# Final Report

Embedded Systems Design - Fall 2019, Team 10

Members: Abhishek Damle, Chris Blackburn, Sean Gallagher, Tsheetiz Tamang

# Table of Contents

# Section 1: Summary

For our project, we designed and prototyped a “chef robot” that selects and fetches ingredients from a pantry area. This was modelled after a 2016 Amazon Picking Challenge and simulated some of the challenges that autonomous picking robots face in a warehouse. The components of the robot were a robotic arm, a sensor module for the robot arm, a three omnidirectional wheeled rover, and the sensor module of the rover. Each of these four components were controlled by Ti CC3220SF microcontrollers running FreeRTOS and used the MQTT protocol to communicate with each other.

The role of the robot arm and its sensor module was to choose the right ingredients based on a recipe and place them onto the rover. The arm achieved this by picking up each ingredient, one by one, and presenting it to the arm sensor module. If the arm sensor module detected the ingredient was a part of the current recipe, the arm placed the ingredient into the rover. Otherwise, the arm placed the ingredient back in its original location.

The role of the rover and its sensor module was to navigate between the pantry and customer. The rover sensor module provided the rover with the heading of the markers in the two areas as well as the distance in front of the rover. A navigation algorithm used this information to direct the rover to the two areas.

# Section 2: Table of Requirements

| Number and Category | Requirement Statement | Satisfied | Explanation |
|---|---|---|---|
| 1. Functional | The System shall consist of<br>1.1 the kitchen<br>1.2 a robot arm<br>1.3 the chef rover<br>1.4 the server and GUI<br><br>The kitchen consists of a pot placed some distance away from a shelf of ingredients.The overall objective is for the chef rover to retrieve ingredients from the shelf and bring them to the pot. The robot arm will be located near the shelf and shall place ingredients from the shelf onto the chef rover. Upon being loaded with ingredients, the chef rover must navigate to the pot on the other side of the kitchen. | yes | The chef rover was able to fetch a series of varied ingredients in multiple consecutive runs. |
| 1.1 kitchen | This is the environment of the project. The kitchen, as shown in figure 1, consists of the | yes | We constructed the environment to meet these specifications. Since the |

| | | | |
|---|---|---|---|
| | pantry and stove areas. Two colored blocks will be used to designate the loading zone and the pot.<br><br>The stove area will consist of a stationary but arbitrarily placed pot where the ingredients must be brought. Meanwhile, the pantry area will house the ingredients, the robot arm, and a marker. The chef rover will be placed between the pot and the pantry, but it will be in-line with them (i.e. it should have the opportunity to move straight on to the pantry loading zone.<br><br>The kitchen will be at least three times as wide as the chef rover, and five times as long. | | rover did not have knowledge of the placement of the markers, their placements were arbitrary. |
| 1.2 robot arm | Once the chef rover arrives at the loading zone, the robot arm must retrieve the ingredients required by a specified recipe and place them on the chef rover.<br><br>The ingredients will be placed next to arm and will wait for the recipe. Once it gets the ingredients needed it will start picking up the blocks and check if it is required, and if it is then the robot arm will place the ingredient onto the rover which will be in the loading zone. If the ingredient is not right then it will place it back in its original position.<br><br>Successful operation will be determined through<br>1. The robot arm's ability to select the correct ingredients<br>2. The robot arm's ability to place the ingredients into the chef rover | yes | The arm successfully placed the correct ingredients from shelf into the rover, rejecting the ingredients not in the recipe. |
| 1.3 chef rover | Rover 2.5 must be used to fulfil the role of the chef rover. The chef rover must navigate between pot and loading zone. The chef rover must be able to account for the arbitrary placement of the pot.<br><br>Successful operation will determined through<br>1. The chef rover's ability to place itself in the loading zone such that the arm can place ingredients on to the rover without dropping them<br>2. The chef rover's ability to navigate between the loading zone and the stove area | yes | The rover was able to navigate between the two markers without knowing their positions beforehand. The rover also stopped correctly in the loading zone such that the arm never missed placing an ingredient in the rover. |

| | | | |
|---|---|---|---|
| 1.5 the server and GUI | All components must communicate with the server and no communication amongst the individual components is permitted. The server shall be responsible for coordinating all of the individual components to place ingredients from a user specified recipe into the pot. A GUI as shown in figure 2 must allow a user to select between existing recipes.<br><br>Successful operation will be determined through<br>1. The server's ability to coordinate the components to reach the final goal of returning the requested ingredients to the pot<br>2. The GUI's ease of use and functionality | yes | The GUI was used to initiate the demo and the rover correctly fetched the ingredients that were requested by the user through the GUI. |
| 2. Cost | | | |
| 2.1 | The total costs excluding the provided materials must not exceed $400 | yes | The total cost of all of the parts was $261.30 |
| 2.2 | The robot arm must cost less than $100 | yes | The arm cost $44.86 |
| 3. Schedule | | | |
| 3.1 - 3.13 | Course Imposed Deadlines | yes | Submitted before deadline |
| 4. Standards | | | |
| 4.1 | The system must use common standards such as UART, SPI, and I2C to interact with sensors. | yes | Only common standards were used to interact with sensors. |
| 4.2 | Sensor data will be collected via interrupts on the microcontroller boards. | yes | All sensor data was collected via interrupts except for the sensors that had UART and analog interfaces. Due to the limitations of the drivers, UART and ADC interfaces used blocking calls meeting the specifications from the help document. |
| 4.3 | The system must use MQTT to communicate with the server using JSON payloads. | yes | The system uses MQTT to communicate with the server and uses JSON payloads. |

| | | | |
|---|---|---|---|
| 4.4 | Each microcontroller board will use FreeRTOS as the scheduler and tasking agent to carry out commands through each robotic component. | yes | Each microcontroller board uses FreeRTOS as the scheduler and tasking agent |
| 4.5 | Message queues will be used to communicate between tasks on each microcontroller board, whether that be in communication with the server or with a sensor on the board. | yes | Only message queues were used to communicate between tasks |
| 5. Ethical and Professional | | | |
| 5.1 | Must comply with intellectual property laws and policies/guidelines.<br><br>Specifically, the license terms of each IP used must be met. | yes | Complied with intellectual property laws and policies/guidelines of all components used in the system. |
| 5.2 | The system and team members must comply with IEEE Code of Ethics - specifically points 3,6,7,8,9,10 | yes | IEEE code of ethics were followed |
| 6. Public health, safety, and welfare | | | |
| 6.1 | The ingredients must never make contact with the floor. | yes | The ingredients were never dropped to the ground during the multiple runs performed in the final demo |
| 9. Social | | | |
| 9.1 | Must employ a simple UI that can be used quickly by both the elderly and restaurant employees. | yes | The UI was simple |
| 9.1 | The UI must display the progress of the ingredient collection. | no | Implementation flaw, although the information was available to the server, the GUI doesn't display the progress. |
| 12. Basic Code Requirements | | | |

| | | | |
|---|---|---|---|
| 12.1 | All of the code must be unit tested and must be written in such a way as to facilitate unit testing | yes | All components were independent and could be independently tested |
| 12.2 | The code size must not be excessive. Each individual contribution must be less than 500 lines and no C function may be more than 25 lines. | no | Implementation flaw, we did not make sure all the code size was not excessive |
| 12.4 | The code must follow general good programming guidelines such as no global variables, descriptive variable and function names, etc... | yes | Good programming guidelines were followed |

# Section 3: Incorporation of Engineering Standards

The main engineering standards used for the system included SPI, UART, MQTT, and JSON. SPI was used to interface with the rover motor encoders as well as Pixy2. UART was used to interface with the rover motors themselves and for debugging purposes. Since the sensors and motors used well developed serial communication standards, we did not have to develop software for these components and could treat them as black boxes. This greatly cut down on the complexity of the project while increasing reliability.

The system also used the MQTT standard to transmit data between the boards and the JSON standard to encode the data. Using well developed standards greatly reduced our development time and increased reliability. Conforming to a commonly used standard also meant that the system was very adaptable and additional components could be incorporated by a third party unaware of the specifics of our work.

# Section 4: Analysis of Robustness

Our system design robustness can be separated into three main categories. First, the system's physical robustness will be considered. Then, the system's software robustness will be considered. Finally, the system will be examined once more from the viewpoint of hardware/software integration to determine if this increases, decreases, or has no effect at all on overall system robustness.

The two Pixy Cams and the IR sensor used in this system are inherently robust from a physical standpoint. If all connections and cables are correct and undamaged, then the sensor hardware should perform in all environments, within reason. Similar reasoning can also be applied to both the arm and the rover platform. If all hardware is undamaged and set up according to appropriate specifications, these two devices should perform in all environments, within reason. For instance, test cases 1 and 2 for the arm are basic joint movement and grip strength. These two test cases always pass in different environments assuming default specifications are met (i.e. the arm joints are not overtightened and the claw servo is not applying an excessive amount of force, causing a stall). For the rover, test cases 1-5 are met in the physical sense simply by the rover being able to move forward, backward, left, right, clockwise, and counterclockwise.

From a software standpoint, the pixy cams and the IR sensor are fairly robust. Once trained on a certain color, the pixy cams are very accurate with no false positives. The rover sensors were able to find the loading and kitchen markers and stop the correct distance away from the marker. However, issues begin to arise if lighting changes in the same environment or the pixy cams are moved to a new environment without retraining. In this case, the pixy cams often did not register the required color, or false positives occur. Varying lighting can also interfere with the IR sensor; however, the IR sensor was more consistent during our testing. The rover is quite robust from a software standpoint. It simply follows direction from the server and corrects in real time according to instructions from the rover sensors or its PID algorithm. The relative speed and PID test cases are easily met in all environments. Similarly, the arm is also quite robust. It receives servo positions from the server and uses a motion smoothing algorithm that is timer-based to step to the arm through positions leading to the new position. This allows the arm to easily pass the test cases for retrieving each ingredient and returning incorrect ingredients to their original locations. It also allows the arm to easily pass the test case of dropping the correct ingredients onto the rover. These positions are hard-coded into the Raspberry Pi server so an environment change/stress would not affect arm motion. There is also no ability to feed incorrect inputs to the arm.

From a hardware/software integration level, this system is very robust. Excluding the possibility of the pixy cams having issues in different lighting, there are no parts of the system that would not work in multiple different environments. There is only one input to the overall system via a GUI with a dropdown list. There are 7 recipe options and an “Execute” button. All that happens when “Execute” is pressed is a state change within the software and an update to the recipe string. Thus, this button can be pressed at almost any time and all it would do is place the system back in its initial state. This would only be a problem if the button was pressed during or after the loading process. Otherwise, there is no other way to provide faulty input and stress to the system. Due to the system specifications that have been laid out, the playing field will always be flat and clear of obstacles. Thus, the rover and its sensors will not have trouble. Therefore, all four system requirements: rover departure from kitchen, rover arrival at loading zone, arm loading, and rover return, are easily met no matter the stress or invalid inputs except for the specific exceptions mentioned above.

The limitations of our system design are differing lighting that can affect the sensors and invalid input from the GUI during the loading/return stage. All other stress possibilities observed during testing were handled flawlessly by the system. Sometimes the rover took a while to arrive at the loading zone due to the sensor-based movement algorithm, but it always arrived successfully. If timing was a strict constraint, then this might cause issues but this will be further discussed in the reliability section below.

# Section 5: Analysis of Reliability

We have four different components that make up our overall design. The four components fell into two categories: arm and rover. For the rover we tested the rover motors, rover sensor, arm

motor and arm sensor. As stated throughout this report, we wanted the rover to act as a “chef” rover, starting at an arbitrary position and finding its way to the loading zone. To achieve this we needed to make sure that the rover could move, use its sensors to find the loading zone, etc. To make sure that the rover could effective execute its task we tested the movement capabilities fully. We tested the rover to make sure that it could take in a directional angle, rotation direction and speed in several test cases. The rover never had any trouble moving throughout the demos, once it found the loading zone it was able to move left, right, backwards, and forward towards the zone.

The rover also had an IR distance sensor and Pixy2 cam which was used to locate the loading zone and destination zone. We had problems with the Pixy2 because it would not properly detect certain colors. To overcome this we trained the Pixy2 to detect darker colors. After solving this problem, the Pixy2 never had a problem detecting the loading zone again. The IR distance sensor had to be adjusted so that the arm could drop the ingredient into the loading mechanism on top of the rover. After these minor adjustments were made to the distance sensor and Pixy2, the rover never had problems detecting the loading zone and stopping at a distance where the arm could properly drop the ingredients off. Moving onto the next component, the arm was very reliable.

Since there was a budget on the arms, we had to settle on a cheap arm that could meet our requirements. This arm did not have many problems, but was a little jittery but was solved when we tweaked the voltage. The main purpose of the arm was to pick up the ingredients and show it to the Pixy2. If it is the right ingredient then the arm would move to the loading position and drop the ingredient into the rover, and if the ingredient was incorrect, it would return the block into its original position. A lot of testing was done to make sure that each movement was smooth, the claw was strong enough to hold the blocks, etc. The sensor that was used for the arm was also a Pixy2, but instead of returning the x, y, height, and width, we just wanted the color signature of the block. Since there were only 3 colors set (Red, Blue, and Green), the Pixy2 did not have any trouble detecting the block. After we set the color signatures in PixyMon, the Pixy2 did not have trouble detecting what ingredient was being presented, allowing us to successfully execute the recipe we wanted.

To tie all of the components together, we created a GUI that would allow the user to select the recipe they wanted to execute the process. We never ran into any problems with the GUI and turned out to be a very creative way of tying all the parts together. Overall, we used the GUI to test all four different components and, during our final demonstration, we were successfully able to run 4 different recipes without any problems.

# Section 6: Design Improvements

The main thing we were lacking was sensing capabilities (primarily due to poor initial planning). Had we included more sensors into our system, the rover would have been able to use its complex movement capabilities, and the arm could have moved more organically.

With our current design, actuators were very static. The arm cycled through predefined complex motions, and would choose which set of motions to take depending on the state of the system. It needed to grab a block, show it to the pixycam2, then either put the block back, or dump it into the rover. To improve movement, we could have added a short-range distance sensor onto the arm and mounted the pixycam2 for the arm such that it oversees all the arm's movement. With that, we could have implemented a *scanning search* for ingredients (colored blocks). Even sticking with the existing movement, a short-range distance sensor could have verified something was in the expected position as opposed to us blindly grabbing at what we expect to be there.

As for the rover, we overestimated the pixycam2's capability. Come to find, it's not extremely reliable and fails unless we are put into the conditions we trained it under. And, without supplementary sensors, system actuation is limited. Adding additional distance sensors would have allowed us to better orient the rover. Specifically, two distance sensors on each side of the rover would have allowed us to create a better model of the rover's surroundings and allowed us to make the rover parallel with any object in its way (if two sensors in the same area have similar readings, we are parallel). With that, we could move omnidirectionally.

Adding another pixycam2 for stereo-vision could have produced a desirable result, but again, the pixycam2 isn't very reliable. We believe that's from the cheap camera module used in the pixycam2. It can't focus like the pixycam1 could, and doesn't perform well even in slight changes in lighting.

# Section 7: Lessons Learned

One thing we didn't do such a good job on was planning out what kind of sensing we'd need. At the beginning of the semester, we had some grandiose idea of what we wanted our final implementation to look like. I'm certain everyone went through something similar. At the same time, we overestimated the capabilities of certain parts of the project. For example, we relied on only two sensors for the rover: a pixycam2 for general vision sense, and a distance sensor more easily add depth to the pixycam2's sensing.

Having done that, we were extremely limited when trying to join each component (namely the rover and rover sensors). Originally, we wanted to be able to navigate much more complicated arenas with obstacles, but with only the pixycam2 and the distance sensor, our sensing capabilities were very one-dimensional. That kept us from using the full capabilities of the rover's complex movement. That is, instead of using the rover's omnidirectional movement, we were limited to rotating and moving forwards.

With computer vision, it could have been possible to setup the arena in such a way to determine the rover's orientation, but again, without proper planning and overestimation of the pixycam2's immediate capabilities, we were left shorthanded.

The main lesson learned here is to use one or two more sensors than you think you'll need, or to at least iron out details of movement and sensing without relying on one main sensor. Even with one extra distance sensor, we could have created a way to make the rover parallel with the loading zones as opposed to limiting its movement.

Additionally, the pixycam2 has trouble detecting learned colors even in slight changes of lighting. Earlier on, we were using a particular shade of light green. With that color, it's almost like the pixycam2 would *untrain* itself. Come to find, the shade of green was just too reflective for the camera to recognize consistently.

Again, the lesson learned here is to *not* overestimate a single sensor's capability and to think more critically about what sensing you will need to help any actuators in the system move.

Beyond that, we learned to *not* underestimate creating a robust, responsive component. That is, it's easier said than done to make a component send and receive messages at high frequencies. The faster we can receive positional data from the rover sensors, the faster the rover can move. It's worth putting time into optimizing each component to be as fast and efficient as possible.