ServoBot - Delivery Robot with a Nodal Network of Order-Taking Modules

Aryan Agarwal, Kenson Nguyen, Marcell Veszpremi, Utkarsh Kumar
ECE180DA: Team 3 Final Report
University of California, Los Angeles

I. ABSTRACT

The ServoBot is not an ordinary product but an ecosystem that makes the lives of chefs, restaurant owners, and workers easier and stress-free. Our team went a step beyond and started this project off with some research on current technologies in the market. There are robots in the kitchen that prepare the food (ex: Flippy) and there are also robots that help with food delivery (ex: StarShip). Surprisingly, we weren't able to find a company that provided a well-rounded product for this market. That is what led us to take this up as a project.

We believe automation and technology should go way beyond what it currently is in the hospitality industry. Another major problem that we came across was the affordability of these robots. Almost all the robots in the industry use technology like LIDAR which is very expensive. Thus, our solution includes Image Processing (Stereo Vision and April-Tags), Speech Processing (Speech-to-text Google API), and IoT (Communication and Component Integration). Our solution is reliable, low-cost, and effective.

II. INTRODUCTION

After the COVID-19 pandemic, restaurants have suffered a significant shortage of staff, especially waiters and hostesses. Moreover, in recent times, contactless delivery has grown in popularity. This provides a need for automation in the hospitality industry which we felt has not been addressed significantly. Restaurants have started slowly integrating features to appeal to the user experience that ensures both safety and cleanliness. In the restaurant sector, however, we have not seen the application of some of the latest technologies such as IoT, Robotics, Image, and Speech Processing. This is the problem that we set out to solve.

Our team has come up with the perfect solution to this problem with ServoBot. ServoBot is the one-stop solution for a seamless user experience right from when the guest/customer enters the restaurant, to when they leave. ServoBot is an integrated ecosystem with three main components: The order-taking module called "*The OrTak,"* the food/drink delivery robot called *"The WOK-er,"* and the order-handling node *"The KitchenNode."*

The ServoBot leverages a networking system that makes the lives of restaurant owners, guests/customers, and other restaurant staff seamless. pAll three components communicate to ensure orders are taken and delivered hassle-free. It also allows all parties involved to have an end-to-end contactless experience.

ServoBot uses the latest technologies in the market to complete its missions, including IoT (communication through the internet), gesture & speech recognition, and cameras. The IoT aspects will look into the communication between the modules and the robot. Meanwhile, Speech and Gesture Recognition will be used to take food orders and control robotic locomotion features. The cameras will be used for obstacle detection and also for ID detection/checking.

In a broader sense, ServoBot can have ranging applications in other parts of the hospitality industry. With certain enhancements, it can be modified to be implemented in elder care centers and to help those with disabilities. The speech recognition and autonomous components ensure minimal human interaction is needed for the components to run smoothly.

III. BASIC DESIGN

Our idea revolves around three main components – each of them solving a unique problem. The three are defined below in extensive detail:

The WOK-er

The WOK-er will be the food/drink delivery robot. The main processing unit will be a Raspberry Pi Zero for all the features other than the motor control, which will be controlled using an Arduino Nano. The WOK-er will be a four-wheeled,

omnidirectional, camera and microphone-enabled, semi-autonomous robot with pathfinding to each of the tables in the restaurant. It will be able to detect obstacles in the direction of motion and has the mechanism to avoid them because of the cameras and ultrasonic sensors on it. It will have an LED screen on it to display the table number it will deliver to. It will be capable of gesture and speech recognition to take commands and execute them. To identify tables, AprilTags are utilized as unique identifiers that the WOK-er can read with its cameras. Obstacle avoidance is achieved with stereovision. To do this, a dual-camera setup is utilized along with processing from the OpenCV library to accurately identify how far the WOK-er is from objects. Speech recognition is provided with the Python library SpeechRecognition, which leverages the Google Cloud Speech API. When a table indicates that they are finished dining, the WOK-er will go collect the dirty plates and leftovers. Below, you can see the WOK-er with the two cameras pointing forward for April Tag and obstacle detection.

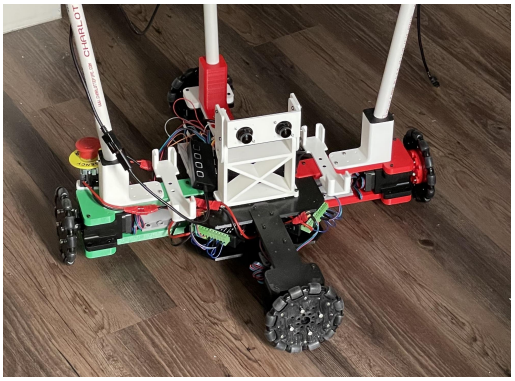

Figure 1: The WOK-er

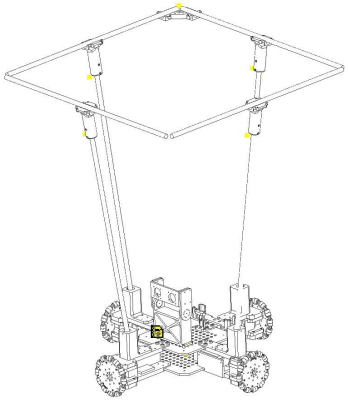

Figure 2: CAD Assembly of the WOK-er

We had to be very careful while choosing the different parts for the WOK-er. We chose the parts based on previous experience of the group members. As the WOK-er is one of the most important components of the ServoBot, we wanted to make it reliable and of good quality. For that reason, we chose the following parts:

The Robot Wheels

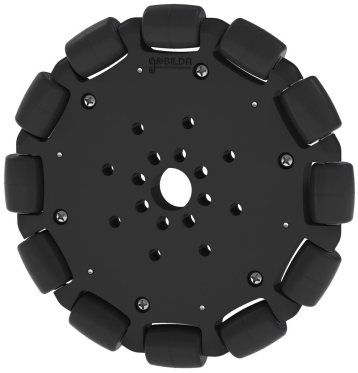

Figure 3: Omnidirectional OD=120mm robotic wheel.

The wheels we chose are the omnidirectional 3604 series wheels from GoBilda. We chose to go with omnidirectional wheels for ease of maneuverability. A common issue with omnidirectional wheels is slippage. Many omnidirectional wheels on the market have plastic rollers, which make them prone to slipping. To prevent this, we purchased wheels with rubber to minimize wheel slippage. Additionally, as our expected load and speed are not extremely high, slippage should not be a huge concern.

The Stepper Motor

Figure 4: Nema 17 Stepper motor with planetary gearbox.

Four Nema 17 stepper motors with gearboxes will be used to achieve high torque and be able to transport large amounts of plates and items. These stepper motors were chosen primarily from their torque ratings and voltage requirements. We need to choose motors that would carry plates and move at a reasonable speed while doing so for an optimal customer experience. These four motors along with the motor driver will allow for extremely precise control of the robot.

The Stepper Motor Driver

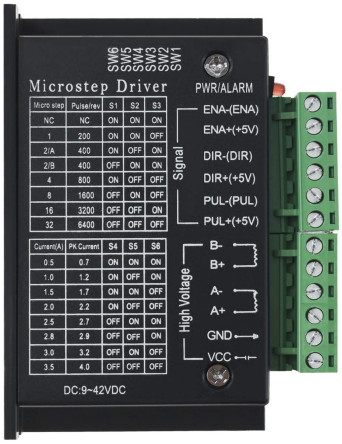

Figure 5: Stepper Motor Driver TB6600

We chose this stepper motor driver as it allows for extremely precise control of our stepper motors. As it is designed for use in CNC machines, the level of precision meets the requirements of this project. Additionally, with the driver being relatively affordable and easily accessible via Amazon, replacing the driver will be easy in case replacements are needed during testing. The requirements for interfacing with the driver are already known as some group members have prior project experience with this specific driver.

The Two Wide Angle 5 Megapixel USB Cameras

We chose to use wide angle cameras so that we can detect the April Tags (used for localization) even when we aren't facing them directly. Moreover, we chose to use the 5 megapixel version so that we have higher resolution and can detect April tags from further away. Furthermore, choosing wide angle cameras allows for a greater overlap region in our Stereo Vision field of view. This in turn, allows for a more robust and reliable detection of obstacles in the robot's path. While we scale down the images for Stereo Vision to accelerate computing, we benefit from being able to average pixels when scaling down, leading to more reliable depth maps.

.

Figure 6: Wide Angle USB Camera

The Microcontroller – Arduino Nano

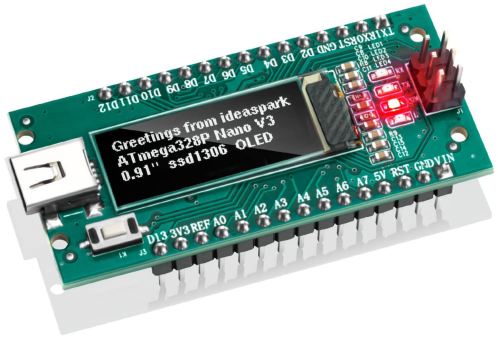

Figure 7: Arduino Nano – ATmega328P CH340 Microcontroller

We will use an Arduino Nano to interface with the motor driver. While the Pi Zero is perfectly capable of sending the switching signal required by the motor driver, we decided that having a dedicated microcontroller was better to avoid the Pi Zero on the robot having to handle both processing and constant switching. Additionally, Arduino Nanos are cheap and accessible, so replacements will be easy to acquire if need be during testing. We chose this Arduino Nano in particular for the built-in OLED display. This will make debugging away from the computer easier, as no laptop USB cable will be required to view debugging messages.

The Battery

Figure 8: Hoovo 7.4V 2S 5200mAH LiPo Battery

Based on our research, the most optimal way to power the system is using two 2S 7.4V LiPo batteries. Each battery has 5200mAh of capacity. Wired in series, we will have a total of 4 cells in parallel and 5200 mAh of capacity. Based on the expected continuous draw of the stepper motors, this should yield 45 minutes of continuous use. But power-saving methods will still be implemented, such as torque disabling of two out of four motors in some scenarios.

Further, we researched various effective path planning technologies such as 'Distance Transform Planner' and 'Lattice Planner.'[1]

The OrTak

The OrTak, or Order-Taker, will be the module present on each of the tables in the restaurant. The main processing unit of each OrTak will be a Raspberry Pi Zero. Power will be provided by 4 AA batteries in series. The OrTak's main feature will be to take food and drink orders from guests/customers. For this, the OrTak will be enabled with a speaker, and a microphone. The OrTak will prompt each patron and take their respective orders through speech recognition. Another important feature of OrTak will be the bill payment integration. Guests/customers will be able to signify to OrTak that their meal is over and they will be able to pay their bill through OrTak as well. All the OrTak devices will be connected to the central node/robot that will relay all the tasks to the kitchen and in turn the WOK-er. Speech recognition will be provided with the Google Cloud Speech API

The Google Cloud Speech API was chosen as we saw the most accurate speech transcriptions with it. Other libraries, like the PocketSphinx library, worked faster but saw noticeable discrepancies between what was said and what was detected. In our application, we decided that some delay for speech transcription was acceptable, so we found that our chosen API was sufficient.

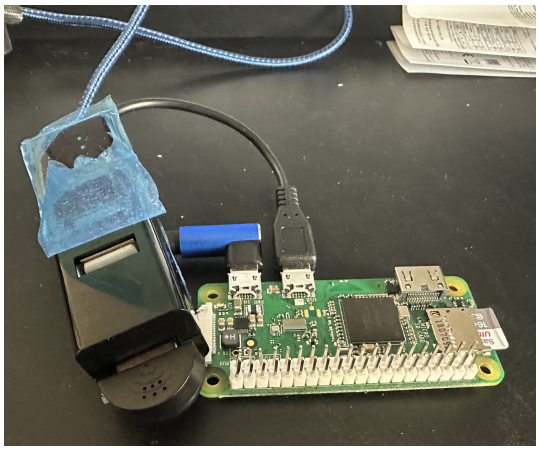

Figure 9: Prototype of the OrTak

The Microphone

We did a lot of research into what kind of microphone works the best with the Raspberry Pi. There were quite a few options but we wanted the OrTak to be a compact system which would take up very less space on a table at the restaurant. We found this microphone online which was very small. We were skeptical at first about the quality of the microphone but to much of our surprise, it worked really well. The one thing we need to check for is the quality of the microphone in a noisy environment.

Figure 10: Microphone for the OrTak

The KitchenNode

[1] https://petercorke.github.io/robotics-toolbox-python/mobile_planner.html#discrete-grid-based-planners

The KitchenNode is the module present in the kitchen of the restaurant. This module receives all orders sent from the network of OrTak's and dictates when and where the WOKer should go. Essentially, the KitchenNode coordinates the entire ServoBot system. The KitchenNode consists of a touch-enabled screen connected to a Raspberry Pi Zero. Kitchen workers will see orders from all tables displayed on the touchscreen. When workers are finished making the food for each order, they simply tap the name of the item to check it off. Once the KitchenNode detects that all orders for one table are finished, it automatically marks that table's order as ready and signals for the WOK-er to stop by the kitchen and receive the food to be delivered. The WOK-er then makes its way to the respective table and the KitchenNode resets to send out future orders. The GUI of the KitchenNode utilizes the Kivy and KivyMD Python libraries. To receive the orders and send out the WOK-er commands, MQTT is leveraged using the PahoMQTT Python library.

The Kivy and KivyMD libraries were chosen as they easily allowed for a sleek and modern GUI to be created. Other libraries were tried, like PySimpleGUI, but we found the documentation for those to be more esoteric and difficult to create good-looking results. We specifically chose to use the KivyMD variant of the Kivy library as we wanted to take advantage of the Material Design framework.

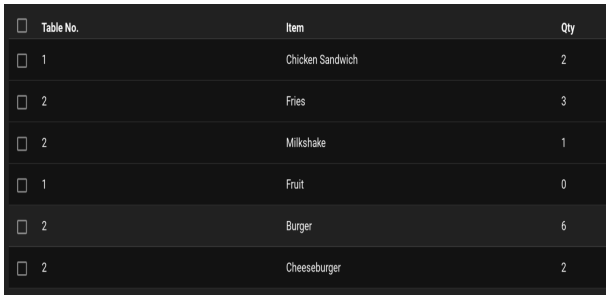

Figure 11: The KitchenNode GUI

IV. PROGRESS TO FINAL PRODUCT

Overall, our progress to the final product has been to our satisfaction. For each of the three components of the ServoBot system, the following goals for this quarter were set:

WOK-er:

- Build WOK-er prototype that can move in all four directions at a suitable speed
- Correctly identify where to go with camera mounted on WOK-er

OrTak:

- Identify which speech recognition library/API to leverage
- Successfully take in food orders with speech recognition on the Raspberry Pi Zero and send to the KitchenNode MQTT topic

KitchenNode:

- Successfully receive and parse all orders from OrTak modules
- Create GUI for KitchenNode that kitchen staff can interact with
- Successfully send commands to WOK-er via MQTT topic

In one way or another, all of our goals for the quarter were accomplished. All goals were able to be demonstrated by the end of the quarter, with minimally-viable prototypes for each node.

For the ***WOK-er***, we made significant progress in the construction of its mechanical components. We were able to successfully create a robust physical prototype which utilizes the selected stepper motors and omni-directional wheels. Code was developed that translated movement directions directly into robot movement. Additionally, the prototype moved at a decent enough speed that would not inhibit acceptable delivery times. We set a goal for this quarter to have a mechanical prototype with some implementation of image processing as well. In order to integrate the recognition capabilities, the Raspberry Pis were used and we aimed to have the code running on a Raspberry Pi independently. For image recognition, there were two primary modules we aimed to implement: a path following system and an obstacle detection system. For path following, we needed landmarking so that the robot could isolate the tables and move towards it in conjunction with obstacle detection which ensures it avoids things in its way. To identify individual tables, AprilTags were chosen for their versatility and robustness. The WOK-er was able to recognize AprilTags at a distance and thus distinguish between unique landmarks within the environment. Stereovision libraries were also utilized and implemented on the two cameras shown in Figure 6. We were able to identify obstacles and the distance they were away from the robot. Whilst we were successful in getting a minimum viable image recognition module running, there was a significant struggle in running the code on the Raspberry Pi processor due to its

limited processing power. Next quarter, we will be optimizing the code to run on the microcontroller as well as implementing a path-planning algorithm that assists the robot in identifying the fastest path to a table and works in parallel with AprilTags which serve as landmarks.

For the ***OrTak***, our goal for identifying which speech recognition library/API we wanted to leverage was met. We tried both the PocketSphinx and SpeechRecognition libraries in Python. After trying both, we found that the SpeechRecognition library was simple to use. Additionally, as the SpeechRecognition library had the ability to utilize the Google Cloud Speech API, the results that we got using the SpeechRecognition were much more accurate. One concern with the Google Cloud Speech API was that there was some delay between the phrase spoken and the program transcribing it, as the library sends a phrase to an external server for transcription. But, as our application is simply taking orders, we thought that some delay was acceptable for our situation. Once we had the library/API chosen, that allowed us to meet our second goal of successfully taking in orders and sending them over MQTT. To do this, we programmed some logic to write out a prompting question to a console screen, wait for a response, and repeat until the order is complete. Users respond with either "no", the food item they want, the quantity they want, or any special requests, depending on which prompt is shown. While this setup is more rudimentary than what we want in our final product, it allowed us to prove that using speech recognition for taking orders was viable.

For the ***KitchenNode***, the first goal of successfully receiving and parsing all orders from the OrTak was met. Receiving was relatively simple as the KitchenNode simply had to listen to the same MQTT topic that the OrTak sent to. Parsing was a little more complex, as we needed to decode a string from the OrTak that contained all relevant order information. To do this, we first settled on an order string format, where we decided on a string delimited by semicolons and colons. Once the order string format was decided upon, we wrote the code to parse the order string. With all the information extracted, we sent this information to be displayed on the GUI. The second goal of creating the GUI. We quickly decided to use the Kivy and KivyMD libraries to leverage Kivy's ease of use and the KivyMD fork's aesthetically-pleasing Material Design components. We decided to show each item and its table on its own line. Once that was decided, creating the GUI was relatively straightforward, although there was some initial difficulty due to KivyMD's lack of documentation for some components. Once the GUI layout was created, it was easy to port in the order information from the OrTak. Our third goal of sending orders to the WOK-er MQTT topic built upon the previous two goals. To accomplish this, we added logic that once all orders from each table were checked off, the order should be sent to the WOK-er. To check off orders, we utilized the KivyMD Checkbox properties. Sending the orders to the WOK-er was relatively simple with the Paho MQTT library, and we made sure to add a queueing system for the tables the WOK-er should go to in the unlikely case that the kitchen prepares food much faster than the WOK-er can bring it out to the patrons.

A general goal for moving forward is to clean up a lot of the code written during this quarter and do more robust testing to catch any unknown edge cases.

V. EXPERIMENTAL VERIFICATION & TESTING

The WOK-er

We did a lot of testing for camera calibration. Many different images were collected to calibrate the camera. The data images were pictures of the calibration chessboard moved through the x and y fields of view of the cameras. A total of about 30 different images were collected. These images can be found in the data folder. The images were collected both from the right and left side cameras. These images were then fed into a calibration method of opencv.

We collected further data for optimizing AprilTag distances. To ensure that the AprilTags work as intended, we collected calibration data which was used to determine the repeatability and reliability of the AprilTag detection. We did this by creating consistent setups with known measurements and distances and then comparing that to the data gathered by the camera running the AprilTag code. A diagram of the setup is included below to illustrate the measurements.

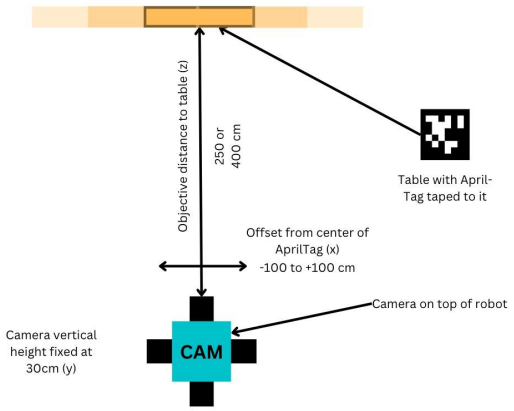

Figure 12: AprilTag testing setup

A data table is also included in Figures 13 and 14. for various trials at different locations camera relative to the AprilTag. The data obtained had mean relative errors of 6.64% for the x-values, 13.94% for the y-values, and 4.16% for the z-values as compared to the objective values. These errors were sufficiently low for our purposes as long as we can implement a robust path-planning algorithm next quarter thus we decided to stick with the AprilTag code with only minor adjustments made due to the nature of the wide-angle camera. Next steps in our data collection is collecting data when multiple AprilTags are in the field of view to see if averaging will increase the accuracy of localization.

April Tag Localization	Sideways Offset	Height from floor	Distance from camera
Angle To Target Objective	Objective x (cm)	Objective y (cm)	Objective z (cm)
-21.801	-100	30	250
-16.699	-75	30	250
-11.310	-50	30	250
-5.711	-25	30	250
0.000	0	30	250
5.711	25	30	250
11.310	50	30	250
16.699	75	30	250
21.801	100	30	250
-7.125	-50	30	400
7.125	50	30	400

Angle To Target Measured	Measured x (cm)	Measured y (cm)	Measured z (cm)
-22.0158388	-93	23	230
-18.14918671	-79	33	241
-11.04630345	-49	31	251
-5.492324557	-25	35	260
0.9203349668	4	29	249
6.044092162	27	30	255
11.22068678	49	26	247
15.9173647	77	34	270
19.51079944	90	26	254
-7.730455332	-60	20	442
7.789285478	58	23	424
STDEV in Error	0.06543	0.09980	0.03488
MEAN of Error	0.06636	0.13939	0.04155

Figure 13: Data tables showing measured vs objective data for AprilTag testing

After collecting data from the AprilTag, we had to ensure that our angle to target objective could be translated into motor control commands. The angle-to-target data is translated into the motor control commands in the Arduino Nano. This is done by multiplying the angel by pre-calculated angle-to-steps value. The stepper motors are then activated to step the calculated number of steps in clockwise or counterclockwise direction.

Finally, we collected data to evaluate the reliability of stereo vision in detecting objects. We collected data for different sized objects at different distances. Since this library had been extensively developed via previous testing, we simply had to validate the code on our camera.

To prepare for the market, we will test the robot in an active environment. We must consider failures from falling plates, and spillage of water. Further, we plan on creating a low platform for stability but will experiment with raised platforms, and if sufficiently stable might utilize a platform extension to raise the platform the plates and items are stored on.

The OrTak and KitchenNode

For the OrTak and the KitchenNode, we started off by working on the two components separately. We first wanted the OrTak speech recognition to start working. We were using the computer's microphone so we were expecting the results with a cheaper USB microphone to be worse than what we got using this.

We started off by using the PocketSphinx library. We tried to just say basic words like "left" and "right" but did not get very accurate results. As seen in the screenshot below, we get homonyms like "night" when we say "right". Thus, we decided to explore other avenues for speech recognition on the OrTak.

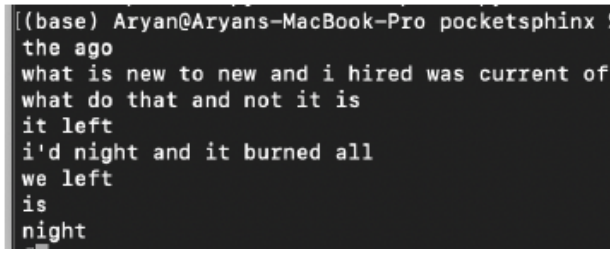

Figure 14: Inaccuracy of PocketSphinx

After doing some research and talking to some more students, we realized that the best speech

recognition software is the Google Speech-to-Text API. We did some testing with it and it worked perfectly with the computer's microphone. Of course, we did have to test using the Raspberry Pi and the USB microphone as well. But, we did a comparison between the Google Speech-to-Text API and PocketSphinx and got very good results. We tried the phrase "testing 1 2 3 4 5" and tried comparing the results of the two softwares. The results can be seen below.

```
C:\Users\Kenson\Desktop>python example.py
Say something!
Sphinx thinks you said testament to do for five
Google Speech Recognition thinks you said testing 1 2 3 4 5
```

Figure 15: Comparison of PocketSphinx and Google Speech-to-Text API

Thus, the decision to use the Google Speech-to-Text API was made. The next phase of testing included the testing using the USB microphone and Raspberry Pi. By the time we got the USB microphone, we had the order parsing code ready. We were able to directly test the speech recognition on the microphone with a prototype of the OrTak. We were able to ask for different kinds of food that it recognized. We were also able to ask customers to repeat their order if they asked for something that was not a part of the menu. We were a little worried about the speech recognition for people with different accents and pronunciations so we tested that as well. To our relief, we did not see any problem on that end either and it worked perfectly fine.

As seen below, we are able to converse with the OrTak and ask it for a particular food item and quantity.

```
pi@raspberrypi:~/Desktop/Team
What would you like to order?
chicken sandwich
How many?
2
```

After this, we moved focus onto getting the customers order into a perfect string formatted in a way for the entire ecosystem to understand. We added code to the order parsing file to take the order and update the string. Below you can see the order string broken down into three different screenshots for proper formatting.

```
TN:1;Items:chicken sandwich-2,
milkshake-3,burger-5;Tot:10;Cost:67.5;
SR:no ketchup please
```

Figure 16: Order String

From here on, the OrTak section was complete. All we needed to do was to send the data over to the kitchen node using the MQTT communication protocol. We first tested the MQTT protocol for basic functions like explained in the lab. We did this between two computers and also between a computer and a Raspberry Pi.

```
pi@raspberrypi:~/180DA-WarmUp $ python mqttsubscriber.py
Connection returned result: 0
Received message: "b'0.054898832962881006'" on topic "ece180d/test" with QoS 1
Received message: "b'0.5167903726137075'" on topic "ece180d/test" with QoS 1
Received message: "b'0.8273297499942227'" on topic "ece180d/test" with QoS 1
```

```
(venv) Aryan@Aryans-MBP 180DA-WarmUp % python3.9 mqttpublisher.py
Connection returned result: 0
Expected Disconnect
```

Figure 17: MQTT Protocol Testing

The MQTT protocol worked very reliably and we were able to send over the string to the KitchenNode very easily and promptly. We then moved onto the testing for the KitchenNode. We wanted to make a basic table displaying all the food and orders. For this we used pysimplegui initially but that gave us a lot of errors and was not reliable. The interface however looked a little better.

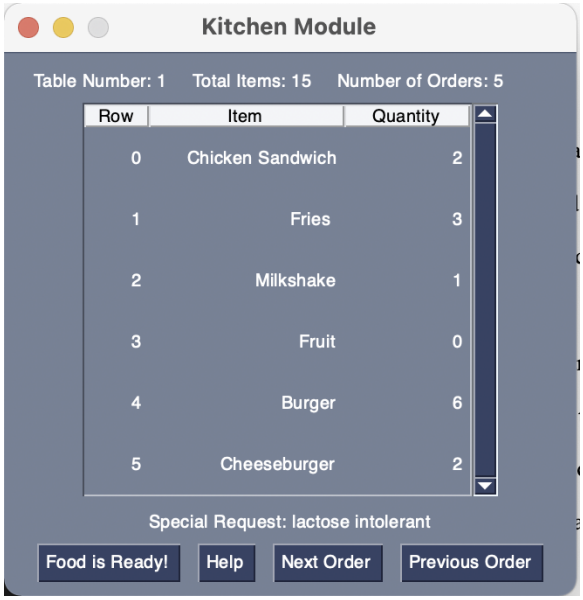

Figure 18: Pysimple GUI version of KitchenNode

We then moved to Kivy and KivyMD which had a library for tables in particular. This helped us a lot because we were able to use the library to make checkboxes to get the rows to disappear on a click.

Table No.	Item	Quantity
1	Chicken Sandwich	2
1	Fries	3
1	Milkshake	1
1	Fruit	0
1	Burger	6

Figure 19: First prototype using Kivy and KivyMD

Thus, we chose the Kivy and KivyMD version. We did all sorts of testing on this. We tried double clicking rows and also clicking them very fast. We tried clicking on different areas on the row as well.

When it comes to the OrTak or the KitchenNode, one thing we need to test extensively in the case where multiple orders are sent from different OrTaks at once. When we are scaling up with multiple OrTaks, order management becomes a very important feature.

We plan on testing using people who don't know much about what we are building to get an idea of how user-friendly the system is. We will also need to do some testing with kitchen staff to see what order format suits them the best and test different variants. We want to talk to restaurant managers, owners and staff to gauge what changes can be made to make the system friendly for not only the customers at the restaurant but also for the customers of our product.

The way we have divided the three components – KitcheNode, OrTak, and WOK-er – is to make them individually useful as well. We can easily test each of these components without the others. The only thing which we need to take care of when it comes to the entire system is communication between these three.

Future Testing

We came up with a plan for testing and validation for the upcoming quarter as well. We need to test the following things:

- Robot localization with multiple AprilTags. This will be tested in a room of known size by comparing the known distance with the distance measured with AprilTag. If we are unable to do this with the current setup, we plan on adding more cameras
- Obstacle avoidance with StereoVision. This will be tested with variety of objects to see if robot will stop and then go around the obstacle.
- Text-to-speech for order taking to eliminate need for screen. The tentative plan is to use pyttsx3 library and a cheap USB speaker.
- Speech-to-text microphone thresholds. We are currently using default settings for the microphone input and want to play around with it to make it as accurate as possible.

VII. WORK DISTRIBUTION

We have allocated the roles based on our previous experiences, and we believe this will maximize the efficiency and reliability of our design. Furthermore, while our responsibilities are clearly outlined we will work together, especially when defining integration between different components.

Aryan Agarwal (IoT & Component Integration)

He will focus on developing the IoT network between the order-taking modules, the hostess module, and the delivery robot. Furthermore, will ensure the integration of the various nodes to ensure correct function.

Kenson Nguyen (IoT & Speech Processing)

He will assist in developing the IoT network and develop speech processing capability for order-taking modules.

Marcell Veszpremi (Robotics)

He will take on a large part of the design and development of the robotic system. Including the physical and software motion components. Will assist with nodal network design. Will also assist with the aesthetic design of the 'OrTak'.

Utkarsh Kumar (Gesture & Visual Processing)

He will focus primarily on image processing and gesture recognition to assist in the path calculation of the robot. He will also be working on the stereovision aspect of the robot.

Each of the members will be working on their assigned tasks over the week and will reconvene every Wednesday to work on the group projects together. The team will be writing the section of the report that they worked on. Before submitting, everybody proofreads the entire document to see if there is anything missing.

VIII. WINTER QUARTER TIMELINE

Week 1-3

We will regroup and look for potential changes we want to make. We will enhance features and expand the menu options. We will further improve the UI of the 'POS' system.

Week 4-6

We will do extensive situational testing and prepare the device for market. Improve aesthetic design where possible and increase reliability. (e.g. making battery replaceable easily)

Week 7-9

Further continued testing trying to eliminate bugs on corner cases. Each week the work allocation will be split up by category as described in the "Work distribution section." Furthermore, throughout each stage we will be debugging extensively amongst each other to ensure that we minimize and don't carry forward too many bugs.

Stretch Goals

Our team likes this idea in particular because we feel that it has a lot of potential and applications not just in the restaurant industry but also in various other industries. Our ultimate stretch goal would surely be to make the product good enough to pitch to investors and get funding to do it on a larger scale. In the future, this idea can be expanded upon in a startup where it will not only be ServoBo marketed, but also the movement platform itself.

Fallback Options

We are confident in our timeline to fully execute the ServoBot. On the off chance that something goes wrong, we could reduce some of the features that we mentioned earlier. For example, if gesture recognition is not working, we can program the robot to just stop and wait until the obstacle is out of the way. The project is flexible enough where the complexity is able to be toned down.

IX. CONCLUSION

The ServoBot ecosystem is the need for the hospitality industry. We believe that even the three components individually have so much to offer that they can be incorporated into different systems as well. The ServoBot has significance in each industry and will present abundantly in the coming years. We do have certain improvements in mind that we want to execute next quarter. Our future goals are as follows:

For the WOK-er:

We want it to be able to localize itself correctly and move to the table number it is sent without any manual assistance. We also want it to have sufficient traction so that it can move around without slippage and at a sufficient speed.

For the OrTak:

The goal is for the OrTak to be able to accurately receive orders in a noisy environment and send orders to the KitchenNode. To do this, the microphone sensitivity thresholds will be adjusted and filtering will be added if needed. Additionally, text-to-speech will be incorporated to eliminate the need for a screen for each OrTak.

For the KitchenNode:

The KitchenNode must be able to show all relevant order information to kitchen staff and easily manage order queue to send orders to the WOKer. We also plan on utilizing Google's Real-Time Firebase to store order information. Doing this will allow the ServoBot system to continue to function in the event that the KitchenNode is somehow power cycled during operation.

We feel we achieved more than what we were aiming for this quarter. We wanted a very basic version of the WOK-er working, a prototype of the OrTak and a conceptual KitchenNode. But, we were able to get a semi-autonomous version of the WOK-er with the April Tags integration. We are very close on getting the stereo-vision obstacle-detection running as well. The OrTak is fully processing speech and is able to communicate with the KitchenNode as well. The KitchenNode is able to receive orders from the OrTak and process them. It is also able to display the different orders and the kitchen staff can interact with it to complete orders. All in all, we have a basic prototype of the entire system running. Everything went as planned and we solved all the problems we ran into as a team with the help of the professor and the TA.

We are all very excited to take this one step further next quarter and complete the project. We are looking forward to building on our progress and creating something useful and impactful!

VII. APPENDIX

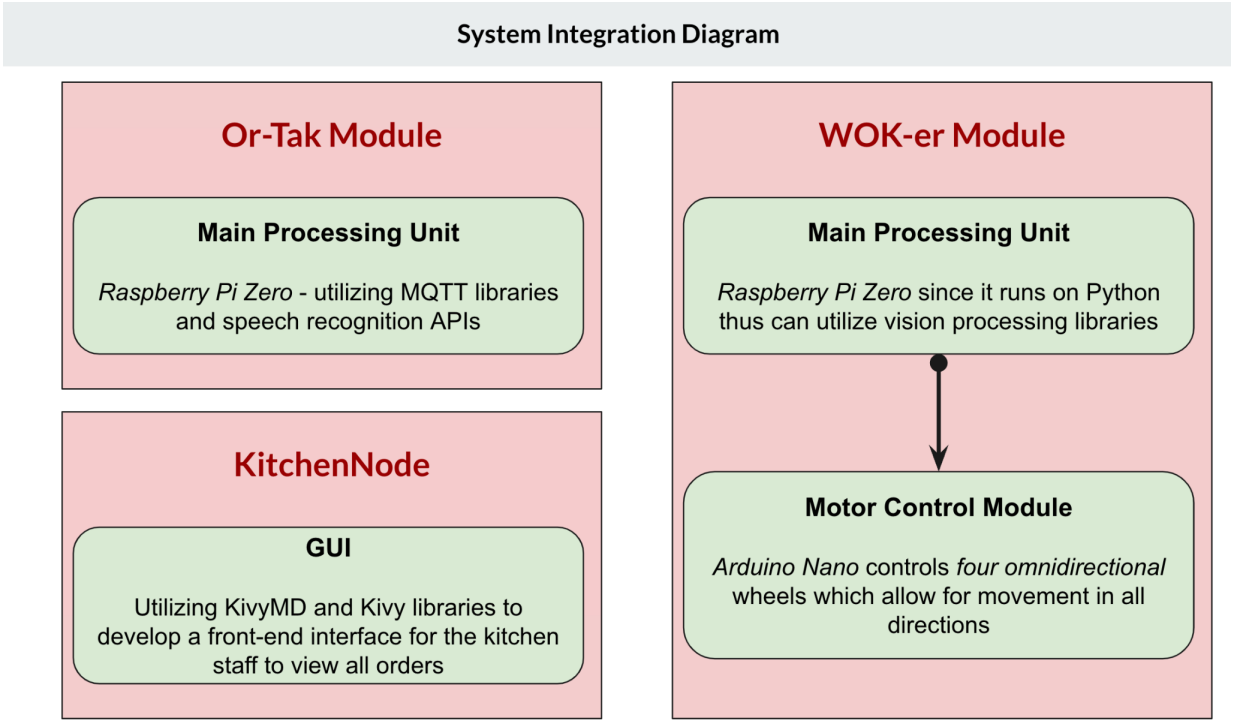

Figure A: (top) Flow chart demonstrating the robot in action

Figure B: (bottom) System process flowchart representing each module in action