Zumo Rescue Operation

Assessment 1 report

Author: Diego Ochando Torres (SHU ID: 30022981)
Course: CS4G Final year (2020/21)
Module: Programming Things

Table of contents

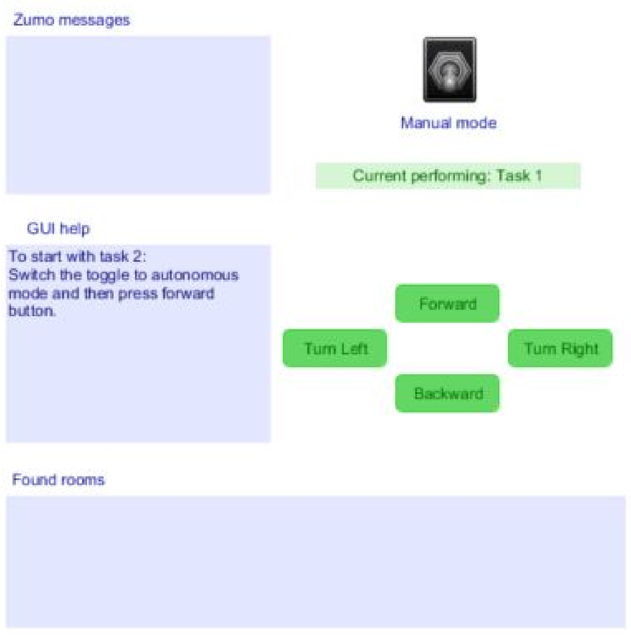
Repository link: https://github.com/Yeyoxando/ZumoRobot

Zumo technical guide

In this section we are going to cover how to perform each task using the developed GUI for it. Help messages are also shown in runtime on the GUI to help the user on each task performance.

How to perform each task?

Task 1

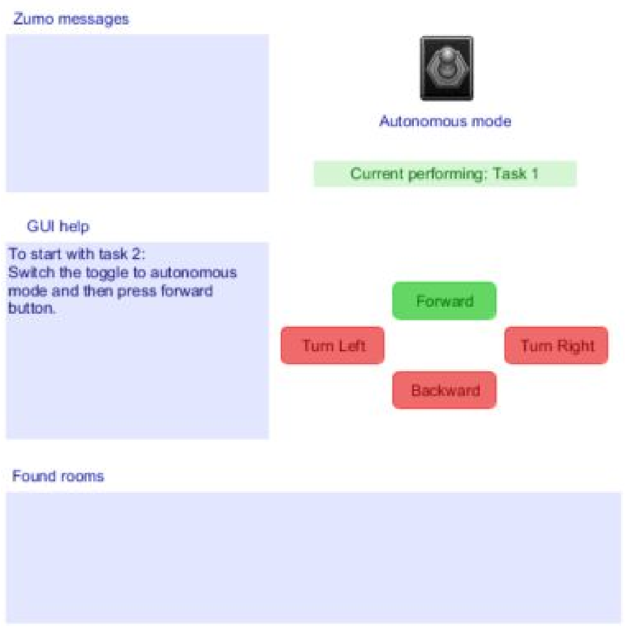


To start with task 2:
Switch the toggle to autonomous mode and then press forward button.

Controls
By pressing one of these four buttons the robot will move manually controlled by the user.

What the robot does
The robot will do the action indicated in the button until the user releases it.

Task 2



To start with task 2:
Switch the toggle to autonomous mode and then press forward button.

Controls
To start with the second task, the user must press the switch button to set the robot on autonomous mode, all buttons except the forward one turn to disabled.
If the user presses the forward button, the zumo will start task 2. All buttons get disabled until it reaches a wall.

What the robot does
When the user starts the second task, the robot will move forward until a maze border is detected on the front line sensor. If the left or right sensor detects a border, the robot will turn a bit to the opposite direction to avoid the "walls".

Task 3

Put GUI Image (I need the robot)
Controls
After the robot detects a wall in front of it, it will stop and the left and right buttons will get enabled.

What the robot does
Initially, the robot returned the control to the user to turn it manually in the desired direction, but this task was modified to accomplish task 4 objective, where the robot turns autonomously and the user has to press one of the buttons making the robot perform a 90 degrees turn in the indicated direction.

Task 4

Put GUI Image (I need the robot)
Controls
Same as task 3.

What the robot does
As it was commented on task 3, the robot turns autonomously 90 degrees in the indicated direction, using the gyroscope with the implemented method to sense the rotation.

Task 5

Put GUI Image (I need the robot)(I need to finish this task)
Controls
When the robot is performing the second task, the user can send a message to the robot, signaling that it is about to enter a room. This is done by pressing the ‘Room!“ button. Right and Left buttons will get enabled.

What the robot does
When the button to start this task is pressed the robot will stop at the entrance of the room and then the user has to press left or right to indicate in which direction the room is. Then the robot will start reckoning the room by doing a zig-zag and storing its data while it is using the proximity sensors to detect any object in the room.
When the complete room is inspectioned the robot will return to the corridor and it will send if there are any people inside the room to the GUI.

Task 6

Put GUI Image (I need to finish this task)
Controls
Not done yet

What the robot does

Task 7

Put GUI Image (I need to finish this task)
Controls
Not done yet

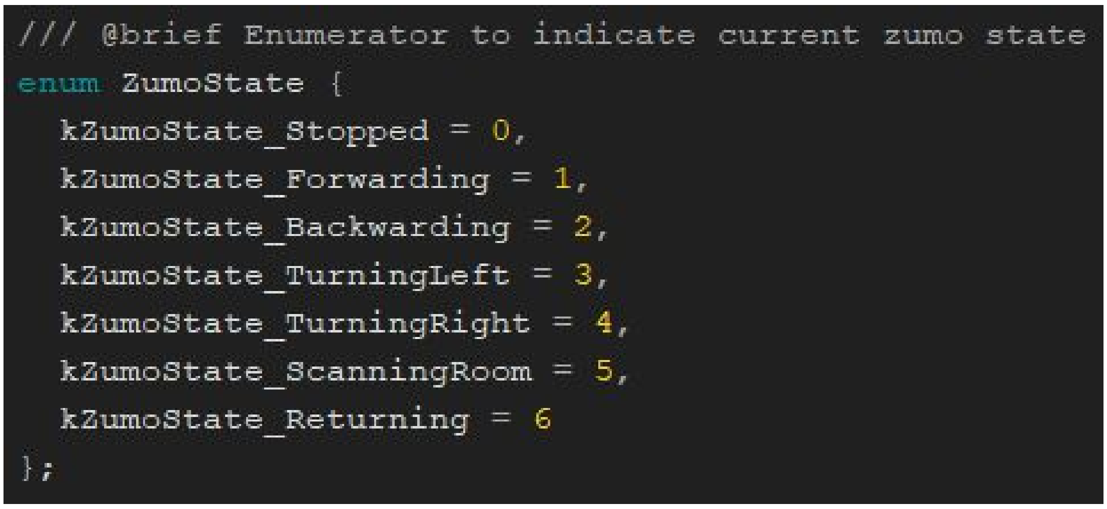
What the robot does

Zumo Robot functioning

In this section we are going to see how the robot has been programmed and how it works internally, although a detailed specification and explanation of the implemented methods for the robot can be found in the project wiki, in a more oriented way to API design.
The robot works with a .ino file to use the Zumo robot board, and a C++ header and source are used for the declaration of a ZumoRobot class that contains all the necessary variables and methods to perform the tasks.

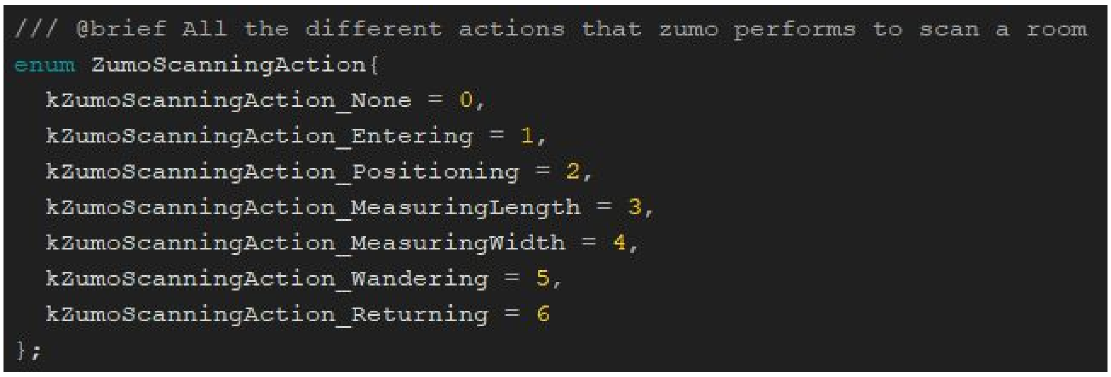
The Zumo robot has been programmed following the finite state machine programming pattern, which is implemented using enumerators to indicate the current state or actions on the robot, which will execute a different case on a switch in the program loop. In the lines below we are going to look at some of these states and see their usage.

ZumoState
This represents its main machine state to perform all the different tasks of the robot, it is used in the Loop code of the arduino program as different cases for a switch, where it executes different functions depending on the state that it has assigned.



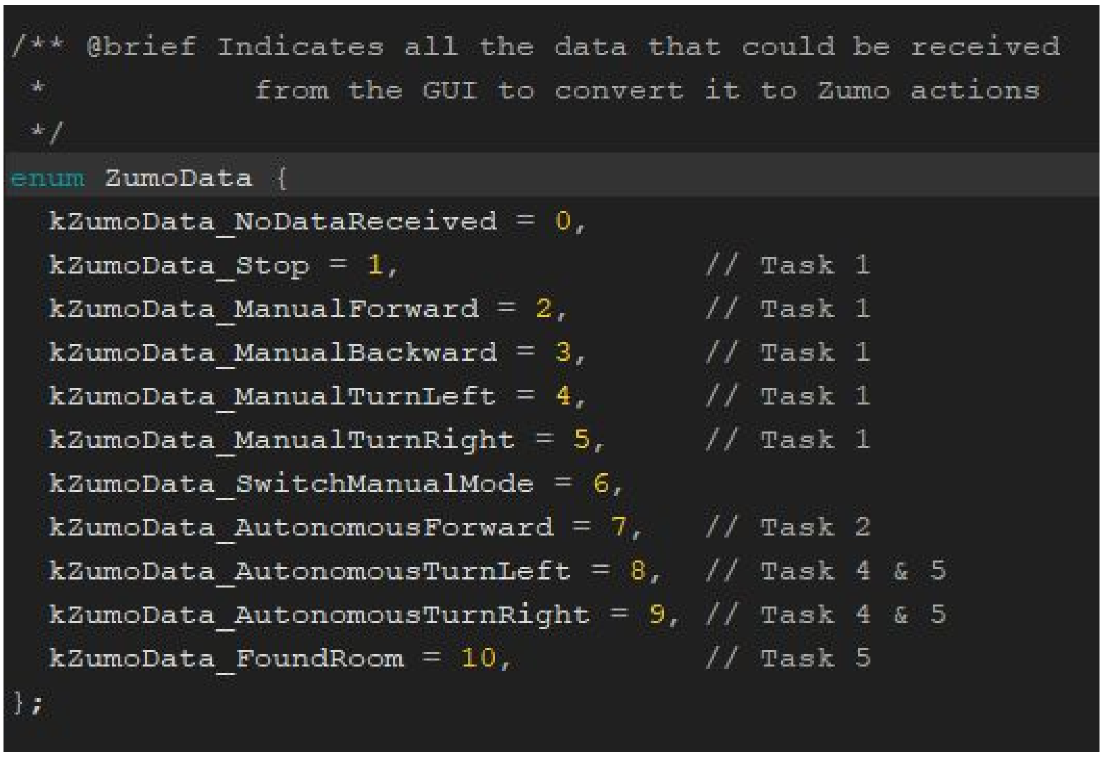
```
/// @brief Enumerator to indicate current zumo state
enum ZumoState {
  kZumoState_Stopped = 0,
  kZumoState_Forwarding = 1,
  kZumoState_Backwarding = 2,
  kZumoState_TurningLeft = 3,
  kZumoState_TurningRight = 4,
  kZumoState_ScanningRoom = 5,
  kZumoState_Returning = 6
};
```

ZumoScanningAction
In a similar way, we have another state machine to control only one task, specifically the task 5, where the robot has to complete a series of steps to correctly perform the task.



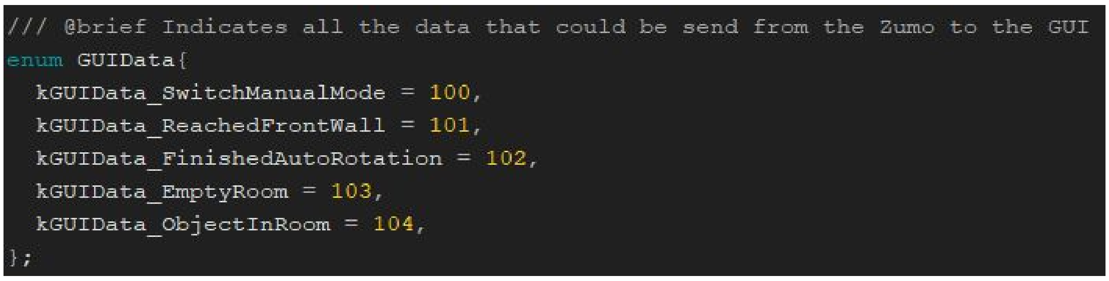
```
/// @brief All the different actions that zumo performs to scan a room
enum ZumoScanningAction{
  kZumoScanningAction_None = 0,
  kZumoScanningAction_Entering = 1,
  kZumoScanningAction_Positioning = 2,
  kZumoScanningAction_MeasuringLength = 3,
  kZumoScanningAction_MeasuringWidth = 4,
  kZumoScanningAction_Wandering = 5,
  kZumoScanningAction_Returning = 6
};
```

ZumoData
It also has an enumerator to interpret the received data from the serial connection with the GUI on the PC, and depending on the receive input, it will execute actions in consequence.



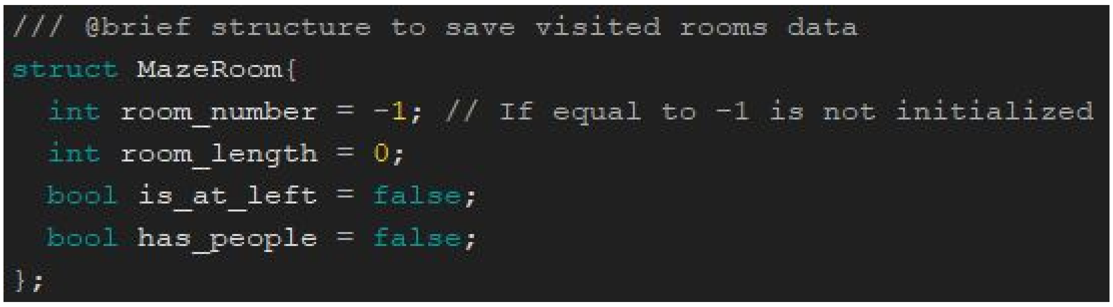
```
/** @brief Indicates all the data that could be received
 *          from the GUI to convert it to Zumo actions
 */
enum ZumoData {
  kZumoData_NoDataReceived = 0,
  kZumoData_Stop = 1,                 // Task 1
  kZumoData_ManualForward = 2,        // Task 1
  kZumoData_ManualBackward = 3,       // Task 1
  kZumoData_ManualTurnLeft = 4,       // Task 1
  kZumoData_ManualTurnRight = 5,      // Task 1
  kZumoData_SwitchManualMode = 6,
  kZumoData_AutonomousForward = 7,    // Task 2
  kZumoData_AutonomousTurnLeft = 8,   // Task 4 & 5
  kZumoData_AutonomousTurnRight = 9,  // Task 4 & 5
  kZumoData_FoundRoom = 10,           // Task 5
};
```

GUIData
Likewise, it also has an enum for the data that has to be transferred from the robot to the GUI, so it can update its layout or show information depending on what is happening.



```
/// @brief Indicates all the data that could be send from the Zumo to the GUI
enum GUIData{
  kGUIData_SwitchManualMode = 100,
  kGUIData_ReachedFrontWall = 101,
  kGUIData_FinishedAutoRotation = 102,
  kGUIData_EmptyRoom = 103,
  kGUIData_ObjectInRoom = 104,
};
```

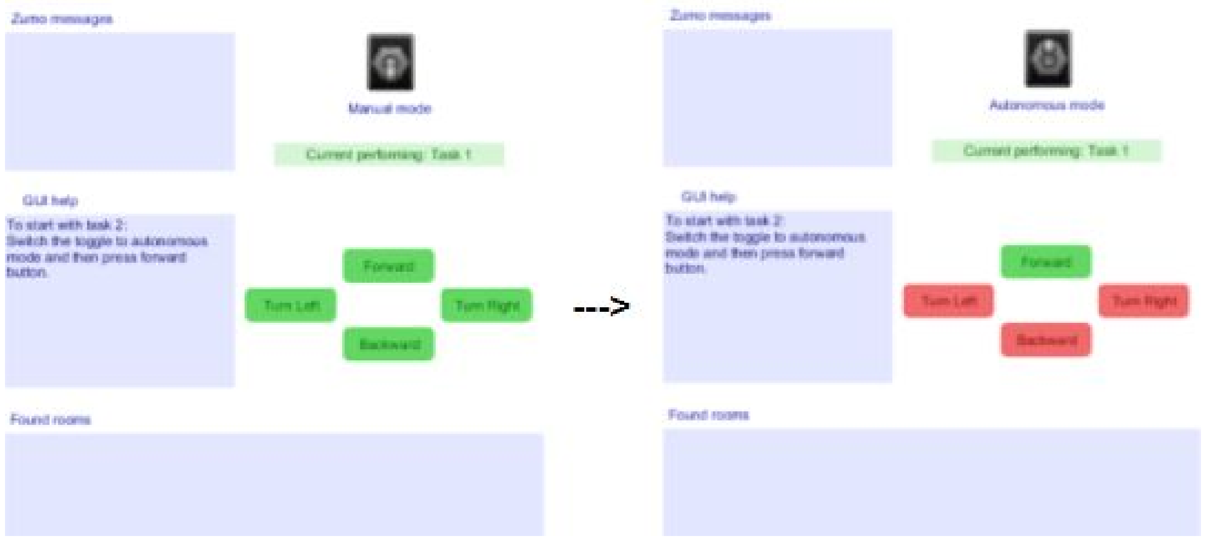
MazeRoom
Finally, a structure to store data from the visited rooms is also included, where data like the number of the room, its direction or if it has people inside.



```
/// @brief structure to save visited rooms data
struct MazeRoom{
  int room_number = -1; // If equal to -1 is not initialized
  int room_length = 0;
  bool is_at_left = false;
  bool has_people = false;
};
```

Zumo GUI functioning

In this section we are going to see how it works the GUI of this project internally. With the provided GUI the user can control the robot remotely from a PC by using the Xbee connection. This GUI was implemented using the processing IDE and language and the G4P library with the GUI builder tool.
As we have seen in the first section of this report, it changes and reacts dynamically to the input actions while it communicates the data to the robot at the same time. As we can see in this example, after changing the switch from manual to autonomous the layout changes.



To start with task 2:
Switch the toggle to autonomous mode and then press forward button.

To start with task 2:
Switch the toggle to autonomous mode and then press forward button.

It also contains labels that change as the user input actions, the use of them is to provide help on how to perform other tasks, or to show information about what is happening. It also includes a label to indicate which rooms the robot has found and if they are empty or not.

The GUI is composed by 4 .pde files which are detailed next:

- **zumoGUI.pde** -> this is the main file of the GUI, which initializes and does the loop for the update on every frame, it will also receive the data from the serial and act consequently by calling different methods and modifying the GUI.
- **gui.pde** -> this file is autogenerated by the GUI builder tool provided by G4P. It setups the GUI layout at the beginning, and it will bind methods to the buttons. Anytime that the tool is open, it deletes the file and creates a new one, because of that, my own code and their comments has been moved to another helper file (guiButtonFunctions.pde).
- **guiButtonFunctions.pde** -> this file is a helper to contain the code and comments of the methods that are called by each different button, as they were deleted after opening the GUI builder tool.
- **guiHelp.pde** -> in this file we can find helper functions such as the ones to enable or disable the buttons, we can also find some text variables that will show up to help the user dynamically.

A detailed specification and explanation of the implemented methods for the GUI can be found in the project wiki, under the files commented right above.

Implementation issues

Certain problems have been faced during the development of the robot, they will be detailed in the next lines.

Rusty with Java-like code on processing -> A lot of time has passed since I programmed in Java in high school, so it was a bit slower compared to coding C/C++ when I started to program the GUI.

Calibrating sensors and first robot programming -> This was the first task where I got stuck, I tried lots of things until I found some examples in the Zumo IDE and some API documentation on Zumo's webpage, which has helped a lot later on other things where I faced the same problem.

Imprecise turns with encoders -> At the beginning I was measuring the turns with wheel encoders, in the first tasks it was working fine, but later when I started to wander the room the robot got some inacceptable offset after performing some turns. This was solved using the gyroscope, with a method that calculates the time that it has to rotate at a certain speed to reach the required degrees, finally it turns out that it works better with this method and I changed the system also for the other tasks.

Doxygen does not work with processing .pde files -> To solve this problem I found out that Doxygen supports java files, as .pde files are based on java, I only had to convert them to java files and then run Doxygen. To speed up this process I have written a batch file to do all of this process automatically (generateDoxygen.bat in project folder).

Can't bring Zumo on the plane to Spain -> I was unable to test the written code for more than one month, I made all the possible tasks before going to christmas break, and finally I requested an extension up to 4th of february and I received another package on xxth january.

Tried to use arv_stl -> I tried to add this library to my project in order to use collection types like vectors to store dynamically the rooms on the maze, so it would be easier to adapt it to any maze. It did not work finally for compilation and compatibility issues, so I finally had to make it with a fixed size array.

Critical evaluation

To close this report I am going to evaluate my own performance on this project, starting with the fact that I did leave my robot things for the project in the UK, and I could not work on it for more than 1 month, may have affected my performance, because I got rusty after all of that time without touching it, and I may be reached further without that stop.

In another way, I think that looking for code examples has helped a lot in some things like the sensors calibration or the use of the gyroscope. Using the repository has also helped me at the hour of implementing some steps where I did not know how to do them, when the code turned into a mess it was easier to start from scratch returning to the last repo commit than trying to fix what I have done.

Finally, I have learned that some tasks as well as some parts of the code could be better, but I think that as my first time programming a robot, messing with electronics and dealing with all of these new things, I had obtained a good result which I am proud of.

Used Resources

External resources have been used for the implementation of some features, these resources are detailed down below.

Processing examples
Retrieved from inside the processing IDE after adding the Zumo library.
Used the ones for serial communication sending and receiving.

G4P documentation
Retrieved from http://www.lagers.org.uk/g4p/ref/index.html
Used for the creation and scripting of the GUI.

Pololu QTR sensors
Retrieved from https://www.pololu.com/docs/0J19/3
Used for sensor calibration tasks.

Zumo 32U4 arduino library examples
Retrieved from inside the Arduino IDE after adding the Zumo library.
Used LineAndProximitySensors example for line and proximity sensors calibration and usage. Encoders and InertialSensors examples used to know how the wheel encoders and the gyroscope works.

Pololu Zumo32U4 Robot User's Guide
Retrieved from https://www.pololu.com/docs/0J63
Used to know about the components of the robot in a more technical way.

TODO

REPORT
Finish explain tasks and put photos of the Gui on tech guide
put received date of the 2nd package in impl issues

CODE
Finish task 5, 6 and 7 implementation

VIDEO
Not possible until robot receiving