# KPI Review and Conclusions:

Note that the initial KPI from the project overview is bolded, and then conclusions drawn from these KPI are attached in non-bold.

* **Detection of proximity to ultrasonic sensors. Success for this criteria is indicated by TurtleBot motion corresponding to variation in ultrasonic ranging readings.** This criteria was successfully met as the TurtleBot’s motion is clearly responsive to changes in ultrasonic readings.
* **Speed control via ultrasonic sensor input. Success is defined by proportional changes in TurtleBot speed with respect to ultrasonic proximity.** This KPI was successfully achieved by having one ultrasonic sensor’s distance reading proportionally correspond to linear velocity of the TurtleBot from 0m/s to its maximum speed of 0.26m/s.
* **Directional control via ultrasonic sensor input. Success is defined by proportional changes in TurtleBot direction with respect to ultrasonic proximity.** This KPI was successfully achieved by having one ultrasonic sensor’s distance reading correspond proportionally to angular velocity of the TurtleBot in its full range of -1.82 to 1.82 rad/s.
* **The ability to enter gesture control mode. Specifically, the press of a button to switch modes and the representation of the system being in "gesture mode" being reflected on the M5 Core UI.** This criteria was successfully met, though representing the system being in “gesture mode” in the UI was not implemented – as this was seen as unnecessary cluttering of the UI and adds unnecessary communication between the devices. Instead, due to well-implemented button debouncing, any press of the user button on the Nucleo board will toggle the mode – and the device will always start in normal mode, so the mode of operation can always easily be tracked. Additionally, the GUI displays when a gesture is received to effectively communicate the robots behaviour to the user.
* **Accurate detection and execution of gestures (system recognises defined hand motions) and representation of this in the UI. Success for this criteria does not involve turtlebot motion, but rather the accurately UI display of the detected gesture.** This criteria has been successfully met, with the detected gesture being displayed in the bottom left corner of the M5 Core.
* **Localisation of the TurtleBot. Success for this criteria is indicated via accurate position estimate display in a 2x2 grid interface on the M5Core2.** The TurtleBot’s position is correctly displayed on the M5 Core grid, which constitutes success for this criteria.
* **Staying within the grid. Success for this criteria is described by the TurtleBot automatically detecting that it has moved outside of the grid, and then turning around so that it can stay within the grid. Expected behaviour is that gesture or ultrasonic input until the TurtleBot has completed its process of re-entering the grid.** This criteria has been successfully implemented, as the TurtleBot detects that it has moved outside of the 2x2m grid and immediately turns around, bypassing all current queued actions for it to perform.

# Challenges/Issues Faced

1. **Overall Software Stability and Robustness:** Maintaining software stability and robustness, especially in complex systems like ROS navigation, posed ongoing challenges. Unexpected crashes or software failures occurred often which disrupted the functionality of the entire system.

**Solution:** We employed rigorous software testing methodologies, to identify and address software bugs and stability issues. Continuous monitoring and logging of system performance helped in diagnosing and resolving issues promptly.

One such occurrence of software crashes was caused by a python queue being full, and attempting to add more information into this queue. This caused the software to block while the queue was filled. The solution was to remove the max queue size and periodically clear the queue to prevent backup of irrelevant data.

1. **Gesture Calibration:** Calibrating the algorithm to detect gesture calibration was a challenge that took several hours to implement.

**Solution:** We employed rigorous testing adjustment of the algorithm to ensure the parameters chosen achieved a desirable result which can detect the 5 different gestures we have added to the project.

1. **ROS:** ROS is a complex framework with a steep learning curve, requiring proficiency in both software development and robotics concepts. As a team we had no experience with ROS and this posed a significant challenge.

**Solution:** To address this, each team member conducted their own individual research to familiarize themselves with ROS. Online tutorials, ROS documentation and videos were utilized to accelerate the learning process. In addition to this, collaborative problem solving was encouraged between team members, this allowed knowledge sharing and facilitated faster learning and increased problem solving abilities.

1. **Insufficient Hardware:** One of the ultrasonic sensors in our setup was found to be faulty, providing inaccurate readings or failing to function altogether. This presented a challenge as it impacted the reliability and accuracy of proximity detection.

**Solution:** To address this issue, we promptly identified the faulty ultrasonic sensor and replaced it with a new, properly functioning sensor.

1. **Localisation with only Odom topic:** Initially, relying solely on odometry data for localization posed significant challenges due to its inherent limitations, such as cumulative errors and drift over time. This resulted in inaccurate position estimates and compromised the overall performance of the localization system.

**Solution:** We improved localization accuracy by integrating the SLAM-Toolbox ROS library, enhancing our TurtleBot's navigation with robust SLAM algorithms. This allowed us to leverage both LiDAR sensor data for real-time mapping and wheel encoder data to provide precise localisation using a Kalman filter.