Wroclaw University of Science and Technology

Faculty of Electronics

Automation and Robotics - Embedded Robotics

Intermediate Project

Android application for controlling window-walking robot's movement

Author:
Piotr Łukomski

Supervisor:
Ph.D. Witold Paluszyński

February 5, 2018

Abstract

Window-walking robot control via Bluetooth. Made using Android Studio and Java programming language. Implementation done with two activities, asynchronous class and separate thread for reading and writing to and from Bluetooth streams. Communication with robot done by sending simple strings to invoke movement.

1 Introduction

Main goals of this project were:

- enable exchange of data between application and robot
- create interface to control robot's movement
- functionality to create route for the robot
- functionality to memorise it's path based on time/distance

Besides there were two "hopeful" goals:

- creation of frame for messages and periodical handshaking the robot
- implementing encryption

Goals were fulfilled partially as obstacles arose along the way.
Firstly, I had serious problems in searching for reliable tutorials as they could greatly speed-up my work's progress. After checking many solutions existing online (most of which didn't work at all or didn't work with HC-05), I had to forfeit that path and develop my own solution. It resulted in many hours and days spent on learning about Java, Android and Bluetooth handling.

Secondly, obstacles were in accessing and changing the "backend" of the robot - not everything could have been done in the app itself. It also became clear that in this particular case controlling one leg of the robot is not an easy task, letting synchronizing four of them alone.

Software:

- Android Studio - it is a very good, free IDE - really helps with downloading necessary packages, libraries, gives hints and helps to keep your code clean. It is most useful if you memorize several commands from the keymap.

Hardware:

- Samsung Grand Trend with Android 5.1.1
- Bluetooth adapter HC-05

2 Implementation

2.1 GUI

First thing I started to do was Graphical User Interface. It changed over time from one activity (one screen) to two - first with list of devices you can connect to, second - after you connect - with control buttons.

In the first version of the application I implemented enabling phone's Bluetooth adapter, discoverability and pairing with other devices, however I abandoned that part - it was redundant, made my code messy and application unstable, while it wasn't in the scope of my application. Finally you have only list of your paired devices in the list, so pairing has to be done by Android built-in tools.

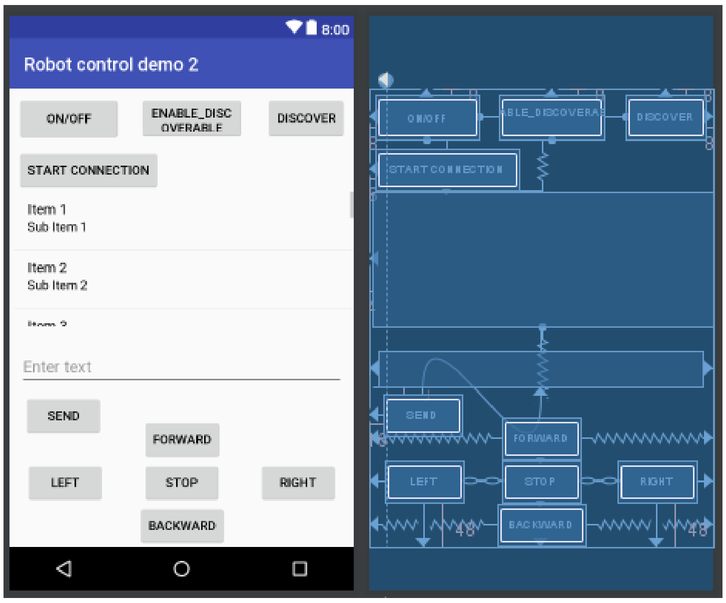


Figure 1: This was first design

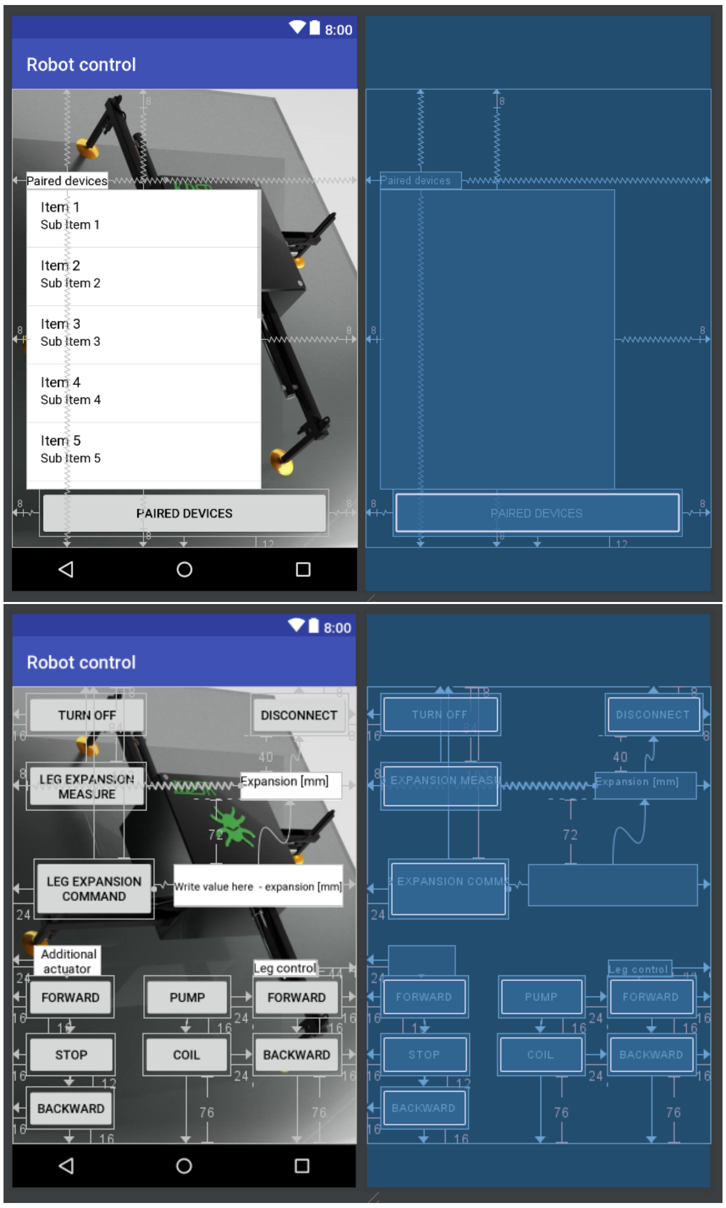
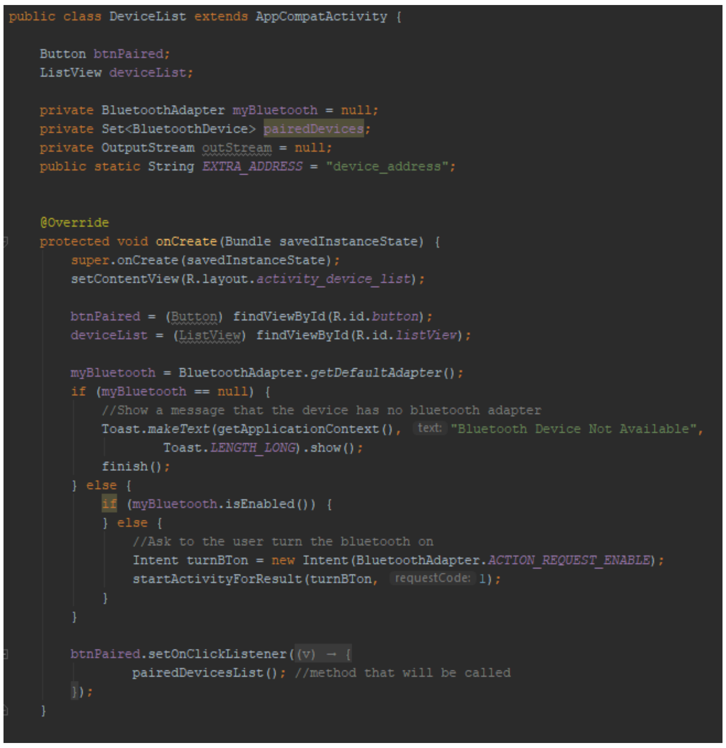


Figure 2: This is final design

As you can see, I abandoned the ”left” and ”right” buttons. Why? Becaused it occurred that the robot is unable to turn left or right.

2.2 The code

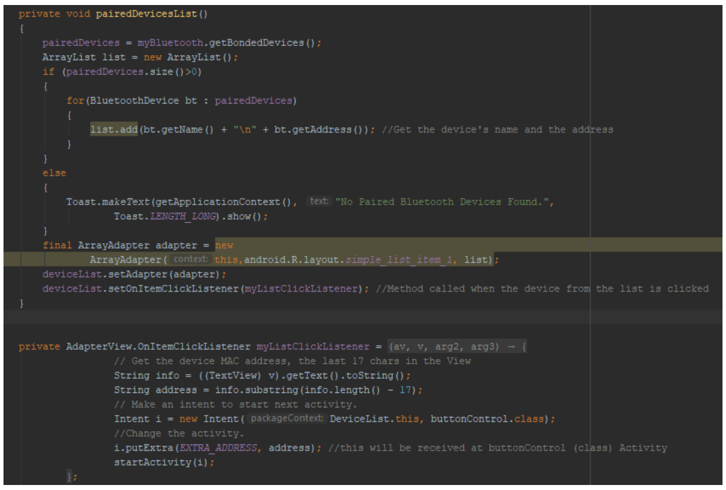
Each activity (screen) contains buttons and other elements that have to be handled in code. After creating GUI we have to declare our elements in our activity's main class and write methods to handle what they do. How it looks in the code of first activity:



```java
public class DeviceList extends AppCompatActivity {

    Button btnPaired;
    ListView deviceList;

    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    private OutputStream outStream = null;
    public static String EXTRA_ADDRESS = "device_address";


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device_list);

        btnPaired = (Button) findViewById(R.id.button);
        deviceList = (ListView) findViewById(R.id.listView);

        myBluetooth = BluetoothAdapter.getDefaultAdapter();
        if (myBluetooth == null) {
            //Show a message that the device has no bluetooth adapter
            Toast.makeText(getApplicationContext(),  text: "Bluetooth Device Not Available",
                    Toast.LENGTH_LONG).show();
            finish();
        } else {
            if (myBluetooth.isEnabled()) {
            } else {
                //Ask to the user turn the bluetooth on
                Intent turnBTon = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(turnBTon,  requestCode: 1);
            }
        }

        btnPaired.setOnClickListener((v) → {
                pairedDevicesList(); //method that will be called
        });
    }
```

Figure 3:

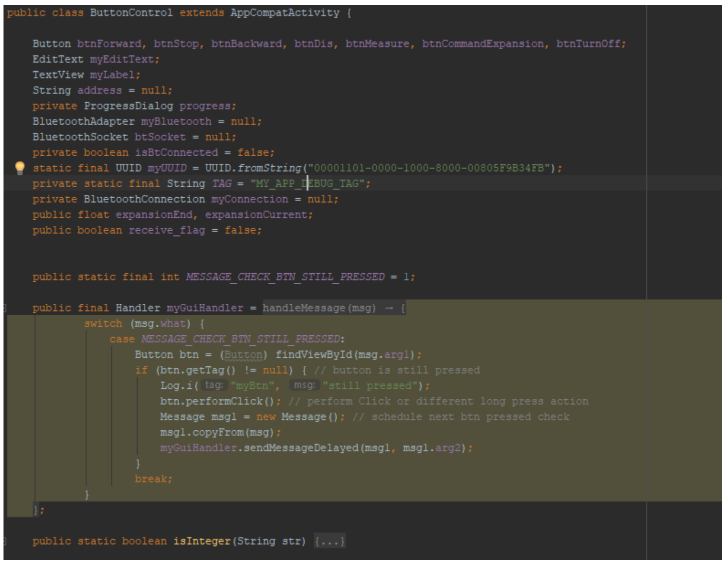
Besides, I had to handle the viewing of paired devices and behaviour after choosing one:



```java
private void pairedDevicesList()
{
    pairedDevices = myBluetooth.getBondedDevices();
    ArrayList list = new ArrayList();
    if (pairedDevices.size()>0)
    {
        for(BluetoothDevice bt : pairedDevices)
        {
            list.add(bt.getName() + "\n" + bt.getAddress()); //Get the device's name and the address
        }
    }
    else
    {
        Toast.makeText(getApplicationContext(),  text: "No Paired Bluetooth Devices Found.",
                Toast.LENGTH_LONG).show();
    }
    final ArrayAdapter adapter = new
            ArrayAdapter( context: this,android.R.layout.simple_list_item_1, list);
    deviceList.setAdapter(adapter);
    deviceList.setOnItemClickListener(myListClickListener); //Method called when the device from the list is clicked
}


private AdapterView.OnItemClickListener myListClickListener = (av, v, arg2, arg3) → {
                // Get the device MAC address, the last 17 chars in the View
                String info = ((TextView) v).getText().toString();
                String address = info.substring(info.length() - 17);
                // Make an intent to start next activity.
                Intent i = new Intent( packageContext: DeviceList.this, buttonControl.class);
                //Change the activity.
                i.putExtra(EXTRA_ADDRESS, address); //this will be received at buttonControl (class) Activity
                startActivity(i);
        };
```

Figure 4:

In the second activity there is more buttons, which means more behaviours and more code:

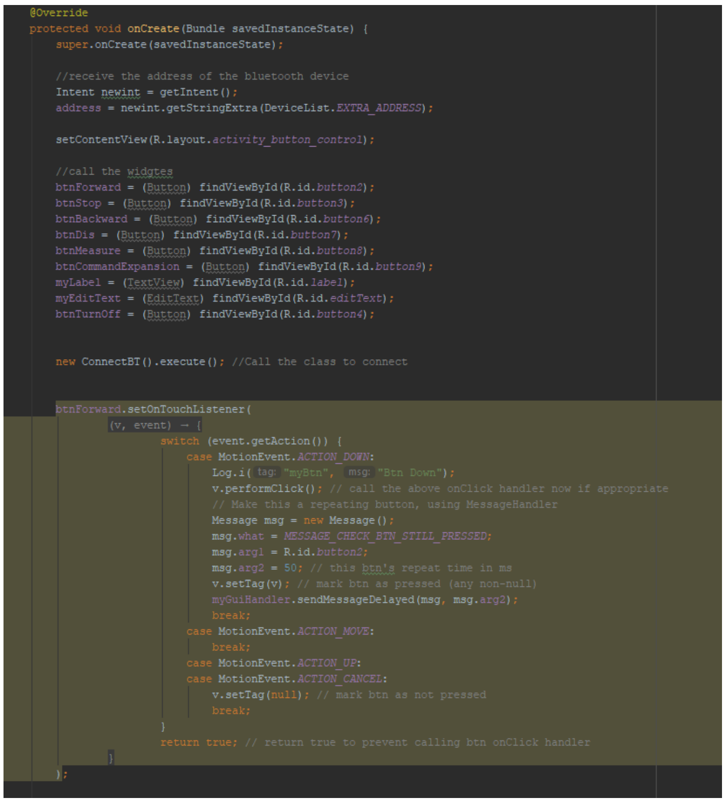


```java
public class ButtonControl extends AppCompatActivity {

    Button btnForward, btnStop, btnBackward, btnDis, btnMeasure, btnCommandExpansion, btnTurnOff;
    EditText myEditText;
    TextView myLabel;
    String address = null;
    private ProgressDialog progress;
    BluetoothAdapter myBluetooth = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    private static final String TAG = "MY_APP_DEBUG_TAG";
    private BluetoothConnection myConnection = null;
    public float expansionEnd, expansionCurrent;
    public boolean receive_flag = false;


    public static final int MESSAGE_CHECK_BTN_STILL_PRESSED = 1;

    public final Handler myGuiHandler = handleMessage(msg) → {
            switch (msg.what) {
                case MESSAGE_CHECK_BTN_STILL_PRESSED:
                    Button btn = (Button) findViewById(msg.arg1);
                    if (btn.getTag() != null) { // button is still pressed
                        Log.i( tag: "myBtn",  msg: "still pressed");
                        btn.performClick(); // perform Click or different long press action
                        Message msg1 = new Message(); // schedule next btn pressed check
                        msg1.copyFrom(msg);
                        myGuiHandler.sendMessageDelayed(msg1, msg1.arg2);
                    }
                    break;
            }
    };

    public static boolean isInteger(String str) {...}
```

Figure 5:

As you can see, there is declared handler which is used to check whether a button is still pressed. It is used to detect end of the long-press on the button which allows you to perform different actions on single-click and long-click. In this case it is used to perform extension of robot's leg only a bit on single-click and as long as you hold the button. Quite nice feature, which I am satisfied with.
There is also custom function "isInteger" which allows me to check whether a String is in fact an Integer.

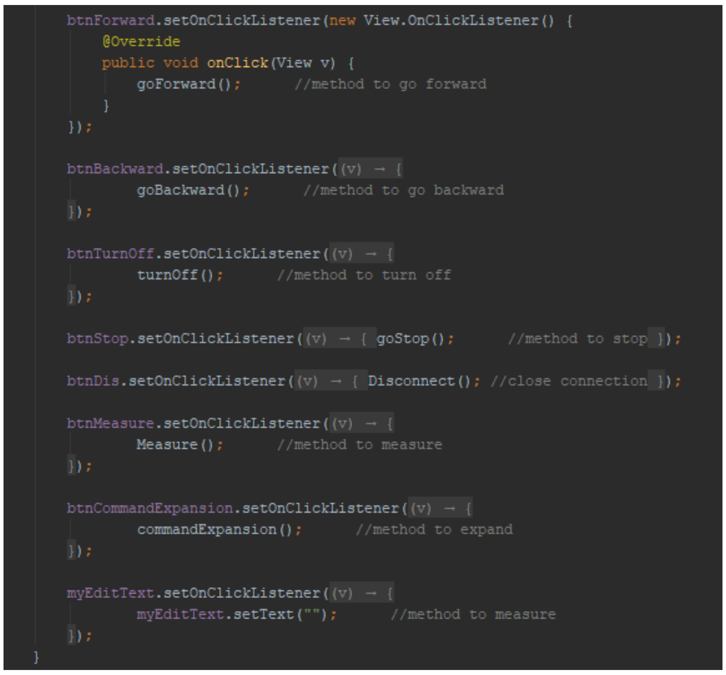
Again declaration of buttons and text fields, invoking asynchronous class to execute and perform connection and further handling long-press, in this particular case on btnForward:



```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //receive the address of the bluetooth device
        Intent newint = getIntent();
        address = newint.getStringExtra(DeviceList.EXTRA_ADDRESS);

        setContentView(R.layout.activity_button_control);

        //call the widgtes
        btnForward = (Button) findViewById(R.id.button2);
        btnStop = (Button) findViewById(R.id.button3);
        btnBackward = (Button) findViewById(R.id.button6);
        btnDis = (Button) findViewById(R.id.button7);
        btnMeasure = (Button) findViewById(R.id.button8);
        btnCommandExpansion = (Button) findViewById(R.id.button9);
        myLabel = (TextView) findViewById(R.id.label);
        myEditText = (EditText) findViewById(R.id.editText);
        btnTurnOff = (Button) findViewById(R.id.button4);


        new ConnectBT().execute(); //Call the class to connect


        btnForward.setOnTouchListener(
                (v, event) → {
                    switch (event.getAction()) {
                        case MotionEvent.ACTION_DOWN:
                            Log.i( tag: "myBtn",  msg: "Btn Down");
                            v.performClick(); // call the above onClick handler now if appropriate
                            // Make this a repeating button, using MessageHandler
                            Message msg = new Message();
                            msg.what = MESSAGE_CHECK_BTN_STILL_PRESSED;
                            msg.arg1 = R.id.button2;
                            msg.arg2 = 50; // this btn's repeat time in ms
                            v.setTag(v); // mark btn as pressed (any non-null)
                            myGuiHandler.sendMessageDelayed(msg, msg.arg2);
                            break;
                        case MotionEvent.ACTION_MOVE:
                            break;
                        case MotionEvent.ACTION_UP:
                        case MotionEvent.ACTION_CANCEL:
                            v.setTag(null); // mark btn as not pressed
                            break;
                    }
                    return true; // return true to prevent calling btn onClick handler
                }
        );
```

Figure 6:

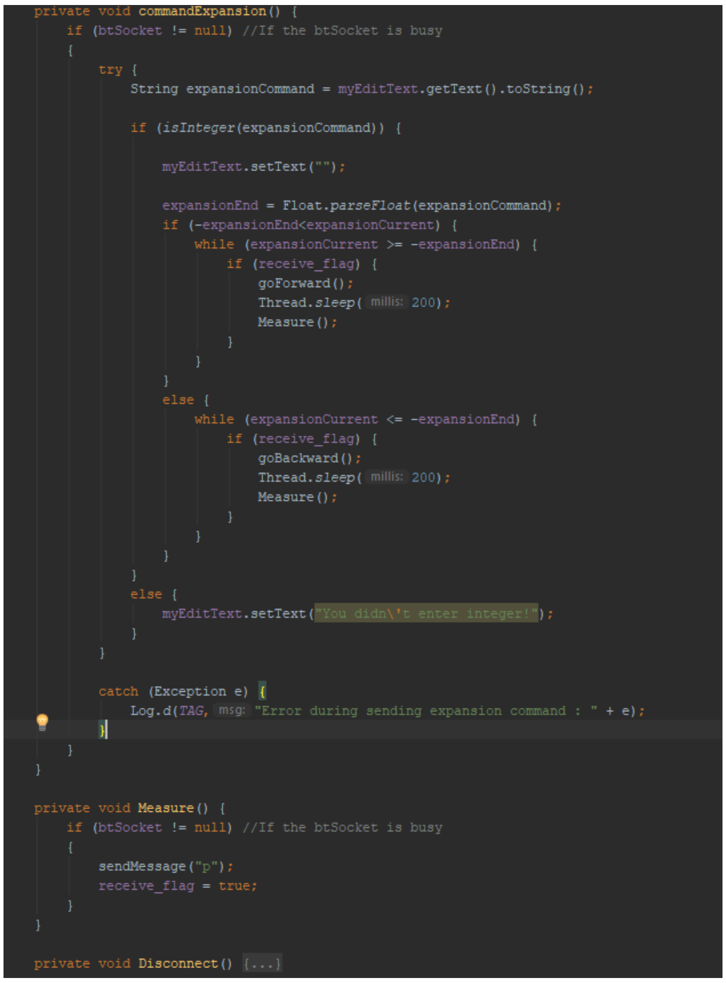
Only btnBackward is handled in the same way, for the rest it wasn't needed, so they only invoke their methods:



```java
    btnForward.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            goForward();      //method to go forward
        }
    });

    btnBackward.setOnClickListener((v) → {
            goBackward();      //method to go backward
    });

    btnTurnOff.setOnClickListener((v) → {
            turnOff();      //method to turn off
    });

    btnStop.setOnClickListener((v) → { goStop();      //method to stop });

    btnDis.setOnClickListener((v) → { Disconnect(); //close connection });

    btnMeasure.setOnClickListener((v) → {
            Measure();      //method to measure
    });

    btnCommandExpansion.setOnClickListener((v) → {
            commandExpansion();      //method to expand
    });

    myEditText.setOnClickListener((v) → {
            myEditText.setText("");      //method to measure
    });
}
```

Figure 7:

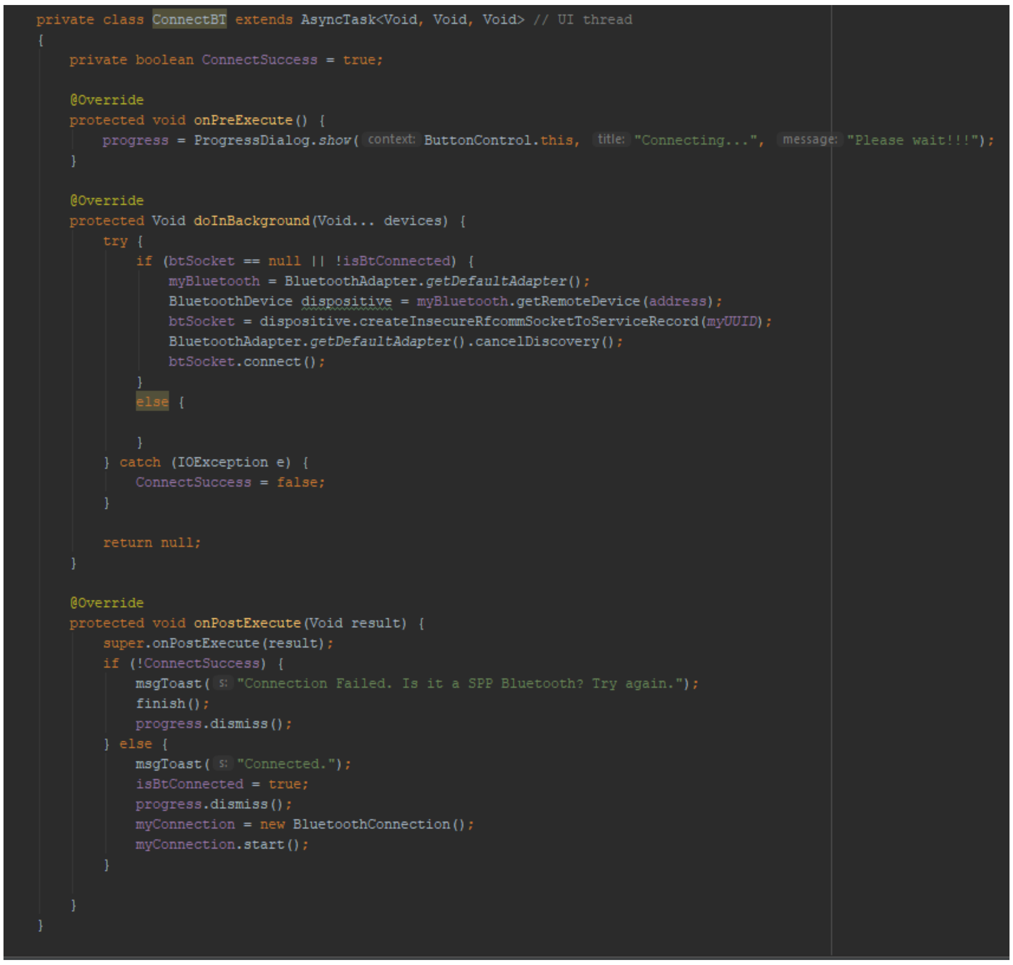
Here we have example of method invoked by button - first one has some logic in it - it sends messages to go forward/backward and to measure as long as the desired measurement value is obtained:



```java
private void commandExpansion() {
    if (btSocket != null) //If the btSocket is busy
    {
        try {
            String expansionCommand = myEditText.getText().toString();

            if (isInteger(expansionCommand)) {

                myEditText.setText("");

                expansionEnd = Float.parseFloat(expansionCommand);
                if (-expansionEnd<expansionCurrent) {
                    while (expansionCurrent >= -expansionEnd) {
                        if (receive_flag) {
                            goForward();
                            Thread.sleep( millis: 200);
                            Measure();
                        }
                    }
                }
                else {
                    while (expansionCurrent <= -expansionEnd) {
                        if (receive_flag) {
                            goBackward();
                            Thread.sleep( millis: 200);
                            Measure();
                        }
                    }
                }
            }
            else {
                myEditText.setText("You didn\'t enter integer!");
            }
        }

        catch (Exception e) {
            Log.d(TAG, msg: "Error during sending expansion command : " + e);
        }
    }
}

private void Measure() {
    if (btSocket != null) //If the btSocket is busy
    {
        sendMessage("p");
        receive_flag = true;
    }
}

private void Disconnect() {...}
```

Figure 8:

Now we come to the part that caused most troubles and hours of contemplation - handling Bluetooth connection:

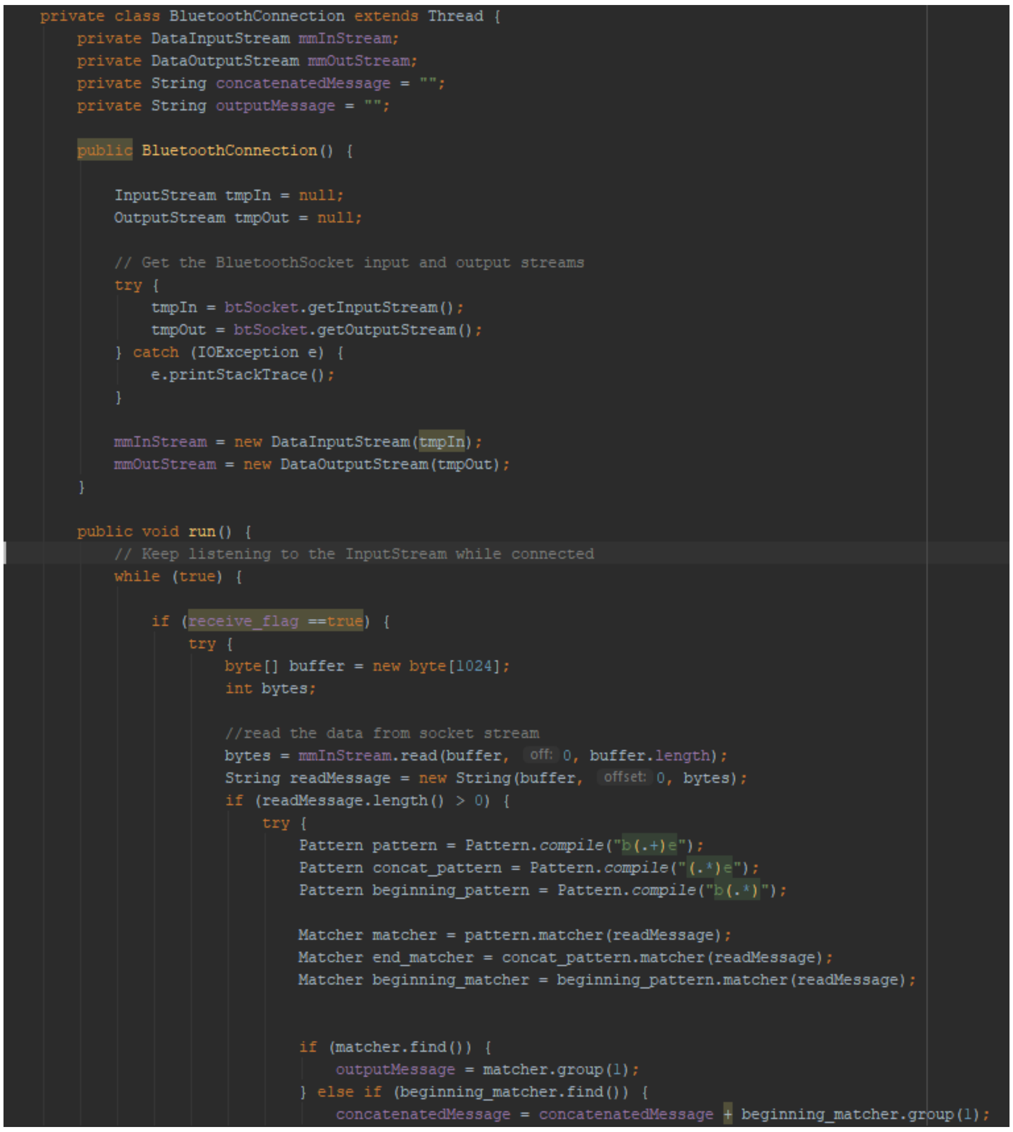


```java
private class ConnectBT extends AsyncTask<Void, Void, Void> // UI thread
{
    private boolean ConnectSuccess = true;

    @Override
    protected void onPreExecute() {
        progress = ProgressDialog.show( context: ButtonControl.this,  title: "Connecting...",  message: "Please wait!!!");
    }

    @Override
    protected Void doInBackground(Void... devices) {
        try {
            if (btSocket == null || !isBtConnected) {
                myBluetooth = BluetoothAdapter.getDefaultAdapter();
                BluetoothDevice dispositive = myBluetooth.getRemoteDevice(address);
                btSocket = dispositive.createInsecureRfcommSocketToServiceRecord(myUUID);
                BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                btSocket.connect();
            }
            else {

            }
        } catch (IOException e) {
            ConnectSuccess = false;
        }

        return null;
    }

    @Override
    protected void onPostExecute(Void result) {
        super.onPostExecute(result);
        if (!ConnectSuccess) {
            msgToast( s: "Connection Failed. Is it a SPP Bluetooth? Try again.");
            finish();
            progress.dismiss();
        } else {
            msgToast( s: "Connected.");
            isBtConnected = true;
            progress.dismiss();
            myConnection = new BluetoothConnection();
            myConnection.start();
        }

    }
}
```

Figure 9:

Above we have an asynchronous class that shows us dialog telling that connection is being made and performs connecting to the specified Bluetooth device. On PostExecute it invokes "Connected" toast message (using myToast custom method), initializes and starts new thread that will run in the background and read/write any data passed to the input or output of the Bluetooth socket.

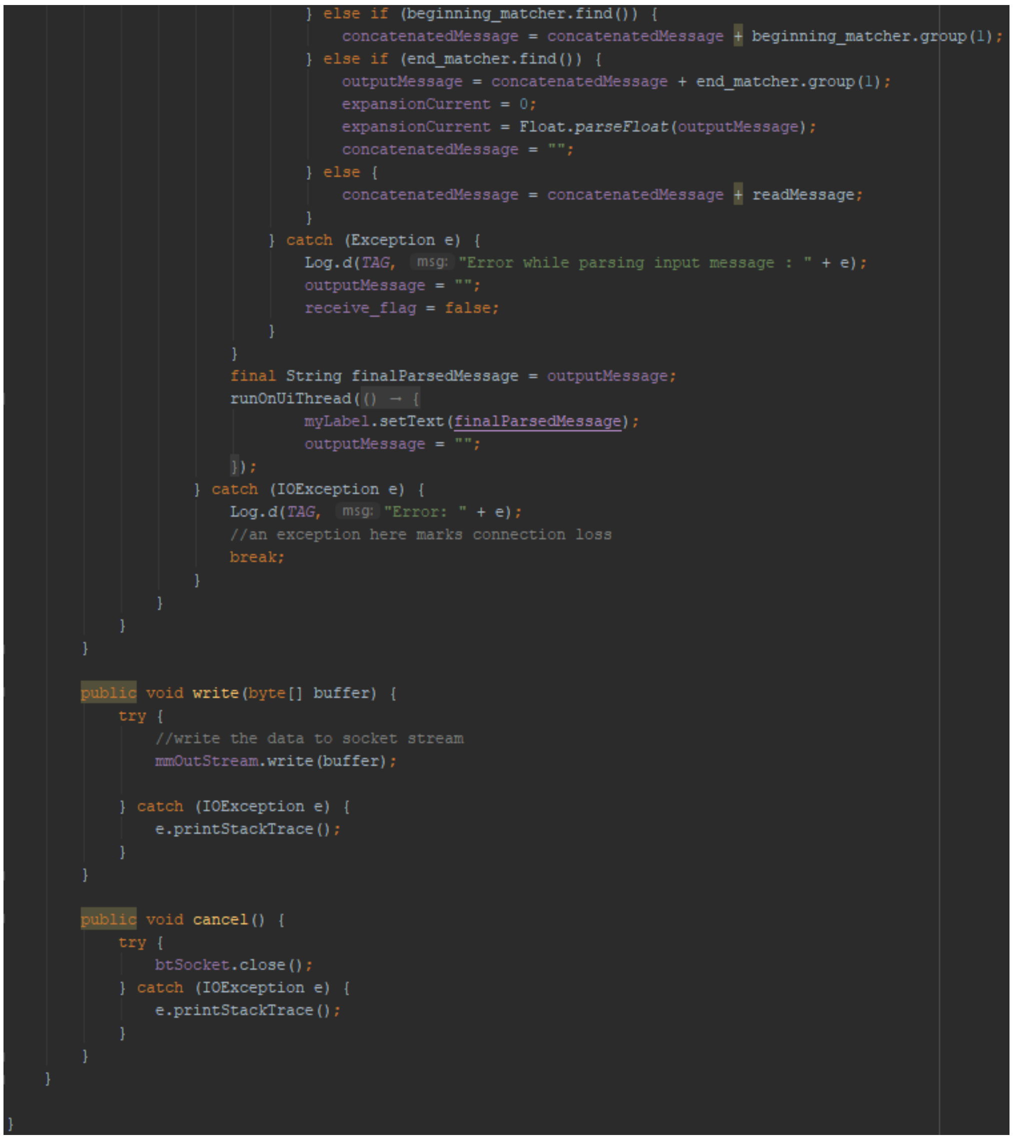
The following screen is a very important one - it handles receiving measurements from HC-05. Because of the fact that we are communicating using Strings I used regular expressions to detect what comes through the input stream. There was a big problem, because it occurred that HC-05 sends measurements of leg expansion character after character in a fast way. It caused input stream to collect that data in an unregular way, sometimes several characters at the time, sometimes one by one. It was the reason I used regular expressions as method to detect beginning and end of each communicate. Later it turned out not to be such great idea, because a lot of logic in this method caused the thread to slow down and bring many problems while implementing "expand to the given measure" functionality, where reading and writing happens very quickly.



```java
private class BluetoothConnection extends Thread {
    private DataInputStream mmInStream;
    private DataOutputStream mmOutStream;
    private String concatenatedMessage = "";
    private String outputMessage = "";

    public BluetoothConnection() {

        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = btSocket.getInputStream();
            tmpOut = btSocket.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }

        mmInStream = new DataInputStream(tmpIn);
        mmOutStream = new DataOutputStream(tmpOut);
    }

    public void run() {
        // Keep listening to the InputStream while connected
        while (true) {

            if (receive_flag ==true) {
                try {
                    byte[] buffer = new byte[1024];
                    int bytes;

                    //read the data from socket stream
                    bytes = mmInStream.read(buffer, off: 0, buffer.length);
                    String readMessage = new String(buffer, offset: 0, bytes);
                    if (readMessage.length() > 0) {
                        try {
                            Pattern pattern = Pattern.compile("b(.+)e");
                            Pattern concat_pattern = Pattern.compile("(.*)e");
                            Pattern beginning_pattern = Pattern.compile("b(.*)");

                            Matcher matcher = pattern.matcher(readMessage);
                            Matcher end_matcher = concat_pattern.matcher(readMessage);
                            Matcher beginning_matcher = beginning_pattern.matcher(readMessage);


                            if (matcher.find()) {
                                outputMessage = matcher.group(1);
                            } else if (beginning_matcher.find()) {
                                concatenatedMessage = concatenatedMessage + beginning_matcher.group(1);
```

Figure 10:

On the following screen we see the rest of ”read” method, along with writing measurement value to screen (which wasn’t so obvious). There is also ”write” method for sending messages and ”cancel” to close the Bluetooth Socket.



```
                                } else if (beginning_matcher.find()) {
                                    concatenatedMessage = concatenatedMessage + beginning_matcher.group(1);
                                } else if (end_matcher.find()) {
                                    outputMessage = concatenatedMessage + end_matcher.group(1);
                                    expansionCurrent = 0;
                                    expansionCurrent = Float.parseFloat(outputMessage);
                                    concatenatedMessage = "";
                                } else {
                                    concatenatedMessage = concatenatedMessage + readMessage;
                                }
                            } catch (Exception e) {
                                Log.d(TAG,  msg: "Error while parsing input message : " + e);
                                outputMessage = "";
                                receive_flag = false;
                            }
                        }
                        final String finalParsedMessage = outputMessage;
                        runOnUiThread(() → {
                                myLabel.setText(finalParsedMessage);
                                outputMessage = "";
                        });
                    } catch (IOException e) {
                        Log.d(TAG,  msg: "Error: " + e);
                        //an exception here marks connection loss
                        break;
                    }
                }
            }
        }

        public void write(byte[] buffer) {
            try {
                //write the data to socket stream
                mmOutStream.write(buffer);

            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public void cancel() {
            try {
                btSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}
```

Figure 11:

3 Summary

As you can see the robot control is not what it was supposed to be due to several reasons. Firstly, leg expansion measurement wasn't ready in the beginning, which prevented me from starting with development of my goals in the first place. In exchange, we focused on implementing other functionalities, like turning on/off, using pump and expanding leg to the desired length. The progress of work is, in my opinion, good, and right now we are in a comfortable place to automate the movement control with significantly less effort than we invested so far. Implementing "hope to" goals is now also quite real, as the basic fundaments are set.

Besides, Android Studio turned out to be a very friendly environment to code in. However, Android programming differs a lot from regular Java usage and requires a lot of effort to learn it's specifics. It's quite rewarding though and enforces to use complicated mechanisms of programming in a flexible way you probably wouldn't use often in other projects.

3.1 References

You can make lists with automatic numbering . . .

- "The Busy Coder's Guide to Android Development" - Mark L. Murphy
- "Android Programming for Beginners" - John Horton
- HC Serial Bluetooth Products User Instructional Manual
- "Bluetooth Tutorial" on youtube - Mitch Tabian
- Internet - developer.android.com, stackoverflow.com