```java
package LFS_IMU_Server_wGUI;

/**
 *
 * @author Ivan
 */
public class GUI extends javax.swing.JFrame {

	private Thread t;
	private UDP_Server udp_server;

	/**
	 * Creates new form GUI
	 */
	public GUI() {
		initComponents();
	}

	/**
	 * 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">
	private void initComponents() {

		jTextField12 = new javax.swing.JTextField();
		UDP_Port = new javax.swing.JTextField();
		jTextField13 = new javax.swing.JTextField();
		PLC_IP_address = new javax.swing.JTextField();
		Blender_IP_address = new javax.swing.JTextField();
		jTextField14 = new javax.swing.JTextField();
		blender_port = new javax.swing.JTextField();
		jTextField5 = new javax.swing.JTextField();
		Run_UDP_Server = new javax.swing.JButton();
		Stop_UDP_Server = new javax.swing.JButton();
		selected_LFS = new javax.swing.JCheckBox();
		selected_IMU = new javax.swing.JCheckBox();
		selecSendToPLC = new javax.swing.JCheckBox();
		selectSendToBlender = new javax.swing.JCheckBox();

		setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
		setTitle("UDP server");

		jTextField12.setEditable(false);
		jTextField12.setText("Blender IP address");
		jTextField12.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent evt) {
				jTextField12ActionPerformed(evt);
			}
		});

		UDP_Port.setText("5555");
		UDP_Port.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent evt) {
				UDP_PortActionPerformed(evt);
			}
		});
```

```
            jTextField13.setEditable(false);
            jTextField13.setText("Blender port");

            PLC_IP_address.setText("158.38.140.113");
            PLC_IP_address.addActionListener(new java.awt.event.ActionListener()
{
                  public void actionPerformed(java.awt.event.ActionEvent evt) {
                        PLC_IP_addressActionPerformed(evt);
                  }
            });

            Blender_IP_address.setText("158.38.140.10");
            Blender_IP_address
                        .addActionListener(new java.awt.event.ActionListener() {
                              public void
actionPerformed(java.awt.event.ActionEvent evt) {
                                    Blender_IP_addressActionPerformed(evt);
                              }
                        });

            jTextField14.setEditable(false);
            jTextField14.setText("PLC IP address");

            blender_port.setText("7501");

            jTextField5.setEditable(false);
            jTextField5.setText("UDP Port");
            jTextField5.addActionListener(new java.awt.event.ActionListener() {
                  public void actionPerformed(java.awt.event.ActionEvent evt) {
                        jTextField5ActionPerformed(evt);
                  }
            });

            Run_UDP_Server.setText("Run");
            Run_UDP_Server.addActionListener(new java.awt.event.ActionListener()
{
                  public void actionPerformed(java.awt.event.ActionEvent evt) {
                        Run_UDP_ServerActionPerformed(evt);
                  }
            });

            Stop_UDP_Server.setText("Stop");
            Stop_UDP_Server.addActionListener(new java.awt.event.ActionListener()
{
                  public void actionPerformed(java.awt.event.ActionEvent evt) {
                        Stop_UDP_ServerActionPerformed(evt);
                  }
            });

            selected_LFS.setText("LFS");
            selected_LFS.addActionListener(new java.awt.event.ActionListener() {
                  public void actionPerformed(java.awt.event.ActionEvent evt) {
                        selected_LFSActionPerformed(evt);
                  }
            });

            selected_IMU.setText("IMU");
```

```
        selecSendToPLC.setSelected(true);
        selecSendToPLC.setText("Send to PLC");

        selectSendToBlender.setSelected(true);
        selectSendToBlender.setText("Send to Blender");

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
                    getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(layout

.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(
                                layout.createSequentialGroup()
                                            .addContainerGap()
                                            .addGroup(

layout.createParallelGroup(

javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(

        layout.createSequentialGroup()

                    .addGroup(

                                layout.createParallelGroup(


javax.swing.GroupLayout.Alignment.LEADING,

                                            false)

                                            .addComponent(

                                                        jTextField12,


javax.swing.GroupLayout.DEFAULT_SIZE,

                                                        120,

                                                        Short.MAX_VALUE)

                                            .addComponent(

                                                        jTextField13)

                                            .addComponent(

                                                        jTextField5)

                                            .addComponent(

                                                        jTextField14))

                    .addPreferredGap(
```

```
javax.swing.LayoutStyle.ComponentPlacement.RELATED,
                                javax.swing.GroupLayout.DEFAULT_SIZE,
                                Short.MAX_VALUE)
                        .addGroup(
                                layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING,
                                                false)
                                                .addComponent(
                                                        UDP_Port,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,
                                                        45,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(
                                                        blender_port,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,
                                                        45,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(
                                                        PLC_IP_address,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE,
                                                        97,
```

```
                                                                Short.MAX_VALUE)
                                                .addComponent(
Blender_IP_address,
javax.swing.GroupLayout.Alignment.TRAILING)))
.addGroup(
      layout.createSequentialGroup()
                  .addGroup(
                              layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
                                                .addGroup(
layout.createSequentialGroup()
.addComponent(
      selected_LFS)
.addPreferredGap(
      javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(
      selected_IMU))
                                                .addGroup(
layout.createSequentialGroup()
.addComponent(
      selecSendToPLC)
.addPreferredGap(
```

```
        javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(

        selectSendToBlender)))
                        .addPreferredGap(

javax.swing.LayoutStyle.ComponentPlacement.RELATED,
                                        12,
                                        Short.MAX_VALUE)
                        .addComponent(
                                        Run_UDP_Server)
                        .addPreferredGap(

javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(
                                        Stop_UDP_Server)))
                                                .addContainerGap()));
        layout.setVerticalGroup(layout

.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(
                                        layout.createSequentialGroup()
                                                .addContainerGap()
                                                .addGroup(

layout.createParallelGroup(

javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(

        jTextField14,

        javax.swing.GroupLayout.PREFERRED_SIZE,

        javax.swing.GroupLayout.DEFAULT_SIZE,

        javax.swing.GroupLayout.PREFERRED_SIZE)

.addComponent(

        PLC_IP_address,

        javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
        javax.swing.GroupLayout.DEFAULT_SIZE,

        javax.swing.GroupLayout.PREFERRED_SIZE))
                                                .addPreferredGap(

javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                .addGroup(

layout.createParallelGroup(

javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(

        UDP_Port,

        javax.swing.GroupLayout.PREFERRED_SIZE,

        javax.swing.GroupLayout.DEFAULT_SIZE,

        javax.swing.GroupLayout.PREFERRED_SIZE)

.addComponent(

        jTextField5,

        javax.swing.GroupLayout.PREFERRED_SIZE,

        javax.swing.GroupLayout.DEFAULT_SIZE,

        javax.swing.GroupLayout.PREFERRED_SIZE))
                                                .addPreferredGap(

javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                .addGroup(

layout.createParallelGroup(

javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(

        jTextField12,

        javax.swing.GroupLayout.PREFERRED_SIZE,

        javax.swing.GroupLayout.DEFAULT_SIZE,

        javax.swing.GroupLayout.PREFERRED_SIZE)

.addComponent(

        Blender_IP_address,

        javax.swing.GroupLayout.PREFERRED_SIZE,

        javax.swing.GroupLayout.DEFAULT_SIZE,
```

```
        javax.swing.GroupLayout.PREFERRED_SIZE))
                                                .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                .addGroup(
layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(
        jTextField13,
        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.DEFAULT_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(
        blender_port,
        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.DEFAULT_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE))
                                                .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                                .addGroup(
layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(
        javax.swing.GroupLayout.Alignment.TRAILING,
        layout.createSequentialGroup()
                        .addGroup(
                                        layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.BASELINE)
                                                .addComponent(
                                                        Run_UDP_Server)
                                                .addComponent(
                                                        Stop_UDP_Server))
```

```
                    .addContainerGap(
                                javax.swing.GroupLayout.DEFAULT_SIZE,
                                Short.MAX_VALUE))
.addGroup(
      layout.createSequentialGroup()
                    .addGroup(
                                layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.BASELINE)
                                            .addComponent(
                                                        selected_LFS)
                                            .addComponent(
                                                        selected_IMU))
                    .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(
                                layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.BASELINE)
                                            .addComponent(
                                                        selecSendToPLC)
                                            .addComponent(
selectSendToBlender))
                    .addContainerGap(
                                javax.swing.GroupLayout.DEFAULT_SIZE,
                                Short.MAX_VALUE)))));
      pack();
}// </editor-fold>

private void jTextField12ActionPerformed(java.awt.event.ActionEvent evt) {
      // TODO add your handling code here:
}
```

```
        private void PLC_IP_addressActionPerformed(java.awt.event.ActionEvent evt) {
                // TODO add your handling code here:
        }

        private void Blender_IP_addressActionPerformed(
                        java.awt.event.ActionEvent evt) {
                // TODO add your handling code here:
        }

        private void jTextField5ActionPerformed(java.awt.event.ActionEvent evt) {
                // TODO add your handling code here:
        }

        private void Run_UDP_ServerActionPerformed(java.awt.event.ActionEvent evt) {
                if ((selected_LFS.isSelected()) ^ (selected_IMU.isSelected())) {
                        udp_server = new UDP_Server(PLC_IP_address.getText(),
                                        UDP_Port.getText(), Blender_IP_address.getText(),
                                        blender_port.getText(), selected_IMU.isSelected(),
                                        selected_LFS.isSelected(),
selecSendToPLC.isSelected(),
                                        selectSendToBlender.isSelected());
                        t = new Thread(udp_server);
                        t.start();
                }
        }

        private void Stop_UDP_ServerActionPerformed(java.awt.event.ActionEvent evt)
{
                if (udp_server.isRunning()) {
                        udp_server.stop();
                }
        }

        private void selected_LFSActionPerformed(java.awt.event.ActionEvent evt) {
                // TODO add your handling code here:
        }

        private void UDP_PortActionPerformed(java.awt.event.ActionEvent evt) {
                // TODO add your handling code here:
        }

        /**
         * @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(GUI.class.getName()).log(
                                java.util.logging.Level.SEVERE, null, ex);
              } catch (InstantiationException ex) {
                    java.util.logging.Logger.getLogger(GUI.class.getName()).log(
                                java.util.logging.Level.SEVERE, null, ex);
              } catch (IllegalAccessException ex) {
                    java.util.logging.Logger.getLogger(GUI.class.getName()).log(
                                java.util.logging.Level.SEVERE, null, ex);
              } catch (javax.swing.UnsupportedLookAndFeelException ex) {
                    java.util.logging.Logger.getLogger(GUI.class.getName()).log(
                                java.util.logging.Level.SEVERE, null, ex);
              }
              // </editor-fold>

              /* Create and display the form */
              java.awt.EventQueue.invokeLater(new Runnable() {
                    @Override
                    public void run() {
                          new GUI().setVisible(true);
                    }
              });
        }

        // Variables declaration - do not modify
        private javax.swing.JTextField Blender_IP_address;
        private javax.swing.JTextField PLC_IP_address;
        private javax.swing.JButton Run_UDP_Server;
        private javax.swing.JButton Stop_UDP_Server;
        private javax.swing.JTextField UDP_Port;
        private javax.swing.JTextField blender_port;
        private javax.swing.JTextField jTextField12;
        private javax.swing.JTextField jTextField13;
        private javax.swing.JTextField jTextField14;
        private javax.swing.JTextField jTextField5;
        private javax.swing.JCheckBox selecSendToPLC;
        private javax.swing.JCheckBox selectSendToBlender;
        private javax.swing.JCheckBox selected_IMU;
        private javax.swing.JCheckBox selected_LFS;
        // End of variables declaration
}

package LFS_IMU_Server_wGUI;

import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
```

```java
/**
 * This class creates a console, and prints the pitch and roll
 *
 * @author Ivan
 */
public class PrintConsole implements Runnable {

	private final JFrame frame;
	private final JTextArea ta;

	public PrintConsole() {
		// setup console frame
		frame = new JFrame();
		frame.add(new JLabel(" Console"), BorderLayout.NORTH);
		ta = new JTextArea();
		frame.add(new JScrollPane(ta));
		Dimension d = new Dimension();
		d.setSize(300, 200);
		frame.setMinimumSize(d);
		frame.pack();
		frame.setVisible(true);
	}

	public void Print(String printPitch, String printRoll) {
		String print = ("Pitch: " + printPitch + " Roll: " + printRoll +
"\n");
		ta.setText(print);
	}

	public void Close() {
		frame.setVisible(false);
	}

	@Override
	public void run() {
		// throw new UnsupportedOperationException("Not supported yet.");
//To
		// change body of generated methods, choose Tools | Templates.
	}
}
```

```java
package LFS_IMU_Server_wGUI;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.logging.Level;
import java.util.logging.Logger;

import net.wimpi.modbus.io.ModbusTCPTransaction;
import net.wimpi.modbus.msg.ReadInputDiscretesRequest;
import net.wimpi.modbus.msg.ReadInputDiscretesResponse;
import net.wimpi.modbus.msg.WriteMultipleRegistersRequest;
import net.wimpi.modbus.net.TCPMasterConnection;
import net.wimpi.modbus.procimg.SimpleRegister;
```

```java
/**
 * This UDP server class receives UDP packets from either LFS (live for speed),
 * or IMU, a mobile app which uses the gyroscope of the mobile to send the roll
 * and pitch of the mobile.
 *
 * @author Ivan
 */
public class UDP_Server implements Runnable {

	private final int port;
	private final int blenderPort;
	private final String PLS_IPaddress;
	private final String blenderIPaddress;
	private Boolean stop;
	private Boolean run;
	private final Thread printThread;
	private final PrintConsole printToConsole;
	private boolean runIMU;
	private boolean runLFS;
	private boolean sendToPLC;
	private boolean sendToBlender;
	DatagramSocket socket;
	private float oldPitch;
	private float oldRoll;

	public UDP_Server(String IPaddr, String port1, String bIPaddr,
			String bPort, boolean imu, boolean lfs, boolean sToPLC,
			boolean sToBlend) {
		PLS_IPaddress = IPaddr;
		port = Integer.parseInt(port1);
		blenderIPaddress = bIPaddr;
		blenderPort = Integer.parseInt(bPort);
		stop = false;
		printToConsole = new PrintConsole();
		printThread = new Thread(printToConsole);
		runIMU = imu;
		runLFS = lfs;
		sendToPLC = sToPLC;
		sendToBlender = sToBlend;
	}

	@Override
	public void run() {
		try {
			run = true;
			printThread.start();
			runUDPserver();
		} catch (Exception ex) {
			Logger.getLogger(UDP_Server.class.getName()).log(Level.SEVERE,
					null, ex);
		}
		throw new UnsupportedOperationException("Not supported yet."); // To

                                        // change

                                        // body

                                        // of
```

```
                                        // generated
                                        // methods,
                                        // choose
                                        // Tools
                                        // |
                                        // Templates.
	}

	private void runUDPserver() throws Exception {

		socket = new DatagramSocket(port);
		while (run) {

			DatagramPacket udpPacket = new DatagramPacket(new byte[1024],
1024);
			// Block until the host receives a UDP packet.
			socket.receive(udpPacket);
			if (runIMU) {
				// Get the relevant data from the IMU UDP packs
				getDataIMU(udpPacket);
			}

			if (runLFS) {
				// Get the relevant data from the LFS UDP packs
				getDataLFS(udpPacket);

			}
		}
	}

	// Stops the server
	public void stop() {
		socket.close();
		printToConsole.Close();
		runIMU = false;
		runLFS = false;
		run = false;
	}

	private void getDataIMU(DatagramPacket request) throws Exception {
		// Obltain references to the packet's array of bytes.
		byte[] buf = request.getData();

		// Wrap the bytes in a byte array input stream
		ByteArrayInputStream bais = new ByteArrayInputStream(buf);

		// Wrap the byte array output stream in an input stream reader
		InputStreamReader isr = new InputStreamReader(bais);

		// Wrap the input stream reader in a buffered reader
		BufferedReader br = new BufferedReader(isr);
		// The message data is contained in a single line
		String line = br.readLine();
```

```
        // The strings in the packets from the IMU are seperated by a ","
        // the string is therefore split into an array by the ","
        String[] stringArray = line.split(",");
        String roll = stringArray[2];
        String pitch = stringArray[3];

        // Filtering the values
        String filter = irrFilter(pitch, roll);
        String[] filterArray = filter.split(",");
        String filterPitch = filterArray[0];
        String filterRoll = filterArray[1];

        // SEND ROLL PITCH AS FLOAT
        float sPitch = Float.parseFloat(filterPitch);
        float sRoll = Float.parseFloat(filterRoll);
        if (sendToBlender) {
              sendDataToBlender();
        }
        if (sendToPLC) {
              sendDataToPLS(sPitch, sRoll);
        }
        printToConsole.Print(pitch, roll);
}

private void getDataLFS(DatagramPacket request) throws Exception {

        // Obtain references to the packet's array of bytes.
        byte[] b = request.getData();

        // Send time of the UDP packet.
        int sendTime = (b[0] & 0xFF) | ((b[1] & 0xFF) << 8)
                    | ((b[2] & 0xFF) << 16) | ((b[3] & 0xFF) << 24);

        // The cars pitch, as an integer.
        int pitchInt = (b[20] & 0xFF) | ((b[21] & 0xFF) << 8)
                    | ((b[22] & 0xFF) << 16) | ((b[23] & 0xFF) << 24);

        // The cars roll, as an integer.
        int rollInt = (b[24] & 0xFF) | ((b[25] & 0xFF) << 8)
                    | ((b[26] & 0xFF) << 16) | ((b[27] & 0xFF) << 24);

        // The cars pitch in its proper type, float.
        float Pitch = Float.intBitsToFloat(pitchInt);

        // The cars roll in its proper type, float.
        float Roll = Float.intBitsToFloat(rollInt);

        float pi = (float) 3.14;

        // Converts radians to degrees
        float sendPitch = Pitch * (180 / pi);
        float sendRoll = Roll * (180 / pi) * (-1);

        if (sendToBlender) {
              sendDataToBlender();
        }
        if (sendToPLC) {
              sendDataToPLS(sendPitch, sendRoll);
```

```
		}
		printToConsole.Print(String.valueOf(sendPitch),
				String.valueOf(sendRoll));
	}

	private void sendDataToPLS(float pitch, float roll) throws Exception {

		try {
			TCPMasterConnection con = null; // the connection
			ModbusTCPTransaction trans = null; // the transaction
			WriteMultipleRegistersRequest req = null;

			// The PLCs address
			InetAddress addr = InetAddress.getByName(PLS_IPaddress);
			int modPort = 502; // default modbus port
			int ref = 12288; // the reference; offset where to start
reading
			// from

			// Open the connection
			con = new TCPMasterConnection(addr);
			con.setPort(modPort);
			con.connect();

			// Creates register to write
			SimpleRegister[] sr = new SimpleRegister[2];
			sr[0] = new SimpleRegister(Math.round(pitch * 1000));
			sr[1] = new SimpleRegister(Math.round(roll * 1000));

			// Prepare the request
			req = new WriteMultipleRegistersRequest(ref, sr);

			// Prepare the transaction
			trans = new ModbusTCPTransaction(con);
			trans.setRequest(req);
			trans.execute();
			con.close();
		} catch (Exception ex) {
			System.out.println(ex);
		}
	}

	private void sendDataToBlender() throws Exception {

		/* The important instances of the classes mentioned before */
		TCPMasterConnection con = null; // the connection
		ModbusTCPTransaction trans = null; // the transaction
		ReadInputDiscretesRequest req = null; // the request
		ReadInputDiscretesResponse res = null; // the response

		InetAddress addr = InetAddress.getByName(PLS_IPaddress); // the

				// slave's

				// address
		int port = 502;
		int ref = 12320; // the reference; offset where to start reading from
		int count = 32; // the number of DI's to read
```

```
            con = new TCPMasterConnection(addr);
            con.setPort(port);
            con.connect();

            req = new ReadInputDiscretesRequest(ref, count);

            trans = new ModbusTCPTransaction(con);
            trans.setRequest(req);

            trans.execute();

            res = (ReadInputDiscretesResponse) trans.getResponse();

            byte[] b = res.getDiscretes().getBytes();

            short pitchShort = (short) ((b[0] & 0xFF) | ((b[1] & 0xFF) << 8));
            short rollShort = (short) ((b[2] & 0xFF) | ((b[3] & 0xFF) << 8));

            // System.out.println("   Pitch: " +pitchShort + "    Roll: " +
            // rollShort);

            // The cars pitch in its proper type, float.
            float pitchFloat = pitchShort / (float) -1000;

            // The cars roll in its proper type, float.
            float rollFloat = rollShort / (float) 1000;

            System.out.println("Pitch: " + pitchFloat + " Roll: " + rollFloat);

            con.close();
            // IP address to the blender computer
            InetAddress IPAddress = InetAddress.getByName(blenderIPaddress);

            // Sends the Pitch & Roll string to blender
            DatagramSocket socket = new DatagramSocket();
            String Message1 = "" + pitchFloat + " " + rollFloat + " 0 ";
            DatagramPacket udpPack = new DatagramPacket(Message1.getBytes(),
                        Message1.length(), IPAddress, blenderPort);
            socket.send(udpPack);
            System.out.println(Message1);
            socket.close();

    }

    // Test to see if the server is running
    public boolean isRunning() {
            return run;
    }

    @Override
    public String toString() {
            StringBuilder sb = new StringBuilder();
            sb.append("PLC IP address: " + PLS_IPaddress + "\n");
            sb.append("Port: " + port + "\n");
            sb.append("Blender IP address: " + blenderIPaddress + "\n");
            sb.append("Blender port: " + blenderPort + "\n");
            return sb.toString();
    }
```

```
	// filtering pitch and roll, separated by ","
	private String irrFilter(String newPitch, String newRoll) {
		Float pitch = (float) ((float) (Float.parseFloat(newPitch) * 0.3) +
(oldPitch * 0.7));
		Float roll = (float) ((float) (Float.parseFloat(newRoll) * 0.3) +
(oldRoll * 0.7));
		oldPitch = pitch;
		oldRoll = roll;
		String sPitch = pitch.toString();
		String sRoll = roll.toString();
		String s = sPitch + "," + sRoll;
		return s;
	}

}
```