CS-308-2014 Final Report

CS 308 Project- Restaurant Bot

Group-5

Shubham Jain, 120050002

Palash Kala, 120050010

Vikas Garg, 120050017

Amol Agarwal, 120110031

Table of Contents

1. Introduction

When we order our food in restaurants, it sometimes takes a lot of time to deliver because of inefficiency of waiters and we have to wait for waiters to deliver our food even it is cooked.
Also it’s costly for restaurant owners to put more and more waiters. The waiters expect a tip from the customers and this goes to about 10-15% of the bill amount in the US.
Moreover when there is heavy load, there’s a lot of chaos, with same order being repeated multiple times to different waiters.

2. Problem Statement

Our solution to this problem is having restaurent bots instead of waiters to deliver the food to customer.
People can order through an interface (An app or a web interface).
Food will be prepared accordingly and given to our restaurant bot.
We will use line follower model of moving on white line on black surface for demo purposes . For controlling multiple bots, we have a central system, which will decide the path for every bot. For every table we can have a position where the bot will reach and customer will take the food from the bot.

3. Requirements

3.1 Functional Requirements

Inputs to the system:

- The table number to be served
- The bot to which above table is assigned
- The food should be placed on required side of the bot
- Map of the hotel along with table numbers
- Address of XBee module used with central server
- White line input from the sensors on Firebird
- Number of bots
- Maximum number of tables
- Maximum number of dishes
- Maximum number of sides on a bot where food can be placed

Output:

- The allocated bot calculates the optimal path to the destination
- Instruction at each node on where to go
- Bot follows white line to reach the destination

3.2 Non-Functional Requirements

- Sufficient and obstacle free space for bot movements
- Optimum lighting condition, for proper functioning of IR sensors/white line detection
- Sync between switching on the bot and the server.
- Sync between order assignment to ensure no deadlock

3.3 Hardware Requirements

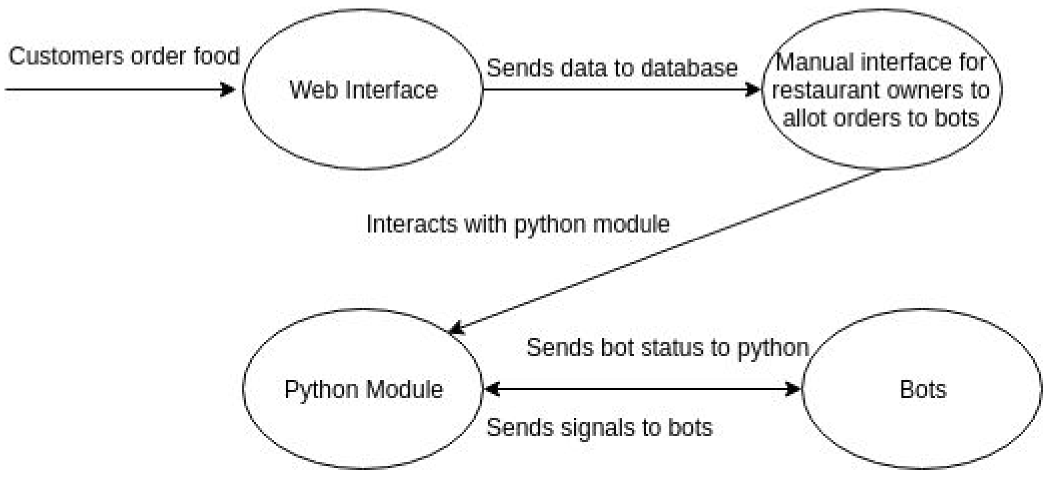
Component	**Quantity**	**Use**
FireBird V	2	Act as bots to serve food
Robotic Arm	2	Two push food on tables. One one each bot
Xbee Module	3	Communicate between Central Server and bots
XBee USB Adaptor	1	Interface Xbee on PC
Arena/Map with White lines	1	To move the bot on
A central system (laptop)	1	To give the commands to the bot

3.4 Software Requirements

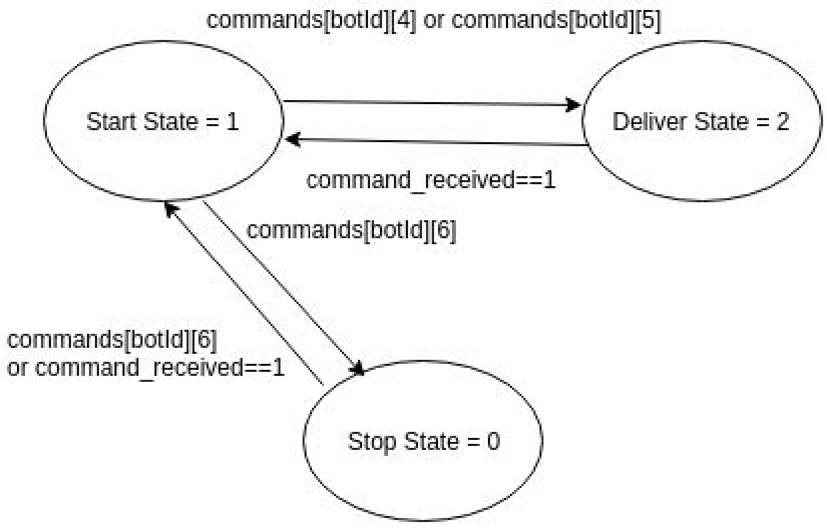
Software	**Use**
AVR Studio/ CodeBlocks	To write and compile firebird code
avrdude_script.sh	To burn the hex file on bot
Python	To run the server side code
XCTU	To configure the XBees

4. System Design

An architecture diagram describing the components of the system and their relationships:


Customers order food
Web Interface
Sends data to database
Manual interface for restaurant owners to allot orders to bots
Interacts with python module
Python Module
Sends bot status to python
Sends signals to bots
Bots

Overall design of your system and include all diagrams related to our work - FSMs, Statecharts, Circuit Diagrams, Snapshots of the mechanical parts.



5. Working of the System and Test results

Firebird Control

- Firebird receives commands at each node, where to move next
- In between nodes, it follows white lines
- On reaching respective tables, the bot receives a command to serve the food
- It pushes the food using the robotic arm. (servo motor control)

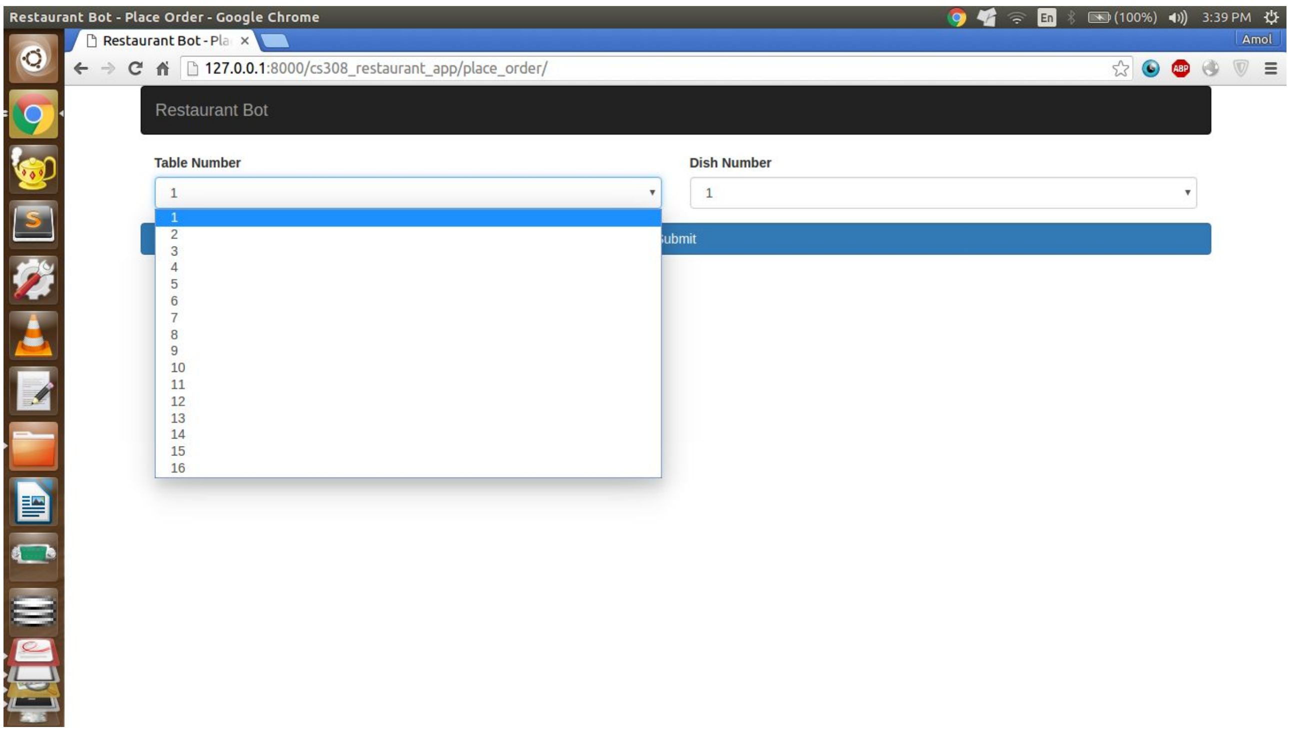
Central Server

- Central Server polls each bot at regular intervals and relays command appropriately based on received status
- There is a web interface to place orders, and assign it to a specific bot
- There is also an API to communicate with data-store about the order specifications and update bot positions

Communication between Server and Firebird

- Communication is achieved using XBee Wireless Module
- Xbee on central server is set on Broadcast Mode
- Xbee on firebirds have their destination set to Xbee on central server
- Each bot acts on specific set of independent commands to which the other bot doesn't act

Web interface screenshots:



Selecting Table No. and Dish Number - Placing the order

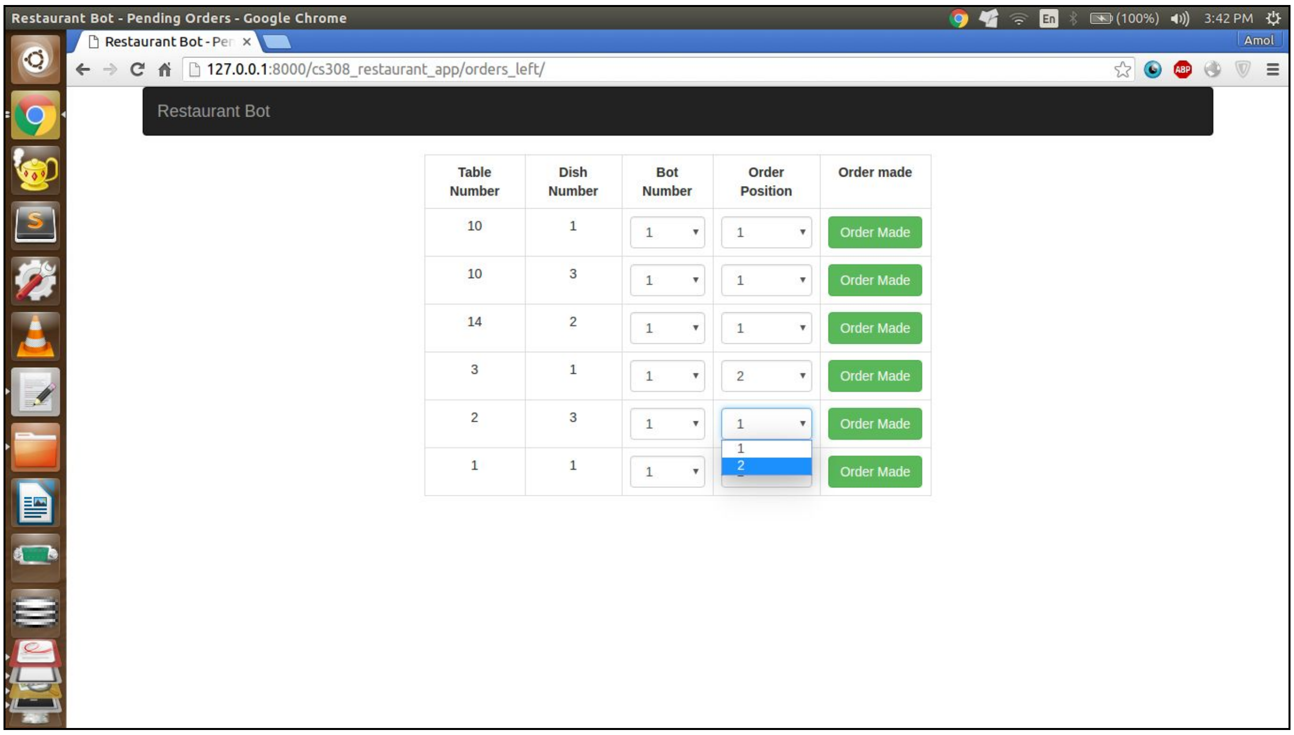
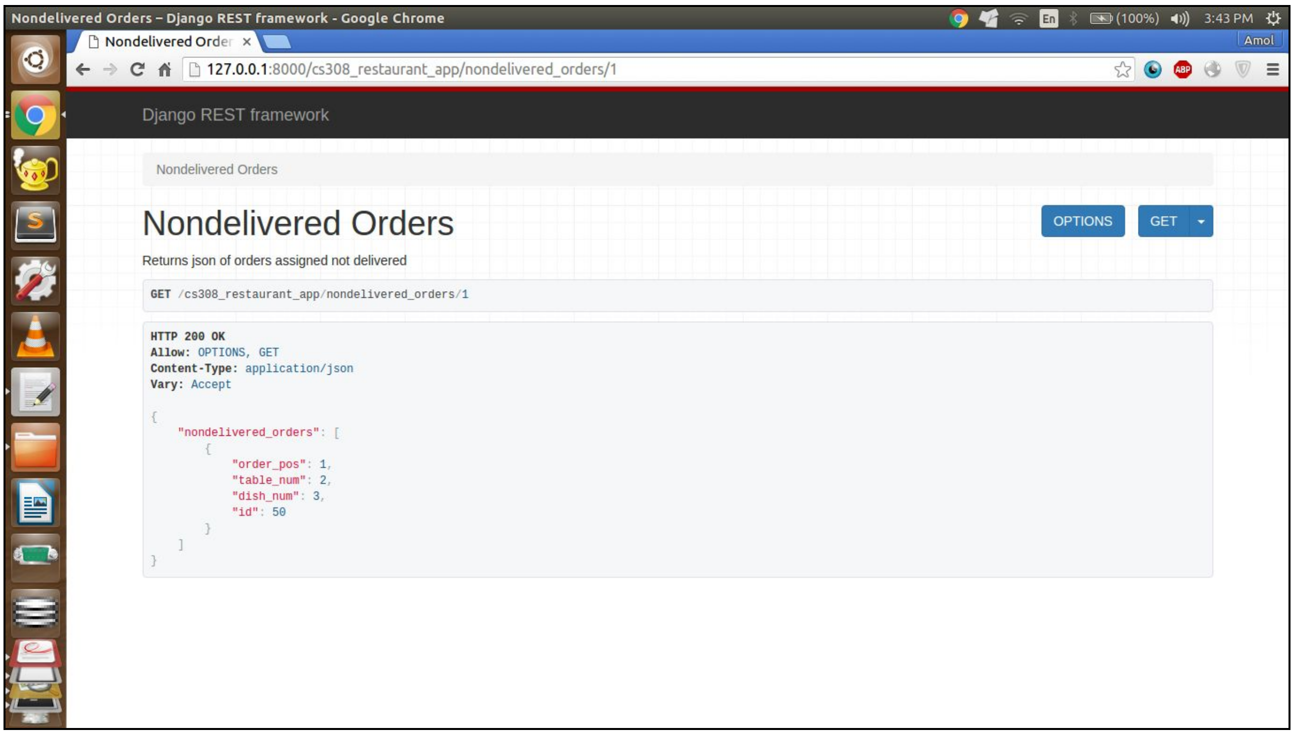

Restaurant Bot - Pending Orders - Google Chrome
Restaurant Bot - Pe
127.0.0.1:8000/cs308_restaurant_app/orders_left/
Restaurant Bot

Table Number	Dish Number	Bot Number	Order Position	Order made
10	1	1	1	Order Made
10	3	1	1	Order Made
14	2	1	1	Order Made
3	1	1	2	Order Made
2	3	1	1	Order Made
1	1	1		Order Made

Chefs Interface, select which order has been made and allot it to the bot


Nondelivered Orders – Django REST framework - Google Chrome
Nondelivered Orde
127.0.0.1:8000/cs308_restaurant_app/nondelivered_orders/1
Django REST framework
Nondelivered Orders
OPTIONS
GET

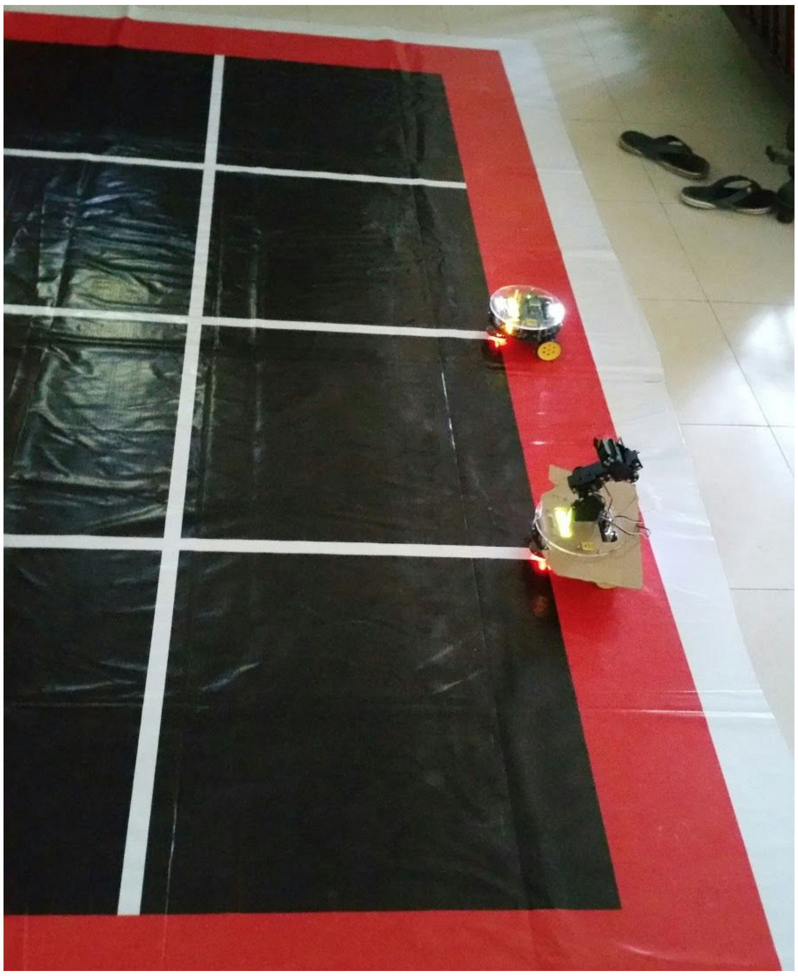
Nondelivered Orders

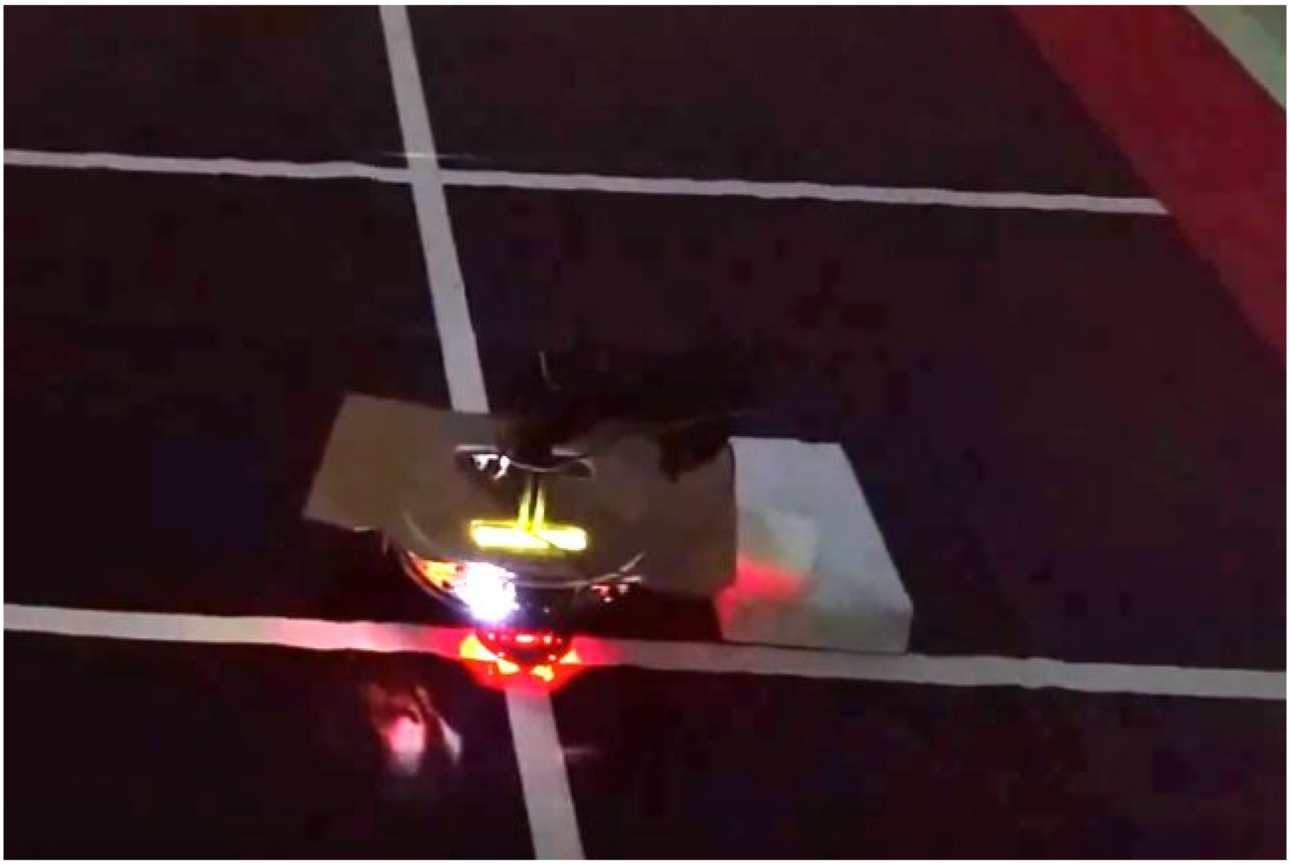
Returns json of orders assigned not delivered

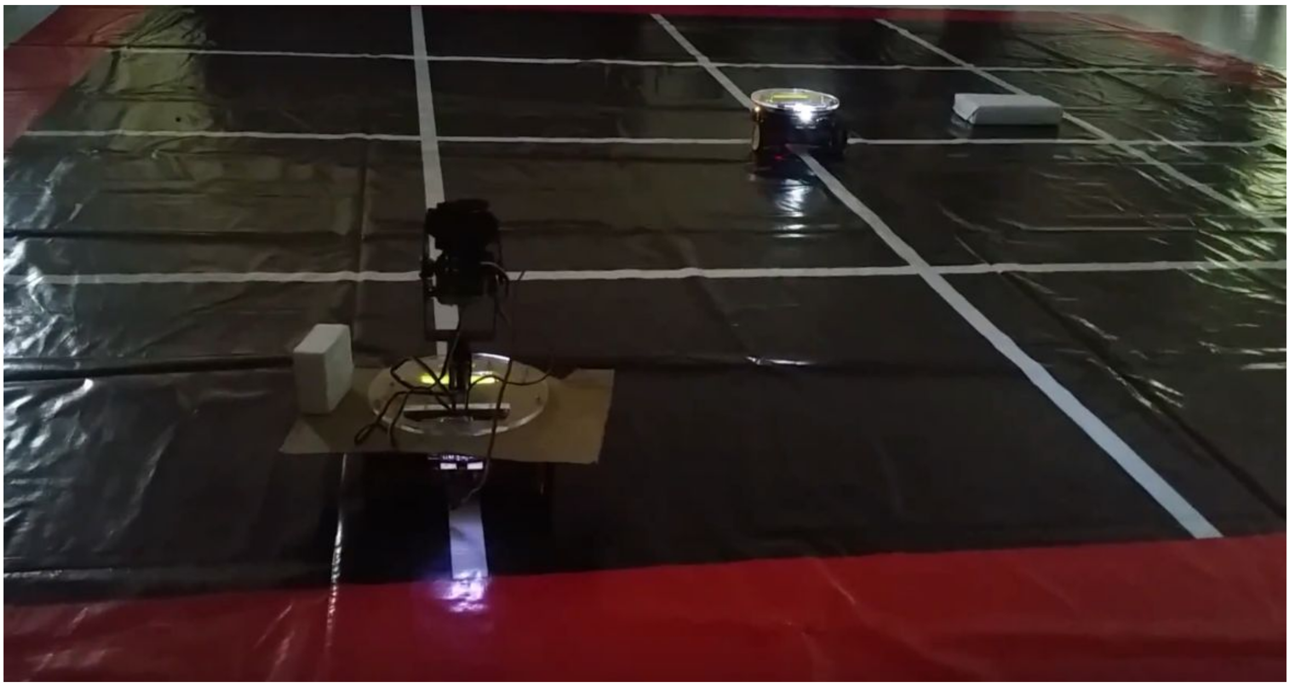
```
GET /cs308_restaurant_app/nondelivered_orders/1
```

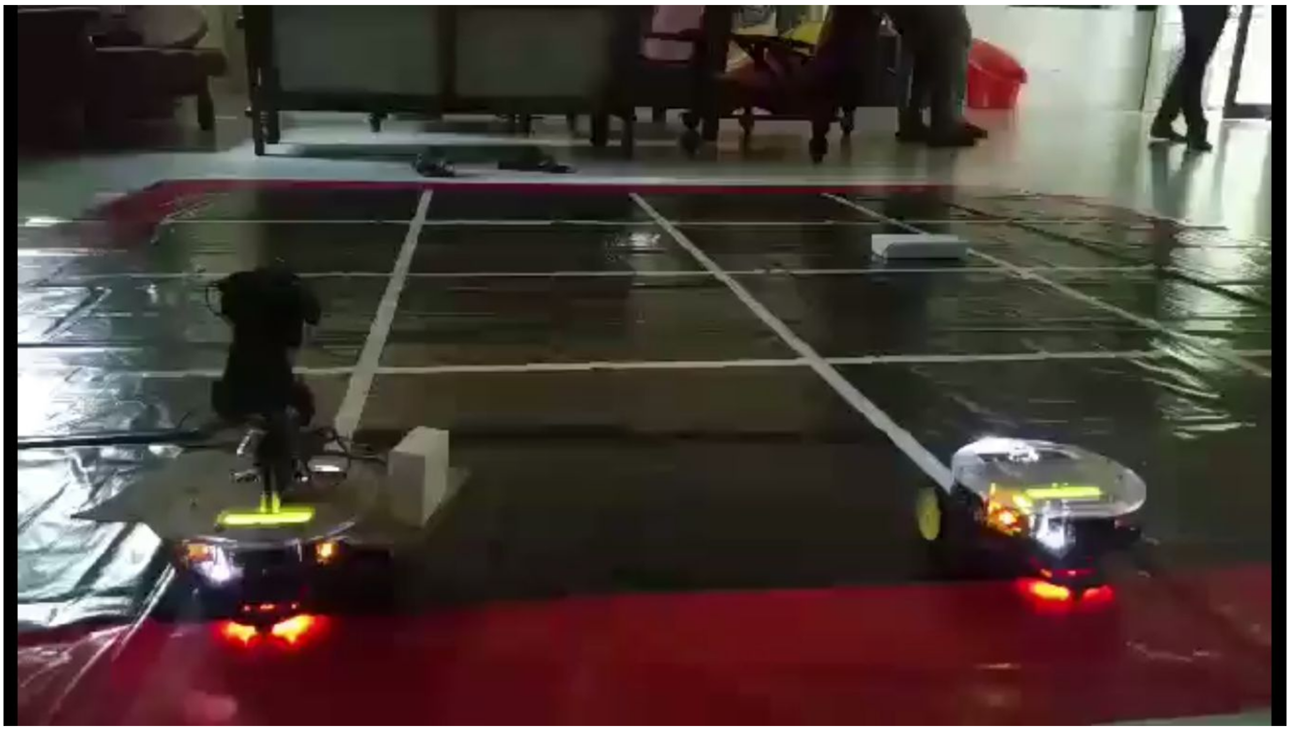
```
HTTP 200 OK
Allow: OPTIONS, GET
Content-Type: application/json
Vary: Accept

{
    "nondelivered_orders": [
        {
            "order_pos": 1,
            "table_num": 2,
            "dish_num": 3,
            "id": 50
        }
    ]
}
```

See the nondelivered orders which either bot is carrying or bot has been allotted but he has not delivered


Bots on initial positions


Bot 1 serving food


Bots in motion


Bots after coming back

Testing of the System

To test the system,we started the server, XBee module and the bot (in the said order). We ordered for a table from the interface. The bots were kept at their initial position and the when we assigned the orders to a bot, the order was assigned and bot moved. We kept a piece of thermocol which was treated as food and pushed by the robotic arm.

To test XBee module we used XCTU software by Digicom. We used it to configure the XBee modules and send it a frame of packet and check does the bot receive it and how does it react to it and thus verify the code on bot.

To test the web module, we run it on a local host and use SQLite Database Browser to see if changes happen in database and whether our App works. To test the APIs we used POSTMAN extension on Google Chrome.

6. Discussion of System

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

i. Communicating with bots from central server via Zigbee Protocol. Created a heuristic where both the bots got different command signals and there was no interference in a bot's motion due to a signal given to the other bot

ii. Bot could not send signals to the server on its own, created buffer problems
    1. We created a signal which when sent to the bot, the bot replies with the mode in which the bot is present currently
    2. After checking the mode, the server replies with the action to take
    3. Hence, it was a three way signaling, rather than a two way signaling

iii. A web app for receiving orders and another portal for restaurant owners to allot orders to the bots. Web app for automatic receiving and serving was difficult because, proper scheduling and flawless motion of the bot was required for that which was not the case
    1. Hence, we divided the problem into two portals, one for ordering and other for allotment
    2. Allotment is done manually by the restaurant owner

iv. The bot pushes food on reaching the node, but tables are not fixed as they obstruct the motion

v. The bots do not collide, but deadlocks haven't been avoided. Paths of the two bots intersected
    1. We had the path beforehand for both the bots and if both the bots had the same next node, we gave the first bot a higher preference and let him complete the path first, while the other bot waited
    2. When both the bots had their next node as the other bot's current location, we haven't avoid this deadlock

vi. People had to choose their order and table number from a good GUI, where they could see their table
    1. Instead, we had table numbers on the table and customers could choose their table from a dropdown

b. Changes made in plan from SRS:

i. We proposed to have bot serve two tables in move from kitchen and then come back. Currently a bot is assigned only one table at a time and has to return to the kitchen after doing so. Our state of bot is designed for accomodating one destination at a time. We can change it to add a new destination after first destination and then return to kitchen. But we didn't had enough time to add more complexity and test as current system in itself wasn't bug free, but went through lot of iterations. So it was a design choice we took.

ii. We proposed to create a GUI as in bookmyshow for ordering food. But instead we created a simple form, because creating such a bug free GUI required lot of frontend efforts for which we didn't had enough time left.

7. Future Work

Reusable features/components

- All components are reusable
- Web app can be reused for some other purpose as the framework is modular and all functions maintain modularity which can be changed a bit and reused
- We have created a class to represent a Bot, Map and XBee communicator module which

can be reused with any other program
- The code of line follower can be used for line following
- The robotic arm code can reused for movement and learning about robotic arm
- The method used for XBee configuration and for communication can be used fr communicating amongst various bots

Possible Extensions

- Extending the system to handle more than 2 bots
- Improving the food serving mechanism, so that it serves accurately on the assigned table without manual readjustments
- Currently a bot is assigned only one table at a time and has to return to the kitchen after doing so. It can be extended to deal with 2 orders at a time and even more if we can somehow figure out food placement on the bot
- Collision detection using Sharp sensors
- Deadlock avoidance in collision avoidance

8. Conclusions

- We have tried to automate waitering process upto some extent.
- Few bots, a central server machine, and a map is all it takes to make the system work
- Our system is capable of receiving and delivering orders to the customers.
- The system is easy to understand and can be improved upon to include scheduling optimizations.

9. References

- DB Browser for SQLite
- XCTU tutorial
- EYantra-DVD

10. Link to video

The Restaurant Bot Video