Module 2 Final Report

1. Table of Contents:

2. Introduction

This project is named the Desktop Launcher. The objective of this project is to create an interesting and appealing toy that is small enough to fit on a desktop. This toy is aimed towards target market of young adults and technology enthusiasts of any age. The basis for this product is a physical launcher with two motors allowing it to rotate 360° on the horizontal axis, and approximately 60° in the vertical axis. The turret is also equipped with a third motor that acts as a firing mechanism, as well as a camera that allows it to capture an image in the direction it is facing.

There are two main methods of controlling the turret. First, as developed in module 1, the user can operate it using an LCD touchscreen display. Second, the user can operate the turret through the use of an android app that controls the turret over bluetooth. This second method is the point of focus of the module 2 project and report.

3. Project Management

The initial conceptual design of the module 2 project stemmed as a direct extension of the module 1 project. This plan was known and agreed upon in advance to all members during module 1 planning. Therefore, there was no need to plan the conceptual design of module 2 and did not require any team discussion.

Moving on from conceptual design,it was decided to split into two teams of two in order to discuss and develop two separate detailed designs for module 2. This was to hopefully produce two distinct designs, with either subtle or major differences. These differences could then be evaluated and weighted by all team members to produce one design with the most optimal features. This method of separation is effective, since it allows a more diverse pool of ideas that can be evaluated, and less chance to get pigeonholed. Both designs were created by meeting outside of class time to brainstorm and record ideas, and discuss them while meeting during class time. This was effective as to optimize lab time.

Once a detailed design was decided upon, project goals and a list of desired features were developed from it. A Trello board was used to maintain a list of high level tasks to be completed. During initial class periods, tasks were ranked in order of importance and marked as such on the trello board. Those tasks were the first to be divided to members in order to get a working foundation of the project completed as soon as possible. This could then be added upon with additional features.

One management method that was particularly effective was using a checklist during meetings. This checklist was unlike the Trello board which listed all the high level tasks that the project encompassed. Instead, at the beginning of meetings (both in the labs and outside) as an extension to scrum, a discussion was held to brainstorm and write down on paper a checklist of all the low level/ specific tasks to be completed that day. This method was particularly effective for three reasons. First, it gives a specific direction and goal for the day, as opposed to a general idea of what should be done. Second, it gives tangible and achievable goals. Breaking up requirements into small individual tasks gives a sense of progress and accomplishment that drive motivation. Lastly, it very useful in keeping track of bugs that arose in the project. When a bug was found during a meeting, or during member's individual work it would be added to the checklist and either be dealt with or kept track of to be dealt with in the future.

The last project management method the team used was to schedule meetings outside of class time as needed in order to keep the project ahead of schedule.

4. Work Accomplished

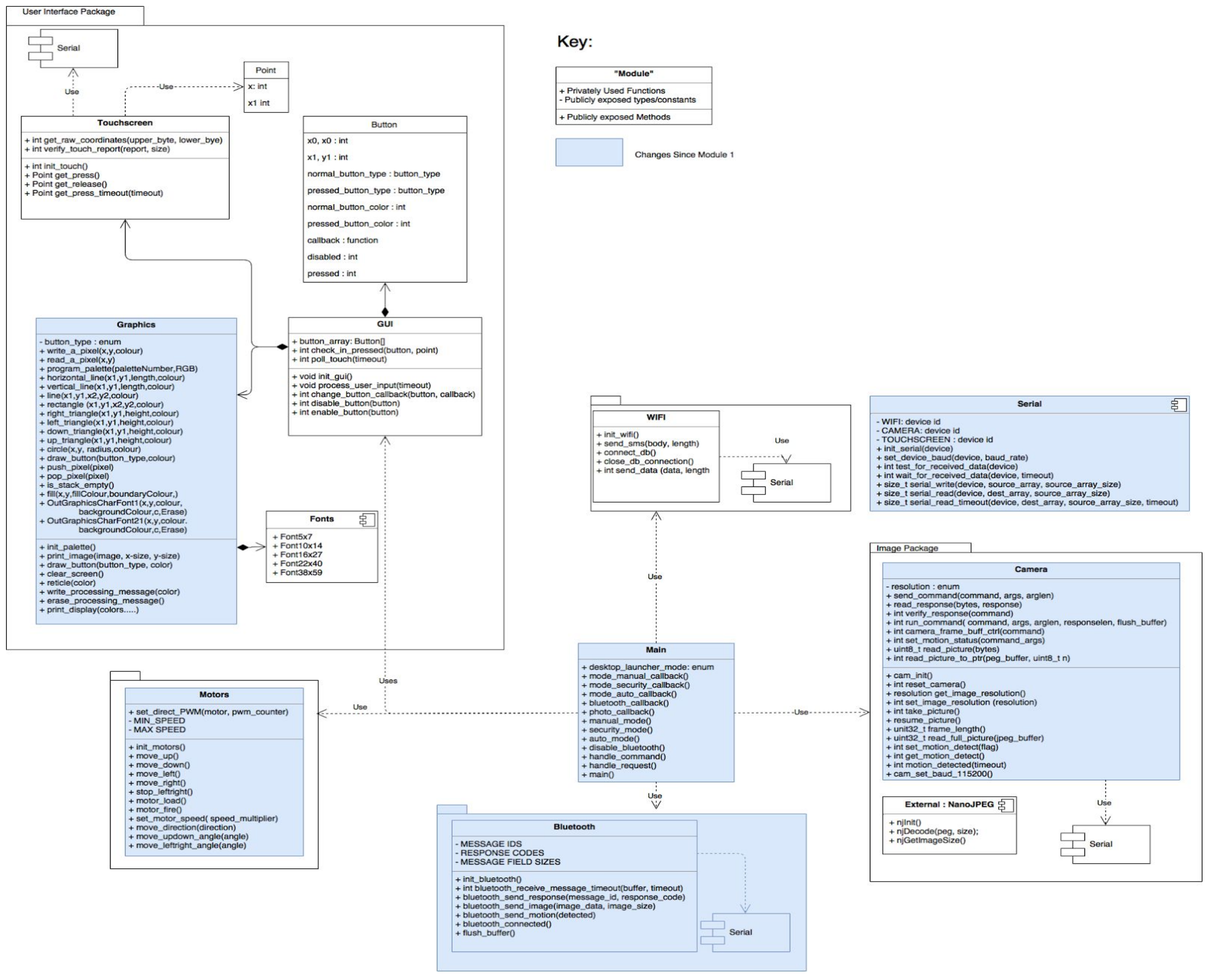
The team successfully implemented all the features available on the NIOS II from module 1 on the Android Application, this includes manual mode, auto mode and security mode. In addition, there is improved NIOS II's performance (discussed below in Detailed Design - NIOS II Improvements). Lastly, these extra features are implemented on the Android application:

- Using Android device's accelerometer detect phone tilt and move turret accordingly
- Point-and-click to move turret
- A photo gallery to view previously taken photos
- A Settings menu to set whether to save photos to device or not and control turret speed
- Tracking mode to track a coloured blob selected by the user

5. Detailed Design:

As mentioned earlier the Module 2 design did not add any new hardware except for the Bluetooth chip. Instead, it focuses on refining the operation of the hardware modules currently implemented. Additionally, Module 2 includes a massive extension to the software capabilities of the NIOS II system in conjunction with the introduction of an Android companion application, the DTL Application.

a) Nios II Improvements:



User Interface Package
Serial
Use
Point
x: int
x1 int
Touchscreen
+ int get_raw_coordinates(upper_byte, lower_bye)
+ int verify_touch_report(report, size)
+ int init_touch()
+ Point get_press()
+ Point get_release()
+ Point get_press_timeout(timeout)
Button
x0, x0 : int
x1, y1 : int
normal_button_type : button_type
pressed_button_type : button_type
normal_button_color : int
pressed_button_color : int
callback : function
disabled : int
pressed : int
Graphics
- button_type : enum
+ write_a_pixel(x,y,colour)
+ read_a_pixel(x,y)
+ program_palette(paletteNumber,RGB)
+ horizontal_line(x1,y1,length,colour)
+ vertical_line(x1,y1,length,colour)
+ line(x1,y1,x2,y2,colour)
+ rectangle (x1,y1,x2,y2,colour)
+ right_triangle(x1,y1,height,colour)
+ left_triangle(x1,y1,height,colour)
+ down_triangle(x1,y1,height,colour)
+ up_triangle(x1,y1,height,colour)
+ circle(x,y, radius,colour)
+ draw_button(button_type,colour)
+ push_pixel(pixel)
+ pop_pixel(pixel)
+ is_stack_empty()
+ fill(x,y,fillColour,boundaryColour,)
+ OutGraphicsCharFont1(x,y,colour, backgroundColour,c,Erase)
+ OutGraphicsCharFont21(x,y,colour. backgroundColour,c,Erase)
+ init_palette()
+ print_image(image, x-size, y-size)
+ draw_button(button_type, color)
+ clear_screen()
+ reticle(color)
+ write_processing_message(color)
+ erase_processing_message()
+ print_display(colors.....)
GUI
+ button_array: Button[]
+ int check_in_pressed(button, point)
+ int poll_touch(timeout)
+ void init_gui()
+ void process_user_input(timeout)
+ int change_button_callback(button, callback)
+ int disable_button(button)
+ int enable_button(button)
Fonts
+ Font5x7
+ Font10x14
+ Font16x27
+ Font22x40
+ Font38x59
Key:
"Module"
+ Privately Used Functions
- Publicly exposed types/constants
+ Publicly exposed Methods
Changes Since Module 1
WIFI
+ init_wifi()
+ send_sms(body, length)
+ connect_db()
+ close_db_connection()
+ int send_data (data, length
Use
Serial
Serial
- WIFI: device id
- CAMERA: device id
- TOUCHSCREEN : device id
+ init_serial(device)
+ set_device_baud(device, baud_rate)
+ int test_for_received_data(device)
+ int wait_for_received_data(device, timeout)
+ size_t serial_write(device, source_array, source_array_size)
+ size_t serial_read(device, dest_array, source_array_size)
+ size_t serial_read_timeout(device, dest_array, source_array_size, timeout)
Image Package
Camera
- resolution : enum
+ send_command(command, args, arglen)
+ read_response(bytes, response)
+ int verify_response(command)
+ int run_command(command, args, arglen, responselen, flush_buffer)
+ int camera_frame_buff_ctrl(command)
+ int set_motion_status(command_args)
+ uint8_t read_picture(bytes)
+ int read_picture_to_ptr(peg_buffer, uint8_t n)
+ cam_init()
+ int reset_camera()
+ resolution get_image_resolution()
+ int set_image_resolution (resolution)
+ int take_picture()
+ resume_picture()
+ unit32_t frame_length()
+ uint32_t read_full_picture(jpeg_buffer)
+ int set_motion_detect(flag)
+ int get_motion_detect()
+ int motion_detected(timeout)
+ cam_set_baud_115200()
External : NanoJPEG
+ njInit()
+ njDecode(peg, size);
+ njGetImageSize()
Use
Serial
Main
+ desktop_launcher_mode: enum
+ mode_manual_callback()
+ mode_security_callback()
+ mode_auto_callback()
+ bluetooth_callback()
+ photo_callback()
+ manual_mode()
+ security_mode()
+ auto_mode()
+ disable_bluetooth()
+ handle_command()
+ handle_request()
+ main()
Uses
Use
Motors
+ set_direct_PWM(motor, pwm_counter)
- MIN_SPEED
- MAX SPEED
+ init_motors()
+ move_up()
+ move_down()
+ move_left()
+ move_right()
+ stop_leftright()
+ motor_load()
+ motor_fire()
+ set_motor_speed(speed_multiplier)
+ move_direction(direction)
+ move_updown_angle(angle)
+ move_leftright_angle(angle)
Bluetooth
- MESSAGE IDS
- RESPONSE CODES
- MESSAGE FIELD SIZES
+ init_bluetooth()
+ int bluetooth_receive_message_timeout(buffer, timeout)
+ bluetooth_send_response(message_id, response_code)
+ bluetooth_send_image(image_data, image_size)
+ bluetooth_send_motion(detected)
+ bluetooth_connected()
+ flush_buffer()
Serial

Figure 1: NIOS II Software Class Diagram

The NIOS II software includes several improvements that centers around the goals of speeding up existing functionality, adding a richer API for the Android client to utilize, and making the device more robust.

i. Image Transfer

One of the areas that required improvement was the image fetch times from the camera to the NIOS II system. Previously, the fetch time was averaging at 8.9 seconds across all tests . Additionally, preliminary testing showed Bluetooth addition increase the time for the image to arrive on the Android client to be 14.1 seconds. Essentially, this rate is not acceptable as it does not meet the project's requirement of being responsive.

To combat the image fetch time bottleneck, the baud rate of the camera was increased from 38400 to 115200, this provided a 3x speed boost in image transfer from the camera to the Nios II.

Moreover, the NIOS II was fetching the image from the camera in 64 byte chunks. This means that for a typical 640x480 image (60KB), 960 messages would have to be sent to the camera to request the data (each message is about 5 bytes). Additionally, the response to these messages has a 7 byte overhead due to data framing. Thus, for each 60 KB of image data being received. There was 11.520 KB worth of overhead (19.2%). To help reduce the overhead, the chunk size was increased from 64 bytes to 128 bytes, reducing the this overhead to 5.760 KB (9.6%).

The final results shows that the time for the Android client to fetch and display the image is reduced to 3.98 seconds, a 72% reduction in time.

ii. Motor Control

The motor control library has been tuned to allow for more speed settings. Moreover, the existing speed settings are now linearly spaced. The motors are also now calibrated allow exact angle rotations in both axis.

These features are instrumental in the operation of the colour tracking and image touch control on the Android client.

iii. Bluetooth Library

A Bluetooth library has been added to enable communication with the Android Client. This library allows the NIOS II system to conform to the Communication Model described in the Communication Model section. It serves two purposes:

1. A Message Framer, abstracting away the process of encoding the outgoing messages behind convenience functions
2. Serves as a wrapper around the serial layer allowing the rest of the NIOS II system to use it to communicate with the Bluetooth chip

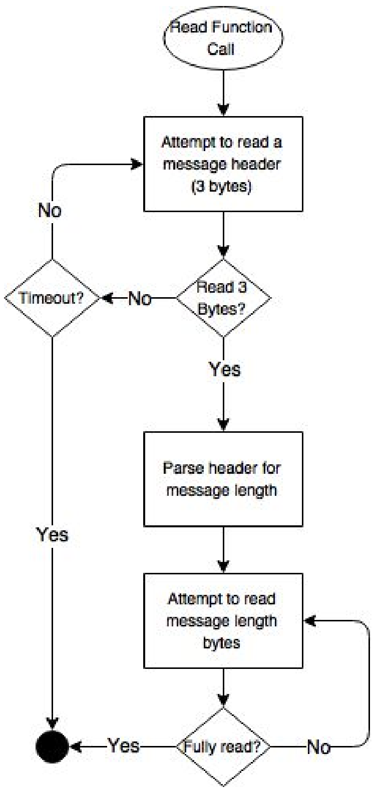
The Bluetooth library follows a very simple and generic receive algorithm to get the messages from the Android client. This is due to the flexible and generic nature of the communication protocol. This facilitates the process of receiving any type of message regardless of length.

The receive timeout call in the library simply loops through attempting a timeout read on a message header. The header then contains the length of the rest of the message, the library then attempts to read length bytes from the Bluetooth chip.

The messages themselves vary in purpose and length, please refer to the submitted code and the Communication Model Section for further details on message structure.

In addition to sending messages , the Bluetooth library allows the NIOS II to:

- Query the Bluetooth device for connection status
- Change the name of the Bluetooth device
- Change the password of the Bluetooth device



Read Function Call
Attempt to read a message header (3 bytes)
No
Timeout?
No
Read 3 Bytes?
Yes
Parse header for message length
Yes
Attempt to read message length bytes
Yes
Fully read?
No

Figure 2: Bluetooth Read Loop

Collectively these features make it possible to design a very responsive user system that is able to handle connections/disconnections on the fly, such that the user is never “locked into” using either Bluetooth or NIOS II touchscreen control.

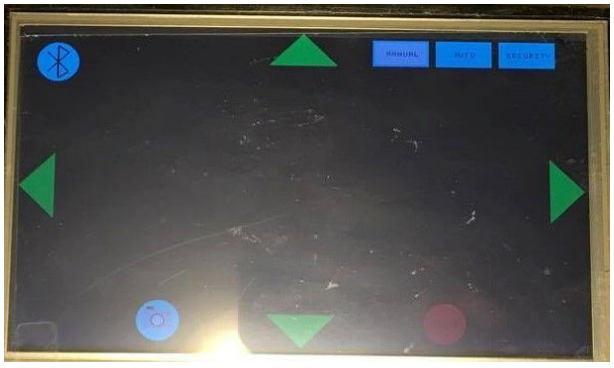


Figure 3: Modified NIOS II Main Screen

iv. Main Loop

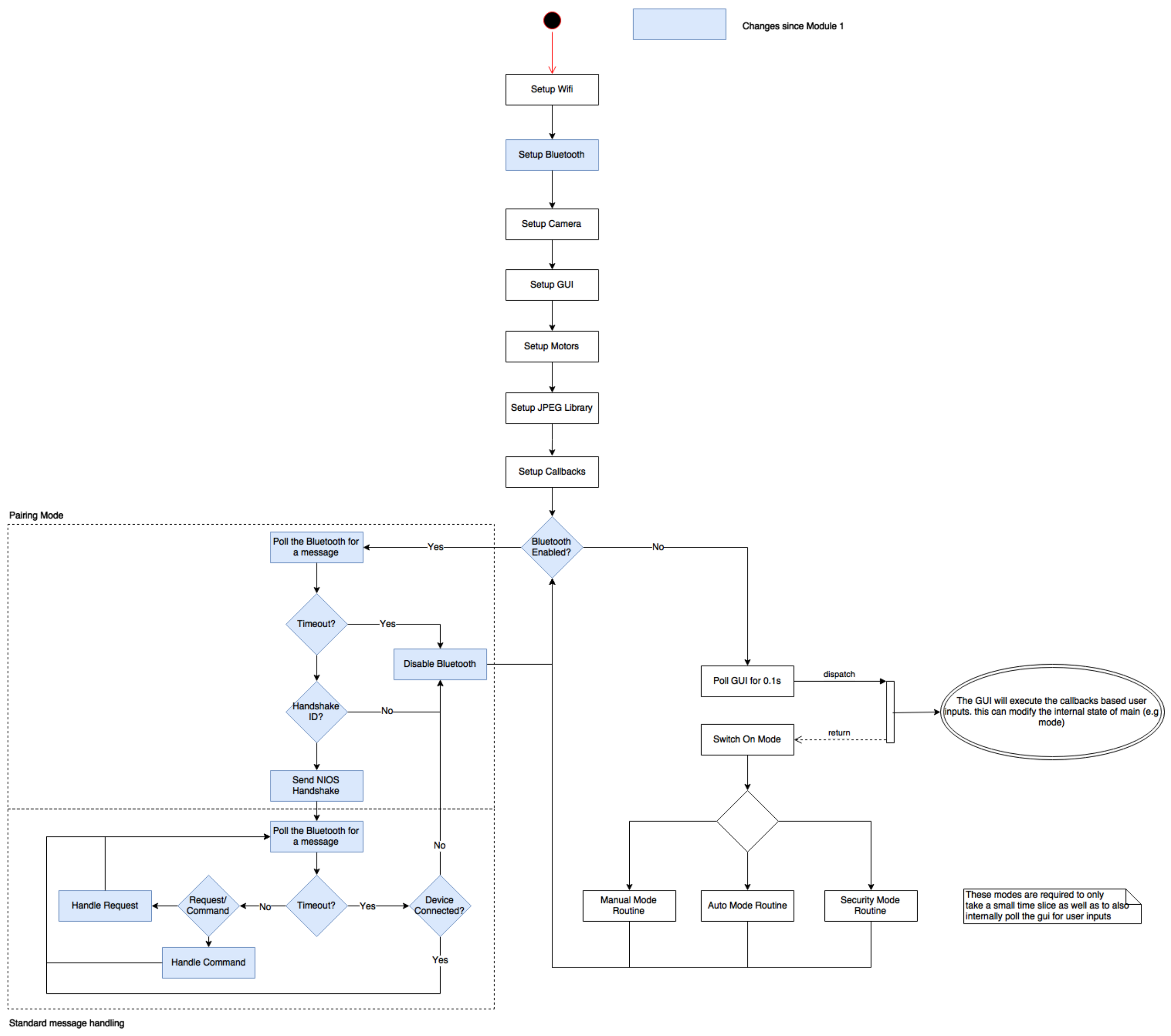


Figure 4: NIOS II Main Loop

As can be seen from the diagram above, the addition of the Bluetooth component to the main loop was was relatively straightforward as the system was originally designed with expandability in mind.

One of the most interesting features of the this design is Pairing Mode. When the Bluetooth button on the NIOS II system is pressed the interface is disabled and the Bluetooth path is now active. If a handshake does not arrive in the first 12 seconds of the connection, either:

- The device connected is not a device running the DTR application
- The device never connected/has disconnected

This allows the main loop to “kick” the NIOS II back to user mode if no connection is active.

Similarly, once pairing is successful and the device is in standard message handling, every time a poll is not successful the NIOS II system checks to see if it should exit out of Bluetooth mode.

b) Communication Model

Communication between the NIOS II and the Android device mimics a server-client model. In this model the NIOS II acts as the server and the Android device as the client. Additionally, communication between the two device was done in a purely synchronous way. This is done because the NIOS II only has a 1 Byte receive buffer, thus if the NIOS II is not listening to the data when a message is sent by the client then there is a high chance that the data will be missed.

The following restrictions are placed on the system to prevent data loss:

1. Only one side must send data at any given time
2. Unless acting on a message's instruction, each side must constantly be listening for new messages
3. Each side must receive data from the other before it can send again
4. Only the Android client can initiate communications with NIOS II

Additionally, as can be seen from figure 5. In order for NIOS II to signal an intent to connect with the NIOS II it must send a heartbeat message. The NIOS will then reply to this heartbeat, indicating that it is now listening to the Android client. This process only needs to happen once to synchronize all devices. It is also used to help avoid cases where a phone without the DTL application connects to the NIOS II.

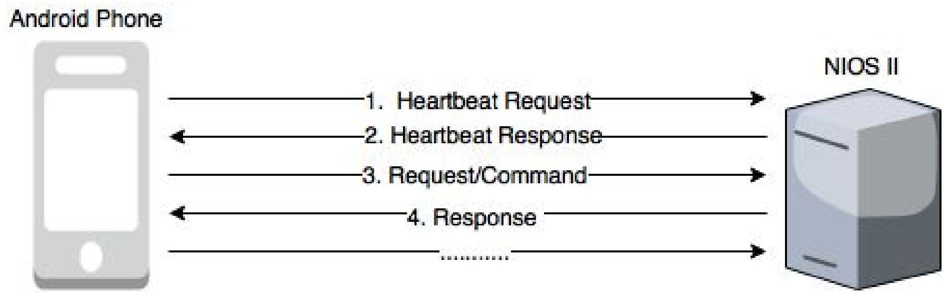


Figure 5: Communication Protocol

c) Messaging Structure

Each message sent between the two devices has the following general structure

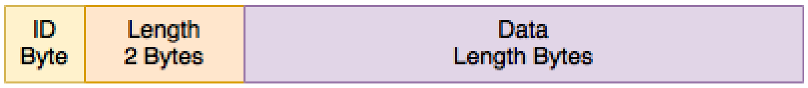


Figure 6: General DTR Message Format

A single Byte for ID was sufficient since the project does not use more than 255 unique message types. Additionally, the maximum length of the data is 2 Bytes which allows a maximum payload of 65536 Bytes. This limit was chosen since the biggest image that would be transmitted had a size of 62 KB. The data is interpreted differently based on the ID of the message.

Note: Note due to Bluetooth communication reliable nature, and guarantees of data ordering and integrity a CRC/Position Dependant checksum is not part of the message structure.

The DTL message allows for 2 types that the Android client can send to the NIOS II system

i. Commands

A command message is classified as any message that causes the NIOS II to perform an action and return a simple response code. Some examples of commands are:

- Move Angle
- Change Motor Speed
- Fire

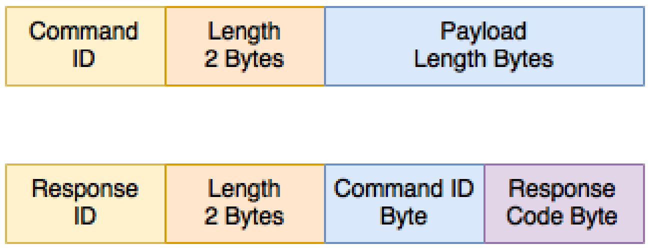


Figure 7: Command & Response Messages

The structure of the command message (Sent from the client) and the response (sent from the NIOS II) can be seen in figure 7.

Each command sent to the NIOS II will can be accompanied by a series of “Arguments” in the payload section. E.g. The move angle command will have 2 int8_t angles (x and y) in the payload section.

Each command will trigger a response message from the NIOS II. This response will be sent after the command has been executed.

All response messages have the same ID. However, the payload contains a Command ID Byte which identifies the command being responded to. Finally, the Response Code is used to relay the status of the command to the client.

For a full list of command please view the submitted code section.

ii. Request Messages

A request message is classified as any message that triggers an exchange of data between NIOS II and the Android client.

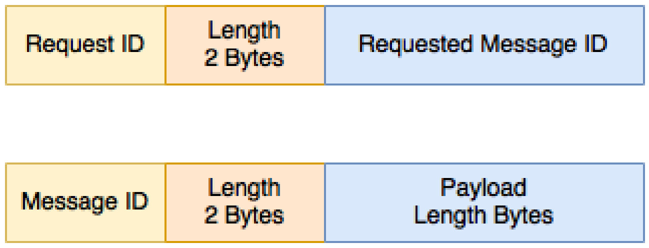


Figure 8 : Request Message Format

All request messages share the same ID. Their payloads contains a single Byte to denote the message being requested. The response to the request is the message being requested. The most prominent example of this is Image Data message, which contains the binary JPEG image.

For a full list of request messages please view the submitted code section.

d) Android Architecture

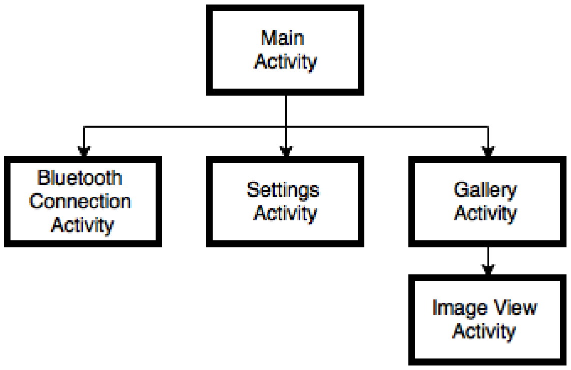


Figure 9: Activity Hierarchy

i. User Interface and User Experience

One of the most critical parts of the DTL application is the user interface and experience. As a result, a tremendous amount of time was spent designing the user interface's look and the expected behaviour of the various elements.

The requirements for the UI boils down to being simple, intuitive, smooth and responsive.

Simple and Intuitive User Interface
The intent of all the buttons are clear because of their corresponding icon and the available modes are labelled and aligned at the top. In addition, each activity is designed for a specific task so the user does not need to guess what the intent of each activity is. For instance, in the Bluetooth Connection Activity, all the user can do is connect and disconnect to their DTL. Similarly, in the Gallery Activity, the user can only view and manage pictures. Lastly, the application clearly indicates its state clearly at all times. For example, upon opening the app, the Main Activity will contain a “Turret Not Connected” label that indicates the user should connect to their DTL first.

Smooth and Responsive User Interface
Designing a responsive user interface requires providing feedback on user interaction. The first step is to provide feedback on every button click. For example, when a button is clicked, the alpha of all buttons lowers from 1 to 0.3 to indicate they are disabled. Furthermore, for tasks that take more than a second to execute, the application shows that the device is currently doing the scheduled task. For instance, when the user clicks the take picture button, a loading circle will be displayed on the center of picture frame to indicate the DTL is currently taking a photo. Similarly, in the Bluetooth Connection Activity, when the user attempts to connect to their DTL, a loading circle will be displayed to indicate the application has registered their request but it takes time to establish a connection. Lastly, the application is resilient against users spamming button clicks or activity switches. The implementation will be discussed in the Main Activity section.

ii. Activities

The majority of the Android application features lie in the Main Activity, but some other features are implemented in other activities. For example, the user has to connect or disconnect their DTL device at the Bluetooth Connection Activity. Similarly, settings change happen in the Settings Activity, this includes automatically backup photos (on/off) and turret speed (slider). Lastly, the Gallery Activity is a well polished photo gallery where the user can see and manage all their previously taken photos.

1. Main Activity

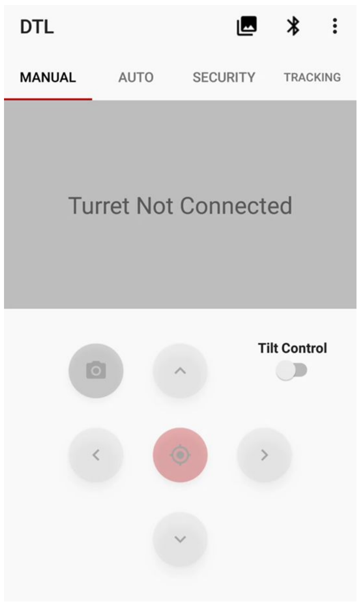


Figure 10. Turret Not Connected

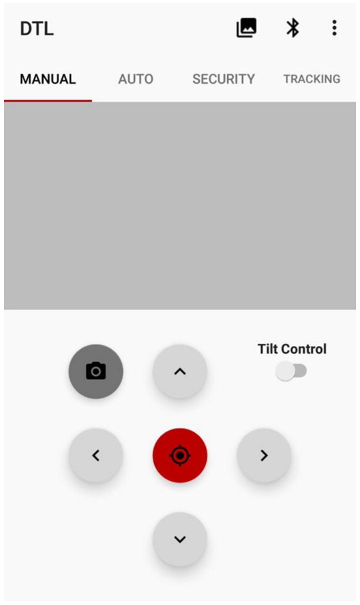


Figure 11. Buttons Enabled

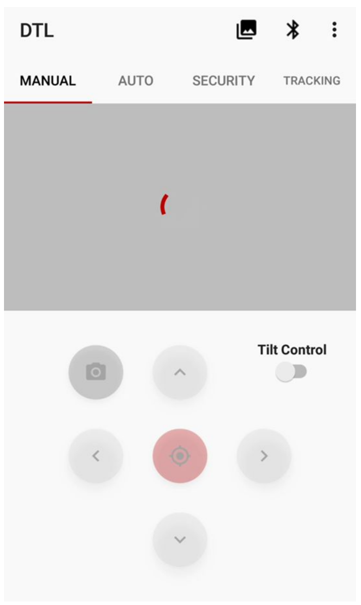


Figure 12. Buttons Disabled (processing)

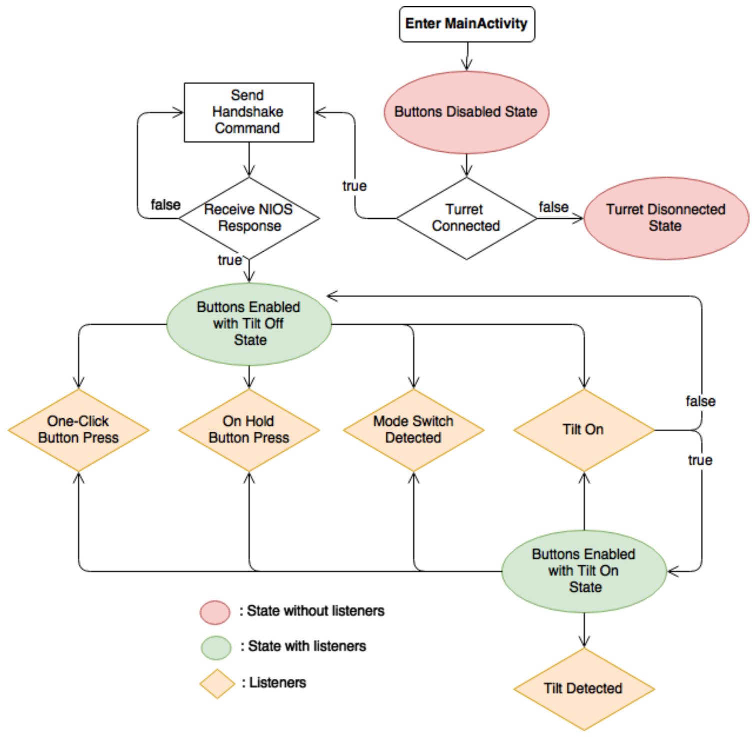


Figure 13: Main Activity

The Main Activity takes the approach of using listeners to respond to user events depending on its current state. With this design, the application becomes less complex and easier to reason about. After setting up the listener framework, adding or removing a listener becomes simple. Currently, Main Activity contains a button hold listener, button click listener, phone tilt detector and mode switch detector.

Upon entering the Main Activity, the application will check if there exists a connection with a DTL device. If not, the Main Activity will enter the “Turret Disconnected” state with all listeners deactivated. Otherwise, the device will initiate a handshake command to the NIOS every second until it receives a response.

After receiving a NIOS handshake response, the Android device will be in a “Buttons Enabled” state where the user can

1. press different buttons such as take a picture, fire a projectile, and move the turret
2. switch the DTL into a different mode (auto, security, or tracking)
3. press anywhere on a picture after taking a picture, to re-center the turret on the space clicked
4. tilt the phone up, down, left, right to move the turret if “Tilt On” is toggled on

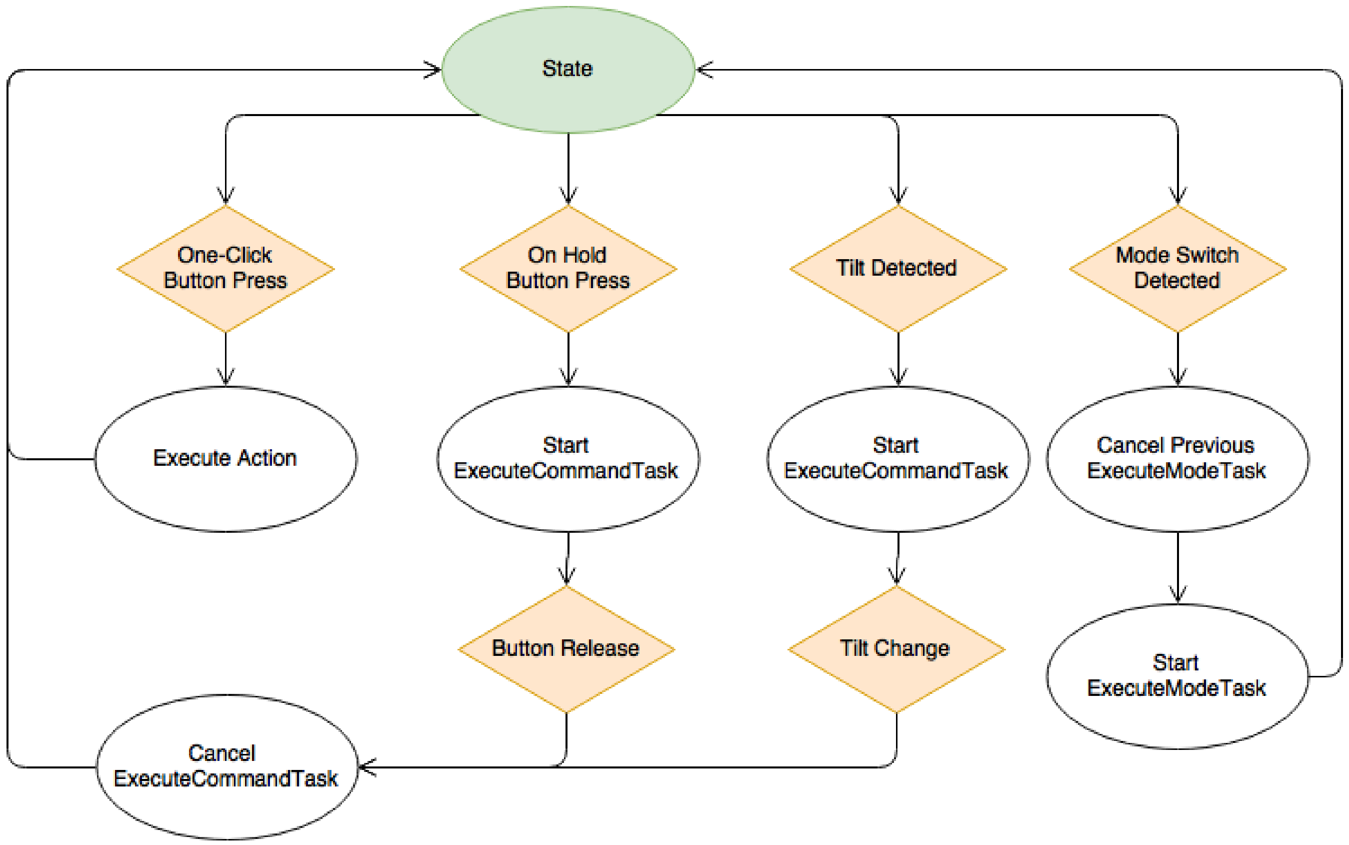


Figure 14. Listeners Flow Diagram

Listeners

One-Click Buttons:

- Photo button to take a picture
- Fire button to fire a projectile
- Click on an area inside a picture to re-center the turret's camera to the area clicked

On Hold Buttons:

- Up, down, left, right buttons to move the turret

Tilt Detector:

- Detects forward, backward, left, right phone tilts to move the turret

Mode Switch Detector:

- Manual, auto, security, tracking mode switches

AsyncTasks

ExecuteCommandTask - an AsyncTask that keeps on sending a movement command (up, down, left, right) to NIOS until the task is cancelled.

ExecuteModeTask - an AsyncTask that keeps on running routines until the task is cancelled. The routines differ for each mode.

Due to the team's communication model, after the Android device initiates a command, it must listen for a NIOS response. This is achieved through a CommunicationHandler that extends Handler in the Main Activity since. The CommunicationHandler listens for any NIOS responses and acts accordingly. Using this approach, all listeners are disabled whenever the Android device sends a command, the the task of re-enabling the listeners can be dispatched to the CommunicationHandler.

Following the communication protocol of only sending one message to NIOS at a time, until it get a response, on-hold buttons and tilt movement are implemented using AsyncTasks that runs forever until they get cancelled. Once a movement button has been pressed, the UI thread will create and execute an ExecuteCommandTask given a direction in an infinite loop. Note that this does NOT freeze the GUI because it is executing on a separate thread, but all buttons are disabled. Upon releasing the button, the AsyncTask will be cancelled and the buttons the CommunicationHandler will enable all buttons after the last command executes. Furthermore, the same idea applies for a tilt movement: start an ExecuteCommandTask when a tilt has been detected and cancel the task when the phone is back at resting position.

A challenge was implementing different modes since the NIOS is a slave in this communication model. Taking Auto Mode for example, the NIOS device cannot simply go to Auto Mode from module 1. Because if the NIOS is executing its Auto Mode routine, then it will not be able to listen for any other Android command. To overcome this challenge, the Android application contains routines that each mode runs. This is done by implementing the routines in ExecuteModeTask, one for each mode. Upon mode switch, the previous ExecuteModeTask cancels and a new ExecuteModeTask starts. Again, this design is robust as it allows simple addition or deletion of modes by implementing routines.

For Manual Mode, no work is needed to be done so there is no routine.
For Auto Mode, the routine is move right followed by a request picture command.
For Security Mode, the Android device requests for motion detection, if a motion was detected, it fires the turrets deadly projectile and takes a picture.

For Tracking Mode the application utilizes openCV. Upon entering tracking mode, the user can press on any object on the picture view. Doing so will sample an 8x8 pixel square. Since the tracker tracks colour the 8x8 square RGB values are converted to HSV. The average of the colours is the calculated and displayed in the "Colour" area in the GUI, the ColorBlobDetector is also configured to detect the colour. This process can be repeated until the "Start Tracking" button is pressed.

Once tracking has started, the DTL application will start taking photos automatically, each time a photo is received it will be passed through the ColorBlobDetector, which will return a list of detected blobs. The activity then filters these blobs to and Identifies the biggest blob and its centroid x,y coordinates. Based on the resolution of the the original image and the known FOV of the camera, the correct angle of rotation is then calculated and a rotation command is sent to the NIOS II system.

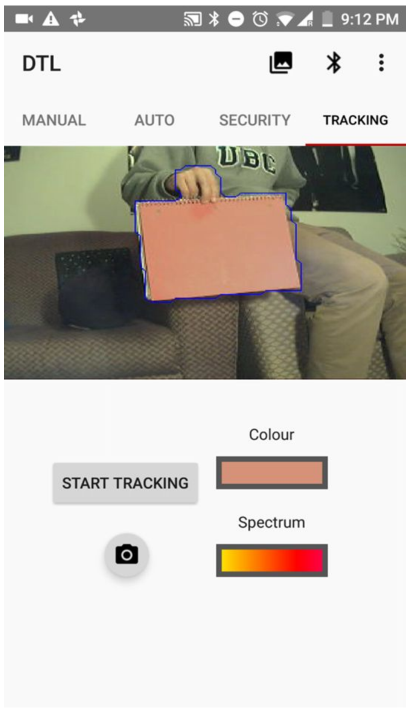


Figure 15: Tracking Mode

2. Bluetooth Connection Activity

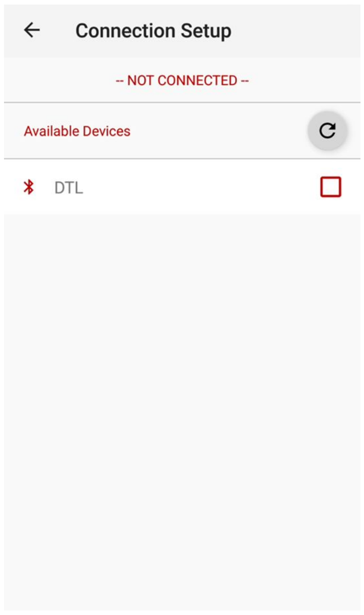


Figure 16: Not Connected

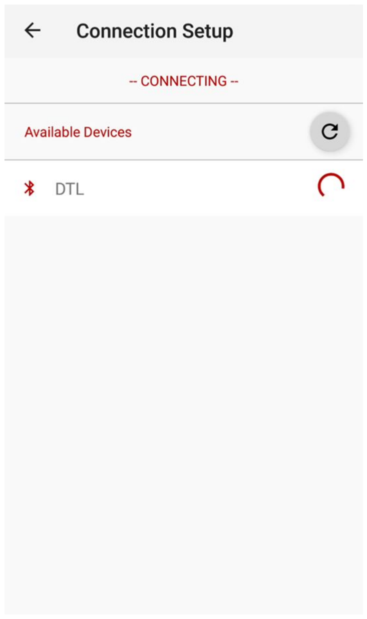


Figure 17: Connecting

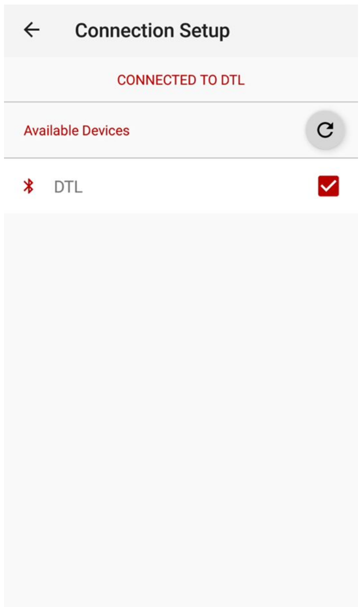


Figure 18: Connected

This is a simple and clean activity that allows the user to connect and disconnect to their DTL device. This Activity starts with a list view lists all Bluetooth devices that have the prefix “DTL” that the Android device is currently paired with - “Not Connected”. At this point, the user can click on any devices listed to establish a Bluetooth connection with the device.

To avoid freezing the UI thread, a BluetoothConnectionThread is used to to initiate the bluetooth connection. During connection, a spinner will be displayed beside the device name to indicate the Android device is currently connecting - “Connecting”. Once a connection has been established, the DTL will have a check-mark beside its name - “Connected”. Otherwise, if the connection fails, the thread will be disposed and the box beside the DTL will remain unchecked - “Not Connected”.

3. Gallery Activity

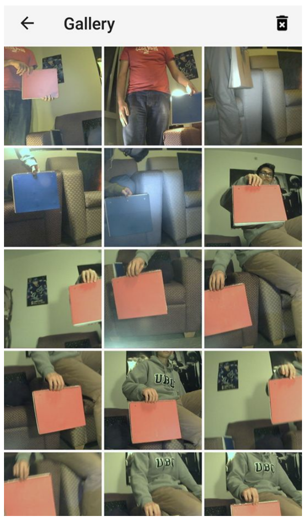


Figure 19: Gallery activity

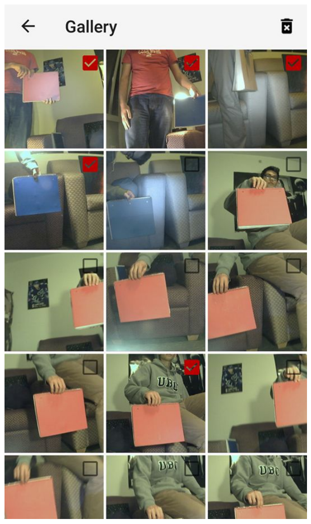


Figure 20: Multi-select option

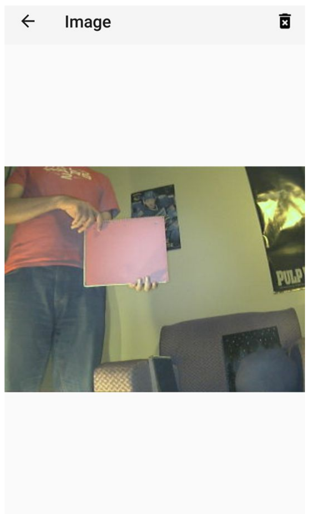
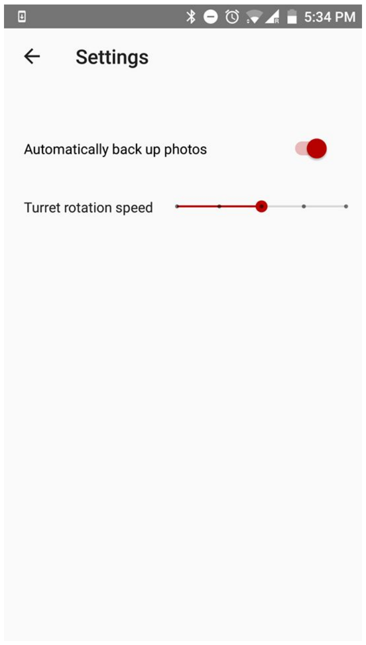


Figure 21: Individual image view

The settings menu gives the option to automatically backup photos to the in-app gallery. When this option is selected, photos that are taken from the turret in all modes are saved to be viewed for later in the apps built in gallery.

In order to achieve this, when an image is taken by the turret, it first creates a file, writes image data, and saves the image to device memory. The gallery activity consists of a gridview that uses a grid adapter populated with a list of these images obtained from memory. In terms of functionality, the gallery activity is scrollable. On click of an image, a new activity is created displaying that individual image. The user can swipe to either side to view the next or previous image, as well as delete the image and return to the gallery. On long click of an image thumbnail, multi-select option appears and allows the user to select and delete multiple images at once. In order to delete an image, it is both removed from the grid adapter's list of images, as well as from device memory.

4. Settings Activity



Automatically back up photos

Turret rotation speed

Figure 22: Settings Menu

Currently, this activity allows two settings:

1. Automatically backup photos, if this is checked, photos are all stored in a DTL Photos folder on the Android device
2. Turret speed, which dictates the speed parameter of movement commands

Designing a separate settings activity allows the designers to easily add or remove settings in the future.

6. Results

Module 2 of this project was overall very successful. Almost every project goal is accomplished except for saving images on a database, which was dropped early in the project. The one drawback in this project is the time it takes for images to process from the camera. The speed of processing increased dramatically from module 1 to module 2 due to the fact that the image did not have to be processed using NIOS, however it was still not as fast as initially hoped for. This meant video display is not possible, and introduced a delay in modes such as tracking. These limitations were partly due to the components used, such as the camera, and the processor available.

In terms of testing, the app was tested for robustness as each component was added. As bugs arose, they were recorded and added to a list of tasks to be done.

7. Integration of Standards

The DTL Application follows many of Android's design principles such as confirmation and acknowledgement pattern, choice of colours, accenting important text, and more.

Confirmation and Acknowledgement Pattern

Upon entering the Application, it asks the user to provide permission to access Bluetooth for device connection and phone storage to store photos. In the Bluetooth Connection Activity, once a device is clicked, a spinner will be displayed beside the device name to indicate a connection is in process. If the connection is successful, the check-box beside the device name will be checked. Otherwise it will be unchecked to indicate the connection was unsuccessful. Moreover, there exists a text box at the top of the activity that indicates each state - “Not Connected”, “Connecting” and “Connected”.

Colour Choice

Following Android's colour selection tips, red (#bb0000) was selected as the primary colour for buttons text and border colours, this overlays onto nicely onto white backgrounds.

Reflection on Design Standards

Imposing these design standards comes with its advantages and disadvantages. For instance, new designers are quickly exposed to how to design well and can learn quick. This will speed up the evolution of technology since they will not have to figure out these helpful patterns on their own. A disadvantage is a potential loss in creativity; If all designers follow these standards then many apps will have a similar feel and designers will be scared to try new ideas.

8. Conclusions and Suggested Future Work

The desktop launcher is a fully functioning, robust product. The reliability of connection and control, as well as user interface of the android app, provide a high quality user control experience.

If this product is to continue on to market stage, a necessary improvement would be refining visual style of the product. This includes improving the working prototype of the physical launcher to a market design with a high visual appeal, as well as refining the GUI of the LCD touch display to a more professional design. In terms of functionality, one improvement that could be made would be to increase image speed by either changing the camera, using a faster processor, or both. Another improvement is to reintroduce the feature of saving images on a remote database, which is not currently in the project due to time constraints.

9. Individual Contributions

Konrad Izykowski

- Added Bluetooth library to Nios
    - Created low level bluetooth chip communication functions
    - Worked with Zeyad to add message types to Nios
- Added calibration functions to Nios to improve motor accuracy
- Added functions to move motors by specific angles
- Added bluetooth mode functionality to Nios
    - Worked with Sawyer on adding the button
    - Added code with Zeyad to handle bluetooth commands from app
    - Worked with Zeyad to create pairing mode on Nios
- Added Motion detect and set photo resolution photo bluetooth commands
- Repaired and improved the physical turret assembly

Sawyer Payne

- GalleryActivity code and layout
- MainActivity layout
- SettingsActivity code and layout
- Worked on tap to touch functionality
- Helped with code functionality in MainActivity with other members

Zeyad Tamimi
- Improve Bluetooth connection reliability and connection with Nicholas
- Communication handling, send messages/respond to NIOS with Nicholas
- Stress test application to ensure reliability with Nicholas
- Implemented color blob tracking using openCV
- Designed the communication protocol and message structure
- Create the pairing mode on the NIOS II with Konrad
- Designed the standard messaging handling framework with Konrad
- Added the save to disk functionality to the Android application

Nicholas Wu
- Improve Bluetooth connection reliability and connection with Zeyad
- Communication handling, send messages/respond to NIOS with Zeyad
- Stress test application to ensure reliability with Zeyad
- Polishing user experience with Sawyer
- Add accelerometer movement feature
- Design state-machine for Main Activity
- Create framework for mode switches (ExecuteModeTask) and on-hold listeners (ExecuteCommandTask)
- Implement and integrate auto mode and security mode
- Create commands for AsyncTasks to communicate with UI thread to update UI (such as showing loading screen in auto or security mode)