|  |  |  |
| --- | --- | --- |
| Module Leader: Mick Marriott | | Level: 6 |
| Module Name: Robotics | | Module Code: 55-608216 |
| Assignment Title: Search & Rescue | | |
| Individual | Weighting: 40% | Magnitude: *2000 words* |
| Submission date/time:  Thursday 2nd March 2023 at 3PM | Blackboard submission | Format: 15 minutes video plus source code. |
| Planned feedback date:  Monday 27th March 2023 | Mode of feedback: Blackboard | In-module retrieval available: Yes |
|  | | |
| **Module Learning Outcomes**     * LO1. Identify and critically assess the elements needed within a physical computing system * LO2. Interface a programmable controller with peripheral devices such as sensors, switches, key pads, motors, lights, sound, displays and other input devices and actuators. * LO3. Determine what types of devices are appropriate for various products and processes. * LO4. Design and implement 'control' algorithms for the relevant hardware platforms. | | |

Robotics Assessment One

# 

In this task you must use a Pololu Zumo 32U4 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 along a series of corridors 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 robot reaches the end of the corridor it turns around and returns to base.

The robot has no prior knowledge of the structure of the building and does not know in advance how many people need rescuing. This means that you must design and code the Zumo so that it can explore a maze to find an unknown number of hidden objects. You will also need to use an XBee module with the robot so that it can communicate wirelessly with your computer.

The building floor is a set of corridors with junctions and adjoining rooms. The "walls” are black lines on a white background. The lines may be drawn or laid out using decorator’s or electrician’s tape. The floor should be card.

The following illustration shows the **default configuration** of rooms and corridors. Your system should, as a minimum, be able to explore this set up. You may add one other more complex configuration if you wish.

|  |
| --- |
| Zumo    **START!**     * This diagram is not to scale. It is indicative of relative sizes of elements of the environment in which the robot is moving. * Your implementation should be 2 metres from top to bottom. * 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. are important. * The corridor is 1.2 times the width of your Zumo. * Rooms are five times the width of the Zumo. * Each room has an open doorway that is 2.5 times the width of the Zumo. * The Zumo must stay inside the corridors and rooms. * The placing of people is indicative. They could be anywhere in the rooms or corridors. * When your Zumo finds a person, it must approach up to them, not crash into them, and report their location to base. |
|  |

# Requirements

Your aim is to get the robot to do the following:

1. Move along the corridors without straying past the walls.
2. Enter each room that it finds through the doorway.
3. Move around the room searching for people.
4. Move up to any people it finds and report their location.
5. Search all corridors and rooms.
6. Return to base by the fastest possible route.

You should implement three control modes.

## Mode one: manual control of the Zumo.

The Zumo can be driven down the corridor by a user who controls it through the GUI (Graphical User Interface) that you will design. The user should be able to enter simple commands either via a text field or using key presses such as w, a, s, d, and space. Communication from the PC to the Zumo is done via the xBee communication channel.

You need to generate a GUI using your chosen programming language. Each programming language has its own libraries to support GUI development and each of them has its pros and cons. In addition, each programming language has libraries to support communications through Serial ports. A non-exhaustive list of language options includes:

* C# and a framework such as
  + <https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor>
  + <https://www.avaloniaui.net/>
  + <https://platform.uno/>
* Java with swing
* C++ with Qt or WxWidgets
* Python with WxPython
* Processing with ControlP5, G4P or GameControlPlus
* Proof of concept C# and Processing solutions are provided on Blackboard.

Comments on GUI development with Processing:

* There are several GUI libraries for Processing that will help you. G4P may be 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 investigate <https://processing.org/reference/libraries/serial/>

## Mode two: Semi-Autonomous control of the Zumo

The Zumo automatically keeps within the corridor by using the line 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. If Zumo hits one of the side walls of the corridor, it modifies its path to keep within the walls of the corridor.

If the Zumo recognises that it has reached a corner, it stops and returns to manual control by:

1. Sending a message using the XBee indicating that fact. The messages received from the Zumo should appear in the GUI.
2. It then deactivates the autonomous behaviour from task 2 (which is keeping the Zumo between the corridor walls); this allows the (human) controller to turn the robot. When that turn is complete, the operator signals that by sending another keypress.
3. This then reactivates the autonomous behaviour so that the Zumo can drive itself down the corridor again.

When the Zumo recognises it is at the end of a corridor, it sends a message through the XBee that appears in the GUI. The user presses 'L' or 'R' to rotate the Zumo 90° to the left or 90° to the right. To complete the rotation and the movements, wheel position encoders can be used to measure the space covered by each wheel. Additionally, the on-board gyroscope may be used to sense the rotation. Once the turn is complete, the Zumo should automatically return to its autonomous behaviour and drive itself down the corridor again.

The robot recognises a doorway and stops. It signals that it has stopped at a doorway by sending a message identifying whether the room is on the left or right of the corridor. The operator initiates the rotation algorithm, as at the end of the corridor. The Zumo enters the room to perform a scan, using its proximity sensors, for objects. Depending on the size of the room, proximity sensors may require the Zumo to roam the room to complete the search. If an object is detected, the Zumo moves to the object and reports its location using the XBee link. Once the scan is complete, the Zumo returns to the doorway, stops, and signals the operator who guides it into the corridor.

At a T-junction, the robot may turn either way to search the remaining corridor and rooms. At the end of any offshoot corridor the Zumo should stop and wait until it is told to turn around and retrace its route to the T-junction.

At the end of the corridor, the Zumo should stop and wait until it is told that it has reached the end by receiving a keypress (‘H’). This time the Zumo should recognise that is must return home, which it does by following the corridors back to base.

## Mode three: Fully autonomous.

The robot behaves as in the semi-autonomous mode but makes all decisions itself without intervention from the operator.

## Additional hints

If the robot has problems detecting the “walls”, use the PVC tape which gives better reflections and improves the robot’s sensitivity.

In some situations, there can be interference from the simultaneous use of multiple sensors, particularly the infrared and ultrasonic sensors. If this happens and you have problems detecting objects in the rooms, you should temporarily switch off unneeded sensors.

# What to hand in…

You should submit, to Blackboard, BY 3:00PM ON THURSDAY 3rd March 2023, A URL LINKING TO A GIT REPOSITORY. Your GIT repository must be accessible to the teaching team during the marking period.

That repository must contain:

1. Your code. This should be documented appropriately through in-code comments and a readme file. You will use many sources in programming this application. Include references to these sources in the readme file. The readme file needs to also contain instructions to build and use your program including performing each of the tasks mentioned above.

1. A video of the robot performing each of the tasks described earlier, and the recording of your screen while the robot performs it. You can do this with your mobile, whilst executing each of the tasks, or combining the recording of the screen and a camera.

# Grading scheme

Your work will be graded on the following facets using the current level six version of the Common Grade Descriptor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Facet** |  | **Available marks** | **At pass level your work may look like** | **At a distinctive level your work may look like** |
| **Mode one** | **Completeness** | **5** | Some attempt to implement the mode. | Implemented.  Can handle errors and edge cases.  Robot behaves as expected. |
|  | **Operation** | **5** | Some functionality is present. | Mode is fully working. |
|  | **Coding** | **5** | Poor use of coding for the robot controller.  A poor-quality UI. | Well-implemented.  Code is readable and easy to follow. |
| **Mode two** | **Completeness** | **5** | Some attempt to implement the mode. | Implemented.  Can handle errors and edge cases.  Robot behaves as expected. |
|  | **Operation** | **5** | Some functionality is present. | Mode is fully working. |
|  | **Coding** | **5** | Poor use of coding for the robot controller.  A poor-quality UI. | Well-implemented.  Code is readable and easy to follow. |
| **Mode three** | **Completeness** | **5** | Some attempt to implement the mode. | Implemented.  Can handle errors and edge cases.  Robot behaves as expected. |
|  | **Operation** | **5** | Some functionality is present. | Mode is fully working. |
|  | **Coding** | **5** | Poor use of coding for the robot controller.  A poor-quality UI. | Well-implemented.  Code is readable and easy to follow. |
| **Graphical user interface** |  | **10** | UI is a simple text field.  Limited interactivity via keystrokes.  May display some messages. | A modern GUI is presented.  All 3 modes are accessible via the UI.  User has full control using both keystrokes and GUI controls.  All messages are displayed. |
| **Readme file** |  | **10** | Some information is presented.  Plain text or limited use of Markdown. | Full build and usage instructions are presented.  Good set of sources are referenced.  Presented in the style of a GitHub Readme, |
| **Demonstration of your understanding in the video** |  | **20** | Low quality video.  Understanding the operation of the robot is difficult. | Good quality video showing the robot and the GUI.  Has a commentary that explains what is happening. |
| **Code quality** |  | **10** | Sensible names.  Some comments.  Some structure. | Well-structured code.  Meaningful comments throughout. |
| **Solving other layouts** |  | **5** | Works only in mode 1. | The robot can follow any layout using mode 3. |