**Programming Things: Assessment 2017\_18**

**Coursework 1 (Individual)**

This assessment is worth 40% of the module marks and contributes to the assessment of all of the module's learning outcomes:

• Identify and critically assess the elements needed within a physical computing system

• Interface a programmable controller with peripheral devices such as sensors, switches, key pads, motors, lights, sound, displays and other input devices and actuators.

• Determine what types of devices are appropriate for various products and processes.

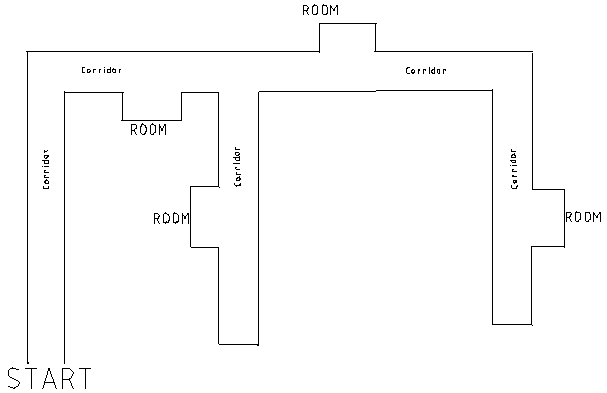
• Design and implement 'control' algorithms for the relevant hardware platforms.

For this assessment, you are required to modify and code a Pololu Zumo robot to perform a (simulated) search and rescue operation.

The scenario **motivating** this assignment is that your robot is trying to find/rescue people trapped on a single floor in a building which is filled with smoke. The robot moves through a corridor and people are to be discovered in rooms or in the corridor. When the robot discovers a 'person' it signals back to ‘base’ so that a search party can come in to rescue that person. The robot, however, continues to search, signalling as and when it finds people in other rooms. When the searchDroid reaches the end of the corridor, it turns around and returns to base (by the quickest route possible, but visiting all the locations where it has found people to confirm they have been rescued).

For the Zumo, this means you will have to 'design' and code the Zumo so that it can explore a ‘maze’ and find a number of 'hidden' objects. You will also need to add an ultrasonic sensor and an xbee to the robot.

The 'building floor' will consist of a corridor with corners and adjoining rooms. The borders will be set out with black lines on a white background. As an illustration, **one possible configuration** of rooms and corridor is given below…



Zumo

*This diagram is not to scale, but is meant to be indicative of relative sizes of elements of the environment the robot is moving in.*

*Everything is 2D: all ‘walls’ are black tape/lines on white paper/card.*

*To help the Zumo ‘know’ where it is, the widths of corridors, rooms doors etc is important.  
The rooms are ‘deeper’ than the corridors.  
The doors are wider than the corridors.  
The corridor is (just) wider than the Zumo.*

*eg: corridorWidth is approx Zumo width\*1.2.*

*The Zumo must stay INSIDE the corridor.*

*Room depth is at least 1.5\*corridorWidth. (you need enough depth for the U/s sensor to work too)*

*Door width is 2\*corridorWidth.*

*The placing of the objects is also indicative. They could be anywhere in the room or in the corridor.*

**TASKS**

Your aim is to get the robot to perform certain tasks:

**Task 1:** The Zumo can be driven down the corridor from a **GUI\* (see comment below)** eg 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).

**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.

**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 (eg 'C' or 'c' for complete). This then reactivates the task 2 behaviour so that the Zumo can drive itself down the corridor again.

**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 (eg ‘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.   
**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.   
**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.

Zumo

Zumo

Zumo

**Task 5:** At the end of the main corridor, the Zumo should stop (as in task 3) and wait until it is told that it has reached the end by receiving a keypress ("E" or "e"). From this point, it should ‘optimise’ the return route and navigate its way back to the start without intervention by the user. It should avoid rooms that were empty and ‘check’ locations that had people in. If a room is still occupied, that should be signalled over the xBee. The robots (pin 13) LED should turn on – to provide a ‘follow me’ guide - which should turn off when the robot gets back to base.

**Comment on GUI development**:

* The expectation is that you will use Processing for the GUI, however there are libraries for both Java and C# which support communications through Serial ports. In fact, Processing uses the Java library. SO, if you are more comfortable with using GUI devpt in VS or Netbeans, feel free to explore those options. A proof of concept C# solution is provided on Bboard.
* There are several GUI libraries for Processing that will help you (eg ControlP5, G4P, GameControlPlus..). G4P is probably the easiest since it also provides a GUI Builder tool.
* From the Processing environment, the File ->Examples->Libraries-> Serial->SimpleWrite sketch does the same thing as the terminal in the Arduino PhysicalPixel example (In windows, you just need to set the portName variable to "COM4" or whatever the port is that the xBee is plugged into for communications). You might also want to look into this: <https://processing.org/reference/libraries/serial/>

Marks will be awarded according to how many and how successfully you have completed these tasks. Additionally, marks will be awarded for the usual things (quality of code, commenting) and also how quickly your robot can navigate the track and how adaptable your robot is (ie you can bring your own track and demo it working, but you’ll get a better mark if your robot can cope with the track I provide…)

**WHAT TO HAND IN…**

**You should submit, to Blackboard, by 10:00am on Monday 15th Jan 2018,**

**A zipped clone of (and a url linking to) a subversion or GIT repository (which I should be a team member on). That repository should contain:**

* **your code, which should be fully documented.** *If you feel confident, this could be (eg) Doxygen’d or Moxygen’d and included within the Repo.*
* **a report/wiki/readme.** This shoulddescribe what you have achieved (against the objectives set), why and how you resolved (or attempted to resolve) key issues. It should also identify and acknowledge any sources for code that you have used and how they have been incorporated/adapted. This COULD be part of the documentation (eg the readme) supporting the repository.
* **a video of your robot navigating your track…**

**MARK SCHEME:**

**The work will be largely marked by demo in week commencing 15th January, 2018.**

**Functionality (80%)**

**Basic Pass (45%)**

Zumo successfully performs tasks 1, 2, 3 **(15 marks each)**

**Reasonable Pass (55%)**

Zumo also successfully performs task 4A **(10 marks)**

**Good Pass (65%)**

Zumo also successfully performs task 4B, C **(10 marks)**

**Very Good Pass (80%)**

Zumo also successfully performs task 5 **(15 marks)**

**Very Very Good ~ Excellent Pass (80 - 100%)**

Extra marks are available for:

* additional features such as speed of robot, adaptability to (changes in) the track/environment (eg light conditions, more than one obstacle, different corners etc), building and displaying a map of robot’s route in the GUI, the getting the robot to do the forward tracking completely autonomously too, etc
* good programming practice eg using the state pattern or pushing the boundaries eg using Collections classes rather than arrays (<http://andybrown.me.uk/2011/01/15/the-standard-template-library-stl-for-avr-with-c-streams/> ) etc…

Marks will be awarded for the tasks proportionately according to a Likert-type scale breakdown:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Task | Not Attempted | Serious attempt but with no real success.  Code is on the right lines, but not quite getting there. | Works sometimes (or silly errors stop it) | Works most of the time but is unreliable and prone to flake-over | Works reliably & robustly all of the time | Mark |
|  | 0 | 20%+ | 45%+ | 60%+ | 85%+ |  |
| Task 1 (15) |  |  |  |  |  |  |
| Task 2 (15) |  |  |  |  |  |  |
| Task 3 (15) |  |  |  |  |  |  |
| Task 4 A (10) |  |  |  |  |  |  |
| B (5)  C (5) |  |  |  |  |  |  |
| Task 5 (15) |  |  |  |  |  |  |
| Extras (20) |  |  |  |  |  |  |
| Total |  |  |  |  |  |  |

**Code quality and report (20%)**

These will be marked using the following Likert Scales:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Code is a mess, nothing works.  No evidence of version control.  Code just submitted as a single file. | Code quality is acceptable: reasonable layout, variable naming etc.  Version Control only used as a last minute depository.  Code from other sources clearly acknowledged. | |  |  | Code quality is excellent, readable and conforming to standards of custom and practice showing good insight into required techniques  Excellent use of version control with the report/wiki mentioned below fully embedded and generated from comments. | |
| 0………….………………4……………………………………………………………………………………………………………………10 | | | | | | | |
|  | Report/wiki/readme is minimal, but acceptable in terms of reporting on what does what, where and why | | Report/wiki/readme gives good explanation of strategies successfully used  Commented code gives and/or proposes a reasonable strategy that was attempted and explains why the implementation didn't succeed.  Mark here depends on depth and range of strategies suggested | | | | All features implemented and  Report/wiki/readme gives excellent explanation of strategies successfully used |
| 0……………………………….4…………..……………………………………………………………………………………..………..…10 | | | | | | | |