School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

LUCIE the Robot Waitress

Paul Brown

Submitted in accordance with the requirements for the degree of

MSc Advanced Computer Science

2016/2017

The candidate confirms that the following have been submitted:

Items	**Format**	**Recipient(s) and Date**
Deliverables 1, 2, 3	Report	SSO (13/09/17)
Deliverable 4	Software code	Supervisor, assessor (13/09/17)
Deliverable 5	User manuals: READMEs	Supervisor (13/09/17)

Type of Project: Exploratory Software (ESw)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)_____

Summary

This project aims to explore how and if the robot known as LUCIE might be capable of working as a waitress. The project began with significant constraints and set-backs, LUCIE is a robot with no limbs and had recently undergone major refurbishment, which required a fresh setup of underlying operating system functionality. However, LUCIE is also a STRANDS robot, which includes significant development in safe navigation in human environments.

A waitress can work within three main environments:

- Formal restaurant, where they will take orders, deliver food and handle payments at the table.
- Informal café, where ordering and payment is at a counter, and only food is delivered to tables.
- Hors d'oeuvres, where a waitress wonders amongst a crowd distributing food.

LUCIE was precluded from handling money, however the project attempted to provide sufficient functionality to allow all other tasks. By making use of STRANDS' topological navigation, LUCIE was able to safely navigate and could be sent to specific tables. However, the navigation can be overly cautious and fail to complete journeys or it can be too blasé and crash into furniture, depending on the configuration. The method of travelling to specific points on a map also proved unsuitable for use in a situation where LUCIE is serving hors d'oeuvres; LUCIE frequently failed to reach a destination due to people occupying it. Environment proved crucial to the success of LUCIE's navigation around furniture, for best results the pathways should be wide enough for LUCIE to spin and furniture should be immovable and substantial at the base so that they can be mapped effectively. Steps down must be avoided, and steps up should also be avoided.

As well as navigation, LUCIE also required a user-interface to interact with both customers and staff. One was developed that took advantage of the touch-screen. This UI proved simple to use for both customers and staff. Customers were able to place and receive orders with no instruction. Staff were able to easily update the menu whilst the application automatically updated the underlying functionality, allowing new items to be ordered. The integration of navigation into the UI deliberately blocks to avoid customers placing orders when LUCIE is between WayPoints. While this may not be in issue in an environment with tables, customers were observed chasing LUCIE, trying to order as it was in motion at a standing event.

With careful planning and consideration for the environment, it would not take a significant amount of work to employ LUCIE as a delivery tool within restaurant or informal café environments. However, the underlying navigational technology would require adaptation to allow LUCIE to work efficiently in an environment serving hors d'oeuvres. The two major areas for further development are to recognise when a human requires LUCIE's attention, navigating to that human, as well as to enable humans to order from LUCIE at any point in time, not just at pre-defined WayPoints.

Acknowledgements

The author would like to acknowledge the significant development and research undertaken by the STRANDS team, particularly those who created topological navigation.
Within the University of Leeds, I would also like to thank Paul Duckworth and Francesco Foglino, who introduced me to LUCIE and STRANDS, pointed me towards useful resources, as well as worked with me on the restoration of LUCIE's existing functionality after the refurbishment.
Thanks also to the staff of the Waterside Café for allowing me to repeatedly drive a robot around their establishment during business hours, as well as the Computer Science department at University of Leeds for allowing me to test LUCIE during an open-day event. Your trust in my development abilities is appreciated.

Table of Contents

Summary....i
Acknowledgements....ii
1 Planning and Project Management....1
1.1 Project Conclusion....3
2 Introduction: LUCIE As A Waitress....5
2.1 Context....5
2.2 Determining Expected Behaviour For LUCIE....6
2.3 Self-imposed Constraints....9
2.4 Initial State Of LUCIE....9
3 Navigation....10
3.1 Routing....10
3.1.1 Further Routing Variations....11
3.2 Determining Delivery Success....12
3.3 Navigation Options....12
3.4 Topological Map Navigation....14
3.5 Navigation Processes And Architecture....16
3.5.1 Robot Architecture....16
3.5.2 Topological Map Maker....17
3.5.3 LUCIE Start Navigation....17
3.6 STRANDS Topological Navigation Configuration....18
3.7 Testing LUCIE's Navigation....18
3.7.1 Testing In The Longroom....19
3.7.2 Testing In The Physics Café....21
3.7.3 Testing In Waterside Café....24
3.7.3.1 Waterside Café Benches....25
3.7.3.2 Waterside Café Step....26
3.7.3.3 Waterside Café Bar....27
3.7.3.4 Waterside Café Obstacles Mitigation....28
3.8 Navigation Evaluation....28
4 UI....29

4.1 Existing STRANDS UIs....29
4.1.1 strands_ui....29
4.1.2 terminal_info_gui....29
4.1.3 marathon_gui....30
4.1.4 Summary Of Existing UIs....30
4.2 Expected Interactions....31
4.2.1 Normal Usage....31
4.2.2 Additional Interactions....32
4.2.2.1 Recovery Interaction....32
4.2.2.2 Login Interaction....33
4.2.2.3 Social Interaction....33
4.3 The URLs....33
4.4 Visual Design....33
4.5 Structural Design....34
4.5.1 Navigation....34
4.5.2 Orders And Logging....35
4.6 Integrating Navigation....35
4.6.1 Asynchronous, Parallel, Or Concurrent Navigation....36
4.6.2 Communication Protocols....37
4.7 Implementation....38
4.8 Testing For Expected Functionality....40
4.8.1 Navigation Testing....40
4.8.2 UI Testing....40
4.8.3 Simple To Setup And Use....40
4.9 UI Evaluation....41
5 Final Testing....42
5.1 Lack Of Interest....42
5.2 Moved Furniture....42
5.3 High Density Of People....43
5.4 A Failed Delivery....44
5.5 Missing Biscuits....44
5.6 Evaluation Of Final Testing....45
6 Conclusion....46
7 References....47

8 Appendix A: External Materials....49
8.1 Robotics....49
8.2 Web Imports And Requirements....49
8.3 Web Included Within Repository....49
9 Appendix B: How ethical issues were addressed....50
10 Appendix C: Product READMEs....51
10.1 Topological Map Maker....51
10.1.1 Tmux useful keys....51
10.1.2 Sessions in Tmux....51
10.1.2.1 Session 0....51
10.1.2.2 Session 1....51
10.1.2.3 Session 2....52
10.1.2.4 Session 3....52
10.1.2.5 Session 4....52
10.1.2.6 Session 5....52
10.1.2.7 Making the map....52
10.1.2.8 Session 6....52
10.1.2.9 Session 7....52
10.1.3 Wrapping up....53
10.2 Robot Waitress: Navigation....53
10.2.1 Tmux useful keys....54
10.2.2 Sessions in Tmux....54
10.2.2.1 Session 0....54
10.2.2.2 Session 1....54
10.2.2.3 Session 2....54
10.2.2.4 Session 3....54
10.2.2.5 Session 4....54
10.2.2.6 Session 5....55
10.2.2.7 Session 6....55
10.2.3 Exiting....55
10.3 Robot Waitress UI....55
10.3.1 Configuration Options....55

1 Planning and Project Management

A successful robot waitress requires integration of navigation and user-interface, which allows the project to be sub-divided into three parts: navigation, user-interface and integration. As the project is exploratory it is not possible to predetermine the exact tasks that will be required. Therefore an agile approach is followed, with each part comprising a single sprint:

1. “Get Moving”: Navigation
2. “Talk To Me”: UI
3. “Go Away”: Integration

Each sprint was concluded with a mini-report submitted to the supervisor, which have been combined to create this final report.

Prior to beginning a sprint, a list of “epics” are chosen, these describe what should be achieved in the sprint but not the tasks required to do so. Examples from the three parts include:

1) “LUCIE must be able to safely navigate to a customer”
2) “A customer must be able to choose items to order”
3) “Support staff must be able to direct LUCIE to a customer”

These “epics” are then investigated and further broken down into single tasks. For example, task 2 required designing a webpage for a large touchscreen followed by writing the HTML, CSS, JavaScript and Python to accomplish a successful order. The granularity chosen for individual tasks was at the “develop webpage” level, as this is enough information to specify what is required and completes a single product, while still leaving enough flexibility to allow for exploration. Individual mini-tasks and progress were then logged in notes under the main task’s card.

To monitor and manage the project, a free online software called “Task Junction” was used, as it also allowed the Supervisor access to monitor progress. This software provided a simple Kanban Board (Fig 1) and Burndown Chart (Fig 2) for organisation and progress monitoring. Notes could be attached to tasks, which were very useful to guide the report writing, track issues and development.

Task durations were estimated relative to each other but the number is not attached to any specific unit. This provided enough information to set the pace of work without generating unrealistic and poorly informed deadlines. The estimates are seen in Fig 1 under “Todo” and are used to generate the Burndown Chart in Fig 2. This also allowed the addition of new tasks without breaking the plan, provided the pace of work doesn’t become unmanageable. A new task was added in Fig 1 on approximately the 7th July.

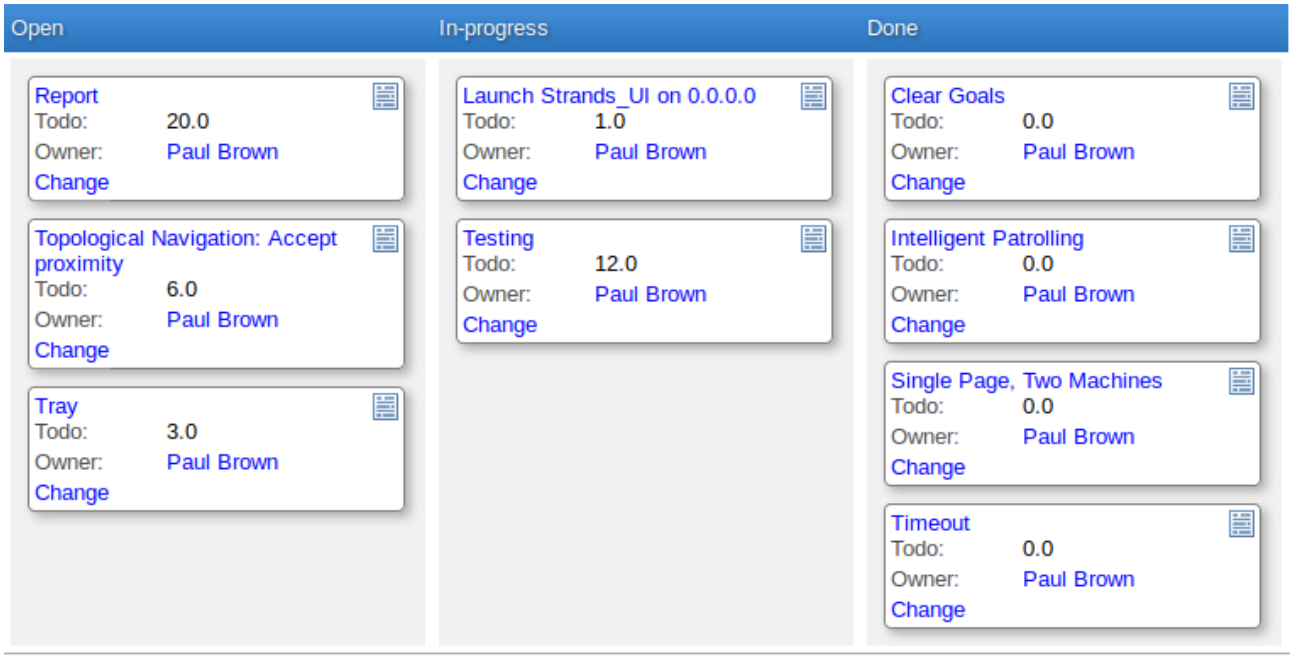


Fig 1: Task Junction Kanban Board

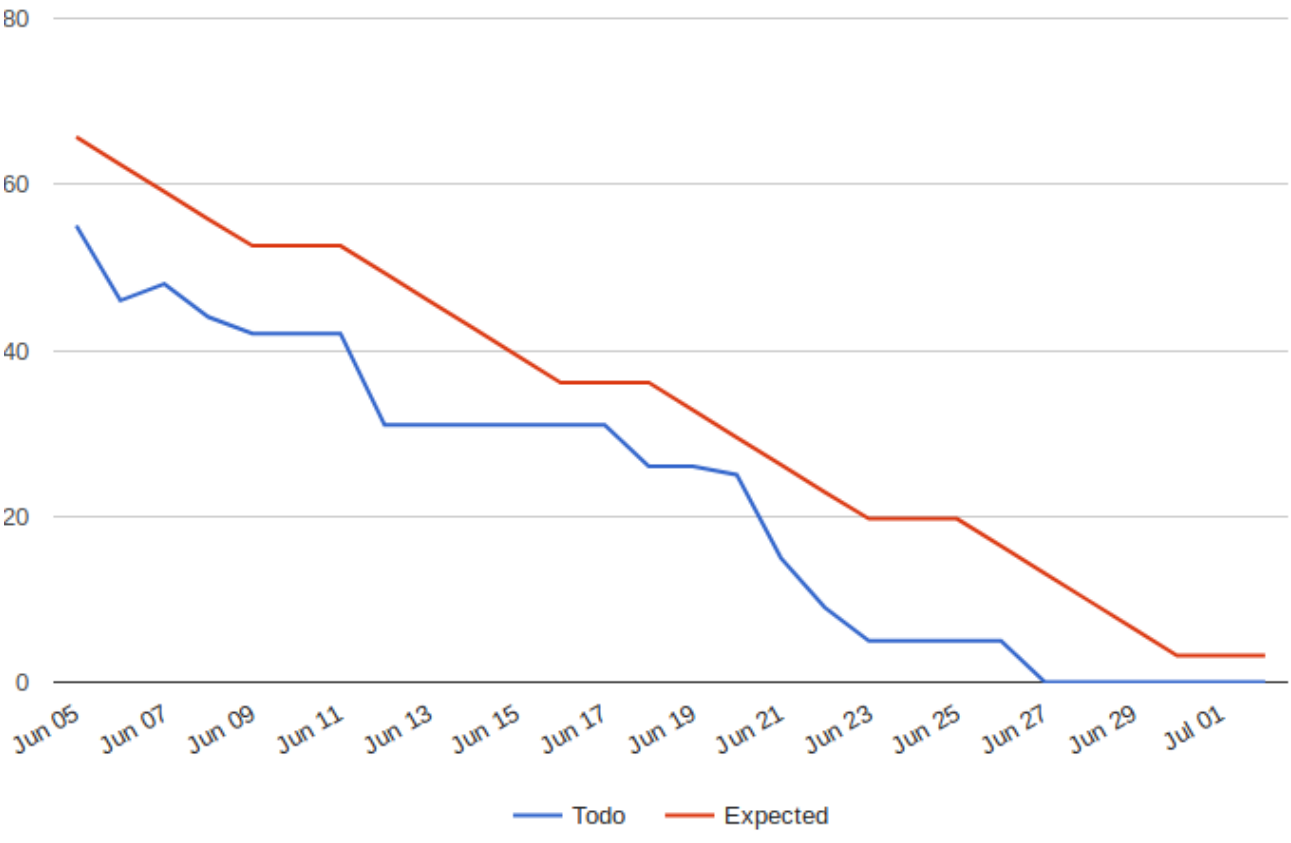


Fig 2: Task Junction Burndown Chart

The flexibility afforded by this approach proved invaluable. The plan could be adjusted after topics were researched and after partially completed tasks were evaluated. During the navigation sprint it became apparent that a simpler way for creating maps was required. This approach allowed the addition of the

creation of a Tmux script to achieve this as well as the omission of creating a new navigational tool as an existing one was discovered.

Although this approach was flexible, some difficulties were still encountered. Having completed the first sprint a week early, the second two, creating the UI and integrating the navigation, were overconfidently merged into one. The intention was to leave the final sprint free to tackle an extra topic.

The initial plan was to create a minimal UI and test it with the navigation before fully developing the UI, however the UI was almost completely developed before testing with navigation. At this point the difficulties with integrating the navigation arose, such as no push notifications and navigation blocking the UI. This required a re-write from AJAX to Web Sockets, because the UI was almost completely developed, this was a much bigger job than it would have been if the original plan was followed. Therefore, the final sprint was spent on re-writing the UI, bug-fixing, testing and evaluation. This completed the planned project.

The code developed was managed using version control, with each project contained within its own repository. In the case of the UI, two branches were also developed. A university provided account was intended to be used, however this wasn't created until after a significant amount of development had been undertaken. Therefore GitHub has been used throughout the project, as version control was required before it was provided. The code for submission has been uploaded to the university provided version control without the commit history to avoid conflict.

1.1 Project Conclusion

The agile approach was very useful for an exploratory project as it is easy to adjust as more information becomes available. It provided ample opportunity for feedback, with weekly meetings, demonstrations and submitted reports. I believe this feedback loop could have been exploited more, however the methodology employed does not match the university requirements. Therefore, I held myself accountable to submitting the reports and received feedback inline with the university guidelines of weekly meetings and a half-way presentation.

Time was wasted by ambitiously merging the last two sprints. By merging the sprints, nothing would have been gained. Therefore, in future projects, it would be better to complete a very short sprint before beginning a new one if they are logically ordered as these two were.

The project management software employed was useful, but not ideal. It wasn't responsive or flexible and at times failed to produce charts. In the past I've used several other similar tools and I've not found one that works adequately well for me to stick with it. I doubt I will use this website again; I believe I could develop a more suitable tool myself.

I was fortunate that the university provided a wide variety of testing environments. The open-day test was added near the end of the project, in an ideal scenario the parameters for this test would have been known at

the start of the project, which would allow the project to be developed for the tested tasks. However, a general approach was followed, which has allowed investigation along a greater breadth and provided more clarity to the direction the university may wish to take further iterations.

2 Introduction: LUCIE As A Waitress

LUCIE is part of STRANDS (Spatio-Temporal Representation and Activities for Cognitive Control in Long-Term Scenarios) (Strands.acin.tuwien.ac.at, 2017), which runs on ROS (Robot Operating System) (Ros.org, 2017) with additional open source packages. The available functionality from the combination of ROS and STRANDS provides a number of options for how to achieve navigation. To determine which approach to implement, a description of the behaviour LUCIE is expected to undertake must first be formalised. From this description the options are compared, one of which is implemented, documented, tested and evaluated. As this project is exploratory, weaknesses discovered are highlighted to guide further work.

2.1 Context

Robotic technology is on the cusp of being advanced enough for satisfactory use within the service industry due to significant research and application in a number of areas (Pinillos et al., 2016), such as interaction, computer vision and obstacle avoidance. Applications such as Amazon's Alexa (https://developer.amazon.com/alexa) demonstrates the potential for natural interactions, whereas the miles driven by Google's self-driving car (https://waymo.com/) demonstrates the potential for safe navigation. A robotic waitress will need to combine these two fields to provide a satisfactory service. The operational model of the robotic waitress can also be integrated into other use-cases, such as domestic care.

Robots that are in existing service positions providing deliveries are still a novelty and thus are favourably reviewed by the consumers who encounter them. However, there is also a lot of fear in the media that robots will take jobs away from low-skilled service workers (Harford, 2017), a theme that spans over 100 years (Wilde, 1915). Some success in allaying these fears has been demonstrated through Baxter, a robot designed to work with humans rather than replacing them (Harford, 2017). In the same manner, the robotic waitress will work with humans as a simple delivery system, allowing their human companion more time to dedicate to serving their customers other needs.

Similar projects have been undertaken; the Roomba has seen commercial success as it made use of navigation to work in service as a Vacuum Cleaner, however it lacks sophisticated interaction with humans (Forlizzi & DiSalvo, 2006). Since then, robots have been deployed in making deliveries in hotels (Pinillos et al., 2016) and in restaurants as wait-staff with some success (Rusu, Gerkey and Beetz, 2008) (Tzou and Su, 2009).

2.2 Determining Expected Behaviour For LUCIE

Before considering how LUCIE should behave as a waitress it is useful to describe how a waiter or waitress fulfils their role. Fig 3 represents a simplified view of the tasks undertaken when serving a table, created from the author's own experiences working in the service industry.

The diagram disguises periods of non-activity, such as waiting for food cooking, or customers to finish eating and talking. The breaks between tasks allow wait-staff to serve tables in parallel. The aim is to relieve the wait-staff of tasks by having LUCIE undertake them.

Initially LUCIE will be capable of making a delivery when told where to go by a human user, then by logging an order location to return to. This potentially allows the activity shown in Fig 4. In this diagram all delivery tasks are shifted to LUCIE. For LUCIE to accomplish a delivery task will require navigation to the destination as well as a method for determining successful completion of the delivery.

For a human-user to give LUCIE a destination, and for customers to place orders, a user-interface will be required. Ordinarily people communicate with wait-staff verbally, although this would be possible with LUCIE by integrating something such as Google's AIY, a complete VUI system would be a project unto-itself. LUCIE also does not have a microphone, however there is an onboard touchscreen, which can be used for interactions.

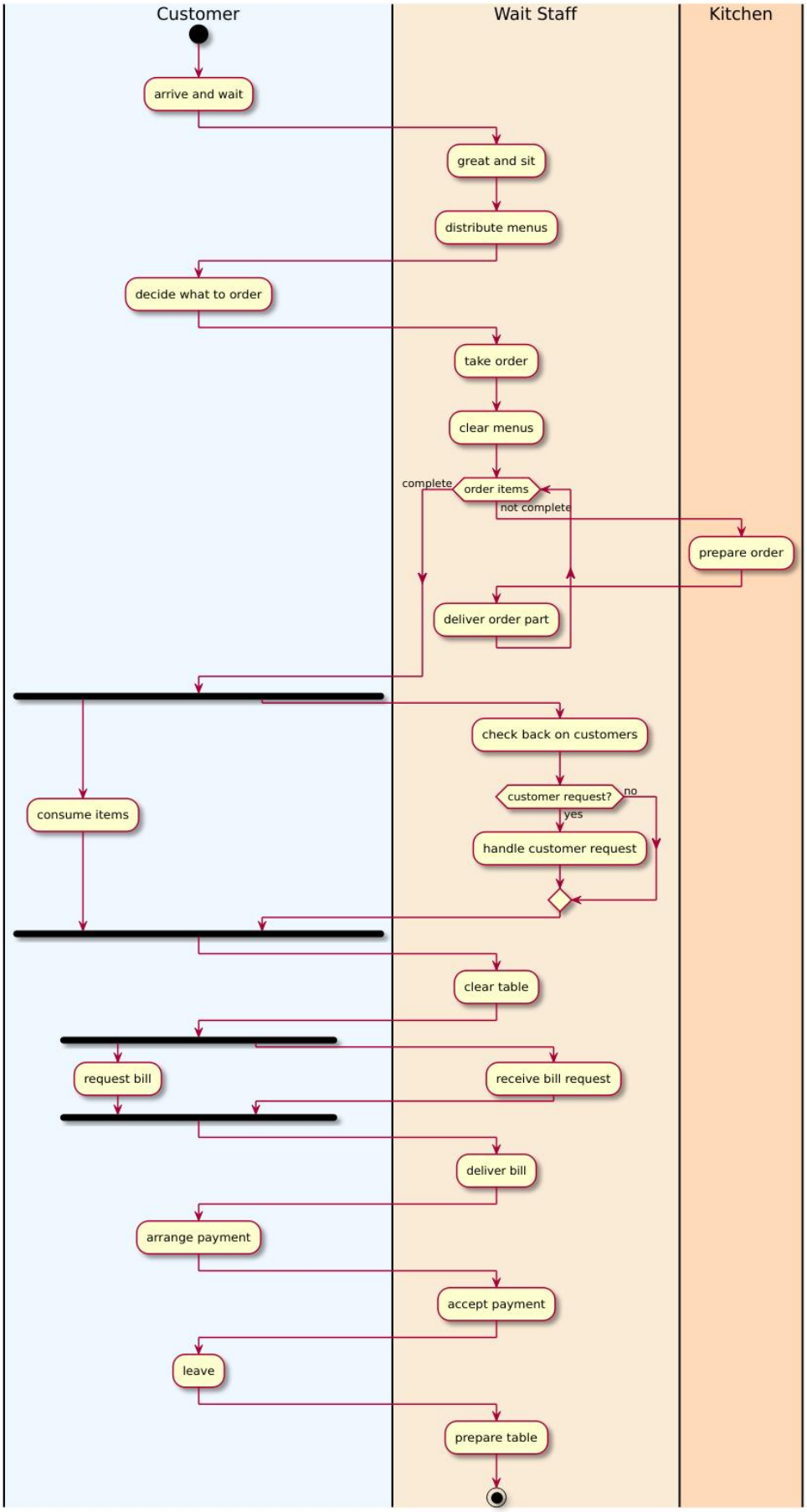


Fig 3: UML Activity diagram for a Waitress

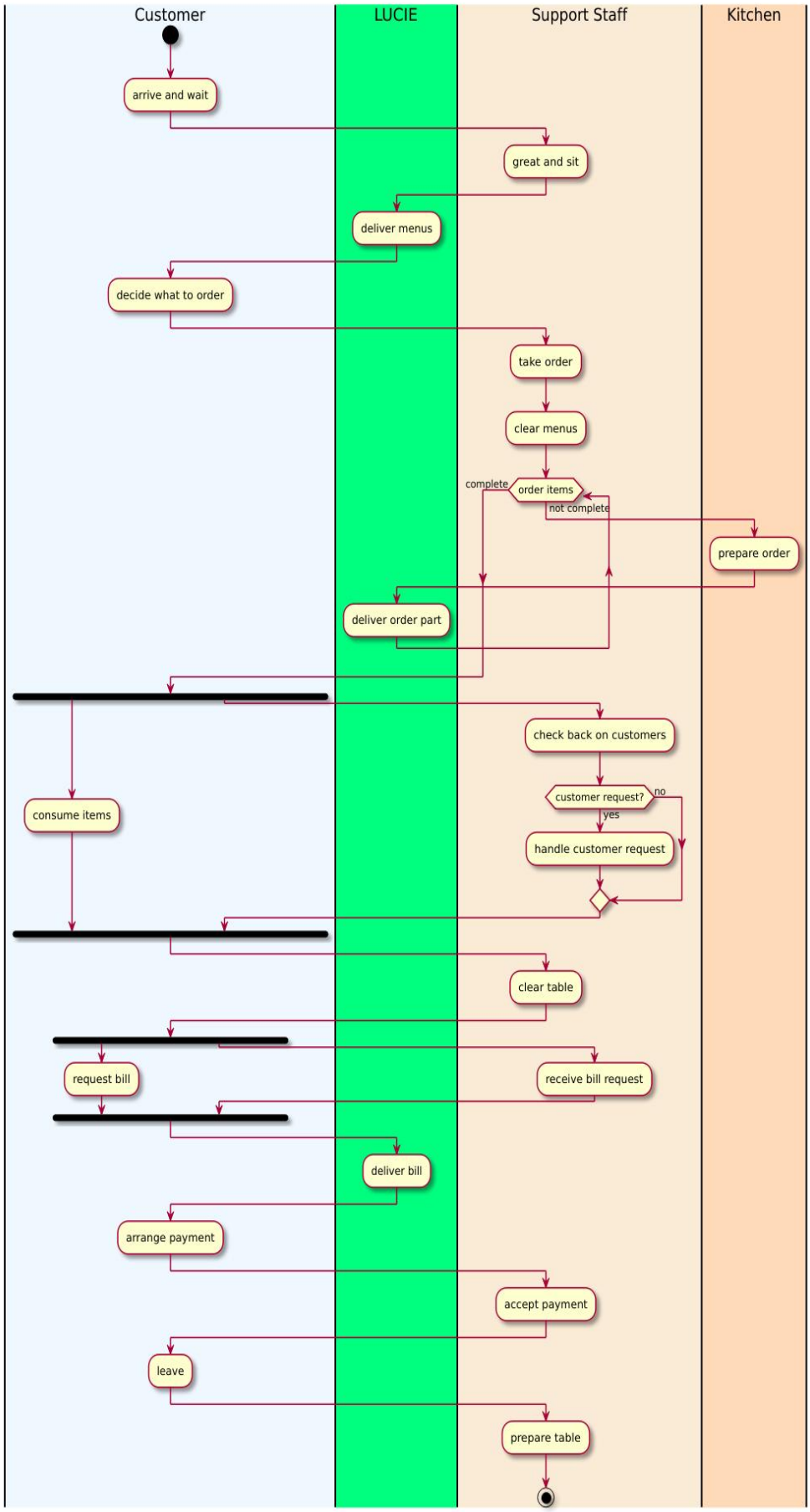


Fig 4: UML Activity diagram with LUCIE

2.3 Self-imposed Constraints

There are a limited number of people who are familiar with LUCIE's systems and who will be available to run LUCIE as a waitress. Therefore, to maximise the usability, the procedure from introducing LUCIE to a new space through to running the waitressing program must be as easy as possible. LUCIE will be operated by Computer Scientists, therefore the command-line interface need not be avoided, but the use of unique ROS/STRANDS programs, such as RoboMongo will be avoided.

Rviz is a unique STRANDS program that is useful for visualisation and monitoring, although it is difficult for a user to customise. Therefore a pre-customised version may be launched that will provide a user who is not familiar with it all the tools they'll require to see what LUCIE is doing and planning for navigation. Interaction with the program beyond looking at is will be avoided.

To make LUCIE easy to use, the required procedures will be packaged in Tmux scripts. This will allow a user to run pre-written commands as they require, so they do not need to learn them. Furthermore, they can observe the output of each script, which is useful for debugging. A user will not be expected to manually update maps or database entries, although this is possible and can be used to improve navigation.

There exists the possibility that the software written will be used with another robot, which is not part of the STRANDS project. Therefore the waitressing program will be developed with minimal dependencies on STRANDS systems. It should be as simple as changing or replacing a navigation class or node to work with that robot's navigation system.

2.4 Initial State Of LUCIE

LUCIE had just been refurbished prior to commencement of the project. Significant set-backs were encountered in getting the basic and prior functionality working. It was discovered after initial testing that LUCIE's head-camera movement was disengaged, which may have effected early results that were then excluded. A team effort was required to re-implement the shared filesystem and networking between the three machines, the LAN, drivers for components, changing file-system paths in scripts to variables to fix software, as well as both determining then setting correct environment variables for STRANDS/ROS.

3 Navigation

LUCIE's navigation system needs to be capable of going to a pre-determined destination in a safe-manner, without worrying humans or colliding with furniture. LUCIE should also avoid bringing harm to itself, although it is expected to be operated in relatively safe environments, it should avoid damaging collisions.

3.1 Routing

Two contenders for LUCIE's default navigation behaviour were considered: Spoke-and-Hub or Point-to-Point. Spoke-and-Hub is a paradigm where LUCIE would return to the Hub after each delivery (Halskau, 2014). In Fig 5, LUCIE's route following Spoke-and-Hub is:

1. Kitchen
2. Red delivery route
3. Kitchen
4. Blue delivery route
5. Kitchen

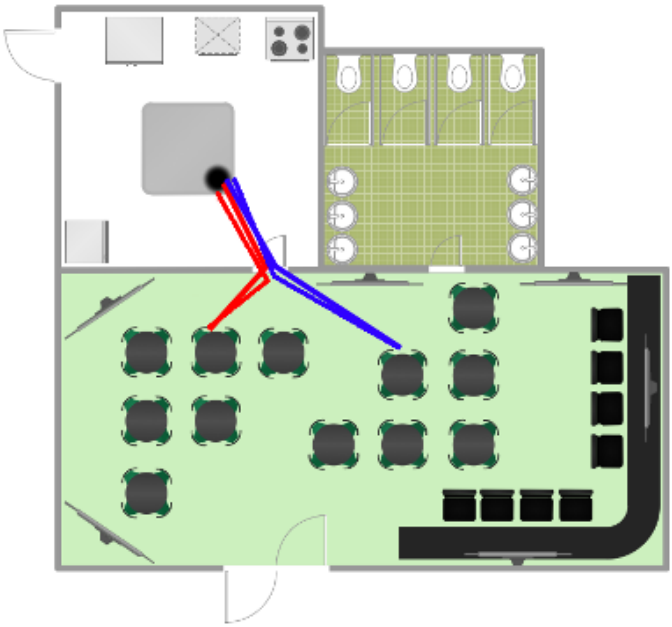


Fig 5: Sport bar floor plan (Conceptdraw.com, 2017) with Spoke-and-Hub route overlaid.

Point-to-Point would allow LUCIE to carry multiple deliveries, dropping off their contents to each respective destination before returning to the Hub (Yin Zhang et al., 2005). This is shown in Fig 6, where LUCIE's route is:

1. Kitchen
2. Red delivery route
3. Blue delivery route
4. Kitchen

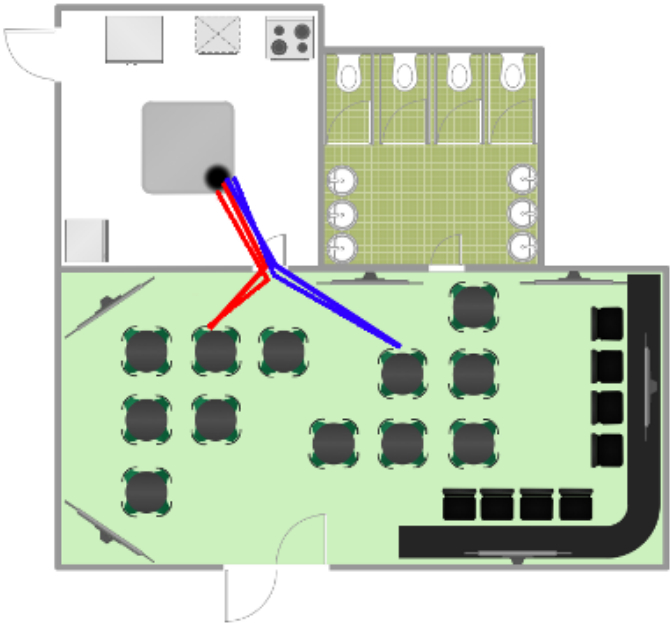


Fig 6: Sport bar floor plan (Conceptdraw.com, 2017) with Point-to-Point route overlaid.

The Point-to-Point routing can be expected to be more efficient as it requires less trips to the kitchen. However there are constraints on the use of this method:

- The delivery vehicle needs to be capable of carrying the items required for multiple deliveries
- The delivery vehicle needs to be capable of segregating the items at delivery in order that the correct items go to the correct places.

LUCIE is not designed as a delivery robot and has no inbuilt hardware for carrying items, this will need to be created and added to LUCIE. Given the constraints this hardware will be in the form of a deep tray, which will require humans to put things in and take things out as LUCIE has no limbs.

Allowing LUCIE to use Point-to-Point routing in this case will introduce issues regarding segregation of items for multiple deliveries. For example, table 2 may not appreciate table 1 handling their food in order to access their own. The container will also be limited in size, constraining LUCIE's ability to carry large quantities of items for multiple deliveries.

Although these issues are not insurmountable, perhaps multiple compartments could be used with a locking mechanism, they also add a level of complexity that goes beyond the scope of this exploration into the use of LUCIE as a waitress. Therefore Spoke-and-Hub routing will be used as the default behaviour, although it may be less efficient in routing, it appears more feasible given the constraints and provides less complexity to the system as a whole.

3.1.1 Further Routing Variations

During evaluating LUCIE's navigational behaviour, other routing paradigms were discovered and explored. At a robotics event LUCIE was sent on a patrol-route, at each point it would wait a while before moving on. This patrolling idea could allow LUCIE to wonder between tables, only heading back to the Hub when a customer places an order. LUCIE would then return to the customer with their delivery before continuing the patrol.

Two versions of this were implemented:

- `PATROL`: LUCIE goes from Waypoint to Waypoint sequentially
- `RANDOM_PATROL`: LUCIE randomly chooses Waypoints to visit without re-visiting any WayPoint until all WayPoints have been visited.

These methods have the same benefits as Spoke-and-Hub but give LUCIE some independence to choose which Waypoint to visit next, relieving support-staff of the task. A user can choose the routing method in `config.py`. In further iterations of this project a more intelligent method of choosing the next destination should be implemented, perhaps using skeletal positioning and facial recognition to determine if a human would like LUCIE's attention.

3.2 Determining Delivery Success

Once LUCIE has arrived at the destination it is expected that the customers will take their items, completing the delivery. At this point LUCIE should return to the Hub, ready to make another delivery quickly. However, LUCIE has no method of determining when the container has been emptied.

The method chosen to determine successful completion also needs to be able to handle unsuccessful completion. It is not enough to simply determine that the items have been taken as this does not indicate that the customer is satisfied with the delivery. LUCIE is the sole point of contact with the customer at this point, therefore if there is an error LUCIE needs to be able to alert staff.

It is tempting to develop a technical solution, such as using image analysis to monitor the container and people removing items. However, any such technical solution would be deterministic and inflexible with incomplete information. Instead it would be better to determine what the customer thinks at this point of contact so that additional service may be offered, perhaps they want Ketchup? No image analysis or sensors could infer this.

The delivery is an interaction with the customer, therefore its success should measured by an interaction. By asking a customer “Do you have everything you need?” LUCIE simultaneously asks if the delivery is complete and provides an additional level of service. If the customer’s response is negative, the interaction can continue to determine the customers intent and to handle it appropriately. This additional service makes LUCIE superior to other novelty food delivery methods, such as conveyor belts, miniature trains or miniature roller-coasters.

There is one caveat to this method, as LUCIE is a robot, people may not respond to the question, therefore a timeout must also be implemented. Furthermore, if LUCIE’s delivery times-out and LUCIE returns with the items, the support staff at the Hub are alerted to an error and can intervene. This requires Spoke-and-Hub navigation or an additional failure behaviour, which would require a technical solution to check the tray is empty.

3.3 Navigation Options

STRANDS provides high-level navigation functionality, which is comprised of components, the structure of which is in Fig 7. This system can be abstracted into 3 layers: action, monitored navigation and topological navigation (Strands.readthedocs.io, 2017). The highest suitable level will be used to avoid repeating already implemented and tested code.

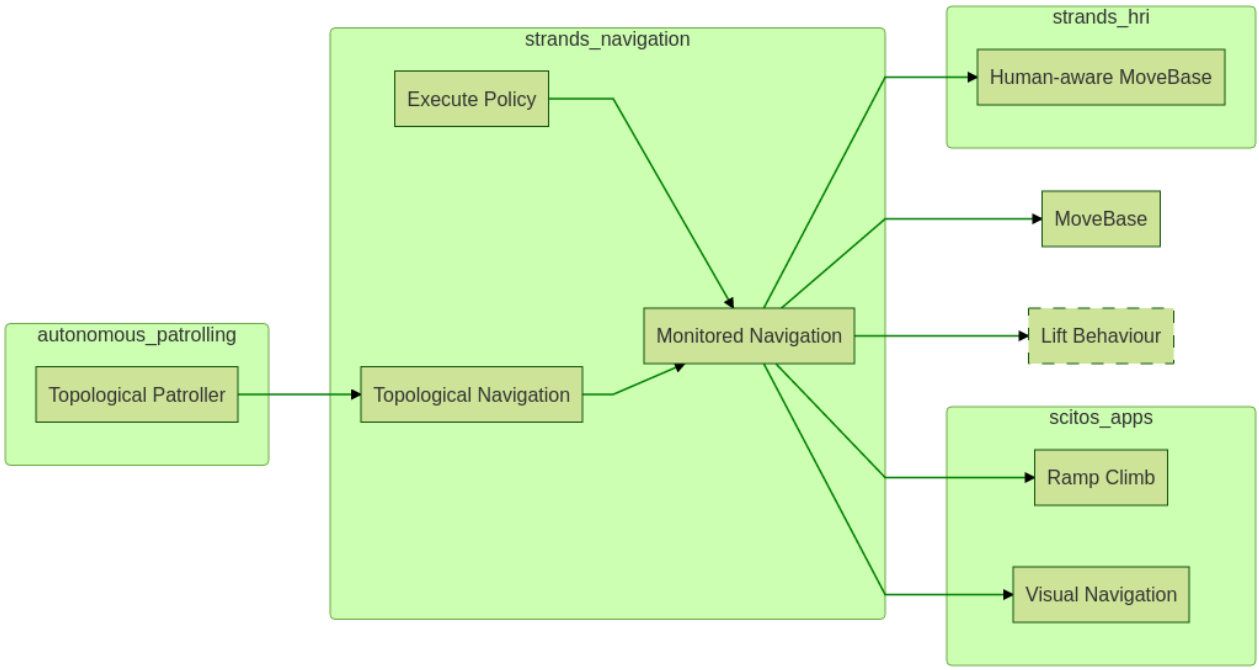


Fig 7: STRANDS navigation (GitHub, 2017)

Each layer in the abstraction is concerned with different functionality:

- **Action**: Concerned with low-level components, such as “MoveBase”
- **Monitored Navigation**: Monitors Action layer, activates specified recovery behaviours in the case of failure.
- **Topological Navigation**: Works with a map to navigate to predetermined WayPoints.

In a restaurant environment each table could be considered a WayPoint, this will allow users to tell LUCIE to go to table 12 by asking LUCIE to go to WayPoint 12. This makes Topological Navigation a feasible and preferential option, as it also inherits inbuilt recovery behaviours.

There are two potential issues that arise from Topological Navigation. The first is the requirement for a map, however this can be created by driving LUCIE around and pressing a button to mark a WayPoint, as described in STRANDS documentation. The second issue is that a table may move slightly as they are often not fixed features, therefore WayPoints are marked in positions that allow for a little movement of furniture.

There is a possibility that LUCIE will arrive near a table and not in an ideal position for a customer to unload their food. In further iterations of this project a second navigation task could be run once LUCIE has arrived at the WayPoint to fine-tune her position.

3.4 Topological Map Navigation

A topological map is built upon a metric map. The metric map consists of an image file (pgm) and a yaml file that describes how to interpret the image file, such as information about scale and colour thresholds. Using this map LUCIE can navigate through the environment.

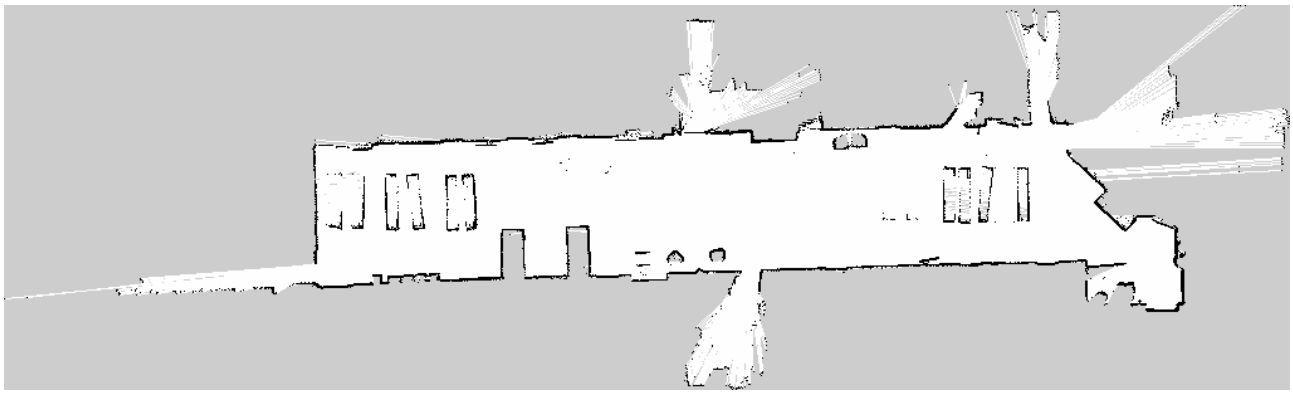


Fig 8: An example metric map image file.

The topological map is the addition of WayPoints and edges aligned with this metric map. The WayPoint is a pose that refers to a position and orientation in the map. An edge describes an action between two WayPoints, for example **move_base** or **human_aware_navigation**. The action on an edge can be viewed using `robomongo`. The topological map is built from a tmap file and is stored in mongodb. A topological map in use is shown in Fig 9. The black on the map is an obstruction from the metric map that cannot be navigated through, the white is clear space from the metric map. Grey denotes areas that have been determined to not be navigable via the current sensor inputs, the cost-map. Multi-coloured spots are where the laser-scan is intersecting obstacles. Green arrows are the WayPoints, the red line is the route planned by LUCIE.

Fig 9: Rviz view of Topological Navigation Map in use.

For LUCIE to navigate from one WayPoint to another, both WayPoints and an edge need to be within the map. Edges can easily be removed using Rviz, however they are much trickier to add using the standard tools. When working with Spoke-and-Hub Routing, edges between tables can be safely removed, but they

are necessary for the patrolling routing methods. Removing edges between tables will force LUCIE to return to the hub to navigate between tables, preventing errors where LUCIE is erroneously sent on a delivery without first collecting food and enforcing Spoke-and-Hub routing.

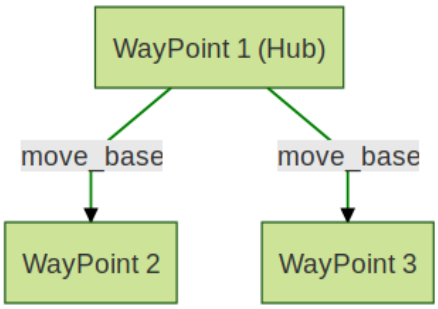


Fig 10: With no edge between Waypoint 2 and WayPoint 3, LUCIE will navigate between them via WayPoint 1

Edges are removed in Rviz by checking the 'EdgeUpdate' box and under 'Update Topic' subscribing to 'topological_map_edges/update'. The edges are shown as red arrows and can be removed with a click. This cannot be undone without additional software or directly updating the database. The names of the WayPoints can also be read in Rviz.

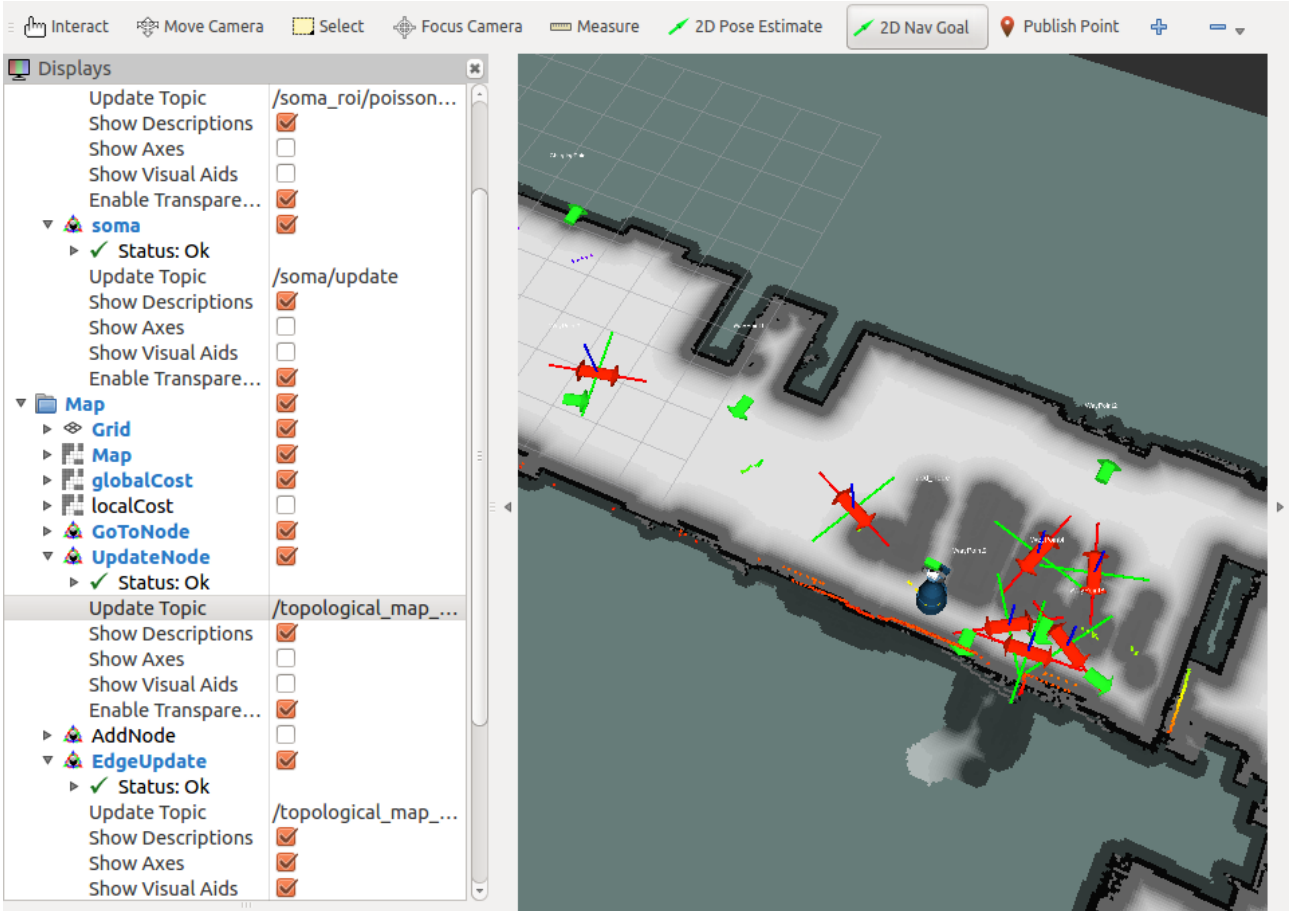


Fig 11: Rviz: the red arrows are edges and are removed with a click.

3.5 Navigation Processes And Architecture

Two products have been created, the first, "Topological Map Maker", enables users to create topological maps without needing a technical understanding of the underlying processes and architecture. The second, “LUCIE Start Navigation” is to simplify the running of all the processes required to bring the robot to the point where a navigation command can be issued.

LUCIE has recently been refurbished, which has also included a change in home directory and user. This has broken some existing scripts that developers had created. As it is expected that this software will be transferred to a new robot, the products have been designed for robustness, making use of inbuilt variables `$HOME` and `$USER` to avoid this issue in the future. However, some assumptions are made about the architecture of the robot and its operating system, therefore some setup on a new robot will still be required.

3.5.1 Robot Architecture

LUCIE contains 3 machines and various input devices. Their current architecture is described in Fig 12.

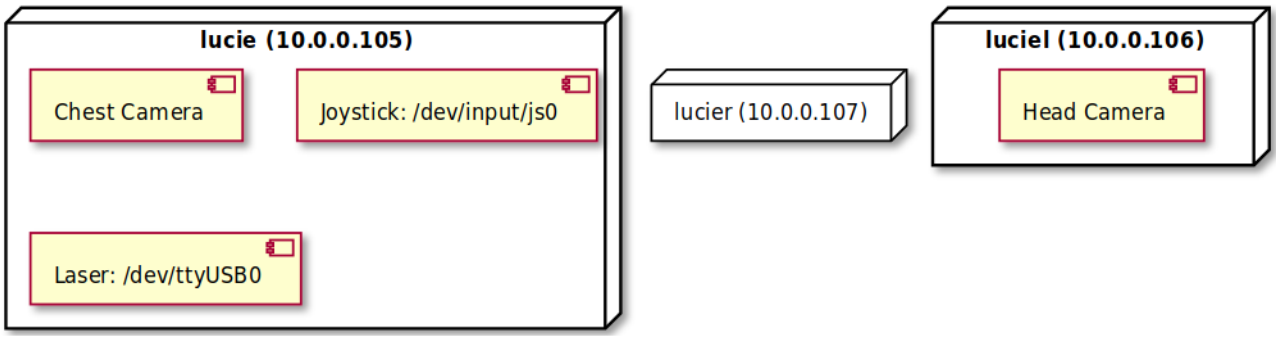


Fig 12: UML Component Diagram of LUCIE's Hardware

For easy communication between machines and to aid readability in code, hostnames are declared in `/etc/hosts` on all three machines. Certificates for SSH have also been created in `~/.ssh/authorized_keys` on respective machines to enable SSH without the need for passwords. This may have led to an error where the machines do not have certificates in `~/.ssh/known_hosts`, appending `export ROS_SSH_UNKNOWN=1` to `~/.bashrc` prevented this error.

The hostnames are used extensively within the products created, to re-use the products in new robots, or if LUCIE's hardware configuration is changed, the hostnames will need to be updated in the Tmux scripts. Within the products, hostnames are set using variables to ease re-use.

3.5.2 Topological Map Maker

The STRANDS project documents how to make a new Topological Map, however the documentation is spread through several resources. In investigating this documentation, features were found that did not work or had incomplete documentation with comments that the documentation "required improvement".

To make the common task of creating a new Topological Map easier, a Tmux script was created that can be followed to produce a useable map. Existing tools were chosen for inclusion based on the goal of running simultaneous metric and topological mapping as well as favouring core tools of STRANDS and ROS. Additional software designed for STRANDS that aids the mapping process exists, however it is expected that this software will be transferred to a new robot, therefore dependencies are kept to a minimum.

Using the tool a user can run the scripts and drive the robot around, pressing a button to mark WayPoints. All necessary files and database entries are created. The documentation for the product is included in the Appendix.

3.5.3 LUCIE Start Navigation

This tool builds upon the Topological Map Maker and was used for testing before the full waitressing program was developed. It is useful as a template for new applications. It has been designed to also use the `name_of_map` variable that was used to create the files to locate them again. This gives the user flexibility

to choose which map they wish to use at runtime. Again, the product is a Tmux script that a user needs to run sequentially, simplifying the process of running all the required processes. The script also makes use of variables for robustness and easy adaptation. The documentation for the product is included in the Appendix.

3.6 STRANDS Topological Navigation Configuration

The topological navigation manager used can be launched with different options regarding the cameras. It can run with a single camera for navigation, either the head-camera or the chest-camera. This mode is sufficient for human-aware-navigation if the furniture can be discovered by the laser-scan, however it fails to see chairs with thin legs. In Fig 13 it can be seen that LUCIE is identifying areas with grey shading that it will avoid while navigating. One of these areas has no black within it, or multi-coloured points, therefore it is being identified via the head-camera only.

For better recognition of furniture, LUCIE should be run with both cameras in the options for the launch command:

```
with_camera:=True camera:=head_xtion with_chest_xtion:=True
```

This improves navigation performance, although it is still not perfect. When run in this mode, if the chest-camera is improperly configured, LUCIE can interpret the floor as unnavigable ground as seen in Fig 14. If this occurs the chest-camera can be configured via an inbuilt program, the result of which is shown in Fig 15. Although Fig 13 and Fig 15 appear quite similar, during testing the navigation with both cameras proved safer.

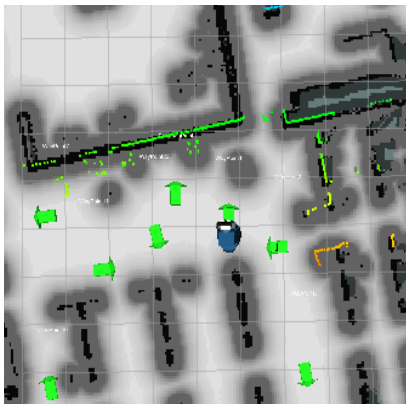


Fig 13: View with no chest-camera.

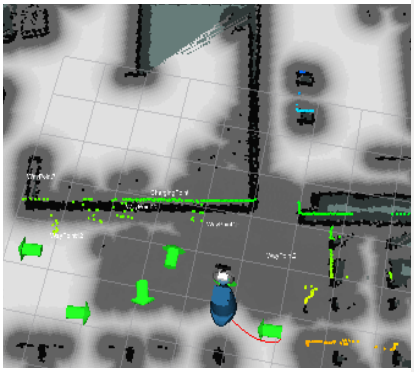


Fig 14: Improperly configured chest-camera

Fig 15: View with configured chest-camera.

3.7 Testing LUCIE's Navigation

The testing consisted of two stages, an initial test at the end of the navigation sprint to ensure LUCIE's navigation was running correctly, followed by testing in three environments with the full waiting software. It

is the final testing that is reviewed here as the initial testing used little more than the inbuilt systems. The three testing environments were:

1. The Longroom: where LUCIE will be deployed during open-days and other events
2. The Physics Café: a closed nearby café with many tables and thin metal legged chairs without much room between them.
3. The Waterside Café: an open café with a variety of furniture, including benches and soft-furnishings as well as a raised platform and bar.

The environments contained enough variety of features to determine LUCIE's strengths and weaknesses.

Fig 16: Longroom

Fig 17: Physics Café

Fig 18: Waterside Café

3.7.1 Testing In The Longroom

The furniture in the Longroom is substantial enough for LUCIE to detect, therefore testing in this area focussed on ensuring correct navigation behaviours, such as recovery and order completion.

ROS provides a tool to record specific logs. To examine the navigation behaviours, the logs from "/waitress_nav", which is the waitressing software node, and "/topological_navigation" were recorded. The logs provide a thorough record, therefore they have been simplified and cleaned up in the table presented below.

This session shows a complete order where WayPoint1 is the Hub and WayPoint17 is the customers location. This is followed by a short random patrol, through WayPoints 28, 10 and 17. At WayPoint17 another order is placed, however navigation failed. LUCIE engaged in the default recovery behaviour, which is to return to WayPoint1, the Hub. LUCIE also asked for help using the inbuilt recovery behaviours, on the second attempt this help was sufficient to allow LUCIE to successfully navigate back to WayPoint1.

The routing sequence from the short random patrol until LUCIE returns to the Hub is illustrated in Fig 19. The visited WayPoints are highlighted in yellow, all of the routes have been combined into one image. LUCIE is choosing sensible, short routes with minor and smooth diversions to avoid furniture along the

route. If a human interrupts the planned route, LUCIE avoids the human but does not show this new route in Rviz.

/waitress_nav	/topological_navigation
[WAITRESS] UI Launched at http://0.0.0.0:5000	
	From WayPoint1 do (human_aware_navigation) to WayPoint21 navigation finished on Tuesday, August 15 2017, at 14:32:10 hours (21/7)
[WAITRESS ORDER] time= 2017-08-15 14:32:18, location= WayPoint21, status= **Open**, items= [(Green Sweet, 0), (Red Sweet, 3), (Blue Sweet, 0)]	
	From WayPoint21 do (human_aware_navigation) to WayPoint1 navigation finished on Tuesday, August 15 2017, at 14:32:34 hours (15/5) From WayPoint1 do (human_aware_navigation) to WayPoint21 navigation finished on Tuesday, August 15 2017, at 14:33:01 hours (21/7)
[WAITRESS ORDER] time= 2017-08-15 14:33:18, location= WayPoint21, status= **Complete**, items= [(Green Sweet, 0), (Red Sweet, 3), (Blue Sweet, 0)]	
	From WayPoint21 do (human_aware_navigation) to WayPoint28 navigation finished on Tuesday, August 15 2017, at 14:33:37 hours (26/7) From WayPoint28 do (human_aware_navigation) to WayPoint10 navigation finished on Tuesday, August 15 2017, at 14:34:48 hours (41/8) From WayPoint10 do (human_aware_navigation) to WayPoint17 navigation finished on Tuesday, August 15 2017, at 14:35:20 hours (22/5)
[WAITRESS ORDER] time= 2017-08-15 14:35:29, location= WayPoint17, status= Open, items= [(Green Sweet, 1), (Red Sweet, 1), (Blue Sweet, 1)]	
	From WayPoint17 do (human_aware_navigation) to WayPoint1 Fatal fail on Tuesday, August 15 2017, at 14:36:05

/waitress_nav	/topological_navigation
	hours (35/0) From WayPoint17 do (human_aware_navigation) to WayPoint1 navigation finished on Tuesday, August 15 2017, at 14:36:37 hours (32/6)

These logs were chosen as they highlight a weakness in the default recovery behaviour implemented, if navigation fails whilst attempting to return to the Hub the recovery behaviour may force an infinite loop of attempting navigation to the Hub and failing. LUCIE's inbuilt recovery behaviours that ask nearby people for help, before sending a tweet to ask for help, should mitigate the risk of this occurring.

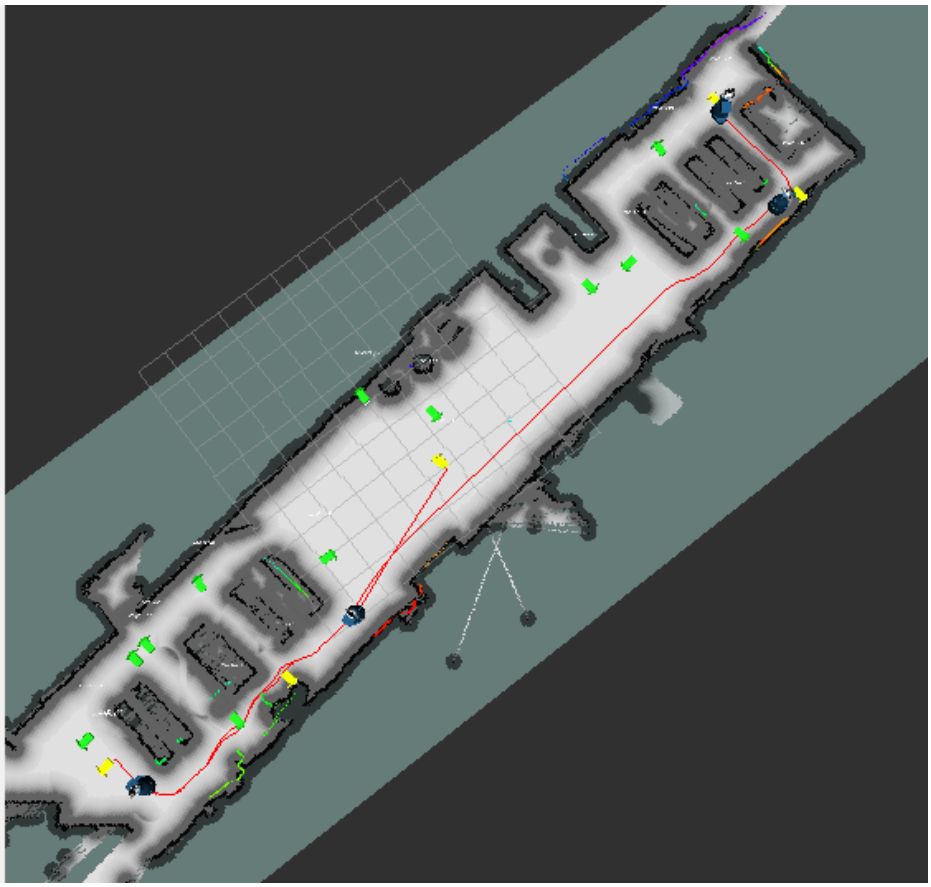


Fig 19: Random Patrol Overlay

3.7.2 Testing In The Physics Café

The Physics Café provided a challenging environment for LUCIE's routing and furniture detection. Testing was undertaken using just the head-camera initially (Fig 20), then using both cameras (Fig 21). However, neither configuration managed to perform satisfactorily, LUCIE would bump chairs and easily get stuck.

Studying the Rviz maps in Fig 20 and Fig 21, the black spots in the main floor are the locations of the furniture legs that were detected during mapping. However, from the photographs it can be seen that this

furniture is easily moved. Ordinarily this map would be cleaned to remove these black spots as they are not accurate over time, however this would contradict the self-imposed constraint of not requiring a user to manually change maps.

The multi-coloured spots show where the laser-scan is intersecting furniture; LUCIE is detecting more furniture legs at run-time than during mapping. These locations are also more accurate. The grey shading shows areas LUCIE will not navigate through as it has determined, via the cameras and laser-scan, these areas contain obstructions. Comparing the two Rviz maps in Fig 20 and Fig 21, it can be seen that the size of these shaded areas changes depending on the direction LUCIE is facing and LUCIE's proximity to the area. LUCIE calculates this cost-map for its local area.

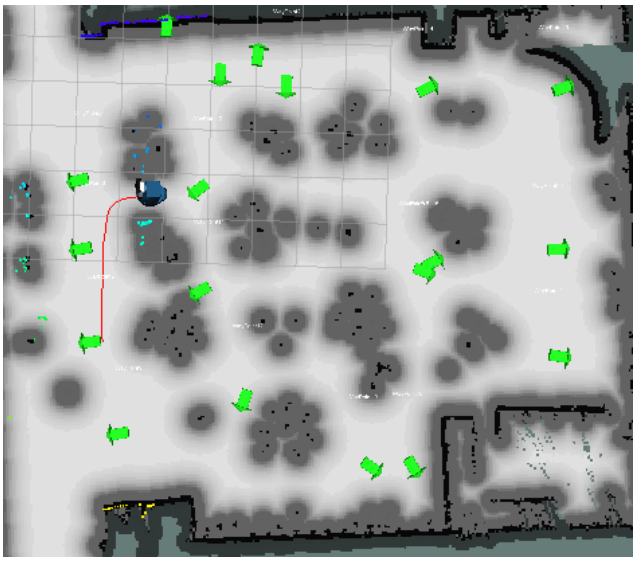


Fig 20: LUCIE stuck in the Physics Café, asking for help.

Fig 21: LUCIE stuck in the Physics Café, again.

The changeable nature of these areas negatively effects LUCIE's route-planning capabilities as LUCIE does not realise the route is blocked until it is too late. At that point LUCIE will be stuck, when it asks for help and is moved to a clear space, it will try the same route again, not remembering that it was blocked. In Fig 21, LUCIE could navigate to the goal if it went the long way around, however it was incapable of determining that route.

The two photographs in Fig 20 and Fig 21 also reveal the situation in which LUCIE gets stuck. On both occasions LUCIE tried to pass between chairs and only once it was in a confined space and did not wish to continue did it try to plan a new route. When LUCIE plans a new route it will circle around to scan the local area, however LUCIE's base is not circular. In both photographs LUCIE has collided with chair legs whilst attempting to scan. LUCIE needs pathways that are at least wide enough for a complete spin in order to not get stuck, whereas the Physics Café pathways are so narrow they can be blocked by a single chair not being tucked in.

3.7.3 Testing In Waterside Café

The Waterside Café was chosen for testing as it remained open through the summer and would provide navigation testing with humans present. However, the café is not busy during the summertime and on only two occasions did LUCIE's path cross a human's. On both occasions the people were amused by the sight of a robot and gave LUCIE a wide berth, therefore LUCIE did not alter its course. However, the Waterside Café's several sections unexpectedly provided three major obstacles for LUCIE.

Fig 22: Waterside Café obstacles: benches, step and bar.

A patrol-route was created to pass through the obstacles show in Fig 22. LUCIE's navigation was not tested around the soft-furnishings area as the pathways in this area were too narrow, as already determined in the Physics Café. This area is also a popular area for people to sit and the risk of LUCIE hitting their chair was

too high. LUCIE's map also did not include the serving area, as the café was conducting business during testing. Each obstacle will be described in turn.

3.7.3.1 Waterside Café Benches

These were not expected to create difficulties for LUCIE, however LUCIE was unable to navigate between them. This is likely due to an incorrect map representation and perhaps chest-camera calibration. Comparing the photograph of the benches in Fig 22 with the map in Fig 23, it can be seen that the only black in the map corresponds to the legs of the tables, the benches are then found in real-time and shaded grey. The route shown in Fig 23 is with an improperly configured chest-camera, which is why LUCIE also believes the floor is untraversable, otherwise LUCIE would not plan a route through the benches.

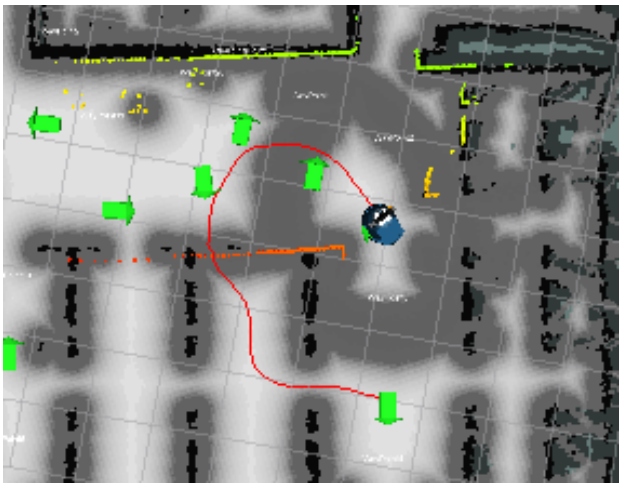


Fig 23: Planning a route through the bench.

Fig 24: Stuck, despite sufficient room.

LUCIE was sent on a patrol route that would pass between these benches, shown in Fig 25, where each robot is positioned at a stop along the route. When planning the route LUCIE determined that there was enough space to pass between the benches, which is true, however LUCIE would only move to between the benches and then refused to continue along the path. With only one camera LUCIE attempted recovery behaviour, as shown in Fig 24 where LUCIE has spun while trying to plan a new route. With both cameras LUCIE became unresponsive, not actioning any of the recovery behaviours and not continuing along the route.

This behaviour is unusual, and the cause could not be determined with certainty or the behaviour rectified. The only hypothesis is that it may be rectified by the chest-camera calibration or changing the threshold for what shade of grey LUCIE will traverse through. This was determined though close study of the Rviz maps, looking carefully at Fig 25, a difference in shading between white and a very light grey can sometimes be seen where the top layer of an image has been erased to allow a lower layer with it's route to show through.

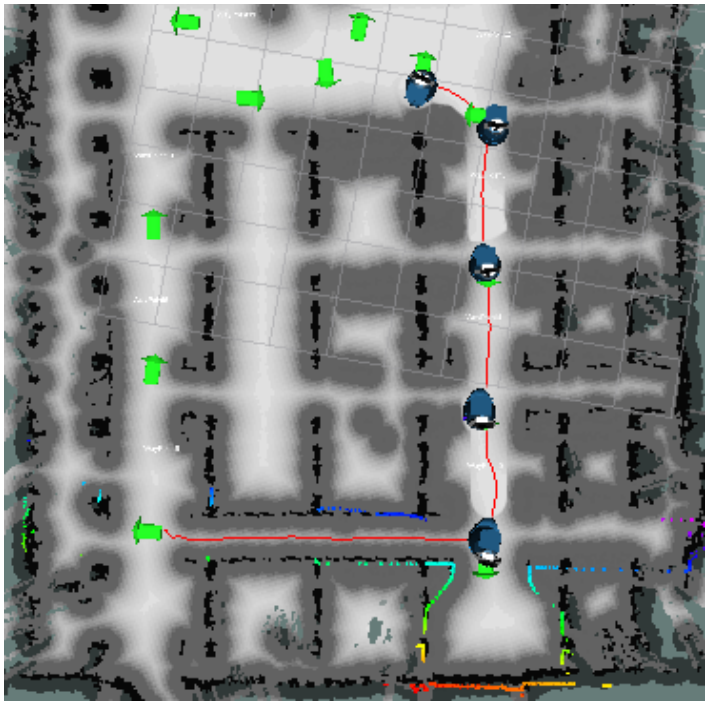


Fig 25: Planning a route between benches, overlay of Rviz maps.

The metric maps in use consist of only black and white, although they can also include grey areas. To allow an editor to choose a grey for shading, a threshold is set in the corresponding yaml file. It may be that as LUCIE moves between the benches the grey becomes too dark for this threshold, i.e. the cost-map for the locale becomes too high for LUCIE to traverse, although this effect could not be observed with the naked-eye via Rviz. Re-calibrating the chest-camera was attempted, however the process gives no indication of how the calibration has changed, therefore a suitable tuning was not achieved.

3.7.3.2 Waterside Café Step

There is a central raised platform in the café that LUCIE would not be able to serve, however LUCIE also needs to be able to avoid crashing into this step. LUCIE's navigation can avoid upward steps if they are high enough for the laser-scan to register them during mapping or if the chest-camera can see them coming, which requires a short distance as the chest-camera looks ahead but pointed slightly downwards.

However, the step in the Waterside Café is too low for the laser-scan and it did not appear in the maps. The area in the center of Fig 27 is the raised platform, the robot to the left of it shows where LUCIE crashed into the step and had to be assisted to move on. The step is also most often approached from the side

Fig 26: Crashing into the step.

and after a short wall, which can be seen in Fig 26, therefore the chest-camera cannot see it coming. No matter the configuration of the cameras, LUCIE would crash into the step.

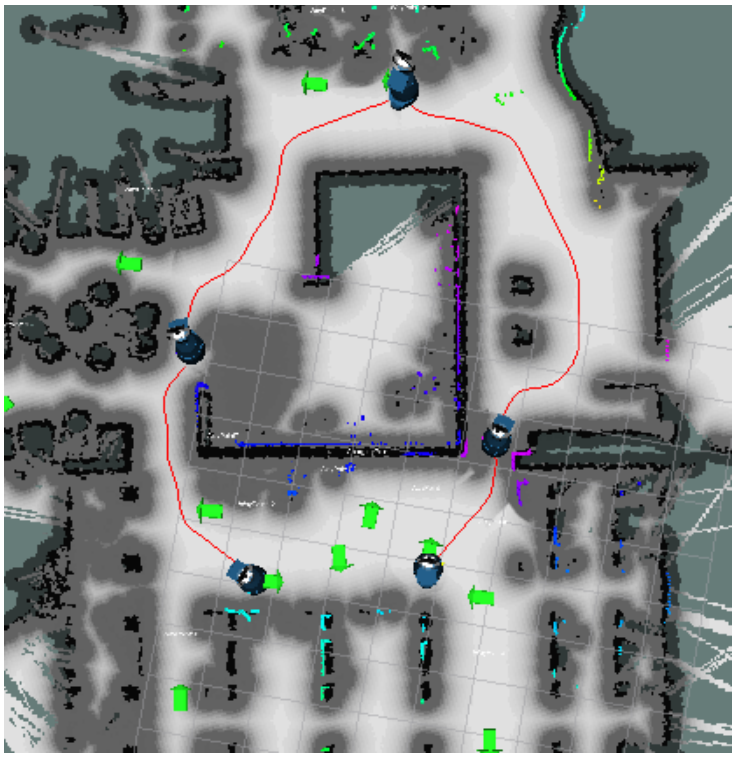


Fig 27: Negotiating the step and bar, moving clockwise.

3.7.3.3 Waterside Café Bar

Around the central raised platform is a bar that protrudes from the wall that is marked in black on the map shown in Fig 27. The difference between the marked wall and protrusion could mislead LUCIE into believing it has more room than it does. This misconception becomes particularly important through the narrow squeeze on the right of Fig 27 and photographed in Fig 28.

With only one camera for navigation, LUCIE believed it could pass through this gap, which is possible. However, because LUCIE did not recognise the bar it positioned itself centrally between the two walls, causing it to crash in spectacular fashion, spinning a full 180 degrees.

With both cameras this crash was prevented, instead LUCIE continually chose a route through the gap, decided the cost was too high to pass through the gap and actioned a recovery behaviour until it was assisted through the gap. Although this behaviour still isn't ideal, it is an improvement on crashing.

Fig 28: Contemplating the squeeze.

3.7.3.4 Waterside Café Obstacles Mitigation

All of the obstacles could be overcome by the placing of additional WayPoints and removal of edges, this could force LUCIE to go around the obstacles rather than through them. Alternatively, the map image could be altered to add in walls. However, this would violate the constraint of not manually updating the maps. With no better alternative, if LUCIE were to be put to work in the Waterside Café, this constraint would have to be violated for navigation to work effectively.

3.8 Navigation Evaluation

Both created products work as required, neither require a technical understanding of the underlying systems to use. However both products are also command-line tools that will require someone familiar with the environment to run.

LUCIE’s topological navigation, although the best available, is still flawed. If an environment is less than ideal it is possible to alter the maps to achieve an effective navigation system. However, there are environments that would require significant changes for LUCIE to be capable of satisfactory navigation, such as the Physics Café or the raised platform in the Waterside Café.

Areas for improvement within navigation include the chest-camera calibration software, which lacks detail and information, as well as LUCIE’s long-term route planning. Currently LUCIE plans in a Markovian manner, however, a recovery behaviour that plans a new route considering the previous one’s failure would be beneficial.

The limitation of WayPoint navigation is that LUCIE aims for a set of coördinates, when the intention is to navigate to a customer. A further iteration on the project should marry the two, perhaps by adding skeletal analysis and facial recognition to determine if a user requires LUCIE’s attention.

4 UI

The User-Interface forms the core of the software as it controls the robot’s behaviour. The UI will be used to control the logic and behaviour, calling other nodes and services as required. In this iteration of the project, the only required major service is the navigation. Therefore, the UI must be instantiated with a client node to send messages to a running topological navigation node.

When re-purposing a robot into a service robot a great deal of attention must be paid to the user interface as this is how it will communicate with those it is expected to serve. At the point of contact between the robot and the customer there will be no support staff present. Therefore the customer must be able to interact with the robot in a natural manner, without instruction. Existing user-interfaces exist within the STRANDS project, these are reviewed prior to choosing appropriate technology to design the user interface. Then the expected interactions will be determined before designing, building and testing the UI along with its integration into the greater STRANDS/ROS system.

4.1 Existing STRANDS UIs

There are three existing UI systems that have been used with LUCIE at events: `strands_ui`, `info_terminal_gui` and `marathod_gui`. The three systems are very different.

4.1.1 `strands_ui`

This is used as the recovery behaviour with standard topological navigation. It includes a GUI within a Web Browser, spoken voice without listening, and optional tweets. The GUI is minimal, only showing the current display number, however when LUCIE requires assistance a modal will appear asking for help. Depending on the assistance required LUCIE may also speak, asking for help. However a human cannot reply by speaking and must use the on-screen modal to reply. The behaviour for the page is written in JavaScript and is included in the page source, although it requires some study to unravel how it works.

4.1.2 `terminal_info_gui`

This is part of a larger project, `aaf_deployment`, whose requirements include making the robot follow a group of people and entertaining them. For events it has been repurposed to show a series of screens such as the news and LUCIE's twitter feed whilst LUCIE patrols a series of WayPoints.

The underlying technology is web-based, making use of Aaran Swartz's “`web.py`” framework. It is run using the framework's inbuilt server.

4.1.3 `marathon_gui`

STRANDS includes its own "web framework", which this UI makes use of. This "web framework" runs a server to show webpages and includes helper functions. However it takes a ROS based approach, using publishing on topics and message passing instead of handling HTTP requests and forms, which is incongruous with standard web frameworks. Therefore, this method provides less flexibility and control over the GUI and its associated behaviours, although it is simpler to integrate into ROS.

4.1.4 Summary Of Existing UIs

Existing solutions are unanimous in their choice of using a Web Browser UI. There are advantages to conforming with this choice for the creation of the waitressing UI:

- Flexible layout will cope with dynamic and changing content, such as reviewing orders or changing menu items.
- Established communication protocols via HTTP, AJAX and Web Sockets
- Potential integration with existing navigation recovery systems, such as `strands_ui`
- Visibility on multiple devices connected to the same LAN by running the web server on IP 0.0.0.0 The visibility on other devices creates the option of having a "admin" machine located at the hub, which can be used to monitor and control LUCIE. It could also serve to notify staff immediately of any issues arising.

The three existing GUIs chose very different approaches: handwritten JavaScript, Python’s Web.py, and a ROS based one. However, this project will not use any of these approaches. The ROS web framework does not expose the established communication protocols to the programmer; for a developer who is experienced in web development it is a confusing and counter-intuitive system. By not exposing the request method it will require a work-around to handle forms, which will be difficult code to maintain and read. Therefore a web framework designed for web development will be used as this can also communicate with ROS using the node, topic, message system.

To make the GUI as easy to maintain as possible, the Python framework Flask (Flask.pocoo.org, 2017) will be used. Flask is taught in an undergraduate module (Leedsforlife.leeds.ac.uk, 2017), therefore it is likely other students who may continue development on LUCIE will have prior experience working with it. Flask is also a well-established, stable and current framework, running with Python 2.7 currently supported by ROS Indigo as well the latest versions of Python, which will be valuable when Python 2.7 is retired from 2020 (Pythonclock.org, 2017).

4.2 Expected Interactions

There are two categories of expected interactions, those that will arise in normal usage and those that help a user resolve any issues. There are also two expected groups of users, the customers and the supporting staff.

4.2.1 Normal Usage

Normal usage requirements are illustrated in Fig 29.

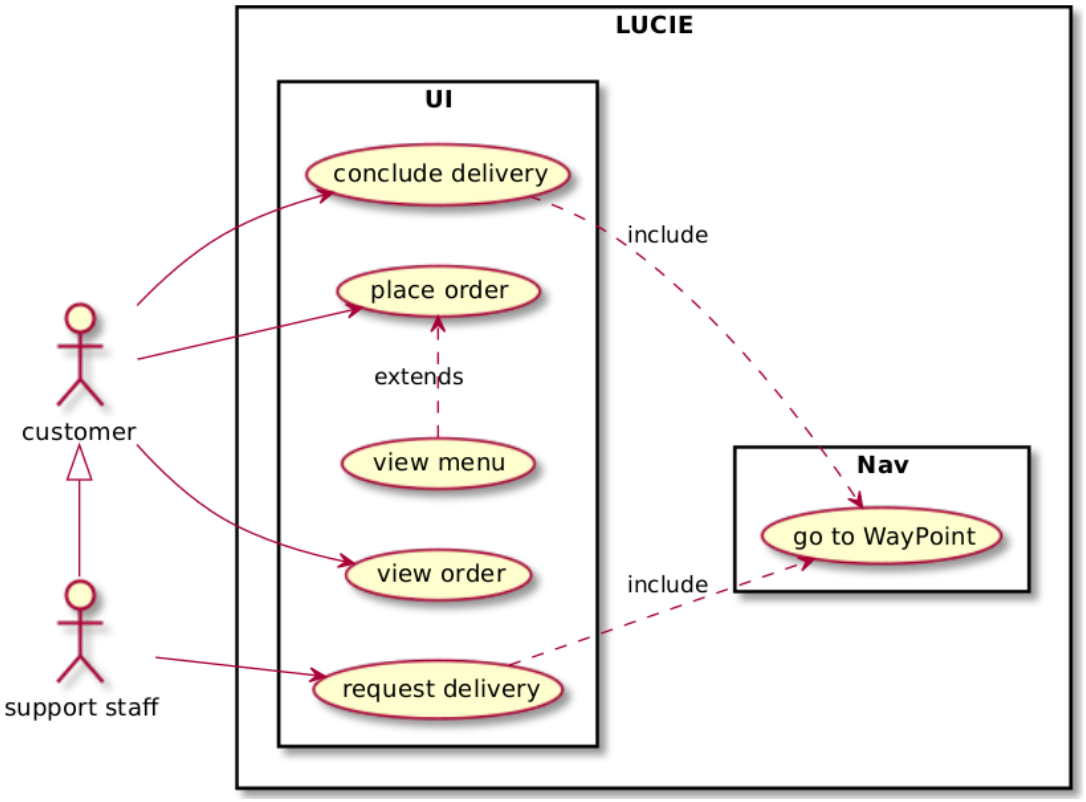


Fig 29: UML Use Case diagram for ordering

As a first approximation a page could be used for each task required in the UI, with the exception of "view menu", which is necessary to place an order and therefore should be included on that page. This gives an initial mapping of pages:

- "/": view menu, place order
- "/order": view order
- "/delivery": conclude delivery
- "/navigation": request delivery

For convenience it would also be beneficial to support staff if they could request the delivery of an order whilst viewing it. The access to different pages will require some security as support staff only should have permission to choose navigational goals for LUCIE to avoid misuse of the robot.

The use of these URLs are shown in Diagram 2, which models a simple interaction with patrolling navigation. URLs to be displayed are shown in speech-marks.

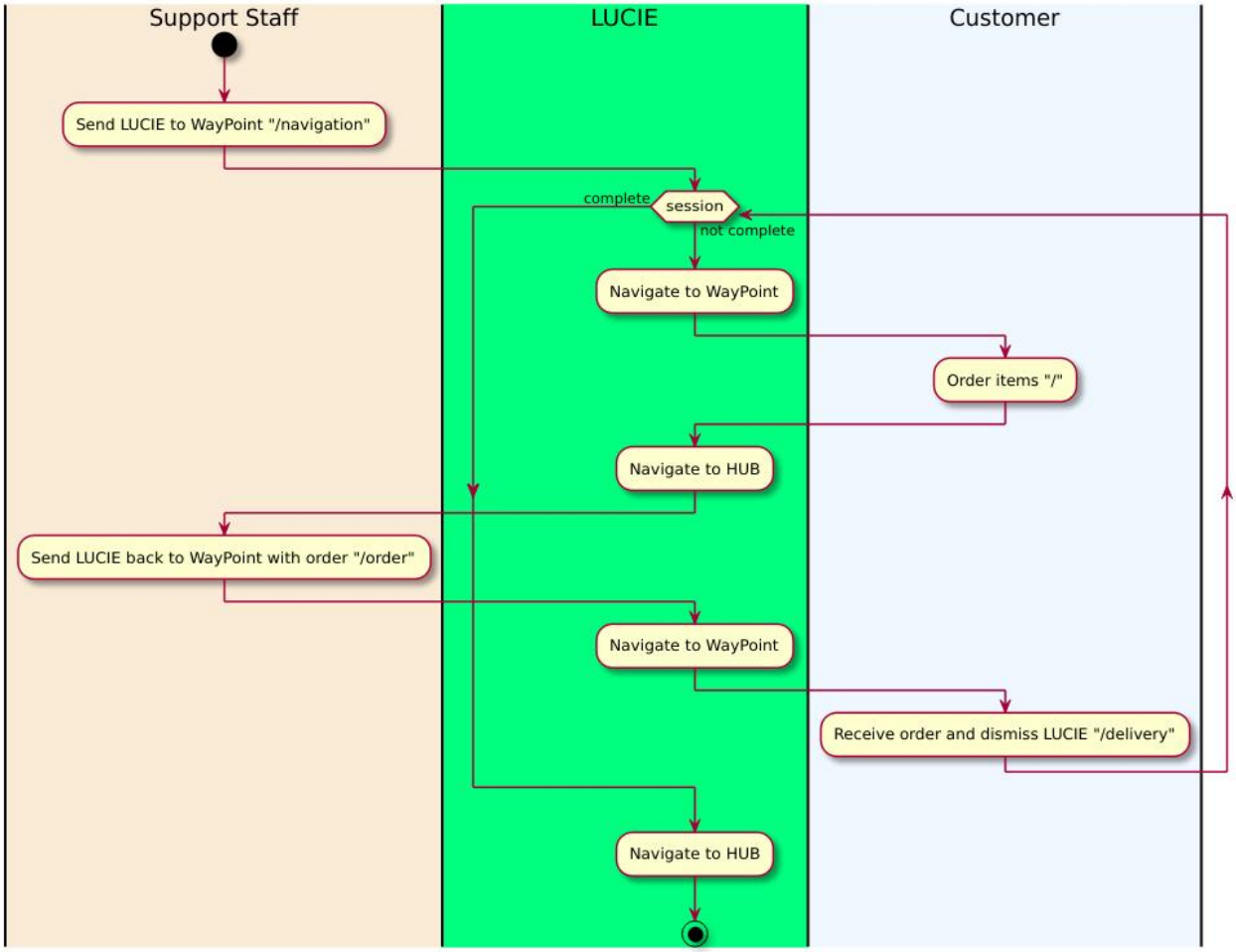


Fig 30: UML Activity diagram for ordering

4.2.2 Additional Interactions

There are three additional interactions that should also be catered for, a recovery interaction, login interaction and social interaction.

4.2.2.1 Recovery Interaction

It will be useful for support staff to have access to views of all orders in case of mistakes in usage. For example, if an order was accidentally cancelled, support staff should be able to look up that order and send it for delivery. To achieve this requires an URL to view all orders ("/all_orders") and a URL that will display a given order ("/order/<orderId>") without navigating back to the Hub, which is the usual behaviour when an order is viewed.

4.2.2.2 Login Interaction

The GUI will provide access to behaviours that a customer should not also have access to, such as giving LUCIE a navigational goal or changing the status of orders. Therefore a login and session is required. This is particularly useful if LUCIE is being run with an additional device at the Hub, which can be logged in and used by support staff. However, there may be occasions when it is undesirable for this security feature to be imposed, therefore it should also be capable of being turned off. This login page is designed with a pin-pad to work easily with a touch-screen.

4.2.2.3 Social Interaction

LUCIE has an inbuilt program that will take a photograph of a user and post it to Twitter. As this feature is popular, beneficial for marketing and requires almost no work to include, it will also be optionally incorporated into the GUI.

4.3 The URLs

To cater to the identified interactions, the following URLs were used, when the technology was switched to use a single page application with WebSockets, the URLs became abstract and map directly to the ContentLoader class methods. This mapping is included in the code comments.

- "/": Menu screen for ordering items
- "/order": Navigate back to HUB, display order. If the user is logged in, allow the order to be sent for delivery or cancelled.
- "/deliver/<orderId>": Navigate to the location where the order was placed. Allow the user to order again or dismiss LUCIE.
- "/navigation": **Available to logged in users only**. Send LUCIE to any given WayPoint.
- "/twitter": Show LUCIE's twitter feed, used when also running program to take photos and tweet them.
- "/login": Used to login, there is no clickable link to this page.
- "/all_orders": **Available to logged in users only**. View the status of all placed orders. Allows support staff to handle mistakes. For example, an order that was cancelled in error can be sent for delivery.
- "/order/<orderId>": **Available to logged in users only**. View an order without navigating back to the HUB.

4.4 Visual Design

LUCIE is equipped with a touch-screen monitor, which customers will use to place their orders. The GUI could also be viewed on other devices with unknown dimensions and unknown peripherals. Therefore the

GUI must have a flexible layout and use large buttons that are easy to tap. It is likely that users will be reading the screen from a small distance, therefore important text must be in a large font-size.

To save time developing the design and features, Bootstrap (Getbootstrap.com, 2017) is used. It provides a modern look with flexible layouts and features such as modals, which are useful for additional communications. An example page is shown in Fig 31, with support-staff logged in.

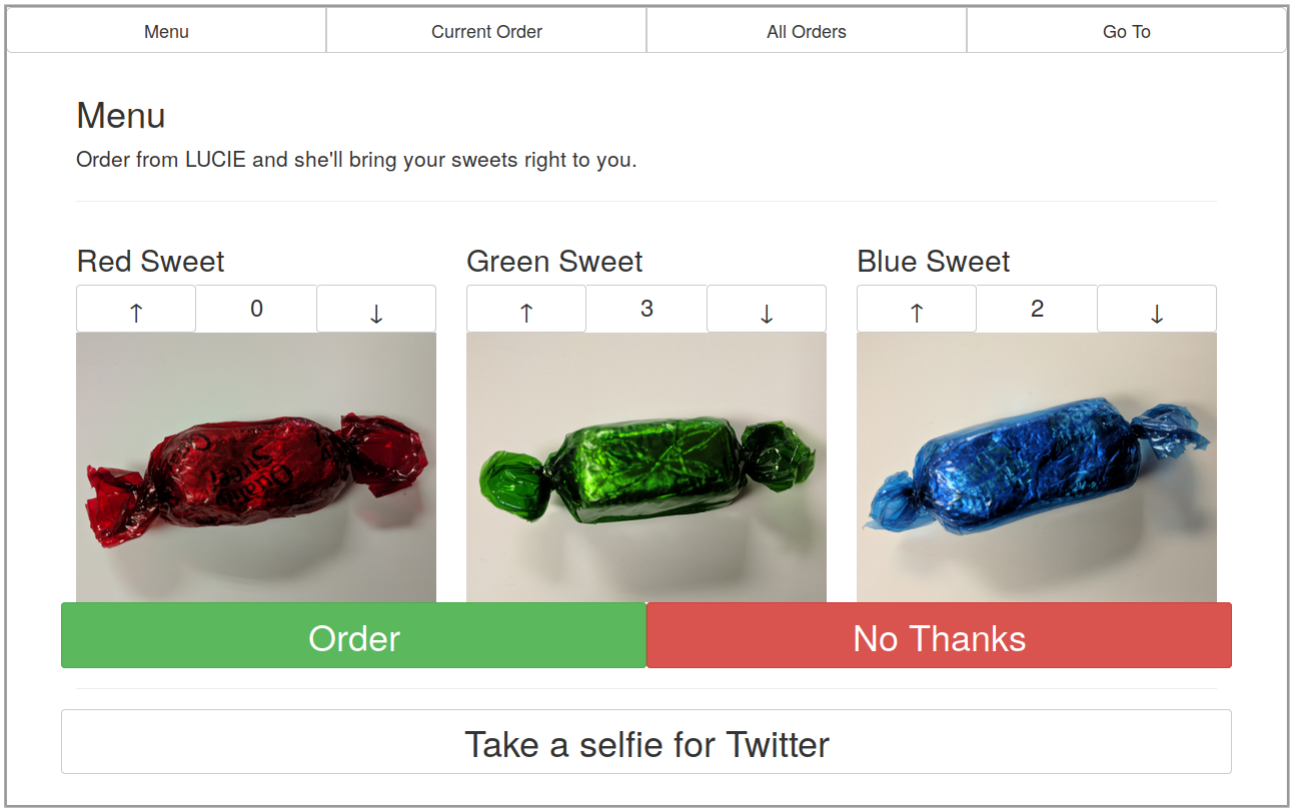


Fig 31: "/" View menu and place order, with support-staff navigation bar.

4.5 Structural Design

The GUI, which is implemented using Flask, requires three objects: one for navigation, the second the orders, and a third for HTML content. For each one a class within their own file was created. The Flask code within `waitress_gui.py` operates as a controller, calling the methods of the classes as required. The view part of the application is uses Jinja2 templates, which are integrated into Flask, rendered and served via WebSocket.

4.5.1 Navigation

The Navigation class is a wrapper around a `SimpleActionLib` client that sends navigational goals. It also includes utility functions for sending LUCIE to various locations and retrieving the current location.

4.5.2 Orders And Logging

The Orders class creates and uses a sqlite3 database in the `/tmp` directory, which provides no long-term persistence. This was chosen because LUCIE will primarily be used for demonstrations, therefore the persistence of a database is not required, however the utility of handling multiple transactions from customers and support-staff is required. To enable analysis of orders, the class integrates with ROS logging. The class contains utility functions for adding, updating and logging orders. If a user wishes to persist the database, only the path to the database need be updated. The database is interacted with via Flask-SQLAlchemy (Flask-sqlalchemy.pocoo.org, 2017), a Flask specific implementation of the SQLAlchemy abstraction for databases. This will allow future changes to the underlying database in use without needing to change the code that interacts with the database.

4.6 Integrating Navigation

Prior to integrating with ROS/STRANDS navigation, LUCIE could be directed with the command `rosrun topological_navigation nav_client.py WayPoint1`. This command creates a temporary client that transmits a `GoToNodeGoal` message. Although it is possible to use this command from the GUI via Python's core subprocess library, it would effectively detach the GUI from the rest of the ROS system, meaning there would be no record of the GUI sending topological navigation messages within `rostopic`.

To integrate into ROS in such a way that the GUI registers its existence as well as which topics it communicates with, it needs to initialise itself as a node. Navigation is then handled within its own class, which creates an `actionlib.SimpleActionClient` to communicate with `topological_navigation`. It can then send a navigation goal and receive feedback on the success of achieving that goal.

There are two items support staff need to configure in `config.py` for LUCIE to operate successfully. Firstly a WayPoint needs to be designated as the Hub, where LUCIE will go to collect items to deliver. Secondly, LUCIE needs to know how many WayPoints there are in total for the options presented in "/navigation", currently this is hard-coded, although it should be possible to extract this information from the MongoDB store.

LUCIE also needs to track the WayPoints where orders are placed, the current WayPoint and target WayPoint if one currently exists. This is handled naïvely by updating a variable upon the successful completion of a goal, however, it should also be possible to extract this information from a topic.

Variations on topological navigation include waiting at a WayPoint or setting a maximum time for a task before moving on to the next task. Both of these methods appear to be relevant to the timeout functionality that allows LUCIE to move on when no customer interacts with the UI. However the interaction is on the client-side of the web application, which will not inform ROS of the need to cancel the timeout. Therefore

this logic is implemented on the client-side, where the interaction occurs. This method also allows more flexibility in when timeouts are set and in what situations they are not required, preventing LUCIE running a default timeout behaviour when at the Hub for example.

4.6.1 Asynchronous, Parallel, Or Concurrent Navigation

When topological navigation is sent the goal message the process needs to wait to receive feedback in order to keep track of LUCIE's position and to ensure appropriate information is displayed. This waiting blocks updating the GUI, preventing any communication whilst LUCIE is moving. To prevent this blocking the command must be run using either Python's `async` and `await` feature, in a parallel process, or concurrently with multi-threading.

For ROS Indigo, rospy is only available for Python 2.7, whereas `async` was introduced in Python 3.5 (Python.org, 2017). Prior to this, third party libraries such as Twistd are required, which would add further complexity and be incompatible with Flask.

Due to the Global Interpreter Lock (GIL), Python is incapable of providing true multi-threaded operations (Wiki.python.org, 2017). However, running the blocking part of the function in another thread does allow further requests to be processed, but this could not be used in the final version as the thread does not retain the context of the request. This prevented the sending of the message back to the client to inform it the navigation has been successful and it should begin the timeout.

The final option available is to fork the process and send the message in parallel. However, in ROS the message must be transmitted from the same Process ID (PID) as the one used to register the node. With all options exhausted, Python 2.7 will block on this transmission.

Python isn't the only language used in the web stack, JavaScript offers the possibility of communicating via AJAX or WebSockets to continue updating whilst the Python process is blocked. However, both solutions require the Python process to respond to determine success, therefore JavaScript is also blocked from achieving this task.

With judicious planning, the blocking may be a useful feature. As LUCIE’s navigation is bound to WayPoints, if a customer were to try to place an order whilst LUCIE is in transit, LUCIE would be unable to return to the customer to deliver the order. This blocking makes the UI unresponsive, preventing anyone interacting with LUCIE and thus prevents such issues. The problem that arises from an unresponsive UI is that the user does not know that their request has been processed. By first responding to the users request and updating the UI, followed by sending a background message to the server for navigation, the system communicates to the user that their request is being handled correctly.

4.6.2 Communication Protocols

Applying traditional web technology communication protocols allows communication from the client to the server, which can also then be used to communicate with ROS.

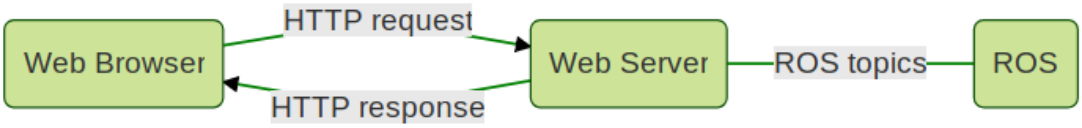


Fig 32: Traditional communication via HTTP

This is the method used when loading new pages, however for the server to gain a message from the user a page refresh is required. This is not an issue when creating a new order as the order page should be shown at this point, however if this method of communicating were used for navigation the new page would not load until the process of navigating were complete. This is caused by the necessity of calling the navigation function while processing the request for a new page, which then cannot return the new page until the navigation is complete.

To circumvent this limitation, AJAX calls can be used to make HTTP requests and accept responses without reloading the page.

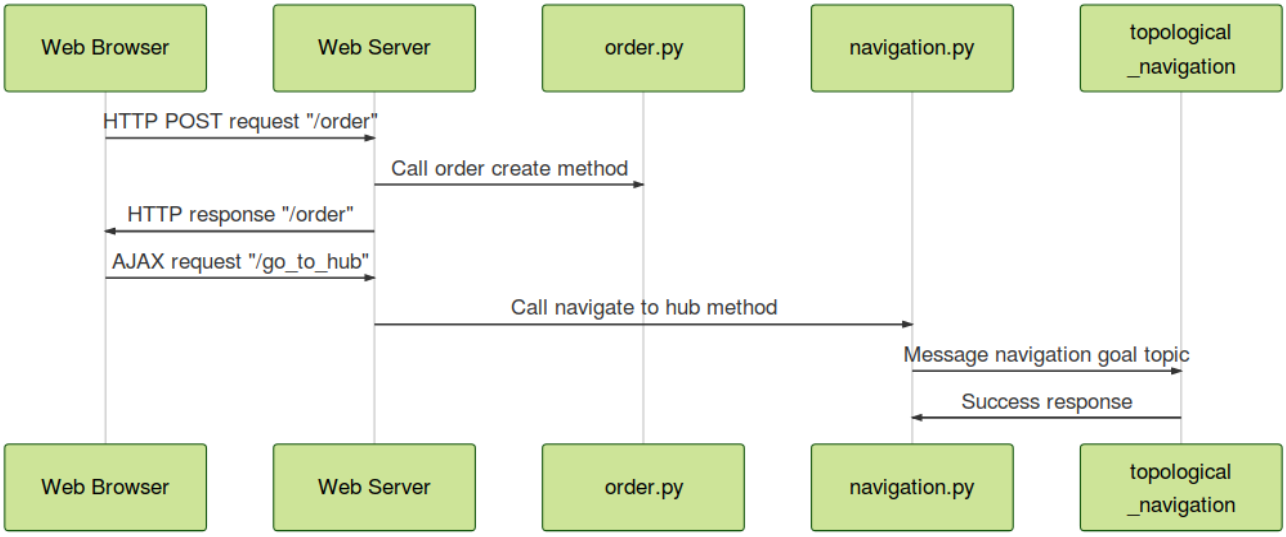


Fig 33: Using AJAX to return the "/order" page before navigation is complete.

This method provides most of the desired behaviour, however it still does not provide a complete route for ROS/STRANDS to communicate with the user. In Fig 33 only `navigation.py` is made aware of the response from the topological_navigation package, the Web Browser cannot receive data from the Web Server without the Web Browser initiating a request.

WebSockets are an alternative communication protocol that allows a Web Server to push data to the Web Browser, as well as to accept data. To fully exploit WebSockets the application is refactored into a single

page application, this also allows an admin machine to force the display on LUCIE to show a given page, or navigation.py to inform the browser that navigation has completed and it should begin the timeout.

Initially the application was developed using AJAX as it most closely mirrors the planned urls. However, the additional functionality afforded by WebSockets of passing messages from the server to the client proved invaluable for the timeout function. To implement WebSockets the Flask-SocketIO (Flask-socketio.readthedocs.io, 2017) extension was used, which provides a relatively simple interface to using WebSockets. Both versions are included in the code repository and both are on LUCIE, however the AJAX version is less complete as it was abandoned.

4.7 Implementation

The final version of the project consists of a complex web of files in various languages. However, they are organised into a relatively simple structure, each file or package has a particular purpose. The application follows standard Flask development principles, using the templates and static directories for HTML and static files such as CSS and JavaScript respectively. The interaction between files is shown in Fig 34, `waitress_gui.py` is the entry point.

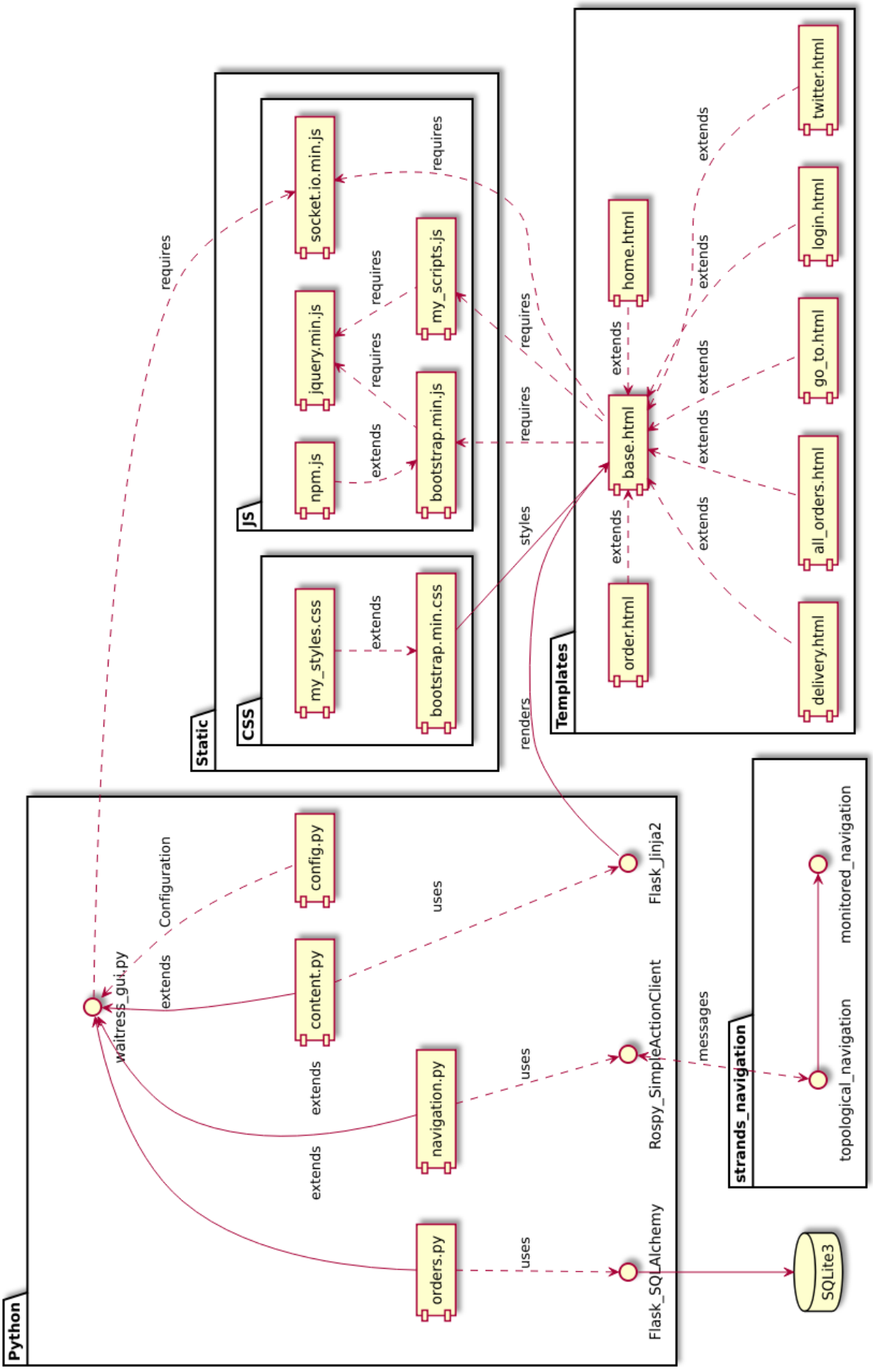


Fig 34: Interaction between code files.

Using WebSockets with Flask-SocketIO instead of normal HTTP requests required the creation of `content.py` to handle the serving of HTML. On the front-end a script in `my_scripts.js` updates the DOM. This script was also required to reload some JavaScript functions so they could register with their respective DOM elements, if they are included in the new content. The normal timeout for WebSockets also had to be increased due to navigation blocking processes for significant periods of time.

4.8 Testing For Expected Functionality.

There are three main goals for this UI:

1. It must result in expected navigational behaviour
2. It must be intuitive for customers to use
3. It should be simple for non-technical users to setup and use.

4.8.1 Navigation Testing

The navigational behaviour was tested in both "One Machine mode", where LUCIE is logged in as an admin user, and with an additional admin machine. LUCIE's expected navigational behaviour was achieved on both platforms as they both call the same functions:

Behaviour	One Machine	With Admin Machine
Go to given WayPoint	✓	✓
Return to hub on order	✓	✓
Remember order location	✓	✓
Deliver order to correct location	✓	✓
Return to hub after delivery or go to a random WayPoint	✓	✓
Return to hub on navigation failure.	✓	✓

This testing is covered in more detail in 3.7 Testing LUCIE’s Navigation, page 18.

4.8.2 UI Testing

The testing of the UI was minimal as only two pages are expected to be interacted with by a customer. In this case five people were asked to interact with LUCIE as a Waitress, but no instruction was given. All users were observed placing orders and accepting them without requesting assistance. This was observed again during the open-day event discussed in 5 Final Testing, page 42.

4.8.3 Simple To Setup And Use

To test this a post-GCSE pupil was allowed to run the programs under supervision. The pupil was read the READMEs for each application and given some practice driving LUCIE. They then created two new maps

and ran LUCIE as a waitress in each of the two locations. In the first instance a couple of bugs were found, a hostname was incorrect and a vital piece of documentation was also incorrect. This was due to the packages on LUCIE not being up-to-date with those in the master branch of version control. The second run went very smoothly with no issues.

The pupil struggled with Tmux, which is not a user friendly application, however the output is very useful for technical operators. As this is a demo-program intended to be used by Computer Scientists within the university, Tmux will continue to be employed. It is expected that the support staff will be familiar with command-line computing and find the output useful when debugging.

4.9 UI Evaluation

For a service orientated robot the interface with those it serves is the core component from which physical behaviours are determined. The user interface developed is functional and provides options for setup and behaviour. There are still technical challenges that need to be resolved and more features to be developed, including:

- Asynchronous navigation
- Extracting map information from ROS/STRANDS
- Complete integration of monitored navigation helper

During testing a need for improving the topological navigation and its integration within the UI also became apparent. If a human is stubbornly occupying the space that is the exact WayPoint LUCIE is trying to reach, LUCIE's navigation will eventually fail. This can be resolved by letting LUCIE accept a close proximity to a WayPoint as successful. The UI also needs to more gracefully handle navigation failure, currently there is no behaviour implemented, such as displaying a “Help Me” message. However, a suitable modal is included within the code so it should be a relatively trivial matter to subscribe to the correct message stream and use `content.py` to push these messages to the UI. Currently this modal is used to inform support-staff when a customer has placed an order if an admin machine is used, which allows them to view and begin preparing the order before LUCIE’s arrival.

5 Final Testing

With the UI developed and integrated with topological navigation, LUCIE was scheduled to work at a University open day event, where it would accept orders for biscuits and deliver them back to the visitors. The event was held in the Longroom, with standing visitors who were free to travel where they wished. Throughout the day multiple issues were encountered that highlighted the strengths and weaknesses of the approach.

5.1 Lack Of Interest

It was hoped the event would be an opportunity to gather data such as delivery times, navigational failure rates and public opinion on robots in the service industry. Unfortunately, although people were curious to see a robot driving, they were more interested in seeing if LUCIE would crash into them than ordering biscuits. As a result, only two orders were completed over the course of several hours. Other data was not gathered in sufficient quantity for analysis due the addressing of further issues.

Talking with attendees at the open-day revealed that the wrong approach may have been taken for that event, the robot may have distributed more biscuits if it had patrolled the area with biscuits on the tray, as opposed to asking people to place an order and wait for it to be returned. This would be more akin to wait-staff distributing hors d'oeuvres at an event, but would not have allowed for testing the ordering system.

5.2 Moved Furniture

When first running LUCIE it became apparent that some of the tables at one end of the room had been moved and some of the existing WayPoints were now positioned within furniture. As guests were arriving an hour earlier than expected, rather than re-map the Longroom, the map was directly edited to remove the incorrect tables. The edit can be seen by comparing Fig 35 with Fig 8.

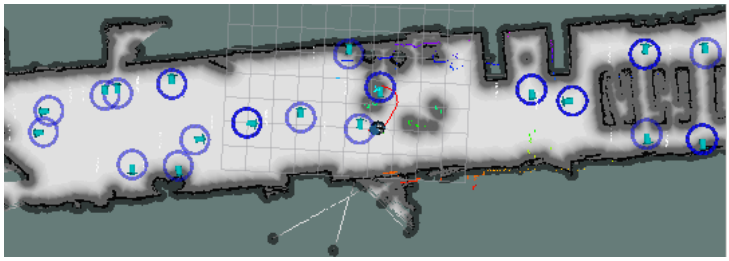


Fig 35: Tables erased from Longroom map.

As WayPoints within furniture were discovered, they were moved using Rviz to more appropriate positions. This caused many navigational failures at first and it would have been quicker to simply remap the room. Once the furniture was removed from the map and WayPoints were adjusted, LUCIE no longer encountered difficulties due to the furniture and was still able to localise well.

5.3 High Density Of People

LUCIE's navigation is quite capable of avoiding humans, however, there are subtleties of navigation around people that LUCIE has not acquired. LUCIE was observed stopping very close to people when finding itself in a narrower path than expected. This was often caused by slight movements of the people adjacent to the planned route. This could lead to LUCIE getting closer to a person than another human would and staying in locations that a human would vacate.

LUCIE managed to navigate around groups of people well, however if there was no path available LUCIE could not reach the destination. Fig 36 captured a moment where LUCIE is taking a route around the groups of people, but the laser-scan is also picking up a person's feet at the destination WayPoint. LUCIE has not yet calculated the cost-map for this area, so the grey around all the people standing there is not yet displayed. In this case, LUCIE's navigation will fail to reach the WayPoint despite the routing, this also happened in Fig 37.

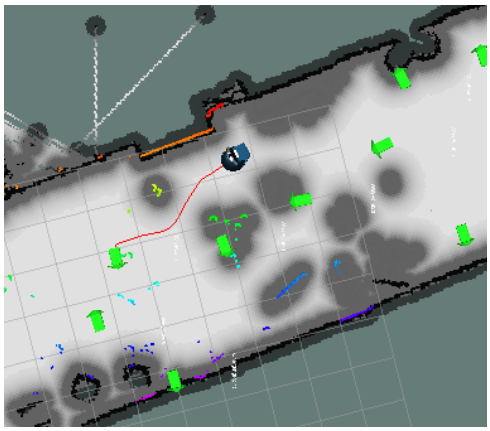


Fig 36: LUCIE navigating around groups of people.

Failure due to people occurred in two situations, either the destination was surrounded or by a group of people, or there was no clear path past the people. In the first situation a waitress would accept reaching the group as reaching the destination, in the second a waitress would politely ask for people to allow them past. Neither solution is trivial.

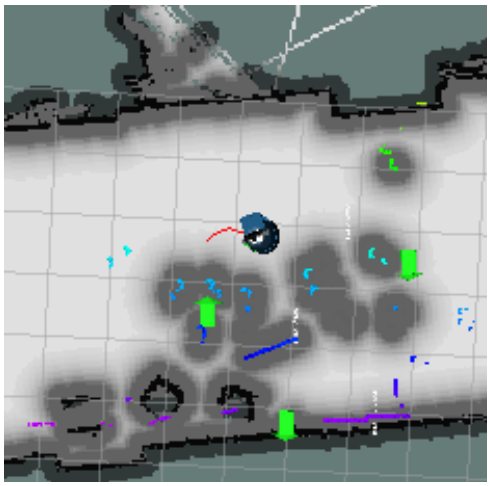


Fig 37: LUCIE unable to reach WayPoint due to people.

To accept close to a destination as success, LUCIE would be required to discern when this is appropriate to apply the correct behaviour. This behaviour would only be appropriate if LUCIE is in range of the customers it is trying to reach, therefore it would have to apply its human detection to the WayPoint to make a decision.

To ask for people to allow LUCIE past would require a new recovery behaviour to be implemented. To determine when this behaviour is appropriate would again be the cause of difficulty. LUCIE would have to ascertain that the only way to reach the destination is through the people. This would require analysis of multiple routes as well as human detection. Fig 36 shows the limited capacity for planning ahead.

As LUCIE is a robot, peoples behaviour around it is also unpredictable. People were observed carrying on conversations whilst LUCIE was trying to pass, ignoring its pleas of help, and deliberately stepping in-front of it to see if it would crash into them. However, people were also observed moving to allow LUCIE to pass.

A trivial solution is possible if the robot does not aim to duplicate the behaviour of a waitress. By carrying hors d'oeuvres and patrolling with a short wait at various WayPoints, if LUCIE's navigation were to fail due to density of people it would appear as though LUCIE was serving those people. By changing LUCIE's recovery behaviour on navigational failure to choose another WayPoint to visit, LUCIE would then carry on patrolling and serving customers. There would be little difference between a navigational fail and a visit to a WayPoint.

However, for the open-day event, a programmed solution was not possible to implement. So the navigational paradigm was changed from `RANDOM_PATROL` to Spoke-and-Hub. Support-staff were then able to observe the room and choose the best destination to send LUCIE to. This proved more successful with no navigation failures and gained LUCIE more attention as it was sent to suitable parties.

5.4 A Failed Delivery

On one occasion LUCIE appeared to complete a delivery, however the application did not register it as complete. On this occasion, LUCIE was close to the WayPoint when the customer accepted the delivery, but it could not actually reach the desired WayPoint. Therefore the UI was unresponsive as LUCIE still believed it was navigating, when the navigation failed, LUCIE returned to the hub as expected.

This again highlights the limitations of WayPoints as the navigational paradigm for the hors d'oeuvres delivery task. However, it also demonstrates a difficulty that arises with blocking communication during navigation, the customer was unable to communicate with the robot that they had completed the order. Solutions are similar to those discussed in 5.3 High Density Of People.

5.5 Missing Biscuits

Several varieties of biscuits were available though the day, however at different times these changed between a maximum of 6 and a minimum of 2, as shown in Fig 38. Fortunately the system was designed to be extremely flexible and new biscuits were added with a photograph and adding one line to `config.py`. To remove biscuits that were no longer available, the corresponding line is removed from `config.py`. By running the server in debug mode, the content was automatically updated when the file was saved.

The layout and JavaScript updated automatically. This was achieved through using Jinja2 to template some parts of the JavaScript in `home.html`. This design made adding new food relatively quick and easy, although perhaps more importantly, removing food that was no-longer available was almost instantaneous.

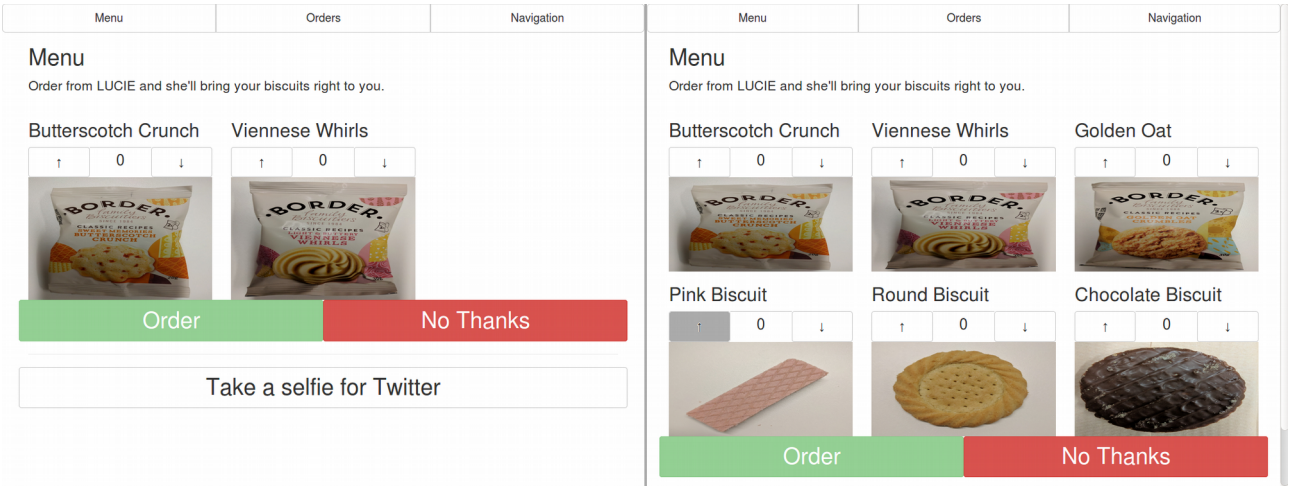


Fig 38: Dynamic design adapting to changing menu.

5.6 Evaluation Of Final Testing

The situation LUCIE was tested in, whilst still under the remit of a waitress, is significantly different from the situation described in 2.2 Determining Expected Behaviour For LUCIE, page 6. This difference has resulted in an unfair test of the entire system within a live environment, however it did raise some interesting issues and proved the adaptability of the system.

It would now be quite simple to adapt LUCIE to work in a situation where it was given a tray of food to distribute by applying a patrolling and waiting routine. There is no need for a UI in this situation, therefore a simple presentation with information could be used to occupy the screen.

Of the issues raised, only proximity to a WayPoint has potential to cause issues within a structured environment where LUCIE is providing table service, relaxing the perimeter of WayPoints may be effective in resolving this. However, the true requirement is for LUCIE to navigate to people, not a set of coördinates, therefore LUCIE's navigation should be extended to achieve this goal.

The UI proved flexible enough to cope with a rapidly changing menu, and it was useable without instruction. The ability to change navigation paradigms with a single configuration variable also proved very useful as it enabled support-staff to switch to a more appropriate one during operation.

6 Conclusion

The aim of this project was to investigate the potential for LUCIE to work as a waitress. This required the integration of two main components, navigation and UI. Although the major part of the project comprised of developing the UI, the majority of issues have resulted from the navigation and its integration.

The primary issue with the navigation system used is its use of a WayPoint to describe a destination as a representation for the true intention, which is most often to navigate to people. This could be resolved in a number of ways, LUCIE could navigate to within the proximity of a WayPoint then navigate to the nearest human, or LUCIE could navigate to visible markers displayed by humans that require its presence, or LUCIE could use its skeletal analysis and facial recognition to attempt to determine if a human requires its attention. This method of navigation will work for both table service and in walk-around situations as an interrupt in a patrol.

The primary issue with the integration of the navigation into the UI is the process blocking, which restricts the flexibility of the system. A crude work-around would be to display a “navigating” screen during navigation, however this still lacks flexibility. The solution is likely to involve breaking up the UI into smaller microservices that are each their own node in ROS, and thus have their own process. This could be done with one node per WebSocket and would allow navigation, ordering and content all to be updated in parallel. This would also circumvent the issue with ROS messages and process ids encountered with multi-processing code. However, the problem encountered whilst implementing multithreaded code with Flask’s requests and contexts when trying to emit messages from the websockets to the client may still arise.

The usability of the system has been hugely improved through Tmux scripts for the semi-automation of generating topological maps. The UI also proved intuitive and flexible. During demonstrations, LUCIE has proven capable of providing adequate table service. During the open-day a solution was found to enable LUCIE to accept orders and delivery food by changing to the Spoke-and-Hub paradigm and giving a high-level of supervision. However, this method is not the most effective for such a situation.

If LUCIE is to continue working as a waitress in the open-day environment, a patrol and wait routine with a quick navigation failure that results in moving to another WayPoint should be implemented. Given a tray of food, LUCIE could then patrol and serve. An extension to the program would be to have LUCIE recognise when it needs to return to the Hub for a restock.

If an entrepreneur were looking to create a restaurant using robotic wait-staff such as LUCIE, it would be advisable to carefully control the environment to make it ideal for the robot. Using fixed furniture with solid bases would enable simpler mapping and accurate navigation. Ensuring pathways have sufficient turning space will allow the robots to manoeuvrer and provide sufficient space for humans and robots to pass. In such a structured environment, LUCIE would be capable of providing the delivery services of a waitress.

7 References

Getbootstrap.com. (2017). *Bootstrap · The world's most popular mobile-first and responsive front-end framework.*. [online] Available at: https://getbootstrap.com/docs/3.3/ [Accessed 13 Sep. 2017].

Conceptdraw.com. (2017). ConceptDraw Samples | Building plans - Cafe and restaurant plans. [online] Available at: http://www.conceptdraw.com/samples/building-plans-cafe-restaurant [Accessed 9 Jun. 2017].

Forlizzi, J. and DiSalvo, C. (2006). Service robots in the domestic environment. *Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction - HRI '06*, pp.258-265.

Harford, T. (2017). *Rise of the robots: What advances mean for workers - BBC News*. [online] BBC News. Available at: http://www.bbc.co.uk/news/business-39296096 [Accessed 23 Apr. 2017].

Flask.pocoo.org. (2017). *Welcome to Flask — Flask Documentation*. [online] Available at: http://flask.pocoo.org/docs/latest/ [Accessed 13 Sep. 2017].

Flask-socketio.readthedocs.io. (2017). *Welcome to Flask-SocketIO's documentation! — Flask-SocketIO documentation*. [online] Available at: https://flask-socketio.readthedocs.io/en/latest/ [Accessed 13 Sep. 2017].

Flask-sqlalchemy.pocoo.org. (2017). *Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (2.2)*. [online] Available at: http://flask-sqlalchemy.pocoo.org/2.2/ [Accessed 13 Sep. 2017].

GitHub. (2017). strands-project/strands_navigation. [online] Available at: https://github.com/strands-project/strands_navigation [Accessed 18 Jun. 2017].

Halskau, Ø. (2014). Offshore Helicopter Routing in a Hub and Spoke Fashion: Minimizing Expected Number of Fatalities. Procedia Computer Science, 31, pp.1124-1132.

Leedsforlife.leeds.ac.uk. (2017). *COMP1011 Programming for the Web*. [online] Available at: https://leedsforlife.leeds.ac.uk/Broadening/Module/COMP1011 [Accessed 13 Sep. 2017].

Pinillos, R., Marcos, S., Feliz, R., Zalama, E. and Gómez-García-Bermejo, J. (2016). Long-term assessment of a service robot in a hotel environment. *Robotics and Autonomous Systems*, 79, pp.40-57.

Python.org. (2017). *PEP 492 -- Coroutines with async and await syntax*. [online] Available at: https://www.python.org/dev/peps/pep-0492/ [Accessed 13 Sep. 2017].

Pythonclock.org. (2017). *Python 2.7 Countdown*. [online] Available at: https://pythonclock.org/ [Accessed 13 Sep. 2017].

Ros.org. (2017). ROS.org | About ROS. [online] Available at: http://www.ros.org/about-ros/ [Accessed 9 Jun. 2017].

Rusu, R., Gerkey, B. and Beetz, M. (2008). Robots in the kitchen: Exploiting ubiquitous sensing and actuation. *Robotics and Autonomous Systems*, 56(10), pp.844-856.

Strands.acin.tuwien.ac.at. (2017). STRANDS. [online] Available at: http://strands.acin.tuwien.ac.at/software.html [Accessed 9 Jun. 2017].

Strands.readthedocs.io. (2017). Using the Strands Navigation System — Strands Documentation documentation. [online] Available at: http://strands.readthedocs.io/en/latest/lamor15/wiki/Tutorial-materials-1.html [Accessed 18 Jun. 2017].

Tzou, J. and Su, K. (2009). High-speed laser localization for a restaurant service mobile robot. *Artificial Life and Robotics*, 14(2), pp.252-256.

Wiki.python.org. (2017). *GlobalInterpreterLock - Python Wiki*. [online] Available at: https://wiki.python.org/moin/GlobalInterpreterLock [Accessed 13 Sep. 2017].

Wilde, O. (1915) *The Soul of a Man under Socialism.* Maisel: New York

Yin Zhang, Roughan, M., Lund, C. and Donoho, D. (2005). Estimating point-to-point and point-to-multipoint traffic matrices: an information-theoretic approach. IEEE/ACM Transactions on Networking, 13(5), pp.947-960.

8 Appendix A: External Materials

This project made use of existing packages, such as STRANDS topological navigation and Flask web framework.

8.1 Robotics

ROS Operating System: Ros.org. (2017). ROS.org | About ROS. [online] Available at: http://www.ros.org/about-ros/ [Accessed 9 Jun. 2017].

STRANDS navigation, including topological navigation: Strands.readthedocs.io. (2017). Using the Strands Navigation System — Strands Documentation documentation. [online] Available at: http://strands.readthedocs.io/en/latest/lamor15/wiki/Tutorial-materials-1.html [Accessed 18 Jun. 2017].

8.2 Web Imports And Requirements

Flask: Flask.pocoo.org. (2017). *Welcome to Flask — Flask Documentation*. [online] Available at: http://flask.pocoo.org/docs/latest/ [Accessed 13 Sep. 2017].

Flask-SocketIO extension: Flask-socketio.readthedocs.io. (2017). *Welcome to Flask-SocketIO's documentation! — Flask-SocketIO documentation*. [online] Available at: https://flask-socketio.readthedocs.io/en/latest/ [Accessed 13 Sep. 2017].

Flask-SQLAlchemy extension: Flask-sqlalchemy.pocoo.org. (2017). *Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (2.2)*. [online] Available at: http://flask-sqlalchemy.pocoo.org/2.2/ [Accessed 13 Sep. 2017].

Also all of the dependencies, as determined by pip, are required for these modules.

8.3 Web Included Within Repository

Bootstrap CSS and JS: Getbootstrap.com. (2017). *Bootstrap · The world's most popular mobile-first and responsive front-end framework.*. [online] Available at: https://getbootstrap.com/docs/3.3/ [Accessed 13 Sep. 2017].

SocketIO JS: Socket.io. (2017). *Socket.IO*. [online] Available at: https://socket.io/ [Accessed 13 Sep. 2017]

JQuery: jquery.org, j. (2017). *jQuery*. [online] Jquery.com. Available at: http://jquery.com/ [Accessed 13 Sep. 2017].

9 Appendix B: How ethical issues were addressed

Robotics in the service industry is a potentially sensitive topic due to fear of robots taking people's jobs, however LUCIE is incapable of working without support-staff and is comparable to existing novelty delivery mechanisms in restaurants such as conveyor belts, miniature trains and miniature roller-coasters. Although such restaurants exist, they have not become wide-spread.

When testing LUCIE in public environments, inquiries were responded to focusing on the navigational aspects of the project to avoid raising concerns. Testing was also confined to the university students and presented only as a novelty.

Military applications would be minimal as this project is less advanced than existing navigation and delivery systems, instead it focuses more on human, robot interaction and acceptance. LUCIE, like the fictional daleks, is incapable of traversing even a small step.

Data gathered was in the form of textual logs and photographs included in this report. Care was taken not to include an humans within the photographs.

10 Appendix C: Product READMEs

10.1 Topological Map Maker

A new topological map can be created alongside a new metric map using the tmux script "lucie_start_mapping.sh" and following the instructions below. The program is run as `./lucie_start_mapping.sh name_of_map path_to_robot_files path_to_database` where `name_of_map` is what the map file will be called (`{name_of_map}.yaml`) and the name of the WayPoints in the database. `path_to_robot_files` is where the yaml and other map files will be stored, the yaml hard-codes this path into its location of the image (pgm) file. If either the database directory or robot_files directory does not yet exist then the script will create it for you. If any arguments are missing, a usage example is printed to stdout and the process exits.

10.1.1 Tmux useful keys

- `ctrl & B then 0` change to session 0. Replace 0 with any number up to 7 to change between the sessions.
- ```ctrl & B then -``` change between panes using the arrow keys.
- `ctrl & B then D` exit the tmux session. Note the session is still running and needs to be killed. Run ./kill_tmux.sh to really exit.

10.1.2 Sessions in Tmux

There are 7 sessions running in tmux, which are designed for the different tasks required in building a topological map.

10.1.2.1 Session 0

This runs roscore and htop to monitor processes. Both of these scripts run automatically so if there are no errors then change to session 1.

10.1.2.2 Session 1

The main pane runs mongodb, which will create your database at the path given and allow the WayPoints to be inserted once they have been created. Run it and leave it until you've finished.

The second pane contains the command to run robomongo, which allows you to view the database in more detail. It is not required.

10.1.2.3 Session 2

This is robot_bringup and provides the lasers and joystick.

10.1.2.4 Session 3

This is to enable the cameras. The locations of the cameras are set by the variables `$HEADCAM_HOSTNAME` and `$CHESTCAM_HOSTNAME` at the start of the script.

10.1.2.5 Session 4

The first pane runs slam gmapping, which scans the environment to build a metric map. It is this that will provide the data for {name_of_map}_raw.pgm and {name_of_map}_raw.yaml files later.

The second pane script is to get WayPoints simultaneously while mapping via the joypad. Run session 5 before beginning mapping.

10.1.2.6 Session 5

This is to run Rviz with some useful options open. It will allow you to watch the generation of the map.

10.1.2.7 Making the map.

LUCIE can be driven by holding [LB] and using the left joystick. If the bumper is hit the motors will stop. Make sure the bumper isn't pressed and hit [start].

WayPoints are set by pressing [LB] & [B], "JOY" will be printed to stdout in Session 4, right pane.

10.1.2.8 Session 6

This is part of saving the map. The top pane will save the map, creating the `{name_of_map}_raw.pgm` and `{name_of_map}_raw.yaml` files. The bottom pane will crop the map to create `{name_of_map}.pgm, {name_of_map}.yaml`.

To get the WayPoints.csv file, exit the process running in session 4 on the right hand side, which is the joy_map_waypoints.py program, by pressing `ctrl & C`. This will then save the `{name_of_map}_waypoints.csv` file as it exits.

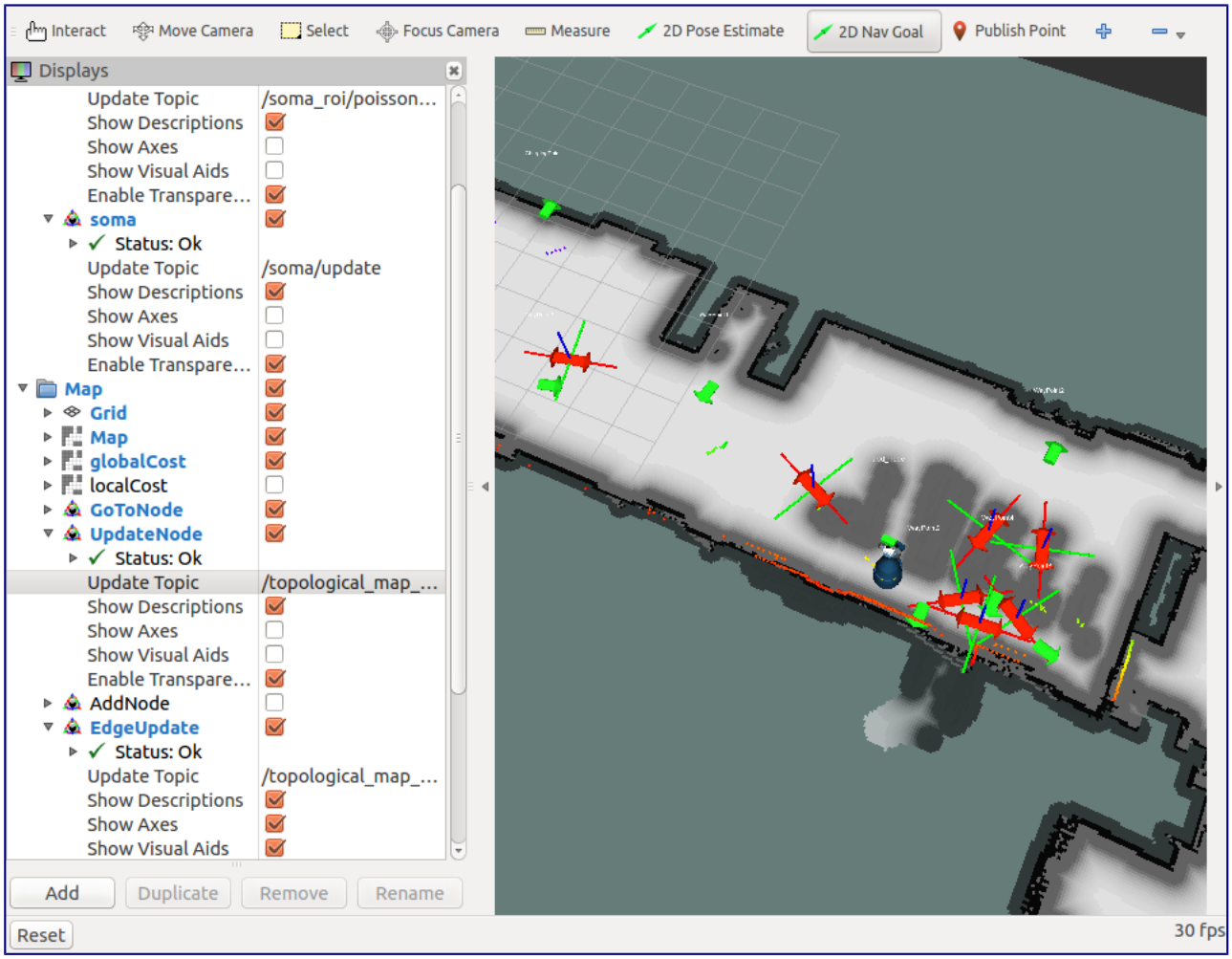
10.1.2.9 Session 7

The first pane creates a tmap file from the other files already made. The second pane then inserts this data into the database.

10.1.3 Wrapping up

The map files have been created in TopologicalMapMaker/robot_files/ and can be moved to your preferred directory. The WayPoints have been inserted into the given database and called "`name_of_map`". The running processes can now be killed. To kill tmux, use the provided script: kill_tmux.sh.

It may be beneficial to clean the map image file using gimp before using it. Node and edges can be edited in Rviz by checking the 'UpdateNode' and 'EdgeUpdate' boxes. Under 'Update Topic' subscribe to '/topological_map_add_rm_node/update' and 'topological_map_edges/update' respectively.



10.2 Robot Waitress: Navigation

This repository contains the scripts and files necessary for running Topological Navigation on LUCIE. The main script is used as `./lucie_start_navigation.sh name_of_topological_map`. It assumes the maps have been created with TopologicalMapMaker as it uses the `name_of_topological_map` variable following the naming conventions for the files created there. The script would need to be manually adapted for other maps. The tmux script can be launched and each session run. The final session gives the option of giving LUCIE a navigation objective.

10.2.1 Tmux useful keys

- `ctrl & B then 0` change to session 0. Replace 0 with any number up to 7 to change between the sessions.
- `ctrl & B then` → change between panes using the arrow keys.
- `ctrl & B then D` exit the tmux session. Note the session is still running and needs to be killed. Run ./kill_tmux.sh to really exit.

10.2.2 Sessions in Tmux

There are 6 sessions running in tmux, which are designed for the different tasks required to run navigation with a topological map.

10.2.2.1 Session 0

This runs roscore and htop to monitor processes. Both of these scripts run automatically so if there are no errors then change to session 1.

10.2.2.2 Session 1

The main pane runs mongodb, which will create your database at the path given and allow the WayPoints to be inserted once they have been created. Run it and leave it until you've finished.

The second pane contains the command to run robomongo, which allows you to view the database in more detail. It is not required.

10.2.2.3 Session 2

This is robot_bringup and provides the lasers and joystick.

10.2.2.4 Session 3

This is to enable the cameras. The locations of the cameras are set by the variables `$HEADCAM_HOSTNAME` and `$CHESTCAM_HOSTNAME` at the start of the script. There is also the option of not running the cameras, an alternative to using the headcam with the strands_cameras.launch file is included in the second pane.

The second pane gives the option of running a people tracking program, which can show where people are in Rviz. This should only be run if `$HEADCAM = False` to avoid conflict.

10.2.2.5 Session 4

The first pane runs strands UI, which integrates with monitored navigation to ask for help when LUCIE is stuck.

The second pane is a convenient script to launch firefox at the correct url for the UI.

10.2.2.6 Session 5

This launches the topological navigation, once this is run LUCIE can be sent navigation objectives. The camera used is set in `$NAV_CAM` at the top of the file.

10.2.2.7 Session 6

The first pane brings up a customized Rviz that loads some useful topics. This makes it easy to set an estimated position (2D pose) for LUCIE, send her navigation goals that are not part of the topological map as well as edit WayPoints and the Edges between them.

The second pane contains the command to issue LUCIE with a navigation goal to reach WayPoint 1.

10.2.3 Exiting

To exit, the running processes can be killed with `ctrl & C`, tmux can be killed using the provided script: kill_tmux.sh.

10.3 Robot Waitress UI

The UI is initialised using the file `waitress_gui.py` as the entry point. The GUI can be accessed by any suitable device on the same LAN as LUCIE. This enables admin staff to issue LUCIE with commands and monitor her behaviour from a stationary machine.

```
rosrun waitress_ui waitress_gui.py
```

10.3.1 Configuration Options

There are constants that can be edited in `config.py`:

- `MENU = [{"name": "Foo", "image": "foo.jpg"}]` used to create the menu. Each item has a name and an image file name. All image files should be available in the `static/images/` directory.
- `HUB = 'WayPoint1'` tells LUCIE which WayPoint to use as the hub, i.e. where to collect delivery items from.
- `NUMBER_OF_WAYPOINTS = 14` tells the UI how many WayPoints are in the Map.
- `ONE_MACHINE = True` Run the UI solely on LUCIE, this will automatically login for admin features.
- `PIN = 1111` set this to a new code, it is what admin staff can use to login. This is not a secure method.

- `WONDERING_MODE = True` make LUCIE go to a random node inbetween orders. If False, LUCIE will return to the HUB.
- `TWITTER = True` provides the link to the Twitter page, use this if the Twitter program is intended to be used.
- `STRANDS_UI_URL` allows the existing helper functions to be embedded in the generic navigation page. This is the url given when strands_ui is launched.