CS 684 Project Report

Group Name: One Piece

Group # 4

Multi-Bot Controller

Gaurav Vijayvargia	133050031
Vaibhav Dave	13V050002
Thyagarajan Radhakrishnan	13305R004
Jyoti Shankar	133050080

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

November 14, 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The objective of this project is to enable the automation and monitoring of the greenhouse using multiple robots. Diversity in the type of tasks to be performed, limitation in the speed and energy requirements of the robot and the need for larger greenhouse coverage area make it necessary to use multiple robots in the greenhouse. However, multiplicity of robots in the greenhouse raises the issue of coordination between them and this is the issue we intend to address in this project. Here, coordination implies the safe allocation of a path in the greenhouse to the robot so as to avoid the possible collisions.

The report is organized as follows:

1. **Chapter 1** provides an overview of the Multi-Bot Controller and the report, and gives the definition of the keywords and acronyms used throughout the document.
2. **Chapter 2** describes the problem statement of the Multi-Bot Controller.
3. **Chapter 3** describes the hardware, functional and non-functional requirements, and the design constraints that are imposed on the Multi-Bot Controller.
4. **Chapter 4** describes the implementation details of the Multi-Bot Controller.
5. **Chapter 5** explains the working of the Multi-Bot Controller with the help of design diagrams like FSM and State Charts.
6. **Chapter 6** describes the testing strategy and data of the Multi-Bot Controller with experimental results.
7. **Chapter 7** describes the design challenges and the open issues faced during the implementation stage.
8. **Chapter 8** describes the future work that could be done on the Multi-Bot Controller.
9. **Chapter 9** concludes the report with a brief overview of our achievements in building the Multi-Bot Controller.

1.1 Definitions, Acronyms and Abbreviations

1. **Acronyms:**

   - GH : GreenHouse
   - CWC: Clairvoyant With Commute algorithm[4]
   - API: Application Programming Interface
   - GUI: Graphical User Interface
   - FSM: Finite State Machine
   - CRC: Cyclic Redundancy Check

2. **Definition:**

   - Task: Traversing of the robot from source to destination node on allocation of a path. On reaching the destination, GH tasks such as temperature sensing, weeding, seeding, etc. are assumed to be "abstract" (imaginary) tasks that will be performed.
   - Sensing point: A location in the GH where (abstract) tasks are to be performed.
   - Waiting slot: A location near the sensing point where the robots can wait for the next task request after performing it's current task. This assumption helps to avoid it from obstructing other robots from traversing via that route.
   - Command packet: The packet generated specifically for the robot by translating (mapping) the minimally shared shortest path (in the form of a list of vertices) into a set of commands to be executed based on the given model arena.

Chapter 2

Problem Statement

The problem statement of our project can be stated as follows:

"Development of a central controller for allocation of minimally shared shortest path in a multi-robot scenario."

Given a GH model arena (Figure 2.1), the objective is to co-ordinate and control the working of multiple robots on the arena in such a way that each robot minimally interferes with other robots in the system while performing the tasks that have been assigned to them. Due to the well-known localization problem of robots, each robot is unaware of not only it's own position but also of the position of other robots in the system. This raises the need for having a central controller that would keep track of the position of all the robots in the system at any point in time and use this crucial information for assigning tasks to them for enabling co-ordination. The onus is on the central controller to assign the tasks, specifically the path to be traversed to reach the destination, in such a manner that not only the possibility of a collision with any other robot is minimal, but also to ensure that the task gets completed in the best possible time with minimum energy consumption.

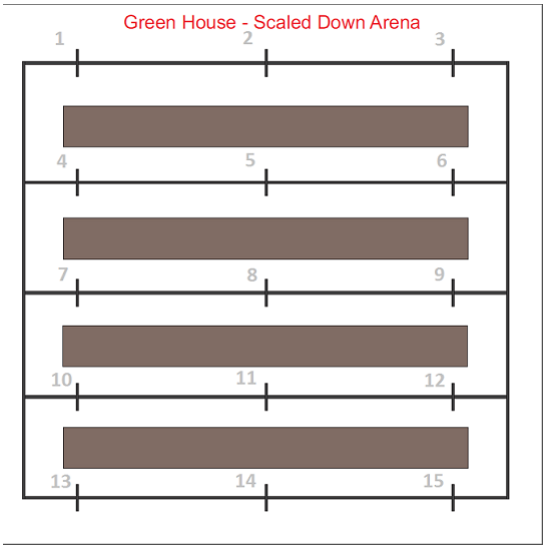

Figure 2.1: GH Model Arena

Chapter 3

Requirements

3.1 Hardware Requirements

- Windows/Linux machine.
- ATmega2560 micro-controller provides the platform for programming the FireBird V robot [1][2] to perform specific tasks.
- ZigBee [3] chip must be configured on the robots to enable wireless communication with the central controller, where broadcast mode is used at the controller-side.
- Serial COM port helps send data (i.e. traversal path in the form of commands) or receive data (acknowledgement for task completion) to the ZigBee protocol for communication between the central controller and robots.
- White line Sensors help in path navigation within the GH model arena.

3.2 Functional Requirements

- A GUI for the user to specify the source and destination node for a particular robot.
- A graph based solution to generate & assign the minimally shared shortest path to the robot.
- Wireless communication for packet exchange between central controller and robots.
- A well-defined packet format for sending commands to a particular robot in a multi-robot scenario.

3.3 Non-Functional Requirements

- Minimizing energy consumption on the robots by performing complex computations at the more-powerful central controller.
- User-friendly GUI.
- Modularity to assure re-usability of the code in future and help towards building a GH API.
- Following conventional coding standards to enhance readability and maintainability.

3.4 Design Constraints

- Size of the GH arena is restricted by the ZigBee range. Each robot must be within the range of ZigBee to ensure that the connection is not lost at any point in time.
- Each robot must have enough battery charge to perform all the assigned tasks.
- Availability of waiting slots at each sensing point is required so that no robot obstructs other robots from performing their tasks once it's own task is completed.

Chapter 4

Design Diagrams

4.1 System Architecture

The System Architecture of the Multi-Bot Controller system consists of the following components as shown in Figure 4.1:

1. GUI (Central Controller)
2. Robots
3. GUI - Robot communication using ZigBee

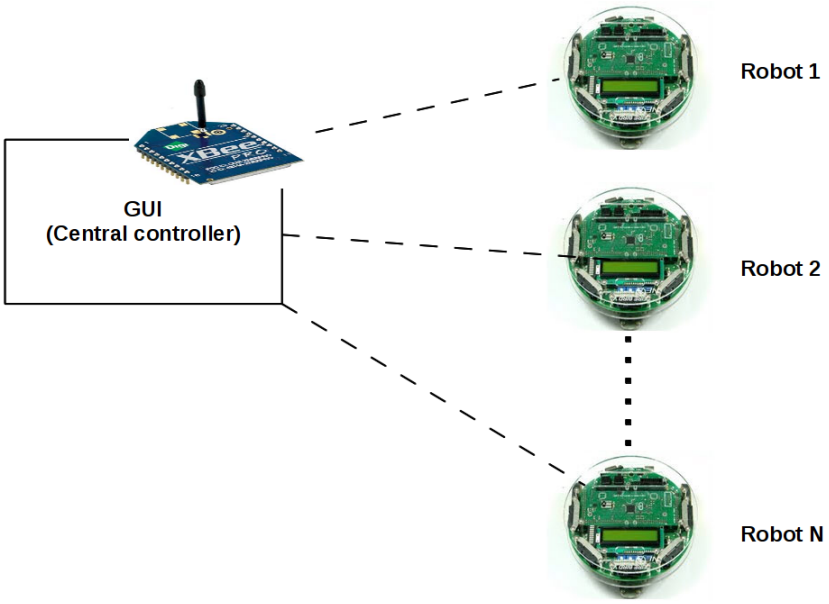

Figure 4.1: System Architecture of the Multi-Bot Controller

4.2 State Chart

The Multi-Bot Controller system can be represented in the form a state chart as shown in Figure 4.2:

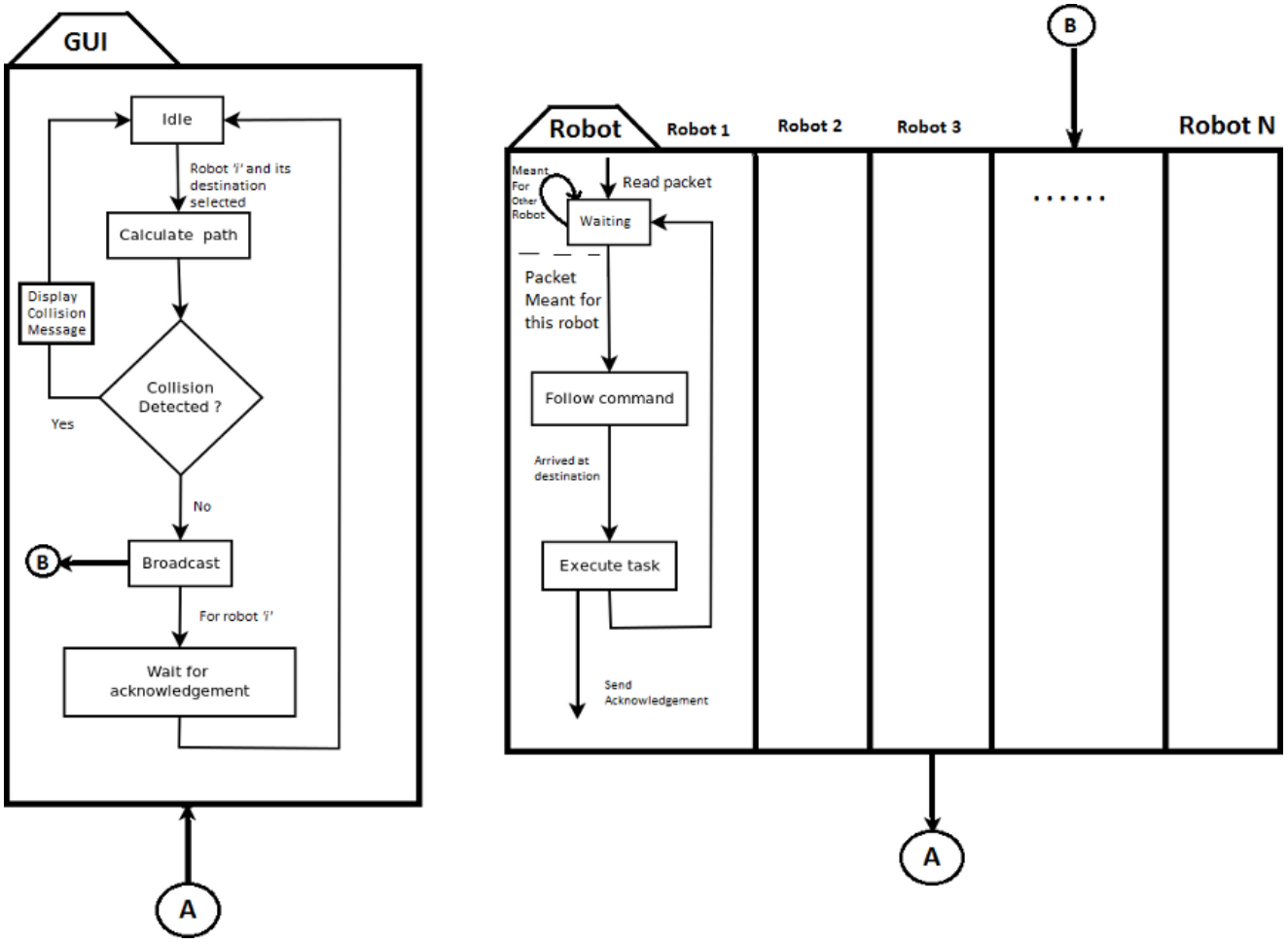

Figure 4.2: State Chart representation of the Multi-Bot Controller

- The figure on the left shows the state chart representation of the GUI, while the figure on the right shows the state chart representation of the Robots.
- The GUI (Central Controller) calculates the minimally shared shortest path to the destination for the selected robot, builds a command packet and broadcasts it to all the robots.
- As the packet is sent in broadcast mode, it will be received by all the robots in the system, and hence, the representation of the Robot state can be seen to be comprising of N AND states (state chart concept) for N robots.
- Only the robot for which the packet is destined will perform the execution of commands, while the rest of the robots will again become idle and wait for the next packet arrival. On finishing execution, the robot will acknowledge the central controller and go back to the idle state.

4.3 Control Flow Diagrams

4.3.1 Control Flow Diagram: GUI (Central Controller)

The control flow diagram of the GUI component of the Multi-Bot Controller system can be represented as shown in Figure 4.3:

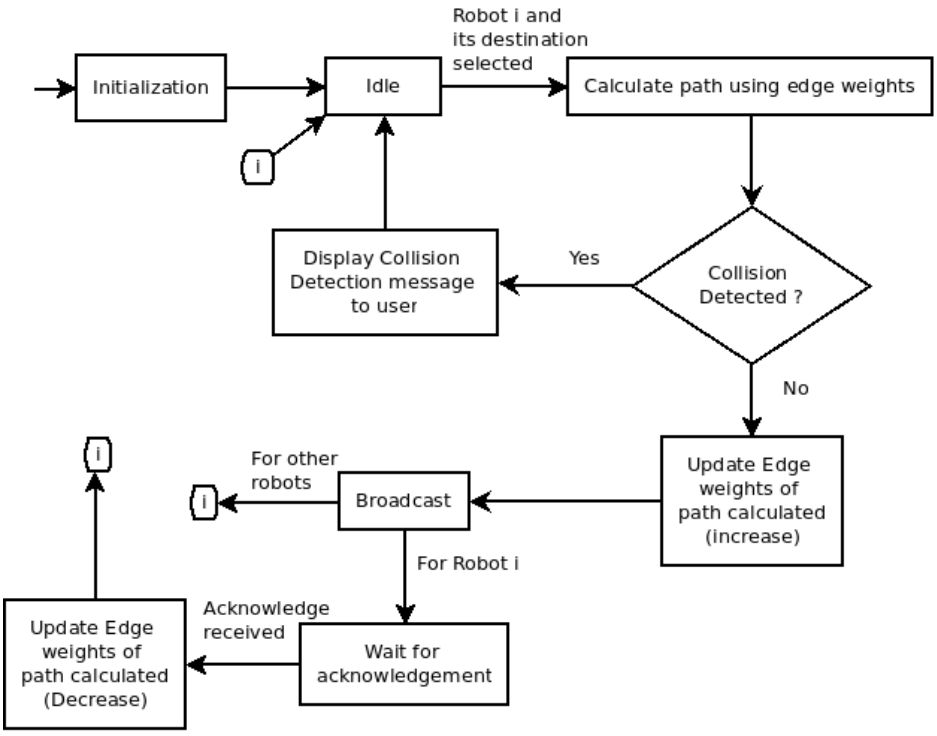

Figure 4.3: Control flow representation of the GUI

- In the "Initialization" stage of GUI, parameters like initial location of robots, direction of robots, initial edge weights, etc. are set. The GUI then waits for user input (Robot ID and its desired destination).
- After getting input from the user, the central controller calculates the minimally shared shortest path from the edge weights using Dijkstra's algorithm, and checks for collision.
- If collision is detected, it displays the message to the user informing about a possible collision. In case of no collision, edge weights belonging to the generated shortest path are increased so as to decrease the probability of selection of these edges during further path generation for other robots (upto the period for which the former robot is busy).
- After updating the edge weights, it broadcasts the command packet to all the robots via wireless medium using ZigBee protocol. It then waits for acknowledgement from the robot selected by user, whereas it goes back to the wait (idle) state for other robots waiting for user input.
- After receiving the acknowledgement from a (busy) robot, it releases (reduces) the edge weights of the path occupied by that robot and goes back to the waiting (idle) state.

4.3.2 Control Flow Diagram: Robot

The control flow diagram of the Robot component of the Multi-Bot Controller system can be represented as shown in Figure 4.4:

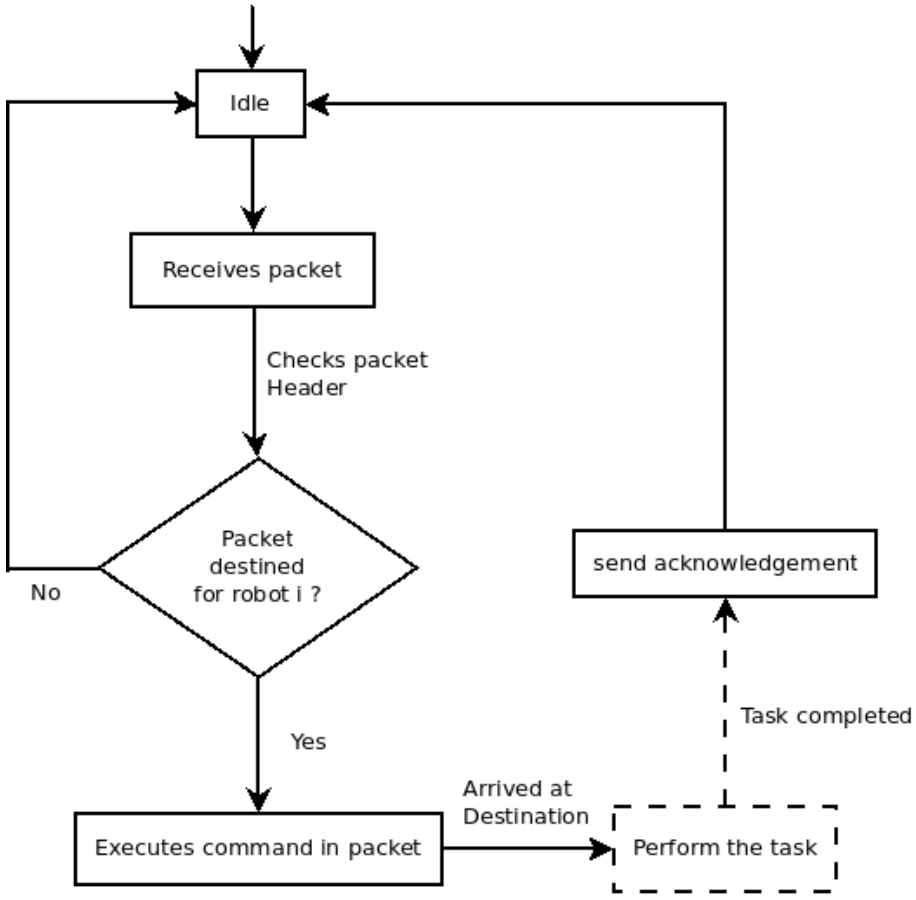

Figure 4.4: Control flow representation of the Robot

- Initially, the robot is in an idle state, waiting for the command packet to arrive from the central controller.
- When it receives a command packet from the central controller, it checks it's header for packet destination.
- If the packet destination (Robot ID for which the packet is destined) matches with it's own Robot ID, then it reads the rest of the command packet and executes the commands in the packet. If they do not match, then it discards the packet and goes back to the idle state.
- After execution of the commands, it should have reached it's destination and performs the GH task (abstraction in our case).
- When this task is completed, it sends a task completion acknowledgement to the central controller that is waiting for it's response.

4.4 Finite State Machine

The working of the packet communication protocol at the robot's end for path traversal can be represented in the form of an FSM as shown in Figure 4.5:

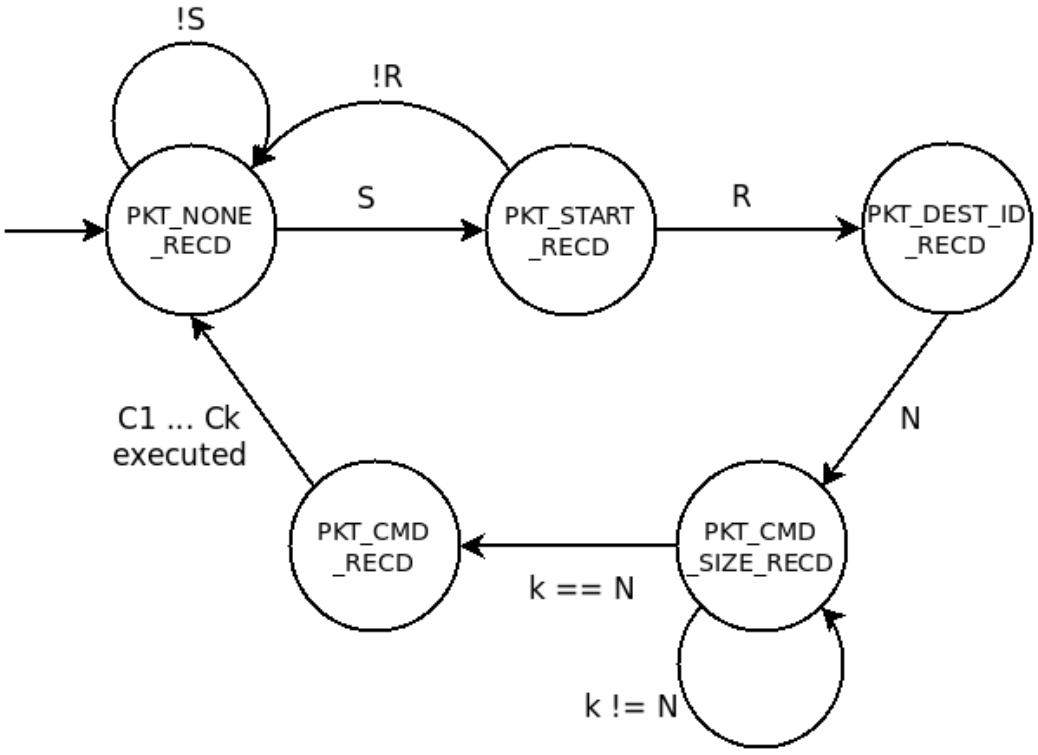

Figure 4.5: FSM representation of the Packet Communication Protocol

The notations used on the transitions in above FSM are described in Figure 5.1 (Chapter 5). In the idle state, when a command packet arrives at a robot, the following sequence of actions take place (reading byte-by-byte):

- If the 1st byte in the packet is the Start byte (S), then the robot will read the next byte. Otherwise, it will discard the rest of the packet.
- If the 2nd byte in the packet, which is the Robot ID (R), matches with it's own Robot ID, then it will read the rest of the packet. Otherwise, it discards the packet.
- It will read the 3rd byte, which is the number of commands to be executed (N = k) and store it.
- It will then read the rest of the packet, byte-by-byte, until it reads and stores all the k commands which it knew is contained in the packet from the 3rd byte (N).
- Then, it will execute the k commands to reach the destination, and go back to the idle state waiting for the next packet arrival.

Chapter 5

Implementation

Our solution to this problem works by finding the minimally shared shortest path from the source to the specified destination for the robots. This has been accomplished by representing the arena as a weighted undirected graph. Dijkstra's shortest path algorithm has been used to find the shortest path from the source node to the destination node for a particular robot. We describe in brief the implementation details of the Multi-Bot Controller in this chapter.

5.1 GUI (Central Controller)

- A GUI for the central controller is provided where the user can specify the inputs i.e. the Robot ID (based on type of work it can perform) for which the task is intended and the Destination sensing point where it needs to reach to perform some work. It's current position is fixed and gets updated after every traversal and hence there is no need to specify the Source sensing point for the robot.

- On submitting the inputs, a minimally shared shortest path for the selected robot is generated using Dijkstra's algorithm based on the edge weights of the graph representation of the model arena [see Algorithm 1]. Once the path is generated (which means no head-on collision is detected), all the edge weights (except the last) belonging to that path are increased by some constant amount. The last edge weight is increased by some large amount because it falls in the collision region (possibility of a collision is highest at the destination node).

- The path generated from the above step is in the form of a list of vertices. However, the robot has no knowledge of the structure of the model arena and hence it becomes necessary to transform this path into a set of commands that the robot can execute to reach the destination by following the black line. We have designed an algorithm for this mapping, which is based on the manner of positioning of the sensing points in the given model arena. This mapping generates a set of commands to be executed in sequence by the robot. We call these commands 'FLRU', where F stands for forward, L for left, R for right, and U for U-turn. These commands are sent to the robot in the form of a packet.

Algorithm 1 Dijkstra's Single-Source Shortest Path Algorithm

Data: (Un)directed graph $G = (V, E, w)$, with non-negative edge weights and source s and destination $d \in V$

Result: Calculation of shortest path to V from source s to destination d

```
for i := 1 ... num(V − {s}) do
|  D(V_i) := ∞;
end
D(s) := 0;
P := φ
while d not in P do
|  u := a vertex not in P with D(u) minimal;
|  P := P ∪ {u};
|  for all vertices v not in P do
|  |  if D(u) + w(u, v) < D(v) then
|  |  |  D(v) := D(u) + w(u, v);
|  |  end
|  end
end
return D(d);
```

5.2 Packet Communication Protocol

FSM representation of the packet communication protocol and it's functioning was described in Section 4.4. The packet format used by this protocol is shown in Figure 5.1:

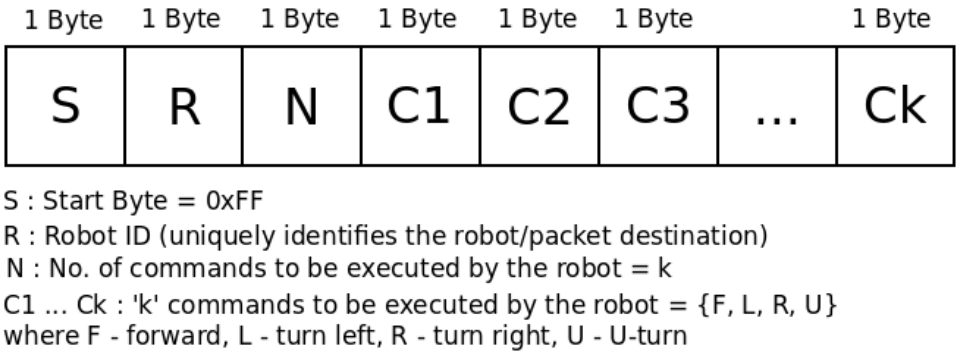

Figure 5.1: Packet Format of the Packet Communication Protocol

- We have designed this packet format for sending the commands to the robot, which consists of a start byte (indicates start of packet), Robot ID, number of commands to be executed, followed by the actual commands ('FLRU' sequence). This packet is sent to the robot using ZigBee protocol through a wireless media by the central controller. On execution of the commands and reaching the destination, the robot acknowledges this to the central controller by sending it's Robot ID. On receiving this acknowledgement, the central controller releases i.e. reduce the edge weights of the path that was assigned to the robot.

- All the above tasks can be performed concurrently for multiple robots in the GH i.e. multiple robots can be traversing the GH at any point in time with a minimally shared shortest path assigned to each one of them to avoid possible collision.

5.3 Program Primitives

The purpose of the (reusable) program primitives that have been defined for our implementation of the Multi-Bot Controller system are stated in Tables 5.1 and 5.2:

5.3.1 Central Controller

Primitives	**Use**
void computePaths(Vertex)	Dijkstra's algorithm: Computes the shortest path for the specified source
List<Vertex> getShortestPathTo(Vertex)	Used to retrieve the shortest path to the target vertex i.e. destination
void setPreviousNull()	This reinitialisation function is called before every call to Dijkstra's algorithm
double computePathWeight(List<Vertex>)	Computes the weight of a specified path
void updatePathWeights(List<Vertex>, int)	Updates the edge weights for the path that has been assigned to a robot or released
void updateAdjacencyWeights (Vertex, Vertex, int, int)	Updates the (adjacency) edge weight between two vertices v1 and v2
String vertexToFLRUMapping(List<Vertex>)	Mapping of the specified path to a set of commands: a string comprising of {F,L,R,U}

Table 5.1: Primitives for Central Controller

5.3.2 Robot

Primitives	**Use**
void black_line_follower()	Black Line Follower
void left_turn()	Turn Left
void right_turn()	Turn Right
void u_turn()	Take U-turn
SIGNAL(SIG_USART0_RECV)	ISR called when a byte is received from ZigBee

Table 5.2: Primitives for Robot

Chapter 6

Testing Strategy and Data

The following testing strategy and data will help to evaluate the Multi-Bot Controller system:

6.1 Configuring ZigBee on Central Controller and Robots

The configuration of ZigBee on central controller and robots is briefly described below: (Refer to the ZigBee Manual [3] Section 2.4 for precise configuration details.)

1. PAN ID on all the ZigBee must be the same.
2. Destination High and Destination Low 64-bit address (32-bit each) on the ZigBee of the central controller must be configured to be in broadcast mode to broadcast the command packets. (Refer Section 2.4.2 of the manual[3] for setting the broadcast mode.)
3. Destination High and Destination Low 64-bit address on the ZigBee of all the robots must be configured to match with the Serial Number High and Serial Number Low address of the ZigBee on the central controller.

6.2 Programming the Robots

1. Each robot must be programmed to receive the command packets, decode it, execute it and follow the path using the black line follower to reach the destination.
2. The code for 2 robots (on which we tested our implementation) are provided. The ideal difference between the codes which was programmed on the robots should have been in the ROBOT_ID for the robot (unique for each robot). However, varying motor speeds and sensor values for path traversal resulted in having to test them for these parameters as well. This needs to be checked for every robot.

6.3 Running the Central Controller

The central controller has been implemented in Java and the code has been provided. Go through the documented code first before running the program to understand the below steps:

1. Connect the ZigBee to the central controller, and turn ON the robots. Assuming that the COM port on which the ZigBee is connected and the COM port specified in the main() function of the program match, running the code will display the following GUI:

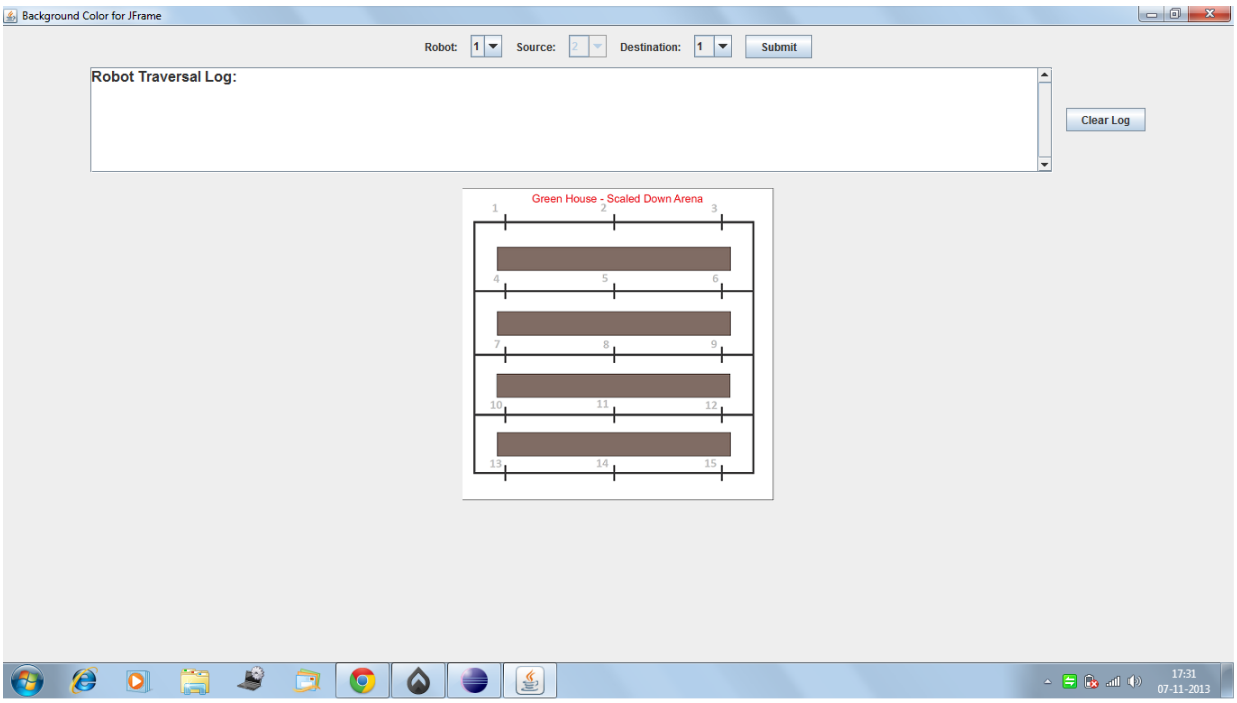

Figure 6.1: GUI of the Central Controller

The GUI will consist of the following components:

- Drop-down menu for selecting the **Robot ID**
- **Source** (current position) of the selected robot
- Drop-down menu for selecting the **Destination** sensing point
- **Submit** button to send the above inputs to the program for minimally shared shortest path calculation
- **Robot Traversal Log** which will display the occurrence of various events in the system
- **Clear Log** button to clear the above log contents
- **GH Model arena map** for reference

2. Choose the Robot ID and the Destination sensing point from the drop-down menu as shown in Figures 6.2 and 6.3:

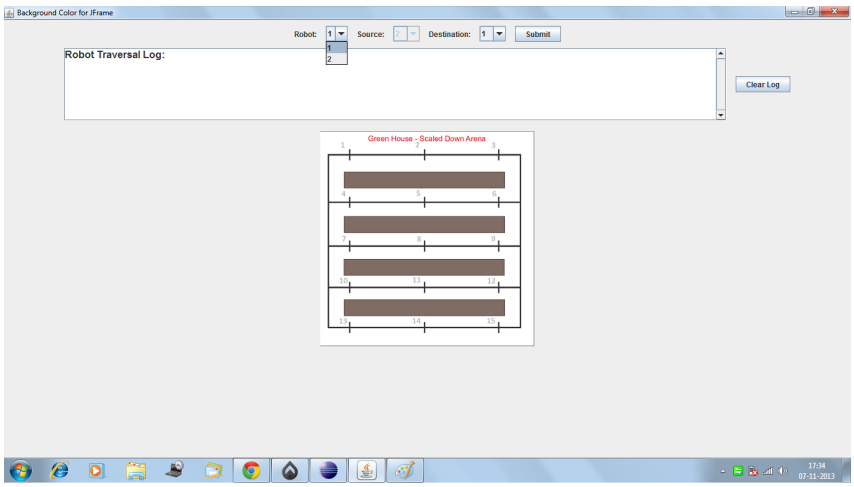

Figure 6.2: Select Robot ID

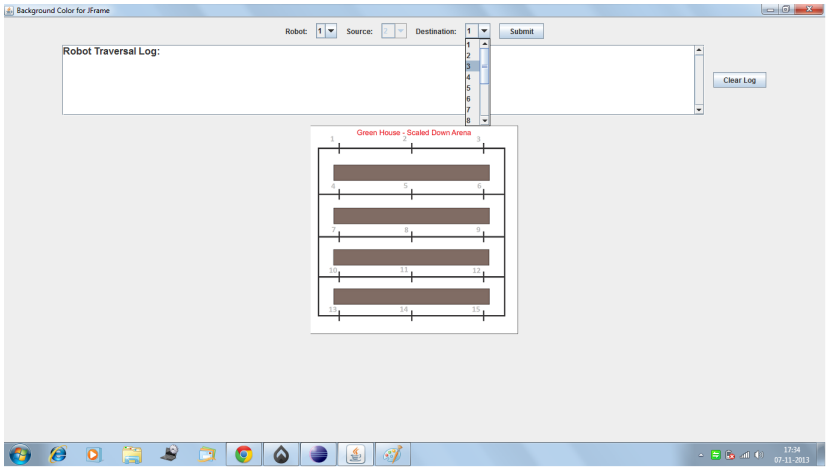

Figure 6.3: Select Destination

3. Suppose you choose Destination '8' for Robot '2' and then Destination '12' for Robot '1', the resulting GUI would look like as shown in Figures 6.4 and 6.5 (see the log - commands that will be executed are also displayed):

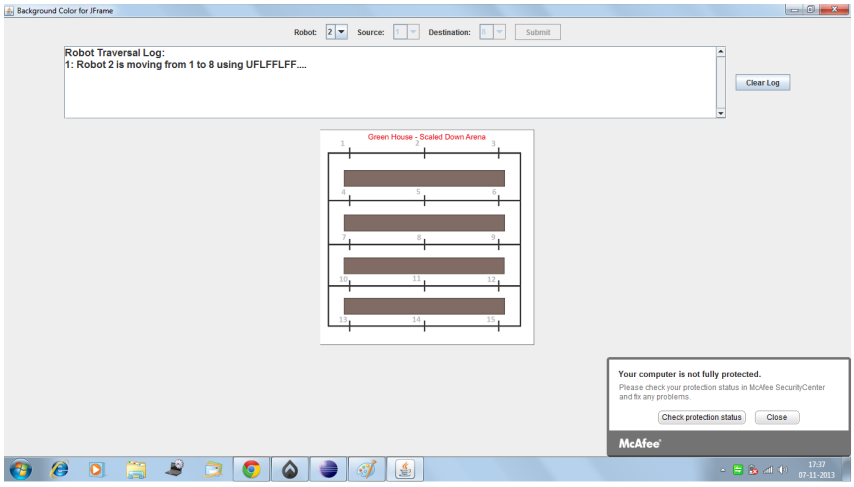

Figure 6.4: Specifying Destination for Robot 2

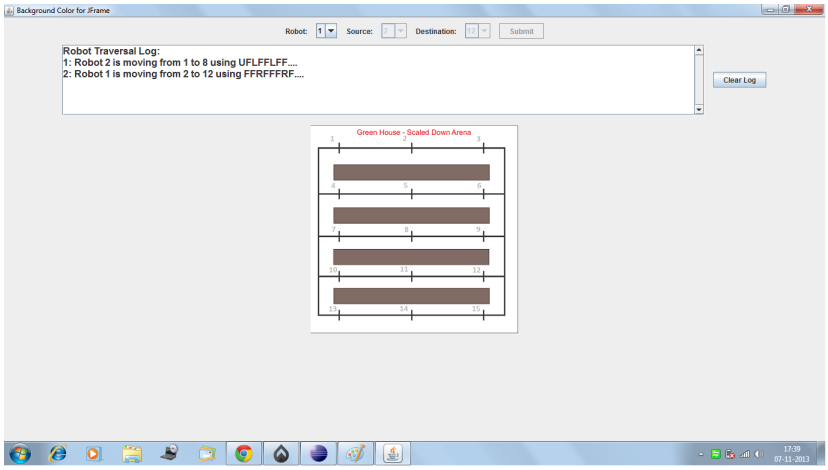

Figure 6.5: Specifying Destination for Robot 1

4. If Robot 2 reaches it's destination first and then Robot 1 reaches it's destination, the GUI would look like as shown in Figures 6.6 and 6.7 (see the log):

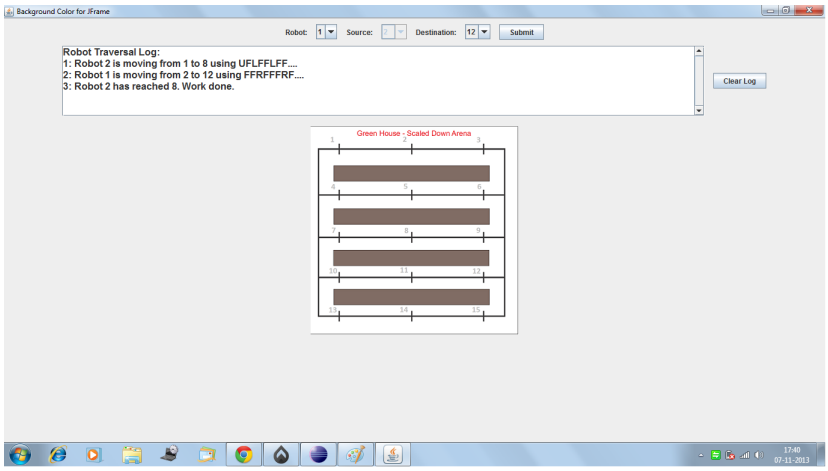

Figure 6.6: Robot 2 reaches Destination

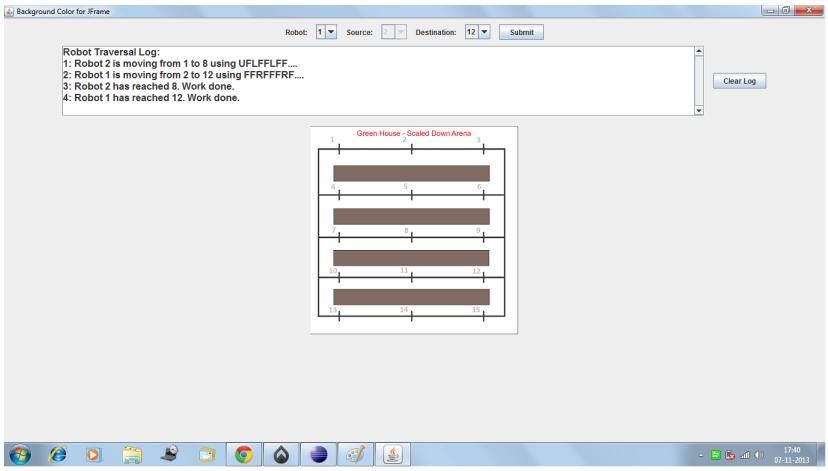

Figure 6.7: Robot 1 reaches Destination

The above procedure can be repeated any number of times for multiple runs of multiple robots.

Chapter 7

Design Challenges and Open Issues

The design challenges and the open issues faced during the implementation stage are addressed in this chapter.

7.1 Issues Encountered

The issues that we encountered while working on the project are enlisted below:

1. Understanding the code of the CWC[4] algorithm, which was identified as a risk during the analysis stage, came across as a major issue in the project due to insufficient documentation. To avoid missing the deadline, the idea of using the CWC[4] algorithm was dropped, and instead, an algorithm to find the minimally shared shortest path for a robot in a multi-robot scenario was proposed and has been implemented to solve the preliminary problem.

2. Black-line following on the given flex sheet of the GH model arena was another major issue that we faced. The reason behind this issue was the smaller width of the black line on the flex. The width of the black line to be followed was less than the space between any 2 consecutive white line sensors on the FireBird V robot [1][2], and this made the robot sense an all white condition even when it was on a black line. Another issue that we faced during the black line following was due to the varying motor speeds of different FireBird V robots even in a fully-charged condition. This consumed a lot of time as we had to test different robots by writing different programs for each robot with varying speeds.

3. The initial solution to the problem which we proposed (based on dividing the arena in the form of a grid) for collision detection and avoidance on the model GH turned out to be arena specific. So, we changed our approach to make the solution more generic by using graphs as a data structure to represent the state of the GH. This transition from arena specific to a generic solution was not easy, as it involved making a lot of changes in the already developed code in a short span of time.

4. The central controller and the robots communicate using the ZigBee protocol over the wireless media. The data sent over this wireless media is susceptible to noise and hence, the problem of packet loss was very much evident and encountered quite often, specially when the central controller and the robots were far from each other (constraints in the ZigBee range seemed to cause the packet losses).

7.2 Insight Gained

The insights gained while working on the project are outlined in the following points:

1. **Modular Approach:** Approaching a large problem by dividing it into smaller and relatively independent problems reduces the time to develop the desired system. It also helps the project team to manage their work and independently work on different modules. Our project typically has been divided into four modules, each having relatively independent responsibilities to be performed, which can be outlined in the following points:

   (a) A GUI for user - central controller - robot interaction on specifying inputs from the user.

   (b) Minimally shared shortest path generation module for robot traversal.

   (c) Module for converting the shortest path generated into a command packet which is arena specific.

   (d) Module running on the robot which accepts the packet from central controller and executes the commands in that packet using a black line follower to reach the intended destination.

2. **Code Documentation:** In the initial phase of the project, we were supposed to understand and implement the code of CWC[4] algorithm and deploy it on the GH model arena. However, insufficient documentation of this code changed the problem definition and course of the project. This gave us the motivation to document our work properly for future extensions. All the modules are properly commented and indented; standard variable naming has been followed throughout the modules. Architecture and design of the project has been documented in the form of State charts, FSMs and Control Flow Diagrams.

3. **Generalization:** Approach to solve any particular problem should be as general as possible because as the scope of the problem grows, a specific or customized solution to the problem acts as a hindrance in carrying forward the specificities of the implementation. Initial implementation of this system was pretty specific towards the GH model arena and wasn't acceptable. Hence, in the next iteration, we followed an approach to make the solution more general; graphs have been used to represent the state of the system and the arena, and Dijkstra's shortest path algorithm has been used to find the route to navigate the robot. This particular implementation made this project independent of the arena for finding the shortest route from source to destination.

Chapter 8

Future Work

This work can be extended to be used in the actual GH instead of a model arena with few modifications. The main areas where scope of improvement lies are as follows:

1. The minimally shared shortest path generation module of our algorithm is generalized and can be used with any arena, but the generation of commands using this path is specific for the model arena which we have considered. This dependency can be removed in future by representing the arena in some standard format, like in a file, and using this file as an input to the different modules.

2. Every robot acknowledges the completion of the path traversal to the central controller on reaching the destination. Once this acknowledgement is received at the central controller, weights of the segments/edges allocated in this route to the robot gets released (reduced). This implementation can be improved by having segment-wise acknowledgement and release of weights from the allocated path for a more efficient minimally shared shortest path generation and better utilization of the arena.

3. The path that needs to be traversed by a robot is sent to it in a packet by the central controller through a wireless link. This packet is susceptible to the noise in the link and hence, packet loss or packet errors are evident in this communication setup. This problem can be resolved in future by having provision for error detection (checksum, CRC etc.), acknowledgements and retransmissions.

4. The GUI can be extended in future to have a better look and feel and can be more user friendly. Currently it only includes the interface to take the robot ID and the desired destination for that robot (It's initial position is known and hence, the source is fixed). In future, it can be extended to specify the robot's initial position from the user. A small video screen can also be included (with camera mounted on each robot) in the GUI for the user to see from the robot's view point. The image of the model arena is currently used just as place holder. In future, it can be used to show the current position of the robot and the path allocated with segment-based acknowledgement feature.

Chapter 9

Conclusion

First and foremost, we would like to thank Prof. Krithi Ramamritham and Prof. Kavi Arya for their support and guidance during the course of this project. The motivation behind building a Multi-Bot Controller for the GH was the need for automation and monitoring of the GH which might consist of multiple robots performing a wide variety of GH tasks like temperature or humidity sensing, seeding, weeding, harvesting, etc. The presence of multiple robots traversing throughout the GH introduces the need for coordination among them so as to avoid collision between them. This is essential for the proper functioning of the GH and this project provides one way to deal with this problem.

To address this problem, we proposed a graph-based solution for coordination among robots in the GH. Deliverables of our final system can be briefly summarized as follows:

- A graph based solution to generate the shortest path with minimal path sharing.
- A GUI for specifying the source and destination node (based on a greenhouse model arena) for a particular robot.
- Wireless communication (using ZigBee) for packet exchange between central controller and robots. Communication between central controller and ZigBee is established through serial port (COM port).

The tasks performed by the Multi-Bot Controller system include:

- Selecting the Robot ID and getting its desired destination node as an input from the user.
- Finding the minimally shared shortest path from source to destination node and checking for possible head-on collision.
- If there is no head-on collision detected, send the command packet containing the commands for path traversal to the respective robot.

A detailed description and working of our proposed solution for the Multi-Bot Controller was provided in Chapter 4, 5 and 6. Chapter 7 described the issues encountered and the insight gained during the course of this project. We concluded with describing the future work that could be done to enhance the functioning of the Multi-Bot Controller in Chapter 8.

Bibliography

[1] FireBird V ATmega2560 robotic research platform hardware manual. IIT Bombay & NEX Robotics Pvt. Ltd.

[2] FireBird V ATmega2560 robotic research platform software manual. IIT Bombay & NEX Robotics Pvt. Ltd.

[3] XBee/XBee-PRO OEM RF Modules Manual.

[4] "Scheduling Issues In Greenhouse Automation Using Robots", VGSN Lohith, Department of Computer Science and Engineering, IIT Bombay, 2013.