CHAPTER 5

TESTING & RESULTS
5.1 Testing work on Emulator

Newly build custom code can be tested on emulator as per following:

Step 1
Building Environment Setup and setting path.

![Figure 5.1 – Running envsetup and setting path](image-url)
Step 2:

Creating SD card

Figure 5.2 – Creating SD Card

mksdcard command can be used to create sdcard as per the requirement. The arguments are size of sdcard and name of the sdcard. Here the command will create an sdcard of 256 MB size and with name sdcard.img
Step 3
Running the emulator with SD Card

The emulator command will run the emulator for custom OS. The -sdcard option if used will require name of the sdcard of be used.

Here emulator will be started with sdcard.img (one which was created in last step)

Note: Emulator can be launched without sdcard but since we need to capture image, we need sdcard for storing captured images and even camera cannot be launched without sdcard.
The emulator screen

![Emulator Screen](image)

**Figure 5.4 – Emulator screen**

Above figure displays android 2.2 emulator with screen, navigation and multimedia buttons and keypad.

The android emulators may differ based on Android OS versions. Here since it is for 2.2 it will look like as a phone having android 2.2 OS.

The emulator is a virtual environment and it behaves same as a device on machine. There are many advantages of using virtual environment (emulator) to test the application or customized code before deploying it to live device. This not only reduces deployment time and bugs but it also can be used to test the application in similar way as it can be done on live device.
When you unlock device by swiping from left to right on the lock icon as shown below (it behaves same as we do in an actual device):

![Unlocking emulator](image1)

**Figure 5.5 – Unlocking emulator**

you will be able to see desktop of the device with 3 options at bottom as shown in following figure.

![Desktop of emulator](image2)

**Figure 5.6 – Desktop of emulator**
Step 4
Select Camera

Once you click on center button (Menu) in above figure you can see icons of applications installed on your device as shown below.

Figure 5.7 – Launching Camera from Emulator
Clicking on camera 4th item in first row will launch the camera as shown below:

![Camera Emulation from Android Emulator](image)

**Figure 5.8** – Camera Emulation from Android Emulator

This is emulation of camera in virtual environment so one cannot take pictures as it can be done using actual camera. No live preview is available here in virtual environment.
When you click on capture button (left most button on the bottom bar), it will capture an image and will display its preview on the right most place holder as shown below:

![Image of camera interface showing captured image]

**Figure 5.9 – Capturing image using Camera**

This is emulation of camera so a default icon has been displayed as captured image preview. The process emulates actual hardware emulation and confirms that camera works properly after changing the code (camera.java).

If there is any error in any of the steps the emulator will raise an exception and will kill the running process forcefully. Here since it worked properly and we are able to capture an image it means the code will work in similar way on the actual device also.
Step 5
Check details of captured image.

To verify that the image is captured successfully and to get the image detail for later use we have to check the image in Gallery.

Gallery can be opened from main menu (Dashboard) as shown below:

![Opening Gallery](image)

**Figure 5.10 – Opening Gallery**
Once you click on Gallery Icon (1st item in the third row) we can see all the albums available there.

The camera album contains images and since this is newly created sdcard there will be only one album (camera) with only one image (the image which we captured in last step).

After clicking on the image it will be seen in full size as shown below:
Figure 5.11 – Viewing Captured Image
When you click on Menu you can see different options like share, delete, more, etc... as shown below

![Image of Android interface with menu options]

**Figure 5.12** – Checking details of the captured image

Clicking on more will expand the options as can be seen in above figure.
The detail screen displays all the details related to selected image.

Here the title of the image represents name of the image which is IMGP_20130718_191156 in our case. The default naming convention for the images are IMG_YYYYMMDD_HHMMSS in android 2.2 devices.

To reflect customization I have changed the naming convention by adding P after IMG, so now all the images capture by the device running my custom OS will have IMGP instead of IMG as a suffix.

Here the extension on the image is PNG so filename of the image captured is IMGP_20130718_191156.png
Checking Phone Number:

![Mobile Phone Number of Running Emulator](image)

**Figure 5.14** – Mobile Phone Number of Running Emulator
5.2 Result

In following section custom Android OS has successfully captured and saved image to the SD card so now in this section I will confirm that work done has achieved expected work.

Step 1

Going to bin directory for getting all commands:

Figure 5.15 – Navigating to /out/host/linux-x86/bin directory
Step 2:

Starting ADB to check attached devices:

![ADB Starting](image_url)

**Figure 5.16** – Starting ADB
Step 3:

Starting ADB Shell:

Figure 5.17 – Starting ADB Shell
Step 4
Listing Directories

Figure 5.18 – Listing Directories of SD Card on Device
Step 5
Navigating to DCIM directory to get exact file name of captured image.

Figure 5.19 – Checking Name of The Captured image
Step 6

Copying Image file from Device's SDCard to local computer using pull command:

```
root@parag-ubuntu:/home/parag/WORKING_DIRECTORY/out/host/linux-x86/bin# /adb pull /mnt/sdcard/DCIM/Camera/IMG_20130718_191156.png /home/parag/Drive1_Parag/phd/Thesis/
109 KB/s (8733 bytes in 0.078s)
```

**Figure 5.20** – Copying file from device to local computer
Step 7

Running independent desktop application to check the results. All the files are compiled and compilation steps are not discussed here. Steps for testing the work and confirming result using independent application are shown below:

7.1
Launching Application : Application can be launched using controller MyStegoC

![Figure 5.21 – Running decode application](image-url)
7.2 First screen of the application

![First screen of decode application](image)

**Figure 5.22** – First screen of decode application

The application launches with an alert to inform user about its usage. This works as a help tip for a new user.
7.3 Selecting file to decode.

![Figure 5.23 – Selecting file to decode](image)

When user clicks on decode menu item from the menu application will prompt it with a file select dialog box. Here I have selected the file captured in last section. Name of the file is IMGP_20130718_191156.png and it is in Thesis folder.
7.4 Preview of the selected file.

Preview of the selected file will be shown in the application as shown below:

![Preview of selected image](image)

**Figure 5.24** – Preview of selected image

The preview confirms that the image which we have selected to decode is the same and also it is not possible to know whether the image contains any hidden text or not.
7.5 Decoding selected image

Once you click on the Decode Now button in previous screen it will call decode function of steganography.java file and will check the file for hidden text.

![The Image was decoded Successfully!](image)

**Figure 5.25** – Message after successful decode

If algorithm can search / extract the hidden message in the file then it will display a message as shown above. There are two other possibilities while it decodes. In case if there isn't any hidden text in the image then it will display cannot decode and if the image is corrupted or not readable then it will display message to inform user that application cannot read the selected image.
7.6 Result Screen

![Image](image.png)

**Figure 5.26 – Output**

Once the image is decoded successfully it will display appropriate message on the place holder. The message here displays mobile number of the user who captured this image from his / her android based phone.

This confirms that the image we captured in previous section had mobile number of the user hidden in it.

Thus the steganographic technique which I created to hide origin (mobile phone number) of the user who captures image gets embedded successfully on all images captured from the native camera of the same phone.

If any image captured from the android phone running the Android OS which I
customized in previous section is used for any criminal purpose then identify of the user who captured the image can be revealed using this complementary decoding application.