Product: Clyde
Team: Cloud 9

Abstract

ClyDe is a system that sanitises desk tops and other flat surfaces in high-traffic spaces such as libraries and offices. For this demo, the team has dedicated significant efforts into continuing the integration work, as well as work on other important tasks, like the user guide. In addition to this we have focused more on usability than before, and have refined details on how the product will be used, as well as developing user interfaces. The team has also focused more on product presentation this week, working on how best to present a coherent idea and value proposition for the industry day.

Introduction

1. Project plan update

The following milestones were planned for this demo:

- Integration - Partly Achieved
- Build team website - Partly Achieved
- Finalise business details - Achieved

Our main work for this final demo involved further integration of system components into a more final 'product'. This included both adding and improving the user interface for visitors, staff, and technicians.
Again, the group has organised goals in the same way as for previous demos: Git branches helped us efficiently integrate the robot's subsystems, Trello allowed us to delegate tasks and daily meetings in Discord meant we kept on top of progress. Google Drive has been useful for other collaboration work, especially to create the user guide, video plans and drafts for this demo report.

1.1. Budget

Money: £10 on database hosting.
Technician Time: Less than 1hr. The technicians were not needed much for this demo, but were consulted for some hardware questions. Since the team has not interacted with the physical system, we needed advice on certain topics like charging.

1.2. Individual Contributions

See the table below for individual contributions by task.

Task	Contributors
User Guide	All team members
Website	All team members
Integration Work	Lars, Adel, Josh
User Interface	Lars, Ivan, Jeffrey, Shining
App	Jeffrey, Shining
Usability Analysis	Ivan
Demo Video	Ryan
Mapping Tutorial Video	Sean

2. Technical details

2.1. Hardware

2.1.1. DFRobot non-contact liquid sensor

The robot needs to be able to sense how much disinfectant is left in its reservoir. This way, the robot can send an alert when its reservoir is empty, so it can be easily refilled. Without a sensor it would result in marking desks as clean when they shouldn't be. This sensor works with the onboard Arduino and is easy to install. It has no special requirements for containers. The sensor being non-contact is an extra benefit as it reduces contamination of the liquid and reduces the chances of hardware touching it, which could cause failures.

2.1.2. Raspberry PI 4B - Controller

We decided that setting up the robot would be made simpler if we included a Raspberry Pi pre-installed with all the required software to interface with the robot at a more direct level. This is because the navigation of the robot has to be configured using a remote computer with ROS2, and we decided that requiring a library computer to install ROS2 is not acceptable. The reasons for this include potential compatibility issues and installation issues which we've experienced first hand, and non-Linux installations of ROS2 are even more challenging. Shipping with a bespoke control computer gets rid of potential compatibility issues. The idea is that this would serve as the control centre for ClyDe, meaning the system has a clear interface for the users to turn to. This Pi could also be used to control the database of tables or to command ClyDe to go to the home position. It would also allow the setup and environment mapping to be done more easily. The Raspberry Pi is powerful, lightweight and affordable, which makes it an excellent candidate for this purpose.
A downside to this is that it requires the library to use a monitor, keyboard and mouse to interface with the Pi, but we found it reasonable to assume it would have this.

2.2. User interface

2.2.1. Raspberry Pi Software - ClyDe Controller

Installed on the host Raspberry Pi will be controller software for ClyDe. Since the all the commands needed for operation are pre-determined, we can wrap them in a simple UI that will guide users through both setting up the system, and normal operation. Implementing this would improve usability by replacing the need to interact with the system via the command line. Due to time constraints only the early parts of this was implemented, with the rest mocked up. However, much like RVIZ, it would display values of certain ROS2 topics, and also publish to some. We believe this would not be too technically challenging to implement. This UI would be important to reduce the amount of obscure ROS2 commands being typed in terminal by any user to an absolute minimum. We believe this would improve the usability of the system significantly.

2.2.2. App User interface - ClyDe Companion

We have improved the appearance of the app since the previous demo. Specifically, we conducted analysis on similar apps such as SeatEd to inform an adjustment of our choice of color, font size, etc. Firstly, the interface layout of the application has been redesigned to make the screen more harmonious. Secondly, we choose royal-blue as our application's main colour, which is consistent with our design philosophy and logo. As well as, this colour is more acceptable for user's eyes. Next, for each app's button, we have selected different small emoji that corresponding to the button activity, that make our app more visually interesting and easy to understand at each step. Lastly, we added clicking animations to each button. The user will feel like they are clicking a real button. With the new appearance of the app, we deployed a survey to check how people react with the app. More details are shown in a Usability Analysis section.

2.3. Software

2.3.1. Robot State Tracking

As part of the integration work, we discovered that it was important to explicitly keep track of the robot's state. This allows components like the image processing and arm movement to trigger at appropriate times. Currently this is implemented using a ROS2 Int16 topic, where each number represents a discrete state. All nodes that have a state-sensitive operation can subscribe to this topic to ensure that their operation triggers at the right time. After the work is completed, they can publish an updated state(i.e one number higher than the triggering number). This distributed method in not optimal, and given more time we would create more complex controllers that dynamically launch and shut down nodes. However, we found it to be an acceptable compromise given the short time frame, our limited ROS2 knowledge and the fact that we had many other tasks to complete these weeks.

2.3.2. Navigation

Mapping Tutorial:
We produced a tutorial for environment mapping with ClyDe, which can be viewed at the following link: Video.

Integration Issues:
Making the Waffle Pi PROTO with the Pincher x100 attached work with the navigator has been an important goal for some time. This would allow us to show off a more complete system, instead of separating the cleaning and navigation. However, this proved a significant challenge. Examples of problems that arose was the robot simply not navigating, and when it did the LIDAR point cloud representing the walls of a room would rotate with it, quickly confusing the navigator. Significant time and effort was put into this, but it seems adding any extra parts to the base Waffle Pi PROTO caused similar issues. Solutions were searched for quite exhaustively and some progress was made, but in the end the root of the issue was not found.
Error Handling: In order to improve our error handling for navigation, we tried to write subscribers to different nodes in the navigator, to pick up any error messages. This would mean we can have different automated responses for each error. Currently, most errors would be handled by the library staff by moving Clyde to home pose and restarting him, which is not ideal. In the future we could continue the work we started, but also get a deeper understanding of the navigator, allowing us to deal with the errors' root cause.

2.3.3. App

As mentioned in last demo, When a user chooses to check in a table, the user will scan the QR code on the table and choose the duration that they wish to stay.
For this demo, we have implemented a timer function for our cleaning booking system. The timer is implemented based on the database. As a result of successfully scanning, the app will use the current time and the booked time slot to estimate the expiration time of the table, then update the duration field of the database with that value. In the background, a program will keep accessing the database and check if there is any table being timeout every 10 minutes.
We have chosen this approach because implementing the timer function based on the app itself have a big limitation. If the user quit the app, the timer will stop.

3. Evaluation

Usability Analysis:
In the last demo we conducted usability analysis based only on self-reflection. For this demo we decided to conduct an experiment with candidates who are from outside our group to evaluate our design.

The experiment - which involved a questionnaire - strictly adhered to all the contents from PIS form and was given the approval from the ethical expert group.

The candidates confirmed they are all University students who have been to any university library before. The vast majority of them are studying in the UK so the samples can only represent the attitude of British university students.

The link to the questionnaire stayed valid for three days from 23/03/2021 to 25/03/2021. We received 24 complete samples in total. However, we removed those that did not spend more than two minutes, as this shows they did not properly consider their answers. Therefore, after filtering the response, the total number of valid and complete samples is 19. There are seven multiple-choice questions and two short answer questions. The questions include various aspects from layout to logic. It is quite hard to present all the data in the report. For more details of the visualised data, please go and see the demo video. In the report we will only demonstrate the way we visualise the data we gathered.

For multiple-choice questions, we used break-down bar charts to visualise the data. As shown in the figure below, we use the area of the rectangle to represent the proportion of choices in the data and mark the sections that account for the largest share. The general impression of this app is

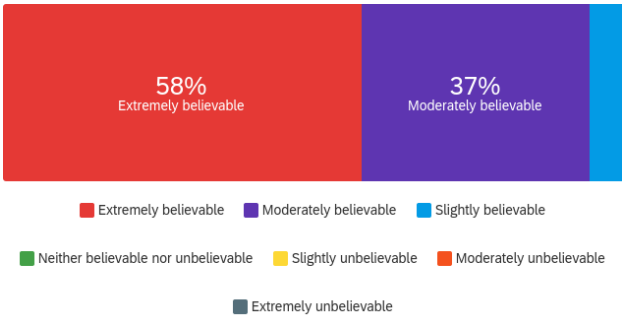

Figure 1. Break-down Bar Chart(Example)

quite positive. 90% of the candidates give general positive feedback to the app. 95% of the candidates think this app is believable and reliable, meaning that there will be few difficulties when they use this app. 95% of the candidates also like the idea and find it relevant to their life. The app may lack some innovation; only 53% of the candidates think this app is extremely or very unique. Since the app remains as a prototype currently, only 53% of the candidates find the layout very appealing with an additional 32% who think it is somewhat appealing.

Overall, it can be concluded that the business idea of the cleaning robot and accompanying app proves to be very successful.

4. Introspection and Reflection

We have added this section to discuss how we believe we performed throughout the development of this project. Since this demo had less technical work and more time spent looking at the direction of the project, we believe it is appropriate to go over these topics in order to demonstrate what we have focusing on this demo. We were inspired to include this section after a meeting with our client.

4.1. Successful Aspects of the Project

Organisation of team

- Team Cohesion - Everybody got along well, with no interpersonal conflicts. People reached out when they needed help with their part, meaning the state of our project was transparent to all members at all times. The extensive use of Discord meant that we were able to neatly separate the project into different tasks and teams.
- Team Structure - Our team structure was adaptive. Overall, we had a portion of our members strictly working on independent tasks such as the image processing and app, while the rest worked on the more dependent tasks. The structure was fluid for those working on the dependent tasks, as information had to be shared amongst everyone in order to keep the design process organized.
- Sticking to the original Gantt Chart - Overall, we ended up working as the Gantt chart suggested we would. The only notable deviations being work on the arm starting early as well as more time allocated on the last demo for investigating the cohesion of our robot rather than integrating it.
- Use of version control - Github has been essential for organising different parts of our robot, allowing parallel development from a shared code base.

Learning new software
Our app team learned how to use android studio with no prior experience in a couple weeks which is an amazing achievement.

Sticking to the scope of the project
We managed to implement all the core features that we looked to have from the beginning of the semester. While our implementation lacks some cohesiveness due to issues with integration, we believe we have enough working parts to prove our concept. The only big thing we would have liked to have done is proper testing on our cleaning methods to guarantee they worked.

Using ROS2
A big reason we decided to use ROS2 is that it translates well to an actual physical project. Since real robots use ROS to work, it meant that most of our code would in theory be portable over to a physical design. Getting our implementation as close as possible to the real thing is something that we wanted to achieve, and working with ROS2 served to make our software more realistic. It also with provided an immense amount of valuable experience. ROS2 also provided us with a wealth of pre-written libraries, like the navigator.

Use of Technician Time

We liberally used technician time, particularly at the start of the semester. This allowed us to quickly gain valuable insight of what would be feasible and what wouldn't.

4.2. Shortcomings and Attempted Solutions

Problem: Instability and documentation issues.
Getting ROS2 to work on everyone's systems proved to be a setback. Identical installations on new Virtual Machines produced different errors. Even at the end of the project this is causing errors for some team members. Parts of our issues stem from there being little documentation on ROS2 as a whole, but also on Webots and how the two interact. We believe this issue will have to be rectified before ROS2 and its Webots integration can be considered accessible to beginners.
Unfortunately, this was inevitable due to the necessity of simulating the robot and so there were no direct solutions to this. We did however have an error channel on our Siscord server to efficiently help anyone who was having compatibility issues, and this helped us solve many.
In hindsight, we believe we would have preferred to go with Gazebo over Webots as it provides a better platform to implement in-depth features at higher efficiency. Gazebo also has far more examples and documentation. Had we been able to work directly with the robot we predict that integration would have been more seamless.
Problem: Navigator being unreliable.
A large blocker for the navigation team was the fact that the navigator packages seem to be inherently unreliable. Much like ROS2 as a whole, it seemed to be more stable on some systems than others. This contributed to the issues with integrating the robot arm, as investigating them were made difficult. Attempted Solution: The team investigated into the source code of the navigator and tried to reinstall to no avail.
Problem: Inability to perform some important qualitative analysis.
There are aspects of the robot that we were not able to test since we were not working with a physical product. An example of this is the cleaning motion of the robot. While we know it is feasible for a robot arm to carry out the motions to properly disinfect a surface according to UK Government guidelines, we were not able to test if the way we programmed the arm nor the cleaning utensil we used is adequate. There is nothing we can do about this, however it is true that the arm and cleaning motion is something that can easily be modified for future versions of ClyDe and would not require a complete overhaul as to render the project unusable.

4.3. Future Direction of ClyDe

Scaled up version.
Currently, we are using the Robotis Waffle Pi as the base for ClyDe. In reality, another base would have to be used since the Waffle Pi is too short to house an arm that could reach on top of a table. This base could be custom built (i.e 3D Printed) since it would not have to carry particularly heavy loads.
More user friendly App.
Our app works, however it is not as feature complete as we would have liked it to be for the full product. Given time, it would be simple to implement many important features that would improve user experience, such as: Text-to-Speech for the visually impaired, ensuring existing screen-readers work, colourblind settings and ensuring font readability. While explicitly made for the Web, the Web Content Accessibility Guidelines(WCAG) 2.1 will be a useful reference for achieving accessibility for a mobile app as well. (W3C, 2018)
Web-app for staff to interface with ClyDe.
As it stands, to start and stop ClyDe, the staff would have to use the ClyDe Controller which is the Raspberry Pi we send out with the robot. While we might not be able to completely get rid of the Pi since it runs the navigation package, we could introduce a web-app (or an addition to the current app) which could allow the staff to start and stop ClyDe as well as add desks individually from their computer or phone. This would eliminate the need to interact with the Pi, and means it just needs to be left running to operate the navigator.
We believe this feature would be straight-forward to implement since we already have the framework set up through the database for the booking system. Additionally ROS2 has packages to interact with the Web.
General Hardware improvements.
Since we were not able to create a physical robot, there are naturally going to be some flaws in our design that will come to light once it is actually built. We estimate that these kind of issues will not require a complete overhaul of the design, and so this point should not be much of an issue, particularly considering that our design is a prototype.

5. Budget

Currently the robot consists of the following parts.

Part	Cost(£)
TurtleBot 3 Waffle Pi	1,276
PincherX 100 Robotic Arm	505
Zeee Lipo Battery	60
Raspberry Pi 4B	30
C270 Logitech USB Webcam	25
DFRobot non-contact water sensor:	10
6-12V R385 DC Diaphragm Pump	5
HDMI cable	4
Total	1,915

Technician Time:
Less than 1hr. Technicians have not been needed much for this demo.

6. Video

Please follow the link below to watch our demo video: Demo Video

References

W3C. Web content accessibility guidelines (wcag) 2.1, 2018. URL https://www.w3.org/TR/WCAG21/.