EfficienSeat

Dennis Donoghue, CSE, Matthew Donnelly, EE, Kristina Georgadarellis, EE, and Aarsh Jain, EE

***Abstract*— The EfficienSeat system is proposed as a solution to address problems and inefficiencies presented by overcrowded dining halls. Through real time monitoring of individual seats via sleek, LED based, modular units on every table, users walking through the dining hall can easily ascertain seating occupancy at a glance and claim their seats with ease. Further, these units will ultimately communicate this information to users outside the dining hall via a phone application in the form of a real time map, giving a simple visual way to determine dining hall occupancy and viable seating locations.**

I. Introduction

Dining halls on our campus suffer from problems in seating that limit their overall efficiency. During especially busy times, incoming patrons have no way of knowing if there will be seats available for them as they commit to entering a dining hall. This leads to masses of people traveling up and down walkways searching for a potential open seat within a sea of people. The more people to a party, the more challenging and frustrating it gets to find seating. Patrons are then faced with the equally unfavorable choices of using their limited time walking throughout the entirety of the dining hall an unknown amount of times until they find a seat, or leaving and wasting the meal swipe or money that they committed to enter the hall. Not only does this waste the patron's time, but also that of the dining hall workers, who are trying to use these same walkways to keep the dining hall operating smoothly by refilling food or dishware, for example. These time delays, though potentially minimal in single instances, add up and are extremely detrimental in a food service setting.

We sought to design a system that would decrease these frustration and time delays, and explored several methods of implementation. One of the big factors we had to decide on was whether to opt for an active or passive system. An active system being one that completely controls where patrons will sit and a passive system that reacts to where patrons choose to sit. A completely active system would be one akin to the restaurant reservation system, one where patrons are told where to sit. This system would be good for maximum seating efficiency, but is not suitable for a dining hall environment. Patrons would be averse to the idea of being told where to sit, especially when the dining hall is relatively empty. Also, the sheer number of seats to manage in a dining hall also makes this approach inefficient.

On the opposite end of the spectrum, we contemplated doing an entirely passive system, one that would indicate seat status based on the seats people chose to sit at. This system would sense when someone was sitting at a particular seat and then relay this information to patrons inside and outside the dining hall. This is favorable because patrons will not have to do anything new in this system, but it would be expensive and complex to implement. For the solution that we came up with, we went with a hybrid between these two approaches.

With the hybrid system in mind, we then considered the specific needs of our solution. The solution needs to satisfactorily address both patrons and dining hall staff in regards to efficiency and time delay, and one that does so via an implementation that is easy to maintain and non-invasive to the normal operation of the dining hall. More specifically, the system needs to be able to accurately indicate the seating status within the dining hall to incoming patrons, allowing them to see seat availability and identify where they are able to and would like to sit. To be effective, the system needs to be able to monitor several hundred seats, as would be present within the dining hall area. At the same time, the system must not be invasive; neither patrons nor the staff should have to work around the system, or be interrupted by it during normal operation. More specific to the dining hall staff the system must also be easy to maintain, requiring easy installation and infrequent maintenance. Ideally, the only maintenance that should be required would be battery replacement once a month for individual monitoring units. Given that this system is to operate in an environment around food, there are also standards to consider regarding both food safety and the safety of the device. In regards to safety of the device, it should be IPX4 compliant, which indicates that it can withstand splashing and spills.

To satisfy these needs, the system will be composed of a number of nodes situated about the dining hall on the tables to directly monitor the seats. These nodes will need to be minimal in size and self-sufficient in terms of power; this coincides with the low maintenance requirement, where routine "refilling" of power should be reasonably infrequent, to the tune of monthly changes. Every node will be connected to a central

TABLE I
Specifications

Specification	Value
Amount seat supported	>=100 seats
Phone App Response Time	<2 seconds
Table Unit Area	<16in^2
IPX4	Compliant
Battery Life	~1 month

D. Donoghue from Dover, Ma (e-mail: ddonoghue@umass.edu).
M. Donnelly from Springfield, Ma (e-mail: matthewdonne@umass.edu).
K Georgadarellis from Dartmouth, Ma (e-mail: kgeorgadarel@umass.edu).
A. Jain from New Delhi, India (e-mail: aarshjain@umass.edu).

hub, creating a network that will reliably monitor the seating situation and ultimately report this to the user via a phone application. Table 1 quantifies these specifications.

A. Overview

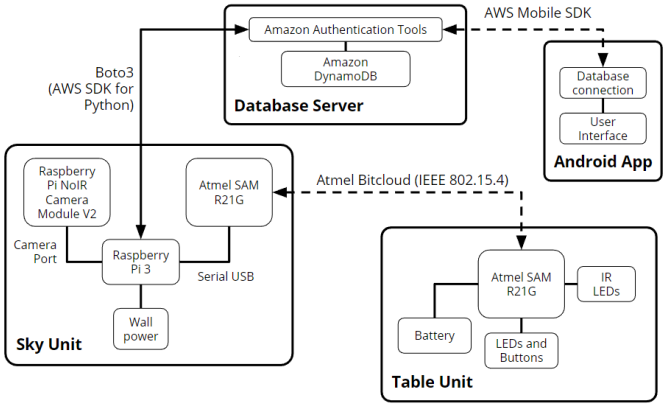

Fig 1. Block diagram of our system. There are four main blocks that make up our system: the Table Unit, Sky Unit, Database Server, and Android App.

To solve this problem, we will design a network to be installed within a dining hall that will allow for real time monitoring of seat occupancy and consistent reporting of this data to users, both inside and outside of the dining hall. As shown in the block diagram in Figure 1, this network will consist of 4 major components: 1) the aforementioned nodes - dubbed "Table Units," 2) a "Sky Unit," 3) a database, and 4) a phone application. Each block's purpose will be outlined in this design section, with more detailed descriptions being provided in the following sections respective to each block.

Patrons of the dining hall, who will be the users of this system, will interact with it from two points: the phone application and the table units. Table units will populate the dining hall as compact units each containing 4 buttons and a number of multi-purpose LEDs, with each unit monitoring 4 seats. Single tables will only require 1 table unit while longer ones will utilize multiple units. These units provide the first point of user interaction by allowing users to claim seats by pressing a button, and seat status will be indicated via the LEDs, providing a simple method of identifying occupancy at a glance for users within the dining hall. By having an easy to interpret system to claim seats the issue of patrons marking their spots with valuable items like phones or keys is eliminated and individual seats will be clearly marked as per their associated LED, instead of an ambiguously placed coat, bag, or phone on the table. Seat occupancy information will be reported from each table unit to the Sky Unit, which will be the central hub of the network. As the central hub, it will process the table unit data and function as the medium of communication between the table units and the database/app side of the system. Alongside this, the Sky Unit will also perform table localization, the information of which will be used to accurately report seating data to users via the app.

From the Sky Unit, seating information will be sent to the database, for which we have chosen to use Amazon DynamoDB. The database will be constantly updated in real time by both the Sky Unit and the phone app so that it will contain up to date information of all seats within the dining hall.

The phone application provides the other end of user interaction with our system. Namely, users will be able to not only observe the seating situation within the dining hall from their phone, but also search for available seats as per the number of people within their party and reserve these seats from the app. Using information queried from the database and the aforementioned table localization data the app will generate a map of the dining hall that indicates the occupancy of seats for the user. Communication will be maintained throughout the entire network so that, for instance, a seat reservation indicated from the phone app will be received by the table unit relatively instantaneously. With a basic outline of the functionality of the system, each block will now be discussed in greater detail in terms of protocol, components, and the like. Although many alternative designs were considered, these were more specific to the technologies used within each block; the overall topology of the system has remained largely the same its inception. These alternatives will be discussed in detail in the following sections.

B. Table Unit

The table unit is the interactive device for the users at the table itself and is a small modular unit containing 4 visual LEDs and 4 buttons, corresponding to 4 seats that the unit is monitoring. It will also contain 4 IR LEDs, which will be discussed later in regards to table localization. The visual LEDs will be used to indicate the status of the seats to the user, namely whether they are taken or not. The buttons will be used to confirm reservations of seats made from the app, and to occupy/vacate seats without the app. The final product will be splash-proof (IPX4 compliance mentioned in the introduction) and easy to maintain with low power consumption. The following table shows the seat state corresponding to the LED responses:

TABLE II
TABLE UNIT CHAIR STATES

Chair State	LED Response
Vacant	OFF
Occupied	ON
Reserved	Slow Pulse
About to Expire	Slow Pulse

The above explanation gives us the requirements for the processor selection.

- The processor needed to be able to communicate through a network with the ability to handle multiple request at any time.
- It should be low powered to keep the maintenance of the unit to minimal.
- It should be able to take input from four buttons and output to four leds.

SAMR21 by Microchip Technology inc. seemed to be the right choice. The microcontroller (SAMR21) has a System on Chip for IEEE 802.15.4 communication protocol. This is a low powered and low bandwidth network protocol generally

used to communicate to low powered remote devices. The stack provided by Microchip, Bitcloud, is well supported for this microcontroller. Some of its features are:

- Network management of more than sixty-five thousand nodes.
- Automatic queuing of incoming packets.
- Confirmation packet for received data.

The project started with a few example codes in Atmel Studio Framework to better understand the overall code that would govern operation of the Table Unit. Simple network communication was established between two SAMR21 boards via their built in RF antennas. The range of communication was established to be beyond 50 feet through testing with a wall barrier which is more than enough to communicate to any corner of a dining common from its center. Further testing was done including button debouncing optimization to give a smoother interface for the user. The Table Unit will be tested using a default set of commands including the following responses:

- Button press toggles state vacant and occupied.
- Message received from Sky Unit shall be processed and signified on leds.
- Reserved state shall go to occupied when button is pressed.
- Timer state shall give a pulse on the corresponding LED.
- Data shall be sent when a button is pressed at the Table Unit which should be properly received at the coordinator in Sky Unit.

C. Sky Unit

The Sky Unit will be composed of the same Atmel SAMR21G processor used in the Table Unit as well as a Raspberry Pi 3 board. For table localization, a Raspberry Pi NoIR camera module will be used. These three components will all operate from wall power. The Sky Unit functions as a central hub for all of the Table Units, and an intermediate link between the Table Units and the database/app. The SAMR21G board will be used to communicate wirelessly with the Table Units via the Atmel Bitcloud stack, as explained in the Table Unit section. While the Sky Unit is the central hub of the system from a block diagram point of view, the Raspberry Pi is the true center; the Pi was chosen for its power and versatility as a computer coupled with easy to use functionality and an abundance of documentation available online. For communicating data between the Table Units and the database the Pi will receive and process data via code written in Python using Boto 3, which is Amazon's own SDK for Python. As we chose to utilize Amazon Web Services (AWS) for our database server, which will be addressed in more detail in the following section, Boto 3 was a reliable choice for integration as it is very well documented by AWS and provides a method of easy to use, low level access to the database from the Pi. Given that Python code is easily written and used on the Raspberry Pi, this is the perfect option that provides a simple, reliable, and efficient way of both writing to and reading from the database with the Pi. By utilizing the Atmel Bitcloud stack and the Boto 3 SDK, we effectively have successful communication between the Table Units and the database.

In terms of interfacing the Pi with the database, alternatives were explored as per the database chosen; Boto 3 was our final choice as we settled on Amazon DynamoDB, with which Boto 3 is closely coupled, but a previous option was that of Amazon RDS - the choice to switch databases will be discussed in the following section. In terms of interfacing the Pi with RDS, simple interaction was available through the use of MySQL at the cost of security due to the nature of the connection. In order to successfully interact with the database, a lot of the security authentications needed to be removed. With DynamoDB, initially connecting to the database involves a bit more work, but the overall process of interacting is simpler with respect to the Pi, and more secure in terms of the connection.

In order to communicate seating data efficiently throughout the system, we needed to design a protocol for the information. To do this, we identified the pertinent information that would be present and required for successful data transmission in terms of accurately monitoring and altering seating states and maintaining system status, which boiled down to Table ID (with what Table Unit are we communicating?), command type (are we changing seat occupancy? telling the Table Unit to sleep?), and data (seat is now occupied, Table Unit action). Our resultant protocol consists of 5-byte strings, as displayed in the table below (Table III).

TABLE III
PROTOCOL

ADDRESS	COMMAND	DATA	PIN
2 Bytes	1 Byte	1 Byte	1 Byte

The first 2 bytes of the string correspond to the Table Unit address, which will be acquired during calibration. The next 3 bytes each correspond to the command type, the data, and the pin (seat), respectively. An example command for a seat reservation would be as follows: "01 1 2 2," where 01 correspond to the address of the Table Unit, and 1 corresponds to a "change seat state" command. Given that the seat state is changing, the fourth byte will correspond to vacant, occupied, or reserved (2 for reserved in this case). Finally, we need to know which seat of the four corresponding to this Table Unit are being affected - here we are changing Seat 2 to reserved. Figure 1A in the Appendix displays all of the possible values for each byte and their corresponding meanings.

To test this block in terms of the Sky Unit's ability to process and transmit data to and from the server, we communicated example data as per the previously described protocol to alter data within the database. By writing a Python program utilizing Boto 3 to establish a connection with the database and both request and modify specific table information within the database, we could exercise a reliable method of wirelessly interacting with the database from the Sky Unit, and more specifically the Pi on the Sky Unit. This functionality will further be tested by automating the process of sending and receiving data to and from the database and having it occur automatically in response to changes elsewhere in the system, such as when a seat's state is changed on the Table Unit itself. The protocol itself was also tested via communication between

the Table Unit and the Sky Unit, where data sent in this protocol from the Sky Unit successfully resulted in the desired change on the Table Unit.

Table Localization: Another important aspect of our system is determining the position and orientation of each Table Unit in the dining hall. This data is needed for an accurate mapping of the tables and seats on the mobile app. Without it, our system could tell the app user that a particular table and seat is open but not where the table is or which of the four seats are available. We could have opted for a static implementation of the Table Unit locations with each unit being mapped to a template of the room before the system was in service, but we wanted our system to have adaptability in table positions in case tables were moved, like if patrons pushed two tables together to seat eight people, for example. We felt as if this was more reflective of a practical implementation of our system. Now let's talk about the methods we considered to implement the table localization functionality.

We first thought about what our system needed in order to have reliable, effective, and dynamic table localization.-In terms of how accurately tables were located and presented the user, we need a realistic level of accuracy; tables do not need to be identified down to the exact inch, but rather within at least half of a table's length. As such, inevitable but minimal error would be tolerated and tables would still be reported correctly enough that users would not fail to find their table. To dynamically respond to changes in the table layout, we decided that periodic updates every hour would be sufficient, since major table movement is not frequent and we do not want to waste power or processing time unnecessarily checking for changes. The implementation also had to be non-intrusive to the operations of the Table Unit, as in, not consume too much power or be distracting to patrons. Keeping these specifications in mind, we first looked at creating an indoor positioning system, however, that idea was quickly abandoned. Since the dining hall is a relatively small area, we would have to be able to detect differences between signals on the order of nanoseconds. We also thought to use ultrasound, as is a common method for indoor localization, but this technology proved to be too complex and robust for our needs.

We consulted Professor Goeckel and Professor Kelly about different ways to implement table localization functionality. Professor Goeckel suggested we try to use the variation in wifi access point strength to map the room, where individual locations would have unique but relatively consistent wifi strengths based on distance, but warned that this is not always the case, even over larger distances than would be used in the dining hall. We did some tests in the Worcester dining hall by downloading software that measures the wifi signal strength at particular locations. Unfortunately, the signal strength was not consistent and it did not provide enough resolution to be able to distinguish between one location and another location one meter away. Professor Kelly suggested that we place a unique pattern on the ceiling and equip the Table Units with small cameras that can take a snapshot of the ceiling. Depending on what piece of the pattern was shown in the snapshot, we could determine the location of the Table Unit. We considered this option but ultimately did not pursue it due to the extra cost the camera would bring to the Table Units.

The method that we are going to implement is by using an IR camera on the Sky Unit and four IR LEDs on board the Table Unit. The IR camera will not pick up the visual spectrum, allowing the IR LEDs to be easily seen when turned on. The position of each Table Unit will be found using the following steps: First, before the system is in service, the Sky Unit will be calibrated to the room. This is done by placing the Sky Unit where it will be mounted, in a corner of the room, and placing an IR LED at every other corner. The dimensions of the room will be given. The IR camera will take a picture of the room and then map the IR LEDs to the corners of the room. This will create a rectangular space. This space will then be divided into a grid so that the table locations can be pinpointed within it. To find the location of each Table Unit, the Sky Unit will send a command to a particular Table Unit to turn on all four of its IR LEDs. The IR camera will take a picture of the dining hall and compare this with the reference grid, finding the position of the table. The Table Unit ID and position will then be sent to the database and finally the mobile app for rendering.

The Table Unit orientation is needed to determine which button and LED corresponds to which seat, since the Table Unit is not fixed to the table (for cleaning purposes). The Sky Unit will determine orientation by sending a command to a particular Table Unit to turn on one IR LED corresponding to one of the seats. The IR camera will then take a picture of the dining hall. From the light intensity, the Sky Unit can determine the orientation of the unit by using a reference guide to match light intensity to orientation.

D. Database/Server

Our system will feature a set of databases, each of which will store table data corresponding to a specific dining hall. These databases will be populated with information obtained from the Sky Unit regarding each Table Unit, as explained in the previous sections, corresponding to table layout and orientation, and seating occupancy status. Our app will fetch data from the database corresponding to the user-selected dining hall and use it to populate the app's dining hall map and respond to user searches. To this end, we decided to use Amazon's DynamoDB. DynamoDB is fast, flexible, and highly scalable noSQL database hosted by Amazon Web Services. DynamoDB is fully integrated with the full suite of AWS services, including identity management and security. The techniques needed to produce this part of are project were acquired both in basic coding classes and through work experience. A test of this part of our system would likely start with the basic CRUD functionality and then test the integration of the database with our greater system. If we can perform CRUD operations properly and correctly connect the parts of the system together, then the test will have passed.

Before using DynamoDB, Amazon RDS was also explored as a potential option, however it proved difficult to interact with in the way we needed (specifically communication with Raspberry Pi) due to how it handled security, and so the other options were explored, which led us to DynamoDB.

E. Android App

Our app is one of the outward-facing parts of our system. The app will allow the users to browse a seating map of the dining halls and search the available tables by party size. Android Studio IDE and Java were utilized to write and build the app. A testing suite for this app would test the user's ability to select a dining hall, the app's ability to update the table map based on database changes, the app's ability to communicate with the Pi, and the app's ability to search the table data by party size. Proper outcomes from these tests would confirm proper app functionality.

II. PROJECT MANAGEMENT

TABLE IV
MDR DELIVERABLES

Deliverable	Percent Complete
Demonstrate basic Table unit functionality: • Change of state via button • Communication to Sky Unit	90
Demonstrate functional app: • UI – Rendered map and ability to search for seats • Communication to AWS	100
Demonstrate Sky Unit Communication to AWS	100

As per the table above, our MDR goals focused on implementing necessary functionality of our system and integration of its components. Part of this was displayed by a functional Table Unit with 4 LEDs to indicate the states of the 4 seats monitored by that unit. The states could be changed physically on the unit via a button press and wirelessly at a distance via the Sky Unit, which shows integration of sky and Table Units and a step towards transmitting data from one end of the system (phone app) to the other (Table Unit). Integration of the Sky Unit and the server/app is currently in place with the app providing a rudimentary user experience of viewing a rendered, dynamic map of the dining hall tables, entering search queries, and viewing the response. The app also uses data from and responds to changes made in the AWS database, which can be successfully interacted with via the Raspberry Pi on the Sky Unit. These functionalities satisfy nearly all of the deliverables lsited.

The MDR deliverable that was only partially delivered upon was the communication between Sky Unit and Table Unit. In effect, the Table Unit and Sky Unit are successfully communicating via the two SAMR21G processors aboard either unit, but the processor aboard the Sky Unit is not yet successfully connected to the Raspberry Pi. Currently, the SAMR21G processor on the Sky Unit can successfully transmit the necessary data when received serially from a computer, but there is an issue with connecting serially to the Pi that will be solved. As the Pi functions as a central hub of sorts, having a successful connection between the Pi and the processor aboard the Sky Unit results in full connectivity of the system from Table Unit to the phone app.

Alongside this issue, table localization is the next major step to tackle in terms of its successful use, implementation, and integration with the rest of our system. We will first test out the IR camera and IR LED interaction, making sure that the camera can detect the IR LEDs from an appropriate distance (the farthest distance across the dining hall). After that is established, we will work on creating a reference grid for a room, starting out small, and then expanding for the size of the dining hall. After that, we will then test the finding of the table positions. As for orientation, we will test that light intensity is a reliable way to tell orientation and create a reference guide based on these tests. Then we will implement this in a small setting, and then expand to the dining hall setting.

There is also still overall optimization and automation to be done to further enhance the basic functionality that we have into what is necessary for the system to work as a whole. More specifically, full Table Unit functionality needs to be implemented in terms of relevant table localization aspects and complete code for all possible table states and the behavior of the LEDs to indicate said states. The code for the Pi aboard the Sky Unit also needs to be completed such that the processing, receiving, and sending of data is automated. A line of communication needs to be opened between the app clients and the Pi, so that the Pi is the only device making changes in the database, in order to prevent reservation conflicts. A more advanced searching functionality must be implemented for the App, as well as a handful of UI updates.

Since our project is easily broken up into components, we split our team to work on each component relatively individually. Discussion as a group, however, occurs regarding each design decision and any changes or issues that come up. Though everyone has an assigned focus on different aspects of the project, there is great team involvement to help flesh out the most efficient approach and to ensure that everyone is consistently on the same page. We hold weekly meetings with our advisor, Professor Wolf, where we discuss the project's current state, alternatives to current designs and potential issues that might arise, and next steps. As a team, we also meet one to two times per week to discuss the project amongst ourselves, to work on components together, and to ensure proper integration of our separate components. The team maintains email communication with Professor Wolf and constant communication within the team via GroupMe. Project documentation and organization is maintained on Google Drive.

Dennis, our computer systems engineer, handles phone app development and integration with the AWS server. Relevant information in terms of functionality and communication protocol are related to the rest of the team when necessary so that the rest of the system may effectively communicate with the database and app when needed. The Table Unit is being primarily handled by Aarsh, who has great familiarity with Atmel processors, code, and functionality. Matt is mainly working with the Raspberry Pi aboard the Sky Unit in terms of ensuring communication with the database and the

Atmel processor aboard the Sky Unit, and processing of data that is transmitted throughout the system. Kristina is working on table localization and how this interfaces with the Sky Unit. As previously mentioned, each component has substantial input from the team as a whole to ensure that the best method to solving the problem and implementing the solution has been found and agreed upon.

Figure 2 below is our Gantt Chart detailing the start of our project up to its end.

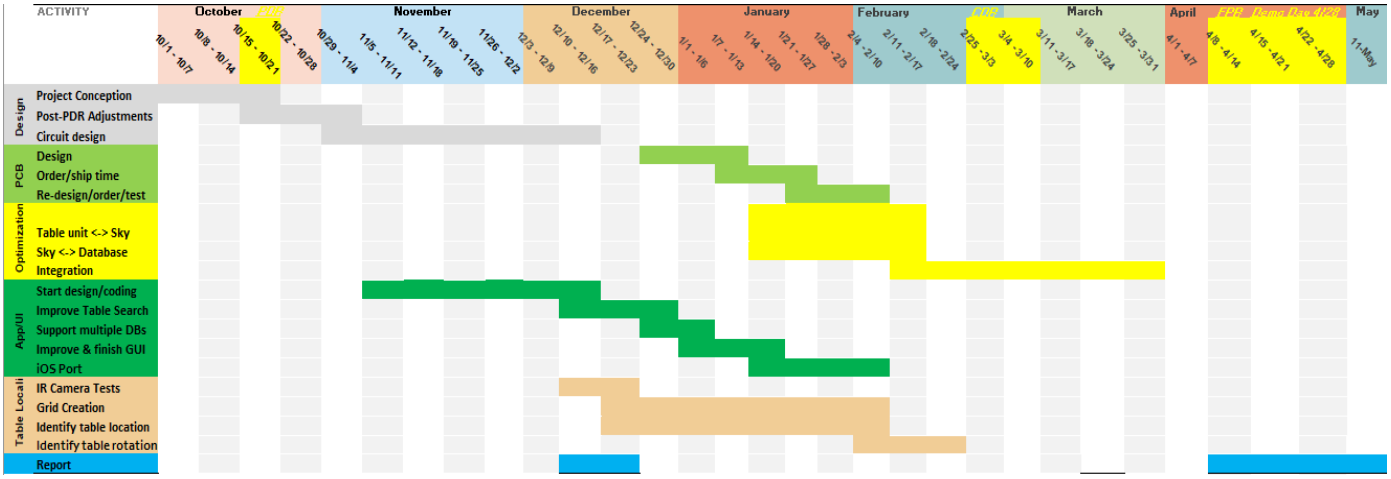

Figure 2. Gantt Chart detailing current progress as well as our plans for the future.

III. Conclusion

By meeting our MDR deliverables, we have established almost complete integration and communication from one end of our system to the other. The fundamental skeleton upon which our project resides has been built and communication protocols have been determined. Piecewise, our system is integrated. We still have yet to link the Raspberry Pi to the SAMR21G aboard the Sky Unit and implement the table localization functionality, the former being trivial and the latter is in the developmental stage. Our project has taken a few twists and turns to arrive at this point. We scaled back our original idea and also sought simpler solutions to the problems we were trying to tackle. We met frequently and held each other accountable for the assigned tasks. Through this, we were able to create the framework for our project and create a clear idea of what our end goal will be. The next step is to build upon this framework and optimize power consumption for the Table Unit. We are aiming for a low power embedded system and will need to utilize the wake and sleep functionalities to our advantage. We will be drafting up the PCB schematic in the coming months so that we can have a running start to next semester. Since we are unfamiliar with the designing of PCBs, this part will prove to be a challenge for us, but starting early on the design will set us up for success. We will start working on the table localization functionality early on in the semester as well, since it is a key component in integrating the entire system. We have a few challenges ahead of us and a lot more mileage to put in before SDP day, but we are well on our way to getting there.

Appendix

ID	Command
0	Invalid
1	Change table state
2	Sleep
3	Wake up
4	Turn IR LED on

ID	Data Entry
0	Vacant
1	Occupied
2	Reserved
`3	About to expire
4	Sleep
5	Turn IR LED on

ID	Pin Name
0	Invalid
1	Seat 1
2	Seat 2
3	Seat 3
4	Seat 4
5	All Seats

Figure 1A. Definitions of the protocol for the Table Unit to Sky Unit Communication.

Acknowledgment

We would like to thank our advisor Professor Tilman Wolf and evaluators Professors Eric Polizzi and Csaba Moritz for their very useful critiques and insightful advice that helped guide our project. We would also like to thank Professors Patrick Kelly and Dennis Goeckel for taking the time to meet with us and discuss potential options for a part of our project. Dennis Donoghue would like to specifically thank the creators and users of the website StackExchange for boundless advice and help.