**Kick Off Report**

**Project Plan**

The aim of this project is to develop an intuitive user interface for controlling a fleet of e-puck2 robots through the use of ROS. The project involves the development of the UI mock-ups – which is followed by the implementation of the UI – in parallel to the setup of the ROS, with integration of the UI and ROS configuration as a key milestone. The final outcome will enable intuitive control and programming of the e-puck2 robots, targeting accessibility and usability for users of all skill levels, with preset examples and interactive tutorials may enhance the user experience as stretch goals. Parallel development of the UI and the ROS setup leads to a more efficient development process. The tools and frameworks that will be utilised include Figma to create the UI mock-ups, React for the UI implementation, Blockly for coding and ROS for setting up the robots and connecting them. Risk mitigation strategies will be implemented throughout the development process to ensure a smooth workflow.

Milestones:

* UI mock-ups on Figma
* Implementation of UI
* ROS Master and configure ROS on robots
* Connect frontend functionality to backend ROS master
* Project showcase (presenting to client)

These milestones can be further broken down into smaller objectives:

**UI mock-ups on Figma**

* Research user requirements and design principles such as accessibility and ease of use.
* Design high-quality mock-ups.
* Collect feedback on the mock-ups from stakeholders for iterative refinement.

**Implementation of UI**

* Set up the development environment with React and integrate necessary libraries.
* Implement basic UI components, such as menus, buttons, and robot controls.
* Add Blockly-based coding functionality for drag-and-drop programming.
* Add written code functionality for more advanced users.
* Develop basic preset code examples to assist users.
* Implement accessibility features such as alt text, dark mode and colour-blind-friendly palettes.
* Conduct frontend testing to ensure usability and fix bugs.

**ROS Master and Robot Configuration**

* Install and configure the ROS environment on the control system.
* Set up the ROS Master and ensure it can handle multiple robot connections.
* Develop ROS nodes for communication with the e-puck2 robots.
* Test basic robot functionality such as movement and sensor data retrieval through ROS commands.
* Troubleshoot hardware and software issues as they arise.

**Connect Frontend Functionality to Backend ROS Master**

* Define communication protocols for the frontend and ROS backend.
* Map frontend commands to ROS functions.
* Implement the API layer to bridge the frontend with the ROS Master.
* Test the integration extensively with individual and multiple robots.
* Debug and optimise performance to ensure smooth communication and control.

**Project Showcase**

* Prepare a demonstration highlighting key features.
* Collect feedback from the client and stakeholders.
* Document lessons learned and identify areas for future development.

Timeline:

* Task 1 – Create UI mock-ups via Figma. The development of task 1 will occur over the Christmas break, with the task ideally being completed before the second semester begins, with January 7th marked as the deadline.
* Task 2 – Configure ROS on the robots and setup ROS master – Jan 7th. This task will be developed in parallel to task 1 as they are independent of each other and entail different aspects of the system.
* Task 3 – UI implementation – Good quality UI designs from task 1 – the mock-ups of the UI – will ensure that the implementation of the UI is projected to be completed by January 25th. Blockly and React will be used to implement the UI.
* Task 4 – Integration of UI functionality to ROS master. Task 4 will begin after the UI has been fully implemented and the ROS system, including the configuration of the ROS master, has been successfully set up on the robots. Considering that this task would occupy the most time, the deadline for this task would be February 26th.

Risks:

* Underestimation of task complexity. Certain tasks could take longer than anticipated due to their complexity. To mitigate, break complex tasks into smaller subtasks and regularly review progress to adjust scheduling accordingly, such as adding slippage time to tasks as necessary.
* Hardware issues with robots. Robots may malfunction. To mitigate, perform rigorous testing early, maintain log of issues.
* Integration challenges between frontend and ROS backend. Could introduce compatibility issues. Conduct integration tests early in the project timeline and maintain clear documentation for the API and web socket connections.
* Prevent damage between robots. To mitigate, add emergency stop button to stop all robots as well as button within software to stop individual robots.
* Scope creep. Additional features that may overwhelm the development timeline. To mitigate, stick to must-haves and should-haves for the MVP. Defer could-have features with a lower priority and as stretch goals.
* Lack of clear documentation. To mitigate issues such as future users potentially struggling to understand the system, create documentation and continuously update throughout the project.