Chapter 1

Introduction to Multi-Agent Robotic Systems

1.1 Introduction:

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. A multi-Agent system is one that consists of a number of agents, which interact with one-another. In the most simple case, all agents are programmed by the same team and they collaborate to complete a task. In the most general case, agents will be acting on behalf of users with different goals and motivations. To successfully interact, they will require the ability to cooperate, coordinate, and negotiate with each other, much as people do.

At the moment widely used robots lack social capabilities to interact with each other in any forms. Therefore, no cooperation is possible among them. Direct implementation of social capabilities, like team formation, negotiations, task assignment and distributed planning into robots would affect both software and hardware of robots. The software of available robots would be changed and made more complex, because additional intelligent components would be needed to implement abovementioned social capabilities. It may demand more complex hardware and some physical means of communication among all robots. As a consequence, significant changes in the widely used robots are mandatory to implement social behavior into robots. Such solution is expensive and lacks the necessary agility. Therefore, we see a need for software that ensures cooperation among robots without specific requirements to them. We propose to address the problem via developing a robot management system that uses data received from robots and performs all the necessary interactions among them. Team formation functionality is part of the management system. The management system decomposes tasks, assigns (and reassigns in case of failures) them to the most suitable robots thereby forming the team and monitors task execution. The goal of the management system is to keep the software and hardware of robots unchanged or change it as minimal as possible. Robots should keep their internal algorithms for task execution and only communication mechanisms to the management system should be added.

1.2 Logical Architecture of Multi-Agent System:

We propose general architecture for robot management systems of different domains that have existing single operating robots, which are capable to execute standard missions, but cannot carry out more labour consuming ones. The architecture allows building multi-robot systems from existing standard robots without making major changes in them. It is suitable for multi-robot systems with the following functionality. To achieve the goal of building the multi-robot systems – to accomplish large missions, the management system has to decompose the task specified by a user until the subtasks can be accomplished by a single robot. The decomposed subtasks must be assigned to particular robots. The allocation should make the overall performance of the system efficient. After tasks are allocated to the robots, the system has to monitor the execution for two main reasons. First, it has to report mission's status to a user and second, various failures may occur during the execution. If a robot fails to complete its task for some reason, this task must be reallocated to other robots.

The proposed architecture has the following main characteristics. The core of the management system is built as a multi-agent system. All interactions among robots are carried out by agents in the robot management system. The robots only have to communicate with the management system or to pass the information to the agents. Each robot has a corresponding (mapped) agent in the system. These agents are named robot agents and represent robots in the management system. Each robot together with its agent makes an autonomous component. Robot agents monitor their robots. They know actual states of the robots, including tasks allocated to the robots and progress of these tasks. Miscellaneous data about each robot, like battery level, errors occurred and robot's pose and location are monitored, too. Still, during the execution of the task each robot uses built in algorithms to execute its tasks autonomously. The robots do not have to implement interactions to other robots and do not have to implement any task allocation mechanisms that are implemented exclusively by the management system. It allows keeping the software and hardware of robots as simple as possible.

Similarly, to robots, the user is represented by an agent, named manager agent. It receives requests for task execution from the user. The agent is responsible for task decomposition into a set of optimal subtasks that could be efficiently executed by individual robots. It has to assign subtasks to the robot agents and reallocate them in case of any failures in their execution by using some kind of negotiation protocol. The architecture is general in the sense that it does not specify the negotiation protocols. It allows using different protocols for task allocation and reallocation. For example, widely applied option of negotiation protocols is the Contract Net protocol [1]. Subtasks can be sequentially allocated using separate Contract Net protocols. Such mechanism is sufficient for the management system's functionality. Still it may lead to suboptimal task assignments. More complex option is the TraderBots approach [2]. It adds market based mechanisms to exchange tasks allocated to the robots if it leads to more optimal allocation. The exchange is done based on some virtual currency. Other options of negotiation protocols are Murdoch and protocols based on beliefs desires and intensions [3]. The generality of architecture is based on the fact that irrespective to the task allocation protocol the agents have to do the same tasks. Firstly, the manager agent has to define the task and initiate the negotiation. The robot agents have to evaluate their abilities to carry out the task and calculate costs (or some number with similar semantics) to indicate how appropriate the robot is for the task. The most important difference in negotiation protocols is existence of direct interactions among robot agents if the TraderBots or any similar approach is used. So, the architecture defines the agents used in the management system, their functionality and general interactions among them irrespectively of the task allocation mechanism used. Actually, the architecture allows realizing various negotiation protocols in the same system.

After the task is allocated to the robot its agent gives appropriate command to the robot. During the execution of the task by the robot, its agent follows the execution and reports the progress to the manager agent. Moreover, in case of any failure during the execution of the task, robot agent reports it to the manager agent, which initiates new negotiation to reassign the task to another robot. The logical architecture of the robot management system is shown in **Figure 1.1**. Optional interactions among robot agents (used only in some negotiation protocols) are denoted with dashed lines.

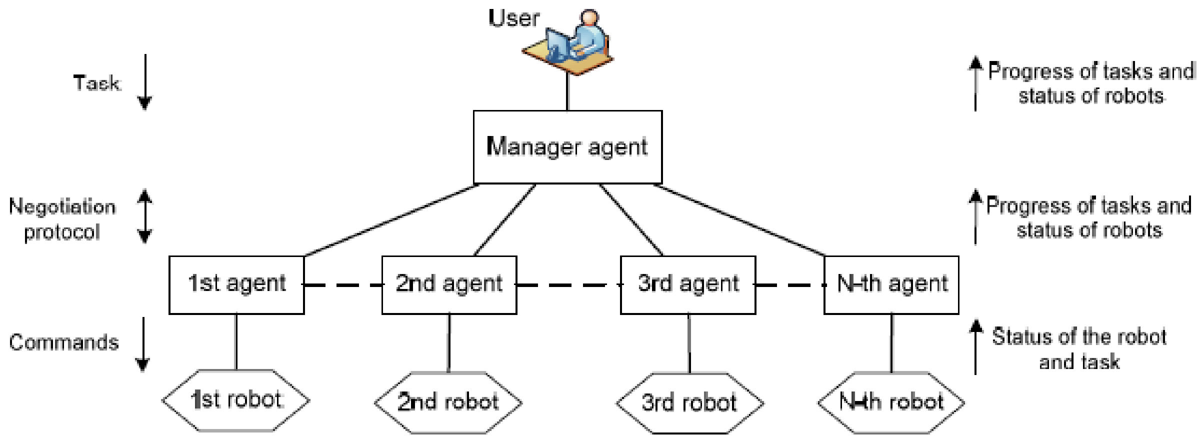


Fig. 1.1 Logical architecture of the management system

1.3 Physical Architecture of Multi-Agent System:

1.3.1 Physical Architecture Definition:

The proposed logical architecture does not include any implementation details. One solution could be to implement it directly i.e. each agent of the logical architecture then is implemented as an agent in some agent development platform, like JADE, JADEX or JACK [4,5].

Still, if the logical architecture is implemented without any additional components, all interactions among agents and robots must be enclosed in agents. The main problem of such solution is usage of different data structures in the robots and agents. While robots use simple bytes of information, the agents use various knowledge representation structures like concepts and predicates. Mapping among these two encodings must be done. Additionally, interfaces with robots are usually programmed in different technologies. Agents as a rule are implemented in Java based platforms [5,6,7] while interfaces with robots are implemented using other languages and platforms, for example MS Robotics Developer Studio [8].

The middleware component is introduced to deal with these problems. So, the physical architecture consists of three layers. The higher layer or deliberative layer is realized as a multi-agent system that is responsible for high level decisions like task decomposition and assignment to robots. The second layer or the middleware layer consists of the middleware server that can be implemented in any technology. The main function of this layer is to carry out communications between robots and agents, including the mapping among the data structures used in the higher and lower levels. There are no direct physical communications among agents and robots. All communications among agents and robots are dispatched by the middleware server. The third or lower layer includes robots. The overall physical architecture of the system consisting of three layers is shown in the **Figure 1.2**. The two higher levels that correspond to the management system and interfaces among layers are described in detail in the following subsections.

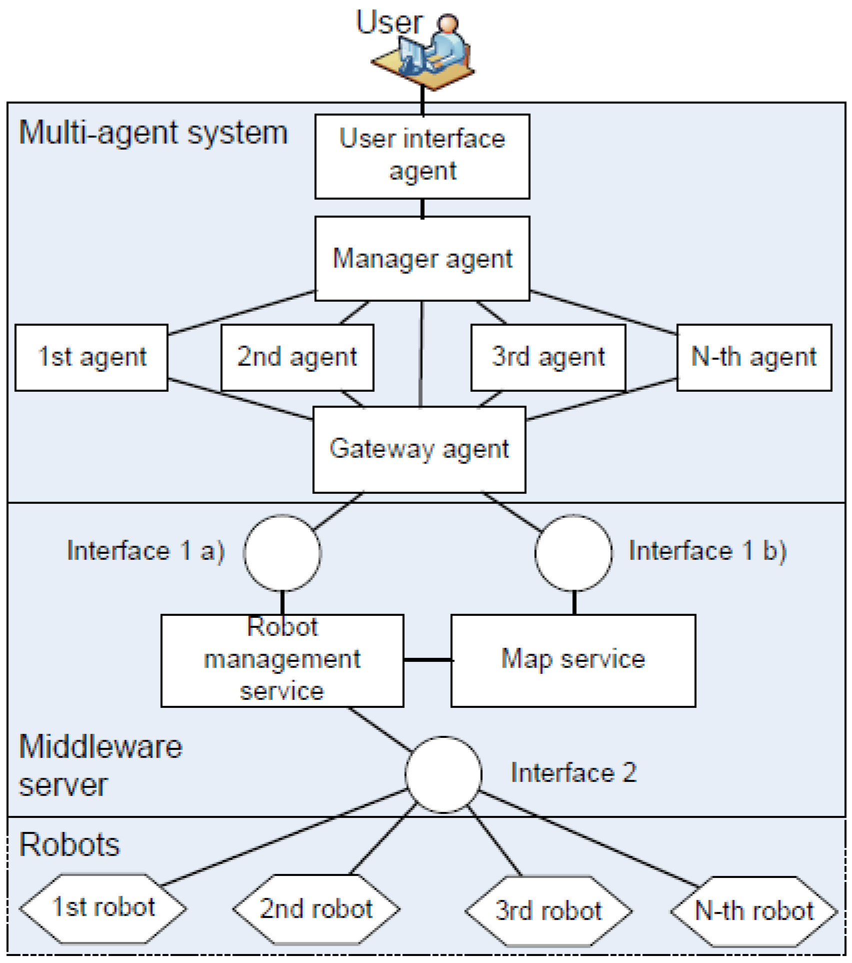


Fig. 1.2 The physical architecture

1.3.2 The multi-agent system:

The main idea of the multi-agent system in the physical architecture remains unchanged from the logical architecture given in **Figure 1.1**. Still, the user interface and the interface with the middleware server must be implemented. Two more agents are introduced in order to separate these interfaces from other components. These agents are introduced because of the following reasons.

Firstly, in case of direct implementation of the logical architecture all agents have to access services from the middleware server. So, all of them would have to implement the interface with the middleware server. Additionally, all agents must be notified about changes in the robots' states, because robot agents need to react to these changes and the manager agent needs to show all changes in the robots' states to the user. Therefore, the agents have to realize not only their responsibilities, but also interactions with the middleware server. To logically separate responsibilities we introduce a gateway agent. It is the only agent communicating with the middleware server. So, it implements interface with the server. All other agents instead of making direct requests to the middleware server send messages to the gateway agent that makes requests to the middleware server. This agent also is the only one that listens to the events that are posted by the robot management service. It informs the corresponding agents about the events. In case of large number of robots and agents the gateway agent may become a bottleneck in the communications among agents and middleware server, because it dispatches all communications. The architecture allows introducing more than one gateway agent in case if such a bottleneck appears.

Secondly, the manager agent of the logical architecture fulfils functions that can be grouped into two independent groups, namely the management of the user interface and task assignment. To keep agents responsible for single type of functions or one role we introduce the user interface agent. The manager agent keeps only the second group of functions (task decomposition and allocation). The user interface agent manages the user interface including the monitoring system that graphically shows statuses of all tasks and robots. It allows the user to follow the execution of tasks. Functions of agents are summarized in the **Table 1.1**.

Table 1.1 Function of agents in the Multi-Agent System

Agent	Functions
The user interface agent	Receive commands from the user and forward them to the manager agent. Monitoring system.
The manager agent	Task decomposition. Management of task allocation process. Task replanning in case of any failures. Reporting monitoring information to the user interface agent.
Robot agents	Task evaluation, to be able to generate proposals for the tasks. Participation in task allocation process. Creating commands for the robots. Monitoring the task execution and reacting on state changes. Reporting status of the task to the manager agent.
The gateway agent	Implementation of the service contracts. Responding to the events generated by the services of the middleware server. Calling services of the middleware server upon requests by other agents.

1.3.3 Middleware server:

The main function of the middleware server is to provide means for communications among the multi-agent system and robots. Additionally, in order to improve the system's overall performance, it is beneficial to collect incoming sensor data from all robots on the same map and ensure data availability to all robots. Here with term "map" we mean shared data structure among robots. Depending on actual mission it can be topological or metric data structure that stores mission specific data [9]. In any case it has to be up-to-date and available for all agents.

The middleware server is implemented in a service-oriented way [9]. The middleware functions of the server are included in one service, named robot management service, while all operations with the map are included in the second service, named map service. The latter creates the map using sensor data from robots. It must be capable of providing actual version of the map upon the request of agents. Additionally, it has to calculate distance and route between given locations.

Interface 1 shown in **Figure 1.2** consists of the service contracts. Interface 1 a) is the interface of the robot management service, while Interface 1 b) is the interface of the map service. Mainly interactions between agents and services are done in the way that agents call service methods, when they need some operations to be executed. Additionally, the robot management service raises events when there are any changes in the state of the robots. The interface agent is notified about all significant changes in robots' states. Event handling in the service environment is done as described in [8].

In order to monitor the robots' status, they need to have reporting functionality. One of our main goals is to create a multi robot system that is capable to accomplish its tasks using as little as possible of any additional infrastructure except the robots themselves. It implies two requirements. First, all data that are acquired about the environment come exclusively from the robots. Second, while the robot sensors are limited in the available information due to physical or functional limitations of the used sensors, the system needs to cope with limited data about the actual state of the environment. Under these assumptions we need the following information reported by robots using the Interface 2: (1) Command executed by the robot or idle state indication to be aware what tasks are being executed and progress of their execution. (2) Position and pose of the robot in terms of its coordinates and heading in any coordinate system that gives possibility to calculate the robot's actual location in the environment. Here authors abstract from the self-localization and positioning problems being irrelevant in the paper's context. (3) Error report indicating if the assignment accomplishment is being threatened by any detected circumstances. (4) Percepts of the robot that are used to build the map. For example, if the robot senses obstacle by its bumper or distance sensor it reports it together with its location and the map is updated accordingly. (5) Various robot and mission dependent data, like battery level, charging report, etc.

Robots report their statuses to the server at regular time intervals that may depend on the nature of the mission. The robot management service has to process data reported by the robots and forward percept information coupled with the position of the robot to the map building service. If there are significant changes in the position of the robot or any other information has been changed since the last report from the robot, its management service has to post an event to the multi agent system. The exact changes in the status of robots that need to be reported depend on the mission of the system. Additionally, the server has to identify loss of communications to the robots and report possible failure of the robot to the robot agent to deal with physical failures. Commands are sent directly to robots by the robot management service and contain the command and any arguments needed for it. Alternatively, plans can be sent to the robot. In this case every command contains a list of actions.

The proposed middleware server with the specified interfaces separates the higher layer from the lower one. It allows using the same management system with any robots that have the same Interface 2. Moreover, the robots can be substituted with some simulation environment that uses the same interface enabling experiments with virtual environment instead of real robots.

1.4 Applications of MAS:

MAS applications cover a variety of domains, including:

- aircraft maintenance
- industrial warehouses
- electronic book buying coalitions
- military demining
- wireless collaboration and communications
- military logistics planning
- supply-chain management
- joint mission planning
- financial portolio management
- Surveillance
- search and rescue missions

We are currently interested in Inter-Agent communication and coordination, and are building reusable multi-agent applications that facilitate interaction among different kinds of agent systems.

Chapter 2

System Components

2.1 Introduction:

Robot system designers must carefully consider each component of their design. The inclusion of sensors, actuators, or additional robots must be justified by contributing to efficient task completion. Components that do not directly contribute add cost without benefit. Communication is another component of multiagent robotic systems that merits careful consideration. The question is not simply whether or not to include inter-robot communication, but what type, speed, complexity and structure. How should these design decisions be made?

2.2 Software Components:

2.2.1 Web Server:

The aim of this part is to develop the Multi-Agent Website which assigns tasks to different agents in the system. In the Server, A web-based interface is developed for clients to interact with the system **Figure 2.1 (a)**.

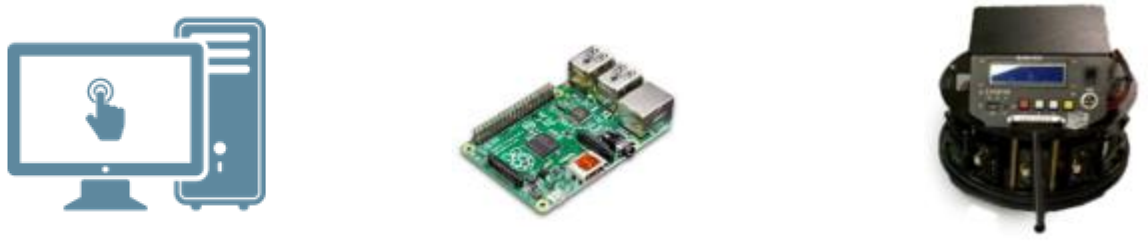
2.3 Hardware Components:

2.3.1 Raspberry pi 3 model (B):

It is the Middleware device between the server and the robots, the main function of it is to provide means for communications among the multi-agent system and robots. Additionally, in order to improve the system's overall performance **Figure 2.1 (b)**.

2.3.2 OMNI-DIRECTIONAL MOBILE ROBOT (ED-7275):

Robots execute subtasks assigned to them in a completely autonomous mode using their built-in algorithms. These robots may execute tasks individual or they could cooperate together to perform a specific task **Figure 2.1 (c)**.



(a)Server **(b)Raspberry pi 3** **(c)Mobile Robot**

Fig. 2.1 System Components

2.4 Project Architecture:

In this book, we present a general introduction to Multi-Agent Robotic Systems. This introduction is presented in **chapter 1** in which we described the general form of the logical architecture as shown in **Figure 1.1**, now for our specific project, the manger agent is a **web server** and the agents are the **raspberries** and the robots are **ED-7275 OMNI-DIRECTIONAL mobile robots** as shown in **Figure 2.2.**

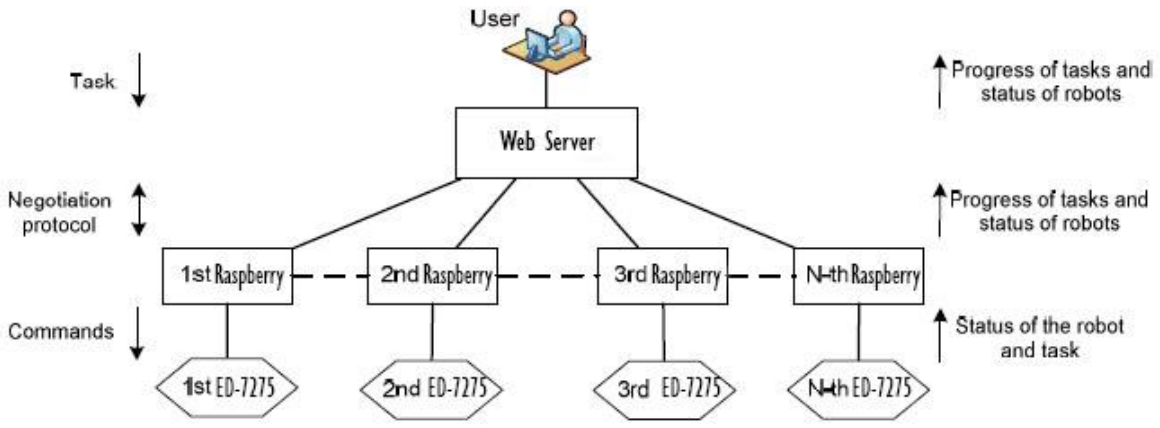


Fig. 2.2 The project architecture

2.5 The Situation to Improve:

Coordination of multi-agent systems has been regarded as a significant topic in research due to its importance in real world applications such as exploration and reconnaissance of a wide area, map building of unknown environments and cooperative transport of large objects. However, there are different challenges in multi-agent control which make it difficult to apply results proposed in research to get reliable products that can be used in industry. The first challenge is real time communication required between agents to perform their tasks. The second challenge is the difference between stability of single agent and that of the whole group of agents. Guaranteeing stability in the response of each agent does not guarantee stability of the multi-agent system. The third challenge is how to define the role of each agent in the system to complete the required task. Different approaches are used in research to develop robust multi-agent robotic systems. In this project, we adopt some of these approaches to apply them to a real-world application.

2.6 Solution:

In this project, we propose two modes of operation for system agents to perform a certain task. A task in our project means an object to be transported from a source location to a destination location avoiding any obstacles in its way as shown in **Figure 2.3 (a)**. any user could assign any number of tasks to the system which assign these tasks to the best robots to do these tasks as shown in **Figure 2.3 (b)**. In the first mode, every individual agent is assigned a task to perform. In this mode, the role of the system is to choose which robot to perform the required task according to a task allocation policy that guaranteeing object transportation with least energy consumption. In the second mode, agents cooperate together to perform a task that can't be performed by an individual agent. In this mode, a formation control algorithm of multi-agent system is adopted to move the group in a certain formation that allows object manipulation.

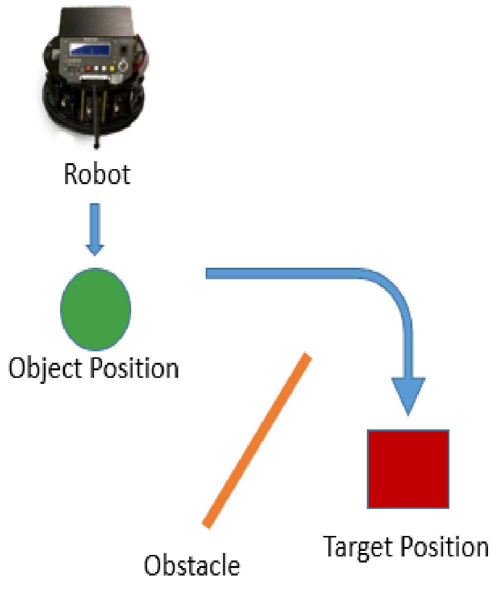


Fig. 2.3 (a) Motion Block diagram

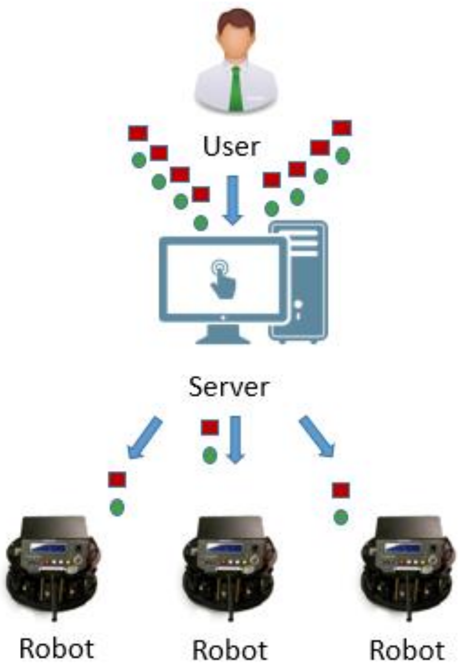


Fig. 2.3 (b) System Block diagram

2.7 Movement Approaches:

2.7.1 Individual Response:

In this approach, each robot does a specific task individually. That each robot goes from an object position to a target position and they coordinate with each other to avoid the deadlock problem as shown in **Figure 2.4**.

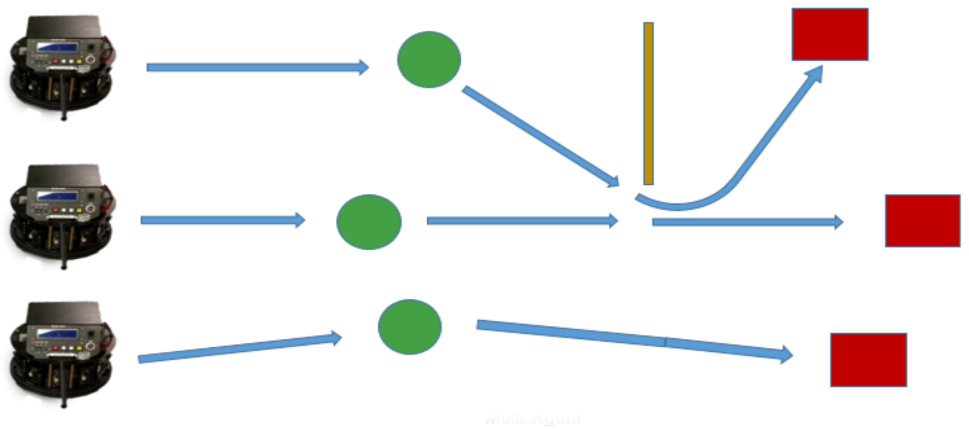


Fig. 2.4 Individual response

2.7.2 Cooperative Response:

In this approach, more than one robot cooperates together to perform a task that can't be performed by an individual agent, a formation control algorithm of multi-agent system is adopted to move the group in a certain formation that allows object manipulation as shown in **Figure 2.5**.

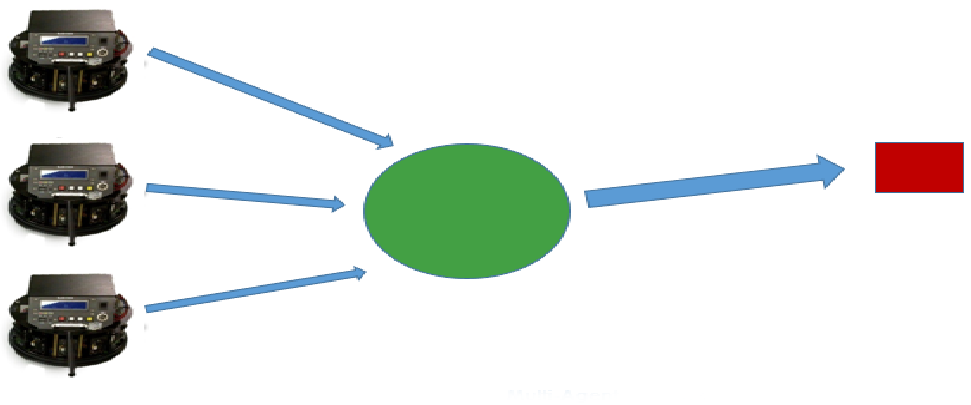


Fig. 2.5 Cooperative response

2.8 Robot Specification:

2.8.1 Robot System:

- Movement: Omni Directional Movement.
- Control: Embedded Micro Based Control.
- Max Driving speed: 50cm/s or so (Rated Driving Speed).

2.8.2 Control Part:

- MCU: ATmega128.
- Motor:
  - Available: 6 axes (Omni wheel 3 axis, External 3 axis).
  - Motor: Dc geared Motor.
  - Operation Mode: Closed Loop.
  - Encoder: Magnetic incremental encoder (26Pluse/ICycle).
- Battery:
  - 4S/1P (Lithium Polymer).
  - More than 80 min use (2Ah).
  - Less than 12 VDC motor voltage.
- Input & Output:
  - LCD: Blue LCD && 2 Lines x 20 character.
  - Switch: 4 input switches.
  - LED: Power LED &&2 User LED.
  - Analog Input: 8 Analog inputs for user.
  - Digital Input: 8 Digital inputs for user.

2.8.3 Instrument:

- Movement: Omni Directional Mobility.
- Number of driving wheels: 3 Omni Wheels.
- Outer Dimensions of: Diameter 255mm &&Height 250mm.
- External Shape: Cylindrical type.

2.8.4 Sensor:

- Floor Detecting System: 5 IR sensors.
- Forward-Detection & distance sensor.
    - 3 PSD Sensor Position front within 3 degrees Range.
    - Detected Distance From 10-80 cm.

2.8.5 Main Board Specs:

- Input voltage: 12 V (2A).
- Output voltage: 12v (User I/o output voltage).
- 2 * 20 Character LCD with Blue Back Light.
- 2 * 8 Digital I/O (8-bit isolation included).
- 8 Analog Input.
- Debug RS-232 Port.
- Max 6 axis DC Servo Motor Control.
- Two wires 12-bit Temperature Sensor.
- 12C Serial RTC (Real Time Clock).

2.9 Motor Controller (LM 629):

2.9.1 Description:

The LM629 are dedicated motion-control processors designed for use with a variety of DC and brushless DC servo motors, and other servomechanisms which provide a quadrature incremental position feedback signal. The parts perform the intensive, real-time computational tasks required for high performance digital motion control. The host control software interface is facilitated by a high-level command set. The LM628 has an 8-bit output which can drive either an 8-bit or a 12-bit DAC. The components required to build a servo system are reduced to the DC motor/actuator, an incremental encoder, a DAC, a power amplifier, and the LM628. An LM629-based system is similar, except that it provides an 8-bit PWM output for directly driving H-switches. The parts are fabricated in NMOS and packaged in a 28-pin dual in-line package or a SOIC-24 package (LM629 only). Both 6 MHz and 8 MHz maximum frequency versions are available with the suffixes -6 and -8, respectively, used to designate the versions. They incorporate an SDA core processor and cells designed by SDA.

2.9.2 Features:

- Holds a 32-bit Position velocity and acceleration register.
- Has a built-in digital PID filter that contains programmable 16-bit coefficient.
- Programmable derivative sampling interval.
- 8-bit sign-magnitude amplitude output data.
- Built-in 4*multiple up-down counter.
- Trapezoidal velocity generator.
- A velocity, target position and filter parameter are correctable during motion.
- Enable Operations in position mode and velocity mode.
- Real-time Programmable host interrupt.
- 8bit Parallel synchronous host interface.
- Built-in incremental encoder interface with index pulse input.
- Has one H-bridge Circuit built-in and supplies 3A current to motor.
- Has a flag out for warning temperature rises and a PIN.

2.10 Omni Wheels:

Omni-Directional wheels roll forward like normal wheels, but slide sideways with almost no friction (no skidding during turns). Use these wheels to make robot turn smoothly or build a holonomic drive train.

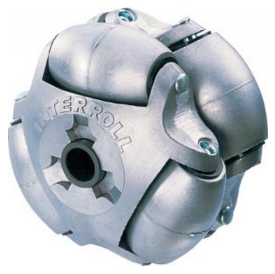


Fig. 2.6 (a)Omni Wheel

2.10.1 Assembling method:

- Drive in shaft with rubber hammer.
- Fix it with an instant adhesive if necessary.
- Push the shaft to end.
- Prevent omni wheel from being pushed out.

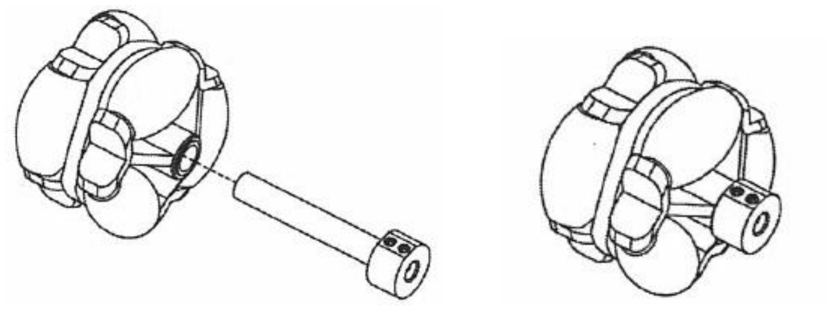


Fig. 2.6 (b)Omni Wheel

2.11 Motor:

2.11.1 Motor Specifications:

Table 2.1 Motor Specifications

#	Part Name	Appearance	Main Specifications	Remark
1	Motor	**Fig. 2.7 DC Motor**	• Rated Voltage 12 V DC. • Rated Torque 110[gf-cm]. • Rated Speed 5950 RPM. • Rated Current 900 mA. • No load speed 7300 RPM. • No load Current 150 mA. • Rated output 7w. • Humidity 20-85%.	Combination type of motor decelerator and encoder.
2	Decelerator	Same as above	• Combination of planet gear + bevel gear. • Reduction ratio 1/27. • Rated torque 1.7 kgf.cm. • Rated speed 220 RPM.	
3	Encode	Same as above	• Magnet/hall sensor mode. • Outside diameter 27.3mm. • Thickness 12.6mm. • Resolution 26 PPR.	

2.11.2 Gear motor characteristics:

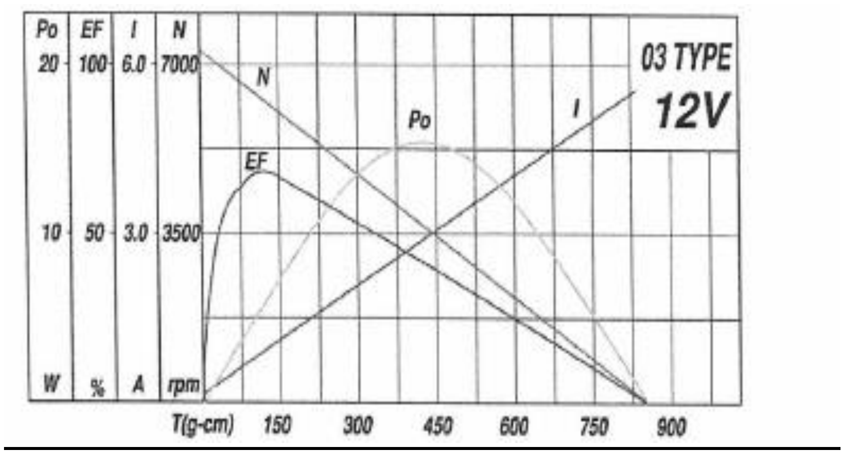
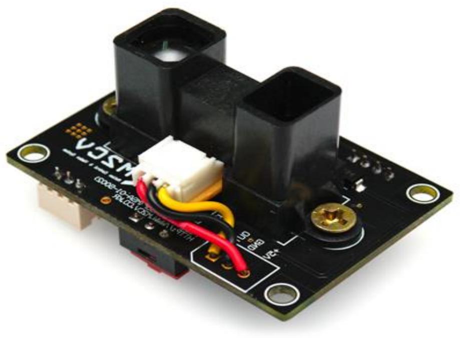


Fig. 2.8 Gear Motor Chart

2.12 PSD Sensor:

The PSD sensor has a separate sensor CPU within a sensor board, so the main CPU can obtain sensor data more accurately through the UART.



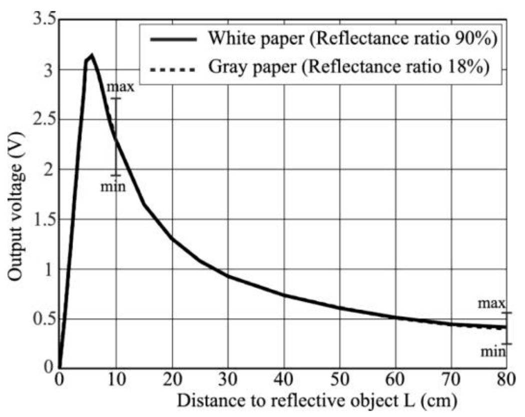

White paper (Reflectance ratio 90%)
Gray paper (Reflectance ratio 18%)
Output voltage (V)
Distance to reflective object L (cm)
max
min
max
min
0
0.5
1
1.5
2
2.5
3
3.5
0
10
20
30
40
50
60
70
80

Fig. 2.9 PSD Sensor

The PSD (Position Sensitive Detector) is a semiconductor that responds to light. Its scope of measurement amounts lOcm~80cm and measures a Distance by using a triangular method, which is the major PSD feature. A light-sending lens, light-receiving lens and distance to PSD reduce according to a proportional relationship of a lens system.

Therefore, a distance d to an object can be calculated. ED-7275 includes three PSD sensors basically, and is also designed to attach additional nine PSD, ultrasonic and pyro electricity sensors. It is unnecessary to change hardware additionally when attaching an additional sensor.

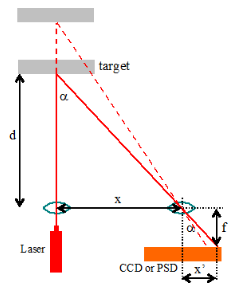


Fig. 2.10 PSD Calculation

2.13 Raspberry pi:

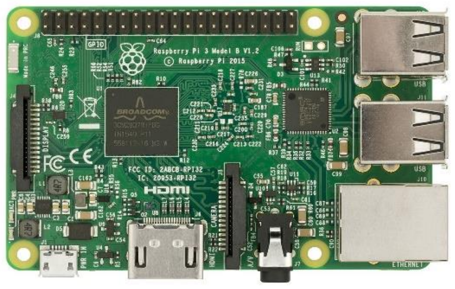


Fig. 2.11 Raspberry pi

A Raspberry Pi is a credit card-sized computer originally designed for education, inspired by the 1981 BBC Micro. Creator Eben Upton's goal was to create a low-cost device that would improve programming skills and hardware understanding at the pre-university level. But thanks to its small size and accessible price, it was quickly adopted by tinkerers, makers, and electronics enthusiasts for projects that require more than a basic microcontroller.

The Raspberry Pi is slower than a modern laptop or desktop but is still a complete Linux computer and can provide all the expected abilities that implies, at a low-power consumption level.

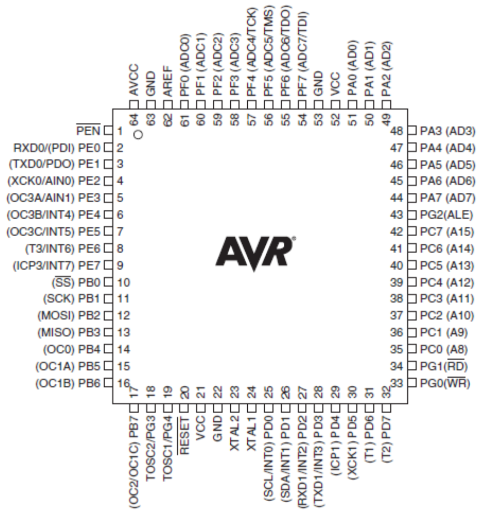
The Raspberry Pi is open hardware, with the exception of the primary chip on the Raspberry Pi, the Broadcom SOC (System on a Chip), which runs many of the main components of the board–CPU, graphics, memory, the USB controller, etc. Many of the projects made with a Raspberry Pi are open and well-documented as well and are things you can build and modify yourself.

2.14 Robot Microcontroller (ATmega128):

- High-performance, Low-power Atmel®AVR®8-bit Microcontroller.

- Advanced RISC Architecture.
    - 133 Powerful Instructions – Most Single Clock Cycle Execution.
    - 32 x 8 General Purpose Working Registers + Peripheral Control Registers.
    - Fully Static Operation.
    - Up to 16MIPS Throughput at 16MHz.
    - On-chip 2-cycle Multiplier.

- High Endurance Non-volatile Memory segments.
    - 128Kbytes of In-System Self-Programmable Flash program memory.
    - 4Kbytes EEPROM.
    - 4Kbytes Internal SRAM.
    - Write/Erase cycles: 10,000 Flash/100,000 EEPROM.
    - Data retention: 20 years at 85° C/100 years at 25° C (1).
    - In-System Programming by On-chip Boot Program True Read-While-Write.
    - Up to 64Kbytes Optional External Memory Space.
    - Programming Lock for Software Security.
    - SPI Interface for In-System Programming.



- QTouch® library support.
  - Capacitive touch buttons, sliders and wheel.
  - QTouch and QMatrix acquisition.
  - Up to 64 sense channels.

- JTAG (IEEE std. 1149.1 Compliant) Interface.
  - Boundary-scan Capabilities According to the JTAG Standard Extensive On-chip Debug Support.
  - Programming of Flash, EEPROM, Fuses and Lock Bits through the JTAG Interface.

- Peripheral Features.
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes.
  - Two Expanded 16-bit Timer/Counters with Separate Prescaler, Compare Mode and Capture Mode.
  - Real Time Counter with Separate Oscillator.
  - Two 8-bit PWM Channels.
  - 6 PWM Channels with Programmable Resolution from 2 to 16 Bits.
  - Output Compare Modulator.
  - 8-channel, 10-bit ADC.
  - 8 Single-ended Channels.
  - 7 Differential Channels.
  - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x.
  - Byte-oriented Two-wire Serial Interface.
  - Dual Programmable Serial USARTs.
  - Master/Slave SPI Serial Interface.
  - Programmable Watchdog Timer with On-chip Oscillator.
  - On-chip Analog Comparator.

- Special Microcontroller Features.
  - Power-on Reset and Programmable Brown-out Detection.
  - Internal Calibrated RC Oscillator.
  - External and Internal Interrupt Sources.
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power- down, Standby, and Extended Standby.
  - Software Selectable Clock Frequency.
  - ATmega103 Compatibility Mode Selected by a Fuse.

- ➢ Global Pull-up Disable.

- I/O and Packages.
  - ➢ 53 Programmable I/O Lines
  - ➢ 64-lead TQFP and 64-pad QFN/MLF

- Operating Voltages.
  - ➢ 2.7 - 5.5V ATmega128L.

- Speed Grades.
  - ➢ 0 - 8MHz ATmega128L.

2.15 Robot Programmer (AVRISP MKII):

2.15.1 Key Features:

- Programs both flash and EEPROM.
- Supports fuses and lock bit programming.
- Upgradeable for future device support.
- Supports target voltages from 1.8V to 5.5V.
- Adjustable ISP programming speed (50Hz to 8MHz SCK frequency).
- USB 2.0 compliant (full speed, 12Mbps).
- Powered from USB, does not require external.
- power supply.
- Target interface protection.
- Short-circuit protection.

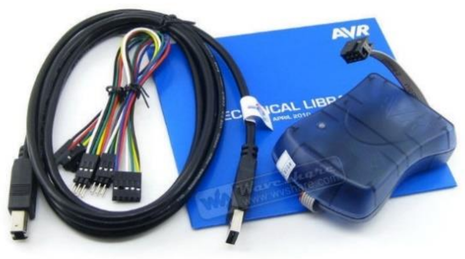


Fig 2.12 (a)AVRISP MKII

2.15.2 Connecting Atmel AVRISP mkII:

This section describes how to connect the Atmel AVRISP mkII to the host PC and the target device for correct operation. Note that Atmel Studio and the USB driver must be installed. See the Atmel Studio documentation for help. AVRISP mkII must be connected to the computer before connecting it to the target device.

Fig 2.12 (b)AVRISP MKII

When the AVRISP mkII is connected to the PC, and if the USB driver is installed, the green LED inside the AVRISP mkII close to the USB connector will be lit. The main status LED will be red before target is detected. After the AVRISP mkII is connected to the PC, it can be connected to the target. The red stripe on the target cable marks pin 1, and this should be mated with pin 1 on the ISP, PDI or TPI connector on the target board. See the section called "Target Interface" for a comparison of the different interfaces.

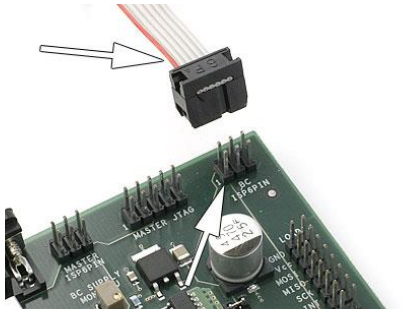


Fig 2.12 (c)AVRISP MKII

When the AVRISP mkII is connected to both the PC and the target board the main status LED should be green indicating that target power has been detected. AVRISP mkII is now ready to be used with Atmel Studio or the programming command line software.

Chapter 3

Robot Model and Control

3.1 Introduction:

Kinematics is the most basic study of how mechanical systems behave. In mobile robotics, we need to understand the mechanical behavior of the robot both in order to design appropriate mobile robots for tasks and to understand how to create control software for an instance of mobile robot hardware. Of course, mobile robots are not the first complex mechanical systems to require such analysis. Robot manipulators have been the subject of intensive study for more than thirty years. In some ways, manipulator robots have been much more complex than early mobile robots: a standard welding robot may have five or more joints, whereas early mobile robots were simple differential drive machines. In recent years, the robotics community has achieved a fairly complete understanding of the kinematics and even the dynamics (i.e. relating to force and mass) of robot manipulators.

The mobile robotics community poses many of the same kinematic questions as the robot manipulator community. A manipulator robot’s workspace is crucial because it defines the range of possible positions that can be achieved by its end-effector relative to its fixture to the environment. A mobile robot’s workspace is equally important because it defines the range of possible poses that the mobile robot can achieve in its environment. The robot arm’s controllability defines the manner in which active engagement of motors can be used to move from pose to pose in the workspace. Similarly, a mobile robot’s controllability defines possible paths and trajectories in its workspace. Robot dynamics places additional constraints on workspace and trajectory due to mass and force considerations. The mobile robot is also limited by dynamics; for instance, a high center of gravity limits the practical turning radius of a fast, car-like robot because of the danger of rolling. But the chief difference between a mobile robot and a manipulator arm also introduces a significant challenge for position estimation. A manipulator has one end fixed to the environment.

Measuring the position of an arm’s end effector is simply a matter of understanding the kinematics of the robot and measuring the position of all intermediate joints. The manipulator’s position is thus always computable by looking at current sensor data. But a mobile robot is a self-contained automaton that can wholly move with respect to its environment. There is no direct way to measure a mobile robot’s position instantaneously. Instead, one must integrate the motion of the robot over time. Add to this the inaccuracies of motion estimation due to slippage and it is clear that measuring a mobile robot’s position precisely is an extremely challenging task. The process of understanding the motions of a robot begins with the process of describing the contribution each wheel provides for motion. Each wheel has a role in enabling the whole robot to move. By the same token, each wheel also imposes constraints on the robot’s motion, for example refusing to skid laterally. In the following section, we introduce notation that allows expression of robot motion in a global reference frame as well as the robot’s 40 Autonomous Mobile Robots local reference frame. Then, using this notation, we demonstrate the construction of simple forward kinematic models of motion, describing how the robot as a whole moves as a function of its geometry and individual wheel behavior. Next, we formally describe the kinematic constraints of individual wheels, and then combine these kinematic constraints to express the whole robot's kinematic constraints. With these tools, one can evaluate the paths and trajectories that define the robot's maneuverability.

3.2 Kinematic model:

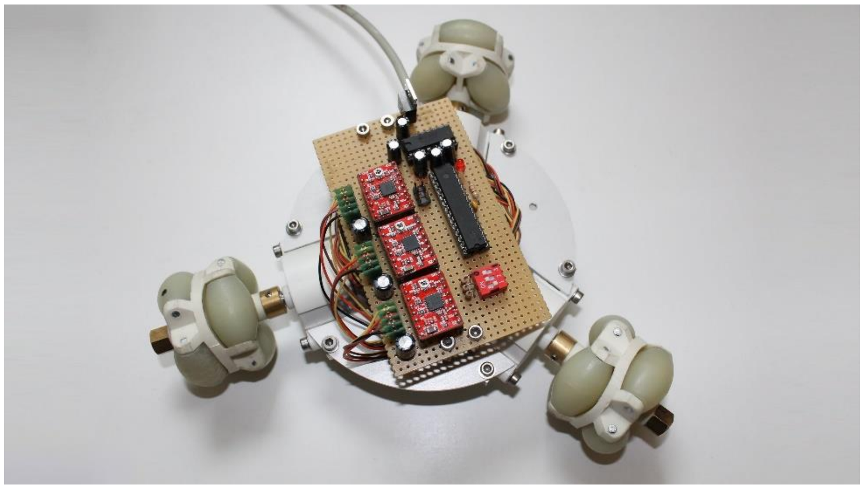


Fig. 3.1 A three-wheel omni-drive robot

Each individual wheel contributes to the robot's motion and, at the same time, imposes constraints on robot motion. Wheels are tied together based on robot chassis geometry, and therefore their constraints combine to form constraints on the overall motion of the robot chassis.

The omni-wheel robot show in **Figure 3.1** This robot has three Swedish 90 wheels, arranged radially symmetrically, with the rollers perpendicular to each main wheel. First, we must impose a specific local reference frame upon the robot. We do so by choosing point P at the center of the robot, then aligning the robot with the local reference frame such that is co-linear with the axis of wheel 2. **Figure 3.2** shows the robot and its local reference frame arranged in this manner.

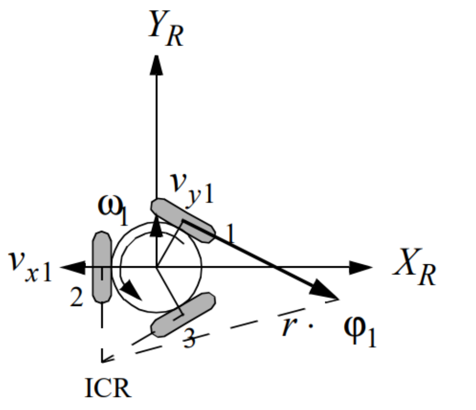


Fig. 3.2 The local reference frame plus detailed parameters

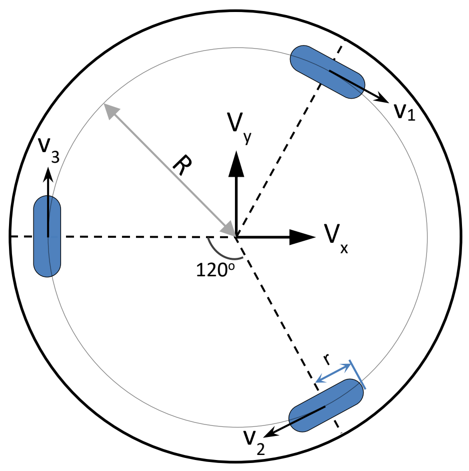


Fig 3.3 Robot frame and velocities

to get the inverse kinematics model, we assume V_R that is velocity of the robot Where:

$$V_R = \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix}$$

v_x is the velocity in the x-axis direction, v_y is the velocity in the y-axis direction and w is angular velocity, $w = \dot{\theta}$ AND:

$$V_{wheels} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Where v_n is the linear velocity of the nth wheel where $n = \{1,2,3\}$

Each wheel velocity has 3 components in v_x, v_y, w and the required is to get $R^{-1}$where:

$$V_{wheels} = R^{-1} * V_R$$

Let’s consider that if we analysis these velocities, we get:

$v_1 = \frac{\sqrt{3}}{2} v_x - \frac{1}{2} v_y + Rw$ eq(4.1)

$v_2 = v_y + Rw$ eq(4.2)

$v_3 = \frac{-\sqrt{3}}{2} v_x - \frac{1}{2} v_y + Rw$ eq(4.3)

Then from equations (4.1,4.2,4.3):

$$V_{wheels} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{-1}{2} & R \\ 0 & 1 & R \\ \frac{-\sqrt{3}}{2} & \frac{-1}{2} & R \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix} = R^{-1} * V_R$$

$$\dot{\theta}_{wheels} = \frac{V_{wheels}}{r} = \frac{1}{r} * \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{-1}{2} & R \\ 0 & 1 & R \\ \frac{-\sqrt{3}}{2} & \frac{-1}{2} & R \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix}$$

In some cases, we need to apply velocities with respect world frame so there is a rotation required:

$$V_R = \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix} = {}^{world}R_{robot}(z, \theta) * V_{world} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} v_{xworld} \\ v_{yworld} \\ w \end{bmatrix}$$

3.3 Control (Tracking):

The server sends a path of points to follow, this path must satisfy a condition that next point must be in any of these directions (forward, left, right, left with 45 degrees and right with 45 degrees) as shown in **Fig 3.5.**

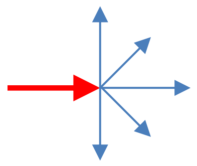


Fig 3.5 Available directions

in the single agent mode, an obstacle avoidance algorithm applied to avoid any dynamic obstacle, and this algorithm uses a sensor called PSD that sense distance and modify the path, this sensor is required to be in front of the moving robot, So the movement must be in one direction of Robot directions.

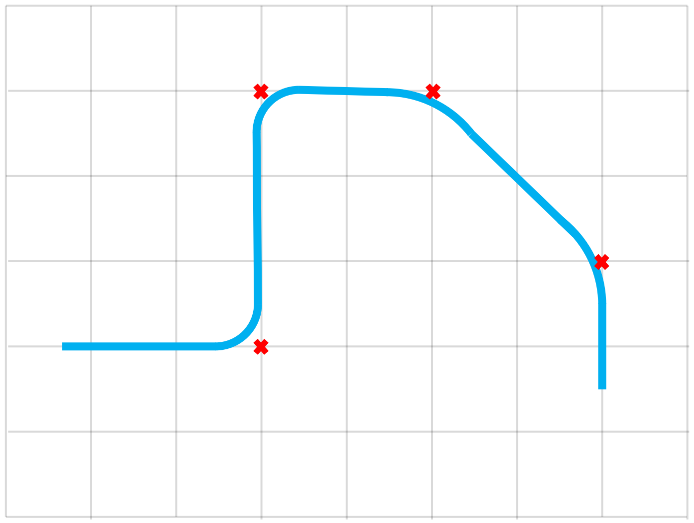


Fig 3.6 Adaptive path followed by robot

between Two adjacent points in the path the robot moves in the forward direction and before reaching next point little rotate with a linear velocity making a curve about the reached point to make the robot always move forward as shown in **Fig 3.6**.

To follow this model in controlling robot, the robot must be equipped in software with a model that take as input both the forward velocity v and angular velocity w

For each point in the path, we calculate the v, w, t for make the robot make a curve from the current direction toward the next point in the path and we have 4 cases as shown in **Fig 3.7**.

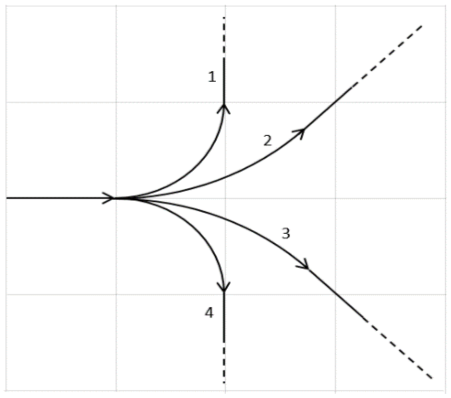


Fig 3.7 Four cases of rotation curve

there is a mathematical relationship between R, v, w and to prove this relation we see **Fig 3.8**.

$$dl = R * d\theta$$
$$\frac{dl}{dt} = R * \frac{d\theta}{dt}$$
$$\therefore v = R * w$$

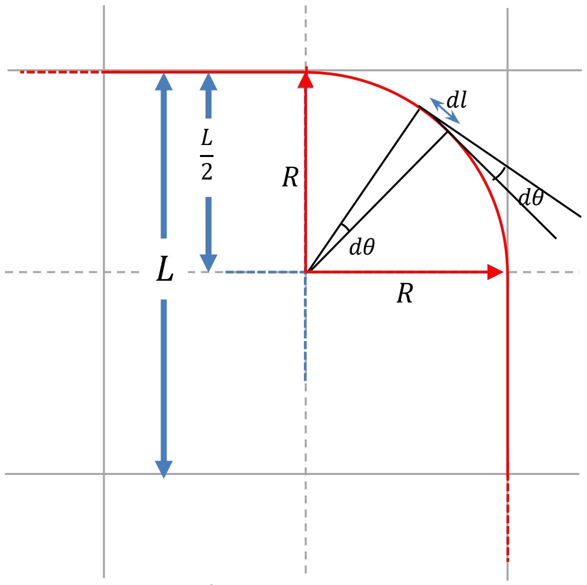


Fig 3.8 90º Curve

So, for each curve we need to find R and assume w to get. in **Fig 3.7** curves 1 and 4 have the same R which equal $\frac{L}{2}$ from **Fig 3.8** where L = the length of the cell used in the map. curves 2 and 3 have the same R which equal to $\frac{L/2}{\tan\frac{\pi}{8}}$ (see **Fig 3.9**)

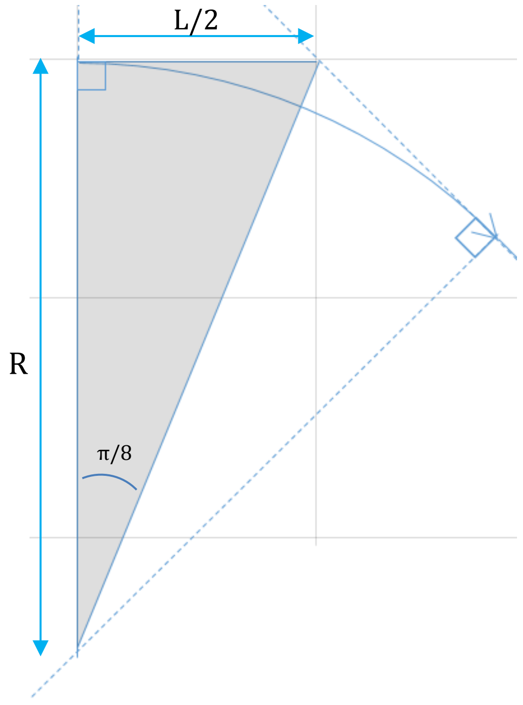


Fig 3.9 45º Curve

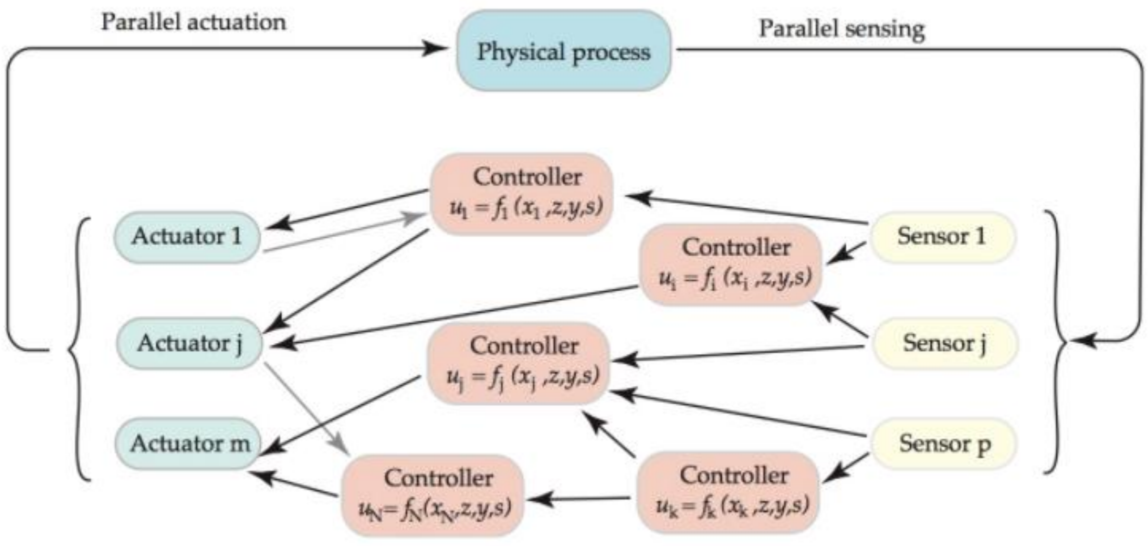
After calculating v, w from last step we get t of the curve by this equation $t = \frac{\theta}{w}$ where θ for both curves 1,4 equal 90 and for curves 2,3 equal 45.

For the forward movement, we tend to make the velocity constant (25 cm/s for example) and of course $w = 0$ and $t = \frac{d}{v}$ where d is the forward distance (distance between points – L).

After these calculations, we apply these velocities to the robot and consider the calculated time.

3.4 Cooperative Control:

Cooperative control [11,12] deals with the problem of controlling a multi-agent robotic system to fulfill a common goal and achieve group behaviors using only local information available to each agent from sensing or communication devices. Such local information may include relative configuration and motion obtained from sensing or communication between agents, agent's sensor measurements, and so on.



Parallel actuation
Physical process
Parallel sensing
Controller $u_1 = f_1(x_1, z, y, s)$
Controller $u_i = f_i(x_i, z, y, s)$
Controller $u_j = f_j(x_j, z, y, s)$
Controller $u_N = f_N(x_N, z, y, s)$
Controller $u_k = f_k(x_k, z, y, s)$
Actuator 1
Actuator j
Actuator m
Sensor 1
Sensor j
Sensor p

Fig. 3.10 General Cooperative control architecture

There is no direct mechanical link between pairs of agents, but rather some wireless sensing and communication links between certain assigned pairs.

The relative sensing and communication dictates the architecture of information flow between agents. Thus, a cooperative system has four basic elements: agents, information topology, control algorithms governing the motion of the agents, and group objective that is difficult to achieve by an individual agent.

A major focus in cooperative control is formation stability, where the group objective is to stabilize the relative distances/positions between the agents to prescribed desired values. Formation maintenance finds natural applications in coverage control, drag force reduction, and space interferometry.

3.5 Types of cooperation:

The control theoretic studies on swarms are concerned with developing appropriate individual control laws for the agents/robots which will lead to achieving a desired swarm behavior for the given agent/robot and swarm settings.

3.5.1 Aggregation:

Cooperative collective behavior usually occurs in aggregated swarms in nature. The swarm moves towards regions with higher food/nutrient concentration (which can be referred to as favorable regions) and away from regions containing toxic or hazardous substances (which can be referred to as unfavorable regions).

These concepts can be easily extended to swarm robotic systems such that favorable regions represent targets or goals, whereas unfavorable regions represent threats or obstacles.

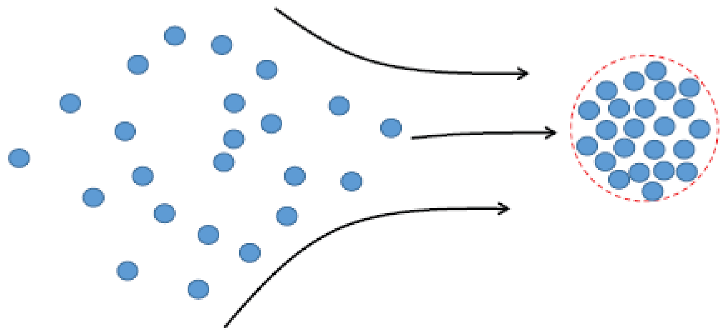


Fig. 3.11 Illustration of typical robot swarm coordination and control problems: Aggregation

3.5.2 Formation Control:

Formation of geometric patterns by robot swarms is an important engineering problem. Such behavior can be seen in swarms in nature during cooperative behaviors such as migration, object transportation, and others. The formation control problem can be divided into various stages such as formation acquisition, formation maintenance, and formation reconfiguration (see **Figure 3.12** for illustration) [13,14].

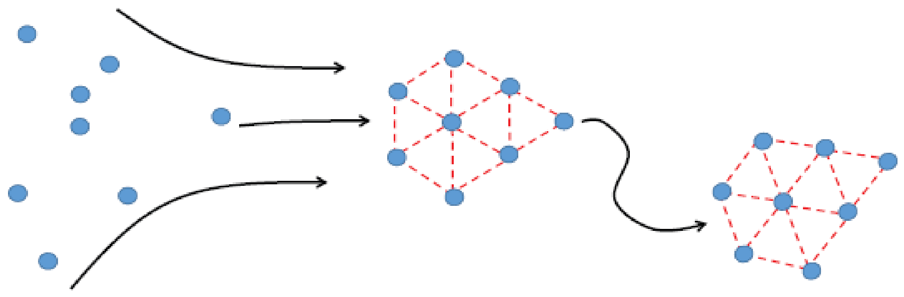


Fig. 3.13 Illustration of typical robot swarm coordination and control problems: Formation acquisition and maintenance

3.6 Problem formulation:

We consider an acyclic rigid formation with a group of mobile autonomous agents moving in a two-dimensional space. In the formation, the following properties are satisfied: (i) there is a leader that does not follow any other agents, it determines where the entire formation goes; (ii) each other agent of the formation follows the leader and maintains a relative distance towards the leader; (iii) the follower is responsible for maintaining a relative distance from the leader, and the leader does not perform any action to maintain the distance.

3.7 Control Design Approaches:

There are various approaches for high-level control or path planning and low-level agent control in multi-agent dynamic systems. It is possible to design high-level path planning or high-level control strategies without considering the individual agent dynamics, although taking them into account can be also beneficial. In contrast, low-level control of individual agents by its nature deals with the agent dynamics. Since the interest in multi-agent dynamic systems is in the collective behavior of the agents the inter-agent distances play an important role and are of particular interest. In the preceding sections, the swarm control problems are defined in terms of constraints on the inter-agent or agent extremum point distances. Therefore, the developed control strategies need to take into account the distance requirements on the agents.

3.8 Implementation:

In our project, we have three agents: one leader and two followers. The server sends the path, which the formation should track, to the leader to determine its velocities and send it with its position to the two followers that follow the leader with known formation but they don't have the path. Each follower determines its velocities based on the error in the distance between it and the leader in order to maintain a relative distance from the leader.

Here we consider a classification based on actuation constraints, and present Single-Integrator modeled Agent and non-holonomic (involving velocity constraints) agent models. Also with low speed, the acceleration dynamics can be removed and the models can be simplified to a kinematic agent models.

3.8.1 Single-Integrator modeled Agent:

Let $x_i(t) = [x_i \quad y_i]^T \in R^2$, $i = 1,\dots,n$ and $v_i(t) = [V_i^X \ ,V_i^Y\ ]^T \in R^2$, $i = 1,\dots,n$ denote the position and the velocity of agent i in the global coordinate system, respectively. The leader of the formation is active, and its motion equations are as follows [15]:

$$\dot{x}_1 = v_1 \qquad (3.1)$$

And the other followers' motion equations are as follows:

$$\dot{x}_2 = v_1 + k_p((\dot{x}_2 - \dot{x}_1) - (x_2 - x_1)) \qquad (3.2)$$

$$\dot{x}_3 = v_1 + k_p((\dot{x}_3 - \dot{x}_1) - (x_3 - x_1)) \qquad (3.3)$$

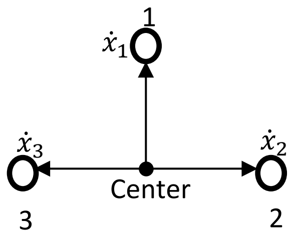


Fig. 3.14 The formation of Single-Integrator modelled agents

Where $\dot{x}_1, \dot{x}_2, \dot{x}_3$ represent the formation pattern positions and x_1, x_2, x_3 are the robot positions also k_p is a proportional gain.

3.8.2 Non-Holonomic Agent:

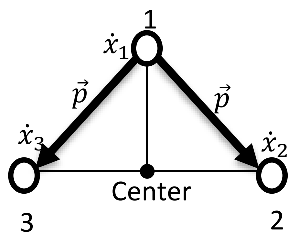


Fig. 3.15 The formation of Non-Holonomic agents

The leader of the formation is active, and its motion equations are as follows:

$$v_l = \frac{\sqrt{\Delta x_l^2 + \Delta y_l^2}}{\Delta t} \tag{3.4}$$

$$\omega_l = \dot{\theta}_l, \ \ \theta_l = tan^{-1}\frac{\Delta y_l}{\Delta x_l} \tag{3.5}$$

Where $\Delta x_l, \ \Delta y_l$ are the errors between leader's desired pose and current pose. And each of the other followers has a position vector $\vec{p} = [p_x \quad p_y]^T$ w.r.t the leader. The other followers' desired position are as follows:

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} cos\,\theta_f & -sin\,\theta_f \\ sin\,\theta_f & cos\,\theta_f \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} x_l \\ y_l \end{bmatrix} \tag{3.6}$$

Where x_f, y_f represent the desired pose of the followers and $x_l, and\ y_l$ represent the actual pose of the leader. And their motion equations are:

$$v_f = \frac{\sqrt{\Delta x_f^2 + \Delta y_f^2}}{\Delta t} \tag{3.7}$$

$$\omega_f = \dot{\theta}_f, \ \ \theta_f = tan^{-1}\frac{\Delta y_f}{\Delta x_f} \tag{3.8}$$

Where $\Delta x_f, \Delta y_f$ are the error between follower's desired pose and current pose.

Chapter 4

Web Server and System Communication

4.1 Introduction:

In this chapter, we will talk about the communication of a client with the robots when he wants to perform the certain task by sending http request through a web browser to the server and waiting for a response back and simultaneously focus on the phases of development a multi agent web application in the java framework. And describes each application’s architecture and sub-architecture their associated interfaces, database schemas, and the considerations behind the chosen design.

The core functionality in the following parts is to develop a multi-agent web application in a java framework. And to store information of robots, tasks and users of it, assign a new task, register a new robot and can edit this information through sending http request to the server.

4.2 Web Server Description:

The aim of this part is the development of a Multi-Agent Website. This website has to store information of robots, tasks and users of it. Each user has the ability to assign a new task, register a new robot and can edit these information through sending http request to the server, the http is a protocol using by the web-based application to facilitate the communication with server side. **Figure 4.1**

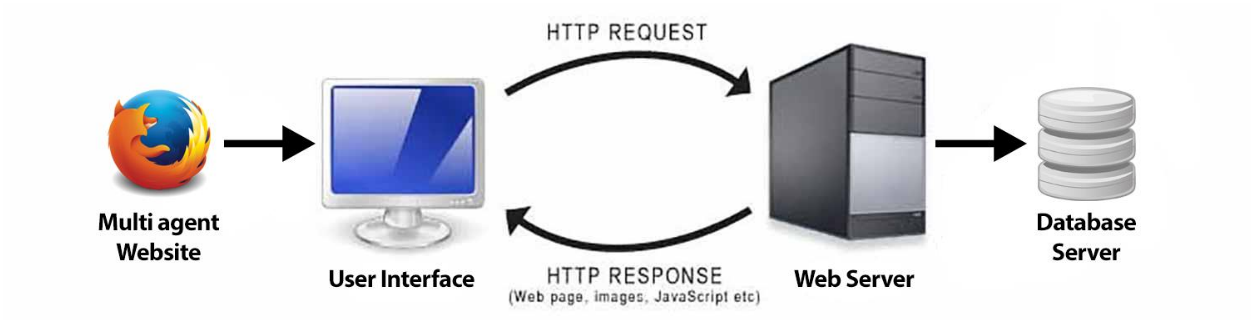


Fig. 4.1 High-level System Description

4.3 Requirements and Objectives:

4.3.1 Requirements:

- Eclipse IDE: Java editor.
- MySQL Query browser: This SQL tool enables an easy way to change the database without an application.
- Apache Tomcat 8.5

4.3.2 Objectives:

The aim of development of this web application is to make the connection between user and robot much easier and faster in performance and time. Here, the user does not need to have an experience or knowledge about the robot or how it will be doing the task. All he does open a browser and assign task or add a robot or a user and so on.

4.4 Server Architecture:

In this context, we describe the phases of the design and development. These phases are categorized into 3 main steps: Database, Application and Graphic User Interface.

4.4.1 Database Architecture:

This section illustrates the architecture of the database using an Entity Relationship Diagram (ERD) and a Relational Schema Definition. The ERD shows the overall structure and communication in the database **Figure 4.2(a).** The Relational Schema Definition describes the tables to be created in the database. In general, it gives the same information as the ERD but in a more specific way. **Figure 4.2(b).** A robot has the availability and status attributes which use in assigning task and trace it. The attributes of each entity are given in a circle with the primary keys. The relationships assigning, Tracing and execution connect the entities in a structured and simple manner [16,17].

The platform is a MySQL database with JDBC support. The database is built for the clerks and the managers of the website. The clerk of the databases must be able to fulfill the wishes of the client. These wishes include assigning tasks to many robots and trace the status of the task. This project team decided to implement the core functionality first and later to attach additional functions. The Core functionality is:

- Add, delete and update Robots information
- Add, delete, and update information about users of the website.
- Insert information about tasks into database during assigning the task.

The team’s minimum target is to show these core functionalities in a user friendly Graphic User Interface (GUI). This consideration is taken into the architecture of the database, which tries to ensure a built-on architecture. Additional functionality has to be integrated in an easy way.

Examples of these additional features can be:

- Overviews of the entities Robot, Task and User.
- Error checking of the application.
- Possibility to add new robot information into database and use it in tasks.
- Possibility to add new users that can assign tasks.
- Possibility to assign new tasks and trace the status of the task.

The creation of the tables is made by using the query language SQL. Two examples of the code are included below:

```
CREATE TABLE `task` (
  `T_ID` int(11) NOT NULL AUTO_INCREMENT,
  `user` int(11) DEFAULT NULL,
  `taskload` float DEFAULT NULL,
  `startX` int(11) DEFAULT NULL,
  `startY` int(11) DEFAULT NULL,
  `endX` int(11) DEFAULT NULL,
  `endY` int(11) DEFAULT NULL,
  `taskname` varchar(45) DEFAULT NULL,
  `taskAvailability` int(11) DEFAULT NULL,
  PRIMARY KEY (`T_ID`),
  KEY `user_idx` (`user`),
  CONSTRAINT `user` FOREIGN KEY (`user`) REFERENCES `user` (`userID`) ON DELETE CASCADE
ON UPDATE CASCADE)
```

While creating the tables using the Relational schema definition for each attribute a variable type is given. All primary keys of the attributes are generated as data type SERIAL. This means that for each entry of the database an ID is generated automatically. Reason is to ensure that all ids do not have to be generated by the user or manager of the website as well as an overall stability.

Mapping cardinalities are illustrated as arrows and explained below:

- One user can assign many tasks.
- One user can trace many tasks.
- Many robots can execute many tasks.

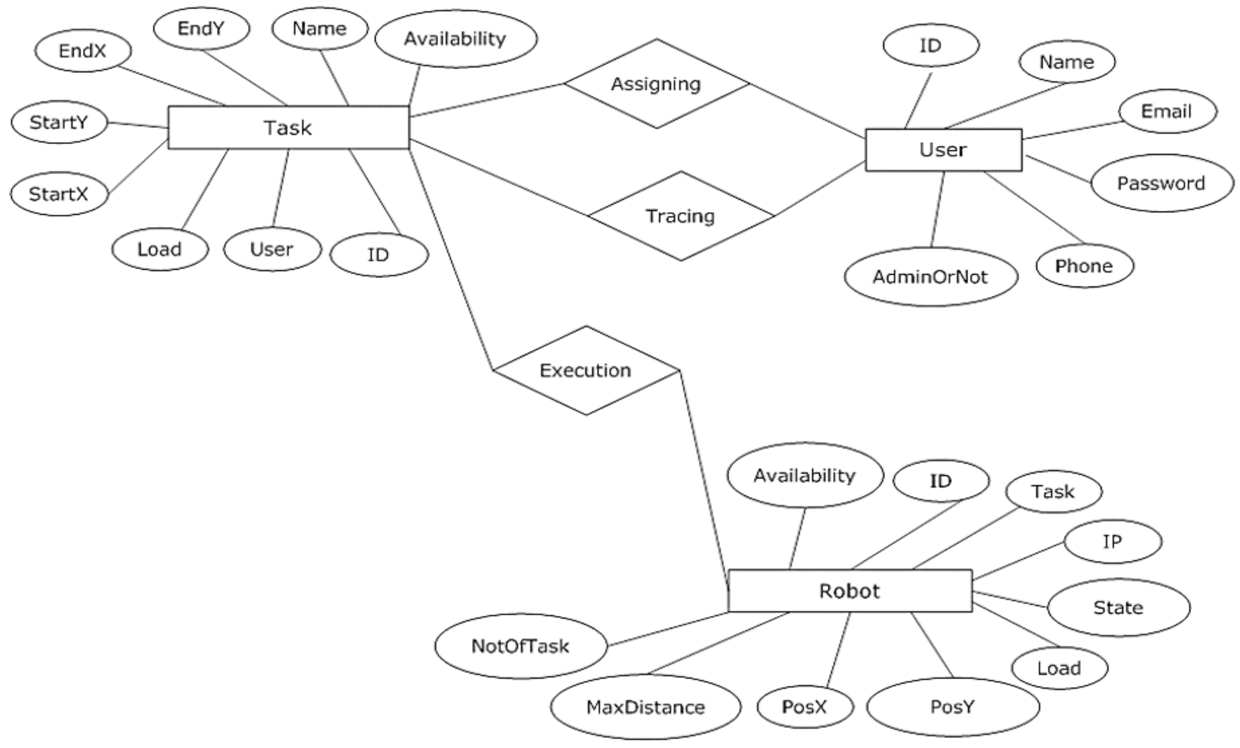
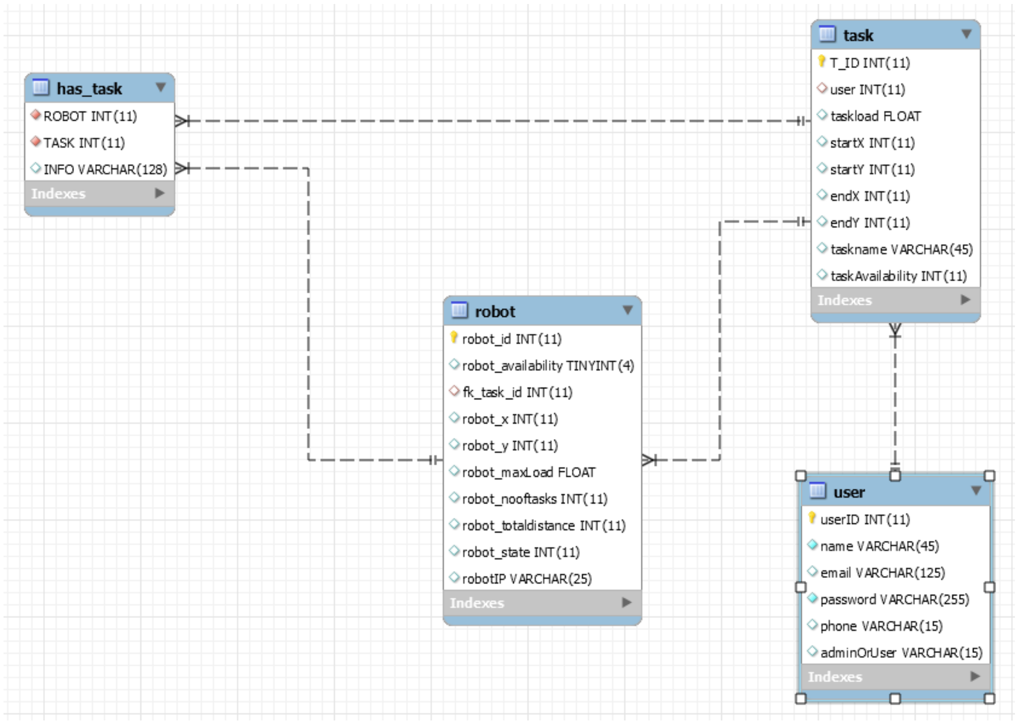


Fig. 4.2(a) Entity Relationship Diagram


has_task
ROBOT INT(11)
TASK INT(11)
INFO VARCHAR(128)
Indexes

task
T_ID INT(11)
user INT(11)
taskload FLOAT
startX INT(11)
startY INT(11)
endX INT(11)
endY INT(11)
taskname VARCHAR(45)
taskAvailability INT(11)
Indexes

robot
robot_id INT(11)
robot_availability TINYINT(4)
fk_task_id INT(11)
robot_x INT(11)
robot_y INT(11)
robot_maxLoad FLOAT
robot_nooftasks INT(11)
robot_totaldistance INT(11)
robot_state INT(11)
robotIP VARCHAR(25)
Indexes

user
userID INT(11)
name VARCHAR(45)
email VARCHAR(125)
password VARCHAR(255)
phone VARCHAR(15)
adminOrUser VARCHAR(15)
Indexes

Fig. 4.2(b) Relational Schema Definition

4.4.2 Application Architecture:

The architecture of the Multi-Agent website uses a 3-tiers MVC (model-view-controller) design pattern. Using this pattern, the application is divided into three modules, every module has own function. design and implement a web-based application. These tiers describe as shown in **Figure 4.3.**

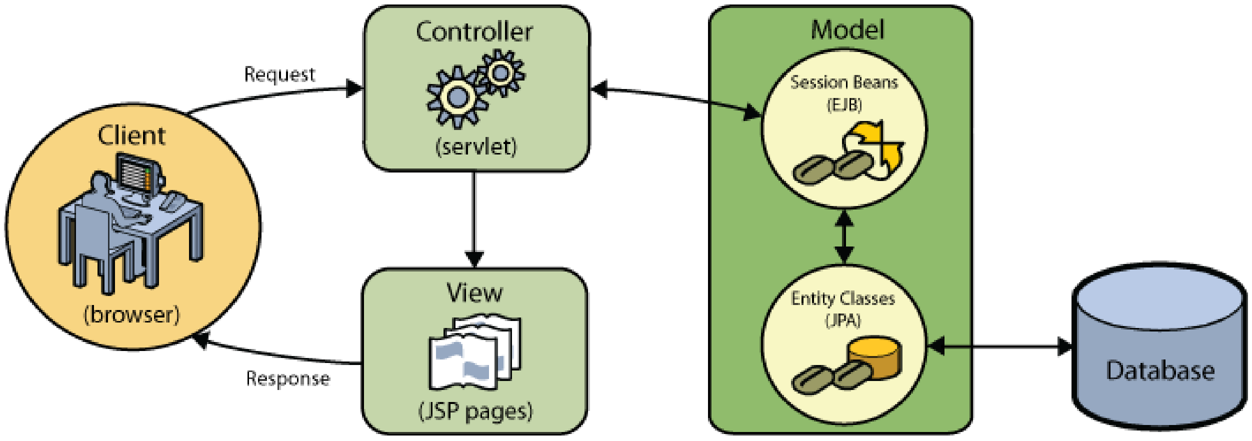


Fig. 4.3 Application Structure Diagram

The model: The model is the principal part of application program. It expresses (Java beans) business data, it is clearly said that one model is a record in database. This makes possible to access the database and create, update or view the data stored. Further explanation is provided in **section 1** related to the Functionality design. An example of the code to connect to the database is given below:

```
try{
 Class.forName("com.mysql.jdbc.Driver");
 myConnection = DriverManager.getConnection( "jdbc:mysql://123.156.5.73/dbpc
t6","username", "password" );
}
catch(Exception e)
{
System.out.println("Failed to get connection");
 e.printStackTrace();
}
```

It is independent of the data form, in other words one model may provide the data for many views, in this way it reduces the repeated code for our application program.

The view: The view is an interrelated part of User Interface (UI) in application program and a seeing and exchanging interface by users. One of advantages is that it can process many different views for the application program using MVC. It consists of JSP pages which definition as:

JSP technology: is a new dynamic web application technology standard. JSP page is composed of traditional HTML web page files (*.html, *.htm), which are inserted Java program files(Scriptlet) and JSP tags. Consequently, it comes into being a dynamic page on the server according to the client request. Also, we used CSS and JSTL to design web pages. We put JSP pages in /multiagentWebApp/WebContent/WEB-INF and some pages kept in /multiagentWebApp/WebContent/WEB-INF/views directories.

Servlet: is a small Java program on the server side and it must realize HttpServlet interface. It can respond and deal with client request, even it can bring a dynamic HTML page. The main components of the servlet as follows:

- init () – the init () function is called when the servlet is initialized by the server. This often happens on the first doGet () or doPut () call of the servlet.
- destroy () – this function is called when the servlet is being destroyed by the server, typically when the server process is being stopped.
- doGet () – the doGet () function is called when the servlet is called via an HTTP GET.
- doPost () – the doPost () function is called when the servlet is called via an HTTP POST. Posts are good way to get input from HTML forms.

The controller: The controller controls UI data displaying and updates the model object state according to the users' input. The controller accepts the user input and calls the model and view to complete the user demand. So, when we click the hyperlinks in the web page and send HTML table list, the controller itself does not have any output and make any processing. It only receives the request and determines calling which model to deal with the request, and it confirms to use which view to display the returning data of the model processing. There is the relationship of the model, the view and the controller. Web server is an important part of the architecture. The main technologies are Servlet and JSP, its function implements UI. Because the essential of JSP and Servlet is a small Java program on the server side, it can exchange Java Applet and HTML with the client. Not only is implementing speed fast and is UI generation flexible, but also is the security very good, at the same time, it can realize some simple application logic [18,19].

4.5 Multi-agent Graphical User Interface:

Snap 1: This is the first page arrived when someone writes this link in their browser, and entered his name and password.

http://localhost:8080/multiagentWebApp/index.jsp

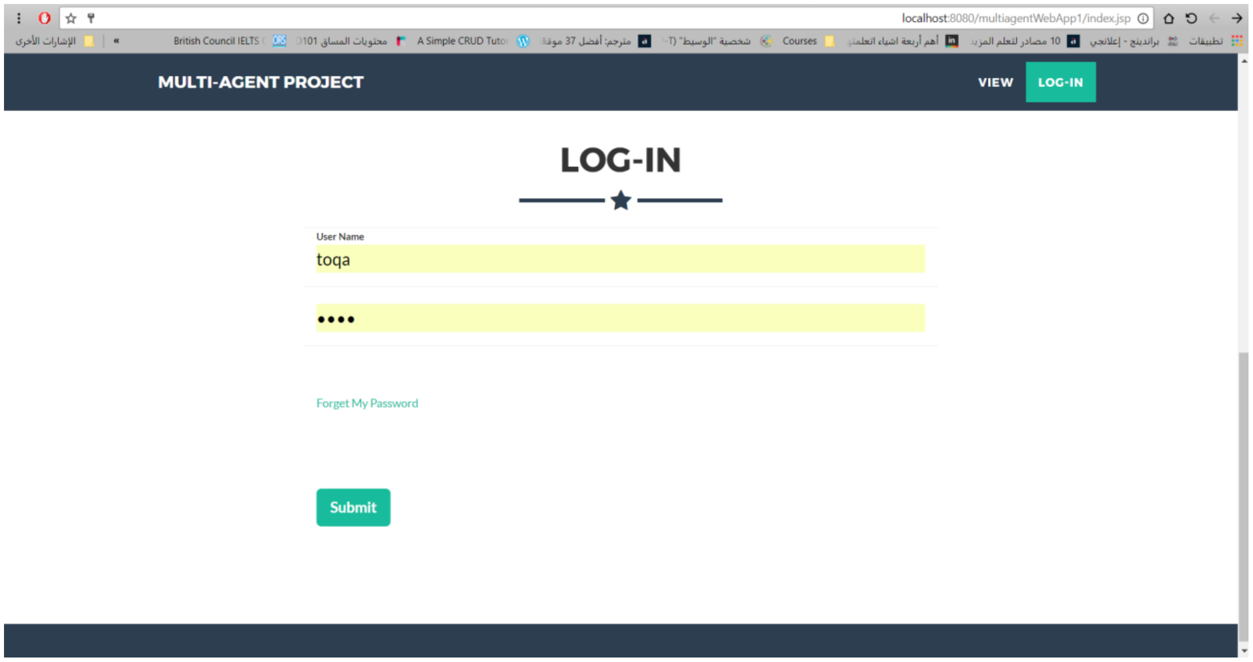


Fig. 4.4 (a)Web application user interface

Snap 2: This page will arrive when administrator login correctly and want to assign a new task

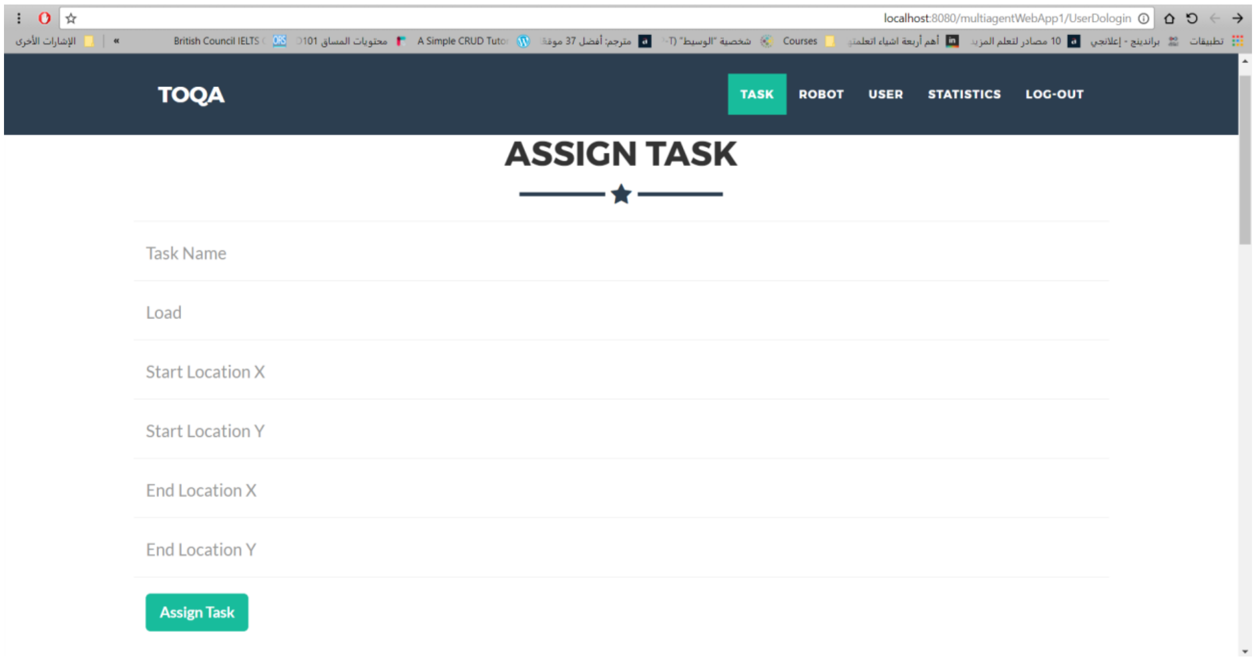
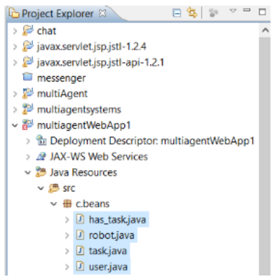


Fig. 4.4 (b)Web application user interface

4.6 Design Flow and Operation:

4.6.1 Design Flow:

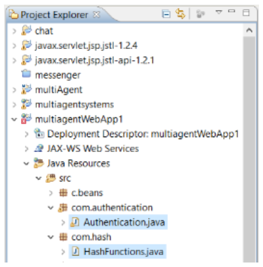
We will be going to explain the detailed design of this application. We create the Javabean classes to simulate the tables we construct in the Database. Javabeans are self-contained software units developed according to the motto **"Developed them once, run and reused them everywhere"**. Or in other words, reusability is the main concern behind the component model. The elements of a Javabean are properties, events and methods. In this application, we are using a method, is same as Java methods and every property should have get and set method. **Figure 5.5 (a)**



```
Project Explorer
> chat
> javax.servlet.jsp.jstl-1.2.4
> javax.servlet.jsp.jstl-api-1.2.1
  messenger
> multiAgent
> multiagentsystems
v multiagentWebApp1
  > Deployment Descriptor: multiagentWebApp1
  > JAX-WS Web Services
  v Java Resources
    v src
      v c.beans
        > has_task.java
        > robot.java
        > task.java
        > user.java
```

Fig. 4.5 (a) Javabean classes

Then, we have two packages each one of them implement a specified function in the web application. The authentication class is responsible of the validation of user and his password also, check if the username is unique or not, that is important to make sure every user registered and can be access the website. The second is a hashFuncion class, which provides a security property to the application and safe the information about users through hashing algorithm. **Figure 4.5 (b)**



```
Project Explorer
> chat
> javax.servlet.jsp.jstl-1.2.4
> javax.servlet.jsp.jstl-api-1.2.1
  messenger
> multiAgent
> multiagentsystems
v multiagentWebApp1
  > Deployment Descriptor: multiagentWebApp1
  > JAX-WS Web Services
  v Java Resources
    v src
      > c.beans
      v com.authentication
        > Authentication.java
      v com.hash
        > HashFunctions.java
```

Fig. 4.5 (b) Authentication and hash classes

4.6.2 Operation:

The website starts with the index page to perform the model of the login function. **Figure 4.6** Also, there is an option in case if the user forgets the password that enables to sit a new password with the name and email.

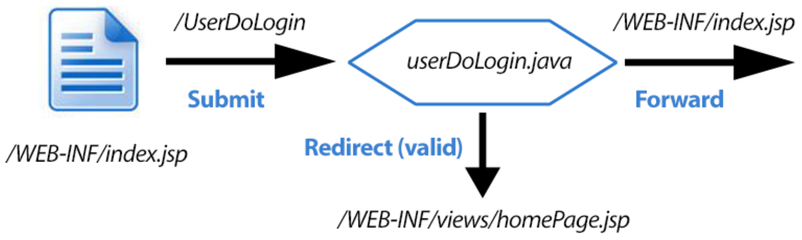


Fig. 4.6 Login Function

4.7 System Communication Infrastructure:

4.7.1 Network Architecture:

Network architecture used to perform communication between the server and the robot and between robots each other. The architecture allows all nodes of the system (Robots and Server) to communicate with each other using wireless technology. We use the infrastructure mode of networking which joins devices together via an access point. The concept of centralization (which is supported by infrastructure mode) makes the access point is a major part of the system.

4.7.2 Wireless Access Points (Wireless AP):

Wireless AP is fixed and provides service to the end systems within range. The end systems connect to the access point to join the network and they are in the same LAN. The AP enables the server to contact with the LAN's end systems making a WAN network.

4.7.3 The End Systems:

They are the devices which run application programs and want to contact with each other (Robots and Server). The robots are considered clients that contact with the server using HTTP protocol.

4.7.4 Protocols and ports:

- HTTP protocol (Port 80).
- SSH protocol (port 22).
- TCP protocol.
- IP protocol.
- WPA-2 for Protected Frame.

4.7.5 LAN:

It is between the robots themselves enables them to contact with each other achieving the concept of multi-agent. Each robot has a different IP in the same subnet (Private IP) and connected with each other using Wireless AP as **Figure 4.7**.

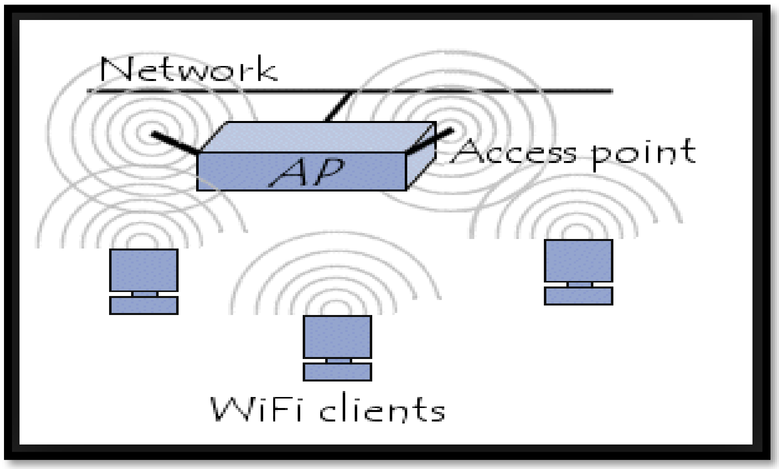


Fig. 4.7 Wireless LAN with 3 end systems

4.7.6 WAN:

It is between the server and each robot independently. The server uses a router or access point to contact with the robots achieving Client-to-Server model as **Figure 4.8**.

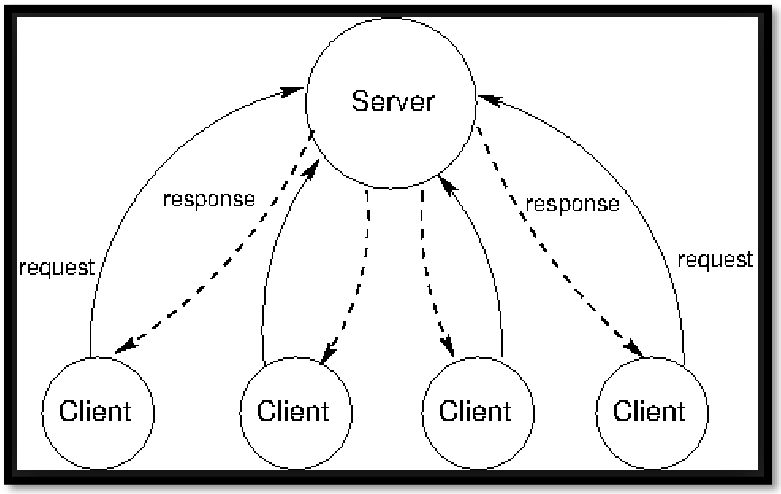


Fig. 4.8 Client-to-Server model

4.8 System Communication Platform:

Communication is an important component of multi-agent robotic systems that merits careful consideration. Multiple cooperating robots are able to complete many tasks more quickly and reliably than one robot alone. Communication between the robots can multiply their capabilities and effectiveness, the levels of communication are progressively more complex and potentially more expensive to implement. For some tasks, communication can significantly improve performance, but for others inter-agent communication is apparently unnecessary. In cases where communication helps, the lowest level of communication is almost as effective as the more complex type.

The question is not simply whether or not to include inter-robot communication, but what type, speed, complexity and structure. How should these design decisions be made?

Based on the type of the task, communication must be redesigned to acquire the maximum performance so we can divide our design into two levels.

1. **Level one:** Website- Agent communication.
2. **Level two:** Communication among the robots (inter-agent communication).

These two levels can be accomplished through 3 phases during the entire task process.

- **Phase 1:** moving from the parking point to the object.
- **Phase 2:** moving from the object to the goal point.
- **Phase 3:** moving from the goal point and return back to the parking point.

And we have accomplished that by choosing the java programming language because of its powerful multithreading and socket programming techniques.

4.8.1 Level one Website- Agent communication:

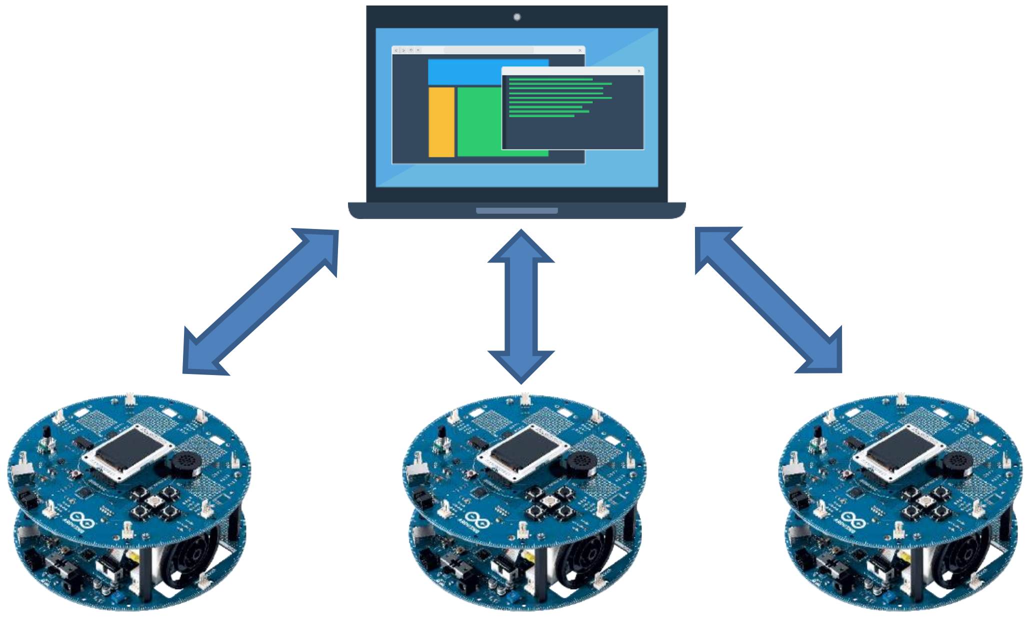


Fig. 4.9 Communication between the website and the robot

In this level, the Communication is just between the website and each one of the robots there is no communication among the robots themselves yet using client-server model which is a communication model for sharing the resource and provides the service to different machines as shown in **Figure 4.9**.

Server is the main system which provides the resources and different kind of services when client requests to use it and it is done through socket programming using TCP/IP as shown in **Figure 4.10**.

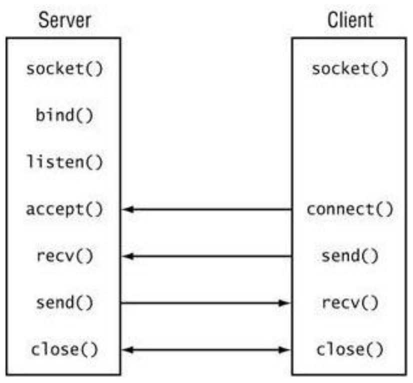


Fig. 4.10 Communication between the server and the client

Firstly, when each one of the robots is turned on, it works as a server and do the following:

- Create a socket for communication.
- Bind the local port and connection address.
- Configure TCP protocol with port number.
- Listen for client connection.
- Accept connection from client.
- Send Acknowledgement.
- Receive message from client.
- Send message to the client.

After that when the user of the website enters the required parameters for the task the website does the follows as a client:

- Create a socket for communication.
- Configure TCP protocol with IP address of server and port number.
- Connect with server through socket
- Wait for acknowledgement from server
- Send message to the server
- Receive message from server

After the robot has created the socket and bounded the port which will listen on to it, the robot waits for a client to connect in this case the client is the website, while the robot is waiting for a connection, the user of the website is entering the task parameters after he finishes and click the assign task button the website will start processing this data with path-planning and task-allocation algorithms which generates a path and sending it to the robot by creating a socket with the IP and port number determined by the robot.

Once the client has connected, the robot will start a separate thread for each connected client using this technique the robot can listen and replay to each connected client separately at the same time.

After the robot receives the path it starts moving towards the object point once it reaches the point it sends back a flag to the website informing it that it has finished this phase (phase 1) and waiting for the next one.

When all the robots reach the object then we start the more complex level.

4.9.2 Level two Inter-Agent Communication:

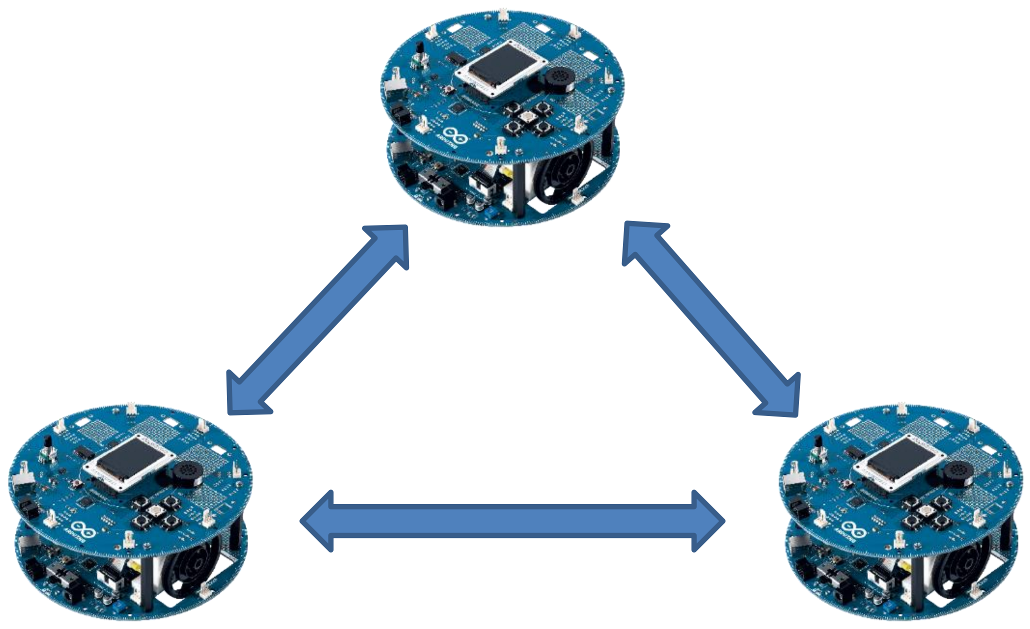


Fig. 4.11 Communication among the robots

At this point we start a more complex communication level which we add an inter-agent communication level to the previous one which the robots start talking to each other's in addition to talking to the website now the agents cannot move as a single agent anymore they need to know each other's position to every determined constant amount of time to adjust their position and speed based on the massages they get we accomplished that by the power of multithreading programming concept which we can start a thread for each connected client to the agent either it was another agent or the website and starts exchanging messages with it at the same time with no delay which is a crucial parameter in our design.

After the robots reached the object they stop and wait for the website to inform them who is the leader and who is a follower because depend on which type of the robot is a different thread will start.

If the robot has been informed that he is the leader he will starts the leader thread and waits for the path of the **second phase** from the website using the same communication of level 1 once it received the path it will start moving towards his goal point and at the same time sending its position to his followers every 200 milliseconds once he reaches his goal he stops sending his position to his followers and send a flag to the website to inform it that he finished the second phase.

Else if the robot has been informed that it is a follower then it will start a follower thread which will start waiting for the leader position every 200 milliseconds to start moving towards the goal point based on the leader position using a specific algorithm once it stop receiving a position feedback from the leader it knows that the leader has reached the goal point as well it did and send a flag to the website informing it that they finished the second phase too.

After all the robots had finished the second phase they all start to wait for the final path from the website using the same communication from level one and start moving towards their parking point as single agents again **phase 3**.

4.10 Low level Communication:

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the

Atmel AVR ATmega128 and peripheral devices or between several AVR devices. The

ATmega128 SPI includes the following features:

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

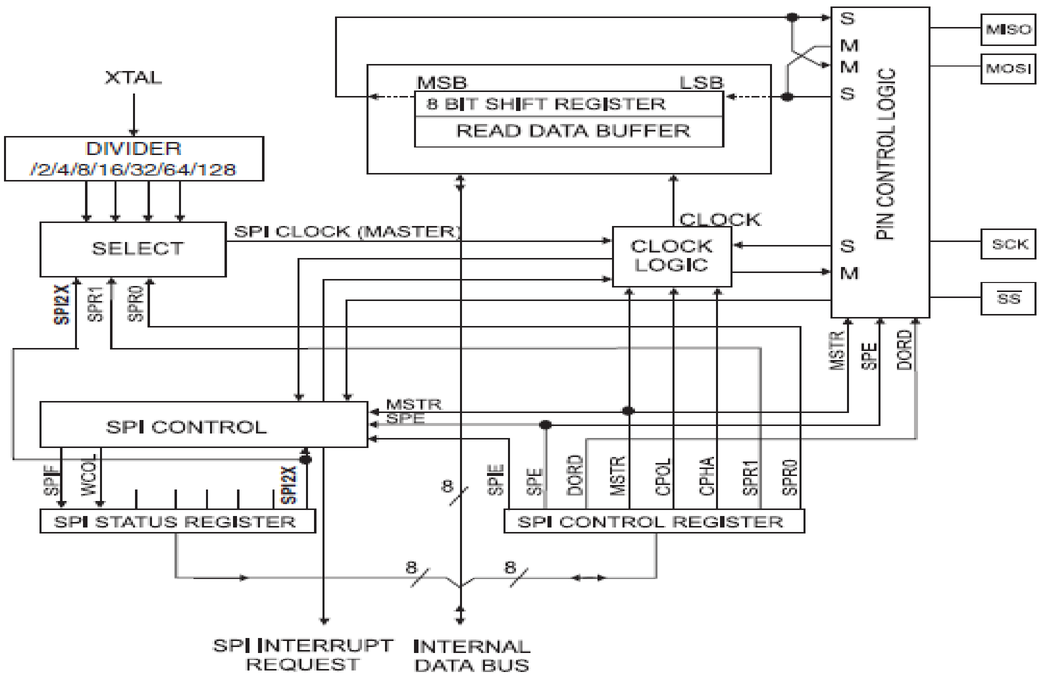


Fig. 4.12 SPI Communication

Chapter 5

Motion Planning

5.1 Introduction:

Robot navigation is the problem of guiding a robot towards a goal. The human approach to navigation is to make maps and erect signposts, and at first glance it seems obvious that robots should operate the same way. However, many robotic tasks can be achieved without any map at all, using an approach referred to as reactive navigation. For example: heading towards a light, following a white line on the ground, moving through a maze by following a wall, or vacuuming a room by following a random path. The robot is reacting directly to its environment: the intensity of the light, the relative position of the white line or contact with a wall. Today more than 5 million Roomba vacuum cleaners are cleaning floors without using any map of the rooms they work in. The robots work by making random moves and sensing only that they have made contact with an obstacle.

The more familiar human-style map-based navigation is used by more sophisticated robots. This approach supports more complex tasks but is itself more complex. It imposes a number of requirements, not the least of which is a map of the environment. It also requires that the robot’s position is always known

5.1.1 Reactive Navigation:

Surprisingly complex tasks can be performed by a robot even if it has no map and no real idea about where it is. As already mentioned robotic vacuum cleaners use only random motion and information from contact sensors to perform a complex task. Insects such as ants and bees gather food and return it to the nest based on input from their senses, they have far too few neurons to create any kind of mental map of the world and plan paths through it. Even single-celled organisms such as flagellate protozoa exhibited goal seeking behaviors. In this case we need to revise our earlier definition of a robot to “**a goal oriented machine that can sense, plan and act**” The manifestation of complex behaviors by simple organisms was of interest to early researchers in cybernetics.

5.1.2 Map-based planning:

The key to achieving the best path between points A and B, as we know from everyday life, is to use a map. Typically, best means the shortest distance but it may also include some penalty term or cost related to traversability which is how easy the terrain is to drive over – it might be quicker to travel further but over better roads. A more sophisticated planner might also consider the kinematics and dynamics of the vehicle and avoid paths that involve turns that are tighter than the vehicle can execute. Recalling our earlier definition of a robot as: **“a goal oriented machine that can sense, plan and act”** This section concentrates on planning.

There are many ways to represent a map and the position of the vehicle within the map. One approach is to represent the vehicle position as $(x, y) \in R^2$ and the drivable regions or obstacles as polygons, each comprising lists of vertices or edges. This is potentially a very compact format but determining potential collisions between the robot and obstacles may involve testing against long lists of edges. A simpler and very computer-friendly representation is the occupancy grid. As its name implies the world is treated as a grid of cells and each cell is marked as occupied or unoccupied. We use zero to indicate an unoccupied cell or free space where the robot can drive. A value of one indicates an occupied or non-drivable cell. The size of the cell depends on the application. The memory required to hold the occupancy grid increases with the spatial area represented and inversely with the cell size. However, for modern computers this representation is very feasible.

In this Project, the map is a dynamic map could be any map to fit the place where we use this multi-agent system and the map which we test the system in is shown in **Figure 5.1.**

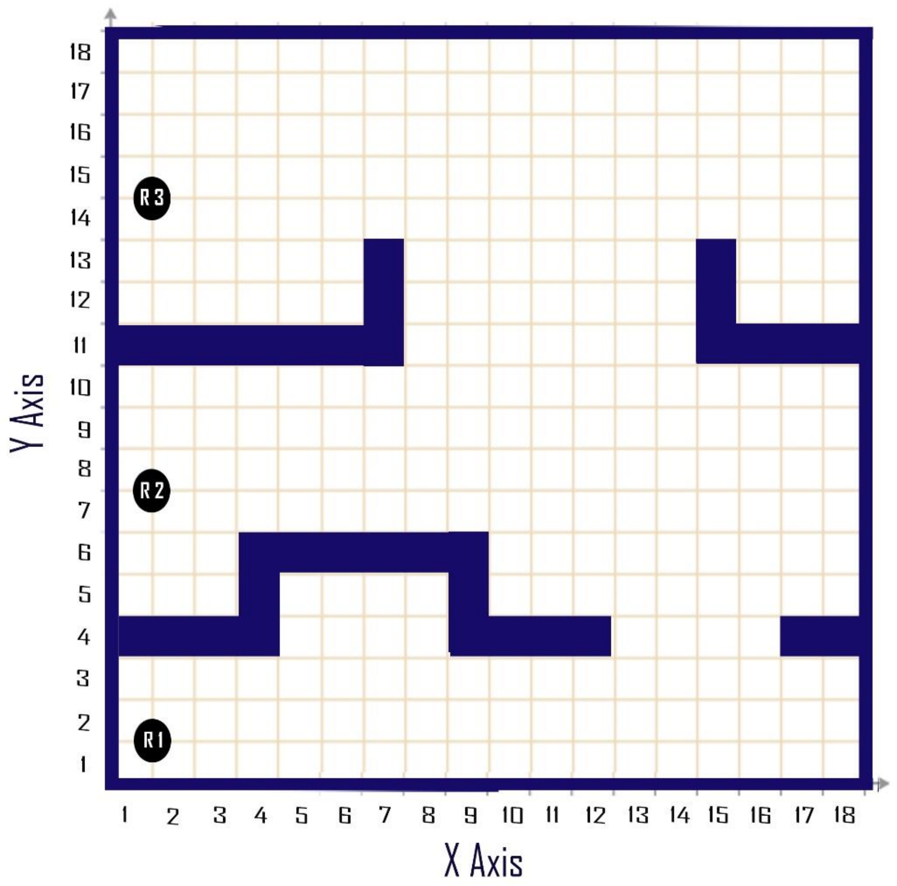


Fig. 5.1 Server Map

5.2 Planning Algorithm:

Now we discuss the algorithm adopted in the project for the planning phase.

5.2.1 Algorithm assumptions:

We state some assumptions for the algorithm. Firstly, the robot operates in a grid world and occupies one grid cell. Secondly, the robot can move to only neighbors on the same direction of its bearing or those require a rotation of the robot by only 45° in any direction. Thirdly, it is able to determine its position on the plane (state estimation knowing that the pulses given to motors for each line segment are known). Fourthly, the robot is able to use the map to compute the path it will take.

And each obstacle in the map has a safety margin to make sure that the robots are away enough from the obstacle to have the best performance from the obstacle avoidance algorithm **Section (5.3)** too, as shown in **Figure 5.2**

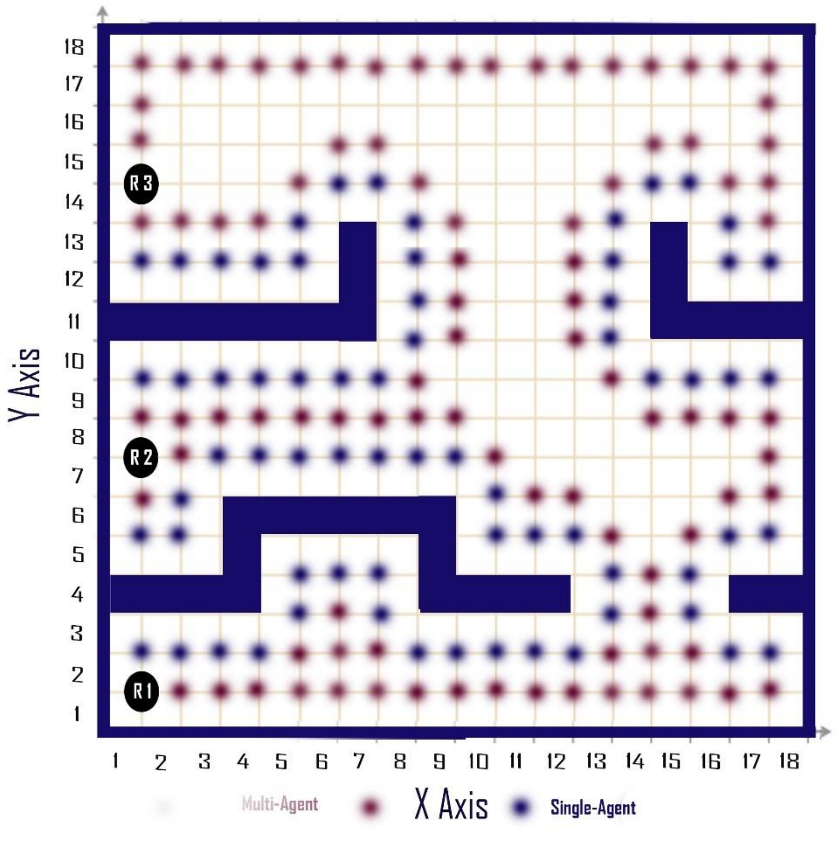


Fig. 5.2 Server Map with margin

Note that, path planning for a single agent needs only one margin of each obstacle, and the multi-agent needs another one to prevent the formation from hitting any obstacle in the map as we show in **chapter 3**.

5.2.2 A* Algorithm:

A* is a path finding algorithm that can be used to find the shortest path from the current location of the robot to the goal location.

A* uses a best-first search and finds a least-cost path from the given initial current location to the goal location. As A* traverses the map, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way. It uses a knowledge-plus-heuristic cost function of cell x (usually denoted f(x)) to determine the order in which the search visits adjacent cells in the map. The cost function is a sum of two functions:
- the past path-cost function, which is the known distance from the starting node to the current node x (usually denoted g(x))
- a future path-cost function, which is an admissible "heuristic estimate" of the distance from x to the goal (usually denoted h(x)).

The h(x) part of the f(x) function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for our application, h(x) might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two cells.

Like all informed search algorithms, it first searches the routes that appear to be most likely to lead towards the goal. What sets A* apart from a greedy best-first search is that it also takes the distance already traveled into account; the g(x) part of the heuristic is the cost from the starting point, not simply the local cost from the previously expanded cell. Starting with the initial cell, it maintains a priority queue of cells to be traversed, known as the open set or fringe. The lower f(x) for a given cell x, the higher its priority. At each step of the algorithm, the cell with the lowest f(x) value is removed from the queue, the f and g values of its neighbors are updated accordingly, and these neighbors are added to the queue. The algorithm continues until a goal cell has a lower f value than any cell in the queue (or until the queue is empty). The f value of the goal is then the length of the shortest path, since h at the goal is zero in an admissible heuristic.

The algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path keeps track of its predecessor. After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start node.

5.2.3 Implementation issues:

For the implementation of the A* algorithm, it requires:

- a two-dimension array to represent the map and each element of the array, defined by raw index and column index, represent a location on the map with (X, Y) values equal to (k*r, k*c) where r and c represent row and column indices and k is a scaling factor. The value of the element in the array will be zero if it represents a free space and one if it represents an obstacle.
- A class contains cell information such as position (x, y), direction of the robot in this cell, the cost from start location to this cell i.e. g(x), and the evaluation function f(x) = g(x) + h(x) where h(x) is the distance to the goal from the cell.
- A priority queue (open set) to contain cells to be traversed (The least cost cell will be traversed firstly). A two-dimensional array identical to the map, “Closed” cells, to represent the cells traversed in the algorithm.
- A two-dimensional array identical to the map, “action”, to represent the direction of the robot in each position on the route to the goal. This array will help in the reverse path traverse after finding the goal to determine cells on the path (path determination).
- An 8*2 array contains the eight possible directions of the robot that would be used to choose cells on the route to goal. Note that in each iteration only three directions of motion are available for the robot to

move in (the same direction of the previous iteration, the previous direction + 45°, the previous direction - 45°).

- Finally, a list of cells path will contain the cells on the path from source cell to the goal cell.

5.3 Obstacle Avoidance:

One of the most important capabilities in mobile robots during their motion is obstacle avoidance. Obstacle avoidance algorithms are essential for any autonomous mobile robot even if the robot does not support map-based planning. We can easily note that planning algorithms are required for map-based planning only (for both structured and unstructured maps). However, obstacle avoidance is required in both reactive navigation and map-based planning.

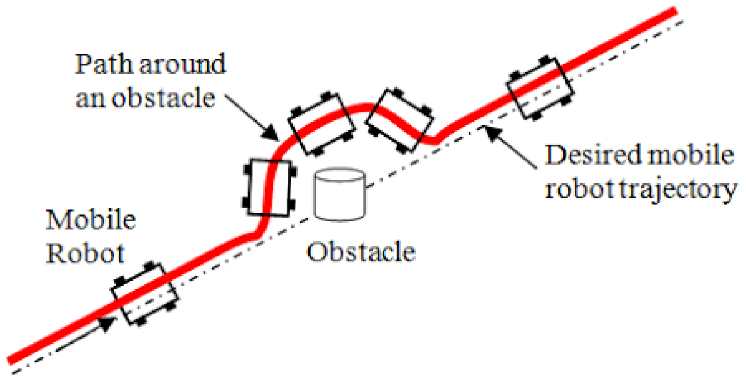


Fig. 5.3 Obstacle Avoidance Technique

In our project, we develop a simple algorithm, in which local sensor readings play an important role in the robot's future trajectory, depending on the Bug-type algorithms [20, 21], but we take into account the resulting path planning trajectory.

The Bug-type algorithms consider the robot's behavior at each instant generally a function of only its most recent sensor readings, but this can lead to undesirable and yet preventable problems in cases where the robot's instantaneous sensor readings don't provide enough information for robust obstacle avoidance.

In order to overcome these limitations and get more optimal behavior, we consider the robot's behavior as a function of most recent sensor readings and the desired trajectory.

Our algorithm steps are:

- We consider each sensor steers the robot into an opposite direction to its location w.r.t the robot $\vec{d}_i$ (i.e. sensor on robot left steers the robot to the right), this direction takes a weight (from 0 to 1) that inversely proportional to the distance that sensor give.
- We also consider the direction of the point next to the next point on the trajectory $\vec{d}_{NP}$ that comes from path planning, in order to return back the path.
- Finally, we calculate a weighted average direction $\vec{D}$ from all these directions, giving positive value to the right direction and negative to left similarly forward and backward.

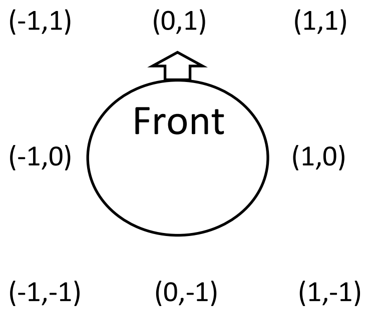


Fig. 5.4 Obstacle Avoidance Possible Directions

$$\vec{D} = \frac{\vec{d}_{NP} + \sum_i^n \vec{d}_i}{no.of\ directions}$$

So, we could determine the steering direction that the robot should apply in order to avoid the obstacle at the same time move toward its goal as possible as it could.

5.4 Task Allocation:

5.4.1 Introduction:

Multi-robot task allocation is a crucial issue in a multirobot system, which concerns on the strategy of matching tasks and robots to get higher performance efficiency according to certain principles. So, whether task assignment is proper has a great influence on the ultimate performance of a multi-robot system. And it has a close relation with how to establish groups and formation, because in most circumstances tasks are performed by group collaboration rather than individuals. With the increase of complexity and scale of the task to be executed, the significance of validity and efficiency of task allocation in a robot team becomes more distinct.

Extensive research nowadays is focusing on multi-robot systems (MRS). These systems offer many advantages as they have high potential to solve a multitude of problems such as industrial warehouses, surveillance and in search and rescue missions. MRS provide high reliability and performance of complex tasks even though they are simple in design [22]. It should also be noted that MRS possess the potential of producing poor results and/or non-deterministic performance if the design did not take into consideration the interaction and interference between the robots [23].

One of the main problems in MRS is task allocation which is the process of assigning tasks to the robots. It has to be done in order to avoid clashes between multiple robots and to increase the overall performance of the system. Therefore, the allocation of the tasks to the proper robots strongly affects the performance of the system [24]. This problem is known as Multi-Robot Task Allocation (MRTA) problem.

5.4.2 Proposed Algorithm:

- We assign number of robots to each task depending on the weight of the load at the start point.

- This algorithm returns a list of robots which is the best available robots to do the task. By dividing the load weight by the max load for each robot, we could obtain the number of robots which each task needs. If the algorithm returned -1 this means that there are no enough available robots for this task and the user have to wait until there is enough robots for this task.
- We divide the map to a number of locations depending on the number of robots in the system which each location has 3 best robots, the first location and the last location have 2 robots by checking the availability of the robots, we assign the task to them if their availability is equal to 1 and for the best robot of the task which its parking location is at the same area of the load we check its state, if its state is equal 3, we assign the new task to it.

Chapter 6

Conclusion and Future work

6.1 Conclusion:

Coordination of multi-agent systems has been regarded as a significant topic in research due to its importance in real world applications such as exploration and reconnaissance of a wide area, map building of unknown environments and cooperative transport of large objects. However, there are different challenges in multi-agent control which make it difficult to apply results proposed in research to get reliable products that can be used in industry. The first challenge is real time communication required between agents to perform their tasks. The second challenge is the difference between stability of single agent and that of the whole group of agents. Guaranteeing stability in the response of each agent does not guarantee stability of the multi-agent system. The third challenge is how to define the role of each agent in the system to complete the required task. Different approaches are used in research to develop robust multi-agent robotic systems.

In this project, we adopted some of these approaches to apply them to a real-world application. We proposed two modes of operation for system agents to perform a certain task. A task in our project means an object to be transported from a source location to a destination location. In the first mode, every individual agent is assigned a task to perform. In this mode, the role of the system is to choose which robot to perform the required task according to a task allocation policy that guaranteeing object transportation with least energy consumption. In the second mode, agents cooperate together to perform a task that can't be performed by an individual agent. In this mode, a formation control algorithm of multi-agent system is adopted to move the group in a certain formation that allows object manipulation.

6.2 Future work:

6.2.1 Further Steps:

- Equip the robots with manipulators to hold objects.
- Extend the server coordinating robots in a local environment to a global server on a cloud which enable the system to be installed in different environment for different users having accounts on system cloud.
- Extend the number of agents in the system to any number according to environment dimensions.
- Extend the coordination of multi-agent system to heterogeneous robots with different specifications.

6.2.2 Business Plan:

Coordination of multi-agent robotic systems is an important topic in research with many objectives to develop real products in industry. Till now the contribution of academia in the field of multi-agent robotic systems is greater than the contribution of industry. In our project, we aim to apply settled concepts in research to develop a real product that can be sold in industry. Our plan is as follows:

- Develop a prototype of 3-robots cooperating together to achieve the proposed concepts using 3 omni-directional lab robots.
- As a proof of concept, the proposed work with application on real hardware will be published in a scientific journal.
- The proven concept can get fund for manufacturers that aim to install this system in their work environment
- This Fund is the basis for system upgrade and purchase for different industrial partners

References:

[1] FIPA Contract Net Interaction Protocol Specification. Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002.

[2] Dias, M.B., TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments. Doctoral dissertation, Carnegie Mellon University, 2004.

[3] Gerkey B.P. On Multi-Robot Task Allocation. PhD Dissertation. University of Southern California, 2003, p. 126.

[4] Bellifemine F. et al, Developing Multi-Agent Systems With JADE, Wiley, 2004. 286 p.

[5] Luck M. et al., Agent Based Software Development. Artech House, 208 p., 2004.

[6] Lavendelis E. Open multi-agent architecture and methodology for intelligent tutoring system development. Summary of Doctoral Thesis. Riga, RTU, 49 p., 2009.

[7] Lavendelis E., Grundspenkis J. Multi-Agent Based Intelligent Tutoring System Source Code Generation Using MASITS Tool. Scientific Journal of RTU. Vol. 43, pp 27-36, 2010.

[8] Vanags M., et al. Service oriented mine hunting classroom simulation system. Applied ICT, April 2010, Latvia, Jelgava, pp. 95-101, 2010.

[9] Thrun S. et al. Probabilistic Robotics, MIT Press, 667 p., 2005.

[10] Josuttis, N. SOA in Practice. The Art of Distributed System Design. O'Reilly, 2007.

[11] Ren, W., Beard, R.W., Atkins, E.M.: 'Information consensus in multivehicle cooperative control', IEEE Trans. Control Syst. Technol., 2007, 27, (2), pp. 71–82

[12] Ren, W., Atkins, E.: ‘Distributed multi-vehicle coordinated control via local information exchange’, Int. J.Robust Nonlinear Control, 2007, 17, (10–11), pp. 1002–1033

[13] Beard, R.W., Lawton, J., Hadaegh, F.Y.: ‘A coordination architecture for spacecraft formation control’, IEEE Trans. Control Syst. Technol. Technology, 2001, 9, (6), pp. 777–790

[14] Anderson, B.D.O., Yu, C., Fidan, B., et al.: ‘Control and information architectures for formations’, IEEE Int. Conf. on Control Applications, 2006, pp. 1127–1138

[15] He, F., Wang, Y., Yao, Y., Wang, L., Chen, W.: ‘Distributed-formation-control-of-mobile-autonomous-agents-using-relative-position-measurements’, IET Control Theory and Applications,

[16] Deployment Patterns (Microsoft Enterprise Architecture, Patterns, and Practices)

[17] Guevara-Masis, Afsarmanesh and Hertzberger, L.O. (2004), Ontology-based automatic data structure generation for collaborative networks 8.

[18] Core J2EE Patterns by Deepak Alur et. al. (Prentice Hall, 2001)

[19] Jump up ^ Fowler, Martin "Patterns of Enterprise Application Architecture" (2002). Addison Wesley.

[20] Lumelsky, V., Skewis, T., “Incorporating Range Sensing in the Robot Navigation

Function.” IEEE Transactions on Systems, Man, and Cybernetics, 20:1990, pp. 1058–1068.

[21] Lumelsky, V., Stepanov, A., “Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape,”.

[22] L. Parker, “Multiple mobile robot systems,” Springer Handbook of Robotics, pp. 921–941, 2008.

[23] A. Hussein and A. Khamis, “Market-based approach to multi-robot task allocation,” International Conference on Individual and Collective Behaviors in Robotics (ICBR), 2013.

[24] Y. Han, D. Li, J. Chen, X. Yang, Y. Hu, and G. Zhang, “A multi-robots task allocation algorithm based on relevance and ability with group collaboration,” International Journal of Intelligent Engineering and Systems, vol. 3, no. 2, 2010.

Appendix: Project Code

Website Code:

https://github.com/multi-agent-Tanta-university-team/web-based-control-of-multi-agent-robotic-systems/tree/master/website%20code

Raspberry Pi Code:

https://github.com/multi-agent-Tanta-university-team/web-based-control-of-multi-agent-robotic-systems/tree/master/raspberry%20code

Low Level Code:

https://github.com/multi-agent-Tanta-university-team/web-based-control-of-multi-agent-robotic-systems/tree/master/low%20level%20code

Full Project Code:

https://github.com/multi-agent-Tanta-university-team/web-based-control-of-multi-agent-robotic-systems