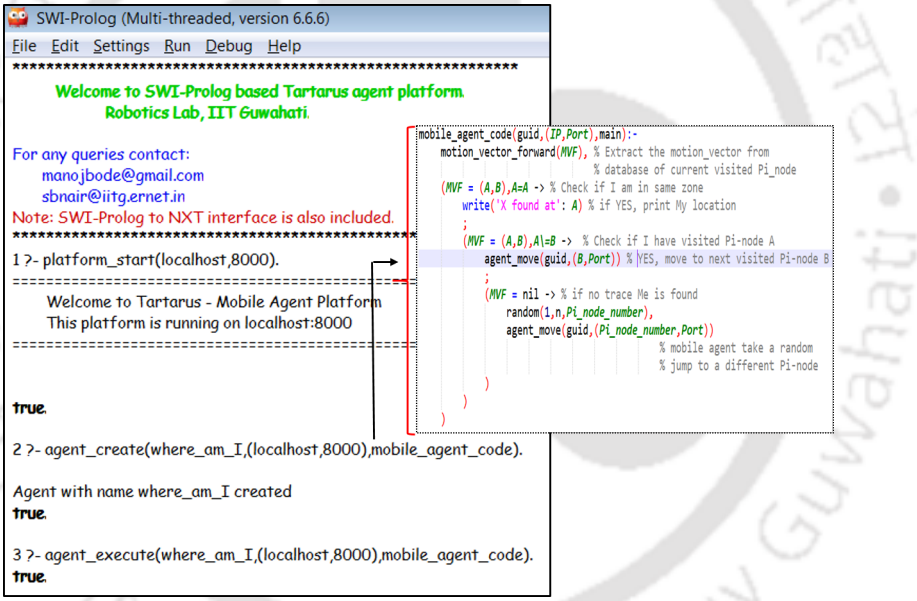


Figure 3.5: Mobile Agent code snippet for the query, *Where am I?*

2. *Where is X?*: A query of this kind is required for a person to know whether X is within the building under consideration and if so, where. This agent based LATS allows for a non-intrusive mechanism to find the location of X. The user packs this query into a mobile agent and transmits it onto the *Tartarus* platform of the closest *Pi*, the one within the zone s/he is in currently. On reaching this *Pi*, the mobile agent scans the database within it to find whether X is/was in this zone. (i) If it discovers that X is within a particular zone currently, it retrieves the location information from the Pi-node and backtracks its path to the user's

system and provides the information on X. (ii) If the agent finds a Motion Vector Forward for X in that zone, then it uses the vector to find the next zone visited by X and migrates to the concerned Pi-node of this zone. It continues to do so till it eventually lands in a Pi-node of a zone where X is currently present. On reaching this, it retrieves the relevant information and retraces its path back to the user's system to provide the information on X. In case X has left the place, the Motion Vector Forward within the Pi-node in the zone where X was last present, will point to INFINITY. The agent would then assume that X is no more in the area and report accordingly to the user. (iii) If no trace of X is found in the database, the mobile agent continues its migration along the Pi-nodes in a conscientious manner [128] [Appendix .1] till it eventually finds that X has been within the zone of some *Pi* or left the place. It may be noted that a user who wishes to know the bearings of another can alter his query to extract a range of information on the person being tracked.

3. *Trace(X)*: This query will provide a list of locations associated with all those zones which X visited in order. The query can again be packed into a mobile agent and sent to the network of Pi-nodes to search the individual databases and retrieve the list. A mobile agent algorithm to trace the path of a BLE tag bearer is shown in Algorithm 1 and an example of mobile agent routing for the same is described in Appendix .2.

3.4 Experiments and Results

Experiments conducted involved users who were asked to move from one zone to another. In addition, experiments involving the acquisition of raw BLE data were also conducted to get more insights into the behavior of the device. Subsequent sections discuss the experiments conducted to acquire and store tracking information which in turn are used and processed by mobile agents to satisfy user queries.

3.4.1 Data Acquisition

A BLE tag bearer was asked to move back and forth across the radial axis of a Pi-node. The actual RSS values received at the Pi-node nominally ranged from -40

Algorithm 1 An algorithm performed by an agent to trace the path of a BLE tag bearer

Result: Path followed by X ; `// X is a person whose path is to be traced`
Stack S = Empty
Queue Q = Empty
while *Path followed by X is not retrieved by Agent ;* `// Agent continues the search until the total path traced by X is found`
do
  MVF(X) = Motion Vector Forward of X at visited Pi-node, P_v
  MVB(X) = Motion Vector Backward of X at visited Pi-node, P_v
  if *(MVF(X) = Nil) OR (MVB(X) = Nil) ;* `// If trace is not found by the agent`
  then
    Select a neighbouring node at random and migrate to it ; `// Agent migrates to another node`
  else
    if *(MVF(X) =* $Z_{P_v} \rightarrow Z_{P_v}$*) OR (MVF(X) =* $Z_{P_v} \rightarrow INFINITY$*) ;* `// If agent has found last node visited by X`
    then
      insertStack(S , v) ; `// Agent inserts the node ID into its internal stack`
      while *X's starting position is not found* **do**
        Use MVB of each earlier visited Pi-nodes to trace back the path
        insertStack(S , Pi-nodes visited before v)
      end
      Path followed by X = *getStack*(S)
      return Path followed by X ; `// Agent returns the path followed by X`
    else if *(MVF(X) =* $Z_{P_v} \rightarrow Z_{P_w}$*);* `// If Agent finds the intermediate node visited by X`
    then
      while *X's starting position is not found* **do**
        Use MVB(X) of each earlier visited Pi-nodes to find the start position
      end
      while *X's last/current position is not found* **do**
        Use MVF(X) of each next visited Pi-nodes to reach the last/current position
        insertQueue(Q , Pi-nodes visited from the start position)
      end
      Path followed by X = *getQueue*(Q)
      return Path followed by X
  end
end

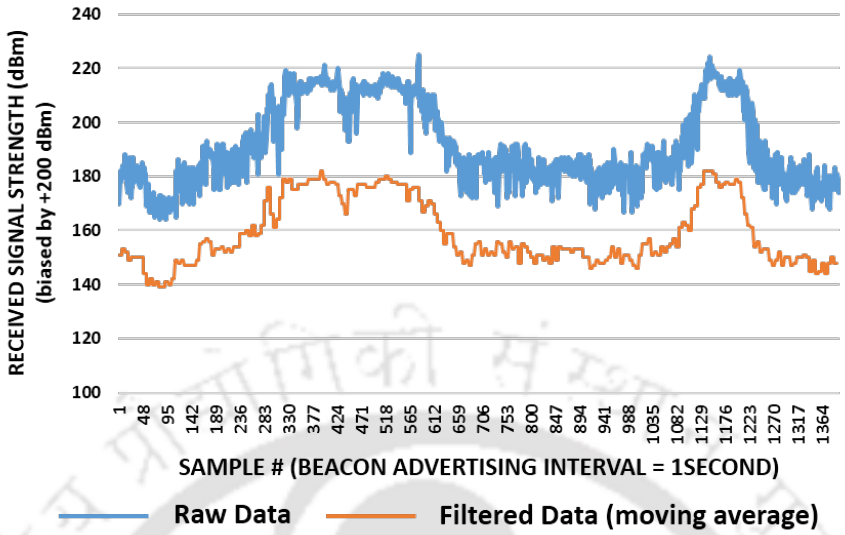


Figure 3.6: BLE raw and filtered data

dBm to +20 dBm[*]. In order to portray the graph in the 1^{st} quadrant for clarity, the values are biased by adding +200 dBm to each of the data points. Figure 3.6 shows the biased raw and filtered BLE data taken over a certain number of sample points. As expected, a trend similar to a sinusoidal wave can be observed in the figure thereby validating the performance of the BLE. The RSS received from a BLE device is subject to noise due to various reasons such as multi-path propagation, signal absorption, signal interference, etc. Based on the analysis by Faragher et al. [51], different filters may be applied to the raw BLE data. After a series of empirical experimentation on data filtering, it was found that a moving average filter with a window size of 6 samples at a time, provides satisfactory results. Analysis revealed a rule of thumb that indicates that as the window size increases, the filtered data becomes more stable. However, this may take more time to produce tracking results. Hence, a compromise needs to be made in terms of accuracy and reactiveness of the deployed tracking system.

An experiment wherein each user was made to wear a BLE tag and asked to move from one zone to another in order to obtain their respective tracking profiles was performed. The experiment was conducted at the ground floor of the Department of Computer Science and Engineering block of the Indian Institute of Tech-

[*]Depending upon the manufacturer, the actual raw RSS values for a BLE device may range from -80 dbm to +25 dbm.

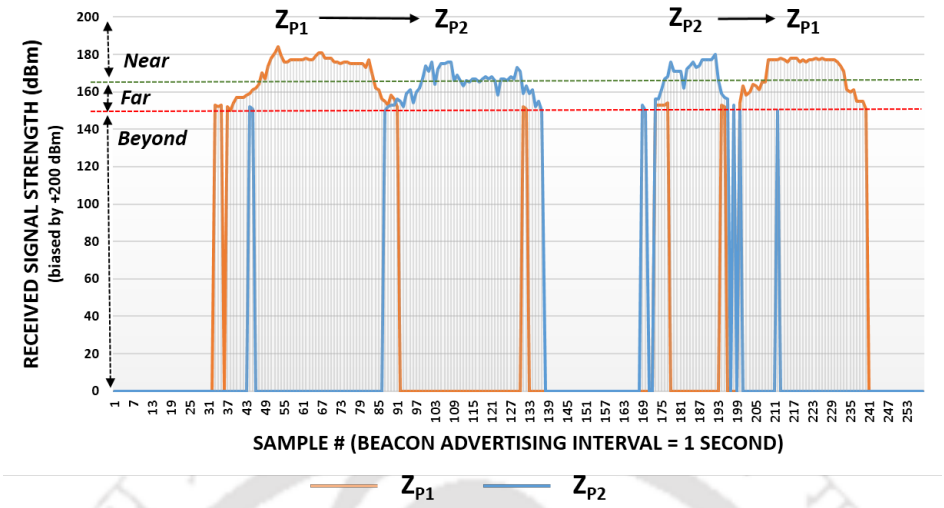


Figure 3.7: Graph showing Inter-Zonal movement for a single BLE tag bearer

nology Guwahati. Since it is logical to assume that the profile generated between two consecutive zones can be extended to other such multiple consecutive zones, the inter-zonal movement for a single user is first described herein. The results portraying a user's movement within two zones, Z_{P_1} and Z_{P_2} along with the three regions, *beyond*, *far* and *near* categorized on the basis of RSS is shown is shown in Figure 3.7. As in Figure 3.6, the Y-axis denotes the filtered and biased RSS values from the BLE receiver at the Pi-node while the X-axis indicates the sampling index ranging from 1 to the number of samples taken at a sampling rate of 1s.

The graph shows two different colored series each corresponding to the RSS at Pi-node within a particular zone. The orange colored series denotes the same for Zone 1 (Z_{P_1}) while the blue colored series indicates that for Zone 2 (Z_{P_2}). Initially, the user is outside the coverage area of both Z_{P_1} and Z_{P_2}. As seen from the Figure 3.7, when the user starts moving towards Z_{P_1}, the RSS (orange color) increases from sample number 41 onwards and attains a maximum when the user is nearest to the associated Pi-node of Z_{P_1}. It then starts to decrease as the user moves away from the Pi-node in Z_{P_1}. When the user enters the periphery of Z_{P_2}, where both the zones overlap to an extent, an increase in the RSS at Z_{P_2} is observed with a corresponding decrease of the same at Z_{P_1}. A similar pattern is exhibited when the user moves away from Z_{P_2} to the next neighboring zone. A similar experiment that was conducted when the person moved from Z_{P_2} to Z_{P_1} is recorded with Z_{P_2} as

entrance zone and Z_{P_1} as the exit zone. The relevant plots are depicted in the latter part of Figure 3.7. It may be observed that there are some random spikes generated due to noise and reflections. Since these peaks cross from the *beyond* region to the *far* region and again go back within a second, the corresponding vectors are not stored in the database.

3.4.2 Query Processing

The post data acquisition step involves satisfying queries fired from the user side. In order to compare the results of query processing using the conventional cloud-based method and the proposed *d*CPS approach, experiments were conducted for the two scenarios described in this section. Since testing on a real system would mean the requirement of a large number of *Pi*s, for both the experiments a multi-floor building was emulated using a 50-node overlay network formed over a network of 4 Pi-nodes and 2 PCs. Each PC hosted 23 emulated Pi-nodes created using *Tartarus*. The BLE tag bearers who move around in the building and need to be tracked, were emulated by mobile agents that move from one node to another. A total of 10 BLE tag bearer were introduced into the network, out of which 6 were made to move randomly within the building. The remaining 4, designated as Head, Professor, Janitor, and Guard were programmed to have predefined movements. In addition, a separate dedicated server acted as the Cloud for both the systems. Figure 3.8 portrays the conceptual layout of the network deployed in a building. For the conventional cloud-based method, the Pi-nodes may or may not be connected to one another. For the proposed mobile agent based approach (as shown in Figure 3.8), these connections are mandatory since there need to be paths for the mobile agents to migrate.

3.4.3 Scenario 1: Conventional Cloud approach

In this approach, every Pi-node was capable of directly communicating with the Cloud. As the BLE tag bearers (mobile agents) move around the building (network), all pertinent data within the Pi-node (such as Timestamp, UUID, Motion Vectors etc.) are directly sent to the Cloud. This is done by each of the Pi-nodes as and when new data is generated within them. Thus, all the data acquired and generated at the Pi-nodes is stored and managed at the cloud. All queries in this scenario are

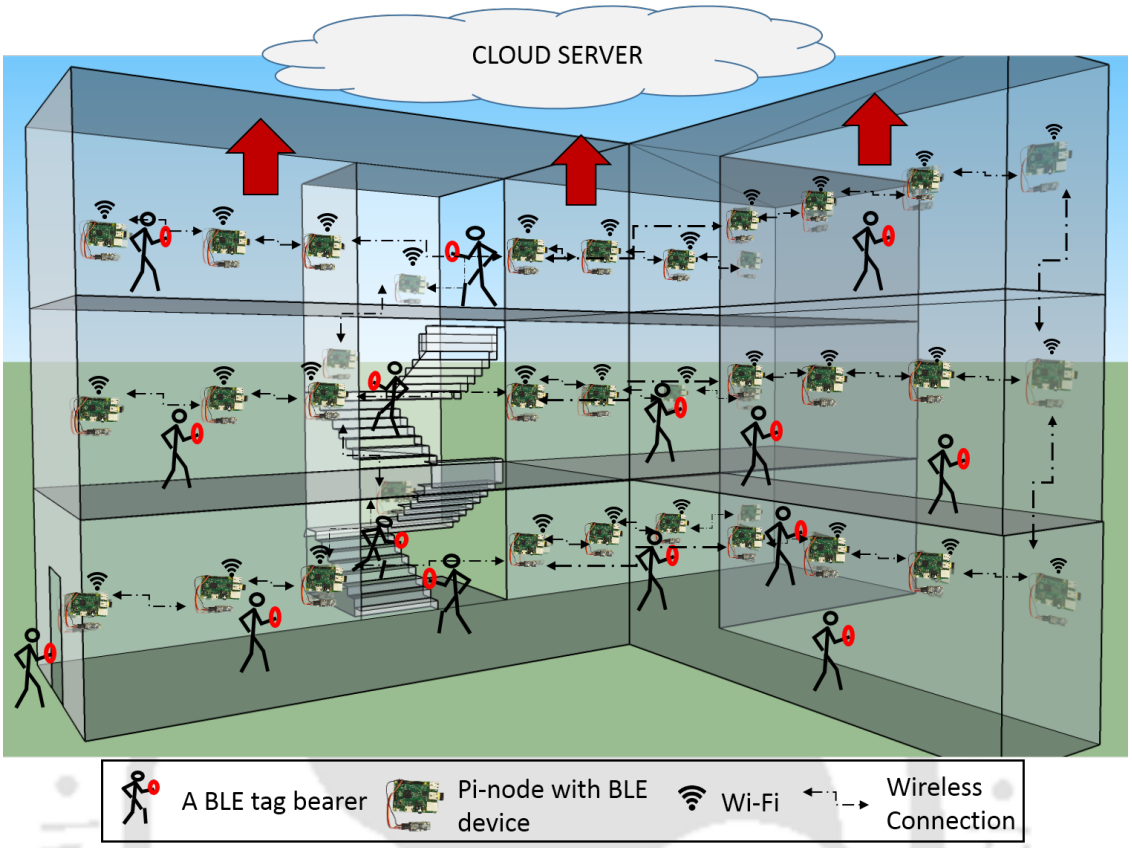


Figure 3.8: System deployed in a multi-floor building

directly sent to the cloud, which are in turn processed at the cloud and returned to the concerned user.

3.4.4 Scenario 2: Proposed *d*CPS approach

In this scenario, a user fires a query in the form of a program within a mobile agent via the UIU. This agent then knits through the connected Pi-nodes in the network, performs the concerned task(s) and processes the data within these nodes thereby processing the query. While doing so, it also sends the acquired data at each node to the cloud. It may be noted that in this case the cloud is updated only with the relevant information pertaining to the query. Unlike the previous centralized scenario, the cloud connectivity is made only from those Pi-nodes where the mobile agent finds query-related information. This drastically reduces data traffic between the networked devices and the cloud.

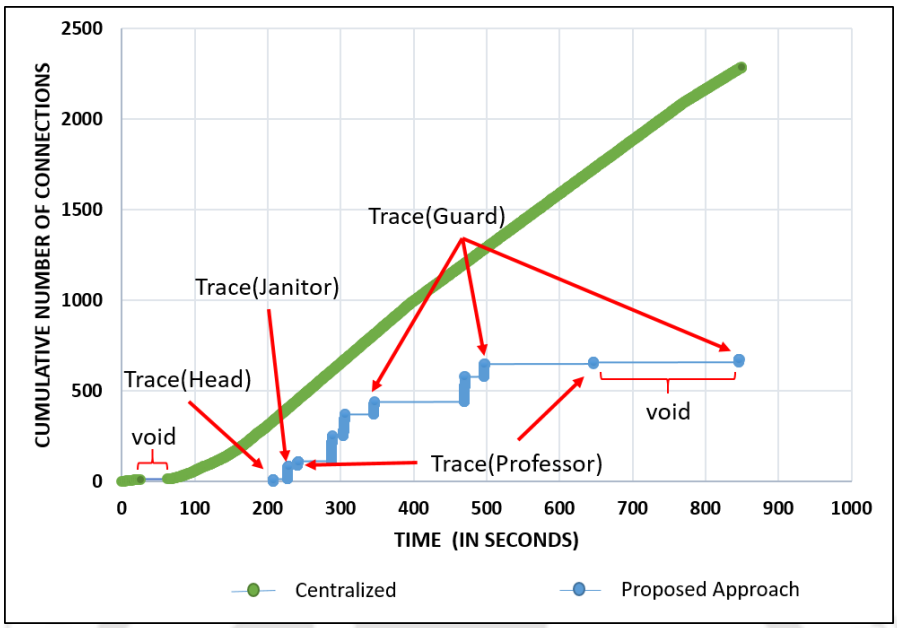


Figure 3.9: Traffic flow for centralized cloud based and mobile agent based systems

3.4.5 Comparison of Scenario 1 with Scenario 2

Experiments were performed where queries were fired by the user in both the centralized cloud-based and the mobile agent scenarios. Data transfer cost in terms of the number of times the Pi-nodes connect to the cloud was logged in both these cases. Figure 3.9 shows the cumulative number of connections made between the Pi-nodes and the cloud server for a set of Trace(X) queries, where X is the person being tracked.

In case of a centralized cloud-based approach, one can infer that the cumulative number of connections made to the cloud increases steadily with time. As mentioned earlier, this is because, for every new data generated at a Pi-node, a connection is made to the cloud.

On the contrary, in the case of the proposed agent based approach, such connections are made to the cloud only when information found is relevant to the query fired. The cumulative number of connections made by the mobile agents during the execution of the queries Trace(Head), Trace(Janitor), Trace(Guard) and Trace(Professor) are shown in the figure. These numbers are far lower than that for the centralized scenario clearly indicating the viability of the proposed approach.

The horizontal flat portions termed as voids in the curves, denote the absence of any connections made to the cloud. Such portions could occur in the centralized scenario when the BLE tag bearers are stationary i.e when no new data is generated at the Pi-nodes. For the *d*CPS scenarios too, such voids could occur provided no queries are fired.

3.5 Chapter Summary

This chapter introduces the concepts of decentralized and distributed computing in CPS and IoT based systems and describes how they can be realized using Multi-Agent paradigms. An agent based Location-Aware and Tracking Service (LATS) application has also been described to bring out the flexibility and versatility of using mobile agents for information sharing and processing on a distributed network of nodes. The system can track people wearing a BLE device indoors with fair reliability and accuracy and also provide answers to a range of queries as regards to the person being tracked. Experimental results reveal that the proposed approach performs better than the conventional cloud-based method. In addition, the use of mobile agents allows multiple queries to be fired using multiple agents concurrently. Queries need not be pre-programmed or preset. Since mobile agents can be released even during runtime, these queries can be fired *on-the-fly*. The use of agents can thus make a network of things smarter and flexible unlike those that do not use agents. Since agents can be created and released even during run-time [163], the system can be scaled, upgraded and programmed to be adaptable.

Centralized systems provide superior control to the designer but are susceptible to single-point failures. In contrast, decentralizing and distributing the decision making process and computing among different nodes makes the system more robust and scalable. However, pure decentralization may not be desirable as it completely removes administrator authorities which may be needed at times. This chapter favors the same viewpoint and opens up avenues for hybrid architectures where the lower layers manage in a decentralized manner and transfer only the necessary information flow to the upper centralized layers. This type of mixed system leverages the best of both the designs.

Though this chapter attempted to lay down the foundations for the use of decentralization in CPS and IoT systems, it must be mentioned that the lack of a central controlling entity does make their realization a challenging task. The same is true for loosely coupled cyber controlled physical systems. Some of the challenging issues include synchronization [87], task-allocation [91] and population control of mobile agents [65] most of which have been discussed in [86] for networked robotic systems. This thesis tries to cater to a broader domain which includes a CPS of robots, embedded systems, vehicles, and devices, all of which have their respective sets of challenges to be addressed. The absence of a central authority for control would naturally mean that each entity comprising the network are peers of others. In a CPS of robots when a resource needs to be used by more than one entity at the same time, it is essential to handle Mutual Exclusion, lest there will be a clash. Implementing mutual exclusion in decentralized environments is a difficult problem especially when some entities (such as robots) take different execution times for the same task/job due to their varying internal conditions (battery charge, wear and tear, etc.). Talking about decentralization without addressing mutual exclusion is thus meaningless. The next chapter describes a novel mechanism to solve this issue by ordering the execution of tasks in the form of an adaptive pipeline of robots which is devoid of a global clock.

"Research is what I'm doing when I don't know what I'm doing."

Wernher von Braun (1912 – 1977)
Aerospace Engineer

4

Ensuring Mutual Exclusion and Ordered Task Executions

Ordering becomes necessary to mitigate or completely avoid chaos. The disorder could be in the form of independent robots or devices vying for a shared resource. If the resources are to be shared by the robots within a CPS for the completion of their assigned tasks, then a mechanism for mutual exclusion of shared resources becomes mandatory in order to avoid contentions and deadlocks. Tasks involving shared resources are common in the real world. For instance, a ticket counter where people wait in a queue is a typical example of a shared resource. In the realm of Multi-Robot System (MRS), a sole battery-charging terminal where at an instant only one robot can plug-in and charge itself forms an example where mutual exclusion needs to be exercised. Thus ordering the execution of allocated tasks during run-time becomes crucial. This is so because, in the real world, there can be several physical tasks that use shared resources and are required to be executed concurrently. Ordering such executions requires a careful understanding of the available resources needed to complete a task within a CPS.

The mutual exclusion problem in a CPS can be fairly solved using a centralized control system. To order the task executions, this central controller can monitor and communicate with all the robots regarding their respective turns to gain access to the shared resource. Though simple and straightforward, this solution could drastically

load the central controller with heavy computational and communication overheads. This is so because the central authority has to communicate with several robots simultaneously and take decisions globally. Further, any change to this CPS of robots in the form of the addition of new tasks or robots would mean bringing down the central server.

This chapter formulates the problem of ordering the execution of sequential, independent and interdependent tasks to be executed by multiple mobile robots within a CPS and proposes a mechanism to solve the same. An agent based distributed approach has been formulated to ensure Mutual Exclusion of Resources (MER) among multiple robots connected to form a dynamic network. A sequence of topologically ordered and interdependent tasks that involves shared resources, forces their execution in the form of a pipeline. Since the number of mobile robots available to execute a set of tasks could vary, this chapter tries to portray these robots as a pipeline wherein the number of processing units could vary during runtime. A conventional pipelined computing architecture requires a clock in order to synchronize and allocate proper time slots for the execution of processes. However, in a real-robot scenario, the execution times for the various tasks performed by a set of robots may vary over time due to several issues such as communication delays, wear and tear, etc. If a pipeline needs to cater to such varying times required for the executions, it should possess an inherently adaptive clocking mechanism so as to compensate for such variations. The proposed mechanism caters to a decentralized and distributed CPS (*d*CPS) comprising nodes such as computers, robots, and sensor nodes, and uses mobile software agents that knit through them to aid the execution of the various tasks while also ensuring mutual exclusion of shared resources. The computations, communications and control, are achieved through these mobile agents. The robots perform physical execution of the tasks in an asynchronous and pipelined manner without the use of a global clock. The mechanism also features addition and deletion of tasks as well as the insertion and removal of robots facilitating *On-The-Fly Programming* of the CPS. The major contributions of this chapter towards the Task Execution Ordering Problem are:

1. A mobile agent based decentralized distributed mechanism for ordering multi-robot task executions.

2. A solution for the MER problem among multiple robots within a CPS.
3. Validation of the proposed mechanism through emulation.
4. Real world implementation of the proposed mechanism in a Warehouse Management System (WMS) type scenario.

4.1 Motivation

While robotic applications are fast making inroads into a plethora of automated systems, the tight coupling between the application and the robotic hardware seem to deter both their scalability and flexibility. The need of the day is to transform such automated systems into ones that are malleable and accessible over a network. Through this transformation, a fair amount of generic nature can be embedded within such systems, thereby allowing for changes to be made in the patterns or nature of executions of the tasks performed. This flexibility can be realized only if we facilitate networking among all the entities within these systems. Networking can allow the entities to communicate with one another and resolve several issues that crop up during runtime. If the entities are mobile, the network becomes dynamic and makes one-to-one communication, a much-disorganized task. A centralized approach for controlling the entities may perform well but makes the system rigid, expensive and hardly scalable. On the contrary, a decentralized and distributed control mechanism coupled with a mobile computing environment can empower these systems with autonomy, flexibility, and scalability. Such automated scenarios can be viewed to be made up of two primary components – a *cyber* component that caters to both computing and networking of the entities and a set of *physical* processes which are executed in the real world by a set of robots using percepts received from either on-board sensors or sensor nodes. Considering the fact that the physical processes are initiated, linked and to some extent controlled by the cyber component, this type of system can be categorized as a Cyber-Physical System (CPS) [11]. Hence, a networked Multi-Robot System (MRS) coupled with a mobile computing environment can provide a fitting framework for a CPS.

Research in MRS has mostly been focused broadly on two main areas viz. *task allocation* and *task partitioning*. In the former [61], tasks are assigned to the ap-

propriate participating entities (robots) in such a way that the desired performance level can be achieved with complete utilization of available resources. The latter, on the other hand, is the process by which a task is divided into a set of subtasks so as to reduce the complexity of its execution [150]. Apart from these, there is also a third objective crucial to an MRS based CPS viz. that of *task execution* which is grossly ignored in MRS specifications. Task execution is an inherent objective (usually defined by the user) that always commences after task allocation or partitioning. While the allocation and partitioning are merely planning models, task execution adheres to the actual implementation which validates the assignments of the tasks. Hence, both task allocation and task partitioning are dependent on task execution without which a task cannot be said completed. Early work on Multi-Robot Task Allocation (MRTA) by Parker [141] describes an architecture where fault tolerance was incorporated in a heterogeneous set of robots for carrying out different tasks. They demonstrated dynamic task allocation (a subclass of task allocation) within an MRS. A formal analysis of the problems faced in MRTA has been presented in [62]. Botelho and Alami [23] describe a technique for allocation and reallocation of tasks. In their work, each robot is provided with details of its own plan. A robot is allowed to make changes in its plan depending upon its capabilities as also those of the other robots. The use of auctioning techniques based on the dynamics of a market has been proposed by Dias and Stentz [46], where the robots are assigned tasks through negotiations with their peers in a distributed manner. Khaluf and Rammig [103] narrow down the scope of task allocation to time-constrained tasks. However, they have ignored the complexities of real-world task execution and provided only the simulation results.

In the domain of distributed computing, the Mutual Exclusion of Shared Resources (MER) is referred to as a classical benchmark problem to resolve resource contentions [151]. MER is required when different nodes need to access a shared resource at the same time, lest a race condition [151] occur. A CPS constituting mobile networked robots could be looked upon as a Mobile Ad-hoc Network (MANET). The problem of MER becomes more complicated in the context of MANETs [14] wherein mobile nodes move in a disorganized manner leading to dynamism in the communication topology. In addition, MANETs are constrained with limited band-

width, low power usage, low computations capabilities, dynamic topology, etc. [53] that increases the complexity of the MER problem as compared to their static counterparts. Solutions to the MER problem in distributed and dynamic networks can be broadly divided into two categories [8] — *token* based and *permission* based. In the token-based approach, a node with a unique token can access the shared resources while others have to wait for the arrival of the token. On the other hand, in the permission-based approach, a node can get access to a shared resource if it can get permissions from all other nodes in the network by exchanging messages. Since, in this approach, a node sends a request for getting access through messages to all the connected nodes, it consumes bandwidth and thereby introduces network latency. Although many variants of the MER problem have been proposed [32, 76, 27, 9], an adaptive and scalable solution in the context of a CPS, wherein the entities performing tasks need to share physical resources in the real world, has still not been proposed. Wu et al. [203], have modeled the problem of mutual exclusion of traffic intersections as a variant of the classical mutex problem. Vehicles compete to get access to the traffic intersection by exchanging messages. A vehicle passes through the intersection when it receives permissions from other vehicles involved in the competition. Their approach, however, uses multiple messages which lead to communication overheads and network latency. Minimizing such overheads while ensuring MER is crucial for the performance of a CPS.

Depending upon their nature, tasks can be divided into two types - (i) Independent and (ii) Sequential. Independent tasks can be executed in isolation and thus do not in any way rely on other tasks. By sequential, it mean that these tasks follow a topological order such that a task say, T_{i+1} is executed only if the execution of the preceding task T_i is completed. In a typical computing environment, when a program includes both concurrent and sequentially executable instructions or methods, the associated compiler separates the independent ones from the sequential ones. Based on the program, it assigns the independent ones to individual cores within a multi-core processor to maximize parallelism. The number of such cores which could be looked upon as independent processing units, naturally do not change. On the contrary in a real-world multi-mobile-robot scenario where robots are synonymous to such computing cores, this may not be the case. The number of robots available

to perform a set of tasks may vary over time. Such variation could be due to the fact that some robots may need to be charged while others could have malfunctioned for some reason. Their number could also increase if more robots are deployed into the scenario. A precompiling procedure to initially allocate sequential and independent tasks to a set of robots, as in a typical multi-core computing environment could be disastrous.

Further, in the physical world, tasks could be *interdependent* by virtue of the fact that they require both robots and resources to get executed. For instance, consider the case where robot R_1 is to execute a task T_1 using a resource Ψ_1 while robot R_2 is to execute task T_2 using the same resource Ψ_1. In this scenario, assuming T_1 and T_2 to be independent tasks, it can be observed that though both robots R_1 and R_2 are free to execute the two tasks, the non-availability of Ψ_1 concurrently to both R_1 and R_2 creates a bottleneck. One of them has to wait for the other to free the resource Ψ_1 forcing T_1 and T_2 to be executed sequentially executed. It may be noted here that, independent tasks may also suffer from similar bottlenecks when they require the same resource. Under such conditions, this *resource dependency* forces these independent tasks to be executed sequentially. One can thus conclude that a technique that can handle the ordering of all types of tasks *on-the-fly* while also catering and effectively utilizing the varying number of mobile processing units, forms a *sine qua non* for CPSs comprising mobile robots. The algorithm proposed in this chapter is novel in the sense that:

1. Intelligent messages in the form of mobile agents have been used to solve the problem of mutual exclusion while executing tasks in a multi-robot distributed environment. Conventional distributed scenarios as in Wu et al. [203], use message broadcasts to share information and ensure mutual exclusion of shared resources. Message broadcasting drastically increases the communication cost [203] and can clutter a network. The proposed mechanism have used a conscientious agent migration strategy [128] which has least inter-node communication cost [69] as compared to other agent based approaches such as CLInG [162], EVAP [37] and Random-walk with cloning [58].

2. Synchronization in distributed settings is a major challenge and is tradition-

ally achieved by using a single node (or a subset of nodes) which provide for clocking. This poses issues of reliability when such nodes fail. In the domain of robotics, the problem of synchronization deteriorates since the time required to execute a given task by a robot can vary due to several environmental factors.

The following sections describe the constituents and system specifications of the proposed approach. The Task Execution Ordering Problem (TEOP) among multiple robots and the inherent objectives for realizing the CPS are discussed in the subsequent sections.

4.2 Work in Brief

Execution of tasks in a *d*CPS wherein the entities such as robots and sensors nodes are independent, is a challenging affair. The situation becomes complex when the resources required for the execution of the tasks are shared among the robots. This calls for a mechanism to provide Mutual Exclusion (MUTEX). One simple solution is ordering of task executions in the form of a pipeline. A pipeline is made up of processing units which executes the atomic commands provided to them. Multiple pipelines could be used to provide concurrency. A traditional pipeline makes use of a global clock to provide synchronization of task executions. However, the assumption is that the task execution will always take the same amount of time.

This work is inspired by the pipeline architecture where the processing units are the robots and the code for the executions of tasks are delivered to them through mobile agents. In a real-world, the assumption that the tasks are atomic and take the same time for execution may not hold. Due to the difference in the designs of robots, environmental conditions and other forms of noise, the execution time will always differ. Thus, this demands for the pipeline to be adaptive and flexible.

The work presented herein uses mobile agents to make the robots adapt to variations in task executions. Each mobile agent carries information and instructions to execute the tasks. The robots are the processing units waiting for the suitable mobile agent to arrive and deliver the instruction sets required for the executions of the tasks. The problem formulation and how it is solved in the real world are

discussed in the subsequent sections.

4.3 Preliminaries

This section presents the entities and characteristics that make up the CPS used herein followed by a formal description of the problem at hand. Mechanisms to ensure MER when the tasks within a CPS need to be executed in a sequential manner are discussed further. The manner in which the tasks can be altered, added or removed *on-the-fly* in/from the sequence is also be illustrated.

4.3.1 Constituents of the proposed CPS

A CPS is an amalgam of both the cyber and the physical worlds where the term *cyber* comprises computations, communications and control while the term *physical* comprises interactions with the real world [168]. The proposed CPS is composed of heterogeneous entities such as a Multi-Robot System (MRS), mobile agents, sensors and computer nodes. Mobile agents form the cyber entities which carry out computations, manages all the communications and control the dynamics of the MRS. The interaction of robots with external surroundings where robots execute the sequential tasks forms the physical component. Following are the basic constituents of the proposed CPS under consideration:

1. Nodes: A node refers to any device that is capable of computations and communications and hosts an agent framework. It can be an embedded system, a personal computer, a robot or even a sensor node. Nodes are connected to each other to form a network $\boldsymbol{W}$.

2. Network: A network $\boldsymbol{W}$ is a dynamic wireless Mobile Ad-hoc Network (MANET) wherein a node can connect or disconnect to another node at any point of time. The connections are inherently managed by the nodes within the network using any of the available mechanisms [45].

3. Robots: A set of networked robots $\boldsymbol{R} = \{R_1, R_2, R_3, \ldots, R_k | k \geq 1\}$ all of which hosts an agent platform within and can connect to the network $\boldsymbol{W}$ in an ad-hoc manner. These are essentially a subset of nodes responsible for the execution

of tasks. Robots are mobile and are equipped with sensors and actuators that enable them to sense their environment and act upon them, respectively.

4. Tasks: A set of finite tasks $\boldsymbol{T} = \{T_1, T_2, T_3, \ldots, T_n | n \geq 1\}$, capable of being executed by the set of robots $\boldsymbol{R}$.

5. Resources: Utilities and nodes other than robots, such as a path, parking/charging bays, a rack containing items which can act as a node, sensor nodes, etc., in the MRS environment required by a robot to accomplish a task, constitute a set of resources $\boldsymbol{\Psi} = \{\Psi_1, \Psi_2, \Psi_3, \ldots, \Psi_r | r \geq 1\}$. Resources need to be shared amongst robots in the set $\boldsymbol{R}$ while a robot executes the tasks in $\boldsymbol{T}$. Once a robot takes over a resource(s), it becomes non-shareable before it is freed by the robot. For clarity, two forms of conventions have been followed in this paper viz. Ψ_i and Ψ^i, where for the task T_i, Ψ_i is a particular resource from the set $\boldsymbol{\Psi}$ while $\Psi^i \subseteq \boldsymbol{\Psi}$.

6. States: States pertain to robots. $S_i^{T_j}$ indicates that a robot is in state S_i and requires to execute the task T_j. All free robots remain in the state designated as S_1^*.

7. Agents: A set of mobile agents $\boldsymbol{\mu} = \{\mu_1, \mu_2, \mu_2, \ldots, \mu_m | m \geq 1\}$, such that each mobile agent $\mu_i \in \boldsymbol{\mu}$, carries the programs of its associated tasks as its payload. It may be noted that each agent carries the programs for a set of task(s) assigned to it along with the information of the required set of associated resources. An agent also carries with it the State Information (SI) in the form of $S_i^{T_j}$ of the robots which it can serve, and the next state to which the robots transit after execution of T_j.

8. Job: A Job J_i is a collection of tasks in $\boldsymbol{T}$ along with the associated set of resources in Ψ, which are required to be executed by the robots in $\boldsymbol{R}$ and constitute the basic inputs to the system. Here, $J_i \subseteq \{(T_1^i, \Psi^1), (T_2^i, \Psi^2), .., (T_n^i, \Psi^n)\}$, T_n^i is the n^{th} task of job J_i and $\Psi^n \subseteq \Psi$. The intersection of subsets of the type Ψ^n need not be a null set indicating that a particular set of resources could be required by more than one task. These jobs are processed and packed into mobile agents by a Job Distributor J_{Dist}. New jobs received by the J_{Dist} could

commence their execution even when their predecessors are being executed.

Here k, n, r and $m \in$ I where I is a set of positive integers.

4.3.2 System Specifications

For a better insight into the complexity of the proposed CPS, the specifications and behavior of the system need to be defined precisely. Listed below are some pertinent points about the system —

1. The number of nodes in the network $\boldsymbol{W}$ is finite.
2. Any node can connect or disconnect from the network $\boldsymbol{W}$ at any instant of time.
3. The system is completely oblivious of the total number of robots $\boldsymbol{R}$ present in the network $\boldsymbol{W}$ at any point of time.
4. The number of mobile agents $\boldsymbol{\mu}$ inhabiting the network $\boldsymbol{W}$ varies dynamically with the change in the sequence of tasks.
5. The sequence in which the tasks in the set $\boldsymbol{T}$ need to be executed may be changed as per the requirements.
6. Each of the robots and the agents are autonomous entities capable of carrying out independent executions.
7. There is no direct robot-to-robot or agent-to-agent communication.

4.4 The Task Execution Ordering Problem

Consider a CPS with a finite number of homogeneous robots. Each robot is required to carry out the execution of a finite number of tasks that are interdependent. Since all the robots are required to execute such tasks, a robot may need a set of resources which are shared among its peers. This invokes the necessity for the mutual exclusion of these resources while executing the tasks. As discussed earlier, in the real world the number of robots available for task execution may vary with time. In addition, one may need to alter, add or delete tasks *on-the-fly*. Under such circumstances,

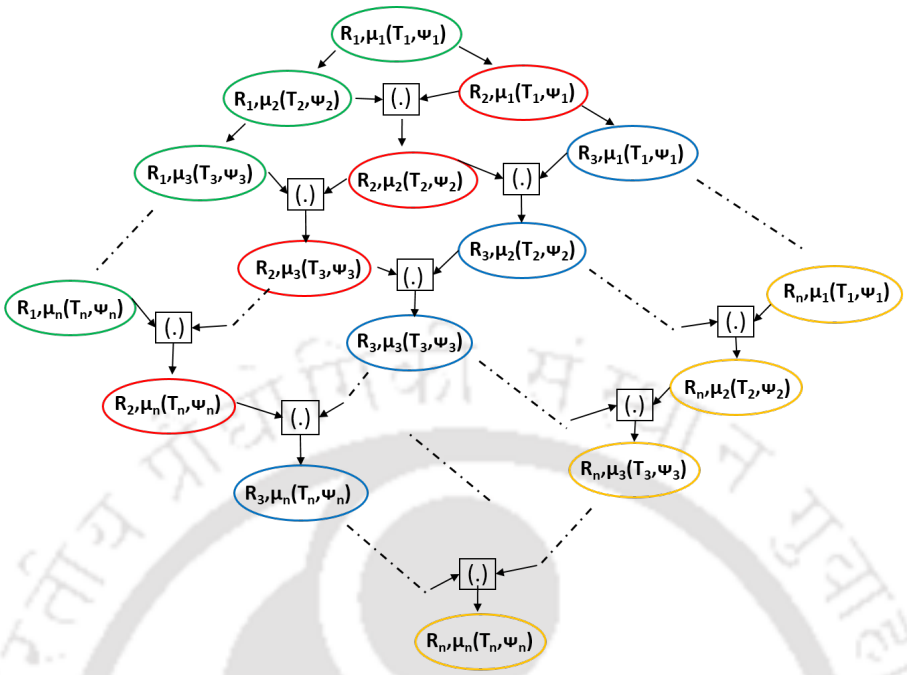


Figure 4.1: Graph depicting the inherent sequential and interdependent nature of execution of tasks

ordering the task executions *on-the-fly*, becomes a challenging problem. This chapter models this problem as a Task Execution Ordering Problem (TEOP) and proposed a solution to the same using a set of mobile agents.

For simplicity, consider a straightforward scenario where each task T_i requires a single resource Ψ_i. The scenario can be easily extended to more complex ones where instead of a single resource, a task may require a set of resource Ψ^i which may be common with those of other tasks. Figure 4.1 represents the problem in the form of a directed acyclic graph. Each node of the graph represents a robot R_i performing a task T_i using the resources in Ψ_i with the help of a mobile agent μ_i. Additionally, there are some nodes which represent the operator - AND (.). This operator makes sure that a certain robot can perform a task T_i if and only if the previous dependent task T_{i-1} is completed (the sequential execution constraint). Since mobile agents carry the programs of the corresponding tasks, this translates the actual dependency of execution onto these agents. Hence, if a robot R_j is vying for resources to execute a task T_i, the mobile agent μ_i carrying the program for the task T_i must be free and available in the network. By free, it means that the mobile agent should not be resident within a robot nor aiding the execution of the

associated task.

The graph shown in the Figure 4.1 depicts the manner in which the robots execute the tasks while also ensuring mutual exclusion. It may be noted that all the nodes having the same color correspond to the same robot. Thus only one of these same-colored nodes can be active at any moment of time. For instance, the red colored nodes stand for the robot R_2. The tasks T_1, T_2 and T_3 thus cannot be executed concurrently since they all need to be executed by the same robot viz. R_2.

As an explanatory example, consider the node $(R_2, \mu_2(T_2, \Psi_2))$ denoting execution of task T_2 by robot R_2 using resource Ψ_2 and mobile agent μ_2. In order to traverse to this node, both inputs to the AND node viz. $(R_1, \mu_2(T_2, \Psi_2))$ and $(R_2, \mu_1(T_1, \Psi_1))$ need to be TRUE i.e. R_1 should have executed task T_2 using μ_2 and Ψ_2 and R_2 should have executed T_1 using μ_1 and Ψ_1. This indicates the sequential nature of execution of tasks by R_2 viz. $(R_2, \mu_1(T_1, \Psi_1)) \rightarrow (R_2, \mu_2(T_2, \Psi_2))$. Additionally, the interdependency between $(R_2, \mu_2(T_2, \Psi_2))$ and its predecessor nodes $(R_1, \mu_2(T_2, \Psi_2))$ and $(R_2, \mu_1(T_1, \Psi_1))$ can also be observed. This means that T_2 (carried only by agent μ_2), which requires resource Ψ_2 for execution, cannot be executed by multiple robots at the same time thereby ensuring mutual exclusion.

4.4.1 Inherent Objectives

With several robots and shared resources, ordering the executions of sequential, independent and interdependent tasks, becomes a complex task especially when the number of executing robots and tasks vary at run-time. This section discusses this problem of ordering in terms of its segregated objectives.

Objective 1

The main objective of the work presented in this paper is to honor the mutual exclusion of the use of resources $\boldsymbol{\Psi}$ by the robots in $\boldsymbol{R}$ while executing all the tasks in the set $\boldsymbol{T}$. Let $R_i^{j,t}$ be a binary function that returns 1 during the time slot when the robot R_i has acquired resource Ψ_j. Hence the objective is

$$\forall R_i \in \boldsymbol{R}, execute(R_i, \boldsymbol{T})$$

subject to

$$\forall \Psi_j \in \Psi, \sum_{i=1}^{n} R_i^{j,t} \leq 1 \tag{4.1}$$

$$R_i^{j,t} \in \{0,1\}, \forall i,j$$

at any time instant t.

Here, *execute*(R_i, $\boldsymbol{T}$) denotes that the robot $R_i \in \boldsymbol{R}$ executes the tasks in the set $\boldsymbol{T}$. As can be observed from Equation 4.1, the constraint of the objective function essentially denotes the MER amongst the robots such that no more than one robot can acquire the same resource at any given time.

The Objective 1 essentially makes the robots in $\boldsymbol{R}$ to align their executions in the form of a *pipeline*. Pipelining [136] is extensively used by the computer processors in order to increase throughput. It facilitates the execution of the several of instructions in a single unit of time. For instance, the three main subtasks performed by processors to complete the execution of an instruction are – *Fetch*, *Decode* and *Execute* [136]. In the absence of a pipeline, the processor has to finish the first instruction which it received from the memory and then move towards the next instruction sequentially. This makes the other functional units of the processor such as the ALU to idle while the *Fetch* instruction is being performed. However, in a pipelined architecture, when the processor is busy executing an instruction, other units within, can perform other subtasks concurrently. However, these subtasks need to be synchronized by a common clock. Any increase or decrease (addition/deletion) in the number of subtasks can cause asynchronism. This gives rise to the second objective.

Objective 2

The second objective is concerned with maintaining the time period of each stage in the asynchronous robotic pipeline. A pipeline in the context of processors comprises a set of cascaded tightly coupled processing elements. The output of one is given as input to the next. These elements are driven *synchronously* by a clock whose time period is set to a value greater than the maximum delay incurred between the elements in the pipeline. Finding this maximum time delay and setting the clock

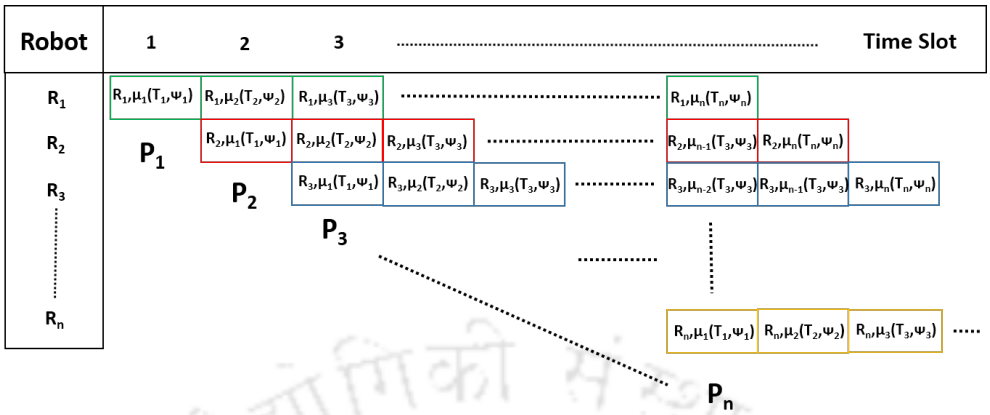


Figure 4.2: Pipelined execution of a set of sequential and interdependent tasks having shared resources in the proposed CPS where different colors indicates different robots

accordingly is possible in the domain of a computing system as the execution and delay times once fixed, never change. However, in real-world robotic scenarios, these timings depend on the task at hand and the conditions and position or location of the robot. In other words, the time to execute a task could vary temporally. This adds another dimension of complexity since the *asynchronously* executed tasks whose execution times vary, could cause problems when the robot(s) try to access a shared resource. Under such conditions, the use of a synchronous lock whose time period is set to a constant value *a priori* could prove to be disastrous. It may be noted that when n robots are executing n tasks, each with distinct resources concurrently, the robotic pipeline is full and operating at its maximum, thus satisfying the following optimality criteria –

$$\forall \Psi_j \in \Psi, \sum_{j=1}^{r} \sum_{i=1}^{k} R_i^{j,t} = r \tag{4.2}$$

at any time instant t.

Objective 3

The final objective of this chapter is to facilitate the concept of *on-the-fly* ordering. In the proposed CPS scenario, one may require to modify, add (insert) or delete tasks in/from the set $\boldsymbol{T}$ *on-the-fly* while the robots are executing tasks in the current set $\boldsymbol{T}$. Further, it may happen that the robots themselves, which are analogous to the

processing elements in a pipeline, need to be inserted or removed due to failures, low batteries, etc. Hence, providing such flexibilities to the end users of such a system is of vital importance.

Taking all these challenges into consideration, the graph in Figure 4.1 is converted into a pipeline model as depicted in Figure 4.2. The vertical axis in herein represents the robots in action while the horizontal one represents the time slots when the robot R_n uses the resource Ψ_n in order to accomplish task T_n. In addition, $P_1, P_2, P_3, .., P_n$ denotes pipelines formed at time slots 1, 2, 3,...,n respectively. As discussed earlier, it can be seen that in time slot 2, the program carried by μ_2 facilitates R_1 to execute T_2 using the resource Ψ_2. Concurrently, R_2 executes T_1 using the program in μ_1 and the associated resource Ψ_1. Both μ_1 and μ_2 remain resident on the respective robots R_2 and R_1 until the tasks are accomplished and thus are not available to any other robot during slot 2, thus ensuring mutual exclusion among the robots. It may also be noted that R_1 executes T_3 whose program is carried by μ_3 using Ψ_3 in time slot 3 only after R_1 and R_2 both have executed T_2 and T_1 respectively as shown in Figure 4.1. It can be seen that in the n^{th} time slot, the pipeline becomes full with all robots R_1, R_2,..., R_n in states S_n, S_{n-1}, S_{n-2},..., $S_1^\star$ executing the allocated tasks T_n, T_{n-1}, T_{n-2},..., T_1 using the associated resources Ψ_n, Ψ_{n-1}, Ψ_{n-2},..., Ψ_1 respectively without any contention. It may be noted that these associated resources could be subsets of Ψ, namely Ψ^n, Ψ^{n-1}, Ψ^{n-2},..., Ψ^1.

The next section presents the mechanism to achieve the objectives listed above

4.5 The Proposed Mechanism

In the CPS scenario used herein, the robots initially reside at a bay or docking station and their states are initialized to $S_1^\star$. This signifies that all the robots in $\boldsymbol{R}$ are currently vying to execute the associated task viz. T_1. As mentioned earlier, every robot hosts an agent framework that allows these agents to knit through them.

In addition, all the mobile agents in $\boldsymbol{\mu}$ are also released into the network $\boldsymbol{W}$ by the Job Distributor J_{Dist}. Task ordering among the jobs is highly dependent on how the J_{Dist} assigns tasks and their associated resources to the agents. Thus, a separate section is provided for its discussion along with an underlying algorithm.

4.5.1 Job Distributor

The Job Distributor J_{Dist} assigns the various subsets ($\{T_i, \Psi^i\}$) within a job to corresponding agents along with the information on the associated set of resources required to execute the tasks. It also embeds the State Information (SI) of the robots it can serve together with the next state to which the robots need to transit. Further, the J_{Dist} also maintains a list of already assigned resources so that the same resource is not assigned to other agents. An agent returns to the J_{Dist} when the assigned task(s) has been executed thereby relinquishing the associated resources. In the example graph shown in Figure 4.1, a simple scenario was chosen where a task requires only a single resource. But in the real world, it is natural to have tasks that require multiple resources which may need to be shared with other tasks. Further, the tasks within a job could be sequential or independent. The task assignment strategies followed by the J_{Dist} for different scenarios are described below.

A. Only sequential tasks

Consider a warehouse scenario wherein an item has to be first fetched from a rack, carried to a packing station and finally packed and shipped to its desired destination. This job comprises a total of 3 sequential tasks - *Fetching* (T_1), *Carrying* (T_2) and *Packing* (T_3), all of which are sequential in nature. For cases when these tasks use different resources, the J_{Dist} assigns a distinct agent for each task together with the associated set of resources. Thus T_1 and Ψ^i, are embedded in μ_1. Likewise, T_2 and Ψ^j and T_3 and Ψ^k are embedded within μ_2 and μ_3. The SI and next SI written onto each of these agents μ_1, μ_2 and μ_3 are – $S_1^\star \rightarrow S_2^{T_2}$, $S_2^{T_2} \rightarrow S_3^{T_3}$ and $S_3^{T_3} \rightarrow S_1^\star$ respectively. Since T_3 is the last task of the job, the next SI is stored as $S_1^\star$ thereby freeing the robot executing this job. These 3 mobile agents are then released into the network.

B. A mix of sequential and independent tasks

Imagine a job that involves a request for exchange of an item. The set of tasks comprising this job could be - first *Fetch* (T_1) the new item, then *Carry* (T_2) it to the packing station and then *Pack* (T_3) the same. While these tasks are being executed

by one robot in a sequential manner, another robot could concurrently *Stamp* (T_4) the item as defective (or some such) and *Place* (T_5) it back to the concerned rack. Since the two sequences ($T_1 \rightarrow T_2 \rightarrow T_3$ and $T_4 \rightarrow T_5$) are independent of each other they do not share any resources. For such kind of jobs, the J_{Dist} sets the SI of the assigned agents μ_1, μ_2 and μ_3 to $S_1^\star$, $S_2^{T_2}$ and $S_3^{T_3}$ and that of μ_4 and μ_5 to $S_1^\star$ and $S_2^{T_5}$, respectively. This concurrent execution of the two sequences ($T_1 \rightarrow T_2 \rightarrow T_3$ and $T_4 \rightarrow T_5$) within a job improves both time and robot utilization.

C. Multiple tasks using the same resource(s)

In the above two cases, it is assumed that the tasks that were sequential did not use a common resource i.e. ($\Psi^i \cap \Psi^j \cap \Psi^l = \emptyset$). It may happen that a job comprises two or more tasks which require the same set of resources. Under such conditions, the task assignment is done purely on the basis of the shared resource needed. For example, if Ψ^i is required to execute tasks T_1 and T_2 while Ψ^j requires T_3. Thus, T_1 and T_2 are interdependent while T_3 is independent (assuming $\Psi^i \cap \Psi^j = \emptyset$). Under such scenarios, the J_{Dist} assigns both the interdependent tasks T_1 and T_2 to a single agent μ_i while T_3 is assigned to another agent μ_2. The State Information (SI) embedded within μ_1 and μ_2 are given below:

μ_1: $S_1^\star \rightarrow S_2^{T_2} \rightarrow S_1^\star$

μ_2: $S_1^\star \rightarrow S_1^\star$

It can be seen from the above SI that μ_1 will find a free robot and make it execute T_1 and T_2 consecutively before freeing it. μ_2 will find a separate free robot and make it execute T_3 concurrently. Since T_1 and T_2 are now within the same agent, mutual exclusion of the resources shared by these tasks are ensured by the agent itself. It may also happen that $\Psi^i \cap \Psi^j \neq \emptyset$. This can be easily reduced to the scenario similar to Ψ^i i.e. all the three tasks T_1, T_2 and T_3 will become interdependent and thus, packed into a single agent by the J_{Dist}.

It may be noted that in a set of sequential tasks within a job, say, $\{T_1, T_2, T_3\}$, the resource for a certain task(s) (T_3) could be free while those of the others are already assigned to agents of the previous jobs. In such scenarios, the J_{Dist} is forced to wait for the agents to return and relinquish the resource(s). However, if T_3 is an independent task, the J_{Dist} will assign it to a separate agent and release it. The

Algorithm 2 Sequence of steps followed by the Job Distributor J_{Dist}

```
Input: A Job J_i in the form of a set of tasks and associated resources {Jobs can arrive at the Job Distributor asynchronously}
Output: μ: A set of mobile agents with each agent containing a task(s) and its associated resource(s)
repeat
  if ∀tasks ∈ J_i == Sequential then
    if resources_available() == True then
      if num_of_tasks_require_same_resource(J_i) > 1 then
        Follow steps as described in Section 4.5.1.C;
      else
        Follow steps as described in Section 4.5.1.A;
      end if
    else
      while(resources_available() == True);
    end if
  else if ∀tasks ∈ J_i == (Sequential OR Independent) then
    if resources_available() == True then
      if num_of_tasks_require_same_resource(J_i) > 1 then
        Follow steps as described in Section 4.5.1.C;
      else
        Follow steps as described in Section 4.5.1.B;
      end if
    else
      while(resources_available() == True);
    end if
  end if
until Job is present
```

agent then follows Algorithm 3 and executes the assigned task. The algorithm for the J_{Dist} is portrayed in Algorithm 2.

4.5.2 Mobile Agent based Mechanism

Consider a scenario with repetitive jobs having similar tasks and associated resources are landing on the J_{Dist} which are then assigned to the corresponding mobile agents and release into the network of robots. Now, as soon as the mobile agent μ_1 lands on robot (say R_1), it verifies the current state of that robot. If a matching state is found (which in this case is $S_1^\star$), μ_1 provides the code for the task T_1 to the robot (R_1 here). Hence, the robot R_1 commences the execution of task T_1 by acquiring the resources Ψ_1 as per the program received from the agent μ_1. After the execution of task T_1 by R_1, the mobile agent μ_1 updates the state of R_1

Algorithm 3 Sequence of steps followed by each mobile agent μ_i for the execution of their assigned task T_i

1: **Input**: State $S_x \in \{S_1^\star, S_x^{T_i}\}$ and Program for task $T_i \in \boldsymbol{T}$; { State is $S_1^\star$ if T_i is the first task to be executed else State is $S_x^{T_i}$}
2: **Output**: Execution of task T_i, $\forall R_k \in \boldsymbol{R}$;
3: **repeat**
4: migrate_to(R_k) ; {Agent migrates to a robot R_k}
5: S=get_state(R_k) ; {Agent fetches the current state of robot R_k}
6: **if** S_x==S **then**
7: commence_execution(T_i,Ψ^i) ; {Agent makes robot R_k execute the code for T_i using Ψ_i}
8: S_{x+1} = get_next_state(); {Agent calls the function to get the next State Information (SI) stored within it}
9: update_state(R_k, S_{x+1}); {Agent updates the state of the robot R_k to the next state carried by the agent}
10: **end if**
11: leave_robot(R_k) ; {Agent migrates into the network to search for other robots}
12: **until** Job is present

Note: A mobile agent carries with it the program or code for a task T_i assigned to it, the state S_x of the robots which it needs to search for and execute the code for T_i along with the very next state the robot should transit (after execution of T_i), in accordance with the job whose task T_i, it carries.

to the next state (depending upon the next task). Consequently, R_1 relinquishes the resource Ψ^1 and waits for μ_2 to arrive. The mobile agent μ_1 then leaves the robot, returns back to J_{Dist} and releases the task along with the associated resource information. This task and resource is then assigned to a new mobile agent for the next job (which in the current scenario is similar to the previous job) by the J_{Dist} and is then released into the network. If μ_1 does not find a matching state, it migrates to another neighboring robot in a conscientious manner, thereby avoiding more recently visited robots.

The mobile agent μ_1 for job J_2 lands up in another robot (say R_2) in state $S_1^\star$ and makes it execute task T_1 using the resource Ψ^1. In this manner, μ_1 continues to make all robots in state $S_1^\star$ to perform task T_1 sequentially. When μ_2, which is also migrating within the network, lands in R_1, it aids the latter in the execution of task T_2 using Ψ^2. Hence, both the robots R_1 and R_2 execute the tasks T_2 (job J_1) and T_1 (job J_2) respectively in a concurrent manner forming a 2-stage pipeline. As time progresses all the k robots start executing distinct tasks concurrently to form

of a k-stage pipeline. Here, the autonomous mobile agents act as tokens to acquire the associated resources in order to carry out an execution. Algorithm 3 depicts the steps that each mobile agent follows for the execution of their assigned tasks. Thus, it can be seen that by virtue of following this algorithm, the set of agents $\boldsymbol{\mu}$ order the execution of tasks, in a manner that ensures mutual exclusion of shared resources among the jobs.

The proposed solution ensures that the free robots are selected and mutually excluded once they start a task within a specific job. Thus, once a robot is booked (mutually excluded) for a job by an agent, the same robot continues to execute all tasks related to this job. The robot is finally released only after the last task (contained within the related agent) is executed. Mutual exclusion is also taken care of when tasks common to multiple jobs require the same resource. Common tasks requiring different resources occurring across multiple jobs are executed concurrently. Mobile agents, once released into the network, act autonomously without any central control. With many networked robots in the scenario and with mobile agents knitting through this network, this proposed CPS as a whole, performs in a decentralized and distributed manner.

4.5.3 Asynchronous Execution Times

Unlike instructions in a CPU, the tasks executed by robots may not have fixed execution times. This could be due to a range of reasons which include wear and tear of various parts, the nature of the paths traversed by a robot, obstacles, charging times and network delays. This issue of non-uniformity in execution times of the various tasks in the pipeline cannot be efficiently handled by the traditional method of using a common clock.

In the present decentralized and distributed CPS, a mobile agent is the only entity that has the code for the execution of a specific task. To mitigate the problem of varying time periods in the robotic pipeline, the mobile agents do not leave a robot until the concerned task is accomplished. Consider a case when R_i is executing task T_j using μ_j and T_j takes more time than T_{j-1} which is being executed by R_{i+1} using μ_{j-1}. This forces R_{i+1} (after the execution of T_{j-1}) to wait for the completion of execution of T_j by R_i. This is because μ_j (currently within R_i) has not yet been

released. Thus, even though the time durations that the robots take to switch from one task to another within the pipeline keep varying over time, the mobile agents facilitate pipelined execution without the use of a common clock. This makes the proposed mechanism adaptive to varying task execution times.

4.5.4 *On-the-fly* Addition/Deletion of task(s)

A real-world system is always prone to changes which could be sudden or gradual. For a system comprising sequential tasks, these changes can be in the form of addition of new tasks or the deletion of already existing tasks to/from the set $\boldsymbol{T}$. There may also be a case where an existing task is required to be replaced by a new or modified version. Traditionally in a centralized system, one would have to bring the whole system down by suspending the executions of all the tasks and then restart the same after the modifications are made. This naturally is a time-consuming and inefficient exercise. The proposed method for the execution of sequential tasks inherently allows for *On-The-Fly Programming* (OTFP) without bringing the system down. In order to ensure the modification of the task sequence, all the state transitions, from one state to the next, are stored *a priori* locally in the *state transition database* of each robot. In this context, the modification could mean addition, deletion or altering the sequence in which the tasks are executed.

The addition of a new task to the set $\boldsymbol{T}$ requires two new mobile agents — one that updates the state change information in the robots (referred to as the Sequence Agent (μ_{seq})) and another that carries the program for the new task (μ_1'). The former agent, μ_{seq}, which is released into the network $\boldsymbol{W}$ with the new modified sequence, migrates within $\boldsymbol{W}$ and updates the state transition database within each robot to reflect the modifications. Thus, if the initial state transition database in all robots comprised the sequence $S_1^\star$, S_2, S_3,..., S_n and the new task to be inserted between T_1 and T_2 is T_1' then this agent updates the sequence to $S_1^\star, S_1', S_2, S_3, \ldots, S_n$ in all robots in R. This would mean that a robot completing the execution of task T_1 (in state $S_1^\star$) would now transit to S_1' instead of S_2 thereby executing the associated task T_1' before T_2 using the second newly released agent μ_1'. Once the modifications are done in each robot, the μ_{seq} terminates itself. The second agent μ_1' is the one that carries the new program as its payload and aids the robots to perform the new

task T_1'. This agent behaves the same way as all the other agents in the set μ.

Deletion is done by merely deleting the concerned state in the transition database by this agent. It may be noted that if any of the task(s) previous to the task currently being executed by the robot gets modified, then the robot continues with the successive tasks and does not redo the entire job. The sequence can also be altered in a similar manner to control the order in which the tasks in $\boldsymbol{T}$ are executed. Both addition and deletion, thus facilitate the shuffling of the sequence of tasks in the pipeline *on-the-fly*. The above feature thus provides OTFP facility to the system.

4.5.5 Mutual Exclusion for Concurrent Tasks

Contrary to the pipelined case, all tasks in the set $\boldsymbol{T}$ can be executed concurrently. Since each task T_i has its own dedicated resource Ψ_i, the agent μ_i can latch on to any robot R_j (provided it is free) and commence executing the associated task. Thus, if there are n tasks (i.e. n agents) and n robots then at any point of time all agents can execute their respective tasks using a robot each. If there are m jobs comprising n tasks each and if all tasks take the same amount of time t for execution, then the total time required for execution of all the jobs would be $m*t$, where * designates the multiplication operator. Mutual exclusion will be preserved, even if the number of robots is greater than the number of jobs. This is so since each task is associated with a single agent which in turn can use only one robot at any moment of time. It may thus be noted that the mechanism described herein can cater to both sequential and concurrent sets of tasks.

4.5.6 Avoiding Deadlocks

According to Coffman et al. [39], a system is in a deadlock state if all the four conditions defined below hold simultaneously –

1. Mutual Exclusion: The resources required are non-shareable and thus requires mutual exclusion.

2. No-Preemption: Resources already assigned cannot be preempted.

3. Circular wait: Presence of circular list or chains of processes waiting for resources acquired by their predecessors.

4. Hold and Wait: A process is holding at least one resource and is also waiting to occupy another resource.

The conditions (1) and (2) hold for the current proposed system. Mutual exclusion is a necessary requirement since the resources become non-shareable once the robots latch on to them. Preemption comes with the risk of indefinite starvation of a resource(s) by the robot preempted by the system and thus adds to the overall cost of execution.

Consider the resource-allocation graph shown in Figure 4.1 which has been converted to a pipeline representation portrayed in Figure 4.2. According to condition (3), if a resource-allocation graph contains at least one cycle, then it can attain a deadlock state. Thus, in order to show that the proposed system is deadlock free, it is sufficient to prove that the graph is acyclic. By applying Kahn [94] algorithm for topological sorting on the graph shown in Figure 4.1, a pipeline representation similar to Figure 4.2 can be obtained. This proves that the graph is a Directed Acyclic Graph (DAG). Hence, condition (3) does not hold for the proposed system thereby making it deadlock free. Depending upon the type of job, condition (4) could hold for certain scenarios and therefore does not guarantee the deadlock-free behavior of the proposed system. Even though condition (3) is sufficient to ensure the deadlock-free behavior of the system, further investigations to remove the condition (4) could be carried out and forms the part of future work of this paper.

4.6 Implementation

In order to validate the efficacy of the proposed mechanism, an automated warehouse as a CPS in order to implement the proposed mechanism has been chosen. This CPS is used to process shipments after the orders are received at the warehouse. The CPS within the automated warehouse comprises a set of networked robots and smart racks. The robots are required to fetch items from the racks and deliver them to the packaging zones. These chores can be decomposed into several tasks such as follow a path to the selected rack, pick the item, traverse towards the packaging zone and place the item there. This sequence of tasks involves the use of shared resources such as the racks and the paths. Warehouses generally optimize on space

which means that the racks are placed close-by thus allowing only one robot to move in between them. This path as also the concerned rack thus form shared resources which can be used by only one robot at any moment of time. This enforces the need to ensure mutual exclusion of resources within the automated warehouse.

In order to ensure that mutual exclusion is preserved, warehouse management systems have to either constantly monitor and control the movement of robots or the robots themselves have to manage and regulate such exclusions. The former method is more centralized and resource intensive where a single or multiple sets of servers constantly monitor and control the robots. Centralized methods have their own drawbacks [127]. The latter method, wherein the robots themselves as a whole manage such mutual exclusions and executions, forms a decentralized and distributed approach which is what this chapter portrays.

The performance of the proposed mechanism for ordering task execution was validated by emulation followed by experiments using real robots. In order to test the practical viability over large networks, the proposed method was emulated on real networked nodes. Emulation (and not simulation) was carried out to ensure that the experiment is closer to the real environment and captures the real-time issues such as network failure, congestion, packet/data loss, etc. in the system. According to the authors in [34], emulation offers more concrete and reliable results than simulation. *Tartarus* [163], a mobile agent platform was used for emulation of the proposed mechanism for sequential and interdependent task execution. Each instance of *Tartarus* running on a computer acts as a node in the network. For the experiments, a 100-node network was created with sets of nodes running on separate computers connected through a LAN. A separate node acted as the Job Distributor (J_{Dist}) which receives the request (job) for the items and converts it in the form of tasks per request. Another additional PC (personal computer) was used to log the status of all the entities and events during the experiments. These logs were used to plot the graphs and analyze the results. It may be noted that these additional nodes (J_{Dist} and PC) did not participate in the proposed mechanism. Experiments were conducted to rigorously test the features of the proposed method for scalability, adaptivity, and OTFP. Each experiment was performed 10 times and the average of

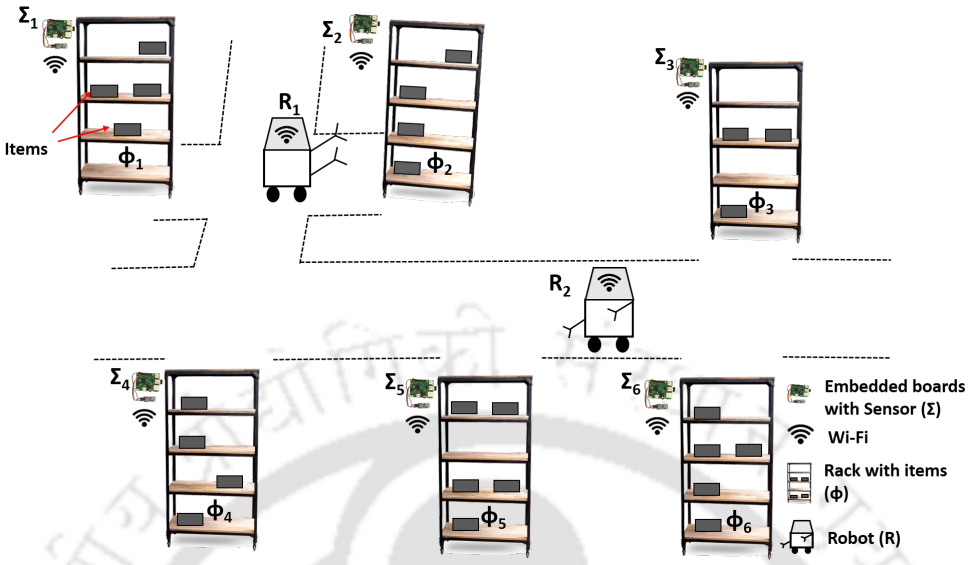


Figure 4.3: A warehouse with racks, robots and embedded boards with sensors

the readings was taken into account while plotting the graphs [▶].

4.7 Experiments and Results

In this section, the experiments conducted and results obtained from both emulation and real-robots, are discussed. The section also compares the proposed approach with its centralized counterpart and highlights the conditions when it is favorable to use the former.

4.7.1 Emulation

As mentioned earlier, a 100-node *Tartarus* based network was deployed on 10 PCs over a LAN wherein each node represented an instance of *Tartarus*. For honoring equal distribution of load, each PC was initiated with 10 instances of *Tartarus*. Depending on their functionality in the real world, the nodes in the emulation scenario are divided into three types. The *Robotic* nodes (R) and the *Shared resource* nodes (Φ) form the *Primary nodes* while the remaining constitute the *Secondary nodes* (Σ). In the present context, these Σ nodes are the inactive nodes which merely allow the agent to flow through in the network such as a router, sensor nodes, etc.

[▶] A video of the experiment performed is available at the following link:
https://www.youtube.com/watch?v=-D9HbtSpe9E

They may, however, be made active so as to perform other tasks such as sensing, data processing, etc. based on the application scenario. A static agent residing on each of the R- and Φ-type nodes performed the job of waiting for the mobile agents in order to receive the code for the task to be performed within them. For emulation, the tasks were designed such that it would take 2 seconds to execute each of them assuming an ideal environment without any unforeseen time lags. Figure 4.3 depicts the primary and secondary nodes for a warehouse scenario.

4.7.2 Comparison with a Centralized Controlled System

In order to fortify the stand on the use of this decentralized and distributed mechanism, it is essential to compare the results of the same with those obtained using a centralized control mechanism. A centralized emulation framework for the experimental set-up was thus made using the same agent platform viz. *Tartarus*. A centralized server operating at a node was responsible for posting the relevant commands using TCP-message based communication. This centralized scenario thus comprised the central node hosting the server and the R and Φ nodes acting as its clients. The setup was obviously devoid of mobile agents. The proposed mechanism for execution of mutually-exclusive sequential tasks was emulated on this centralized framework. Instead of the mobile agents carrying the programs required for the execution of the tasks, the robots herein had the required programs to execute all the tasks, embedded in their respective memories *a priori* on-board. The experiment comprised execution of a series of jobs with increasing number of tasks. Since the experiment was performed on an emulated framework, a total of 5 different soft computational tasks were chosen viz. Sorting (Sort), Merging (Merge), Addition (Add), Subtraction (Sub) and Division (Div) of data within the nodes. The 5 tasks were repeated for jobs containing more than 5 tasks i.e. if a job has 7 tasks, than the sequential tasks within this job comprises – Sort, Merge, Add, Sub, Div, Sort, Merge. The tasks were designed in such a way that the total computational time for each task was equal to 2 seconds in an ideal environment without any lags.

Initially, no resources are occupied until the execution commences. The central server thus sends a message to the robot node R_1 to commence the execution of task T_1. In order to ensure mutual exclusion, messages are passed to the central server

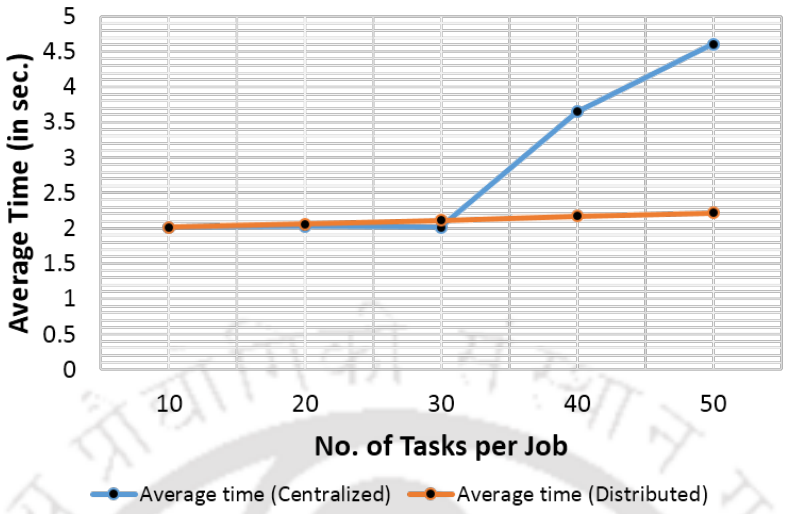


Figure 4.4: Centralized versus the proposed decentralized and distributed approach

by each robot as and when a task is to be initiated or completed. On receiving a message from a robot node, say R_j after the completion of task T_i $(i, j \geq 1)$, the centralized server sends a message to the $R_{j+1^{th}}$ node informing that the resource Ψ_i has been relinquished and that the task T_i can now be executed. In this way, each robot executes a task T_i only when the central server gives it a green signal to do so. The central server thus manages task execution for all robot nodes and hence serves to ensure mutual exclusion.

The variation in the performances of the centralized and the proposed mobile agent based decentralized and distributed mechanisms have been portrayed in Figure 4.4. As can be observed, when the number of tasks per job is below 30, the centralized mechanism seems to perform a tad better than the proposed version. However, as the number of tasks grows (beyond 30), the throughput of the centralized system degrades rapidly since the average time for execution of a task increases. With increasing number of tasks, the volume of information to be exchanged (between the robot nodes and the server) in order to manage the execution of these tasks and ensure mutual exclusion, also increases drastically. Such a large number of server-to-client communications results in a majority of time being wasted on acknowledging and replying to the various nodes. When the central server is loaded with such a large number of requests, it takes more time for the completion of the

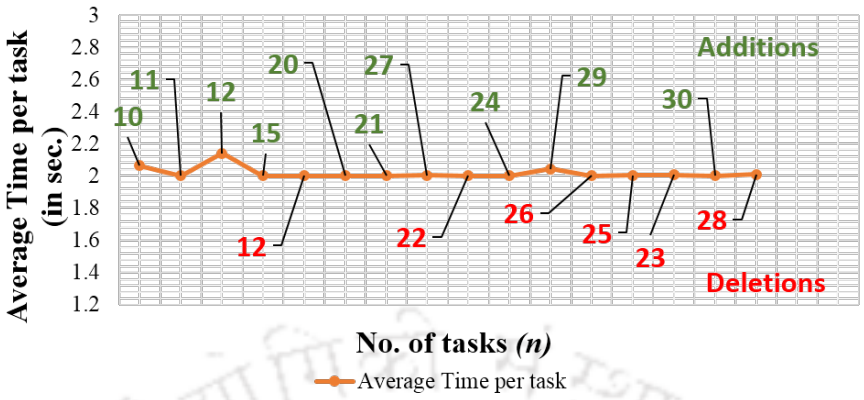
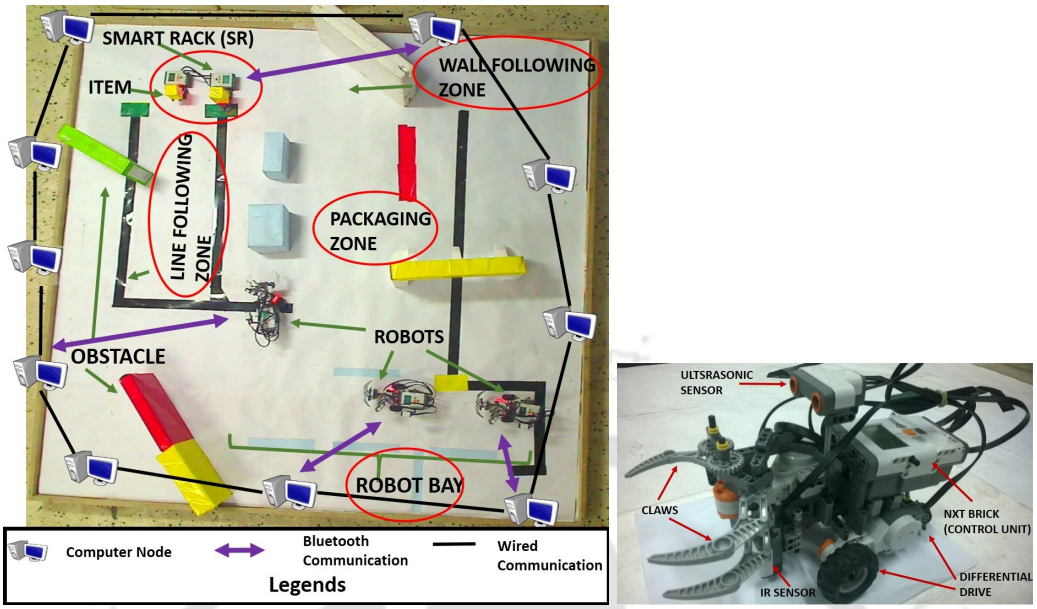


Figure 4.5: Addition/Deletion of tasks *on-the-fly*

tasks due to these computational constraints. In the case of the proposed decentralized approach, the rate of increase in the average time required for completion of the jobs can be observed to be gradual thus indicating the superiority of this approach. These results also show that the system is scalable in the sense that it is hardly affected by the increase in the number of tasks per job (aka. mobile agents).

4.7.3 Task Addition and Deletion

As previously mentioned, each of the tasks takes 2 seconds to complete execution in an ideal environment. To study the effect of addition/deletion of tasks *on-the-fly*, tasks were added and deleted during execution. This caused the length of the sequence of tasks in a job to vary at runtime. Figure 4.5 shows the average time spent per task after addition or deletion of tasks in the set $\boldsymbol{T}$. The numbers above and below the curve in the graph depict the number of tasks comprising the job being executed. As can be seen in Figure 4.5, the experiment initially started off with 10 tasks. While these tasks were being executed, a new task was inserted into the system by the Job Distributor (J_{Dist}). This was achieved by following the procedures mentioned in Section 4.5.4, eventually modifying the total number of tasks in the task sequence to 11, 12, 15, 12, 20...and so on as depicted in Figure 4.5. It may be noted that some of the tasks were also deleted in between the run. One can observe that the graph is almost linear which clearly indicates that, though tasks are added/deleted through OTFP, there seems to be no significant impact on the net time taken for execution. This is so because the extra delays due to communication



(a) (b)

Figure 4.6: (a) Top view of the test-bed (b) Structure of one of the LEGO® MINDSTORMS® NXT robot used in the experiments

overheads and computational requirement are distributed among the nodes of the underlying network. This concurrency reduces the effective increase in such lags. In a centralized approach, all these overheads would add up on the controlling entity, thereby degrading its performance.

4.7.4 Experiments on a Robotic Warehouse

In order to provide a proof-of-concept of the actual working of the proposed technique in the real world, a prototype of the warehouse automation scenario was implemented using a set of mobile robots. The job of each mobile robot was to pick an item from a rack, carry it to the packaging zone situated at another location and place it there, from where on it could be parceled and dispatched. As shown in Figure 4.6a, the experimental setup contains mobile robots and shared resources. The latter include the smart racks and three zones viz. a *line following zone*, a *wall following zone* and a *packaging zone*, all of which need to be shared i.e. they need to be used by the robots in a mutually exclusive manner. A job is divided into four sequential tasks designated T_1 to T_4, which are briefly described below:

1. T_1: A mobile robot waiting at the robot bay executes T_1 by virtue of which it

moves forward before it detects a wall. On detecting the wall, it takes a right turn and again moves forward to finally stop when it detects a black line.

2. T_2: This task makes the mobile robot to open its claws and start following the black line before a green marker is detected. This marker denotes the location of the smart rack.

3. T_3: The robot picks the item from the smart rack using its claws and then follows the wall until a red marker is detected. It then places the item in the packaging zone, thus executing task T_3.

4. T_4: The robot follows a black line before a yellow marker is detected which denotes the start location, thus reentering the bay once again, thereby accomplishing task T_4.

For experimentation with real robots, a set of LEGO® MINDSTORMS® NXT robots are used. Since these robots cannot host the *Tartarus* platform within their controller block, they were connected to respective computers (hosting *Tartarus*) via Bluetooth. An interface similar to LPA-PRO-NXT interface [88] was used to control the robots via these computers that formed the nodes of the network. The movement of the robots was based on one castor wheel and a two-wheel differential drive. A pair of claws attached to the front of the robots facilitated the picking of items while an on-board ultrasonic sensor gauged the distance of objects in front. The robot was capable of following a black line path using a pair of IR sensors. In addition, a color sensor detected different markers laid on the floor. Figure 4.6b shows the structure of one such robot.

As soon as a request for an item is received by the J_{Dist}, the same is converted into a job J and the corresponding set of mobile agents carrying their respective programs (one per agent) is released into the network. A set of four experiments were performed with the number of robots varying from 1 to 4. A total of 4 jobs were assumed to be always fed into the system. In the first experiment, only one robot R_1 was made available at the bay. This signifies a case when one robot needs to perform all the tasks in a sequential manner. R_1 which is initially in state $S_1^\star$ is thus the only robot available to receive the program for task T_1 from agent μ_1.

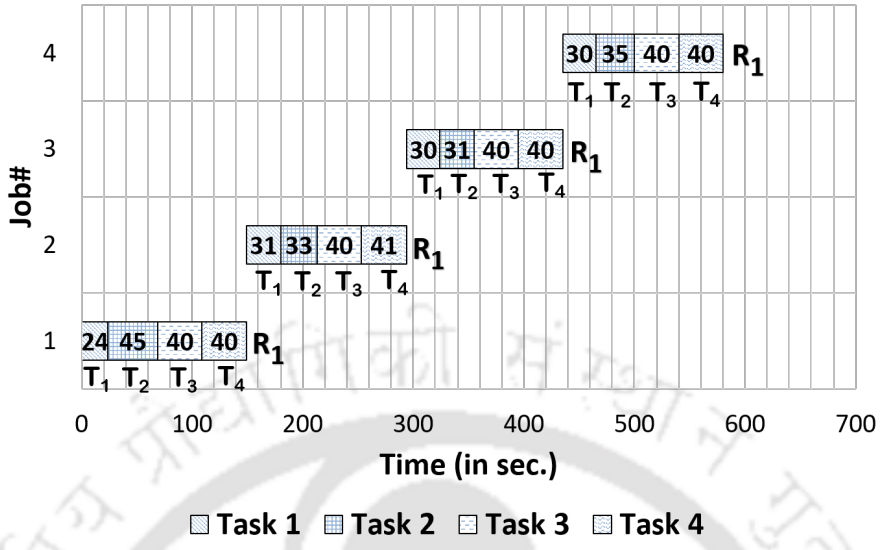


Figure 4.7: Execution of tasks using a single robot

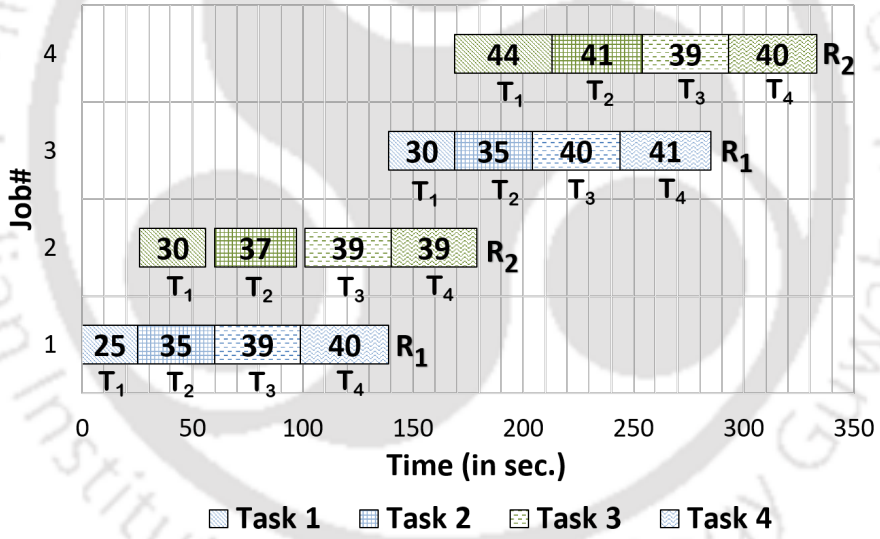


Figure 4.8: Pipelined execution of tasks by 2 robots

After receiving the program, R_1 starts executing T_1 since the shared resources viz. all the zones and smart racks are free. As mentioned earlier, a mobile agent resides within the robot before the latter completes the associated task. After the task is completed, the mobile agent leaves the robot and starts migrating into the network in search of another robot in a similar state that requires the associated program.

After task T_1 is accomplished, R_1 goes ahead in the pipeline only if the resources required to execute T_2 are available. This is only possible when R_1 receives the

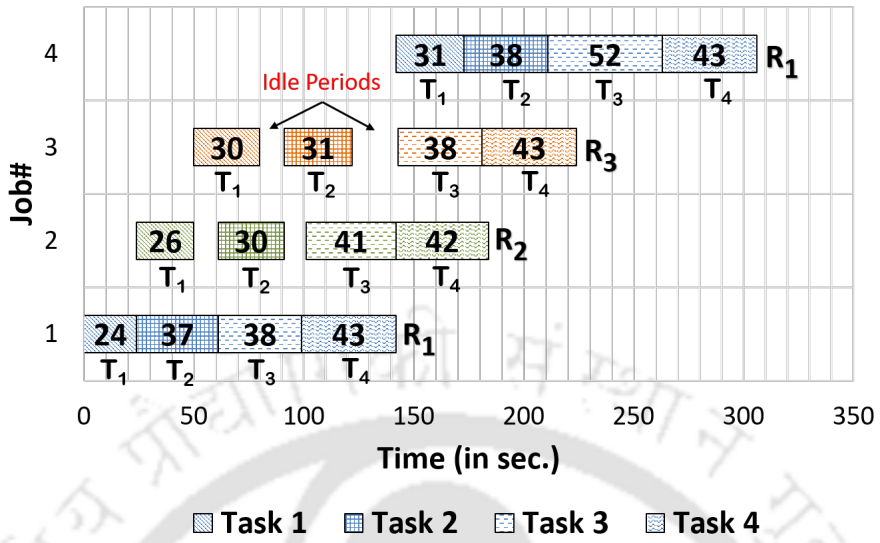


Figure 4.9: Pipelined execution of tasks by 3 robots

program to execute T_2 via the corresponding agent μ_2. Task T_2 is that of picking an item from the rack. The third task T_3 is to place this item at the packaging zone. After T_3 is accomplished R_1 performs the final task T_4 of following the black line and returning to the bay from where it started. This experiment is one of the conventional ways of performing a sequence of tasks with a single robot. It can thus be used as a baseline while comparing the results obtained from experiments using the proposed decentralized and distributed method using pipelined execution.

In the second, third and fourth set of experiments, the same jobs were executed but now in a pipelined manner with the numbers of robots enumerated as 2, 3 and 4 respectively.

Figures 4.7, 4.8, 4.9 and 4.10 depict four graphs where each one corresponds to the results obtained when the number of robots were varied from 1 to 4 respectively. The X-axis in each graph represents the time consumed (in seconds) by the tasks while the Y-axis represents the jobs to be done. The numbers imprinted on the boxes within the graphs denote the execution times taken by the corresponding tasks.

Figure 4.7 shows the results obtained when there is only one robot available to complete the jobs. The different patterns filled inside the boxes in each row denote the corresponding tasks viz. T_1, T_2, T_3 and T_4. As can be seen in the figures, each

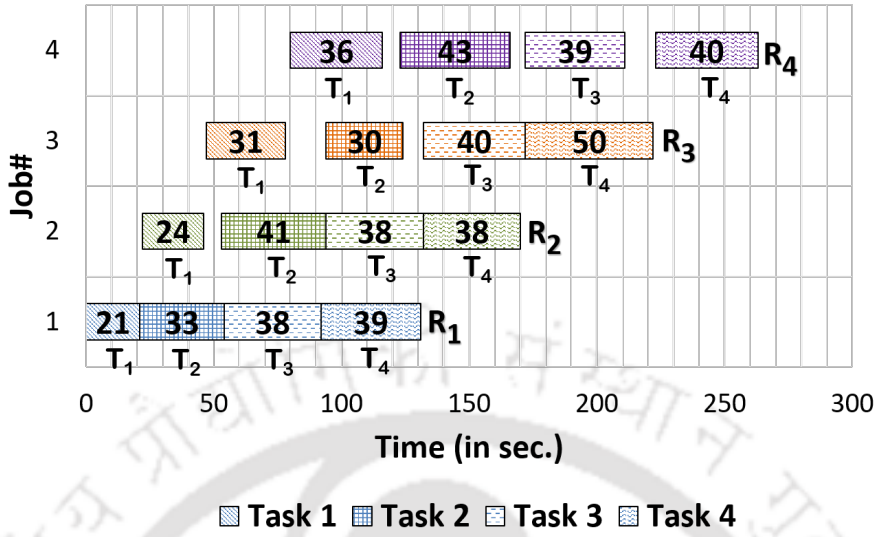


Figure 4.10: Pipelined execution of tasks by 4 robots

row of boxes i.e. tasks denotes a specific job. As soon as all the tasks comprising a job are completed, the robot switches to executing the first task of the next job. In the case when there is only a single robot there are no inter-job execution delays. This is so because the robot does not have to wait for any shared resources to become free for its use.

Figures 4.8, 4.9 and 4.10 show results for cases when the number of robots available are 2, 3 and 4 respectively, in the experiments. In the Figure 4.8, two distinct colors have been used to show the two robots — blue signifying R_1 while green representing R_2. It can be observed that when R_1 is executing T_1, R_2 waits in the robot bay before the resource Ψ_1 acquired by R_1 is relinquished. When R_1 completes the task T_1, it relinquishes the associated resource Ψ_1, which in turn triggers R_2 to commence execution of T_1 while R_1 switches to execute T_2 using resource Ψ_2. Execution commences only after the associated agent (μ_1 for T_1 and μ_2 for T_2) reach the concerned robot and provide the relevant programs. Thus, when the mobile agent μ_1 triggers execution of T_1 by R_1, the rest of the robots at the bay cannot execute T_1 since the associated mobile agent μ_1 is now busy with R_1. This inherently ensures proper ordering of execution of tasks while also obeying mutual exclusion.

Subsequently, both R_1 and R_2 enter the pipeline and concurrently execute tasks T_2 and T_1 respectively. It can be observed that the robots seem to take

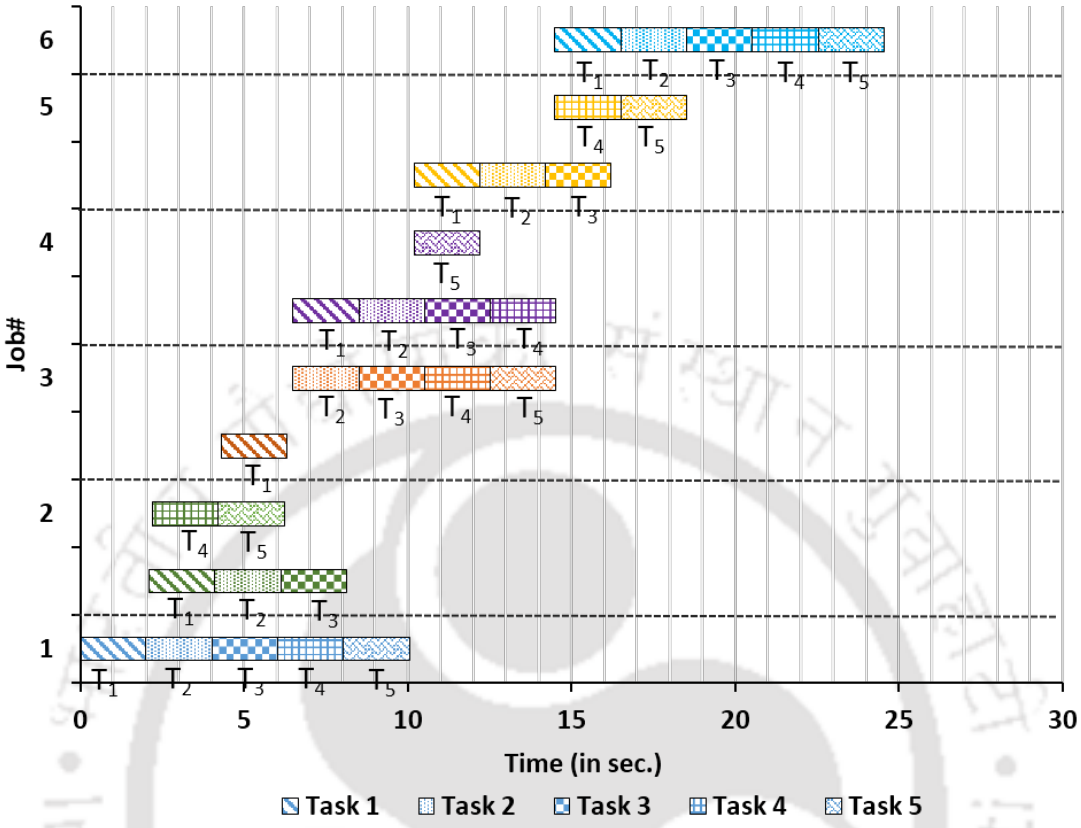


Figure 4.11: Execution of jobs comprising both sequential and independent tasks

unequal times to execute the same tasks which in turn cause *idle periods* between the executions of two consecutive tasks. Each *idle period* between the tasks along the row indicates the extra time the robot waits for the resource of the subsequent task to be relinquished by its predecessor. In the real world, execution times depend on the wear and tear that the robots undergo, their controllers, charge on the battery and other environmental conditions. In Figures 4.9 and 4.10, these *idle periods* are more prominent due to the presence of more robots. Thus, one cannot provide guarantees that a task will take the same amount of time to complete as it did earlier as in the emulation experiments. The speed-up obtained when 2, 3 and 4 robots were used were found to be 1.75, 1.84 and 2.21 respectively.

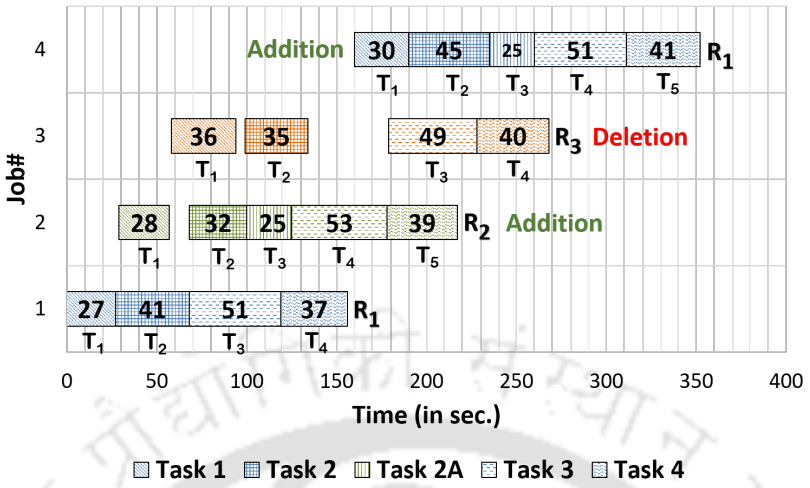


Figure 4.12: Execution of tasks using 3 robots - Addition and Deletion of task T_{2A} *on-the-fly*

4.7.5 A mix of Sequential and Independent tasks within a job

Real-world jobs usually comprise heterogeneous tasks where a few of them could be sequential while the rest may be independent. Results for the experiments conducted for such jobs have been portrayed in Figure 4.11. The experiment was conducted in an emulated environment similar to the one discussed in Section 4.7.1. As can be seen from the figure, job J_1 and J_6 are composed of purely sequential tasks and thus are executed in sequence. Jobs J_2, J_3, J_4 and J_5 comprises two sequences of tasks which are independent of each other. Thus, as shown in the figure, the proposed system manages to execute the two sequences of tasks within a job in sub-optimal concurrent manner.

4.7.6 Modification of the Sequence of Tasks

Figure 4.12 shows the graph when a task is added and deleted *on-the-fly* from the sequence of tasks T. In this experiment, a new task T_{2A} was introduced via μ_{2A} during the execution of T_2 in Job 2. It was then deleted immediately after its execution in Job 2. The same task was again added during the execution of T_2 in Job 4. The additions and deletion were performed using mobile agents as described in Section 4.5.4. The new task in the context of the warehouse scenario was a detour from the normally used path which constituted the mobile agent μ_{2A} and the

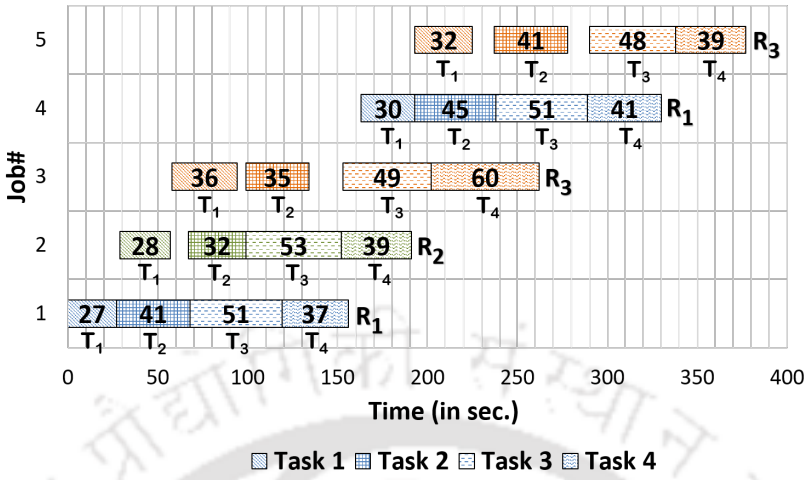


Figure 4.13: Execution of tasks using 3 robots - Removal of a robot (R_2) *on-the-fly*

associated resource Ψ_{2A}. After R_2 completed task T_2, μ_{2A} caused it to execute T_{2A} in Job 2. Since Ψ_2 is now free, R_3 (saffron colored row box) commenced execution of T_2 using μ_2 concurrently, as seen in Figure 4.12.

It can be seen that the introduction of task T_{2A} introduced a large *idle period* in Job 3. This was because when R_3 was executing task T_2, R_2, having finished execution of T_{2A}, commenced the execution of T_3. The case arose because the time for executing T_{2A} by R_2 was less than that for executing T_2 by R_3. Addition and deletion of tasks *on-the-fly* seemed to have no effect on the other concurrently running tasks.

4.7.7 Removal of robots

The batteries of all the robots need to be recharged after their energy levels go below a certain predefined threshold. The threshold was chosen in such a way that even if this threshold were to be crossed at the commencement of a job, the robot would have enough energy to complete all the remaining tasks in that job and then return to the robot bay for charging. This is synonymous to the removal of a robot from the system. It can be observed in Figure 4.13, that R_2 was removed after it completed Job 2 in a 3-robot scenario. The absence of R_2, caused R_3 to perform the task T_1 in Job 5 just after R_1 completed T_1 in Job 4. As can be observed the pipeline continues to execute tasks concurrently in spite of the absence of R_2.

It can thus be observed from the results of the experiments, both emulation and real-world, that though the task execution times vary, the system adapts to these changes and maintains the pipeline even in the absence of a clock. Also, the resources associated to the tasks are used in a manner that mutual exclusion is preserved without any direct communication.

4.8 Chapter Summary

This chapter portrays a mechanism for ordering the execution of a set of interdependent tasks within a CPS of robots and sensor nodes operating in the real-world under the constraint of mutual exclusion. A CPS, which forms part of an automated warehouse has been used as the target application. The mechanism, however, can be easily ported to other decentralized and distributed scenarios where shared resources are utilized and mutual exclusion needs to be implemented. It has also been shown that a centralized solution could no doubt be an option to solving the mutual exclusion problem but its performance degrades as the system is scaled upward. The use of mobile agents makes the mechanism described herein decentralized, distributed and scalable and also allows for changes to be made in the tasks as also the set of robots operating within the system, during runtime.

A robot could fail or be removed from the CPS environment leading to the unpredictability of the nature of the execution of the pipeline. Uncertain incidents such as these can conventionally cause the stalling of the pipeline. The absence of a common clock for the pipeline aids the mechanism in coping up with such a situation. Since the mobile agents manage the execution and ensure mutual exclusion, the waiting times differ based on the availability of the shared resources which in turn aids in re-synchronizing such unforeseen changes in the environment. This also addresses the challenge posed by fluctuations in the execution times taken by the physical processes or tasks performed by the robots. Additionally, since mobile agents carry the related programs for task execution, this mechanism can support heterogeneity in terms of the use of a range of computational entities including robots and sensor nodes. New entities can be plugged into the CPS along with mobile agents carrying the programs for the associated tasks. The use of mobile

agents facilitates the OTFP feature which allows such injection or addition and deletion of tasks on-the-fly. This feature can also help in purging deprecated or faulty programs from the system and replacing them with the updated ones during runtime. One can also envisage augmenting this mechanism with Self-healing as in [87] so that a failure within the CPS is quickly noted and the concerned agent is drawn towards the node so that the task(s) can be re-executed if required.

This chapter alleviates the problem of Mutual Exclusion which may arise in a distributed network of nodes such as CPS and IoT with no central controlling authority. However, it has been assumed that the tasks or problems to be solved do not change during the running life of the system. In the real-world, this may not be so. Problems could change or new ones could take their place asynchronously across the CPS network. This may impede the functioning of the system. It may also happen that its parameters need to be re-tuned offline. Such a catastrophic scenario could be avoided if the system were able to adapt by itself and continuously learn and evolve during its lifetime. Systems empowered with such features will not only cope up with changes in the problems that need to be solved but also learn to find the more optimal solutions for the problem at hand. The next chapter presents a method to find the best solution which can cater to a given problem, in a distributed and decentralized manner. From an incoming stream of new solutions, the system tries to adapt and find the best mappings between the given problems and solutions. Concepts imbibed from the Biological Immune System have been used to achieve embodied life-long learning in *d*CPS.

"Nature holds the key to our aesthetic, intellectual, cognitive and even spiritual satisfaction."

Edward Osborne Wilson (1929)
American biologist

5

Immunological Inspiration and Metaphors

Robustness, adaptability, and decentralization found in biological systems have always been the inspiration for many researchers to mimic these properties. One such system is the Biological Immune System (BIS), known for its remarkable ability to recognize and learn the pattern of incoming antigens (pathogens or foreign cells) during the life of the host. The BIS is composed of several components which coordinate through local interactions to serve the global purpose of protecting the host body. One of the most significant features of the BIS is its distributed manner of functioning. Heterogeneous populations of antibodies attack populations of antigens across the body of the host in a distributed, and concurrent manner. Based on the learned information, the more effective antibodies formed as a result of previous antigenic attacks, facilitate a faster response to similar attacks in the future thereby making the system self-adaptive and one that evolves continuously. Exploiting this paradigm would naturally pave the way for realizing mechanisms for decentralized and distributed CPS. This chapter attempts to provide a review of the immunological metaphors and the manner in which they are adopted in the computational world. The contents herein form a prelude to the work presented in the subsequent chapters which describe mechanisms and algorithms inspired by the BIS.

5.1 Biological Immune System

The Biological Immune System (BIS) is a two-tier defense system comprising the *innate immune system* and the *adaptive immune system*. The former is present from the very birth of a vertebrate and remains constant throughout the life of an individual. The cells that contribute to this system do not require prior exposure to the foreign cells or *antigens* and are thus immediately available to curb a set of antigenic attacks. While the innate system remains fixed to the type of antigens it can counter, the *adaptive* immune system learns the patterns of new antigens and develop responses specific to each of them, thereby, making it the primary source of inspiration behind the subsequent contributions made in this thesis. It may be noted that henceforth, the acronym BIS shall refer to the adaptive immune system.

5.1.1 The Adaptive Immune System

The ability of the BIS to learn and quickly counter the effects of a new or already encountered antigen provides tremendous insights and motivation to develop robust, yet simple selection and learning mechanisms suited to decentralized and distributed computational scenarios. The BIS consists of several types of immune cells such as the B-, T-, Killer and plasma cells, all of which together defend the biological being. For simplicity, these cells will be referred to as *immune cells*. The *antibodies* on the immune cells within a BIS are responsible for the recognition of a given *antigen*. The *paratope*, which forms the antigen-binding site of the antibody and the corresponding complementary-shaped *epitope* on an antigen, facilitate antigenic recognition. Figure 5.1a shows an antigenic detection by the complementary paratope-epitope matching for three different antibodies. The immune cells associated with each of the antibodies are shown in Figure 5.1b. The term antigen has been used synonymously for all types of foreign pathogens. As soon as an antigen is detected within the body of an individual, the process of generating and/or attracting the right kind of antibodies at the site of detection is initiated. The antibodies make their way to the antigenic site through the fluids present inside the body. Different antibody-antigen interactions and recognition mechanisms which form the basis of the adaptive immune system [43] have been proposed; the more prominent ones

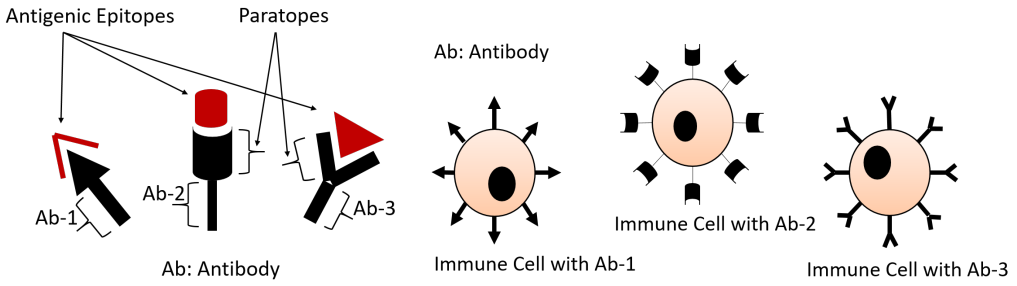


Figure 5.1: (a) Different antigenic epitopes and complementary paratopes of antibodies (b) Immune cells with different antibodies

being from the theories of Clonal Selection, the Immune Network, and the Danger theory. A detailed review on these three theories and resulting mechanisms is provided in Chapter 5. The following sections describe the significant contributions and provide a brief chapter-wise summary of the thesis.

5.2 Theories of Biological Immune System

A theory intends to explain the activities of a complex system. Though theories are overwritable, one should be open enough to salvage the general ideas and use them as a computational tool to solve problems in different domains [43]. The same applies to the Clonal Selection, Immune Network, and the Danger theories. All these theories have been separately used for solving computational problems. However, this thesis attempts to portray a hybrid approach that aims to combine them in order to adopt the best. In other words, this thesis emphasizes the fact that from a computational perspective, the mechanisms from these theories can be used in tandem to mimic the best possible selective and learning behaviors of the BIS.

5.2.1 Clonal Selection

Clonal Selection [28] is one of the earliest and widely accepted theories. According to this theory, in the presence of an antigenic attack, the immune cells capable of successfully recognizing an antigen, proliferate and clone into multiple cells. The clones in turn, undergo somatic *hypermutation* [43] depending upon the activation events. Unlike its conventional counterpart, hypermutation is a controlled form of

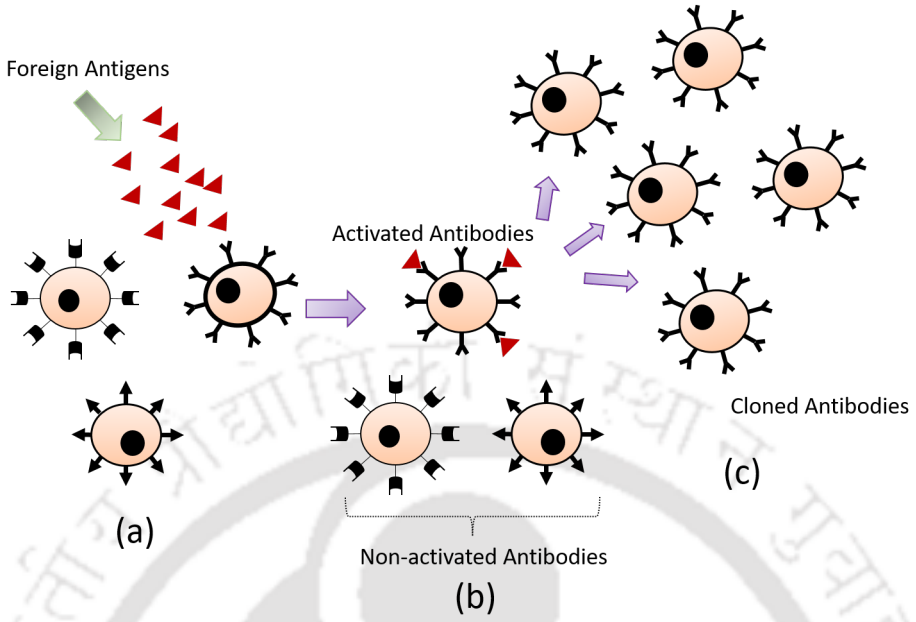


Figure 5.2: The clonal selection. Antibodies which are able to recognize the antigens, binds and get activated resulting in a clone

mutation wherein the search attempts to narrow down onto the optimum. Cloning and hyper-mutation together add diversity to the antibody repertoire of the adaptive immune system which in turn helps ameliorate the antigenic recognition. The antibodies which are more efficient in this recognition process eventually become *memory cells* that remain for a longer time within the biological being. These cells, in turn, reduce the secondary response times to the same antigen. Figure 5.2 presents a simplistic schematic process of clonal selection.

5.2.2 Idiotypic Network

The Idiotypic or Immune Network (IN) theory proposed by Jerne in 1974 [85], portrays a novel way of looking into how the BIS works. Unlike clonal selection, this theory states that the immune cells interact with one another even in the absence of an antigenic attack. According to Jerne [85], in addition to a paratope, an antibody on an immune cell harbors an idiotope, which can be looked upon as analogous to an epitope on an antigen. A paratope of an antibody can interact and recognize the idiotope of another even in the absence of an antigen (epitope). Several such interactions can aid in the formation of a dynamic network, also known as the

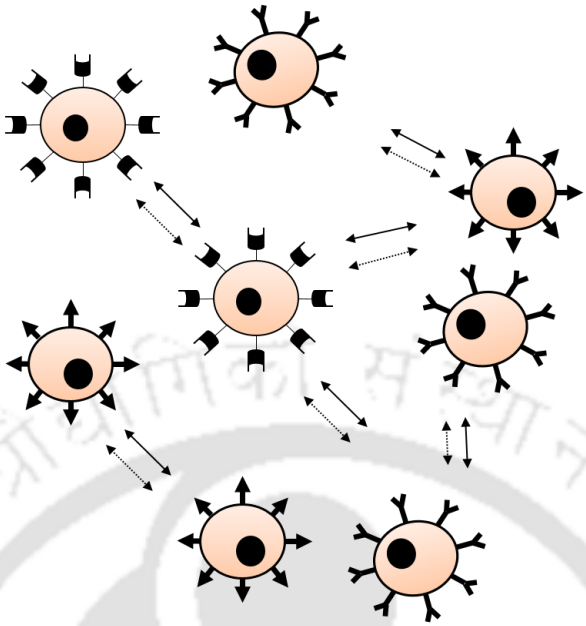


Figure 5.3: Antibody to Antibody interactions in Immune Network Theory. The solid lines are the stimulations while the dotted lines represent the suppressions.

idiotypic network. The interactions (as shown in Figure 5.3) can be either positive (stimulation) or negative (suppression) depending on whether a paratope recognizes an idiotope. These stimulations and suppressions finally result in the proliferation or decay of the antibody molecules.

5.2.3 Danger Theory

This theory [124] proposes that the immune system responds to events that are harmful and cause damage, which in turn signifies danger. Danger signals are however not flagged in normal cell death (apoptosis). A danger zone formed at the site of an antigenic attack attracts the attention of the concerned immune cells. On reaching this zone, these cells get stimulated, undergo clonal expansion and in turn contain the attack. Figure 5.4 shows the danger signals being released under an antigenic attack. Researchers have been inspired by mechanisms exhibited by the BIS, to formulate algorithms for a variety of applications.

The theories presented above, thus point towards the adaptive, learning and information processing properties inherent in the BIS. The following section lists some of the relevant features associated with the BIS and presents an Artificial Immune System built on the immune metaphors.

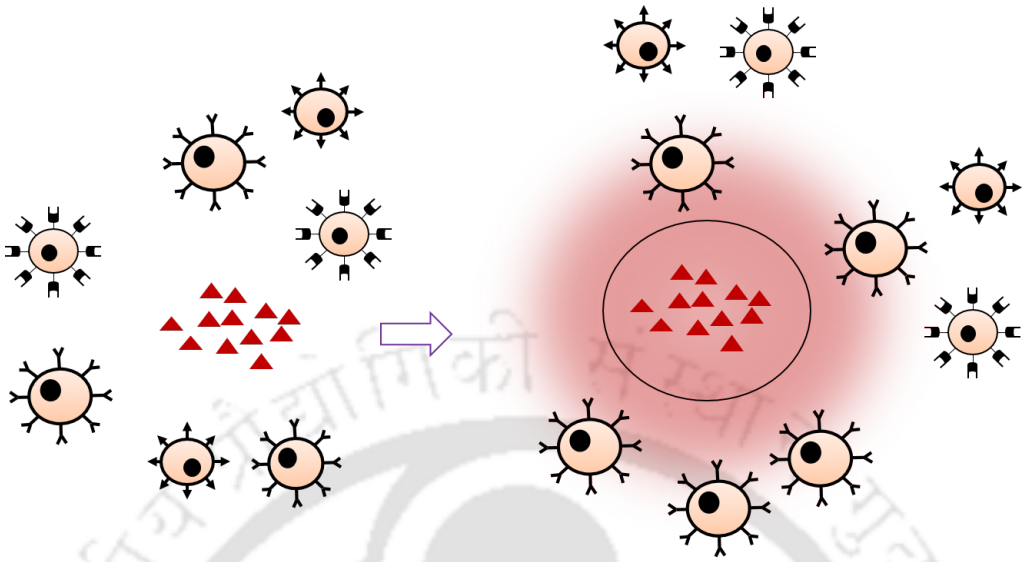


Figure 5.4: Release of danger signals and attraction of specific antibodies towards the portion of host body part under attack

5.3 Features

Though there are several properties [43] which can be found in the BIS, those that are important from the computational perspective are discussed below.

- Decentralization and Autonomy: A BIS is devoid of any central controlling entity. Thus the components autonomously interact with the environment they live in and provide for a global aim of recognizing foreign cells.

- Pattern recognition: A BIS uses novel mechanisms for identifying antigenic patterns. One way is a lock and key mechanism wherein paratopes on the antibodies match complementarily with the epitopes of the antigens.

- Dynamic repertoire and Generalization: To cater to the varying antigenic patterns that can force their way into the host body, the BIS maintains a dynamic repertoire of antibodies which adapts to the changing patterns once encountered. This provides for a better immune response to a secondary attack of a similar kind.

- Distributedness: The immune cells and the organs are distributed across the host body and thus do not have a central point of failure.

- Fault tolerance: If an immune cell of a specific type that has generated the response for an antigen is removed, other cells become activated on encountering the antigen thereby reassigning the task of matching.

- Robustness: The BIS is a population-based system and thus offers diversity in the different patterns of antigens it can detect by maintaining a repertoire of matching antibodies. Thus, even if some of the cells die, the diverse population still is capable of curbing the attack. This provides for robustness to unseen attacks.

- Adaptiveness: During the lifetime of the host body, the BIS generates responses to the attacks it has encountered in the past. It learns and memorizes the patterns and adapts by producing a population of immune cells having varying antibodies.

- Learning and Memory: For a prolonged response, some immune cells adapt significantly than others and thus reside in the host body for a more extended period of time contributing towards learning and memory in the BIS.

- Self-organization: According to the principle of clonal selection, whenever an antigenic attack happens, the matching immune cells become activated, clone and hypermutate to contain the attack autonomously. Without the aid of any external entity, the different components of the BIS self-organize to the incoming foreign invasion.

Such intriguing features have attracted the research on developing computational tools inspired by the BIS. The Artificial Immune System (AIS) is one such paradigm which is based on the mechanisms found in the natural immune system.

5.4 Artificial Immune System

Jerne's theory of the Idiotypic network is still controversial in the domain of immunology [43]. However, from a computational point of view, it offers functions and tools which can be applied to tasks such as optimization, autonomous robot

navigation, and computer security. According to Castro and Timmis [43], Artificial Immune Systems are,

> "...adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving."

Thus Artificial Immune System (AIS) not only involves non-mathematical theories but is an amalgam of mathematical functions, models and metaphors. It includes all the aspects ranging from pattern recognition to optimization and clonal selection to danger theory. Some of the application areas where AIS has been applied includes classification [189], data mining [183], computer security [77], optimization [176], robot navigation and control [82], machine learning [189], scheduling [38] and agent-based systems [92]. Castro and Timmis [43] identify three basic elements to design an AIS:

1. **Representation:** An abstraction of components such as immune cells, danger signals and mutation in the computation world.
2. **Affinity Measures:** A set of functions to quantify the extent of interactions between the different components within the BIS.
3. **Immune Algorithms:** A set of algorithms which regulate the meta-dynamics within the system.

The following sections describe the computational counterparts of the BIS which have been applied to the work proposed in this thesis.

5.4.1 Shape-Spaces and Cross-Reactivity Threshold

Perelson and Oster [145] proposed the concept of shape-space ($\mathcal{S}$) to represent the amount of interactions between the immune cells and antigens. Each antibody (Ab) has a *Paratope* (Pt) which aids in recognizing an *Epitope* (Ep) of the antigen (Ag). The Ep and Pt can be described by L parameters thereby forming an L-dimensional point in the shape-space $\mathcal{S}$. A shape-space could be a real-valued, integer, symbolic or binary, depending upon the application. The extent of the complementarity or the affinity in the *shapes* of Ep and Pt contributes to a recognition. Greater the

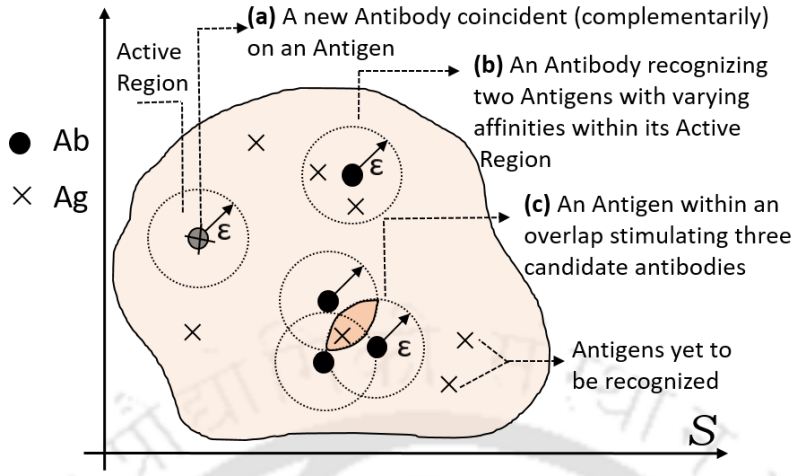


Figure 5.5: A Shape Space $\mathcal{S}$

affinity, greater is the potential of that *Ab* to curb the corresponding *Ag*. An *Ab* is selected for an *Ag* only if the complementary shape of the *Ep* of that *Ag* lies within a small region of the shape space surrounding that of the *Pt* of *Ab*. This small region which is referred to as the *Active Region (AR)*, is characterized by the *cross-reactivity threshold* (ϵ) [43]. All antibodies within this region are attracted towards the antigen *Ag* and hence are *stimulated* by it. Stimulated antibodies clone and proliferate proportionate to their affinities [28] and thus quell the antigenic attack. Figure 5.5 shows a 2-D shape space $\mathcal{S}$ wherein the black dots signify antibodies and the crosses indicate the antigens. When a newly detected antigen has no antibodies to suppress it, an *Ab* which is complementarily coincident to the *Ag* is created (Figure 5.5(a)). All *Ag*s that fall within the active region of this *Ab* can now be catered to, by this *Ab*. As can be seen in Figure 5.5(b) the *Ab* can cater to two of the *Ag*s that lie within its *AR*. Several types of *Ab*s could recognize the same *Ag* if the latter lies within the overlap of their active regions (Figure 5.5(c)). In brief, the role of the BIS is to create, select and evolve a *repertoire* of *Ab*s which are best suited to curb antigenic attacks.

5.4.2 Affinity Measures

Affinity measure (ψ) is the distance between the points on the shape-space $\mathcal{S}$, i.e., between the *Ag* and *Ab*. For a real-valued shape, ψ can be calculated by using a

simple *Euclidean* distance given by,

$$\psi = \sqrt{\sum_{i=1}^{L}(Ab_i - Ag_i)^2} \tag{5.1}$$

where L is the dimension of real-valued vectors Ag and Ab. For binary shape-space, hamming distance could be a measure of affinity between the Ag and Ab.

5.5 Applications

Though AIS has figured in a wide range of closed-world problems, this brief survey describes only those encountered in decentralized, distributed and real-world scenarios.

The primary function of the BIS is to distinguish between the *self* cells and the *non-self* cells where the former are part of the host and the latter include everything else. Based on this classical definition, initial research in AIS catered to the area of security in the computational world. Of late, security has become a necessary requirement in the deployment of IoT based applications. Authors in [118] and [119] have proposed AIS based intrusion detection models for IoT scenarios. Each of the nodes within the IoT develops its own local intrusion detection antibody-like submodules which are shared among the nodes, thus making it a distributed system. Zhang et al. [208] have used CLONALG [44] and AIRS [189] to provide a three-layered hierarchical system for communication to improve security in smart grids. Though the work on security, cited above, emphasized distributed sharing of knowledge in simulated worlds, the AIS mechanisms used were inherently centralized. Implementing these mechanisms in real, decentralized and distributed scenarios is the need of the day. Besides, a prescription for an embodied version is also missing.

In the beginning, AIS was mainly used to implement security. It was only later that the cognitive and learning properties of AIS were found to be useful in other application areas [79]. The Immune Network theory was first used in mobile robotics by Watanabe et al. [187]. They implemented adaptive behavior arbitration in a robot situated in a dynamic environment. The antigens were mapped to the sensor readings (states) obtained from the environment while the antibodies

were taken to be hand-coded atomic actions. Whitbrook et al. [193, 192, 194] introduced evolutionary strategies to learn new actions. The actions were evolved in the training phase while the state-action mappings were learned later during execution. Augmenting the above work with sharing and learning mechanisms in multi-robot scenarios can greatly enhance the performance of such systems.

Watkins et al. [189] introduced an Artificial Recognition System (AIRS) as a supervised classifier. Though AIRS was initially developed for a single computing system, its efficacy on a network of devices has been discussed in [188]. Watkins [188] has stressed the need for a communication mechanism to interact and share knowledge amongst different components in the AIS.

AIS was introduced as a life-long learning system by Sim et al. [176]. This system was used to solve the 1D bin-packing problem. Inspired by the Immune Network theory, Sim et al. proposed a self-sustaining network of interacting entities which is computationally efficient and scalable. The implementation was however confined to a closed-world optimization problem. There is no mention of mechanisms to extend this work to suit real-world decentralized and distributed scenarios, as in a BIS.

In the above discussed immuno-inspired work, problems are represented as antigens while solutions form the antibodies. In order to sustain good solutions, their concentration or population is increased as in the BIS. Controlling these antibody populations (concentration) in a decentralized and distributed manner is yet another challenge. Early work has reported the use of the concept of resources [183, 189] to control the population of antibodies. However, such a mechanism can work only on a single machine or system wherein the information about all its resources is known and accessible globally. One of the ways by which nature tackles this issue in decentralized and distributed environments is by using an indirect form of communication termed *stigmergy* [65]. This mode of communication operates when each entity senses and acts based on its respective local environment which in turn translates to the desired global change. Godfrey et al. [65], describe a cloning controller that uses stigmergy to achieve controlled on-demand cloning in a distributed and decentralized networked system. The work presented in the next chapter also uses the same principle to manage and control the heterogeneous antibody populations.

5.6 Chapter Summary

This chapter presented the fundamental concepts required in understanding the BIS. The features which make it a befitting solution for realizing a decentralized CPS were also described. The chapter further presented the three theories propounded in the immune system and their computational counterpart, the Artificial Immune System, was also explained. The different application domains provide sufficient evidence to apply these simple yet powerful mechanisms in real-world decentralized cyber-physical systems. As mentioned earlier, the aim behind the inclusion of this chapter was to provide the readers with a brief overview and significance of immune-inspired mechanisms used in the following chapters.

"Success is not the key to happiness. Happiness is the key to success. If you love what you are doing, you will be successful."

Albert Schweitzer (1875 – 1965)
Alsatian theologian, organist and physician

6

On Immuno-inspired Solution Selection

Several hyper- and meta-heuristic methods have been successfully employed to find better solutions for problems in closed-world scenarios. However, when the problems change or new problems crop up, the system trying to solve them needs to be halted and its parameters re-tuned offline. In a real-world scenario, the complexity further increases when these problems are distributed across a physical network of devices and robots with no central controlling authority. Circumventing these challenges using conventional approaches such as client-server based methods is not robust and difficult to scale. From many times, researchers have looked into non-traditional methods which are inspired by Nature. Colonies of ants and bees with each entity having limited intelligence can give rise to impressive patterns resulting from coordinated activities instigated by stigmergy (indirect communication). Even the Vertebrate Immune System boasts of such decentralized mechanisms to defend the host body from antigenic attacks. Such decentralized and co-operative mechanisms found in the immune system forms the inspiration of the work reported herein.

This chapter presents a mechanism that integrates the three main theories in immunology viz. the Clonal Selection theory, the Immune Network theory, and the Danger theory, to find the best solution for a given set of problems in a *d*CPS. Problems can occur in any of the nodes comprising this networked system. The system evolves the mapping between the problem and its pertinent solutions and shares

this information across the network of nodes thereby accelerating the search for the best solution. Sharing has proved to be beneficial in enhancing the performance of decentralized and distributed systems [90]. This chapter addresses the challenge of finding the best solution in scenarios which are devoid of an all-knowing global entity.

6.1 Introduction

Finding the best solution for a given problem has always been the motivation for research in the field of Artificial Intelligence (AI). Several conventional techniques have been employed where an algorithm, a meta-heuristic or a hyper-heuristic is tuned to search for a good solution which can cater to a set of problems. Though successfully applied, many a time such approaches require cumbersome re-tuning of the associated parameters if a new problem is added or the nature of the problem itself changes with time [176]. In such scenarios, a system that stores and exploits the past knowledge, self-organizes and learns *on-the-fly* could be the need of the day, especially in the domain of Cyber-Physical Systems (CPS) [11] such as a network of multiple robots and embedded systems. A CPS is an interconnection of various heterogeneous devices over a network. It is made up of *cyber* software components which control and actuate their *physical* counterparts.

Centralized cloud-based computing is traditionally a mainstream architecture for CPSs. A central authority could collect all the data from such devices and process the same to decide the best solution on their behalf. This centralization, no doubt, provides a global control and is relatively simple to implement. However, it suffers from issues such as scalability, a central point of failure, unreachability in remote areas, privacy issues [21, 169], to name a few. Decentralizing control and distributing the computing among the devices could be an alternate architecture for better intelligence sharing especially in networked embedded systems. For instance, recent work on a decentralized email architecture [185] tested on low-cost devices such as Pi, strongly backs the feasibility of decentralized and distributed approaches.

Decentralized systems are those which do not have a central authority to control and coordinate a given process. They are deemed to be distributed if there are

multiple entities (nodes) which are part of the networked system and can compute and communicate concurrently and autonomously. The devices forming such a CPS share their information across the network to which they are connected and make decisions based on collective knowledge. Semwal et al. [166] have described some of the advantages of peer-to-peer (P2P) local sharing of knowledge in such *decentralized* Cyber-Physical Systems (*d*CPS).

Since there is no central authority having a global knowledge, finding the best solutions for a varying stream of problems at each node, is a challenging affair especially in such *d*CPSs. To illustrate this, consider a scenario described in [164] wherein mobile robots and devices are connected across a dynamic network within a warehouse. New mobile robots may connect to this dynamic network at any time while existing ones could disconnect themselves for some reason (e.g., need to recharge their batteries). These nodes may need to share information on the state of the warehouse and the items within, asynchronously. In such a robotic scenario, a *task* to be executed could be viewed as a *problem* which needs to be solved. For example, the task of retrieving an item from a shelf by a robot is a problem that needs to be solved by it. New kinds of problems or tasks could also crop up or be presented to any of these robots. Vehicular Networks (VANETS), where the vehicles need to communicate with their peers to share information, forms another example of a *d*CPS. Service providers can provide value-added advertisements to customers to attract them. Displaying a right advertisement matching to the needs and profile of passengers in vehicles forms an interesting and a challenging application area.

In Nature, most of the problems are solved in a decentralized manner. For instance, insects such as ants, and bees which individually possess limited capabilities, tend to exhibit complex behaviors through simple peer-to-peer interactions and stigmergy [19]. Researchers have proposed several algorithms which are inspired by Biology mainly due to their simplicity and evolving nature. The *Artificial Immune System* (AIS) [42], a computational learning model inspired by the Biological Immune System (BIS) uses such a set of algorithms. The BIS comprises numerous antibodies capable of tackling antigenic invasions. The BIS can select the best antibody (solution) to curb the invasion by a given antigen (problem). The best antibody increases in population, proliferates and subsequently quells the antigenic

population. The AIS, which is the computational counterpart of the BIS, is different from other biological counterparts such as Neural Networks, in the sense that it is inherently a distributed system with no *central controlling* entity. This makes it robust, highly-tolerant and self-organized [30]. Many different theories and models which represent different aspects of the AIS have been proposed and used, the more prominent ones being - Clonal Selection [28], the Immune Network [85] and the Danger Theory [124]. These theories have formed the basis for realizing a plethora of applications in the areas of security [77], optimization [44], robotics [82], data mining [183], machine learning [189], etc. Though a BIS manifests itself in a decentralized manner, most researchers have merely reported their AIS implementations in a centralized manner within closed-world scenarios. A few [189, 119, 118] have proposed decentralized solutions but demonstrated their working in closed world. Shrivastava et al. [170] and Jha et al. [92] describe how an Idiotypic Network could be implemented in a decentralized manner. The authors in [176, 80] describe an AIS based hyper-heuristics for solving the hard problems of 1-D bin-packing and job scheduling. However, these problems too are of the closed-world type.

This chapter proposes a mechanism that integrates the three main theories in immunology viz. the Clonal Selection theory, the Immune Network theory, the Danger theory, to find the best solution for a given set of problems in a decentralized and distributed manner. Problems can occur in any of the nodes comprising a CPS. The system evolves the mapping between the problem and its pertinent solutions and shares this information across the network of nodes thereby accelerating the search for the best solution. Using the problem of sorting numbers as an example domain, an extensive analysis of the proposed system in *emulation* environments has been carried out. The problem of sorting was used only because the mapping between a sorting algorithm and the type of data set it can handle best is evident. This helps in authenticating the viability of the proposed approach. This chapter also presents a real-world robotic application wherein, given a dynamic-set[✿] of path-following algorithms, the robots need to toil in the physical world and find the best among them. New path-following algorithms were also injected into the system *on-*

[✿]In this chapter, a dynamic-set refers to a set into which elements can be inserted or deleted *on-the-fly*.

the-fly to test its learning ability. Depending on the features of the path such as its straightness and crookedness, the robots cooperate and search the best algorithm to tackle the situation. With new algorithms coming into the fray during run-time, the robots were still able to find the better-suited ones thus contributing to decentralized learning. Sharing has been proved to be beneficial in enhancing the performance of the decentralized and distributed systems as in [90]. This chapter addresses the challenge of implementing distributed sharing in *d*CPSs and proposes a mechanism to realize the same using a hybrid of theories propounded in immunology. In order to emphasize the novelty, some of the features of the proposed mechanism are described below:

1. Unlike closed-world problems reported in [176], mechanisms proposed herein cater to the learning and sharing across a real physical network of participating nodes. Nodes could mean computing devices and robots; static or mobile.

2. The proposed approach integrates concepts not only from the Immune Network theory but also those from the Danger theory and the Clonal Section theory to eventually provide adaptivity, self-learning, and self-organization in real decentralized and distributed CPSs.

3. Instead of conventional broadcasting methods, this chapter leverages the use of mobile agents, to share the knowledge amongst the nodes. The mobile agents, dubbed as Intelligent Packets (*iPkts*), are used to carry the mappings between the problem and solution.

4. Finally, how the proposed mechanism can be emulated and also implemented in a real-world warehouse scenario comprising mobile robots, is shown.

The goal was thus to develop an embodied, decentralized and adaptive system which can learn the pattern of incoming problems, with time, and be able to select the best out of a given set of solutions to tackle them, for applications devoid of any central controller. It may be noted that the word *solution* could be either a heuristic, an algorithm, a program, or a system.

The next section discusses the motivation and describes the three theories on immunology. Subsequent sections present the proposed approach, the related

terminologies and immune metaphors, followed by the results obtained in both the emulated and real-world environments and conclusions arrived at.

6.2 Motivation

One of the application scenario which motivates the work presented herein is the exploration of remote areas by a fleet of networked mobile robots. With no centralized server, these robots need to work in a decentralized manner sharing vital information amongst themselves. Further, imagine that these robots have to map an unknown terrain (e.g., Mars Exploration Rover mission) and characterize a wide variety of rocks and soil. Since the environment is unknown and the changes that could occur therein are uncertain, the robots have to continuously learn and adopt new strategies to prolong their survival. A study reported in Nature [12] showcase how cells could be disseminating information through simple peer-to-peer interactions, forming a network of cells. Similarly, the mobile robots could also share the strategies (solutions) which are congenial to their survival. This requires a mechanism which not only shares the information but also maps the problems to the best solutions in a decentralized manner since there is no central system controlling them. One may argue that a specific robot among the fleet could act as a leader and take the necessary decisions. Such an approach is, however, prone to failures [205]. For example, the leader itself could be a single of point of failure which could render the fleet inoperable. Mechanisms which facilitate distributed knowledge sharing and decentralized decision making thus become mandatory in such scenarios.

Autonomous Vehicles (AV) carrying passengers by road also form another scenario. AVs need to optimize on both on-road comfort and efficiency in terms of low fuel consumption. Modern-day AVs update their driving profile based on the experience collected from the environment. One of these updates, for instance, could be to decrease the speed based on the road conditions. However, in order to be proactive, the AVs need to share their experience with other AVs such that they can leverage this prior information and reduce redundant learning. For instance, two AVs driving in opposite directions could inform one another about the respective road conditions. Based on this information, the AV going towards a road which is

in a bad state can adapt its driving strategy. The Information passed between AVs could thus be represented as a feature vector that encodes the road health status such as bumps, detours, debris, etc. Thus, in this scenario, the feature vectors of different roads form the problems while the driving strategies to be used constitute the solutions. Selecting the best solution for a particular road condition in a decentralized manner forms an appealing research area.

Though the presented problem of selecting the best solution for a problem in a decentralized and distributed manner may seem naíve, it puts up many interesting challenges crucial in the realization of information sharing and mapping. These include:

1. Finding mechanisms for the sharing of mappings across the network of nodes.
2. Deciding the meta-dynamics of the system such that the good solutions are rewarded while bad ones are suppressed.
3. Handling asynchronous arrival of solutions and problems which can modify the mapping information.
4. Continuous learning post the deployment of the system.

The next section presents briefly some of the underlying mechanisms found in the BIS followed by the proposed approach and its associated meta-dynamics.

6.3 Brief Survey

Though AIS has figured in a wide range of closed-world problems, this survey describes only those encountered in decentralized, distributed and real-world scenarios. A brief survey on work relevant to the one proposed herein also follows.

The primary function of the BIS is to distinguish between the *self* cells and the *non-self* ones where the former are part of the host while the latter includes everything else. Based on this classical definition, initial research in AIS catered to security in the computational world. Of late, security has become a requirement in the deployment of IoT based applications. Authors in [118] and [119] have proposed AIS based intrusion detection models for IoT scenarios. Each of the nodes within

the IoT develops its own local intrusion detection antibody-like sub-modules which are shared among the nodes, thus making it a distributed system. Zhang et al. [208] have used CLONALG [44] and AIRS [189] to provide a three-layered hierarchical system for communication to improve security in smart grids. Though the work on security cited above emphasized distributed sharing of knowledge in simulated worlds, the AIS mechanisms used were inherently centralized. Implementing these mechanisms in real, decentralized and distributed scenarios is the need of the day.

In the beginning, AIS was mainly used to implement security. It was only later that the cognitive and learning properties of AIS were found to be useful in other application areas [79]. The Immune Network theory was first used in mobile robotics by Watanabe et al. [187]. They implemented adaptive behavior arbitration in a robot within a dynamic environment. The antigens were mapped to the sensor readings (states) obtained from the environment while the antibodies were taken to be hand-coded atomic actions. Whitbrook et al. [193, 192, 194] introduced evolutionary strategies to learn new actions. The actions were evolved in the training phase while the state-action mappings were learned later during execution. Augmenting the above work with sharing and mapping mechanisms in multi-robot scenarios can greatly enhance the performance of such systems.

A recent work by Raza and Fernández [152] describes a multi-tier immunology inspired framework to control heterogeneous mobile robotic systems. Their proposed system uses mechanisms inspired by clonal selection and idiotypic network theories.

Watkins et al. [189] introduced an Artificial Recognition System (AIRS) as a supervised classifier. Though AIRS was initially developed for a single computing system, its efficacy on a network of devices has been discussed in [188]. Watkins [188] has stressed the need for a communication mechanism to interact and share knowledge amongst different components in the AIS.

AIS was introduced as a lifelong learning system by Sim et al. [176]. This system was used to solve the 1D bin-packing problem. Inspired by the Immune Network theory, Sim et al. discussed a self-sustaining network of interacting entities which is computationally efficient and scalable. The implementation was however confined to a closed-world optimization problem. There is no mention of mechanisms to extend this work to suit real-world decentralized and distributed scenarios, as in

a real BIS.

In the above discussed immuno-inspired work, problems are represented as antigens while solutions form the antibodies. In order to sustain good solutions, their concentration or population is increased as in the BIS. Controlling this antibody population (concentration) in a decentralized and distributed manner is yet another challenge. Early work has reported the use of the concept of resources [183, 189] to control the population of antibodies. However, such a mechanism can work only on a single machine or system wherein the information about all its resources is known and accessible globally. One of the ways by which nature tackles this issue in distributed and decentralized environments is by using an indirect form of communication termed *stigmergy* [65]. This mode of communication operates when each entity senses and acts based on its respective local environment which in turn translates to the desired global change. Godfrey et al. [65], describe a cloning controller that uses stigmergy to achieve controlled on-demand cloning in a distributed and decentralized networked system. The work presented herein also uses the same principle to manage and control the heterogeneous antibody populations.

6.4 The Proposed Approach

The proposed mechanism runs on each of the nodes or devices (or robots) connected to each other and forming a *d*CPS. Through an input channel stream, the problem instances can be added to these nodes on-the-fly. Each of the problem instances is assumed to be represented in the form of a Feature Vector (FV), $\mathcal{F}_x$, which could be binary, symbolic or real-valued. The solutions which can cater to the problem instances are also associated with the feature vector, $\mathcal{F}_y$. Figure 6.1 shows an antigen binding with an antibody and their equivalent $\mathcal{F}_x$ and $\mathcal{F}_y$ in the computational world.

The nodes concurrently search for an appropriate mapping between the FVs of a heterogeneous set of problem instances and a repository of solutions and select the best solution. Sharing of the mapping information among the nodes accelerates the search for the best solution. Intelligent Packets (*iPkts*) are used to share the knowledge (mapping information) among the nodes comprising the physical network.

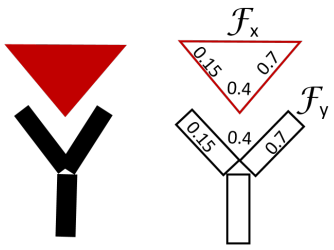


Figure 6.1: Computational counterparts of an antigen and antibody

Finally, the overall system maintains a minimal repertoire of solutions and the associated mappings which can cover the entire problem space for a node. Each of the nodes has initially a minimum number of solutions which may or may not cater to the sets of problem instances. This chapter uses the concept of Localized Idiotypic Networks (LIN) [92] which are basically formed from local interactions (stimulations and suppressions) among the available solutions within a node. A solution receives a performance feedback if it solves the problem instance within the node. The better-performing solutions are stimulated (rewarded) while the lesser ones are suppressed (penalized) mutually. This results in an increase in the population of the best local solutions and the dwindling of the others as in the conventional Idiotypic Network [85]. As shown in Figure 6.2, such LINs are formed concurrently across the *physical network*[*] of nodes so that eventually the best solution(s) survive in the entire physical network. This makes the proposed mechanism decentralized and distributed. Formally,

let $\mathcal{P}$ be the set of instances of a problem to be solved. For example, if we consider the problem of sorting a sequence of numbers then the sequence {4,2,6,8,1}, is an instance of this problem. Let $\mathcal{S}$ be the repository of solutions for $\mathcal{P}$. Elements in both $\mathcal{P}$ and $\mathcal{S}$ can be added to or deleted at any moment of time. A problem instance $x \in \mathcal{P}$ and a solution $y \in \mathcal{S}$ are represented in the form of $\mathcal{F}_x$ and $\mathcal{F}_y$, respectively. Let $\mathcal{M}$ be the mapping between x and y. The objective is to find a mapping $\mathcal{M}^*$ such that for a given x:

$$|\zeta(\mathcal{M}^*(\mathcal{F}_x, \mathcal{F}_y))| \geq |\zeta(x, y)| \ \forall y \in \mathcal{S} \tag{6.1}$$

[*]In this thesis, a *physical network* refers to an actual network of hardware nodes which are interconnected through a wired or wireless medium. An Idiotypic Network, on the contrary, is a meta-level network formed by the interactions between the solutions.

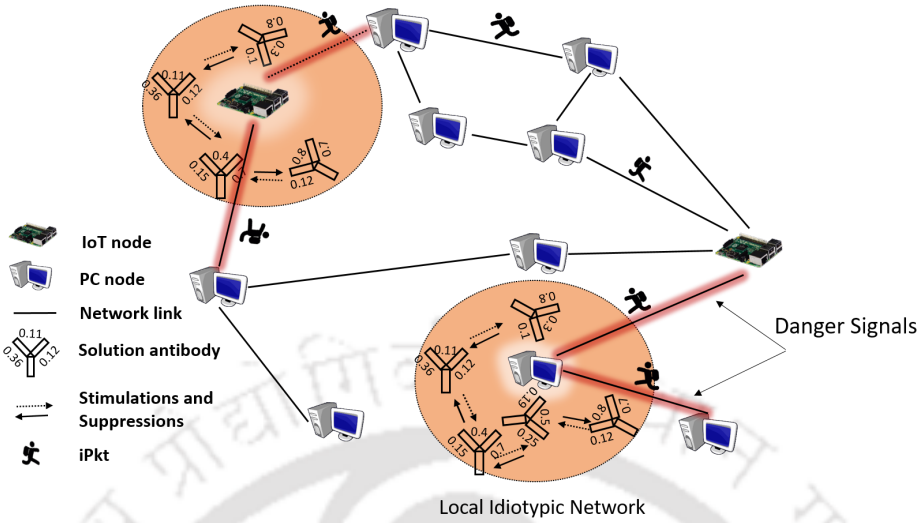


Figure 6.2: Local Idiotypic Networks formed across a *d*CPS

where $\zeta(x, y)$ is a real-valued function used to evaluate the performance of a solution. As can be observed from Equation 6.1, the goal is to search for a mapping such that for a given problem instance, the solution with the maximum performance score is selected. Mapping $\mathcal{M}^*$ also allows *many-to-one* mapping between a subset of problem instances and the best solution. Thus, alternatively, the purpose of the proposed mechanism is to cluster the sets of similar problems instances with centroid being the best solution. Hence, as discussed, the proposed approach maintains a repertoire $\mathcal{S}$ that contains the minimal number of required best solutions. How to share and exploit the knowledge amongst a network of devices is the challenge.

The following sections describe the critical components of the proposed mechanism.

6.4.1 Intelligent Packets Migration

Sharing of information within a collective of nodes is traditionally realized using broadcasting. Though this method of sharing is simple and effective, it consumes a fair amount of energy. Energy conservation is vital in multi-robots and IoT scenarios wherein frequent charging may not be feasible. This chapter therefore extends and use an Intelligent Packets Migration (IPM) scheme as the one proposed in [165].

In this scheme, each mobile agent, dubbed as Intelligent Packet (*iPkt*), decides its migration strategy based on the current state of the environment. They can be

either autonomous or controlled by the participating nodes in the physical network. Each *iPkt* is associated with a unique solution and contains the FV ($\mathcal{F}_y$) and related meta-information about a solution namely an ID, activation value and a performance score. As can be seen in Figure 6.2, the nodes wherein LINs are formed are the ones where problems have currently occurred. Thus, based on the state of the nodes, the *iPkt*s are migrating towards it which is in contrast to broadcasting based methods.

6.4.2 The Proposed Mechanism

The pseudo-code for the overall functioning of the proposed mechanism is given in Algorithm 4. The terminologies and symbols used are explained defined below.

- Node (N): A node N is a low-end computational device which has communication capabilities. A Pi board, robots, laptops, mobile devices, servers, etc. form examples of typical nodes. These nodes when connected to each other, form a physical network $\mathcal{W}$. Nodes may also be envisioned to be connected to sensors and actuators for sensing and control.

- Mapping (M): This mapping is a relation between the two entities viz. the instances in set $\mathcal{P}$ and the associated best solution.

- Affinity (ψ): Affinity ψ is the strength of mapping between the $\mathcal{F}_x$ and $\mathcal{F}_y$. A high ψ value corresponds to a strong mapping and vice-versa.

- Performance Score (ζ): After a solution solves a problem instance, it receives a performance score ζ which provides the measure of the efficacy with which the problem was solved. This score, for instance, could be a sum of output feedbacks such as execution time, memory consumed, etc.

- Activation (τ): It is the overall sum of the stimulations (Sti) and suppressions (Sup) received by a solution.

- Internal Repertoire ($IRep$): The repertoire of solutions and mapping information available within a node are stored in $IRep$.

- External Repertoire ($XRep$): Solutions and mapping information received from other nodes (through sharing) are maintained in $XRep$.

Algorithm 4 The proposed mechanism running concurrently at each node

$\mathcal{P} = \emptyset$; {The set of current problems}
$\mathcal{S} = \emptyset$; {The set of current solutions}
$\mathcal{F}_x$ = FV of problem instance x in $\mathcal{P}$
$\mathcal{F}_y$ = FV of problem instance y in $\mathcal{S}$
while True **do**
    Add new problem instances to $\mathcal{P}$; {if available}
    $DiffuseDangerSignals()$; {diffuse signals to the neighbouring nodes to attract $iPkts$ using equations 6.4 and 6.5}
    $\mathcal{S} \leftarrow \mathcal{S} \cup IRep \cup XRep$; {append the solutions available in the internal and external repertoire to $\mathcal{S}$}
    for all $\forall\ x \in \mathcal{P}$ **do**
        for all $\forall\ y \in \mathcal{S}$ **do**
            $Aff \leftarrow CalcAffinity(\mathcal{F}_x, \mathcal{F}_y)$; {using equation 6.6, calculate the affinities between the FVs of the problem instances and solutions}
            if $psi \geq \psi_{th}$ **then**
                $CandAbs \leftarrow append(y)$
            end if
        end for
        Calculate Stimulations using equation 6.8;
        Calculate suppressions using equation 6.9;
        Update the net activation after stimulations and suppressions using equation 6.10;
        Select the solution having the maximum activation value;
        Execute the solution and get the feedback score;
        If allowed, clone the $iPkt$ associated with the best solution and Hyper-mutate the $\mathcal{F}_y$ of each cloned $iPkt$; {cloning controller clones based on resources}
        Allow the $iPkts$ and its clones to migrate and share the new mappings;
    end for

end while

The proposed approach integrates concepts not only from the Immune Network theory but also from the Clonal Selection and the Danger theories to suit both static and dynamic network scenarios. Nodes may asynchronously receive different sets of problem instances which are needed to be solved using the best available solution. New solutions may also be added to the nodes. The task of the proposed mechanism is to find the mapping which selects the best solution for a given problem instance. Since new problem instances and solutions could crop up or change at the various nodes, the associated mappings may also vary making the search complex. To tackle this dynamic scenario, the proposed mechanism leverages the concepts from Danger theory to first attract the candidate solutions with the required mappings. This is then followed by interaction between the different solutions and problems instances as in Immune Network theory. Finally, the best solution is selected, cloned and the mapping is mutated and shared among other participating nodes. The Clonal Selection theory based mutation and cloning contribute towards the evolution within the

system. The proposed mechanism has three essential elements - *Selective Search*, *Localized Idiotypic Network* and *Decentralized Population Control*. *Selective Search* (steps 7-15) corresponds to the search and screening of the candidate solutions which may have the required mappings to select the best solution. Search is enhanced by emanating danger signals while affinity decides the candidate solutions among all the solutions that are considered to be capable to solve the problem instance. The *Localized Idiotypic Network* is responsible for the interactions between the solutions and the problems as described in Sec. 6.4.2.C and in steps 16-18 of Algorithm 4. After a solution is selected and used to solve the problem instance, it is cloned, mutated and shared. In decentralized and distributed systems, communication cost is desired to be low. Since mapping information of each solution is wrapped and shared using *iPkt*s, excessive cloning of these packets can cause a population to explode. This could result in higher communication cost and may choke the physical network in the worst case. To avoid such networking failures, a decentralized population controller has been used. The metadynamics of the proposed mechanism are presented below.

Metadynamics

A. Danger Signals

Whenever a problem instance which needs to be solved occurs at a node, the node is said to be distressed and diffuses Danger Signals (DGS) across its neighbors. These DGS have been modeled based on the pheromone diffusion mechanism as described by [69]. The signals have two parameters —*strength* (Sg) and *lifetime* (T_{life}) which wane at each hop across the distressed node that emanated it. This forms a DGS gradient, with signals in the outer periphery having lesser strengths. The extent to which these signals diffuse in an *omni*-directional manner is set by a constant called the *signal spanning length* (l_{span}). The signals also carry with them the FV of the antigenic problem instance and penetrate into the neighboring nodes in the form of a diffused signal gradient. When the *iPkts* that move around the network in a conscientious manner [128], encounter these DGS at a node, they compare the FV within them with the FV within the signal at that node. If this amounts to a recognition (i.e., $\psi(\mathcal{F}_x, \mathcal{F}_y) \geq \psi_{th}$), they follow the signal strength gradient towards

the distressed node; else they ignore the signals and move on conscientiously as in [69]. DGS also have a lifetime and thus are volatile.

The dynamics of these DGS are expressed by the DGS gradient (ΔSg) and lifetime gradient ΔT_{life} given by—

$$\Delta Sg = Sg_{max}/l_{span} \tag{6.2}$$

$$\Delta T_{life} = Sg_{max}/l_{span} \tag{6.3}$$

where Sg_{max} is the maximum signal strength diffused between distressed node and its one-hop neighbor.

The strength $Sg(H)$ and lifetime $T_{life}(H)$ of DGS at neighboring nodes with hop-count H is given by—

$$Sg(H) = \begin{cases} Sg_{max}, & \text{if } H = 0 \\ Sg(H-1) - \Delta Sg, & 1 \leq H \leq l_{span} \end{cases} \tag{6.4}$$

$$T_{life}(H) = \begin{cases} Sg_{max}, & \text{if } H = 0 \\ T_{life}(H-1) - \Delta Sg, & 1 \leq H \leq l_{span} \end{cases} \tag{6.5}$$

After the solutions have reached the distressed node, an affinity measure is used to filter out the candidate solutions, as described in the next subsection.

B. Affinity

An antibody is said to recognize an antigen if its paratope has a high affinity with the epitope of the antigen. In the BIS, affinity is based on the degree of complementarity between the shapes of the paratopes and the epitopes. In the computational realm, instead of complementarity, the concept of similarity is often defined as the affinity measure and could be taken to be proportional to the Euclidean distance between the FVs the problem instance and the solution. Affinity $\psi(\mathcal{F}_x, \mathcal{F}_y)$ between

the $\mathcal{F}_x$ of the problem instance and $\mathcal{F}_y$ of the solution is given by,

$$\psi(\mathcal{F}_x, \mathcal{F}_y) = \psi_{max} - \sqrt{\Sigma_{i=1}^{n}(\mathcal{F}_{x_i} - \mathcal{F}_{y_i})^2} \tag{6.6}$$

such that, $\psi_{max} = \sqrt{n}$ where n is the length of the FV and $i \leq n$.

A solution is a candidate solution for a problem instance, if the affinity ψ between the associated FVs is greater than or equal to an affinity threshold ψ_{th}, given by

$$\psi_{th} = \mu * \psi_{max} \tag{6.7}$$

where μ is a system constant. Lower values of ψ_{th} will attract more numbers of candidate solutions towards the distressed node or site of the antigenic attack and vice versa.

Once the candidate solutions have been found, they take part in the interaction among themselves, forming an Idiotypic Network, as advocated in [85]. Since such idiotypic networks are forms concurrently among various nodes, they can be termed as Localized Idiotypic Network (LIN), as described in the following subsection.

C. Localized Idiotypic Network

Even in the absence of new problem instances from the input stream, stimulations and suppressions occur between the solutions concurrently across the nodes in the network thus forming an LIN. Depending upon which solution has the better performance, rewards and penalties in the form of Stimulations (Sti) and Suppressions (Sup) are added to the activation (τ) of the interacting solutions. In the computational world, the nodes connected to a network are the local interaction sites where the $iPkt$s carrying mapping information of the solutions, interact and exchange their respective Sti and Sup.

A solution gets stimulated by another if its ζ is better than the other, else it gets suppressed. The stimulations Sti_i^j received by solution y_i from the solution y_j is given by—

$$Sti_i^j = \psi(\mathcal{F}_{y_i}, \mathcal{F}_{y_j}) + \alpha * \tau_j, \ 0 < \alpha \leq 1 \tag{6.8}$$

where, $\psi(\mathcal{F}_{y_i}, \mathcal{F}_{y_j})$ is the affinity between the solutions y_i and y_j, α is the scaling factor and τ_j is the activation value of y_j.

Similarly, the suppressions Sup_j^i received by y_j from y_i is given by—

$$Sup_j^i = \gamma * \psi(\mathcal{F}_{y_i}, \mathcal{F}_{y_j}) + \delta * \tau_i,\ 0 < \delta \leq 1 \tag{6.9}$$

where γ is the discounted penalty factor, δ is the scaling factor and τ_i is the activation value of y_i.

Activation τ received by an solution y is equal to the sum of total stimulations and suppressions received from the antigens and interacting antibodies. In this work, Farmer's discrete equation model [52] for an idiotypic network has been leveraged to calculate the total activation which is given by—

$$\tau = \tau_{prev} + Sti - Sup \tag{6.10}$$

where τ_{prev} is the activation value of a y after the last LIN interaction.

The solution with the maximum τ value is chosen to solve the problem instance. This best solution then receives the performance score and is eligible to clone and mutate after packing its associated mapping information into an $iPkt$.

D. Cloning and Distributed Population Control

Survival of the fittest defines the rationality of nature. Only those individuals who can compete and survive are allowed to produce off-springs. The rest are eventually purged from the environment. The BIS follows the same rational behavior for the immune cells it harbors. Only those immune cells which are better than the rest in curbing the antigens are allowed to proliferate. While this behavior seems obvious, the mechanisms that facilitate this selective and efficient proliferation within large IC populations in a distributed and decentralized manner within a vertebrate body remain intriguing especially in the absence of a centralized population tracking system. Besides, these biological population control mechanisms also inhibit the exponential growth of the population of the immune cells lest it bloats the whole body.

In the proposed mechanism, the $iPkt$s carrying the mapping and other meta

information of a solution, meet at interaction site within the node. This interaction site is implemented in the form of a queue. When the migrating $iPkt$s reaches a node $\boldsymbol{N}$, they first enter this queue where interactions in the form of Sti and Sup takes place. The total number of clones which an $iPkt$ can produce depends on three factors– the maximum queue length (Q_{max}), activation (τ) and maximum allowed activation (τ_{max}) where Q_{max} is the maximum allowed number of $iPkt$s that can enter the interaction site (queue) and τ_{max} is the maximum value of activation an $iPkt$s can have. The equations behind the population control for a node $\boldsymbol{N}$ are provided below—

$$NC = CloningSpace * \tau/\tau_{max} \tag{6.11}$$

s.t.

$$CloningSpace = Q_{max} - Q_{current} \tag{6.12}$$

where NC is the allowed number of clones for a node, $CloningSpace$ is the maximum possible number of clones for the concerned node and $Q_{current}$ is the current length of the queue.

Based on the conditions above, if an $iPkt$ clones itself, then depending upon the value of NC and minimum activation value (τ_{min}) required to generate a clone, its activation value τ is reduced to the value given by—

$$\tau = \tau_{prev} - NC * \tau_{min} \tag{6.13}$$

Hence, by sensing the number of $iPkt$s already available at an interaction site, the cloning controller through stigmergic means, approximates the global population of other $iPkt$s across the network. If the queue of interaction site is full then it is safe to conclude that the overall network is bloated with $iPkt$s. This forms the basic principle behind the distributed population control.

E. Hyper-mutation

Some of the immune cells undergo somatic *hyper-mutation* [43] causing more effective changes in the surface receptors of their paratopes, thus fortifying the attack on the antigens. This process adds diversity to the BIS and thus aids in the evo-

lutionary process. In the computational world discussed herein, this is achieved by using *affinity-proportional mutation* to candidate solutions that can cater in a better way to new problem instance. A Gaussian mutation function is applied to each of the attributes of $\mathcal{F}_y$ within the cloned $iPkt$. For affinity-proportional mutation, the following exponential function has been used—

$$\lambda(\psi_{norm}) = e^{-\rho * \psi_{norm}} \tag{6.14}$$

where $\lambda(\psi_{norm})$ is the function representing affinity-proportional mutation and ψ_{norm} is the normalized affinity calculated by, $\psi_{norm} = \psi/\psi_{max}$.

Thus, the overall mutation of each attribute of a FV is given by—

$$\mathcal{F}_{y_i}^m = \mathcal{F}_{y_i} + \lambda(\psi_{norm}) * G(0, \nu) \tag{6.15}$$

where $\mathcal{F}_{y_i}^m$ is the i^{th} attribute of hyper-mutated $\mathcal{F}_y$, G(0,ν) is the Normalized Gaussian Probability Distribution function and ν is the desired variance.

F. Apoptosis Life time of an $iPkt$ is represented as the remaining number of hops it can take. Activation also represents the amount of energy within a cell. Both of these factors decide whether or not an $iPkt$ is allowed to be reside in the network. With each hop to a node, the time to apoptosis Γ is decreased by unity and is given by–

$$\Gamma(H) = \begin{cases} \Gamma_{max}, & \text{if } H = 0 \\ \Gamma(H-1) - 1, & 1 \leq H \leq \Gamma_{max} \end{cases} \tag{6.16}$$

An $iPkt$ is purged out of the system if,

$$(\Gamma(H) = 0) \vee (\tau < \tau_{min}) \tag{6.17}$$

The mechanism proposed in this chapter are designed to cater to the resource-constrained devices. The next section provides the computational complexity of Algorithm 4.

6.4.3 Time Complexity Analysis

The equations presented above are linear having a maximum time complexity equivalent to $\mathcal{O}(n)$. Lines 6 - 8 and 16 - 22 all adds up to the complexity of $\mathcal{O}(n)$. Loop from lines 9 - 15 have a worst case time complexity of $\mathcal{O}(n^2)$. This loop though appears to be the most resource intensive part of Algorithm 4, however, since the environment is distributed, the complexity of the loop is shared across the nodes. Greater the nodes, lesser will be the complexity of the loop. Thus, the overall complexity of Algorithm 4 varies between $\mathcal{O}(n^2/N)$ to $\mathcal{O}(n^2)$.

6.5 Experiments and Results

This section discusses the experiments and results from both the emulation and real-robots scenarios.

6.5.1 Emulation

The problem of sorting of input sequences (problem instances) at different nodes of a *d*CPS has been chosen as a problem to test the efficacy of the proposed mechanism. This section initially discusses the problem and its features and then present the experiments and results obtained.

Sorting is one of the most basic and practical problems in the computational domain. According to [108], more than 25% of CPU time is involved in sorting operations. Sorting algorithms can have different time complexities depending upon the type of input data. For example, in general, it is known that Insertion-Sort is worse than Quick-Sort. However, Insertion-Sort performs better when the input comprises nearly sorted permutations of sequences of numbers. It can be inferred thus that the choice of the algorithm is dependent on the characteristics or features of the problem instance. In the work reported in [75], the author has used a total of four features to form the FV. These are - the length ($\mathcal{L}$), the Number of Inversions (INV), the Number of Runs (RUN) and the Longest Ascending Subsequence (LAS) where, $\mathcal{L}$ denotes the length of the input sequence, INV is the total number of inversions present in a sequence, RUN is the maximum number of continuous ascending subsequences and LAS is the length of longest ascending subsequence.

Table 6.1: System parameters and their values

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
μ	0.5	Sg_{max}	100	l_{span}	2	ρ	2
α	0.4	δ	0.2	σ	1	ν	0.6
Γ_{max}	1000	τ_{min}	5	τ_{max}	20	v	5

INV ranges from 0 for a completely sorted sequence to $\mathcal{L} * (\mathcal{L} - 1)/2$ for a reversely ordered sequence. Similarly, RUN is 1 for a sorted permutation and is equal to $\mathcal{L}$ for a reversely ordered sequence while LAS is $\mathcal{L}$ for a sorted sequence and 1 when the same sequence is in reverse order. Thus, the shape space to be chosen in the problem of sorting is real-valued. In the experiments, the same set of features as in [75] have been used and thereby, the value of n is equal to 4. It may be noted that feature extraction is itself a challenging task. Fortunately, with advancement in various feature extraction and machine learning methods, it is possible to represent a given problem instance with its features. In the course of the experiments, it is assumed that the features are available as a part of the dataset.

Though decentralized and distributed systems have their advantages, experimental evaluation becomes difficult since there is no central authority. Hence, one separate dedicated computer node served as a log server to record events such as sharing, executions, etc. The results are thus presented with reference to a *step-count* managed by the log-server. It may be noted that the log-server does not interfere with the functioning of the proposed approach. Each of the experiment is repeated 10-times and the average of all the trials is reported.

6.5.2 Experiment on a 10-node physical network

The first set of emulation experiments were performed on a 10-node network connected in a grid topology and created using 5 Personal Computers (PC), each running two instances of *Tartarus* [163, 167], a multi-agent platform, developed in-house. *Tartarus*[Ω] facilitates the building of an overlay network and supports programming and cloning of mobile agents [113, 211]. The $iPkt$s are implemented using mobile agents [113] though other variations such as peer-to-peer messaging could also be used. Each instance of *Tartarus* represents a node running on an IP.

[Ω] https://github.com/tushar-semwal/ProjectTartarus

Unique Port numbers are used for TCP/IP socket connections between the nodes. Initially, the Intrinsic Repertoire (*IRep*) is fed with four algorithms as the potential solutions – Quick-Sort (QS), Insertion-Sort (IS), Shell-Sort (ShS) and Selection-Sort (SelS). The problem dataset is divided into four sets– $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$. $\mathcal{P}_1$ comprised all 40,320 permutations of length 8 ($^8P_8$) from the set {1,2,3,4,5,6,7,8}. Set $\mathcal{P}_2$ contained 2000 randomly permuted sequences of length 1000 each. $\mathcal{P}_3$ constituted a set of 2000 nearly-sorted permutations of length 1000 each and $\mathcal{P}_4$ contained 2000 randomly permuted input sequences of random lengths varying from 8 to 1000. Feature vector $\mathcal{F}_y$ carried by each $iPkt$ is randomly initialized using one of the FVs from the problem dataset chosen. Algorithm 4 was used to conduct the experiments. Table 6.1 provides the values of the constant parameters used during the course of the experiments.

The following approach has been adopted for better clarity and ease in the analysis:

- Since the proposed mechanism does not have a training phase, a fixed number of problem instances are randomly chosen from each of the problems sets $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$ and are then introduced to half of the total number of nodes in a random manner.

- After specific stopping criteria is met, the system's learned knowledge is tested on the remaining number of problem instances.

- The trend in the learning of the mechanism and the number of quality solutions selected is then analyzed and reported in this chapter.

In this experiment, at each iteration of the Algorithm 4, antigens in the form of input sequences along with their FVs were allowed to appear in a controlled and defined manner, at random nodes concurrently (this thesis will refer to this as one barrage of antigenic attacks). The antigenic attacks caused the associated node to diffuse the DGS to its neighbors. Each experiment was repeated 10-times and the average of the results are reported. Figure 6.3a, 6.3b, 6.3c and 6.3d provides the plots for the population of the algorithms on different antigenic set of input sequences $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$ respectively. For convenience, $iPkt$ containing map-

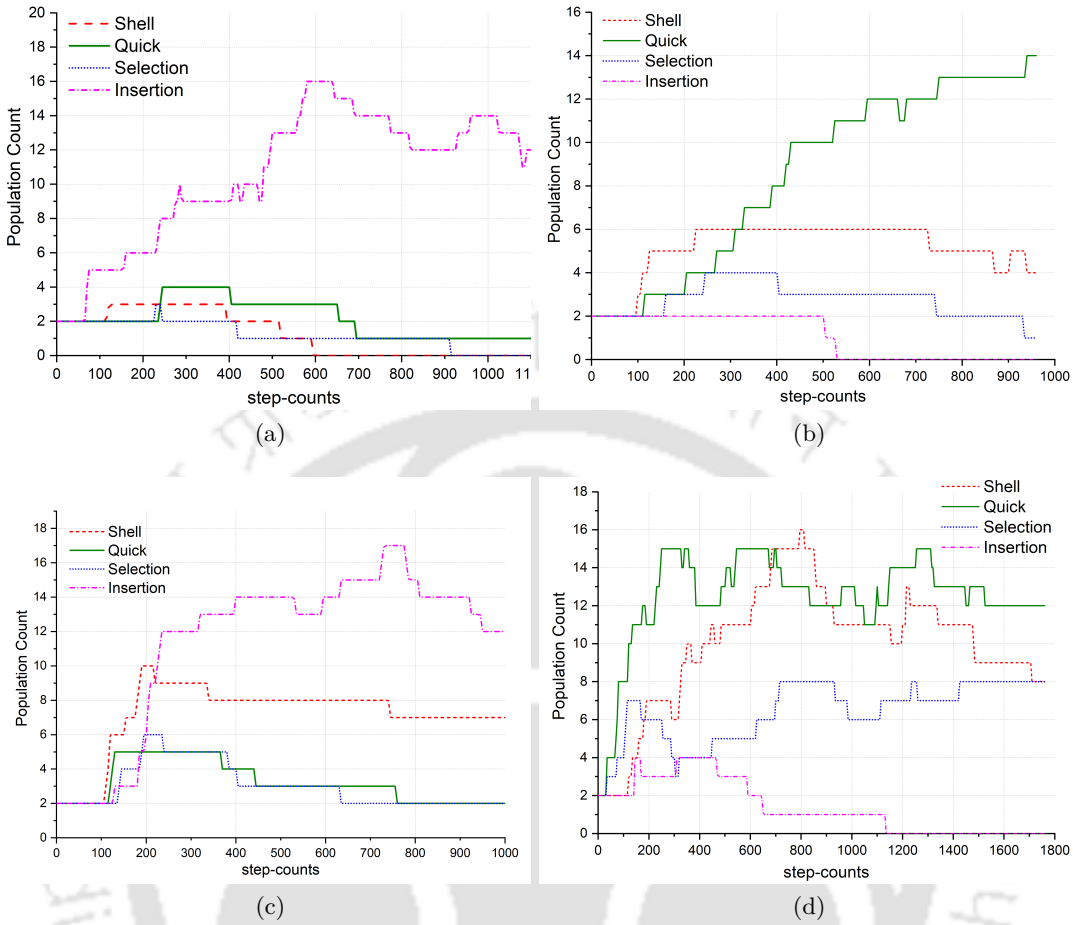


Figure 6.3: Population curves for a 10 node network: (a) Problem instances comprising all possible permutations of length 8 (b) Problem instances comprising randomly permuted sequences of length 1000 (c) Problem instances comprising nearly sorted sequences of length 1000 (d) Problem instances comprising randomly permuted sequences of length varying from 8 to 1000

ping information associated with Insertion, Shell, Quick and Selection Sort will be referred to as IS-$iPkt$, ShS-$iPkt$, QS-$iPkt$ and SelS-$iPkt$, respectively.

It may be noted that the population count represents the trend in the learning of the system running the proposed mechanism. Since the nodes select only those solutions with highest populations, an increase in the population of a particular solution across the network signifies that the nodes have arrived at the best solution for the corresponding problem instance.

Problem set $\mathcal{P}_1$: As can be seen from Figure 6.3a, the population of IS-$iPkt$s emerged out to be the winner by increasing in population while suppressing those carrying the other algorithms. This happened because Insertion-Sort performed

better than the other algorithms and hence received more rewards and therefore stimulations from the $iPkt$s of the other populations (ShS-$iPkt$s, QS-$iPkt$s and SelS-$iPkt$s), causing an overall increase in its population (concentration) in the network. As a result, the others which performed badly received more of penalties and were suppressed greatly by the IS-$iPkt$s. Repeated penalties and hence suppressions from IS-$iPkt$s, drastically lowered the population of those $iPkt$s which were carrying mappings of badly performing sorting algorithms (ShS, QS, and SelS). It is thus clear that after step-count = 500, the remaining population is dominated by IS-$iPkt$s. It may also be noted that the trends show an on-demand selective domination in the better/ best-performing algorithm in a distributed manner across the network. At step-count = 850, a copy of a previously fired barrage of antigens was introduced in the network causing a secondary response [43] as can be seen from the peak at around step-count = 980. Beyond this, the population of the best $iPkt$s (IS-$iPkt$s) seems to wane, indicating that the rate of rewards and hence stimulations are dying down due to the absence of any antigenic attacks. Beyond step-count = 1000, only the best $iPkt$s increase in number to contain the fresh attack. The other $iPkt$s do not tend to increase in population as they are by no means effective in containing the attack by the same kind of antigen.

The results presented reveal some interesting aspects. Though it is well known that Insertion-Sort is theoretically worse than Merge-Sort and Quick-Sort, there is a rapid increase in the population of IS-$iPkt$s (Figure 6.3a) indicating that it is the best choice when the input problem instance to be sorted is of shorter length. One may thus infer that the proposed mechanism can wean out the best-suited algorithm (viz. Insertion-Sort) for set $\mathcal{P}_1$ as confirmed by [75]. What is more interesting is that the population of the $iPkt$s representing the better/best algorithm suppresses those of the others carrying the less performing ones. The winning population thus emerges concurrently across the network in a distributed and decentralized manner.

Problem set $\mathcal{P}_2$: Figure 6.3b shows the plot for data set $\mathcal{P}_2$ which contains randomly permuted input sequences of length 1000. As can be seen, after the first barrage at step-count = 80, all populations seem to increase except that of IS-$iPkts$. After step-count = 120, the population of ShS-$iPkts$ and SelS-$iPkts$ also

Table 6.2: A summary of the results on the 10-node network

Problem set	Type	Known best	Achieved best
$\mathcal{P}_1$	All permutations of short length sequences	Insertion Sort	Insertion Sort
$\mathcal{P}_2$	Randomly permuted long length sequences	Quick Sort	Quick Sort
$\mathcal{P}_3$	Nearly sorted long length sequences	Insertion Sort	Insertion Sort
$\mathcal{P}_4$	Randomly permuted sequences of length varying from 8 (short) to 1000 (long)	Quick Sort	Quick Sort

seem to stagnate. At step-count = 470, it is evident that QS-$iPkts$ is the best for the kind of data in the set $\mathcal{P}_2$. QS algorithm seems to gain most of the rewards and stimulations while the others eventually decline in population.

It can be seen that for instance set $\mathcal{P}_2$, there is no increase in the population of IS-$iPkts$. This emphasizes the fact that these $iPkt$s received suppressions from others and hence were not allowed to proliferate into the network. For the first three barrages, QS-$iPkts$, ShS-$iPkts$ and SelS-$iPkts$ underwent multiple interaction scenarios. The first scenario is when they all received stimulations from the IS-$iPkts$ as all of them carry algorithms which are definitely better than the Insertion-Sort for the set $\mathcal{P}_2$. The second scenario is when SelS-$iPkts$ and ShS-$iPkts$ interacted and mutually stimulated one another. Since QS-$iPkts$ is the winner the chances of receiving stimulations from it are rare. The third kind of interaction is where QS-$iPkts$ received stimulations from ShS-$iPkts$, IS-$iPkts$ and SelS-$iPkts$. After the fourth barrage, QS-$iPkts$ seems to dominate and win the solution space clearly. It can be further analyzed that the IS-$iPkts$ are heavily suppressed by the other $iPkt$s due to which their population dwindles.

The initial rise in the population of other three algorithms is due to the fact that they received stimulations from IS-$iPkts$ but eventually QS-$iPkts$, due to their high valued performance, rapidly proliferated and suppressed other types of $iPkt$s.

Problem set $\mathcal{P}_3$: In Figure 6.3c, antigenic data was in the form of nearly sorted sequences of size 1000. The populations of both ShS-$iPkts$ and IS-$iPkts$ seem to increase initially due to their higher performances but are soon dominated by that of the latter. The increased population causes increased proliferation across the

network, thus suppressing the populations of SelS-*iPkts*, QS-*iPkts* and ShS-*iPkts*. When the observations with that from Figure 6.3a are coupled, one may conclude that Insertion-Sort seems to be the best option when it comes to sorting of small length sequences or nearly sorted long length sequences. Further, Figures 6.3b and 6.3c show that Shell-Sort performs satisfactorily for instances from sets $\mathcal{P}_2$ and $\mathcal{P}_3$, possibly indicating that this algorithm could emerge as the winner when it comes to problem instances comprising long length nearly sorted or randomly permuted sequences. However, the same requires more investigations and thus provides scope for future work.

Problem set $\mathcal{P}_4$: Figure 6.3d shows the population graph when the length of the input sequences varies from 10 to 1000 and the sequences are randomly permuted. Since the data set $\mathcal{P}_4$ contains a mix of different types of input sequences, it is clear from the figure that both ShS-*iPkts* and QS-*iPkts* have about the same population size. It is then at step-count = 1300, QS-*iPkts* is the winner. It may be noted that since the antigenic sequences taken out of the set $\mathcal{P}_4$ could have had drastically different features, other algorithms such as Selection-Sort also seem to increase in population. In brief, the heterogeneity of the input data can cause multiple algorithms to dominate in a network.

A summary of the results on a 10-node network is shown in Table 6.2. As can be seen from the table, the results obtained validates the correctness of the proposed mechanism.

6.5.3 On-the-fly Addition of Solutions

The previous experimental results show the case wherein the number of solutions (viz. algorithms) was fixed *a priori*. Each of the 10 nodes contained all the solutions and the task was to find a mapping such that the best solution was selected for a given problem instance. Though this may seem trivial, the experiments validated the correctness of the proposed approach. A new set of solutions could also be added to the existing repertoire of solutions, on-the-fly. It may be noted that, the system will now find mappings taking these new solutions too into consideration. Figure 6.4a shows the learning phase when new solutions were added to the repertoire randomly

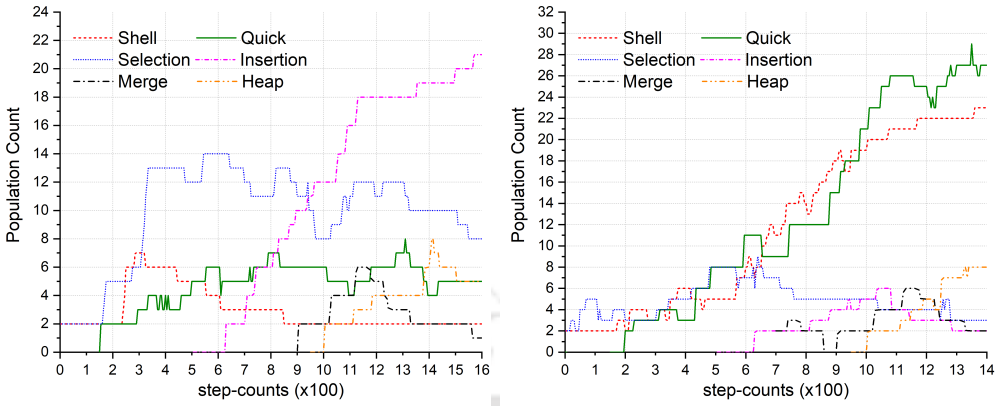


Figure 6.4: On-the-fly addition of new solutions for the experiment on problem instances comprising sets (a) $\mathcal{P}_1$ and (b) $\mathcal{P}_4$

for the problem set $\mathcal{P}_1$. As can be seen, initially there were only two solutions (viz., Shell and Selection) and the mechanism found the best (Selection) among them. It may be observed that at step-counts 150, 620, 900 and 1005, the new solutions Quick, Insertion, Merge and Heap respectively, were added. The addition of these solutions changed the dynamics within the system and made it adapt and arrive at the new best solution viz. Insertion. Figure 6.4b shows the case when the set $\mathcal{P}_4$ which is a mix of sequences having different features, yields multiple best solutions. As can be seen from Figure 6.4b, for set $\mathcal{P}_4$ both Quick and Shell sort emerged to be the best solution.

The mechanism was thus capable of finding the best solutions from amongst the old and newly added ones. Freshly added solutions tend to change the course of mappings thereby exhibiting the adaptive and self-organizing abilities of the proposed mechanism.

6.5.4 Experiments on a 50-node physical network

To understand the behavior of the mechanism when scaled, experiments using a network of 50 nodes deployed over 10 PCs with each PC hosting 5 instances of the Tartarus platform, were performed. The system constants were taken to be the same as in Table 6.1. Experiments using $\mathcal{P}_2$ and $\mathcal{P}_3$ data sets, the results of which have been portrayed in Figures 6.5a and 6.5b respectively. As can be seen, the trend is

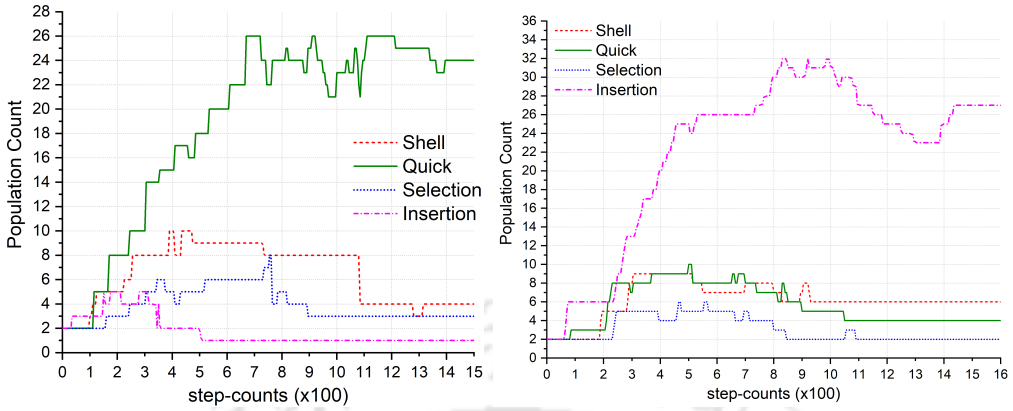


Figure 6.5: Population curves for the experiment on problem instances comprising sets (a) $\mathcal{P}_2$ and (b) $\mathcal{P}_3$ repeated on a 50-node network

similar to that in Figures 6.3b and 6.3c with Quick-Sort and Insertion Sort having the maximum population counts, respectively. Since the increase in the number of nodes in the physical network did not alter the behavior of the system, one may conclude on the scalability of the mechanism.

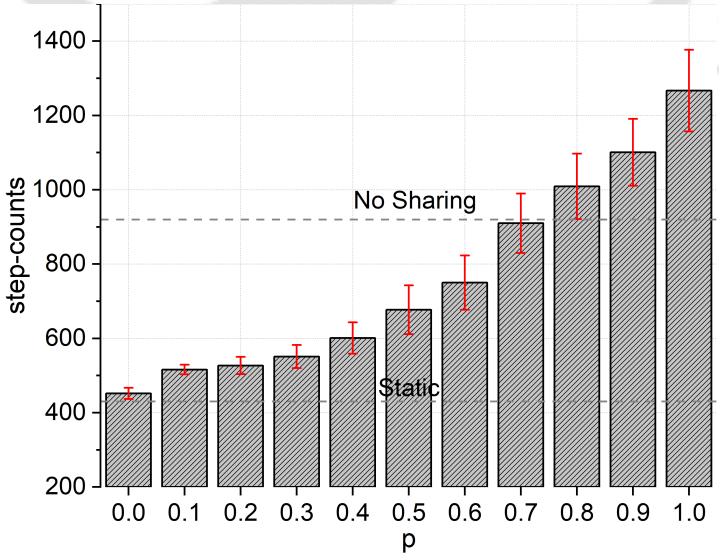
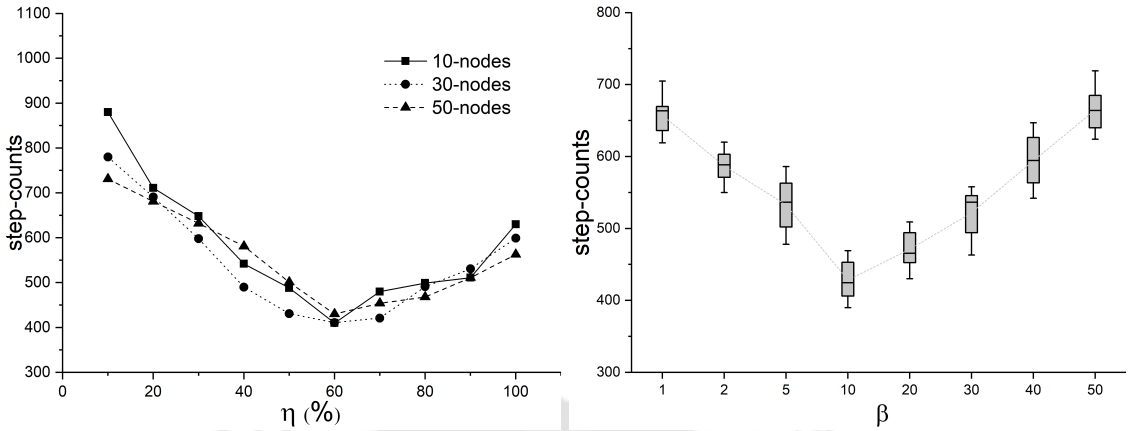


Figure 6.6: Performance in a 50-node dynamic network for the experiment on set $\mathcal{P}_4$



(a) (b)

Figure 6.7: Effect of (a) Influx Rate and (b) Barrage Size on learning

6.5.5 Performance in a Dynamic Network

A dynamic network is a physical network wherein the nodes may connect or disconnect randomly thereby changing topology. In all the previous experiments, the network was assumed to be static with no new nodes being added or removed from the system. However, in real-world scenarios, this may not be the case. For instance, in a Mobile Ad-hoc Network (MANET), the nodes are mobile and thus could easily connect or disconnect from the network. Depending on the application, this dynamism could be high or low. To study the performance of the proposed mechanism in dynamic networks, a 50-node physical network wherein nodes could break off and re-connect to the network with a probability p, was used. The value of p was varied from 0 (static network) to 1 (highly dynamic network). $p = 1$ represents a case where a node will break or reconnect 10 times in a second. Figure 6.6 shows the number of step-counts taken to converge on set $\mathcal{P}_4$, for different values of p.

As can be seen, the convergence is faster for lower values of p. As p increases, the number of step-counts also increases. Due to the high dynamism ($p \geq 0.7$), the probability of diffused DGS getting severed is high. This in turn causes the $iPkt$s containing the mappings to be shared, to be misled thereby delaying convergence.

6.5.6 Effect of Influx Rate

The Influx Rate (η) is the percentage of the total number of nodes which receive a barrage of problem instances at any instant of time. Figure 6.7a shows the learning trends for different values of η. As can be seen from the figure, as η increases, the number of step-counts required for convergence decreases till around a value of 60% in this case. This is so because as η increases, more number of nodes become involved in finding the solution for the given problem instance, thereby increasing the rate of sharing and hence convergence.

However, beyond $\eta = 60\%$, the step-count increases thereby slowing the convergence. This is because due to high η, the number of $iPkts$ generated by the corresponding nodes, increases. On a limited bandwidth scenario, such an effect can choke up the network thereby slowing down the convergence [65].

6.5.7 Effect of Barrage Size

Barrage Size β, the number of problem instances per barrage, has a direct effect on the learning rate of the proposed mechanism. For low values of β, it is obvious that generalization and hence learning is difficult. On the other hand, for large values of β, the mechanism may try to overfit which could mean a waste of computational resources. This can be seen in Figure 6.7b where the number of step-counts required to find an appropriate mapping decreases with an increase in β initially. This trend lasts up to a certain point when $\beta = 10$. Beyond this, the mechanism requires an increasing number of step-counts for convergence. One may conclude that the range of values of β for optimal performance is application-specific and may require to be found empirically.

6.5.8 Effect of Danger Signals

If no Danger Signals (DGS) were diffused by a distressed node, the $iPkt$s would possibly be mislead and many a time be late in reaching such nodes. The presence of DGS facilitates selective guidance of these $iPkt$s towards the distressed node. $iPkt$ are inherently programmed to compare the affinity between the feature vectors, $\mathcal{F}_y$ and $\mathcal{F}_x$. If this affinity is higher than the affinity threshold τ, the $iPkt$ starts to move

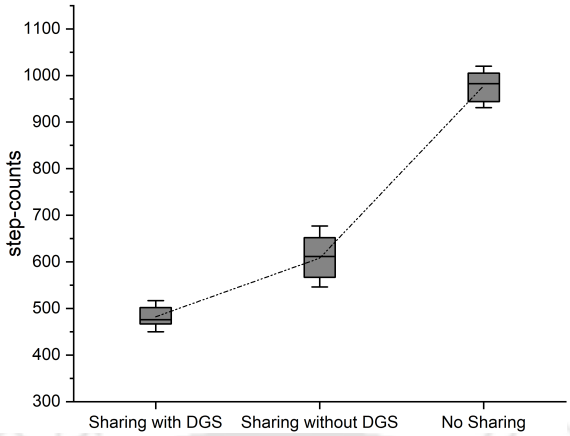


Figure 6.8: Effect of Danger Signals (DGS)

along the danger signal gradient towards the distressed node. Due to the presence of the signal gradient, they tend to move towards the distressed node along the shortest path. If such signals were to be diffused farther into the network, the amount of time taken by the pertinent $iPkt$ to reach the distressed node would be drastically reduced. However, this could be computationally expensive and at times a futile exercise in case of dynamic networks. Figure 6.8 depicts boxplots for step-counts for both cases viz. with and without the use of DGS. As expected, the mean step-count required for finding the best solution when DGS are used is less than that when these signals are not used. Further, it can be seen from the boxplots that when the proposed approach does not use DGS, the variance in the step-counts required to find the best solution, is high. This, as already mentioned, is because of the absence of any selective guidance offered to the $iPkt$s which results in an increase in the convergence time. On the contrary, the use of DGS, causes the system to arrive at the best solutions in almost the same time increasing repeatability to an extent. For the reference, a boxplot of step-counts when sharing is disabled is also shown in Figure 6.8.

6.5.9 Experiment on an IoT scenario

As a proof of concept, the proposed mechanism was also tested on a 50-node network formed within 10 computational devices. Each of these 10 devices was attached to a Temperature sensor so that it constituted an IoTized network. The data from these 10 devices were distributed to the emulated nodes formed within them, with

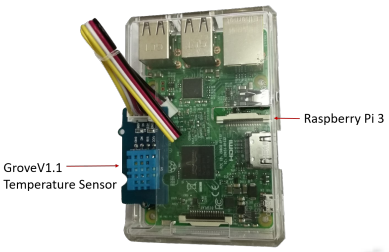


Figure 6.9: An IoT node comprising a Grove V1.1 Temperature sensor and Raspberry Pi 3

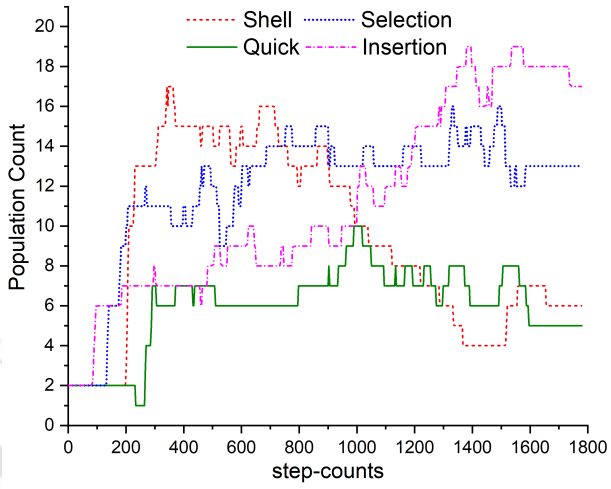


Figure 6.10: Population curves for a 50-node IoT

each containing 5 nodes. Temperature sensors were of the Grove V1.1 make (shown in Figure 6.9). The experiments performed comprised sorting the sensory values obtained from these sensors at each of the 50 nodes. Figure 6.10 presents the results of the experiment in a graphical manner. As can be seen, during the initial learning phase of the mechanism, the population of all the four different algorithms increased but eventually, Insertion-sort and Selection-sort emerged as the winners. Analysis of the raw sensory data revealed that the values of $\mathcal{INV}$ and $\mathcal{RUN}$ were low which means that the values generated by the temperature sensor are usually the same for fairly large and continuous periods of time. This is so because the ambient temperature changes at a very slow rate due to which the data is most often nearly sorted. This means that Insertion-Sort should be the winner. The results obtained from the experiment authenticate the same.

6.5.10 Comparison with the non-sharing approach

A system which shares the knowledge with the participating entities will generally converge faster than the one where the entities learn in isolation. To demonstrate the efficacy of sharing among peers, a comparison of performance of the proposed mechanism with and without sharing has been shown in Figure 6.11.

When sharing is absent, different nodes will arrive at the best solution at different step-counts. In Figure 6.11, each dotted line thus, represents the trend in

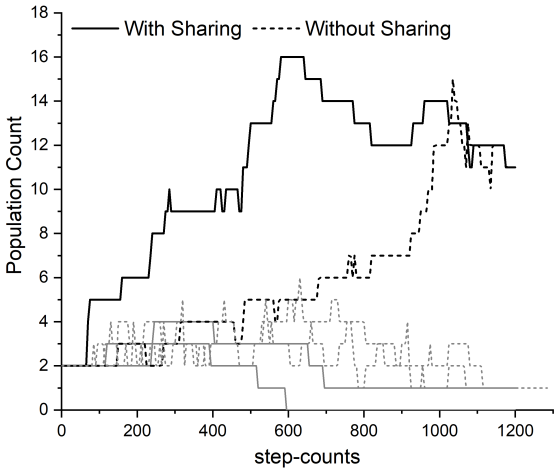


Figure 6.11: Comparison of results: With and Without sharing

the average of the populations of a particular solution taken across all the nodes. As can be observed, the best solution (dotted black lines) was found at the 960th step-count which is far later than that when sharing was used (step-count 480 for solid black lines). The faster convergence results due to the fact that sharing most of the time allows the best mappings to reach the nodes while the search is still on. Sharing thus makes the system act akin to a typical parallel and distributed search scenario. For clarity, only the best solutions for each of the case (i.e., with and without sharing) are shown in dark lines while the population of other solutions is grayed out.

6.5.11 Experiments using Real-Robots

To test the efficacy of the proposed mechanism in real-world scenarios, it is essential to embody it onto real situated robots. This section presents a proof-of-concept of the mechanism running on a physical network of real-robots.

With a surge in the popularity of e-commerce, companies such as Amazon and Flipkart maintain large warehouses to store their inventory. In conventional Warehouse Management Systems (WMS), a human operator uses manual loading vehicles to pick and place items within. Robots, from Kiva and GreyOrange, are now been widely used by such e-commerce firms to smoothen the process of picking, packing and delivering items ordered online. As soon as an order arrives, a centralized entity

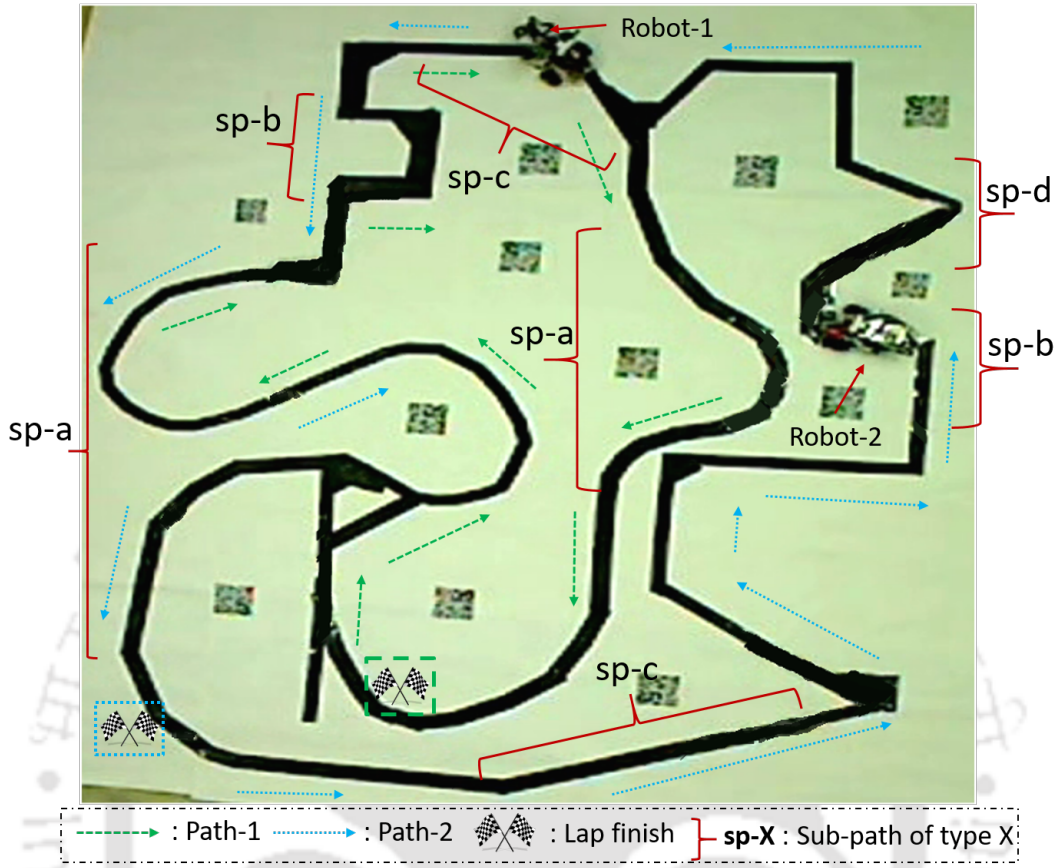


Figure 6.12: Experimental arena with robots, paths and their sub-paths

decides a path for the robot to follow so that it can reach the rack which contains the corresponding item. QR codes laid on the floor of the warehouse aid in guiding the robots to their respective destinations. Depending on the characteristics of the path created, the robots may need to adapt their velocities to ensure that they stay on the path or track. Further, based on the weight of the rack and the type of items within, the robot may also need to minimize any form of vibrations.

The robots populating such scenarios may encounter different types of sub-paths such as straight, curved, U-turns, etc., as they move. Hence, they may need to use different strategies or *Path-Following Algorithms* (PFA) to cater to these sub-paths. They could choose from a dynamic-set of PFAs and eventually adapt by finding out the best one for a given sub-path. Considering the changing nature of the environments the robots work in, formulating a generic PFA which can cater to all the sub-paths, is a difficult problem. On the contrary, designing a system which receives a stream of PFAs and then selects the best suited for the given sub-

Table 6.3: Sub-paths and their features

sub-path (sp-X)	Description	Linear-ity	Curva-ture	90°	Sharpness
sp-a	curved path	0.3	0.7	0	0.2
sp-b	path with 90° turns	0.5	0.3	0.9	0.7
sp-c	straight path	0.7	0.2	0	0
sp-d	acute turns	0.4	0.5	0.3	0.9

path, would be simpler and viable. Thus, the proposed mechanism is mapped into a warehouse scenario to try and solve this issue. In this context, the solutions are the different PFAs (or heuristics or rules) while the problem instances constitute the paths having different features (linearity, curvature, 90°, sharpness of turn as shown in Table 6.3). Figure 6.12 depicts a small warehouse area of size 3m x 3m populated by two real-robots used to prove the practical viability of the proposed mechanism experimentally. The black strips act as the paths to be followed. Two main paths, Path-1 and Path-2 and their directions are shown using differently colored (green and blue) arrows. As can be seen, each path is composed of several sub-paths with varying linearity, curvature, and 90° turns as described in Table 6.3. For clarity, a few of these sub-paths have been labeled in Figure 6.12. Path-1 is made up of sub-paths *sp-a*, *sp-b* and *sp-c* while Path-2 comprises *sp-a*, *sp-b*, *sp-c* and *sp-d*. It may be noted that sub-paths of the same type need not have exactly the same features. For instance, the sub-paths of type sp-a in Path-1 and that in Path-2 are noticeably different. The figure also shows two separate cross-checkerboard flags, one for each path (green for Path-1 and blue for Path-2), which signify the start and end points of a lap for each of the paths, respectively. Since each path is a cycle, a robot is said to have completed one lap if it reaches the respective start point of the corresponding path.

For experimentation with the real-robots, a pair of LEGO MINDSTORM NXT robots has been used. The robots were based on a two-wheel differential drive with one castor wheel. Two light sensors attached to the front of the robot were used to sense the black path. Since these robots are incapable of hosting the *Tartarus* mobile agent platform on their controller blocks, they were controlled by two different connected computers hosting a *Tartarus* platform each, via Bluetooth. Tartarus

was needed to create and allow the migration of the $iPkt$s carrying the *PFA to sp-X* mapping information. An interface similar to LPA-PRO-NXT [88] was used to control these robots. As a proof-of-concept, the robots were made to follow two different paths (Path-1 and Path-2) simultaneously as shown in Figure 6.12. The information regarding the sub-paths and their corresponding features are initially made known to the robot in the form of a map.

The task of the proposed mechanism was to make the robots execute the algorithms received from an input stream of PFAs, and learn the best *algorithm-to-sub-path-type* mapping. When the robot encounters a sub-path, it emanates *danger signals* to attract those $iPkt$s which carry mapping information for this sub-path type. These attracted $iPkt$s constitute the external repertoire ($XRep$) for that robot. The $iPkt$s already available within the robot form the internal repertoire ($IRep$). The $IRep$ and $XRep$ interact and compete with each other to form a LIN. It may happen that the $XRep$ is empty in which case the LIN is formed due to interactions of $iPkt$ within $IRep$. Finally, the $iPkt$ with the best mapping is selected and the associated PFA is executed by the robot for that sub-path. If the corresponding PFA is currently not available within the robot, then it requests the same from the input stream and executes it. Every time a robot finishes crossing a sub-path, the associated mapping information and the performance measure of the best PFA is packed into an $iPkt$, cloned and shared with the other robot(s). As discussed earlier, the number of clones produced is proportional to the cloning resource available within the robotic node. Since the cloned information percolates concurrently across the physical robotic network, sharing and hence learning is accelerated.

The use of multiple robots in the real-world scenario described herein, triggered the execution of several PFAs for different paths, concurrently in the arena. This accelerated the learning process and based on the features of the path, only the best-matched PFAs were selected. The inherent sharing thus reduced redundancy of trials and all robots eventually learned the best algorithm to be used for each type of sub-path.

Eight PFAs using different strategies, listed in Table 6.4 were used in the experiments. For each lap of a path and each robot, a distinct PFA was randomly chosen and introduced into the robots via Tartarus. Experiments were repeated 10 times.

Table 6.4: PFAs and the strategies used

PFA	Strategy
A1	If both sensors detect white then move straight with maximum speed else switch off either of the motors to turn
A2	If both sensors detect white then move straight with maximum speed else move the motors in opposite directions to turn
A3	Reduce the speed of the left motor by a fixed amount to steer left and vice-versa for smooth movement
A4	Based on the deviation in the light sensor value from the edge of the path, proportionally decrease the speed of either of the motors
A5	Proportional and Derivative (PD) control
A6	Proportional and Integral (PI) control
A7	Proportional, Integral and Derivative (PID) Control
A8	Same as A1 but has an extra case which checks whether both the sensors are away from the black path and takes a sharp turn

The system was reset before every experiment. The graph between the time taken and the lap# for a single experiment is shown in Figure 6.13a. As can be seen, the robots on Path-1 and Path-2 in their very first lap received algorithms A2 and A5, respectively. Since there is no other algorithm available initially, A2 and A5 were executed for all the sub-paths of paths 1 and 2 individually by each robot. Further, as A2 and A5 cannot be the best-suited PFAs for all the sub-paths, the overall time taken for the first lap is considerably high, especially for the Path-2. In the second lap, A1 and A7 are received by the robots on Paths 1 and 2, respectively. By now, PFAs A2, A1, A5 and A7 are tried by the robots concurrently on different sub-paths. The results portrayed in Figure 6.13a being just one experimental instance, it can be observed that there are fluctuations in the times consumed in the initial laps. This is due to the fact that the robots are struggling to adapt and learn the best PFA (solutions) for a given sub-path (problem instance). Eventually, the time per lap stabilizes at around 22 and 32 seconds for the Paths 1 and 2 respectively. It may also be noted that though a maximum of eight PFAs were used, new PFAs can still be injected through the stream on-the-fly. This may cause initial turbulence resulting in higher execution times which will eventually settle as the mechanism adapts using the new PFA(s).

After repeating the experiment 10 times, the average of the time taken to complete a lap of a path by a robot is shown in Figure 6.13b. As can be comprehended

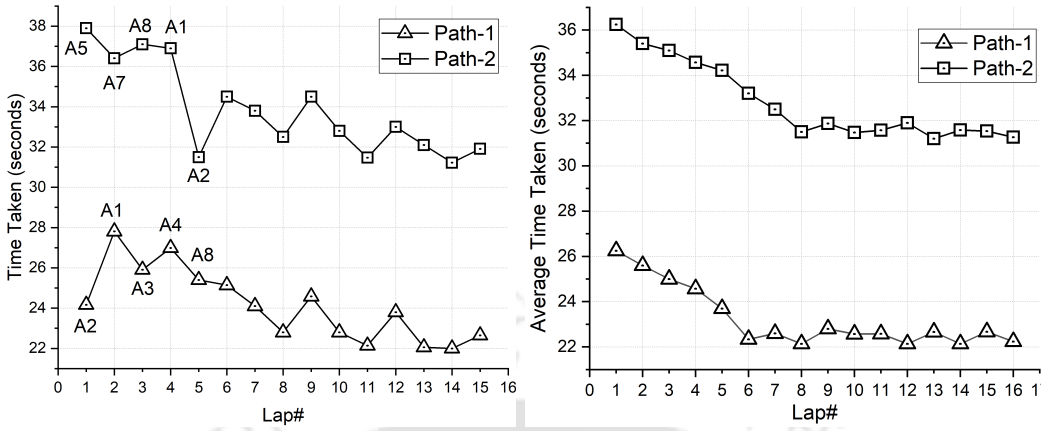


Figure 6.13: (a) Time taken per lap for the two paths in one experiment (b) Average time taken per lap for the two paths

from the figure, initially the robots consumed a higher amount of time since the mappings between the PFA and the sub-path were not yet learned. At laps 6 and 8 for paths 1 and 2, the average time per lap plateaued out at around 22 and 31 seconds, respectively. This is due to the fact that with the increase in the number of laps, the mechanism evolved better mappings and hence consumed lesser times. As can be seen in Figure 6.12, the sub-paths are not structured or well-defined but are handmade which make them rugged. This exerts more selective pressure on the mechanism which in turn authenticates the efficacy of the proposed mechanism in real-world scenarios. The final best mappings between the PFA and the corresponding sub-path found by the mechanism are listed in Table 6.5. It can be seen that only three PFAs were selected to cater to four different sub-path types (A2 is common for *sp-b* and *sp-d*) which necessarily means that the mechanism can also arrive at generic solutions[▶].

It may be observed that the overall execution on the part of the robots is distributed. Further, since there is no central controlling authority, the robots work in a decentralized manner much like the biological immune system. The results thus substantiate the continuous learning and adaptive properties earlier stated to be presented in the proposed mechanism.

[▶] A video can be accessed via the link: https://goo.gl/EzVUyo

Table 6.5: Best PFAs found for a given sub-path type

sub-path	**Best PFA found**	**Remarks**
sp-a (curved path)	A4	Since the change in acceleration is proportional to deviations from the path, the movement of the robot is smooth. Hence, A4 is well suited for such curved sub-paths
sp-b (sharp 90° turns)	A2	Moving motors in opposite directions can make the robot rotate about its axis. Thus appropriate for right-angled turns.
sp-c (straight path)	A1	For straight paths, moving at max speed could be a viable solution as compared to other more complex PFAs.
sp-d (sharp acute turns)	A2	Since both sp-b and sp-d are sharp turns, A2 seemed to the best option.

6.6 Chapter Summary

An Immunology-inspired approach to find the best solution(s) for a given set of problems in a decentralized and distributed manner has been portrayed in this chapter. Mechanisms from the Clonal Selection theory, the Immune Network theory, and the Danger theory, have been combined to selectively search the mapping(s) associated with the best solution(s). Using sorting as a problem in an emulated distributed scenario, an extensive analysis of the results obtained in both static and dynamic networks have been discussed. The approach also proved to be scalable in terms of both the number of solutions and the number of nodes. On-the-fly addition of new solutions allows the system to adapt and find better mappings for the given problem. Sharing eventually induces all nodes or robots to have the best mappings between the problem and the solutions. If we now envisage a scenario wherein a node from such a converged network, say N_1, is removed and added to another network, say N_2, the better-performing solutions in the repertoire of this newly added learned node could proliferate and transfer knowledge to nodes within N_2, thereby hastening convergence in N_2. This implies that it is possible to transfer the knowledge learned by one system to another. In addition to emulation, a proof-of-concept of the proposed system using multiple real-robots in a warehouse

scenario has also been presented. This approach could also be used in the domain of autonomous and connected Vehicles. The paths in a warehouse could be looked upon to be synonyms to the roads while the features of the sub-paths could be grabbed using maps such as those hosted by Google. Depending on the road conditions, connected vehicles could then share their driving algorithms and arrive at better ones.

The approach presented in this chapter makes an inherent assumption that the stream of solutions, from which the best one is to be selected, is already available. However, in the real systems, solutions suitable for the underlying problem are required to be generated and evolved from scratch. This calls for a mechanism to not only select but also evolve new solutions which can be executed for the problem encountered. Though the mechanisms for the evolution of solutions along with selection for closed-world scenarios have been discussed in [176], modifying them to suit real decentralized and distributed environments, still remains a challenge. For a CPS of robots with an objective to learn a task in one such environment would mean that it needs to evolve a set of robot-controllers (solutions) and then select the best among them for the concerned task. The application areas for the proposed approach include terrain exploration, warehouses, and service robots, wherein it is desirable to extend the running time of a robot by reducing the related energy consumption. Hence the mechanism should also be energy efficient and use intelligent communication techniques to work under power constraints. The next chapter describes one such mechanism to provide embodied evolution cum selection of robot-controllers in a decentralized and distributed system of multiple robots. The mechanism leverages the use of mobile agents to transfer the information among the robots efficiently.

"Don't say you don't have enough time. You have exactly the same number of hours per day that were given to Helen Keller, Pasteur, Michelangelo, Mother Teresa, Leonardo da Vinci, Thomas Jefferson, and Albert Einstein."

H. Jackson Brown Jr. (1991 – 1994)
American author

7

Augmenting Embodied Evolution

While in the biological realm, survival may be the only intrinsic motivation behind *evolution*, in the field of robotics, the accomplishment of a set of tasks forms an additional component. These two motives together are deemed necessary in both single and multi-robot scenarios. Due to their inherent resistance to noise and smooth input to output mapping, several researchers have used Artificial Neural Networks (ANN) [54] to evolve controllers (solutions) for such a CPS of robots [134].

Traditional evolutionary approaches incorporate an iterative evolutionary search for a single *monolithic* robot controller which can aid a robot to achieve the desired tasks. The search for a single optimal robotic controller is performed using several iterations that involve multiple runs of the associated algorithm on centralized (and sequential) simulation environments. This separately evolved controller is then embedded into real robots. Single controllers have been evolved to implement different tasks such as obstacle avoidance, gait learning and search [134]. However, bootstrapping the evolutionary process becomes difficult especially when the tasks to be learned are complex for a single controller to cater to. In such scenarios where a complex task may contain bootstrap [71] and deception [195] issues, researchers have opted to employ *behaviour-decomposition* [175] based techniques wherein, separate subcontrollers are evolved to cater to each of the different subtasks. An arbitrator that sits on top of these subcontrollers and performs the task of selecting the

appropriate subcontroller for the current task [49, 116], is then evolved.

In contrast to traditional evolutionary approaches, the term *Embodied Evolution* (EE) applies to a CPS of robots that autonomously and continuously adapt their behaviors in accordance with the changes in the environment [26]. In EE, robot controllers learn both onboard and online, and continue to do so even when the robots are actually deployed in an environment, thereby reducing the *reality gap* [84]. EE is still susceptible to bootstrapping and deception issues when the task to be accomplished is complex for single-controller based robots to learn. Traditional approaches have been known to use behavioral decomposition [116] to avoid bootstrap issues, but they have seldom been applied in an *embodied*, onboard and online manner. What needs to be explored is a technique that is a distributed, fully embodied and evolutionary version of such traditional approaches.

In this chapter, an immunology-inspired embodied action-evolution cum selection algorithm has been proposed to evolve different subcontrollers for different *regions* of the search space. Sensor values sampled from the environment constitute the antigen while the associated subcontroller (actions) corresponds to an antibody. Similar to the way antibodies are evolved and primed for different antigens, the algorithm evolves and selects different subcontrollers for an associated *region* of the antigenic (sensory) space. Some of the salient features of the proposed algorithm are:

1. On-the-fly and onboard Evolution: The subcontrollers are evolved on-the-fly and onboard the robots.

2. Distributed Learning: While the encapsulated version of the algorithm within one robot seeks and learns to choose the best subcontroller, sharing of these controllers across its peers in the collective of robots speeds up the convergence process [26].

3. Single Parameter Tuning: A single system parameter viz. the cross-reactivity threshold (explained in Chapter 5) can be used to tune and vary the granularity of search in the antigenic space.

This chapter describes an alternative algorithm to evolve and select different subcontrollers on-the-fly for different regions (ranges) of antigenic space (sensor values)

sampled from the environment, in a decentralized and distributed manner. The use of an immunology inspired algorithm ushers a new era wherein multiple subcontrollers or solutions can be created, evolved and arbitrated online in a *d*CPS.

7.1 Background

Though a copious amount of research on evolutionary approaches can be cited, this section discusses the behavior-decomposition and embodied evolution based techniques which are more pertinent to the work presented in this chapter.

7.1.1 Behavioral decomposition

In this method, instead of just one robot controller, different subcontrollers are evolved to solve distinct subtasks. After these subcontrollers have adequately evolved, another (usually ANN based) controller is trained to map the input states of the robot to one of the already evolved subtask specific controllers. Lee [116] describes each of the low-level subtask specific controllers as *behavior primitives* while the top-level controller that has learned to map the inputs to these subcontrollers was termed the *behavior arbitrator*.

Lee [116] implemented a Genetic Programming (GP) based controller to solve a box-pushing task. This task was manually divided into two subtasks. Separate subcontrollers (primitives) were evolved for each subtask on a simulator. Finally, a GP based arbitrator was evolved to combine these subcontrollers hierarchically. Though their approach was implemented on a real robot, each subcontroller was evolved separately in an offline manner. Further, there is no evidence that there proposed architecture can be used across multi-robot scenarios in a decentralized manner. Moioli et al. [131] proposed the GasNet system where subcontrollers for two different tasks were activated or inhibited based on the production and secretion of virtual hormones. Their homeostasis-inspired controller was able to select appropriate subcontrollers depending on internal and external stimuli.

Duarte et al. [49] portrays a search and rescue task using a real e-puck robot. The overall task was divided into a few subtasks and separate subcontrollers were evolved for each of them. The experimenter provided the fitness function for each

subtask. After the appropriate subcontrollers were found, a behavior arbitrator was evolved which delegates a subtask to the best subcontroller based on the sensory inputs. Though a complex task was solved using their hierarchical controller, the behavior primitives and the behavior arbitrator do not seem to have been evolved in an online and onboard manner. A similar line of work by Duarte et al. [50] also demonstrates an approach for the incremental transfer of their evolved hierarchical control system from simulation to real robots. Of late, Duarte et al. [48] have presented EvoRBC, an approach to evolve a control system for robots with arbitrary locomotion complexity and implemented it on a simulated robot. They have used a Quality Diversity algorithm to build a repertoire of behavior primitives (e.g., move straight, turn right). This repertoire is then used to evolve an ANN which maps the sensory inputs of the robot to an appropriate primitive.

In most of the above-reported approaches, the subcontrollers are evolved separately and lack the essence of continuous, distributed, decentralized and on-the-fly learning.

7.1.2 Embodied Evolution

A recent comprehensive paper by Bredeche et al. [26] presents a review of the research published since the inception of the term Embodied Evolution (1997-2017). EE involves continuous and online learning in a collective of robots. A population of robots learns in a decentralized manner by sharing the controllers evolved among the peer robots. Although there has been a recent surge in papers [26] where EE has been successfully applied, most of the work that cites the use of real robots [190, 81, 177, 138] are constrained to a single controller that can solve relatively simple tasks such as obstacle avoidance and phototaxis. Only recent, a work by Heinerman et al. [81] implements a relatively complex task of foraging using a single controller. A robot may require learning several such tasks. Under such conditions, the subcontroller will need to be re-trained to take in the set of new tasks. In an ANN-based subcontroller, such re-training may not be a viable exercise. Intuitively, one may conclude that evolving a single subcontroller to cater to an ever-increasing number of tasks, is extremely cumbersome if not impossible. Using behavior primitives and an associated arbitrator may be a logical step to circumvent

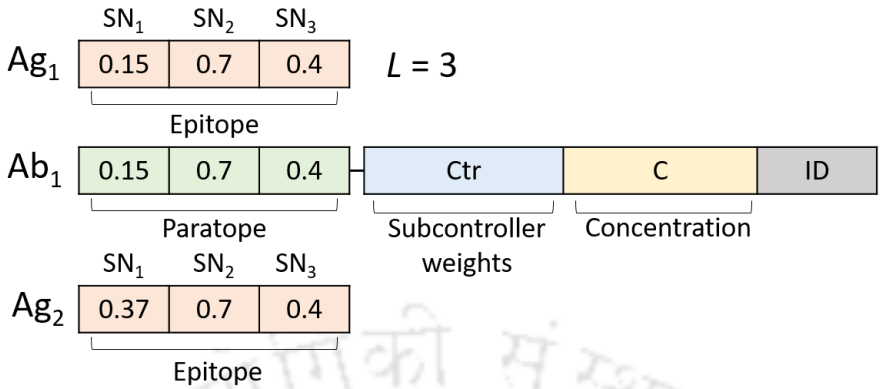


Figure 7.1: An Antibody tackling two different Antigens (SN_is are the values obtained from sensors onboard the robot)

this drawback. However, the drawback is that for every incremental addition of a new subtask the behavior arbitrator has to be trained offline all over again. This calls for an online and onboard continuous learning mechanism for both the subcontrollers and that arbitrator which will consequently lower the reality gap. The algorithm proposed in this chapter distinguishes itself from earlier presented techniques in a way that instead of subtask division, it keeps dividing the whole sensor-sampled search area within the given environment, into separate regions. A subcontroller is then evolved on-the-fly for each such region.

7.2 Methodology

The method proposed in this chapter is inspired by the novel action-selection mechanism exhibited by the Biological Immune System (BIS). This section initially discusses the immunological metaphors used, followed by a description of the computational counterparts and the explanation of the proposed algorithm.

7.2.1 From Immunology to the Real World

In the real world, the antigenic epitope Ep can be visualized as an L-dimensional vector sampled from the environment via sensors onboard a robot. L corresponds to the number of sensors attached to the robot. Figure 7.1 depicts an epitope Ep (top) formed when a robot encounters an obstacle in front. The corresponding antibody Ab, as shown in Figure 7.1, comprises four components - a paratope Pt,

the associated subcontroller Ctr, a concentration C value and an ID number. A new Ab is created if the Ag is not recognized by any of the Abs present in the repertoire. The Pt of such a new Ab is initialized to the Ep of the current Ag so that both Pt and Ep are dimensionally equivalent and have the same shape space. The Ctr is a vector comprising the weights of an associated neural network while the C denotes the fitness value returned after executing the Ctr when the corresponding Ab is chosen to quell Ag. The fitness function depends on the task and is provided by the experimenter based on the application. Abs, with better performing Ctrs, evolved based on fitness values, are assigned proportionately higher concentrations. The ID number aids in uniquely identifying an Ab. Figure 7.1 also shows another Ag (bottom) which falls in the AR of the same Ab.

The Euclidean distance between an Ep and a Pt is used as the affinity measure (ψ). Lesser this distance more is the affinity. An Ab is chosen to tackle a given environment state (Ag) if the affinity ψ between the Ep and the Pt is less than ϵ, a system constant akin to the cross-reactivity threshold [43]. Antibodies that satisfy this criterion are referred to as the *candidate antibodies*. All such candidates antibodies are stimulated by the corresponding antigen Ag located within the overlap of all active regions of these antibodies as shown in Figure 5.5(c) of Chapter 5. This antigenic stimulation ($AgSti$) results in increasing the C of all the candidate antibodies by an amount proportional to their respective ψ values. In the computational world, a specific subcontroller is synonymous to an Ab while an environmental state acts as an Ag. Thus, different subcontrollers could be evolved, each of which is tuned to specific environmental states. This chapter presents a BIS inspired novel algorithm to evolve subcontrollers (antibodies) for specific actions required to counter different environment states (antigens) sampled by the sensors onboard a robot. Just as the BIS, this algorithm is decentralized and distributed in nature and evolves subcontrollers on-the-fly.

7.2.2 The Proposed Algorithm

The proposed algorithm is outlined in Algorithm 5. The algorithm runs on every robot belonging to the swarm. It works in tandem with a communication routine which facilitates foreign antibodies (subcontrollers) to be received from the

Algorithm 5 The Proposed Algorithm

1: $\epsilon \leftarrow Constant$;{Initialize cross-reactivity threshold}
2: $IRep, XRep, CandAb \leftarrow \varnothing$;{Create empty lists}
3: **while** True **do**
4: $Ep \leftarrow fetchSensorValues()$;{Get sensor values}
5: **for** each Ab_i in $IRep$ **do**
6: $\psi \leftarrow measureAffinity(Ep, Pt_i)$; {Calculate affinity between Ag and Ab}
7: **if** $\psi \leq \epsilon$ **then**
8: $agStimulation(\psi, Ab_i)$;{Stimulate Ab_i}
9: $CandAbs \leftarrow append(Ab_i)$;{Add Ab_i to the list of candidate antibodies}
10: **end if**
11: **end for**
12: **if** $CandAb = \varnothing$ **then**
13: $NewAb \leftarrow createNewAntibody(Ep)$;{Create a new Ab if there are no candidate antibodies to select}
14: $CandAbs \leftarrow append(NewAb)$;{Add new Ab to the list of candidate antibodies}
15: $IRep \leftarrow append(NewAb)$;{Add new Ab to the internal repertoire}
16: **end if**
17: **if** $(random() < P_{sharing})$ OR $(XRep.size = 0)$ **then**
18: $BestAb \leftarrow selectBestAntibody(CandAbs)$; {Select the best Ab from list of candidate antibodies}
19: **else**
20: $BestAb \leftarrow selectBestAntibody(XRep)$; {Select the best Ab from the external repertoire}
21: **end if**
22: **while** Current Ag space is within the AR of $BestAb$ **do**
23: Execute and Evolve $BestAb$ using an EA;
24: **end while**
25: $CandAb, XRep \leftarrow \varnothing$;
26: **end while**

peer robots in the collective and stored in an *extrinsic* antibody repertoire ($XRep$) within the robot. During the task execution by a robot, its *intrinsic* antibody repertoire ($IRep$) is broadcast to all the peer robots within a communication range. Instead of all, only a set Abs from $IRep$ selected based on a fitness criterion can also be shared.

Initially, the $IRep$ within a robot is a *tabula rasa*. As mentioned, the sensor values sampled from the environment form the epitope Ep of the antigen Ag. As soon as Ep of the current Ag has been sampled, the distances (ψ) between this Ep and the Pts of all the Abs in $IRep$ are calculated using the $measureAffinity$ function (line 6 of Algorithm 5). The Abs for which the ψ values are less than ϵ

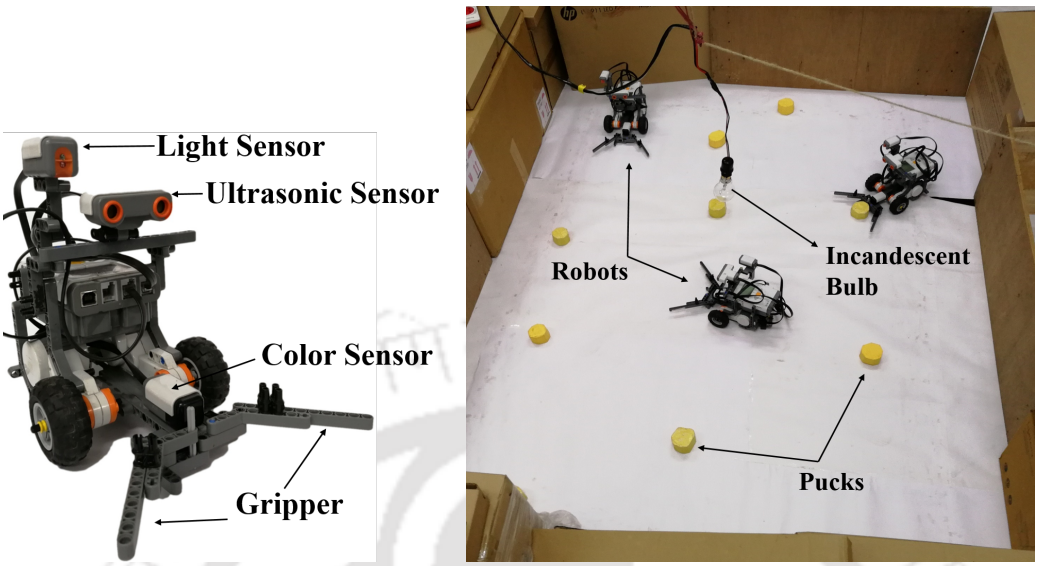
are made to be stimulated (using *agStimulation* function) by the antigen *Ag* and are then appended to a list of candidate antibodies (*CandAbs*). As in the BIS, the *AgSti* raises the concentration *C* of all antibodies within the *CandAbs* by an amount proportional to their respective ψ values. In the beginning, since no *Ab*s exist in the *IRep*, a new *Ab* is created whose *Pt* is initialized to *Ep* and the weights of the associated *Ctr* are randomly set (line 13). The ID is provided sequentially, starting with 1. The value of its *C* is set to an initial non-zero minimum value lest it be discarded immediately. Antibodies with *C* equal to zero, are purged from *IRep*. The new *Ab* with ID 1 is then added to the currently empty list *CandAbs*.

For final execution by the robot, the best *Ab* is selected from either *XRep* or the list of *CandAbs* based on a probability $P_{sharing}$. The *selecBestAntibody* function (lines 18 and 20) selects the matching *Ab* ($\psi \leq \epsilon$) having the highest *C* value. The *Ctr* of the best *Ab* is evolved in order to produce adequate behavior within the associated *AR*. When the robot is executing a controller of the selected *Ab*, the environmental state is continuously sampled. It may be noted that during the task execution, if the *Ep* sampled from the environment is outside the *AR* of the currently selected best *Ab*, the current task execution is stopped and the process to select and evolve a new *Ab* recommences.

The algorithm partitions the environment (antigenic) space sensed by the robot based on ϵ which in turn defines the area of the *Active Region* (*AR*) of a subcontroller (antibody). More the value of ϵ, more is the environment space catered to by that subcontroller. Increasing ϵ will mean a lesser number of subcontrollers for a given environment space. If ϵ were to cover the entire environment space then, the algorithm would try to evolve just one subcontroller that can cater to all conditions, as in [81, 138]. On the other hand, a very low value of ϵ would mean a smaller *AR* resulting in too many subcontrollers catering to particular tasks.

(1+1)-Online Evolutionary Algorithm

The evolution of the *Ctr* of the best *Ab* that has been selected can be achieved using an Evolutionary Algorithm (EA). This work leveraged the (1+1)-Online evolutionary strategy [25] to evolve the antibodies with controllers that deliver satisfactory performance. In this strategy, the controller weights are mutated using a Gaussian



(a) The Robot (b) The Experimental Arena

Figure 7.2: (a) Structure of the LEGO® MINDSTORMS® NXT robot used in the experiments (b) The Experimental Arena

function with $N(0, \sigma)$, where the value of σ doubles if the offspring of the controller of the selected *Ab* performs lower than the currently used parent controller. An offspring replaces the parent if the former outperforms the latter one.

7.3 Experiments

For experimentation, a set of LEGO® MINDSTORMS® NXT robots each having a Colour Sensor (CS), an Ultrasonic Sensor (US) and a Light Sensor (LS) are used and shown in Figure 7.2a. The US mounted in front of the robot aids in obstacle detection while the CS facilitates identifying the color of an object encountered by the robot. The two LSs are fixed parallel to the two motors in order to detect the light from both directions. The range of values from the ultrasonic, color and light sensors vary from 0 to 200, 0 to 9 and 0 to 100, respectively. High values from the US denote that the obstacle is far from the robot while low values indicate the presence of the obstacle. High values from the LS show that the robot is near to the light while lower values specify that the robot is far away from the light source. For CS, the default value is calibrated to 7. The high and low values were decided based on the size of the arena. For instance, if the size of the area is more than the

detectable range of the US, then the a value near the maximum 200 will correspond to the high values. For tasks involving foraging, a Lego gripper was also attached to the lower frontal part of the robot. With two motors attached to two rear wheels and a caster wheel at the front, the robot used a differential drive to move around in its environment. The operating speed of the motors was set to 70% of the maximum speed. A Pi mounted onboard the robot and interfaced with the sensors, and the motors constituted the main hardware controller. As depicted in Figure 7.2b, 8 to 12 lightweight yellow colored hexagonal-puck shaped objects were scattered randomly across a 2m x 2m arena which constituted the environment. For the yellow color, the CS returned a value 3. The only obstacle in the arena are the walls surrounding the arena. A Wi-Fi router placed near the arena facilitated the interconnection between the robots and hence the sharing of antibodies. Experiments were performed in both static and dynamic networks. Since Wi-Fi range covers the whole arena, dynamism was implemented by randomly creating and breaking the connection between the robots. The *Ctr* constituted a feed-forward ANN with four input nodes, five hidden nodes, and two output nodes and used the *hyperbolic-tangent* (tanh) as the activation function. Out of the four input nodes, two were connected to the two light sensors, while the remaining were provided inputs from each of the ultrasonic and the color sensors. The sensor readings were normalized to the range between 0 and 1 and then fed to the inputs of the controller ANN.

7.3.1 Scenarios

The proposed algorithm was tested on a set of three scenarios explained below in the order of increasing complexity.

Scenario S_1 - Static Environment

In this scenario, a light source (an incandescent bulb) was fixed at some minimum height in the center of the arena. The objective was to make the robot move towards the light source (phototaxis) and then stay near it. If the robot encountered an obstacle, then it had to avoid it. Using a bare minimum of just a single US and one LS puts additional pressure on the algorithm while searching and evolving an optimum controller. For a single robot controller, the task of phototaxis together

with obstacle avoidance thus becomes non-trivial. Similar to online evolutionary systems presented in [81], the objective function used herein rewards behaviors (Ab) for each of the sub-goals achieved. The sub-goals herein include motion of the robot towards the light source and avoiding obstacles. The objective function for an evaluation period of $t^{'}$ timesteps is as follows:

$$\mathcal{F}_{T_1} = \sum_{t=0}^{t=t^{'}} (f_{obs} + f_{light}) \tag{7.1}$$

where, $f_{obs} = v_{trans} * (1 - v_{rot}) * d$

$f_{light} = \max_{1 \leq j \leq 2} (lightSensor_i)$

f_{obs} is a classical function adapted from [135] wherein v_{trans} is the translational speed, v_{rot} is the rotational speed and d is the distance between the obstacle and the US onboard the robot. v_{trans}, v_{rot} and d, are all normalized between 0 and 1. The fitness function f_{light} rewards movement towards the light source. $lightSensor_i$ is the normalized value from the LS between 0 (no light) and 1 (brightest light).

Scenario S_2 - Static Environment with Puck Pushing

The primary goal here was to push the yellow colored pucks towards the light source while also avoiding the walls. If no pucks are encountered, the robot should continue moving towards the light source. The CS fixed behind the gripper facilitates the detection of the puck based on its colour. Along with the sub-goals defined in S_1, the task S_2 also comprises a fitness function to reward the controlled pushing of the puck using the gripper attached to the robot. The objective function for S_2 is given by:

$$\mathcal{F}_{T_2} = \sum_{t=0}^{t=t^{'}} (f_{obs} + f_{light} + f_{puck}) \tag{7.2}$$

where, $f_{puck} = b_{puck} + v_{trans}$

b_{puck} is a binary variable which is equal to 1 if a puck is within the gripper else it defaults to 0. The meanings of f_{obs} and f_{light} are same as that defined in scenario S_1.

Scenario S_3 - Dynamic Environment

In this scenario, a change in the environment needed to be compensated by a reversal of a behavior. Here, the light source was switched ON or OFF in an asynchronous manner during runtime. When the light was ON, the goal remained the same as in scenario S_2 where the robot needed to push the puck towards the light source. However, when the light source was switched OFF, the robot needed to learn to repel these object whenever encountered. Irrespective of whether the light source was ON or OFF, the robot also needed to learn to avoid the walls or any other static obstacle. The objective function used in this scenario is given below.

$$\mathcal{F}_{T_3} = \sum_{t=0}^{t=t'} (f_{obs} + f_{light} + b_{light} * f_{puck} + \bar{b}_{light} * f_{antipuck}) \tag{7.3}$$

where, b_{light} is a boolean variable equal to 1 if the light intensity is above a certain threshold. The average of the values returned from the LS at different positions in the arena while the light is ON, forms the threshold. $f_{antipuck}$ models the repulsion of a robot when it encounters a puck.

Two sets of experiments were carried out - 1) Real-robot 2) Energy Saving. In the real-robot experiments, the sharing module of the proposed algorithm was switched off and only a single real-robot was used to learn the respective tasks in the three scenarios explained later. In addition, a set of three real robots were was also used to learn the same tasks with the sharing module enabled. This allowed the robots to share their subcontrollers (antibodies) mutually. The second set of experiments were designed with a completely different motive. These experiments were intended to introduce and test a new and energy efficient sharing mechanism. Experiments herein were carried out in an emulated environment with 80 soft or virtual robots connected through a dynamic network. Besides, experiments wherein attempts to evolve single robot controllers for each of the scenarios were also performed. Though not crucial, this was necessary to validate the fact that the used scenarios are substantially complex for a single controller.

7.4 Results

This section initially discusses the results on the attempts to evolve a single controller for all the tasks, followed by analyses of the set of experiments which are performed using the real-robots. Finally, it showcases the results obtained with the IPM scheme running on an emulated network of 100 nodes.

7.4.1 Evolving a Single Robot Controller

A total of ten runs per scenario were performed to evolve a single controller using (1+1)-Online EA running on a single robot. The controller was allowed to evolve for 200 iterations in scenario S_1 while the same was 400 for the other two scenarios. In scenario S_1, the controllers which learned to move straight towards the light or avoid the obstacle, easily evolved during the first half of the total of 200 evaluations in all the ten runs of the same experiment. It was only after an average 145^{th} evaluation count that controllers which learned partial phototaxis together with obstacle avoidance emerged. These controllers were of inferior quality in the sense that they followed a curved path while moving towards the light source, instead of the preferred straight movement.

In the scenario S_2, a controller which learned all the three subtasks, namely foraging, phototaxis and obstacle avoidance, was not evolved. From the pool of controllers found during the 400 evaluations, some of the evolved controllers learned only obstacle avoidance while some others learned foraging and phototaxis. No controller learned all the tasks. Scenario S_3 had two subtasks which are switched based on the two conditions: 1) Condition C_1, when the light source is turned ON, and the robot had to push the pucks towards the light while also avoiding obstacles. 2) Condition C_2, wherein the light source is turned OFF and the robot had to learn to avoid obstacles and to repel the pucks encountered. The conditions C_1 and C_2 were triggered asynchronously. The duration for which each condition lasted was kept such that a minimum of 5 consecutive controller evaluations could take place under each condition. This asynchronous switching made it impractical for a single controller to learn the whole task. It may thus be concluded that it is difficult for a single monolithic controller to learn all the tasks described herein.

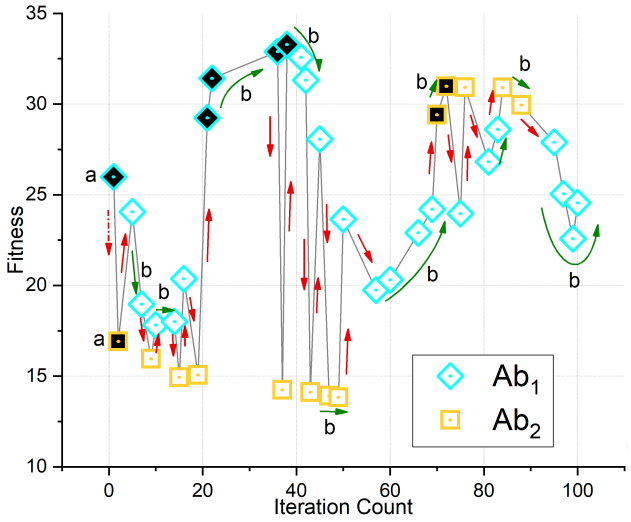


Figure 7.3: Variations in antibody fitness for scenarios S_1

7.4.2 Experiments using Real-Robots

Ten trials for each of the scenarios[1] were performed. The value of ϵ was empirically found and set to 0.45, 0.4 and 0.25 for the scenarios S_1, S_2 and S_3, respectively. For a given scenario, the desired value of ϵ can be found by first initializing it to half the normalized range. The value may then be gradually increased or decreased based on whether the number of subcontrollers generated is sufficient to make the robot(s) learn the scenario. Using the proposed algorithm, the subcontrollers were allowed to evolve for 200 iterations in scenario S_1 while the same was 500 for the other two scenarios.

Scenario S_1

Figure 7.3 shows a typical evolution-selection curve of the antibodies created, evolved and selected during one of the trials. The X-axis denotes the iteration count and Y-axis indicates the fitness values returned by the objective function defined in equation 7.1. For this particular run, the robot was initially placed near the boundary wall of the arena, facing the light source. Following is a description of the events that resulted in the graph shown in Figure 7.3. The robot initially sampled the Ag from its environment and the first antibody Ab_1 was created, depicted by the cyan colored marker on the graph (Figure 7.3). The robot then executed the randomly

[1] For the video: https://goo.gl/8qtJtd

initialized Ctr of Ab_1 which caused it to move to a place whose Ag, when sampled, was found to be outside the AR of Ab_1. The random behavior executed by the robot was that of rotating around its position. Since initially, the robot was facing an open area, rotation around its axis caused it to face the wall leading to a drastic decline in their US and LS values, justifying the need to generate a fresh Ab to tackle this new and unknown antigenic space. Having sampled this new Ag (a vector having small values in the US and LS fields) the robot created a corresponding fresh antibody Ab_2 (orange colored marker) and executed it. These very first versions of the antibodies Ab_1 and Ab_2 corresponds to the case (a) in Figure 5.5, which shows the creation of new antibodies coincident on their corresponding Ags. The transitions from Ab_1 to Ab_2 (and vice-versa) are shown using red colored arrows on the graph in Figure 7.3. These transitions from one Ab to another indicate the *on-the-fly* selection property of the proposed algorithm. The green colored arrows denote the selection of the same Ab after an iteration ends. This indicates that the robot has sensed the next Ag within the AR of the same antibody.

In both cases (pointed by the arrows), the evolution of an Ab is carried out using an EA where the parent Ctr of Ab is mutated to produce an offspring which is then executed based on a probability. If the offspring perform better than the parent, the offspring Ctr replaces the parent Ctr of the same Ab. Since the robot is in continuous motion, the sampled Ag is always changing. Thus, the same Ab is selected when different Ags appear within its AR, as previously highlighted in case (b) of Figure 5.5. The best antibodies evolved are indicated with the markers filled with black color as shown in Figure 7.3. Iteration 3 onwards, $IRep$ has two antibodies from which the robot could choose one. These include Ab_1 which can be triggered for the antigenic space wherein the robot is in an open area and Ab_2 which can be chosen when there is an obstacle in front. This process of evolution and selection continues until the training period ends. The best antibodies evolved can then be used during the testing phase. For the remaining 9 out of 10 runs, the robot is placed in different positions and orientations. It may be noted here that the concept of action-selection should not be confused with those in incremental learning approaches [20], as it may seem so from Figure 7.3. The antibodies herein are selected based on an event without any human interference or for that matter

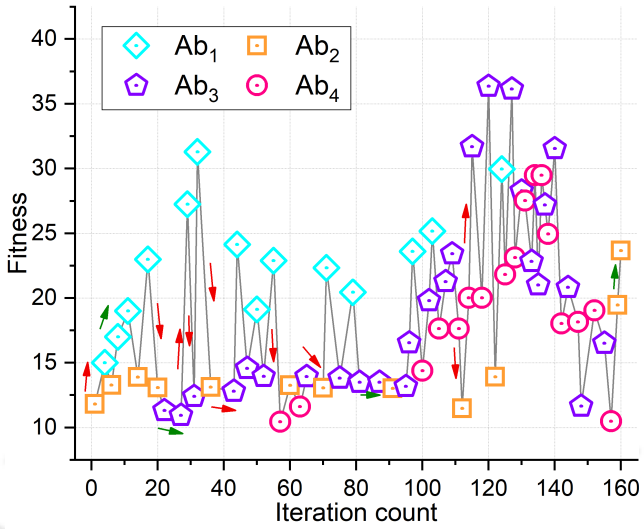


Figure 7.4: Variations in antibody fitness for scenarios S_2

any other entity.

Scenario S_2

This scenario is a complex extension of S_1 wherein along with phototaxis and obstacle avoidance, the robot also needs to learn to push the puck towards the light source as and when it encounters one. In addition, the robot needs also to learn to avoid an obstacle while pushing this object without mislaying or losing them. Figure 7.4 shows a similar evolution-selection curve (as in Figure 7.3) wherein the antibodies were evolved and selected during the training period. In this trial, the robot was randomly placed in a position facing the wall of the arena. The repertoire, $IRep$ is set to NULL before initiating the algorithm. Since this scenario is an extension of S_1, the antibodies that could tackle the antigenic spaces involving obstacles and phototaxis, emerged to be of similar nature as found during the training in S_1. In addition, two new antibodies (Ab_3 and Ab_4) were created, evolved and selected. Ab_3 was selected whenever the robot encountered a puck object with no obstacle in front. Ab_4 evolved to tackle the case when the robot encountered an obstacle while pushing the object. The flow of evolution and selection can be seen in Figure 7.4. The meanings of the red and green arrows are the same as mentioned in the earlier graph. It can be seen that while on one side the proposed algorithm evolves new antibodies, it is also capable of selecting a pertinent one amongst these

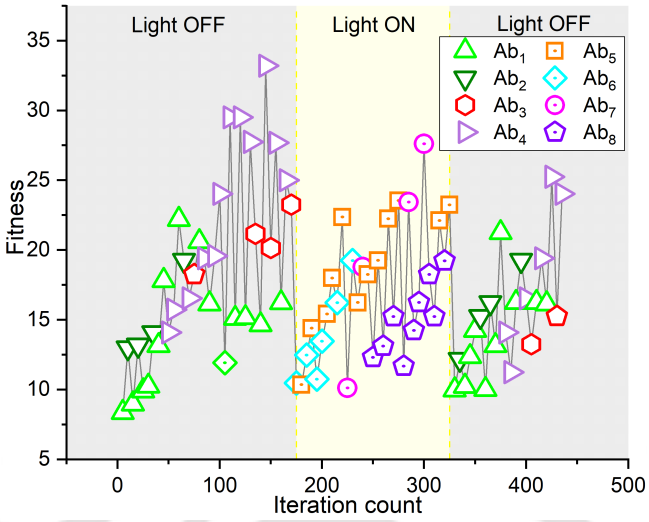


Figure 7.5: Variations in antibody fitness for scenarios S_3

for execution.

Scenario S_3

The environment in this scenario changes dynamically and asynchronously with the robot having no control over it. The robot, in turn, needs to change its behavior based on this change in its environment. As mentioned in earlier, scenario S_3 requires different behaviors, and thus different *Ab*s for each of the light conditions (ON or OFF). As can be seen from Figure 7.5, the light source was initially in the OFF position from the 1^{st} to the 176^{th} iteration. After this, it was kept switched ON till the 325^{th} iteration and turned OFF again. The *Ab*s specific to the light's OFF state were initially created, evolved and selected during the time the same OFF state was maintained. As soon as the light source was switched ON, the antigenic space changed, leading to the creation of a new set of *Ab*s. These *Ab*s then followed the same evolution-selection journey in order to produce suitable *Ctr*s. As can be seen from the figure, when the light OFF state occurs again, the previously evolved *Ab*s present in the repertoire were triggered and again and evolved further. It may be noted that the *Ab*s in all the runs of each of the experiments for scenarios S_1, S_2 and S_3, were learned from scratch.

A total of 5 independent trials for S_3 were also conducted using three real robots which were allowed to share their respective *IRep*s among the peers with similar

results. Increasing the number of robots further will aid a parallel search, which in turn can enhance the learning. This experiment also validated that the algorithm can be run across a collective of robots. The advantage of sharing, however, is put down to an extent by the energy consumed in the sharing operations. In the next set of experiments, presents and implement an alternative way to share knowledge among the robots in an energy conservative manner.

7.4.3 Sharing Scheme for Energy Conservation

Sharing of information within a collective of robots is traditionally realized using broadcasting on the part of individual robots. This method, though simple and effective, consumes a fair amount of energy. Energy conservation is vital in mobile robot scenarios since charging points may not be available or could be far away from its current location. Thus, every attempt to reduce energy consumed needs to be made. This work used mobile agents to share the information among the robotic nodes. This communication strategy was found to be more conservative in energy consumption than the conventional broadcast method. For experimentation, an emulation environment *Tartarus* [163] was chosen instead of simulation since the former allows for better analyses of real network parameters such as data transfer speed, bandwidth and energy consumed.

Each mobile agent is a piece of code which has the ability to migrate from one node to another in an autonomous manner. These mobile agents or *Intelligent Packets* (*iPkt*) can migrate from one node to another in a network, take decisions and also execute the code they carry based on predetermined conditions. An *iPkt* can also carry information in the form of its payload and transport the same to the other nodes during its sojourns across the network. These *iPkt*s are used to carry the controllers (weights of the ANN) evolved by a robotic node to facilitate sharing with its peers in the network. To restrict the extraneous movement of such a packet, the same has been programmed to move in a *conscientious* manner wherein the *iPkt* maintains a list of already-visited robot nodes. Every time an *iPkt* visits a node, it appends the node identifier (node address) in the list. In the conscientious movement, an *iPkt* migrates to that neighboring node which is either not visited or has been least visited. Unlike broadcasting, the conscientious method provides

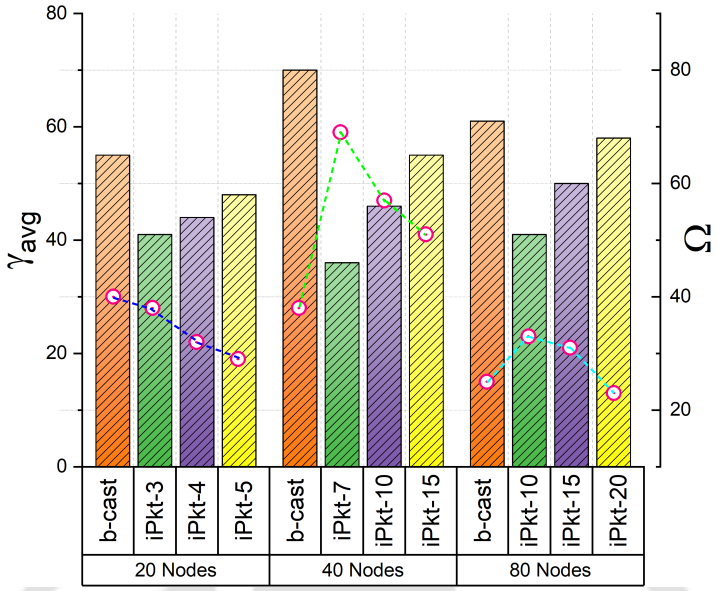


Figure 7.6: Plots of γ_{avg} and Ω for different networks for Broadcast and IPM schemes

for a controlled movement and thus greatly aids in the reduction of unnecessary packet migrations. In addition, an *iPkt* can also be programmed to move to another robotic node under certain conditions such as when a controller with a desired fitness is found. This could further contribute to reducing the energy spent on communication. It may be noted that to know the neighbors, an *iPkt* at a node needs to broadcasts a small *hello* message packet to create a routing table. Since the *hello* broadcast is done only by those nodes that currently have a packet within and when it wants to migrate to a neighboring node, the overall communication overheads turn out to be less than that in case of broadcasting. For comparing the efficacy of the sharing schemes, a standard EA such as the one proposed in [81] has been implemented to solve the soft task of learning an OR gate function. An ANN with 2 input nodes, 3 hidden nodes, and 1 output node was used. For broadcasting, the social learning method prescribed in [81] was used while in the case of mobile agent based strategy, the encapsulated version (individual learning) of the same method as in [81] was implemented wherein the *Pkt*s were used to share the evolving controllers among the emulated robot nodes forming the network.

In Figure 7.6, the left Y-axis denotes the inter-robotic node communications (sending and receiving) per robot node averaged (γ_{avg}) over 10 runs of the exper-

iments. The right Y-axis is based on a metric called the *Convergence count* (Ω), which is the number of iterations spent from the point when the first best solution was found by a node till 90% of the robot population converges to the same solution due to sharing. In the X-axis, *b-cast* denotes the broadcast method while the terms *iPkt-x* correspond to the cases when x *iPkt*s were used. The value x, i.e. the number of *iPkts* is chosen proportionately to the total number of robotic nodes in the network. The emulation experiments were carried out on 20, 40 and 80-node dynamic networks. Dynamism was introduced by varying the number of neighbors per node from 0 (isolated node) to a maximum of 5. The neighbors of a node were changed randomly over time making it the equivalent of a dynamic network.

As can be seen from Figure 7.6, with an increase in the number of nodes, the average number of communications made per node using the broadcast method remains higher than that reported using the IPM scheme. The graph also shows that as the number of *iPkt*s is increased, the sharing too is enhanced (lower convergence counts). This can be observed for each of 20, 40 and 80-node scenarios where the convergence counts drops with the increase in *iPkt*s. It may be noted that, though one may infer that an increase in *iPkt*s would accelerate the sharing process, it comes at the cost of higher communication overheads per node. Overall, one may thus conclude that if saving power is prime, then the IPM method seems to have an edge over the conventional broadcast mechanism.

7.5 Chapter Summary

This chapter describes an immunology-inspired distributed and embodied action-evolution cum selection algorithm for learning of specific subcontrollers in a CPS of robots. Robots in a collective evolve individual subcontrollers for coping up with the environment just the way the BIS creates and tunes antibodies to quell antigenic attacks on-the-fly. Subcontrollers are shared amongst the robots to facilitate other robots to get better solutions and to cope up with the environment. Based on the value of ϵ, the environment (antigenic) space sensed by the robot can be partitioned into different *Active Regions* (AR). A separate subcontroller is then evolved for each AR. ϵ thus governs the number of subcontrollers generated. Since power can be a

significant source of concern in mobile robots, the use of *iPkt*s for sharing allows for a fair amount of saving in the same. Further investigations into adjusting the number of *iPkt*s and their subsequent routing will provide better insights into making the sharing mechanism more energy efficient.

"Study Nature, Love Nature, Stay Close To Nature, It Will Never Fail You."

Frank Lloyd Wright (1867 – 1959)
American architect

8

Conclusions and Avenues for Future Research

With the large-scale proliferation of sensors and embedded devices including robots, the need for algorithms that are scalable, robust, intelligent and work in conjunction with decentralized Cyber-Physical Systems (*d*CPS), has become mandatory. This thesis addressed the importance and challenges of realizing *d*CPSs in a bottom-up fashion. Assimilating the past and the needs of the modern-day, the contributions made in this thesis are two-fold. While the first half introduced *Tartarus* as a middleware and solved the classical problem of mutual exclusion in *d*CPS, the second half took a non-conventional approach and proposed mechanisms inspired by the Biological Immune System (BIS). As in the previous work by Godfrey et al. [67, 64] and Jha [86], mobile agents were found to be a viable tool for in-network information processing and sharing of knowledge within the *d*CPS. This chapter presents a summary of the contributions made and discusses their applications envisioned in the real world. This chapter concludes with the avenues for future research and directions for their extension.

8.1 Summary of Thesis

This thesis aimed at introducing the concept of decentralization in Cyber-Physical Systems and the Internet of Things. It is envisaged that the contributions made in this thesis will apply to scenarios wherein a centralized control is impractical. Such milieus include a swarm of robots and drones exploring remote areas, a dense network of IoT devices, modern warehouses, offshore asset management, and other similar moderate and large-scale systems. With a bottom-up approach, the first contribution (Chapter 2) laid a foundation on top of which the future decentralized mechanisms can be built. *Tartarus*, a multi-agent middleware, was enhanced and presented as a tool to bridge the gap between the *cyber* and *physical* systems. It included support not only for the general high-level operating system based computers, but also encompassed embedded devices, sensors and robots. In contrast to its predecessor, *Typhon*, which was built on top of a proprietary brand of Prolog, *Tartarus* was developed using SWI-Prolog, a free software thereby embracing the principles of Open Science[*]. Some of the significant features of *Tartarus* are heterogeneity, multiple hardware support, payload capability in agents and on-the-fly programming, which portrays it as a versatile tool for the realization of a real-world *d*CPS. Its modular software architecture facilitates easy upgrade and addition of interfaces to cater to new hardware. The agents in *Tartarus*, both static and mobile, were tested against a real-world scenario comprising robots, a Pi mounted with a camera forming a monitoring node and a human-administrator in the loop. The system displayed its unique characteristics as a platform for realizing a *d*CPS.

With this platform in place, the next contribution (Chapter 3) comprised a mobile agent based methodology to realize decentralized CPS and IoT systems. The primary aim of this chapter was to realize a *d*CPS and compare it with its centralized counterpart. For the same, a Location-Aware and Tracking System (LATS) was used as an application. Several Pi-nodes comprising a Pi and a Bluetooth Low Energy (BLE) module, were deployed across a building at the Indian Institute of Technology Guwahati. Users who need to be tracked were made to wear a BLE beacon transmitter. If a user enters the zonal area of a Pi-node, his/her location

[*] https://www.fosteropenscience.eu/content/what-open-science-introduction

is registered on the Pi-node. In the decentralized version of the LATS, the data remained within the Pi-nodes. This contribution proposed a novel *motion vector* based approach using which the direction of the person was calculated through local interactions among the neighboring Pi-nodes. Finally, the queries regarding the *where* and *when* about such persons were wrapped into a mobile agent and released into the network of Pi-nodes. The mobile agents utilized the motion vectors to trace the user. The results of the proposed decentralized agent-based methodology when compared with the centralized version, validated the former's efficacy over the latter. The use of mobile agents to share the information can be easily extended to different IoT environments, for instance, an IoT for data collection in museums.

The next contribution (Chapter 4) deals with a classical problem of mutual exclusion of shared resources in a *d*CPS of robots. Centralized systems have their advantages, one of which is global control. However, in decentralized systems, coordinating the access of resources required to accomplish a task by a set of mobile robots, is a challenging affair. The proposed mechanism solved the problem of mutual exclusion by ordering the tasks received by a job distributor in the form of a pipeline. Each job is made of a sequence of tasks which were allocated dynamically to the set of robots acting as processors modeled as a pipeline. Though this resembles a pipeline processor, it needs to be emphasized that the processors (robots) herein are physically mobile and have varying execution times for the same task. The emulation experiments conducted demonstrated the versatility of the proposed mechanism against the varying periods of task execution. *On-the-fly* addition and deletion of both tasks and robots form the enticing features that make the mechanism suitable for industrial applications. The experiments on real robots further strengthen the usability of this approach. Besides, the proposed mechanism was also proved to be deadlock free. However, the proposed system is limited by the size of the network of nodes. For extensive networks (nodes > 1000), there could be considerable delays while searching for a robot to be serviced by a mobile agent. The situation may further worsen (due to an increase in delays) if the available bandwidth for communication is low. One solution to control such a limitation is to use pheromone-conscientious migration strategy [69], which allows for a bi-directional search on the part of the robotic nodes and the agents across the network. Tackling

the complexity of the tasks whose programs are carried by the mobile agents, opens up future scope for the research. While on one side a box-pushing operation may need just a single robot, complex tasks, such as carrying an extended object, may require multiple robots to synchronize their actions. After reserving the necessary number of robots, the concerned agent within this proposed system could release a new set of agents or clones capable of performing the relevant tasks as also synchronizing them using stigmergy based mechanisms as discussed in [87]. The Job Distributor (J_{Dist}) is the only central entity in the proposed system which could be replaced by its distributed version. This means that instead of a single entity that does the task of book-keeping of the allocated resources, each node could act as a scaled-down version of the proposed J_{Dist}. These mini versions of the J_{Dist} could then communicate with one another to ensure a distributed locking mechanism. This work can also be extended to scenarios where different robots have different abilities such that some subset of robots are capable of executing only specific tasks. The challenge would then be to choose just those robots which are fit for the execution of the required tasks. Since physical processes are performed in the real world, the entities of a typical CPS are bound to encounter unpredictable changes in the environment. Further, these physical processes inherently operate at a different time-scale leading to the creation of an asynchronous environment within the system.

In the previous contribution (Chapter 4), the solution to execute a task by a robot was assumed to be the only best one available. However, in real situations, this may not be the case. There could be several ways to perform the same task. It is desirable that, from the many solutions, only the one which is the best in terms of, for instance, the least task completion time or maximum battery conservation, be chosen. Hence, the approach developed in the previous chapter was extended and complemented with a truly decentralized and distributed solution selection system. In this contribution (Chapter 6), an mechanism to find the best solution for a given set of problems in *d*CPS scenarios was proposed. The developed approach was inspired by the selection properties found in the BIS. An extension to the work by Jha et al. [89], this contribution not only applies Idiotypic Networks to reward the better solutions but also integrates concepts from the clonal selection

and Danger theories to attract and clone superior solutions. The problems occurring at a node in a *d*CPS forms an antigen while the solutions represent the antibodies. The objective was to find a mapping which selects the best solution for a given problem without any global knowledge. The distressed nodes release danger signals in the form of problem signatures across their peers. The mobile agents carrying the relevant mappings are attracted towards this node facing a problem. This is equivalent to an enhanced concurrent search in the decentralized networked system. Once the desired mobile agents reach the distress node, an Idiotypic Network is formed wherein better solutions are stimulated while inferior ones are suppressed. This is similar to a reinforcement-based mechanism wherein the superior quality of solutions are rewarded. The results gathered from extensive simulation experiments validate the objectives of the proposed approach to select the best solutions in both static and dynamic environments. Further, a proof-of-concept implementation on a set of robots proved the applicability of the technique in a real-world *d*CPS. This mechanism can also be used to deliver services in a vehicular network or to a music recommendation application in a locally formed IoT devoid of any cost-intensive cloud server.

The final contribution of this thesis (Chapter 7) ameliorates the previously proposed decentralized and distributed online selection algorithm. The work in Chapter 6 assumed an infinite stream of solutions. However, in real environments, the system may need to generate and evolve new solutions based on the performance feedback of the currently available solutions. This enhanced contribution augments the selection mechanism with an evolutionary strategy to create fresh solutions as per the needs thereby making the *d*CPS self-sustainable and adaptive. In contrast to traditional evolutionary approaches wherein a single solution is evolved to cater to the whole task, the mechanism proposed in this work partitions the entire problem space into subtasks and evolves different solutions for each of the subtasks. Using cross-reactivity threshold as a single parameter, the division of the problem space can be controlled. Three scenarios with varying complexity were constructed to test the mechanism. The experiments conducted using a set of real robots validated the feasibility of the approach. The comparison with a traditional approach proved the efficacy of the proposed mechanism. Besides, the use of mobile agents for sharing

the solutions among the robots further aided in conserving the power consumption. The results favored the utilization of mobile agents when the network is dense and suggested the use of broadcasting for sparse networks. Along with applications to a *d*CPS of robots, the mechanism can also be employed for selecting and evolving avatars in computer games.

8.2 Future Research Avenues

The work reported in the chapters of this thesis provide ample scope and promulgate several clear directions for future research endeavors. This thesis in no way dissuades the readers from the use of centralization but points in the direction of persuading multi-layered hybrid architectures for control. For instance, the horizontal exploration could be decentralized while retaining centralization among the vertical layers. Though *Tartarus*, the first contribution of this thesis is still in its nascent form, it has a strong potential to act as a middleware. One immediate upgrade to *Tartarus* could be the support for Android-based mobile devices. The smartphones continuously collect data from a user. A *Tartarus* instance running on such devices can greatly extend the scope of applications such as opportunistic location-aware message passing, local mobile intelligence, appliance control, etc. As an agent platform, FIPA [137] compliance will also be an essential add-on to the *Tartarus*. Plugins for interfacing other embedded systems and robots could increase the heterogeneity of this platform. Support for inter-language communication such as Python and Java can further add to the ubiquity of *Tartarus*. Providing mobile agent based mechanisms proposed by Jha et al. [92, 90, 91] and Godfrey et al. [67, 65] in addition to the ones proposed in this thesis as packages, will greatly contribute towards making *Tartarus* as a rapid prototyping tool. Last but not least, ensuring the security of the agents needs to be looked upon. The use of Blockchain Technology [210] can provide some pointers for securing agents in *Tartarus*.

The work on ordering task executions in a *d*CPS of multiple robots currently forms a single sequential pipeline. A significant extension to this work could be to allow multiple parallel pipelines to overlap the type of tasks the robots perform. A further research direction can be to create agent behaviors which will enable more

than one robot to share the same resource alternatively or simultaneously to speed up the pipelined processing. Another research perspective is the job distributor whose current implementation is naïve. The robotic nodes can themselves distribute the tasks as and when they arrive. This will aid in reducing the central component and balancing the overall *d*CPS. The mechanisms for selection and evolution of solutions described in Chapters 6 and 7 can be augmented with techniques such as Genetic Programming (GP) [13] to create novel heuristics. This will contribute towards unique and better quality of solutions. Communication techniques which can reduce the power consumption in the devices and robots forming the *d*CPS is another avenue of research that needs to be investigated.

With the mass growth in the number of sensing and computing devices, the limit of a pure centralized notion of intelligence is becoming visible. This could be the right time to exploit non-traditional inspirations such as those from Biology to develop algorithms which are scalable, robust, distributed, versatile, incorporate decentralization, and learn and evolve continuously during their lifetime.

Bibliography

[1] The boeing 787 produces over 500gb of data during every flight. https://www.geek.com/geek-cetera/the-boeing-787-produces-over-500gb-of-data-during-every-flight-1542105/. Accessed: 2018-10-11.

[2] The first electrical appliance turns 100 years. https://newatlas.com/go/4824/. Accessed: 2018-10-11.

[3] Internet of things (IoT) history. https://www.postscapes.com/internet-of-things-history/. Accessed: 2018-10-11.

[4] Least realistic new year's resolution ever. https://goo.gl/fMkQZE. Accessed: 2018-10-23.

[5] The little-known story of the first iot device-industries blog. https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/. Accessed: 2018-10-11.

[6] Networked robots. http://www.ieee-ras.org/technical-committees/117-technical-committees/networked-robots/146-networked-robots. Accessed: 2018-10-23.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[8] Anchal, P. Saini, and C. Krishna. An Efficient Permission-Cum-Cluster Based Distributed Mutual Exclusion Algorithm for Mobile Adhoc Networks. In *2014 International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pages 141–146, Dec 2014.

[9] H. Attiya, A. Kogan, and J. Welch. Efficient and robust local mutual exclusion in mobile ad hoc networks. *Mobile Computing, IEEE Transactions on*, 9(3):361–375, March 2010.

[10] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[11] R. Baheti and H. Gill. Cyber-physical Systems. *The Impact of Control Technology*, (1):161—-166, 2011.

[12] M. Baker. How the internet of cells has biologists buzzing. *Nature News*, 549(7672):322, 2017.

[13] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco, 1998.

[14] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic. *Mobile ad hoc networking*. John Wiley & Sons, 2004.

[15] F. Bellifemine, A. Poggi, and G. Rimassa. JADE: a FIPA2000 compliant agent development environment. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 216–217, 2001.

[16] A. Bieszczad, B. Pagurek, and T. White. Mobile agents for network management. *IEEE Communications Surveys & Tutorials*, 1(1):2–9, 1998.

[17] J. Bloomer. *Power programming with RPC*. " O'Reilly Media, Inc.", 1992.

[18] M. Bode, S. S. Jha, and S. B. Nair. A Mobile Agent based Autonomous Partial Green Corridor Discovery and Maintenance Mechanism for Emergency Services amidst Urban Traffic. In *The First International Conference on IoT in Urban Space*, pages 13–18, Rome, 2014. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering.

[19] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, G. Théraulaz, G. Theraulaz, et al. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[20] J. Bongard. Behavior chaining-incremental behavior integration for evolutionary robotics. In *ALIFE*, pages 64–71, 2008.

[21] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[22] E. Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014.

[23] S. C. Botelho and R. Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1234–1239, 1999.

[24] A. Boukerche, R. B. Machado, K. R. Jucá, J. a. B. M. Sobral, and M. S. Notare. An agent based and biological inspired real-time intrusion detection and security model for computer network operations. *Computer Communications*, 30(13):2649–2660, 2007.

[25] N. Bredeche, E. Haasdijk, and A. Eiben. On-line, on-board evolution of robot controllers. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 110–121. Springer, 2009.

[26] N. Bredeche, E. Haasdijk, and A. Prieto. Embodied evolution in collective robotics: A review. *arXiv preprint arXiv:1709.08992*, 2017.

[27] S. Bulgannawar and N. Vaidya. A distributed k-mutual exclusion algorithm. In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pages 153–160, May 1995.

[28] F. M. Burnet. The Clonal Selection Theory of Acquired Immunity. *Vanderbilt University Press, Nashville, Tennessee*, 1959.

[29] M. Calle and J. Kabara. Measuring energy consumption in wireless sensor networks using gsp. In *2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–5, Sept 2006.

[30] N. Capodieci, E. Hart, and G. Cabri. Designing self-aware adaptive systems: From autonomic computing to cognitive immune networks. In *2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, pages 59–64, Sept 2013.

[31] L. Catarinucci, D. De Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone. An iot-aware architecture for smart healthcare systems. *Internet of Things Journal, IEEE*, 2(6):515–526, 2015.

[32] K. M. Chandy and J. Misra. The Drinking Philosophers Problem. *ACM Trans. Program. Lang. Syst.*, 6(4):632–646, Oct. 1984.

[33] M. Chen, S. Gonzalez, and V. Leung. Applications and design issues for mobile agents in wireless sensor networks. *Wireless Communications, IEEE*, 14(6):20–26, 2007.

[34] R. Chertov, S. Fahmy, and N. Shroff. Emulation versus Simulation: A case study of TCP-targeted denial of service attacks. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM*, pages 10 pp.–325, 2006.

[35] D. Chess, C. Harrison, A. Kershenbaum, and T. J. Watson. Mobile agents: Are they a good idea?, 1995.

[36] M. Chetty, J.-Y. Sung, and R. E. Grinter. How smart homes learn: The evolution of the networked home and household. In *International Conference on Ubiquitous Computing*, pages 127–144. Springer, 2007.

[37] H. N. Chu, A. Glad, O. Simonin, F. Sempe, A. Drogoul, and F. Charpillet. Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*, volume 1, pages 442–449, Oct 2007.

[38] C. A. C. Coello, D. C. Rivera, and N. C. Cortes. Use of an artificial immune system for job shop scheduling. In *International Conference on Artificial Immune Systems*, pages 1–10. Springer, 2003.

[39] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, June 1971.

[40] M. Cusumano. Cloud computing and saas as new computing platforms. *Communications of the ACM*, 53(4):27–29, 2010.

[41] J. Dangerfield, T. McCarthy, and W. Ellery. The mound-building termite macrotermes michaelseni as an ecosystem engineer. *Journal of tropical Ecology*, 14(4):507–520, 1998.

[42] D. Dasgupta. *An Overview of Artificial Immune Systems and Their Applications*, pages 3–21. Springer, 1999.

[43] L. N. De Castro and J. Timmis. Artificial immune systems: a novel approach to pattern recognition. 2002.

[44] L. N. De Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002.

[45] S. Dhenakaran and A. Parvathavarthini. An Overview of Routing Protocols in Mobile Ad-Hoc Network. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(2):251 —- 259, 2013.

[46] M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.

[47] D. M. Dobkin. *The RF in RFID: UHF RFID in Practice*. Newnes, 2012.

[48] M. Duarte, J. Gomes, S. M. Oliveira, and A. L. Christensen. Evorbc: Evolutionary repertoire-based control for robots with arbitrary locomotion complexity. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 93–100, 2016.

[49] M. Duarte, S. Oliveira, and A. L. Christensen. Hierarchical evolution of robotic controllers for complex tasks. In *Development and Learning and Epigenetic*

Robotics (ICDL), 2012 IEEE International Conference on, pages 1–6. IEEE, 2012.

[50] M. Duarte, S. M. Oliveira, and A. L. Christensen. Hybrid control for large swarms of aquatic drones. In *Proceedings of the 14th International Conference on the Synthesis & Simulation of Living Systems. MIT Press, Cambridge, MA*, pages 785–792. Citeseer, 2014.

[51] R. Faragher and R. Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+'14)*, 2014.

[52] J. D. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1):187–204, 1986.

[53] L. D. Fife and L. Gruenwald. Research Issues for Data Communication in Mobile Ad-hoc Network Database Systems. *ACM SIGMOD Record*, 32(2):42–47, June 2003.

[54] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. 1994. In *Proceedings of the Third International Conference on Simulation of Adaptive behavior: From Animals to Animats*, volume 3.

[55] F. S. Fogliatto, G. J. Da Silveira, and D. Borenstein. The mass customization decade: An updated review of the literature. *International Journal of Production Economics*, 138(1):14–25, 2012.

[56] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent agents III agent theories, architectures, and languages*, pages 21–35. Springer, 1996.

[57] A. A. Freitas. A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explorations Newsletter*, 6(2):77–86, 2004.

[58] J. Gaber and M. Bakhouya. Mobile Agent-Based Approach for Resource Discovery in Peer-to-Peer Networks. In *Agents and Peer-to-Peer Computing*, pages 63–73, 2008.

[59] N. Gangwar, T. Semwal, and S. B. Nair. Care: An iot based system for passenger service and comfort in railways. In *Communication Systems and Networks (COMSNETS), 2017 9th International Conference on*, pages 55–62. IEEE, 2017.

[60] Gartner. Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015, 2015. [Online; accessed 20-February-2016].

[61] B. P. Gerkey and M. J. Mataric. Principled Communication for Dynamic Multi-Robot Task Allocation. *Lecture Notes in Control and Information Sciences*, 271:353–362, 2001.

[62] B. P. Gerkey and M. J. Mataric. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *IEEE International Conference on Robotics and Automation, ICRA '03*, volume 3, pages 3862–3868, Sept 2003.

[63] A. Ghosh, R. Ratasuk, B. Mondal, N. Mangalvedhe, and T. Thomas. Lte-advanced: next-generation wireless broadband technology. *IEEE wireless communications*, 17(3), 2010.

[64] W. W. Godfrey, S. S. Jha, and S. B. Nair. On a mobile agent framework for an internet of things. In *International Conference on Communication Systems and Network Technologies (CSNT), 2013*, pages 345–350. IEEE, 2013.

[65] W. W. Godfrey, S. S. Jha, and S. B. Nair. On stigmergically controlling a population of heterogeneous mobile agents using cloning resource. In *Transactions on Computational Collective Intelligence XIV*, pages 49–70. Springer, 2014.

[66] W. W. Godfrey and S. B. Nair. An immune system based multi-robot mobile agent network. In *Artificial Immune Systems*, pages 424–433. Springer, 2008.

[67] W. W. Godfrey and S. B. Nair. Mobile agent cloning for servicing networked robots. In *Principles and Practice of Multi-Agent Systems*, pages 336–339. Springer, 2010.

[68] W. W. Godfrey and S. B. Nair. A Mobile Agent Cloning Controller for Servicing Networked Robots. In *International Conference on Future Information Technology IPCSIT*, volume 13, pages 81–85. IACSIT Press, 2011.

[69] W. W. Godfrey and S. B. Nair. A Pheromone Based Mobile Agent Migration Strategy for Servicing Networked Robots. In *Bio-Inspired Models of Network, Information, and Computing Systems*, pages 533–541. Springer, 2012.

[70] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[71] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[72] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[73] R. S. Gray. Agent tcl: A transportable agent system. In *Proceedings of the ciKM Workshop on intelligent information Agents, Fourth international conference on information and Knowledge Management (ciKM 95), Baltimore, Maryland*, 1995.

[74] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[75] H. Guo. *Algorithm Selection for Sorting and Probabilistic inference: A machine learning approach*. PhD thesis, Department of Computing and Information Sciences, College of Engineering, Kansas State University, 2003.

[76] V. Hadzilacos. A note on group mutual exclusion. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, PODC '01, pages 100–106, New York, NY, USA, 2001. ACM.

[77] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont. An artificial immune system architecture for computer security applications. *IEEE Transactions on Evolutionary Computation*, 6(3):252–280, Jun 2002.

[78] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile Agents : Are they a good idea ? Technical report, 1995.

[79] E. Hart, D. Davoudani, and C. McEwan. Immunological inspiration for building a new generation of autonomic systems. In *Proceedings of the 1st international conference on Autonomic computing and communication systems*, page 9. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[80] E. Hart and K. Sim. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation*, 24(4):609–635, 2016.

[81] J. Heinerman, A. Zonta, E. Haasdijk, and A. E. Eiben. On-line evolution of foraging behaviour in a population of real robots. In *European Conference on the Applications of Evolutionary Computation*, pages 198–212. Springer, 2016.

[82] A. Ishiguro, T. Kondo, Y. Watanabe, and Y. Uchikawa. Dynamic behavior arbitration of autonomous mobile robots using immune networks. In *IEEE International Conference on Evolutionary Computation*, volume 2, pages 722–727. IEEE, 1995.

[83] D. E. Jackson and F. L. Ratnieks. Communication in ants. *Current biology*, 16(15):R570–R574, 2006.

[84] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

[85] N. K. Jerne. Towards the network theory of the immune system. *Ann. Immunol.(Inst. Pasteur)*, 125:373–389, 1974.

[86] S. S. Jha. *On Mobile Agents for Learning & Coordination in a Networked Robotics Milieu*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, 2016.

[87] S. S. Jha, W. W. Godfrey, and S. B. Nair. Stigmergy-Based Synchronization of a Sequence of Tasks in a Network of Asynchronous Nodes. *Cybernetics and Systems*, 45(5):373–406, June 2014.

[88] S. S. Jha and S. B. Nair. A Logic Programming Interface for Multiple Robots. In *3rd National Conference on Emerging Trends and Applications in Computer Science, NCETACS*, pages 152–156. IEEE, 2012.

[89] S. S. Jha and S. B. Nair. An idiotypic solution sieve for selecting the best performing solutions in real-world distributed intelligence. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2015 3rd International Conference on*, pages 71–76. IEEE, 2015.

[90] S. S. Jha and S. B. Nair. On a multi-agent distributed asynchronous intelligence-sharing and learning framework. In N. T. Nguyen, editor, *Transactions on Computational Collective Intelligence XVIII*, pages 166–200, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[91] S. S. Jha and S. B. Nair. Tansa: Task allocation using nomadic soft agents for multirobot systems. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(4):308–318, 2018.

[92] S. S. Jha, K. Shrivastava, and S. B. Nair. On emulating real-world distributed intelligence using mobile agent based localized idiotypic networks. In R. Prasath and T. Kathirvalavakumar, editors, *Mining Intelligence and Knowledge Exploration*, pages 487–498, Cham, 2013. Springer International Publishing.

[93] D. Johansen, R. van Renesse, and F. B. Schneider. Supporting Broad Internet Access to TACOMA. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, number 1, pages 55–58, 1996.

[94] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[95] D. Kahneman and P. Egan. *Thinking, fast and slow*, volume 1. Farrar, Straus and Giroux New York, 2011.

[96] D. Kahneman and A. Tversky. Prospect theory: An analysis of decision under risk. In *Handbook of the fundamentals of financial decision making: Part I*, pages 99–127. World Scientific, 2013.

[97] Y. Kambayashi and M. Takimoto. Higher-order mobile agents for controlling intelligent robots. *International Journal of Intelligent Information Technologies (IJIIT)*, 1(2):28–42, 2005.

[98] Y. Kambayashi, Y. Tsujimura, H. Yamachi, M. Takimoto, and H. Yamamoto. Design of a multi-robot system using mobile agents with ant colony clustering. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10. IEEE, 2009.

[99] K. Kamei, S. Nishio, N. Hagita, and M. Sato. Cloud networked robotics. *IEEE Network*, 26(3), 2012.

[100] K. Kant and P. Mohapatra. Scalable internet servers: issues and challenges. *SIGMETRICS Performance Evaluation Review*, 28(2):5–8, 2000.

[101] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. *IEEE communications surveys & tutorials*, 13(4):584–616, 2011.

[102] T. Kawamura and K. Sugahara. A mobile agent-based p2p e-learning system. *IPSJ Journal*, 46(1):222–225, 2005.

[103] Y. Khaluf and F. Rammig. Task Allocation Strategy for Time-Constrained Tasks in Robots Swarms. In *European Conference on Artificial Life*, number 2009, pages 737–744, 2013.

[104] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. Future internet: The internet of things architecture, possible applications and key challenges. In *10th International Conference on Frontiers of Information Technology (FIT)*, pages 257–260, Dec 2012.

[105] K.-D. Kim and P. Kumar. An overview and some challenges in cyber-physical systems. *Journal of the Indian Institute of Science*, 93(3):341–352, 2013.

[106] J. Kinnunen, G. Krishnamurthy, K. Huhtanen, P. Jussila, and K. Ratschunas. Location dependent services, 2004. US Patent 6,813,501.

[107] M. Klusch, S. Lodi, and G. Moro. Agent-based distributed data mining: The kdec scheme. In *Intelligent Information Agents*, volume 2586 of *Lecture Notes in Computer Science*, pages 104–122. Springer Berlin Heidelberg, 2003.

[108] D. E. Knuth. *The art of computer programming: Sorting and Searching*, volume 3. Addison-Wesley, 1981.

[109] P. Korst and H. Velthuis. The nature of trophallaxis in honeybees. *Insectes Sociaux*, 29(2):209–221, 1982.

[110] S. Kumar and S. K. Singh. Monitoring of pet animal in smart cities using animal biometrics. *Future Generation Computer Systems*, 2016.

[111] V. Kumar, D. Rus, and G. S. Sukhatme. Networked robots. In *Springer Handbook of Robotics*, pages 943–958. Springer, 2008.

[112] H. Kuwahara. Experiences teach us the future of autonomous decentralized systems. In *Autonomous Decentralized Systems, 1997. Proceedings. ISADS 97., Third International Symposium on*, pages 169–175. IEEE, 1997.

[113] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Commun. ACM*, 42(3):88–89, Mar. 1999.

[114] E. A. Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015.

[115] J. Lee, B. Bagheri, and H.-A. Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[116] W.-P. Lee. Evolving complex robot behaviors. *Information Sciences*, 121(1-2):1–25, 1999.

[117] T. Leppänen. *Resource-Oriented Mobile Agent and Software Framework for the Internet of Things*. PhD thesis, 2018.

[118] C. Liu, J. Yang, R. Chen, Y. Zhang, and J. Zeng. Research on immunity-based intrusion detection technology for the internet of things. In *2011 Seventh International Conference on Natural Computation*, volume 1, pages 212–216, July 2011.

[119] C. M. Liu, R. Chen, and C. Chen. An artificial immune-based distributed intrusion detection model for the internet of things. In *Advanced Research on Material Engineering, Architectural Engineering and Informatization*, volume 366, pages 165–168. Trans Tech Publications, 1 2012.

[120] R. B. Machado, A. Boukerche, J. Sobral, K. Juca, and M. Notare. A hybrid artificial immune and mobile agent intrusion detection based model for computer network operations. In *19th IEEE International Proceedings on Parallel and Distributed Processing Symposium, 2005*, pages 1–8. IEEE, 2005.

[121] P. Maes. Artificial life meets entertainment: Lifelike autonomous agents. *Communications of the ACM*, 38(11):108–114, 1995.

[122] P. Maes, R. H. Guttman, and A. G. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, 1999.

[123] J. Matani and S. B. Nair. Typhon - A Mobile Agents Framework for Real World Emulation in Prolog. In *Multi-disciplinary Trends in Artificial Intelligence*, pages 261–273. Springer, 2011.

[124] P. Matzinger. The Danger Model: A renewed sense of self. *Science*, 296(5566):301–305, 2002.

[125] M. S. Michael. Universal asynchronous receiver/transmitter, 1992. US Patent 5,140,679.

[126] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[127] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes. Hive: Distributed agents for networking things. *IEEE Concurrency*, 8:24–33, 2000.

[128] N. Minar, K. Kramer, and P. Maes. Cooperating mobile agents for mapping networks. pages 34–41. MIT Media Lab, in Proceedings of the First Hungarian National Conference on Agent Based Computing, 1998.

[129] R. Minerva, A. Biru, and D. Rotondi. Towards a definition of the internet of things (iot). *IEEE Internet Initiative*, 1:1–86, 2015.

[130] F. G. Miskelly. Assistive technology in elderly care. *Age and ageing*, 30(6):455–458, 2001.

[131] R. C. Moioli, P. A. Vargas, F. J. Von Zuben, and P. Husbands. Towards the evolution of an artificial homeostatic system. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 4023–4030. IEEE, 2008.

[132] S. Mostinckx, T. Van Cutsem, S. Timbermont, E. G. Boix, E. Tanter, and W. De Meuter. Mobile-C: a mobile agent platform for mobile C/C++ agents. *Software - Practice and Experience*, 39(July):661–699, 2009.

[133] S. Moyer, D. Maples, S. Tsang, and A. Ghosh. Service portability of networked appliances. *IEEE Communications Magazine*, 40(1):116–121, 2002.

[134] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, 2009.

[135] S. Nolfi. Evolutionary robotics: Exploiting the full power of self-organization. 1998.

[136] L. Null and J. Lobur. *The essentials of Computer Organization and Architecture*. Jones & Bartlett Publishers, 2014.

[137] P. D. O'Brien and R. C. Nicol. Fipa—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[138] P. J. O'Dowd, M. Studley, and A. F. Winfield. The distributed co-evolution of an on-board simulator and controller for swarm robot behaviours. *Evolutionary Intelligence*, 7(2):95–106, 2014.

[139] A. Outtagarts. Mobile agent-based applications: A survey. *International Journal of Computer Science and Network Security*, 9(11):331–339, 2009.

[140] M. P. Papazoglou. Agent-oriented technology in support of e-business. *Communications of the ACM*, 44(4):71–77, 2001.

[141] L. E. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220–240, 1998.

[142] L. Parolini, N. Tolia, B. Sinopoli, and B. H. Krogh. A cyber-physical systems approach to energy management in data centers. In *Proceedings of the 1st acm/ieee international conference on cyber-physical systems*, pages 168–177. ACM, 2010.

[143] P. N. Pathirana, N. Bulusu, A. V. Savkin, and S. Jha. Node localization using mobile robots in delay-tolerant sensor networks. *IEEE transactions on Mobile Computing*, 4(3):285–296, 2005.

[144] H. Peine and T. Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *Proc of the First International Workshop on Mobile Agents MA97*, volume 1219, pages 50–61. 1997.

[145] A. S. Perelson and G. F. Oster. Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self-non-self discrimination. *Journal of theoretical biology*, 81(4):645–670, 1979.

[146] V. A. Pham and A. Karmouch. Mobile software agents: an overview. *IEEE Communications Magazine*, 36(7):26–37, July 1998.

[147] M. Pilkington. 11 blockchain technology: principles and applications. *Research handbook on digital transformations*, page 225, 2016.

[148] M. Pipattanasomporn, H. Feroze, and S. Rahman. Multi-agent systems in a distributed smart grid: Design and implementation. In *IEEE/PES Power Systems Conference and Exposition, 2009. PSCE '09.*, pages 1–8, 2009.

[149] J. Posadas, J. Poza, J. Simó, G. Benet, and F. Blanes. Agent-based distributed architecture for mobile robot control. *Engineering Applications of Artificial Intelligence*, 21(6):805–823, Sept. 2008.

[150] F. L. W. Ratnieks and C. Anderson. Task partitioning in insect societies. *Insectes Sociaux*, 46(2):95–108, 1999.

[151] M. Raynal. Algorithms for mutual exclusion. *The MIT Press, Cambridge, MA*, 1986.

[152] A. Raza and B. R. Fernández. A multi-tier immuno-inspired framework for heterogeneous mobile robotic systems. *Applied Soft Computing*, 71:333 – 352, 2018.

[153] W. Ren, R. Beard, and E. Atkins. A survey of consensus problems in multi-agent coordination. In *American Control Conference, 2005. Proceedings of the 2005*, pages 1859–1864 vol. 3, June 2005.

[154] B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. Ieee, 2009.

[155] T. Robles, R. Alcarria, D. M. de Andrés, M. Navarro, R. Calero, S. Iglesias, and M. López. An iot based reference architecture for smart water management processes. *JoWUA*, 6(1):4–23, 2015.

[156] N. C. Rowe. *Artificial Intelligence Through Prolog*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[157] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3293–3298. IEEE, 2012.

[158] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[159] W. Sadiq and F. Cummins. *Developing Business Systems with CORBA*. Cambridge University Press, New York, NY, USA, 1998.

[160] I. Satoh. Mobilespaces: A framework for building adaptive distributed applications using a hierarchical mobile agent system. In *Proceedings of 20th International Conference on Distributed Computing Systems, 2000*, pages 161–168. IEEE, 2000.

[161] P. Schreyer. The contribution of information and communication technology to output growth. *OECD Science, Technology and Industry Working Papers*, No. 2000/02, 2000.

[162] F. Sempe and A. Drogoul. Adaptive patrol for a group of robots. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2865–2869 vol.3, Oct 2003.

[163] T. Semwal, M. Bode, V. Singh, S. S. Jha, and S. B. Nair. Tartarus: A Multi-agent Platform for Integrating Cyber-Physical Systems and Robots. In *Proceedings of the 2015 Conference on Advances In Robotics*, AIR '15, pages 20:1–20:6, New York, NY, USA, 2015. ACM.

[164] T. Semwal, S. S. Jha, and S. B. Nair. On ordering multi-robot task executions within a cyber physical system. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 12(4):20, 2017.

[165] T. Semwal, D. D. Kulkarni, and S. B. Nair. On an immuno-inspired distributed, embodied action-evolution cum selection algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 141–148. ACM, 2018.

[166] T. Semwal and S. B. Nair. Agpi: Agents on raspberry pi. *Electronics*, 5(4):72, 2016.

[167] T. Semwal, S. Nikhil, S. S. Jha, and S. B. Nair. Tartarus: A multi-agent platform for bridging the gap between cyber and physical systems. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1493–1495. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[168] J. Shi, J. Wan, H. Yan, and H. Suo. A survey of Cyber-Physical Systems. In *2011 International Conference on Wireless Communications and Signal Processing, WCSP 2011*, pages 1–6, 2011.

[169] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[170] K. Shrivastava, S. S. Jha, and S. B. Nair. Autonomous mobile robot navigation using artificial immune system. In *Proceedings of Conference on Advances In Robotics*, pages 1–7. ACM, 2013.

[171] F. Shrouf, J. Ordieres, and G. Miragliotta. Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm. In *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 697–701, Dec 2014.

[172] P. Siano. Demand response and smart grids—a survey. *Renewable and sustainable energy reviews*, 30:461–478, 2014.

[173] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146 – 164, 2015.

[174] A. Silva, M. M. Da Silva, and J. Delgado. An overview of agentspace: a next-generation mobile agent system. In *Mobile Agents*, pages 148–159. Springer, 1998.

[175] F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen. Open issues in evolutionary robotics. *Evolutionary computation*, 24(2):205–236, 2016.

[176] K. Sim, E. Hart, and B. Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary computation*, 23(1):37–67, 2015.

[177] E. D. Simoes and K. R. Dimond. Embedding a distributed evolutionary system into a population of autonomous mobile robots. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 2, pages 1069–1074. IEEE, 2001.

[178] H. Tai and K. Kosaka. The Aglets project. *Communications of the ACM*, 42(3):100–101, 1999.

[179] M. Takimoto, M. Mizuno, M. Kurio, and Y. Kambayashi. Saving energy consumption of multi-robots using higher-order mobile agents. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 549–558. Springer, 2007.

[180] P. Tarau. Binprolog: A continuation passing style prolog engine. In M. Bruynooghe and M. Wirsing, editors, *Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 479–480. Springer Berlin Heidelberg, 1992.

[181] P. Tarau. Jinni: Intelligent mobile agent programming at the intersection of java and prolog. In *Proc. of PAAM*, volume 99, pages 109–123, 1999.

[182] G. Theraulaz and E. Bonabeau. Modelling the collective building of complex architectures in social insects with lattice swarms. *Journal of Theoretical Biology*, 177(4):381 – 400, 1995.

[183] J. Timmis, M. Neal, and J. Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1):143–150, 2000.

[184] M. Tortonesi, A. Morelli, M. Govoni, J. Michaelis, N. Suri, C. Stefanelli, and S. Russell. Leveraging internet of things within the military network environment—challenges and solutions. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 111–116. IEEE, 2016.

[185] G. Y. Tsipenyuk. *Evaluation of decentralized email architecture and social network analysis based on email attachment sharing*. PhD thesis, 2018.

[186] L. M. Vaquero and L. Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *SIGCOMM Comput. Commun. Rev.*, 44(5):27–32, Oct. 2014.

[187] Y. Watanabe, A. Ishiguro, and Y. Uchikawa. Decentralized behavior arbitration mechanism for autonomous mobile robot using immune network. In

Artificial immune systems and their applications, pages 187–209. Springer, 1999.

[188] A. Watkins. *Exploiting immunological metaphors in the development of serial, parallel and distributed learning algorithms*. PhD thesis, Computing Laboratory, 2005.

[189] A. Watkins, J. Timmis, and L. Boggess. Artificial Immune Recognition System (AIRS): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, 2004.

[190] R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 2002.

[191] J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014.

[192] A. Whitbrook, U. Aickelin, and J. Garibaldi. An idiotypic immune network as a short-term learning architecture for mobile robots. In *International Conference on Artificial Immune Systems*, pages 266–278. Springer, 2008.

[193] A. M. Whitbrook, U. Aickelin, and J. M. Garibaldi. Idiotypic immune networks in mobile-robot control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(6):1581–1598, 2007.

[194] A. M. Whitbrook, U. Aickelin, and J. M. Garibaldi. Two-timescale learning using idiotypic behaviour mediation for a navigating mobile robot. *Applied Soft Computing*, 10(3):876–887, 2010.

[195] L. D. Whitley. Fundamental principles of deception in genetic search. In *Foundations of genetic algorithms*, volume 1, pages 221–241. Elsevier, 1991.

[196] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12:67–96, 1 2012.

[197] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[198] O. Wolfson, S. Chamberlain, K. Kalpakis, and Y. Yesha. Modeling moving objects for location based services. In *Developing an Infrastructure for Mobile and Wireless Systems*, pages 46–58. Springer, 2001.

[199] O. Wolfson, P. Sistla, B. Xu, J. Zhou, S. Chamberlain, Y. Yesha, and N. Rishe. Tracking moving objects using database technology in domino. In *Next Generation Information Technologies and Systems*, pages 112–119. Springer, 1999.

[200] D. Wong, N. Pacictek, T. Walsh, J. Dicelie, M. Yotmg, and B. Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. In *Mobile Agents*, pages 86–97. Springer Berlin Heidelberg, 1997.

[201] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2009.

[202] M. Wooldridge and N. Jennings. Agent theories, architectures, and languages: A survey. In M. Wooldridge and N. Jennings, editors, *Intelligent Agents*, volume 890 of *Lecture Notes in Computer Science*, pages 1–39. Springer Berlin Heidelberg, 1995.

[203] W. Wu, J. Zhang, A. Luo, and J. Cao. Distributed mutual exclusion algorithms for intersection traffic control. *Parallel and Distributed Systems, IEEE Transactions on*, 26(1):65–74, Jan 2015.

[204] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[205] M. Yogeswaran and S. Ponnambalam. Swarm robotics: An extensive research review. In *Advanced Knowledge Application in Practice*. InTech, 2010.

[206] Y. Yoshimura, A. Krebs, and C. Ratti. An analysis of visitors' length of stay through noninvasive bluetooth monitoring in the louvre museum. *CoRR*, abs/1605.00108, 2016.

[207] O. Zaiane. Building a recommender agent for e-learning systems. In *International Conference on Computers in Education, 2002. Proceedings.*, pages 55 – 59, 2002.

[208] Y. Zhang, L. Wang, W. Sun, R. C. Green, and M. Alam. Artificial immune system based intrusion detection in a distributed hierarchical network architecture of smart grid. In *2011 IEEE Power and Energy Society General Meeting*, pages 1–8, July 2011.

[209] P. Zhao, S. Suryanarayanan, and M. G. Simões. An energy management system for building structures using a multi-agent decision-making control methodology. *IEEE Transactions on Industry Applications*, 49(1):322–330, 2013.

[210] Z. Zheng, S. Xie, H.-N. Dai, and H. Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 1:1–25, 2016.

[211] Y. Zuo and J. Liu. A reputation-based model for mobile agent migration for information search and retrieval. *International Journal of Information Management*, 37(5):357 – 366, 2017.

Publications

Journals

- **Tushar Semwal**, Shashi Shekhar Jha and Shivashankar B Nair. "On Ordering Multi-Robot Task Executions within a Cyber Physical System", ACM Transactions on Autonomous and Adaptive Systems, 12, 4, Article 20 (November 2017), 27 pages.

- **Tushar Semwal** and Shivashankar B. Nair (2016). "AgPi: Agents on Raspberry Pi", Special Issue of the Journal - Electronics: "Raspberry Pi Technology", MDPI Open Access.

- **Tushar Semwal** and Shivashankar B. Nair. "On an Immuno-inspired Selection Mechanism for Decentralized Cyber-Physical Systems", Applied Soft Computing, Elsevier. [Under Review]

Conferences

- **Tushar Semwal**, Divya D Kulkarni, Shivashankar B. Nair (2018). On an Immuno-inspired Distributed, Embodied Action-Evolution cum Selection Algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference* (GECCO), Kyoto, Japan. pp. 141-148.

- **Tushar Semwal**, Nikhil S., Shashi Shekhar Jha, Shivashankar B. Nair (2016). TARTARUS: A Multi Agent Platform for Bridging the gap between Cyber and Physical Systems. In *Proceedings of 2016 Autonomous Agents and Multi Agent Systems Conference* (AAMAS), Singapore. pp. 1493-1495.

- **Tushar Semwal**, Manoj Bode, Vivek Singh, Shashi Shekhar Jha, Shivashankar B. Nair (2015). Tartarus: A Multi-Agent Platform for Integrating Cyber-

Physical Systems and Robots. In *Proceedings of 2015 Conference on Advances In Robotics* (AIR), Goa, India. Article No. 20.

- **Tushar Semwal**, Shivashankar B. Nair (2018). MAVNet: A Mobile Agent based framework for Vehicular Networks. In *Proceedings of third International Conference On Internet of Things: Smart Innovation and Usages* (IoT-SIU), Bhimtal,pp. 1-6.

Other publications (Not related to thesis work)

- Ishita, Divya D Kulkarni, **Tushar Semwal**, Shivashankar B. Nair (2019). On Securing Mobile Agents using Blockchain Technology. In *Second International Conference on Advanced Computational and Communication Paradigms*, Sikkim, India. [Under Review]
- Rahul Mishra, **Tushar Semwal**, Shivashankar B. Nair (2018). A Distributed Epigenetic Shape Formation and Regeneration Algorithm for a Swarm of Robots. In *Workshop of Genetic and Evolutionary Computation Conference* (GECCO), Kyoto, Japan. pp. 1505-1512.
- **Tushar Semwal**, Gaurav Mathur, Promod Yenigalla, Shivashankar B. Nair (2018). A Practitioners' Guide to Transfer Learning for Text Classification using Convolutional Neural Networks. In *Proceedings of SIAM Conference on Data Mining* (SDM), San Diego, USA. pp. 513-521.
- Nitu Gangwar, **Tushar Semwal**, Shivashankar B. Nair (2017). CARE: An IoT based System for Passenger Service and Comfort in Railways. In *Proceedings of* 9^{th} *International Conference on COMmunication Systems & NETworkS* (COMSNETS), Bengaluru, India. pp. 55-62.
- Nikhil S., **Tushar Semwal** and Shivashankar B. Nair (2016). Immuno-Inspired Behaviour Adaptation in Multi-Robot Systems. In *Proceedings of IEEE Conference on Systems, Man and Cybernatics* (SMC), Budapest, Hungary. pp. 3199-3204.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

Appendices

.1 Conscientious Migration Strategy

In the Conscientious Migration Strategy [128], the mobile agents migrate to the neighboring node only when that node has not been not visited or happens to be the least visited one. In order to keep track of the visited nodes, a mobile agent appends the recently visited node to a list, comprising the nodes already visited, maintained within itself. Thus before moving to the next node, an agent checks if the next visited node is a member of this list. If so, it chooses another neighboring node or the least visited neighbor.

.2 Query Processing

Figure 1 shows 10 Pi-nodes connected to each other in a topology similar to the geographical map of a floor of a building. The 10 Pi-nodes are denoted by a, b, c, d, e, f, g, h, i, j and their corresponding zonal areas by Z_{P_a}, Z_{P_b},.., and Z_{P_j} respectively. For the sake of simplicity, only part of the database relevant to agent routing is shown in each Pi-node. Let us assume that the path followed by a BLE tag bearer X is: $d \rightarrow e \rightarrow f \rightarrow i \rightarrow h$.

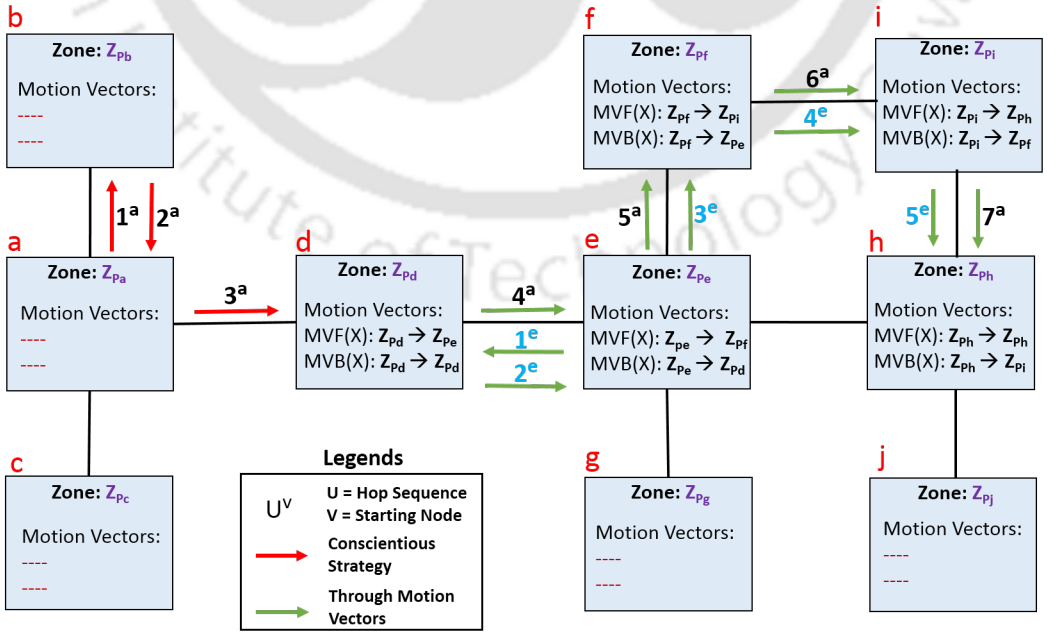


Figure 1: Agent migration in the proposed approach

In the figure, MVF(X) and MVB(X) denotes Moving Vector Forward and Moving Vector Backward for X respectively. Imagine a user fires a query from node a to trace X. The associated mobile agent now has 3 neighboring nodes (b, c and d) to migrate. Since there is no motion vector for X in the node a, the mobile agent opts for the conscientious strategy and chooses one of these neighbors. Assume that it chooses node b and migrates to it. Since the motion vectors for X are absent in b, the conscientious strategy forces it to backtrack to a. If it now selects node d and migrates to it, the motions vectors of X within d will force the agent to switch to the strategy of following motion vectors. From d onwards, the motion vectors will guide the agent through nodes e, f, i and h in that order and thereby retrieve the trace for X.

If the query was fired from node e , since the MVB for X point to node d, the agent migrates to d, after which it follows the MVFs to discover the trace $d \rightarrow e \rightarrow f \rightarrow i \rightarrow h$ as in the previous case.