|  |
| --- |
| Laboratory 5:  Graphical User Interface Development SEG4145 April 9, 2015 Martin Moncion 6146324 Jeremy Sivaneswaran 3116318 |
| Objective The objectives of this lab is to Control the operations of an autonomous robot in real-time using a graphical user interface. Design and Algorithms The algorithm we used on the arduino robot was the same as for lab 4. In our main loop are program is placed into a do while loop and waits for an instruction from the controller and then performs the instruction. The disconnects itself and prints disconnect when the stop command is sent to the robot.  The readInstruction and conductAction method both use switch case methods to determine what action has been read and needs to be performed. If no instruction is received conduct action does nothing. After every action performed it send an acknowledgment to the GUI. The GUI only accepts numerical input anything else will be rejected.   * Case1: Move Forward * Case2: Move Backwards * Case3: Rotate Clockwise * Case4: Rotate Couter-Clockwise * Case5: Get Distance To Nearest Object * Case6: Get Temperature * Case7: Exit   On exit the Robot will exit are while loop and display disconnected on the LCD screen. Flowchart  Discussion We had some problems with communicating with the robot originally. At first we sending the Arduino char signals, the robot was only able to read two characters at time for some reason. Any sequence longer would get cut off. We were never able to figure out why instead we sent string signals to the robot and it ended up responding properly. Are UI had no problems. We got stuck for a short while because we were sending two messages to our robot at one point but we were able to resolve the issue.  Overall we were able to field test our implementation and our robot is able to execute our algorithm flawlessly. Controller Souce Code /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package lab5;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.InetAddress;  /\*\*  \*  \* @author Jeremy  \*/  public class Control {    private static int PACKETSIZE = 100;  private static int PORT = 9876;  private static int BYTE\_LENGTH = 64;    private static InetAddress host;  private static int receivePort;  private static DatagramSocket socket;    public Control() {  int choice = 0;  try {  socket = new DatagramSocket(PORT);  }catch (Exception e){  System.out.println(e);  }    }    public String waitForRemoteGreeting(){  String message = "";  byte[] receiveData = new byte[BYTE\_LENGTH];  boolean waitingForGreeting = true;  try{  do{  System.out.println("Listening for remote greeting: ");  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);    do{  socket.receive(receivePacket);  }while(receivePacket.getLength() == 0);  message = new String (receivePacket.getData(), 0, receivePacket.getLength());    if(message.equals("Hello server!")){  host = receivePacket.getAddress();  receivePort = receivePacket.getPort();  waitingForGreeting = false;  return new String("Received Greeting: " + message + "\n");  }  }while(waitingForGreeting);  }catch(Exception e){  System.out.println(e);  }    return null;  }    public String waitForMessage(){  String message = "";  byte[] receiveData = new byte[BYTE\_LENGTH];    try{  System.out.println("Listening for message: ");  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);    do{  socket.receive(receivePacket);  }while(receivePacket.getLength() == 0);    message = new String (receivePacket.getData(), 0, receivePacket.getLength());  System.out.println("Received: " + message + "\n");    host = receivePacket.getAddress();  receivePort = receivePacket.getPort();    }catch(Exception e){  System.out.println(e);  }    return message + "\n";  }    public boolean sendMessage(String message){  try{  byte[] data = message.getBytes("UTF-8");  System.out.println("Sending " + message + " to " + host.getAddress().toString() + " on port " + receivePort);  DatagramPacket sendPacket = new DatagramPacket (data, data.length, host, receivePort);    socket.send(sendPacket);  }catch(Exception e){  System.out.println(e);  return false;  }  return true;  }  }  /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package lab5;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.InetAddress;  import java.text.DateFormat;  import java.text.SimpleDateFormat;  import java.util.Date;  import javax.swing.JOptionPane;  import javax.swing.JScrollPane;  /\*\*  \*  \* @author Jeremy  \*/  public class StingRayGUI extends javax.swing.JFrame {    private static int PACKETSIZE = 100;  private static int PORT = 9876;  private static int BYTE\_LENGTH = 64;    private static InetAddress host;  private static int receivePort;  private static DatagramSocket socket;  private static String sStatusDialog;  private static Control control;    /\*\*  \* Creates new form StringRayGUI  \*/  public StingRayGUI() {  initComponents();  tStatus.setEditable(false);  this.setVisible(true);  disableAll();  bStart.setEnabled(true);  }    private void disableAll(){  tDistance.setEnabled(false);  tAngle.setEnabled(false);  bMoveForward.setEnabled(false);  bMoveBackward.setEnabled(false);  bRotateRight.setEnabled(false);  bRotateLeft.setEnabled(false);  bExit.setEnabled(false);  bTemperature.setEnabled(false);  bSonar.setEnabled(false);  bStart.setEnabled(false);  }    private void enableAll(){  tDistance.setEnabled(true);  tAngle.setEnabled(true);  bMoveForward.setEnabled(true);  bMoveBackward.setEnabled(true);  bRotateRight.setEnabled(true);  bRotateLeft.setEnabled(true);  bExit.setEnabled(true);  bTemperature.setEnabled(true);  bSonar.setEnabled(true);  bStart.setEnabled(false);  }  /\*\*  \* This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.  \*/  @SuppressWarnings("unchecked")  // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  private void initComponents() {  jPanel1 = new javax.swing.JPanel();  bMoveForward = new javax.swing.JButton();  bMoveBackward = new javax.swing.JButton();  jLabel1 = new javax.swing.JLabel();  tAngle = new javax.swing.JTextField();  bRotateRight = new javax.swing.JButton();  bRotateLeft = new javax.swing.JButton();  jLabel2 = new javax.swing.JLabel();  tDistance = new javax.swing.JTextField();  bTemperature = new javax.swing.JButton();  bSonar = new javax.swing.JButton();  bExit = new javax.swing.JButton();  jScrollPane2 = new javax.swing.JScrollPane();  tStatus = new javax.swing.JTextArea();  bStart = new javax.swing.JButton();  setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("StingRay"));  bMoveForward.setText("MoveForward");  bMoveForward.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bMoveForwardActionPerformed(evt);  }  });  bMoveBackward.setText("MoveBackward");  bMoveBackward.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bMoveBackwardActionPerformed(evt);  }  });  jLabel1.setText("Rotate Angle");  tAngle.addKeyListener(new java.awt.event.KeyAdapter() {  public void keyPressed(java.awt.event.KeyEvent evt) {  tAngleKeyPressed(evt);  }  });  bRotateRight.setText("Rotate Clockwise");  bRotateRight.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bRotateRightActionPerformed(evt);  }  });  bRotateLeft.setText("Rotate Counter-Clockwise");  bRotateLeft.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bRotateLeftActionPerformed(evt);  }  });  jLabel2.setText("Move Distance");  tDistance.addKeyListener(new java.awt.event.KeyAdapter() {  public void keyPressed(java.awt.event.KeyEvent evt) {  tDistanceKeyPressed(evt);  }  public void keyReleased(java.awt.event.KeyEvent evt) {  tDistanceKeyReleased(evt);  }  public void keyTyped(java.awt.event.KeyEvent evt) {  tDistanceKeyTyped(evt);  }  });  bTemperature.setText("Temperature");  bTemperature.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bTemperatureActionPerformed(evt);  }  });  bSonar.setText("Distance to Obstacle");  bSonar.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bSonarActionPerformed(evt);  }  });  bExit.setText("Exit");  bExit.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bExitActionPerformed(evt);  }  });  tStatus.setColumns(20);  tStatus.setRows(5);  jScrollPane2.setViewportView(tStatus);  bStart.setText("Start");  bStart.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  bStartActionPerformed(evt);  }  });  javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  jPanel1.setLayout(jPanel1Layout);  jPanel1Layout.setHorizontalGroup(  jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGroup(jPanel1Layout.createSequentialGroup()  .addComponent(bRotateRight)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 116, Short.MAX\_VALUE)  .addComponent(bRotateLeft))  .addGroup(jPanel1Layout.createSequentialGroup()  .addComponent(bMoveForward)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addComponent(bMoveBackward))  .addGroup(jPanel1Layout.createSequentialGroup()  .addContainerGap()  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGroup(jPanel1Layout.createSequentialGroup()  .addComponent(jLabel1)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  .addComponent(tAngle))  .addGroup(jPanel1Layout.createSequentialGroup()  .addComponent(jLabel2)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(tDistance))))  .addGroup(jPanel1Layout.createSequentialGroup()  .addComponent(bTemperature)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addComponent(bSonar))  .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()  .addComponent(bStart)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addComponent(bExit))  .addComponent(jScrollPane2)  );  jPanel1Layout.setVerticalGroup(  jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGroup(jPanel1Layout.createSequentialGroup()  .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(jLabel2)  .addComponent(tDistance, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(bMoveForward)  .addComponent(bMoveBackward))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(jLabel1)  .addComponent(tAngle, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(bRotateRight)  .addComponent(bRotateLeft))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(bTemperature)  .addComponent(bSonar))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 15, Short.MAX\_VALUE)  .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(bExit)  .addComponent(bStart)))  );  javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  getContentPane().setLayout(layout);  layout.setHorizontalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  );  layout.setVerticalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  );  pack();  }// </editor-fold>//GEN-END:initComponents  private void bMoveForwardActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bMoveForwardActionPerformed  if(tDistance.getText().length() == 0){  JOptionPane.showMessageDialog(null, "Enter a distance between 1 and 20");  }else if (Integer.parseInt(tDistance.getText()) < 0 || Integer.parseInt(tDistance.getText()) > 20){  JOptionPane.showMessageDialog(null, "Enter a distance between 1 and 20");  }else{  disableAll();  control.sendMessage("1");  displayMessage("Sending Instruction: Move Forward \n");  displayMessage("Waiting For Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  control.sendMessage(tDistance.getText());  displayMessage("Sending Distance: " + tDistance.getText() + " \n");  displayMessage("Waiting For Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  enableAll();  }  });  }  });  }  });  }  }//GEN-LAST:event\_bMoveForwardActionPerformed    private void bStartActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bStartActionPerformed  displayMessage("Waiting for Robot Greeting... \n");  bStart.setEnabled(false);  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForRemoteGreeting());  enableAll();  }  });  }//GEN-LAST:event\_bStartActionPerformed  private void displayMessage(String message){  DateFormat df = new SimpleDateFormat("HH:mm:ss");  Date date = new Date();  sStatusDialog += "[" + df.format(date) + "]: " + message;  tStatus.setText(sStatusDialog);  tStatus.repaint();  }    private void bTemperatureActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bTemperatureActionPerformed  disableAll();  control.sendMessage("6");  displayMessage("Sending Message: Get Temperature \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage("Temperature Received: " + control.waitForMessage());  enableAll();  }  });  }//GEN-LAST:event\_bTemperatureActionPerformed  private void bSonarActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bSonarActionPerformed  disableAll();  control.sendMessage("5");  displayMessage("Sending Message: Get Distance to Obstacle \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage("Distance Recieved: " + control.waitForMessage());  enableAll();  }  });  }//GEN-LAST:event\_bSonarActionPerformed  private void tDistanceKeyTyped(java.awt.event.KeyEvent evt) {//GEN-FIRST:event\_tDistanceKeyTyped  }//GEN-LAST:event\_tDistanceKeyTyped  private void tDistanceKeyReleased(java.awt.event.KeyEvent evt) {//GEN-FIRST:event\_tDistanceKeyReleased  }//GEN-LAST:event\_tDistanceKeyReleased  private void tDistanceKeyPressed(java.awt.event.KeyEvent evt) {//GEN-FIRST:event\_tDistanceKeyPressed  if(evt.getKeyCode() < 48 || evt.getKeyCode() > 57){  JOptionPane.showMessageDialog(null, "Please enter only digits ");  evt.consume();  }  }//GEN-LAST:event\_tDistanceKeyPressed  private void bMoveBackwardActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bMoveBackwardActionPerformed  if(tDistance.getText().length() == 0){  JOptionPane.showMessageDialog(null, "Enter a distance between 1 and 20");  }else if (Integer.parseInt(tDistance.getText()) < 0 || Integer.parseInt(tDistance.getText()) > 20){  JOptionPane.showMessageDialog(null, "Enter a distance between 1 and 20");  }else{  disableAll();  control.sendMessage("2");  displayMessage("Sending Instruction: Move Backward \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage("Waiting For Acknowledgement " + control.waitForMessage());  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  control.sendMessage(tDistance.getText());  displayMessage("Sending Distance: " + tDistance.getText() + " \n");  displayMessage("Waiting For Acknowledgement " + control.waitForMessage());  enableAll();  }  });  }  });  }  }//GEN-LAST:event\_bMoveBackwardActionPerformed  private void tAngleKeyPressed(java.awt.event.KeyEvent evt) {//GEN-FIRST:event\_tAngleKeyPressed  if(evt.getKeyCode() < 48 || evt.getKeyCode() > 57){  JOptionPane.showMessageDialog(null, "Please enter only digits ");  evt.consume();  }  }//GEN-LAST:event\_tAngleKeyPressed  private void bRotateRightActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bRotateRightActionPerformed  if(tAngle.getText().length() == 0){  JOptionPane.showMessageDialog(null, "Enter an angle between 1 and 359");  }else if (Integer.parseInt(tAngle.getText()) < 0 || Integer.parseInt(tAngle.getText()) > 360){  JOptionPane.showMessageDialog(null, "Enter an angle between 1 and 359");  }else{  disableAll();  control.sendMessage("3");  displayMessage("Sending Instruction: Rotate Clockwise \n");  displayMessage("Waiting For Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  control.sendMessage(tAngle.getText());  displayMessage("Sending Angle: " + tAngle.getText() + " \n");  displayMessage("Waiting For Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  enableAll();  }  });  }  });  }  });  }  }//GEN-LAST:event\_bRotateRightActionPerformed  private void bRotateLeftActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bRotateLeftActionPerformed  if(tAngle.getText().length() == 0){  JOptionPane.showMessageDialog(null, "Enter an angle between 1 and 359");  }else if (Integer.parseInt(tAngle.getText()) < 0 || Integer.parseInt(tAngle.getText()) > 360){  JOptionPane.showMessageDialog(null, "Enter an angle between 1 and 359");  }else{  disableAll();  control.sendMessage("4");  displayMessage("Sending Instruction: Rotate Clockwise \n");  displayMessage("Waiting For Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  control.sendMessage(tAngle.getText());  displayMessage("Sending Angle: " + tAngle.getText() + " \n");  displayMessage("Waiting For Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  enableAll();  }  });  }  });  }  });  }  }//GEN-LAST:event\_bRotateLeftActionPerformed  private void bExitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_bExitActionPerformed  disableAll();  control.sendMessage("7");  displayMessage("Sending Instruction: Shutdown \n");  displayMessage("Waiting for Acknowledgement: \n");  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  displayMessage(control.waitForMessage());  bStart.setEnabled(true);  }  });  }//GEN-LAST:event\_bExitActionPerformed  /\*\*  \* @param args the command line arguments  \*/  public static void main(String args[]) {  /\* Set the Nimbus look and feel \*/  //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html  \*/  try {  for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  if ("Nimbus".equals(info.getName())) {  javax.swing.UIManager.setLookAndFeel(info.getClassName());  break;  }  }  } catch (ClassNotFoundException ex) {  java.util.logging.Logger.getLogger(StingRayGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (InstantiationException ex) {  java.util.logging.Logger.getLogger(StingRayGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (IllegalAccessException ex) {  java.util.logging.Logger.getLogger(StingRayGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (javax.swing.UnsupportedLookAndFeelException ex) {  java.util.logging.Logger.getLogger(StingRayGUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  }  //</editor-fold>  //</editor-fold>  /\* Create and display the form \*/  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  sStatusDialog = "";    control = new Control();  StingRayGUI srg = new StingRayGUI();    }  });  }    // Variables declaration - do not modify//GEN-BEGIN:variables  private javax.swing.JButton bExit;  private javax.swing.JButton bMoveBackward;  private javax.swing.JButton bMoveForward;  private javax.swing.JButton bRotateLeft;  private javax.swing.JButton bRotateRight;  private javax.swing.JButton bSonar;  private javax.swing.JButton bStart;  private javax.swing.JButton bTemperature;  private javax.swing.JLabel jLabel1;  private javax.swing.JLabel jLabel2;  private javax.swing.JPanel jPanel1;  private javax.swing.JScrollPane jScrollPane2;  private javax.swing.JTextField tAngle;  private javax.swing.JTextField tDistance;  private javax.swing.JTextArea tStatus;  // End of variables declaration//GEN-END:variables  } |
|  |

## Arduino Source code

#include <Wirefree.h>

#include <WifiClient.h>

#include <SoftwareSerial.h>

#include <Servo.h>

#include <Wire.h>

//Define constants

#define TEMPSENSOR 0x68

#define SONAR\_SENSOR 22

#define BOARD\_LED 13

#define LCD\_PIN 18

#define LEFT\_MOTOR 45

#define RIGHT\_MOTOR 8

#define RIGHT\_SENSOR 49

#define LEFT\_SENSOR 48

#define CENTER\_MOTOR 7

SoftwareSerial LCD(0, LCD\_PIN);

int reg = 0x01; // Ambient temperature = 0x01

WIFI\_PROFILE wireless\_prof = {

"Elwe",

"asdfghjkl",

"192.168.43.26",

"255.255.255.0",

"192.168.43.1",

};

String remote\_server = "192.168.43.90";

String remote\_port = "9876";

WifiClient client(remote\_server, remote\_port, PROTO\_UDP);

void setup()

{

Serial.begin(9600);

Wire.begin(); // Join I2C bus

pinMode(RIGHT\_SENSOR, INPUT);

pinMode(LEFT\_SENSOR, INPUT);

pinMode(SONAR\_SENSOR, OUTPUT);

pinMode(BOARD\_LED, OUTPUT);

pinMode(LCD\_PIN, OUTPUT);

pinMode(CENTER\_MOTOR, OUTPUT);

LCD.begin(9600);

Servo Center ;

Center.attach(CENTER\_MOTOR);

Center.write(82);

// Serial.println("test");

Wireless.begin(&wireless\_prof);

clearScreen();

if (client.connect()) {

Serial.println("connected");

// Send message over UDP socket to peer device

client.print("Hello server!");

//Serial.println("Waiting for command...");

}

else {

// if connection setup failed:

Serial.println("failed");

}

displayStudentNumbers();

}

void loop()

{

// if there are incoming bytes available

// from the peer device, read them and print them:

int instruction;

do{

Serial.println("Waiting for Instruction");

instruction = readInstruction();

Serial.println("Conducting Action");

conductAction(instruction);

}while(instruction != 7);

if(instruction == 7){

Serial.println("disconnecting");

client.stop();

while(1);

}

}

void conductAction(int instruction){

String temp;

int sonar;

String message;

switch(instruction) {

case 1:

Serial.println("Waiting For Distance");

message = getMessage();

Serial.println("Received Distance");

Serial.println("Moving Forward " + message);

moveForward(message.toInt());

break;

case 2:

Serial.println("Waiting For Distance");

message = getMessage();

Serial.println("Received Distance");

Serial.println("Moving Backward " + message);

moveBackward(message.toInt());

break;

case 3:

Serial.println("Waiting For Angle");

message = getMessage();

Serial.println("Received Angle");

Serial.println("Rotating Clockwise " + message);

rotateRight(message.toInt());

break;

case 4:

Serial.println("Waiting For Angle");

message = getMessage();

Serial.println("Received Angle");

Serial.println("Rotating Counter Clockwise " + message);

rotateLeft(message.toInt());

break;

case 5:

sonar = getSonarDistance();

Serial.println("Sending Sonar Distance "+ String(sonar));

client.print(String(sonar));

Serial.println("Message Sent");

break;

case 6:

temp = getTemperature();

Serial.println("Sending Temperature " + temp);

client.print(temp);

Serial.println("Message Sent");

break;

case 7:

Serial.write("Recieved Instruction: Exit");

clearScreen();

delay(100);

printMessage("Disconnected", 1, 0);

break;

default:

break;

}

}

int readInstruction(){

String in;

int instruction;

do {

in = getMessage();

instruction = in.toInt();

switch(instruction) {

case 1:

Serial.write("Recieved Instruction: Move Forward\n");

break;

case 2:

Serial.write("Recieved Instruction: Move Backward\n");

break;

case 3:

Serial.write("Recieved Instruction: Rotate Clockwise\n");

break;

case 4:

Serial.write("Recieved Instruction: Rotate Counter Clockwise\n");

break;

case 5:

Serial.write("Recieved Instruction: Get Distance To Nearest Object\n");

break;

case 6:

Serial.write("Recieved Instruction: Get Temperature\n");

break;

case 7:

Serial.write("Recieved Instruction: Exit");

break;

default:

break;

}

}

while (instruction < 1 || instruction > 7);

return instruction;

}

/\*

Name: getDistance

Description:receives a wireless signal and returns the distance in centimeters of what it receives

Parameters: none

Return: distance in centimeters of the signal sent

\*/

int getDistance()

{

char in;

do{

delay(100);

}

while(! client.available());

in = client.read();

int distance;

distance = in - '0' ;

distance = distance \* 10;

in = client.read();

distance = distance + (in-'0');

return distance;

}

/\*

Name: getAngle

Description:receives a wireless signal and returns the angle in degrees of what it receives

Parameters: none

Return: angle in degrees of the signa sent

\*/

int getAngle()

{

char in;

do{

delay(100);

}

while(! client.available());

Serial.println("angle");

in = client.read();

Serial.print(in);

int distance2;

int distance;

distance = in - '0' ;

distance = distance \* 100;

in = client.read();

Serial.print(in);

distance2 = client.read() - '0';

distance2 = distance2 \* 10;

do{

delay(100);

}

while(! client.available());

in = client.read();

Serial.print(in);

distance = distance + distance2 + (in-'0');

return distance;

}

/\*

Name: getSonarDistance

Description: find the distance to how close the robot is to objects in fron of it

Parameters: none

Return: the distance away from any obstruction that the sonar has found

\*/

int getSonarDistance(){

pinMode(SONAR\_SENSOR, OUTPUT);

digitalWrite(SONAR\_SENSOR, LOW);

delay(2);

// Write a HIGH value to sonar

digitalWrite(SONAR\_SENSOR, HIGH);

delay(5);

digitalWrite(SONAR\_SENSOR, LOW);

pinMode(SONAR\_SENSOR, INPUT);

unsigned long duration = pulseIn(SONAR\_SENSOR, HIGH);

return duration / (29 \* 2) ;

}

/\*

Name: rotateLeft

Description: rotate the robot by x degress

Parameters: input is the degrees of the rotation

Return: none

\*/

void rotateRight(int x){

clearScreen();

String d = "Right ";

d.concat(x);

printMessage("Rotating", 4, 0);

printMessage(d, 4, 1);

flashLED(3);

Servo Left\_Motor, Right\_Motor;

int left\_sensor, right\_sensor, old\_left\_sensor, old\_right\_sensor, left\_count, right\_count, count;

boolean right\_active, left\_active;

delay(1000);

left\_count = 0;

right\_count = 0;

left\_active = false;

right\_active = false;

old\_left\_sensor = digitalRead(LEFT\_SENSOR);

old\_right\_sensor = digitalRead(RIGHT\_SENSOR);

Right\_Motor.write(120);

Left\_Motor.write(120);

while ((left\_count <= round((x \* 53)/100)) || (right\_count <= round((x \* 53)/100))){

//Serial.print(left\_count);

//Serial.print(" ");

//Serial.println(right\_count);

left\_sensor = digitalRead(LEFT\_SENSOR);

right\_sensor = digitalRead(RIGHT\_SENSOR);

if(old\_left\_sensor != left\_sensor){

left\_count++;

}

if(old\_right\_sensor != right\_sensor){

right\_count++;

}

old\_left\_sensor = left\_sensor;

old\_right\_sensor = right\_sensor;

if (left\_active){

if(left\_count > right\_count){

Left\_Motor.detach();

left\_active = false;

}

}

else{

if(! (left\_count > right\_count)){

Left\_Motor.attach(LEFT\_MOTOR);

Left\_Motor.write(120);

left\_active = true;

}

}

if (right\_active){

if(right\_count>left\_count){

Right\_Motor.detach();

right\_active = false;

}

}

else{

if(!(right\_count > left\_count)){

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(120);

right\_active = true;

}

}

delay(1);

}

Right\_Motor.detach();

Left\_Motor.detach();

delay(1000);

}

/\*

Name: rotateLeft

Description: rotate the robot by x degress

Parameters: input is the degrees of the rotation

Return: none

\*/

void rotateLeft(int x){

clearScreen();

String d = "Left ";

d.concat(x);

printMessage("Rotating", 4, 0);

printMessage(d, 4, 1);

flashLED(3);

Servo Left\_Motor, Right\_Motor;

int left\_sensor, right\_sensor, old\_left\_sensor, old\_right\_sensor, left\_count, right\_count, count;

boolean right\_active, left\_active;

delay(1000);

left\_count = 0;

right\_count = 0;

left\_active = false;

right\_active = false;

old\_left\_sensor = digitalRead(LEFT\_SENSOR);

old\_right\_sensor = digitalRead(RIGHT\_SENSOR);

Right\_Motor.write(0);

Left\_Motor.write(0);

while ((left\_count <= round((x \* 53)/100)) || (right\_count <= round((x \* 53)/100))){

//Serial.print(left\_count);

//Serial.print(" ");

//Serial.println(right\_count);

left\_sensor = digitalRead(LEFT\_SENSOR);

right\_sensor = digitalRead(RIGHT\_SENSOR);

if(old\_left\_sensor != left\_sensor){

left\_count++;

}

if(old\_right\_sensor != right\_sensor){

right\_count++;

}

old\_left\_sensor = left\_sensor;

old\_right\_sensor = right\_sensor;

if (left\_active){

if(left\_count > right\_count){

Left\_Motor.detach();

left\_active = false;

}

}

else{

if(! (left\_count > right\_count)){

Left\_Motor.attach(LEFT\_MOTOR);

Left\_Motor.write(0);

left\_active = true;

}

}

if (right\_active){

if(right\_count>left\_count){

Right\_Motor.detach();

right\_active = false;

}

}

else{

if(!(right\_count > left\_count)){

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(0);

right\_active = true;

}

}

delay(1);

}

Right\_Motor.detach();

Left\_Motor.detach();

delay(1000);

}

/\*

Name: movediagonal

Description: move the robot Dagonal across a certain number of tiles diagonal lengths

Parameters: input is number of tiles to traverse

Return: none

\*/

void moveDiagonal(int x){

Servo Left\_Motor, Right\_Motor;

int left\_sensor, right\_sensor, old\_left\_sensor, old\_right\_sensor, left\_count, right\_count, count;

boolean right\_active, left\_active;

delay(1000);

left\_count = 0;

right\_count = 0;

left\_active = false;

right\_active = false;

old\_left\_sensor = digitalRead(LEFT\_SENSOR);

old\_right\_sensor = digitalRead(RIGHT\_SENSOR);

Right\_Motor.write(0);

Left\_Motor.write(90);

while ((left\_count <= x\*160) || (right\_count <= x\*160)){

//Serial.print(left\_count);

//Serial.print(" ");

//Serial.println(right\_count);

left\_sensor = digitalRead(LEFT\_SENSOR);

right\_sensor = digitalRead(RIGHT\_SENSOR);

if(old\_left\_sensor != left\_sensor){

left\_count++;

}

if(old\_right\_sensor != right\_sensor){

right\_count++;

}

old\_left\_sensor = left\_sensor;

old\_right\_sensor = right\_sensor;

if (left\_active){

if(left\_count > right\_count){

Left\_Motor.detach();

left\_active = false;

}

}

else{

if(! (left\_count > right\_count)){

Left\_Motor.attach(LEFT\_MOTOR);

Left\_Motor.write(90);

left\_active = true;

}

}

if (right\_active){

if(right\_count>left\_count){

Right\_Motor.detach();

right\_active = false;

}

}

else{

if(!(right\_count > left\_count)){

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(0);

right\_active = true;

}

}

delay(1);

}

Right\_Motor.detach();

Left\_Motor.detach();

delay(1000);

}

/\*

Name: moveforward

Description: move the robot forward across a certain number of tiles

Parameters: input is number of tiles to traverse

Return: none

\*/

void moveForward(int x){

clearScreen();

String d="Forward ";

d.concat(x);

printMessage("Moving", 5, 0);

printMessage(d, 4, 1);

flashLED(3);

Servo Left\_Motor, Right\_Motor;

int left\_sensor, right\_sensor, old\_left\_sensor, old\_right\_sensor, left\_count, right\_count, count;

boolean right\_active, left\_active;

delay(1000);

left\_count = 0;

right\_count = 0;

left\_active = false;

right\_active = false;

old\_left\_sensor = digitalRead(LEFT\_SENSOR);

old\_right\_sensor = digitalRead(RIGHT\_SENSOR);

Right\_Motor.write(0);

Left\_Motor.write(90);

while ((left\_count <= x\*4) || (right\_count <= x\*4)){

//Serial.print(left\_count);

//Serial.print(" ");

//Serial.println(right\_count);

left\_sensor = digitalRead(LEFT\_SENSOR);

right\_sensor = digitalRead(RIGHT\_SENSOR);

if(old\_left\_sensor != left\_sensor){

left\_count++;

}

if(old\_right\_sensor != right\_sensor){

right\_count++;

}

old\_left\_sensor = left\_sensor;

old\_right\_sensor = right\_sensor;

if (left\_active){

if(left\_count > right\_count){

Left\_Motor.detach();

left\_active = false;

}

}

else{

if(! (left\_count > right\_count)){

Left\_Motor.attach(LEFT\_MOTOR);

Left\_Motor.write(90);

left\_active = true;

}

}

if (right\_active){

if(right\_count>left\_count){

Right\_Motor.detach();

right\_active = false;

}

}

else{

if(!(right\_count > left\_count)){

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(0);

right\_active = true;

}

}

delay(1);

}

Right\_Motor.detach();

Left\_Motor.detach();

delay(1000);

}

/\*

Name: diagonal

Description: move the robot Dagonal across a certain number of tiles diagonal lengths

Parameters: input is number of tiles to traverse

Return: none

\*/

void diagonal(int x)

{

clearScreen();

String d= "Diagonal ";

d.concat(x);

printMessage("Moving", 5, 0);

printMessage(d, 3, 1);

flashLED(3);

Servo Right\_Motor;

Servo Left\_Motor;

Left\_Motor.attach(LEFT\_MOTOR);

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(0);

Left\_Motor.write(90);

int count = 0;

int new\_sensor;

int old\_sensor = digitalRead(LEFT\_SENSOR);

//int old\_sensor = digitalRead(RIGHT\_SENSOR);

for(int i = 0; i<1950 \* x; i++)

{

new\_sensor = digitalRead(LEFT\_SENSOR);

//new\_sensor = digitalRead(RIGHT\_SENSOR);

//Serial.print(new\_sensor);

if (new\_sensor != old\_sensor){

count++;

}

old\_sensor = new\_sensor;

delay(1);

}

//Serial.print("\n");

Serial.println(count);

Left\_Motor.detach();

Right\_Motor.detach();

delay(2000);

}

/\*

Name: movebackward

Description: move the robot backards across a certain number of tiles

Parameters: input is number of tiles to traverse

Return: none

\*/

void moveBackward(int x){

clearScreen();

String d="Backward ";

d.concat(x);

printMessage("Moving", 5, 0);

printMessage(d, 4, 1);

flashLED(3);

Servo Left\_Motor, Right\_Motor;

int left\_sensor, right\_sensor, old\_left\_sensor, old\_right\_sensor, left\_count, right\_count, count;

boolean right\_active, left\_active;

delay(1000);

left\_count = 0;

right\_count = 0;

left\_active = false;

right\_active = false;

old\_left\_sensor = digitalRead(LEFT\_SENSOR);

old\_right\_sensor = digitalRead(RIGHT\_SENSOR);

Right\_Motor.write(90);

Left\_Motor.write(0);

while ((left\_count <= x\*4) || (right\_count <= x\*4)){

//Serial.print(left\_count);

//Serial.print(" ");

//Serial.println(right\_count);

left\_sensor = digitalRead(LEFT\_SENSOR);

right\_sensor = digitalRead(RIGHT\_SENSOR);

if(old\_left\_sensor != left\_sensor){

left\_count++;

}

if(old\_right\_sensor != right\_sensor){

right\_count++;

}

old\_left\_sensor = left\_sensor;

old\_right\_sensor = right\_sensor;

if (left\_active){

if(left\_count > right\_count){

Left\_Motor.detach();

left\_active = false;

}

}

else{

if(! (left\_count > right\_count)){

Left\_Motor.attach(LEFT\_MOTOR);

Left\_Motor.write(0);

left\_active = true;

}

}

if (right\_active){

if(right\_count>left\_count){

Right\_Motor.detach();

right\_active = false;

}

}

else{

if(!(right\_count > left\_count)){

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(90);

right\_active = true;

}

}

delay(1);

}

Right\_Motor.detach();

Left\_Motor.detach();

delay(1000);

}

/\*

Name: forward

Description: move the robot forward across a certain number of tiles

Parameters: input is number of tiles to traverse

Return: none

\*/

void forward(int x)

{

clearScreen();

String d="Forward ";

d.concat(x);

printMessage("Moving", 5, 0);

printMessage(d, 4, 1);

flashLED(3);

Servo Right\_Motor;

Servo Left\_Motor;

Left\_Motor.attach(LEFT\_MOTOR);

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(0);

Left\_Motor.write(90);

int count = 0;

int new\_sensor;

int old\_sensor = digitalRead(LEFT\_SENSOR);

//int old\_sensor = digitalRead(RIGHT\_SENSOR);

for(int i = 0; i<1375 \*x; i++)

{

new\_sensor = digitalRead(LEFT\_SENSOR);

//new\_sensor = digitalRead(RIGHT\_SENSOR);

//Serial.print(new\_sensor);

if (new\_sensor != old\_sensor){

count++;

}

old\_sensor = new\_sensor;

delay(1);

}

//Serial.print("\n");

Serial.println(count);

Left\_Motor.detach();

Right\_Motor.detach();

delay(2000);

}

void reversediagonal()

{

Servo Right\_Motor;

Servo Left\_Motor;

Left\_Motor.attach(LEFT\_MOTOR);

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(120);

Left\_Motor.write(0);

int count = 0;

int new\_sensor;

int old\_sensor = digitalRead(LEFT\_SENSOR);

//int old\_sensor = digitalRead(RIGHT\_SENSOR);

for(int i = 0; i<1925; i++)

{

new\_sensor = digitalRead(LEFT\_SENSOR);

//new\_sensor = digitalRead(RIGHT\_SENSOR);

//Serial.print(new\_sensor);

if (new\_sensor != old\_sensor){

count++;

}

old\_sensor = new\_sensor;

delay(1);

}

//Serial.print("\n");

Serial.println(count);

Left\_Motor.detach();

Right\_Motor.detach();

delay(2000);

}

void reverseforward()

{

Servo Right\_Motor;

Servo Left\_Motor;

Left\_Motor.attach(LEFT\_MOTOR);

Right\_Motor.attach(RIGHT\_MOTOR);

Right\_Motor.write(120);

Left\_Motor.write(0);

int count = 0;

int new\_sensor;

int old\_sensor = digitalRead(LEFT\_SENSOR);

//int old\_sensor = digitalRead(RIGHT\_SENSOR);

for(int i = 0; i<1325; i++)

{

new\_sensor = digitalRead(LEFT\_SENSOR);

//new\_sensor = digitalRead(RIGHT\_SENSOR);

//Serial.print(new\_sensor);

if (new\_sensor != old\_sensor){

count++;

}

old\_sensor = new\_sensor;

delay(1);

}

//Serial.print("\n");

Serial.println(count);

Left\_Motor.detach();

Right\_Motor.detach();

delay(2000);

}

/\*

Name: printStopMessage

Description: Prints a stop message on the LCD display

Parameters: none

Return: none

\*/

void printStopMessage(){

clearScreen();

printMessage("Stopped", 5, 0);

flashLED(3);

delay(3000);

}

/\*

Name: printMessage

Description: Prints a given message to the LCD display

Parameters:

text - String - In - the message to print

col - int - In - the column of the display to start the message at

row - int - In - the row of the display used to display the message

Return: none

\*/

void printMessage(String text, int col, int row){

LCD.write(0xFE);

LCD.write(col + row \* 64 + 128);

LCD.print(text);

}

/\*

Name: flashLED

Description: Instructions to blink the LED light for a certain period of time

Parameters: x - int - In - the number of seconds to blink the LED

Return: none

\*/

void flashLED(int x) {

for (int a = x; a > 0; a--){

digitalWrite(BOARD\_LED, HIGH);

delay(500);

digitalWrite(BOARD\_LED, LOW);

delay(500);

}

}

/\*

Name: displayStudentNumbers

Description: Instructions to display the student numbers on the LCD screen

Parameters: none

Return: none

\*/

void displayStudentNumbers(){

clearScreen();

printMessage("3116318", 5, 0);

printMessage("6146324", 5, 1);

flashLED(5);

}

/\*

Name: displayTemperature

Description: Instructions to display the scurrent room temperature on the LCD screen

Parameters: none

Return: none

\*/

String getTemperature(){

String temp="";

Wire.beginTransmission(TEMPSENSOR);

Wire.write(reg); // Indicate temperature value to read

Wire.endTransmission();

Wire.requestFrom(TEMPSENSOR, 1); // Request data

while(Wire.available() < 1); // Wait for data

byte temperatureData = Wire.read(); // Temp. value

clearScreen();

printMessage("Temperature", 2, 0);

printMessage(String(temperatureData) + " Degrees", 2, 1);

flashLED(5);

temp = String(temperatureData);

return temp;

}

/\*

Name: clearScreen

Description: Instructions to clear the LCD display screen

Parameters: none

Return: none

\*/

void clearScreen (){

LCD.write(0xFE);

LCD.write(0x01);

delay(10);

}

String getMessage(){

String message;

message = "";

int in;

do{

while (client.available()){

in = client.read();

if(in != 255 && in != -1){

message += (char) in;

}

}

}while(message=="");

return message;

}