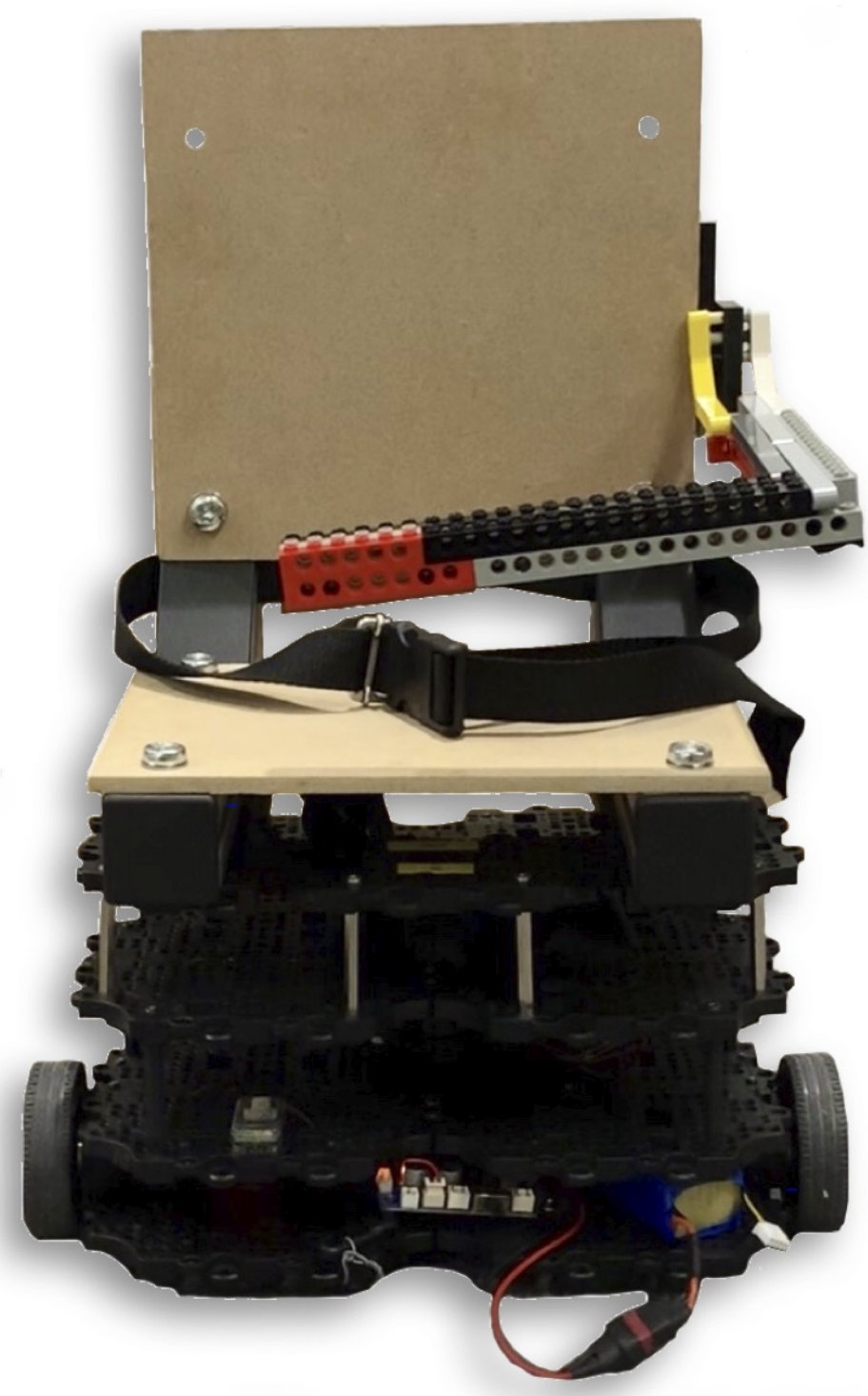
IAN

Interactive Airport Navigator

User Guide



User Guide

Group 6

Group Members

Yussef Soudan
Matt Shostak
Yuexiang Jin
Eloise Milliken
Ethan Soreide
Erodotos Terpizis
Ion Bruckner
Maria Sliacka
Daragh Meehan
Chongli Shen

Contents

1 Introduction

IAN is the interactive airport navigator created by Group 6 of the System Design Project course in 2020. IAN was created to help elderly travellers, primarily, reach points of interest in airports by combining autonomous navigation with a chair, thereby making IAN an autonomously-moving chair.

1.1 Overview

The full system consists of:

- **The Turtlebot and its Raspberry Pi** - contains the core navigation logic.
- **The User Pi** - a Raspberry Pi needed by the Turtlebot Pi to act as the 'master' for the Turtlebot's navigation. The provided User Pi is labeled 'Meowth'.
- **The Server Pi** - a Raspberry Pi that contains the database and acts as an airport server. The provided Server Pi is labeled 'Clefairy'.
- **The UI** - lives on the User Pi and is displayed on a 7" touchscreen.
- **The Camera** - scans boarding passes, and is connected to the User Pi.
- **The User Seat** - built on top of the Turtlebot.

The system network is shown in Figure 1.

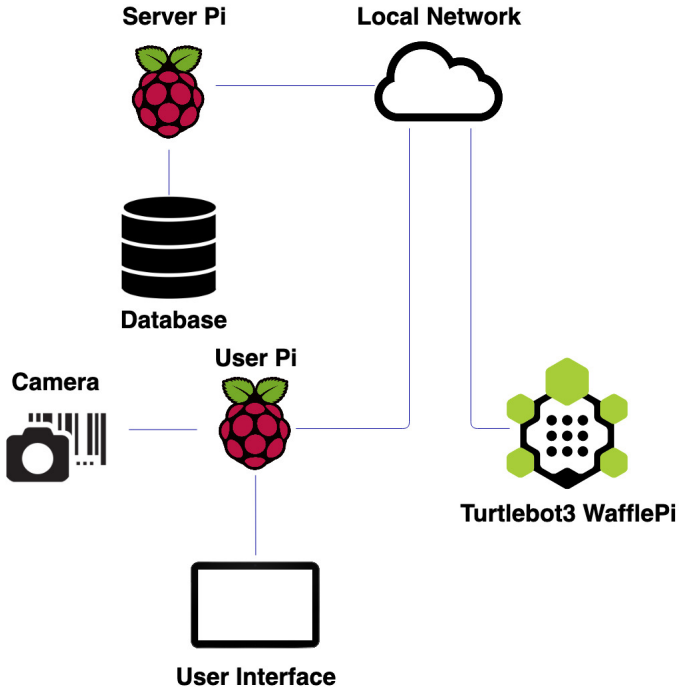


Figure 1: *User Pi*: acts as the 'master' for the Turtlebot: handles all the business logic, is connected to the camera which scans boarding passes, and is connected to the screen which displays the user interface. *Server Pi*: acts as a cloud server; it contains the database that stores all the flight and user information. *Turtlebot3 WafflePi*: the Turtlebot's Raspberry Pi, which contains the core navigation functionality of the Turtlebot. *Local Network*: SDProbots, across which the 3 Pi's communicate.

1.2 Intended Functionality

The user approaches the robot and presses the Start button on the screen. The user will be asked to scan their boarding pass. After, the system will check if the user's boarding pass is valid by querying the remote database. If successful, the user's corresponding flight information and status will be displayed. The user will then be asked to choose where IAN should take them. They can choose to go to their gate or to other points of interest (e.g. restaurants). Once the user selects their destination, IAN should begin moving there. A change of flight status (e.g. gate number or flight time) could happen mid-journey. If it does, the user will be notified by the means of a screen pop-up. Once IAN reaches the destination, the user can press the exit button which returns IAN back to its initial position. At all times throughout the journey, the user will be able to pause and change their destination.

The intended functionality of the prototype is demonstrated by the flow chart in Figure 2.

2 Setup

Assuming you have the provided equipment outlined in section 1.1, you will now be able to setup each component of the system. However, if you choose to use your own, refer to Appendix B before starting the setup.

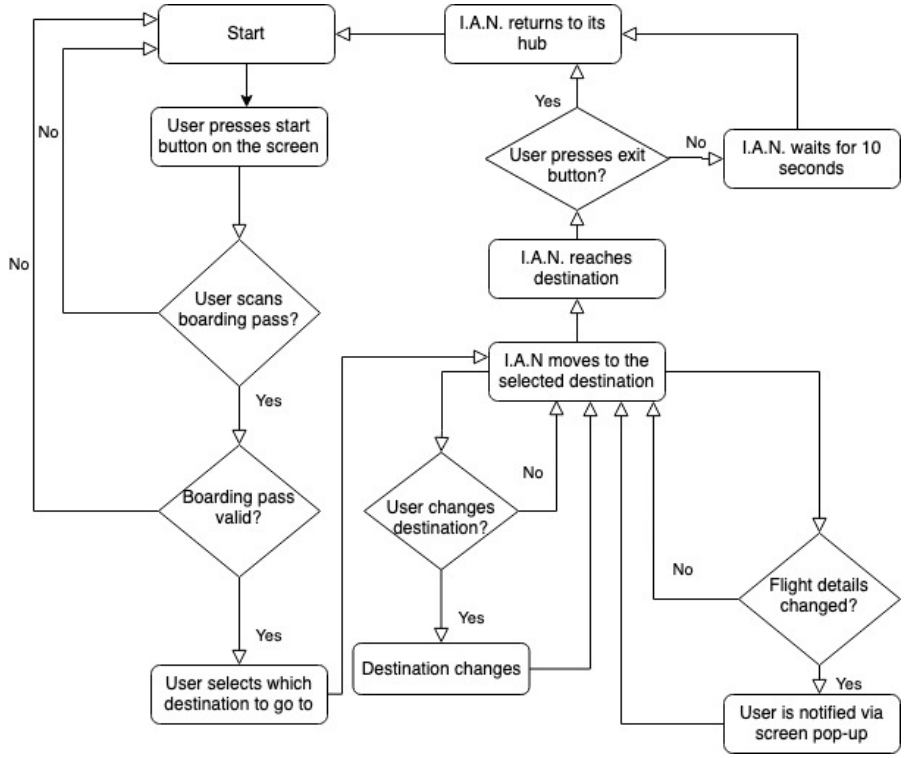


Figure 2: A flow chart of the intended functionality of IAN

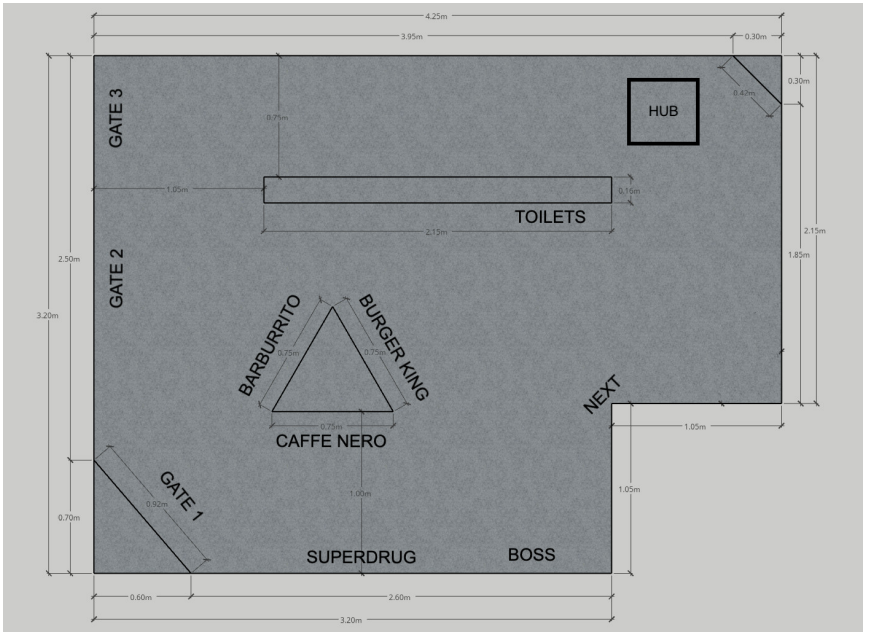


Figure 3: The physical environment used in the making of IAN with the 'HUB' being the start position of IAN, courtesy of the School of Informatics.

2.1 Software

The software component of the system comprises the navigation, UI, the database, and the boarding pass scanning software.

The Raspberry Pi connected to the screen is the **User Pi**, and it acts as the 'master' for the Turtlebot. Both the Turtlebot Pi and the User Pi need to be connected on the same local network for them to communicate, and you need to know their respective IP addresses. This can be done by running `ifconfig` in a terminal. Do this for all the Pi's and the Turtlebot Pi. Once you know their respective IP's, you need to complete the network configuration setup between the User Pi and the Turtlebot Pi by following the PC Setup Robotis tutorial, section 6.1.4 [1].

Now that you know the various IP's, you will need to change the IP addresses used throughout the code. This can be done by changing the User Pi, the Turtlebot IP and the Server Pi IP's in the `env.py` file in the `~/sdp-ian/` directory on both the User Pi and the Server Pi.

Additionally, you will need a personal device connected to the same network with the repository [2] on it with the updated IP's in the `env.py` file.

Finally, you will have multiple mock boarding passes provided that can be scanned. More on making your own boarding passes in Appendix D.

2.2 Physical Environment

The physical environment is where the Turtlebot roams. There are several coordinates already in the code of various locations (outlined in the UI) that IAN can go to – these are all based on the environment outlined in Figure 3. You could build the same environment given the measurements in the figure. The physical environment used when building this system was wooden; however, any other solid material should work. **The walls should be at least as high as the LIDAR sensor on the Turtlebot: 28.1cm.** If you opt for building your own physical environment, the coordinate mapping

process and the process of integration with the UI are outlined in Appendix C.

2.3 Configuration & Starting the System

If you have followed the above steps, the system should be correctly setup. Here's a checklist to ensure things will work together as intended:

- The User Pi is running, and is connected to the local network, the display screen, and the webcam.
- The Server Pi is running, and is connected to the local network.
- The Turtlebot is in its start position (the hub – see Figure 3), is running, is connected to the local network, and has a fully charged battery.
- You, the user, have a valid boarding pass. You also have a personal PC or laptop that is connected to the same network as the Pi's, and has the repository [2] on it.

Once the configuration is complete, you need to do the following on the **User Pi**:

1. Open a terminal and run:
   ```
   roscore
   ```
2. Open another terminal, and ssh into the Turtlebot (if you are asked for a password, it's `turtlebot`):
   ```
   ssh pi@<turtlebot-ip>
   ```
   and then run:
   ```
   roslaunch turtlebot3_bringup turtlebot3_robot.launch
   ```
3. Open yet a third terminal and run the following **two** commands:
   ```
   export TURTLEBOT3_MODEL=waffle_pi
   roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=
   $HOME/map.yaml
   ```

You can now start the system by running the following on a User Pi terminal:

```
python ~/sdp-ian/ui/IanUi.py
```

From here, you can start interacting with the system as outlined in Figure 2. You could start causing delay to all flights, for the user to receive a pop-up on the screen by launching a terminal **on your device** that's on the same network and running:

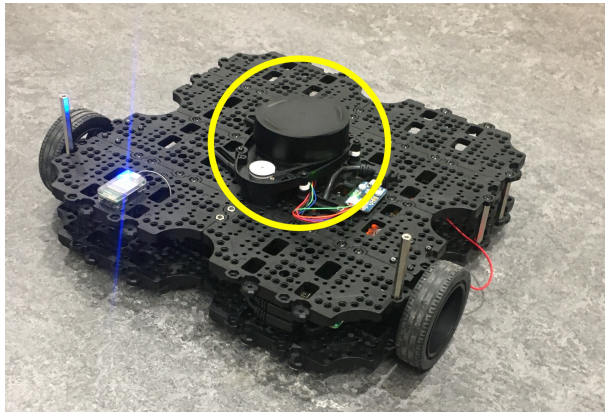


Figure 4: The LIDAR sensor, circled in yellow, on the second level (exposed) of the Turtlebot.

```
python ~/sdp-ian/Live/dice/delay_flights.py
```

When you are done, you could restore the flight times, and statuses, by running:

```
python ~/sdp-ian/Live/dice/restore_flights.py
```

3 System Restrictions

Turtlebot Payload

The maximum additional weight a WafflePi Turtlebot can carry is 30kg. The Turtlebot provided has a chair weighing roughly 4kg, which means you can add around 26kg of weight to it.

Sensor Blockage

The Turtlebot uses a saved map of its environment as well as the LIDAR sensor (shown in Figure 4) to navigate and avoid obstacles. If the LIDAR sensor, placed on the second level of the provided Turtlebot, gets blocked in any way (for example, by a set of wires or a piece of wood), the Turtlebot will stop as a safety precaution. Therefore, you need to make sure that nothing is in the way (except for the metal rods supporting the higher levels).

Battery Constraints

Assuming you use the system as intended, the Turtlebot battery should last for up to 30 minutes. You are provided with another battery, which should be kept charged when not in use. When changing the batteries, make sure

to turn the Turtlebot off first, as taking out the battery while it's on will result in the Turtlebot falling out of sync with the clock.

User Pi Overheating
Since it handles the intensive navigation logic and the UI software, the User Pi will shut down due to overheating after about 30 minutes of use. To prevent the system from overheating you can use a cooling device, such as a fan.

4 Usage Instructions for Clients

To start using IAN, press the "Start" button on the screen in front of the chair. Press the "Touch to Continue" button to progress through the explanation of what IAN is.

4.1 Scanning a Boarding Pass

You will then be prompted to scan your boarding pass. Place the barcode on the boarding pass in front of the camera and wait for it to be scanned. If successful, your flight information will be displayed. If unsuccessful, ensure that the barcode is not folded, and that it is not too far away from the camera, and try again.

4.2 Viewing Your Flight Information

Your gate, name, flight number, and departure time will be displayed after scanning your boarding pass as well as during your journey. See Figure 5.

4.3 Going to Destinations

1. Press the "Go Somewhere" button on the flight information screen.
2. Press on an icon representing a destination category. See Figure 6.
3. Press on an icon representing a specific destination.
4. After ensuring that your seatbelt is fastened, confirm your destination by pressing on the blue check mark. IAN will start moving.
5. Your flight information as well as your current location and destination will be displayed. See Figure 7.
6. Press the "Pause" button while IAN is moving to stop your journey. You can also press the "New Goal" button to change your destination.
7. After IAN has reached its destination, the journey will terminate.

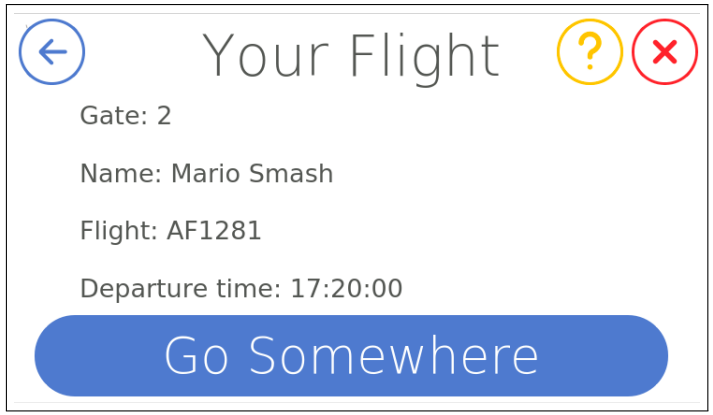


Figure 5: The flight information screen. Your gate, name, flight number and departure time are listed. Press the "Go Somewhere" button to select a destination. On the top-left of the entire screen is the Back button. You can press this to return to the Start screen. On the top-right of the entire screen are the Help button (question mark) and the Exit button (cross).

Figure 6: The location selection screen. In the center are the location categories. From top-left clockwise there are: gates, toilets, shops and restaurants. On the top-left of the entire screen is the Back button. You can press this to return to the flight information screen. On the top-right of the entire screen are the Help button (question mark) and the Exit button (cross).

8. You will be asked if you will need further assistance in the future. Press "Yes" to keep IAN where it is. Press "No" to stop using IAN.

Figure 7: The journey viewer screen. Your flight information is displayed on the bottom-left. Above this is the "Pause" button used to pause your journey. You can press the "New Goal" button to change your destination. On the right is a map indicating where you are now (the circle) and where the destination is (the square). On the top-right of the full screen are the Help button (question mark) and the Exit button (cross). After pressing "Pause", you can press the "Resume" button located in the same position to resume your journey.

9. You may receive updates to your flight at any time. If this happens, a pop-up window will appear. Read the information and press the cross button to close the window.

4.4 Getting Help

Press on the question mark on the top-right of the screen to get help. A list of frequently asked questions will be displayed. See Figure 8.

4.5 Exiting and Returning IAN

The Exit button (cross) on the top-right of the screen will prompt you to return the robot to where you first started using it, referred to as the "hub". A confirmation window will appear. Press the BACK button to cancel the action and return to the previous screen you were on. Press the EXIT button to make the robot go to back to the hub. See Figure 9.

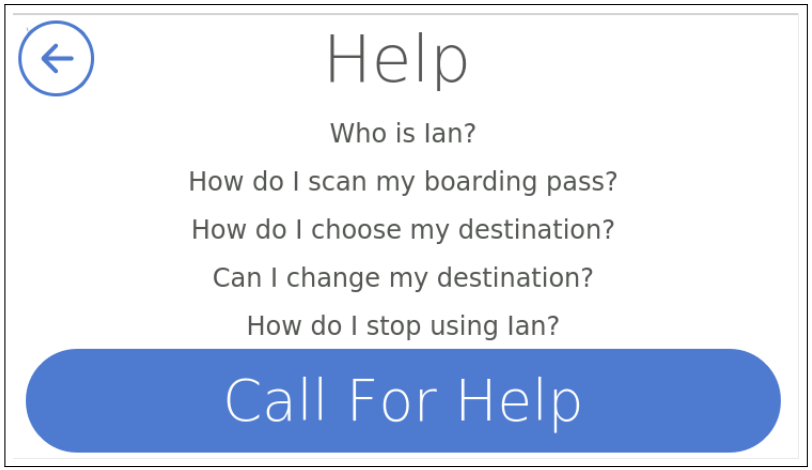


Figure 8: The help screen. A list of questions is displayed. Press on a question to get the answer. On the top-left of the entire screen is the Back button. When pressed, this return you to the screen you were previously on.

Figure 9: The Exit screen. Press the BACK button to return to the previous screen. Press the EXIT button to stop using IAN.

IMPORTANT: Please make sure that you have finished using the robot before doing so. Ensure that you are not sitting on the chair and that nothing is attached to it, such as a bag handle, before pressing EXIT. If you have any doubts, ask a member of staff.

5 Troubleshooting

The table below is broken down into subsections displaying where the navigation, database, and user interface may be misbehaving, and colour coded **respectively** from lightest to darkest. Any issue that does **not** fall into one of these three categories will **not** be colour coded, and appears at the end of the table. It should be further noted that the following problems will most likely be faced by the developer rather than the client, unless otherwise stated.

Issue	Cause	Solution
The robot keeps rotating or will backtrack in the general destination area, and is unable to identify that it has reached the goal.	There is a parameter tuning mistake.	The following parameters must be incremented: 1) `yawgoal_tolerance` must be > 0.3. 2) `xy_goal_tolerance` must be > 0.1.
The following error message appears in one of the open terminals: `Map update loop missed its desired rate of <rate> Hz... the loop actually took <seconds> seconds.`	1) There is not enough processing power. 2) There is a parameter tuning mistake.	1) Close unnecessary background tasks. 2) The value of the parameter `sim_time` is too high for the processing power of the Raspberry Pi and must be decreased to 2.0.

The following error messages appear in one of the open terminals:
 	
`No route to host.`
`Unable to communicate with master.`
	There are network connectivity issues.	Ensure User Pi and Turtlebot Pi are connected to the same network, which can be done by pinging each other. Furthermore, confirm the IP addresses on the `env.py` files are correct.

One of the following error messages appear in one of the open terminals: 1) `Costmap2DROS transform timeout. Current time: <time>, global_pose stamp: <time>, tolerance: <tolerance>.` 2) `Failed to compute odom pose, skipping scan (Lookup would require extrapolation into the future. Requested time <time> but the latest data is at time <time>, when looking up transform from frame [base_link] to frame [odom]).`	The User Pi and Turtlebot Pi are not time synced.	1) Ensure User Pi and Turtlebot Pi are connected to the same network and have the same time. Use the `datetimectl` command to check the time on the Pi's 2) Manually set the time by using the command: `sudo timedatectl set-time <hh:mm:ss>`. 3) If the network is offline, use an NTP server by following the NTP Server Setup tutorial [3].
No pop-ups appear on the screen after running `delay_flights.py` on personal device.	There are network connectivity issues.	Ensure your device and the Server Pi are connected to the same network, which can be checked by pinging each other.
User Interface suddenly stops after scanning the boarding pass.	The `update.txt` file contains a record change from a previous use.	Delete all the content in the file `sdp-ian/Ui/update.txt` on the Server Pi.
User Interface freezes.	The User Pi is overheating.	Turn the User Pi off and let it cool down for a few minutes.
The robot beeps every 20 seconds or so.	The battery is low on charge.	The battery must be replaced or recharged.
The screen does not display anything.	1) There may be a disconnected cable. 2) The screen may be faulty.	1) Ensure all wires are connected. 2) The screen unit must be replaced.

Appendix

A Overriding Default Configuration

Navigation

The navigation packages used by the Turtlebot (found on the Turtlebot's Pi) have certain parameters, which can be tuned to enhance the Turtlebot's navigation. These parameters are located in files in your `catkin_ws/param` directory. In `dwa_local_planner_params_waffle_pi.yaml`:

- `path_distance_bias`: sets how close the local planner should stay to the global path. A high value makes the local planner prefer trajectories on the global path.
- `xy_goal_tolerance`: the tolerance in meters from the goal. A value of 0.25 allows a tolerance of 25 cm to reach the given goal.
- `min_vel_x, max_vel_x`: the minimum/maximum velocity of the Turtlebot.

In `costmap_common_params_waffle_pi.yaml`:

- `inflation_radius, cost_scaling_factor`: control how far away the Turtlebot aims to be from obstacles while it moves. More on this in page 11 of the ROS Navigation Tuning Guide [4].

UI

There are two modes for the UI: a dark mode and a light mode. The default mode is the light one. If you would like to change it to the dark mode, edit line 14 in `sdp-ian/ui/IanUi.py` to:

```
qtdesigner_file  = "IanDark.ui"
```

Database

To make changes to the database via the Server Pi - adding, or deleting tables or entries - simply connect the Server Pi to a screen, a keyboard and a mouse. Then, access the `mysql` server console via:

```
mysql -u root -p
```

The password here should be `ubuntu`. Once you have the console open, you could use the basic MySQL tutorials [5] to make your changes.

B Using Your Own Pi's

Requirements

Assuming you are provided with the Turtlebot WafflePi, **you will additionally need two Raspberry Pi's**. If you choose to work with Raspberry Pi's other than the provided ones, the minimum requirements for each are as follows:

- 1GB RAM
- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- Full-size HDMI
- 40-pin extended GPIO
- 4 USB 2 ports

For those Raspberry Pi's to work as intended, their OS has to be **Ubuntu 16.04** and they need to have ROS (Robot Operating System) installed. You will also need **a webcam that can automatically focus** - a recommended minimum requirement is Logitech C270 HD Webcam [6], which is provided. A 7" display touch screen [7] is also provided; if you choose to work with a different screen, it has to be compatible with the Raspberry Pi.

Setting Up the Navigation Modules on Your Own Pi

We advise that you download the Ubuntu 16.04 image from Ubiquity Robotics [8] as it comes with ROS pre-installed. First, you should follow the Robotis PC Setup tutorial (section 6.1.4) [1] to configure the network once you know the IP's. Second, you should copy and paste the `launch` and `param` directories from the repository [2] into your home directory's `catkin_ws` folder.

Setting Up the Database on Your Own Pi

If you have opted for another Raspberry Pi, then you first need to follow this MySQL Server Installation tutorial [9], up to and including 'Start the mysql shell', to install a mysql server on the third Pi (the Server Pi). Make the password `turtlebot` for simplicity. You also need to create another user, `secondPi` with the same password and privilges using this MySQL User Creation tutorial [10]. Once that is done, all you need to do is run the `create_db.sql` file from the root folder of the sdp-ian repository [2] as follows:

```
mysql -u root < create_db.sql
```

You should then add a trigger event for whenever a change happens to a flight's time or status (in the table `main_sdp_db.flights`), an entry should be created in a dedicated table (`main_sdp_db.flight_changes`). The official MySQL Trigger Events tutorial [11] walks you through how create such triggers.

This should create the database with all its tables and its users, and the setup should be complete.

C Making Your Own Environment

If you opt for using a different physical environment than the one we have outlined in Figure 3, we will cover in this section how to integrate a different physical environment into the system.

We begin by assuming that you have followed the instructions in 2, except for, of course, 2.3. To continue with this customised setup, you will need the User Pi and the Turtlebot up and running as instructed. You will need a screen, a keyboard and a mouse connected to the User Pi.

Step 1: Making the map

On the **User Pi**:

1. Open a terminal and run:
   ```
   roscore
   ```
2. Open another terminal, and ssh into the Turtlebot (if you are asked for a password, it's `turtlebot`):
   ```
   ssh pi@<turtlebot-ip>
   ```
   and then run:
   ```
   roslaunch turtlebot3_bringup turtlebot3_robot.launch
   ```
3. Open yet a third terminal and run the following **two** commands:
   ```
   export TURTLEBOT3_MODEL=waffle_pi
   roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
   ```
4. On a 4th terminal window, run:
   ```
   export TURTLEBOT3_MODEL=waffle_pi
   roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
   ```

The last command allows you to control the robot to perform the mapping operation. It is important to avoid vigorous movements such as changing the speed too quickly or rotating too fast. When building a map using the robot, the robot should scan every corner of the environment to be measured. You will be able to see on the User Pi screen which parts of your physical environment have been mapped and which parts have not been mapped.

Once you are satisfied with your map, you can save it by running the following in another terminal:

```
rosrun map_server map_saver -f ~/map
```

This will create two files, `map.pgm` and `map.yaml` in the home directory of the User Pi. For the sake of consistency, copy those files in the root folder of the `sdp-ian` repository on your Pi to overwrite the ones already provided.

When you start up the system as in section 2.4, the navigation packages will use the map you have just created.

Step 2: Setting Destinations in Your Map

For this part, follow these steps to begin with:

1. Open a terminal and run:
   ```
   roscore
   ```
2. Open another terminal, and ssh into the Turtlebot (if you are asked for a password, it's `turtlebot`):
   ```
   ssh pi@<turtlebot-ip>
   ```
   and then run:
   ```
   roslaunch turtlebot3_bringup turtlebot3_robot.launch
   ```
3. Open yet a third terminal and run the following **two** commands:
   ```
   export TURTLEBOT3_MODEL=waffle_pi
   roslaunch turtlebot3_navigation turtlebot3_navigation.launch
   map_file:=$HOME/map.yaml
   ```
4. On the visualisation tool, RViz, that just opened, you should point to where your Turtlebot is and its orientation via the `2D Pose Estimate` button.
5. Open another terminal, and run:
   ```
   rostopic sub /move_base/feedback
   ```
   This will keep printing the coordinates where the Turtlebot thinks it is on the map until it reaches the destination and stops. **Record the final pair of coordinates printed in the terminal.**
6. Then, use the `2D Nav Goal` button to select a goal. As soon as you select it on the map, the Turtlebot will start moving in its direction.

The final pair of coordinates are the coordinates of the selected destination on your customised map.

When you are done repeating this process to select your possible destinations, open `sdp-ian/ui/get_pos.py` and change the positions of the gates and the shops (lines 36-68) to the ones you chose on your map.

Step 3: Putting the New Map on the UI

This might be the most difficult step, since it requires a bit of manual and mathematical work to get it right.

The goal of this part is to get 'live tracking feature' working correctly, where you can see the Turtlebot's current location on the map as it moves towards its goal (which you can also see on the map).

The first step is to create an approximation of your map in QtDesigner. Next, you need to figure out the x and y coordinates of the various edges of your map on the 7" screen.

Next, using those values, you need to create a conversion formula that converts the coordinates of the Turtlebot on its map to that on the UI map. This is to make the illusion of GPS tracking. Once you know this formula, you can put it in the `convert_coordinates(x,y)` function in `sdp-ian/ui/get_pos.py`.

D Making Your Own Boarding Passes

If you choose to make boarding passes of your own, the process is fairly simple. You can choose to have either a barcode or a QR code on your boarding pass. In either case, the code corresponds to a string. You could make either via the Online Barcode Generator web tool [12].

Once you have printed your code(s), be sure to make an entry in the `main_sdp_db.passengers` table corresponding to that user, specifying a `flight_id` for the passenger that exists in the `main_sdp_db.flights` table.

References

[1] "PC Setup - Turtlebot3". In: *docs.ros.org* (). URL: http://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/#pc-setup.

[2] "SDP Group 6 Repository". In: (). URL: https://github.com/yussefsoudan/sdp-ian.

[3] "NTP Setup". In: (). URL: http://raspberrypi.tomasgreno.cz/ntp-client-and-server.html.

[4] Kaiyu Zheng. *ROS Navigation Tuning Guide*. URL: http://kaiyuzheng.me/documents/navguide.pdf.

[5] "Basic MySQL Tutorial". In: (). URL: https://www.mysqltutorial.org/basic-mysql-tutorial.aspx.

[6] "Logitech Webcam C270". In: (). URL: https://www.logitech.com/en-gb/product/hd-webcam-c270.

[7] "Raspberry Pi 7" Screen Display". In: (). URL: https://www.amazon.co.uk/Raspberry-Pi-7-Inch-Screen-Display/dp/B014WKCFR4.

[8] "Ubiquity Robotics". In: (). URL: https://downloads.ubiquityrobotics.com/pi.html.

[9] "Install MySQL Server on the Ubuntu operating system". In: (). URL: https://support.rackspace.com/how-to/install-mysql-server-on-the-ubuntu-operating-system/.

[10] "How to Create MySQL Users Accounts and Grant Privileges". In: (). URL: https://linuxize.com/post/how-to-create-mysql-user-accounts-and-grant-privileges/.

[11] "Trigger Syntax and Examples". In: (). URL: https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html.

[12] "Online Barcode Generator". In: (). URL: https://barcode.tec-it.com/en/?data=Lorem%20Ipsum%0A.