https://github.com/ajay/ROSE

Second report (Design only) - Full Report #2

Software Engineering (14:332:452)

Professor Ivan Marsic

Due: March 13, 2016

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**
Project management	16	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 1: Interaction Diagrams	30	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 2: Class Diagram and Interface Specification	10	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 3: System Architecture and System Design	15	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 4: Algorithms and Data Structures	4	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 5: User Interface Design and Implementation	11	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 6: Design of Tests	12	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 7: Plan of Work	2	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Section 8: References	5	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %
Total	100	14.3	14.3	14.3	14.3	14.3	14.3	14.3	100
Total Percentage	-	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	14.3 %	100 %

Section 1: 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
Controller - receives order from user input and sends the order to the database and each available item to the customerInterface
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.
Terminal - monitors the robot and sends information about the robot to the managerInterface
ManagerInterface - Interface for the manager to be able to monitor the robot's movements 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

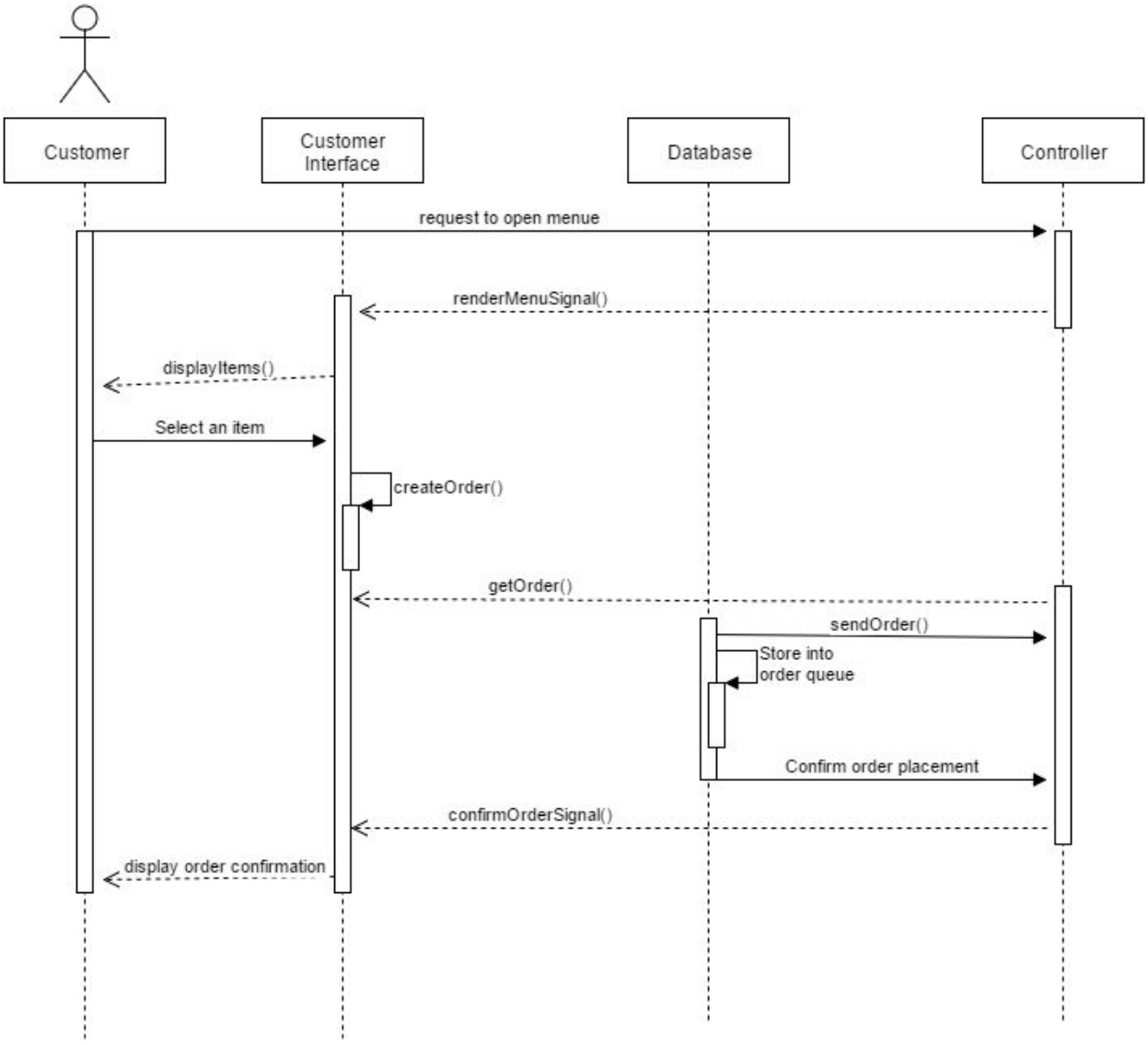


Figure 1: this use case describes how the customer will be able to send information to the database to be received by the robot. This use case involves a controller that will pass information between the events on the customer interface and the database itself. the database will pass information in a first in, first out fashion.

Use Case 2: Delivering the Order

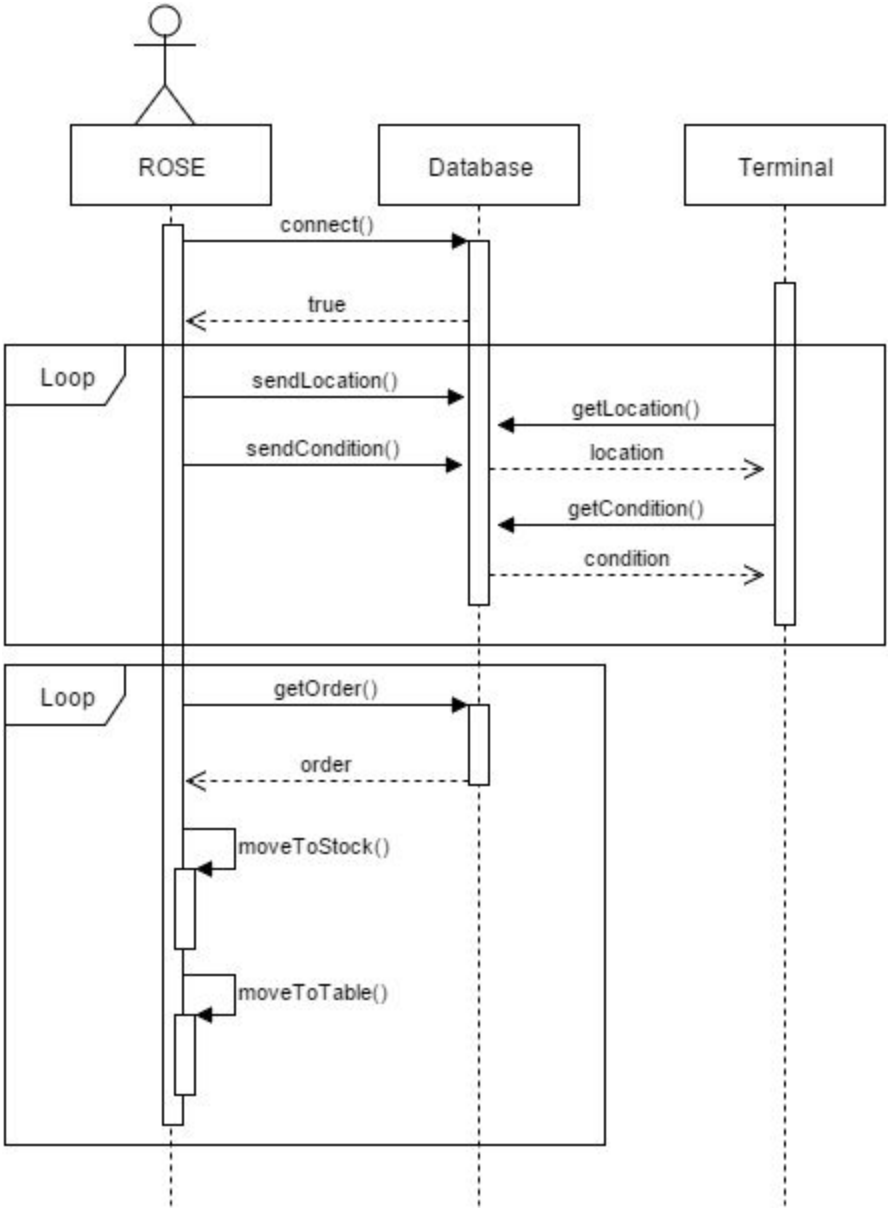


Figure 2: ROSE begins to connect to the database and sends its status as well as location. Additionally, the terminal updates the current location and condition from the data it receives from the database. This loops every second. Inside the second loop, we have to robot getting the order (order is the class containing item ordered, tableID and orderID) from the database, picking up the item(s) ordered and delivering them to the customer.

Use Case 3: Reviewing the Order

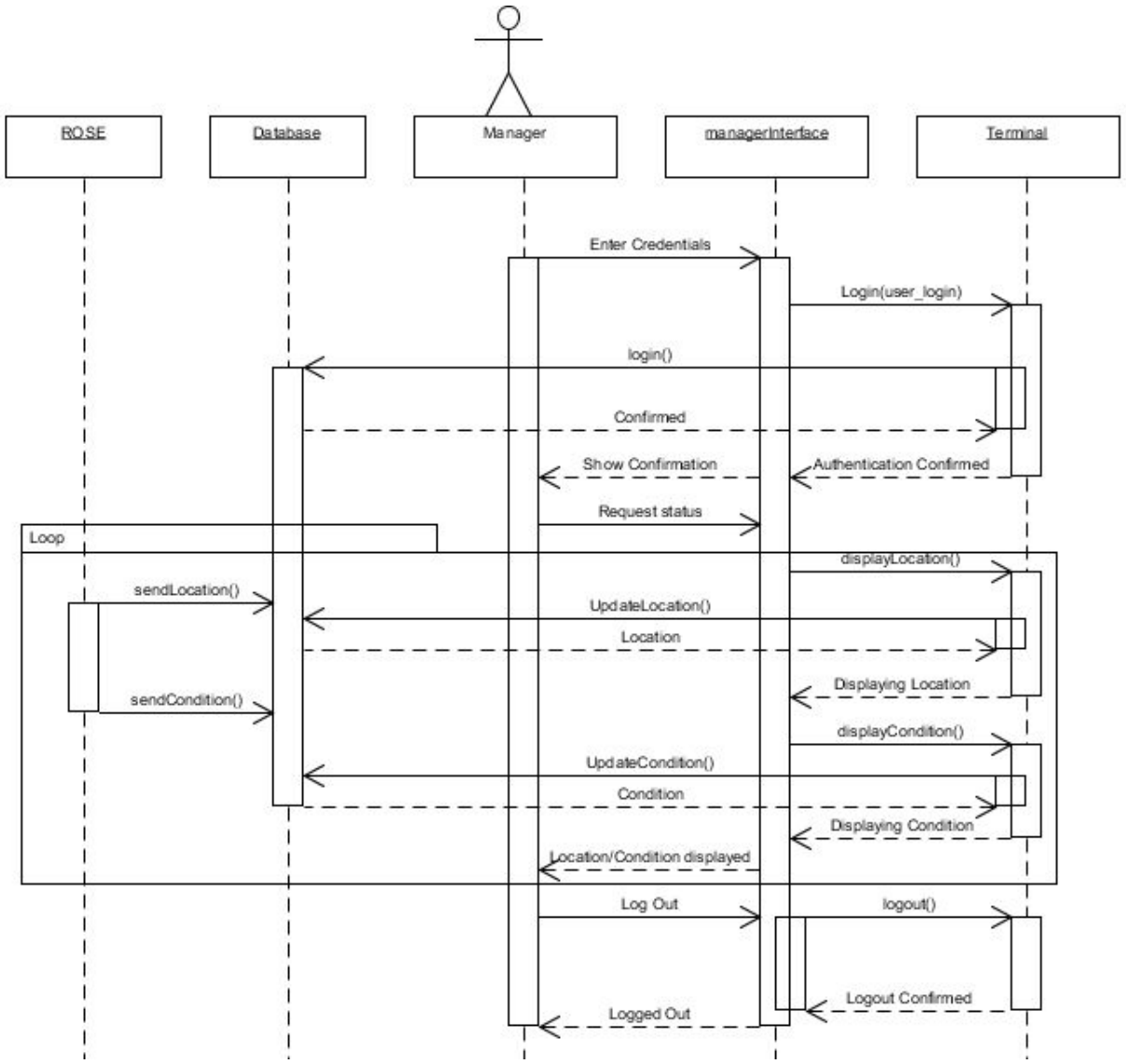


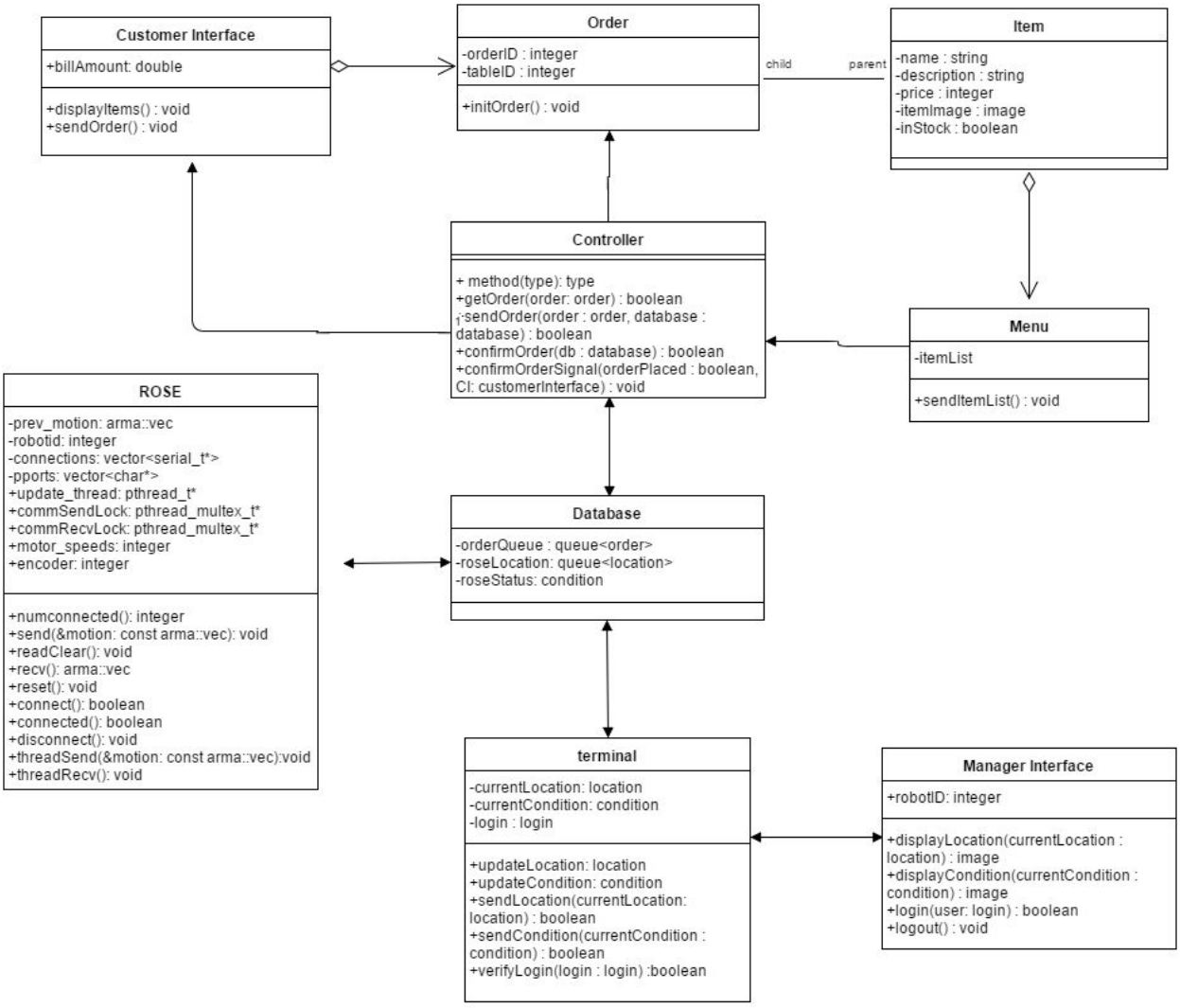
Figure 3: This diagram illustrates the interaction between objects in Use Case 3. Given this drawing, we see that the manager logs into the managerInterface to check the status of the robot. The manager must first log into the terminal, which authenticates the session using the database. He or she then issues the request to check the status, which the terminal grants by requesting the status of ROSE from the database. ROSE continually updates the database of its location and condition. The terminal's calls to the database returns ROSE's status for the manager to review. Afterwards, the manager logs out, and the terminal confirms the logout.

Design Principles

Our system in this use case is designed so that ROSE, the controller, and the terminal are independent systems that interact among the five objects depicted in the interaction diagram. In terms of the principles, we use interface segregation, where some objects do not depend on every class in the system. We also use single responsibility principle such as the managerInterface, where its main role is to allow the manager to interact with the terminal, which then accesses the database to give the manager a real-time update of the robot's location and condition.

Section 2: Class Diagram and Interface Specification

a) Class Diagram



b) Data Types and Operation Signatures

Customer Interface

Attributes

+billAmount: double	Variable which corresponds to the customer's bill total.

Methods

+displayItems(): void	This method display the menu items for the customer to view and select.
+sendOrder(): void	This method will send the order to the Controller

Manager Interface

Attributes

+robotID: Int	Variable which corresponds to the ID of the robot

Methods

+displayLocation(): image	This method will be used to display an image of the location of the robot.
+displayCondition(): image	This method will be used to display the condition of each part of the robot.
+login(): void	This method will be used by the manager to request access to the terminal.
+logout(): void	This method will be used by the manager to exit the access to the terminal.

Menu

Attributes

-itemList: list<item>	A list will be used to store all of the items which are available on the restaurant menu.

Methods

+sendItemList(itemList: list<item>): void	This method will be used to send the list of items from the menu to the Controller.

Item

Attributes

-name: String	Variable used to store the name of the menu item.
-description: String	Variable used to store the description of the menu item.
-price: double	Variable used to store the price of the menu item.
-itemImage: image	Image used to display the appearance of the menu item.
-inStock: boolean	Boolean used to check if the item is in stock.

Order

Attributes

“Item” class inheritance	Inherits all attributes from the “Item” class
-orderID: Int	An integer will be used to keep track of the order.

-tableID: Int	Each table will be assigned a number so the robot knows where to deliver the order.

Methods

+initOrder() : void	Allows an order to be created at the time of the customer's request with the initial values of the customer's desired item

Controller

Methods

+renderMenuSignal(): void	This method will signal that the customer interface is ready to render the menu.
+getOrder(order : order): boolean	This method will return true if the order is received from the customer interface.
+sendOrder(order: order, database: database): boolean	This method will send the order to the database.
+confirmOrder(db : database) : boolean	Will return true if the order has been placed successfully, false otherwise
+confirmOrderSignal(orderPlaced: boolean, CI : customerInterface): void	This method will command the customer interface to display to the customer if their order was placed successfully

Database

Attributes

-orderQueue: queue<order>	The orders sent to the database will be stored in a queue and delivered in a FIFO basis.
-roseLocation: queue<location>	table used to store the locations of the robot.

-roseStatus: condition	table used to store the most recent condition of the robot.

Terminal

Attributes

-currentLocation: location	Variable used to store the current location of the robot.
-currentCondition: condition	Variable used to store the current condition of the robot.
-login: login	Variable used to store the user login credentials.

Methods

+updateLocation(): location	This method will get the location from the database.
+updateCondition(): condition	This method will get the condition from the database.
+sendLocation(): boolean	This method will send the location of the robot to the manager interface and return true if successful.
+sendCondition(): boolean	This method will send the condition of the robot to the manager interface and return true if successful.
+verifyLogin(): boolean	This method will use the login credentials passed to it by the manager interface and allow the manager to access the terminal if the credentials are correct.

ROSE

Attributes

-prev_motion: arma::vec	Vector used to record previous motion carried out by the robot.
-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.
-pports: vector<char*>	Vector used to contain the ports.
+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.
+motor_speeds: integer	Variable used to set the speed of the motor.
+encoder: integer	Variable used to convert information to another format.

Methods

+numconnected(): integer	This method gives the number of connections.
+send(&motion: const arma::vec): void	This method locks the data before setting it and then sends the motion command.
+readClear(): void	This method goes through every device and reads all of the contents.

+recv(): arma::vec	This method adds a lock to wait until the commthread is done setting the vector.
+reset(): void	This method is used to set the previous motion to zero.
+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.
+threadSend(&motion: const arma::vec):void	This method sends serial data in its own thread.
+threadRecv(): void	This method receives serial data in its own thread.

(c) Traceability Matrix

	Customer Interface	Order	Item	Controller	Menu	ROSE	Database	Terminal	Manager Interface
Controller				**X**					
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			X						
OrderDeliverer						X			
StockChecker			X						
Authenticator								X	
User							X		
UserList							X		
DataDisplayer									X
ActionQueue									X
DataReceiver								X	

Note: 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.

CustomerInterface Class

This class evolved from several domain concepts: TouchInterface, Bill, 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.

Order Class

Derived from the Order and OrderAdder domain concepts, this class is responsible for assigning unique order and table IDs for correct delivery. It will also add the price of the items to the customer's bill.

Item Class

Both the ItemChecker and StockChecker domain concepts are part of the Item class, which will be used to identify items and check to see if those items are in stock.

Controller Class

The Controller class was derived from the following domain concepts: Controller, ItemAdder, OrderPusher, and DatabaseConnection. This is the class that exchanges data with as well as commands several other classes to ensure that customer orders are being properly stored and that the database is reachable for storing those orders.

Menu Class

This class was derived from the FoodList domain concept. It contains all items that the restaurant currently offers and will send this information to the controller to be sent to the Customer Interface.

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.

Database 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.

Terminal Class

The terminal class is derived from the domain concepts: StatusChecker, Authenticator, and DataReceiver. These concepts will be used by the terminal to find the location and condition of the robot from the database. It will also attempt to verify the credentials with the database user list coming from the managerInterface. This class is seen as the back-end processor that will send and receive information to the database on behalf of the Manager Interface.

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.

Section 3: 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

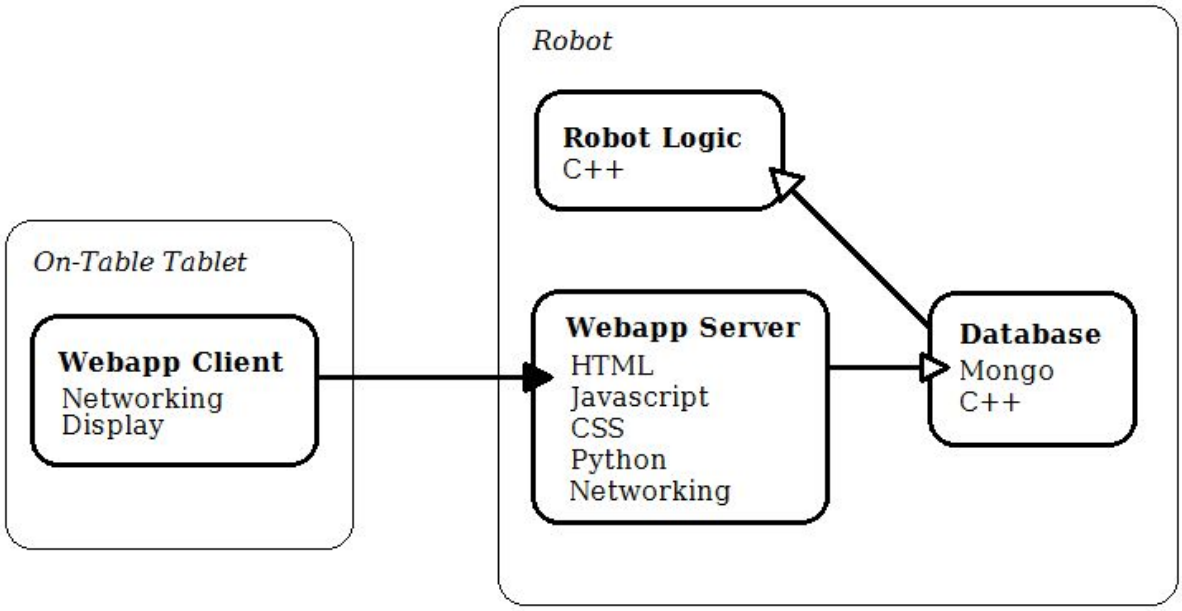


Figure 3.1: UML Package Diagram

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 two collections:

1. The collection that contains placed orders
2. The collection holding the pending actions for the robot to complete (for instance, move forward, backward, left, right a certain distance).

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: Ordered Items

Attributes	Descriptions
Item	The name of the item that was ordered.
Quantity	The quantity of this item that was ordered.
Table Number	The table number where this order should be delivered.

Collection 2: Pending Actions

Attributes	Descriptions
Action	The name of the action to be completed.
Action Parameters	A list of parameters for this action (e.g. for move forward, a single parameter for the distance to move forward).

Note: Collection 2 will be used for debugging purposes only; in the future, we will implement a controller that provides these instructions to the robot internally.

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 will be using either 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). 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 a 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 Arduino's, we have a sender thread, a listener thread and a communications manager thread. For web application communications we have another thread and one more thread for mongo. There is communication between all the communication threads. 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 depends on the following resources:

- ➔ Raspberry Pi / Laptop (To run Mongo)
- ➔ Vex Parts (to build robot frame)
- ➔ 8-12 Vex Motors
- ➔ 8-12 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
- ➔ 2 Cameras (PS3-eye)
- ➔ Tablet/ Smartphone (To run restuarant application)
- ➔ Circuit Board and Wires (For encoders...)
- ➔ 2-3 Arduinos (For motors...)
- ➔ Router
- ➔ USB Hubs

Section 4: 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).

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

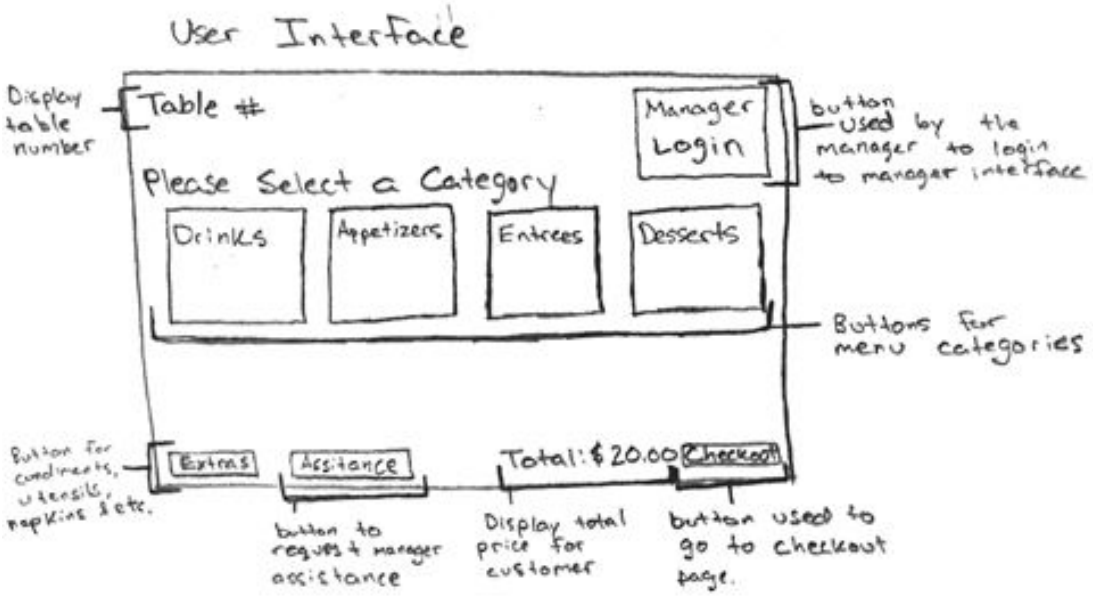
This collection will gather information about the kinematics of the robot’s base. This collection will have 8 documents in total, which corresponds to the displacement and velocity of each of the 4 wheels on the robot.

3) Positioning of the robot's arm

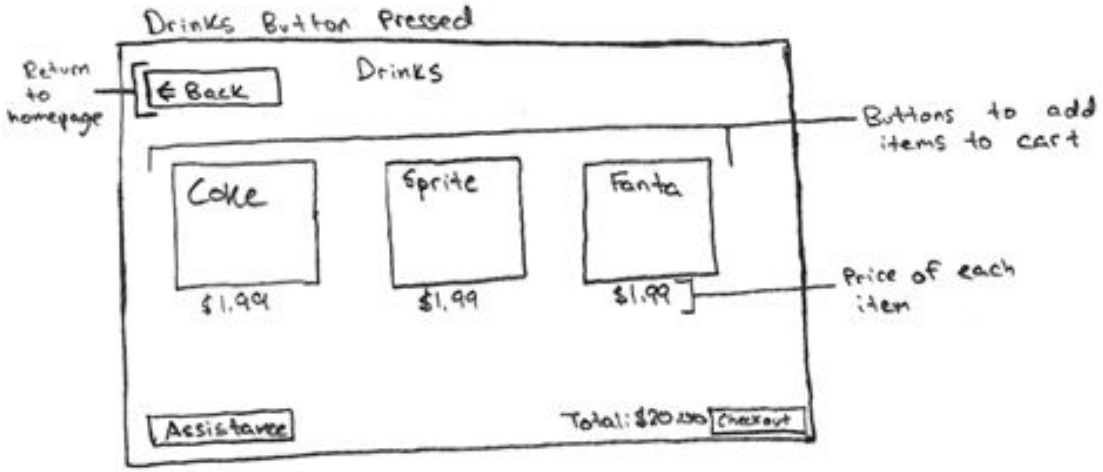
This collection will contain information about the positioning of the robots arm. Will have 3 documents in total which corresponds to the x, y, and z components of the arm.

Section 5: User Interface Design and Implementation

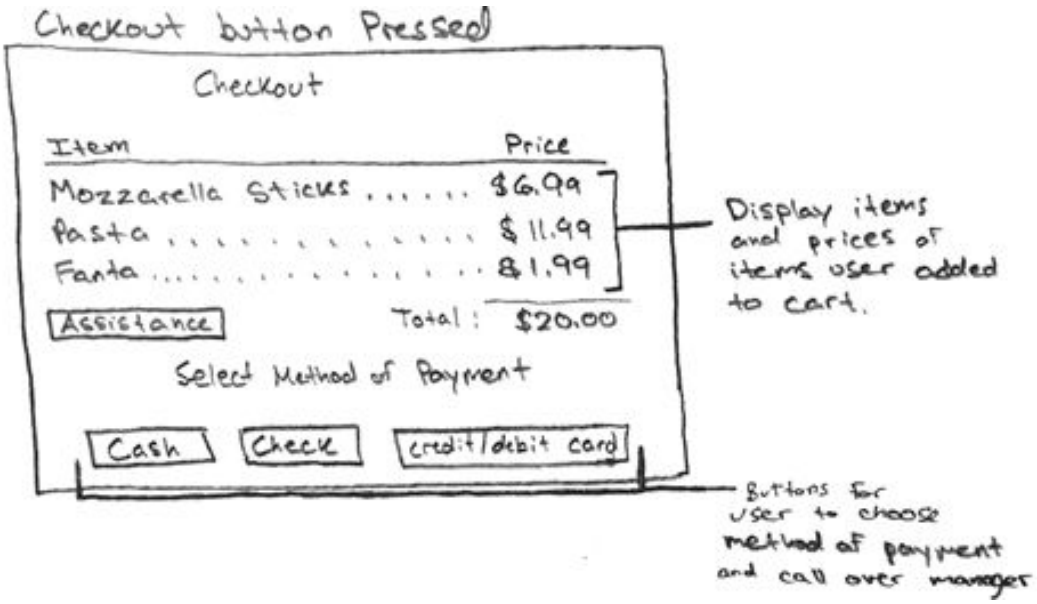
From our last user interface design, we have made numerous changes that are enumerated in the diagrams below (with comments in the margins indicating specific changes).



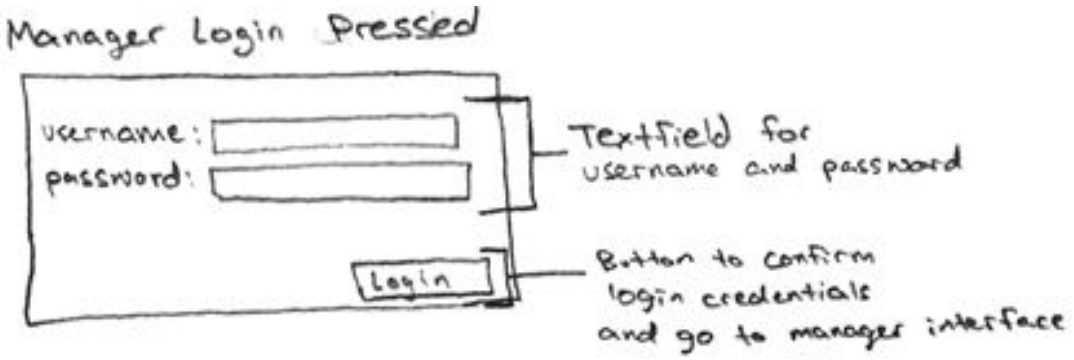
UI Design Figure 1



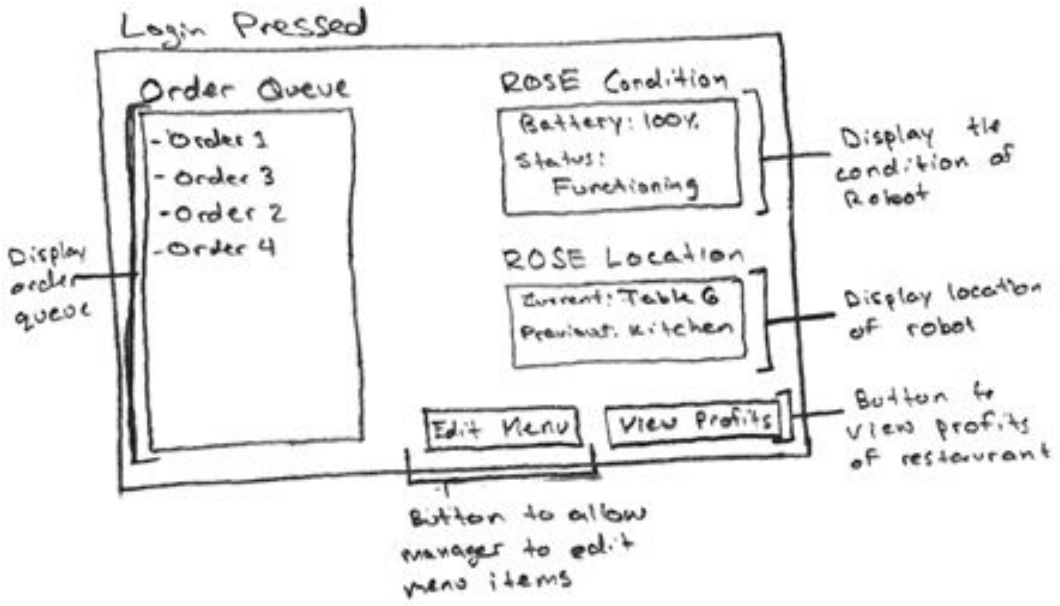
UI Design Figure 2



UI Design Figure 3



UI Design Figure 4



UI Design Figure 5

While we have not yet completely implemented the user interface, we have identified how we plan to do so:

We will use Python with the Flask microframework, rendering dynamic webpages using HTML/CSS/Javascript templates upon certain user actions and pushing customer orders to the locally- (i.e. robot-) hosted MongoDB database when any customer does so. This application will also use Ajax to avoid page reloading upon user actions, to provide a cleaner user experience.

Section 6: 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.
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.
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

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 7: Project Management & Plan of Work

a) Contributions of Individual Members

Merging the Contributions from Individual Team Members. All members worked on their parts of the report due for the week, which were divided up during our weekly meetings. We used Google Docs to collectively contribute to the report, and then one member finalized the format and updated the table of contents preceding submission by the deadline. One problem we ran into was lack of communication through social media mediums. We have solved this by creating an excel sheet consisting of other contact information such as phone numbers and emails. This ensures that should anyone be unable to respond through that medium, we could reach out to them through other mediums.

b) Project coordination and progress report

We've been able to allow the customer (user) to send orders to the database requesting a beverage. We have also successfully implemented the ability of the robot to accept orders from the database based on user input, and send back a confirmation to the database and the user that the order has been placed. We also built the base of the robot, which is currently operational but not autonomous. We implemented chilli tags to identify objects.

We are currently working on the robot's movement and its coordination. Concurrently, our group is working on the robot's image recognition for the drink so that it can grab the correct drink for the user. Alongside this, we are simultaneously programming and calibrating the arm to grab the item.

In terms of organization, we designated a consistent group meeting time once a week to discuss what needs to be done for the upcoming deadlines. Our discussions included splitting up work for documentation and finding out what hardware or software we need to work on for the demo. In addition, we had split our groups into sub-groups to work on specific aspects of the project, which included the database, the web application, and the robot.

c) Plan of Work

We are going to upgrade the UI to control the robot remotely (for testing purposes only). We will also implement PID controller so that the robot can travel in a straight line. We will also have the robot pass data to the database regarding its displacement and velocity. Then, we will program the Arduinos to control the motors of the arm. We also will want to map out the EIT lab, which we will utilize as the restaurant for the demo. We will attach the arm to the robot before the demo date.

d) Breakdown of Responsibilities

Coordinating the Integration

Ajay is coordinating the integration and testing of the robot hardware and software. Other team members will aid him in both aspects, but will contribute to separate portions of the final project needed for the first demo.

Database (Cedric and Jon)

- Implemented the server code for mongodb with python so the webapp can transfer data to the server remotely
- Will Implement mongodb with C++ so ROSE can communicate with the same database that the webapp communicates with.

Web-app (Vineet, Neil, Brice)

- Began implementing the web application by designing a simple application that will display two buttons and will add corresponding entries to the database upon pressing them. This test application can be hosted on either the local machine, or on the cloud via MongoLab.
- We will finish implementing the web application for the first demo by adding more buttons for ROSE's various functions, so that each button press on the application can induce a corresponding action on ROSE.
- After completing the first demo, we will move from testing the individual functions of ROSE to creating the actual customer-oriented application, which will internally call these functions for the robot (i.e. automating these functions).

Robot (All team members, with Ajay and Srihari as lead)

- We have tested the latency of reading the database on a remote server with ROSE being the client and have determined that there is too much latency for ROSE to be the client. Therefore we have decided to store the database locally on ROSE itself
- Will implement code to allow ROSE to send information about its velocity and displacement to the database
- Will program the arduinos in charge of controlling the position of the arm
- Once the robot's base has been complete, we shall begin testing movement of ROSE and its ability to accept arbitrary commands using the web app.
- Finally, once the mechanics of ROSE has been tested and confirmed to be working, we will begin to implement autonomy for item delivery

Plan of work chart up to first demo

Name	Begin date	End date
Python implementation for database	2/22/16	2/23/16
inf passing from db to server	2/24/16	2/26/16
design simple web interface	2/22/16	2/26/16
test latency of remote server	2/29/16	3/2/16
hardware design of base	2/22/16	2/26/16
create base of arm	3/3/16	3/8/16
Finish hardware design of arm	3/7/16	3/11/16
robot to server communication	3/14/16	3/15/16
robot autonomy of base	3/14/16	3/18/16
robot autonomy of arm	3/14/16	3/29/16
preparation for first demo	3/25/16	3/31/16

Section 8: References

http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter5.pdf