Hardware Platform Development and Android Apps Verification for the RoboCat Design

A report presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Haiquan Zhang

December 2013

This report titled

Hardware Platform Development and Android Apps Verification for the RoboCat Design

by

Haiquan Zhang

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Jianchao Zhu

Professor

Dennis Irwin

Dean, Russ College of Engineering and Technology

ABSTRACT

HAIQUAN ZHANG, M.S., December 2013, Electrical Engineering

Hardware Platform Development and Android Apps Verification for the RoboCat

Director of Project: Jianchao Zhu

The RoboCat is a quadruped biomimetic robot. The RoboCat design is to build a biomimetic bobcat robot such that it can step forward and turn, be manipulated by the end-users, and retrieve environmental information. The hardware platform is the center part to manipulate the RoboCat. The hardware platform is needed to manipulate the servos, to communicate with the end-users, and to provide the environment for the software development and the implementation. Meanwhile, the previous version of the RoboCat was too heavy to step forward.

This project is to build the hardware platform for the RoboCat and develop the software to verify the communications and related functions on the hardware platform for the RoboCat. By using the hardware platform, the software for the RoboCat can be developed. Thus the servos can be manipulated and the communication between the RoboCat and the end-users can be enabled.

To build up the hardware platform, first, a proper operating system (OS) needs to be selected based on the requirements of the RoboCat design. Then, the appropriate hardware boards need to be found and integrated together. Finally, the software needs to be developed to validate the hardware design.

The main accomplishments of the project are as follows. First, the Android OS is selected for the hardware platform. Then, for the central board of the RoboCat, the

ARM Cortex-A8 based BeagleBoard-XM is selected for the development and test purpose, and the Galaxy Nexus cellphone is mounted on the RoboCat, which is more than 1 lb lighter than the previous version of the central board. Meanwhile, the software is developed to enable and validate the communication between the central board and its peripherals, i.e., the Pololu Maestro board, the Arduino Mega board, etc., via the USB interface. Finally, the Android apps are developed to enable the manipulation of the servos from the cell phone through the seek bars and the record of the gait data.

To summarize, in this project, the hardware platform for the RoboCat is built up, and the communication, the integration, and the weight reduction tasks are completed. Future work for the RoboCat may include the modeling and the integration of the current sensors.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1: BACKGROUND

In this chapter, the brief introduction to the RoboCat and its framework are provided.

The RoboCat is a quadruped biomimetic robot. The functionalities of the RoboCat includes to keep balance, to complete fundamental movement, e.g., stepping forward and turning, to record and playback the audio, to record and playback video, to detect, recognize, stare and track a specific object with the onboard vision system, and to interact with the environment.

In addition to the mechanical parts, the RoboCat mainly comprises i) the central board, ii) the servos and the servo controller, iii) the servo current sensor and the related processing board, and iv) the peripherals for the audio and video processing.

The central board is responsible for integrating all the other hardware devices for the RoboCat, to obtain, process, and respond to the information retrieved from the environment and the end-users, and to send the command to manipulate the servos.

The servos and the servo controller are needed to maintain and adjust the attitude of the RoboCat. Totally there are 14 servos mounted on the RoboCat. Each leg has 3 servos to modify the positions of the shoulder, the knee, and the lateral, correspondingly. The remaining 2 servos are used to manipulate the head of the RoboCat.

The servo controller board receives the data from the central board via the USB port or the TTL. Then, the data can be interpreted by the servo controller board and thus the position commands can be sent to the servos.

The value of the current flowing through the servos needs to be obtained in order to monitor the output torque of the servos, whence the command to the servo controller may be adjusted accordingly by the central board. In addition to obtain the current value, the analog current data need to be converted to the digital ones for the central board to retrieve. Therefore, the current sensor processing board is needed.

The RoboCat design has been the EECS Senior Design capstone project for several years. In the previous design, the total weight of the RoboCat was 7 lb 4 oz. It was using the PC-104 as the central board, where the Linux-based operating system (OS) was installed. The Pololu Maestro was selected as the servo controller. Meanwhile, the current and therefore the output torque of the servos are monitored by using the ACS712 current sensor and the Arduino Mega board. The ACS712 current sensor transforms the current signal to the voltage signal proportionally. Then, the Arduino Mega board processes the voltage signal and sends the results to the central board.

CHAPTER 2: PROBLEM STATEMENT

In this chapter, some fundamental problems of the previous RoboCat design are described from the RoboCat functionalities, the OS, the central board, and the application software aspects.

In the previous design, the RoboCat can neither step forward nor turn. One reason is that, the RoboCat itself was too heavy for the servos to maintain the proper positions. Meanwhile, the vendor's software for the Pololu Maestro needed the offline operations and cannot be modified online through the previous central board, i.e., PC-104 [1].

In addition, numerous drivers and apps for the peripherals were unavailable for the Linux based OS, the Ubuntu 10.04, which was installed on the PC-104. An enormous amount of time and effort have been spent on resolving the compatibility and the driver development problems. Furthermore, the Ubuntu 10.04 OS needed a dedicated keyboard and LCD display, which are unavailable on the PC-104. In addition, the user interface (UI) of the Ubuntu 10.04 was inconvenient for the end-users.

The PC-104 was used as the central board. It had several disadvantages. First, the weight of PC-104 was 1 lb 6 oz., which is too heavy. Second, according to the above discussion, the OS installed on PC-104, i.e., the Ubuntu, was inconvenient for the development purpose, while other poplar OS, e.g., the Android OS, is not supported by PC-104. Third, the interface resources for the peripherals were limited. Finally, the external keyboard and LCD display were needed for the development, and hence the modification of the software cannot be completed onboard (RoboCat) for the RoboCat.

Due to the disadvantages of the hardware and the OS described above, the application software was unavailable in the previous design of the RoboCat.

Meanwhile, the vendor's servo controller software cannot be used onboard for the RoboCat.

To overcome the above obstacles, an alternative OS needs to be selected. Meanwhile, a new hardware platform needs to be built up, and the corresponding application software needs to be developed.

CHAPTER 3: OBJECTIVES AND SCOPE

In this chapter, the objectives of this project are described based on the remaining problems discussed in Chapter 2.

1. RoboCat

In this project, the software and hardware platform should be provided for the developers to enable the RoboCat to step forward and turn. Meanwhile, the overall weight needs to be reduced such that the servos can maintain and adjust the positions continuously.

2. OS

The proper OS needs to be selected and tested to facilitate the application software development and to provide a friendly UI for both the developers and the end-users.

The drivers for the widely used protocols, e.g., the USB, should be ready to use for the software development.

The OS should have long life cycle, whence the endeavors for the hardware and software development can be valuable to the up-to-date RoboCat design for a long period and the developers can focus only on the RoboCat design itself.

3. Hardware platform

The first task for the hardware design is to select the proper central boards. To facilitate the end-user's operations, the central board should have the integrated LCD touchscreen. The central board should also have abundant interface resources. In addition, the weight of the central board should be less than 10 oz.

After the central board has been selected, the OS should be tested and may need to be modified as well.

All the hardware devices need to be integrated together and the communication among them should be enabled and validated.

4. Application software

The software development environment should be set up. The software for the communication between the central board and the peripherals, and the onboard (RoboCat) manipulation of the servos needs to be developed and validated.

CHAPTER 4: SIGNIFICANCE

In this chapter, the significance of the project for the RoboCat design is discussed.

From the discussions of the Chapter 2 and Chapter 3, the RoboCat design will be benefited from this project since the developers will be free from endless compatibility problems. Meanwhile, the previous efforts in the hardware platform design and the software development for the RoboCat design can be easily reused.

As a result, the development life cycle will be shortened, such that the possibility for the hardware platform to be obsolete before the RoboCat design is

completed is eliminated. The developers can thus focus on the tasks directly related the RoboCat design itself. The comparison of the new accumulated RoboCat design progress with the previous one is illustrated in Figure 1.

The friendly development environment and UI provided by this project can be more attractive to both the potential developers and users.

CHAPTER 5: APPROACH

In this chapter, the approaches for the tasks of the project are discussed.

1. RoboCat functionalities

The software will be developed and the hardware platform will be built up, based on which, the RoboCat can step forward and turn.

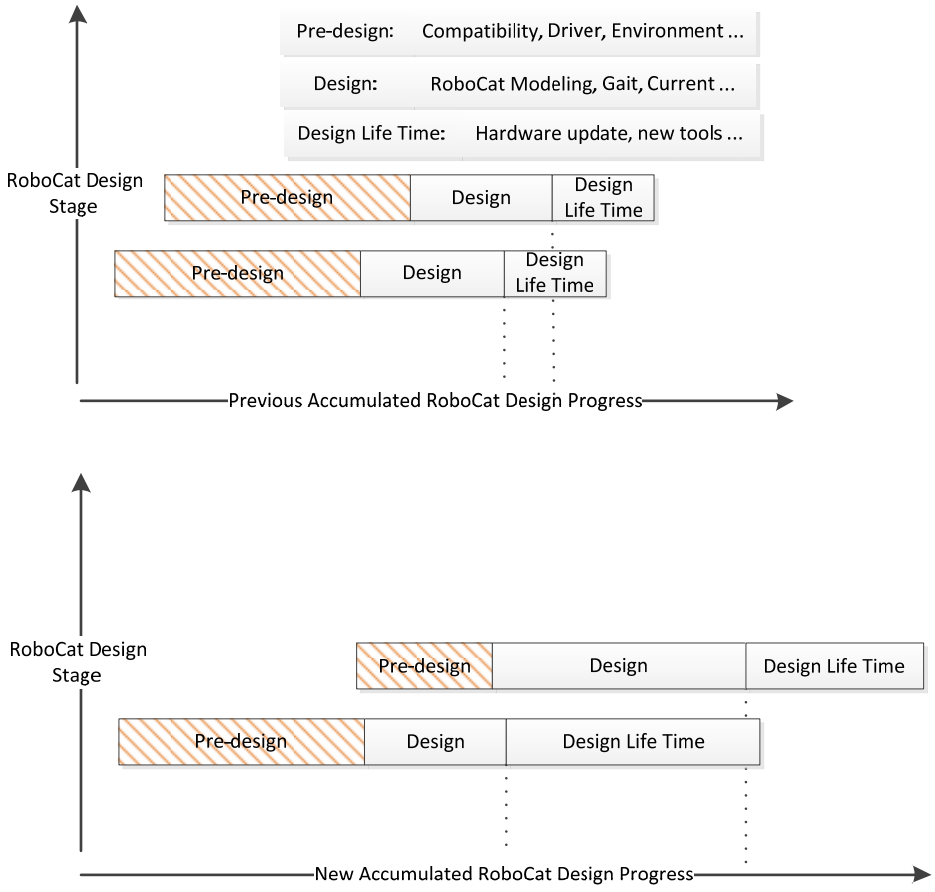

Figure 1. Accumulated RoboCat Design Progress Comparison

Now consider the weight reduction problem. A light central board will be selected for the RoboCat. Meanwhile, the central board needs to be highly integrated to reduce the weight.

2. OS

Most of the popular OS, e.g., the Android, the Linux based OS (Ubuntu, Fedora, etc.), the iOS, and the WinCE, will be considered and compared. The OS matching the requirements provided in the previous chapter best will be selected.

The OS will be tested on the hardware platform of the RoboCat and the necessary modification to the OS will be provided.

3. Hardware platform

One of the main objectives for the hardware platform design is to select the appropriate central boards and to integrate the peripherals of the central boards. The central boards need to i) have abundant interface resources and ii) be light. But it is in general difficult for the two requirements to be fulfilled at the same time. Thus, an evaluation board, which fulfills the requirement i), can be selected for the development purpose, and a light and highly integrated board will be mounted on the RoboCat.

The USB is a standard and widely used port on various platforms and devices. To use the USB as the major communication port for the hardware platform will save the efforts in the interface compatibility problems. Also, the hardware platform will have an extended life cycle and broad compatibility. Thus, the future expansion and

update of the hardware platform will be convenient. Based on these advantages, the USB is selected as the major communication interface for the hardware platform.

Since the OS to be installed on the central board is the Android, the central board will be selected among the ARM based boards, e.g., the Beagleboard, the Beagleboard XM, the Pandaboard, the Tiny210, and the TI evaluation kit.

4. Application software

The application software development comprises the following 2 steps:

i) Development environment set up and

ii) Software coding and onboard validation.

The development environment needs to be open source to facilitate necessary modification. The development tool needs both to be popular and to have good support. In this case, the trouble shooting may be easy since the solutions to the possible problems might already be available from the online resources.

CHAPTER 6: MAIN RESULTS

In this chapter, the main results of the projects are presented and summarized. There are too many technical details in the hardware platform and application software development to be included in the report, and the details have already been documented in separate documents, where the descriptions for these documents are provided in the Section 5 of this chapter. Thus, in this chapter, only the key points that the developers working on the platform developed in this project have to learn are described in detail. Other results may only be briefly summarized.

1. RoboCat functionalities

An Android app is developed to manipulate the servos and thus the gait of the RoboCat. The positions of the servos on the RoboCat can be manipulated on the touchscreen and then be recorded and replayed. The RoboCat developers can continue to tune the gait such that the RoboCat can move forward and turn.

Now consider the weight reduction problem. The weight of the previous design for the RoboCat mainly comprised that of the central board, the servos, and the mechanical parts. The weight of the mechanical parts for the RoboCat may not be easily reduced unless the materials can be replaced and the RoboCat can be re-assembled. Meanwhile, the total weight of the servos is more than 2 lb, which cannot be reduced.

The Galaxy Nexus is to be mounted on the RoboCat to replace the PC-104 central board. The weight of the Galaxy Nexus with the battery included is 4.67 oz, which is 1 lb 1 oz less than that of the PC-104. Thus, the weight of the RoboCat is greatly reduced.

The objectives and the current status of the RoboCat functionalities development are summarized in the following table:

Table 1

The Objectives of the Project and the Status: RoboCat Functionalities

Objectives	**Status**
Step forward and turn	Developed Android apps to enable step forward and turn
Reduce 1 lb of the weight	Selected the Galaxy Nexus (4.67 oz.) as the central board (mounted) for weight reduction, which is more than 1 lb lighter than the previous central board

2. OS

As discussed in Chapter 5, the OS for the RoboCat needs to have a large user base. In this case, the solutions for the problems encountered when building the OS and developing the application software may have already been provided in some public forums. Thus, the trouble shooting can be greatly facilitated. Meanwhile, it is preferred for the OS to be open source, such that the necessary modifications can be made to the OS.

The OS platforms considered include the Linux based OS, e.g., the Ubuntu and the Fedora, the Android, the iOS, and the WinCE. The only OS satisfying all the above requirements is the Android.

Meanwhile, the hardware devices communication might be completed via the USB port. Thus, the USB host mode needs to be available on the OS. The Android OS with the version of 3.1 or higher provides the USB host support. Therefore, the Android 3.1+ OS is selected as the OS for the RoboCat [2].

In this evaluation central board, the Android Jellybean is installed and tested. The kernel of the OS is modified such that the USB host is enabled. The source code of the Android OS is available in [3].

The folder ‘image_folder’ contains the compiled Android OS installation files. The Linux based OS, such as Ubuntu and Fedora, is needed to flash an Android bootable SD card [4].

The procedure to install the Android OS is as follows:

a) Prepare a micro SD card with the volume of at least 2 GB, and then use an SD card to USB adapter to connect the SD card to USB port on the PC;

b) Open a terminal under Ubuntu, type in ‘df -h’. The information ‘/dev/sdX’ will be displayed, where ‘X’ is a lower case letter;

c) In the folder ‘image_folder’, run ‘./flashsbX.sh’, where ‘X’ is the lower case letter obtained from the above step. Meanwhile, to enable theUSB host mode, the file ‘android.hardware.usb.host.xml’ available in the folder ‘image_folder’ needs to be copied to the path ‘/system/etc/permissions’.Then the android boot image will be written to the SD card, where the Android OS can boot up,

d) After the OS image has been written to the SD card, there will be 3 partitions on the SD card. The 3 partitions will be automatically generated by the ‘./flashsbX.sh’ file, where the OS boot information and the OS image will be written. Then, when the Android OS on the SD card needs to be refreshed, the 3 partitions may have to be erased to ensure the successful Android OS refresh. A format tool ‘HP USB Disk Storage Format Tool’ can be used to erase the 3 partitions.

The objectives and the current status of the OS are summarized in Table 2.

Table 2

The Objectives of the Project and the Status: Operating System (OS)

Objectives	**Status**
Select proper OS to facilitate operation and development	Selected the Android OS (version 3.1+ to enable USB OTG)
Test/modify/validate the OS for the RoboCat development	Modified the OS to be compatible with the central board’s display and USB host

3. Hardware platform

The hardware platform design mainly comprises selecting the proper evaluation central board, selecting the proper mounted central board, designing the hardware platform architecture, and evaluating the central board interface resources.

a) Select the proper evaluation central board

Since Android is selected as the OS for the project, the boards to be considered to be the central board are limited to ARM-based platforms, including the Tiny210, the Beagleboard, the Beagleboard XM, the Pandaboard, the Real210, and the TI TMDXEVM3358.

The platform should

- have 4.7” or larger touch screen LCD, and 4.7” LCD is preferred in order to reduce weight,
- have WIFI,
- have UART RS232 and TTL ports,
- support SD and/or JTAG,
- have at least 3 USB host ports, and
- support CAN or I2C, and the CAN bus is preferred since i) the communication between DSP module and ARM-based platform requires high transmission rate, ii) the transmission speed of CAN bus is 100 times over I^2C bus, and iii) most of the new TI DSPs support CAN, where the DSP can be added to the RoboCat hardware platform in the future to implement the feedback control algorithm.

Based on the above requirements, the Beagleboard–XM is selected as the evaluation central board. It is a popular and economic ARM-based development board. Based on [5], the features of the board are listed in Figure 2.

b) Select the mounted central board

Since the application software development for the RoboCat will only be processed in the evaluation central board, the main requirements for the mounted central board are to be light and to be well integrated with the peripherals such as the camera, the speaker, and the touchscreen. In addition, the Android device needs to support the USB on-the-go (OTG) [6], and thus it can serve as a host to send commands to the other hardware devices for the RoboCat.

	Feature	
Processor	OMAP3530DCBB72 720MHz	
POP Memory	Micron	
	2Gb NAND (256MB)	2Gb MDDR SDRAM (256MB)
PMIC TPS65950	Power Regulators	
	Audio CODEC	
	Reset	
	USB OTG PHY	
Debug Support	14-pin JTAG	GPIO Pins
	UART	LEDs
PCB	3.1" x 3.0" (78.74 x 76.2mm)	6 layers
Indicators	Power	2-User Controllable
	PMU	
HS USB 2.0 OTG Port	Mini AB USB connector	
	TPS65950 I/F	
	MiniAB	
HS USB Host Port	Single USB HS Port	Up to 500ma Power
Audio Connectors	3.5mm	3.5mm
	L+R out	L+R Stereo In
SD/MMC Connector	6 in 1 SD/MMC/SDIO	4/8 bit support, Dual voltage
User Interface	1-User defined button	Reset Button
Video	DVI-D	S-Video
Power Connector	USB Power	DC Power
Expansion Connector (Not Populated)	Power (5V & 1.8V)	UART
	McBSP	McSPI
	I2C	GPIO
	MMC	PWM
2 LCD Connectors	Access to all of the LCD control signals plus I2C	3.3V, 5V, 1.8V

Figure 2. Features of BeagleBoard-XM (from the BeagleBoard-XM Rev C System Reference Manual)

Based on the above requirements, the Android based cell phones, e.g., the Galaxy Nexus, the Nexus 4, the Galaxy S II, and the Galaxy S III, are considered and compared.

The Galaxy Nexus is selected as the mounted central board since it supports the USB OTG, is light with the weight of 4.67 oz., and is economic.

c) Design the hardware platform architecture

The hardware platform mainly comprises i) the central board, ii) the servos and the servo controller, iii) the servo current sensor and the related processing board, iv) the peripherals for the audio and video processing, and v) a DSP board for the closed-loop motion control that may be added in the future.

The hardware platform architecture is provided in Figure 3. As illustrated in Figure 2, the central board is the brain of the hardware platform. Other hardware devices, e.g., the Pololu Maestro servo controller and the Arduino Mega board, are integrated to the central board. The central board sends command to the peripherals to manipulate the RoboCat. At the same time, the central board retrieves, processes, and responds to the information from the environment, the peripherals, and the end-users.

d) Evaluate the central boards interface resources

On the Beagleboard-XM, the USB host, USB slave, and TTL port are tested.

For the two devices connected via USB port, one needs to be a USB host and the other one needs to be USB slave. In general, an Android device can always serve

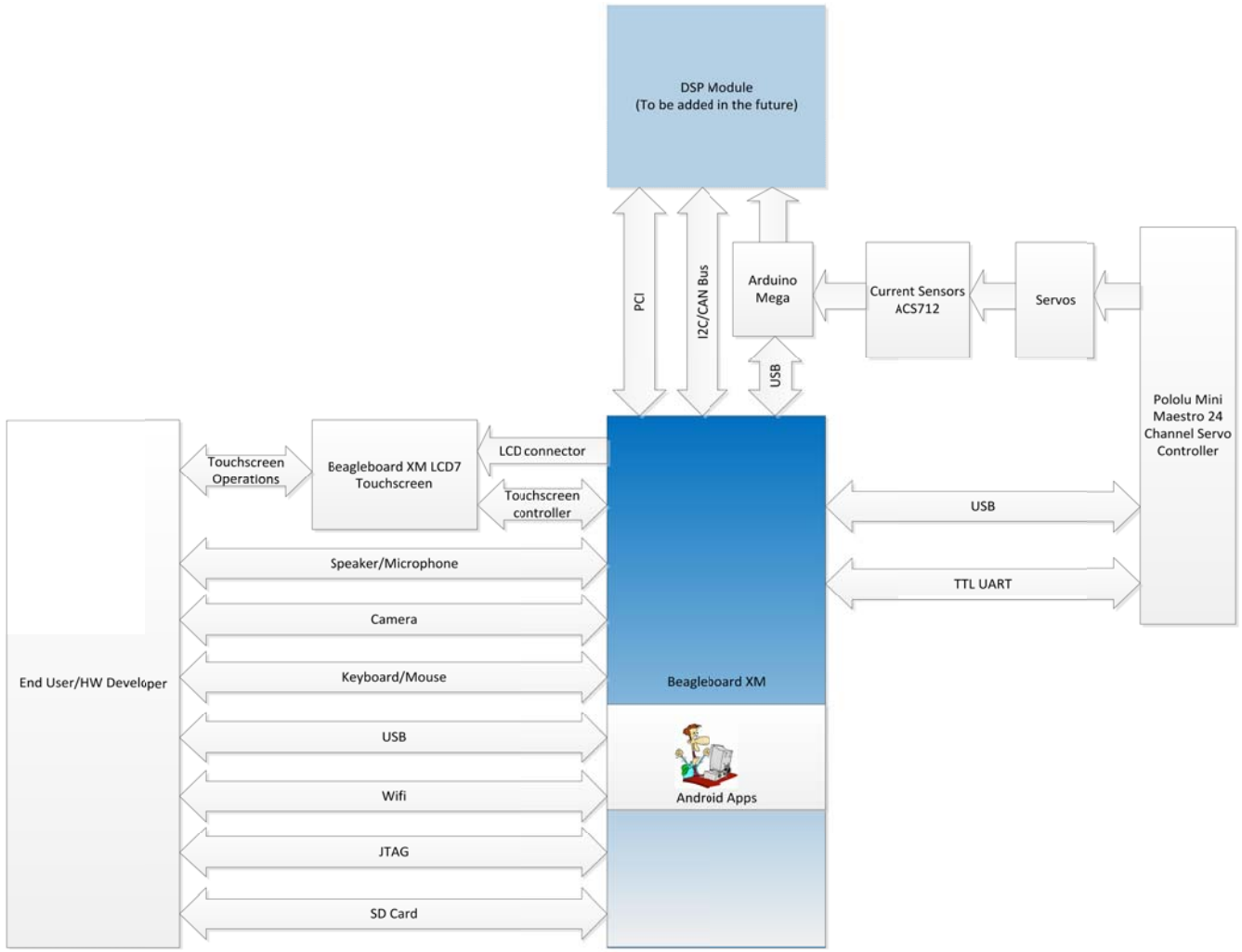

Figure 3. RoboCat hardware platform architecture

as the USB slave but does not have to serve as the USB host. In the RoboCat hardware platform, the Android device needs to function as a USB host device, since it needs to send command to the peripherals.

An Android device with the USB OTG port can always serve as both the USB host and the USB slave. Thus, the USB OTG must function well on the central boards. The USB OTG is validated on both the Beagleboard-XM and the Galaxy Nexus. The objectives and the current status of the hardware platform development are summarized in Table 3.

4. Android apps

Table 3

The Objectives of the Project and the Status: Hardware Platform

Objectives	**Status**
Select proper central board with display, abundant interface resources, low weight, ...	– Evaluation board • Compared Beagleboard XM, Tiny210, Pandaboard, TI Evaluation kit, etc. • Selected Beagleboard XM and Beagleboard touch – Mounted board • Compared Galaxy S ii, S iii, Galaxy Nexus, Nexus 4, etc. (Price, USB OTG, weight, ...) • Selected Galaxy Nexus
Design the hardware platform architecture	The hardware platform architecture was provided
Evaluate the interface resources for the peripherals on both the evaluation board and the mounted board	– Evaluation board • Abundant hardware interface • Tested both USB host and TTL – Mounted board • Validated the USB OTG function

An Android app is developed to validate the communications between the central board and the peripherals. In addition, the Android app is used to provide a replacement of the Pololu Maestro controller center in the Android OS. The Android apps development comprises the following steps:

a) Set up the development environment

The code of the Android apps is mainly written in JAVA by using the Android Developer Tools (ADT) bundle. The ADT bundle is available in the website 'www.android.com' [7], [8]. Meanwhile, the Java development kit (JDK) 6 is needed to compile the Android source code.

b) Software development and validation

The software is developed to detect and identify the specific USB device (Pololu Maestro, and Arduino, etc.), and to manipulate the servos by using the central board touchscreen. Every USB device has a unique vendor ID and a product ID. The apps for the RoboCat need to identify the Pololu Maestro and the Arduino boards, and therefore the vendor ID and the product ID need to be found.

On Windows OS, there is an application software called 'USBVIEW', by using which the USB ID can be identified. In ADT, the USB device ID can be obtained by using the command 'usbManager.getDeviceList()'.

After the vendor ID and the product ID have been found, in order to detect the Pololu Maestro and the Arduino Mega via the Android apps, the file 'device_filter.xml' needs to be created in the path </resource/xml> with the following contents [9]:

```
<resources>
    <!-- Pololu Maestro Servo Controller vendor 8187 and product 140  -->
    <usb-device vendor-id="8187" product-id="140" />
    <!-- Arduino Mega mVendorId=9025,mProductId=66 -->
    <usb-device vendor-id="9025" product-id="66" />
</resources>
```

The software to manipulate the servos on the RoboCat is developed and validated. The user interface (UI) of the software is illustrated as Figure 4, where the 'Record' button is to record the current servo positions, the 'Run Gait' button is to replay the gait, and the 'Reset servos' button is to reset the servos to the neutral positions.

Some skills for the Android apps development are provided in [10].

5. The Documentation and the Source Code Descriptions

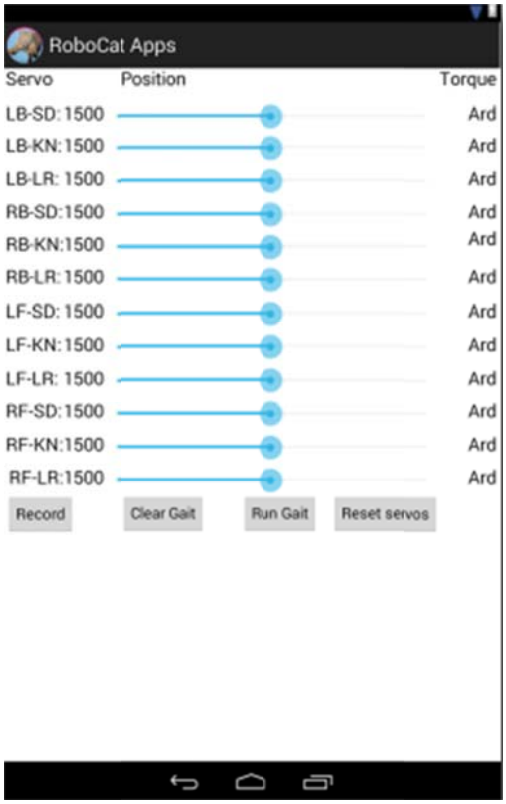

Figure 4. Android apps interface

In this section, the description for the documents of the RoboCat hardware platform and the source code of the Android apps are provided.

All the documents and the source code for the RoboCat are provided in the folder 'Formal RoboCat Harware Platform Docs and Source Code 120113'. The description for the documents and the source code in the above folder is provided in Table 4.

The objectives and the current status of the Android apps development are summarized in Table 5. The android app is developed in the ADT and the Android app

Table 4

The Documents Description

Document or Subfolder	**Description**
Android Apps development skills.doc	This file provides some Android apps development skills
Android apps start.doc	This file describes how to build the Android apps project
Flash Android OS on SD card.docx	This file describes the procedure to flash the Android OS on an SD card for the Beagleboard XM
Linux OS Environment.doc	This file describes the how to build up the Linux OS environment
Pololu Maestros Servo Controller and Arduino Mega.docx	This file provides the information for the Pololu Maestro and the Arduino Mega board
image_folder	This folder contains the Android OS image to be flashed on the SD card for the Beagleboard XM
RoboCatProject	This folder contains the source code developed for the UI and hardware communication Android app
CS_Android_App_Integrate_EE_App	This folder contains the source code of the Android app developed by the CS team, which has integrated the source code in the folder 'RoboCatProject'

Table 5

The Objectives of the Project and the Status: Android Apps

Objectives	**Status**
Set up the development environment	– Built the Android development environment • ADT bundle • JDK
– Develop the application software to enable and validate • communication between central board and the peripherals, and • onboard (RoboCat) manipulation of the servos	– Developed the Android Apps to • enable and detect the specific USB device (Pololu Maestro, Arduino, etc.) • enable the communication between central board and peripherals • provide UI to manipulate 12 servos, record/reset gait, replay gait, etc..
Validate the application software	Validated apps on both BeagleBoard XM and Nexus cell phone

project comprises more than 20 files totally, and thus, only the source code of the main program is provided in the Appendix.

6. The CS RoboCat team Android app

The CS RoboCat team is responsible to develop the Android app in order to provide the UI for the hardware developers, the ender-users, and the multi-touch functions. The hardware communication Android app developed in the project is integrated in the CS Android app. The source code for the integrated Android app is available in the folder ‘CS_Android_App_Integrate_EE_App’.

The UI of the CS Android app is illustrated in Figure 5 to Figure 8. The CS Android app contains 3 submenus, i.e., ‘HW DEV’, ‘END USER’, and ‘MULTI TOUCH’, which are designed for the hardware developers, the end users, and the multi-touch operations, correspondingly. Figure 5 shows the UI for ‘HW DEV’ submenu.

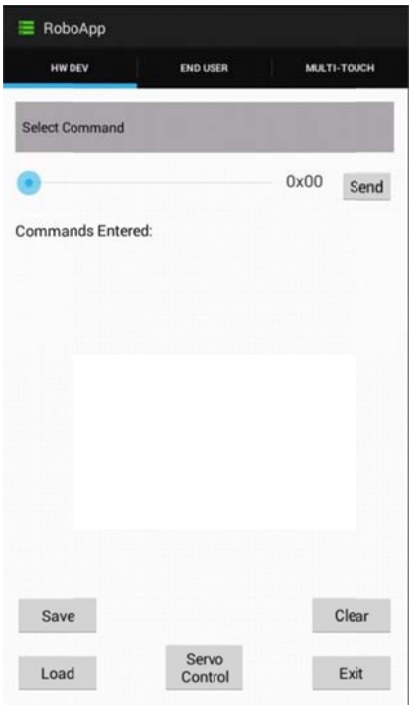

Figure 5. CS RoboCat team Android App Hardware Developer Submenu

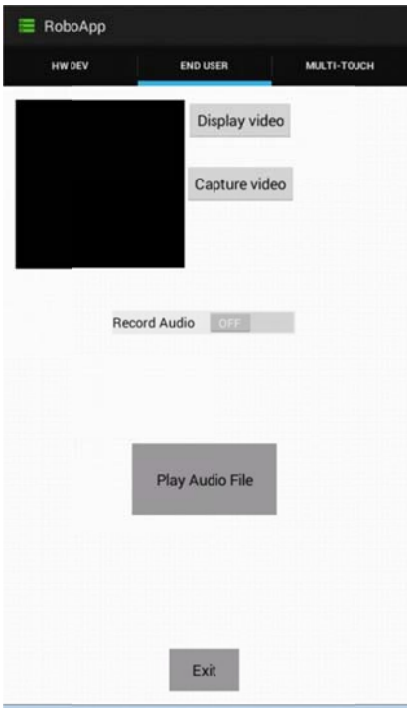

Figure 6. CS RoboCat team Android App End User Submenu

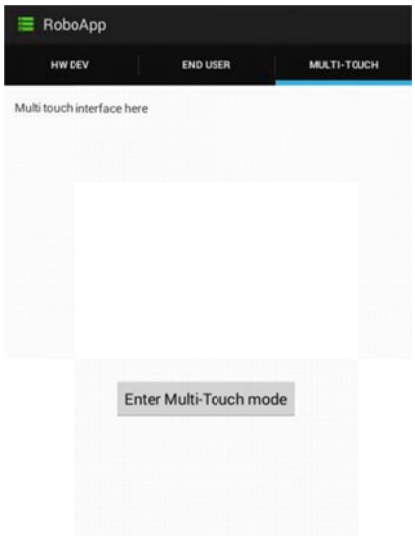

Figure 7. CS RoboCat Team Android App Multi-touch Submenu

Figure 8. CS RoboCat team Android App Multi-touch Interface

By clicking on the ‘Servo Control’ button, the UI in Figure 4 will popup. Since the UI in Figure 4 contains all other functionalities in Figure 5, the remaining buttons and seekbar shown in Figure 5 can be omitted. Figure 6 shows the UI of the submenu for the end users, where the audio and video can be recorded and replayed.

Figure 7 shows the submenu to enter the multi-touch mode and Figure 8 shows the multi-touch operation interface.

In the following content, the method to modify the Android app source code in the folder ‘CS_Android_App_Integrate_EE_App’ is discussed. In the file ‘/RoboApp/src/com/robocat/roboappui/MultiTouchActivity.java’, the following function provides the interface to send the command according to the multi-touch operations:

```
private String Send_command(TypeOfAction Final) {
  // map event to command
  String res = "None";

  String [] a = control.getKnownCommands();
  Context context = getApplicationContext();
  String[] Audio_files;
  Audio_files = getResources().getStringArray(R.array.audio_files);
  try{
      switch (Final){
          case OneDown:
              res=a[0];
              control.sendCommand(res, commandHistory);
              break;
              ……
              }
}
```

where the corresponding command needs to be sent to the Pololu Maestro via the following function in the file ‘/RoboApp/src/com/robocat/roboappui/communication/PololuMaestroUSBCommandProcess.java’:

```
public void sendCommand(byte[] buffer, int length) {
       Log.d(TAG, "sendCommand(" + buffer[2] + ")");

       maestroDevice.sendCommand(buffer, length);
}
```

By modifying the file '`MultiTouchActivity.java`', the commands sent to the Pololu Maestro board can be adjusted.

CHAPTER 7: CONCLUSIONS

This project provided the hardware platform for the RoboCat and developed the Android apps to enable the communications between the central board and the peripherals. Meanwhile, the overall weight of the RoboCat has been significantly reduced.

The OS, the communication method of hardware platform, and the development environment selected in this project ensure that the platform has the extended life time and the desirable expansion and compatibility properties.

Based on the results of this project, future RoboCat developers can directly tune the positions of the servos for the RoboCat to step forward and turn. They can also enhance the Android apps and expand the hardware devices based on the software and the hardware platform provided in this project.

CHAPTER 8: FUTURE WORK

In this chapter, the major future work for the RoboCat design is discussed.

1. The modeling of the RoboCat

One critical problem with the RoboCat design is that, the kinetics and dynamics model of the RoboCat are unavailable. One consequence is that, the tuning of the gait and the trouble shooting may only be completed through tedious experiments. Furthermore, with the adjustment to the RoboCat, the previous procedure may have to be repeated. For example, the servo positions for each leg of the RoboCat are coupling to each other, and thus in some specific positions, the servos may be working against each other.

2. Further weight reduction

Although the weight has been significantly reduced due to the hardware platform design in this project, there is still promising potential in the weight reduction. Further weight reduction does not have to be completed together with significant upgrading, and the weight reduction can be completed through small details upgrading.

3. Gait tuning

Once the modeling of the RoboCat is completed, the gait tuning can be completed first by using the Matlab/SIMULINK and then onboard.

4. Android apps

Though the Android app developed in the project has been integrated with that developed by the CS group for the RoboCat (see, Section 5 in Chapter 6). Further tuning may still be needed.

REFERENCES

[1] “Pololu Maestro Servo Controller User's Guide,” Pololu Corporation, 2012, http://www.pololu.com/docs/0J40
[2] “USB Host," Google Inc., 2013, http://developer.android.com/guide/topics/connectivity/usb/host.html
[3] “Rowboat BeagleBoard”, Rowboat google group, 2013, https://groups.google.com/forum/#!forum/rowboat
[4] Linux OS Environment.doc, *Formal RoboCat Harware Platform Docs and Source Code 120113*
[5] “BeagleBoard-xM Rev C System Reference Manual Revision 1.0” Apr. 2010, http://beagleboard.org/static/BBxMSRM_latest.pdf
[6] “USB_On-The-Go”, 2013, http://en.wikipedia.org/wiki/USB_On-The-Go
[7] “Android Developer Tools”, Google Inc., 2013, http://developer.android.com/tools/help/adt.html
[8] Android apps start.doc, *Formal RoboCat Harware Platform Docs and Source Code 120113*
[9] Flash Android OS on SD card.docx, *Formal RoboCat Harware Platform Docs and Source Code 120113*

[10] Android Apps development skills.doc, Formal RoboCat Harware Platform *Docs and Source Code 120113*

APPENDIX: THE SOURCE CODE OF THE ANDROID APP'S MAIN PROGRAM

```
/*
 * This is the main activity file for the RoboCat project. In this file, the
following tasks can be completed:
 * i)  detect the correct Pololu Maestro USB device;
 * ii)       detect the correct Arduino Mega USB device;
 * iii) obtain the progress from the seek bar, and map the no. of the seek bar
to the correct Pololu Maestro channel
 * iv) complete the correct functionality for the RoboCat, e.g., sending
command to the Pololu Maestro, record the gait values
 *           for the RoboCat, reset the servos, ...
 * Author: Haiquan Zhang
 */

package com.robocatpackage;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.channels.FileChannel;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.Timer;
import java.util.TimerTask;

import android.widget.Toast;

import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.app.Activity;
import android.view.Menu;
import android.content.Context;
import android.content.Intent;
//import android.content.ServiceConnection;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbManager;
import android.text.format.DateFormat;
```

```
import android.text.format.Time;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;

import com.robocatpackage.R;


public class RoboCatActivity extends Activity implements View.OnClickListener,
SeekBar.OnSeekBarChangeListener {
	private Button resetServosButton, recordButton, clearGaitButton,
runGaitButton;
	private static final String TAG = "MaestroSSCActivity";
	private PololuMaestroUSBCommandProcess maestroSSC;
   // private Handler mHandler = new Handler();
	int channelCount = 12;
	// the seekBar range is [900, 1500], and there is no way to modify the
beginning value of the seek bar. therefore, the actual progress value and the
	// seekbar progress value are recorded separately
	int progressResetActual =1500;
	int progressOffset =900;
	// int array to record the current gait values
	int[] arrayGaitChannelProgress = new int[2*(channelCount+1)];
	//ArrayList<Integer> arrayGaitChannelProgress = new
ArrayList<Integer>(Collections.nCopies(2*channelCount, -1));
	TextView[] textViewChannelPositionArray = new TextView[channelCount];
	private SeekBar[] channelPositionBarArray = new SeekBar[channelCount];
	//the map between the seekbar no. and the pololu maestro channel
	int[] channelNoMapArray=new int[]{1,2,3,5,6,7,12,13,14,16,17,18};

	String gaitDefaultFileName = "GaitShared.txt";

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		View view = getLayoutInflater().inflate(R.layout.robocat_layout,
null);
		setContentView(view);
		// initialize the gait array value to -1. declare in the
beginning, but have to initialize here
		Arrays.fill(arrayGaitChannelProgress,-1);

		resetServosButton = (Button)
view.findViewById(R.id.reserServosButtonid);
		resetServosButton.setOnClickListener(this);
		recordButton = (Button)
view.findViewById(R.id.buttonpololurecordid);
		recordButton.setOnClickListener(this);
		clearGaitButton = (Button)
view.findViewById(R.id.clearGaitButtonid);
```

```
            clearGaitButton.setOnClickListener(this);
            runGaitButton = (Button) view.findViewById(R.id.runGaitButtonid);
            runGaitButton.setOnClickListener(this);


            // Initialize the textView and seekBar
            for(int i = 0; i < channelCount; i++) {
                      textViewChannelPositionArray[i] = new TextView(this);
                      channelPositionBarArray[i] = new SeekBar(this);
                  }


            channelPositionBarArray[0] = (SeekBar)
view.findViewById(R.id.channel1PositionBar);
            channelPositionBarArray[1] = (SeekBar)
view.findViewById(R.id.channel2PositionBar);
            channelPositionBarArray[2] = (SeekBar)
view.findViewById(R.id.channel3PositionBar);
            channelPositionBarArray[3] = (SeekBar)
view.findViewById(R.id.channel4PositionBar);
            channelPositionBarArray[4] = (SeekBar)
view.findViewById(R.id.channel5PositionBar);
            channelPositionBarArray[5] = (SeekBar)
view.findViewById(R.id.channel6PositionBar);
            channelPositionBarArray[6] = (SeekBar)
view.findViewById(R.id.channel7PositionBar);
            channelPositionBarArray[7] = (SeekBar)
view.findViewById(R.id.channel8PositionBar);
            channelPositionBarArray[8] = (SeekBar)
view.findViewById(R.id.channel9PositionBar);
            channelPositionBarArray[9] = (SeekBar)
view.findViewById(R.id.channel10PositionBar);
            channelPositionBarArray[10] = (SeekBar)
view.findViewById(R.id.channel11PositionBar);
            channelPositionBarArray[11] = (SeekBar)
view.findViewById(R.id.channel12PositionBar);
            // set listeners for channelPositionBar
            for(int i = 0; i < channelCount; i++) {

      channelPositionBarArray[i].setOnSeekBarChangeListener(this);
            }



            textViewChannelPositionArray[0] = (TextView) findViewById
(R.id.channel1positionvalue);
            textViewChannelPositionArray[1] = (TextView) findViewById
(R.id.channel2positionvalue);
            textViewChannelPositionArray[2] = (TextView) findViewById
(R.id.channel3positionvalue);
            textViewChannelPositionArray[3] = (TextView) findViewById
(R.id.channel4positionvalue);
```

```
            textViewChannelPositionArray[4] = (TextView) findViewById
(R.id.channel5positionvalue);
            textViewChannelPositionArray[5] = (TextView) findViewById
(R.id.channel6positionvalue);
            textViewChannelPositionArray[6] = (TextView) findViewById
(R.id.channel7positionvalue);
            textViewChannelPositionArray[7] = (TextView) findViewById
(R.id.channel8positionvalue);
            textViewChannelPositionArray[8] = (TextView) findViewById
(R.id.channel9positionvalue);
            textViewChannelPositionArray[9] = (TextView) findViewById
(R.id.channel10positionvalue);
            textViewChannelPositionArray[10] = (TextView) findViewById
(R.id.channel11positionvalue);
            textViewChannelPositionArray[11] = (TextView) findViewById
(R.id.channel12positionvalue);



            UsbManager manager = (UsbManager)
getSystemService(Context.USB_SERVICE);
            HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
            maestroSSC = new PololuMaestroUSBCommandProcess(manager);


      }


      // function to toastMessage
      private void toastMessage(String string) {
            // TODO Auto-generated method stub
            Toast.makeText(getApplicationContext(),
               string, Toast.LENGTH_LONG).show();

      }

      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is
present.
            getMenuInflater().inflate(R.menu.robo_cat, menu);
            return true;
      }
      // resume actions
      @Override
      public void onResume() {
            super.onResume();
            Intent intent = getIntent();
            String action = intent.getAction();
            if (action.equals("android.intent.action.MAIN")) {
           Toast.makeText(getApplicationContext(),
```

```
                        "Reconnect the USB devices.", Toast.LENGTH_LONG).show();
                finish();
                 }

                 if
(action.equals("android.hardware.usb.action.USB_DEVICE_ATTACHED")|action.equal
s("android.intent.action.MAIN")) {
                        UsbDevice device = (UsbDevice)
intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
                        if
(UsbManager.ACTION_USB_DEVICE_ATTACHED.equals(action)) {
                        Log.d(TAG, "DEVICE TO ATTACH:" + device.toString());
                        maestroSSC.setDevice(device);
                 Toast.makeText(getApplicationContext(),
                        "USB device connected.", Toast.LENGTH_LONG).show();

                        } else if
(UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action)) {
                       Log.d(TAG, "DEVICE TO ATTACH: " + device.toString());
                        maestroSSC.setDevice(device);
                 Toast.makeText(getApplicationContext(),
                        "USB device disattached.", Toast.LENGTH_LONG).show();
                        } else {
                        Log.d(TAG, "Unknonw error: =" + action.toString());

                        }
                 }
                 else
                 {
                Toast.makeText(getApplicationContext(),
                        "No valid USB device found!", Toast.LENGTH_LONG).show();
                 }
        }

        /**
         * Handle button click
         */
        @Override
        public void onClick(View v) {
               Log.d(TAG,"onClick(" + v.getId() + ")");
               // Get button id first
               final Button b = (Button) v;
            String buttonText = b.getText().toString();


               if (v.equals(resetServosButton))
               {
                        resetServos();

               }
               else if (v.equals(recordButton))
               {
```

```
                recordGaitButtonAction();
            }
            else if (v.equals(runGaitButton))
            {
                try {
                    runGaitButtonAction();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            else if (v.equals(clearGaitButton))
            {
                clearGaitButton();
            }
        }

        // clear the Gait
        public void clearGaitButton()
        {

            // test: delete the Gait.txt file

            File root = new File(Environment.getExternalStorageDirectory(),
"Gait");
          if (!root.exists()) {
              root.mkdirs();
          }
          //File gpxfile = new File(root, "Gait"+currentDateandTime+".txt");
          File gpxfile = new File(root, gaitDefaultFileName);

            boolean deleted = gpxfile.delete();


        }

        public void runGaitButtonAction() throws IOException {
            // TODO Auto-generated method stub

///////////////
Runnable r = new Runnable() {
@Override
public void run()
{
        try
        {
        File root = new File(Environment.getExternalStorageDirectory(), "Gait");
    if (!root.exists())
        {
          root.mkdirs();
        }
    File gpxfile = new File(root, gaitDefaultFileName);
```

```
      BufferedReader buffReader = new BufferedReader(new InputStreamReader(new
FileInputStream(gpxfile)));
      String line;
    while ((line = buffReader.readLine()) != null)
      {
      int[] gaitLineVal=parseGait(line);
      oneGaitAction(gaitLineVal);
      for (int iloop =0; iloop<100000000; iloop++){}
      }

    buffReader.close();

      }
            catch (IOException e)
                  {
            // TODO Auto-generated catch block
            e.printStackTrace();
                  }
      }
};

Handler h = new Handler();
h.postDelayed(r, 1000); //  delay in milliseconds.
//////////////////////


}




      private void oneGaitAction(int[] gaitLineVal) {
            // TODO Auto-generated method stub
      for (int iRunGait=0, lenRunGait = gaitLineVal.length/2; iRunGait
<lenRunGait; iRunGait++)
      {
            // double iRunGait to point to the right channel
            int iRunGaitDouble = iRunGait*2;
            if (gaitLineVal[iRunGaitDouble] == -1) continue;
            if (gaitLineVal[iRunGaitDouble] <0 )
                        {
                  break;
                  }
            else
            {
                  final int channelNoMapped = gaitLineVal[iRunGaitDouble];
                  final int progressActual = gaitLineVal[iRunGaitDouble+1];
                  int channelNoSeekBar=iRunGaitDouble/2;
                  progressChangeAction(channelNoMapped, progressActual,
channelNoSeekBar);
```

```
            }

      }

      }


      private void progressChangeAction(int channelNoMapped, int
progressActual, int channelNoSeekBar)
      {
            maestroSSC.setServoPosition(channelNoMapped, progressActual);
            // modify the offset for the progress of the seek bar
            int progressSeekBar = progressActual - progressOffset;

      channelPositionBarArray[channelNoSeekBar].setProgress(progressSeekBar);

      textViewChannelPositionArray[channelNoSeekBar].setText(String.valueOf(pr
ogressActual));
      }

      private int[] parseGait(String line) {
            // TODO Auto-generated method stub

            String[] byteValues = line.substring(0, line.length() -
1).split(",");

            //String[] byteValues = arr.replaceAll("\\[",
"").replaceAll("\\]", "").split(",");

            int[] results = new int[byteValues.length];

            for (int i = 0; i < byteValues.length; i++) {
                try {
                    results[i] = Integer.parseInt(byteValues[i]);
                } catch (NumberFormatException nfe) {};
            }

            return results;
      }


      private void recordGaitButtonAction() {
            // TODO Auto-generated method stub
            generateGaitOnSD(gaitDefaultFileName, arrayGaitChannelProgress);

      }

      public void generateGaitOnSD(String sFileName, int[]
arrayGaitChannelProgress){
          try
          {
              File root = new File(Environment.getExternalStorageDirectory(),
```

```
"Gait");
                if (!root.exists()) {
                    root.mkdirs();
                }
                File gpxfile = new File(root, sFileName);
                Toast.makeText(this, "Saved", Toast.LENGTH_SHORT).show();
                BufferedWriter outputWriter = null;
                if (gpxfile.exists())
                {
                     outputWriter = new BufferedWriter(new
FileWriter(gpxfile,true));

                }
                else
                {
                    outputWriter = new BufferedWriter(new FileWriter(gpxfile));
                }

                //outputWriter = new BufferedWriter(new FileWriter(gpxfile));
                for (int i = 0; i < arrayGaitChannelProgress.length; i++) {

                     {outputWriter.append(arrayGaitChannelProgress[i]+",");}
                }
              outputWriter.newLine();
                 outputWriter.flush();
                 outputWriter.close();
             }
             catch(IOException e)
             {
                  e.printStackTrace();
             }
            }

// function to reset servos
       public void resetServos() {
              for (int channelNoSeekBar = 0; channelNoSeekBar < channelCount;
channelNoSeekBar++)
              {
                     // obtain the mapped channelNo
                     int channelNoMapped = channelNoMapArray[channelNoSeekBar];
                           progressChangeAction(channelNoMapped,
progressResetActual, channelNoSeekBar);
                     }

       }

       /**
        * SeekBar Listener
        */
       public void onProgressChanged (SeekBar seekBar, int progressSeekBar,
boolean fromUser) {
              int progressActual = progressSeekBar + progressOffset;
```

```
            Log.d(TAG,"onProgressChanged(" + seekBar.getId() + ", " +
progressActual + ")");
            for (int channelNoSeekBar = 0; channelNoSeekBar < channelCount;
channelNoSeekBar++)
            {
                  if
(seekBar.equals(channelPositionBarArray[channelNoSeekBar])) {
                        // save position
                        int channelNoMapped;
                        channelNoMapped =
channelNoMapArray[channelNoSeekBar];

      arrayGaitChannelProgress[channelNoSeekBar*2]=channelNoMapped;

      arrayGaitChannelProgress[channelNoSeekBar*2+1]=progressActual;
                        // pololu operation

                       progressChangeAction(channelNoMapped, progressActual,
channelNoSeekBar);
                        break;
                  }

            }
      }

      /**
       * SeekBar Listener
       */
      @Override
      public void onStartTrackingTouch(SeekBar seekBar) {
      }

      /**
       * SeekBar Listener
       */
      @Override
      public void onStopTrackingTouch(SeekBar seekBar) {
      }
}
```