

# O.L.I. User guide



# Contents:

Topics	Page no.
1. System overview	
1.1. Introduction -----	2
1.2. Server -----	2
1.3. Robot -----	2
1.4. App -----	3
2. Unpacking	
2.1. Robot -----	3
3. Installation and set up	
3.1. Server -----	3
3.2. Simulation -----	4
3.3. App -----	4
4. Operation of system	
4.1. Running the app -----	5
4.2. Running the O.L.I. simulation -----	6
5. Troubleshooting -----	6
6. Contact us -----	7

# 1. System Overview

## 1.1. Introduction:

The OLI platform is designed to help people at risk of falling in their homes to live more independent and safer lives. The prototype platform we have put together works by using a mobile app to detect and register a fall, before autonomously navigating towards the user. OLI then steadies itself using a deployable leg system, allowing the user to help themselves back to a standing position. OLI's goal is to ensure that the inconvenience of minor falls no longer means a loss of independence whilst ensuring the user is covered in the event of a serious fall. OLI achieves this through the use of a timer, started upon detection of a fall, that calls emergency services upon completion. If the fall is serious all the user needs to do is wait and OLI will ensure that help will arrive.

## 1.2. Robot:

### 1.2.1 Navigation

O.L.I. is equipped with a 360° field of view and 3.5 meters range LiDAR making it capable of building an accurate occupancy grid. The implemented SLAM algorithm allows O.L.I. to start its movement from a random position in an unknown environment and to build the occupancy grid of the area by keeping track of its own position relative to the built map. Once the occupancy grid is created, it can plan a path to go from one point (start) to the other (target) using a rapidly-exploring random tree algorithm hence avoiding all obstacles. Because it is a differential wheeled robot (i.e a robot whose movement is based on two separately driven wheels placed on either side of the robot body), we can easily make it rotate toward a goal by varying the relative rate of rotation of its wheels. The robot's controller has been written in MATLAB and the simulations are done in Webots.

### 1.2.2 CAD

The CAD model is made up of a Robotis Turtlebot Burger model, along with a custom designed exoskeleton. The external frame is made up of a base plate, support rods, top cap and the support bar surrounding the frame. It also features deployable feet mounted to the base plate in order to provide additional support to O.L.I. when aiding a user in getting up. The support bar surrounding the frame, its joints and the deployable feet are all rated for a load of 83.9kg. The feet and support bar are operated via linear actuators mounted at the bottom of the robot.

### 1.3. App:

The O.L.I. robot comes with a handy mobile application that allows the user to set up an account, call O.L.I. when required and monitors the user's movements to detect a fall and call for help. The app comes with a front-end written in Dart using the Flutter framework, and a backend written in Python using Flask.

The app comes with two buttons - Call O.L.I. and Set up. For testing purposes, there is a third button - fall detection demo. The first step is to set up the app to connect to OLI. If the robot and your phone are both connected to the same WiFi network, the app will show that OLI is connected. The user will also be given the option to enter their name, postcode, and emergency contact number. This will personalise the experience of the user and allow the app to call emergency services to the right location, as well as contact your emergency contact.

### 1.4. Server:

The OLI prototype we have developed detects falls via a mobile app before relaying this information onto the OLI platform itself. Wifi communication was determined as the most reliable way for this communication to happen since it greatly increases the range at which communication can occur between app and robot. Whilst the simulation and the app can be run independently of one another on a local machine, it is important to demonstrate that we have the capability to connect the two as if we were to deploy our prototype in the real world.

## 2. Unpacking:

### 2.1. Robot

Due to the constraints of the current pandemic, no physical prototype of the device is available. However, we have listed what we would include with such a device below.

**Included in the box:**

*1x O.L.I.- robot platform*

*1x charging cable*

*1x user guide leaflet*

**Downloaded from the web:**

*1x O.L.I. app*

## 3. Installation and set up:

### 3.1. Server

The Web Server is used to link the Webots simulation and the OLI app. In a prototype deployed in the real world, this server would run on the OLI platform and facilitate communication between itself and the mobile app via a wifi network.

To install the server, please enter the following command into a terminal.

```
git clone https://github.com/SDP-Group-1/server.git
chmod +x OLI-server-install
sudo ./OLI-server-install
```

This will install and start the web server used for communication. The web-server uses ftp and will listen on port 22. In addition to this, a new user named 'OLI' is added to the system, with a home directory that will contain all the files used for communication. An uninstall script is also created, which will completely revert all changes made by the install script.

### 3.2. Simulation

To run the robot simulation you will need to access both the MATLAB controller and the webots worlds (simulations). The following steps will help you set up Webots, MATLAB and the repository:

- Clone the git repository for the robot on your local machine using the following command:

```
git clone https://github.com/SDP-Group-1/nav.git
```

- Install Webots R2021a using the installer from cyberbotics [here](#).
- Install MATLAB R2021a using the installer from mathworks [here](#).
- Install the necessary MATLAB toolboxes. You can do this during the installation but you can also do this afterwards by clicking on Add-Ons > Get Add-Ons. You will need the navigation toolbox (version 2.0), the Lidar toolbox (version 1.1), the computer vision toolbox (version 10.0) and the image processing toolbox (version 11.3).
- The robot's controller is a MATLAB controller that can be edited from MATLAB or directly on Webots.
- You can now start editing the controller and the simulation as you wish!


### 3.3. App

The app accompanying the O.L.I. robot uses the [Flutter](#) framework. The following steps will help you set up the app and the editor:

- Install Git using the instructions [here](#).

→ Clone the git repository for the O.L.I. app on your local machine using the following command:

```
git clone https://github.com/SDP-Group-1/oli.git
```

- Install Flutter [here](#). This app runs on version 1.22.6 (DO NOT USE FLUTTER 2). Once you complete installation, open your local command line console and run `flutter --version`. If the version is higher than 1.22.6, then run `flutter downgrade`, followed by `flutter doctor1`.
- The backend of the app is a server that is programmed using Python and the Flask framework. Install Python [here](#). Use [this article](#) to install Flask<sup>2</sup>.
- The app can be maintained and edited through Android Studio or Visual Studio Code.
  - ◆ **Android Studio:** Install Android Studio [here](#). After installation, click on “Open Existing Android Project”, and open the oli project folder. To add in relevant plugins for Flutter and Dart, on an existing project, click on File > Settings > Plugins. Over there you will be able to search for the Dart and Flutter plugins for Android Studio.
  - ◆ **Visual Studio Code:** Install Visual Studio Code [here](#). Once installed, open the editor, click on File > Open Folder and navigate to the oli project folder. To better facilitate programming in Dart, go to Extensions (Symbol-) and install the ‘Dart’ and ‘Flutter’ extensions.
- Set up a device: you can either use an emulator or your own smartphone to run the app. It is recommended to use your own smartphone as the fall detection requires the sensors that are present in a smartphone.
  - ◆ **Emulator:** To set up and use an emulator on Android Studio, follow [this article](#). When an emulator is used on Android Studio, Visual Studio Code recognises it as well. However, it is not supported well and thus, is **not recommended**.
  - ◆ **Personal smartphone:** On your preferred Android smartphone, enable Developer Options using [this article](#). Once enabled, turn on USB Debugging. This will allow your device to show up on your preferred editor. Additionally, install ADB [here](#), which will allow you to access files on your Android device and efficiently debug the app.

That’s it! You can now start editing the app in your preferred editor.

---

<sup>1</sup> This will ensure that the other SDKs involved such as Dart will be updated corresponding to the version of Flutter

<sup>2</sup> You do not need to set up an environment to use Flask - that is optional, but recommended. You can also [install Flask](#) through [pip](#), the Python package manager directly on your machine.

## 4. Operation of system

### 4.1. Running the app

The app consists of two parts - a frontend in Dart (Flutter) and a backend in Python (Flask). The Flask server needs to be run prior to running the app. Follow these steps to get the app running on your machine<sup>3</sup>:

- Open your preferred text editor that you set up in Section 3.3., and open the project in the editor.
- Open `/fall-detection/flask_backend.py` in your editor, and edit line 7 to your local IP address (format: `http://<your IP>:5000`). Use [this article](#) to find your IP address.
- Open your local command line console and navigate to the oli app directory. When inside the directory, open `/fall-detection`.
- Run the `flask_backend.py` file to start the backend server.  
`python flask_backend.py` (may need to use `python3`).
- To run the app on Android Studio, click on the Play icon in the toolbar, or click on Run > Run. To run the app with breakpoints, follow [this article](#).
- To run the app on Visual Studio Code, open `/lib` folder. This folder contains all the Dart files that build and render the app. Open `main.dart`, and above the main function, there should be two options: Run | Debug. Click on Run and the Command Palette will open all the devices connected. Click on your Android Device.
- The app will take around 1-2 minutes to build and will then open on your device. Make sure your Android smartphone is unlocked.

The app should now be running on your machine. You can use the debug console present in your editor to see any `print()` messages to confirm that the app is running correctly.

### 4.2. Running the O.L.I. simulation

The GitHub repository currently contains 5 branches including main. The main branch contains the latest working version. The other 4 branches contain the features in development.

- If you wish to work on any of these features you can

```
git fetch origin <branch-name>
git checkout <branch-name>
```

- Once you are on the desired branch you will be shown two folders: `demos` and `webots`. Select `webots`.
- The `webots` folder itself contains multiple other folders each showcasing a feature such as `lidar-readings`, `route-planning` or `occupancy-grid creation`.
- Based on what you wish to see/ work on, select a folder.

---

<sup>3</sup> These steps are for running the app on your own personal device.

- The directory that you have chosen should now show 2 subdirectories: The Worlds folder and the Controllers folder. The worlds folder contains the webots simulations whereas the controllers folder contains the MATLAB controllers.
- Simply choose the simulation you wish to run and open it on Webots. The associated controller will be automatically displayed next to it.
- If you wish, you can choose to edit the controller in MATLAB.
- Note that, once you run the simulation a MATLAB Command window will appear.

## 5. Troubleshooting

### **Q - Null safety error while running 'pub-get' on Flutter.**

A - Flutter has recently introduced a new version 2.0 which introduces a number of new programming safeties. However since we are building our app on an older version of Flutter, it can not support all dependencies that have been updated to support Flutter 2.0. It is then up to the developer to go through which dependencies have been updated for Flutter 2.0, set their versions to the appropriate ones, and run `flutter downgrade`, followed by `flutter doctor`.

### **Q - Flask server does not receive the request from the app with the sensor data / SocketException in fall detection.**

A - Ensure that the host IP is set to your laptop's IP address in `/lib/flaskApi.dart` (line 7). To find your laptop's IP address, follow this [article](#).

### **Q - Invalid depfile message in the debug console for the app.**

A - Run `flutter clean` in your project directory from the command line to resolve the issue.

### **Q - The server is giving me a 500 OOPS error when I try to connect?**

A - This issue is likely the result of a server misconfiguration or a faulty connection. To fix, uninstall and reinstall the server using the steps on pages 3 and 4. If this fails, try using the 'kill' command on other programs that might be trying to use the same port as the OLI web server

### **Q - Webots is not linking to MATLAB**

A - 64-bit versions of Webots are not compatible with 32-bit versions of MATLAB. Webots comes only in 64-bit flavors and therefore it can only inter-operate with a 64 bit version of MATLAB.



**Q - Webots is showing this error message on the console: “error using ⇒ calllib Method was not found” or “error in ⇒ launcher at 66 calllib(‘libController’, ‘wb\_robot\_init’)**

A - On some platforms the MATLAB interface needs perl and gcc to be installed separately. On some macOS systems the MATLAB interface will work only if you install the Xcode development environment

**Q - Webots is using the wrong controller**

A - When Webots tries to start a controller it must first determine what programming language is used by this controller. So, Webots looks in the project's controllers directory for a subdirectory that matches the name of the controller. The first file that is found will be executed by Webots using the required language interpreter. It won't be possible to execute "xyz\_controller.m" if a file named "xyz\_controller.py" is also present in the same controller directory.

**Q - Webots is using a void controller instead of my controller**

A - In the case the filename specified in the .wbt file doesn't exist or if the required language interpreter is not found, an error message will be issued and Webots will start the "void" controller instead. Make sure that you have done all the steps mentioned in 3.2 of the user guide and that the controller file is in the right subdirectory.

## 6. Contact us

For any queries, please contact us on [group1@ed.ac.uk](mailto:group1@ed.ac.uk).

To give us feedback on how we can improve our product, please send an email to [feedback-group1@ed.ac.uk](mailto:feedback-group1@ed.ac.uk).

If you believe you can contribute to our project, [open a pull request](#) in the respective repository on GitHub and request a review from a member of that team. Read [this article](#) to learn how to contribute to open-source projects.