CS-684-2016 Final Report

Auto Serving bot

Amit Pathania 163054001
Manjunath K 163050008
L Goutam 163050091

Table of Contents

1. Introduction

In today's busy world, every street has a restaurant. But a restaurant with tasty food is hard to find among these restaurants. Even if you do find one, it may be always busy with a long queue due to unpunctual waiters who cannot serve the customers with proper scheduling of orders. Some of these waiters may also be aggressive and one may not find good ambience in these restaurants. Also, cheap and trust worthy human factor is hard to find these days. By replacing these waiters with automated bots, we can reduce the cost as well as the time taken to serve huge number of customers. The diners at a restaurant will not have to deal with human errors from their waiters. The customer experience is also enhanced with the reliability and flexibility of the bots. A Bot can never provoke or annoy a customer. It is always submissive. With its less maintenance cost and high reliability, these bots are highly superior to human waiters.

"Auto-Serving Bot" is firebird robot which can automate the serving process in a restaurant and can be used to serve orders at the customer table. The robot also accommodates best features like find the shortest route to the counter, obstacle detection and collision avoidance, rerouting dynamically. Multiple ASB(Auto-Serving Bots)s are used to improve customer service and share workloads among bots.The customer will use android app to place an order from table and the bots will be used to deliver the order on the table once order is prepared. The robots use black line following technique to reach counter or desired table and ZigBee for communication with the counter.

2. Problem Statement

The Project aims to automate the process of serving orders in restaurant by using firebird V robot. We have successfully created an android application from which order can be placed by a customer and the order is served to his table by a firebird V robot.

The task was made more challenging by adding two more bots for serving and hence there was requirement of coordination among three bots to ensure proper delivery of orders and avoiding collisions and deadlocks. The same was achieved by using round robin scheduling algorithm for tasking bots and carefully designing onward and backward path from start point to destination and planning re-routing algorithms in real time to avoid collisions and deadlocks by using techniques similar to exponential backoff algorithm used for collision avoidance in computer networks using variable delays.

3. Requirements

3.1 Functional Requirements

3.2 Non-Functional Requirements

Performance: The order placed by a customer is placed into the database without any significant delay. And after the order has been prepared, the bots serve the orders in a round robin fashion which increases the delivery speed by using multiple bots.

Ease of operation: The Android Application user interface should be user friendly, and the

user should not find it difficult to place an order or enquire the details about the cost of each item.

Open Source Software: Using Open source softwares to reduce cost of building the project.

The Bots should stop and serve the order in a position in which the user is comfortable to take the order

3.3 Harwdare Requirements

1. Firebird V ATMEGA 2560 – 03 Nos. To act as serving bots.

2. Proximity/IR sensors – for collision avoidance.

3. Android tablet/phone – To install and place order from the Android application.

4. Zigbee modules – 04 Nos. For communication between counter/admin and bots for checking whether bot is free and passing desired destination table number for order delivery.

3.4 Software Requirements

1. AVR studio : To program instruction onto a given bot

2. XCTU: Xigbee programmer to setup Zigbee parameters

3. Android Studio: For creating the android application from which an order is placed

4. Python : To read table number for delivery and send same to bot using Zigbee.

5. Wamp64 : To use MySQL, PHP database for creating website and apps

4. System Design

Each bot is having Zigbee module for communication and one Zigbee module is kept at counter. Each table is having Android tablet/phone. The order will be placed from android app. Once the order is prepared, the bot will collect the order from counter and then the bot will deliver it to the table as per the input table number. On delivery of the order, it will return to the counter automatically to attend the next order. Navigation to the desired table would be on pre-defined path. In case of obstacle enroute, it will re calculate the path.

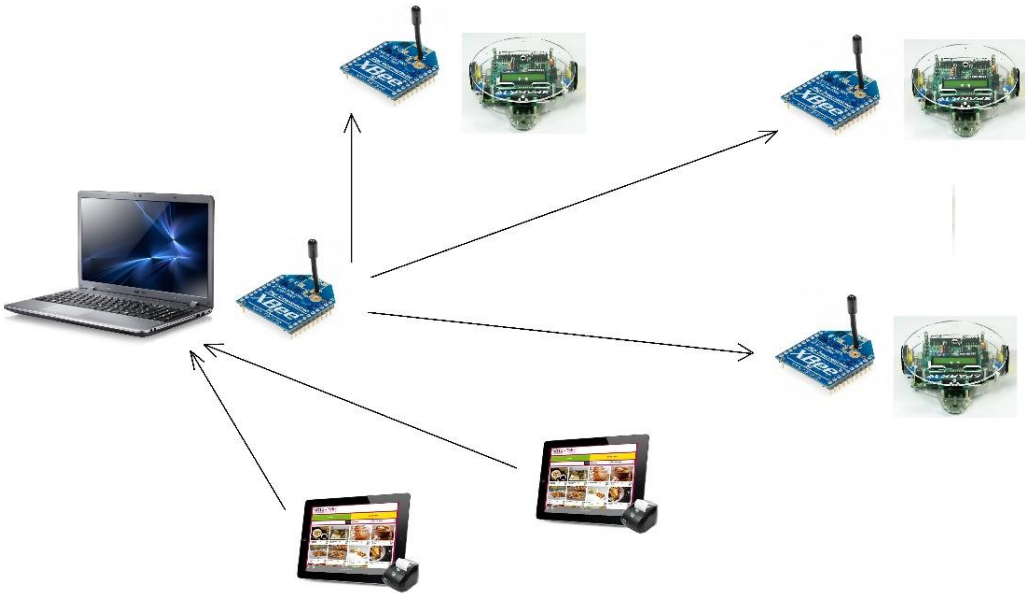


Illustration 1: System architecture

List of states. The auto serving robot can be in one of four modes. Each mode has its own distinct behavior. Each mode is designated by a module (state) representing a sub-state machine having required functionality.

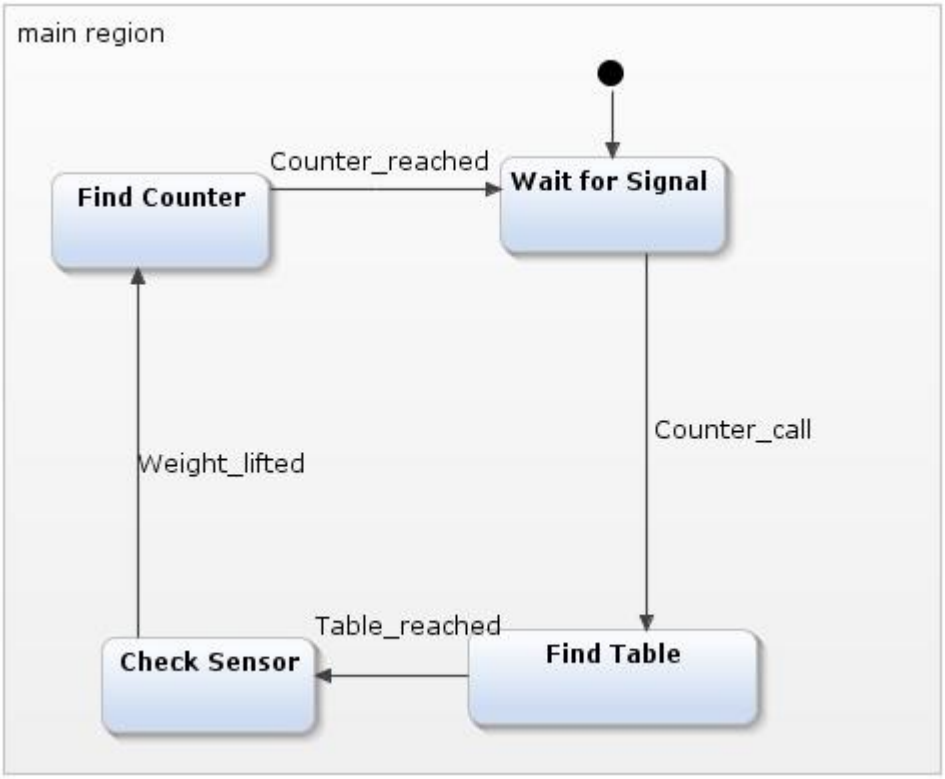


Illustration 2: List of states

- **Wait for Signal state :** Robot will looking for incoming signals on Zigbee during this state.
- **Find Table state:** It carries out all activities to find the table where order has to be delivered.

- **Check Sensor state:** provides functionality to check the order kept on the robot once customer table is reached.

- **Find Counter state:** carries out all activities to find the counter

Find Table state is further sub divided into three sub state machines. Each of these sub states performs concurrently inside the Find table state machine.

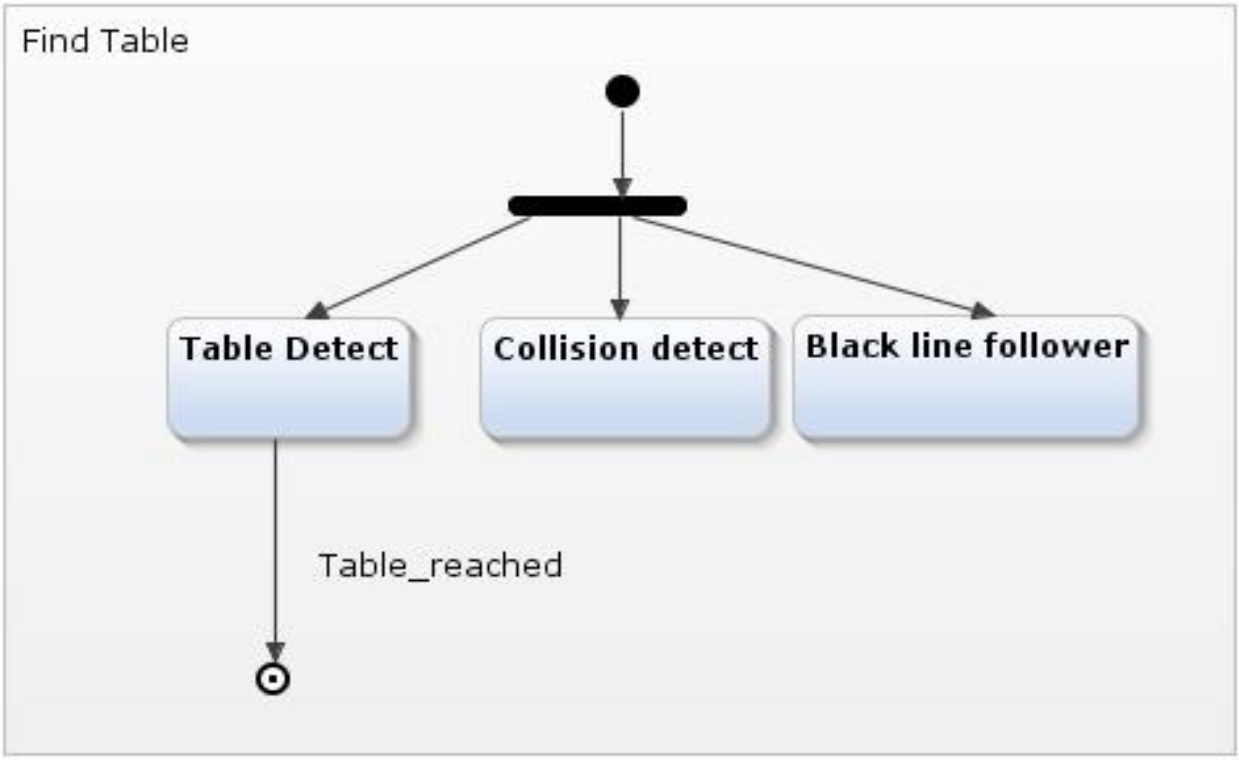


Illustration 3: States in Find table event

- **Table Detect module** continuously checks to see whether robot has reached the desired table number. Reaching table causes triggers Table_Reached event.

- **Collision Detect module** looks for other robots, people and surrounding items to avoid collisions. The robot takes required action based on obstacles.

- **Black Line Follower** module carries out activities needed to keep the robot moving along the specified pre defined path to each table.

Find counter state is further sub divided into four sub state machines. Each of these sub states performs concurrently inside the Find counter state machine.

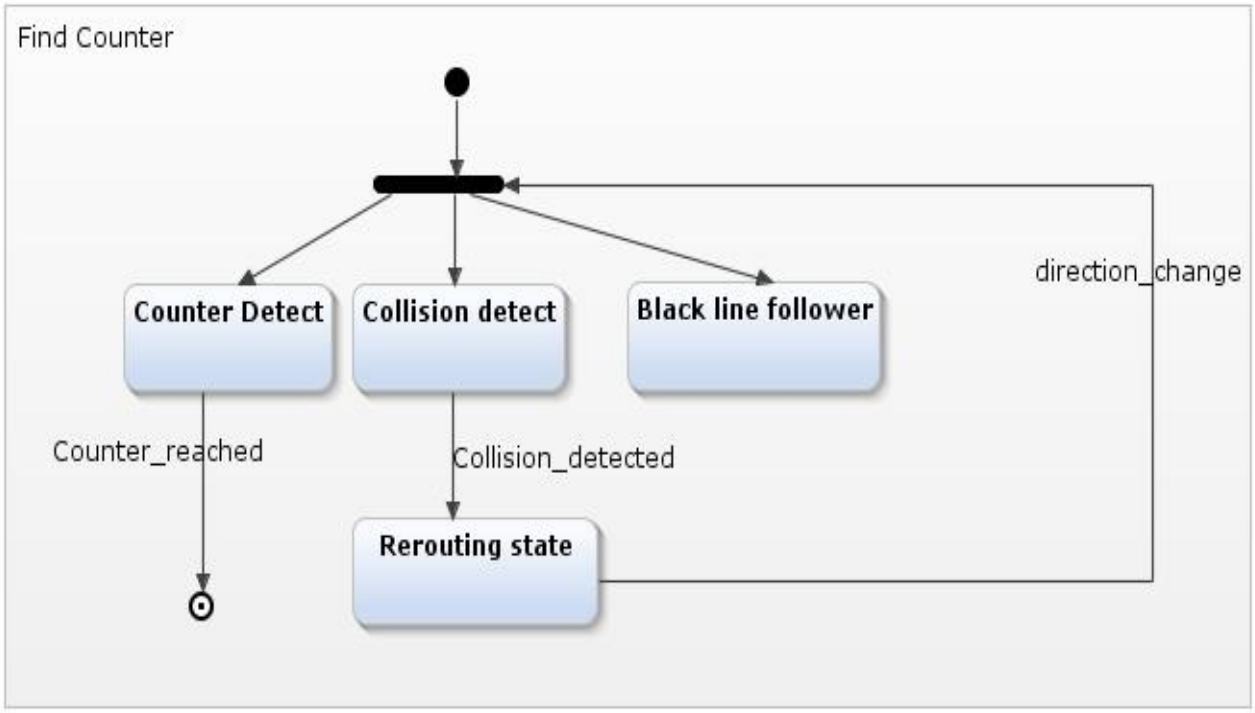


Illustration 4: Substates in Find counter state

- **Counter Detect module** continuously checks to see whether robot has reached the counter. Reaching the counter triggers Counter_Reached event.

- **Collision Detect module** looks for other robots, people and surrounding items to avoid collisions. On obstacle detection triggers Collision_detected event.

- **Rerouting state module** changes the direction of the bot. The robot recalculates its route based on obstacles.

- **Black Line Follower** module carries out activities needed to keep the robot moving along the specified pre defined path to counter.

- On occurrence of signal Counter_reached the Find counter Module terminates its execution.

List of events. Following events causes transitions between different states:

- **Table_reached**. Whenever desired table is reached,this event is triggered.

- **Counter_call**. Whenever customer order is prepared, the counter boy presses ORDER ID button and the Xigbee module kept at the counter will be used to call the robot, this event is triggered.

- **Counter_reached**. Whenever counter is reached, this event is triggered.

- **Weight_lifted**. When customer lifts the order. The sensor checks order has been lifted and this event is triggered.

Checkpoint detection

Here 1,2,3,4 and 5 indicate sensors.

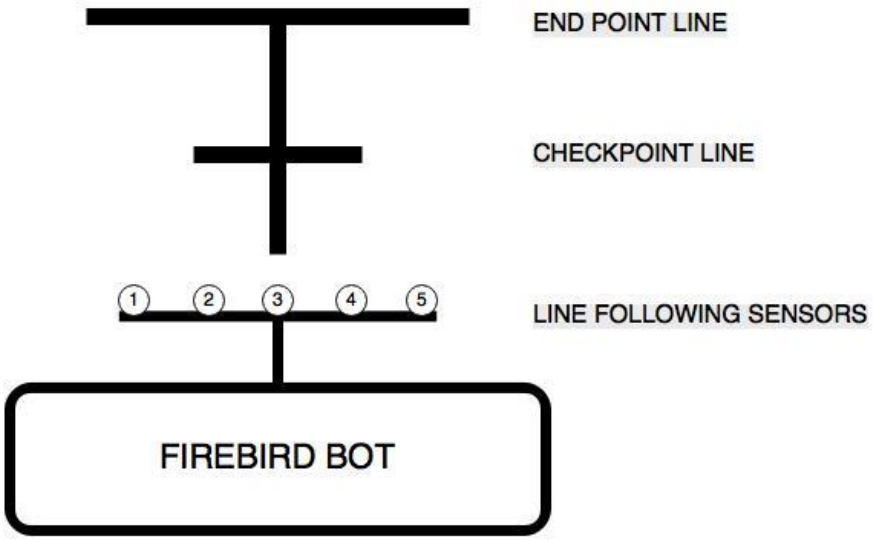


Illustration 5: Checkpoint detection

Obstacle detection

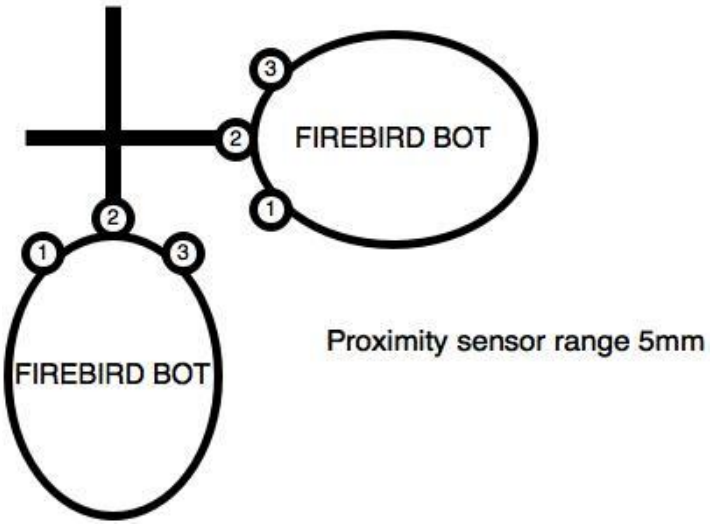


Illustration 6: Obstacle detection

Weight detection. The weight detection being implemented by using IR sensor No1 Whenever order is kept on the bot, IR sensors is blocked and generate the signal that order has been placed and when bot reached the table and order is lifted, it detects it and returns back .

5. Working of the System and Test results

Implementation setup. Each table is numbered and table position is fixed. There is WAMP server which contains runs the restaurant website and contain database for android app. The zigbee is connected with this laptop for communication with bots. Once the customer arrives and he will place an order on android tablet kept on the table using Android app.The order will be displayed at the counter webpage. Once the order is prepared, the counter will click on that table number. Once it is clcikd, python script will run in background to communicate with the bots using Zigbee. The python script will check which order is free and establish communication, pass table number and the free bot will collect the order and then the bot will deliver it to the table as per the input table number. On delivery of the order, it will return to the counter automatically to attend the next order. Navigation to the desired table would be on pre-defined path based on X-Y grid. In case of obstacle enroute, it will re calculate the path and use a new path. This is achieved by separating onward and backward paths and assigning different priorities. The same was done by using round robin scheduling algorithm for tasking bots and carefully designing onward and backward path from start point to destination and planning re-routing algorithms in real time to avoid collisions and deadlocks by using techniques similar to exponential backoff algorithm used for collision avoidance in computer networks using variable delays.

The laptop running WAMP server and connected to Zigbee:-

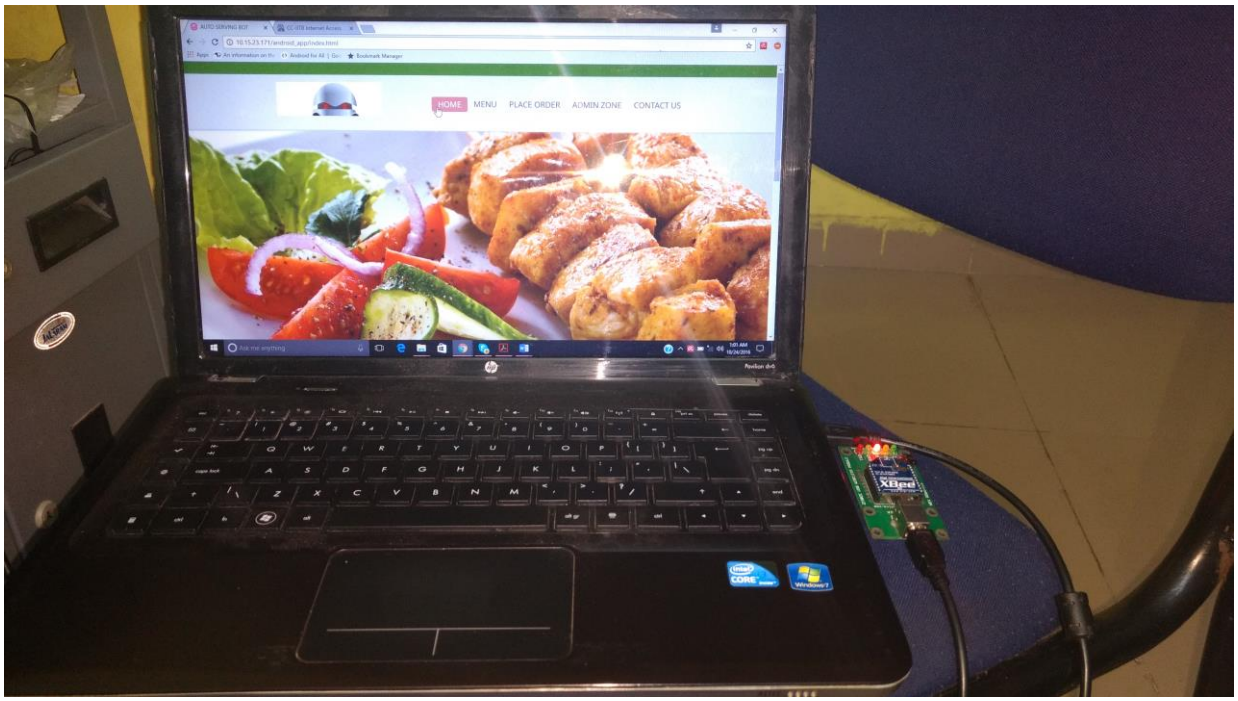


Illustration 7: WAMP server

The homepage of the website:

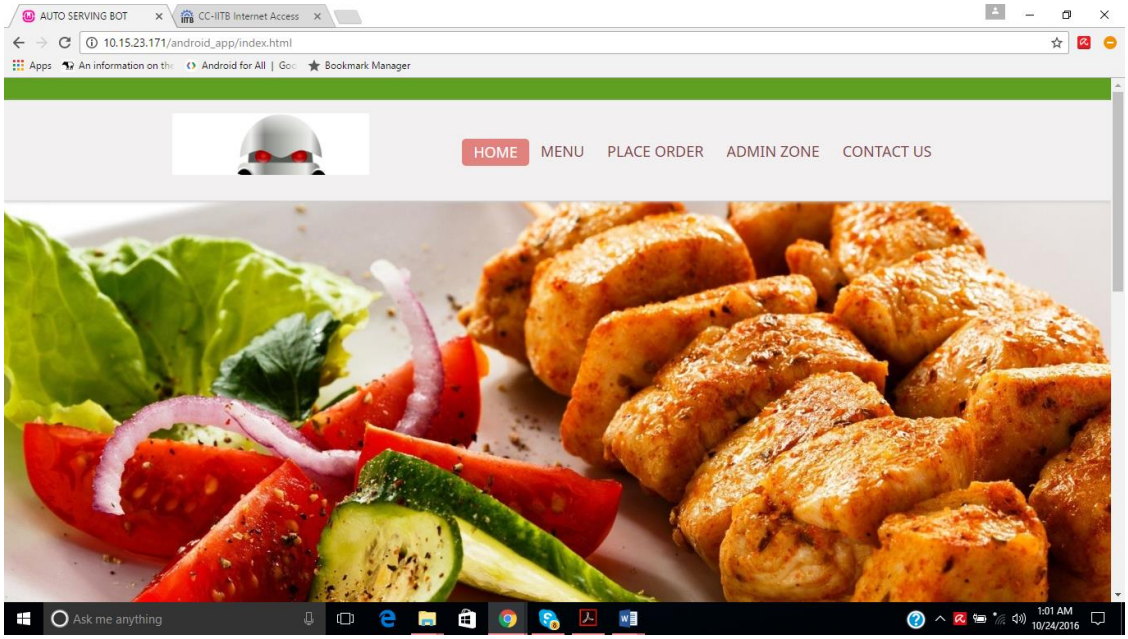


Illustration 8: Website

The option to view menu in the website

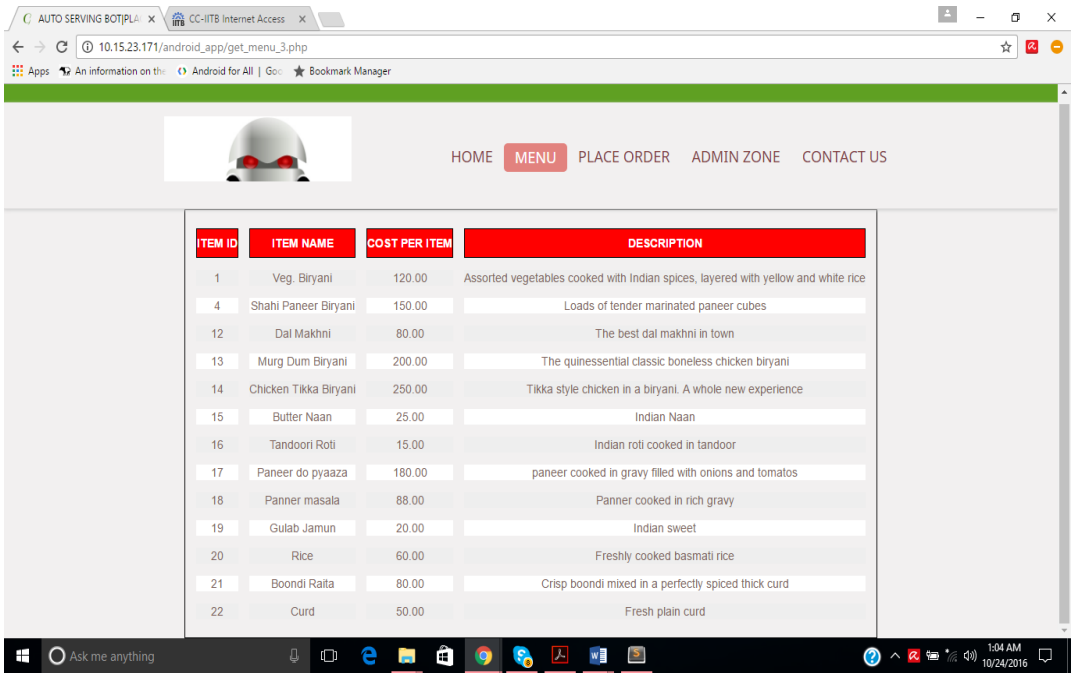


Illustration 9: Menu

The page to place order from laptop or mobile web browser:

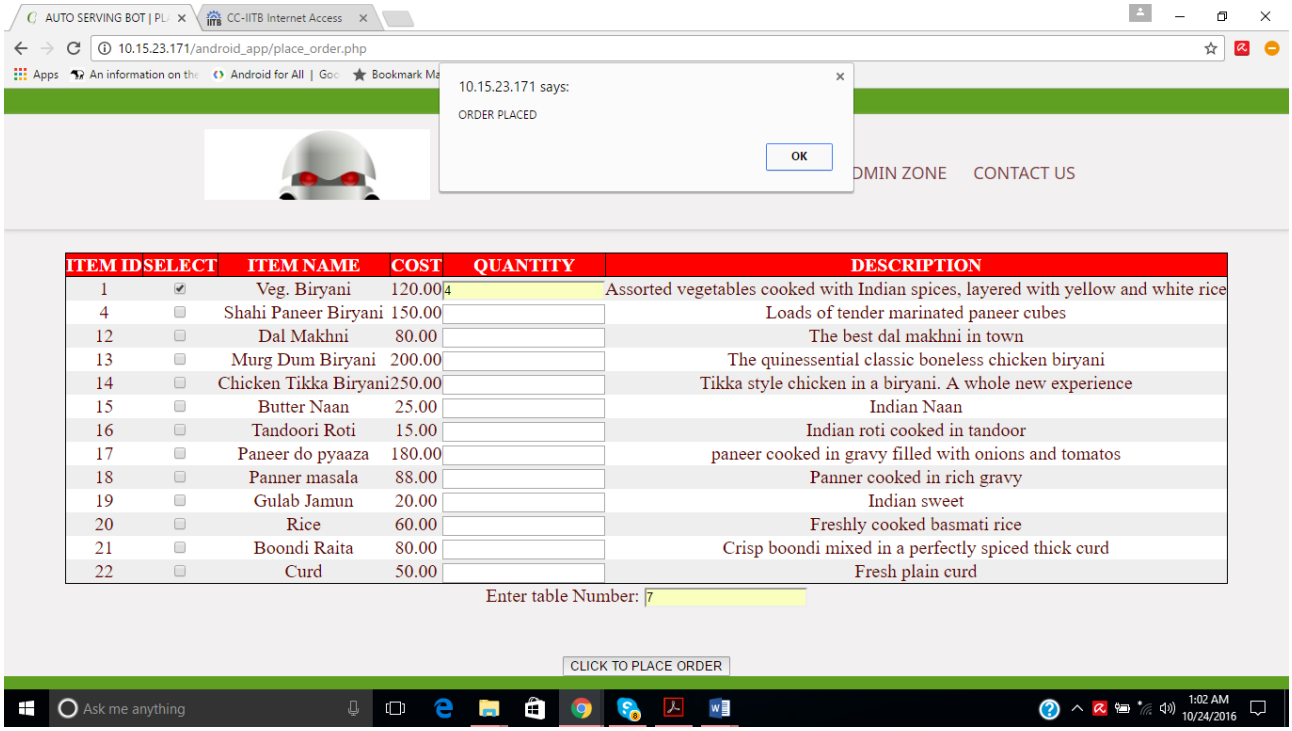


Illustration 10: Placing order

The android app **"Food buddy"** to place orders online using android mobiles: For testing of the placing order from the android application, each entry the user makes has an error checking to check the correctness of the input in the context. Also different combinations of inputs which can generate errors are tested and appropriate toast messages are shown to the user.

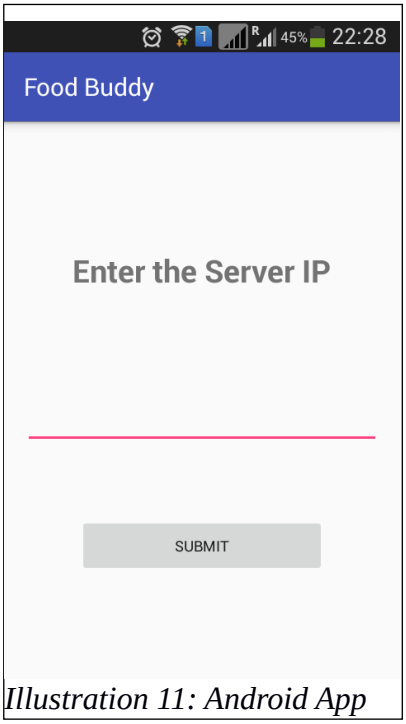


Illustration 11: Android App

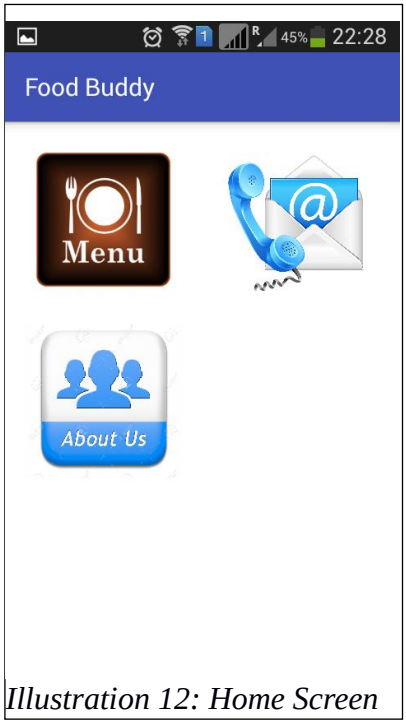
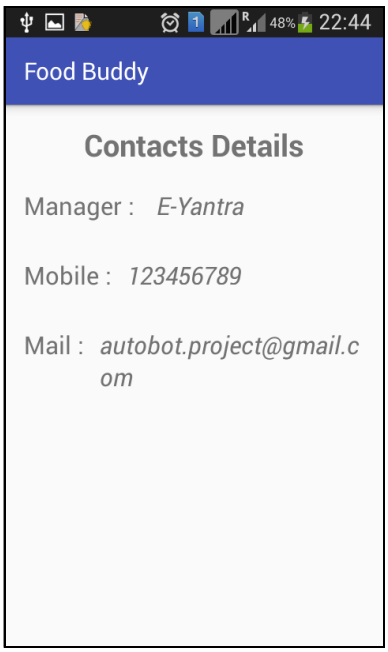
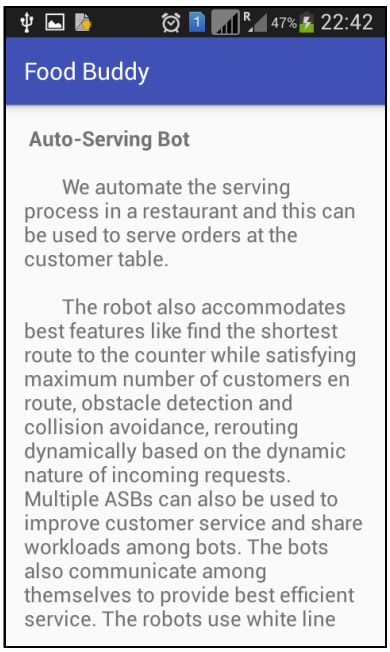


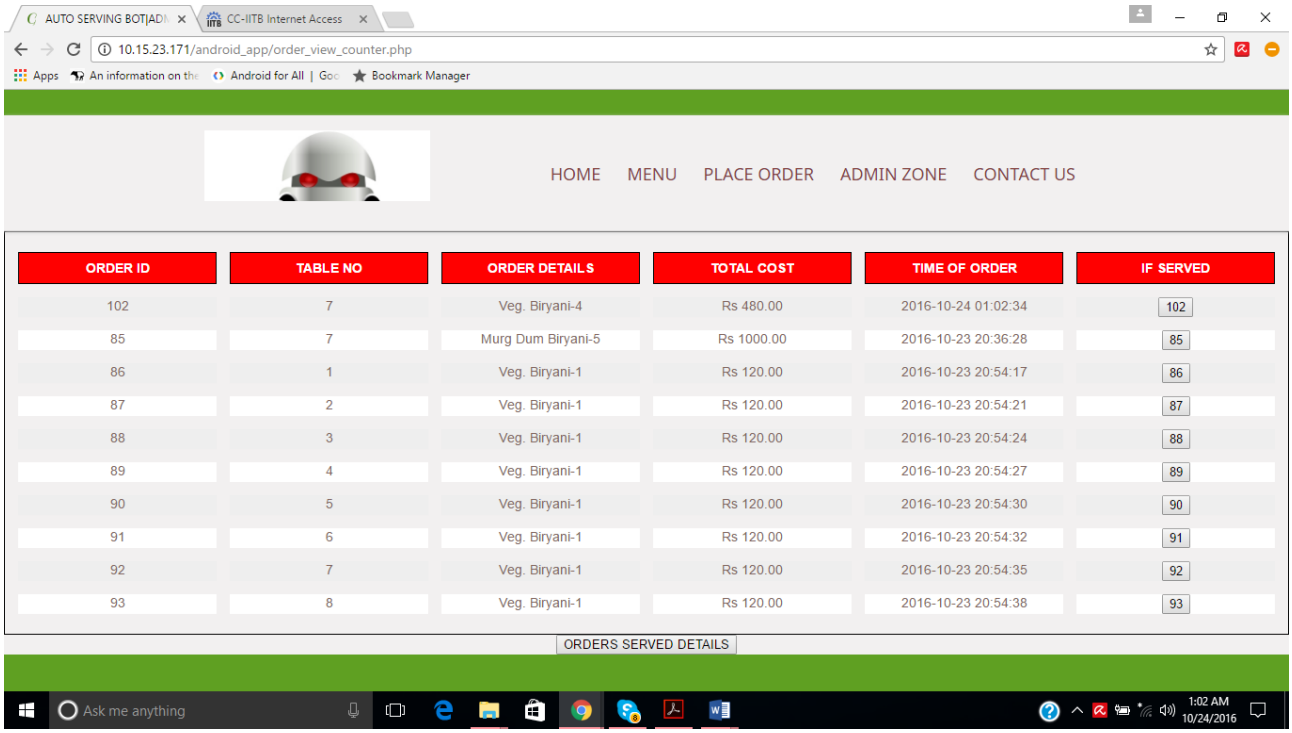
Illustration 12: Home Screen

Illustration 13: Menu

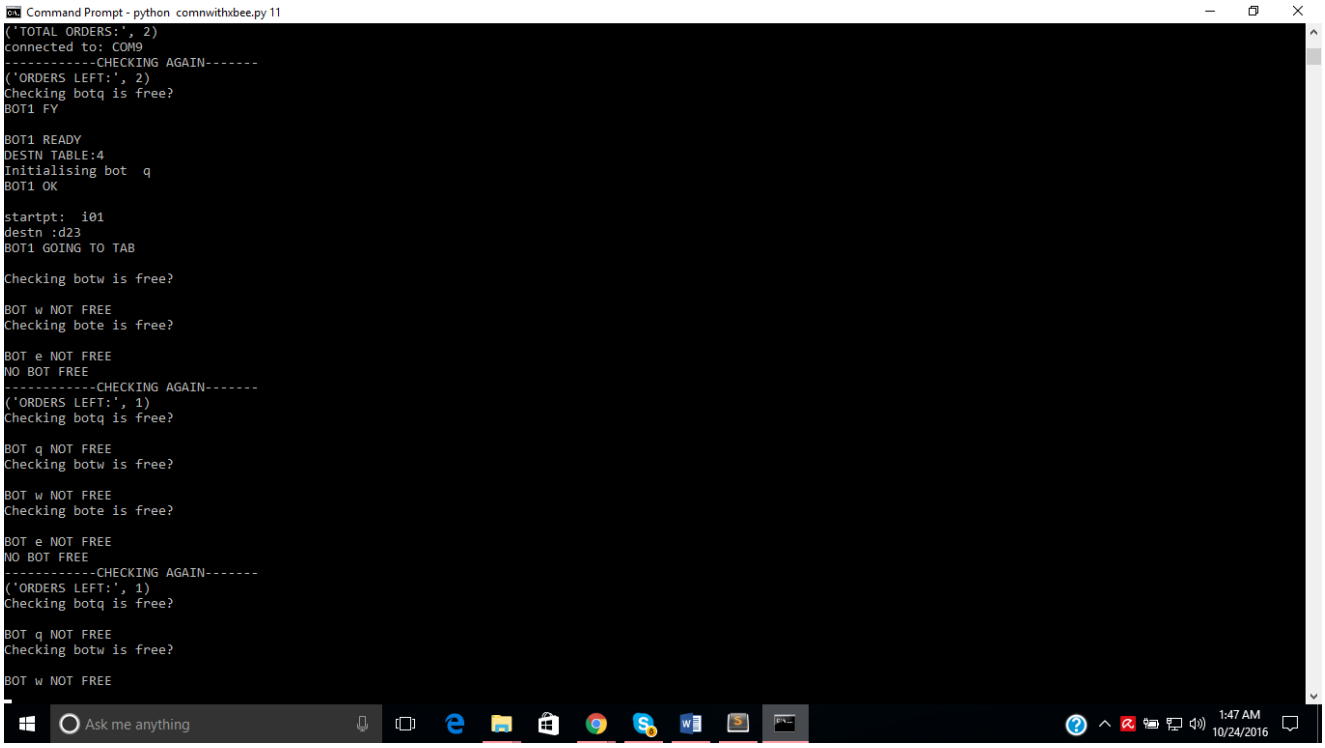




The admin page to manage orders. If order prepared, admin will click on the table ID and python script will run in background to communicate with bots:

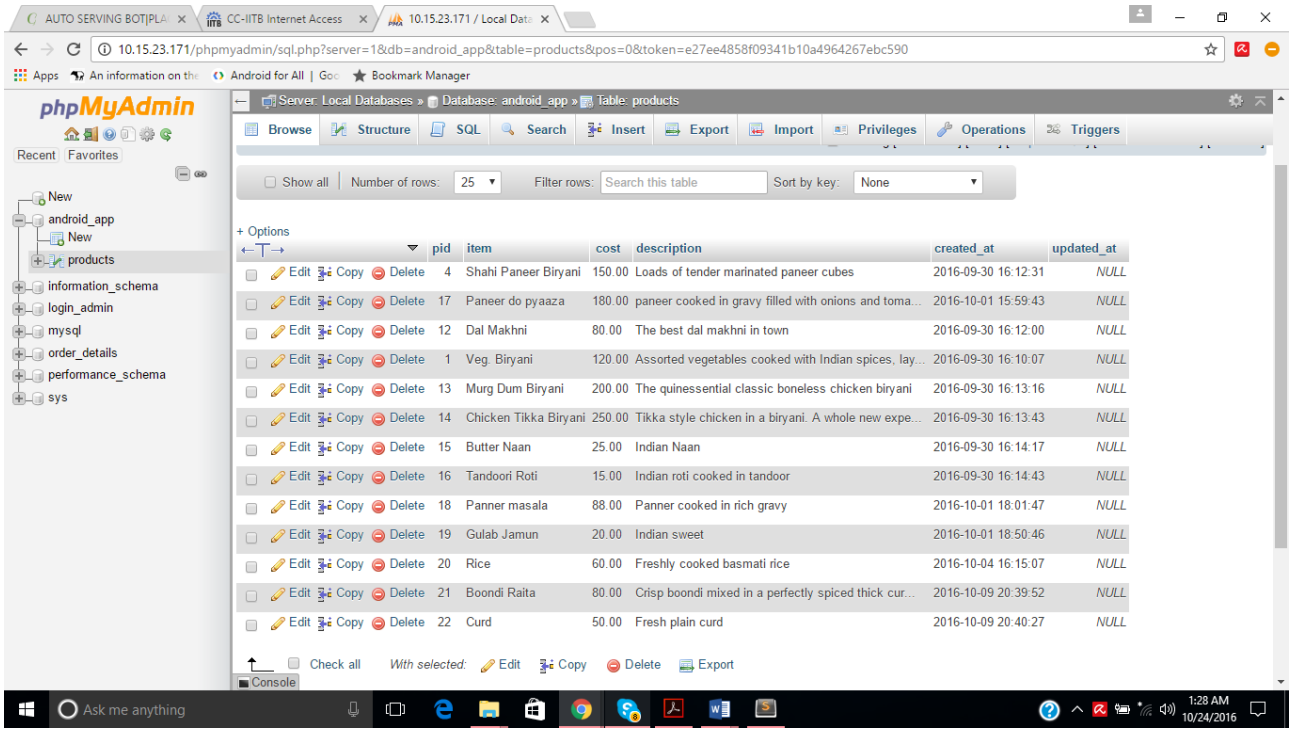


The python script running in background to check which bot is free and then giving source and destination to free bot

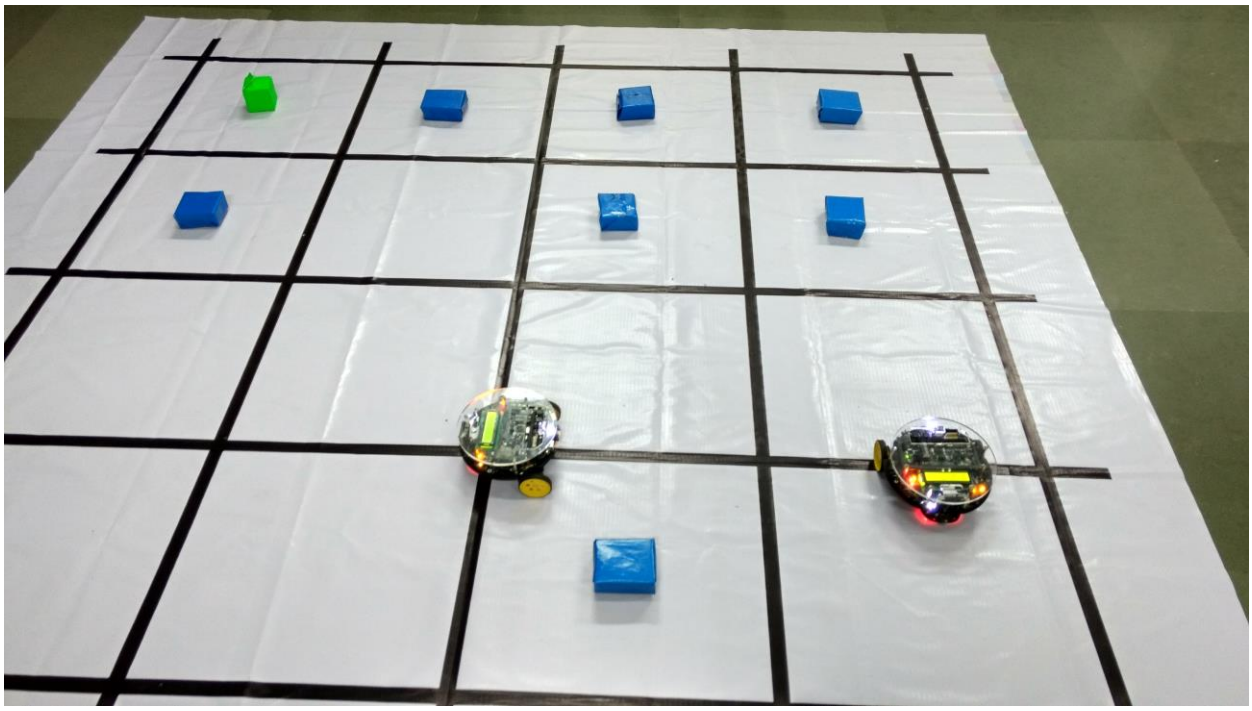


The admin page to view which order is served, the time of delivery and bot details who served the order:

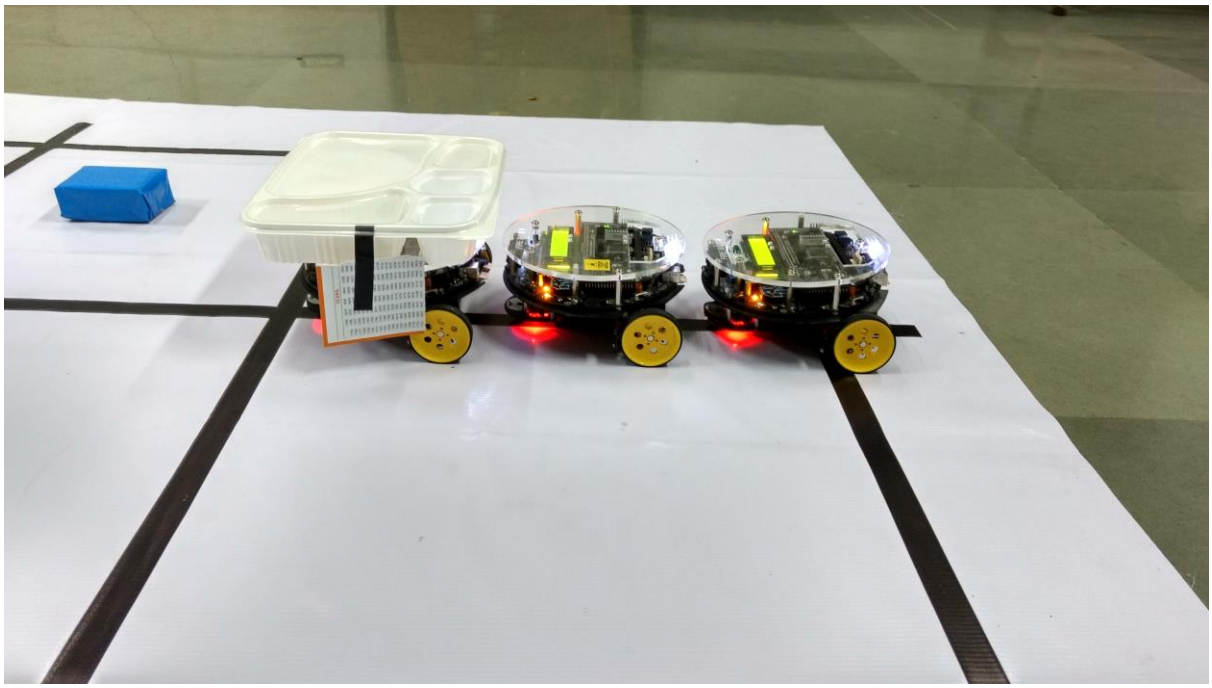
The php database view for admin:



The implementation arena:Grid of 4x4 matrix. Each grid contain one table as indicated by color boxes.



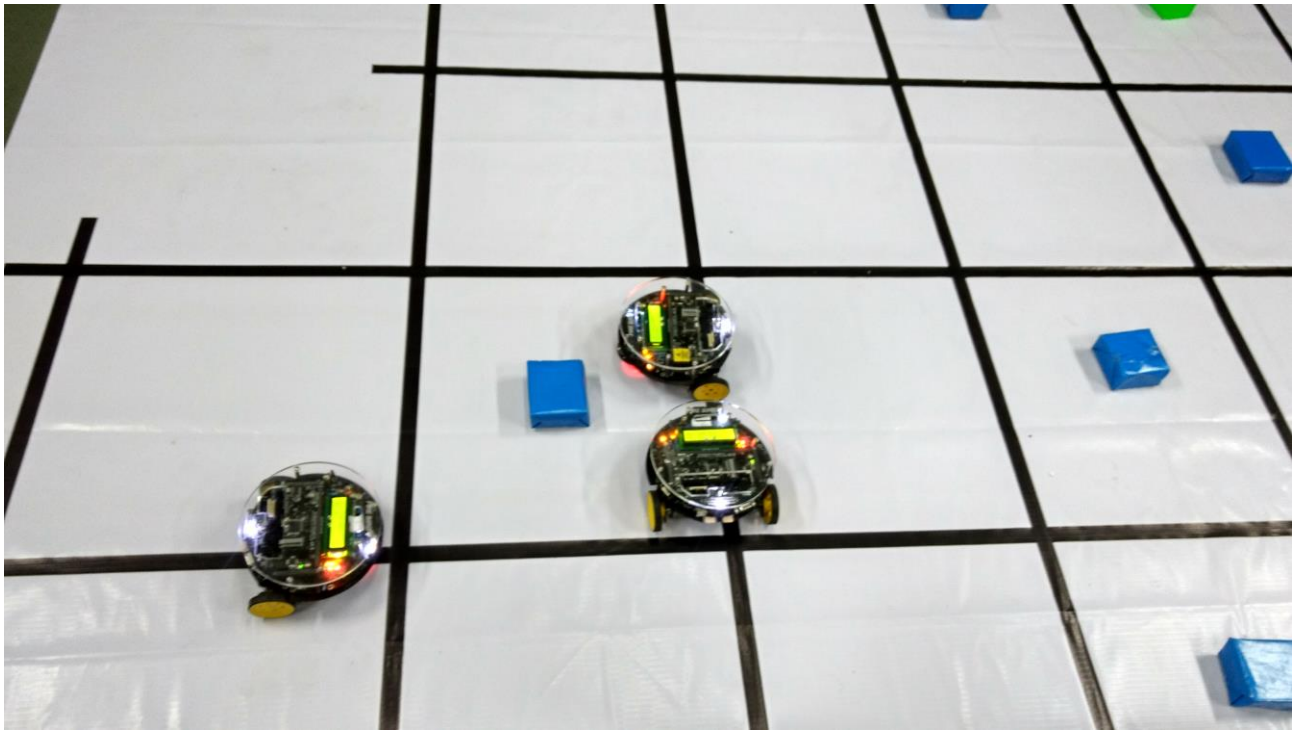
Bots lined up at counter for receiving orders. Initially all bots will be lined up for receiving orders. Once the order is received, it starts moving towards the table and clearing the path for the subsequent orders.



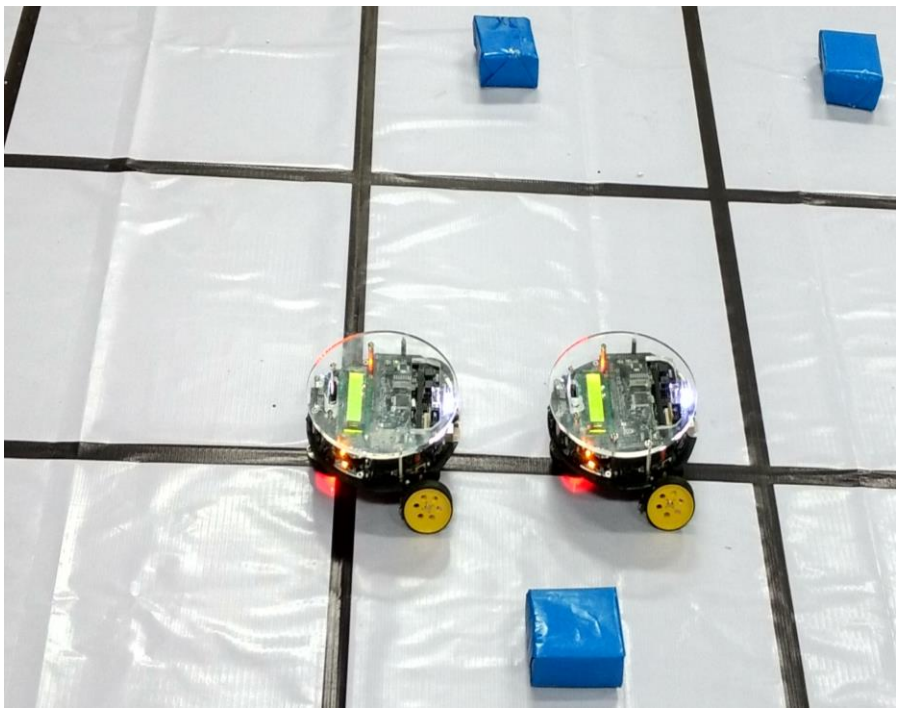
Collision avoidance/ rerouting:

Testcase1 : Following picture depicts the scenario, where in the first bot is delivering the order at the table, and the destination for the second bot is also comes in the same path. As the second bot knows that the path will be cleared immediately after first bot delivers the order, it doesn't reroutes itself and waits for bot1 to finish its delivery of order. First bot,

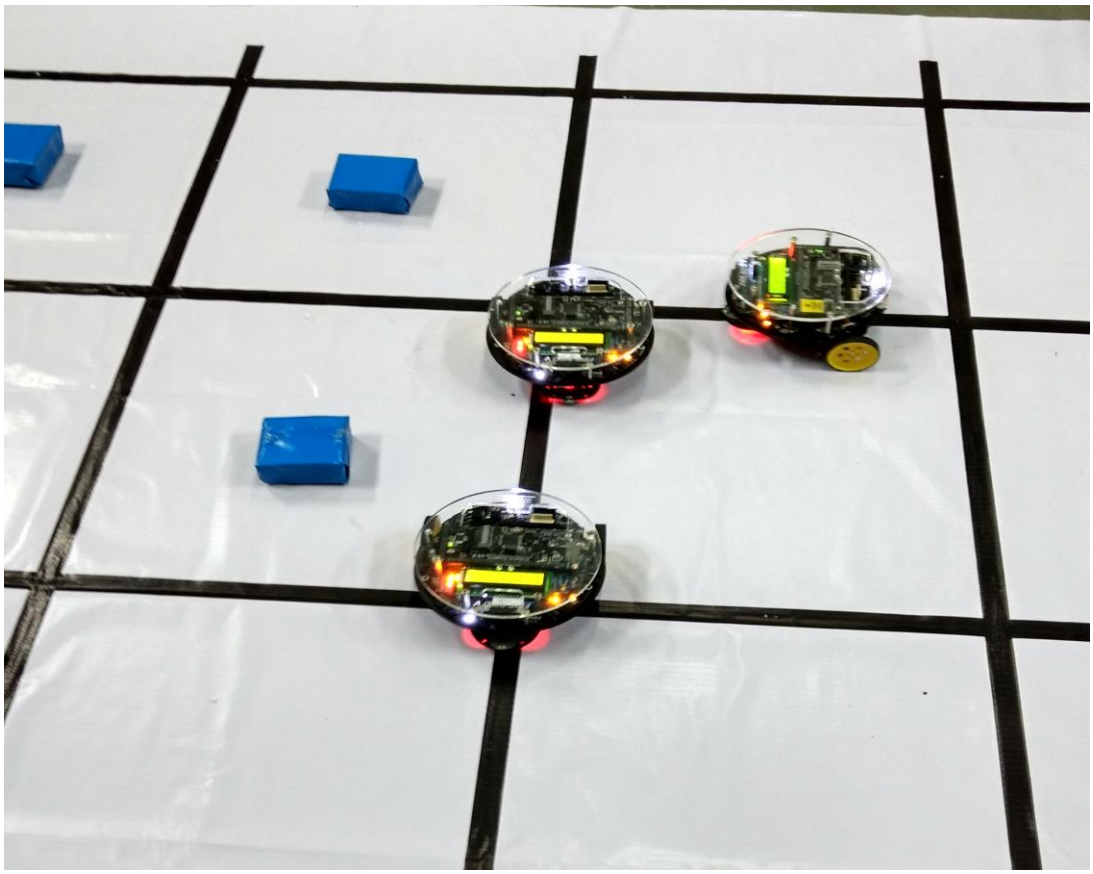
after delivery, falls back to its return path thus clearing the path for the second bot and then second bot will continue towards its destination table. Sharp sensors and IR sensors stops the second bot from colliding to first bot.



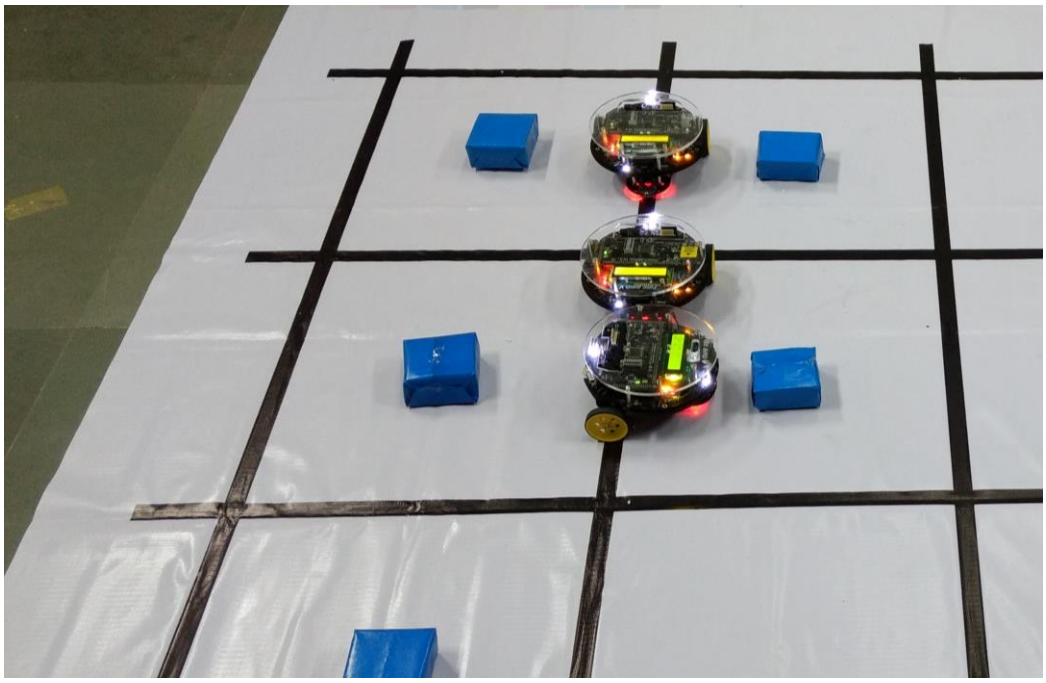
Testcase2 : If two bots are moving along the same path, the following bot will control its speed and follow the ahead bot to avoid collisions(Adaptive cruise control).



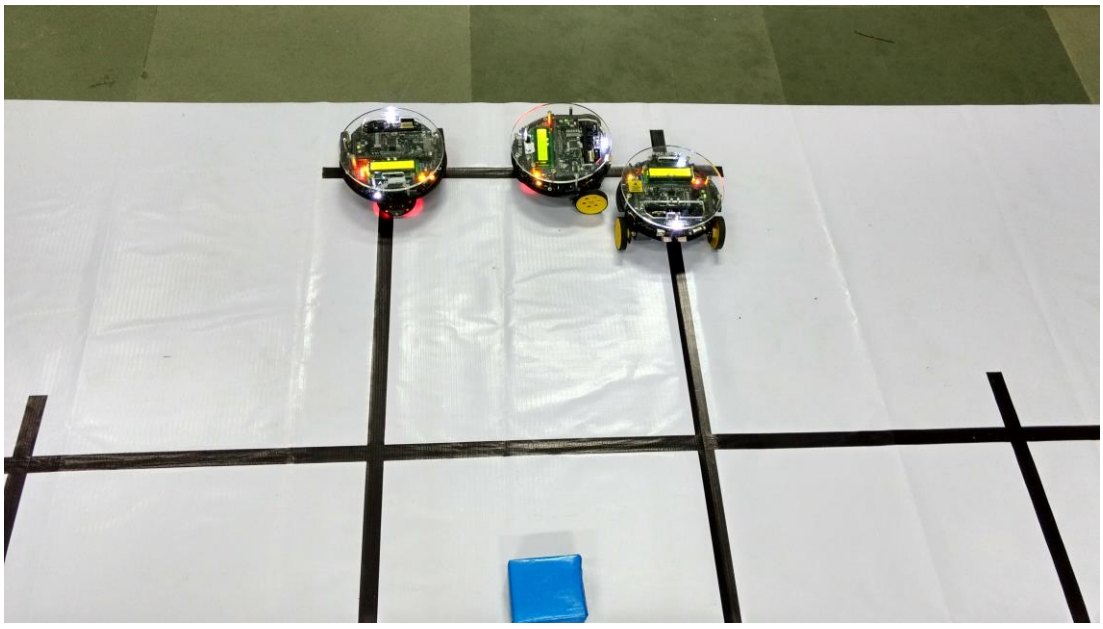
Testcase3 : If two bots are on the same path, then the following bot will wait on the turnings to avoid collisions.



Testcase4 : The following picture depicts a situation where in the first bot1 has already delivered the order and returning back to counter, bot 2 is going to deliver the order and bot 3 is delivering at the table(reached the table and waiting for the customer to pickup the orders). So the bot 2 is stuck in between bot1 and bot3. As the return path is blocked for bot1 and delivering order is more important than returning to the counter, bot1 reroutes itself and takes a 180 degree turn and follows the outermost path and reaches the counter. Bot2, as it knows that the path would be clear once bot3 starts moving towards the counter after delivery, so bot2 doesn't reroutes itself and waits till the path is clear. Once order lifted from bot3, Bot3 will move towards the counter and bot2 will continue to move along its earlier path towards destination table.



Bots lined up at counter after delivery of orders for receiving next orders. Only first bot will be in ready state and rest two bots will remain busy state to avoid collisions and to ensure order is given only to one bot.



6. Discussion of System

a) What all components of your project worked as per plan?

- Making an android app for placing orders.
- Installing WAMP server and maintaining database of menu items, orders placed and delivered.
- Passing destination table coordinates to a free bot from admin webpage using python.
- Communicating with bots using Zigbee.
- Navigating to the table and coming back to counter (Black line following) using grid system.
- Avoiding collisions using IR sensors and proximity sensors and rerouting in case of obstacle detection using exponential backoff technique with different delays and finding new route using rerouting algorithm.

b) What we added more than discussed in SRS?

- The extra online web ordering system by which orders can be placed using our website for non android users. Such users can order directly using web browser.

- The online website where admin can manage orders, view orders delivered and add/remove items to menu list.

c) Changes made in plan from SRS:

- For the testing purpose of the database system and users not having android phones, we have created an extra online web ordering system by which orders can be placed using our website also.

- Instead of using additional proximity sensors, we used existing IR sensor on bot for checking whether order was lifted or not. The left IR sensor no1 was used for the same innovatively to avoid making any changes in the existing bot.

7. Future Work

1. Our work can be improvised by replacing the proximity sensors with weight sensor to check if the order tray has been picked up by the customer.
2. A Robotic Arm can also be used to deliver the order at the customer table.
3. The Android application can be improved by adding a “status of the order” feature to track the live status of each item.

4. The theme can be made more interesting by making multiple counters and making bots decide dynamically which counter to go based on counter call and the distance from it.

8. Conclusions

"Auto Serving Bot" can make dining experience more satisfying when implemented in real life. This is alternative to existing dependence on human resource whose behavior is not predictable and prone to errors. This solution is economically feasible as it is one time investment with no recurring cost and drawbacks like pay hikes or bonus or strikes. The existing solutions are costly and are not been used extensively. This solution can be further extended to other hospitality services like supermarkets for moving items between various sections or hospitals to deliver medicines or reports from one department to other or in big warehouses for movement of goods.

9. References

[1] http://www.chinadaily.com.cn/business/2014-11/26/content_18978177.htm [2016].

[2] https://github.com/eyantra/Autonomous_Waiter_Robot_using_Firebird_Tmega2560