**Individual Zumo Assignment Report**

*Task 1*

Implemented basic WASD code from a previous lab session to test out my xBee link and check everything was in order. Also added in Serial inputs read for future code to be implemented.

*Task 2*

For this task, I made use of the of the example code ‘BorderDetect’. Due to this example being based on a black surface with a white border, I had to figure out how to invert this. I found a great post on the Pololu forum (<https://forum.pololu.com/t/sumo-collision-detection-on-white-surface-with-black-border/12462>), which helped me overcome this issue. This code worked well enough to keep the Zumo within the corridor by itself. I had to make some minor adjustments to the QTR threshold, speeds, timings etc. to make it suitable for my track however. I implemented this example code in to a while loop so that it would stop running once the Zumo detects a line in front of it (reaches a corner).

*Task 3*

For this task, I had to figure out how the sensors would know it had a black line directly in front of it. I decided that it would be best to carry on using the outer edge sensors (0 and 5), I outputted their readings to the Serial monitor to check what was coming back and realised they would detect a line in front of them once they both exceeded the set QTR threshold. With some minor adjustments to the threshold, this now worked really well.

I now had to set up my GUI in processing so I can control the Zumo from the GUI and display messages back to the user. As recommended on the spec, I installed the G4P library and used the builder tool to neatly add all my required buttons to a GUI along with a text area box to display messages to the user. Once I had done this, I needed to find a way to pass data back and forth between Arduino and Processing. I found a really useful guide online that explained exactly how this all worked (<https://learn.sparkfun.com/tutorials/connecting-arduino-to-processing#shaking-hands-part-1>). I was specifically interested in the ‘Shaking Hands’ section of this guide as it showed step by step a great way to allow for data flow both ways. I followed this as I was unable to find a better way to do it and it worked really well.

Once all of the above was set up, I re-implemented the WASD code in to Arduino so the user was able to manually navigate the Zumo around the corner, before being able to hit the Complete button so the Zumo could resume its behaviour of navigating down the corridor with border detect.

*Task 4*

First up, I implemented the code for ‘S’ so the user would be able to manually stop the Zumo and deactivate the self-navigating code. I had to include a check for ‘S’ being pressed within my border detect while loop so the loop could be broken at any time by the user. Stopping the Zumo would then enable the user to tell the Zumo whether it is about to enter a side-corridor or a room.

I then hooked up ‘Room’ and ‘Corridor’ buttons, along with ‘Left’ and ‘Right’ buttons so the user could state whether we were about to enter a side-corridor or a room and what side of the Zumo it is on.

Once this has been selected, the Zumo will store all of this data using an array of objects. Rooms/Corridors will have their own unique ID, along with other attributes to help the Zumo understand where everything is on the course. The GUI will display the room/corridor data that has been selected by the user (it will call this from the array of objects), then the user will be able to continue.

When Zumo enters a sub-corridor, a new corridor object will be created with a sub corridor flag. This enables the Zumo to know it is not in a normal corridor anymore and will know that when it reaches the end, it will be expected to be turned around and keep going until it exits the corridor. This flag will also come in useful in task 5. I have implemented code so the Zumo will know when it exits a sub-corridor and pass back an appropriate ‘exited sub corridor’ message back to the GUI. Due to the class objects I have created and the attributes they contain, my code will be able to calculate what way the Zumo needs to turn to continue its search and not to turn the way it has already searched. I have implemented code that disables either the ‘A’ or ‘D’ button so the user doesn’t have the option to turn the wrong way.

When Zumo has entered a room, the user will need to hit the ‘Scan Room’ button I have implemented. This executes a ‘for’ loop that will turn the Zumo incrementally each way whilst scanning for a person (object). I ended up using the NewPing library for this as it is a very easy library to use and works well. However, I came across this GitHub repo (<https://gist.github.com/flakas/3294829>). The code within this repo was very useful as it is very well commented, this helped me to understand how the sensor actually worked which I struggled to understand from the NewPing examples. The sensor works by firing a pulse of sound out from the sensor, when the pulse hits an object of some form, it will rebound the sound right back to the sensor. From this sensor reading we will be able to figure out how long this has all taken in milliseconds. We can then convert this reading from milliseconds to cm. Even if no objects are close, the sensor will obviously still return a reading but it will be a very high reading. In my Zumo code, I will do the conversion and check for if an object is within 20cm. This is the room depth, plus an extra couple of cm to allow for it to work if an object is right on the edge of the room and the Zumo isn’t far in to the room. I made sure the motors did a sweeping action to cover the whole room.

*Task 5*

In preparation for task 5, I have switched away from using object arrays to making use of vectors. To make vectors work in Arduino I had to add in library from <https://github.com/maniacbug/StandardCplusplus>. This is a library created by the development community to allow use of the standard template library for C++ within Arduino. We will be able to iterate through the newly created vectors with ease in order to read the last element and remove as we go backwards towards the start of the track.

After implementing the vectors and having these work properly, I decided to change how my Zumo tackled corners. After my change, the user would now press A or D and the Zumo would turn that way until the user presses stop again to stop the turn. This would allow me to time the duration the Zumo is turning for and store these readings in a vector so we can make use of them when turning corners on the return journey.

Similarly to the above, I wanted to calculate the distances/durations of corridors, how far rooms are down corridors etc. I made use of the ‘millis()’ function for this and the turn durations. This function returns an unsigned long variable that says how long the program has been running in milliseconds. I figured out that if I stored this in a start time variable, then read the millis function again, deducted the start time, I would result in the difference in how long a certain action has been running. In my case, how long a Zumo has been travelling down a corridor etc.

At this stage I realised I would need to split the ‘main’ corridor down in to separate corridors as I had been treating the main corridors as one corridor which would provide me with issues when trying to navigate the Zumo back round the track by itself. I had to make this work by creating a new corridor instance when the Zumo is turned with A or D. I obviously had to create a flag to stop a new instance being created when the Zumo is being turned in to a room, for example.

I have decided to calculate distances by recording a corridors total duration and saving this to an attribute. Sub corridors total duration attribute will actually hold the duration of how far the entrance to it is down its parent corridor. A room object will also have a duration attribute so it knows how far down its parent corridor it is. My calculation will work by taking a corridors total duration, deducting a rooms duration and being left with the required return journey duration. If a corridor has a room and a sub corridor, I will have to apply logic to accommodate this and adjust the durations accordingly.

Once the end track button is clicked, the Zumo will 180 and read the last rooms duration from the end of the rooms vectors, as long as this room needs checking again if it had a person in. Once it has travelled this distance, it will turn the correct way towards the room making use of the rooms attributes, re-scan, turn on a follow me light if a person is still in there and continue.

My logic will be checking for rooms and sub corridors on the current corridor so it knows what to be expecting next (a room, a sub corridor or a corner).

The Zumo will take advantage of the turn durations and other attributes to get itself round corners, and it will know the distances of rooms and sub corridors from my calculations being stored on attributes in the vectors.

As it makes its way back through the track it will make use of the push\_back() vector function to remove rooms and corridors from the last position in the vector after it is finished with them. Once there is only one corridor remaining the Zumo will know it was the original starting corridor and drive down it until it reaches the start line and displays a message stating it is complete to the GUI.

*Final Comments*

Unfortunately, I ran out of time and was unable to get task 5 working as well as expected. It works well without border detect, so it will travel all the correct distances, do the correct room checks etc. but it struggles with the corners a little and obviously staying within the corridor lines.

If I had more time, I feel I could have got this working perfectly. I also didn’t manage to refactor my code as much as I would have hoped, apart from that I am happy with what I have learnt and achieved from this assignment.