Read Me

GitHub repo: <https://github.com/Rasesh94/Programming-Things-Assignment-1.git>

Navigation Video: https://drive.google.com/open?id=1AKkXRMMEi4v1HZDu0FSOqQihNQqLzOiw

**Assignment Summary**

This assignment was a brand-new learning curve for me within the programming domain as it was my first experience with development on embedded systems, and it opened me up to a whole host of challenges. One of the biggest challenges I came across with development in the Arduino environment in particular was the lack of breakpoints which made me realise how reliant I am on them! I found myself making a lot of silly mistakes, which would usually be swept up via a debugging system.

Within the assignment I faced a lot of time issues due to my negligence with my time management. It was a strong learning curve; Regardless I am confident that I have completed the assignment to a reasonable degree.

**Objectives / Achieved**

**Task 1:** *The Zumo can be driven down the corridor from a* ***GUI\* (see comment below)*** *e.g. using w, a, s, d and ‘stop’ ‘buttons’ or a text field.* ***You*** *are controlling the Zumo at this point. Communication is via the xBee’s (not over a USB cable).*

The criteria for task 1 has been fully satisfied within the project. This was a simple adaptation of the ‘WASD’ program we developed within the lab sessions. During initial development for the GUI I wasted a lot of time. In order to create the GUI on processing, I followed a tutorial on google which involved me creating the GUI from scratch complete with buttons. I did this via creating classes for buttons and other objects which was very slow. I then re-read the specification and found it mentions the G4P GUI Builder which does everything for you instantly.

**Learning Points:** Fully read the specification before diving in!

**Task 2:** *The Zumo automatically keeps within the corridor by using the reflectance sensors to turn away from the walls (this is an adaptation of the boundary checking and line-following examples looked at in the tutorials). This means you only start the Zumo moving, after that it is navigating itself along the corridor. It stops when it encounters a ‘wall’ in front of it and you navigate it around the corner in task 3.*

The criteria for task 2 has been fully satisfied. This task proved harder than I anticipated due, and I found myself working on this right up until the last minute. This, again, was a consequence of me not fully reading the assignment specification before diving in. Initially I didn’t realise the Zumo must be kept between the lines (I just had the zumo stop at corners which was a very simple adaptation of the line follower example). Eventually I added the functionality to allow the zumo to rebound from the walls whilst simultaneously stopping when it reaches a wall. This was achieved by adding a plethora of clauses and stops when ANY of the sensors detect a value above the threshold.

**Difficulties:** The biggest gripe with task 2 derived from the inability for the sensors to detect the reading of the stopped device from when it was flagged. The initial logic I used when first attempting to implement this was as follows:

If sensor[0] (left most sensor) && [sensor[6] (right most sensor) > QTR\_Threshold Then It would indicate we’re at a wall. Which is true, however in practicality it didn’t work. This is due to the fact that the initial reading that flagged one of the right or left sensors as greater than the threshold would change by the time the function would begin. The way I eventually circumvented this issue is by adding multiple delays, and re-readings of the sensors before the function ends in order to get the latest reading to make a decision. I feel like the way I implemented this function was not very efficient and looked a bit messy. I would be interested to see the ‘preferred’ method.

**Task 3:** *The Zumo recognises that it has reached a corner, stops and sends a message using the xBee indicating that fact. The messages received from the Zumo should appear in a text area in the GUI. It then deactivates the behaviour from task 2 (which is keeping the Zumo between the corridor walls); this allows the (human) controller to turn the robot. The controller signals that the turn is complete by sending another keypress (e.g. 'C' or 'c' for complete). This then reactivates the task 2 behaviour so that the Zumo can drive itself down the corridor again.*

Task 3 is fully operational, and I didn’t come across any additional challenges with this task (other than the issues I mentioned with task 2). Whenever the zumo approaches any wall, it asks the user to manoeuvre the Zumo towards the corridor. Once the user clicks ‘complete’ on the GUI, the zumo begins automatically navigating down the corridor, whilst simultaneously creating a class instance for the corridor.

***Task 4:*** *The Zumo searches a room/side-corridor. The (human) controller will first stop the robot (outside the room or entrance to next corridor) and then signal that the robot is about to enter a room/corridor by sending an appropriate button press or text field data (e.g. ‘Co’ for corridor, “Ro” for room and 'R' or ‘L’ for right/left). They will then turn/drive the robot into the room/corridor. The Zumo should recognise this behaviour and an appropriate message should appear in the GUI.*

I successfully implemented task 4, and I also added an additional feature. Whenever the user signals the robot is outside a sub corridor/room, the user does **not** need to specify the room direction. The zumo automatically calculates whether the room is to the left or right based on the compass. The zumo will initially turn left 90 degrees and check for a wall. If there is no wall, the user proceeds ahead, if there is a wall however, the zumo rotates 180 degrees and proceeds to move forwards. The zumo also saves the direction (Left or Right) to the class instances.

***4A)*** *If the robot enters a room, the message should also give a room a number and identify whether the room is on the left or right of the relevant corridor. The Zumo should also record that information. The Zumo should only move into a room a short way and then stop to perform a scan of the room, using the U/s sensor, for objects. If an object is detected, the Zumo reports that back using the xBee link. This report needs to be seen inside the GUI you have created. The Zumo/message should identify which room the object is in (including the corridorID). After the scan is complete, the Zumo should stop and wait for the human controller to navigate the robot out of the room and turn back into the corridor. The same keypress as that used to signal that a corner-turn is complete should signal that the robot is back in the corridor and being driven as in task 2. – classes, and add to live stream data*

Task 4A is fully operational. Again, with the use of the compass the user does not need to specify the direction of the door. The zumo successfully enters a room/sub-corridor, uses the sensor to scan for objects. It performs a 360-degree spin whilst searching for objects to account for a wide distribution. If an object is detected within the room, it is flagged within the instance created for the room and a message appears on the GUI.

***4B)*** *If the robot enters a corridor, the robot should recognise that it is in a different corridor, and store relevant room data recognising that the room is in that side-corridor. Inside the corridor, it behaves as in tasks 3 & 4A.*

***4C)*** *At the end of the sub-corridor, the robot should stop (as in Task 2) the user (controller) should tell the robot to turn around and then the robot navigates its way directly to the end of the corridor. The robot should recognise the exit of the sub-corridor. It should behave as in Task3, but it should ONLY allow the controller to turn it the direction that will allow the Zumo to continue searching.*

*Note: when a robot is exiting a side-corridor, there is a wall in front of it, but no walls immediately to the sides, so it has two choices for which way to turn. However, at a corner there is a wall in front of it and a wall to one side: it only has one choice for the direction to turn. At the end of the corridor, there are walls on 3 sides of the robot.*

Tasks 4B & C were both implemented successfully (and also one of my favorite aspects about my program). The compass configuration played a great role in the completion of these objectives. The zoom successfully differentiates between a standard corridor and a sub corridor, and modifies its behavior respectively. When the zoom exits a room within the sub corridor, upon reaching the end it performs a 180-degree spin and drives to the exit. Once it has exited the sub corridor, based on the previous turn (the left/right prior to entering the sub corridor), the program prohibits the Zumo turning in that direction. It will only let you carry on going forwards but not back.

Unfortunately, due to time constraints I did not manage to complete task 5. This was disappointing for me, since I did have a clear strategy in mind on how I would have implanted it using timings and the compass. I discuss the reasons for my time constraints below.

**Learning Curves**

1. The Compass. Whilst it proved to be extremely useful by the end of the assignment, configuration of the compass was very time extensive. I had a problem with my XBEE shield which made the compass glitch with my Zumo, however prior to diagnosing it down to the XBEE shield, I wasted a lot of time tinkering the code thinking it was something to do with the way I wrote it. After the XBEE shield was replaced, I still wasted a lot of time with the compass because I already spent so long on it that I felt I needed to implement it otherwise it would have all been a waste of time! This was one of the biggest reasons I didn’t have enough time for task 5.
2. IDE. Another mistake I made was developing on Visual Micro initially. Visual Micro was the answer to most of my problems I had with the lack of intellisense on the Arduino IDE, however I soon shot myself in the foot when I tried to implement a vector using the IDE. There was an issue with adding libraries with visual micro which meant I had to move back to Arduino IDE which consumed a lot of time towards the end of the project.

**Libraries Used:**

Arduino\_Vector – Lightweight library used to get C++ Vectors (I found the other libraries such as the ArduinoSTL

LSM303 – For Compass

NewPing – For Sonar detector

Pushbutton – For the button to start

QTRSensors – Sonar Detectors

ZumoBuzzer – Zumo Libraries

ZumoMotors

ZumoReflectanceSensorArray