Contents

Testing

The Application

Real World Resting

To validate our efforts, the application was tested again on all emulators after 5.0, and also on real mobile phones running android versions 6.0 and 7.0 respectively.

This led to further problems however; In running the devices on real mobile phones, it was discovered that while the multidexing problems were resolved, the application would crash when the emergency services call button functionality was attempted. This confused the team as the emulators ran the function without problem.

After some testing of this specific functionality it was noticed that the application required permissions to be turned on in the settings of the real device for “Phone call access”. After this, the call function works perfectly, as it does on the emulator.

Some other problems occurred in the testing of the application for the first time.

It was clear there was something not quite right with the Boolean values of the notification toggle button, as there were times when notifications were not being received, yet the button was toggled on. This was followed up with more detailed instrument testing in android studio.

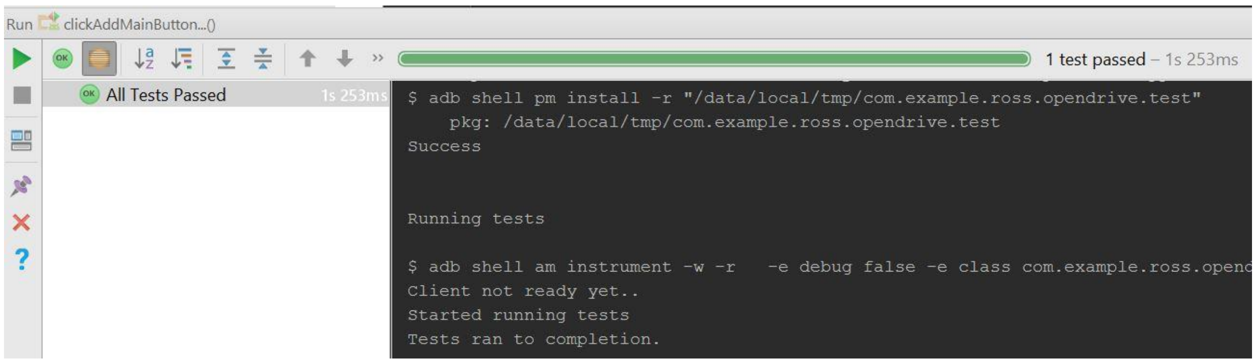
As well as this, there was a problem which would result in the entered neighbor’s number not matching that of the edit text and as such, the number entered on the quick text screen did not always match that of the user entered number.

Aside from this, the first look at the application did not show any immediately obvious issues.

Automated Instrument / User Interface Tests

The following are areas highlighted from our real world testing, that needed to be examined through the more detailed lens of espresso test units.

<u>Test 1: Button remained visible after being clicked repeatedly in quick succession.</u>

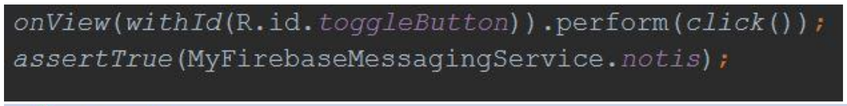


```
$ adb shell pm install -r "/data/local/tmp/com.example.ross.opendrive.test"
    pkg: /data/local/tmp/com.example.ross.opendrive.test
Success


Running tests

$ adb shell am instrument -w -r   -e debug false -e class com.example.ross.openc
Client not ready yet..
Started running tests
Tests ran to completion.
```

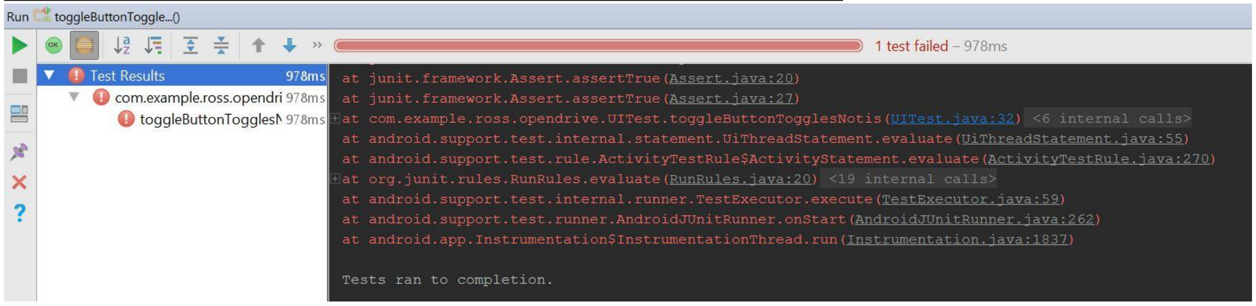
<u>Test 2: Checking Boolean Value Changes Correctly on click of toggle button</u>

Expected Result: Failure

Result: Failure (VALIDATED)



```
onView(withId(R.id.toggleButton)).perform(click());
assertTrue(MyFirebaseMessagingService.notis);
```


Run toggleButtonToggle...()
1 test failed – 978ms
Test Results 978ms
com.example.ross.opendri 978ms
toggleButtonTogglesN 978ms

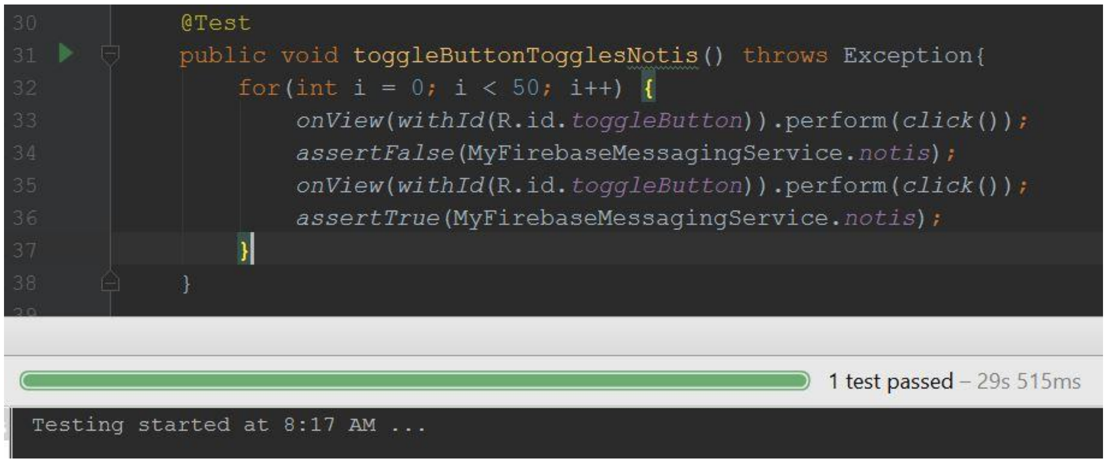
```
at junit.framework.Assert.assertTrue(Assert.java:20)
at junit.framework.Assert.assertTrue(Assert.java:27)
at com.example.ross.opendrive.UITest.toggleButtonTogglesNotis(UITest.java:32) <6 internal calls>
at android.support.test.internal.statement.UiThreadStatement.evaluate(UiThreadStatement.java:55)
at android.support.test.rule.ActivityTestRule$ActivityStatement.evaluate(ActivityTestRule.java:270)
at org.junit.rules.RunRules.evaluate(RunRules.java:20) <19 internal calls>
at android.support.test.internal.runner.TestExecutor.execute(TestExecutor.java:59)
at android.support.test.runner.AndroidJUnitRunner.onStart(AndroidJUnitRunner.java:262)
at android.app.Instrumentation$InstrumentationThread.run(Instrumentation.java:1837)

Tests ran to completion.
```

Comments: This must mean that the above notification issue was simply due to network problems at the time which of course are out of our control, however to be sure the function was stress-tested.

<u>Test 3: Ensuring the above holds through under stressful extremes (Boundary Value Testing)</u>

Expected Result: Success

Result: Success (VALIDATED)



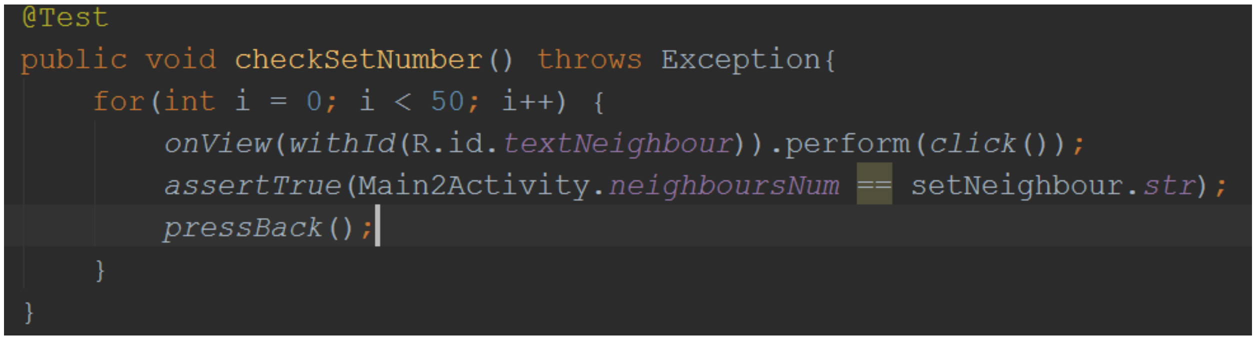
```
@Test
public void toggleButtonTogglesNotis() throws Exception{
    for(int i = 0; i < 50; i++) {
        onView(withId(R.id.toggleButton)).perform(click());
        assertFalse(MyFirebaseMessagingService.notis);
        onView(withId(R.id.toggleButton)).perform(click());
        assertTrue(MyFirebaseMessagingService.notis);
    }
}
```

```
1 test passed – 29s 515ms
Testing started at 8:17 AM ...
```

<u>Test 4: Checking Neighbor number always matches input of editText</u>

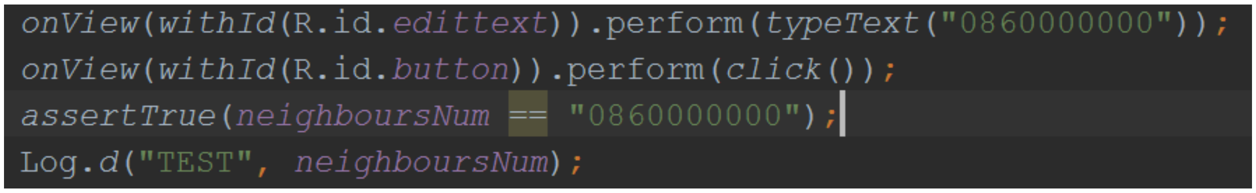
Expected Result: Success

Result: Failure (INVALID)



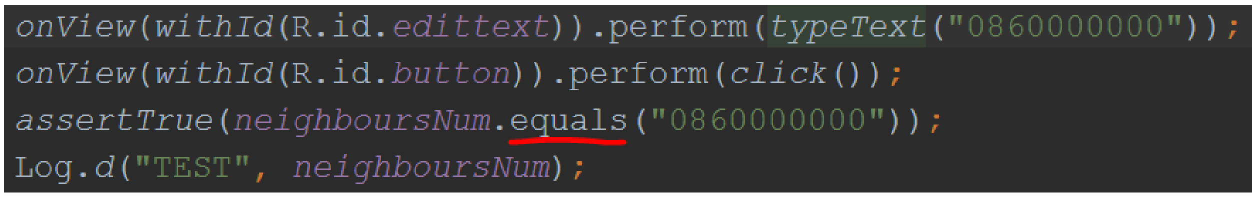
```
@Test
public void checkSetNumber() throws Exception{
    for(int i = 0; i < 50; i++) {
        onView(withId(R.id.textNeighbour)).perform(click());
        assertTrue(Main2Activity.neighboursNum == setNeighbour.str);
        pressBack();
    }
}
```

For some Reason, the neighbor number variable was not being set when I clicked the button after editing the editext. I had a suspicion at this point, that this is to do with the recently added shared preferences, which are necessary to save data even when the app is shut down.



```
onView(withId(R.id.edittext)).perform(typeText("0860000000"));
onView(withId(R.id.button)).perform(click());
assertTrue(neighboursNum == "0860000000");
Log.d("TEST", neighboursNum);
```

For some reason this code resulted in a failed test, however when I printed the neighboursNum, variable in the log as can be seen below, the number matched perfectly, so even though the test was a fail on android studio, as far as the functionality goes as far as I was concerned it passed. However, the error was in the testing and not in the code itself, as when I realized the above highlighted error, and changed the code to the following:



```
onView(withId(R.id.edittext)).perform(typeText("0860000000"));
onView(withId(R.id.button)).perform(click());
assertTrue(neighboursNum.equals("0860000000"));
Log.d("TEST", neighboursNum);
```

Test passed.

Test 4 Amended: Validated

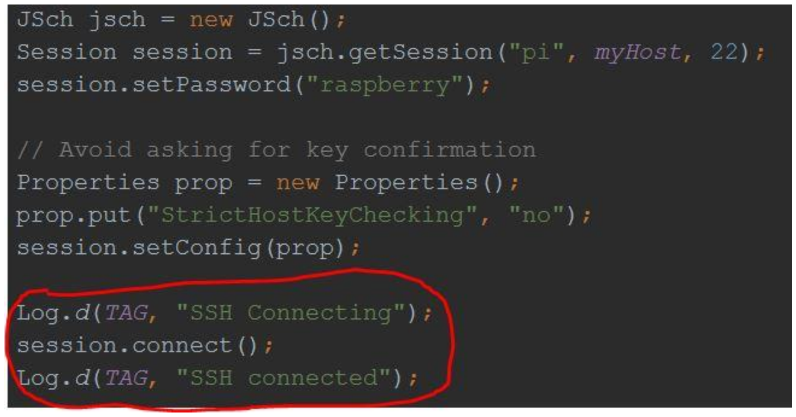
Functionality testing & Verification

Throughout the development of the project, after each minor change to a build, manual Unit testing was carried out, and the task/activity affected by the changes was repeated multiple times under different scenarios to try to produce an error.

As the application's code is not particularly algorithmically complicated, and is used largely as a control for the raspberry Pi, we felt unit testing programmatically, while beneficial had there been more time, was not as important as testing the User Interface and various interactions that occur. Also, the application is small enough for the code to be tested manually without a large time-sink. For example, if a revision saw changes to the openFileActivityBuilder (which accesses a user's google drive), the corresponding test was to run through the full functionality of the method containing the changed code, so opening the drive, selecting a file, opening a file in app, deleting a file, and ensuring that the process can be repeated without faults. This generated a lot of crashes at the beginning of our development phase but as the project developed these were ironed out.

Activity Testing

After each major build, the team would do a clean install of the application and run through the whole installation process with different values for the neighbor number, IP addresses and the various options of the alarm sound available, to ensure the SSH connection was working each time, and that changes had not effected the applications interaction with the Pi. Log statements were used in android studio using the TAG of the relevant class. Here is an example of using log statements to validate the working state of the SSH connection.



```
JSch jsch = new JSch();
Session session = jsch.getSession("pi", myHost, 22);
session.setPassword("raspberry");

// Avoid asking for key confirmation
Properties prop = new Properties();
prop.put("StrictHostKeyChecking", "no");
session.setConfig(prop);

Log.d(TAG, "SSH Connecting");
session.connect();
Log.d(TAG, "SSH connected");
```

The team tested each location in code whereby there was a migration from one activity to another.

The team used principles from boundary value testing, and input values that were extreme cases.

The following errors Occurred:

Problems Identified using this method, Solution, and Validation:

Error 1: This extreme value testing alerted the team that the set neighbor number functionality would accept numbers of any value.

Amendment: After adding appropriate controls to only accept numbers of length 10, and re-testing, the tests all behaved as expected.

Re – Test: Success

Error 2: Regardless of whether the neighbor number had been set up, upon closing and opening of the application it would not remain stored, meaning it would have needed to be set up upon every startup of the app.

Amendment: Used SharedPreferences Class to store data rather than simple Strings.

Re – Test: Success

Error 3: Each time the main activity page was loaded it would repeat the message (Registered for notifications).

Amendment: A flag was set in the onPause method and while this flag is set, the message will not be displayed.

Re – Test: Success

Structural Testing

Main2Activity:

Statement	*Branch*	*Expected*	*Result*
Toggle button checked	True	Notis = true	Valid
	False	Notis = false	Valid
On Activity Result	Code = Resolution	Client Connect	Valid
	Code = Opener	Open Drive	Valid
	Code = Deleter	Open Drive	Valid
On Resume (null Client)	True	Build new Client	Valid
	False	Connect	Valid
On Pause (null Client)	True	Disconnect	Valid
On Connect Flag = 0	True	Registration msg	Valid
	False	No registration msg	Valid
On Click View	openButton	RequestCodeOpener	Valid
	textNeighbour	Open Messages	Valid
	emergencyCall	emergencyCheck ()	Valid
	startCamera	Arm Cam (Blue light)	Valid
	stopCamera	Disarm Cam (light off)	Valid
Emergency Permissions	True	Call 999	FAILED
	False	Permission change msg	FAILED
Menu Selection	Sign out	Sign out + finish task + start logout activity	Valid
	Set IP	Start SSH activity	Valid
	Delete Files	RequestCodeDelete	Valid
	Set Neighbor	Start setNeighbour activity	Valid
	alarmSound	Start alarmSound activity	Valid
SignOut Client == null	True	Disconnect	Valid
	False	Prompt Log in	FAILED

Failures:

1. Emergency Permissions Failure (True): Permissions in phone turned off, change Call Intent to Dial Intent.
2. Emergency Permissions Failure (False): No message to handle case of lack of permissions, added message.
3. SignOut Client == null(False): No message to handle signing out before logged in, message added.

AlarmSound:

Statement	*Branch*	*Expected*	*Result*
Choice	0	Fire Alarm	Valid
	1	Dog Bark	Valid
	2	Lullaby	Valid
	3	Siren	Valid

MyFirebaseMessagingService:

Statement	*Branch*	*Expected*	*Result*
Notis	True	Get Notifications	Valid
	False	No Notifications	Valid

SetNeighbour:

Statement	*Branch*	*Expected*	*Result*
Edit Text length = 10	True	Set neighbor's number and save in shared preferences	FAILED
	False	Prompt Correct Length msg	Valid

Failures:

1. Edit Text length == 10 (True): Shared Preferences syntax was incorrect and as a result the message was not being saved when the application was closed down.

Compatibility/Portability tests

In order to test the forward compatibility of the application, devices of each android build from marshmallow 5.0 up to current were tested via the android studio emulator. This brought to light the fact that often on the earlier builds, the device was crashing due to CPU constraints. The Android studio Debugging tool was used to detect that the problem was that multidexing was disabled, and our activity was doing too much work in the main thread causing skipped frames. Thus, the team re-wrote much of the mainActivity code to ensure that multithreading was used to take load off of the CPU.

The Raspberry Pi

Stress - Test

To test the raspberry Pi – side code, the team opted for functional stress testing methods. This involved leaving the raspberry Pi powered on, with the script running for 48 hours in succession, to ensure that there would be no problems discovered.

Error 1: Of course, this identified certain problems, in particular with the alarm sounding functionality. After a while, the frames being captured would freeze while the alarm sound was being played, and would resume thereafter. Also upon exiting the program and restarting IO error messages were displayed and the program was un-runnable. After changing the software used to play the music, from OmxPlayer to VLCPlayer, this issue was greatly reduced, but not fully resolved, in that the team had managed to fix the IO error, and the freezing issue was reduced but not fixed. To resolve this, the team attempted to separate the function that handled playing the sound into a separate thread, but this unfortunately did not resolve the issue. The team consulted our supervisor about this issue, however he too was unsure as to what the source of the problem was.

However, this will not be an issue in real world use of the system, as although there was a delay in the live feed, frames that had detected movement were still collected and uploaded as normal, and as the security feed screen on the raspberry pi will never actually be turned on in a realistic scenario, the Pi would simply be powered on and its functionality accessed through SSH. This also did not impact on the Pi being able to run for long periods of time as proven by our ability to leave the script running for 48 consecutive hours.

The team feels that with more time and a more proficient knowledge of python, considering this was our first time using the language, that perhaps a more suitable solution could have found. Threading could perhaps still be the way forward as it may not have been fully utilized in our code. Another possibility is a python library called ‘subprocess’ which appeared to allow functionality to run in the background, however time constraints resulted in the unfortunate omission of the traversal of this potential cure.

Branch Testing

Each setting in the conf.json file option was enabled/disabled and re-run to ensure that each possible function was working correctly. The phone number for the SMS service was replaced with an incorrect phone number, i.e too many digits, the program thus failed.

The python code itself is not too complicated in terms of each function has very specific functionality which will cause the entire process to fail if one of them is not working. There are no real obsolete or optional functions, beside the ones related to the ‘conf.json’ file which was tested beforehand. We could therefore determine that the code was functioning correctly as files that had detected movement were uploaded as expected, in the manner we expected. Nevertheless, we ran the ‘inspect’ functionality on PyCharm which detected no anomalies in the code and ran the debugger just for clarity.

Testing for Various Web Cams

Multiple different webcams were tested on the raspberry pi to ensure that the code was fully functioning for a universal set of USB webcams. The team used 3 different webcams for this purpose, a ‘Microsoft Lifecam HD-300’, a ‘Trust Webcam’, and a ‘Mikomi 7130’. This test passed conclusively with each camera booting up upon the script running and capturing images as expected.

As with the webcams we also decided to test using 2 separate speakers to ensure no options were needed to be altered upon the use of a different speaker. Again, this test passed conclusively with both speakers outputting sound as expected.

Network Test

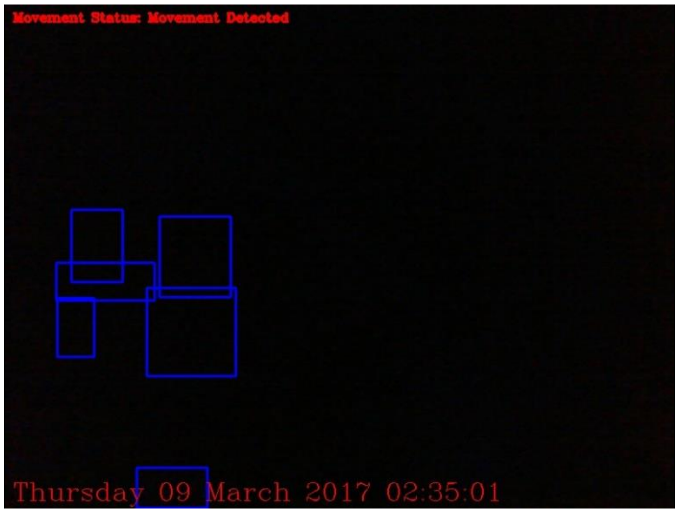
The Raspberry Pi was also tested by connecting it to various networks, be it home networks, or mobile hotspots from the team’s own phones, to ensure that the system would run not depending on a specific network IP.

Detecting movement

One of the main functionalities of the pi itself that was tested was of the image capture itself and how well it worked in various conditions. Firstly, it had to be ensured that images were only captured and uploaded when actual movement was detected. This was done through running the camera feed and testing detection both using slow movements and rapid movements. Both were detected well but the team noticed that the amount of motion needed was quite large. This led to the decision to reduce the “min_area” variable in ‘conf.json’ from 2000 to 1000 so that the contour size detected in the frame didn’t have to be as large in order for it to be classed as movement. This ensures that smaller motions will still be detected. However, the team felt that although this helped movement to be detected, the frequency of uploads was still too small. This was due to the “min_motion_frames” variable, which was initially set to 8, declaring that a minimum of 8 consecutive frames with detected movement were needed for an image to be uploaded. We initially reduced this to 4, but found pictures to be saved with too little movement, so finally decided on 6 frames being the ideal number through more testing.

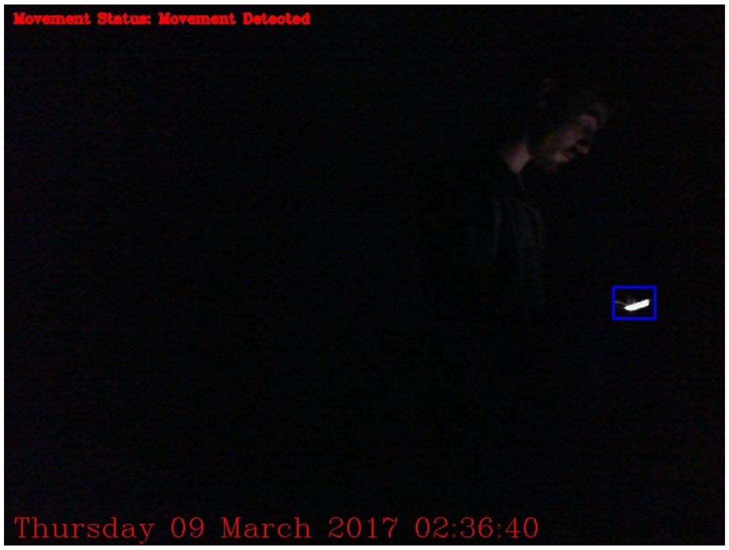
Completely dark room

When developing the code, the process had been run many times to the point whereby we knew that images were easily and well captured at close range, in a well-lit room, as most often the camera would be situated in front of us on a desk, or in a living room. The team deduced that while this was optimal conditions for initially ensuring the detection worked, we wanted to test that movement could be detected in a variety of common conditions. This was tested under the following conditions.


Movement Status: Movement Detected
Thursday 09 March 2017 02:35:01

As can be seen from the picture above, the team tested the camera in conditions of a completely dark room where nothing is visible. Through this testing we discovered that the camera did in fact detect movement, however not quite to the extent of when the room was fully lit, in that it had to be quite distinct movements in order to be detected. This surprised the team as it was honestly not expected to be able to distinguish the movement without visible light. Clearly, the image is not entirely useful from a security standpoint in that it's still black, but this can still be viewed as useful as the alarm functionality will still work, scaring off potential threats, or perhaps soothing an agitated child. Also, the notification will still be sent to the user.

Small light presence

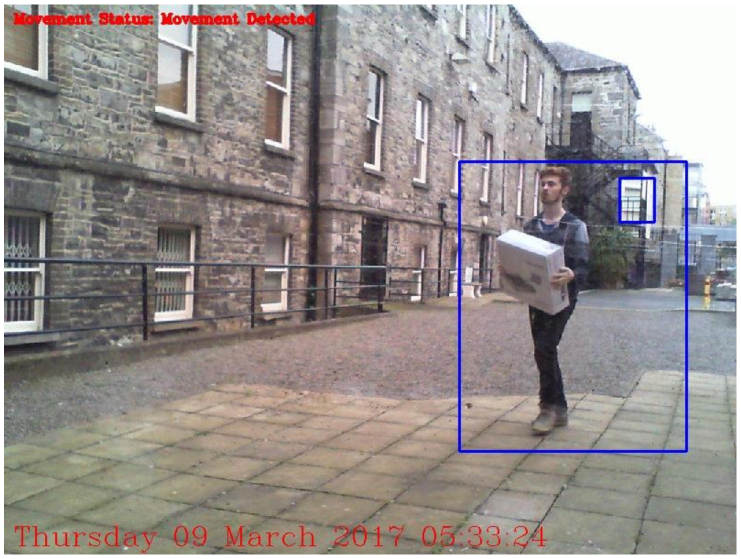

Movement Status: Movement Detected
Thursday 09 March 2017 02:36:40

As can be seen from the above image the team also decided to test motion detection by standing in a completely dark room with a small source of light (shown in the above picture). Again, this test was more successful than expected and as you can see, the movement of the phone light being switched on immediately caused the camera to detect motion around this shift in the image. This test passed with surprisingly responsive results, and illustrates the usefulness of the system's ability to perform even in low light situations.

Drastic change in lighting conditions

The team also tested lighting conditions by testing if a large shift in the light would be detected as movement. We started off in a completely dark room and then switched the room's lights on full immediately. Considering our motion capture is developed using the weight of the frame's background as a reference there is no surprise that this sudden burst of light caused a motion event to be triggered. This could be a potential downfall, considering no actual movement was detected, only light. However, it could also be potentially useful in certain security scenarios.

Detecting images at a distance


Movement Status: Movement Detected
Thursday 09 March 2017 05:33:24

As seen in the above image and in the video demonstration scenario of a home delivery, the team tested the camera for longer distance image capturing. This would be mainly used if a user wanted to set up their camera outside, or as tested, from an inside room looking outside through a window. Again, the testing went very well in that the image is still clear, the box detecting movement was accurate, and the image was captured and processed as expected. The distance and outdoor lighting conditions were something the team saw as absolutely essential when determining criteria in which the camera should work, and as seen above the test did pass.

Overall the testing of the Raspberry Pi went about without any major drawbacks other than the sound issue described above, any issues that were present were quickly ratified upon discovery and were generally small issues with the written code rather than the software.

User Testing

As part of the testing for SecuriPi the team decided that user testing would be a good way of demonstrating the ease at which our system could be set-up and used. With the given time constraints, since the system has been fully built, it has been tested on one external user. The user, who opted to remain nameless, is a 20-year-old, female, computer science & infrastructure student currently in 3rd year in DIT.

The user was given a list of tasks to do with regards to basic set-up and use of both the raspberry pi and the android app. When going into this test scenario the user had prior knowledge of what the project was but had no experience in viewing the system's interface or using any of the features. The user had plenty of experience with phone applications but had never used a raspberry pi before.

The raspberry pi and android app were given to the user with most of the necessary files and software installed, this was to save time as installing things like the application on android studio and OpenCV on the pi would take far too much time for this test. The user's tasks came from the user manual we supplied as part of our documentation, both in terms of the pi and the android application itself.

Tasks

1) The user's first task was to open the 'conf.json' file on the Raspberry Pi and to enable settings as they saw fit. Our user chose to turn the live camera feed off, SMS notifications on, automatic authentication on and sound functionality on. They left the camera settings as default. For the purpose of this test the user would be using SMS credentials linked to the team's TextLocal account.

Result:

There were no issues in performing this task, with it being relatively straightforward especially for a proficient computer user.

2) The user's next task was to create their own Google Developer project to link with the Pi, this consisted of following instructions in the Technical Specification, creating the project enabling credentials and API for Drive, and downloading a json file into the working directory.

Result:

The user completed this task easily and efficiently following the outlined steps both on the website and in our specification. They struggled to find the button to download the json file as it is quite small and not obvious upon first glance, in fact the team even struggled with this same task upon set-up. Perhaps more detailed directions to the button's location need to be provided in the Technical Specification document.

3) The next task the user had was to run the script, this was done locally on the pi as initial authentication was needed. They were then instructed to check the 'MyCreds.txt' file to ensure

it had been written to, and then re-run the script to ensure that no further manual authentication was needed.

Result:

Again, this was a relatively straightforward task for a user familiar with command line arguments. It was completed easily and quickly.

4) Next, the user was tasked with entering their raspberry pi's IP address into the "Connect via SSH" setting in the app, so that they may interact with the pi directly from the app.

Result:

The user initially struggled getting the IP address as they entered the Windows command "ipconfig" rather than the Linux equivalent "ifconfig" on the pi. But they soon realized their mistake and found the correct address and entered it successfully in the app. (*the user was asked how they found the button as first time using the app and noted that because of the simplistic design there was only one clear place to look*).

5) The user was next asked to save a number for their neighbor contact and send a text through our quick text function. The user was told to enter their own phone number for validation sake.

Result:

The user quickly found the setting having navigated to the same setting bar in the previous task and set up the number with ease. They then navigated back to the main app screen and sent the pre-saved text to their number.

6) The user was next tasked with changing the alarm sound that would be outputted from the pi from the app. They could pick whichever sound they wanted, except 'Siren' as that was the option already set and the goal was to demonstrate the change.

Result:

The user again found the option quickly as it was in the same set of settings accessed by the button in the top right corner. The user noted that the options were clear to choose from and that the on-screen validation that a sound had been selected was good. We then showed the user the code on the pi whereby the sound file had been changed to "Dog" as she had selected.

7) The user was then instructed to start the camera via the application, Ross (stood in an adjacent room) would trigger movement on the camera, and then the user was asked to view the images via the web browser link in the app.

Result:

The user quickly initialized the camera and opened her drive account via the labelled buttons. In fact, she completed these tasks too quickly as we were left waiting for Ross to detect movement for the files in drive to refresh after she already had it open. Once refreshed the images of Ross in the next room appeared on the user's drive account. The user had successfully completed set-up and basic use of the app.

8) As the camera was still running we instructed the user to go back to the app, stop the camera and then view these same files in the app version of drive.

Result:

The user again completed this quickly as she did not have to go anywhere new to access any of the functionality and at this stage was getting comfortable in using our app. The camera was successfully stopped and the same images viewed on the browser were available to view from the app itself also.

9) Lastly the user was tasked with deleting the files that were just uploaded from the app itself.

Result:

The user was initially confused as to why the delete activity seemed so similar to the opening of files in app, and was unsure if the file had been deleted once the select option was hit (unfortunately select cannot be changed to delete as it is a part of the Drive API). However once the user opened drive to check whether the delete had been a success, it was clear how the function worked.

Observations and Feedback:

The user was very pleased with the system as a whole, noting that, the app in particular, was "very easy to navigate. All main functionality is located directly on the main page and the simple design of the app meant that I was never lost when issued a task". She did however state that the pi was "slightly more difficult to navigate through" and that some "less tech-y users" may struggle to perform some of the tasks necessary. Although on further discussion she agreed that "there isn't really much you can do about that when considering installations ", and explained how the set-up documentation did have "very clear steps" for each necessary process on the pi.

The user stated that while it was aesthetically "basic", the design of the app actually helped her to more quickly perform tasks as everything was clearly laid out and that the app was "easy to get to grips with". She noted how the on-screen feedback was a very good addition, especially when changing settings, that there was "very clear feedback if I had changed something or confirmed a setting".

Overall the user stated that they thought the premise of the project was a good idea and that the "quick and simple" nature of the app was necessary especially when considering a main function of the app is to provide quick responses to security alerts.

We were very happy with this feedback as we had put a lot of thought into what functionality needed to be quickly accessed from the main screen of the app, and which could be put behind the settings button in the top-right corner. We were also very happy that considering the user had no experience using a raspberry pi before that, what few tasks we did ask her to perform, were done with relative ease because of the available documentation. We were also happy that the user had could complete all the tasks and that the actual functions of the app all worked as intended.

End of Document

Thank you for Reading.

-SecuriPi Development Team