Zumo Search and Rescue Report

Tasks Achieved

Out of the tasks that were set in the brief my robot achieved the following:

- Task 1 - bot moves down the corridor
- Task 2 - bot turns corners
- Task 3 - bot detects end of corridor, turns around and comes back to start
- Task 4 - bot enters and leaves an empty room
- Task 5 - Enters and leave all rooms , returns to base at corridor end
- Taks 6 - Communicates robot state via bluetooth xBee
- Task 7 - Searches for people in rooms, signal via XBee

Acknowledgement and Sources

My robot makes use of libraries and ideas from three sources, Pololu the robot manufacturer , Texas Instruments, who had interns who wrote libraries for working with the Zumo robot using TI hardware / Energia and Tim Eckel the developer of the NewPing library.

Pololu

The core libraries that the robots uses to operate are from Pololu:

Library	Usage
ZumoBuzzer	Used to operate the buzzer on the Zumo
PushButton	Used to operate the push button on the back of the Zumo
ZumoReflectanceSensorArray	Used to operate the sensor array at the front of the Zumo to look for lines
ZumoMotors	Used to control the Zumo's DC motors
L3G	Used to operate the Zumo's built in gyroscope
zumo-32u4-arduino-library	

The TurnSensor library that I used was one that I created by using code from a **Pololu** **example** which used the built in Gyro to resist rotation of the robot - I abstracted the code from the example in the files TurnSensor.cpp/.h into a class that could be used as a general purpose library for working the Z axis of the robot's gyro.

Texas Instruments:
These libraries were developed by interns, it is the result of, "*a project completed by the TI Santa Barbara interns in the summer of 2013 in which the CC3200 is used to control a motorized robotics platform that includes Inertial Measurement (IMU) sensors*".

The library was useful for utilities used to work with the Zumo IMU and learning techniques for controlling the Zumo using it.

Library	Usage
Utilities	Used for general useful utility functions such as wrapping an angle , or checking if a value is within a given range
PIDController	Used to abstract the implementation of PID control calculations, a proportion, integral and derivative constants are supplied and the library performs the necessary calculations when given an error.
ZumoCC3200-library	

The Texas Instruments repo had many other libraries in particular a library called **TurnAngleCommand**, from which I learnt techniques for using the gyro to rotate the Zumo to a specific angle.

TurnAngleCommand uses PID control to perform this - my implementation also uses PID control at the heart of it and uses the same proportional and integral constants but with a much smaller stopping threshold for more accurate turns. My implementation also uses a different library, which I implemented based on a Pololu example, called **TurnSensor** to access the gyroscope and uses its readings to track the robot's orientation.

In learning how to use PID and the gyro to make the robot perform accurate turns, I developed a method of using the gyro readings to keep the robot driving in a fairly straight line using PID to control the motor speeds keeping the robot heading constant. The examples in the Texas instruments library were very useful in learning how to combine PID and the gyroscope to control the robot.

The implementation of this driving straight and rotation to a specified angle will be explained in the section “Key Issues”.

Tim Eckel
Tim Eckel is the developer of the NewPing library which is used for operating ultrasonic sensors, which my robot uses to detect objects in rooms. This library was used to easily work with the ultrasonic sensor and sweep the rooms looking for objects, an explanation of how this is used is featured later.
NewPing-library

Quick Operation Overview

The way in which my robot completes the course, is by building up an “image” of its current surroundings step by step using the line sensor. The robot is driven forward for a short distance, stopping when a timer has expired or a line is hit, it then turns left and right driving up to the wall on each side to ensure that it remains within the walls. This is repeated as the robot makes its way through the map.

The robot uses a data structure to store what kind of wall it just sensed to the left, right and in front of it (eg a partial wall, full wall, no wall) - this data structure is then examined to estimate where the robot currently is in the map, the previous position estimate is also used to calculate the new position estimate in some situations. Examples of position estimates are : in a right room, in a left room , in corridor , in right corner, end of maze.

The estimate is then used to decide what to do next, this table specifies how the robot acts with a given position estimate.

Position Estimate	Behaviour
In Corridor	Move forward and search, then turn and search left and right for walls
In Right Room	If the room hasn’t been searched move into the room and search it for people
In Left Room	If the room hasn’t been searched move into the room and search it for people
In Right Corner	Turn right then continue as normal move forward and search left / right
In Left Corner	Turn left then continue as normal move forward and search left / right

End of Maze	Turn the robot around 180 degrees and do the maze in reverse
Partial wall on the Left	Move twice as far forward as normal to get past the partial wall quicker, then search left / right as normal
Partial wall on the Right	Move twice as far forward as normal to get past the partial wall quicker, then search left / right as normal
Uncertain of position	Continue as normal, move forward and search left right, hopefully we will get enough sensory data to have a firm position estimate on the next movement
At the Start	Move forward and search left and right as normal.

With these estimates the robot is fully equipped with instructions of how to handle each situation it may encounter in the maze - fulfilling its job of going through the entire maze and finding people in rooms.

Key Issues

Staying within the walls: This issue is key to being able to complete the course at all, originally I thought it would be best to make the robot zig zag hitting each wall through the course until it detected rooms on either sides of the wall, but the problem with this is that it is difficult to differentiate between corners and rooms, I could not guarantee that the robot would not zig zag into a corner, in a way that would appear to be a room to the robot's sensors - or the opposite situation.

The solution that I thought up of to keep the robot within the walls was to have it move forward then rotate left and right driving up to the walls on each side. This ensures that the robot is still within the walls after each movement and has the added benefit of allowing us to differentiate between different parts of the map (corners , rooms, corridor) in a consistent way by checking what walls are surrounding the robot.

Turning Using the Gyro *(motion.ino : **rotateToAngle()**)* **:** with my decision to turn the robot left and right to each of the surrounding walls, I needed a method to accurately rotate a given amount of degrees. Since this robot has no encoders it is impossible to accurately tell how much the wheels have spun in a given span of time, if this was available it could be used to calculate how much to spin each wheel to

spin to a specific angle. Instead the robot does have a built in gyroscope that can be used to measure angular velocity and convert this into a measurement of where the robot is facing - my robot uses the gyroscope to make turns by doing the following:

- Zeros the gyroscope so that wherever it is facing is considered zero degrees
- Adds the target angle to the current heading to calculate where the heading should be at the end of the turn (wrapping this between -180 to +180 degrees (the operating range of the gyro)
- The current heading of the robot is then read to calculate how much error there is between the current heading and the target angle
- This error value is then used in a Proportional Integral control algorithm to calculate speeds for the motors to rotate the robot
- The robot keeps checking the current heading and adjusting the motor speeds until the current heading is within 1 degree of the target angle, stopping the robot spinning when it is.

Correcting the Robot after a turn: after each turn that the robot makes it has the potential to be off by about 1 or 2 degrees, this makes the robot turn fairly accurately but over time these errors can accumulate with the result being the robot facing in an unexpected direction after about 10 turns.

To fix this I had the robot correct its orientation after each turn to the left and right by using each wall as a reference point to correct against, the robot slowly drives up to wards each wall (when facing left / right) with the aim being to get the reflectance sensors on the right and left edges of the sensor array on the wall.

When a reflectance sensor hits the wallthe motor on that side would set its speed to zero, this had the effect of making the robot parallel with the wall on each left and right turn. So if the right reflectance sensor reached the wall before the left, the right motor would set its speed to zero letting the left motor keep driving forward until the left reflectance sensor hit the wall and vice versa. By correcting the robot's orientation on each of its turns any errors that were created during a turn are wiped out, preventing them from accumulating and causing issues.

Detecting Partial Walls: implementing the correction feature described above allows the robot to stop accumulated errors from turns causing issues but it also creates a potential problem. Should the one of the reflectance sensors on the edge of the reflectance sensor array hit a wall ,the robot will begin to correct by moving the opposite motor, but if there is a gap in the wall (for a corner or room) to robot will over correct and end up facing the wrong way.

To solve this issue, I implemented a feature that times how long on average it takes the robot to perform a correction and should a correction that the robot is performing take longer the 1.5 times the average correction time - it stops correcting and signals that a partial wall has been detected. This prevents the robot from over correcting and facing the wrong way.

Detecting Rooms and Corners: to detect rooms and corners a similar timing feature as described above is used. It times how long on average it takes for the robot to encounter a wall after turning left or right and driving forward. Should the robot take longer than twice the average time it takes to travel to a left or right wall it stops moving turns around to keep travelling as normal and signals that no wall has been detected on that side.

To detect corners the robot combines this feature with detecting if there is a wall in front of the robot to determine if it is a corner and whether the corner is a right or left turn.

When the robot hits a wall in front of it, it is probable that the robot did not hit the line straight on and may be misaligned with it, the robot uses a function *(motion.ino : **performReverseCorrections**)* that reverses the robot off the line until the robot is parallel with the line it drove into, this is done so that the robot is not too close to the wall it drove into when we perform further movement, particularly when taking corners or reaching the end of a corridor.

Detecting objects: Once the robot has determined that it is in a room it checks to see if has searched the room by consulting a boolean flag "*roomSearched*", the flag is reset to false when the robot is not in a room and true once it has scanned the room it is in. This ensures that it only searches a room once.

The robot searches a room by performing a 180 degree sweep at the mouth of the room and polling to see if it detects anything within 30cm of its ultrasonic sensor. If an object is detected the robot makes a tick sound as it detects objects in real time, leaves the room then sounds a siren to signal that a person is found - it also sends an Xbee message of this occurrence.

Driving Straight Using the Gyro (motion.ino : ***driveForwardFor()***)**:** to move the robot forward through the course I needed functionality that could move the robot forward in a straight line without it drifting to the left and right, as we established earlier we cannot track the rotations made by each wheel as there are no encoders on the wheels.

However we can use the gyro to drive in a fairly straight line by keeping the robot's current heading constant, the functionality works as described:

- Zeros the gyroscope so that wherever it is facing is considered zero degrees
- The target heading is set to zero
- The current heading of the robot is read and the amount of error from the target heading is calculated from this
- A Proportional Integral control algorithm is used to calculate speeds for each motor based on the amount of error from the target heading, these speeds adjust to keep the robot heading at zero
- The robots keeps adjusting the speeds of the motors to keep the heading at zero until the travel time exceeds the set amount or the robot hits a line in front

This covers the key issues that I encountered while working on a solution for these tasks.