



TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Department of Mechatronics

Chair of Mechatronics systems

Giorgi Peranidze

Android-MATLAB/Simulink interface

Android-MATLAB/Simulink liides

MSc thesis

MHK70LT

The author applies for

The academic degree

Master of Science in Engineering

Tallinn

2016

Author's declaration

I declare that I have written this graduation thesis independently.

These materials have not been submitted for any academic degree.

All the works of other authors used in this thesis have been referenced.

The thesis was completed under _____ supervision

Author

_____ Signature

The thesis complies with the requirements for graduation thesis.

Supervisor

_____ Signature

Accepted for defense.

_____ Chairman of the defense committee

_____ Signature

Abstract

The main goal of this work is to create Android libraries, which will visualize the result of a simulation, done in MATLAB/Simulink, of two devices: Magnetic Levitation System (MLS) and 3D crane.

The Android application created using libraries, created during this work, will communicate to a PC, running MATLAB/Simulink simulation, using internet connection.

The libraries should be scalable, reliable and easy to implement. Support different screen resolutions and Android OS version.

The Magnetic Levitation System is a nonlinear, open-loop, unstable time varying frictionless dynamical system that keeps the metallic ball levitated.

3D crane is The three-dimensional model of industrial crane, which integrates with MATLAB/Simulink and operates in real time.

1 TABLE CONTENTS

2	Foreword.....	7
3	EESSÕNA.....	8
4	Introduction.....	9
5	Previous work	11
6	Literature study	13
6.1	Android.....	13
6.2	Application model: applications, tasks, thread and processes	13
6.3	Android application.....	14
6.4	Android software development kit (SDK)	15
6.5	Challenges in Android app development	15
7	Matlab	17
7.1	Simulink.....	17
8	System description	18
8.1	3D crane	18
8.2	Magnetic levitation system.....	20
9	Development	22
9.1	Communication between devices.....	22
9.1.1	User datagram protocol.....	22
9.1.2	UDP in Android	22
9.1.3	Sending data from Android.....	23
9.1.4	Receiving data in Android	24
9.1.5	Communication in Matlab	27
9.1.6	Sending data from Matlab.....	27
9.1.7	Receiving data in Matlab	29

9.2	3D crane view.....	30
9.2.1	Drawing the CraneView	33
9.2.2	CraneAngleView.....	33
9.2.3	Drawing CraneAngleView.....	34
9.2.4	User input.....	35
9.3	Magnetic levitation system view	35
9.3.1	Scalable vector image	35
9.3.2	SVG in Android	36
9.3.3	Drawing the MLSView.....	38
10	Manual	40
10.1	Importing library to project	40
10.2	Using 3D crane library	42
10.3	Using MLS library.....	46
11	Demonstration applications	48
11.1	Magnetic Levitation System app	48
12	Simulation in MATLAB/Simulink	53
12.1	Magnetic levitation system.....	53
12.2	3D crane.....	54
13	Summary.....	57
14	kokkuvõte.....	59
15	References.....	61
16	Appendices.....	64
	Appendix 1. CraneView code.....	64
	Appendix 2. CraneAngleView code	74
	Appendix 3. SendDataTask	77

Appendix 4. Receiving service code.....	79
Appendix 5. Attributes for 3D crane.....	82
Appendix 6. MLS view code	83
Appendix 7. MLS view's attributes	87

Figure 4.1 Basic data flow	9
Figure 5.1 MLS animation in Simulink	11
Figure 5.2 iPad app UI	12
Figure 8.1 3D crane.....	18
Figure 8.2 3D crane setup [25]	19
Figure 8.3 Magnetic Levitation System.....	20
Figure 8.4 MLS laboratory setup [20]	21
Figure 9.1 Data sending on Android.....	24
Figure 9.2 Data receiving in Android	26
Figure 9.3 Board setup.....	27
Figure 9.4 Data for output, 3D crane	28
Figure 9.5 Package Output block for 3D crane.....	29
Figure 9.6 Packet input	30
Figure 9.7 Crane view.....	32
Figure 9.8 CraneAngleView different crane positions	34
Figure 9.9 MLS model in Blender, SVG generated with FreeStyle, colored SVG image	36
Figure 9.10 MLS View	38
Figure 10.1 Add module	41
Figure 10.2 Add module dependency	41
Figure 10.3 Inputs	42
Figure 11.1 User interface from MLS app.....	49
Figure 11.2 Settings screen from demonstration apps3D crane app.....	50
Figure 11.3 3D Crane app in portrait mode	51
Figure 11.4 3D crane app in landscape mode	52
Figure 12.1 MLS model in Simulink3D crane.....	53
Figure 12.2 Checking incoming data	53
Figure 12.3 Simulation result.....	54
Figure 12.4 3D Crane model for simulation in Simulink	55
Figure 12.5 Controlling simulation.....	55
Figure 12.6 Crane's deviation	56

2 FOREWORD

This thesis represents the work done by me for a Master's degree in Mechatronics at Tallinn University of Technologies. The main aim of this project was to create an Android library which would communicate with simulation or control system running on the PC.

The task was provided from Control Systems Research Laboratory (A-Lab). The work was done in the laboratory provided by the said laboratory. Programming was done in Android Studio and MATLAB R2015b.

The supervisors were Leo Teder from faculty of mechanical engineering and Aleksei Tepljakov from Control Systems Research Laboratory.

I would like to thank my family and friends for support during the work period. Would like to thank the A-Lab for providing well equipped laboratory and my supervisors for their help and guidance.

3 EESSÕNA

See töö on tehtud Tallinna Tehnikaülikooli magistrikraadi saavutamiseks. Peamine eesmärk selles projektis oli luua Android raamatukogu, mis suudab suhelda simulatsiooni eesmärgil juhtimissüsteemi või arvutiga.

Ülesanne selle töö jaoks andis Control Systems Research Laboratory (A - Lab). Töö teostati nende poolt pakutud laboris. Programmeerimine tehti Android Studio ja MATLAB R2015b süsteemidega.

Juhendajad olid Leo Teder Mehaanikateaduskonnast ja Aleksei Tepljakov Control Systems Research Laboratory-st.

Tahaksin tänada oma pere ja sõpru toetuse eest projekti kallal töötamise ajal. Suur tänu ka A - Labile kes pakkus võimaluse töötada hästi varustatud laboris ja kelle juhendajaid olid abiks töö tegemisel.

4 INTRODUCTION

The purpose of this work is to research and develop methods in order to interface with MATLAB/Simulink for an Android application running on a tablet or on a mobile phone. The expected result is a working libraries that can be used to create Android applications capable of receiving data from MATLAB as well as sending data back for control purposes and visualizing the result of the simulation or the output of the actual physical device. The communication between devices should be established over the air via a Wi-Fi network. The basic dataflow diagram is displayed on Figure. 2.1. Application should be easy to use without need for training, with easy and intuitive user interface.

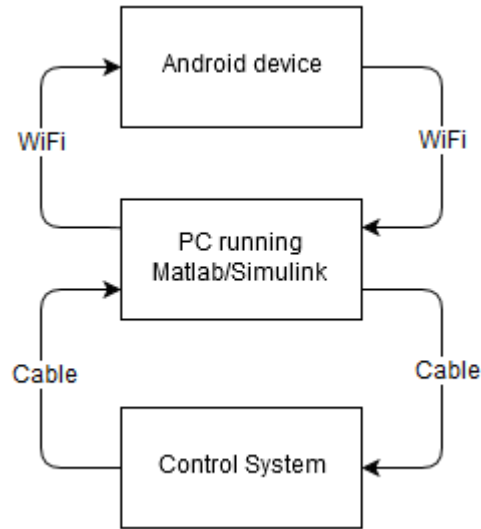


Figure 4.1 Basic data flow

The problem Control Systems Research Laboratory, Alpha Control Lab (A-lab) [39], a research laboratory within the Department of Computer Control at Tallinn University of Technology, is facing is the limited number of same devices. The device list includes, but is not limited to, the Magnetic Levitation System and 3D crane. The Magnetic Levitation System's typical teaching applications are [20]:

- System identification
- SISO, MISO, BISO controllers design

- Intelligent/Adaptive control
- Frequency analysis
- Nonlinear control
- Hardware-in-the loop
- Real-Time control
- Closed loop PID control

While the teaching applications are not few, the available devices for students is limited to one. Same applies to 3D crane. The processes are simulated in MATLAB/Simulink, but having an ability to see the results and control the system from device which is presented in the pocket of more the 80% of mobile user's population [22] would make teaching process more accessible.

A-lab works in different directions to bring control systems available to all people interested in them. One direction is virtual reality, there are several projects developed for Oculus Rift [23]. The finished product will give users ability to see the visualized simulation in virtual reality and control it using joystick or computer keyboard and mouse. These projects make it easier to use devices virtually, but are not as cheap as mobile phone applications would be. For each person there has to an Oculus Rift device and personal computer equipped with high end video card. Oculus Rift's price start from 699 euros and the video card, the one used at this moment in a-lab for development is GeForce GTX 980, from 399 euros. This price get higher if considering a class of twenty students.

Students have developer similar applications, like the ones created during this thesis work, in the past and the results have been promising and furthermore development is always in the consideration.

5 PREVIOUS WORK

Currently the result of a simulation can be animated in MATLAB, the Simulink model of the Magnetic Levitation System is equipped with an animation block. When the simulation updates after each sample time. The position of ball is updated accordingly and screen shows ball velocity and the coil current (Figure 4.1). This feature of Magnetic Levitation System is however never used.

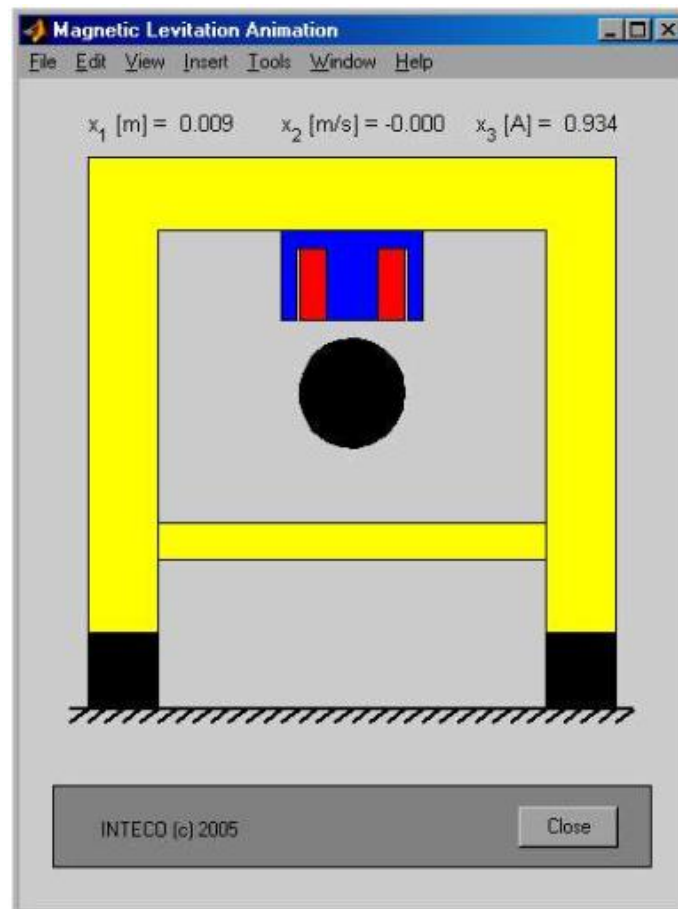


Figure 5.1 MLS animation in Simulink

For 3D crane system a control application was created in 2013. The application runs on tablet devices running on iOS operating system, known as iPad. The application only supports landscape mode. The crane is controlled using touch gestures as well as using manual input of control values [21]. The interface is shown on Figure 4.2.



Figure 5.2 iPad app UI

Similar application for Android devices was developed in past for inverted pendulum. The application was created only for specific screen resolutions and it lacks support for modern devices with higher resolutions. Application runs in only landscape mode. Application was created using processing software sketchbook [31]. As the application is created in non-native environment it is not possible to reuse the code and functionality in other native Android applications.

6 LITERATURE STUDY

6.1 Android

Android OS is free and open source platform, based on Linux kernel, primarily for touchscreen mobile devices and tablets. Modified versions on Android OS runs on televisions, cars, smart-watches, game consoles, photo cameras and notebooks. Android OS is primarily developed by Google. Android OS is released with Apache Software License (ASL). ASL means that the code under this license can be used in closed source products. Because of this other companies such as Samsung, HTC, ASUS and others have modified the operating system and are using it for their devices. This fact lead to wide spread and popularity of the devices running under the Android operating system. The wide variety of software on device with different price tags brought Android operating system to position where it has largest install base on mobile devices and tablets since the fourth quarter of 2010.

Main user interface in devices running on Android OS is direct manipulation, using touch inputs like tapping, swiping and pinching to manipulate on screen objects. Beside touch inputs Android input subsystem supports many different device classes, including keyboard, joystick, trackball and mouse. These input devices connect to Android device with USB or Bluetooth connection [32].

6.2 Application model: applications, tasks, thread and processes

In Android application, there are three important terms to consider:

- Android package
- Task
- Process

In Android OS there is more fluent coherence between process, executable image and the application, unlike other operating systems [5].

The Android package is a file that contains the source and resources needed for the application. Android package is how Android applications are distributed. The installation process is done using these packages. After installing the app, an icon is created for user. [33]

Clicking the icon of app will result in opening a new window. The user might think that they are dealing with application, but actually it's a task. An Android package contains several activities, one of them is launcher or the top level entry point activity. When the app is launched a task is created for that activity. Every other activity started within the app will be managed by the same task. An activity can also be started in new task, using the flag `Intent.FLAG_ACTIVITY_NEW_TASK`.

A process is something that basic users are not aware of. By default all the source code in a package is run in low-level Linux kernel. Android might decide to shut down a process at some point if free memory get low and other, higher priority processes need them [5].

6.3 Android application

An Android app is a software application running on the Android platform. Because the Android platform is built for mobile devices, a typical Android app is designed for a smartphone or a tablet running on the Android OS [34]. Applications are usually developed in Java programming language using the Android software development kit (SDK), but other development environments are also available.

For this work, the applications were developed using Java programming language and Android SDK. As development environment Android Studio 2.1 was used [28]. Android Studio is the official IDE for Android app development, based on IntelliJ IDEA. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhances app development productivity.

The main interaction between user and Android app happens with activities. An activity represents a single screen with a user interface. All user interface elements in an activity are built using View and ViewGroup objects. A View is an object that draws something on the screen that

the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface [35].

Android application package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware. To make an APK file, a program for Android is first compiled, and then all of its parts are packaged into one file. An APK file contains all of that program's code, resources, assets, certificates, and manifest file [36]. Users can get Android applications from Google Play app store or from third party app stores, like Amazon Appstore.

6.4 Android software development kit (SDK)

The Android SDK includes a variety of tools that help developers develop mobile applications for the Android platform. SDK tools are platform independent and are required no matter for which Android platform application is developed for. The SDK is free and can be downloaded from Android developers' web page.

6.5 Challenges in Android app development

The process of development is complex due to barriers such as: screen resolution, device model and device capabilities. Hardware performances vary between devices. Factors such as Power, Memory, and Graphical User Interface (GUI) affect the performance of an application. During application development the developer must be very aware and anticipate these concerns so that the code is written in such a way that the resource utilization in system is balanced and distributed optimally to all the required applications.

Screen resolution is crucially important part in application. Device manufacturers support wide range of resolutions in order to address needs of every user segment.

A developer's application development approach should be future centric and precise in the requirements. When applications are designed, developers should consider aspects such as scalability, usability and coding standards, in order to enhance the quality of the application.

7 MATLAB

MATLAB (“matrix laboratory”) is a multi-paradigm numerical computing environment and fourth-generation programming language. A proprietary programming language developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python [37].

The control systems Android libraries was created for are modeled using special tool Simulink, which is conjunction with MATLAB.

7.1 Simulink

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems.

Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis [38].

8 SYSTEM DESCRIPTION

8.1 3D crane

The three-dimensional model of industrial crane is a highly nonlinear MIMO system equipped with a dedicated system of sensors unique 2D angle measuring unit. The system is fully integrated with MATLAB and Simulink and operates in real-time. The software enables rapid prototyping of real-time control algorithms. The C-code writing is not required. 3D Crane is delivered with the library of basic controllers. The model has three control DC motors and five position measuring wheel encoders. An example of the 3D P controller is shown. The Real Time Workshop and Build Model items chosen from the pull-down menus execute the RTW routines. As a result one obtains the executable file which operates as a real-time controller after connecting it to Windows Target [29].



Figure 8.1 3D crane

The 3D crane setup (Figure 7.2) consists of a payload hanging on a pendulum-like lift-line wound by a motor mounted on a cart.

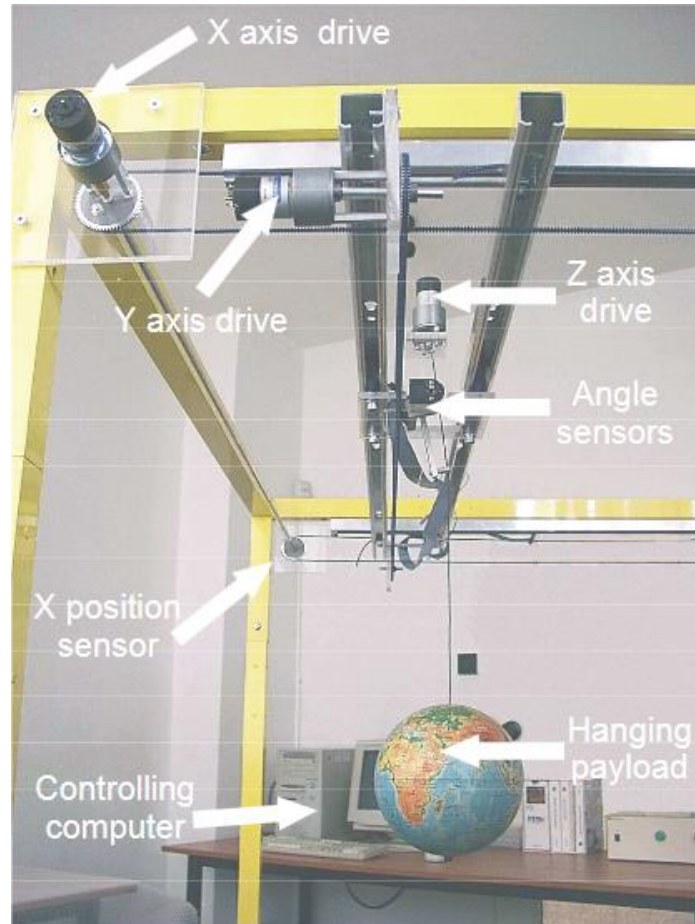


Figure 8.2 3D crane setup [25]

The payload is lowered and lifted in the z direction. Both the rail and cart are capable of horizontal motion in the x direction. The cart is capable of horizontal motion along the rail in the y direction. Therefore the payload attached to the end of the lift line can move freely in three dimension and it driven by three DC motors.

The power interface amplifies the control signals, which are transmitted from the pc to the Dc motors. It also converts the encoders pulse signals to the digital 1-bitform to be read in PC.

To communicate with device the PC has to be equipped with RT-DAC/PCI multipurpose digital I/O board, which communicates with the power interface board. The while logic necessary to activate and read the encoder signals and to generate the appropriate sequence of the pulses of PWN to control the DC motors is configure in the Xilinx chip of RT-DAC/PCI board. Al

functions of the board are accessed from the 3D crane toolbox, which operates directly in the MATLAB/Simulink environment [25].

8.2 Magnetic levitation system

The Magnetic Levitation System (MLS) is a nonlinear, open-loop, unstable time varying frictionless dynamical system. The basic principle of MLS operation is to apply the voltage to an electromagnet to keep a ferromagnetic sphere levitated. Moreover, the sphere can follow a desired position value varying in time. The coil current is measured to explore identification and multi loop or nonlinear control strategies. To levitate the sphere a real-time controller is required. The equilibrium stage of two forces (the gravitational and electromagnetic) is maintained by the controller to keep the sphere in a desired distance from the magnet [19].



Figure 8.3 Magnetic Levitation System

The Magnetic Levitation System is an ideal tool for studying and demonstrating magnetic levitation. Which is a classic control problem used in wide range of practical applications, for example, transportation – magnetic levitated trains. [41]

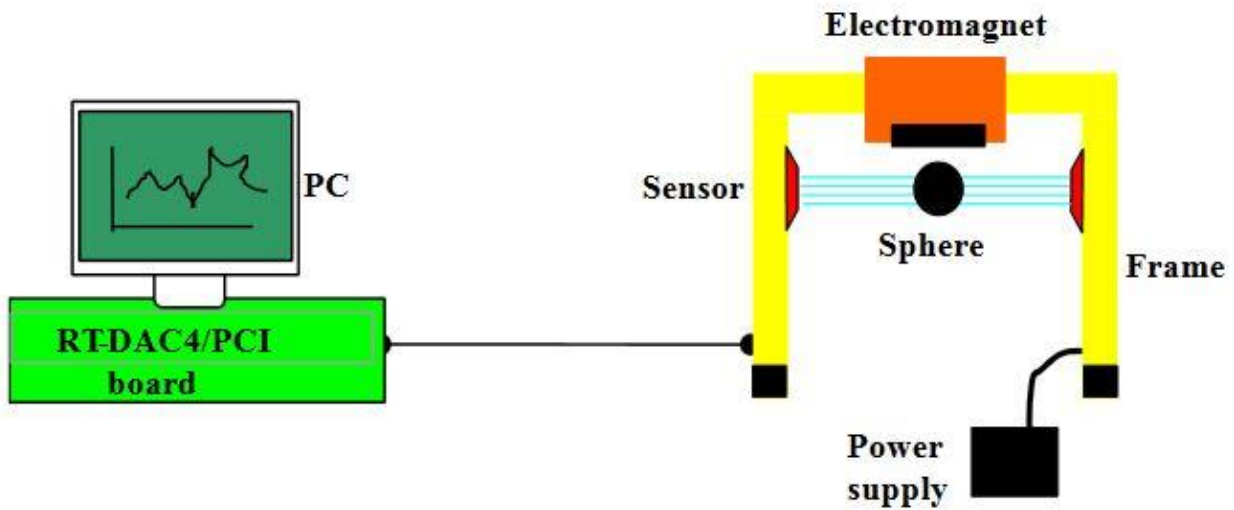


Figure 8.4 MLS laboratory setup [20]

The hardware of MLS consists of:

- Aluminum construction
- Electromagnet
- Ferromagnetic object
- Position sensor
- Current sensor
- Power interface

To operate the system it needs power supply, interface to a PC and dedicated RTDAC4/PCI I/O board configured in the Xilinx technology. The software runs in real time under Microsoft Windows 98/NT/200/XP using MATLAB 6.5 with RTW and RTWT toolboxes.

The control software for the MLS is included in the MLS toolbox, which uses the RTWT and RTW toolboxes from MATLAB.

MLS toolbox is a collection of M-functions, MDL-models and C-code DLL-files that extends the MATLAB environment and solves MLS modelling, design and control problems [20].

9 DEVELOPMENT

9.1 Communication between devices

9.1.1 User datagram protocol

User datagram protocol (UDP) uses simple connectionless transmission model with a minimum protocol mechanism. Connectionless communication referred to as CL-mode communication, is a data transmission method used in packet switching networks by which each data unit is individually addressed and routed based on information carried in each unit. Under connectionless communication between two network end points, a message can be sent from one end point to another without prior arrangement.

With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. As application developed in this thesis are time-sensitive applications, UDP used because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system [1].

9.1.2 UDP in Android

In order to send and receive UDP datagrams in application developed for Android OS, a class DatagramSocket has to be used [2]. This class implements a UDP socket for sending and receiving DatagramPacket [3]. DatagramPacket is class that represents a datagram packet which contains data either to be sent or received through a DatagramSocket. It holds additional information such as its source or destination host. A DatagramSocket object can be used for both endpoints of a connection for a packet delivery service.

DatagramPacket uses IP address and port of machine that is listening for receiving the data. And for receiving it listens to specified port number. The port numbers for sending and receiving has to be different.

9.1.3 Sending data from Android

The need for sending data to Matlab arises only when a specific action is performed on Android device, for example changing cranes position. As sending is not continuous process, for it as solution an AsyncTask was implemented [4]. When an application is launched, the system creates a thread of execution for the application, called "main." This thread is very important because it is in charge of dispatching events to the appropriate user interface elements, including drawing events. It is also the thread in which application interacts with user inputs. As such, the main thread is also called the UI thread [5]. All time consuming operations have to be run in different thread, in order not to block main one and make user interface freeze while waiting, for example data to be send.

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers. AsyncTask is an ideal solution for short operations (a few seconds at the most.) like ours.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread (the main thread).

When the event happens and data has to be send to Matlab application checks if there is and data sending AsyncTask running, if there is it kills it, as newer updated data is more important to be deliver to simulation running on Matlab. After assuring that there is no AsyncTask running, previous was killed or finished, a new AsyncTask is created. The data, which has to be send, is passed to it and executed.

In executing part the data, which consists of decimal numbers in case of 3D crane these numbers are position on the x, y and z axis, is serialized, meaning that the data is converted to byte array, as DatagramPacket sends data with byte array format. A DatagramPacket is created, which needs the byte array, storing the data, the length of this byte array, the IP address on machine which runs the Matlab simulation and port on which Matlab is listening and receives data. The IP address of machine and the port number are defined in the application. Then the DatagramSocket is created with the DatagramPacket created in previous step. And an attempt is made to send the

packet. If packet is send information about it is returned to the UI thread and it can be updated accordingly. The sending process can fail because of several reasons: the internet connection is not available on Android device, the IP address and/or the port number of receiving machine are not correct, the receiving machine has no internet connection or blocks incoming messages from the specified port number. Or the two device, the Android one and a pc running Matlab, are not in same local network, which is requirement from Matlab side and will be discussed in part of this these about Matlab communication.

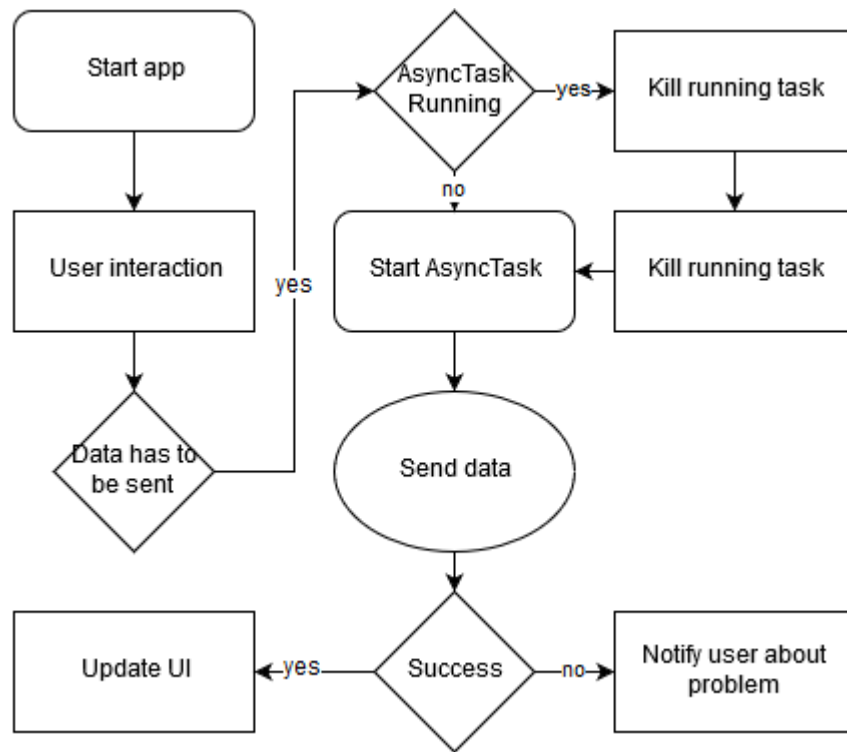


Figure 9.1 Data sending on Android

9.1.4 Receiving data in Android

The approach which was used for sending data cannot be used for receiving the incoming UDP packages from Matlab. As this process is continues process and the data is received non-stop, especially in the magnetic levitation system.

Following the best practices for this type of situation, usage of Android service [6] was decided to be implemented. A Service is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. The main UI thread will start service that will listen and receive data from Matlab. A service is a good way to handle long running network transactions from the background.

To interact with main UI thread, in order to update user interface components, I use a BroadcastReceiver [7]. A broadcast receiver is an Android component which allows developer to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens [8].

In these application LocalBroadCastManager [9] is used for communication between service and activity, which displays and manages user interface. LocalBroadCastManager is subclass of BroadcastReceiver and is used in situations when developer does not want other applications to receive this broadcast and it's more efficient than using system broadcasts.

The service will send broadcasts every time it gets data which is important for user interface, for example in the case of magnetic levitation system this data will be the position of ball. The main activity which implements the user interface will be listening for incoming broadcasts and upon receiving one will act accordingly, which in most cases is to apply the update to user interface elements.

The data is send via broadcast is packed in intent [10]. An intent is an abstract description of an operation to be performed. It can be used with broadcastIntent to send it to any interested BroadcastReceiver components, and `startService(Intent)` to communicate with a background Service. In these applications scope we are using this two methods to send data to service, the data consist of information about the Matlab, to which port the UDP socket has to listen in the service, and to communicate with UI thread, the received data from Matlab.

In the activity which manages user interface we listen to incoming broadcast, which will be send from service. Upon receiving one we unpack the data stored in intent object and apply updates to user interface.

The receiving service works according to the logic: after executing start command from activity the local Boolean variable in service becomes true and indicates that the service is running. While this local variable is true, meaning that the service has to listen incoming packages from Matlab, service create a byte array to store incoming data, creates socket with port number, which was received via intent, and waits for incoming DatagramPacket. If DatagramPacket arrives the data is converted to an integer in the case of magnetic levitation system or to the coordinates of crane in the case of 3D crane system. After the data is packed in an outgoing intent and is sent to system with broadcast. The main UI activity is waiting for this type of broadcasts and upon receiving one, reads data from intent and updates user interface.

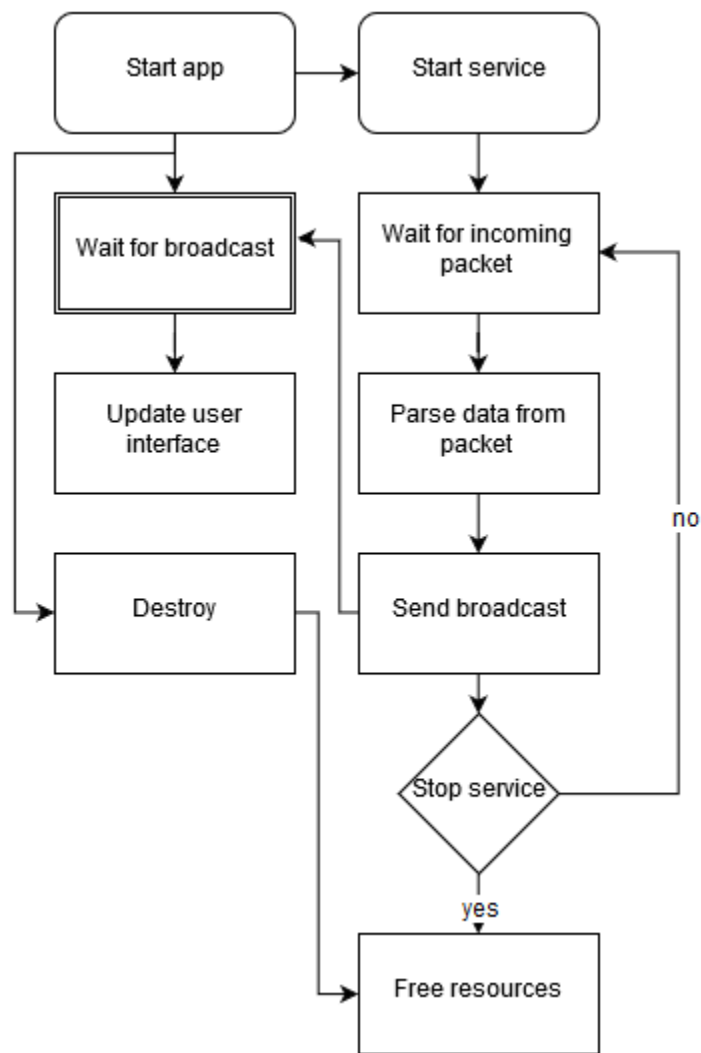


Figure 9.2 Data receiving in Android

9.1.5 Communication in Matlab

Packet Input/output blocks in MATLAB/Simulink are used to transfer data through UDP communication protocol using binary encoding. The block sends data from one computer's one UDP port to another device. And receives the incoming data on different UDP port. As data acquisition board, which is packet input/outputs' parameter, Standard Devices UDP Protocol [9010h] is used. Board's configuration has three parameters: the IP address of remote device, receiving and sending UDP ports (Figure 8.3). This board is used in both, input and output blocks.

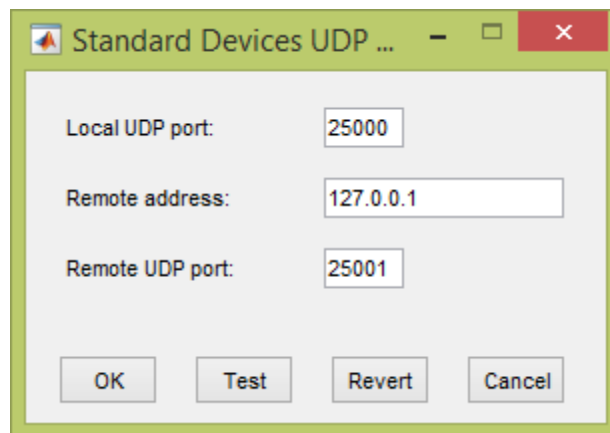


Figure 9.3 Board setup

9.1.6 Sending data from Matlab

The data which has to be sent from MATLAB/Simulink is passed to the packet output block (Figure 8.4). In case of 3D crane, there are five values to be sent:

- X coordinate of cranes position
- Y coordinate of cranes position
- Z coordinate of cranes position
- X angle
- Y angle

In case of the MLS passing data to output packet block looks same as shown on Figure 8.4, the difference is that the MLS only send one value for spheres position.

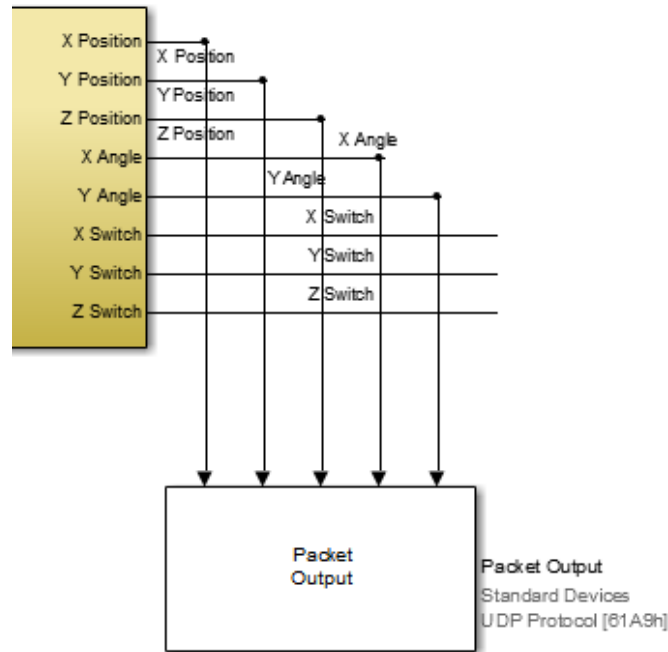


Figure 9.4 Data for output, 3D crane

The block's parameters are displayed on Figure 8.5. The difference between the MLS and 3D crane is output packet size and output packet field data type. In case of the MLS output packet size is eight, as one double value takes eight bits of memory, and in case of 3D crane it is forty. And output packet field data types: `'1*double'` and `{'1*double', '1*double', '1*double', '1*double', '1*double'}`, meaning that we are sending one value of double and five values of double. In case of 3D crane block will have five inputs, one for each output value. For the MLS the block will have only one input.

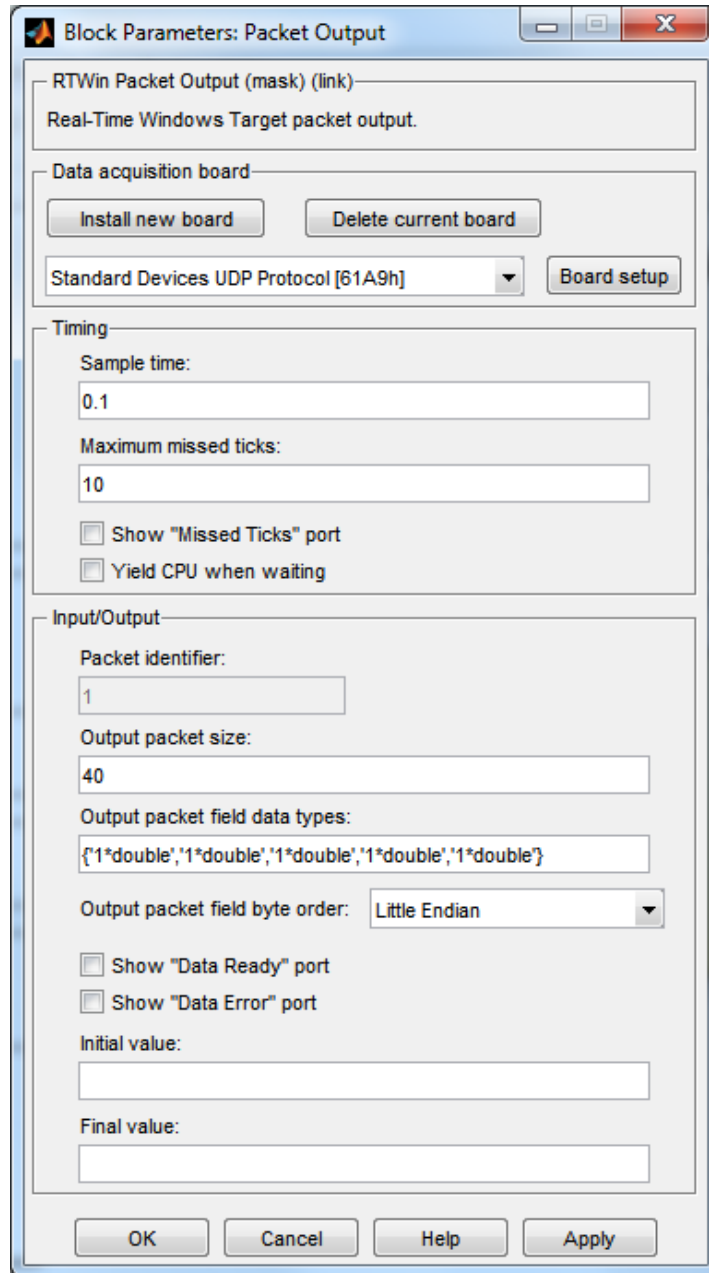


Figure 9.5 Package Output block for 3D crane

9.1.7 Receiving data in Matlab

For sending data Packet input block is used [30]. For this block the same board setup is used like one in Packet output block (Figure 8.3). Packet input block's parameters are shown on Figure 8.6. The image shows settings for 3D crane, the difference in case for the MLS is the input

packet size, which is eight instead of 3D crane's eighty four, and block output data types, which is '1*double' as the MLS only takes one input.

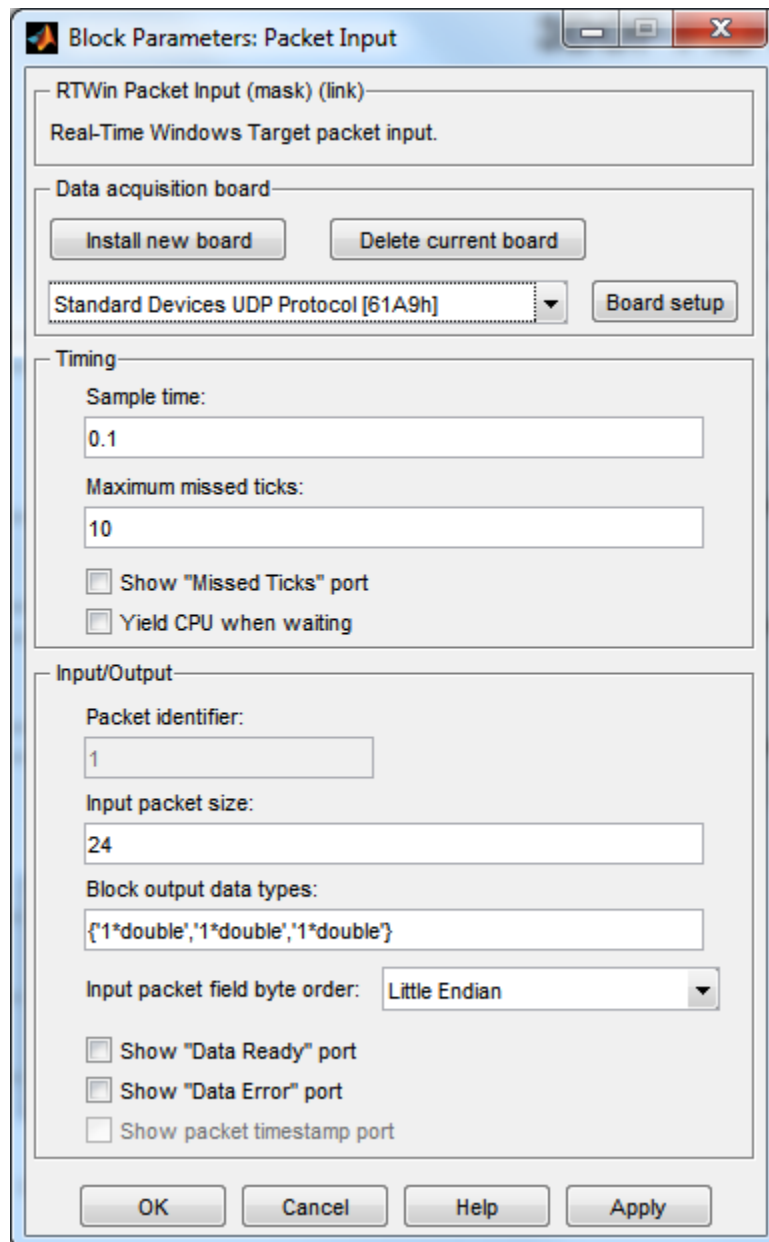


Figure 9.6 Packet input

9.2 3D crane view

In order to show user visualized process of 3D crane and give him/her ability to control the real device by touch interaction with Android device. The CraneView and CraneAngleView classes have been created. The CraneView and the CraneAngleView are custom views, which inherits all the properties from general view class, these properties include and are not limited to: `android:layoutWidth` and `android:layoutHeight`, the two required attributes which define the size of view on the user interface screen, the attribute `android:id`, which is used to assign a name for accessing the view from code.

Eight unique attributes have been create for this CraneView, in order to simplify the process and give customizable control to developers who will implement the CraneView in future. These attributes are displayed in a Table 8.1.

Table 9.1 The custom attributes of the 3D crane view

Attribute name	Value type	Required
lineColor	Color resource	False
circleColor	Color resource	False
borderColor	Color resource	False
axisX	Integer value	True
axisY	Integer value	True
axisZ	Integer value	True
border	Integer value	True
Interval	Integer value	False

First three attributes are to customize the view, set the color of lines connecting the crane to the edges, color of the circle that represents a crane and color of the border which represent the color of border. The next four attributes are used to adjust real life parameters and visual representation of device. These parameters are the size of device in centimeters, `app:axisX`, `app:axisY` and `app:axisZ` are sizes of the width, breadth and height of crane's frame. The border property is the width of the frame of crane device, it's used to make realistic proportions of visual representation.

The last attribute, interval is used to draw the grid on the sides of crane. The interval value represents distance between marks on the grid. Also at the end of border the last mark is drawn. The marks have text, representing the distance in centimeters.

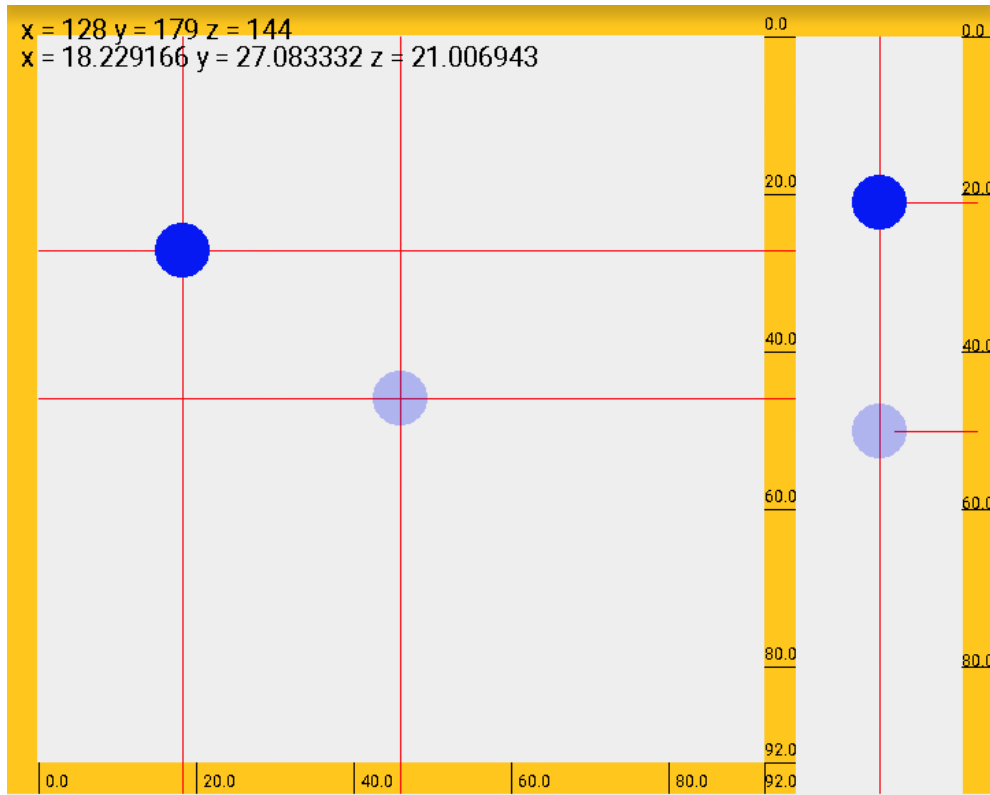


Figure 9.7 Crane view

The Figure 8.8 shows screen of user interface from a demonstration application that implements the crane view, the attributes are set as follows: the color of lines are red, color of crane blue and color of borders yellow. The width and breadth of actual device in a-lab is 0.6 on 0.8 meters and with height of one meter. The border size is four centimeters. The intervals were set to be twenty centimeters long. These attributes were set according to 3D crane device and the result was shown on the phone screen. In this case on Motorola Moto X device, which has screen resolution of 720 x 1280 pixels, one centimeter of real life device is scaled to 5.76 pixels on screen. For debugging purposes there are coordinates written in the upper left corner of the screen, first line writes coordinates of the crane on the screen and second line show coordinates of crane in real device.

Crane view has two representation of crane, one is the user controllable crane, which responds to touches and sends data to MATLAB/Simulink and second one, drawn with transparent color represents simulations or real devices position, its position can only be changed by incoming data about its position.

The left square represent the 3D cranes view from top. Touching and moving the controllable crane on this plane will send real life coordinates as data to Matlab. The right side shows the crane from the front viewpoint, the controllable crane can be moved up and down by touching and moving it with touch inputs. Changing the position will send coordinate data to Matlab.

9.2.1 Drawing the CraneView

Upon creating the CraneView during runtime, application gets the allocated maximum width and height for drawing the view. Proportional to the real world size and available pixel it choses best possible dimension to draw the most realistic view. The front view part which is drawn on the right side and represent the crane in Z axis takes 10% of whole view. After measurements and size calculations are finished, application draws frame's borders, the lines and the ball for both controllable and representable cranes. Then checks settings for distance between interval lines, calculates real life values and according pixel length and draws the grid.

9.2.2 CraneAngleView

The CraneAngleView displays the cranes deviation from vertical line, Z axis. The CraneAngleView's look is shown on Figure 8.9.

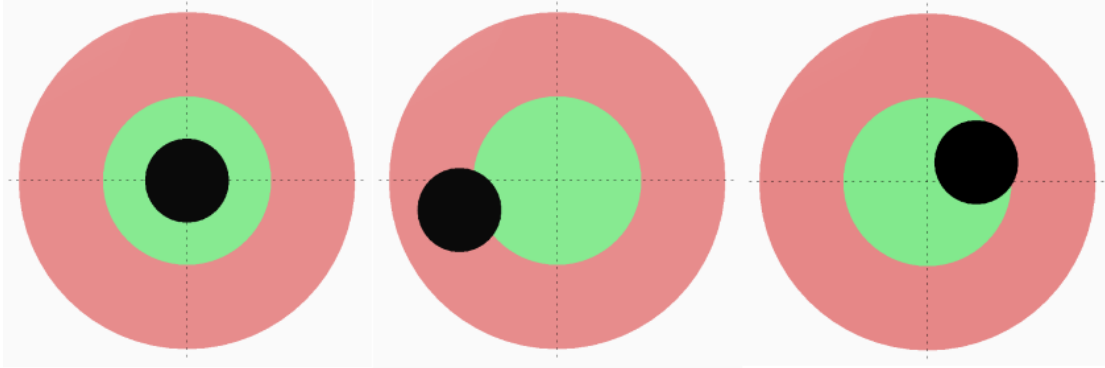


Figure 9.8 CraneAngleView different crane positions

The view has circular design and has two circles, the center one in green color and outer circle with red color. The circle with green color gives visual information that the crane deviates in admissible range. If crane goes outside this green circles and will be on the red background, this will indicate that the cranes swing is dangerously. The crane is represented with black colored circle, which is in the center at the relaxed position, when there is no deviation from vertical line. The view has to intercrossing dashed lines. These lines represent cranes X and Y axis.

Table 9.2 Custom attributes of the 3D crane angle view

Attribute name	Type	Default value
craneColor	Color	#000000
dangerColor	Color	#e68787
safeColor	Color	#82e98d

CraneAngleView has three custom attributes, these attributes are displayed in Table 8.2. Having this attributes gives developer customization privilege and can modify view as pleased. Setting the same color values for `app:dangerColor` and `app:safeColor` will result in combining those two circles and removing distinction between safe and danger areas.

9.2.3 Drawing CraneAngleView

After creating CraneAngleView, application measures available width and height in the layout for the view and gives most suitable, equal width and height. The biggest circle, which is in red

color as default takes 45% of available space as radius and the circle is drawn. The green radius or green circle is third of the radius of red circle. Green circle is drawn with this radius and the center in on the center of the red circle. Two perpendicular dashed lines are drawn afterward, the lines cross each other at the center of circles. After drawing this background, the crane is drawn in the center of all.

9.2.4 User input

When user touches the screen, application checks the position of touch. If touch is inside left part of crane, it means that the crane has to be moved in X and Y axis and moves the crane to the touched point. If user touches and moves finger the crane will follow the finger's movement until it reaches the edge of crane or user removes the finger from the screen. If touch is in the right side of CraneView, meaning user wants to control the position of crane in Z axis, the crane follows fingers movement only up and down no leaving the borders.

When the crane position changes in any direction, the view notifies the activity which implement the CraneView about the change and sends real life values to it. The activity upon catching the position change event start data sending task, described in part about sending data from Android.

9.3 Magnetic levitation system view

9.3.1 Scalable vector image

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics. We considered using scalable vector graphics, because SVG keeps high quality graphics at any resolution. It is zoomable and the image can be zoomed without degradation. These qualities of SVG image would guarantee most realistic look of magnetic levitation system on Android device with any screen resolution [15].

A-lab has created a 3d model of magnetic levitation system in Blender [13]. Blender is a professional free and open-source 3D computer graphics software product used for creating

animated films, visual effects, art, 3D printed models, interactive 3D applications and video games.

Using Blender's add-on FreeStyle [14], the 3d model was exported as SVG image. And colored using free drawing software InkScape [15]. The results of each step are presented on Figure 8.10. From left to right: blender model's low quality render, exported SVG with FreeStyle and colored result using InkScape.

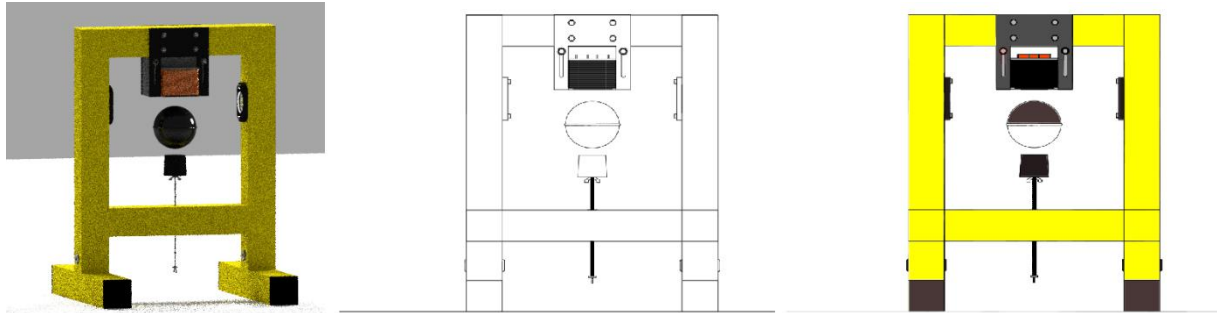


Figure 9.9 MLS model in Blender, SVG generated with FreeStyle, colored SVG image

9.3.2 SVG in Android

Scalable vector graphics images support was added recently to Android applications. And the main advantages using SVG in Android is when application's icon is SVG image, as it reduces size of whole app. Without SVG icons Android app needs five different icons for each type of resolution.

Using SVG image besides the icon is only supported in devices running on Android OS 4.4 or above. To support the devices with OS version below then Android 4.4 all SVG images have to be converted to PNG format [16]. Portable Network Graphics (PNG) is a raster graphics file format that supports lossless data compression [17].

Currently Android devices are grouped in six different screen density groups (Table 8.3) [18]. This group may expand as new technologies come, like it did in past.

Table 9.3 Screen density groups

Name	Density
ldpi (low)	120dpi
mdpi (medium)	160dpi
hdpi (high)	240dpi
xhdpi (extra-high)	320dpi
xxhdpi (extra-extra-high)	480dpi
xxxhdpi (extra-extra-extra-high)	640dpi

Depending on screen density a device running on Android OS will be assigned to one element from this table. During runtime of the app it checks this resources available for its defined screen density and uses those resources. This means that in case of providing five different sized same image will look same on any device. This type of approach alludes us from using partially supported SVG images.

Considering the fact that limitations of SVG images in Android, the complications with image generated using InkScape software, as Android lacks the support of SVG created or modified using this software, and the general recommendation from android developers documentation, that using PNG images will scale lossless on different resolutions. The further development with SVG images has been abandoned. But the ability to import SVG image instead of the PNG is in the application.

The PNG image of magnetic levitation system was created using SVG image which was acquired from Blender model. The image was converted into five different sizes, according to density groups shown in table 8.4, the ldpi (low) was not created as device with this screen are not around anymore, but if app runs on device with ldpi the image will be taken from mdpi and scaled down. The image sizes and their corresponding screen densities are shown in table #.

Table 9.4 Used image resolution

Screen density	Image size
mdpi (medium)	759x1013
hdpi (high)	1139x1519

xhdpi (extra-high)	1519x2026
xxhdpi (extra-extra-high)	2279x3039
xxxhdpi (extra-extra-extra-high)	3038x4052

9.3.3 Drawing the MLSView

Figure 8.11 displays how the Magnetic Levitation System is drawn in application, which implements the MLSView. To draw the magnetic levitation system on the view, a screen resolution corresponding image is taken from the resources, the image which was created from SVG image discussed in Scalable Vector Image part of this thesis. The image is realistic representation of actual device.

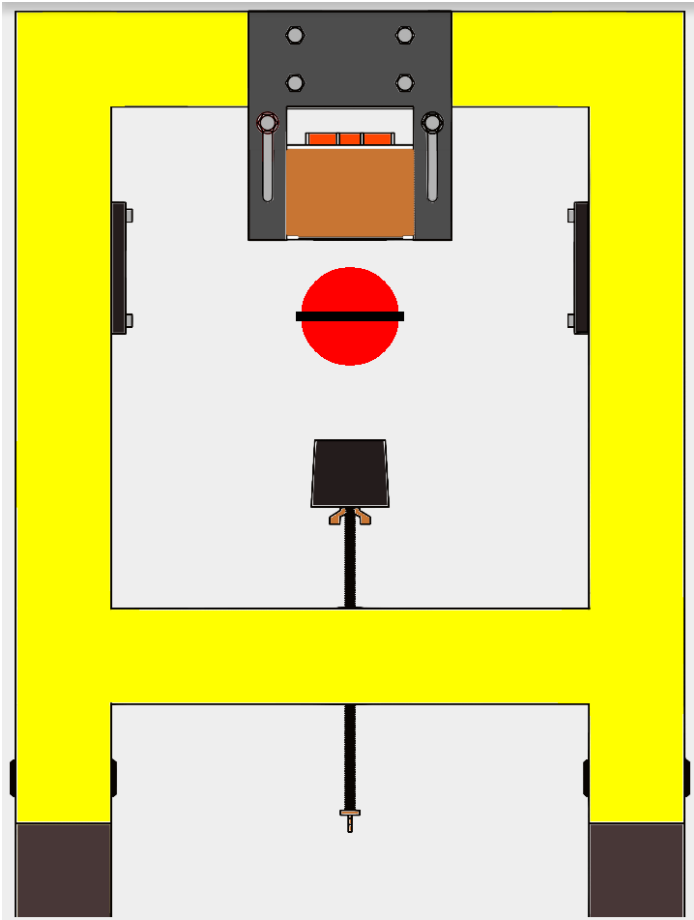


Figure 9.10 MLS View

Above the MLS image is drawn a sphere, which is levitated in simulations and in real device. The sphere is in center in horizontal and sits just above the rubber when the process is not active. The MLSView is listening to changes in sphere's position and upon receiving one acts accordingly and moves the sphere. The event about ball position change is registered in an activity that implements this view and also implement the receiving data service, which was discussed in part about receiving the data in android.

The position of sphere can be changed by user touch input. The position of sphere relative to magnet can be set and controller by MATLAB/Simulink simulation. To set this parameter application uses data send task to pass the parameter to MATLAB/Simulink.

10 MANUAL

10.1 Importing library to project

Both libraries are distributed as android archive library (aar) bundle. The aar bundle is the binary distribution on an Android Library Project [23].

In case of 3D crane library, the aar bundle contains necessary files to draw the crane view, the crane angle view, data receiving service, data sending task class and custom attributes for the views.

In case of MLS library the bundle contains files for drawing the MLS view, data receiving service, data sending task class, custom attributes and images of physical device of magnetic levitation system.

In order to import the library bundled in .aar file the developer has to have a copy of it on his/her machine and an Android Studio software. In Android studio after creating new Android project or on existing one, depending the phase of the app development, developer has to choose next operations: right click on the app root in Android view, then choosing new and module, (Figure 9.1) a new dialog windows opens and there import .JAR/.AAR package has to be selected and after that in new dialog windows a path to desired library's location has to be supplied.

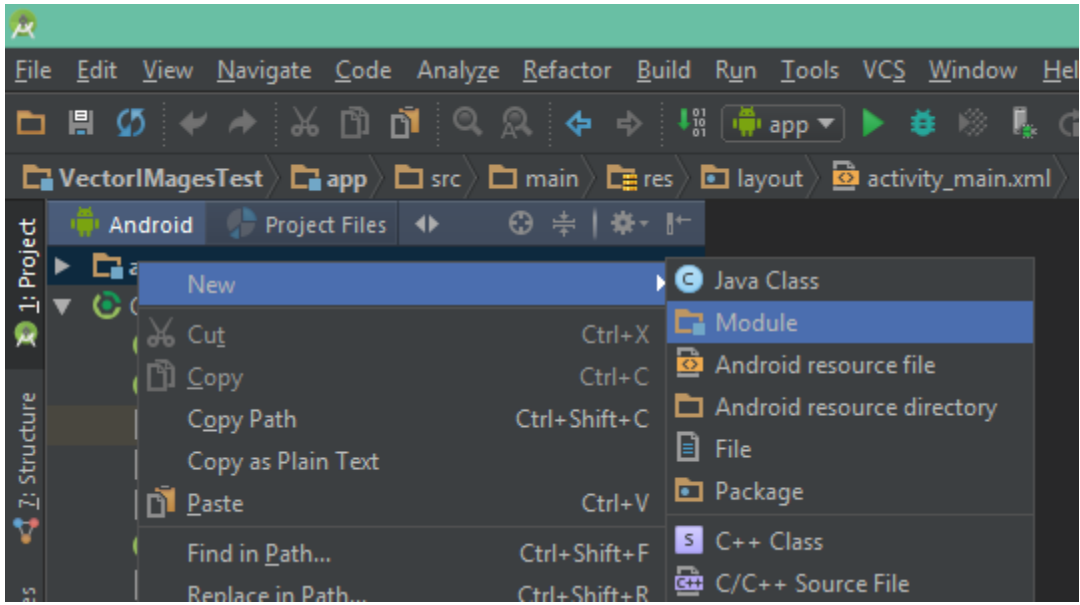


Figure 10.1 Add module

After adding a module to application, it has to be added as module dependency. To do so, in file menu we choose Project Structure... or click short-key Ctrl+Alt+Shift+s and in the apps dependencies we add new module dependency and choose the library we added in previous step. (Figure 9.2)

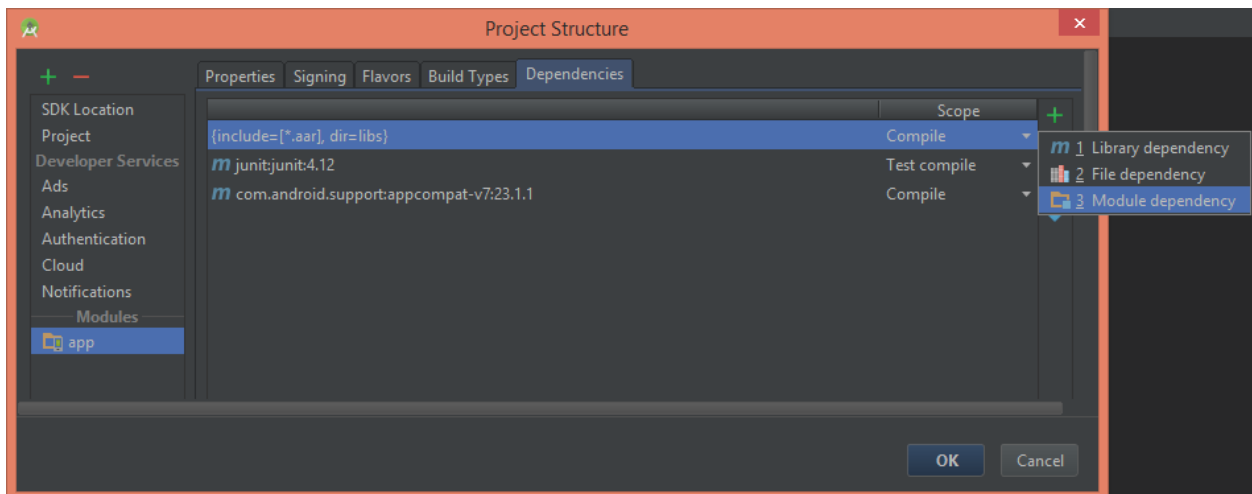


Figure 10.2 Add module dependency

After these steps, an Android application has support of the library and can be imported in java classes and used as needed.

10.2 Using 3D crane library

To use 3D crane library's functionality in an Android app, 3d_crane.aar has to be added as dependency to the project, according to instructions explained in importing library to project part of this paper. All classes: the crane view `CraneView.java`, the crane angle view `CraneAngleView.java`, the data sending task: `SendDataTask.java` and data receiving service `MatlabConnection.class`, are in `ee.a_lab.a3dcranelibrary` package. These classes contain all necessary methods and resources to fulfill required functionality.

A Java method, or may be called function, is a collection of statements that are grouped together to perform a specific reusable operation.

A class is a blueprint or a template for creating different objects which defines its properties and behaviors. Java class objects exhibit the properties and behaviors defined by its class. A class can contain fields and methods to describe the behavior of an object [26].

A Java package is a technique for organizing Java classes into namespaces similar to the modules of Modula, providing modular programming in Java. Java packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually [27].

To uses this classes they first have to be imported in desired Java class, this can be class which takes responsibility for user interface or a class with purpose to extend the functionality of the library.

```
import ee.a_lab.a3dcranelibrary.CraneView;
import ee.a_lab.a3dcranelibrary.CraneAngleView;
import ee.a_lab.a3dcranelibrary.SendDataTask;
import ee.a_lab.a3dcranelibrary.MatlabConnection;
```

Figure 10.3 Inputs

Other than base methods of Android's `AsyncTask` class, the `SendDataTask` class has additional four public methods:

- `boolean setIp(String ip)`
- `String getId()`

- `boolean setPort(int port)`
- `int getPort()`

`setIp()` method is used to set the IP address parameter of the receiving machine, which runs MATLAB/Simulink simulation. The method takes string type value as parameter, which represent the IP address written as string, for example “127.0.0.1”. The method will return true if IP string is correct and if successfully sets it, it will return false if there is any error and will print the error description in logcat window.

`getIp()` method will return the string representation of IP address which is used in the moment of calling the method.

`setPort()` method is used to set the port number on which the receiving machine is listening, the method takes one integer type parameter. If the port number is correct, meaning it in the range of available port numbers, the method will set it and return true, otherwise it will return false and write description about error in the logcat window.

`getPort()` method will return the port number which is used at the moment of method call.

To execute the `SendDataTask`, after creating its object, developer has to set IP address and port number using methods: `setIp(String ip)` and `setPort(int port)`. To run the task developer has to call `execute()` method on `SendDataTask` object and pass three float type parameters one each for x, y and z coordinates. About success or failure of executing the task, the developer will be informed with message in logcat window.

In order to show crane view on user interface, developer has to add `ee.a_lab.a3dcranelibrary.CraneView` to their layout file. Other than inherited attributes from view class, seven unique attributes can be set to it, this attributes are discussed in topic about 3D crane view. Four attributes: `axisX`, `axisY`, `axisZ` and `border` are required and not assigning them will result applications crash during runtime, this is caused intentional to prevent unexpected behavior. If this message about required fields will be written in logcat for developer to debug and correct.

`CraneView` class additionally to inherited methods, has following two method: `public void setPosition(double cranePositionX, double cranePositionY, double`

`cranePositionZ`), which is used to update cranes position on user interface. The method takes three double type parameters, each for x, y and z coordinates. This parameters are real life values and are converted to reflect corresponding pixel values for screen. Second one is `setRealPosition(double cranePositionX, double cranePositionY, double cranePositionZ)`, this method work same as previous one, but is used to change transparent, the real cranes position. The class has two helper methods for converting centimeter to corresponding length in pixel and opposite, converting length in pixel to length in centimeters. These methods are

- `int cmToPixel(double lengthInCm, int axis)`
- `double pixelToCm(int lengthInPixels, int axis)`

Where the first parameter is value we want to convert and second one on which axis are we converting this value for, these axis is one of the following constant values:

- `public static final int AXIS_X = 0;`
- `public static final int AXIS_Y = 1;`
- `public static final int AXIS_Z = 2;`

`CraneView` class has custom event `OnPositionChangeEvent(double x, double y, double z)` which is called every time user changes the cranes position. The class which will use crane view also has to implement `OnCranePositionChangeListener` with events implementation. In `OnPositionChangeEvent` method and will receive cranes coordinates in space, developer has to implement data sending task or any other functionality, depended on requirements of the application.

To add `CraneAngleView` on user interface, developer has to use `ee.a_lab.a3dcranelibrary.CraneAngleView` in layout file. Color for danger circle, the color for background circle and color for crane circle can be set from layout file using attributes mentioned in topic `CraneAngleView`. View will use default colors if these are not supplied. `CraneAngleView` does not register user inputs as it has only displaying function.

To change cranes position in `CraneAngleView`, a method `setCranePosition(int angleVerical, int angleHorizontal, double height)` has to be used. The

`angleVertical` is the angle value of the cranes deviation from the Y axis. The `angleHorizontal` is angle deviation from X axis. The height parameter is the length of cranes position in Z axis. These parameters are used to calculate on how big is deviation in centimeters. To calculate cranes deviated position on the plane the formulas are used (1) and (2).

$$x = \cos(\text{angleVertical}) * \text{height} \quad (1)$$

Where x – position of crane on X axis

angleVertical – Deviation on x axis of crane

height – The length of cranes position on Z axis

$$y = \cos(\text{angleHorizontal}) * \text{height} \quad (2)$$

Where y – position of crane on Y axis

angleHorizontal – Deviation on Y axis of crane

height – The length of cranes position on Z axis

The values of X and Y is converted to pixel values and the position of crane is redrawn.

To receive data from MATLAB/Simulink, the developer has to use class `ee.a_lab.a3dcranelibrary.MatlabConnection`. This class is a service and to start it developer has to create and pass port number with intent, the key of intent is in `MatlabConnection` class: `MatlabConnection.PORT_NUMBER`. Code example:

```
Intent    serviceIntent    =    new    Intent(this,
MatlabConnection.class);

serviceIntent.putExtra(MatlabConnection.PORT_NUMBER,
25001);

startService(serviceIntent);
```

To receive data from service into application developer has to create Broadcast listener, using intent filter and use `MatlabConnection.UDP_BROADCAST` static value as its parameter. Broadcast receiver has to registered and unregistered according to applications needs. The receiver will receive data packet in intent. The intent contains three float values for cranes coordinates, Table 9.1 shows key value pairs.

Table 10.1 Constant values

Key	Value
<code>MatlabConnection.INTENT_DATA_X</code>	X coordinate
<code>MatlabConnection.INTENT_DATA_Y</code>	Y coordinate
<code>MatlabConnection.INTENT_DATA_Z</code>	Z coordinate
<code>MatlabConnection.INTENT_DATA_X_ANGLE</code>	X angle
<code>MatlabConnection.INTENT_DATA_Y_ANGLE</code>	Y angle

This values have to be used with `CraneView`'s `setPosition(x, y, z)` method to update user interface, data also can used for other purposes, depending on applications needs.

10.3 Using MLS library

For application to use capabilities of MLS library, it has to have dependency on the `mls_library`. Following steps described in section 9.1. Importing library to Android will make library available in application.

The send task work in the same manner as described in section 9.2. Using 3D Crane library. Only difference is parameters passed in execute method. Instead of three float values, data send task for MLS takes one parameter with type of float, which is the distance between sphere and magnet.

For user interface view class `ee.a_lab.mls_library.MLSView` has to be used, by adding `ee.a_lab.mls_library` to the layout file and supplying necessary attributes. `MLSView` as

custom attributes has: distance between magnet and rubber holder and diameter of sphere. These parameters are integer type and are measured in centimeters.

To receive data from Matlab/Simulink, the developer has to use class `ee.a_lab.mls_library.ReceiveDataService`. This class, similar to one used in 3D crane library, is a service and to start it developer has to create and pass port number with intent, the key of intent is in `ReceiveDataService` class: `ReceiveDataService.PORT_NUMBER`. The code example:

```
Intent    serviceIntent    =    new    Intent(this,
ReceiveDataService.class);

serviceIntent.putExtra(ReceiveDataService.PORT_NUMBER,
25000);

startService(serviceIntent);
```

In order to get received data from service into application developer has to create Broadcast listener, using intent filter and use `ReceiveDataService.UDP_BROADCAST` static value as its parameter. Broadcast receiver has to registered and unregistered according to applications needs. The receiver will receive data packet in intent. The intent contains one float value for sphere's position, the key, to get value of position from intent, is saved in static field `ReceiveDataService.INTENT_DATA`.

11 DEMONSTRATION APPLICATIONS

To demonstrate the ability and simplicity of the implementing process of the libraries in new Android applications, two applications was created. The Magnetic Levitation System app and 3D Crane app. Both applications use the functionality of the libraries and are customized to meet specific requirements.

11.1 Magnetic Levitation System app

The demonstration Android application was created for the Magnetic Levitation System, as the system has only one control parameter no extra control user interface elements were added to the UI rather than the MLSView. The application works in landscape and portrait modes. The user interface of the portrait mode is shown on the Figure 10.1. The landscape looks same as portrait but the MLS is smaller as the height of the available space decreases in landscape mode.

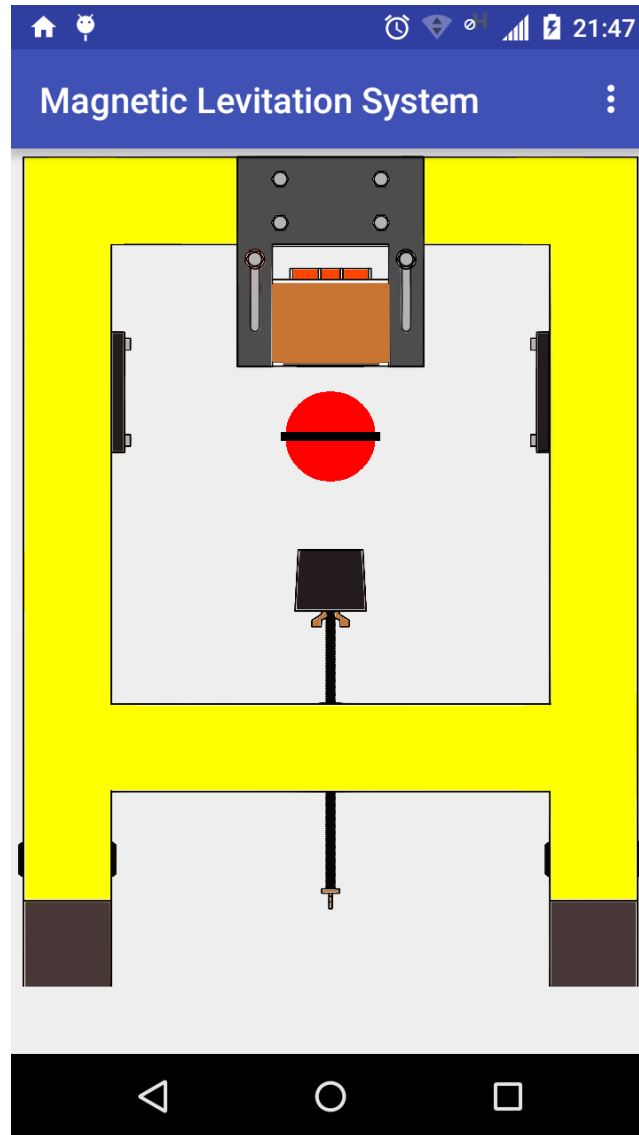


Figure 11.1 User interface from MLS app

The application has menu, with only one option which is settings. Clicking settings will open new windows with three field that can be set by the user. (Figure 10.2) This fields are the IP address of the machine running the MATLAB/Simulink simulation, the receiving and sending UDP ports. These value are retrieved by application during runtime and are passed to data send task and data receiving service. For sending task the methods `setIp()` and `setPort()` are used, the usage of these methods are described in topic about Using 3D crane library. Same topic describes how to pass receiving UDP port to the service.

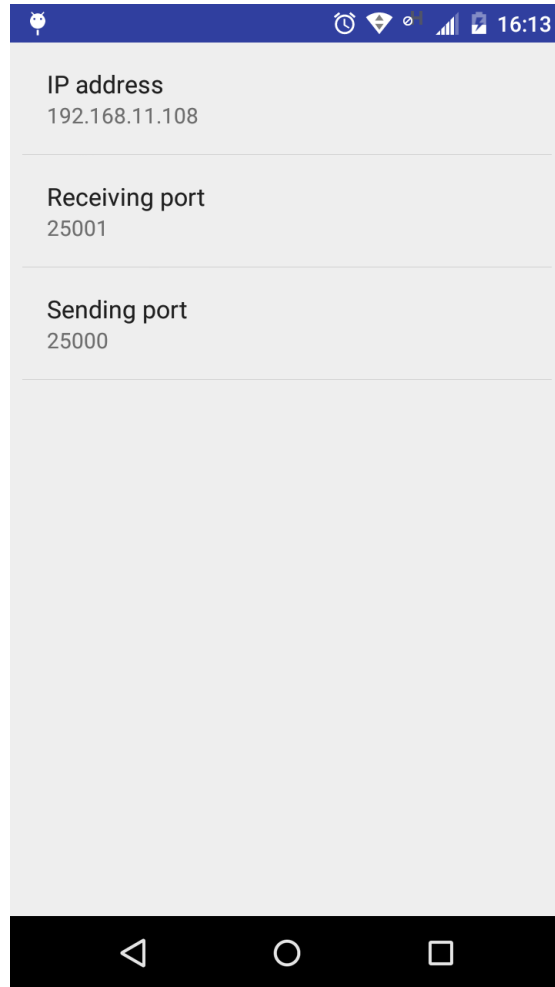


Figure 11.2 Settings screen from demonstration apps3D crane app

The 3D crane demonstration application has two different user interfaces, as the same layout would not be appropriate for the landscape and portrait mode. (Figures 10.3, 10.4) Landscape is mode when phone is in horizontal position and portrait mode is when phone is in vertical position. This two layouts have CraneView and CraneAngleView, also there are tree fields for numeric input and a button with text set. This input field will display the cranes position and also can be modified by clicking and entering desired values using virtual keyboard. Clicking the button set will send the data to Matlab/Simulink and the cranes position will change.

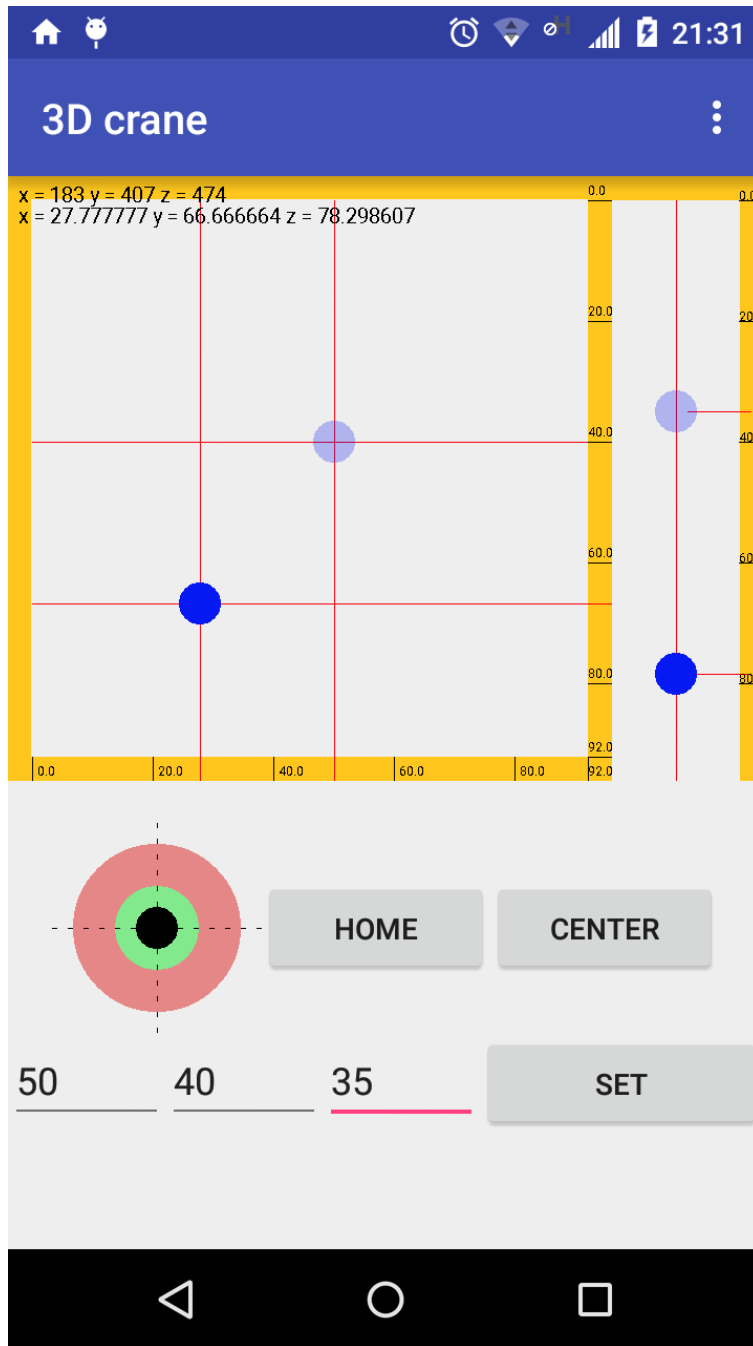


Figure 11.3 3D Crane app in portrait mode

The user interface in 3D Crane app also has two additional buttons, the home button and the center button. Clicking the home button will move the cranes X and Y position in the coordinates of 0, 0, this operation is done using a3dcranelibrary's method `setPosition()`, which is in CraneView class and the usage of this method is described in part about using 3D crane library in android project. After setting cranes position on view, the data is sent to MATLAB/Simulink

using a3dcranelibrary's SendDataTask, using functionality of this class is described in the same topic. The simulation will be updated in MATLAB/Simulink and real device as well. Clicking the center button will move crane in the middle of X and Y axis and will do the same on the MATLAB/Simulink simulation and/or real crane device. The method calls are same as in home buttons description.

These buttons are not presented on the landscape mode as the screen of testing device is not giving the enough space to put an extra user interface elements.

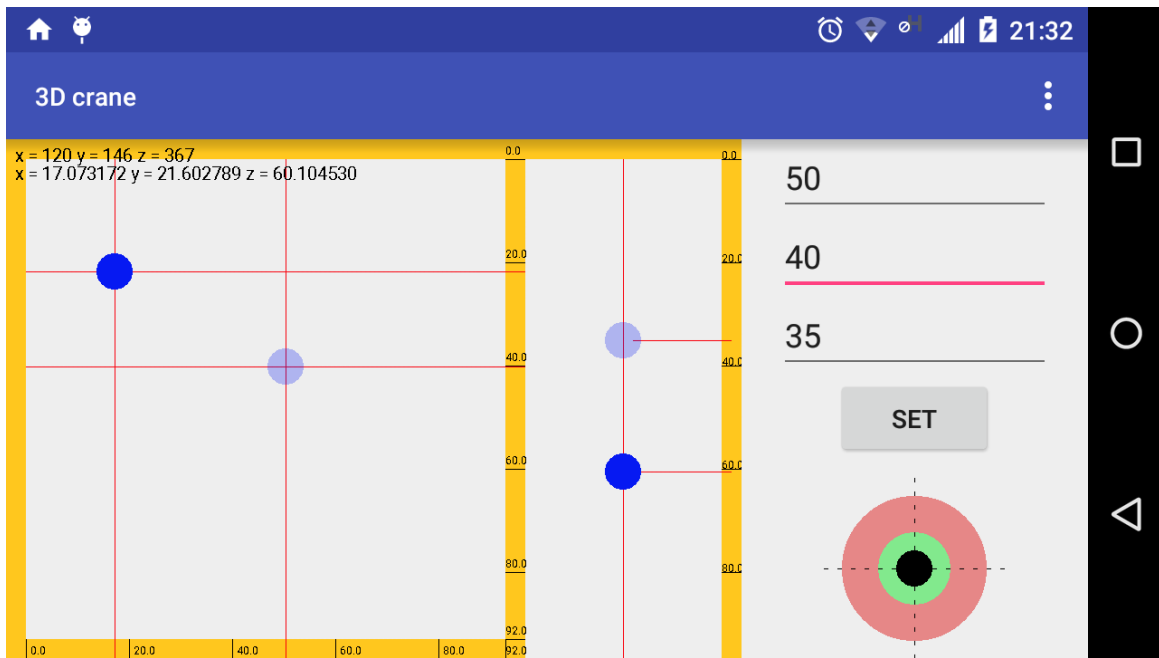


Figure 11.4 3D crane app in landscape mode

App has a menu where the user can open settings window and set the IP, receiving and sending UDP ports. Settings window look same as shown in the case of the MLS app (Figure 10.2). These settings are saved on device and vales are used for data sending and receiving calls in application.

12 SIMULATION IN MATLAB/SIMULINK

12.1 Magnetic levitation system

The model for the Magnetic Levitation system was taken from A-Lab [40], to make it work with incoming input and send result data back, a packet input and packet output blocks were added.

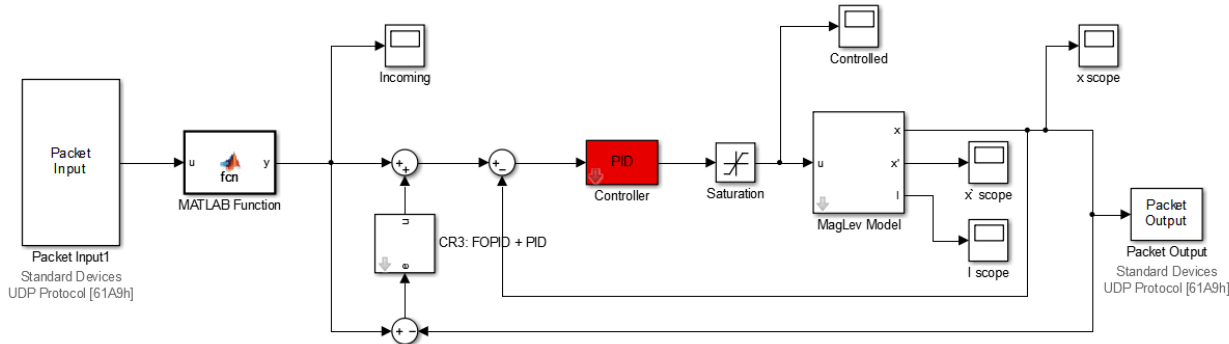


Figure 12.1 MLS model in Simulink3D crane

Packet output block outputs incoming data, but if there is no data it outputs 0. Zero as an input for control system is unacceptable as it tries to recalculate for zero and not for the number desired from Android device. To solve this problem a logic was added using MATLAB function.

```
MATLAB Function x
1 function y = fcn(u)
2
3     persistent saved_value
4
5     if isempty(saved_value)
6         saved_value = 0.01;
7     end
8
9     if u > 0
10        saved_value = u;
11    end
12    y = saved_value
```

Figure 12.2 Checking incoming data

The function has one persistent field: `saved_value`. Persistent field means that over the calls of the function the value of field stays the same, if there is no value assignment. This field is

initialized once with value of 0.01, as a default value. When data is passed to this function from packet input, the functions checks if value is greater than 0 and it is the value will be saved in persistent field, if not function will not modify the persistent field. The function block will output the value of the persistent field. The code of this function in shown on Figure 11.2.

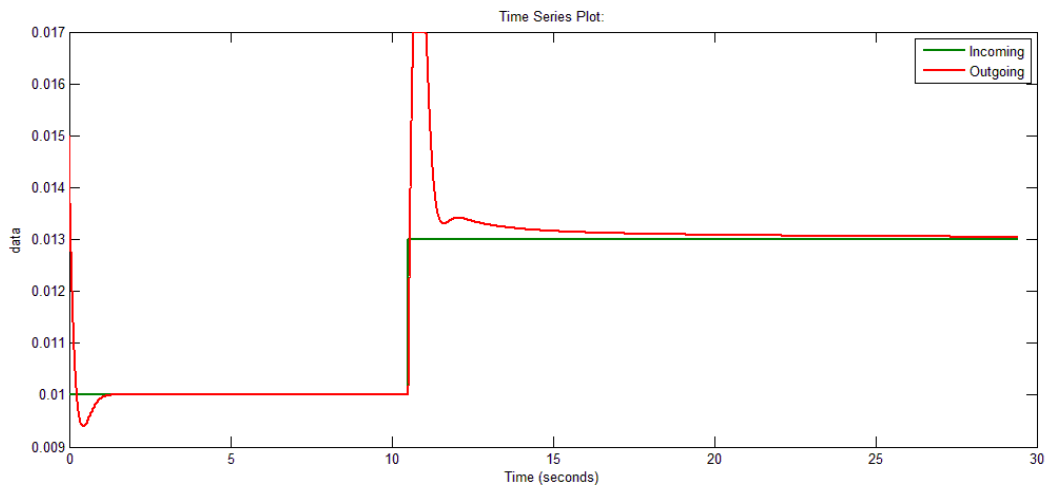


Figure 12.3 Simulation result.

The result of simulation run in MATLAB/Simulink while data was received from and send back to the Android device, running the MLS demonstration application. The simulation stabilized spheres position for default position which is 0.01, then the position was changed from Android and was set to be 0.013. The MLS system controller managed to stabilize the spheres position according to incoming data. The data was also send back and the visualization on android was updated accordingly.

12.23D crane

The template for the 3D crane model was taken from a demo provided by Inteco LLC. To make model work with remote device, the packet input and packet output blocks were added and connected to the controller. Simple PID controllers were used to control the position of crane.

3DCrane Simulation Model

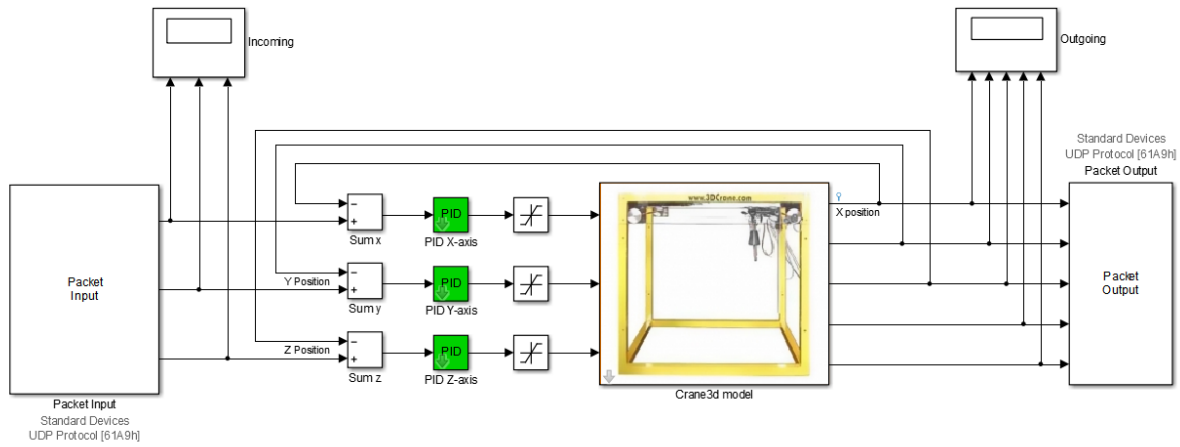


Figure 12.4 3D Crane model for simulation in Simulink

The application was tested against simulation. The results of test is shown on Figure 12.5, the incoming coordinates for crane’s X and Y positions are registered by 3D crane model and the crane’s position is changed according to the input. The results are send back to the application running on Android device and the visualization is updated accordingly.

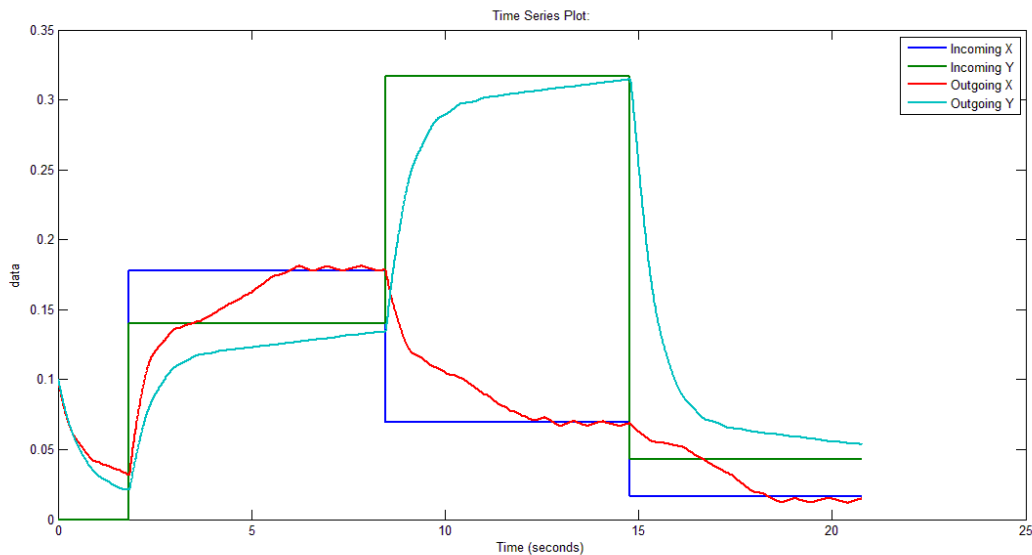


Figure 12.5 Controlling simulation

The position of crane was dictated four times, the angular deviation, during crane's travel time to the desired position, is shown on Figure 12.6.

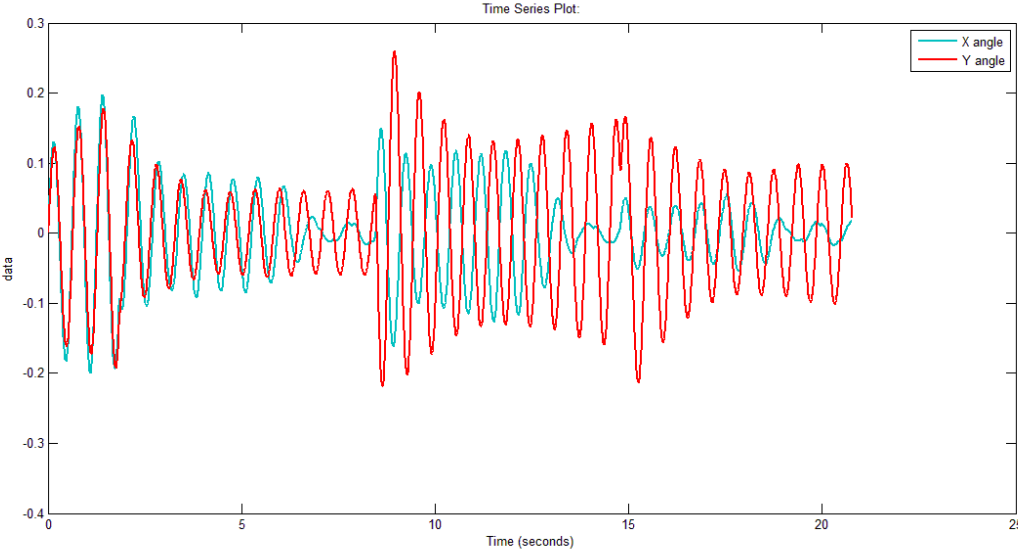


Figure 12.6 Crane's deviation

13 SUMMARY

The main goal of this work was to create an Android library, which would implement basic functionalities to visualize and control two systems: the 3D crane and the Magnetic Levitation System. The control is simulated in MATLAB/Simulink or on real device. The data needed for visualization comes from MATLAB/Simulink, with or without physical device connected to PC which runs the system model.

The main disadvantages the previous, similar works had, which was the limited supported devices, only worked for screens with specific sizes, and lack of ability to reuse solutions and apply them to different applications, have been solved. Applications build using the libraries created during this work support screens with wide variety of resolutions and sizes. The applications work in both, landscape and portrait mode. They support mobile phones and tablets, running on Android operating system.

The library is written using native programming language for Android application development. It is simple to implement and follows the standards of creating such library. Developers are given functionality for implementing the systems, customize them and add extra functionality if required.

The application logic is divided in different classes and the developer can create applications for different systems rather than exclusively the two one, the MLS and 3D crane, and only use parts that are generally same for these type of systems, like the data sending and receiving functionality.

For communication between devices, Android and PC, UPD protocol was choose and implemented on both ends of the communication. In Android the incoming packages are managed by a service, which is always on during the applications lifecycle. The outgoing packets are sent when certain action is invoked on Android. The communication executes in the background thread and the main, the UI thread, works flawlessly, without freezes and lags.

For the Magnetic Levitation System the view or a user experience element was created using the 3D model which is replica of the physical device. The view looks realistic and visualizes the levitated spheres position. The view registers user touch inputs like tapping and moving. The

user can change spheres position and control the system, set desired distance for the sphere from the magnet.

For the 3D crane there were two views created. One to visualize the three, X, Y and Z, axis of the crane and second one to display the deviation of the weight hanging on the crane. The first view, with name CraneView, registers user touch input and gives ability to control the system, by tapping and/or moving and also draws the real time position of the simulated or actual crane. The second one, the CraneAngleView, has only visualization purpose. These two views are separate elements and can be shown on user interface together or separately.

The usage of the libraries are documented and explained in the manual, which is part of this thesis paper. Using these libraries, two Android applications were developed. One for the Magnetic Levitation System and second for the 3D crane. For visualization of the process and controlling the system, only the functionality from library was used. The applications demonstrate the simplicity of implementing the systems and flexibility, as 3D crane app is equipped with extra functionality, with buttons for moving crane in the center or starting position. The 3D crane demonstration app also shows how can user interface be implemented in different ways, the app has two layouts, one for portrait and one for landscape modes and they are fitted for the better design.

The application were tested with simulations running on MATLAB/Simulink. From feedback of test the minor changes were applied to the libraries. For further testing would be nice to see how the application behaves on devices with different screen resolutions and hardware capabilities.

14 KOKKUVÕTE

Käesoleva töö eesmärk oli luua Android teek, mis võimaldaks ellu viia ja visualiseerida kaks kontrollisüsteemi põhifunktsiooni: 3D kraana ja magnetilise levitatsiooni süsteemi. Juhtimist simuleeritakse MATLAB/Simulinkis või reaalse seadme peal. Visualiseerimiseks vajalikud andmed pärinevad MATLAB/Simulinkilt, või ilma füüsilise seadmega ühendatud arvutist, mis käitab süsteemi mudelit.

Peamisi puudusi eelmistel, sarnastel töödel oli piiratud arv toetatud seadmed, ekraanide muutumatud suurused ja suutmatus taaskasutada lahendusi ja rakendada neid erinevates rakendustes. Need probleemid on lahendatud. Käesolevas töös on loodud tugi erinevate resolutsioonidega ekraanidele. Rakendused töötavad nii maastiku kui portree režiimis. Nad toetavad mobiiltelefone ja tabearvuteid, töötavad Android operatsioonisüsteemil.

Teek on kirjutatud, kasutades Android rakenduste arendamise programmeerimiskeelt. Seda on lihtne rakendada ja luua vajalikke raamatukogu standardeid.

Android ja PC seadmete vaheliseks sutlemiseks valiti UPD protokoll.

Magnetilise levitatsiooni süsteemi kuva element oli loodud kasutades seadme 3D-mudelit. Vaade tundub realistlik ja visualiseerib kera levitatsiooni. Vaade reageerib kasutaja puute sisenditele, nt puudutades ja liigutades. Kasutaja saab muuta kera asukohta ja, seada soovitud vahemaa kera ja magneti vahele.

3D kraanast on loodud kaks vaadet. Üks visualiseerib kolm kraana telge X, Y ja Z ja teine kraanale riputatud massi kõrvalekalde kuvamiseks. Esimene vaade nimega CraneView, registreerib kasutaja puutesisestuse ja annab võimaluse kontrollida süsteemi. Teine, CraneAngleView, on loodud ainult visualiseerimise eesmärgil. Neid vaateid on võimalik kuvada kordamööda ja eraldi.

Teekide kasutamine on dokumenteeritud ja on kirjutatud ka käsiraamat mis on selle töö osaks. Neid teeki kasutades töötati välja kaks Android rakendust. Üks magnetilise levitatsiooni süsteem ja teine 3D kraana. Protsessi visualiseerimiseks ja süsteemi juhtimiseks kasutati ainult teegi

funktsionaalsust. Loodud rakendused on lihtsad kasutada ja paindlikud. 3D kraana app omab kahte kasutajaliidest mida rakendatakse erinevalt, kuva võib olla portree ja maastiku asendis.

Rakendust katsetati MATLAB/Simulinkis simulatsioone rakendades. Testide põhjal tehti väiksemaid muudatusi teekides. Täiendavalt testides oleks tore näha, kuidas rakendus käitub erinevatel seadmetel erineva riistvaraga.

15 REFERENCES

1. Kurose, J. F.; Ross, K. W. (2010). Computer Networking: A Top-Down Approach (5th ed.). Boston, MA: Pearson Education.
2. Web site of Android development documentation [WWW]
<http://developer.android.com/reference/java/net/DatagramSocket.html>
3. Web site of Android development documentation [WWW]
<http://developer.android.com/reference/java/net/DatagramPacket.html>
4. Web site of Android development documentation [WWW]
<http://developer.android.com/reference/android/os/AsyncTask.html>
5. Web site of Android development documentation [WWW]
<http://developer.android.com/guide/components/processes-and-threads.html>
6. Web site of Android development documentation [WWW]
<http://developer.android.com/guide/components/services.html>
7. Web site of Android development documentation [WWW]
<http://developer.android.com/reference/android/content/BroadcastReceiver.html>
8. Web site for Eclipse, Android and Java training and support [WWW]
<http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html>
9. Web site of Android development documentation [WWW]
<http://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html>
10. Web site of Android development documentation [WWW]
<http://developer.android.com/guide/components/intents-filters.html>
11. Web site of MathWorks [WWW]
<http://se.mathworks.com/help/simulink/slref/datatypeconversion.html>
12. Web site of MathWorks [WWW]
<http://se.mathworks.com/help/rtwin/ref/packetoutput.html>
13. Web page for Blender projects and software [WWW] <https://www.blender.org/>
14. Web page for Blender projects and software [WWW]
<https://www.blender.org/manual/fr/render/freestyle/index.html>
15. Web page for InkScape software [WWW] <https://inkscape.org/en/>

16. Web site of Android development documentation [WWW] <http://developer.android.com/tools/help/vector-asset-studio.html>
17. Web site of official PNG reference library [WWW] <http://www.libpng.org/pub/png/>
18. Web site of Android development documentation [WWW] http://developer.android.com/guide/practices/screens_support.html
19. Web site of Alpha Control research laboratory [WWW] http://a-lab.ee/equipment#qt-laboratory_devices-ui-tabs3
20. Web site of Alpha Control research laboratory [WWW] <http://a-lab.ee/man/MagLev-user-manual.pdf>
21. Web site of Alpha Control research laboratory [WWW] http://www.a-lab.ee/edu/sites/default/files/Preobrazhenskiy_BSc.pdf
22. The Statistics Portal for Market Data, Market Research and Market Studies [WWW] <http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>
23. Android Studio project site [WWW] <http://tools.android.com/tech-docs/new-build-system/aar-format>
24. Web page of Oculus Rift [WWW] <https://www.oculus.com/en-us/>
25. Web site of Alpha Control research laboratory [WWW] <http://a-lab.ee/man/3DCrane-user-manual.pdf>
26. Web site of Java documentation and tutorials [WWW] <https://docs.oracle.com/javase/tutorial/java/concepts/class.html>
27. Web site of Java documentation and tutorials [WWW] <https://docs.oracle.com/javase/tutorial/java/package/packages.html>
28. Web site of Android development documentation [WWW] <http://developer.android.com/tools/studio/index.html>
29. Web site of Alpha Control research laboratory [WWW] http://a-lab.ee/equipment#qt-laboratory_devices-ui-tabs1
30. Web site of MathWorks [WWW] <http://se.mathworks.com/help/rtwin/ref/packetinput.html>
31. Web page of processing software and projects [WWW] <https://processing.org>
32. Description of Android OS [WWW] <http://developer.android.com/about/android.html>

33. Web site of Android development documentation [WWW]
<http://developer.android.com/tools/projects/index.html>
34. Web page of tutorial on Android Software development [WWW]
http://www.cprogramming.com/android/android_getting_started.html
35. Web site of Android development documentation [WWW]
<http://developer.android.com/guide/components/activities.html>
36. Wikipedia article about APK [WWW]
https://en.wikipedia.org/wiki/Android_application_package
37. Wikipedia article about MATLAB [WWW] <https://en.wikipedia.org/wiki/MATLAB>
38. Web page of Simulink product [WWW] <http://se.mathworks.com/products/simulink/>
39. Web site of Alpha Control research laboratory [WWW] <https://A-Lab.ee>
40. A. Tepljakov, E. Petlenkov, J. Belikov, and E. A. Gonzalez, “Design of retuning fractional PID controllers for a closed-loop magnetic levitation control system,” in ICARCV 2014 : The 13th International Conference on Control, Automation, Robotics & Vision, 2014, pp. 1345–1350.
41. R.-J. Wai, J.-D. Lee, and K.-L. Chuang, “Real-Time PID Control Strategy for Maglev Transportation System via Particle Swarm Optimization,” IEEE Trans. Ind. Electron., vol. 58, no. 2, pp. 629–646, Feb. 2011.

16 APPENDICES

Appendix 1. CraneView code

```
public class CraneView extends View {

    OnCranePositionChangeListener mListener;

    public boolean isDebug() {
        return debug;
    }

    public void setDebug(boolean debug) {
        this.debug = debug;
    }

    private boolean debug = false;

    //color resources
    int circleColor;
    int lineColor;
    int borderColor;

    //paints for lines and circle
    private Paint mLinePaint;
    private Paint mShadowLinePaint;
    private Paint mCirclePaint;
    private Paint mShadowCirclePaint;
    private Paint mBorderPaint;
    private Paint mIntervalPaint;

    //circles coordinates
    private int mPositionX = -1;
    private int mPositionY = -1;
    private int mPositionZ = -1;

    //shadow circle coordinates
    private int mShadowPositionX = 10;
    private int mShadowPositionY = 10;
    private int mShadowPositionZ = 10;

    //width and height of our view
    private int mViewWidth;
    private int mViewHeight;

    //dimensions of physical crane
    private int mSizeX;
    private int mSizeY;
    private int mSizeZ;
    private int mSizeBorder;

    //ration between physycal device measurements and pixels
    private static double mRatioX;
    private static double mRatioY;
    private static double mRatioZ;

    //interval size in cm, default is 10cm
    private int mIntervalSize = 20;

    //cranes representations radius
```

```

private int mCraneRadius = 20;

public static final int AXIS_X = 0;
public static final int AXIS_Y = 1;
public static final int AXIS_Z = 2;

private int mBorderWidth;

private int mTopViewWidth, mTopViewHeight, mFrontViewWidth, mFrontViewHeight;

public CraneView(Context context) {
    super(context);
    init();
}

public CraneView(Context context, AttributeSet attrs) {
    super(context, attrs);

    TypedArray a = context.getTheme().obtainStyledAttributes(
        attrs,
        R.styleable.CraneView,
        0, 0
    );

    try {
        circleColor = a.getColor(R.styleable.CraneView_circleColor, 0xff000000);
        lineColor = a.getColor(R.styleable.CraneView_lineColor, 0xff000000);
        borderColor = a.getColor(R.styleable.CraneView_borderColor, 0xff000000);

        if (!a.hasValue(R.styleable.CraneView_axisX))
            throw new RuntimeException("Defining axisX is required for
CraneView");
        if (!a.hasValue(R.styleable.CraneView_axisY))
            throw new RuntimeException("Defining axisY is required for
CraneView");
        if (!a.hasValue(R.styleable.CraneView_axisZ))
            throw new RuntimeException("Defining axisZ is required for
CraneView");
        if (!a.hasValue(R.styleable.CraneView_border))
            throw new RuntimeException("Defining border is required for
CraneView");

        mSizeX = a.getInt(R.styleable.CraneView_axisX, -1);
        mSizeY = a.getInt(R.styleable.CraneView_axisY, -1);
        mSizeZ = a.getInt(R.styleable.CraneView_axisZ, -1);
        mSizeBorder = a.getInt(R.styleable.CraneView_border, -1);

        mIntervalSize = a.getInt(R.styleable.CraneView_interval, 20);
        init();
    } finally {
        a.recycle();
    }
}

public CraneView(Context context, AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);

    TypedArray a = context.getTheme().obtainStyledAttributes(
        attrs,
        R.styleable.CraneView,
        0, 0

```

```

);

try {
    circleColor = a.getColor(R.styleable.CraneView_circleColor, 0xff000000);
    lineColor = a.getColor(R.styleable.CraneView_lineColor, 0xff000000);
    borderColor = a.getColor(R.styleable.CraneView_borderColor, 0xff000000);

    if (!a.hasValue(R.styleable.CraneView_axisX))
        throw new RuntimeException("Defining axisX is required for
CraneView");
    if (!a.hasValue(R.styleable.CraneView_axisY))
        throw new RuntimeException("Defining axisY is required for
CraneView");
    if (!a.hasValue(R.styleable.CraneView_axisZ))
        throw new RuntimeException("Defining axisZ is required for
CraneView");
    if (!a.hasValue(R.styleable.CraneView_border))
        throw new RuntimeException("Defining border is required for
CraneView");

    mSizeX = a.getInt(R.styleable.CraneView_axisX, -1);
    mSizeY = a.getInt(R.styleable.CraneView_axisY, -1);
    mSizeZ = a.getInt(R.styleable.CraneView_axisZ, -1);
    mSizeBorder = a.getInt(R.styleable.CraneView_border, -1);

    mIntervalSize = a.getInt(R.styleable.CraneView_interval, 20);
    init();
} finally {
    a.recycle();
}
}

public interface OnCranePositionChangeListener {
    void onPositionChangeEvent(double x, double y, double z);
}

public void setCranePositionChangeListener(OnCranePositionChangeListener
eventListener) {
    mListener = eventListener;
}

private void init() {
    mLinePaint = new Paint();
    mLinePaint.setStyle(Paint.Style.STROKE);
    mLinePaint.setColor(lineColor);

    mShadowLinePaint = new Paint();
    mShadowLinePaint.setStyle(Paint.Style.STROKE);
    mShadowLinePaint.setColor(lineColor);
    mShadowLinePaint.setAlpha(70);

    mCirclePaint = new Paint();
    mCirclePaint.setColor(circleColor);

    mShadowCirclePaint = new Paint();
    mShadowCirclePaint.setColor(circleColor);
    mShadowCirclePaint.setAlpha(70);

    mBorderPaint = new Paint();

```

```

    mBorderPaint.setStyle(Paint.Style.STROKE);
    mBorderPaint.setColor(borderColor);
    mBorderPaint.setStrokeWidth(10);

    mIntervalPaint = new Paint();
    mIntervalPaint.setStyle(Paint.Style.STROKE);
    mIntervalPaint.setColor(Color.BLACK);

    Resources r = getResources();
    mCraneRadius = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
14, r.getDisplayMetrics());
}

    public void setPosition(@NonNull double cranePositionX, @NonNull double
cranePositionY, @NonNull double cranePositionZ) {

        this.mPositionX = mBorderWidth + cmToPixel(cranePositionX, AXIS_X);
        this.mPositionY = mBorderWidth + cmToPixel(cranePositionY, AXIS_Y);
        this.mPositionZ = mBorderWidth + cmToPixel(cranePositionZ, AXIS_Z);

        invalidate();
    }

    public void setShadowPosition(@NonNull double cranePositionX, @NonNull double
cranePositionY, @NonNull double cranePositionZ) {

        this.mShadowPositionX = mBorderWidth + cmToPixel(cranePositionX, AXIS_X);
        this.mShadowPositionY = mBorderWidth + cmToPixel(cranePositionY, AXIS_Y);
        this.mShadowPositionZ = mBorderWidth + cmToPixel(cranePositionZ, AXIS_Z);

        invalidate();
    }

    public double getXInCm() {
        return pixelToCm(mPositionX - mBorderWidth, AXIS_X);
    }

    public double getYInCm() {
        return pixelToCm(mPositionY - mBorderWidth, AXIS_Y);
    }

    public double getZInCm() {
        return pixelToCm(mPositionZ - mBorderWidth, AXIS_Z);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);

        int desiredWidth = 100;
        int desiredHeight = 100;

        int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        int widthSize = MeasureSpec.getSize(widthMeasureSpec);
        int heightMode = MeasureSpec.getMode(heightMeasureSpec);
        int heightSize = MeasureSpec.getSize(heightMeasureSpec);

        int width;
        int height;

        //Measure Width
        if (widthMode == MeasureSpec.EXACTLY) {
            width = widthSize;

```

```

    } else if (widthMode == MeasureSpec.AT_MOST) {
        width = widthSize;
    } else {
        width = desiredWidth;
    }
}

//Measure Height
if (heightMode == MeasureSpec.EXACTLY) {
    height = heightSize;
} else if (heightMode == MeasureSpec.AT_MOST) {
    height = heightSize;
} else {
    height = desiredHeight;
}

mViewWidth = width;
mViewHeight = height;

mFrontViewWidth = width / 5;

mTopViewHeight = width > height ? height : width;
if (height > width) {
    mTopViewHeight -= mFrontViewWidth;
}

mTopViewWidth = mTopViewHeight;
mFrontViewHeight = mTopViewHeight;

if (mPositionX == -1)
    mPositionX = mTopViewHeight / 2;
if (mPositionY == -1)
    mPositionY = mTopViewWidth / 2;
if (mPositionZ == -1)
    mPositionZ = mFrontViewHeight / 2;

try {
    mRatioX = (double) mTopViewWidth / mSizeX;
    mRatioY = (double) mTopViewHeight / mSizeY;
    mRatioZ = (double) mFrontViewHeight / mSizeZ;
} catch (Exception e) {
    e.printStackTrace();
}

mBorderWidth = cmToPixel(mSizeBorder, AXIS_X);

//MUST CALL THIS
setMeasuredDimension(mTopViewWidth + mFrontViewWidth, mTopViewHeight);
}

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    mBorderPaint.setStrokeWidth(mBorderWidth);

    //draw borders
    canvas.drawRect(mBorderWidth / 2,
        mBorderWidth / 2,
        mTopViewWidth - mBorderWidth / 2,
        mTopViewHeight - mBorderWidth / 2,
        mBorderPaint);
    canvas.drawLine(mTopViewWidth,
        mBorderWidth / 2,

```

```

        mTopViewWidth + mFrontViewWidth,
        mBorderWidth / 2,
        mBorderPaint);
canvas.drawLine(mTopViewWidth + mFrontViewWidth - mBorderWidth / 2,
        mBorderWidth / 2,
        mTopViewWidth + mFrontViewWidth - mBorderWidth / 2,
        mFrontViewHeight,
        mBorderPaint);

//draw vertical and horizontal lines
canvas.drawLine(mPositionX, mBorderWidth, mPositionX, mTopViewHeight,
mLinePaint);
canvas.drawLine(mBorderWidth, mPositionY, mTopViewWidth, mPositionY,
mLinePaint);
canvas.drawLine(mShadowPositionX, mBorderWidth, mShadowPositionX,
mTopViewHeight, mLinePaint);
canvas.drawLine(mBorderWidth, mShadowPositionY, mTopViewWidth,
mShadowPositionY, mLinePaint);

//draw center
canvas.drawCircle(mPositionX, mPositionY, mCraneRadius, mCirclePaint);
canvas.drawCircle(mShadowPositionX, mShadowPositionY, mCraneRadius,
mShadowCirclePaint);

//draw up and down
canvas.drawLine(mTopViewWidth + mFrontViewWidth / 2 - mBorderWidth / 2,
        mBorderWidth,
        mTopViewWidth + mFrontViewWidth / 2 - mBorderWidth / 2,
        mTopViewHeight,
        mLinePaint);
canvas.drawLine(mTopViewWidth + mFrontViewWidth / 2,
        mPositionZ,
        mTopViewWidth + mFrontViewWidth - mBorderWidth / 2,
        mPositionZ,
        mLinePaint);

canvas.drawLine(mTopViewWidth + mFrontViewWidth / 2,
        mShadowPositionZ,
        mTopViewWidth + mFrontViewWidth - mBorderWidth / 2,
        mShadowPositionZ,
        mLinePaint);

canvas.drawCircle(mTopViewWidth + mFrontViewWidth / 2 - mBorderWidth / 2,
mPositionZ, mCraneRadius, mCirclePaint);
canvas.drawCircle(mTopViewWidth + mFrontViewWidth / 2 - mBorderWidth / 2,
mShadowPositionZ, mCraneRadius, mShadowCirclePaint);

Paint paint = new Paint();

paint.setColor(Color.BLACK);
paint.setTextSize(20);
if (debug)
    canvas.drawText(String.format("x = %d y = %d z = %d", mPositionX,
mPositionY, mPositionZ), 10, 25, paint);

double actualPositionX = pixelToCm(mPositionX - mBorderWidth, AXIS_X);
double actualPositionY = pixelToCm(mPositionY - mBorderWidth, AXIS_Y);
double actualPositionZ = pixelToCm(mPositionZ - mBorderWidth, AXIS_Z);

if (debug)
    canvas.drawText(String.format("x = %f y = %f z = %f", actualPositionX,
actualPositionY, actualPositionZ), 10, 45, paint);

```

```

        drawIntervals(canvas);
    }

    private void drawIntervals(Canvas canvas) {

        int intervalLengthInPixelsX = (int) cmToPixel(mIntervalSize, AXIS_X);
        int intervalLengthInPixelsY = (int) cmToPixel(mIntervalSize, AXIS_Y);
        int intervalLengthInPixelsZ = (int) cmToPixel(mIntervalSize, AXIS_Z);

        int intervalCountX = (mSizeX - 2 * mSizeBorder) / mIntervalSize;
        int intervalCountY = (mSizeY - 2 * mSizeBorder) / mIntervalSize;
        int intervalCountZ = (mSizeZ - 2 * mSizeBorder) / mIntervalSize;

        for (int i = 0; i <= intervalCountX; i++) {
            canvas.drawLine(mBorderWidth + intervalLengthInPixelsX * i,
                mTopViewHeight,
                mBorderWidth + intervalLengthInPixelsX * i,
                mTopViewHeight - mBorderWidth,
                mIntervalPaint);
            if (debug)
                canvas.drawText(Double.toString(i * mIntervalSize),
                    mBorderWidth + intervalLengthInPixelsX * i + 5,
                    mTopViewHeight - 5,
                    mIntervalPaint);
        }
        canvas.drawLine(mTopViewWidth - mBorderWidth,
            mTopViewHeight,
            mTopViewWidth - mBorderWidth,
            mTopViewHeight - mBorderWidth,
            mIntervalPaint);
        if (debug)
            canvas.drawText(Double.toString(mSizeX - 2 * mSizeBorder),
                mTopViewWidth - mBorderWidth,
                mTopViewHeight - 5,
                mIntervalPaint);

        for (int i = 0; i <= intervalCountY; i++) {
            canvas.drawLine(mTopViewWidth - mBorderWidth,
                mBorderWidth + intervalLengthInPixelsY * i,
                mTopViewWidth,
                mBorderWidth + intervalLengthInPixelsY * i,
                mIntervalPaint);
            if (debug)
                canvas.drawText(Double.toString(i * mIntervalSize),
                    mTopViewWidth - mBorderWidth,
                    mBorderWidth + intervalLengthInPixelsY * i - 5,
                    mIntervalPaint);
        }
        canvas.drawLine(mTopViewWidth - mBorderWidth,
            mTopViewHeight - mBorderWidth,
            mTopViewWidth,
            mTopViewHeight - mBorderWidth,
            mIntervalPaint);

        if (debug)
            canvas.drawText(Double.toString(mSizeY - 2 * mSizeBorder),
                mTopViewWidth - mBorderWidth,
                mTopViewHeight - mBorderWidth - 5,
                mIntervalPaint);

        for (int i = 0; i <= intervalCountZ; i++) {
            canvas.drawLine(mTopViewWidth + mFrontViewWidth - mBorderWidth,

```



```

        mBorderWidth + intervalLengthInPixelsZ * i,
        mTopViewWidth + mFrontViewWidth,
        mBorderWidth + intervalLengthInPixelsZ * i,
        mIntervalPaint);

    if (debug)
        canvas.drawText(Double.toString(i * mIntervalSize),
            mTopViewWidth + mFrontViewWidth - mBorderWidth,
            mBorderWidth + intervalLengthInPixelsZ * i,
            mIntervalPaint);
    }
}

@Override
public boolean onTouchEvent(MotionEvent event) {

    boolean handled = false;
    boolean throwEvent = true;

    int xTouch;
    int yTouch;

    // get touch event coordinates and make transparent circle from it
    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_DOWN:
            xTouch = (int) event.getX(0);
            yTouch = (int) event.getY(0);

            if (isTopSideView(xTouch, yTouch)) {
                mPositionX = xTouch;
                mPositionY = yTouch;
            } else if (isFrontSideView(xTouch, yTouch)) {
                mPositionZ = yTouch;
            } else {
                throwEvent = false;
            }

            invalidate();
            handled = true;

            if (mListener != null && throwEvent)
                mListener.onPositionChangeEvent(pixelToCm(mPositionX -
                    mBorderWidth, AXIS_X),
                    pixelToCm(mPositionY - mBorderWidth, AXIS_Y),
                    pixelToCm(mPositionZ - mBorderWidth, AXIS_Z));

            break;

        case MotionEvent.ACTION_MOVE:
            xTouch = (int) event.getX(0);
            yTouch = (int) event.getY(0);

            if (isTopSideView(xTouch, yTouch)) {
                mPositionX = xTouch;
                mPositionY = yTouch;
            } else if (isFrontSideView(xTouch, yTouch)) {
                mPositionZ = yTouch;
            } else {
                throwEvent = false;
            }

            invalidate();
    }
}

```

```

        handled = true;

        if (mListener != null && throwEvent)
            mListener.onPositionChangeEvent(pixelToCm(mPositionX -
mBorderWidth, AXIS_X),
                pixelToCm(mPositionY - mBorderWidth, AXIS_Y),
                pixelToCm(mPositionZ - mBorderWidth, AXIS_Z));

        break;

    case MotionEvent.ACTION_CANCEL:
        handled = true;
        break;

    default:
        // do nothing
        break;
}

return super.onTouchEvent(event) || handled;
}

public static int cmToPixel(double lengthInCm, int axis) {
    switch (axis) {
        case AXIS_X:
            return (int) (lengthInCm * mRatioX);
        case AXIS_Y:
            return (int) (lengthInCm * mRatioY);
        case AXIS_Z:
            return (int) (lengthInCm * mRatioZ);
    }
    return 0;
}

private boolean isTopSideView(int x, int y) {
    if (x >= mBorderWidth
        && x <= mTopViewWidth - mBorderWidth
        && y >= mBorderWidth
        && y <= mTopViewHeight - mBorderWidth)
        return true;
    return false;
}

private boolean isFrontSideView(int x, int y) {
    if (x >= mTopViewWidth
        && x <= mTopViewWidth + mFrontViewWidth
        && y >= mBorderWidth
        && y <= mFrontViewHeight - mBorderWidth / 2)
        return true;
    return false;
}

public static double pixelToCm(int lengthInPixels, int axis) {
    switch (axis) {
        case AXIS_X:
            return lengthInPixels / mRatioX;
        case AXIS_Y:
            return lengthInPixels / mRatioY;
        case AXIS_Z:
            return lengthInPixels / mRatioZ;
    }
    return 0;
}

```

} }

Appendix 2. CraneAngleView code

```
public class CraneAngleView extends View {

    private int mViewWidth;
    private int mViewHeight;

    private int mOutsideCircleRadius;
    private int mInsideCircleRadius;
    private int mCraneCircleRadius;

    private int mCranePositionX;
    private int mCranePositionY;

    private int mViewCenterX;
    private int mViewCenterY;

    private Path mPath;

    private Paint mDangerBackgroundPaint;
    private Paint mOkBackgroundPaint;
    private Paint mCranePaint;
    private Paint mLinePaint;

    private int mAngleHorizontal;
    private int mAngleVertical;

    public CraneAngleView(Context context) {
        super(context);
        init();
    }

    public CraneAngleView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    public CraneAngleView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        init();
    }

    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    public CraneAngleView(Context context, AttributeSet attrs, int defStyleAttr, int
defStyleRes) {
        super(context, attrs, defStyleAttr, defStyleRes);
        init();
    }

    private void init() {

        mDangerBackgroundPaint = new Paint();
        mDangerBackgroundPaint.setColor(ContextCompat.getColor(getContext(),
R.color.danger));

        mOkBackgroundPaint = new Paint();
        mOkBackgroundPaint.setColor(ContextCompat.getColor(getContext(),
R.color.lightGreen));

        mCranePaint = new Paint();
        mCranePaint.setColor(ContextCompat.getColor(getContext(), R.color.black));
    }
}
```

```

        mLinePaint = new Paint();
        mLinePaint.setColor(ContextCompat.getColor(getContext(), R.color.black));
        mLinePaint.setStyle(Paint.Style.STROKE);
        mLinePaint.setPathEffect(new DashPathEffect(new float[]{5, 10}, 0));
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);

        int desiredWidth = 200;
        int desiredHeight = 200;

        int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        int widthSize = MeasureSpec.getSize(widthMeasureSpec);
        int heightMode = MeasureSpec.getMode(heightMeasureSpec);
        int heightSize = MeasureSpec.getSize(heightMeasureSpec);

        int width;
        int height;

        //Measure Width
        if (widthMode == MeasureSpec.EXACTLY) {
            width = widthSize;
        } else if (widthMode == MeasureSpec.AT_MOST) {
            width = Math.min(desiredWidth, widthSize);
        } else {
            width = desiredWidth;
        }

        //Measure Height
        if (heightMode == MeasureSpec.EXACTLY) {
            height = heightSize;
        } else if (heightMode == MeasureSpec.AT_MOST) {
            height = Math.min(desiredHeight, heightSize);
        } else {
            height = desiredHeight;
        }

        height = height > width ? width : height;
        width = width > height ? height : width;

        mCranePositionX = mViewCenterX = width / 2;
        mCranePositionY = mViewCenterY = height / 2;

        mOutsideCircleRadius = width / 2 - 20;
        mInsideCircleRadius = mOutsideCircleRadius / 2;
        mCraneCircleRadius = mInsideCircleRadius / 2;

        mViewWidth = width;
        mViewHeight = height;

        setMeasuredDimension(width, height);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        canvas.drawCircle(mViewCenterX, mViewCenterY, mOutsideCircleRadius,
            mDangerBackgroundPaint);
    }

```

```

        canvas.drawCircle(mViewCenterX, mViewCenterY, mInsideCircleRadius,
mOkBackgroundPaint);
        canvas.drawCircle(mCranePositionX, mCranePositionY, mCraneCircleRadius,
mCranePaint);

        mPath = new Path();
        mPath.moveTo(0, mViewHeight / 2);
        mPath.quadTo(mViewWidth / 2, mViewHeight / 2, mViewWidth, mViewHeight / 2);
        canvas.drawPath(mPath, mLinePaint);

        mPath.moveTo(mViewWidth / 2, 0);
        mPath.quadTo(mViewWidth / 2, mViewHeight / 2, mViewWidth / 2, mViewHeight);
        canvas.drawPath(mPath, mLinePaint);
    }

    public void setCraneAngle(double angleHorizontal, double angleVertical, double
height) {
        double aH = Math.cos(angleHorizontal) * height;
        double aV = Math.cos(angleVertical) * height;

        mCranePositionX += aH;
        mCranePositionY += aV;

        invalidate();
    }
}

```

Appendix 3. SendDataTask

```
public class SendDataTask extends AsyncTask<Double, Void, Void> {

    private static final String TAG = SendDataTask.class.getSimpleName();

    private int mPort;
    InetAddress inet_addr;
    DatagramSocket socket;

    public String getIp() {
        return "1";
    }

    public boolean setIp(String ip) {
        try {
            inet_addr = InetAddress.getByName(ip);
        } catch (UnknownHostException e) {
            Log.e(TAG, e.getMessage());
            return false;
        }
        return true;
    }

    public int getPort() {
        return mPort;
    }

    public boolean setPort(int port) {
        if (port > 0 && port <= 65535) {
            this.mPort = port;
            return true;
        }
        return false;
    }

    private byte[] toByteArray(double[] doubleArray) {
        int times = Double.SIZE / Byte.SIZE;
        byte[] bytes = new byte[doubleArray.length * times];
        for (int i = 0; i < doubleArray.length; i++) {
            ByteBuffer.wrap(bytes, i * times, times).putDouble(doubleArray[i]);
        }

        return bytes;
    }

    private double[] toDoubleArray(byte[] byteArray) {
        int times = Double.SIZE / Byte.SIZE;
        double[] doubles = new double[byteArray.length / times];
        for (int i = 0; i < byteArray.length; i++) {
            doubles[i] = ByteBuffer.wrap(byteArray, i * times, times).getDouble();
        }
        return doubles;
    }

    @Override
    protected Void doInBackground(Double... params) {
        double x = params[0];
        double y = params[1];
        double z = params[2];
    }
}
```

```

Log.i(TAG, "Sending data");

ByteBuffer bf = ByteBuffer.allocate(24);
bf.order(ByteOrder.LITTLE_ENDIAN);
bf.putDouble(x);
bf.putDouble(y);
bf.putDouble(z);
byte[] sendBuffer = bf.array();

DatagramPacket packet = new DatagramPacket(sendBuffer, sendBuffer.length,
inet_addr, mPort);
try {
    socket = new DatagramSocket();
    socket.send(packet);
    Log.i(TAG, String.format("x = %f, y = %f, z = %f", x, y, z));
} catch (IOException e) {
    Log.e(TAG, e.getMessage());
}

return null;
}
}

```


Appendix 4. Receiving service code.

```
public class MatlabConnection extends Service {
    public static final String UDP_BROADCAST = "UDPBroadcast";

    public static final String INTENT_DATA_X = "a3dcrane.intent_data.x";
    public static final String INTENT_DATA_Y = "a3dcrane.intent_data.y";
    public static final String INTENT_DATA_Z = "a3dcrane.intent_data.z";
    public static final String INTENT_DATA_X_ANGLE = "a3dcrane.intent_data.xAngle";
    public static final String INTENT_DATA_Y_ANGLE = "a3dcrane.intent_data.yAngle";

    public static final String PORT_NUMBER = "a3dcrane.intent_data.port_number";

    private int mPortNumber;

    private Thread UDPBroadcastThread;
    private Boolean shouldRestartSocketListen = true;

    DatagramSocket socket;

    private static final String TAG = MatlabConnection.class.getSimpleName();

    private void listenAndWaitAndThrowIntent() throws Exception {

        byte[] recvBuf = new byte[40];
        if (socket == null || socket.isClosed()) {
            socket = new DatagramSocket(mPortNumber);
            socket.setBroadcast(true);
        }
        //socket.setSoTimeout(1000);
        DatagramPacket packet = new DatagramPacket(recvBuf, recvBuf.length);

        DoubleBuffer db =
        ByteBuffer.wrap(recvBuf).order(ByteOrder.LITTLE_ENDIAN).asDoubleBuffer();

        Log.i(TAG, "Waiting for UDP broadcast");
        socket.receive(packet);

        //db.limit(packet.getLength()/Double.SIZE);

        String senderIP = packet.getAddress().getHostAddress();

        double x = db.get(0);
        double y = db.get(1);
        double z = db.get(2);
        double xAngle = db.get(3);
        double yAngle = db.get(4);

        Log.i(TAG, String.format("Got UDB broadcast from: %s x = %f, y = %f, z = %f,
xAngle = %f, yAngle = %f", senderIP, x, y, z, xAngle, yAngle));

        broadcastIntent(x, y, z, xAngle, yAngle);
        socket.close();
    }

    private void broadcastIntent(double x, double y, double z, double xAngle, double
yAngle) {
        Log.i(TAG, "broadcasting intent");
        Intent intent = new Intent(UDP_BROADCAST);
        intent.putExtra(INTENT_DATA_X, x);
    }
}
```

```

        intent.putExtra(INTENT_DATA_Y, y);
        intent.putExtra(INTENT_DATA_Z, z);
        intent.putExtra(INTENT_DATA_X_ANGLE, xAngle);
        intent.putExtra(INTENT_DATA_Y_ANGLE, yAngle);
        sendBroadcast(intent);
    }

    private byte[] toByteArray(double[] doubleArray) {
        int times = Double.SIZE / Byte.SIZE;
        byte[] bytes = new byte[doubleArray.length * times];
        for (int i = 0; i < doubleArray.length; i++) {
            ByteBuffer.wrap(bytes, i * times, times).putDouble(doubleArray[i]);
        }

        return bytes;
    }

    private double[] toDoubleArray(byte[] byteArray) {
        int times = Double.SIZE / Byte.SIZE;
        double[] doubles = new double[byteArray.length / times];
        for (int i = 0; i < doubles.length; i++) {
            doubles[i] = ByteBuffer.wrap(byteArray, i * times, times).getDouble();
        }
        return doubles;
    }

    void startListenForUDPBroadcast() {
        UDPBroadcastThread = new Thread(new Runnable() {
            public void run() {
                try {
                    while (shouldRestartSocketListen) {
                        listenAndWaitAndThrowIntent();
                    }
                    //if (!shouldListenForUDPBroadcast) throw new ThreadDeath();
                } catch (Exception e) {
                    Log.e(TAG, "No longer listening for UDP broadcasts cause of error
" + e.getMessage());
                }
            }
        });
        UDPBroadcastThread.start();
    }

    public void stopListen() {
        shouldRestartSocketListen = false;
        if (socket != null)
            socket.close();
    }

    @Override
    public void onDestroy() {
        stopListen();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        mPortNumber = intent.getIntExtra(PORT_NUMBER, 25001);
        shouldRestartSocketListen = true;
        startListenForUDPBroadcast();
    }

```

```
        Log.i(TAG, "Service started");
        return START_STICKY;
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

Appendix 5. Attributes for 3D crane

```
<resources>
  <declare-styleable name="CraneView">
    <attr name="lineColor" format="color" />
    <attr name="circleColor" format="color" />
    <attr name="borderColor" format="color" />
    <attr name="axisX" format="integer" />
    <attr name="axisY" format="integer" />
    <attr name="axisZ" format="integer" />
    <attr name="border" format="integer" />
    <attr name="interval" format="integer" />
  </declare-styleable>

  <declare-styleable name="CraneAngleView">
    <attr name="craneColor" format="color" />
    <attr name="dangerColor" format="color" />
    <attr name="safeColor" format="color" />
  </declare-styleable>
</resources>
```

Appendix 6. MLS view code

```
public class MLSView extends View {

    private int mSpherePositionX;
    private int mSpherePositionY;
    private int mSphereRadius;
    private Paint mSpherePaint;

    private int mViewWidth;
    private int mViewHeight;

    //
    private int mRubberStart;
    private int mMagnetStart;

    private int mDistanceBetweenRubberAndMagnetInCm;
    private int mBallRadiusCm;

    private int cmInPixel;

    private SpherePositionChangeListener mListener;

    public interface SpherePositionChangeListener {
        void onSpherePositionChange(double x);
    }

    public void setOnBallPositionChangeListener(SpherePositionChangeListener
eventListener) {
        this.mListener = eventListener;
    }

    public MLSView(Context context) {
        super(context);
        init();
    }

    public MLSView(Context context, AttributeSet attrs) {
        super(context, attrs);
        TypedArray a = context.getTheme().obtainStyledAttributes(
            attrs,
            R.styleable.MLSView,
            0, 0
        );

        try {
            mDistanceBetweenRubberAndMagnetInCm =
a.getColor(R.styleable.MLSView_distanceMagnetRubber, 50);
            mBallRadiusCm = a.getColor(R.styleable.MLSView_ballRadius, 20);

            init();
        } finally {
            a.recycle();
        }
    }

    public MLSView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        TypedArray a = context.getTheme().obtainStyledAttributes(
            attrs,
            R.styleable.MLSView,
```

```

        0, 0
    );

    try {
        mDistanceBetweenRubberAndMagnetInCm =
a.getColor(R.styleable.MLSView_distanceMagnetRubber, 50);
        mBallRadiusCm = a.getColor(R.styleable.MLSView_ballRadius, 20);

        init();
    } finally {
        a.recycle();
    }
}

```

```

@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

    super.onMeasure(widthMeasureSpec, heightMeasureSpec);

    int desiredWidth = 100;
    int desiredHeight = 100;

    int widthMode = MeasureSpec.getMode(widthMeasureSpec);
    int widthSize = MeasureSpec.getSize(widthMeasureSpec);
    int heightMode = MeasureSpec.getMode(heightMeasureSpec);
    int heightSize = MeasureSpec.getSize(heightMeasureSpec);

    int width;
    int height;

    //Measure Width
    if (widthMode == MeasureSpec.EXACTLY) {
        width = widthSize;
    } else if (widthMode == MeasureSpec.AT_MOST) {
        width = Math.min(desiredWidth, widthSize);
    } else {
        width = desiredWidth;
    }

    //Measure Height
    if (heightMode == MeasureSpec.EXACTLY) {
        height = heightSize;
    } else if (heightMode == MeasureSpec.AT_MOST) {
        height = Math.min(desiredHeight, heightSize);
    } else {
        height = desiredHeight;
    }

    if (height * 3 / 4 > width) {
        height = width * 4 / 3;
    } else {
        width = height * 3 / 4;
    }

    mViewWidth = width;
    mViewHeight = height;

    cmInPixel = mViewWidth / 28;

    mRubberStart = height * 47 / 100;
    mSphereRadius = height / 20;
}

```

```

    mSpherePositionX = width / 2;
    mSpherePositionY = mRubberStart - mSphereRadius;

    mMagnetStart = height / 4;

    setMeasuredDimension(width, height);
}

private void init() {
    mSpherePaint = new Paint();
    mSpherePaint.setColor(Color.RED);
    mSpherePaint.setStyle(Paint.Style.FILL_AND_STROKE);
    mSpherePaint.setStrokeWidth(10);
}

@Override
public void onDraw(final Canvas canvas) {

    //Get background image for MLS and draw on canvas
    Drawable mlsBackground = ContextCompat.getDrawable(getContext(),
R.drawable.mls);
    mlsBackground.setBounds(0, 0, canvas.getWidth(), canvas.getHeight());
    mlsBackground.draw(canvas);

    //Draw sphere
    mSpherePaint.setColor(Color.RED);
    canvas.drawCircle(mSpherePositionX,
        mSpherePositionY,
        mSphereRadius,
        mSpherePaint);
    mSpherePaint.setColor(Color.BLACK);
    canvas.drawLine(mSpherePositionX - mSphereRadius - 10,
        mSpherePositionY,
        mSpherePositionX + mSphereRadius + 10,
        mSpherePositionY,
        mSpherePaint);
}

@Override
public boolean onTouchEvent(MotionEvent event) {

    boolean handled = false;
    boolean throwEvent = true;

    int xTouch;
    int yTouch;

    // get touch event coordinates and make transparent circle from it
    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_DOWN:
            xTouch = (int) event.getX(0);
            yTouch = (int) event.getY(0);

            if (isSphereInFreeArea(xTouch, yTouch)) {
                mSpherePositionY = yTouch;
            } else {
                throwEvent = false;
            }

            invalidate();
}

```

```

        handled = true;

        if (mListener != null && throwEvent)
mListener.onSpherePositionChange(distanceFromMagnetToSphereCenterInCm());

        break;

    case MotionEvent.ACTION_MOVE:
        xTouch = (int) event.getX(0);
        yTouch = (int) event.getY(0);

        if (isSphereInFreeArea(xTouch, yTouch)) {
            mSpherePositionY = yTouch;
        } else {
            throwEvent = false;
        }

        invalidate();
        handled = true;

        if (mListener != null && throwEvent)
mListener.onSpherePositionChange(distanceFromMagnetToSphereCenterInCm());

        break;

    case MotionEvent.ACTION_CANCEL:
        handled = true;
        break;

    default:
        // do nothing
        break;
}

return super.onTouchEvent(event) || handled;
}

private boolean isSphereInFreeArea(int xTouch, int yTouch) {
    if (yTouch > mMagnetStart + mSphereRadius
        && yTouch < mRubberStart - mSphereRadius)
        return true;
    return false;
}

public void setBallPosition(double y) {
    int inCm = (int) (y * 100);
    mSpherePositionY = mRubberStart - mSphereRadius - inCm * cmInPixel;
    invalidate();
}

private double distanceFromMagnetToSphereCenterInCm() {
    return (double) (mSpherePositionY - mMagnetStart - mSphereRadius) / cmInPixel;
}
}

```


Appendix 7. MLS view's attributes

```
<resources>  
  <declare-styleable name="MLSView">  
    <attr name="distanceMagnetRubber" format="integer" />  
    <attr name="ballRadius" format="integer" />  
  </declare-styleable>  
</resources>
```