4.2.2. Linux Kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

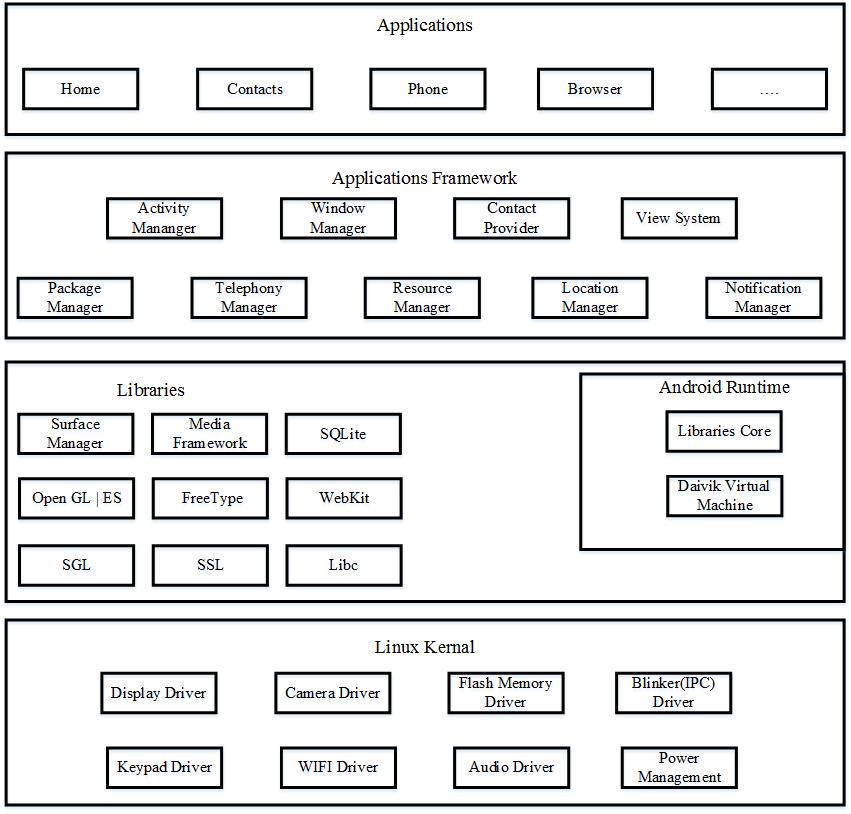


Figure 4.1. Android Architecture

4.2.3. Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows –

* android.app: Provides access to the application model and is the cornerstone of all Android applications.
* android.content: Facilitates content access, publishing and messaging between applications and application components.
* android.content: Facilitates content access, publishing and messaging between applications and application components.
* android.opengl: A Java interface to the OpenGL ES 3D graphics rendering API.
* android.os: Provides applications with access to standard operating system services including messages, system services and inter-process communication.
* android.text: Used to render and manipulate text on a device display.
* android.view: The fundamental building blocks of application user interfaces.
* android.widget: A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

4.2.4. Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

4.2.5. Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications. Application frameworks became popular with the rise of graphical user interfaces (GUIs). The Android framework includes the following key services –

* Activity Manager: Controls all aspects of the application lifecycle and activity stack.
* Content Providers: Allows applications to publish and share data with other applications.
* Resource Manager: Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
* Notifications Manager: Allows applications to display alerts and notifications to the user.
* View System: An extensible set of views used to create application user interfaces.

4.2.6. Applications

All the Android application at the top layer. Written applications are installed only on this layer. Examples of such applications are Contacts Books, Browser, and Games etc. Robotic Arm Controller application will be described on section 4.5.

**4.3. Java Language**

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The target of Java is to write a program once and then run this program on multiple operating systems. The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the statesmanship for Java. In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called OpenJDK. The base of java programming structure are following.

4.3.1. Class

A class is a template that describes the data and behavior associated with an instance of that class. A class is defined by the class keyword and must start with a capital letter. The body of a class is surrounded by {}.



Figure 4.2. Class keyword use in Java program

The data associated with a class is stored in variables; the behavior associated to a class or object is implemented with methods. A class is contained in a text file with the same name as the class plus the .java extension. It is also possible to define inner classes, these are classes defined within another class, and in this case you do not need a separate file for this class.

4.3.2. Object

An object is an instance of a class. The object is the real element which has data and can perform actions. Each object is created based on the class definition. The class can be seen as the blueprint of an object, i.e., it describes how an object is created.

4.3.3. Package

Java groups classes into functional packages. Packages are typically used to group classes into logical units. For example, all graphical views of an application might be placed in the same package called “tukaly.aungye.robotarm.views”.

It is common practice to use the reverse domain name of the company as top level package. For example, the company might own the domain, vogella.com and in this example the Java packages of this company starts with “tukalay.aungye”.

Other main reason for the usage of packages is to avoid name collisions of classes. A name collision occurs if two programmers give the same fully qualified name to a class. The fully qualified name of a class in Java consists of the package name followed by a dot (.) and the class name.

Without packages, a programmer may create a Java class called Test. Another programmer may create a class with the same name. With the usage of packages you can tell the system which class to call. For example, if the first programmer puts the Test class into package report and the second programmer puts his class into package “xmlreader” you can distinguish between these classes by using the fully qualified name, e.g, xmlreader.Test or report.Test.

4.3.4. Inheritance

Class can be derived from another class. In this case this class is called a subclass. Another common phrase is that a class extends another class. The class from which the subclass is derived is called a superclass. Inheritance allows a class to inherit the behavior and data definitions of another class. The following codes demonstrates how a class can extend another class. In Java a class can only extend a maximum of one class.



Figure 4.3. MyExtensionClass inherit from MyBaseClass

4.3.5. Exception Handling

In Java an exception is an event to indicate an error during the runtime of an application. So this disrupts the usual flow of the application’s instructions. In general exceptions are thrown up in the call hierarchy until they get catched.

4.3.6. Interfaces

An interfaces is a type similar to a class and is defined via the interface keyword. Interfaces are used to define common behavior of implementing classes. If two classes implement the same interface, other code which work on the interface level, can use objects of both classes.

Like a class an interface defines methods. Classes can implement one or several interfaces. A class which implements an interface must provide an implementation for all abstract methods defined in the interface.



Figure 4.4. Example implementation of an interface.

**4.4. XML Design**

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. XML tags are not predefined in XML. We must define our own Tags. Xml as itself is well readable both by human and machine. Also, it is scalable and simple to develop. In Android we use xml for designing our layouts because xml is lightweight language so it doesn’t make our layout heavy.

4.4.1. Basics of User Interface

The whole concept of Android User Interface is defined using the hierarchy of View and ViewGroup objects. A ViewGroup is an invisible container that organizes child views. These child views are other widgets which are used to make the different parts of UI. One ViewGroup can have another ViewGroup as an child element as shown in the figure given below’

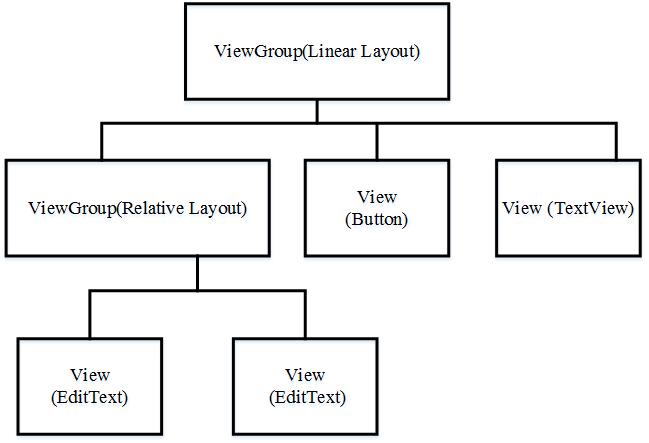


Figure 4.5. View Group in android UI

Here in above Diagram ViewGroup (Linear Layout) contains one ViewGroup (i.e. Relative Layout) and two View (Button and TextView). Further two more View (i.e. 2 EditText) are nested inside Relative Layout ViewGroup. It is important to note that one layout can be nested in another layout.

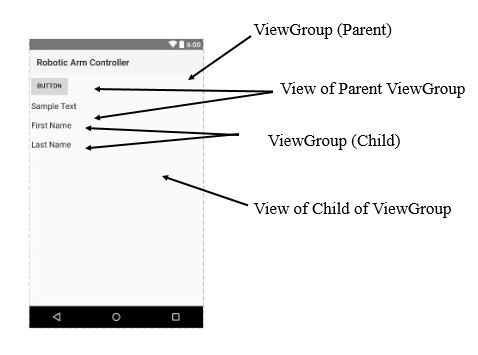


Figure 4.6. Basic UI Explanation in Android

Every Android application screen has some components like button, Text or images. These are contained inside the ViewGroup. Layouts are the best examples for ViewGroups. The different types of layout in android are Linear Layout, Relative Layout, Absolute Layout, Table Layout and Frame Layout.

**4.5. Android Studio**

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, mac OS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

Android Studio was announced on May 16, 2013 at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0. The current stable version is 3.1.3 released in June 2018.

4.5.1. Features

The following features are provided in the current stable version:

* Gradle-based build support
* Android-specific refactoring and quick fixes
* Lint tools to catch performance, usability, version compatibility and other problems
* ProGuard integration and app-signing capabilities
* Template-based wizards to create common Android designs and components
* A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations
* Support for building Android Wear apps
* Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine.
* Android Virtual Device (Emulator) to run and debug apps in the Android studio.

Android Studio supports all the same programming languages of IntelliJ, and PyCharm e.g. Python, and Kotlin; and Android Studio 3.0 supports "Java 7 language features and a subset of Java 8 language features that vary by platform version." External projects backport some Java 9 features.

**4.6. Main Activity of Robotic Arm Controller**

In Mobile SDK apps, the main activity begins immediately after the user logs in. Once the main activity is running, it can launch other activities, which in turn can launch sub-activities. When the application exits, it does so by terminating the main activity. All other activities terminate in a cascade from within the main activity. Main activity class in Robotic Arm Controller application provides the program to control the robotic arm and for UI registration. All scripts in Main Activity are as following.

4.6.1. Initialization

The program need to import packages from android and java libraries because the program use the features from this like activities, user interfaces, and mathematics.

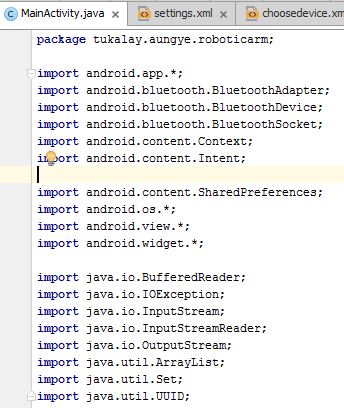


Figure 4.7. Initialization of MainActivity.java

4.6.2. Main Activity Class

The application Main Activity class extends the abstract Mobile SDK activity class, “android.os.Activity.class”. In Robotic Arm Controller application, text-views are used to show text on UI. Button widgets are also declared to be clicked from the user. String variables are used for father use of text views.

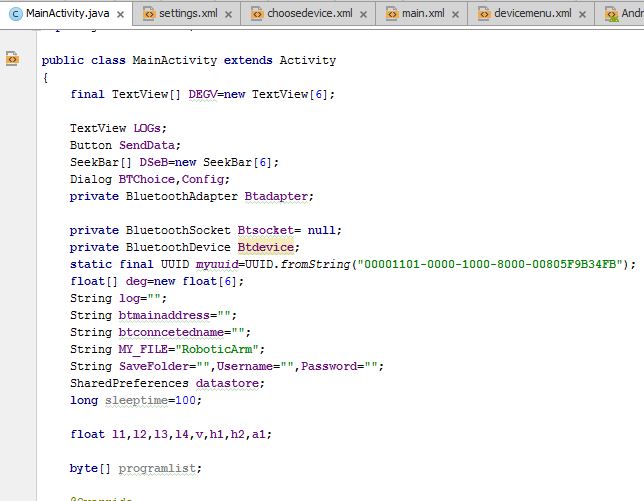


Figure. 4.8. Initialization of MainActivity.Class

4.6.3. Create Methods

When an Activity first call or launched then “onCreate()” method is responsible to create the activity. Whenever orientation (i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then “savedInstanceState” i.e. object of Bundle Class will save the state of an Activity. After Orientation changed then “onCreate()” will call and recreate the activity and load all data from “savedInstanceState”. Basically Bundle class is used to store the data of activity whenever above condition occur in app. The “onCreate()” is not required for apps. But the reason it is used in app is because that method is the best place to put initialization code. You could also put your initialization code in “onStart()” or “onResume()” and when you app will load first, it will work same as in “onCreate()”.

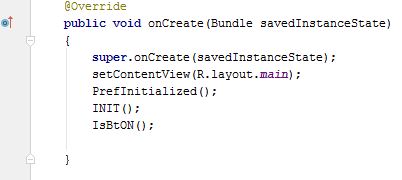


Figure. 4.9. “onCreate” Method in Main Activity

4.6.4. Initialization of Bluetooth

The application require the Bluetooth adapter activity so that the program need to declare “IsBtOn” function. If the adapter is disabled or empty, the program will return null.

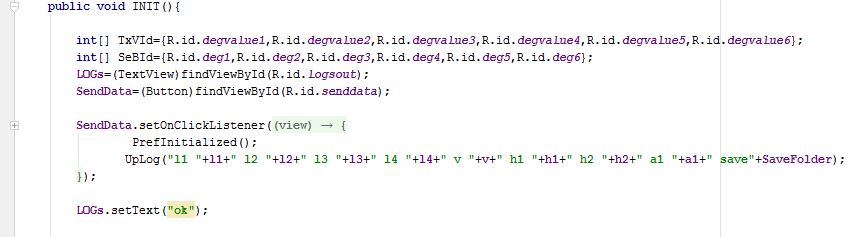


Figure. 4.10. Initialization program for checking Bluetooth Adapter

4.6.5. Bluetooth Device Connection

When the user want to connect the Bluetooth device of robotic arm, the program will create insecure frequency socket with a UUID by calling the “BTHC()” class. This class extends the “AsyncTask<>” class to run the scrip in background. The program will vibrate 250ms after the android phone if the Bluetooth device is connected by programming “vl.vibrate(250)” function. The function “while()” is used to receive the messages from robotic arm.



Figure. 4.11. BTHC class for Bluetooth socket

4.6.6. Data Initialization

The program set a byte array from an array of degrees which have seven units because robotic arm uses five axes, one griper and for start and stop byte.

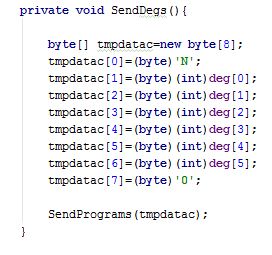


Figure. 4.12. SendDegs() function for initialization data

4.6.7. Data Transmission Function

The microcontroller of robotic arm receives the data by byte units so that the program declare array of byte with the function of “SendPrograms(byte[])”. The program also declare the script “Btsocket.getOutputStream()” as the transmission need the output stream from the Bluetooth socket. For the error of getting output stream, the program use “try-catch” function for exception of input socket and output socket.



Figure. 4.13. Send program() function for data transmission

**4.7. Robotic Arm Controller Application**

Robotic Arm Controller application is the android software used to control the robotic arm via Bluetooth. This is developed using java program with Android Studio software. The package name of this application must be unique to avoid override installation and it is “tukalay.aungye.roboticarm”.

4.7.1. Permissions

Android apps must request permission to access sensitive user data (such as contacts and SMS) or certain system features (such as the camera and internet access). Each permission is identified by a unique label. For example, an app that needs to send SMS messages must have the following line in the manifest:

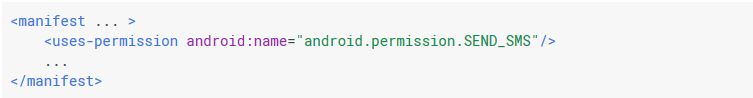


Figure 4.14. Example of request permission line in manifest

Robotic arm controller application need to request the following permission

* Bluetooth
* Read/Write External Storage
* Vibrate

4.7.2. Controller UI design

The layouts for this application is programmed using xml language. There are four UIs used in Robotic Arm Controller application. They are

* Main Interface
* Menu Interface
* Bluetooth Devices Interface
* Configuration Interface

4.7.2.1. Main Interface

Main interface is the first interface when the application is lunched. This interface contains controller joint stick and joints seek bars to control the robotic arm.



Figure 4.15. Main user interface in Robotic Arm Controller

4.7.2.2. Menu Interface

This interface will appear when the menu button is clicked in android phone running Robotic Arm Controller application. This interface has a list of option to lunch other interfaces.



Figure 4.16. Menu UI in Robotic Arm Controller

4.7.2.3. Bluetooth Device Interface

This interface will appear when the user choose Choose Device option from the Menu Interface. List of Bluetooth device with physical address will be shown in this interface and must be choose the device (HC-05) that is used in robotic am. Scan button is used to scan when the devices are missing.

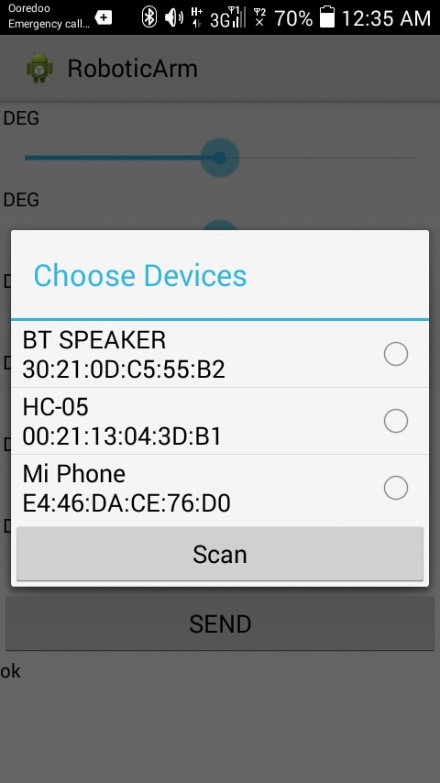


Figure 4.17. Bluetooth Devices Interface

4.7.2.4. Configuration Interface

This interface is used to configure the dimensions of the robotic arm. Many edit-text box will be shown in this interface. Save button can be click when the configuration is finish or to save the configuration.

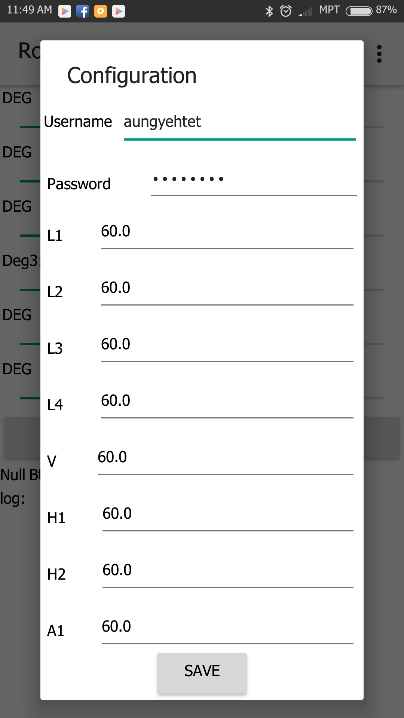


Figure 4.18. Configuration Interface

**4.8. Summary**

The android software implementation for robotic arm is programmed using the Android Studio Software. The test and result of the system will be described in the next chapter.