https://github.com/ajay/ROSE

Final Report

Software Engineering (14:332:452)

Professor Ivan Marsic

Group 9

Ajay Srivastava, Srihari Chekuri, Cedric Blake, Brice Howard,

Vineet Sepaha, Neil Patel, Jonathan Cheng

Table of Contents

Individual Contributions Breakdown

		Team Member Name							
Responsibility Levels	Points	Ajay Srivastava	Srihari Chekuri	Cedric Blake	Brice Howard	Vineet Sepaha	Neil Patel	Jonathan Cheng	Totals
Summary of Changes	5	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 1: Customer Service of Requirements	6	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 2: Glossary of Terms	4	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 3: System Requirements	6	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 4: Functional Requirements Specification	30	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 5: Effort Estimation	4	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 6: Domain Analysis	25	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 7: Interaction Diagrams	30 + 10	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %

Section 8: Class Diagram and Interface Specification	10 + 10	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 9: System Architecture and System Design	15	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 10: Algorithms and Data Structures	4	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 11: User Interface Design and Implementation	11	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 12: Design of Tests	12	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 13: History of Work, Current Status, and Future Work	5	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 14: References	-	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Project Management	13	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Total	200	28.6	28.6	28.6	28.6	28.6	28.6	28.6	200
Total Percentage	-	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %

Summary of Changes

Section 4: Functional Requirements

- Added descriptions for every Actor and Stakeholder
- Created new Sequence Diagrams for each use case and modified some of the use cases

Section 7: Interaction diagrams:

- Updated to resemble our current implementation. Removed the “controller” and the “terminal” classes
- Added specific design patterns and explained how it improved our design

Section 8: Class Diagram and Interface Specification:

- Complete overhaul of the Class Diagram to resemble our current implementation.
- Updated the “Data Types and Operational Signatures” section to match the new class diagram
- Added “Design Patterns”
- Added OCL Contracts for Classes

Section 11: User Interface Design and Implementation

- Updated diagrams of design
- Added brief explanation of the implementation of the UI

Section 1: Customer Statement of Requirements (CSR)

a) Problem Statement

Establishing a new restaurant is like having a child: no one will take as much interest or pride in it as you. There are very few jobs that make you feel as proud and happy as owning a successful business. You are your own boss, but at the same time, you also have many responsibilities, and having a restaurant does not necessarily make you successful. There are many hardships and obstacles that I face, such as:

- Long days and nights - As an owner of a restaurant, I have to put in very long hours to make sure everything goes smoothly. And even when I'm home, I have to make several calls a day to make sure everything is fine. I'm constantly thinking of how I can improve my restaurant, and what I can do in order to reach more customers and improve their overall experience.
- No weekends - Weekends are the busiest time for most restaurants and are the time when the majority of customers have free time to stop by my restaurant. This means I have to sacrifice the time I originally had with my family and my free time in order to try to make my restaurant the best it can be.
- Unstable Income - I remember when business slowed down, and my checkbook was in the red, and guess who the first to go without a check was? Me. The person who does the most work, is the person who is often paid the least. Of course, sometimes small business owners have to make these sacrifices in order to keep the business afloat.
- Benefits - Health insurance is a major concern for all small business owners. I have to buy private insurance for myself and my employees and provide 401K plan and/or some sort other additional benefits as well.

As a restaurant owner, I want to give my customers the best possible dining experience I can, so that I can turn them into regular customers. Even if my food is great, if the customer doesn't like my service, it is highly unlikely that they will return to eat at my restaurant again. Oftentimes, customer service falls short due to the lack of policies set by myself, or the lack of quality in those I hire (such as waiters and chefs). Several problems that I have run into with my employees include:

- Tardiness - An employee who is late once or twice may need a reminder about company policies. An employee who is consistently tardy can throw off the shift schedule and strain his co-workers which will impact the quality of your restaurant. Controlling employees and changing their habits are hard; a

habitually late employee will most likely continue to be late even after being given warnings. Repercussions such as firing employees will then lead to other issues that require time and money, such as searching for new employees and then training them when they are found.

- Hygiene or appearance problems - One employee's foul hygiene or poor appearance will give rise to poor perceptions of the food quality, cleanliness of the restaurant and overall atmosphere. Issues similar to tardiness will arise.
- Poor attitude - Customers come to a restaurant for friendly, polite service and a pleasant experience. Service from someone with a poor attitude taints the entire dining experience and reduces patronage. It is hard to control attitudes of employees, as well as control specific moods they are in.
- Poor job execution - When an employee cannot perform his or her job at an acceptable level, additional training may be required, unless there is a deeper problem like poor attitude or laziness. Again, training requires additional time and money to be spent.
- Poor customer service - It is essential that all employees be well versed in customer service and table etiquette. Hosts and hostesses, servers and bussers should all exhibit quality service techniques.
- Insubordination - Problems with supervisors or managers are common plights in many businesses. A disrespectful staff member can leave a blemish in an otherwise positive workplace.
- Theft - Employees have been known to find methods of stealing money, supplies or food from a restaurant. This not only hurts business profits but also fosters an environment of dishonesty.

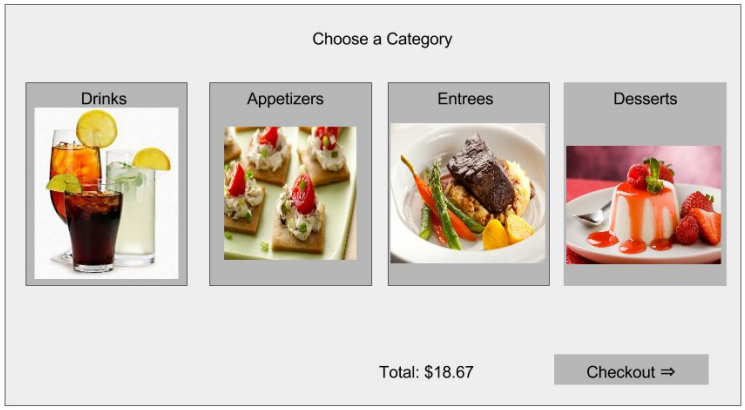
As a restaurant owner, I have more than enough on my plate already and I want to minimize the number of factors that are not in my control. This includes the efficiency at which my restaurant is running and reducing the number of employees I have to depend on.

When customers order a meal and don't receive what they ordered, they can either suffer through it or send it back. Either way, the situation isn't ideal because eating something they don't really want is unfavorable, and sending it back can lead to an awkward confrontation which will ruin their dining experience. If they were to have a confrontation, they're usually afraid of upsetting the server or receiving a meal that was rushed and not made properly, or they come across a jerky and a little entitled waiter, in which case you, as the restaurant owner have to deal with more than you bargained for. To avoid such problems, you as the restaurant owner should first figure out how to avoid such disturbances in the dining process. Like many existing restaurants such as

Chili's and Applebee's, I would like a customized web application that can be accessed via tablets at each table in my restaurant. The application should allow my customers to order food at their own pace and they no longer have to ask for the menus every time they wish to order something else. By enabling the customer to make changes to their order personally, I can reduce the number of mistakes in the orders, due to a waiter's negligence.

By introducing the web application, the restaurant will run more efficiently and hopefully prompt the customers to come back again. I also want to reduce the number of employees I hire because it adds onto my responsibility of running the restaurant. Most waiters and waitresses do not cause problems, but paying them and providing them with benefits cuts into my paycheck every month. A waiter's major role includes taking orders, delivering the food, giving the customer their check and then cleaning up. I would like a robot that is able to navigate through my restaurant without running into any obstacles and avoiding collisions with customers and/or staff. The robot should be able to differentiate between different foods and drinks. Then based on the selections made on the web application, the robot should bring the ordered food and drinks to the customers' table. This robot should reduce the number of employees required to run the restaurant efficiently while increasing profits.

I would also like administrator access to the robot. In the web application, I would like a staff member to have login access to the robot status. They should be able to use a simple login and see the robot's current status such as its location, battery level and the option to turn the robot on or off. Having access to such information is necessary for me to maintain the robots so I don't run into problems with them (It would not be very nice if a robot ran out of battery in the middle of my restaurant).



We conclude with the features we would like to have in our software/hardware as follows:

Web application requirements:

1. Order food - choose between entrees, appetizers, drinks and make modifications to the order
2. Request an employee (for some problem)
3. Pay their bill at the end of their meal.
4. An administrator login
5. Access to robot status (Location/battery and such)
6. Ability to turn the robot on and off

Robot Requirements:

1. Navigate through my restaurant without running into any obstacles
2. Differentiate between different types of foods and drinks
3. Be able to grab different types of food and drinks after recognizing them
4. Localization (It should know where the tables are)
5. Deliver food from kitchen to the customers' table
6. Differentiate between different customers (or where they are seated) in order to deliver the correct order to the correct customer

Section 2: Glossary of Terms

b) Glossary of Terms

Technical Terms

➔ Robot terms:
- Robot - a machine that is capable of carrying out a series of actions automatically

➔ Web application terms:
- User interface (UI) - refers to the “screen” of the application and the components within it that users (i.e. customers) see and interact with.
  i) Customer-side UI - The user interface with which the customers interact.
  ii) Administrator-side UI - The user interface with which the manager interacts.
- Web Application (webapp) - a program accessible through the internet that performs some task. In the case of this project, “webapp” refers to the application used by customers to request food or drinks, and used by managers to synchronize with the restaurant’s menu and attend to customers who seek them.
- Menu - a graphical display on the webapp containing different selectable options. Not to be confused with the non-technical term, Menu.
- Flask- A lightweight web application framework used on the application backend. Implemented in Python, it incorporates easy-to-use functions such as routing, template rendering, and JSON for transforming database records that we utilize in our application.

➔ Server/database terms:
- Database - a collection of information that is organized for ease of access, management, and updating
- Server - a computer program or machine that waits for and responds to a request sent by another computer
- Request - in the perspective of computers, this is defined as a message sent from one device to another
- PyMongo- A Python library used to communicate with a MongoDB database; our web application utilizes this to add orders to the database and process them through Python.

Non-Technical Terms

- ➔ Manager - the individual who controls the operations of the restaurant
- ➔ Customer - an individual who is served by the restaurant
- ➔ Order - a request of food and/or drink specified by a customer
- ➔ Menu - a list of food and drink items available to select. Not to be confused with the technical term, Menu.
- ➔ Administrative Options - the options provided to the manager to update the restaurants menus. This includes options to add/ remove items, change item descriptions, and change item prices.
- ➔ Robot Conditions - This includes the robot's battery life and the mechanical conditions or the robot.
- ➔ Stock - a generalization of any location in which food or drinks can be served. This document assumes that the food or drink is premade and ready for the robot to retrieve.

Section 3: User Stories (In Place of System Requirements)

Size Key

For each user story we include a size of 1 - 10 points. The sizes are rough estimates of how much effort is needed to implement each user story based on our group's technical skills and each user story's perceived complexity.

Priority Key

ST-X-#: High priority (essential feature)
ST-X-#: Middle priority (Important feature)
ST-X-#: Low priority (less important feature)
ST-X-#: Lowest priority (optional feature)

Structure

- The user stories have a very specific structure:
- As a <insert actor here>, I am able to <insert action here> by means of <insert device here>, which will allow me to <insert benefit here>
- ST-Cu-# for customer stories, ST-M-# for manager stories, and ST-Ch-# for chef stories

a) As the customer...

Index Number	User Story (As the customer)	Size (Points)
ST-Cu-1	I am able to request food and drinks by way of the web application, which ROSE will then bring to me when it is ready.	7 points
ST-Cu-2	I am able to traverse the restaurant without ROSE getting in my way.	4 points
ST-Cu-3	I am able to check the subtotal for what I ordered, so that I know what I am paying for.	2 points
ST-Cu-4	I am able to request a change in my order before ROSE sends the order request.	2 points

b) As the manager…

Index Number	User Story (As the manager)	Size (Points)
ST-M1	I am able to access the customer menu from the restaurant's PC in order to determine what my customer sees when viewing the menu.	2 points
ST-M2	I am able to access administrative options of the restaurant's menu on the restaurant's PC with a username and password so that I will be the only person that can change the restaurant's menus.	3 points
ST-M3	I can add, remove, and alter menu items through the administrative options on the restaurant's PC for the sake of updating customers about the available items and their prices.	5 points
ST-M4	I am able to view the quantity of purchase for each item and the restaurant's profits with administrative access so that I can monitor the state of the restaurant and make changes to the menu based on item popularity.	5 points
ST-M5	I can monitor the current status of the robot, which includes the position, table assignment, and its current conditions, through the restaurant's PC so that I can determine if the robot has malfunctioned and, in the event that it has, find out where the robot is so that I can find and troubleshoot it.	8 points

c) The User Experience

Our project will in essence have 2 types of users, the manager/restaurant owner, and the restaurant customer.

The manager should have administrative access to the user interface. They should be able to do things such as add/ remove items, alter price of items, display when items are temporary available/ unavailable (i.e. item of the day, ran out of materials for that item, etc.). In addition, the manager should be able to monitor and control various properties of the robot such as table assignment of the robot, current action of the robot, condition of robot (i.e. battery life, any mechanical malfunctions).

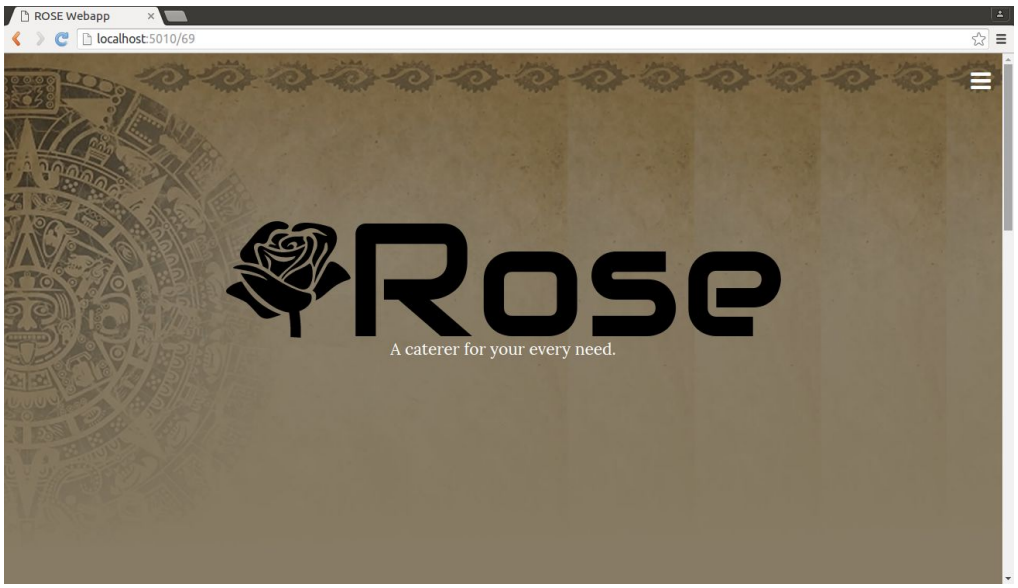
The customer should be able to view the restaurant menu items. This includes the Item description, the price of the item, and possibly a picture of the item. in addition, the customer should be able to place an order and purchase for their meal through the device. This includes purchasing for multiple people from one device (allow customers to select their meals from separate devices, but accept the purchase of all items from one device). Furthermore, the customer should be allowed to monitor the state of their meal and be given an estimated time of service.

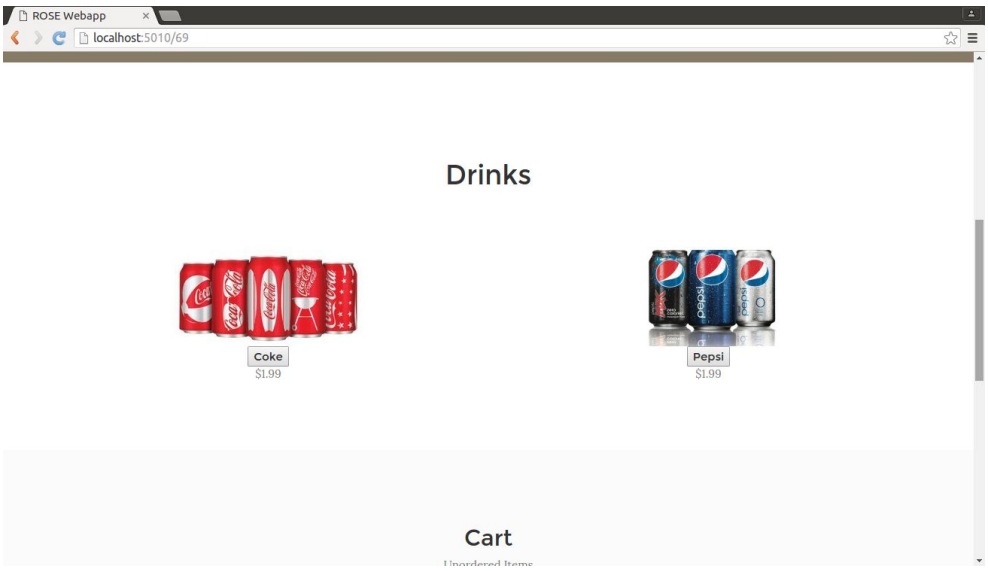
The process of interacting with the robot and its features will depend on the actor that wants to communicate with the robot. Their will be two types of actors. Actor 1 will be the restaurant customer, which will order items from a device provided to them by the restaurant. Actor 2 will be the restaurant manager, which will have access to the administrative features of the web application.

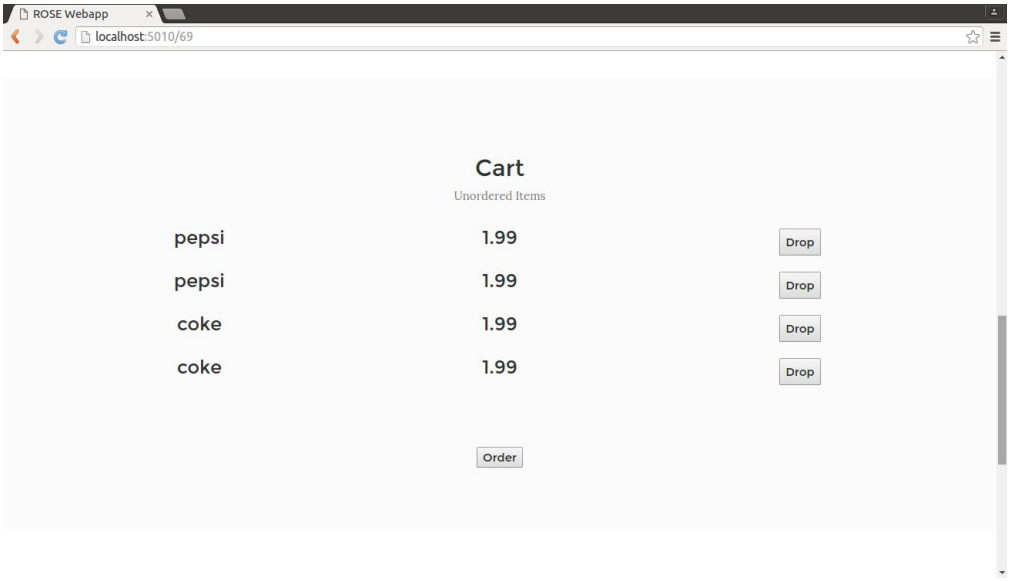
The manager of the restaurant will be able to access the web application through a personal computer within the restaurant. Once accessed, the manager will have a variety of features to access for the sake of monitoring the state of the restaurant and the state of the robots in service. As far as monitoring and updating the restaurant is concerned, the manager will have administrative access to the menu of the restaurant. Administrative access will be checked by a userID and password, so that the system not only knows that the user is an administrator, but also knows which administrator is using the web application. In addition, the userID will be used so that any administrator knows

whoever made changes to the web application and when the made the changes. Once logged on, the administrator will be able to monitor data pertaining to their restaurant and will be able to make changes to the menu items that the restaurant customers will see. From the restaurant menu, in addition to the item data that is seen by the restaurant customer, the administrator will be able to view data about menu items such as the quantity that has been sold of that food item and the profit generated by selling that item throughout the day.

Customer User Interface:







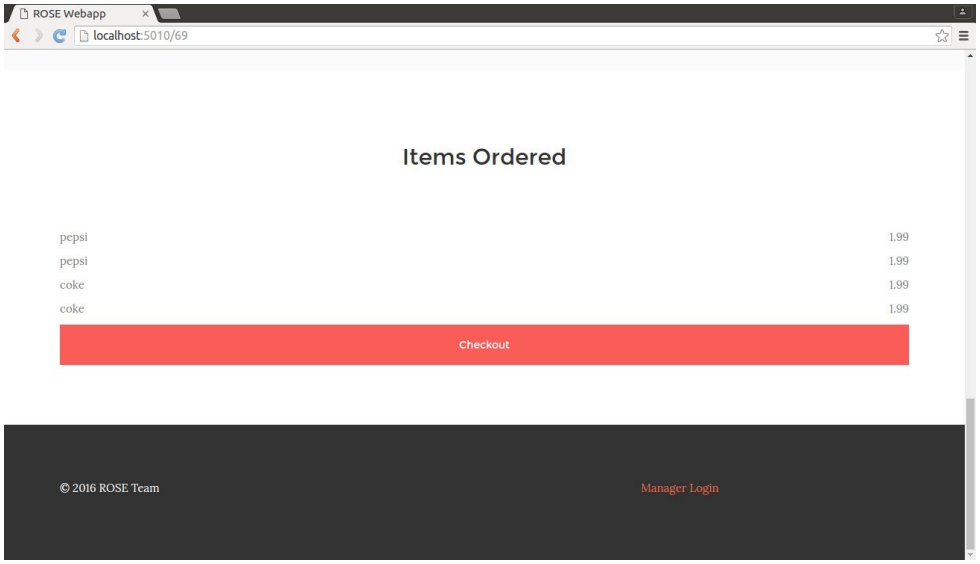


Figure 3.1: These screenshots are the finalized web application pages for the customer user interface. It allows customers to order a drink with the press of a button, review the items in a card, and confirm the order via the “checkout” button.

Manager User Interface:

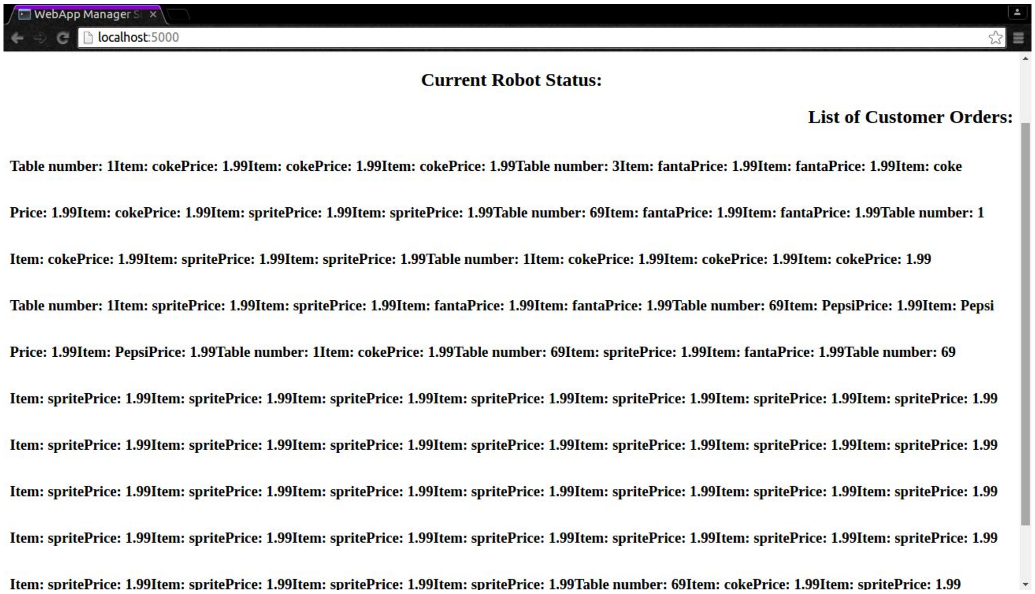


Figure 3.2: This screenshot shows the updated manager interface. This part of the web application allows the manager to view the list of orders, prices, and table number.

Section 4: Functional Requirements Specification

a) Stakeholders:
   - Manager - ROSE minimizes the need to hire human waiters and thus, will decrease the amount of money the manager needs to spend, as they would otherwise have to pay their waiters annually.
   - Restaurant organization - A similar benefit as the manager, the Restaurant Organization will not have to spend money annually on hiring waiters.
   - Chef - Eliminates the need for human interaction with waiters, which will negate any chances of miscommunication in terms of order delivery
   - Customers - ROSE minimizes the chance that the customer will receive an incorrect order or that their order will be forgotten. In addition, customers will not be required to tip ROSE, which allows them to save money
   - Manufacturing companies - Manufacturing companies will have an increased income due to the demand from restaurant organizations to manufacture ROSE robots.
   - Developers - A similar benefit as the manufacturing companies, developers will be in demand to create, improve, and specialize ROSE's design for restaurant organizations.

b) Actors and Goals:
   Initiating Actors:
   - Robot - ROSE will initiate the delivery of the food item to the customer as soon as ROSE receives the order of the desired item from the customer's web app. ROSE will have completed its goal when It serves the desired item to the table which ordered that item.
   - Customer - The customer will initiate ordering the desired item. ROSE will not be able to initiate delivering an item until the customer has completed their order. The goal of the customer is realized when ROSE has confirmed that the customer's order has been placed.
   - Manager - The manager will initiate the monitoring of ROSE as it makes its delivery to the customer. The manager's goal will be realized when the Information about ROSE's location and battery life has been displayed to the main terminal of the restaurant.

Supporting Actors

- Tablet - The tablets will generate an interface for the restaurant customer to use. This interface will allow the customer to view the available items of the restaurant and order any of the available items via web application.
- Main terminal - The main terminal will communicate with ROSE and request information about ROSE's status such as battery life and location. It will then proceed to update an Interface for the manager which will display the status of ROSE in real time.

Off-Stage Actors

- Stock - The stock is any location in which ROSE will travel to in order to retrieve the item that has been requested by the customer. This location will provide the Item that ROSE is searching for.

Casual Descriptions of Use Case:

Our group's user stories will serve as the casual descriptions of our use cases

Use Case Diagram

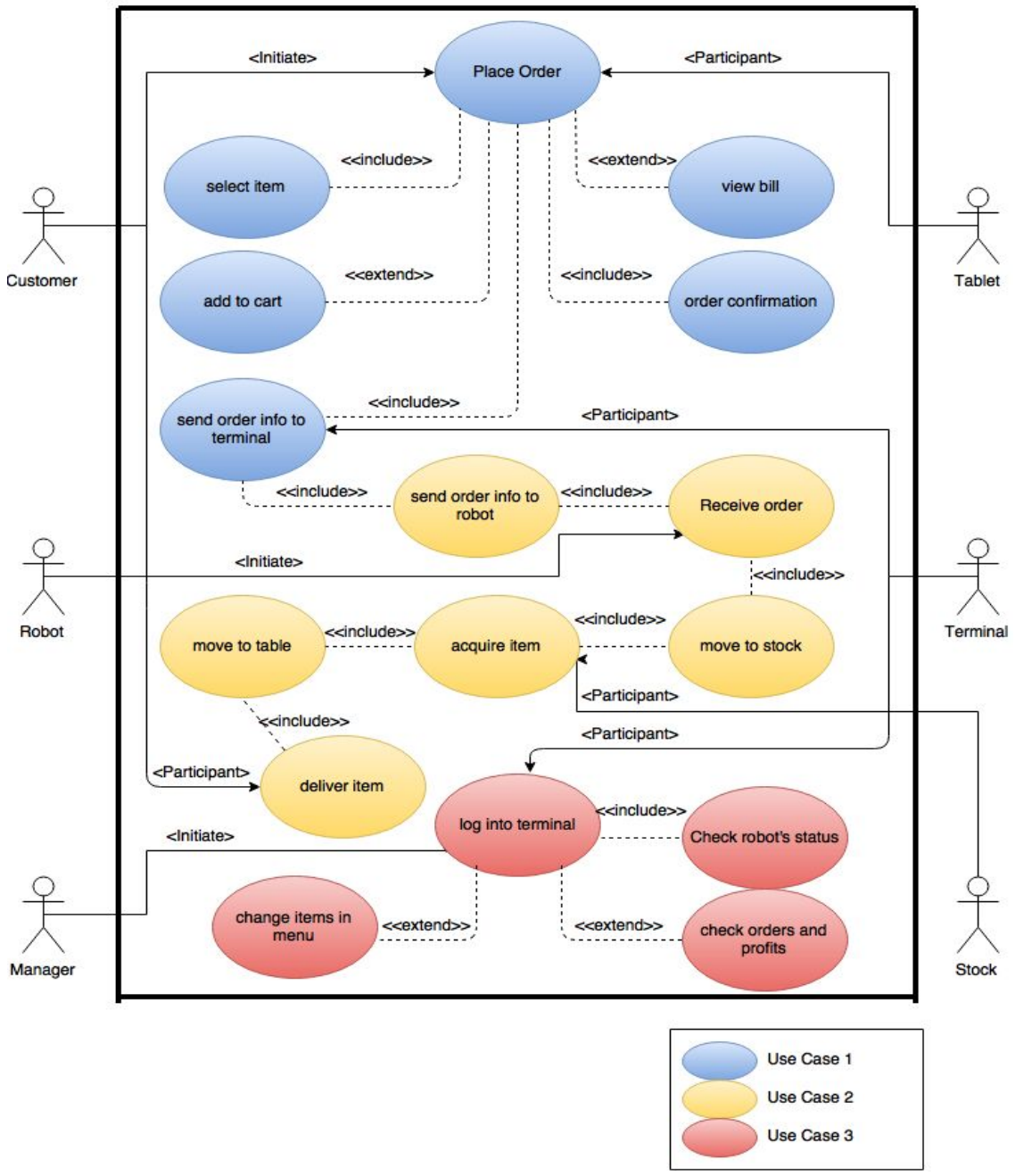


Figure 4.1: Use case diagram, illustrating the interactions between the actors and the components of the system. Note that while finalizing our project, we decided to replace the terminal with a web application called the Manager Interface.

Use Cases

1. Placing the Order

Initiating Actor: Customer
Goal: Receive food using a Web App on tablet
Participating Actors: Web Application, Database

Precondition: The customer has a tablet present on the table to place their order
Postcondition: The customer's order is placed into the database queue

Main Success Scenario:

- Customer opens restaurant menu on web app
- Customer views the items present on the menu through categories
- Customer will add items to cart
- Customers continue the process until they have all the desired items in the cart
- Customers press the order button to send the order to the database
- Customers will receive a confirmation on screen that the order has been placed
- Customers will press checkout to get the bill

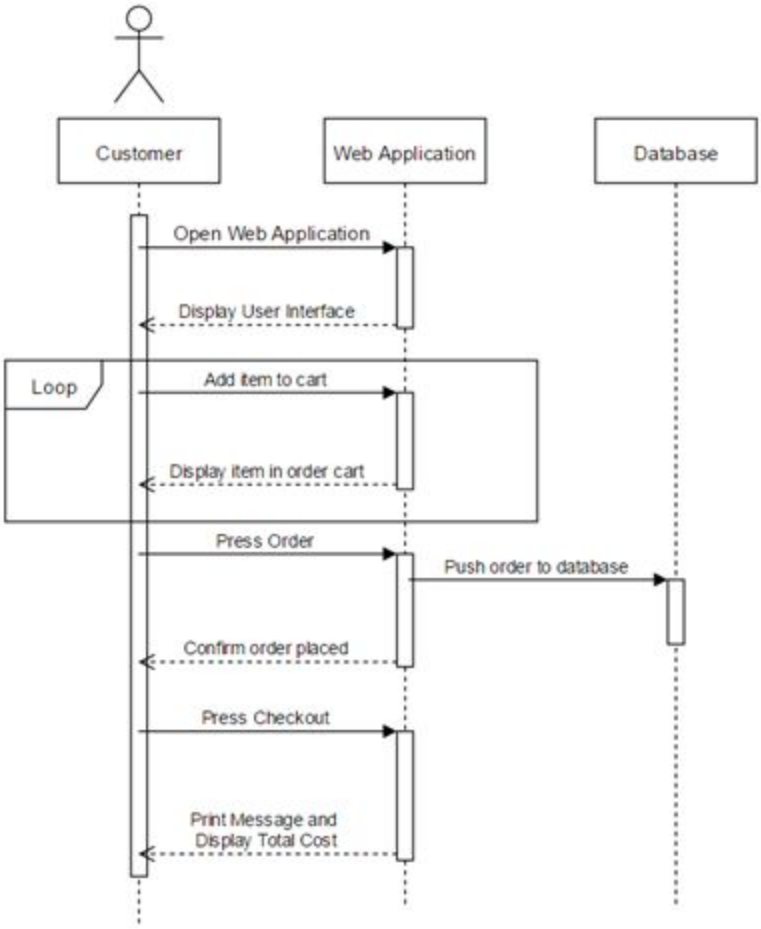


Figure 4.2: Sequence diagram for placing the order. Note that we decided to have the web application interact with the database instead of through a main terminal.

2. Delivering the Order

Initiating Actor: Robot
Actor's Goal: deliver food to the customer
Participating Actors: Stock, Customer, Database

Precondition: The requested item is present in the database of the main terminal
Postcondition: The customer has received his/her order

Main Success Scenario:

- Robot request order from the database
- Order information is delivered to the robot
- Robot moves to stock
- Robot acquires the order from stock
- Robot updates its status every certain time intervals
- Robot transports the order to the customer's table
- The customer receives the order from the robot

Alternate Scenario (Out of Stock):

- Robot does not find the requested item
- Robot tells the main terminal that the food item is out of stock
- The main terminal sends information to the web app that the food item is out of stock

Alternate Scenario (Robot Malfunction or Connection Issues):

- Robot fails to send update status to main terminal or stops performing a certain action
- Terminal sends a distress signal to the manager
- Manager receives signal from the terminal
- Manager handles the malfunction as needed
- Terminal alerts the user of this malfunction and asks them to wait while the manager resolves the issue

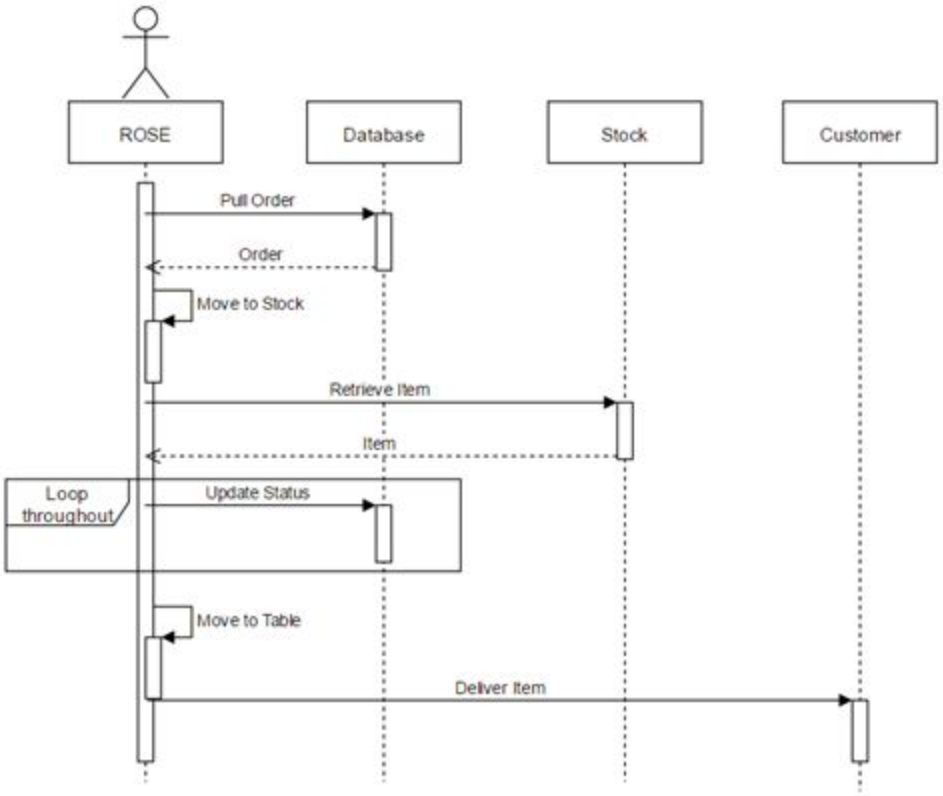


Figure 4.3: Sequence diagram for the main success scenario for delivering the order. Here, the robot interacts with the database, stock, and customer so that delivery is ensured. Instead our previous reports, we had a main terminal; this has been replaced by the database.

3. Review the System

Initiating Actor: Manager
Actor's Goal: View status and condition of the robot
Participating Actors: Web Application, Database

Precondition: Robot updates the main terminal of its current condition and status
Postcondition: Manager is notified on all aspects of the robot in real time

Main Success Scenario:

- Manager opens the manager interface through the web application
- The manager checks the robot's status through the web application which is retrieved from the database
- The manager checks the orders on the web application which is retrieved from the database

Alternate Scenario:

- Robot sends signal to the main terminal
- The main terminal stops sending orders and signals to the robot
- The main terminal decides which type of malfunction occurred
- The main terminal pings the manager to let him know that a malfunction has occurred
- The manager can check and see what type of malfunction has occurred

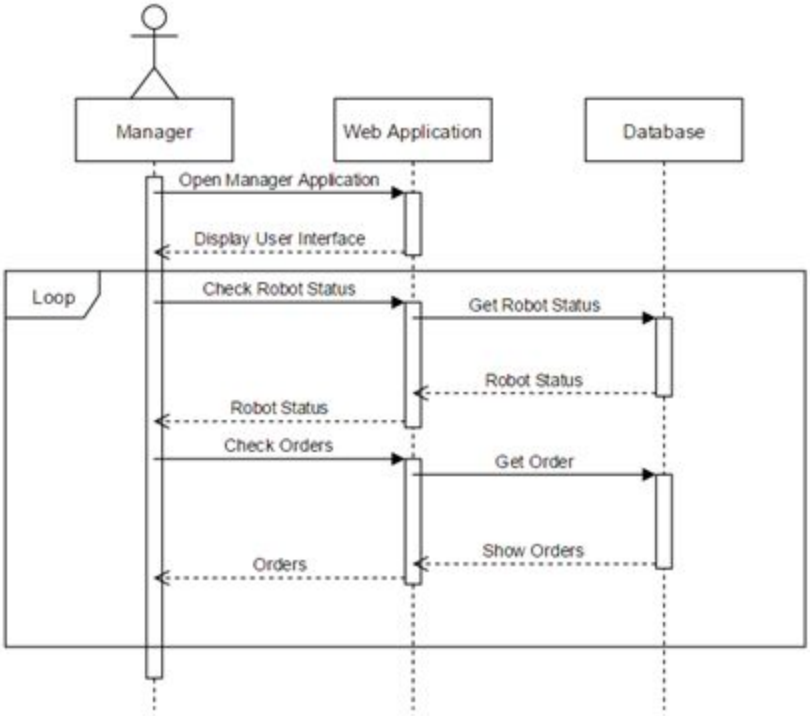


Figure 4.4: Sequence Diagram for main success scenario of Reviewing the system. Note that in previous reports, we had the manager interacting with a main terminal. Our finalized design has a manager web application to check the status and updates of robot status through the database. Login, confirmation, and logout were not implemented due to time constraints.

Traceability Matrix

Requirement	PW	UC-1	UC-2	UC-3
1	3			
2	2			
3	3			
4	3			
5	4			
6	5			
7	5			
8	3			
9	5			

Figure 4.5: The traceability matrix, linking the requirements with the use cases for this project.

Requirements:

1. Provide user interface for the customer
2. Allow the user to select items from the user interface
3. Notify the terminal of the items selected when the order is placed
4. Notify the robot of the items selected when the order is placed
5. Robot selects items to deliver from the stock based on the items ordered by the customer
6. Monitor the robot's location, actions, and conditions
7. Deliver the requested items by robot
8. notify the terminal that the robot has completed delivery and is ready to make a new delivery
9. Notify the terminal when the robot malfunctions

Section 5: Effort Estimation

Placing an order:

1) Web Application - 35 hours

For the placement of the order, all of the time was gone into creating the customer side of the web application. This was broken down into two parts: one was the customer interface, and the second was the functionality of the interface. In order to perform these tasks, we had to learn HTML and CSS for formatting and creating the aesthetics of the customer interface and python/javascript for the functionality. Testing included: visually adding those same items to the cart for the customer to view on the web application. The final version was displayed in our second demo.

2) Database - 15 hours

We had to learn about database communication with the web application using Python. Tests included: pushing ROSE's control input into a database and seeing if the correct values were pushed and pushing orders to the database for ROSE to receive.

Delivering an order:

1) Database - 20 hrs

The majority of the time spent on the database was about learning how the database functions. We had to learn how to: push information to the database, create a server/ client connection through MongoDB, pull and convert BSON document values to types in C++, pulling and parsing item strings. Testing has included: pulling control input for ROSE from the database, pushing ROSE values to the database, and pulling orders from the database. Finally, a simulation was created of database manipulation for demoing.

2) Computer Vision - 100 hours

Attempted a variety of different computer vision techniques, such as Support Vector Machines, Hard Cascades and Color Mapping. All these techniques failed so we had to resort to implementing edge detection using Chilitags and localization.

3) Robotics - 400+ hours

Preparation of the robot included: designing and building the robot's frame, Testing physical connections (power supplies, motors, potentiometers, encoders, arduinos), Implementation of Proportional Integral Derivative (PID) for both the base and the arm of the robot, Implementation of Inverse kinematics for the arm of the robot,

Integration of the robot with computer vision, and implementation of autonomy.

Updating ROSE status:

Section 6: Domain Analysis

a) Domain Model

In the following model derivations, the following definitions are used:

Responsibility - a necessary action that the system must fulfill

Type - a classification of aforementioned responsibilities

Type D, "Doing": a responsibility that requires action, such as manipulation of data or physical interaction of components

Type K, "Knowing": a responsibility that designates the storage of data

Concept Name - label that represents aforementioned responsibilities

(i) Concept Definitions

Use Case 1: Placing the Order

Responsibility	**Type**	**Concept Name**
Coordinate and delegate responsibilities of all concepts.	D	Controller
Accept user's touch input.	D	TouchInterface
Container for user's order data, consisting of list of foods and prices.	K	Order
Container for all of a given user's orders.	K	Bill
Container for all food items in the restaurant.	K	FoodList
Push order to the user's bill.	D	OrderAdder
Container for all customers' orders.	K	OrderQueue
Add item to user's order.	D	ItemAdder
Confirm with user that he/she would like to take an action.	D	WarningPopup
Push order to the robot's order queue.	D	OrderPusher
Receive payment for bill.	D	PaymentAcceptor
Display menu items for the user.	D	MenuDisplay
Show confirmation that the order has been pushed.	D	ConfirmPopup

Figure 6.1: Extracted concepts and their responsibilities for Use Case 1.

Use Case 2: Delivering the Order

Responsibility	Type	Concept Name
Coordinate and delegate responsibilities of all concepts.	D	Controller
Implement a procedure to pull and remove database records as robot processes orders.	D	DatabaseConnection
Block additional requests for delivering orders if the robot is already processing an order.	D	StatusChecker
Split a given order into the queue of actions that constitute it.	D	OrderParser
Correctly identify the item to be delivered from the available stock.	D	ItemChecker
Continuously check a timer to determine when to display a robot's status.	D	StatusChecker
Deliver the order to the correct table.	D	OrderDeliverer
Allow web app to receive messages regarding out-of-stock items.	D	StockChecker

Figure 6.2: Extracted concepts and their responsibilities for Use Case 2.

Use Case 3: Review the System

Responsibility	Type	Concept Name
Authenticate user credentials.	D	Authenticator
Container for user's information.	K	User
User information storage module.	K	UserList
Display the map of the restaurant, and robot's location, the current action, and current operating status of the robot.	D	DataDisplayer
List of queued actions.	K	ActionQueue
Receive information (location, operating status, current action) from the robot.	D	DataReceiver

Figure 6.3: Extracted concepts and their responsibilities for Use Case 3.

(ii) Association Definitions

Identifying associations for Use Case 1: Placing the Order

Concept Pair	Association Description	Association Name
Controller ⇔ TouchInterface	The Controller receives input from the TouchInterface, which is generated by user interaction, to ensure the user can interact with the device.	Touch Request
Order ⇔ OrderAdder	OrderAdder will store data to the Order container so that the user's bill is updated	Provide Data
OrderPusher ⇔ OrderQueue	The OrderPusher data is stored in OrderQueue to ensure that the order information is saved for the user	Order Request
MenuDisplay ⇔ Controller	The Controller should initiate MenuDisplay so that the user will be able to see what is available to order	Show Menu
PaymentAcceptor ⇔ Bill	Payments should be paid according to the amount calculated in the Bill, so that the PaymentAcceptor debits the correct amount from the user.	Pay bill
WarningPopup ⇔ TouchInterface	The user must be able to confirm his or her action by touching the choices that WarningPopup has, through the TouchInterface.	Confirmation
ConfirmPopup ⇔ TouchInterface	TouchInterface will display a message to the user indicating that the confirmation has been received.	Confirmation Received
FoodList ⇔ MenuDisplay	MenuDisplay should extract container of information regarding items from the	Prepare data

	FoodList to show users the restaurant's menu.	

Figure 6.4: Relationships between concepts seen in Use Case 1 and their importance.

Identifying associations for Use Case 2: Delivering the Order

Concept Pair	Association Description	Association Name
Controller ⇔ DatabaseConection	Controller ensures that there is a connection between the application and the server by checking DataBaseConnection. The server connection is needed to store data from users and associated systems.	Check Connection
Controller ⇔ StatusChecker	Controller would see if robot is delivering an order so that the robot can finish its current task. The Controller will also establish an interval by which to execute this concept.	Check status
Controller ⇔ OrderParser	The Controller will break down the order so that the robot logically follows specific steps to obtain the requested items in that order.	Generate Actions
Controller ⇔ ItemChecker	The Controller allows the robot to distinguish features a requested item to ensure that the item is correctly identified.	Identify item
Controller ⇔ OrderDeliverer	OrderDeliverer receives coordination data from the Controller so that the robot delivers the order to its correct destination.	Delivery
Controller ⇔ StockChecker	Controller returns processed data to give StockChecker an updated count of	Check stock

	what items are available in the restaurant.	

Figure 6.5: This table describes the relationships between each concept in Use Case 2 and explains why they need to work together.

Identifying associations for Use Case 3: Review the System

Concept Pair	Association Description	Association Name
Authenticator ⇔ User	Authenticator takes username and password entered through User and temporarily holds the login request for verification.	Begin Authentication
Authenticator ⇔ UserList	Authenticator attempts to match credentials stored in UserList to match the credentials of a user that is attempting to log in.	Verify User
DataDisplayer ⇔ ActionQueue	The ActionQueue has a list of queued actions that DataDisplayer checks in order to give an update on the display	Show actions
DataReceiver ⇔ DataDisplayer	The DataReceiver obtains data about the robot that DataDisplayer continually checks in order to provide live updates.	Send Info

Figure 6.6: This table lists relationships based on Use Case 3 and explains why they need to work together.

(iii) Attribute Definitions

Attributes for Use Case 1: Placing the Order

Concept	Attributes	Attribute Description
Controller	Coordination	Used to interact with various components simultaneously to ensure smooth transitions and multi-component effort to execute tasks.
	Task Distribution	Ensures that information and commands are sent to the correct component that can execute such requests.
TouchInterface	Deciphering user input	Ability to convert touch of screen by user into readable data by the system. Necessary to simulate interaction between user and display.
PaymentAcceptor	Transaction Record	Tracks transactions for restaurant's records to document method of payment, approval of credit or debit card, and time of transaction.
MenuDisplay	Data-to-image conversion	Any data, information, or confirmation that the device is programmed to output must be readable to the user via the display.

Figure 6.7: Attribute definitions based on Use Case 1.

Attributes for Use Case 2: Delivering the Order

Concept	Attributes	Attribute Description
Controller	Coordination	Exchanges data with robot for positioning and movement tracking in order to deliver the correct order in a reasonable time.
	Task Distribution	Breaks down data from orders so that robot can follow a sequential set of commands to retrieve items and orders to be delivered.
DatabaseConnection	Pull request	Obtains and, if needed, removes data from the database that the webapp stores from orders and other information about the user.
ItemChecker	Object identifier	Program to visually identify items by key features.
StatusChecker	Order interrupter	Temporarily holds a series of commands that delivers an order if another order is being delivered.
StockChecker	User notifier	Allows message to be displayed after stock for a specific item is unavailable.
OrderDeliverer	Delivery confirmation	Once an order is delivered to its correct table, the next order in queue is looked at.
OrderParser	Data division	Information lumped into an order is split into logical steps that the robot can follow.

Figure 6.8: Attribute definitions based on Use Case 2.

Attributes for Use Case 3: Review the System

Concept	Attributes	Attribute Description
Authenticator	Verify identity	Checks to see if user input matches pre-existing credentials that are stored in a database.
DataReceiver	Find location	Locates robot and records position within restaurant.
	Check status	Pings the robot to see if it is operating correctly
	Track actions	Retrieves information about what the robot is doing
UserList	Check role	While credentials are being verified, if a match is made, the role of the user logging in is checked for access privileges.
User	Username container	Field provided to user to enter user name
	Password container	Field provided to user to enter password
DataDisplayer	Periodic checking	Makes sure that data being displayed is up-to-date by performing status checks in intervals.

Figure 6.9: Attribute definitions based on Use Case 3.

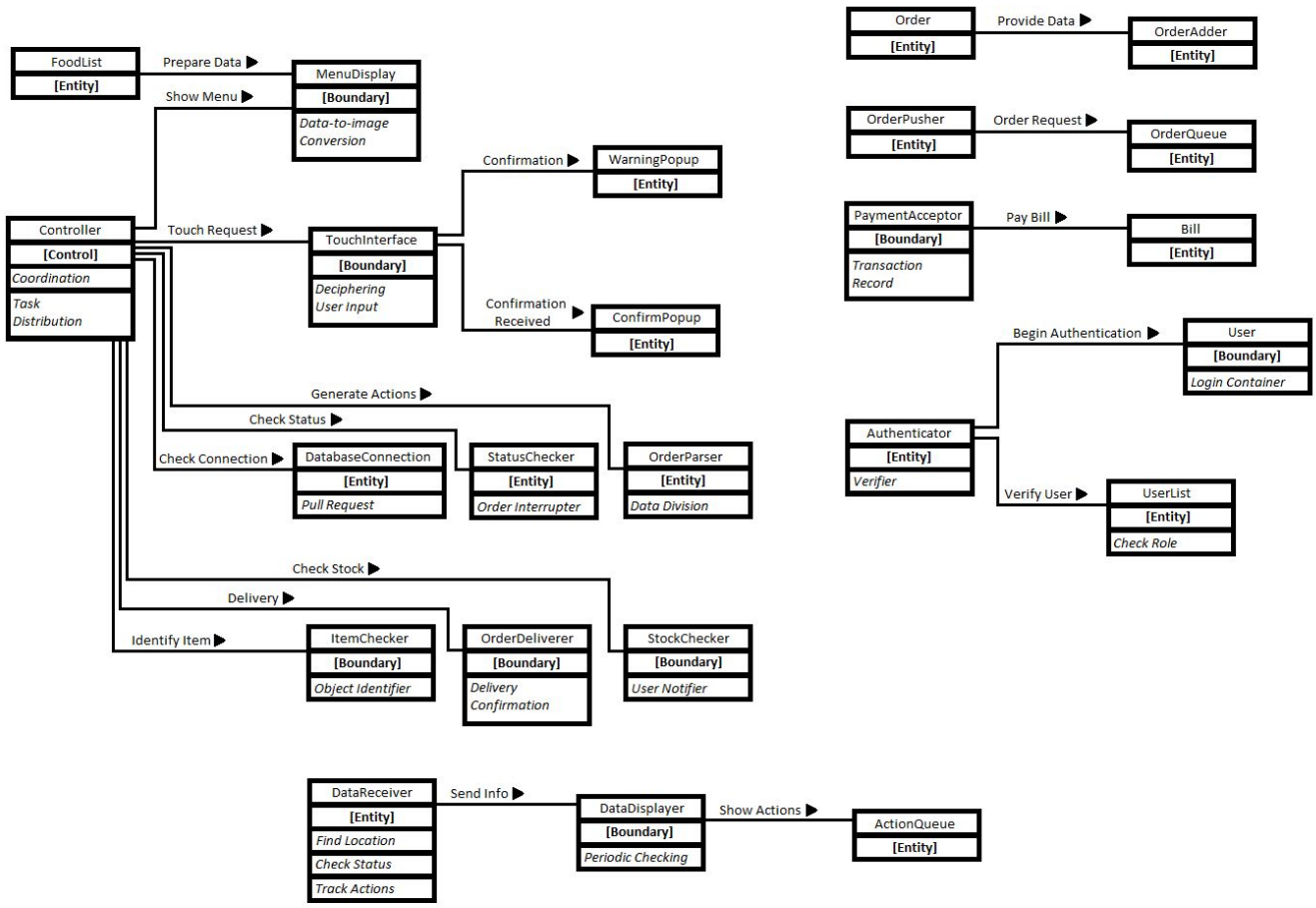


Figure 6.10: The Domain Model

The domain model was constructed by first extracting the responsibilities that needed to be completed from each use case, and by delegating these tasks to appropriate concepts. After this, the attributes of each concept were extracted so that these concepts can contain the information needed to perform their operations. Moreover, associations/relationships between different concepts were determined; these attributes and associations were then incorporated into the concept illustration, and the domain model was completed. For convenience, each concept was labelled as one of the following: control (which is responsible for delegating tasks to specific concepts), boundary (a concept that directly interacts with the external actors), or entity (a concept that is internal to the system and that therefore does not interact with the actors).

(iv) **Traceability matrix**

Use Case	PW	Domain Concepts																							
		Controller	TouchInterface	Order	FoodList	Bill	OrderAdder	OrderQueue	ItemAdder	WarningPopup	OrderPusher	PaymentAccepter	MenuDisplay	ConfirmPopup	DatabaseConnection	StatusChecker	OrderParser	ItemChecker	OrderDeliverer	StockChecker	Authenticator	User	UserList	DataDisplayer	ActionQueue	DataReceiver
UC-1	8	X	X	X	X	X	X	X	X	X	X	X	X	X												
UC-2	28	X													X	X	X	X	X	X						
UC-3	19																				X	X	X	X	X	X

Figure 6.11: Shows how our use cases map to our domain concepts.

b) Systems Operation Contracts

Operations Contract Table 1

Name	Place Order
Responsibilities	Allow customer to give order to robot through Web App on a mobile device
Use Cases	UC-1
Exception	No exceptions
Preconditions	Display a menu through a Web App on a tablet or mobile device.
Postconditions	Order is placed in the database to be carried out by the robot.

Operations Contract Table 2

Name	Deliver Order
Responsibilities	Based on items requested by customer, the robot will obtain the item from the stock and deliver it to the customer.
Use Cases	UC-2
Exception	Item is out of stock
Preconditions	Order is placed in the database by the customer.
Postconditions	Order has been received by the customer.

Operations Contract Table 3

Name	Request information from terminal
Responsibilities	Allow manager to access information about the robot through the restaurant's server.
Use Cases	UC-3
Exception	No exception
Preconditions	Robot sends information to the terminal about its' condition.
Postconditions	Manager is informed about the state of the robot and the robot's current action.

c) Mathematical Models

Particle Filter / Localization:
We use the particle filter algorithm for localization of our robot. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, a hypothesis of where the robot is. The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning the robot has no information about where it is and assumes it is equally likely to be at any point in space. Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are resampled based on recursive Bayesian estimation (how well the actual sensed data correlate with the predicted state). Ultimately, the particles should converge towards the actual position of the robot. The use of fiducial markers to act as landmarks (chilitags in our case) helps with the implementation of this algorithm on our physical robot.

Edge Detection / Chilitags:
We will use edge detection to aid in the localization, object detection and identification. Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction. The algorithm used to detect chilitags utilizes edge detection to determine if a set of edges is a chilitag and also determine its id, in the case it is one.

PID / Motion:
We will be using PID(proportional–integral–derivative) controller to make the robot's movements more accurate. A PID controller continuously calculates an error value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error over time by adjustment of a control variable, such as the position of a control valve, a damper, or the power supplied to a heating element, to a new value determined by a weighted sum. With the PID controller the robot will take into consideration the external forces and still function accurately (for example moving in a straight line or picking up an object).

Inverse Kinematics / Motion of arm:
We will use kinematics equations of the robot to determine the joint parameters that provide a desired position of the end-effector. Inverse kinematics transforms the motion plan into joint actuator trajectories for the robot. The movement of a kinematic chain of a

robot is modeled by the kinematics equations of the chain. These equations define the configuration of the chain in terms of its joint parameters. Forward kinematics uses the joint parameters to compute the configuration of the chain, and inverse kinematics reverses this calculation to determine the joint parameters that achieves a desired configuration. Inverse kinematics formulas allow calculation of the joint parameters that position a robot arm to pick up a part. Rather than using an established numerical/graphical approximation, we use an iterative model that determines a particular joint angle by varying from 0 to 180 degrees in 5-degree increments.

Section 7: Interaction Diagrams

Class descriptions

ROSE - Describes how the robot will send and receive data and how the robot will move
Menu - stores every item onto a list to be displayed to the customer
CustomerInterface - Interface for the customer to be able to view and choose items that are available from the restaurant
Database - Stores information of the customers orders and the robot's status. This information will be requested by the terminal, the robot, and the customer device as needed
ManagerInterface - Interface for the manager to be able to monitor the robot's movement and conditions
Order - stores the item ordered by the customer, the order ID and the table ID from which the customer placed the order. This class is stored in the database class as the customer places the order.

Use Case 1: Placing an Order

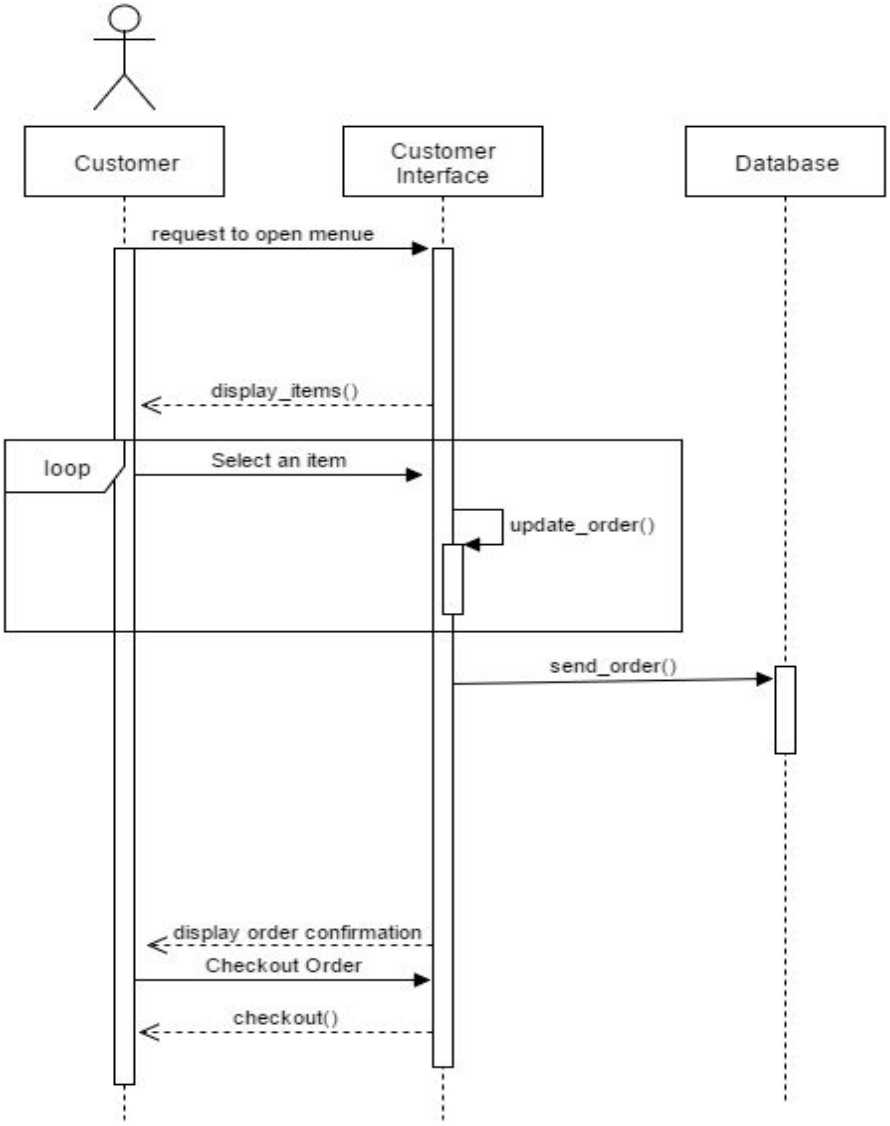


Figure 7.1: This use case describes how the customer will be able to send information to the database to be received by the robot. This use case involves a controller that will pass information between the events on the customer interface and the database itself. the database will pass information in a first in, first out fashion.

Design Pattern: The Mediator Pattern is depicted here, where the customer interface acts as an exchange medium between the customer and the database. This improves our design by having some entity act as an interpreter between the front-end and back-end when a customer places an order, making the ordering experience seem seamless.

Use Case 2: Delivering the Order

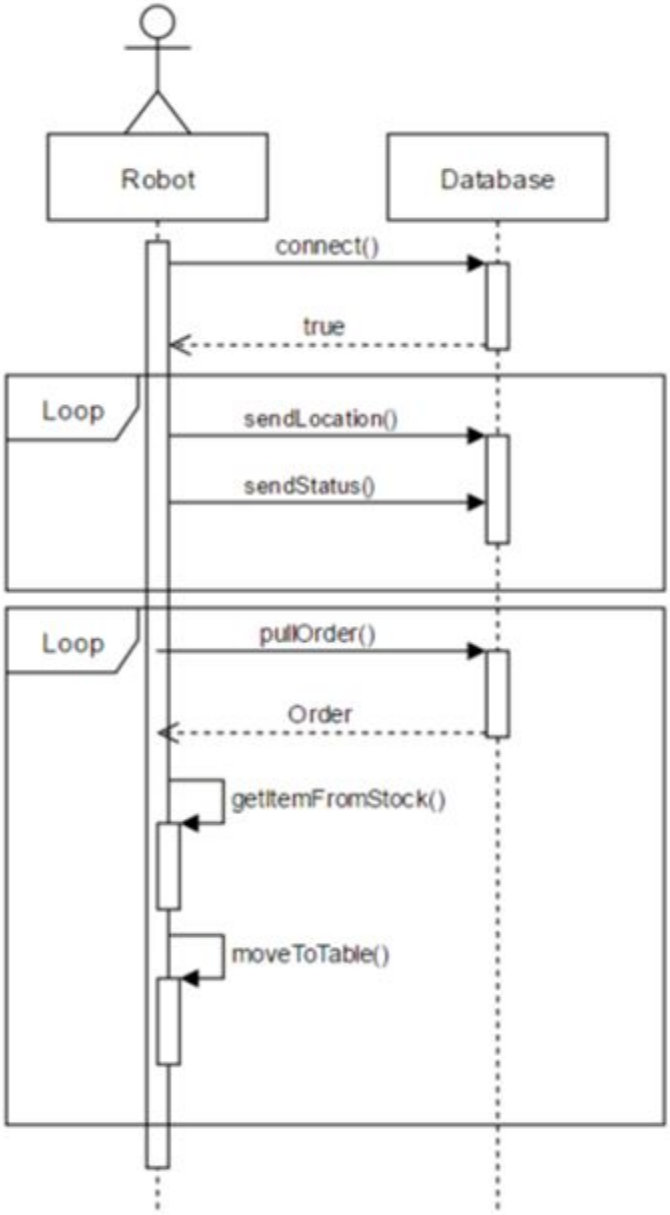


Figure 7.2: ROSE begins to connect to the database and sends its status as well as location.This loops while the robot is active. Inside the second loop, we have to robot getting the order (order is the class containing item ordered, tableID and orderID) from the database, picking up the item(s) ordered and delivering them to the customer.

Design Pattern: An Interpreter Pattern is visible here, where the robot retrieves data from the database so that it can fulfill the order request that the customer made through the customer interface as seen in **Figure 7.1**. It contributes to our automation design.

Use Case 3: Reviewing the Order

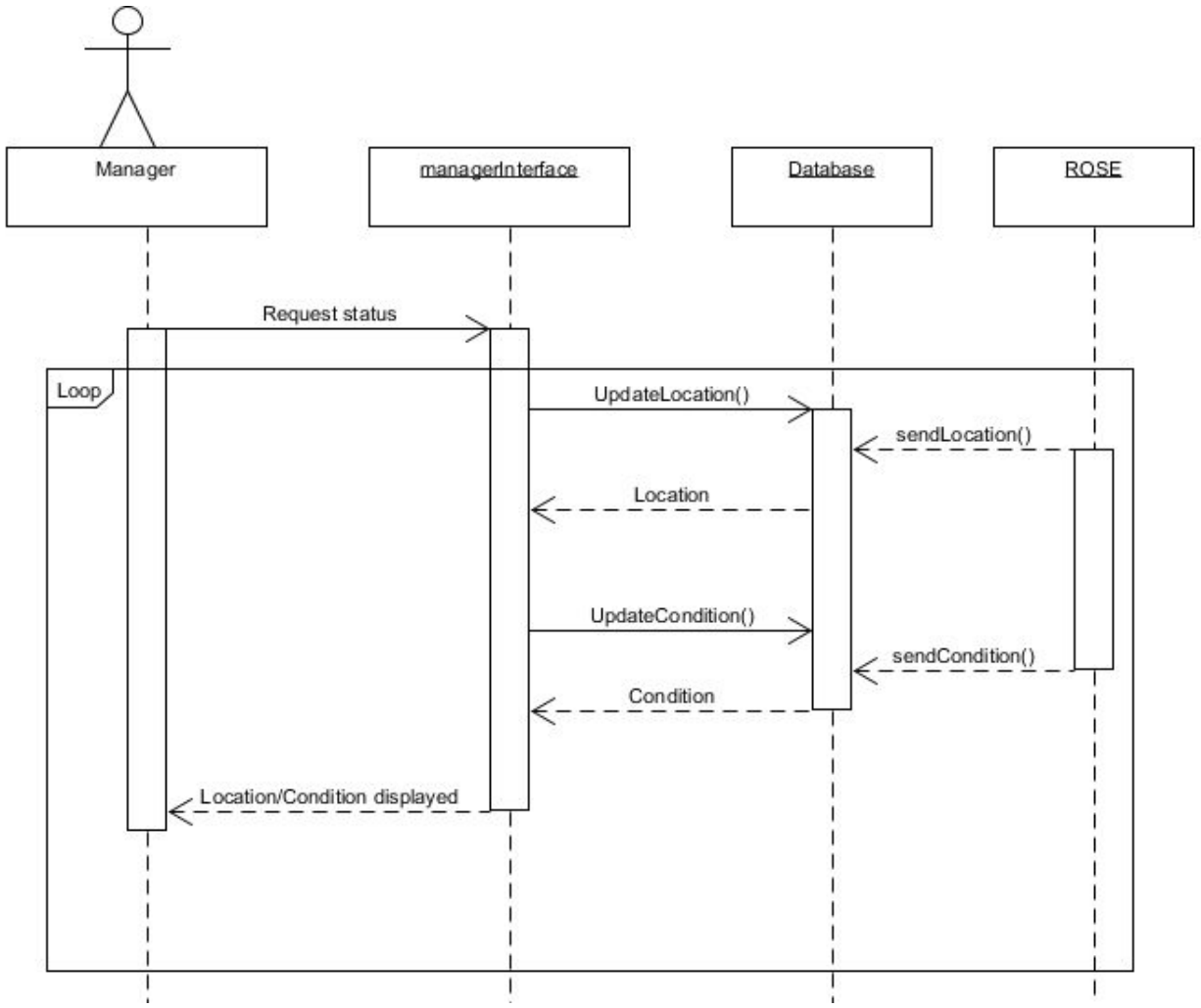


Figure 7.3: This diagram illustrates the interaction between objects in Use Case 3. Given this drawing, we see that the manager requests the status of the robot through the managerInterface. ROSE continually updates the database of its location and condition on its own. The terminal's calls to the database returns ROSE's updated status for the manager to review.

Design Pattern: the primary design pattern here is an Observer Pattern, which essentially tells us that the manager interface is merely used to monitor the robot's status. It is one of the important designs that allows the manager to oversee operations.

Section 8: Class Diagram and Interface Specification

A. Class Diagram

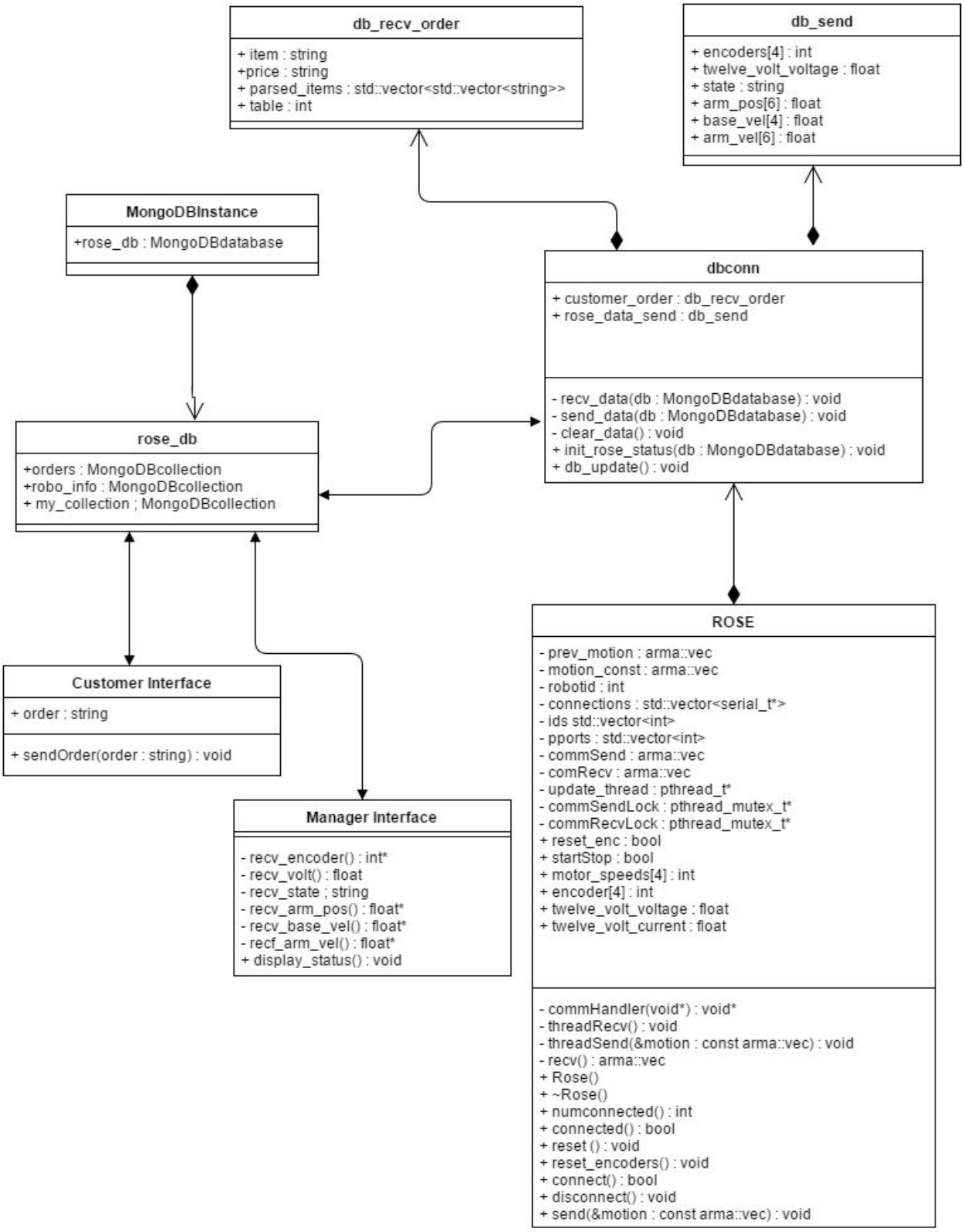


Figure 8.1: Class Diagram for all Use Cases

B. Data Types and Operation Signatures

Customer Interface

Attributes

+ order : string	Holds all items that the customer orders so that their request may be sent to the database

Methods

+sendOrder(): void	This method will send the order to the Controller

Manager Interface

Attributes: None

Methods

- recv_encoder() : int*	Retrieves ROSE's current encoder value from database
- recv_volt() : float	Retrieves ROSE's current voltage value from database
- recv_state : string	Retrieves ROSE's current action from database
- recv_arm_pos() : float*	Retrieves ROSE's current arm position from database
- recv_base_vel() : float*	Retrieves ROSE's current speed value from database
- recv_arm_vel() : float*	Retrieves ROSE's current arm velocity from database
- display_status() : void	Runs all other methods in this class and displays their values on the Manager UI

MongoDBInstance

Attributes:

+ rose_db : MongoDBdatabase	The database that ROSE is programmed to connect to on system startup

rose_db

Attributes:

+ orders : MongoDBcollection	A MongoDB collection that contains the information of all orders delivered by the web app
+ robo_info : MongoDBcollection	A MongoDB collection that contains the information of the status of ROSE during system operation
+ my_collection : MongoDBcollection	A MongoDB collection that contains the control commands from the Manager UI while ROSE is in manual control mode

db_recv_order

Attributes:

+ item : string	Contains all items of an order.
+ price : string	Contains all prices of each item in an order
+ parsed_items : std::vector<std::vector<string>>	2-dimensional vector that contains each item and its corresponding price on each row of the vector.
+ table : int	Contains the table number in which the order came from.

Methods: None

db_send

Attributes:

+ encoders[4] : int	Array that contains each encoder value to determine the position of ROSE
+ twelve_volt_voltage : float	Holds the voltage value of ROSE
+ state : string	Describes the current action that ROSE is performing
+ arm_pos[6] : float	Array that holds the angles of each joint on the arm
+ base_vel[4] : float	Array that contains the velocity of each motor on the base of ROSE
+ arm_vel[6] : float	Array that contains the velocity of each motor on the arm of ROSE

Methods: None

dbconn

Attributes:

+ customer_order : db_recv_order	Contains the order information each customer request. See “db_recv_order” above
+ rose_data_send : db_send	Contains the status of ROSE. See “db_send” above

Methods:

- recv_data(db : MongoDBdatabase) : void	Retrieves an order from the database if the “customer_order” member variable is empty
- send_data(db : MongoDBdatabase) : void	Sends ROSE status information to the database
- clear_data(): void	Removes the order information from the “customer_order” member variable
+ init_rose_status(db :	Initializes the database so that ROSE can

MongoDBdatabase) : void	update the data about its current status
+ db_update() : void	Connects ROSE to a MongoDB instance and runs the “recv_data” and “send_data” methods periodically.

Rose

Attributes

-prev_motion: arma::vec	Vector used to record previous motion carried out by the robot.
- motion_const : arma::vec	Vector holding values of 255. Used to convert double from -1 to 1 to a pwm value from -255 to 255.
-robotid: integer	Variable used to store the id of the robot.
-connections: vector<serial_t*>	Vector used to hold all of the serial data connections.
- ids : std::vector<int>	Vector holding DEV_IDs of all current connections.
-pports: vector<char*>	Vector used to contain the ports.
- commSend : arma::vec	Handles outgoing communications to arduinos
- commRecv : arma::vec	Handles incoming communications to arduinos
+update_thread: pthread_t*	Thread variable used to hold the updated thread.
+commSendLock: pthread_multex_t*	Thread used to hold the lock which will be sent when the corresponding function is called.
+commRecvLock: pthread_multex_t*	Thread used to hold the lock that is being received.

+ reset_enc : bool	Used to hold the state of if the encoders need to be reset on the arduino or not
+ startStop : bool	Used as a sort of "signal handler". False by default, Rose will stop if it is set to true.
+motor_speeds: integer	Variable used to set the speed of the motor.
+encoder: integer	Variable used to convert information to another format.
+ twelve_volt_voltage : float	Holds voltage of 12V battery powering TK1
+ twelve_volt_current : float	Holds current draw from 12V battery powering TK1

Methods

- commHandeler(void*) : void*	Private method for handing data transfer over serial. Sets up and maintains mutex locks for sending & receiving
- threadRecv() : void	This method receives serial data in its own thread.
- threadSend(&motion : const arma::vec) : void	This method sends serial data in its own thread.
- recv() : arma::vec	This method adds a lock to wait until the commthread is done setting the vector.
+ Rose()	Creates Rose object to interact with robot, as well as connects to robot using connect()

+ ~Rose()	Disconnects Rose object from robot
+numconnected(): integer	This method gives the number of connections.
+ connected() : bool	return The status of Rose's connection
+send(&motion: const arma::vec): void	This method locks the data before setting it and then sends the motion command.
+reset(): void	This method is used to set the previous motion to zero.
+ reset_encoders() : void	Resets values of encoders back to zero, on the arduino side
+connect(): boolean	This method is used to connect to all the arduinos currently mounted on the system.
+connected(): boolean	This is the default connection method which returns true if connected.
+disconnect(): void	This method is used to disconnect to all the arduinos currently mounted on the system.

C. Traceability Matrix

	Customer Interface	Manager Interface	ROSE	dbconn	rose_db	MongoDBInstance	db_send	db_recv_order
Controller								
TouchInterface	**X**							
Order								**X**
FoodList	**X**							
Bill	**X**							
OrderAdder	**X**							
OrderQueue					**X**			
ItemAdder	**X**							
WarningPopup	**X**							
OrderPusher	**X**							
Payment Acceptor	**X**							
MenuDisplay	**X**							
ConfirmPopup	**X**							
Database Connection						**X**		
StatusChecker		**X**						
OrderParser								**X**
ItemChecker								

OrderDeliverer			**X**					
StockChecker								
Authenticator								
User								
UserList								
DataDisplayer		**X**						
ActionQueue		**X**						
DataReceiver				**X**				

Figure 8.2: In the figure above, the classes from domain concepts are in the first column, while the classes that evolved from the domain concepts are in the first row. This has changed since last report because we no longer use terminal or controller in our finalized project design. Additionally, some of the classes that we have now were not derived from the domain concepts. Note that some domain concepts are no longer used as it was not implemented in the final project.

CustomerInterface Class

This class evolved from several domain concepts: TouchInterface, FoodList, Bill, OrderAdder, WarningPopup, PaymentAcceptor, MenuDisplay, and ConfirmPopup. Most of these concepts have been integrated into the class because they are interdependent in allowing the customer to interact with the virtual menu. The pop-ups give the customer feedback when the order is about to be and is confirmed to be placed into the system. It also should display the bill and accept payments when the customers are ready to leave. The touchInterface allows the customer to interact with the various buttons we will implement on the web application. FoodList shows the items available, and OrderAdder pertains to having the web application retain an order cart for everything that the user requests. This interface is also in charge of pushing order data to the database.

ManagerInterface Class

Domain concepts DataDisplayer and ActionQueue will display the position and status of the robot in real-time as well as the list of actions that the robot is currently or will be executing. Information returned from the terminal will allow the manager to monitor the whereabouts of the robot, and he or she can choose to modify actions from this interface.

dbconn Class

This class receives data from the customer interface in the form of orders as well as sends that data to ROSE so the robot can deliver the order to the customer.

rose_db Class

Derived from domain concept OrderQueue, this class holds the orders in a FIFO method so that customers are served in the priority of who sends their request out first.

MongoDBInstance Class

This class was derived from the Database Connection domain concept. It simply checks to see if a database connection is established.

ROSE Class

The domain concept of OrderDeliverer is part of the ROSE class, where the robot is in charge of delivering the orders to the correct table.

Db_recv_order Class

Derived from the domain concepts OrderQueue, OrderParser, User, and UserList, this class stores the customer's orders in queue, and breaks it down into smaller commands that the robot can understand. Additionally, it stores credentials that the terminal will verify against the manager, whose login request is sent through the managerInterface, then through the terminal.

db_send Class

This class was not derived from the domain concepts; it is used to send encoder values to the robot so that we can use the web application to adjust the speed of its movement as well as its direction and rotation.

D: Design Patterns

The ROSE System utilizes a variety of different design patterns in its implementation:

- Mediator Pattern: When a customer places an order, the information contained in the order (items, prices, and table number) is passed to a Mediator (the webapp server) that handles all orders, and passes them into the database.
- Interpreter Pattern: Code hosted on the robot then uses the Interpreter pattern by taking the information from the database and deciphering it for later use.
- Observer Pattern: The manager's page is to serve as an Observer pattern (except in circumstances requiring a manual control override), since it merely exists to observe the changes in the queue that holds the orders, as well as any status changes in the database concerning the robot.
- State Pattern: When in manual override, the robot's movement is determined in a manner that reflects a State pattern, because each input is dealt with as a behavioral state.
- Strategy Pattern: When it is performing regularly and autonomously, it performs under a pattern that more resembles a Strategy pattern, in that it uses different algorithms to determine its behavior under certain conditions.

E. Object Constraint Language (OCL) Contracts

List important contracts (invariants, preconditions, postconditions) for classes and their operations

db_recv_order Class

```
Context db_recv_order inv:
    Self.table >= 0
```

db_send Class

```
Context db_send inv:
    self.encoders.size() == 4
    self.arm_pos.size() == 6
    self.base_vel.size() == 4
    self.arm_vel.size() == 6
    -- Note: the size of these arrays must be exact in order to work properly
```

Manager Interface Class

Context Manager Interface inv:

self.recv_volt() > 0

-- Note: voltage should be greater than 0; otherwise, we would not receive

ROSE class

Context ROSE inv:

Self.numconnected() >= 0

-- Note: cannot have negative number of robots connected

Section 9: System Architecture and System Design

a) Architectural Styles

The system implements client-server and component-based architectural models, and also utilizes facets of domain-driven design. Explanations of each of these styles are as follows:

- This system will involve a client-server architectural model, where the robot will function as the server hosting the database and web application, and the tablets available on the restaurant tables will represent clients accessing the web application (thus drawing on the application server model). The client and server will be connected locally (on the same network), and the client will make requests to the server (i.e. order menu items) using the available GUI illustrated previously in this report.
- The system also utilizes a component-based architecture, with the component architecture of the wep app consisting of various buttons that users can click in order to place orders (which will then be placed onto the database for processing by the robot).
- The system draws on the domain-driven design, since it is only applicable in an environment that contains properties that allow the robot to identify a table location (e.g. chilitags). It also requires the presence of “stock”, or the currently available inventory of items from which the robot can retrieve ordered items.

b) Identifying Subsystems: UML Package Diagram

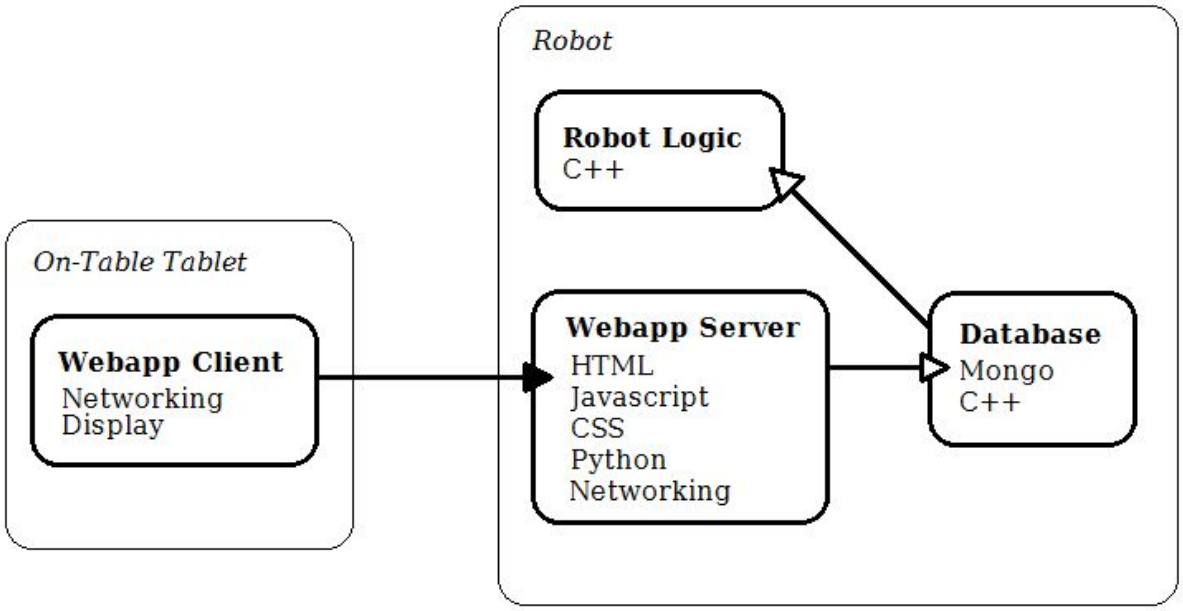


Figure 3.1: UML Package Diagram

Figure 9.1: An illustration of the subsystems present in our system, along with the software tools implemented within each of them.

The above UML package diagram illustrates that our system is composed of 4 main components:

1. Client-facing web application: This is the component that the user (restaurant customer) interacts with, through an on-table tablet. It requires a reliable, secure connection to the restaurant's network, and is composed of HTML/CSS in the form of templates that are displayed to the user (with buttons to select items).
2. Server-side web application: The server is configured using the Flask framework and uses it to reroute users to different pages, process input (i.e. placed orders), and add these items to the MongoDB database.
3. Database: This subsystem is a MongoDB instance that is used to hold various data, such as placed orders and current robot state.
4. Robot logic: This subsystem consists of ROSE, the robotic server that will process orders and bring them from the kitchen area to the respective table. ROSE incorporates a host of algorithms and data structures outlined in Section 10 of this report, and is implemented in C++.

c) Mapping Subsystems to Hardware

As can be gleaned from Figure 3.1, the software subsystems have been divided into two hardware systems; their division is such that the On-Table Tablet handles the smallest load possible. This serves three main purposes:

1. This division of subsystems minimizes the load placed on the network, since the only connection required by this system is between a Tablet and the Robot.
2. This minimizes the latency between customer orders and the robot's actions.
3. This minimizes the cost of production of the collection of systems.

The only alternate, sensible division of subsystems would be to place the Database and WebApp on an additional piece of hardware; however, while this would minimize the processing load on the robot, it would require the utilization of networking capabilities between both other systems. This increases the latency between the orders and the robot's actions significantly, and makes the system much more sensitive to an increase in the number of Tablets.

An additional benefit to this system is that it allows for a minor change in the way the system interacts with the restaurant. Instead of requiring the restaurant to purchase a tablet for each table, the design may change to allow customers to access the server and order their food from their own devices. This would significantly decrease the overall cost of the system.

d) Persistent Data Storage

The system will need to store persistent data, specifically three collections:

1. The collection that contains information on placed orders.
2. The collection that contains information regarding the robot while being operated through the manager override feature.
3. The collection holding information regarding the robot outside of the manager override feature (i.e. as it autonomously processes orders).

These collections need to persist beyond a single instance of program execution so that a hardware malfunction does not "clear" the future actions for the robot to complete. This data will be stored specifically in MongoDB BSON-format collections.

The schema of these collections is as follows:

Collection 1: Orders

Attributes	Descriptions
Prices	A string representing the prices of the items that form this order.
Table Number	The table number where this order should be delivered.
Items	A string representing the items that form this order.
parsed_items	Contains the parsed strings of each item and their prices

Collection 2: mycollection

Attributes	Descriptions
State	The state entered by the manager through the manager override in the web application. This can be one of the cardinal or a combination of the cardinal directions (e.g. N, NE, etc.)
Speed	The speed of the robot set by the manager, ranging from 0 to 1 in 0.05 increments.
Rotation	The direction of rotation of the robot set by the manager; can be set to CW, CCW, or NONE.

Collection 3: robo_info

Attributes	Descriptions
twelve_volt_voltage	Holds the current voltage.
state	Holds the current action of ROSE

encoders	Holds the encoder value of each arduino on ROSE's base
arm_pos	Holds the angles of each joint on ROSE's arm
base_vel	Holds the velocity value of each motor on ROSE's base
arm_vel	Holds the velocity value of each motor on ROSE's arm

e) Network Protocol

ROSE will be communicating with a server hosted on MongoDB. The Mongo Wire Protocol is a simple socket-based, request-response style protocol. The client (ROSE) will communicate with the database server through a regular TCP/IP socket. We will also be using SSH to remotely access ROSE. For the web application we are using the TCP/IP protocol. We are using Flask to make our web application and it is a WSGI framework. It will only work on TCP/IP and HTTP protocols.

f) Global Control Flow

- Execution orderliness: Our project is procedure-driven. A user will order an item on the application and the order will get placed in a queue(first in, first out procedure). So regardless of how users make orders, ROSE will deliver them in a linear fashion.
- Time Dependency: ROSE is an event-response type system with no concern for real time. It will deliver items when an order gets placed, so it is not a real-time system.
- Concurrency: ROSE will be a multithreaded system. For the Robot itself we have an SDL thread for input and display, a main robot thread for taking updates and a PID thread for accurate motion. To do localization we use a planner thread, filter thread and an image processing thread. For thread communications with arduinos, we have a sender thread, a listener thread and a communications manager thread. For web application communications we have one thread and another thread for mongo. All communication threads can communicate with each other. There are mutex locks to prevent corrupted data when sending and receiving. There is communication between the PID and main robot thread to make the robot move in the appropriate direction.

g) Hardware Requirements

The system utilizes the following resources:

- ➔ Raspberry Pi / Laptop (To run Mongo)
- ➔ Vex Parts (to build robot frame)
- ➔ 22 Vex Motors
- ➔ 16 Vex-Encoders
- ➔ 2-3 7V Batteries (for Arduinos)
- ➔ 1 12 V Batteries (for Jetson TK-1)
- ➔ Jetson TK-1 (ROSE)
- ➔ Battery for USB HUB
- ➔ 8 port USB HUB
- ➔ 1 Camera (PS3-eye)
- ➔ Tablet (To run restuarant application)
- ➔ 3 Circuit Board and Wires (For encoders...)
- ➔ 2-3 Arduinos (For motors...)
- ➔ Router
- ➔ USB Hubs

Section 10: Algorithms and Data Structures

a) Algorithms

Particle Filter / Localization:

To do localization using a particle filter, the robot first detects various chilitags placed in a predetermined location throughout a room. Then based on the distance compared to various landmarks (chilitags) we calculate the distance between the robot and the landmarks to determine the robot's location in a predetermined setting. The particle filter algorithm uses a number of simulated robots, with each have their own random amount of noise. Each one of these simulated robots, or particles, is then allowed to move about the map in its own way. Each particle takes in information from the landmarks the robot sees, as well as motion information coming from the robot itself. At the most basic core of the algorithm, an average of all the particles is taken, along with some measurements about error and deviation, and this is taken as the location of our robot.

Edge Detection / Chilitags:

We will use edge detection to aid in the localization, object detection and identification. The algorithm used to detect chilitags utilizes edge detection to determine if a set of edges is a chilitag. Edge detection algorithm involves 3 steps. First, we use *filtering* to improve the performance of an edge detector with respect to noise. More filtering to reduce noise results in a loss of edge strength. In order to facilitate the detection of edges, it is essential to determine changes in intensity in the neighborhood of a point. Second, *enhancement* emphasizes pixels where there is a significant change in local intensity values and is usually performed by computing the gradient magnitude. Next we have *detection*. We only want points with strong edge content. However, many points in an image have a nonzero value for the gradient, and not all of these points are edges for a particular application, so we set thresholds based on the criterion used for detection(for example colors).

Forward/Inverse Kinematics:

We will use forward and inverse kinematics in the movement of the robot's joints. These algorithms are used to compute the current location of the arm end effector given the joint angles, and to determine the joint angles given the spatial location of the end effector, respectively.

The algorithms are explained as follows using the following arm diagram:

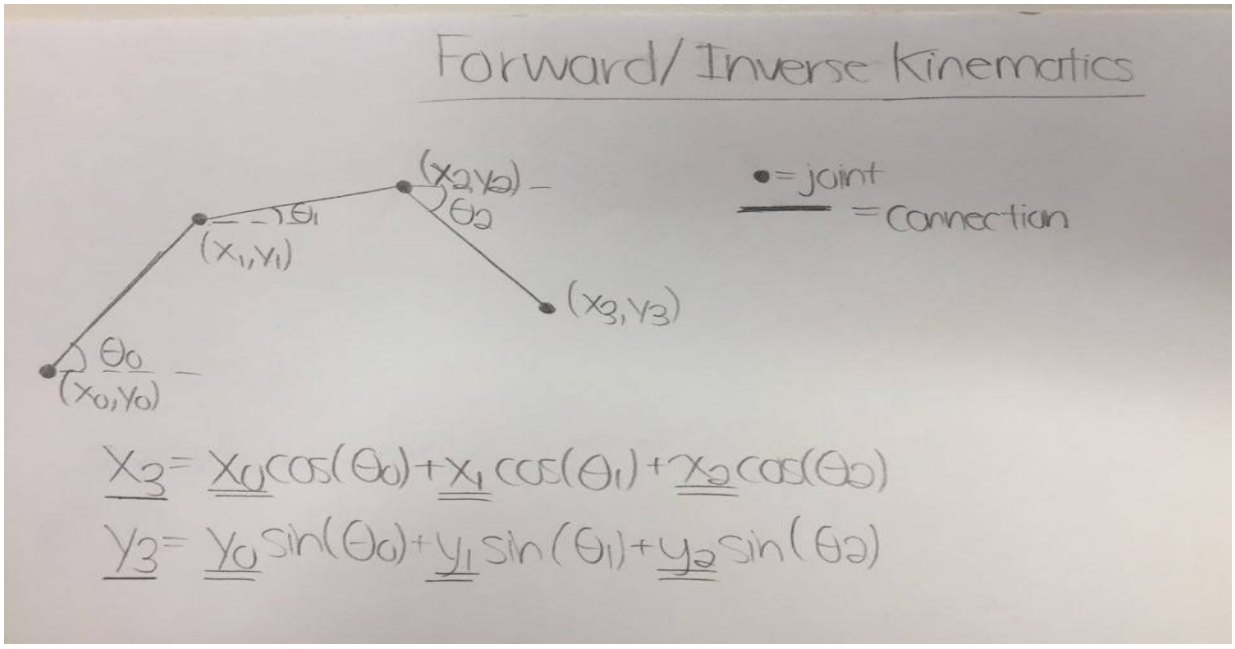


Figure 10.1: A sample illustration of a robot arm, with 4 joints and 3 joint connectors in 2 dimensions. The spatial coordinates are shown as $(x_0,y_0),...,(x_3,y_3)$, and the joint angles with respect to the +x axis as $\theta_0,...,\theta_2$, and the two equations needed to calculate the rightmost joint location are given as well.

Forward Kinematics:
This algorithm simply requires solving a system of linear equations as shown above for the coordinates (x_3,y_3), given the joint angles; since this requires solving two equations with 3 known variables, this can be solved. In fact, in our case of 3 dimensions, we can still solve 3 equations for (x_n, y_n, z_n) using this procedure, since all angles are still known.

Inverse Kinematics:
Using the same illustrated example above, we can see that it requires solving for the 3 unknown angles given the coordinates of the joints (2 known variables); thus, in this case (as well as our system's case), the angles cannot be determined exactly. Instead, numerical and graphical approximations must be used. Our system utilizes a binary search algorithm that draws on the ideas of Monte-Carlo algorithms, in that it adjusts joint angles from 0 to 180 degrees in 5-degree increments.

PID:
Our system implements a Proportional-Integral-Derivative (PID) controller as an error-correcting mechanism for both robot and joint movement. This controller is implemented as follows:

Continuously move straight, computing error factors (one for proportion, integral, and derivative) based on the difference between the average encoder value and the value of each encoder. The proportion error's value is the error itself, that of the integral is the sum of the errors, and that of the derivative is the difference between the encoder values. After doing so, we adjust the motion of each wheel by taking into account each of these terms, as follows:

motion[i] = speed + (error[i] * k_p) + (error_sum[i] * k_i) + (error_diff[i] * k_d),
Where motion[i] indicates the speed of wheel i, k_p, k_i, and k_d are the proportion, integral, and derivative constants, and the errors are the error factors defined above.

Using this algorithm, our robot can accurately move to a destination and adjust its arm to grab an item.

Database Algorithms:

1) recv_data(db : database):
This algorithm allows ROSE to extract an order from the database and save the order information on a struct. After ROSE reads an order from the database, It will proceed to destroy the order within the database. However, because ROSE stored the order information into a struct first, it will not forget this order until the order has been carried out. In addition, each item string of an order will be parsed and organized onto a 2-dimensional vector such that ROSE can access any element of any item it receives. When there is not any orders available to carry, this method will return without attempting to store any information within ROSE's data structure.

2) send_data(db : database) :
This algorithm will be used by rose to send various data about ROSE's status. With it's status stored within a data structure, ROSE is designated to send its information periodically to the manager UI.

3) db_update():
This algorithm will connect ROSE to a MongoDB instance and the database "rose_db". Afterwards, it will run recv_data() and send_data() within a while loop, allowing ROSE to continuously carry out orders and send information to the manager UI.

4) init_rose_status(db : database):

This algorithm will initialize the database with default values for each piece of data that ROSE must send to the manager interface. This is done for two reasons. One, this resets values on system startup, so that information about ROSE's status is always accurate. Two, in order for ROSE to update the documents of the MongoDB collection, each document must first exist within the collection.

5) clear_data():

This algorithm is used to clear the "customer_order" instance of the "data_recv_order" data structure so that ROSE can receive new orders once it is done fulfilling its current order.

Web Application Algorithms:

1. Client and server communication: We form this communication using XMLHttpRequest() to form a request to the server (either 'GET' or 'POST' requests), and send responses from the server by retrieving database data and using jsonify() to create JSON objects to return to the client for them to parse and process.

b) Data Structures

Menu list:

Prior to rendering the customer interface, the information about the items to be displayed on the customer UI will be stored within a list so that the item information may be accessed again by a separate device. All items will be stored within a linked list so that the manager will be able to easily add or remove items from the list at their discretion.

Mongodb database:

Used to collect information of various types. Each type of information is contained in a collection of the database, which has an arbitrary number of documents associated with it.

1) Order information from the customer

The documents of this collection include parameters that defines an order placed as an “item” (includes item name and price), as well as the timestamp in which the order has been placed, the order number, and the table number in which the order came from. Orders within this collection will be removed in a FIFO fashion. As each order has been delivered, the order will be removed from the list and the next order to be fulfilled will be the oldest order of the collection.

2) Information about the location and velocity of the robot

This collection will gather information about the kinematics of the robot’s base. This collection will have 8 documents in total, which corresponds to the displacement and velocity of each of the 4 wheels on the robot.

3) Positioning of the robot's arm

This collection will contain information about the positioning of the robots arm. Will have 3 documents in total which corresponds to the x, y, and z components of the arm.

4) The “dbconn” class

THis class contains methods and structs that allow ROSE to exchange information with the web app. Details of all methods are in the “algorithms” section

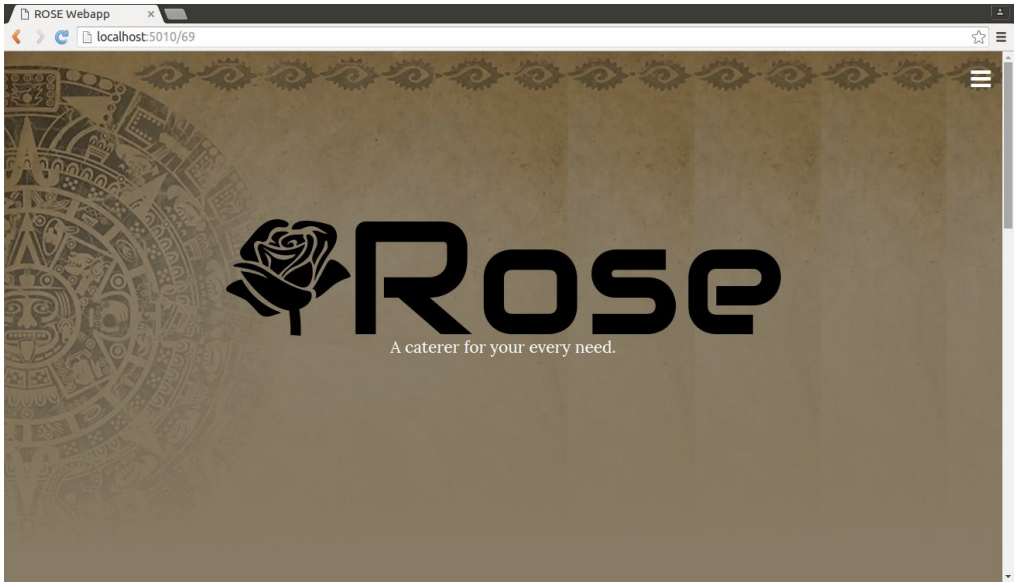
- db_recv_order: Struct that allows ROSE to remember an order after ROSE has destroyed the order from the database
- Db_send_order: Struct that contains all the information that ROSE is designated to send to the manager interface.
- recv_data(db : database): Private method that allows ROSE to retrieve order information from the database
- send_data(db : database): Private method that allows ROSE to send status information to the database

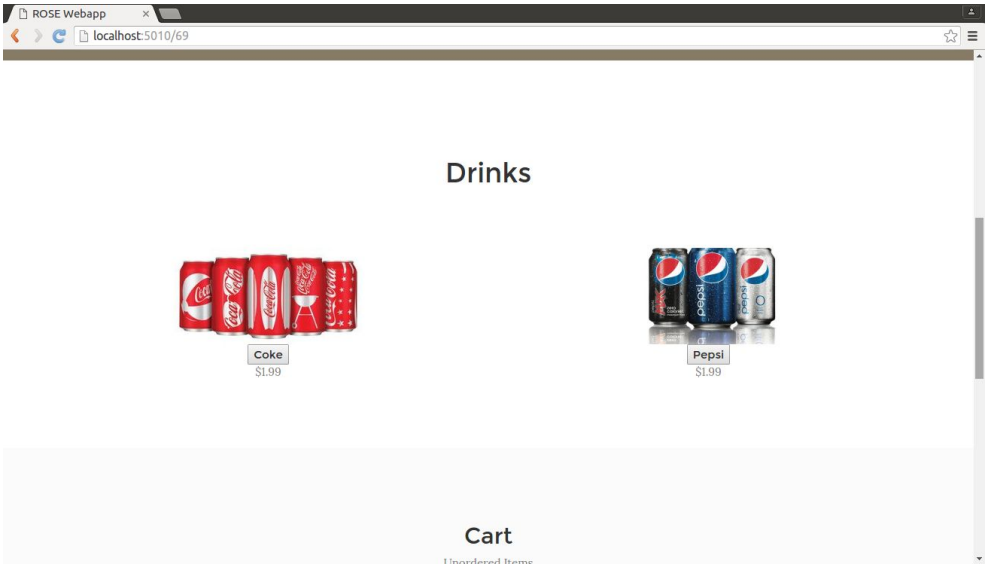
- db_update(): Public method that connects ROSE to a MongoDB instance and proceeds to run recv_data() and send_data()
- init_rose_status (db : database): Public method that initializes the database in preparation for sending ROSE's status.

Section 11: User Interface Design and Implementation

A: UI Design

i. *Customer User Interface*





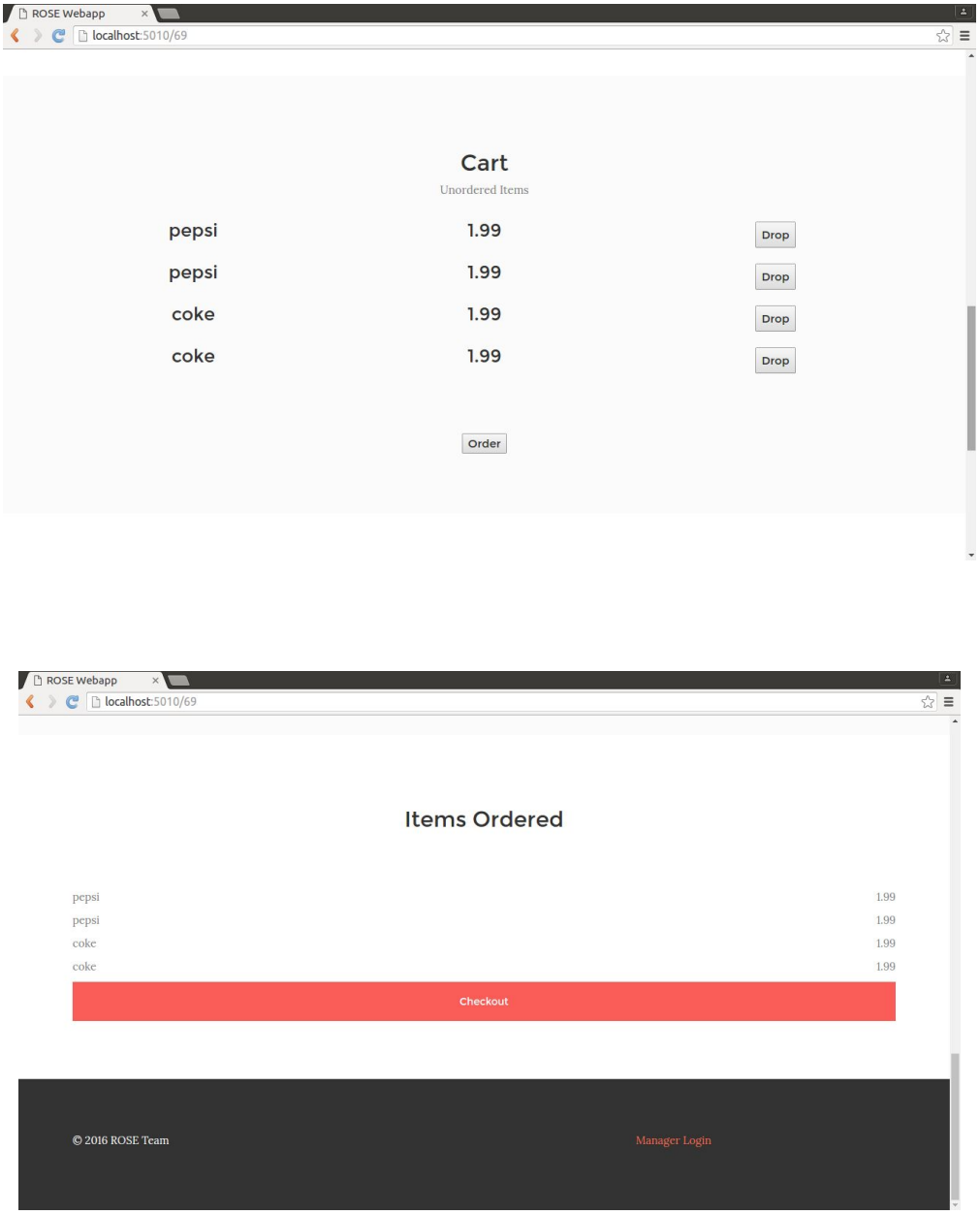


Figure 11.1: These screenshots are the finalized web application pages for the customer user interface. It allows customers to order a drink with the press of a button, review the items in a card, and confirm the order via the “checkout” button.

ii*. Manager User Interface*

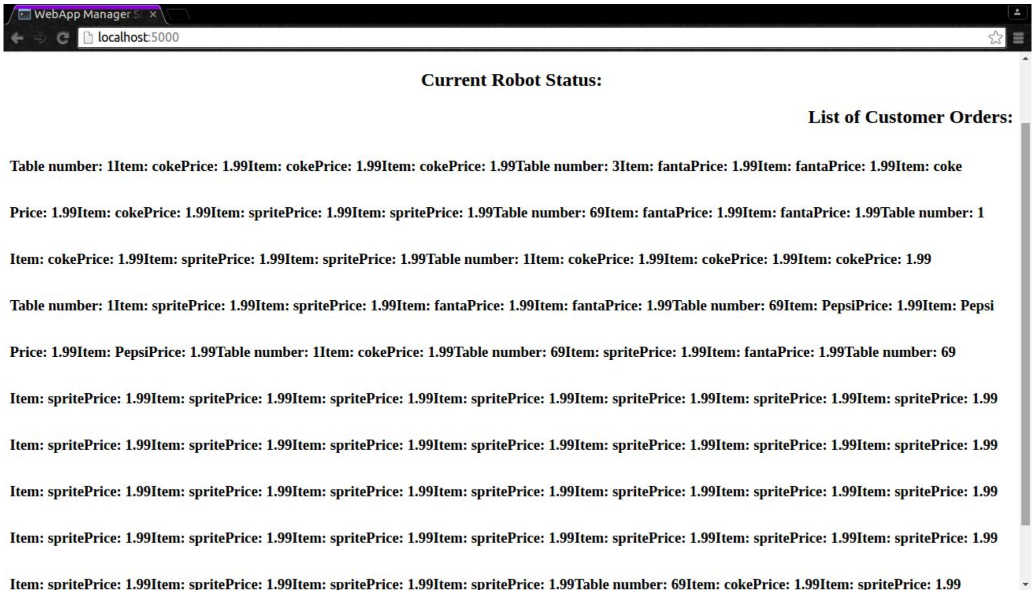


Figure 11.2: This screenshot shows the updated manager interface. This part of the web application allows the manager to view the list of orders, prices, and table number.

B: UI Implementation

i. *Customer Interface*

The customer interface is displayed when a device connects to the wireless router on the robot, and routes to a URL the IP of the robot and the table number using its browser. When a customer then places an order, the information contained in the order (items, prices, and table number) is passed to the webapp server, which inserts it into the database for later use by the robot's order queue handling code. The information is then moved to a different collection within the page that represents items that have yet to be paid for, which the customer can clear from the page by pressing the "Checkout" button. To avoid mistakes in orders, any items that have been added to the cart may be dropped before they are ordered, and confirmation popups will rise after the customer attempts to either place an order or check out.

ii. *Manager Interface*

The manager interface is meant to keep tabs on the state of the robot and the orders within the database, to make sure that the system is working correctly. If the robot begins to behave in a manner that deviates from its standard behavior, the manager may route to the manual override page, which disables the autonomous functionality temporarily and gives the manager control of the robot's movement.

Section 12: Design of Tests

a) Test Cases:

List and describe the test cases that will be programmed and used for unit testing of your software

1. Edge Detection - First, we wrote a program such that it can recognize tennis balls. The program will take in camera feed and when there is a tennis ball shown in the camera feed, the program will recognize it and focus on it(a small circle will appear around the tennis ball). For our second test, we first printed some chilitags out. Then we started taking in camera feed and placed the chilitags in front of the camera to check if the program was able to recognize them. For our next test case we passed in various images and asked it to recognize all the red dots on it.
2. Particle Filter - The premise behind the particle filter is that it will allow us to localize based off of landmarks, which we know the already locations of. Currently, we are using chilitags for these landmarks. After knowing that detection for the chilitags was working as well as possible, we decided to go ahead and set up a small experiment in the B hallway of the engineering building. With chilitags set up on the walls, as well as a predetermined map of the hallway created, we tested the particle filter algorithm to see if we would be able to track our location relative to our predetermined map. Our map was created so that 1 pixel was equivalent to 1 inch, so we were able to verify if the displayed location was accurate. We found that when many landmarks are in view, we were able to determine our location very precisely, but when landmarks are sparse, our determined location was not usable. We now know that we have to improve the algorithm when landmarks are not in view, so we are taking approaches such as improving the motion model, as well as attempting to predict where we will move based on our current location and the direction we were moving in previously. For our test, we set the starting position as next to the kitchen and moved forward and watched the simulation update itself on the current location of the robot after moving forward for 2 seconds.
3. Inverse Kinematics - We made an application that allows us to control the arm on our robot with keys. This showed that arm could be controlled manually to pick up objects. With the arm functioning with manual control, we needed to calculate the actual position we wanted to arm grab in the frame the camera was seeing. This algorithm is still under development, so we have not had a chance to fully test it yet. To make sure the arm was working, we wrote a program so that the arm to a position based on the x,y,z coordinates specified. The arm was

successfully able to go the position with a small margin of error, that can be fixed using PID.

4. PID - We made an application that allows us to control the motion of our robot with keys. We then held the key for moving front, back, left and right to test whether the robot was capable of moving in a straight line. Next, we placed a chilitag 20 meters in front of our robot. Then we tested whether the robot was able to drive straight, autonomously to the chilitag.
5. The database - We created a simple program in both python and C++ to test pushing information to a remote database. In both python and C++, we created a simple hello world document that would be assigned a timestamp whenever we pushed the information to the database. In addition, we have tested connecting to a database remotely by the use of a local router so that a tablet can connect with the robot and exchange information with it by means of the database. Lastly, we have tested database entry and retrieval of items ordered from the web app.

b) Test Coverage:

Discuss the test coverage of your tests.

Our tests cover all the main functions of our robot. They check if the robot has the capability to recognize objects using edge detection algorithms. We check if the robot can move in an efficient manner using PID. For example, without PID, if we hit the right arrow key to move east, the robot may actually move north east. This will cause the robot to do a lot of readjustments when we run the robot autonomously. We test localization to check if the robot is able to continuously update its location based on its current location and the landmarks around it. We also test if the arm is able to move and is able to grasp an object. This covers all the major functions of the robot. We also test our webapp which is an addon to our robot. It will allow a customer to order something, which will then a push a request to the database. Then we test if the robot is able to get this information from the database.

c) Integration Testing:

Describe your Integration Testing strategy and plans on how you will conduct it.

For our integration testing, we will first combine various unit tests to check if they work together and if they do not, then we will revise our software such that it works. First, we will try sending data from the robot to the database. This will let us know that the robot is connected to the database and it is able to send information to the database about itself. We will then combine edge detection algorithms with the particle filter to confirm that the robot is able to recognize chilitags and determine its location based on the probability models. Next, we will add PID to the process, such that the robot is able to send information, determine its location, move in an accurate way and update its location as it moves around in the setting. This will be the hardest step because the robot has to keep updating its location and if it doesn't recognize a landmark, then it will affect our results. Also, if PID doesn't work properly, it makes it hard for the robot to move in an efficient manner. Similarly, we will order something to the webapp to successfully send an order to the database. This shows that the webapp is connected to the database as well and is able to send information to the database. The we add one more step to the process, which is having the robot access the information sent by the webapp. Once the robot, gets the information from the database, it will go to a predetermined location and pick up an object. Then it will go from that location to another predetermined location in the setting with that object. We will combine small parts together to complete an actual task such as sending information to a database, recognizing objects and moving based on that.

Section 13: History of Work

Web App:

- Researched various frameworks to utilize and decided to use Flask
- Created simple Flask scripts using online tutorials to familiarize ourselves with the framework, as well as simple templates to understand the template rendering functionality
- First demo: Created a basic functional Web Application, featuring the ability to move the robot in 8 directions by pushing to a Mongo database
- Created a basic functional Customer Interface Web App, which allowed customers to order items by pushing them to a database
- Second demo: Added aesthetic elements to the Customer Interface; reorganized the Customer Interface for ease of use by the customer
- Created a basic functional Manager Interface Web App, which allowed managers to observe the order queue in addition to the operating status of the robot

- Streamlined initial “basic Web App” and converted it into a manual override option

One of our main goals for the web application was to have a functioning interface by March 29. However, we were unable to complete this goal. By our second demo, however, we had a fully functioning web application.

Looking forward, the web application can be extended by allowing customers to select appetizers and entrees as well. Moreover, we can also allow the manager to see more information regarding the robot, such as its current status, its exact location in the restaurant, and encoder/potentiometer readings to check for a malfunction.

Database:

- Jan 11 - Jan 25: Download MongoDB for Python and C++. Learned how to write a python script and a C++ script which incorporated MongoDB.
- Jan 25 - Feb 2: Learned how to create a client/ server connection with MongoDB. created Hello World scripts in Python and C++ which incorporated server/ client connection
- Feb 8 - Feb 15: Tested cross-platform communication by pushing to a MongoDB database with a Python script and pulling from the same database with a C++ script.
- Feb 15: Began incorporation of database with web app by pushing in key strokes that are recorded from the web app
- Feb 25: Began incorporation of of database with robot by pulling values from the database that were sent there by the web app, thus confirming communications with the web app.
- Mar 7 - Mar 21: upgraded the database code for both the web app and the robot such that the robot’s speed, cardinal direction, and rotation could be controlled directly from the web app
- Mar 28 - Apr 11: changed the database code such that the customers can order drinks from the database. The information about their order was then passed to the robot. Finally, the robot would parse the Items it received from the database.

Future improvements will involve optimization of database code for maximum performance and testing and implementing database code to synchronize for multiple requests to the database so that multiple robots can be used in the system at once.

Computer Vision:

- February - Setup openCV and fiducial marker detection.
- March - Tested various object detection methods such as SVM’s, haar cascades
- April - Tested color segmentation and neural networks for object detection.

Finally worked with chilli tags to detect objects and landmarks. Added these algorithms on the the robot.

Robotics:

- January - built the base of the robot and implemented motion
- February - built an arm for the robot and implemented IK for movements of the arm
- March - Started testing localization using a particle filter
- April - Integrated webapp, database, computer vision and localization on the robot for performing the tasks of a waiter.

Section 14: References

http://www.learnaboutrobots.com/forwardKinematics.htm
http://cdn.intechopen.com/pdfs-wm/379.pdf
http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_ChApter5.p
http://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html
https://en.wikipedia.org/wiki/PID_controller
https://en.wikipedia.org/wiki/A*_search_algorithm
https://en.wikipedia.org/wiki/Particle_filter
http://chili.epfl.ch/software
https://en.wikipedia.org/wiki/Inverse_kinematics
http://www.mathworks.com/discovery/edge-detection.html
https://en.wikipedia.org/wiki/Speeded_up_robust_features
https://en.wikipedia.org/wiki/Support_vector_machine