Product: DR.PHIL
Team: Klean
Student: Harry Wixley

Abstract

DR.PHIL (Disinfecting Robot Prioritising High Interaction Locations) is an autonomous robot which can navigate a single floor, and disinfect door handles.

My main contributions to this project were being the software leader for my group, developing the entire iOS app by myself, and developing the vision system.

The main lessons I learned were leadership, communication and organisation. This was the first large scale group project I have worked on and being assigned as the Software Leader required a significant responsibility to ensure all team members were on top of their work in order to stay on top of our Gantt chart and meet the demo deadlines.

1. Contribution

In this project I was allocated as the software leader, and was involved in many software tasks from all different domains. These tasks included: logo design, the task management API, building the custom CAD door for simulation, building all the Gazebo worlds, developing the door and handle recognition model, building the sanitisation path planning algorithm by myself, and building the iOS app by myself.

My most significant contributions to this project were being the software leader for my group, developing the entire iOS app by myself, and developing the vision system.

1.1. Software leader

Being software leader entailed taking care of large decisions with the other department leads, assigning the weekly software tasks to team members, checking team members progress on software tasks, and reviewing all pull requests in our GitHub repository to ensure they were of good quality before merging.

Thus as software lead at the beginning of each week I would send a message to the Slack software channel detailing the goals for the given week, and what tasks needed to be done. To ensure all my team was happy with the tasks they were allocated I used a poll at the beginning of each week to allow everyone to choose what they wanted to do. When there were too many people wanting to do a given task I would just ask if anyone would be willing to change, luckily all of my team were very easy and so there were no conflicts in this respect. If there was ever a task that was not chosen I or Maks would take it. Once all tasks were assigned I would add these to Trello with a more in depth task description, quantify a goal for the task, and specify a deadline for completing the task.

To ensure all my team were making progress and able to do the tasks they were allocated I would send messages to the Software channel in Slack asking how people were getting along. This worked very well as it served as a reminder for people to get on with their tasks and posed an opportunity for them to ask for help.

Lastly, in order to ensure good code version control in our GitHub repository, at the start of semester I posted some guidelines to the group to ensure that members used Git appropriately. This mainly entailed: that all new features should be put on a new branch, all commits should have informative descriptions, and that pull requests should only be made after sufficient testing. This worked out great as everyone followed these guidelines, and all pull requests were reviewed by me meaning that I was able to test all the code on my own system to ensure it was of good quality before merging it into the main branch.

1.2. iOS app

I developed the iOS app entirely by myself. This involved building all the user interfaces, adding functionality to these user interfaces, and integrating Firebase to allow for fast and secure communication with the robot.

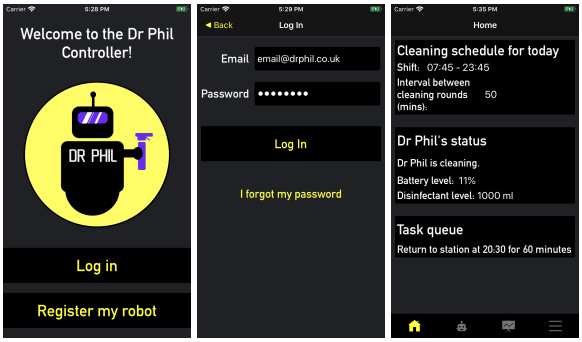

Figure 1. iOS app Start, Login and Home pages respectively

Due to the fact that our robot is fully autonomous the aim of the app was to provide a very simple way for users to monitor and control the robot. The only control needed was setting cleaning schedules, and being able to make the robot stop and return to it's station. The app also allowed a user to monitor the status of the robot by retrieving real time updates from DR.PHIL this included: location info, task queue info, cleaning statistics, power usage, and disinfectant usage.

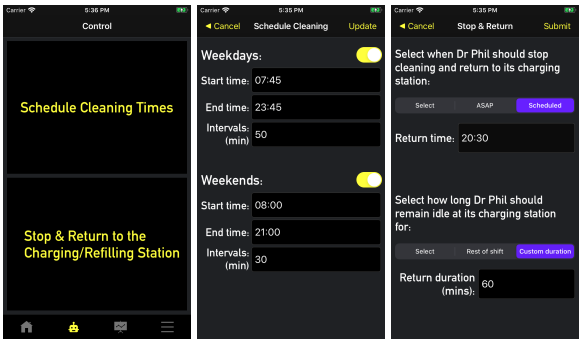

Figure 2. iOS app Control section (schedule cleaning, stop & return)

We initially planned to use a ROSbridge server to connect the app and the robot, however, I decided using Firebase's Firestore (an online NoSQL database) would be a far more useful option due it's scalability, security, and vast amount of features (in particular that of account creation).

In the account creation process I had to find a secure and easy way to verify a user's identity and what robot they wanted to connect

to. I decided to do this using predefined robot credentials (these credentials were a unique robot ID, and a unique key). Thus in order to be able to create an account a user would first have to verify their robot's identity by entering their robot ID and key (which would be included in the packaging for the robot). The app would then check the database to see if an unregistered robot with the entered robot ID existed, if so it would check if the entered key matched that in the database. If this was successful the user could then create an account.

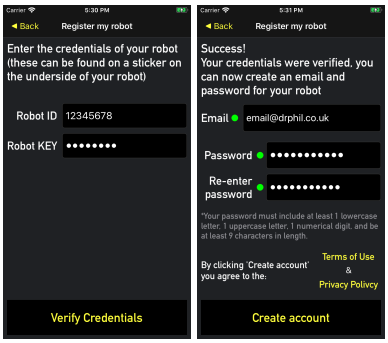

Figure 3. iOS app Create Account steps

Furthermore, I had to structure the database not only to maximise read/write efficiency but also so that the data read/write rules could be configured to maximise security. These rules define which directories users have access to and what privileges they have in these directories (read/create/edit/delete). Without these rules malicious users could potentially try to tamper/spoof database read/write requests and thus manipulate other users' data to try control their robot. These rules were configured by only allowing authenticated users to access data, and only allowing these users to access data that they own (ensuring their user ID matches that stored in the document). Since every user has a unique user ID that is set and stored by the Firestore API adapting this user ID for a database read/write request to access other users' data would be infeasible.

When building the user interface my main priorities were simplicity, accessibility, and aesthetics. I used a distinctive colour scheme in line with our brand colours so it was not only aesthetically pleasing but also easy to read. I also made sure to use large text for all the UI elements to ensure accessibility for those with poor eyesight. I used dynamic UI designs in order to ensure each page would be able to resized for different sized screens. This was done by adding a dynamic keyboard feature, and using UI element constraints. The dynamic keyboard feature is used for when a user inputs text into an text entry field located at the bottom of the screen (resulting in the software keyboard covering the user's view of the text field), this was done by moving the screen view upwards if the software keyboard ever covered a text field. UI element constraints were used to define the relative spacing and sizing between different UI elements on a given page to ensure that all the UI elements were adapted to each screen to maximise legibility, screen coverage, and aesthetics.

Integration into the full system was very simple due to Firebase's large array of platform support. We just had to create a ROS client using the Firestore API to connect to the database. From then given the way the database was structured (documents made identifiable using the robot's ID) it was very easy for the robot to retrieve it's data as it all it needed was it's predefined RID value.

1.3. Door and handle recognition model

For our door and handle recognition model I generated the synthetic dataset, built various classification models on this dataset and helped train the YOLOv3 model.

The problem we found when facing developing a door and handle recognition model was that our custom door was quite dissimilar in shape from regular doors so using a real door dataset would prove rather inaccurate upon implementation with this DR.PHIL prototype. Thus I was tasked to develop a synthetic dataset that used our custom door.

To create a synthetic dataset I decided to just extract frames from our robot's front camera while it explored various Gazebo worlds. In order to do this I first created 6 new large Gazebo worlds filled with lots of different types of obstacles. These obstacles consisted of random items (such as cones, pallets, boxes etc.) and more typical obstacles one would find in a building (such as tables, chairs, and people). I made sure to include lots of people in these worlds to ensure that the model was resilient against falsely classifying a person as a door for obvious safety reasons. These worlds were made purely for generating this dataset and not for testing our implementation as otherwise the accuracy of this model would be biased given it would be classifying images from worlds it had seen before (which would not be the case in a real life scenario where DR.PHIL would have to be able recognise doors it has never seen before).

After this, my teammate created a ROS node for extracting frames from the robot's camera that I could execute while exploring the various worlds. In order to prevent collecting samples that looked very similar we had to ensure that we only extracted an image every 20 frames. Once this was configured I explored all the various Gazebo worlds using the turtlebot's teleoperation package (allowing me to control it using my WASD keys) and ended up with over 1000 images. However, we wanted our single model to be versatile at classifying doors in both simulation and the physical world so we decided to integrate real door images into this dataset too.

Now since we decided to use a bounding box recognition model (so we could localise the location of the door and handle for navigation) we had to label each of our samples manually using YoloMark. Once the labelling was complete, I created a script to split the dataset into train and test splits (split of 70:30) whilst ensuring the class priors in each of these splits were equivalent (equal number of door and non-door samples in each test and train split) so that there was no bias to either class in our model.

After training the YOLOv3 model on this new dataset, it ended up performing very well as shown by its implementation in our real-time footage. The only downfall to this model was that it was very computationally expensive. To solve this I thought of using a binary classifier model (such as a logistic regression or SVM model) to determine if a frame contained a door (as it would be much less computationally expensive than YOLOv3), and if so, only then run the YOLOv3 model on this frame to localise the door and the handle. Despite this seemingly elegant solution upon implementation testing we found these models performed poorly for detecting doors from afar. This was because these models were overfit to the closer samples in our dataset (unlike YOLOv3 which combats overfitting using batch normalisation and weight decay) so we decided to simply use the YOLOv3-Tiny-Prn model for this prototype (less computationally expensive than YOLOv3 since it uses less convolutional layers), and for the commercial product integrate a USB accelerator to allow us to use the YOLOv3 model.

Integration into the main system was relatively simple as all we had to do was create a ROS node that connected to our turtlebot's camera and then execute our YOLOv3-Tiny-Prn model against

each frame.

Figure 4. Example outputs of the YOLOv3 and YOLOv3-Tiny-Prn models respectively

2. Lessons learned

Despite all the stress of the various demo deadlines and group organisation I found this course to be extremely insightful and fun. This was my first experience doing such a large scale informatics group project, being a leader for an informatics project, working with robotics, and integrating so many hardware/software components into a single working system.

2.1. Technical skills

This was my first time ever collaborating on such a large scale project, working with ROS, working with CAD, working with Gazebo, developing an object recognition model and integrating so many different software/hardware components from various domains into a single working system.

The main technical challenge of working with a large group was ensuring everyone knew how to use GitHub effectively, to ensure that version control was good and no sensitive code was overwritten accidentally. Since this was my first large scale collaborative informatics project I learned a lot about the different tools Git has to offer. When developing the large dataset for our door and handle recognition model I learned how to use Git LFS (Large File System) which is used for pushes containing 500MB+ of data. Furthermore, I became familiar with how to integrate CI (Continuous Integration) tests into a repository as an easy way to automate tests on a branch before merging it into main. Since I was in charge of all Pull Request reviews this became very useful and saved me a lot of time.

This was my first time ever working with ROS which was very fascinating and insightful. I was initially quite apprehensive of using ROS as it seemed very complicated but after help from some of my team members who had taken the IVR (Introduction to Vision Robotics) course last semester I became quite comfortable with it.

It was my first time using CAD to create 3D models. I was thrown in the deep end as I had to replicate the physical door produced by our technician and hardware team for simulation. I managed to grasp this quite quickly due to my experience with SketchUp. However, the main difficulty I found with this task was learning how to retain colour in 3D model files. This was essential to ensure our simulation door was consistent with our physical door so that our door and handle recognition model would perform the same. After a lot of fiddling and research I realized that STL files could not actually retain colour and that I needed to export my 3D models as Collada files. This entailed using a different CAD development software that supported the exportation of Collada files.

In addition, it was my first time working with Gazebo to develop

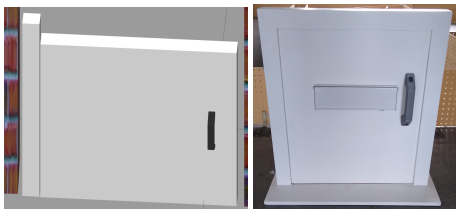

Figure 5. The CAD door I built and the physical door respectively

3D worlds with working physics. This was quite difficult as the Gazebo world editor software is quite buggy and clicking the wrong combination of tools would result in a program crash. However, after a tedious learning curve I got the hang of it and managed to create all the simulation worlds with the custom CAD door I made.

Figure 6. An example of one of the Gazebo worlds I built - mimicking an unusual shaped room filled with obstacles

2.2. Teamwork skills

The main skills I learned from working in a group for such a large scale project were leadership, project management, group management, and communication.

With the new Covid protocols in place it meant we had to work with students we had never met before in real life so being able to communicate effectively in order to stay on top of our scheduled tasks was crucial.

I have been a captain of my sports teams in the past, however, being a leader in the domain of informatics was completely new to me and required many different responsibilities. I learnt how to manage a team under a time-sensitive schedule, allocate weekly tasks to my team, and ensure that everyone was happy with what tasks they were allocated. I found the easiest way to do this would be by using polls for people to choose what they would want to do, and resolve any conflicts in these polls if presented.

Project management required weekly discussions with the department leads to address what tasks needed to be done in each department for the given week. This project management was essential to stay on top of our Gantt chart, and thus meet our milestones for each of the demo days and industry day.

Group management required group meetings for large decisions/progress updates, and splitting the group into its different departments to optimise peoples strengths and make things more efficient.

Lastly, working on such a large scale project with a group requires streamlined communication channels. This was done by doing weekly group meetings in our private Teams channel, and all other communication via Slack which was subdivided into different channels for each department. These proved very effective as all of our team members were very engaged in these throughout the semester.

Appendix

YOLOv3 - You Only Look Once (version 3). An object recognition model that uses convolutional neural networks to localise objects in an image.

References

Alexey Bochkovskiy, Darknet - neural networks for object detection, Github Repository available at https://github.com/AlexeyAB/darknet

Miguel Arduengo and Carme Torras and Luis Sentis(2019),Robust and Adaptive Door Operation with a Mobile Manipulator Robot, Github Repository available at https://github.com/MiguelARD/DoorDetect-Dataset

Joseph Redmon, Ali Farhadi(2018), YOLOv3: An Incremental Improvement, available at https://arxiv.org/pdf/1804.02767.pdf

Chien-Yao Wang1, Hong-Yuan Mark Liao1, Ping-Yang Chen2, and Jun-Wei Hsieh(2019), Enriching Variety of Layer-wise Learning Information by Gradient Combination https://openaccess.thecvf.com/content_ICCVW_2019/papers/LPCV/Wang_Enriching_Variety_of_Layer-Wise_Learning_Information_by_Gradient_Combination_ICCVW_2019_paper.pdf

Chien-Yao Wang1, Hong-Yuan Mark Liao1, Ping-Yang Chen2, and Jun-Wei Hsieh(2019), Enriching Variety of Layer-wise Learning Information by Gradient Combination, https://openaccess.thecvf.com/content_ICCVW_2019/papers/LPCV/Wang_Enriching_Variety_of_Layer-Wise_Learning_Information_by_Gradient_Combination_ICCVW_2019_paper.pdf

SDP Group 13, 2021, Demo 1 report

SDP Group 13, 2021, Demo 2 report

SDP Group 13, 2021, Demo 3 report

SDP Group 13, 2021, Demo 4 report