# Programming Things Assignment 1 Report

## Moving down the corridor

After playing with the Zumo somewhat, I soon realized that it has real issues when attempting to drive straight. I believe these issues stem from a mix of uneven surfaces and power distribution through the tracks, meaning the Zumo struggles to move more than a few centimeters without veering left or right. I attempted to compensate these inaccuracies but it seems that they are random and can happen either left or right and with different degrees of severity.

Due to the veering issues, I decided to utilize the reflectance array to line up to the wall. My plan was to start facing down the corridor, turn 90 degrees clockwise and check to see if there is a wall there, if there was a wall then I would turn 180 degrees and check to see if there was a wall to the left of the corridor too. If there was a wall to the left and right, I would ask the Zumo to move down the corridor slightly and check to see if the wall is still there. If the wall is not there at any point, we must either be entering a room or turning a corner.

## Turning

Initially, I attempted to do turning by giving power to the motors for a set time to turn 90 degrees. This worked in the first instance but was inaccurate and caused the robot to be as much as 180 degrees out, this is mainly because the paper slips under the Zumos’ tracks, giving unpredictable results.

Knowing the Zumo does contain a gyroscope I decided that was the next best way to do accurate turns. Unfortunately, there are no basic examples for this provided by Pololu for the Zumo shield for Arduino, so I had to look at some code for a Zumo 32u4 library and adapt it to use with the shield. The code that I found was a library written by Pololu (<https://github.com/pololu/zumo-32u4-arduino-library/tree/master/examples/RotationResist)> and that code has a class called TurnSensor which uses some basic gyro code to turn the Zumo. After implementing this class on my Zumo, I found that turns were still inaccurate and could be 5 to 10 degrees out and would occasionally just under or overshoot wildly. I assume this was down to the hardware I was using not been directly designed for this class.

In an attempt to improve my gyro accuracy, I searched the internet again and found a project by Texas Instruments (<http://processors.wiki.ti.com/index.php/ZumoCC3200)> that some summer interns had written. Although the majority of the code was useless to me, there was a class they had written called TurnAngleCommand, which could combine with the TurnSensor class written by Pololu to make a half decent turning algorithm which can be found in my project as Motion::turn().

## Entering Rooms

Once I had the robot turning, entering rooms in its most basic form was a fairly simple task to complete. If I ever found when checking for walls to the left and right that one did not exit, there was a timeout, so the robot would know there is no wall. This timeout usefully puts the Zumo about half way into the room. The Zumo can then check to make sure it is in a room and not turning a corner by searching left and right for walls. If no walls are found to the left or right, we know that the Zumo is in a room.

The issues with entering rooms come when you hit the ‘frame’ of the door, so half of the Zumo is in the room and the other half of the Zumo thinks this is a wall. To try and combat this, I had to have a function that would return the amount of sensors over the wall after it had tried to align with the wall. If there were still sensors not aligned, the Zumo is against a frame so we need to reverse off, move down the corridor slightly and attempt to enter the room again.

## Searching Rooms

Once confirmed to be inside a room, the Zumo will always end up facing the wall to the left of it, due to this I ask the Zumo to do a 450 degree turn, to the left, so it will end up facing back towards the door. During this sweep, the ultrasonic sensor is looking for objects in a small area and sends back “an object has been found in {room number}” Every time an object is found, so if there are 3 objects, it will return them 3 times.

The big problem with this was there is no way to align while doing the 450 degree turn and even though doing it in 5 lots of 90 degree turns was more accurate, the Zumo could easily end up 20 degrees off. There was no way I could combat this as with my design, the Zumo effectively though if there were no walls, it was in a room, to go and use a wall to align it would directly go against that theory. Alignment has to be done on exit, which is not very useful as the Zumo could end up coming back the way it came in a diagonal line, which happened more often than not.

## Exiting Rooms

After searching the room, we just needed to exit it. As mentioned previously, the search will leave the Zumo pointing towards the door so to exit we just drive forward until we find the wall, which will be opposite the door as we cannot have two rooms opposite each other. Usually the zumo would be coming out of the room at an angle so would need to align against the opposite wall. Once aligned, the Zumo can turn and carry on its merry way down the corridor.

One of the issues was if we carried on checking to the left and right as before, the same room would be hit multiple times due to the wide doors. To combat this, I asked the Zumo to look only towards the way opposite the room for the first time it advances down the corridor. This means if the door was to the left, the Zumo would come out of the room, align itself, turn left, move slightly, look to the right and align itself again and, turn left again and then carry on as normal.

## Cornering

Cornering is one of the easier tasks in the requirements. Using the same functions to detect a wall when turning left or right, I was checking to see if there was a wall while going straight forward. If we hit a wall while going straight forward we have either got to the end or hit a corner.

If it is a corner then one wall to either the left or right will be present. The robot will attempt to turn right, if it finds that is where the wall is then it will turn 180 degrees and the turn is left. The Algorithm then restarts back at moving down the corridor.

The main restrictions with the cornering is that it must be a right angle, if there is anything but a perfect 90 degrees then there is a high chance of errors. If there is a curved corner, the robot will not be able to proceed due to the reflectance sensor not knowing the situation where the edge sensors are activated and the rest are not.

## Detecting the end

Once we have the cornering algorithm, we can use this exact same principle to detect the end of the corridor, except when there is both a left and a right wall present, we know we have reached the end as there is only back to go.

The only time this could become an issue is if there is a small room so the robot thinks that is an ending to the corridor.

## Reporting to Base (XBee)

Using the XBee to report back to the computer was very useful. It was made tricky due to the only tutorials online that were available used the SoftwareSerial library and did very basic communications, such as a ‘chat client’. There were very few resources available on how this needs to be set up.

In the end, I used the built-in serial class to send the messages and the command line tool ‘screen’ to see the output.

## Conclusion

This task was made difficult by a few factors, such as the inability to turn accurately, the inability to drive in straight lines and the difficult methods needed to detect the location on the map. In a real life situation where there is 3D corridors, we would be able to have an ultrasonic module on each of the edges and use this to build a picture of the environment.