



Dharma Teja Srungavruksham

## **Development of Teleoperation Software for Wheeled Mobile Robot**

**School of Electrical Engineering**

**Department of Electrical Engineering and Automation**

Thesis submitted in partial fulfillment of the requirements for the degree of  
Master of Science in Technology

Espoo, May 23, 2016

Instructor:

Professor Riku Jäntti  
Aalto University  
School of Electrical Engineering

Supervisors:

Professor Ville Kyrki  
Aalto University  
School of Electrical Engineering

Ms. Anita Enmark  
Luleå University of Technology

# Acknowledgement

I thank Prof. Riku Jäntti for providing me with the working opportunity to study and implement teleoperation software as part of their efforts to realise better living through the use of cellular based network technologies like Machine-to-Machine and Internet of Things. I highly appreciate his patience throughout my thesis work, his assistance and access to department's resources helped me to work in the premises and conduct experiments. I would also like to extend my thanks to the research team, Shariatmadari Hamidreza for his help with demo testing and advice, Ali Yusein for his assistance in using communication nodes and Iraji Sassan. I am grateful for all the help extended by resource manager Viktor Nässi, without whom it would have been difficult to manage experiment logistics.

I thank my supervisor Prof. Ville Kyrki for letting me pursue thesis work with another department and his understanding of the work. I appreciate the assistance of Tomi Ylikorpi and Salama Annika throughout my spacemaster studies at Aalto Univeristy.

Finally, I thank Ms. Anita Enmark for being my supervisor and Victoria Barabash from Luleå University of Technology and Erasmus mundus programme for their support to spacemaster which gave me an international exposure and great friends for life. I thank my family and friends who were always there for me and understood my purpose to pursue this degree.

Espoo, May, 2016

Dharma Teja Srungavruksham

<b>Author:</b>	Dharma Teja Srungavruksham	
<b>Title of the thesis:</b>	Development of Teleoperation Software for Wheeled Mobile Robot	
<b>Date:</b>	May 23, 2016	<b>Number of pages:</b> 10+78
<b>Department:</b>	Department of Electrical Engineering and Automation	
<b>Programme:</b>	Master's Degree Programme in Space Science and Technology	
<b>Professorship:</b>	Automation Technology (AS-84)	
<b>Supervisors:</b>	Professor Ville Kyrki (Aalto) Ms. Anita Enmark (LTU)	
<b>Instructor:</b>	Professor Riku Jäntti	
<p>Wireless technology in our daily lives is giving a way to more and more inter connected devices. More increasingly innovative applications are being developed for daily usage ranging from simple sensing devices to autonomous robots. With all these extra additions, the burden on wireless technologies such as WLAN and 3G/4G/LTE is leading to the development of new network architectures and protocols. Teleoperation of remote devices are finding their way into common places using those technologies. In this thesis work a teleoperation software was developed to use wireless serial devices to control a remote robot over the network and perform autonomous tasks under the supervision of an operator. The robot is an indoor wheeled mobile robot and operator's device is a computer, both running on Windows OS. This system is similar to a Wireless Sensor Networks with actuators and sensors being on the robot device while the control brain is on a remote computer. The full system has several components like graphics for robot parameters, settings for communications, modes of operations for operator and robot's own localization and safety tasks. Field tests validated the full functionality of the system but in four out of nine trials failure of wireless devices caused complete system paralysis. An autonomous trajectory following operation was implemented to study the effects of packet loss in communication, it was found that control was reliable even with 26% drop. With a linear driving test it was also observed that robot's free moving wheel was causing an orientation error adding an extra <math>0.06^\circ</math> when moving backwards. A continuous transmission of data packets in the network ensures reliability in the system, this is very important from operator's perspective.</p>		
<b>Keywords:</b> teleoperation, telerobotics, mobile robots, Pioneer 3-Dx, wireless networks		

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Objectives . . . . .	2
1.3	System overview . . . . .	3
1.4	Thesis overview . . . . .	5
<b>2</b>	<b>Related Work</b>	<b>6</b>
2.1	Control architecture . . . . .	7
2.1.1	Direct control . . . . .	9
2.1.2	Telecommanding . . . . .	10
2.1.3	Behaviour based model . . . . .	10
2.1.4	Hierarchy level based model . . . . .	11
2.2	Communications . . . . .	12
2.2.1	TCP/UDP . . . . .	14
2.2.2	IEEE 802 family . . . . .	15
2.3	Control interface . . . . .	15
2.3.1	GUI and input device . . . . .	16
2.3.2	Camera views . . . . .	16
2.3.3	Time followers vision . . . . .	17
2.3.4	Operator command strategy . . . . .	19
2.4	Summary . . . . .	20
<b>3</b>	<b>Trajectory Tracking</b>	<b>21</b>
3.1	Nonlinear State Tracking Controller . . . . .	22
<b>4</b>	<b>Tools and Framework</b>	<b>27</b>
4.1	Wheeled Mobile Robot (WMR): Pioneer 3-DX . . . . .	28



4.1.1	Robot Hardware . . . . .	28
4.1.2	Robot Software . . . . .	30
4.2	User Interface framework . . . . .	33
4.3	Wireless Communication . . . . .	36
<b>5</b>	<b>Implementation</b>	<b>39</b>
5.1	Communication data packet . . . . .	39
5.1.1	Payload Data . . . . .	40
5.2	Robot application development . . . . .	43
5.2.1	ArRobot . . . . .	44
5.2.2	Transmission . . . . .	47
5.2.3	Reception . . . . .	48
5.3	Desktop application development . . . . .	49
5.3.1	GUI . . . . .	49
5.3.2	Transmission . . . . .	54
5.3.3	Reception . . . . .	54
<b>6</b>	<b>Experiment setup</b>	<b>57</b>
6.1	Setup . . . . .	57
6.2	Test description . . . . .	61
<b>7</b>	<b>Results and discussion</b>	<b>62</b>
7.1	Functionality . . . . .	62
7.1.1	Navigation . . . . .	62
7.1.2	Linear drive operation . . . . .	64
7.1.3	Trajectory following operation . . . . .	64
7.1.4	Anti collision safety . . . . .	66
7.2	Performance . . . . .	68
<b>8</b>	<b>Conclusions</b>	<b>72</b>
8.1	Conclusive remarks . . . . .	72
8.2	Future Work . . . . .	73
	<b>References</b>	<b>74</b>

# List of Tables

4.1	Useful Server Information Packets (SIP) contents . . . . .	31
5.1	Header and Trailer elements description . . . . .	41
5.2	Control velocity command data payload from <b>Controller</b> . . .	41
5.3	Transmission interval command payload from <b>Controller</b> . . .	42
5.4	Velocity command data payload from <b>Controller</b> . . . . .	42
5.5	Set transmission command payload from <b>Controller</b> . . . . .	42
5.6	Position data payload from <b>Robot</b> . . . . .	42
7.1	Packet drops in testing . . . . .	71
7.2	Elliptic path completion time and packet drop . . . . .	71

# List of Figures

1.1	Block diagram of complete systems operation . . . . .	4
2.1	Components of a teleoperation system [1] . . . . .	7
2.2	Basic control paradigms [2] . . . . .	8
2.3	Direct control model [3] . . . . .	9
2.4	Quadrifolium drawing with direct control [3] . . . . .	9
2.5	Telecommanding model [4] . . . . .	10
2.6	Behaviour based model [5] . . . . .	11
2.7	Level architecture with advanced speed and gesture input [6] . .	12
2.8	Standard OSI model . . . . .	13
2.9	Standard protocols distributed on OSI model <sup>1</sup> . . . . .	13
2.10	Comparing local control loop with remote control loop [3] . . . .	14
2.11	Cockpit interface [7] . . . . .	16
2.12	Different camera view for robot environment [8] . . . . .	17
2.13	Three camera view points for rescue mission [9] . . . . .	18
2.14	Operator performance with three different views [9] . . . . .	18
2.15	Operator interface with a robot model [10] . . . . .	19
2.16	Block diagram of the system [10] . . . . .	19
2.17	(a) Navigation time and (b) position error for command strategies [11] . . . . .	20
3.1	Trajectory tracking control diagram . . . . .	21
3.2	Robot motion in global reference frame . . . . .	23
3.3	Trajectory tracking of robot state . . . . .	24
4.1	Teleoperation setup . . . . .	27
4.2	Pioneer 3-DX robot . . . . .	28
4.3	Advanced Robot Control and Operations Software (ARCOS) server- client control architecture [12] . . . . .	30

4.4	ArRobot task cycle . . . . .	33
4.5	Signal and slot mechanism . . . . .	34
4.6	Communication node . . . . .	36
4.7	Hardware stack . . . . .	37
5.1	Structure of a standard packet with its size . . . . .	40
5.2	Robot application flow . . . . .	43
5.3	Zone check task in ArRobot's flow . . . . .	44
5.4	Safety zones and maximum velocities . . . . .	45
5.5	Robot packet transmission (Tx) . . . . .	47
5.6	Robot packet reception (Rx) . . . . .	48
5.7	Ui application flow . . . . .	50
5.8	User interface application . . . . .	51
5.9	Interactive graphics scene zooming . . . . .	51
5.10	Ui packet transmission (Tx) . . . . .	55
5.11	Ui packet reception (Rx) . . . . .	56
6.1	Experiment area images . . . . .	58
6.2	Packet transmission settings . . . . .	58
6.3	Experimental area map . . . . .	59
6.4	Operator interface in action . . . . .	60
7.1	Robot path using dead reckoning . . . . .	63
7.2	Robot path with laser localization . . . . .	63
7.3	Actual robot position (far end of the image) vs calculated position (graphics scene of UI) . . . . .	64
7.4	Linear robot motion between two point . . . . .	65
7.5	Velocities for linear controlled drive . . . . .	65
7.6	Trajectory following map . . . . .	66
7.7	Trajectory tracking errors . . . . .	66
7.8	Safety activated on the map . . . . .	67
7.9	Velocity alteration for safety . . . . .	67
7.10	Free wheel effect on orientation . . . . .	68
7.11	Free wheel effect on orientation . . . . .	69
7.12	Robot control with packet loss . . . . .	70

# List of Algorithms

5.1	Velocities update for zone check . . . . .	46
5.2	Reference target generator . . . . .	52
5.3	Motion control . . . . .	53
5.4	Linear control . . . . .	54

# Symbols and Abbreviations

**ARCOS** Advanced Robot Control and Operations Software

**ARIA** Advanced Robotics Interface for Applications

**ARNL** Advanced Robotics Navigation and Localization

**AROS** ActivMedia Robotics Operating System

**SIP** Server Information Packets

**SONARNL** Sonar based Advanced Robotics Navigation and Localization

**WMR** Wheeled Mobile Robot

# Chapter 1

## Introduction

*“Everything should be made as simple as possible, but not simpler.”*

- Albert Einstein

Since its inception wireless communication has penetrated into every field of work and everybody’s life, demand for connectivity and information sharing is increasing exponentially. In mobile phone usage alone there are over 6.8 billion devices [13] and predictions promise for very steep growth. Remote systems could be anything from simple sensor relaying raw data to an autonomous robot operating in other planets. In the area of field and service robotics mobile robots are quite commonly used to solve a remote task while a human controller might directly operate the robot or set an action for remote execution. Robots can also be programmed to have a level of intelligence that does not require human interventions which make it easy to execute repetitive tasks on its own. We see many such mobile robot applications in space and military usage. In space application the robot might have to survive the harsh climates of Mars or Moon by reacting to changing environment conditions by using sophisticated sensors yet, conduct the basic research tasks it was meant to do with the guidance of commands sent from thousands of miles away [14]. In military applications reliable communications are essential to operating very expensive and sophisticated robotic equipment and save lives, most importantly the ones used for drones, bomb disposal robot and land mine detection [2, 15]. Other types of mobile robots in extensive use are in warehouse management where they have to navigate through a complex maze of shelves and conduct storage/retrieval activities. New concepts like auto driving road vehicles, mail delivery drones and many more are emerging. Another application where teleoperation is taken with utmost care is in safety, security and rescue operations [16, 17]. In all the before mentioned scenarios the complex interaction between machine and human can be divided as machine-to-machine, human-to-machine, human-to-machine-to-human etc. operations. Such connectivity requires a reliable, safe, efficient and modular teleoperation architecture.

## 1.1 Motivation

This project work is a part of [Machine-to-Machine Redefining Information Sharing and Enablers \(M2MRISE\)](#) [18] initiative, partnering universities and multinational companies. A test bed for different configurations of machine-to-machine communications is being carried out in Communications and networks department of Aalto university. As the commercial use of communications getting more personal and number of devices connected to it are increasing in use the need to test smart solutions configuring the existing networks will help in achieving more reliable and dependable network solutions.

In order to study those different communication setups and their properties a test bed was needed so, teleoperation type setup with an indoor wheeled mobile robot was chosen. The goal was to develop software with a user interface that can be used by a human operator to command and control a robot remotely. It should have the capability to monitor and modify packet data communication rates, provide visuals of robot's sensor information and enable by default to accommodate any type of communication node that can receive a serial data via USB. The software on the robot side should have basic safety build into it and ability to localize. In order to operate a robot with different speed and take into consideration the communication related effects like data packet loss and radio interference an appropriate strategy had to be created. With basic operational software the communication nodes can be tested in different scenarios and study its properties.

With an indoor environment for robot operations an experimental area with some narrow and wide areas was chosen. It is possible to drive the robot to different corners of the room and make studies. Some useful teleoperation mode behaviours were selected to simulate varying speeds of robot and communication node distances. The ability to alter the transmission rates on the fly provides a real life operational scenario for testing. All data packets from operator/controller side and robot side are time stamped and stored in local files for later retrieval and to conduct further analysis.

## 1.2 Objectives

The primary goal of this research work is to develop software that would enable teleoperation of a Wheeled mobile robot called Pioneer 3-Dx using Communications and networking department's lab-made wireless nodes. A simulator had to be used for most of the development part finding errors in code and debugging. The robot API library was supposed to be integrated into robot application. It was important to ensure all sharing of memory objects among parallel processes is done properly. To achieve these goals the whole problem is decomposed into several sub-parts as listed below:



1. Research the previous work done in implementations of teleoperation software development, its components like a control interface and feedback, communication, safety and autonomous behaviours.
2. To form a 2D digital “map” of experimental lab area which is to be used for robot’s localization.
3. Data packet definitions to accommodate several command packet types and robots sensory information.
4. Develop software for robot’s on-board computer to use the library called ARIA (Advance Robot Interface for Applications) provided by Adept MobileRobots and have the following features:
  - (a) Interpret a command packet received via wireless node.
  - (b) Globally localize the robot in a map using laser and sonar sensors using the proprietary library called ARNL from Adapt MobileRobots.
  - (c) Implement a smart collision avoidance with on-board sensors.
  - (d) Transmit the robot state, current velocities and sensor information.
5. Develop software for a remote desktop computer to have the following features:
  - (a) User Interface that allows operator to send commands to the robot, view robot’s location on the map, visualize sensor information, monitor packet data rates and change setting.
  - (b) Implement three modes of operation first: to control the robot with the keyboard, second: rapid linear testing where the robot moves between two points in a straight line and third: an automatic trajectory tracking controller.
6. Conduct experiments and test for functionality of the system with different operation modes. Analyse results to understand the performance issues and system limitations.

## 1.3 System overview

A complete system architecture is shown as a block diagram in Figure 1.1. This embodies thesis objectives mentioned earlier. The main parts of this teleoperation software are operator and a mobile robot. The communication link between them is a wireless node using IEEE 802.15.4 protocol.

The operator has control over operational modes of robot and the transmission (Tx) settings of the robot and operator. A control interface shows robot’s

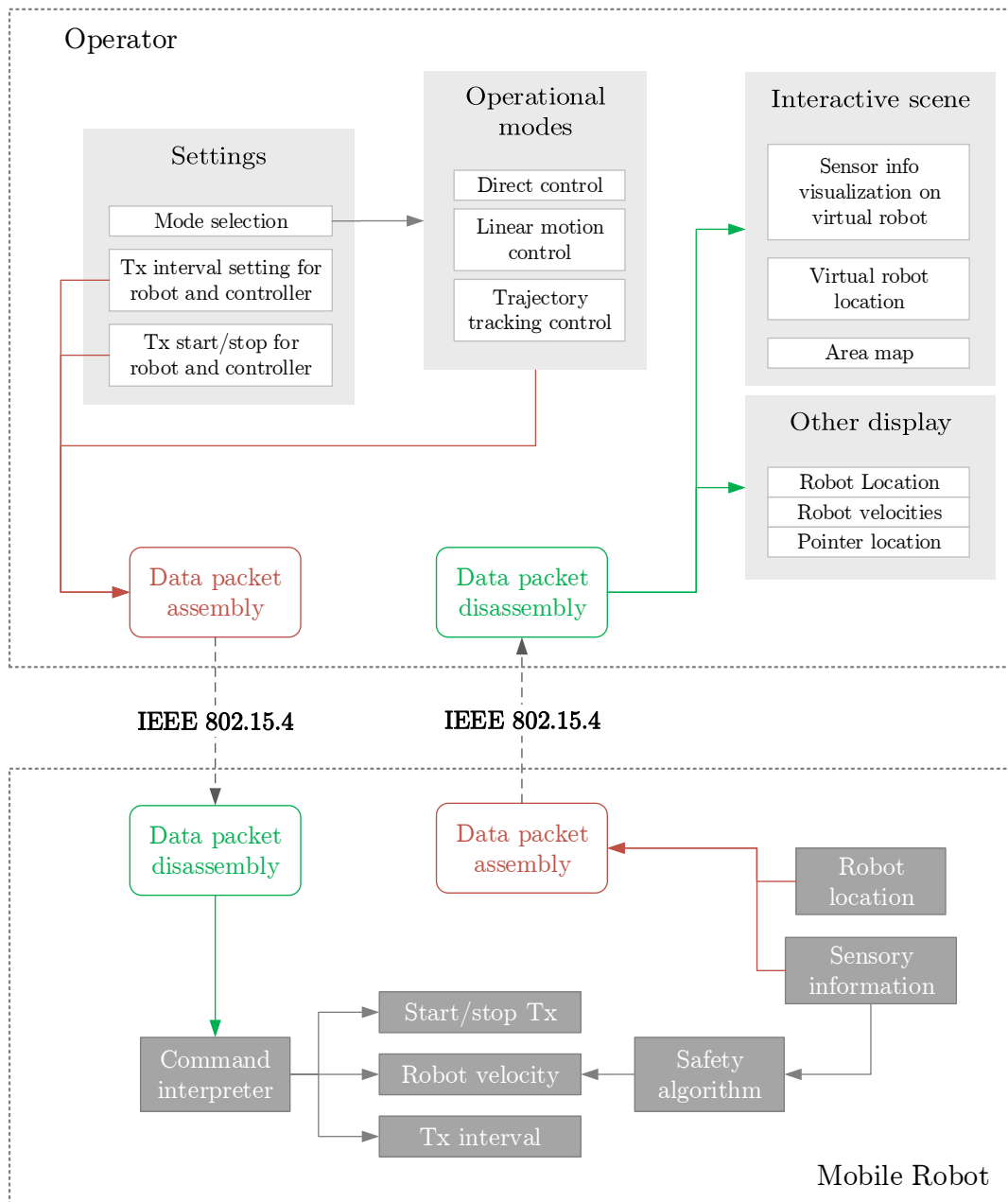


Figure 1.1: Block diagram of complete systems operation

location with a graphically drawn accurate robot, the map of the experimental area, sensor information projected on the robot and current velocities.

On the Mobile robot side there is a command interpreter to parse relevant commands for starting/stopping of Tx, setting translational and rotational velocities and changing Tx interval. A safety algorithm is always running in parallel that can directly alter robot's velocities. The robot's location is determined either by odometry information or laser localization, which together with other sensory readings is assembled into a data packet.

There is a data packet assembler and a disassembler on both sides that are

transferring packets to the wireless node via USB link.

## 1.4 Thesis overview

This thesis discusses the development and testing of various software components in teleoperation system they include a graphical user interface, data packets, a safety algorithm, trajectory tracking control algorithm and a transmission/reception module.

Thesis briefing is as follow :

1. Current chapter discusses why a teleoperation study is important, motivation behind building one. The objectives of the thesis work.
2. Chapter 2 is all about state-of-the-art work done in this field showing clearly the architecture of use, control interfaces, sensor use and wireless communication.
3. Trajectory tracking algorithm related theory is discussed in chapter 3. It shows how the algorithm was derived from the basic kinematic model of the robot and how it fits into the whole trajectory tracking and control loop.
4. Chapter 4 discusses the tools and frameworks used in the project work. The robot that was used, communication nodes, software libraries, API's and frameworks.
5. Chapter 5 discusses the implementation part of the project, here the designs of several software components are detailed with flowcharts and algorithms behind them.
6. To conduct experiments, it is better to know the system before. In chapter 6 a description of using the system, setting it up and testing objectives are discussed.
7. Chapter 7 discusses the results obtained from testing to validate functionality and analyse system performance. The observations are mentioned with supporting graphs and tables.
8. Conclusions are made in chapter 8, discussing what was achieved in this thesis work and what limitations were observed. It also proposes some promising future expansion based on current work.

# Chapter 2

## Related Work

This chapter discusses on the literature behind teleoperation of mobile robots. There are several components of a teleoperation that are discussed with reference to previous work, each with their own benefits but it has to be noted that different applications must require certain configurations of teleoperation components.

### Definitions

- Teleoperation “to operate a vehicle or system over a distance” [2].
- Teleoperator Any machine that is enabling human to sense its environment and access to control its actuators [19].
- Telerobot “is a subclass of teleoperator in which the machine acts as a robot for short periods, but is monitored by a human supervisor and reprogrammed from time to time” [19].
- Operator “A human operator is the person who monitors the operated machine and takes the control actions needed” [2].

In telerobotics the two main applications are in mobile robots and mobile manipulators of which mobile robots fit into this thesis context. Teleoperation of a mobile robot is fundamentally a control problem with wider loops involving one or more wireless radio communication nodes. One of the earliest work defines teleoperation to have three basic components namely sensors, controls and man-machine interface [20]. Controls and sensory information is transmitted by the operator and robot respectively via some communication medium i.e. wired, radio, WLAN, etc., in order for the operator to have an optimal perspective of the robot environment an appropriate interface should be provided. The arrangement of these components, complexity of shared information

among them and involvement of human control over the robot can be done in several different ways, design of such a ‘Control architecture’ requires understanding of application goals. Figure 2.1 show a basic control architecture in a teleoperation.

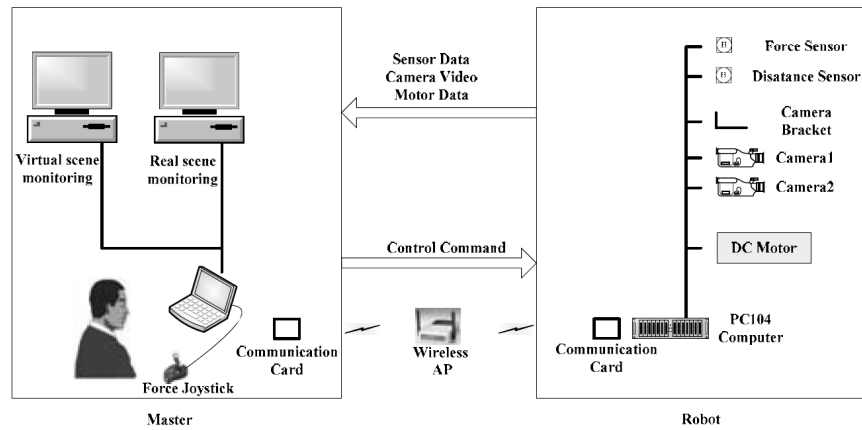


Figure 2.1: Components of a teleoperation system [1]

Re-arranging the original idea of important teleoperation components to a more modern sophisticated systems [2], the system can be grouped into three parts:

- Control architecture - Higher level overall design
- Communications - Involving one or more protocol
- Control interface - Sensory information presentation and obtaining user commands

## 2.1 Control architecture

The designing of higher control loops and connecting them in different ways with lower or basic control loops to form a teleoperation system can be thought of as an architectural task. When an operator sends a direct signal that alters robot state and in turn receives the feedback of the changed state then such a paradigm is a simple master-slave arrangement [21] (direct control). This basic architecture is the root for many teleoperation systems, at some point in any architecture this sort of direct control must be used. Going a step further there might be certain tasks that robot has to execute on it is own with a little control loop and the task selection is done by an operator, such an arrangement is called task-based control [2, 22] (or supervisory control).

As shown in Figure 2.2a operator commands are direct values to robot actuators, in Figure 2.2b the local control loop drives actuator to a desired set point coming from the operator and in Figure 2.2c operator has ability to choose

from different operational tasks from his interface thus giving him supervisor ability to have multiple types of control over robot actions. From this basic design, more operator-robot interactions can be added like behavior control, supervised autonomy, shared control, coordinated control, collaborative control etc. All such paradigms demand an integration of an operator throughout a teleoperation process.

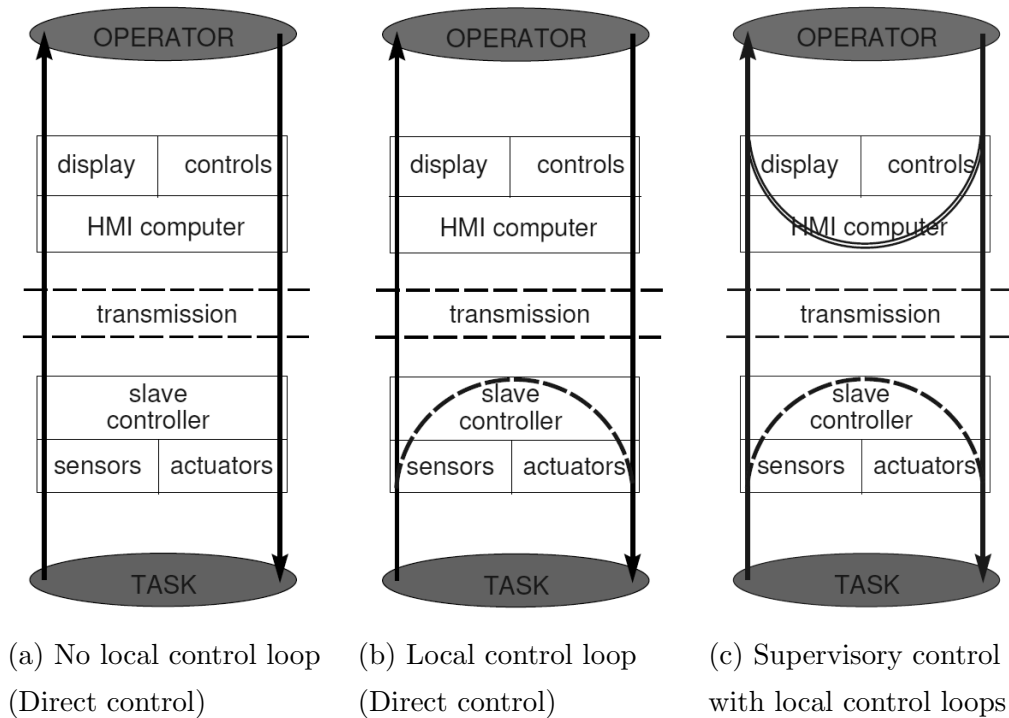


Figure 2.2: Basic control paradigms [2]

The main functions of a supervisor according to [19] are:

1. Planning - this involves reading of the robot environment and forming a strategy while also taking into account the physical limitations of the system.
2. Teaching - sending commands relevant to the situation and deciding on a way to deliver that command.
3. Monitoring - it involves reading the robot state and digesting the priority information from the robot, this in order to detect anomaly and take actions.
4. Intervening - in case of totally unpredicted behaviour taking emergency initiative or taking direct control of the robot or making minute adjustments in the middle of a task execution.

Communications can prove difficult when large amount of data is being exchanged between an operator and the robot due to reasons like limited band-

width, external interference, time-delays, off-order packet delivery and information loss, every developing teleoperation solution must face such a situation at some point. To solve such issues, solutions as network models, predictive interfaces and others have been used [23]. They are not discussed in this thesis as it is one of those problems in a teleoperation system that has its own research challenges. Following subsections discuss on some of the architectures that have been used in literature.

### 2.1.1 Direct control

To understand a basic direct control design with an example [3], a remote 2DOF robot arm with a pen was operated using a visual feedback from a robot camera. Operator command was basically end-effector's coordinate position which the robot had to reach with an encoder based feedback control loop. For communications, they used UDP (User Datagram Protocol) with packet delay time of over a minute. Though for some critical teleoperation scenarios where network delay can cause major chaos this may not be practical. Figure 2.3 shows the block diagram of that model. The experiment was done with open loop and closed loop arm control to draw a quadrifolium, the difference in performance is shown in Figure 2.4b and 2.4c. This teleoperation system with its basic components can prove to be efficient.

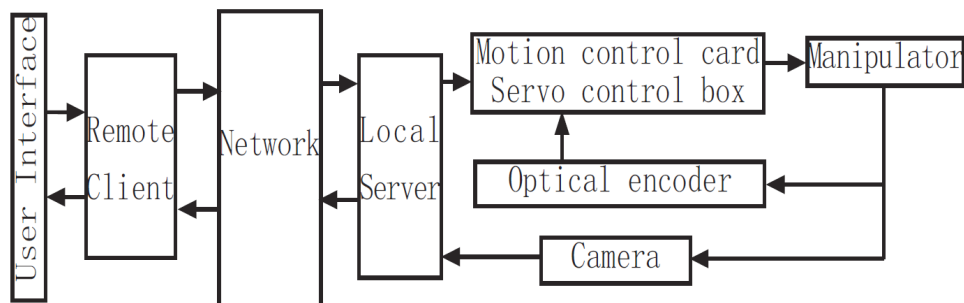


Figure 2.3: Direct control model [3]

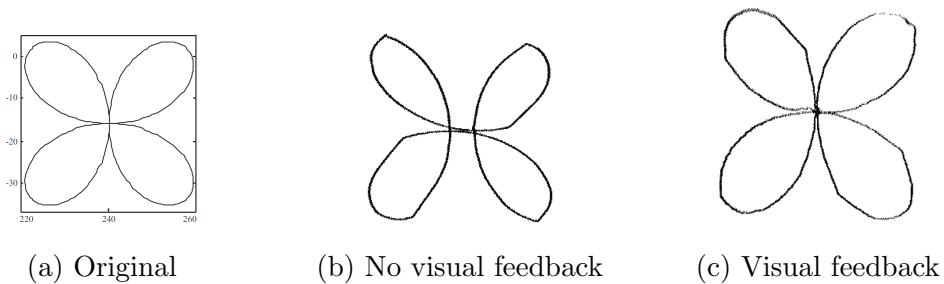


Figure 2.4: Quadrifolium drawing with direct control [3]

### 2.1.2 Telecommanding

Telecommanding is basically sending commands to a remote robot over a network. To deepen the human-machine interaction [4] proposes a telecommanding paradigm with a basic joystick command and advanced ‘linguistic’ commands. This implementation had wireless LAN and a radio transceiver for commanding and media purposes respectively. This type of model is termed ‘interactive teleoperation’, whose block diagram is shown in Figure 2.5. Just like in a natural scenario where people give verbal directions to a certain location, this advanced telecommanding is designed to take similar linguistic commands and parse them at the robot to execute it with sensor information forming a local control loop. These linguistic commands are typed on operator’s GUI. Whenever the operator chooses to use basic telecommanding (i.e. joystick) the command parser overrides any previous values from control loops with new values.

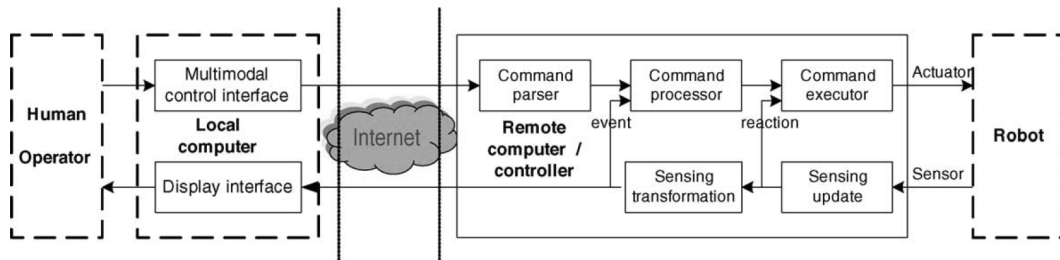


Figure 2.5: Telecommanding model [4]

Such a model enables for continuous operator command following by the robot. All the components are tightly integrated making it function like a single unit. This is a measure of an increase in human-machine interaction and giving priority to operators command even in the middle of a task.

### 2.1.3 Behaviour based model

In this model, behaviours are programmed to perform some actions, they are activated on the robot either on command by the operator or based on sensory input. In the example below [5] defines three behaviours ‘user following’, ‘obstacle avoidance’ and ‘goal reaching’. ‘Supervisory layer’ monitors the sonar information and network status like delay in the packet arrival, loss of packet and jumbled packet order, to select an appropriate behaviour.

They have used a joystick for operator’s direction input over ADSL internet and on the robot side a 3G wireless node for communication. User following behaviour uses fuzzy logic to determine operator’s forward, backward, left and right commands which are then translated to left and right wheel velocity. Obstacle avoidance uses a similar logic but the data comes from sensors which activate left, forward and right direction based on fuzzy rules. In goal reaching



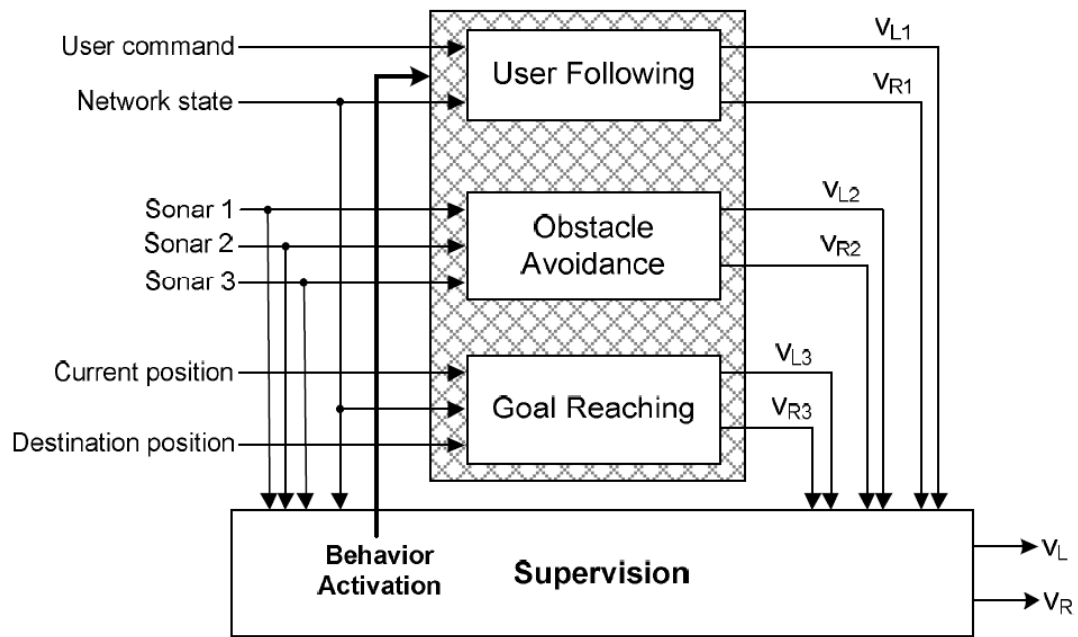


Figure 2.6: Behaviour based model [5]

behaviour the operator gives new position to which robot needs to steer and the corresponding velocities are generated by robot kinematic laws. Behaviour based control is also very much advantageous in a multi-robot teleoperation system where robots need to coordinate among themselves to solve a task [24]. Only limitations mentioned in [5] are due to unknown network behaviour and speed of teleoperation i.e. time delay between the operator and the robot. Under time-delay conditions some tasks can be left to robots discretion given the robot has enough sensory knowledge of its environment.

#### 2.1.4 Hierarchy level based model

When the robot environment is unknown and there are too many variables on the ground, a simple collection of robot behaviours is not enough there needs to be an architecture as in [25, 6] that has reasonable intelligence to integrate an operator and robot as tightly as possible and maybe also involve other teams, practically deciding upon what level of operations should be allowed with a situation in hand.

[6] implemented such an intelligent architecture giving the operator a view of the environment with the help of name tags for different visible objects [6, Figure 11] and ability to command the robot with natural verbal sentences and gestures. It has three levels of operations combined called ‘world model’, they are mission level, interaction level and attention level. Here operator and robot are termed as team mates, similar to a real world scenario where team of people try to accomplish a mission. At all these levels, the generated sensor data is

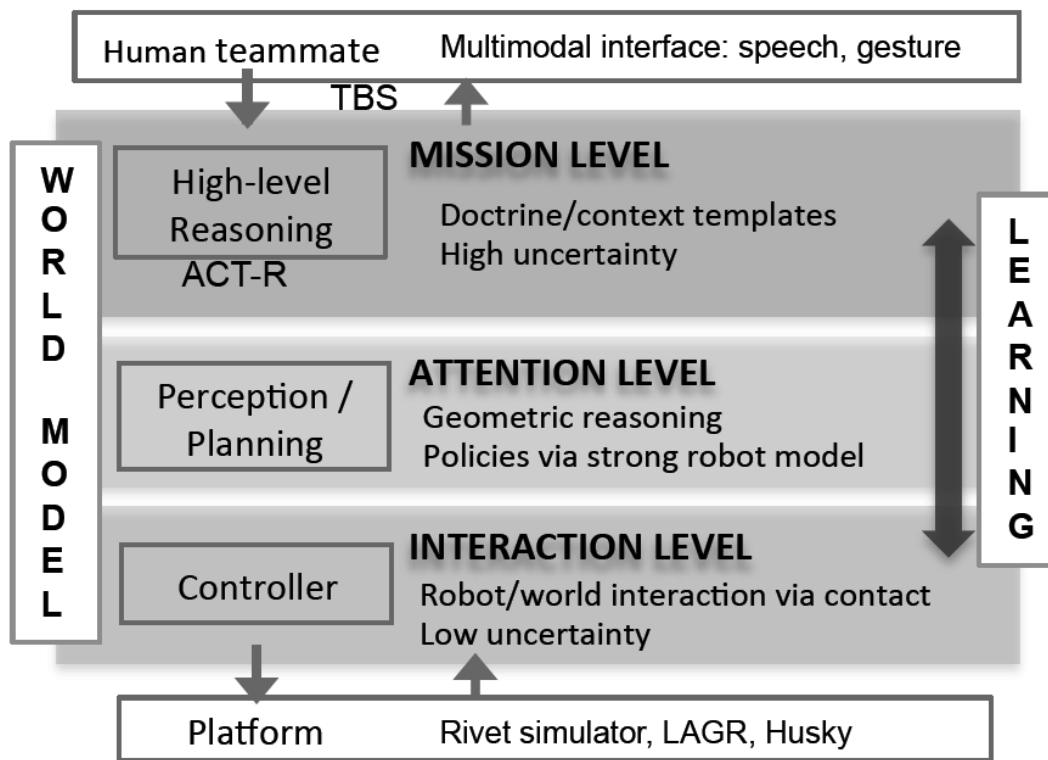


Figure 2.7: Level architecture with advanced speed and gesture input [6]

stored to be computed for environment mapping on operator's interface. Direct attention of the operator is required mainly in interaction level, the doctrines of goals are preset by the operator for mission level operations. With predictive algorithms running on the system to parse operator's voice or gesture input it decides on which level should the robot run and what tasks to execute from the database of doctrines.

This level of intelligence in teleoperation is more practical to implement with increasing advances in networks and sensors, as it can be said that an evolving teleoperation system would end up with some sort of hierarchy architecture.

## 2.2 Communications

Wireless networks are the most commonly used communication channels in mobile robotics, they fall into 'wireless robotics' category [26]. A communication channel can be made up of radio waves, bluetooth, RFID, Infrared or some other wavelength of electro-magnetic spectrum [27]. It is the standard protocols that gives value to the popular use of wireless technologies. The standard OSI model is shown in Figure 2.8 its abstraction can be found in every type of communication functions. Standard protocols which are used commonly like IPV4/6, IEEE 802.XX, HTTP, DNS, SSH, 2G/3G/LTE and others are projected on an OSI

model in Figure 2.9. This brings clarity in the wide range of choices available for usage in networking devices.

In one of the earliest World Wide Web network teleoperation, [28] used HTTP protocol to operate an industrial robot arm with the help of a CCD camera. Coming to the present day the networks have grown complex and protocols have diversified, coming up with higher order standards for wireless robotics have become a necessity [26], there are several applications where teleoperation is being actively used [2, 19, 15]. From the usage popularity in wireless robotics, the discussions are drawn into two groups, TCP/UDP based and IEEE 802 family based communications. One more area worth mentioning is cloud robotics which mainly uses 2G/3G/LTE cellular networks and M2M (Machine-to-Machine) communication systems [29], this is gaining popularity because of the availability of huge mobile phone infrastructure.

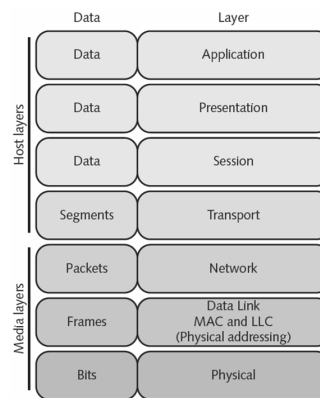


Figure 2.8: Standard OSI model

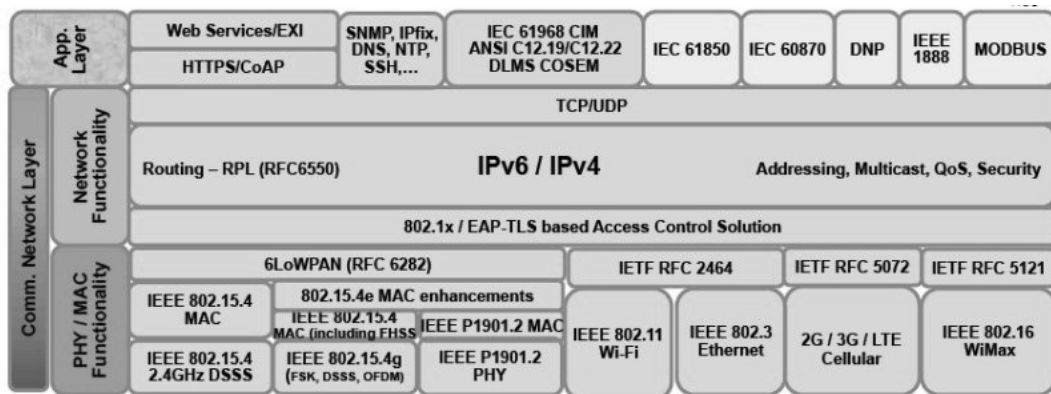


Figure 2.9: Standard protocols distributed on OSI model <sup>1</sup>

<sup>1</sup>The Internet of Every Thing - steps toward sustainability Keynote, China Computer Federation Technical Committee on Sensor Network (CWSN 2011), Sept. 26, 2011 <http://www.cs.berkeley.edu/culler/talks.html>

### 2.2.1 TCP/UDP

This is the most commonly used protocol for daily information digest everywhere, this is the reason why its the first preference in many teleoperation systems [30, 4, 31, 3, 32, 11]. As seen in the OSI stack diagram of Figure 2.9, to form a TCP/UDP data packet it needs to include headers from several layers and such a design makes this protocol reliable throughout the OSI layers. Any teleoperation system using this would benefit immensely from the huge internet infrastructure that we already have.

From studies done in [31] on a path following control loop, a wireless device was used inside that loop between the operator and robot, the performance of the system was observed with a 5 % and 50 % packet drop. This was studied against the same control loop running locally on the robot. They used UDP protocol and in their implementation every loss of packet occurred on operator and robot side was replaced by previous instance. The green mark is local controller and red is the remote controller, Figure 2.10a shows paths to be almost close but Figure 2.10b with 50 % packet drop shows the robot to be off course.

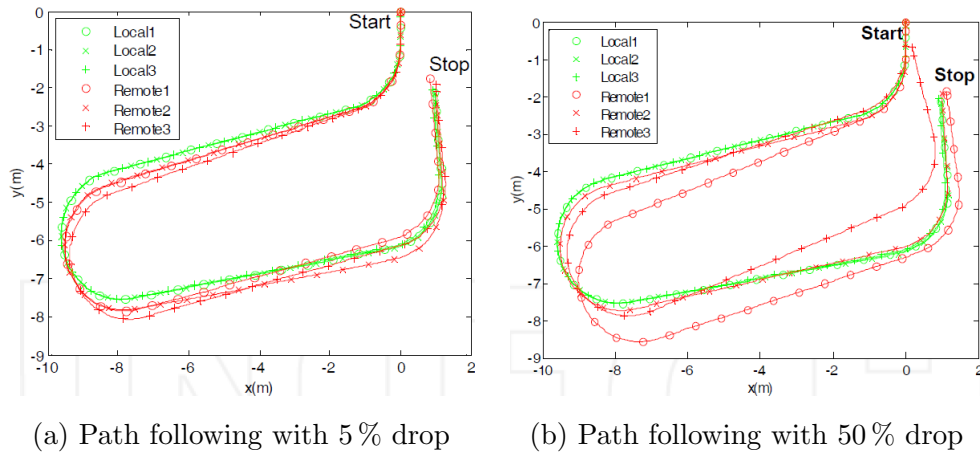


Figure 2.10: Comparing local control loop with remote control loop [3]

In [3] a remote manipulator was used to draw with the UDP protocol. It is a system with image feedback so they had to take care of compression of the image to fit in a single data packet. The operator was acting as a server while the client was the manipulator. The network performance of teleoperation show that the total time it took to completely draw a quadrifolium was 4'10" without visual feedback and 7'49" with visual feedback. The increase in time was mainly due to image processing.

TCP and UDP are not the perfect solution for usage in teleoperation. TCP with its current protocol design is meant for full duplex transmissions, auto error handling, packet re-ordering and delivery guaranteeing. This is not ideal for a real-time scenario like teleoperation mainly because of the unpredictable time of a TCP packet arrival. UDP on the other hand cannot guarantee a

proper packet delivery though the overall network time delay might be low. Some work is being done in network related predictive/compensative methods based on network models to increase the real-time usability of these protocols [33]. Other efforts are underway to make IP based protocols for applications that require real-time solutions like IOT(Internet Of Things) and WSN(Wireless Sensor Networks) [32].

### 2.2.2 IEEE 802 family

The protocols like IEEE 802.11 family, IEEE 802.15.4 etc. are designed to work with physical and MAC layer of OSI model. They are meant for low-power and low-range devices which is ideal for real-time short range communications between two nodes. Its use in teleoperation is as popular as TCP/UDP because of low power consumption, low cost of nodes and ease of use. The usages are found in the works of [9, 34, 27, 1].

The abilities of these devices can be increased many folds by having a network of such nodes. They are used for robot-robot, robot-base station, robot component-robot component communications in networks like WSN [35, 36] and Network Control Systems(NCS) [37]. This type of architecture closely resembles that of teleoperation if sensor, robot and operator are considered to be a node then communication between them would result in NCS. This area is being heavily researched to create better protocols and models for reliable, efficient and fault tolerant networks [38].

## 2.3 Control interface

The operator's perspective of the robot environment is the key characteristic of a good teleoperation system. The way robot's sensor information is presented, the ease of sending commands, information update delays, robot's orientation and communication status [39] are all the measures that amount to successful operator-robot integration. Number of sensors and their types are critical to presenting robot's environment as accurately as possible. A poor interface design is described in [7] as "resulting in spatial disorientation, attentional bottlenecks, lack of situational awareness, confusion and frustration". In designing any interface, the key elements are GUI, sensors and ease of commanding, a good interface would integrate these three components very tightly. From operator's point of view GUI provides visual information of robot's environment which is generated by sensors on-board robot and in order for the operator to steer the robot he needs to be equipped with a good commanding device.

### 2.3.1 GUI and input device

In [40] a 6 DOF robot arm was teleoperated using a ball-shaped external device, the purpose was to input commands to some of the robot's autonomous tasks. The parameters of joints were being shown in a simple GUI. In [28] a camera was attached as end-effector of a robot arm and operator was clicking at a point on the image to move the camera to the new coordinate. These two ways are the basis for other interfaces, GUI can be seen as a command input element and information display element or just as input element and external devices such as keyboard, joystick, master manipulator etc. to input commands. A hand held joystick is commonly used to send direct motion commands to the robot [4, 1, 27, 9].

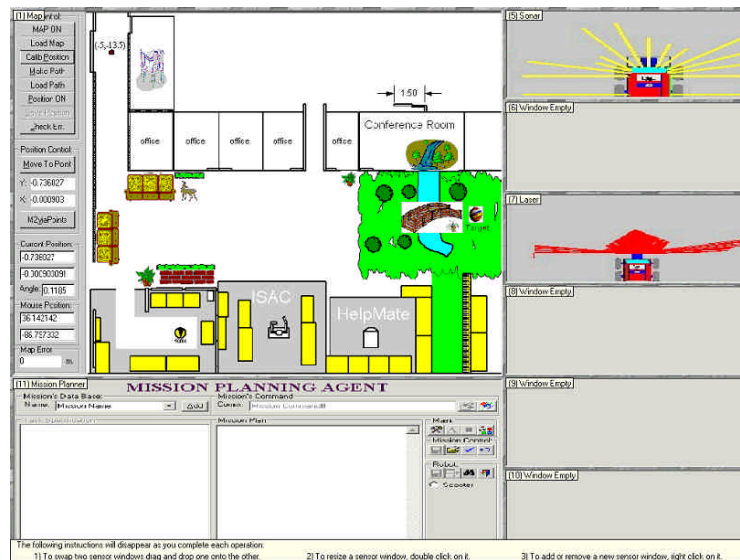


Figure 2.11: Cockpit interface [7]

In the study conducted by [7] among three different types of graphical user interfaces, operators find the cockpit style interface to be more intriguing and simple. As shown in Figure 2.11 the UI occupies full screen to make maximum use of screen real estate. Its well designed with all the critical information like robot's position, 2D map with marked objects in different colors, graphical representation of sensory information and lot of optional buttons and text editor for input. This interface was preferred by 100% of the experiment participants as their first choice.

### 2.3.2 Camera views

**Example 1** In the experiment conducted in [8] placement of different types of cameras on the robot were studied from the operator's usability point of view. Five different views were presented to an operator's monitor as shown in Figure



2.12; an ordinary camera with robot's front view, an omni-directional camera with 360° surround view, a 1 m long pole mounted camera with circular fisheye lens with wide field of view, a panoramic view from the omni-directional camera and a camera mounted outside overlooking the testing area (direct view). The test subjects rated the views in the following order: fish-eye view, omni view, direct view, panoramid view and ordinary camera view. Clearly the more information an operator has of the robot's environment the better is the commanding.

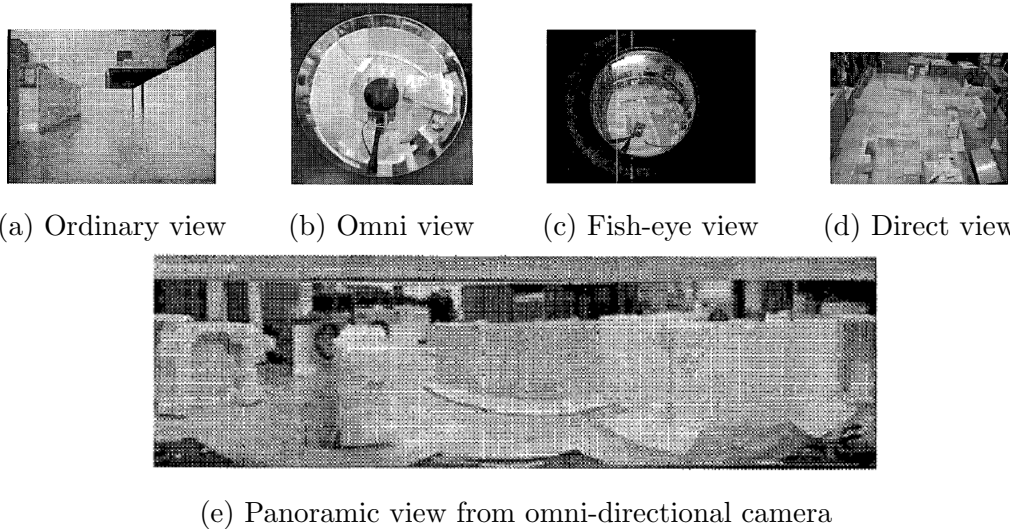


Figure 2.12: Different camera view for robot environment [8]

**Example 2** Operator performance assessment was conducted in [9] on a simulated rescue mission scenario. Three camera view points were presented to each operator and was asked to navigate through a random maze to rescue survivors. A camera view in front of the robot, a view from the camera suspended on a pole and an autonomous quadcopter programmed to follow the robot (Figure 2.13). As shown in Figure 2.14 the pole mounted camera leads to all performance criteria. On further analysis of quadcopter guided navigation time it can be seen how an operator senses the environment, clearly the pole mounted camera is in more sync with operator actions than an external camera.

### 2.3.3 Time followers vision

One of the innovative interface solution is presented in work done by [10], a simple camera and 3D sensor were used to store all the data with timestamps and robot's positions in a database when the operator was driving around. Their system calculates the robot's current position and evaluates data from previous instances to project the relevant images and 3D data on the operators interface. Considering the current location of the robot, camera orientation and field of

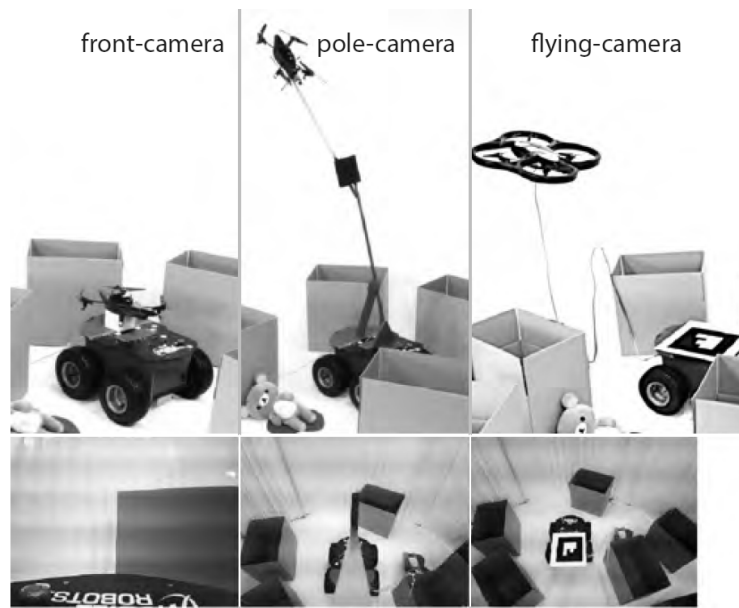


Figure 2.13: Three camera view points for rescue mission [9]

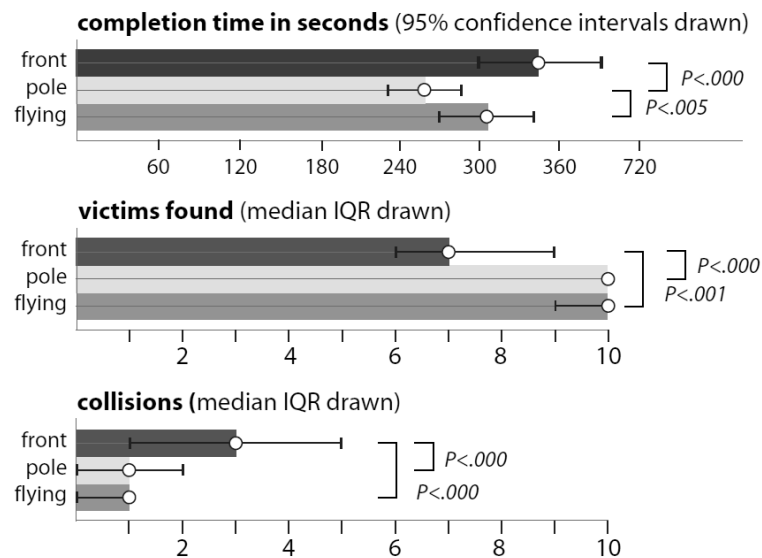


Figure 2.14: Operator performance with three different views [9]

view, a full 3D model of the robot is put in the view thus, giving a perspective that robot is being driven with the aid of external sensor. Figure 2.15 shows how the interface looks like from operator's perspective. The full block diagram of the system is shown in Figure 2.16.

The operator can comfortably be immersed in the environment with this method but the field of view could be one of the limitations. Though this innovative solution makes use of as much sensory information as possible from the database, the dynamic environment with movable objects might be very challenging for the system to make calculations on previous sensor data relative to current sensor data.



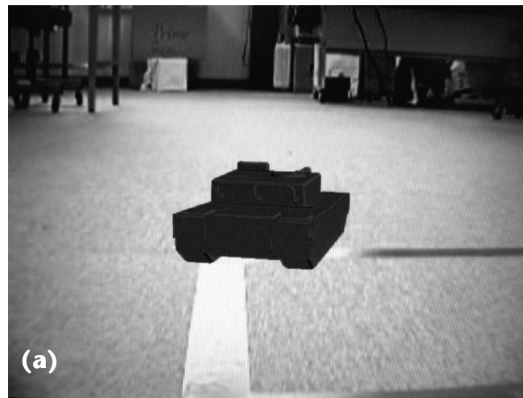


Figure 2.15: Operator interface with a robot model [10]

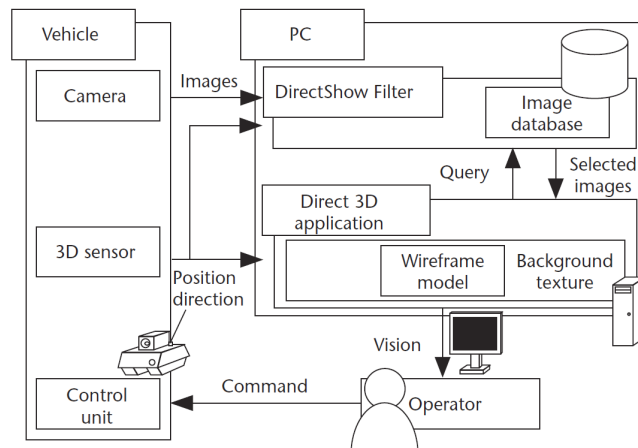


Figure 2.16: Block diagram of the system [10]

### 2.3.4 Operator command strategy

In the study done by [11] the end result of operators performance was the balance between productivity and performance of teleoperation system. It was the combination of sensor feedback and command strategy. The operator was using a master device (mechanical arm) with change in position of the device either robot speed values are generated or robot's position values are generated, they call it 'position-speed' command and 'position-position' command. Position-speed can drive the robot large distances but not very accurate reaching a target position while position-position can generate small robot movements for precision driving. They have visual feedback of robot parameters and camera views and force feedback on the master device that is directly proportional to robot-obstacle distance. The conclusion was that speed command demonstrated the highest productivity while combined command strategy had more accuracy as shown in Figure 2.17. When force feedback was used the operator was much more sure of driving the robot with high speed since he can feel the approaching obstacle and switch the command strategy.

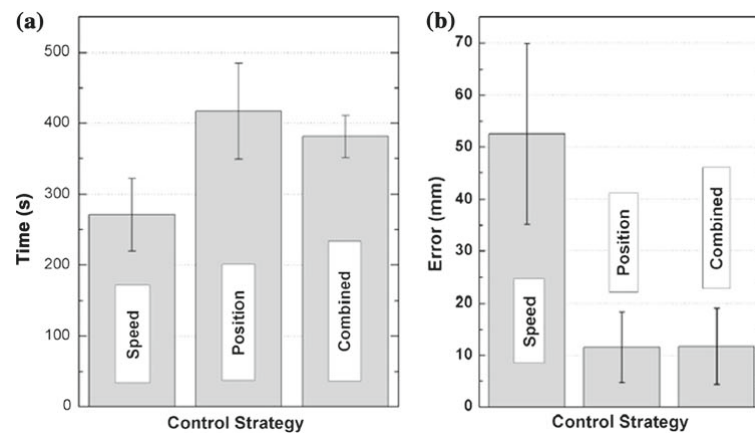


Figure 2.17: (a) Navigation time and (b) position error for command strategies [11]

## 2.4 Summary

In summary, sufficient models and types were discussed in each topic to provide a clear comparison among them. The control architecture choice is purely application dependent but for scalability and maintenance best methods must be observed. Having a level based architectural design in mind throughout the lifetime of teleoperation system is recommended.

Several interface related implementations are discussed from the operator point of view, they have been statistically ranked. The basic idea of having enough sensory information in a very simple and understandable graphics is valid in all the teleoperation systems. The choice of an input device for operator commands can be of operator's choice.

Communications are a very important part of teleoperation and choice of which is not a simple decision. In a simple operator-robot scenario depending on the distance between them, a Wireless serial node or WLAN can be chosen. The challenge for this thesis work is to choose appropriate components design and implement a full teleoperation system that is reliable and scalable.

# Chapter 3

## Trajectory Tracking

An algorithm was designed to automate the robot's motion on a given path hence a trajectory controller was implemented, this chapter describes the theory behind it. In order to implement a closed loop controller for a system its kinematic and/or dynamic model has to be established and some level of feedback gains have to be calculated, a control diagram depicted in Figure 3.1 has been implemented in the following chapter 5.

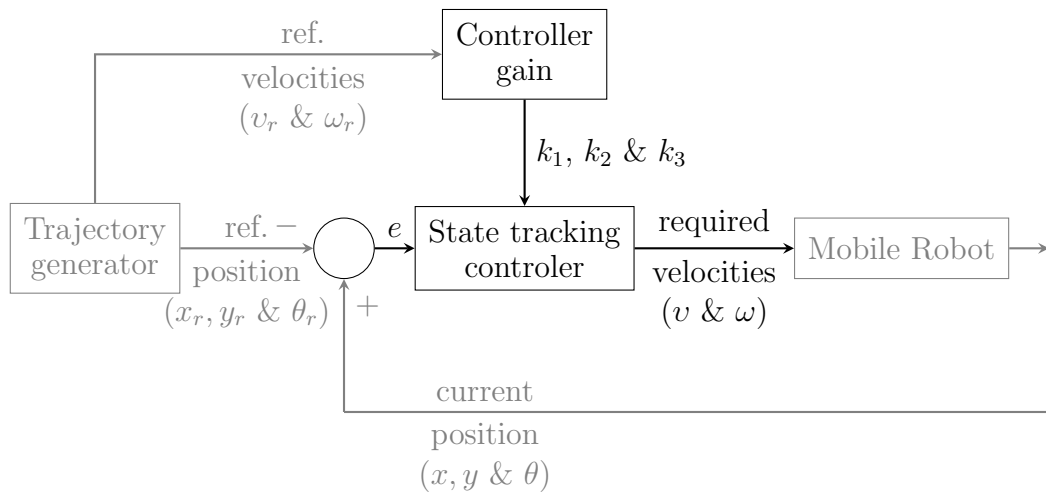


Figure 3.1: Trajectory tracking control diagram

The controller used here is called *Nonlinear State Tracking Controller* from [41] which is based on a kinematic model of differential wheel drive system. The robot system used in this project is explained in Section 4.1 and it falls in the category of non-holonomic drive systems where controllable degrees of freedom ( $v$  linear and  $\omega$  angular velocities) is less than system's total degrees of freedom ( $x, y$  position and  $\theta$  orientation). A comparison of this controller with other commonly used controllers is discussed in [42], where a better performing

*Lyapunov-based Guidance Control* theory in terms of taking less time to track a path is proposed. In the current work, the objectives do not include fast tracking hence *Nonlinear State Tracking Controller* is chosen as its linear and angular velocity tracking errors minimise better in comparison to others, it is also relatively less complex implementing with C++ language. Other benefit is that the proportional gains are calculated from reference velocities set by the trajectory generator thus, giving a dynamic behaviour.

[42] also discusses some other controller implementations based on fuzzy logics and adaptive methods. Some have also used dynamic models based on Lagrange formulation [43], these type of systems would use parameters such as robot's mass, mass of the wheels, electric motor parameters, moment of inertia of the wheel, chassis and wheel axis. In one of the earliest versions of dynamic controllers [44] shows how a kinematic model can be combined with a torque controller. In [45] dynamic and kinematic models are included in the path planning model. In [46] an adaptive controller was designed where the control architecture includes kinematic and dynamic based controllers here, a parameter updation law is formulated where reference velocities generated by a kinematic controller and robot's current velocities are used to better estimate the orientation for the dynamic controller. [47] proposed a fuzzy logic based adaptive dynamic controller where the position measurement is the only state of the robot that is needed by the controller as input since, linear and angular velocity components are integrated into the dynamic model.

### 3.1 Nonlinear State Tracking Controller

This is designed to carry out robot motions in an **obstacle-free environment**, two types of motions can be described for any scenario: *Point-to-point* and *trajectory tracking*. Current project work addresses trajectory tracking though point-to-point can also be achieved but again the trajectory to be followed from one point to another sums up to be a pure trajectory tracking. Stabilizing the robot's position about a point is one challenge that is not addressed here as it falls beyond the thesis objectives. Now, assuming wheels roll without any slippage the basic type of non-holonomic system applicable here is of a unicycle type, where the two wheels attached by an axil are motorised thus forming a differential drive mechanism while a third one is a freely moving caster wheel which acts as support, as shown in Figure 3.2 in a top view.  $v$  is transational velocity,  $\omega$  is rotational velocity,  $x$ ,  $y$  and  $\theta$  are robot's configuration and  $P$  is geometric center axis about which it can turn in its own radius. This shape of the robot is exactly scaled version of the actual robot Pioneer 3-Dx.

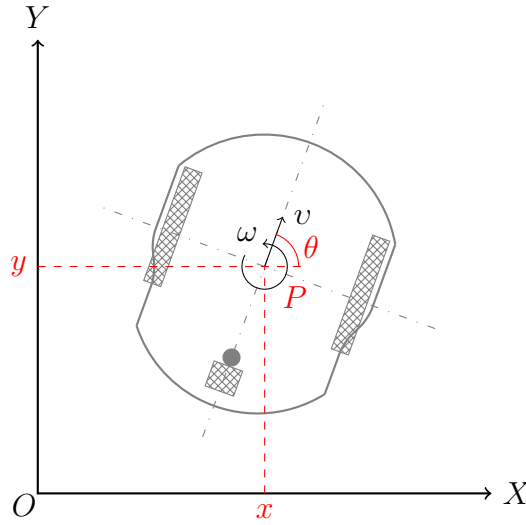


Figure 3.2: Robot motion in global reference frame

### WMR model

The robot configuration in a 2D space of Cartesian coordinate system ( $X - O - Y$ ) can be written as  $q = (x, y, \theta)^T$ ,  $q \in \mathbb{R}^2$  and  $\theta$  being positive in the counter-clockwise direction. The robot's motion makes its configuration a time varying quantity thus its linear velocity  $v$  expresses two components  $\dot{x}$  and  $\dot{y}$ , while angular velocity  $\omega = \dot{\theta}$ , their relation is a first order kinematic model of the robot.

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (3.1)$$

$v$  and  $\omega$  are taken as control inputs, this driftless nonlinear system applies to most of the WMR's. This type of non-holonomic system has a constraint to move the robot in lateral direction which is expressed as

$$\dot{x} \sin \theta - \dot{y} \cos \theta = 0$$

In feedback based trajectory tracking it is necessary to zero-in on required robot's configuration, let us say the error variables approaching zero is given by  $q_e = (x_e, y_e, \theta_e)^T$ . Suppose the current and reference robot configuration is represented by  $q$  and  $q_r$  respectively as shown in Figure 3.3 then line of sight errors  $x_e$  and  $y_e$  and  $\theta_e$  are related as in (3.2) where error is between the two transformation matrices.

$$q_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x - x_r \\ y - y_r \\ \theta - \theta_r \end{bmatrix} \quad (3.2)$$

## Trajectory

If  $(x_r(t), y_r(t))$  is the reference trajectory for  $t \in [0, T]$ , to drive the robot in a desired trajectory path translational, rotational velocities and orientation have to be calculated from the this reference path. They are related as follow.

Trasational velocity, + or - denotes robot direction forward or reverse respectively.

$$v_r(t) = \pm \sqrt{\dot{x}_r^2(t) + \dot{y}_r^2(t)} \quad (3.3)$$

Angle at each  $(x_r, y_r)$  is,  $k$  being 0 or 1 for forward or reverse direction respectively

$$\theta_r(t) = \arctan 2(\dot{y}_r(t), \dot{x}_r(t)) + k\pi \quad (3.4)$$

As per  $\dot{\theta}$  of (3.1) time derivative of (3.4) gives rotational velocity as

$$\omega_r(t) = \frac{\dot{x}_r(t)\ddot{y}_r(t) - \dot{y}_r(t)\ddot{x}_r(t)}{\dot{x}_r^2(t) + \dot{y}_r^2(t)} \quad (3.5)$$

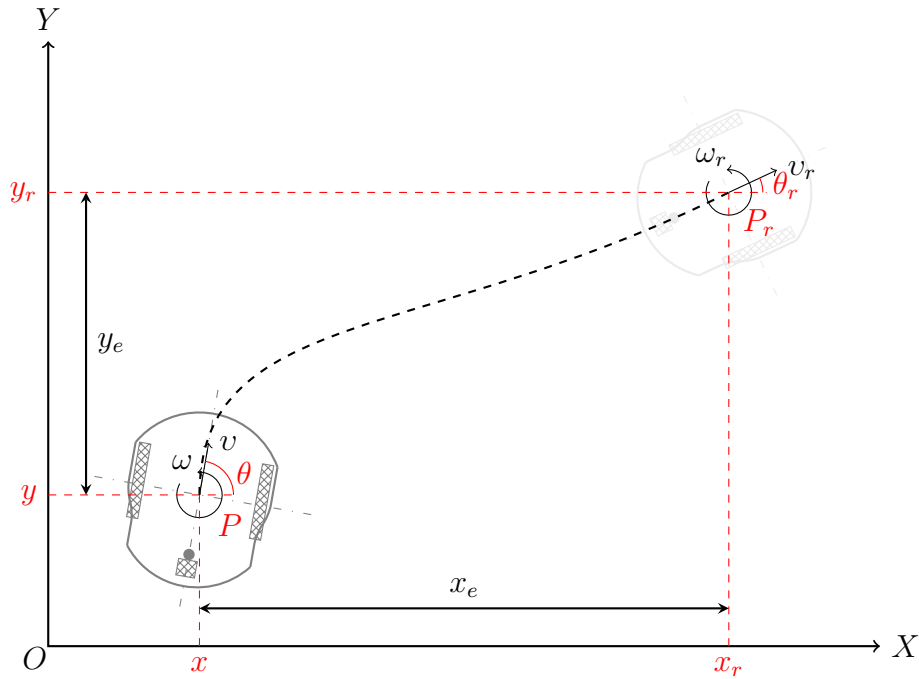


Figure 3.3: Trajectory tracking of robot state

## Controller

A moving robot trying to follow a path has to always zero in on position error as expressed in (3.2) hence a controller would make sure that this zero check is valid

$\forall t \in [0, T]$  forcing the robot to stay on track. To calculate the controller model a derivative of (3.2) is made considering equation (3.1) as  $\dot{x}_r \sin \theta_r = \dot{y}_r \cos \theta_r$  the result is eq. (3.6), a non-linear dynamic model.

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} \cos \theta_e & 0 \\ \sin \theta_e & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} + \begin{bmatrix} -1 & y_e \\ 0 & -x_e \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} v_r \cos \theta_e - v \\ \omega_r - \omega \end{bmatrix} \quad (3.6)$$

Lets define a matrix  $\begin{bmatrix} u_v \\ u_\omega \end{bmatrix} = \begin{bmatrix} v_r \cos \theta_e - v \\ \omega_r - \omega \end{bmatrix}$ , rewriting the eq. (3.6) in terms of closed-loop inputs  $q_e$ ,  $v_r$ ,  $u_v$  and  $u_\omega$

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} + \begin{bmatrix} 0 \\ \sin \theta_e \\ 0 \end{bmatrix} \cdot v_r + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_v \\ u_\omega \end{bmatrix} \quad (3.7)$$

Linearizing about equilibrium point i.e.  $x_e = y_e = \theta_e = 0$  and  $u_v = u_\omega = 0$  we obtain.

$$\Delta \dot{\mathbf{q}}_e = \begin{bmatrix} 0 & \omega_r & 0 \\ -\omega_r & 0 & v_r \\ 0 & 0 & 0 \end{bmatrix} \cdot \Delta \mathbf{q}_e + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \Delta \mathbf{v} \quad (3.8)$$

For a valid controllability  $v_r \neq 0$  and  $\omega_r \neq 0$ , the above equation is of the form  $\Delta \dot{\mathbf{q}} = \mathbf{A} \cdot \Delta \mathbf{q} + \mathbf{B} \cdot \Delta \mathbf{u}$  with full rank  $rank(\mathbf{B}, \mathbf{A}\mathbf{B}, \mathbf{A}^2\mathbf{B}) = 3$ . Defining the full state space controller with three states and two inputs as

$$\mathbf{v} = \mathbf{K} \cdot \mathbf{q}_e \quad (3.9)$$

From 3.3 a controller structure can be established, it can be visualized that transational velocity of the robot influences error in  $x_e$  to compensate error in  $y_e$  and  $\theta_e$  its simply possible to control rotational velocity, from this discussion a controller would be

$$\begin{bmatrix} u_v \\ u_\omega \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} & \mathbf{K}_{13} \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \mathbf{K}_{23} \end{bmatrix} \cdot \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} \quad (3.10)$$

$$\begin{aligned} u_v &= -\mathbf{K}_{11}x_e \\ u_\omega &= -\mathbf{K}_{22}sign(v_r) - \mathbf{K}_{23}\theta_e \end{aligned} \quad (3.11)$$

Renaming controller gains from  $\mathbf{K}_{11}$ ,  $\mathbf{K}_{22}$  and  $\mathbf{K}_{23}$  to  $k_1$ ,  $k_2$  and  $k_3$  respectively. Comparing the real and desired polynomials of above equations would yield controller gains. It takes the form of

$$(s + 2\xi\omega_n)(s^2 + 2\xi\omega_n s + \omega_n^2) \quad (3.12)$$

For this second order equation  $\xi \in (0, 1)$  and  $\omega_n > 0$ , now writing the polynomial form of (3.11)

$$s^3 + (k_1 + k_3)s^2 + (k_1k_3 + k_2v_r + \omega^2)s + k_1k_2qv + k_3\omega^2 \quad (3.13)$$

Comparing equations (3.13) with (3.12) we get,

$$\begin{aligned} k_1 + k_3 &= 4\xi\omega_n \\ k_1k_3 + k_2v_r + \omega^2 &= 4\xi^2\omega_n^2 + \omega_n^2 \\ k_1k_2qv + k_3\omega^2 &= 4\xi\omega_n^3 \end{aligned} \quad (3.14)$$

Solution in the suggested form of  $(u_v, u_\omega)^T$  yields

$$\begin{aligned} k_1 &= k_3 = 2\xi\omega_n \\ k_2 &= \frac{\omega_n^2 - \omega_r^2}{|v_r|} \end{aligned} \quad (3.15)$$

From [48] when  $v_r$  goes to 0  $k_2$  gain becomes  $\infty$  which is not of any practical use in case of zero velocities hence that is modified as  $k_2 = b.|v_r|$  Now the system characteristic frequency is

$$\omega_n = \sqrt{\omega_r^2 + b.v_r^2} \quad (3.16)$$

Parameter  $g > 0$  addition makes the gain approach zero when robot is no longer in motion controllability is lost. Finally from [41] the nonlinear tracking controller based on Lyapunov function also can be referred to as Samson's controller is given by

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v_r \cos \theta_e + k_1 x_e \\ \omega_r + k_2 v_r \frac{\sin(\theta_e)}{\theta_e} y_e + k_3 \theta_e \end{bmatrix} \quad (3.17)$$

where  $k_1$ ,  $k_2$  and  $k_3$  from (3.15) and (3.16) are

$$\begin{aligned} k_1 &= 2\xi\sqrt{\omega_r^2 + b.v_r^2} \\ k_2 &= b.|v_r| \\ k_3 &= 2\xi\sqrt{\omega_r^2 + b.v_r^2} \end{aligned} \quad (3.18)$$



# Chapter 4

## Tools and Framework

In this chapter, a description of the required software framework and API's and hardware devices are described in detail. The Figure 4.1 shows the setup of the whole system. The wheeled mobile robot has an on-board computer which has a dedicated serial communication link with the on-board microcontroller unit, the communication node is connected via USB to the computer. On the remote controller side similar computer connection to network node is made. The interface used by the wireless communication node device is IEEE 802.15.4 digital radio with integrated MAC hardware.

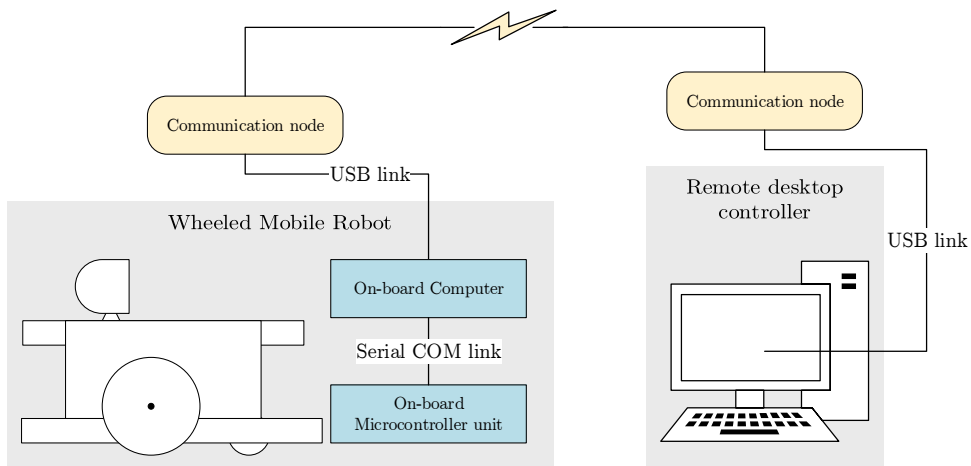


Figure 4.1: Teleoperation setup

## 4.1 WMR: Pioneer 3-DX

A mobile robot is a machine which has the ability to perform independent tasks with its own intelligence or can be monitored and controlled from a remote location, this capability to operate remotely is possible with wireless communications technologies. A WMR with two differential wheels and a free rotating balancing wheel is called Differential wheeled robot with a tricycle configuration, its designed in such a way that it can swing about a vertical axis which includes all the robot's physical body inside that imaginary circle. This robot comes with a free C++ Software Development Kit (SDK) and some proprietary packages for localization. Figure 4.2 shows the robot's front and back view with all its visible sensors.

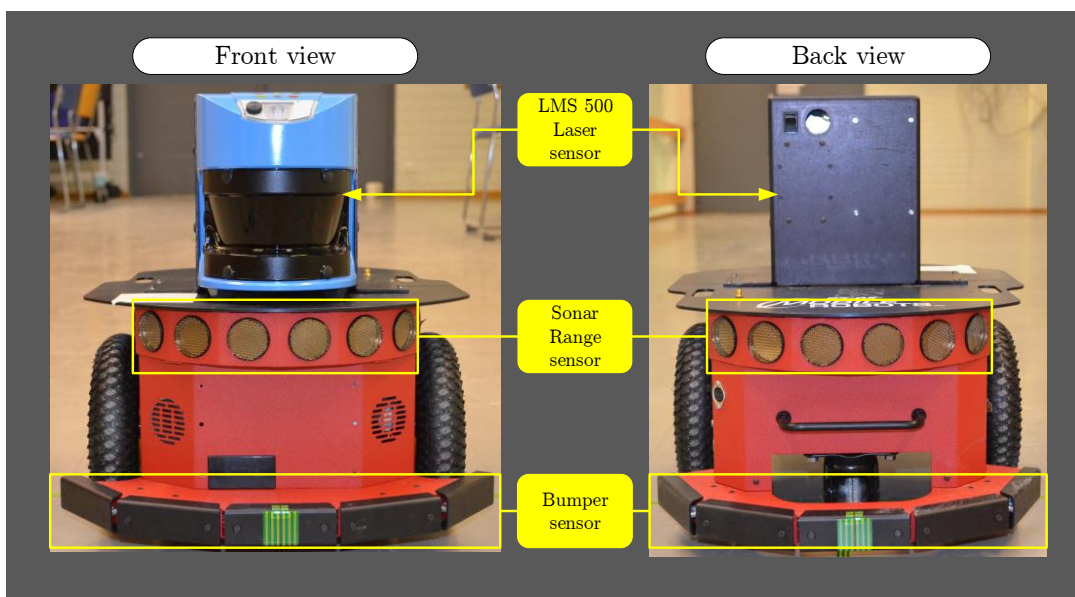


Figure 4.2: Pioneer 3-DX robot

### 4.1.1 Robot Hardware

The robot platform consists of two wheel differential drive system, motor control and drive electronics, reversible DC motors, motion encoders and a battery unit, all these are managed by an on-board 32-bit Renesas SH2-7144 RISC microprocessor including the P3-SH microcontroller with ActivMedia Robotics Operating System (AROS) software. The features of the microprocessor and microcontrollers are mentioned in [12, Page 4]. This onboard microcontroller unit handles all the low level tasks of the robot like maintaining motor drive speeds and heading, collecting data from sonar sensors and bumpers. For the higher intelligent task an onboard Windows 7 based computer is used which is serially (RS232) connected to the microcontroller unit thus, forming a client-server communication link. It is possible to tweak the sensors' precision and

encoders' precision, this hardware is also flexible enough to supply electrical power to external devices like camera, manipulator or other sensors. A Sick LMS 500-2100 laser range finder is used for high precision and long range localization of the robot in a given 2D space, this laser sensor's interface is Ethernet based and directly connected to the on-board computer for acquiring readings.

The specifications are as follow [12]:

1. Robot dead weight is around 9 Kg with one lead acid battery and it can accommodate a maximum of three batteries, the robot chassis is built of aluminium.
2. Physical characteristics:
  - Payload capacity is 23 Kg
  - height : 23.7 cm + 18.5 cm laser range sensor
  - width : 39.3 cm
  - length : 44.5 cm
3. Several parts form the robot assembly includes deck, motor stop button, user control panel, body-nose-and-accessory panel, sonar arrays, motors-wheels-and-encoders and batteries-and-power
4. DC power
  - 3 Warker WKA12-9F2 lead acid batteries with 12V and 9Ah each
  - Battery runtime with PC 3-4 hours
  - Charging time is 6 hours per battery
5. Mobility
  - Two foam filled wheels with diameter of 19.53 cm and width of 4.74 cm
  - Differential steering with swing radius of 26.7 cm
  - Maximum rated translational speed is 1.4 m/s and rotational speed is 300 deg/sec
6. Sensors
  - Sonar : 8 in front and 8 in the back
  - Laser : scanning 181 degrees with 1 degree resolution, data transmission rate of 100Mbps and scanning frequency of 75 Hz
  - Bumpers : 5 in front and 5 in the back
  - Encoders : 33,500 counts per rotation

### 4.1.2 Robot Software

#### ARCOS

Microcontroller level **ARCOS** firmware acts as a server accepting command packets from client and sending **SIP** over the **HOST** connection or external **SERIAL** port on the user control panel, in this thesis work **HOST** connection is used and is directly connected to the on-board computer where Advanced Robotics Interface for Applications (**ARIA**) library acts as a client. Figure 4.3 shows a clear control architecture for **ARCOS**. There are some firmware configuration parameters like robot's motion related parameters, physical dimensions and sensor parameters that are stored in its memory. While executing client commands it uses these parameters as references, MobileRobots Inc. provided software to make changes to these configurations if necessary.

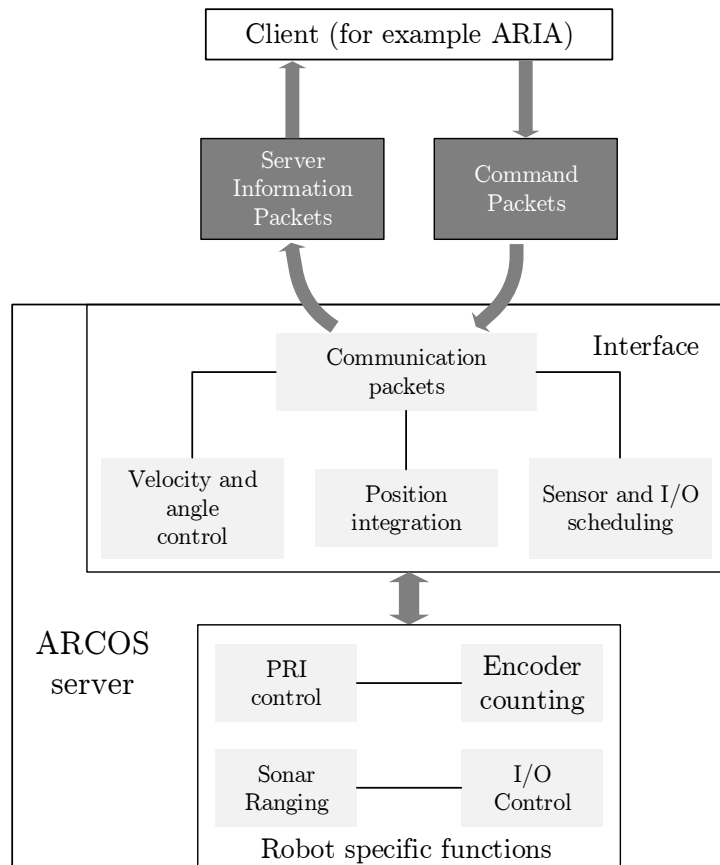


Figure 4.3: **ARCOS** server-client control architecture [12]

### Arcos server-client connection steps

There are a few basic steps for establishing and maintaining server-client connection with **ARCOS** as give in [12]:

1. Sending a series of 2 synchronisation commands to which **ARCOS** replies by echoing those packets back to the client, after sending 3rd synchronization command server replies with robot identification information containing the name of the robot, its type and sub-type.
2. After establishing the link with the server, client sends **OPEN** command to start motor and sensor servers and **ARCOS** begins transmitting **SIP**'s at regular intervals.
3. In order for **ARCOS** to know that a client is still connected and receiving **SIP**'s, it is the responsibility of the client to keep sending a **PULSE** command at an interval less than or equal to **watchDog** seconds, this is an **ARCOS**'s watchdog which anticipates at least one client command packet in that time interval.
4. To close the connection **CLOSE** command is used.

Some useful contents of an **SIP** packet are:

	Label	Description
Robot state	XPOS	Wheel encoder integrated coordinates in mm
	YPOS	
	THPOS	Orientation in degrees
Robot velocities	L VEL	Wheel velocities in mm/s L-Left and R-Right wheel
	R VEL	
	SONAR COUNT	Number of new sonar reading included in <b>SIP</b>
	NUMBER	Sonar disc number (Only for Sonar count > 0)
	RANGE	Corresponding sonar range value in mm
	.....	rest of the sonar disc numbers and readings
	.....	
	BATTERY	Battery charge in tenths of volts

Table 4.1: Useful **SIP** contents

## ARIA

In order to develop an application that would use **ARCOS**, MobileRobots has provided with Software Development Kit called **ARIA** which is an object oriented C++ library API (Application Programming Interface) used for running programs on the robot's on-board computer. This API comes with all the necessary classes like server-client communication with **ARCOS**, sensor device interface classes, motion command classes, callback classes, utility classes and range of other useful classes. There is also **Advanced Robotics Navigation and Localization (ARNL)** and **Sonar based Advanced Robotics Navigation and Localization (SONARNL)** proprieter library in C++ that comes with purchase of the sick laser, in this project they are used for localization of the robot in a given 2D space, these libraries are built to work seamlessly with the **ARIA** library. There is a parameter file which includes motion control, robot dimensions and sensor related parameters which are used as references, **ArRobot** has functions which can be used to change those parameters during run time.

### Useful C++ classes from API:

**ArRobotConnector**: This class is the one that executes the **ARCOS**es server-client connection procedure as mentioned in paragraph 4.1.2

**ArRobot**: There is the main class of **ARIA** called **ArRobot** which runs in a cycle of 100 ms (can be changed with `ArRobot::setCycleTime`) time period, as shown in Figure 4.4 the **ArRobot** task cycle handles all the necessary run to read the packet from **ARCOS** and send packet from cached direct motion commands. A useful feature of this class is that you can write a user task in a function of your class and add it as a functor (function pointer) to the 'call back' list this way it runs in the same cycle as **ArRobot**'s task cycle where all the latest sensor data is available for usage downside is that the motion commands if any are sent to **ARCOS** only in the next cycle. **ARIA** also has something called action classes with **ArAction** as base class, this solves the same purpose as user task classes but the robot's motion commands are sent to **ARCOS** in the same cycle. An issue with the library files of **ARIA** was causing a compiler error when subclassing **ArAction** class hence user tasks were implemented as necessary.

**ArLaserConnector**: This class connects to the SICK laser via Ethernet interface and adds it to the sensor range devices list of **ArRobot**, during the main task cycle the sensor readings are interpreted automatically.

**ArLocalizationTask**: It uses laser data and odometry data from the robot to determine robot's position and orientation in a given 2D map (preloaded **ArMap** class). This task is run asynchronously and the robot position is updated every  $\approx 30$  ms. This is implemented with Monte Carlo Localization Algorithm, the related settings are loaded from the library files of the API. The default Number of samples and the grid resolution are 2000 and 100 mm respectively.

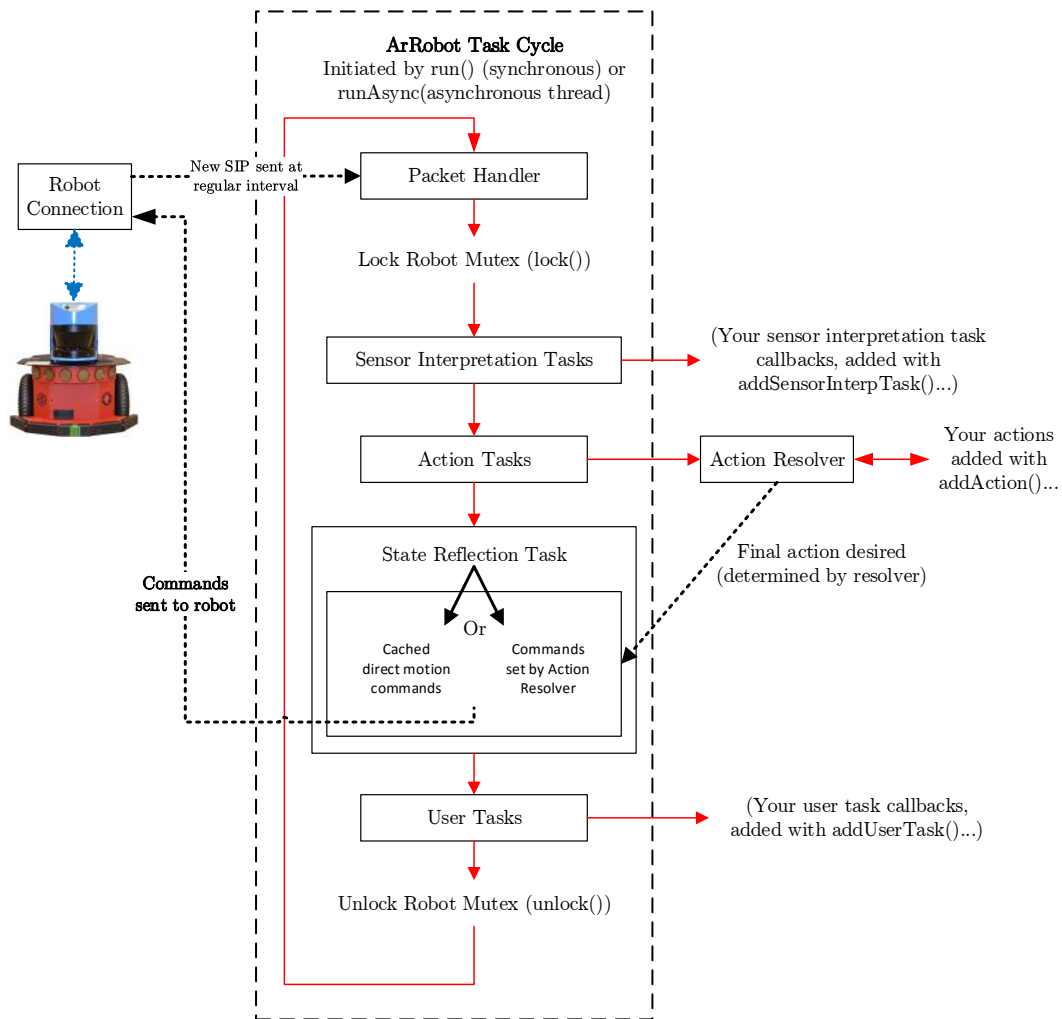


Figure 4.4: ArRobot task cycle

## 4.2 User Interface framework

**Qt5** is a cross-platform C++ application development framework with modular class libraries, in this project GUI (Graphical User Interface) libraries were extensively used for accepting user inputs through widgets and visually showing a graphical representation of robot and its track, some other useful information like robot speed and communication packet rates is also displayed. Since the libraries are meant to be compiled on different computer platforms using them would not change the feel of application if compiled for other OS/architecture this positive aspect gives a big advantage using this particular framework. The important Qt classes are discussed in this section.

```
class objectA : public QObject {
    Q_OBJECT
```

signals:

```
void signalFunction_1();
void signalFunction_2();
```

slots:

```
void slotFunction_1();
void slotFunction_2();
};
```

```
class objectB : public QObject {
    Q_OBJECT
```

signals:

```
void signalFunction_1();
void signalFunction_2();
```

slots:

```
void slotFunction_1();
void slotFunction_2();
};
```

```
connect(objectA,SIGNAL(signalFunction_1()),objectA,SLOT(slotFunction_2()));
connect(objectA,SIGNAL(signalFunction_2()),objectB,SLOT(slotFunction_1()));
connect(objectA,SIGNAL(signalFunction_2()),objectB,SLOT(slotFunction_2()));
connect(objectB,SIGNAL(signalFunction_1()),objectA,SLOT(slotFunction_1()));
```

After meta-object compiler a newly generated  
C++ source files contains

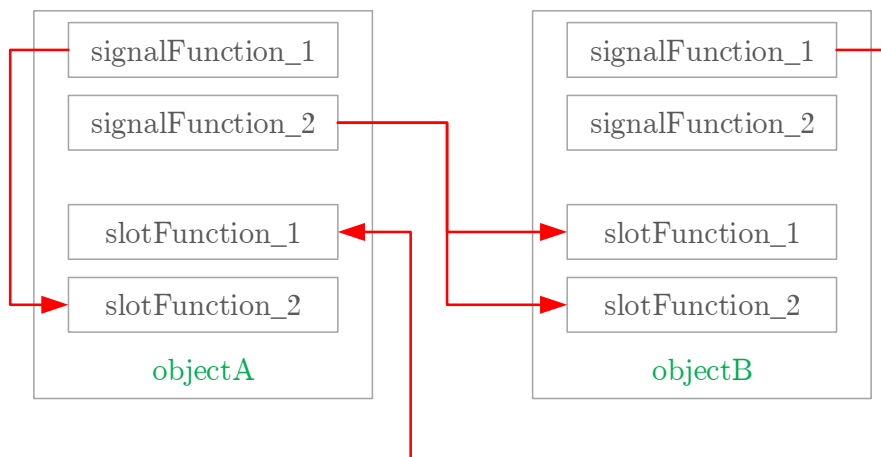


Figure 4.5: Signal and slot mechanism

## Signals and slots

The most important things to understand when using Qt is its signal and slot mechanism, this is Qt's own innovation and it is an ideal feature to implement communication between Qt objects (QObject is the base class). During the code compilation, a meta-object compiler which comes with Qt installation goes through the source code and where ever there is a macro called Q\_OBJECT in the class it automatically generates C++ source code with meta-object



information which is then later integrated into the final executable file. This meta-object information source files define links between signal and slot functions. A function `QObject::connect(const QObject* sender, const char* signal, const QObject* receiver, const char* method)` is used to make this connection. Using `emit function()` which emits the signal for `this` object's `function()`. This emitter and sender objects do not have the knowledge of each other, it is just that they should be type compatible and the receiver should accept equal or less number of function parameters the sender is passing. It is also possible to make connections between objects from different threads. Figure 4.5 illustrates the code and diagram view of the mechanism.

**QObject** This is the base class for all Qt objects, this class holds all the basic features Qt provides. Some important ones are:

- Handling events, creating the object tree automatically with children and parent object information and integrates with signals and slots mechanism
- They by default live in a single thread in which its created but it is possible to change the thread affinity just by calling `QObject::moveToThread(QThread *)`
- Auto connection of default signals and slots of its subclasses and their children are performed during runtime

**QApplication** It performs a lot of initialization before any widgets are shown on screen for GUI based Qt application, hence an instance of this has to be created before any other widgets are initialized. In order to keep the application running and enter the main event loop `QApplication::exec` is used, it only exits when the main widget generates a `QApplication::lastWindowClosed` signal. The features of this class are to:

- Initialize the application with user's desktop setting
- Perform event handling from the computer system to relevant widgets
- Set the applications native look style, cursor settings, information on all the widgets used and handles text and color space

## 4.3 Wireless Communication

The main block in teleoperation is a radio communication module, in this project custom made wireless nodes are used for operator-robot communication. This node is called UWASA node and it is developed at University of Vaasa by [Yusein Ali](#) who is currently a researcher at Department of Communications and Networking of Aalto University. The node's design objective is to fulfil the needs of a reliable wireless sensor networking application hence, this node is made to be modular and stackable with slave boards as per the application needs. The radio module of the node uses IEEE 802.15.4 protocol and is linked with Omni-directional antenna. Some of its features are:

Interface	USB 2.0
Max baud rate	921600
Range	100 - 300 m
LED indicators	packet transmitting, packet receiving, power on, battery power
Power supply	Li-ion battery or USB power
Size	4.5 cm(L) 5 cm(B) 2 cm(H)



Figure 4.6: Communication node

For a user, it provides [49] :

1. Stackable process power and memory option.
2. Low power consumption with addable external power supply.
3. Use of the microcontroller unit with a power management circuit and a wireless communication circuits.
4. The option to stack more slave boards to meet the application needs.
5. Overall small size (45.6 mm X 45.6 mm).

## Hardware

The basic stack was used for this project which consists of the *Main module* and the *Power module* as in Figure 4.7. The data transfer to/from the device is made through USB connector.

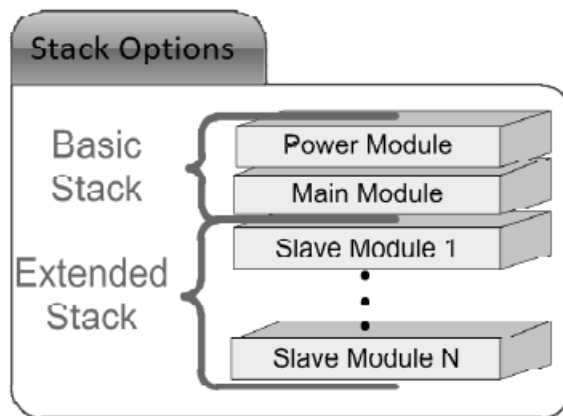


Figure 4.7: Hardware stack

### Main module

This is the master module and its different sub-modules present inside are:

- Wireless Transceiver/ RF controller
- Main controller to manage in-node data processing, performing data processing and decision making and
- Hardware stack controller

## Power module

Major purpose of this module is to fulfil the power requirements of other modules, its different sub-modules are:

- Power source and regulation, it can use power from USB or can be boosted with external Li-ion battery
- Dynamic power path management
- Battery fuel gauge
- Battery charger to charge an external Li-ion battery if connected.
- Hardware stack controller

## Software

To make use of full hardware features a custom middleware abstraction API's (Application Programming Interface) were developed on top of FreeRTOS (Real Time Operating System). These API's give an application on the module to easily run parallel processes for different purposes like for e.g. Power management and Time synchronization. On the external computer device driver libraries are installed, these are a set of C libraries with function for buffer storage, writing/reading stacks of buffer to/from the communication module.

The functions used from this library are:

- Initialize the serial port with baud rate and port number and initialize the stack of buffers to store packets.
- Use the previously initialised buffer to read and write data packets.
- Use the previously initialised 'serial port' handler to send and receive the buffer.

# Chapter 5

## Implementation

The details of implementation of software and its components are discussed in this chapter. As per discussion in the Chapter 2 a behaviour based architecture model is chosen, the interface is designed in such a way that all the sensory information is available to the operator in a graphical way and settings are grouped for easy access.

Designing a communication data packet is one of the important tasks as it needs to fully make use of wireless device capability and deliver maximum packet size. Packets from the robot and controller side are transmitted in regular interval, this design is to give teleoperation maximum reliability in usage. Communication nodes are programmed to be half-duplex.

File storage functions have been implemented to store all the experimental data for further analysis. All data packets transmitted and received are stored with time stamps, this way its easy to simulate data. Two time stamps are stored for each entry into the file, one came from the data packet (transmission time) and the other is local time on packet reception.

### 5.1 Communication data packet

Communication nodes are used as a serial device with a baud rate of 921600. To have a proper exchange of data, a standard model of packet protocol has to be used which contains a header, payload data and a trailer. Header and trailer are common for all of the packets while Payload data varies based on the type of packet being assembled. A packet size varies between 11 bytes and 106 bytes, its structure is shown in table 5.1.

The CRC(Cyclic Redundancy Check) is a polynomial based error-checking code very useful for validation of the data packet on the receiving node. Here an implementation of 16bit CRC is used<sup>1</sup>. Two functions were coded, one to verify CRC and another to append CRC into the trailer. Any data packet that isn't correct is usually discarded from further parsing.

The header, trailer and payload are made up of a single data structure (struct in C language) to get easy access to each element. Each table of data structures is shown in 5.1, 5.2, 5.6, 5.5, 5.3 and 5.4 with their actual element names on the left side. There are also packet creation functions for each type of packet and on the receiving side a parser uses the 'sub\_type' element from header to decide what data to parse.

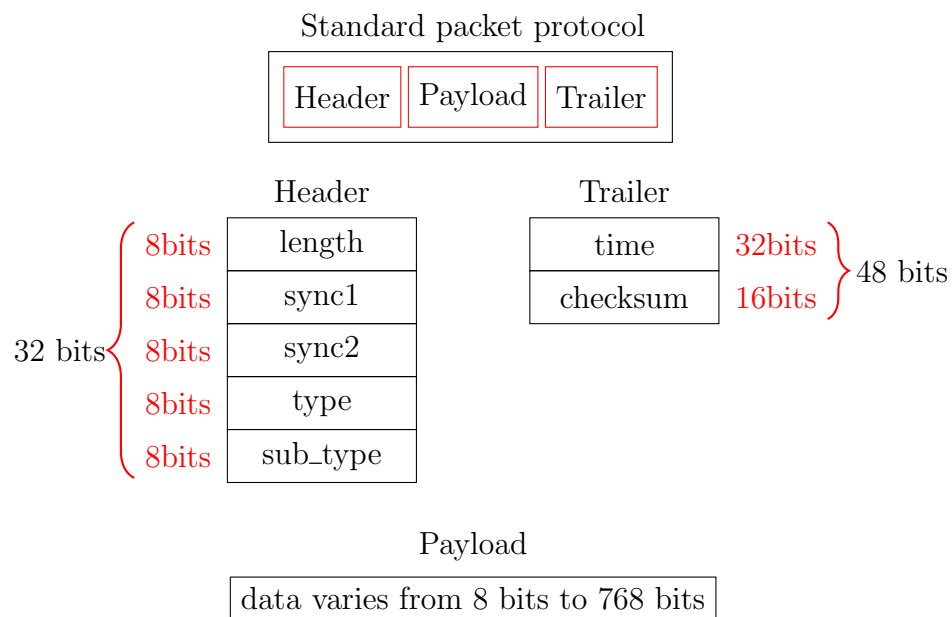


Figure 5.1: Structure of a standard packet with its size

The Table 5.1 shows the size of each of the elements present in Header and Trailer part of a standard packet.

### 5.1.1 Payload Data

The payload data consists of one type of Position information (table 5.6) from packets send by **Robot** application and four types (table 5.1, 5.2, 5.5, 5.3 and 5.4) of command information from packets send by **Controller** application.

<sup>1</sup>CRC-CCITT <http://srecord.sourceforge.net/crc16-ccitt.html>

<b>Header</b>	
length:	The total length of the packet in bytes
sync1:	First signature byte used to verify packet authenticity (0xFF)
sync2:	Second signature byte used to verify packet authenticity (0xFA)
type:	Type of packet <ul style="list-style-type: none"> <li>- position type (0xA1), used for packets sent from Robot to Controller</li> <li>- command type (0xA2), used for packets sent from Controller to Robot</li> </ul>
sub_type:	Used for command type packets. They are: <ul style="list-style-type: none"> <li>- none (0)</li> <li>- set Tx (1), to set the start of transmission on Robot</li> <li>- set Tx spacing (2), to set the transmission interval on Robot</li> <li>- set velocity (3), to set left and right wheel velocities of Robot in mm/s</li> <li>- set control velocity (4), to set the translational and rotational velocities of Robot in mm/s and degrees/s respectively</li> </ul>
<b>Trailer</b>	
time:	tick count from the start of a computer in ms
checksum:	a CRC-16 based checksum to verify if the received packet is complete and original

Table 5.1: Header and Trailer elements description

<b>Payload: Control velocity command data (32 bits)</b>	
trans_vel:	set translational velocity of the Robot( $\text{mm s}^{-1}$ ) <16 bits>
rotation_vel:	set rotational velocity of the Robot( $^{\circ} \text{s}^{-1}$ ) <16 bits>

Table 5.2: Control velocity command data payload from Controller

<b>Payload: Tx interval command data (16 bits)</b>	
tx_spacing:	set packet transmission interval on the Robot (ms) <16 bits>

Table 5.3: Transmission interval command payload from Controller

<b>Payload: Velocity command data (32 bits)</b>	
left_vel:	set left wheel velocity of the Robot( $\text{mm s}^{-1}$ ) <16 bits>
right_vel:	set right wheel velocity of the Robot( $\text{mm s}^{-1}$ ) <16 bits>

Table 5.4: Velocity command data payload from Controller

<b>Payload: Set Tx command data (8 bits)</b>	
set_tx:	set/unset packet transmission on the Robot, values are 0 for unset and 1 for set <8 bits>

Table 5.5: Set transmission command payload from Controller

<b>Payload: Position data (768 bits)</b>	
pos_x:	robot position in the 2D world coordinate system in X direction (mm) <16 bits>
pos_y:	robot position in the 2D world coordinate system in Y direction (mm) <16 bits>
pos_heading:	robot heading in the 2D world coordinate system; 0 to +180 for counter-clockwise and 0 to -180 for clockwise ( $^{\circ}$ ) <16 bits>
trans_vel:	transational velocity of the Robot ( $\text{mm s}^{-1}$ ) <16 bits>
rotation_vel:	rotational velocity of the Robot( $^{\circ} \text{s}^{-1}$ ) <16 bits>
laser[35]:	array of uniformly divided 35 laser range readings out of 191 readings (mm) <560 bits>
sonar[8]:	array of eight sonar readings from the back side of the robot (mm) <128 bits>

Table 5.6: Position data payload from Robot



## 5.2 Robot application development

Software application on the robot side is divided into several elements:

1. Robot main - robot cycle thread, which handles laser localization, position update, robot safety and velocity setting.
2. Transmission - this thread handles packet forming, CRC append and packet transmission.
3. Reception - this thread handles packet reading from the buffer, CRC validation, packet parsing and operator command setting.

As explained in sub section 4.1.2 the main Robot task cycle (`ArRobot`) runs asynchronously with the localization task (`ArLocalizationTask`) in a separate thread. The program has to be started from a command window passing two argument values called `-m2mmap` and `-m2mcommpport`. Figure 5.2 shows different tasks started when the program executes.

`-m2mmap` It is the location of a 2D map the program has to use  
`-m2mcommpport` The port to which the radio communication node is connected  
 E.g. `robot.exe -m2mcommpport COM5 -m2mmap C:/User/Document/office.map`

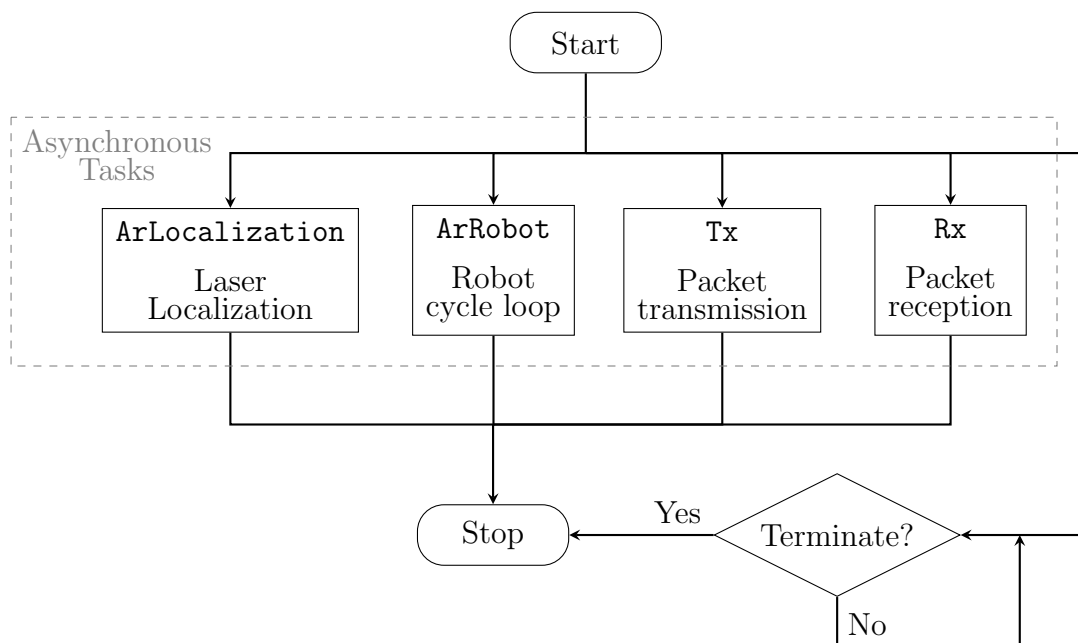


Figure 5.2: Robot application flow

### 5.2.1 ArRobot

As discussed in Figure 4.4 of the Section 4.1 a user task is added to the ArRobot's cycle where a method is called at the end of the cycle. Figure 5.3 shows the function call to `doZoneCheck()` inside which a safety algorithm is applied at every cycle.

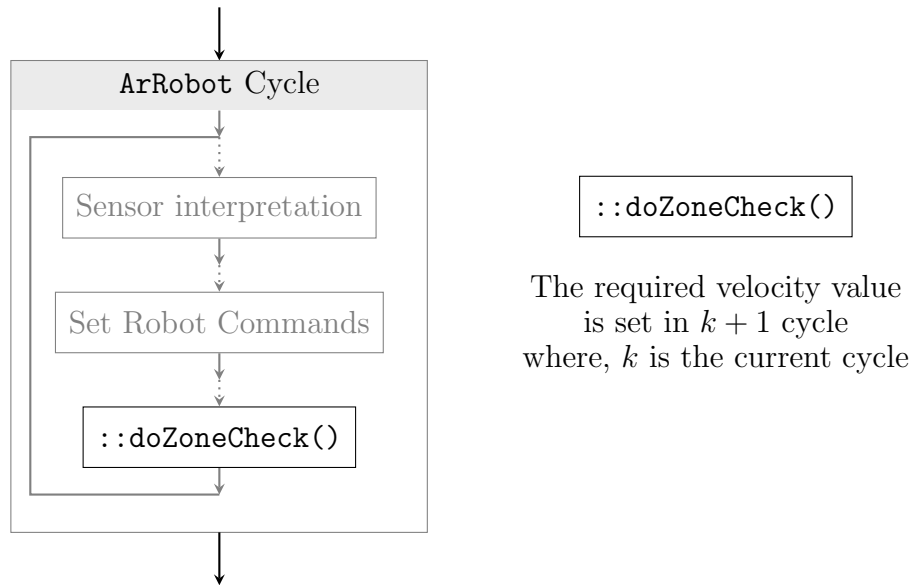


Figure 5.3: Zone check task in ArRobot's flow

### Zonal safety

This task implements the algorithm to limit the robot's speed based on its distance from obstacles. A graphical illustration of predefined boundaries and their capped maximum translational and rotational velocities are shown in Figure 5.4. Since robot is covered with sensors on all sides a function called `ArRobot::checkRangeDevicesCumulativePolar` from ARIA library 4.1.2 is used to get the obstacle range in all direction i.e.  $360^\circ$ . That is how a boundary based approach gives the operator a complete confidence in driving the robot with high speeds.

Calculations for velocities based on their boundary are done by taking a ratio of current velocities to their maximum velocity of an applicable boundary. The equations 5.2a & 5.2b shows how new velocities values are decided, corresponding ratio's are as per equations 5.1a & 5.1b. Where  $v$  and  $\omega$  are translational and rotational velocities respectively,  $v_{max}$  and  $\omega_{max}$  are maximum translational and rotational velocities respectively.

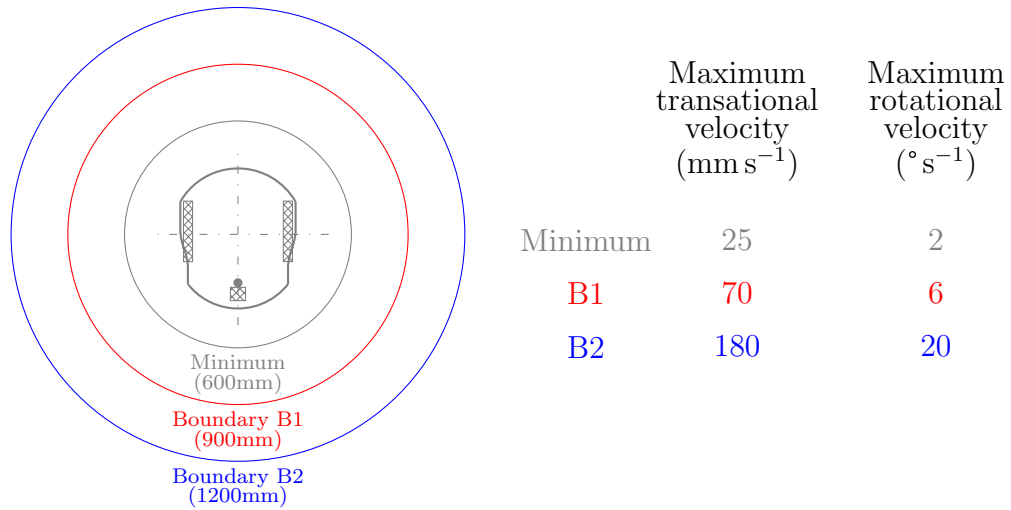


Figure 5.4: Safety zones and maximum velocities

$$v_{ratio} = \frac{v}{v_{max}} \quad (5.1a)$$

$$\omega_{ratio} = \frac{\omega}{\omega_{max}} \quad (5.1b)$$

$$v = \begin{cases} v_{max} & \begin{cases} \text{if } v > v_{max} \ \& \ \omega = 0 \\ \text{if } v_{ratio} > \omega_{ratio} \end{cases} \\ \frac{v}{\omega_{ratio}} & \text{if } \omega_{ratio} > v_{ratio} \end{cases} \quad (5.2a)$$

$$\omega = \begin{cases} \omega_{max} & \begin{cases} \text{if } \omega > \omega_{max} \ \& \ v = 0 \\ \text{if } \omega_{ratio} > v_{ratio} \end{cases} \\ \frac{\omega}{v_{ratio}} & \text{if } v_{ratio} > \omega_{ratio} \end{cases} \quad (5.2b)$$

The equation 5.1 explains the execution of the safety task clearly where  $v$  and  $\omega$  are variables updated every ArRobot's cycle while  $R$ ,  $Min$ ,  $B_1$  and  $B_2$  are global constants.

- $v$  Current translational velocity commanded
- $\omega$  Current rotational velocity commanded
- $R$  267  $\Rightarrow$  Robot swing radius in mm
- $Min$  600  $\Rightarrow$  Minimum boundary in mm
- $B_1$  900  $\Rightarrow$  B1 boundary in mm
- $B_2$  1200  $\Rightarrow$  B2 boundary in mm

```

get the commanded  $v$  and  $\omega$  values
if packet reception rate < 1 packet/s then
    set  $v = 0, \omega = 0$  to the robot
end if
if zonal distance <  $(B_2 - R)$  then
    if  $v > v_{B_2max}$  or  $\omega > \omega_{B_2max}$  then
        set  $v$  and  $\omega$  with appropriate ratios to max. values of B2
        Equations 5.1 & 5.2
    else
        set  $v$  and  $\omega$  to the robot
    end if
    return
end if

if zonal distance <  $(B_1 - R)$  then
    if  $v > v_{B_1max}$  or  $\omega > \omega_{B_1max}$  then
        set  $v$  and  $\omega$  with appropriate ratios to max. values of B1
        Equations 5.1 & 5.2
    else
        set  $v$  and  $\omega$  to the robot
    end if
    return
end if

if zonal distance <  $(Min - R)$  then
    if  $v > v_{Minmax}$  or  $\omega > \omega_{Minmax}$  then
        set  $v$  and  $\omega$  with appropriate ratios to max. values of Min
        Equations 5.1 & 5.2
    else
        set  $v$  and  $\omega$  to the robot
    end if
    return
end if
set  $v$  and  $\omega$  to the robot
return

```

Algorithm 5.1: Velocities update for zone check

### 5.2.2 Transmission

The packet transmission flow chart is shown in Figure 5.5 it consists of a timer to send packets in a fixed interval, the Tx program which is running in parallel to ArRobot gathers the necessary data from ArRobot and packages it before sending via serial link. As shown in Figure the 'transmission enable' flag is always checked, its setting causes an immediate end to transmission. The packet transmitted here is only position data from table 5.6. To properly terminate this thread an 'end' flag is used which is set just before closing of the program.

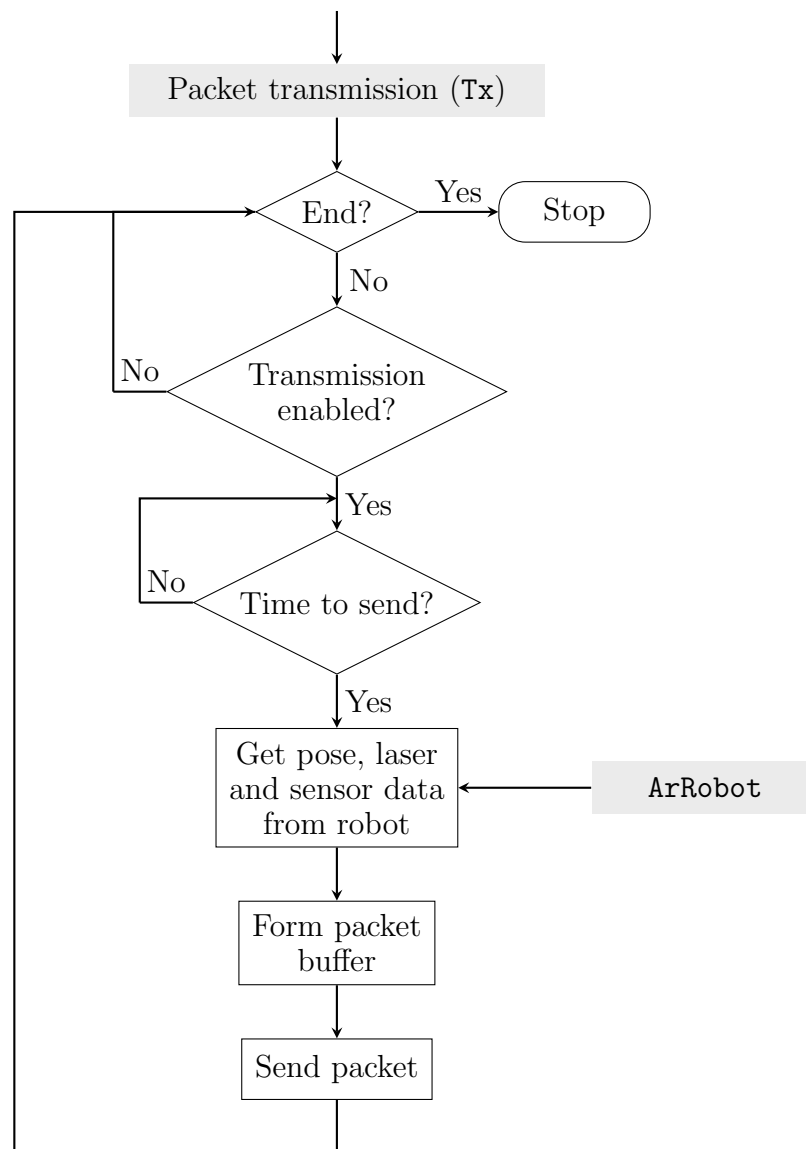


Figure 5.5: Robot packet transmission (Tx)

### 5.2.3 Reception

The reception packet flowchart is shown in Figure 5.6. This program parses the received packets from controller UI and verifies authenticity of each packet in the following steps:

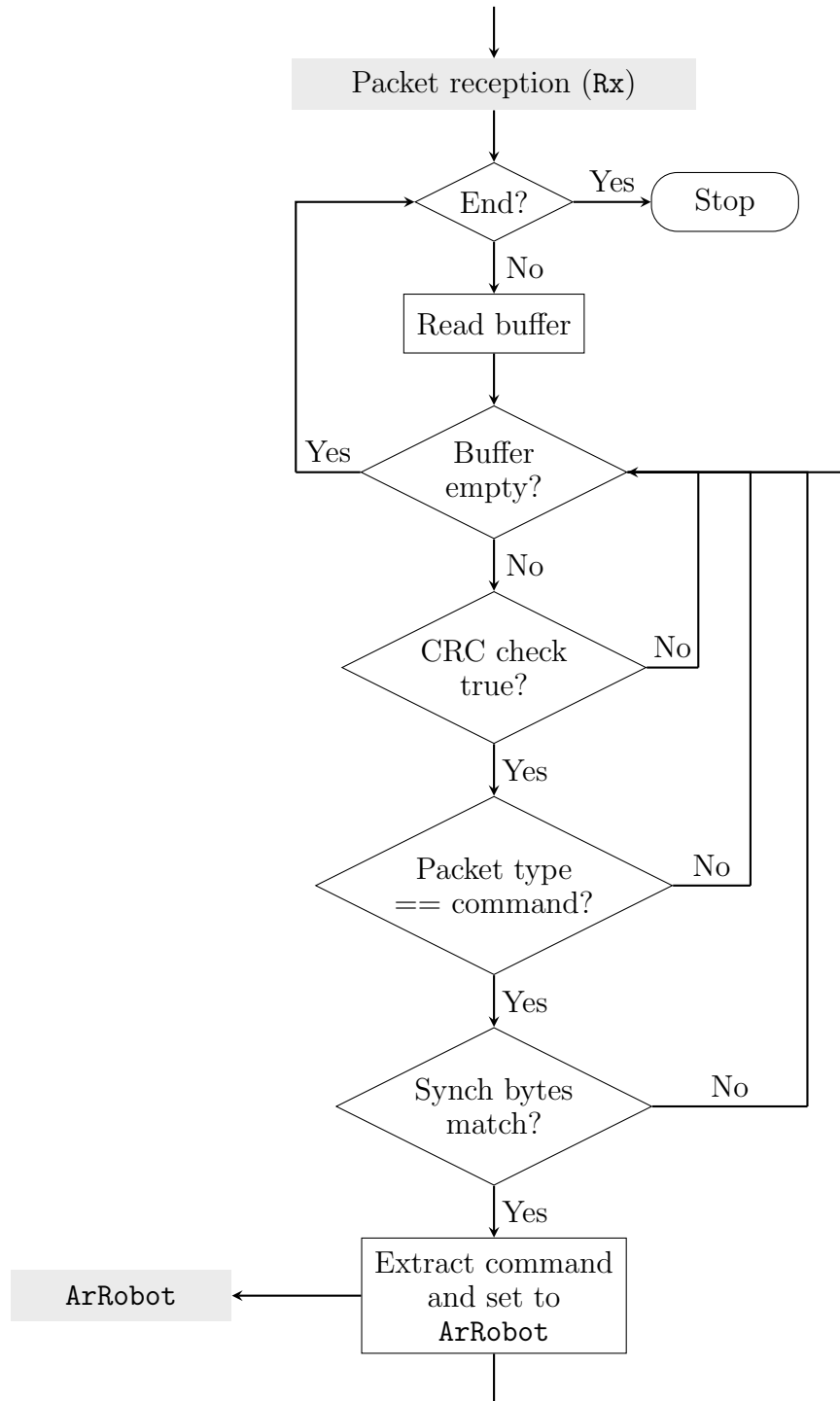


Figure 5.6: Robot packet reception (Rx)

CRC checking

Type of the packet

Sync bytes matching

After a proper packet is verified the commands are extracted and set to parallelly running `ArRobot` program. The buffer is only read if its full and its cleared after the packet has been extracted. The thread is terminated properly on program shutdown using 'end' flag.

## 5.3 Desktop application development

The components on the controller side are divided into:

1. GUI - User interface thread to handle operator inputs and display sensor graphics and robot parameters.
2. Transmission - this thread handles packet forming, CRC append and write to the transmission buffer.
3. Reception - this thread handles packet reading from buffer, CRC validation, packet parsing and applying them to graphics and send data from the transmission buffer.

The program overview is shown in Figure 5.7 where UI runs as a parallel task with transmission and reception tasks. The program can be stopped by simply closing UI. It has to be started from a command window by passing the argument for the communication node's port number. **E.g. `robotUserInterface.exe 3`**

### 5.3.1 GUI

The cockpit style approach as in Figure 2.11 was taken in designing the User interface. It is built with the help of Qt framework described in Section 4.2, the whole dialog consists of:

- 'Interactive graphics scene' to display robot orientation and position inside a given map, and takes in keyboard input while driving the robot in manual mode
- 'Rx and Tx graphs' for real time data reception and transmission visualization

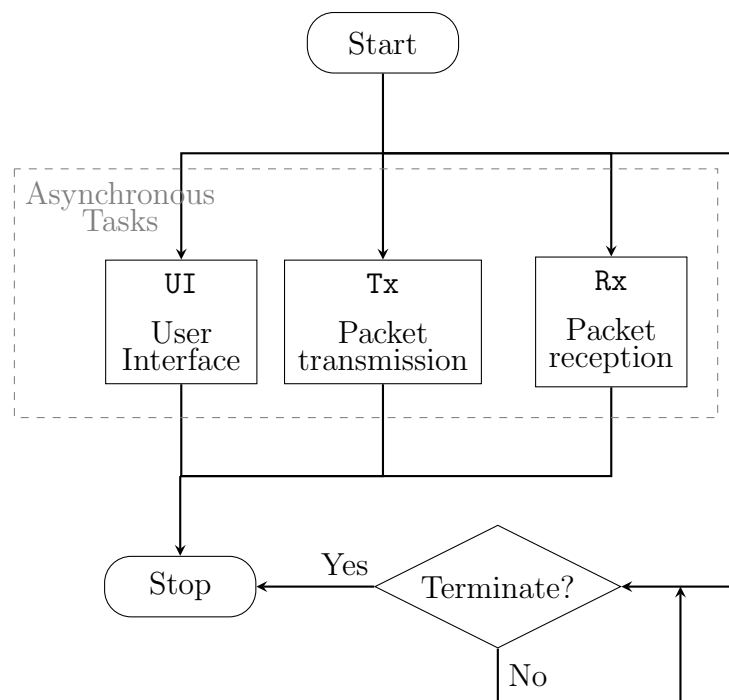


Figure 5.7: Ui application flow

- ‘Robot control’ to drive **WMR** in different modes (Motion controlled, linear controlled and manual)
- ‘Packet rate control options’ to enable transmissions and change transmission interval for the robot and operator
- ‘Current velocity and position’ indicator

As shown in Figure 5.8, a zoomed closeup of the ‘graphics scene’ showing laser and sonar sensor readings are also plotted as in Figure 5.9.

Using signal and slot mechanism as described in section 4.2 widget update functions are called whenever a new data is available to be displayed, that is how Rx and Tx graphs, Interactive graphics scene and velocity and position indication widgets are rendered in real time giving a continuous visual display of parameters.

### Modes of robot operation

There are three modes to drive the robot using controller, the latest command packets are formatted and sent to Tx task through the use of a single function thus ensuring minimum inter-thread functional calls. At any moment only one mode can be used as each of them produce unique inputs.



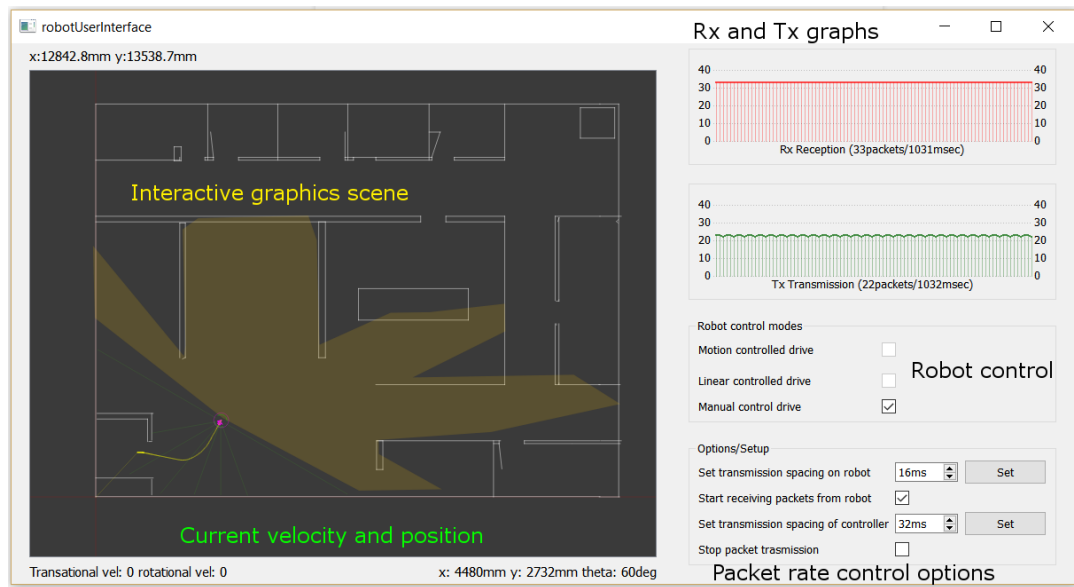


Figure 5.8: User interface application

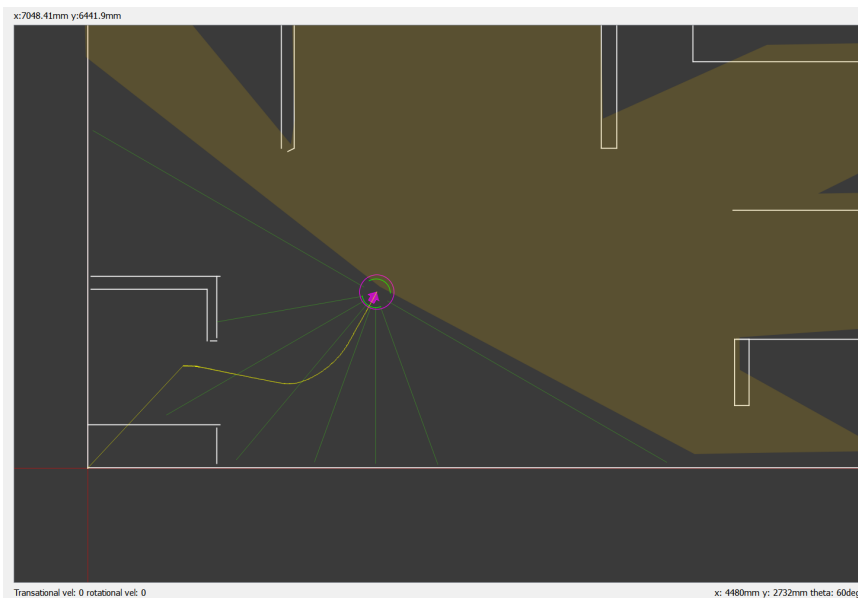












Figure 5.9: Interactive graphics scene zooming

**Manual control** Under this mode the 'nteractive graphics scene' accepts keyboard input. The arrow keys (  ,  ,  ,  ) and space bar (  ) events are processed to generate velocity values as follow:

	$v = +20 \text{ mm s}^{-1}$
	$v = -20 \text{ mm s}^{-1}$
	$\omega = +2 \text{ }^\circ\text{s}^{-1}$
	$\omega = -2 \text{ }^\circ\text{s}^{-1}$
	$v = 0 \text{ and } \omega = 0$

These velocities values are directly set to Tx task. The velocity values incremented here are the lowest according to the robot's operation manual [12].

**Motion control** The algorithm implemented here for motion control of the robot come from Chapter 3 which is a non-linear state tracking controller (eq. 3.17). A trajectory or reference target generator is started that monitors the current robot state, whenever a current state  $q$  is reached it generates a new state  $q_{+1}$  for the motion controller to calculate correction in velocity which is then sent to the robot via Tx task.

**Require:**  $allTargets$

**while**  $allTargets! = 0$  **do**

**while**  $q_{current}! = q_{reference}$  **do**

        wait

**end while**

    Generate next target references from  $x$  and  $y$  of  $allTargets$

    Equations 3.3 and 3.5

$q_{reference} = (x_{reference}, y_{reference}, \theta_{reference})$

    Set to motion control algorithm 5.3

**end while**

Algorithm 5.2: Reference target generator

$$v_{ratio} = \frac{v}{v_{min}} \quad (5.3a)$$

$$\omega_{ratio} = \frac{\omega}{\omega_{min}} \quad (5.3b)$$

$$v = \begin{cases} v_{min} & \begin{cases} \text{if } v < v_{min} \ \& \ \omega = 0 \\ \text{if } v_{ratio} < \omega_{ratio} \end{cases} \\ \frac{v}{\omega_{ratio}} & \text{if } \omega_{ratio} < v_{ratio} \end{cases} \quad (5.4a)$$

$$\omega = \begin{cases} \omega_{min} & \begin{cases} \text{if } \omega < \omega_{min} \ \& \ v = 0 \\ \text{if } \omega_{ratio} < v_{ratio} \end{cases} \\ \frac{\omega}{v_{ratio}} & \text{if } v_{ratio} < \omega_{ratio} \end{cases} \quad (5.4b)$$

Equations 5.3 receives reference targets from algorithm 5.2 whenever a current target state is reached. The motion controller then calculates new trajectory values and modifies velocities if their values exceed  $max$  and  $min$  boundary

limits. In case of readjusting values for *min* condition sometimes calculated ratios might produce a value which exceeds *max* allowed velocity, which is then re-adjusted as per equations 5.3 and 5.4. These limits in robot velocities are preset and compiled into programs on both robot side and controller side.

**Require:** current and reference robot configuration and velocities limits

```

while task running do
  Calculate gains  $k_1$ ,  $k_2$  and  $k_3$  for Eq. 3.18
  Calculate velocities  $\omega$  and  $v$  from Eq. 3.17
  if  $v > v_{max}$  or  $\omega > \omega_{max}$  then
    set  $v$  and  $\omega$  with appropriate ratios to max. values
    Equations 5.1 & 5.2
  else
    keep  $v$  and  $\omega$ 
  end if
  if  $v < v_{min}$  or  $\omega < \omega_{min}$  then
    set  $v$  and  $\omega$  with appropriate ratios to min. values
    Equations 5.3 & 5.4
    if  $v > v_{max}$  then
       $v = v_{max}$ 
    end if
    if  $\omega > \omega_{max}$  then
       $\omega = \omega_{max}$ 
    end if
  else
    keep  $v$  and  $\omega$ 
  end if
  send the  $\omega$  and  $v$  to Tx task
end while
set  $v = 0$  and  $\omega = 0$ 

```

Algorithm 5.3: Motion control

**Linear control** This is a simple test case where a robot is driven along x-axis and the speeds are set to keep the robot moving between two states along x-axis which is governed by equation 5.5. A fairly simple algorithm in 5.4 is run in

this mode.

$$v = \sqrt{|x_{reference} - x_{current}| * 50} \quad (5.5)$$

**Require:** *direction*, current, start and stop configurations

```

while task running do
  if  $x_{current} \leq x_{reference_{start}}$  then
     $direction = forward$ 
  end if
  if  $x_{current} \geq x_{reference_{stop}}$  then
     $direction = reverse$ 
  end if
  calculate  $v$  from eq. 5.5
  if  $direction = forward$  then
    set  $v$  and  $\omega = 0$  to Tx task
  else
    set  $-v$  and  $\omega = 0$  to Tx task
  end if
end while

```

Algorithm 5.4: Linear control

### 5.3.2 Transmission

The radio device being used is configured for half-duplex operations hence to keep the data packet collision to minimum Tx program is designed differently from robot application's Tx counterpart. A global buffer parameter is used here to write the transmittable command packets at a specific transmission interval. This global buffer is then read by Reception program and transmitted with minimalistic additional delay. The figurative illustration is drawn in flowchart 5.10.

### 5.3.3 Reception

The Rx program is similar in operation to robot application's Rx with the addition of a transmission function. In the ever running program loop the global buffer is checked for its content if its full the data packet is transmitted and the

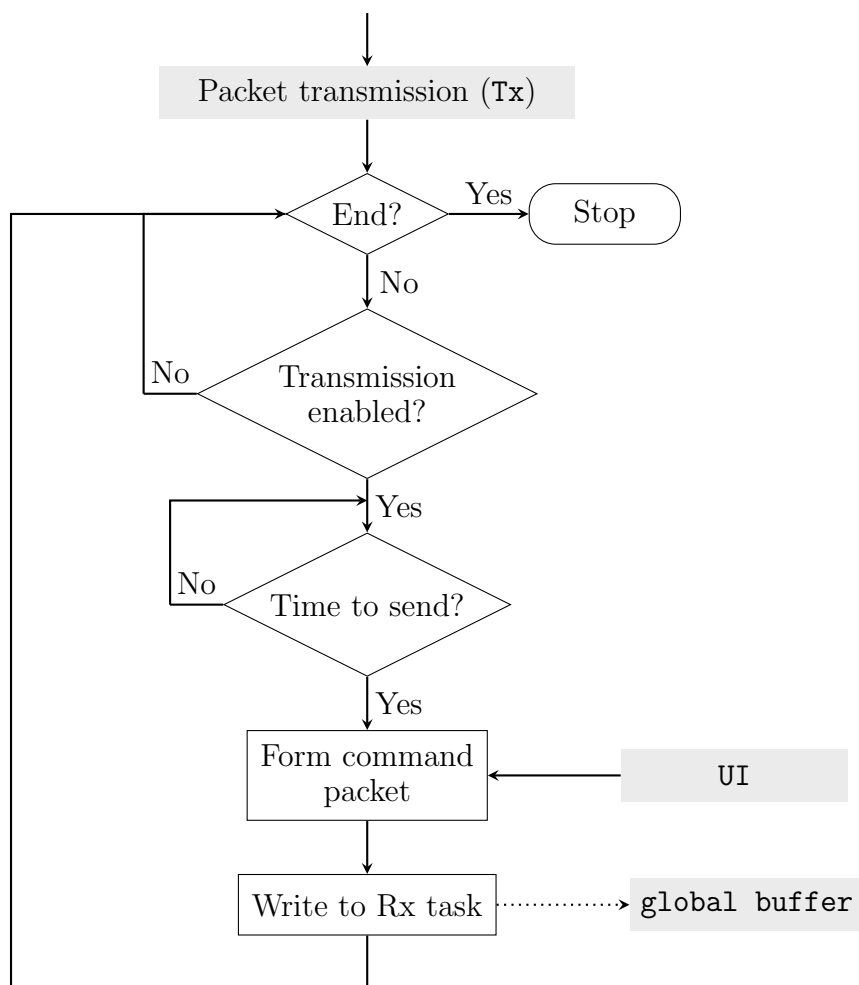


Figure 5.10: Ui packet transmission (Tx)

buffer is cleared. This way only one program has access to the radio device at a time to read or to write data. The time it takes to fully read the reception data might add additional delay to the packet transmission but at these high speed data transfer rates the robot control is always maintained given the radio signal strength is at full and wireless devices are communicating. The flowchart in Figure 5.11 illustrated the program operation.

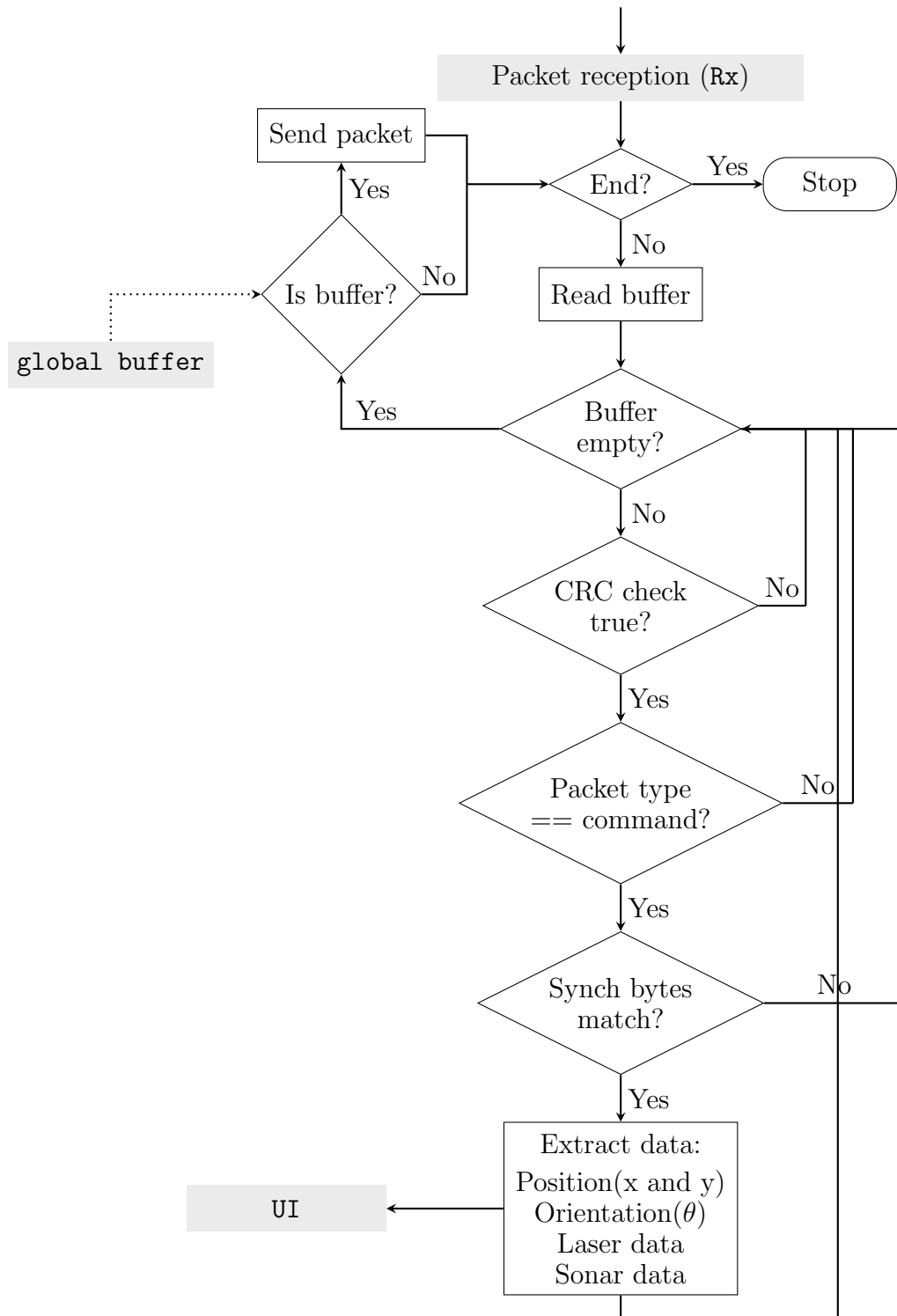


Figure 5.11: Ui packet reception (Rx)

# Chapter 6

## Experiment setup

This chapter discusses the experiments related setup and usage of UI. Testing of the system was done for functional verification and performance analysis. The functional testing is selected to show a working of the full system features i.e., mode of operations and zonal safety behaviour. Performance analysis is done on the data collected from several field trails.

### 6.1 Setup

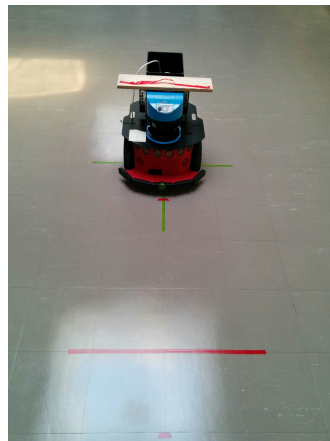
**2D map** First thing needed to be done before using laser localization with ARIA library 4.1.2 was to acquire a 2D map information of the field, [MobileRobots](#) provided an application called Mapper3<sup>1</sup> for that purpose. During testing, this map is loaded into operator's graphics scene where robot movements and sensor readings are overlaid. Indoor field testing was conducted as the robot was meant for indoor use, the real picture of the area is shown in Figure 6.1, 2D map of the field is shown in Figure 6.3.

**System start** The operator's computer was a laptop and robot has an on-board computer with serial and Ethernet interfaces. The IP addresses for Ethernet were fixed for both so that the robot program could be run via remote desktop client of windows OS. The program `robot.exe` was run everytime a new experimental trail was undertaken. Packet transmission from the robot can only be activated on demand by the operator so, ones the Ethernet cable is removed operator moves to his designated position in the field and the program

---

<sup>1</sup>Mapper application to convert laser data (.2d) to 2D map (.map)

<http://robots.mobilerobots.com/wiki/Mapper3>



(a) Green cross tape is origin



(b) View from robot behind



Figure 6.1: Experiment area images

`robotUserInterface.exe` is run on the operator side. By clicking on GUI options (Figure 6.2) for transmission setting he can start communicating with the robot.

On the robot side there is a main task cycle which has a regular 100 ms timing (Figure 4.4). Since the transmission interval from the operator is 47 ms i.e. two packets per task cycle, the commands are cached and set at one go. This has no impact on robots overall performance but it is a measure of communication nodes speed and program capability.

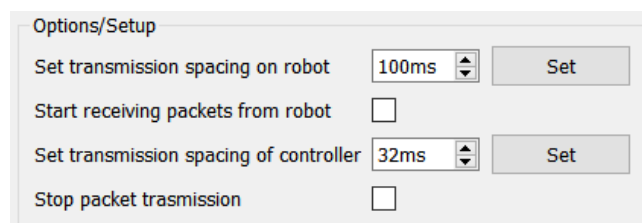


Figure 6.2: Packet transmission settings



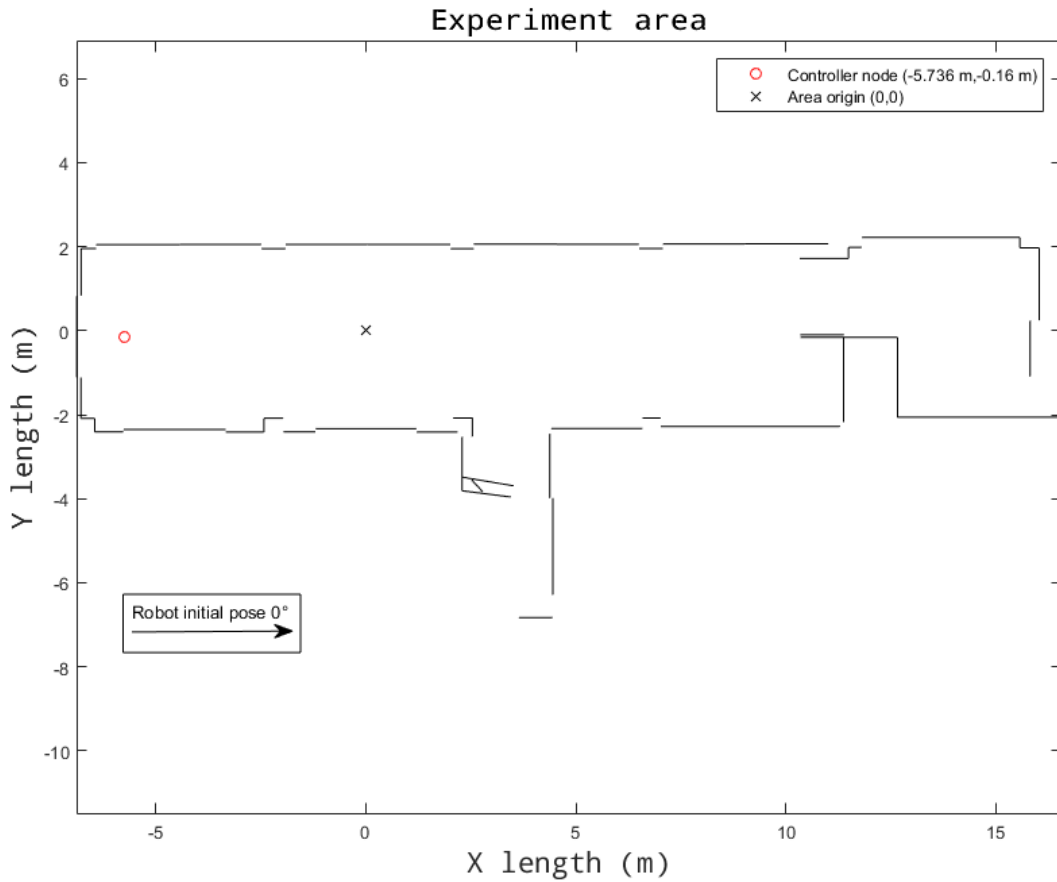


Figure 6.3: Experimental area map

## Constants

Some parameters that are kept as constants throughout testing are:

1. The controller node position  $(-5.736m, -0.16m)$  (red circle in Figure 6.1) is in direct line of sight with the robot node in most of the field.
2. Robot origin position  $(0m, 0m)$  ('x' mark in Figure 6.1). This start configuration is  $(x, y, \theta)^T = (0, 0, 0)^T$  and is set as a default initial position by hard coding the values at the start of the localization algorithm.
3. Packet transmission rates are chosen in such a way that packet loss and collisions are to the minimum, this was done by keeping the nodes at constant distance and varying the values using GUI.
  - The robot transmission rate is 31 ms.
  - The controller (or UI) transmission rate is 47 ms.

In practise the communication rates and packet loss can be effected due to other external factors like distance between the nodes, radio interference, battery power and reflective surfaces.

4. The elliptic path for trajectory tracking was generated for 101 points each as a reference vector values  $(x_r, y_r, \theta_r, v_r, \omega_r)^T$  where

$$\begin{aligned} x_r &= 1.8\cos\left(t\frac{\pi}{50}\right) & y_r &= \sin\left(t\frac{\pi}{50}\right) & \theta_r &= \operatorname{atan2}\left(\frac{\dot{y}_r(t)}{\dot{x}_r(t)}\right) \\ v_r &= \sqrt{\dot{x}_r^2(t) + \dot{y}_r^2(t)} & \omega_r &= \frac{\dot{x}_r(t)\ddot{y}_r(t) - \dot{y}_r(t)\ddot{x}_r(t)}{\dot{x}_r^2(t) + \dot{y}_r^2(t)} \end{aligned} \quad (6.1)$$

5. Maximum operational velocities are:

$$\begin{aligned} \text{Motion control} & \quad 500 \text{ mm s}^{-1} \text{ and } 50^\circ \text{ s}^{-1} \\ \text{Linear control} & \quad 400 \text{ mm s}^{-1} \\ \text{Direct control} & \quad 500 \text{ mm s}^{-1} \text{ and } 50^\circ \text{ s}^{-1} \end{aligned}$$

6. Trajectory controller gain values for equations in 3.18 are  $b = 40$  &  $\xi = 0.8$ .

7. Axes length for elliptical reference path are 2 m and 3.6 m for minor and major axis respectively.

**Operator interface** Operator's interface is shown in Figure 6.4, it is a typical operations view. To control the robot operator simply needs to select the mode of operation from the right side panel, he can use start/stop buttons for linear and motion controlled drive. Whenever user wants to take hold of the robot in the middle of an operation its possible by just using arrow keys on the graphics scene panel the commands are automatically forwarded. On the bottom left corner there are velocity value indicators that the operator currently wants, he can also hover over graphics scene to view the physical location of that point in the map. Scene zooming is possible for better information viewing.

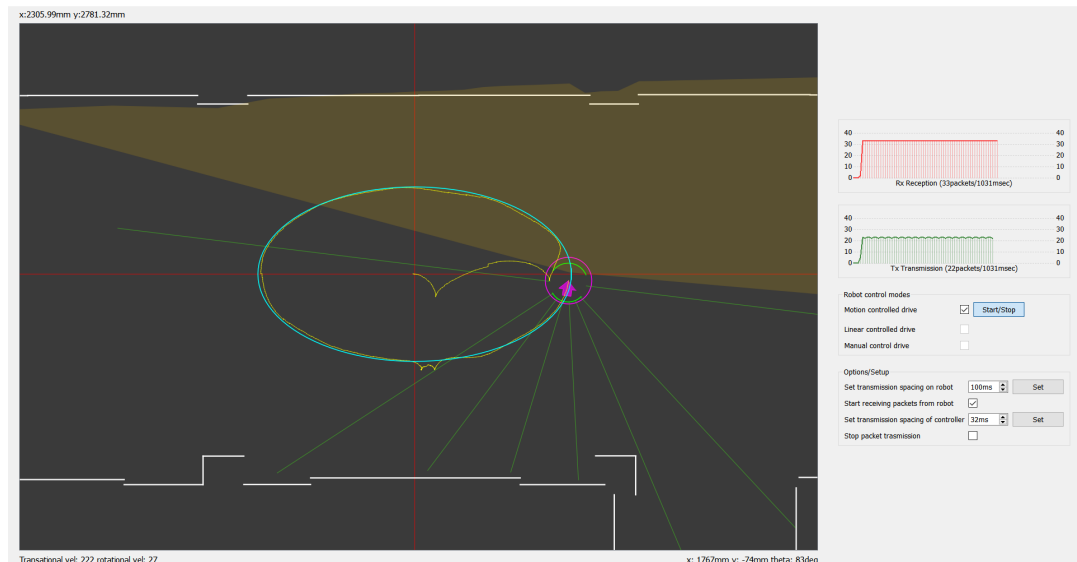


Figure 6.4: Operator interface in action

## 6.2 Test description

The interface is operator centric and all the operations are supervised hence from that point of view the testing purposes are to check navigation with different sensors, use operation modes and verify functioning of the zonal safety algorithm. Analyse data from these results for understanding overall system performance.

**Navigation** The 2D laser data available for the operator can aid in more reliable and accurate commanding in the field, this is compared with sonar and dead reckoning based navigation. In this experiment, two manual drive tests were conducted one with laser switched off and other with laser on.

**Linear drive** This was implemented to check initial remote controllability of the robot. To test this the robot was run with laser localization and between two points which are 3m apart along x axis.

**Motion control drive** A trajectory control algorithm was used to automatically follow an elliptic path, here the effectiveness of laser localization and remote control ability of the system is tested.

**Performance** Not everything works as planned hence its necessary to find out the reasons behind errors and unpredicted behaviour. All the test trails were considered in order to study a problem among them and find out candidate values effecting the systems overall usability. The main reasons were linked to laser, free moving wheel of robot and communication loss.

# Chapter 7

## Results and discussion

This chapter discusses results from the experiments conducted for the three modes of operation 5.3.1 and the zonal safety 5.2.1. Testing objectives are to validate functionality of the system and bring out some key performance related issues.

### 7.1 Functionality

Functionality related results are to check operator navigation, operation modes and working of safety.

#### 7.1.1 Navigation

Laser localization was found to be precise in giving out accurate information for the operator, user navigation in wander mode for dead-reckoning and laser localisation is shown in Figures 7.2 and 7.1 respectively. In case of dead-reckoning the main cause for such a huge error for such a short run was due to the free moving third wheel of the robot. When the robot is driven backwards the free wheel flips its direction and adds that extra motion to slide the robot a bit and change its orientation. Figure 7.3 shows the actual physical location of the robot. Each time robot moves minute error accumulates overtime due to wheel slippage, change in wheel size because of wearing and encoder calibration change, this is the major drawback to dead-reckoning based navigation.

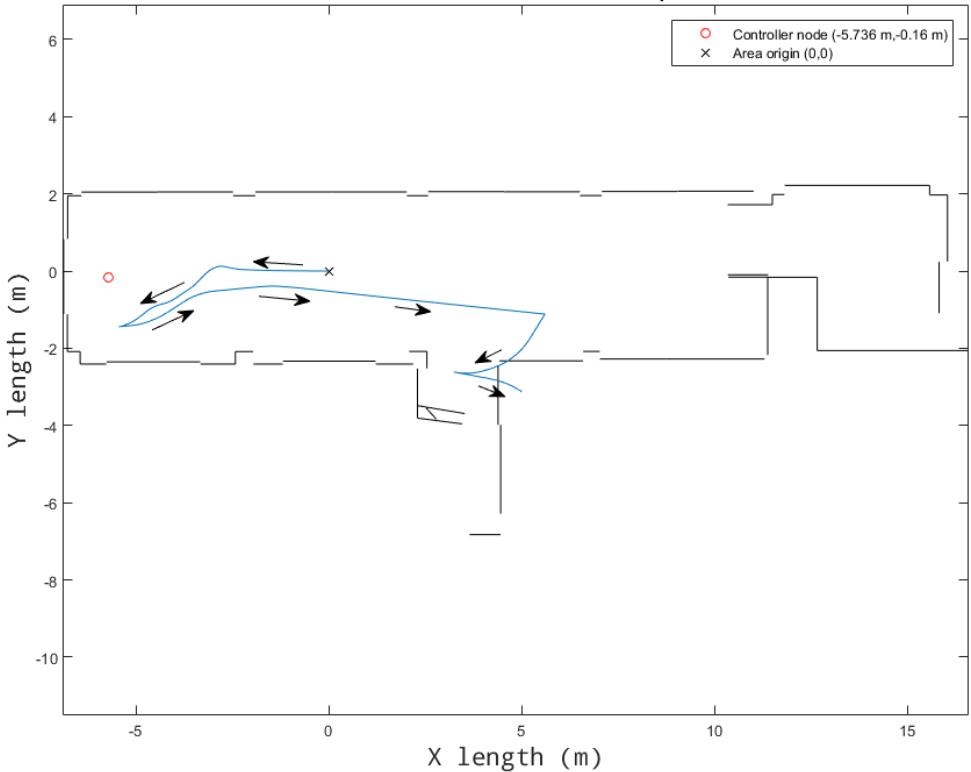


Figure 7.1: Robot path using dead reckoning

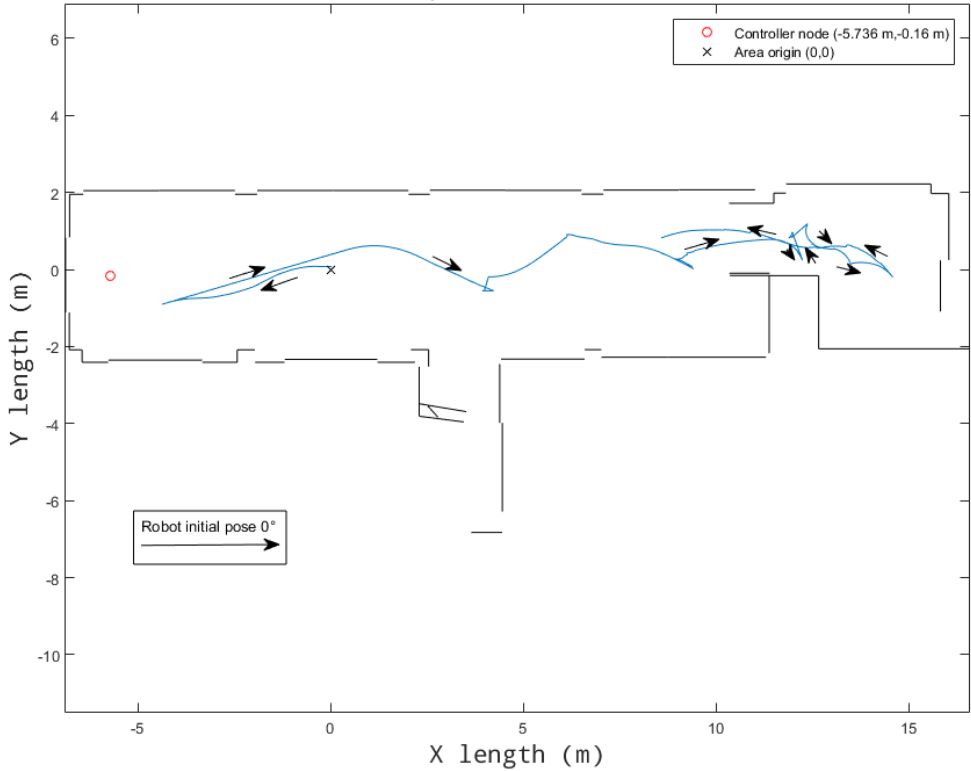


Figure 7.2: Robot path with laser localization

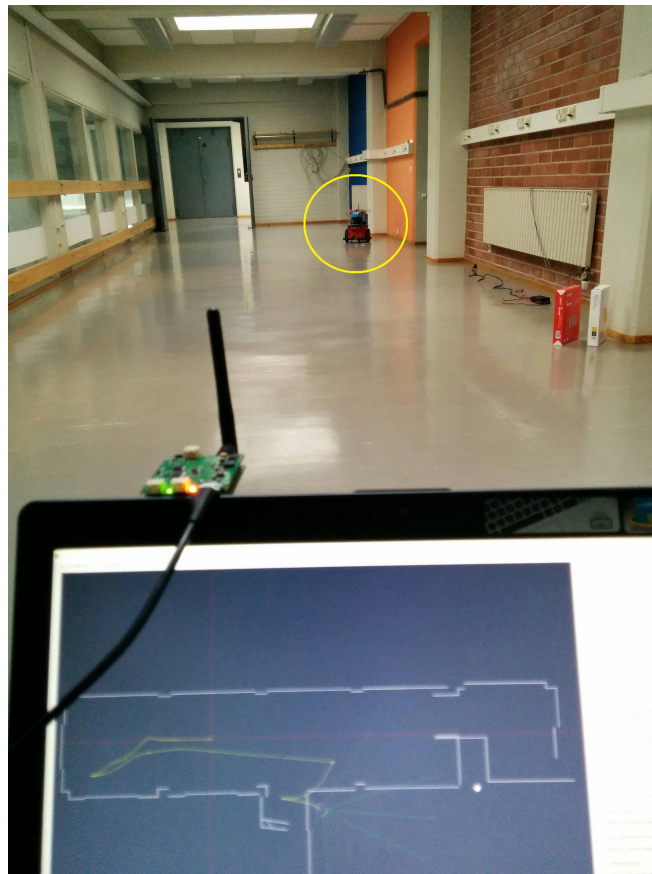


Figure 7.3: Actual robot position (far end of the image) vs calculated position (graphics scene of UI)

### 7.1.2 Linear drive operation

In this mode, the robot is driven along x-axis between points at 3 m distance. Figure 7.4 shows an example of drive on the map. The purpose of this mode is for rapid testing of teleoperation in automatic mode. In performance section effects of the free moving wheel is studied on the robot's orientation with this mode. Figure 7.5 shows distance based velocity generation algorithm in use, the actual velocity set on the robot is slightly different from calculated velocity. This is because of the velocity control algorithm on the robot's middleware, in the forward direction the velocity is gradually increased and on the backward direction it can flip instantly.

### 7.1.3 Trajectory following operation

The robot is made to follow an elliptic path, the gains of controller and capping speeds are set as per 6.1. In this trial, the total time robot took from the static position to completing the oval path was 52.6 s. Figure 7.6 shows its path on

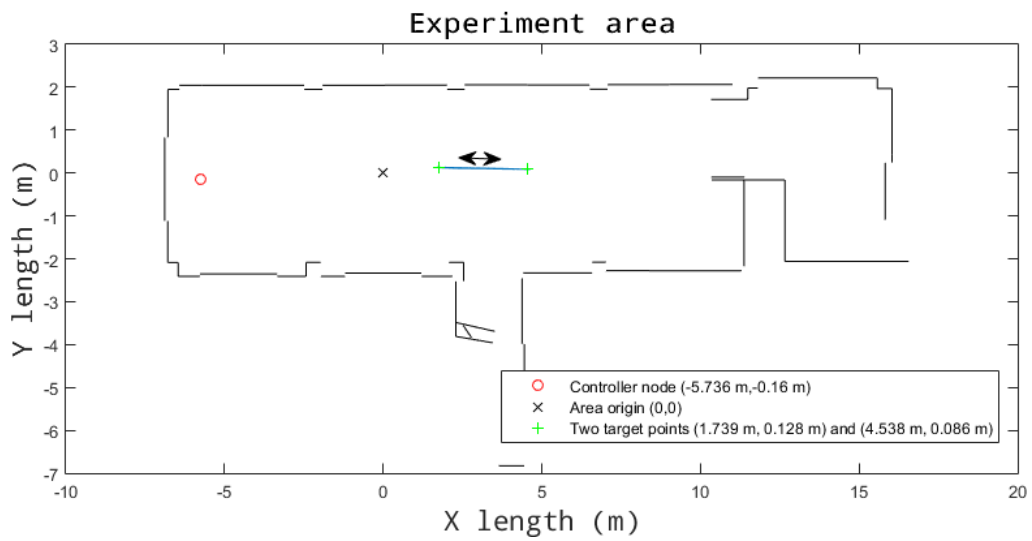


Figure 7.4: Linear robot motion between two point

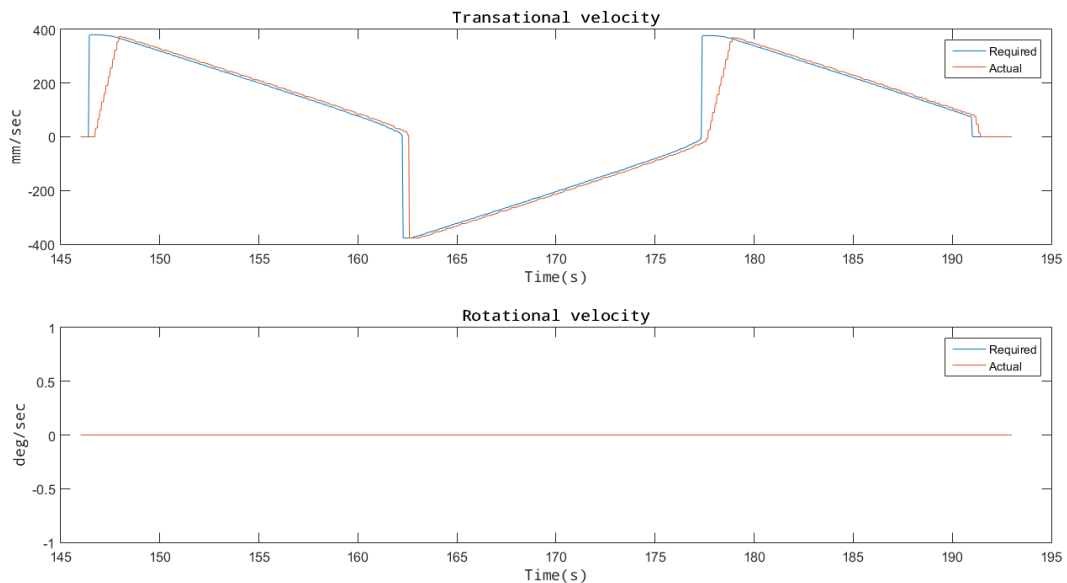


Figure 7.5: Velocities for linear controlled drive

the map, the two big gaps from the actual track are due to the angle shift from  $360^\circ$  to  $0^\circ$ . This is a programming issue that can be fixed but time constraints prevented from redoing tests with new changes.

Figure 7.7 shows the graphs of ideal (red) and actual (blue) path in x, y coordinate and  $\theta$  orientation with error differences (yellow). The reliable tracking was achieved with this algorithm, in all other test cases robot successfully completed the path.

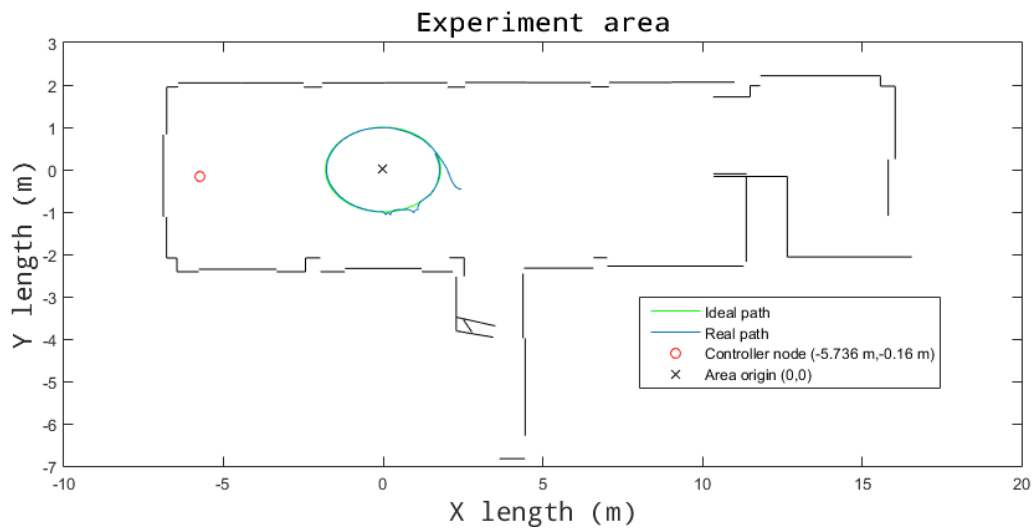


Figure 7.6: Trajectory following map

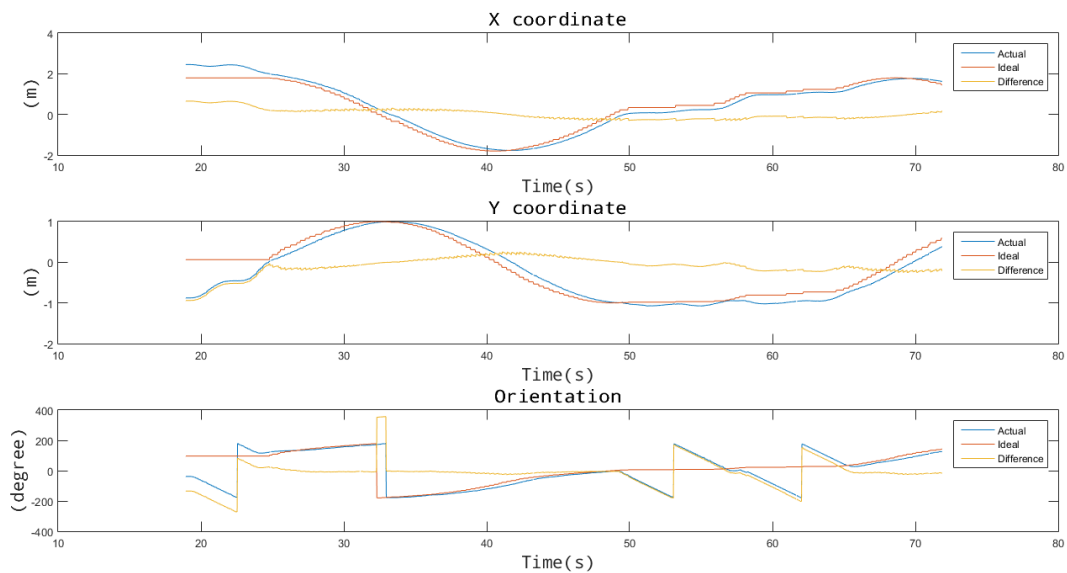


Figure 7.7: Trajectory tracking errors

### 7.1.4 Anti collision safety

During all the testing trials it was observed that safety algorithm always managed to handle robot speeds, even under situations where communication was lost completely it didn't collide with any obstacle. The algorithm is show in 5.1 the only values needed to be assigned are the velocity maximums as mentioned in 6.1. The effectiveness of this can be particularly seen in an experiment where the robot was driven through a narrow part of the area. Below, Figures 7.8 and 7.9 show the points at which a particular zone was activated.

Shades of gray color associated with zones are:



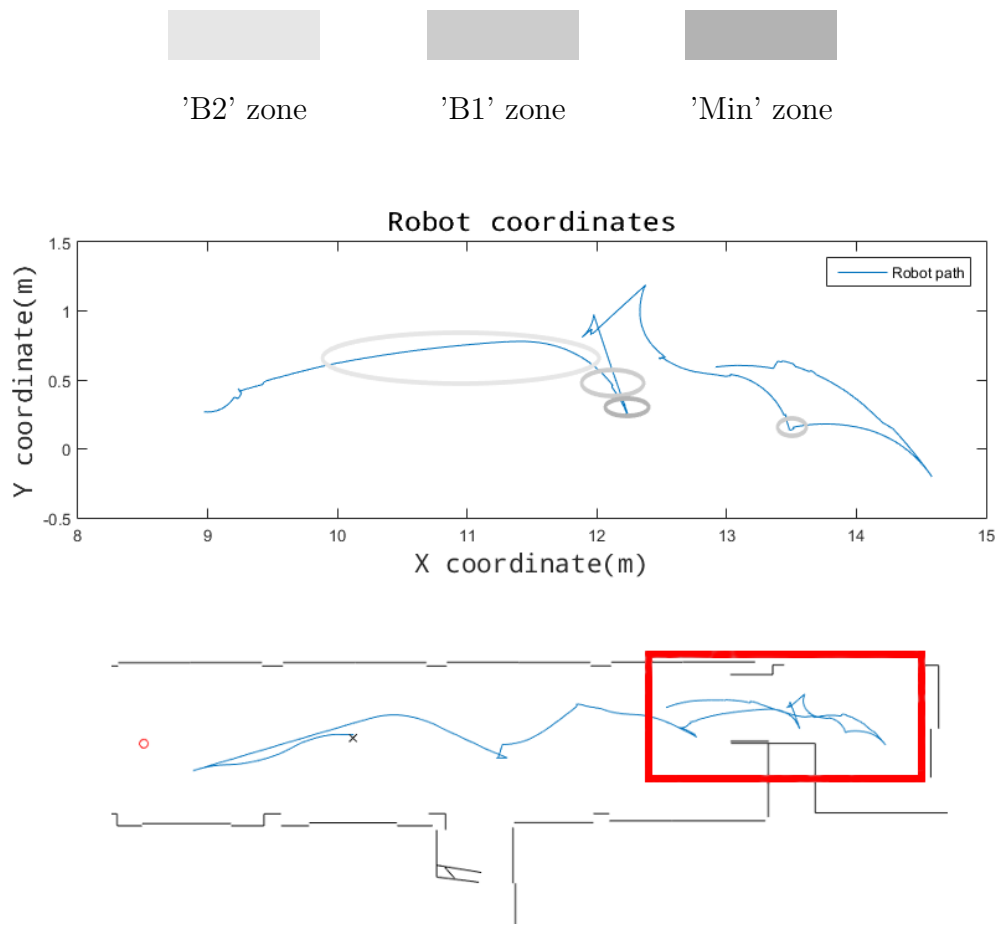


Figure 7.8: Safety activated on the map

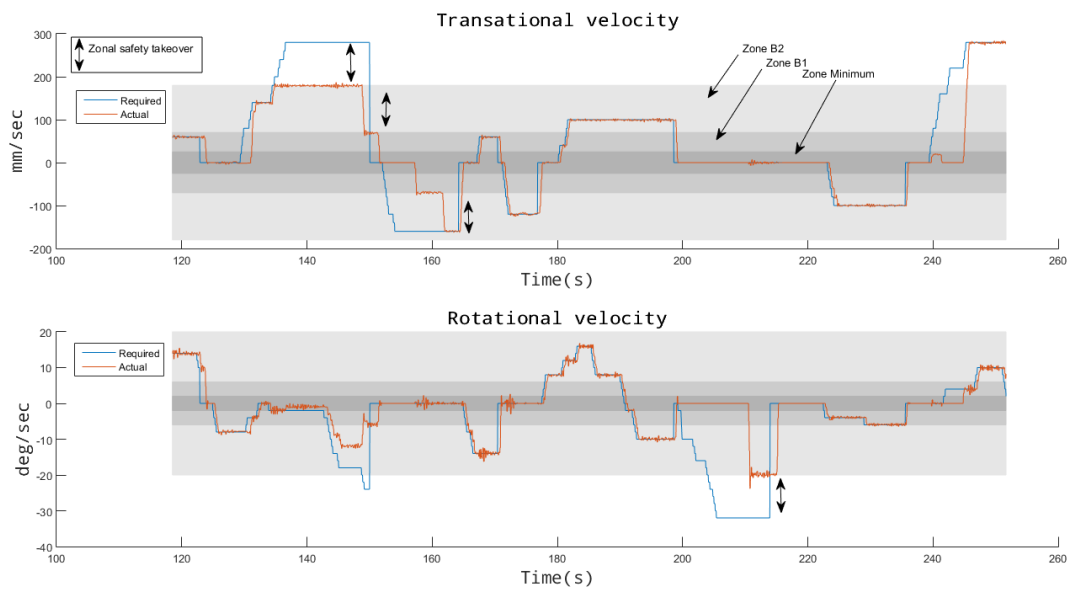


Figure 7.9: Velocity alteration for safety

## 7.2 Performance

Test trials were the combination of different operational modes the discussion here is on the study done on the results obtained from those tests. There were a total of 9 test trials.

Some key observations made from all the tests can be summarised here:

1. Robot collision with an obstacle never occurred.
2. The free moving wheel of the robot had an uncontrollable orientation error.
3. Overall system reliability didn't depend on data packet loss.
4. Communication devices sometime stopped working due to packet collision because of its half-duplex nature.
5. Sometimes laser localization was not effective in updating robot state.

**Free wheel effect** The linear drive was tested between two points along the x axis, it was found that the free moving third wheel had an effect on the simple straight driving of the robot. In ideal conditions, orientation should be the same as when the 'linear mode' operation was started. The median values of orientations from a test in forward-reverse-forward operation was found to be  $-1.1720^\circ$ ,  $-1.2350^\circ$  and  $-1.1840^\circ$  respectively. Orientation of robot would gradually shift from its initial value at  $0.06^\circ$  adding a diversion error in each turn.

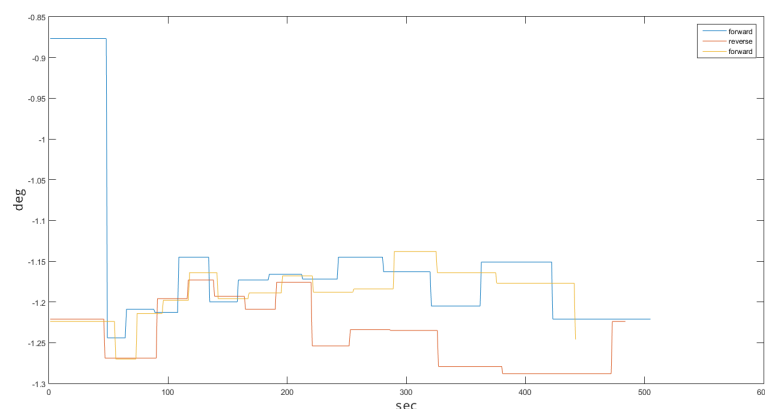


Figure 7.10: Free wheel effect on orientation

**Laser update delay** During the trials it was found that laser localization library from [ARIA](#) was taking time to update the robot state, its effect though

negligible most of the time. In this one case where the robot was operating in ‘trajectory following’ mode there was a 22.17s delay in completing the elliptic path due to this error. The trajectory tracking controller was correcting the path during this time. Figure 7.11 shows in red dots the data points robot path would generate ideally, they should be spread in close formation evenly but there is a gap showing the skip in the robot’s state.

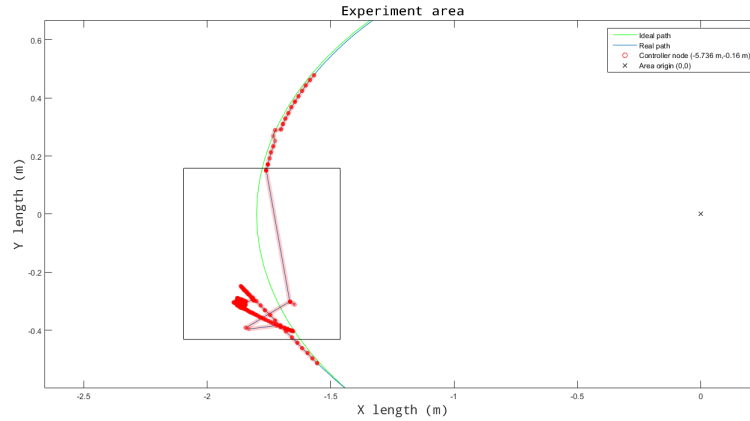
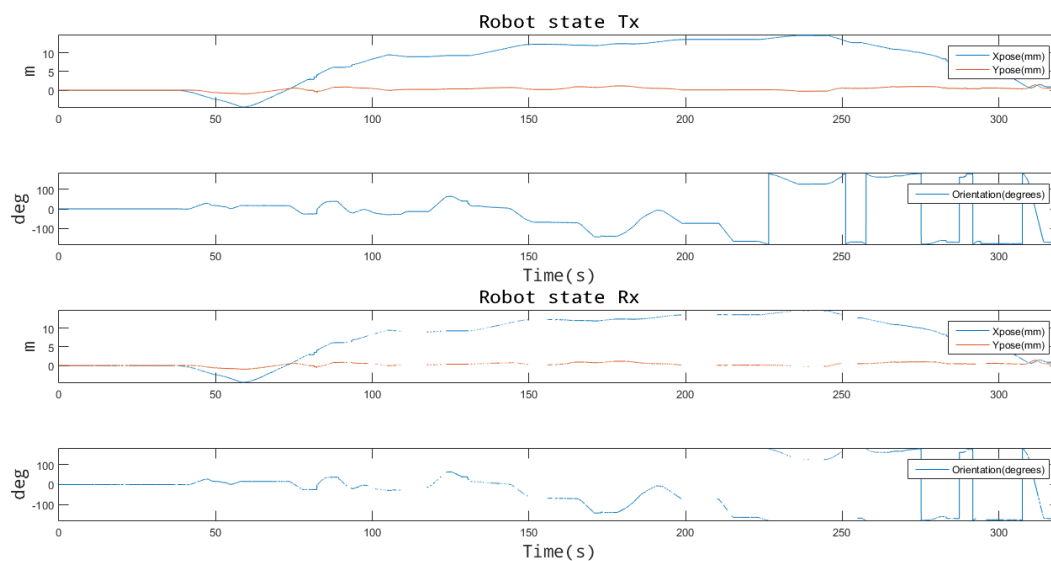


Figure 7.11: Free wheel effect on orientation

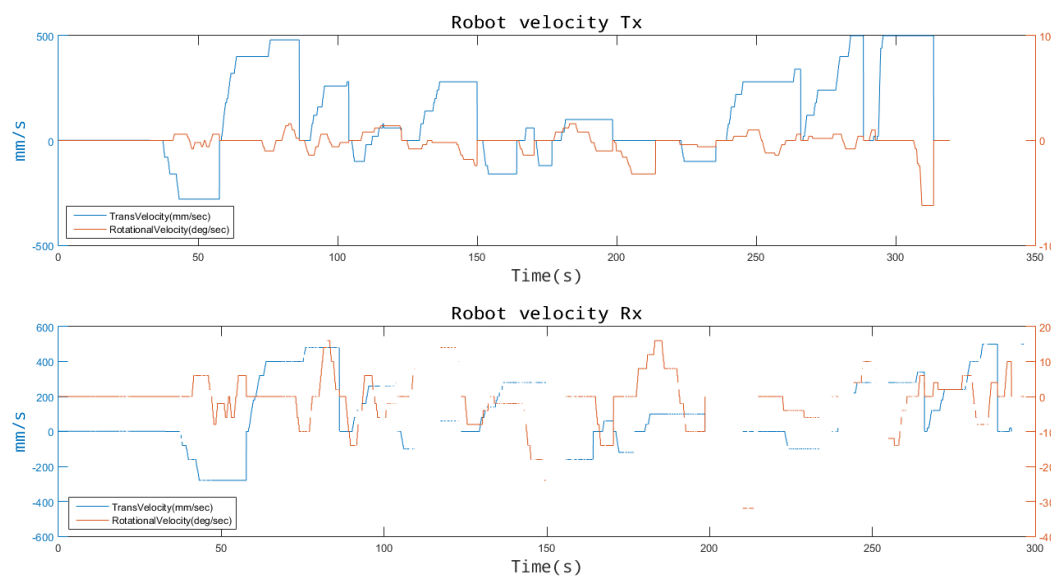
It took 4s to update the state during which the trajectory controller was receiving current state information as  $x = -1760$  mm,  $y = -149.3$  mm and  $\theta = 256^\circ$ , hence it was generating control velocities  $v = 201$  mm s<sup>-1</sup> and  $\omega = 24^\circ$  s<sup>-1</sup>. When the position was updated as shown in figure above it appears as if the robot has teleported. On subtracting the time taken for path correction from total completion time we obtained result of 42.3s which is more close to the ideal time of elliptic path operation.

Typical time for elliptic path	40 s
Time took for elliptic path	64.47 s
Delay caused by laser update	22.17 s

**Communications packet drop** Packet drops in all of the trials are calculated in percentage of the total transmitted packets to the number of unreceived packets. It is observed that packet drop sometimes causes the total lapse of the system because of failure in communication devices. Reason behind this is the half-duplex nature of devices and its prone to cause packet collision in the middleware, the whole system had to be restarted because of this. In all other cases packet drop showed almost no effect. Figure 7.12 shows one such trial where operator commands and robot information were exchanged effectively despite 23.9618% and 28.4325% of packet drops at robot and operator side respectively. Here in the table 7.1 below all the trials and their total packet drop percentages are shown, there is also a column showing the situations where the wireless devices totally failed to operate, it is a randomly occurring event.



(a) Robot state transmission



(b) Velocity command transmission

Figure 7.12: Robot control with packet loss

The effect of the packet drop when studied from trajectory following operations, in three such trials it was found that the packet drop did not contribute to the total time of completion, the reasons for not achieving typical timing of  $\approx 40$  s is trajectory error and laser state updation delay. Table 7.2 shows the packet drops occurred in all those trials.

Modifying the trajectory control algorithm to support  $360^\circ$  to  $0^\circ$  switch could save around 2 s in elliptic path time completion. Operations are highly depen-

	Ui $\rightarrow$ Robot Packet drop (%)	Robot $\rightarrow$ Ui Packet drop (%)	Node fail	Operation time s
Trail 1	1.08	36.65	yes	228.8
Trail 2	24.8	45.8	yes	189.14
Trail 3	9.293	14.69	no	324.9
Trail 4	4.17	18.9	no	432.55
Trail 5	9.45	23.74	no	250.7
Trail 6	8.49	53.4	yes	140
Trail 7	26.4	18.75	yes	208.43
Trail 8	24	28.4	no	319.4
Trail 9	0.3	2.5	no	351.9

Table 7.1: Packet drops in testing

	Total time (s)	Ui $\rightarrow$ Robot Packet drop	Robot $\rightarrow$ Ui Packet drop	Why not $\approx 40$ s
Trial 9	45	0.1771%	0.295%	Trajectory control error
Trial 7	64.47	6.2831%	4.978%	Trajectory control error & Laser state update delay
Trial 4	42.7	4.5317%	2.2063%	Trajectory control error

Table 7.2: Elliptic path completion time and packet drop

dent on laser localization finding a fix for the update problem can make it more reliable.

# Chapter 8

## Conclusions

This final chapter concludes with the overall project observations made throughout the development and testing phase. Mentioned here, are also some limitations and possible future extension of thesis work.

### 8.1 Conclusive remarks

Teleoperation software was successfully tested and the minimum features necessary for a complete teleoperation was fulfilled. Three modes of operations developed can be used in complement to each other. Most important being trajectory tracking which was demonstrated successfully through experiments. Manual drive gives the operator a complete control of the robot simply by relying on the sensor information represented graphically on the UI screen. Monitoring of data packet rates is a clear indicator of communication status and a sign of visual assurance whenever transmission rates are changed. The Experimental area of  $\approx 243\text{m}^2$  was a big testing bed with differently spread wall borders to give variety of testing scenarios.

A continuous packet transmission was the key for a very successful teleoperation in all modes of operation as it ensured proper control even in worst packet drop scenarios, minimum transmission rates achieved were around 16 ms in one direction. Optimal transmission rates chosen were 31 ms and 47 ms in both direction.

In trajectory tracking, minute path adjustments were not possible physically even though the algorithm could generate such values as the minimum allowed rotational speed and translational speed were  $0.2\text{m s}^{-1}$  and  $2^\circ\text{s}^{-1}$  respectively, a maximum error of  $\pm 30\text{mm}$  is inevitable.

The robot was successfully driven through the narrow path of the testing area to demonstrate the safety algorithm ability to react to approaching obstacle and alter the velocities. The three concentric virtual zones defined around the robot fit snugly with robots differential wheel drive ability and spread of sensors around robot covering almost 360°.

Through performance analysis it was concluded that there were cases when laser localization update of robot status effected its operation, in trajectory following it is critical that robot state is updated regularly or it appears as robot teleportation to the operator. It was found that packet loss had no real effect on the overall operation of the system other than the full collapse of the device which paralysed it from communicating. The robot's non-holonomic design with a free moving wheel is shown to have effect on robot's orientation when it is moved in the backward direction, this might be really the critical problem for very narrow driving but if laser localization is reliable then it can be managed to a certain extent.

## 8.2 Future Work

Throughout the development of the project, our idea was to use different communication modules for testing. The most forward looking development would be to increase the range of the teleoperation capability of the whole system. Bringing it more towards real world operational scenario. Increasing range has its own challenges one most significant being communication delays, several methods have been proposed in literature to solving this. There can be implementations of safety based on communications.

More can be done in trajectory generation in avoiding moving/stationary obstacles since, the current trajectory following algorithm is well established. Present localization algorithms can be modified to include mapping that way new obstacles can be remembered in the area which can be used for trajectory generation.

Implementations related to auto communications recovery can be adapted both on the controller and robot side. This could be a significant asset to the reliability of teleoperation system.

# References

- [1] L. Zhang, Z. Chen, J. Wang, and S. Yan, “A networked teleoperation system for mobile robot with wireless serial communication,” in *Proceedings of the International Conference on Robotics and Biomimetics*. IEEE, 2009, pp. 2227–2231.
- [2] S. Lichiardopol, “A survey on teleoperation,” *Dept. Mech. Eng., Dynamics Control Group, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, Tech. Rep. DCT2007*, vol. 155, 2007.
- [3] S. Cong and J. Wang, “Internet-based and visual feedback networked robot arm teleoperation system,” in *International Conference on Networking, Sensing and Control (ICNSC)*. IEEE, 2010, pp. 452–457.
- [4] M. Wang and J. N. Liu, “Interactive control for internet-based mobile robot teleoperation,” *Robotics and Autonomous Systems*, vol. 52, no. 2, pp. 160–179, 2005.
- [5] M. D. Phung, T. T. Van Nguyen, and Q. V. Tran, “Navigation of networked mobile robot using behavior-based model,” in *International Conference on Control, Automation and Information Sciences (ICCAIS)*. IEEE, 2013, pp. 12–17.
- [6] J. Oh, A. Suppé, F. Duvallet, A. Boularias, L. E. Navarro-Serment, M. Hebert, A. Stentz, J. Vinokurov, O. J. Romero, C. Lebiere, and R. Dean, “Toward mobile robots reasoning like humans.” in *The Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*. AAAI, 2015, pp. 1371–1379.
- [7] R. Olivares, C. Zhou, B. Bodenheimer, and J. A. Adams, “Interface evaluation for mobile robot teleoperation,” in *Proceedings of the 51st ACM Southeast Conference (ACMSE03)*, vol. 112, Savannah, GA., March 2003, p. 118.
- [8] N. Shiroma, N. Sato, Y.-h. Chiu, and F. Matsuno, “Study on effective camera images for mobile robot teleoperation,” in *13th IEEE International Workshop on Robot and Human Interactive Communication*. IEEE, September 2004, pp. 107–112.



- [9] D. Saakes, V. Choudhary, D. Sakamoto, M. Inami, and T. Lgarashi, “A teleoperating interface for ground vehicles using autonomous flying cameras,” in *23rd International Conference on Artificial Reality and Telexistence (ICAT)*. IEEE, 2013, pp. 13–19.
- [10] M. Sugimoto, G. Kagotani, H. Nii, N. Shiroma, F. Matsuno, and M. Inami, “Time follower’s vision: a teleoperation interface with past images,” *Computer Graphics and Applications*, vol. 25, no. 1, pp. 54–63, 2005.
- [11] I. Farkhatdinov, J.-H. Ryu, and J. Poduraev, “A user study of command strategies for mobile robot teleoperation,” *Intelligent Service Robotics*, vol. 2, no. 2, pp. 95–104, March 2009.
- [12] A. M. Inc, *Pioneer 3 Operations Manual*, version 3 ed., January 2006.
- [13] Wikipedia, “list of countries by number of mobile phones in use,” August 2015. [Online]. Available: [https://en.wikipedia.org/wiki/List\\_of\\_countries\\_by\\_number\\_of\\_mobile\\_phones\\_in\\_use](https://en.wikipedia.org/wiki/List_of_countries_by_number_of_mobile_phones_in_use)
- [14] T. Fong, J. R. Zumbado, N. Currie, A. Mishkin, and D. L. Akin, “Space telerobotics unique challenges to human–robot collaboration in space,” *Reviews of Human Factors and Ergonomics*, vol. 9, no. 1, pp. 6–56, 2013.
- [15] T. Theodoridis and H. Hu, “Toward intelligent security robots: A survey,” *Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE*, vol. 42, no. 6, pp. 1219–1230, 2012.
- [16] R. R. Murphy, J. Kravitz, S. L. Stover, and R. Shoureshi, “Mobile robots in mine rescue and recovery,” *Robotics and Automation Magazine, IEEE*, vol. 16, no. 2, pp. 91–103, 2009.
- [17] R. R. Murphy, “A decade of rescue robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5448–5449.
- [18] Wireless@kth, “Machine-to-machine redefining information sharing and enablers (m2mrise),” August 2015. [Online]. Available: <https://wireless.kth.se/m2m/projects/m2mrise/>
- [19] T. B. Sheridan, “Teleoperation, telerobotics and telepresence: A progress report,” *Control Engineering Practice*, vol. 3, no. 2, pp. 205–214, February 1995.
- [20] A. K. Bejczy, “Sensors, controls, and man-machine interface for advanced teleoperation,” *Science*, vol. 208, no. 4450, pp. 1327–1335, 1980.
- [21] G. Niemeyer, C. Preusche, and G. Hirzinger, “Telerobotics,” in *Springer handbook of robotics*. Springer, 2008, pp. 741–757.
- [22] W. R. Ferrell and T. B. Sheridan, “Supervisory control of remote manipulation,” *IEEE Spectrum*, vol. 4, no. 10, pp. 81–88, October 1967.

- [23] P. F. Hokayem and M. W. Spong, "Bilateral teleoperation: An historical survey," *Automatica*, vol. 42, no. 12, pp. 2035–2057, September 2006.
- [24] W. Zheng, Y. Wang, and N. Xi, "Behavior coordination in the internet-based multi-robot teleoperation system," in *IEEE International Conference on Robotics and Biomimetics. ROBIO'06*. IEEE, 2006, pp. 988–993.
- [25] A. Birk, S. Schwertfeger, and K. Pathak, "A networking framework for teleoperation in safety, security, and rescue robotics," *IEEE Wireless Communications*, vol. 16, no. 1, pp. 6–13, 2009.
- [26] A. Knoll and R. Prasad, "Wireless robotics: A highly promising case for standardization," *Wireless Personal Communications*, vol. 64, no. 3, pp. 611–617, April 2012.
- [27] R. BOBOC, H. MOGA, and D. TALABĂ, "A review of current applications in teleoperation of mobile robots," *Bulletin of the Transilvania University of Brasov, Series I: Engineering Sciences*, vol. 5, no. 2, 2012.
- [28] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley, "Desktop teleoperation via the world wide web," in *International Conference on Robotics and Automation, Proceedings*, vol. 1. IEEE, 1995, pp. 654–659.
- [29] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, 2012.
- [30] A. F. Winfield and O. Holland, "The application of wireless local area network technology to the control of mobile robots," *Microprocessors and Microsystems*, vol. 23, no. 10, pp. 597–607, 2000.
- [31] F. Espinosa, D. Pizarro, F. Valdes, and M. Salazar, *Remote and Telerobotics*. InTech Open Access Publisher, March 2010, ch. Electronics Proposal for Telerobotics Operation of P3-DX Units, pp. 1–17.
- [32] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities," *Wireless Communications, IEEE*, vol. 20, no. 6, pp. 91–98, 2013.
- [33] Y. Bo, L. Suju, and L. Dongjie, "Time delay study on internet based teleoperation system," in *Proceedings of the 27th Chinese Control Conference*. Kunming, Yunnan, China: IEEE, July 2008, pp. 379–383.
- [34] J. Han and R.-l. Chang, "Research and developing on intelligent mobile robot remote monitoring and control system," *Procedia Engineering*, vol. 16, pp. 840–845, 2011.
- [35] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards," *Computer communications*, vol. 30, no. 7, pp. 1655–1695, 2007.

- [36] K. S. Low, W. N. N. Win, and M. J. Er, "Wireless sensor networks for industrial environments," in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, vol. 2. IEEE, 2005, pp. 271–276.
- [37] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, p. 138, January 2007.
- [38] L. Zhang, H. Gao, and O. Kaynak, "Network-induced constraints in networked control systems—A survey," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 403–416, 2013.
- [39] J. Y. Chen, E. C. Haas, and M. J. Barnes, "Human performance issues and user interface design for teleoperated robots," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 37, no. 6, pp. 1231–1245, 2007.
- [40] T.-J. Tara, A. K. Bejczy, C. Guo, and N. Xi, "Intelligent planning and control for telerobotic operations," in *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, vol. 1. IEEE, 1994, pp. 389–396.
- [41] G. Oriolo, A. De Luca, and M. Vendittelli, "Wmr control via dynamic feedback linearization: design, implementation, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, November 2002.
- [42] M. Amoozgar and Y. Zhang, "Trajectory tracking of wheeled mobile robots: A kinematical approach," in *IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications (MESA)*. IEEE, 2012, pp. 275–280.
- [43] A. De Luca and G. Oriolo, *Kinematics and Dynamics of Multi-Body Systems*. Springer Vienna, 1995, ch. Modelling and Control of Nonholonomic Mechanical Systems, pp. 277–342.
- [44] T. Fukao, H. Nakagawa, and N. Adachi, "Adaptive tracking control of a nonholonomic mobile robot," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 5, pp. 609–615, 2000.
- [45] S. L. Francis, S. G. Anavatti, and M. A. Garratt, "Dynamic model of autonomous ground vehicle for the path planning module." in *Proceedings of the 5th International Conference on Automation, Robotics and Applications*. Wellington, New Zealand: IEEE, December 2011, pp. 73–77.
- [46] F. N. Martins, W. C. Celeste, R. Carelli, M. Sarcinelli-Filho, and T. F. Bastos-Filho, "An adaptive dynamic controller for autonomous mobile

- robot trajectory tracking,” *Control Engineering Practice*, vol. 16, no. 11, pp. 1354–1363, 2008.
- [47] T. Das and I. N. Kar, “Design and implementation of an adaptive fuzzy logic-based controller for wheeled mobile robots,” *Control Systems Technology, IEEE Transactions on*, vol. 14, no. 3, pp. 501–510, 2006.
- [48] G. Klančar, D. Matko, and S. Blažič, “Mobile robot control on a reference path,” in *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*. IEEE, 2005, pp. 1343–1348.
- [49] R. A. Yusein, *The UWASA Node Reference Manual*, University of Vaasa, Department of Computer Science and Aalto University Department of Communications and Networking. [Online]. Available: [http://wsn.aalto.fi/en/tools/uwasa\\_node/](http://wsn.aalto.fi/en/tools/uwasa_node/)