**Technical Report – User Interface for WiFi Communication Between Laptop and Raspberry Pi**

**Requirements:**

Write a program so the user can enter a command and value (e.g. “Go forward for 3 seconds” or “Turn left for 2 seconds”), then press a button which sends that information via WiFi to the Raspberry Pi attached to the robot, which then executes the command. A desirable feature would also allow for multiple commands to be sent at once to be executed consecutively. In addition, the robot should be able to send values (e.g. the value from the ultrasonic sensor ) back to the program upon request of the user or at regular intervals.

**Class Analysis:**

There are seven classes, ClientServer, Client, CommandPanel, ChoosePanelNumber, ConditionPanel, UltrasonicGetter and RobotUser:

The ClientServer contains methods for reading from and writing to the input and output streams.

The Client class extends ClientServer and contains a Client constructor which creates the socket to communicate with the Raspberry Pi. It also contains methods for handling events from the GUI.

The CommandPanel extends JPanel and contains a constructor which creates the drop down menu and textfield so the user can input commands. It also contains event handling methods which standardise inputs and pass to methods in the Client class to communicate with the Raspberry Pi.

The ChoosePanelNumber class extends JPanel and contains a constructor which creates a drop down menu and button to choose how many instances of CommandPanel to create and add to the GUI to allow for multiple commands to be sent at once. It also contains event handling methods which create and add instances of CommandPanel to ChoosePanelNumber when the ‘Ok’ button is pressed and calls methods in CommandPanel and ConditionPanel if the ‘Go!’ button is pressed.

The ConditionPanel class extends JPanel and contains a constructor which contains a checkbox, a drop down menu containing commands and two text fields for input so the user can choose what the robot should do and how long for if an object has been detected with a certain sensor and/or within a certain distance. It contains methods to get the selected item in the drop down menu or the input in the textfields so that they can be called in the handleConditionEvent() method, which then passes to a method in the Client class to send the information to the Raspberry Pi.

The UltrasonicGetter class extends Thread and contains a run method which reads the ultrasonic sensor value from the input stream every two seconds if a button is pressed using a method in Client. It then prints the value to a textfield in RobotUser. It stops reading when the button is pressed again.

The RobotUser class extends JFrame and contains a constructor which adds an instance of ChoosePanelNumber and ConditionPanel in addition to arrow buttons for control of the pan and tilt servos on the robot, buttons to make the robot follow a line and stop, and a button and textfield to receive distance values from the ultrasonic sensor. It also adds a menu bar with an option to shut down the Raspberry Pi. When the buttons are pressed, methods from the Client class are called to read from or write to the input and output streams as appropriate. The main method in this class creates a new instance of Client and RobotUser, starts the UltrasonicGetter thread, sets the size of the window and exits the program if the window is closed.

**Class Diagram:**

RobotUser

* messageDisplay: JTextArea
* ultraDisplay: JTextArea
* messageScrollPane: JScrollPane
* ultraScrollPane: JScrollPane
* ultra: JButton
* followLine: JButton
* stop: JButton
* clearUltra: JButton
* clearMessage: JButton
* servoInit: JButton
* left: BasicArrowButton
* right: BasicArrowButton
* up: BasicArrowButton
* down: BasicArrowButton
* mainPanel: JPanel
* pantilt: JPanel
* otherPanel: JPanel
* ultraPanel: JPanel
* messagePanel: JPanel
* menuBar: JMenuBar
* raspberryPi: JMenu
* shutdown: JMenuItem
* tabbedPane: JTabbedPane
* instructionPanel: ChoosePanelNumber
* conditionPanel: ConditionPanel

+ actionPerformed(ActionEvent)

+addBorder(JPanel, String)

+main(String[])

ChoosePanelNumber

* numbers[]: String
* numberOfPanels: JComboBox
* prompt: JLabel
* ok: JButton
* go: JButton
* mainPanel: JPanel
* commandPanels: JPanel
* panelArray: List<CommandPanel>

+ actionPerformed(ActionEvent)

+handleEvent()

CommandPanel

* instructionNo: int
* number: String
* commands[]: String
* choices: JComboBox
* input: JTextField
* labelSeconds: JLabel
* forLabel: JLabel
* commandNo: JLabel

+handleUserEvent()

JFrame

ConditionPanel

* checkbox: JCheckBox
* commands[]: String
* sensors[]: String
* commandChoice: JComboBox
* sensorChoice: JComboBox
* distanceInput: JTextField
* valueInput: JTextField
* ifLabel: JLabel
* withLabel: JLabel
* unitLabel: JLabel
* timeLabel: JLabel
* forLabel: JLabel

+ getTriggerDistance(ConditionPanel): String

+getCommand(): String

+getValue(): String

+handleConditionEvent(ConditionPanel)

+itemStateChanged(ItemEvent)

Client

- socket: Socket

- IP: String

- port: int

+ handleNetworkEvent(String)

+getValue(): String

Thread

UltrasonicGetter

- getUltra: boolean

+ run()

ClientServer

+ readFromSocket(Socket)

+writeToSocket(Socket)

**Implementation and testing:**

The code has been written to implement the above classes and methods and can be found on GitHub at legorovers/PiRover. The testing will test that the program reacts as expected to user input and also gives expected outputs. It will also test that the user interface stays responsive throughout.