Research and Development of a Visually Programmable Robot integrated with Augmented Reality Engine for academic users

by

Rahul R (MT2013119)

A thesis submitted

in partial fulfilment of the

requirements for the degree of

Master of Technology

in

Information Technology

International Institute of Information Technology, Bangalore.

June 2015

Thesis Certificate

This is to certify that the thesis titled **Research and Development of a Visually Programmable Robot integrated with Augmented Reality Engine for academic users** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Technology** is a bona fide record of the research work done by **Rahul R (MT2013119)** under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Madhav Rao

IIIT-Bangalore,

The 18th of June, 2015.

Abstract

"In this thesis, an attempt was made to build a new interactive educational-tool for students to inspire their interest in learning programming. The research objective in this thesis is an educational robot which the students can interact through a visual programming language. Two versions of the hardware platform is made available, a ready made robotic development platform known as the Electric Ray Robot and an easily buildable robotic platform, known as the InBot, thus giving a hands on experience in introductory robotics as well. A drag and drop programming environment is developed for desktop (MiniBloq) and mobile (InBlocks) devices to program the robots. An augmented reality simulation engine is integrated to the programming environment in a view to reach out to students with limited access to the hardware platforms. The results show that there is a promising and huge market potential to apply this new interactive technology to the development of educational robots."

Acknowledgements

This work might never have been completed without the generous help and support of my supervisor, Prof. Madhav Rao, who was always eager to help and his door was always open for discussion. I would like to thank him for being my mentor during the past six months and for all those long discussions that we had. I thank him for introducing me to the field of visual languages and most importantly for teaching me how to convey ideas in writing.

I would also like to thank Prof. Jaya Sreevalsan Nair and Mr. Ramesh Sundararaman for agreeing to be a member for my guiding committee.

Last, and not the least, I would like to thank my family who have always supported me. Without their support, encouragement and love this work would have never been completed.

— Rahul R

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A programming curriculum in an introductory programming syllabus requires students to learn foundational programming skills and concepts. But most introductory programming classes involve students with exercises that are dull and irrelevant. Even with proper examples, many programming languages add to the student's frustration because they require an understanding of the cryptic syntax before anything remotely interesting can be accomplished. Consider a simple program to display Hello World on the console. Following are the programs written in three popular programming languages: C, Java and Python, as shown in the listing 1.1, 1.2 and 1.3.

Listing 1.1: Hello World program in C

```
#include <stdio.h>
int main()
{
    printf("Hello World!");
    return 0;
```

```
}
```

Listing 1.2: Hello World program in Java

```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World!");

    }

}
```

Listing 1.3: Hello World program in Python

```
print "Hello World!"
```

Each programming language presents its own set of difficulties for a beginner to understand. In C, the challenge for students is to understand the concepts of main, preprocessor directive, header file and return type in a first instance. The Java program too involves concepts of class, access modifier, static keyword and method call. The Python program is more intuitive than C or Java. However the novice programmer has to remember the function to print the text. It was felt that the beginners tend to learn the intricacies of programming if they are motivated. The idea is to introduce to a set of building block functions which is easy to program and then introduce students with similar concepts in any of the programming languages mentioned above.

In this visual approach to programming, a programmer need not remember any of the constructs of a programming language, but only on the approach to solve a problem. Visual programming has been an active research area in recent years resulting in many visual programming languages. In computing, a visual programming language

(VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. Scratch shown in Figure 1.1 developed by the MIT Media Labs in 2005 is a popular visual programming language. Scratch is used by students, scholars, teachers, and parents to provide a stepping stone to the more advanced world of computer programming.

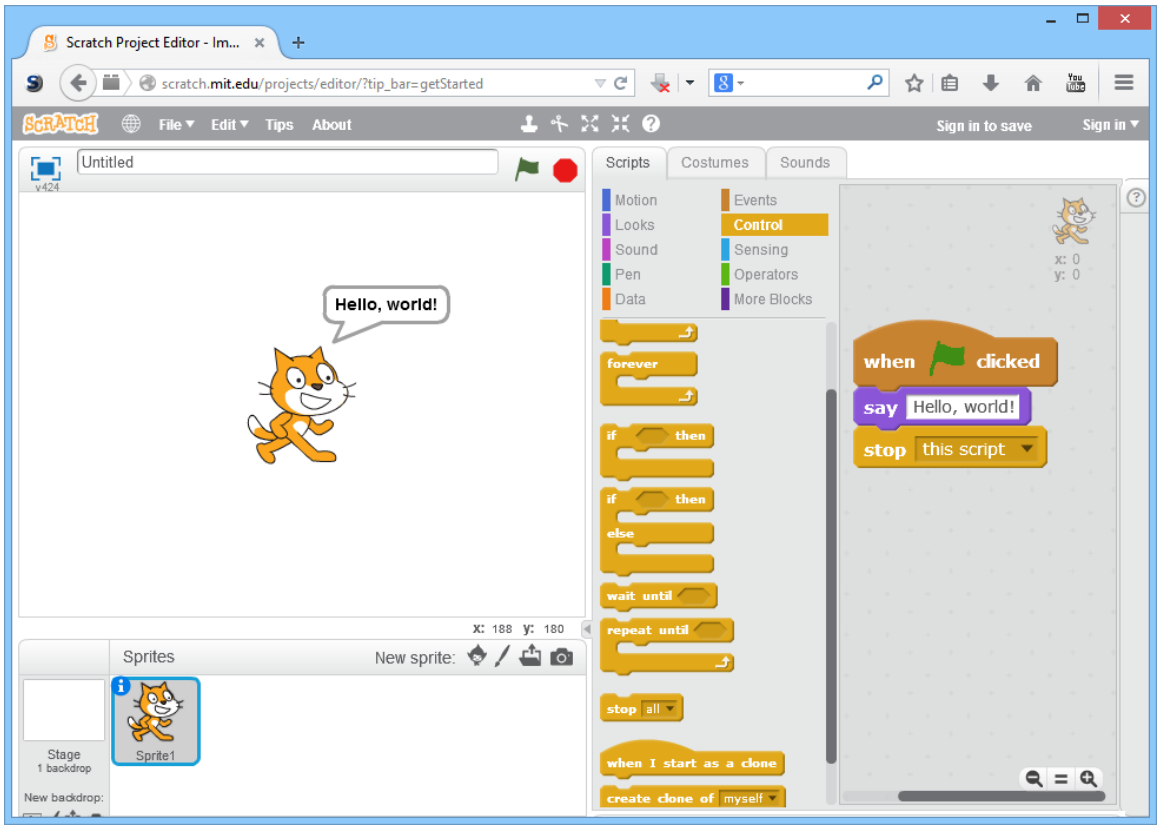

Figure 1.1: Hello World program in Scratch

In this research, we extend the idea of using visual programming language with a real robot to make it more engaging for the students. This also gives a platform to learn fundamentals of programming as well as robotics. Past reports have also suggested that robotics were used in teaching science, math and computing skills [13,

10, 7, 8].

LEGO [6] is known for its educational robotics line through the Mindstorms robotics kit since 2006. The Education EV3 Core Set consists of EV3 programmable brick (ARM9-based processor), motors, different types of sensors and around 500 accessories. Very simple programs for the robot can be created using the programmable brick itself. For complex programs, a visual programming software is included in the package. Lego's Mindstorm kit was adopted by Department of Computer Science of US Air Force Academy [9] to teach programming. However the LEGO kits are not affordable and the building blocks developed is not based on open source designs. The open source design implies that both hardware and software can be independently extended if required. A similar work was done by University of Alabama Computer Science Department which used a 3D graphical environment Alice [14] and iRobots [2] for its introductory CS course. Their course topics cover objects, methods, variables, loops, nested ifs, user input, parameters, events, random numbers, arrays and recursion. The software development was made open source, yet the hardware design was restricted to iRobot proprietary design.

Hence a need for an economical robotic kit with an open source hardware design and software platform is required for the beginners. The software platform should also support a simulation engine in case the hardware platform access is limited. This research proposes an augmented reality simulation engine, where the simulated robot is augmented on to the real world thus eliminating the need of designing a virtual world for the simulated robot. This research brings together diverse computer science streams such as Software Engineering, Computer Vision, Computer Graphics, Inter Device Communication and Robotics to build an engaging educational platform for

students.

Table 1.1: Bringing together various streams of Computer Science

No	Area	Tasks Involved
1	Software Engineering	Design and Development of the visual programming environment
2	Computer Vision	Image Processing techniques to identify a pre-defined marker for augmented reality
3	Computer Graphics	Modeling, Texturing, Exporting and Positioning the virtual content for augmented reality
4	Inter Device Communication	Bluetooth communication between the mobile device and the robot platform
5	Robotics	Robotic platform development

In my observation, visual programming is not trying to replace a textual programming language like C or Java. The visual language developed in this thesis is to help students learn fundamentals of programming as well as introductory robotics in an engaging manner without losing their interest. Once mastered the art of developing the programs using blocks, students should be able to easily graduate to a textual programming platform and program the robots.

Chapter 2

Electric Ray Robot

Electric Ray robot developed by Protocentral Inc, as shown in Figure 2.1, is an Arduino compatible [1] robot. It is an entry-level, low-cost complete robotic platform. It is simple enough to be used out-of-the-box for teaching and learning robotics and Arduino programming.

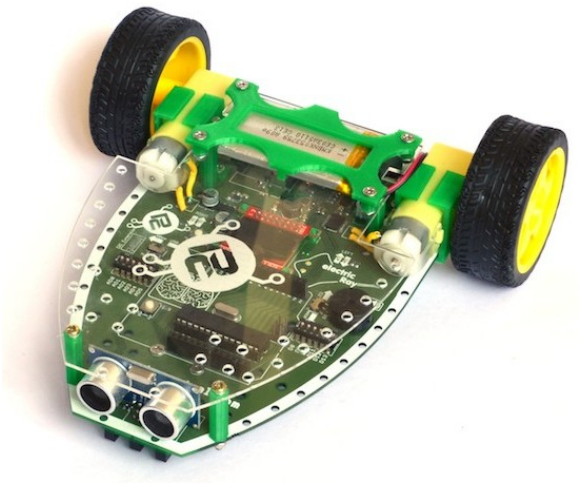

Figure 2.1: Electric Ray Robot

2.1 Specifications

- Atmega328 microcontroller with Optiboot bootloader (Arduino-Uno compatible)
- Arduino shield expansion headers
- Ultrasonic distance sensor
- 3x IR sensors
- OLED display
- Piezo buzzer
- FTDI USB-to-serial coverter
- TB6612FNG Motor Driver
- 2x 3V geared motors
- 2x wheels with rubber tyres
- 1100 mAH, 3.7V Li-Poly battery pack
- Clear acrylic sheet to cover the PCB

2.2 Component Details

The electric Ray robot is a single PCB-based platform which includes all the components required for a basic robotics project including microcontroller, sensors, motors, motor drivers and power supply.

From the top, an ultrasonic distance sensor module is placed in the front of the robot to measure the distance to the nearest obstacle in the robot's path. This produces a digital echo signal proportional to the distance to the closest object. On the bottom of the board at the same place, three IR sensors are mounted for line detection and following.

The ATmega328 (running at 16 MHz at 5V power) microcontroller onboard contains a Arduino-Uno compatible Optiboot bootlaoder that allows you to upload code through the Arduino IDE. The onboard FTDI USB-to-serial converter IC enables the connection of the microcontroller's serial port to the PC's USB port for programming and serial data transfer. Also connected to this microcontroller are female headers with a standard Arduino Uno footprint to take any Arduino shield compatible with the Arduino Uno.

A 2-color 128x64 pixels 0.96" Organic LED (OLED) display is present to display any messages from the code. This is connected to the ATmega328 through an SPI interface. The Adafruit SSD1306 OLED driver library can be used to control this display. An LED is also available onboard for user control and indication.

The motor driver is based on the TB6612FNG chip which is capable of driving two DC motors at up to 1.2 A continuous current. This driver controls two 3V DC motors with integrated gearboxes for speed control. The motor driver is also capable of driving the motors at adjustable speeds and in both directions.

Power supply is from a 3.7V, 1100 mAH Lithium-Polymer rechargeable battery pack. The 3.7V voltage is stepped up by an on-board boost DC/DC converter to the 5V required by all the components on the board. The battery can be recharged by connecting the robot to a PC through USB. Charging will continue even when

the power switch is in the OFF position. Battery charging will be shut off once the battery has been completely charged.

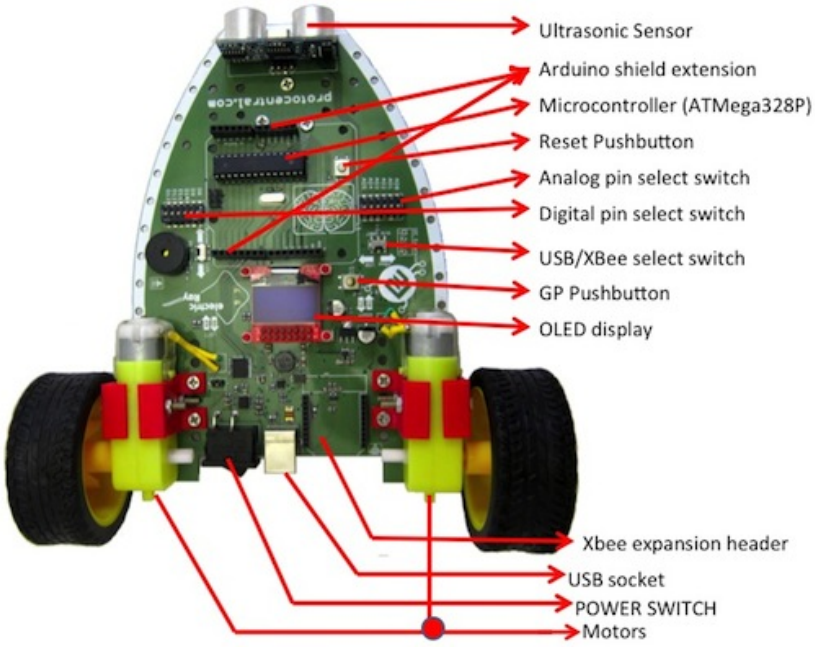

Figure 2.2: Electric Ray Robot

2.3 Programming the robot

To program the robot, connect the robot to your computer via USB. Open the Arduino IDE, and load the sketch located in Appendix B.

You need to tell the IDE which Arduino board you are targeting with your software, so open the Tools - Board menu and choose Arduino Uno.

The Arduino IDE must know which of your USB ports the robot is connected to. The Tools - Serial Port menu lists the available ports. If only one item is shown, click on that one. If two or more are shown, you can disconnect the robot and re-open

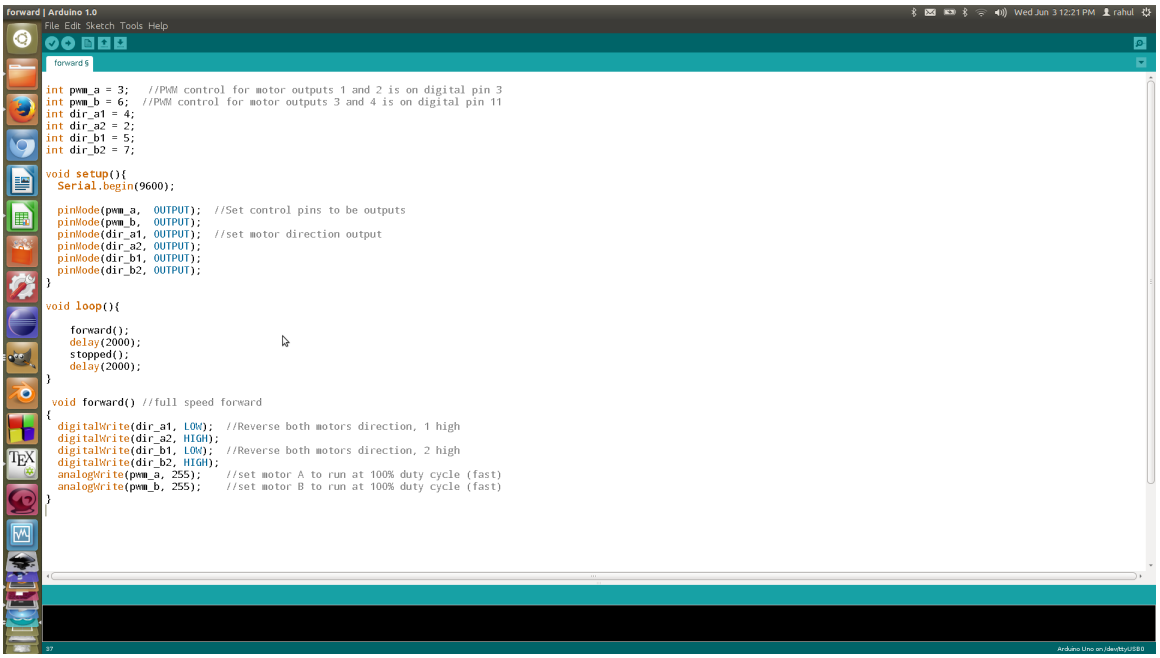

Figure 2.3: Arduino IDE

the menu; the entry that disappears should be the robot. Reconnect the board and select that serial port.

Click the "Upload" button in the top left of the IDE window. Wait a few seconds - you should see the RX and TX leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar of the software. Once this appears, you can disconnect the robot from the USB cable

With batteries in the robot, turn on the power switch and put it on the ground. The robot should start moving forward.

2.4 Summary

In my observation, a current limitation of the Electric Ray platform is the absence of an inbuilt wireless communication feature. Neverthless, provision has been provided in the open source hardware to gear the platform with a bluetooth shield or wifi shield enabling wireless communication. Also, another aspect to analyse is the memory limitations of the robot. As per the official documentation of [3] ATmega328 chip found on the Uno has the following amount of memory:

1. Flash 32k bytes (of which .5k is used for the bootloader)
2. SRAM 2k bytes
3. EEPROM 1k byte

Its noticable that there's not much SRAM available in the chip. If the program runs out of SRAM, it may fail in unexpected ways though it will appear to upload successfully, but not run, or run strangely. This scenario has to be explored further which would further determine the maximum number of blocks a student can program.

Chapter 3

MiniBloq

MiniBloq is a graphical development environment for Arduino and other platforms. Its main objective is to help in teaching programming using a drag and drop blocks called Action blocks. It is specially used at elementary, middle and high schools. The MiniBloq version v0.82 used in this research [4] comes in Windows version.

3.1 Enhancing MiniBloq for robotic platform

Minibloq is a graphical code generator with IDE capabilities. It's self-contained and every distribution includes the complete tools needed to compile (or interpret, depending on the selected target) and deploy the code to the selected hardware target. Every code block is configured in XML. To solve a given problem, the blocks are arranged in a logical manner and then the compiled executable file is transferred to the hardware via data cable. An advantage of Minibloq compared to other graphical programming softwares is that it can be enhanced to add new blocks with functionality specific to your hardware.

MiniBloq is specifically enhanced for Electric Ray robot by adding new blocks by abstracting the pin level details of the hardware. Thus, various blocks for for controlling movement, sensors, display and buzzer of Electric Ray Robot are available to program. The new blocks developed is shown in the Figure 3.1. The enhanced interface of MiniBloq is shown in Figure 3.2.

Figure 3.1: New blocks developed for Electric Ray

The blocks are designed to be intuitive. A "Hello World" program in the developed graphical tool is written by dragging and droping a block for OLED display, followed by text in the form of string literal as shown in the Figure 3.3. The program when ported to the Electric Ray robot will display the string "Hello World" in OLED.

Internally the programming environment converts building blocks to its corre-

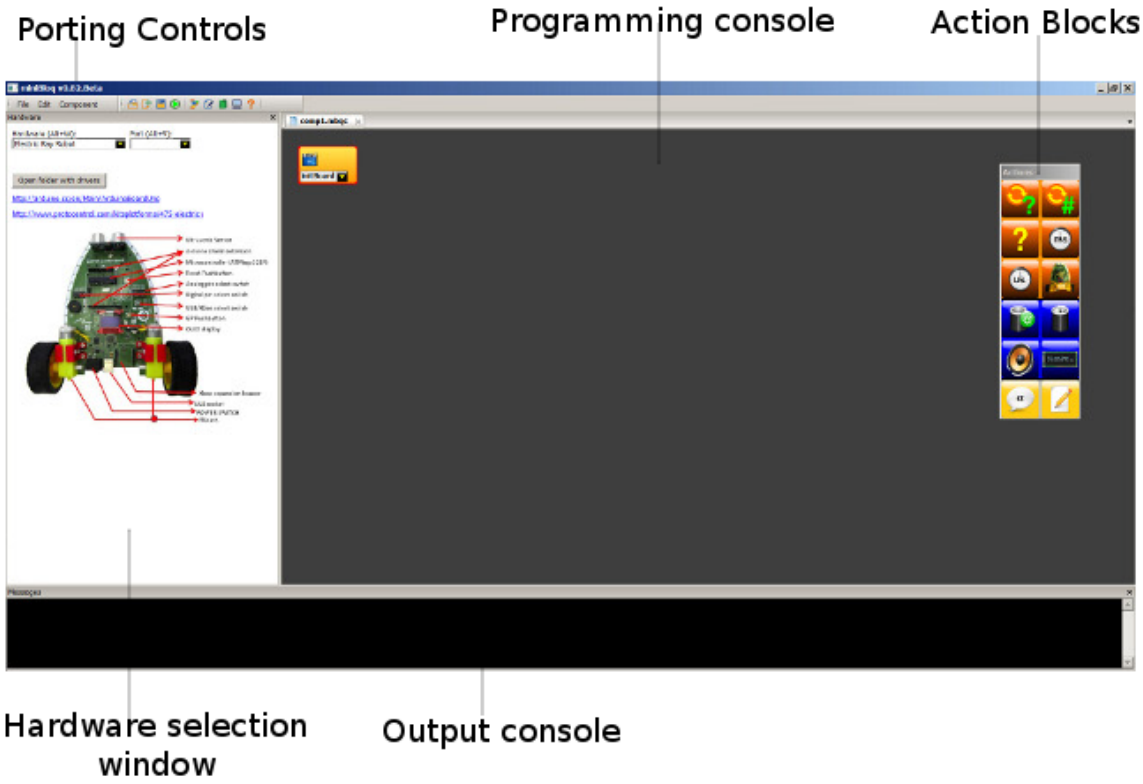

Figure 3.2: Enhanced Minbloq programming environment for Electric Ray

Figure 3.3: Hello World program in Minibloq

sponding C++ code compatible with Arduino. Students interested in modifying the existing property of the building block are allowed to edit the generated C code. For example a code generated for forward movement in the form of block is shown in the Figure 3.4.

Connect the robot using a data cable and choose the connected port in MiniBloq interface. Click on the green play button at the top. This will compile the program

```
void forward(int speed) {
   DigitalWrite(D4, false);
   DigitalWrite(D2, true);
   DigitalWrite(D5, false);
   DigitalWrite(D7, true);
   AnalogWrite(PWM3, speed);
   AnalogWrite(PWM6, speed);
}
```

Figure 3.4: Code generated for forward block

and port the program to the robot. A detailed manual on programming with MiniBloq and is made available at Appendix A

3.2 Conference Paper

Apart from fulfilling the technical objectives, a conference paper [11] based on this topic was recently presented to the 2nd IEEE International Conference on MOOCs, Innovation and Technology in Education(MITE). The aim of MITE 2014 was to provide a forum for academicians and professionals from various educational fields and with cross-disciplinary interests to network, share knowledge and engage in dialogue around the theme of fostering innovation and excellence in engineering education. As of 20th Decemner 2014, the paper has been accepted.

3.3 Summary

In my observation, the enhanced MiniBloq can be used as a learning tool for students of an age 13 and above. This drag-and-drop visual programming tool can teach fundamental programming concepts by programming a robot including:

1. Algorithm Design

2. Command Sequences and Control Flow

3. Conditionals

4. Loops

5. Sensors and Events through creative problem solving.

There is also provision to add new blocks to the exising robot which is explained in the manual available at Appendix A. A provision is also made to see the code running behind each block to understand the code generated behind the scenes. This feature will be useful for students when they want to graduate to a textual programming platform after expertising with the visual programming technique. A current limitation of the MiniBloq is with adding new hardware to visual programming platform. It is not a generic programming platform and supports only hardware boards belonging to the Arduino family. Further work is planned in future to analyse the platform and to make it more generic for different family of hardware and robots.

Chapter 4

InBlocks

Inventor's Blocks also known as InBlocks is an application developed for mobile devices running on Android platform. Similar to MiniBloq, it has a drag-and-drop programming language that snaps together like puzzle pieces. The created program can be then run on a real robot using bluetooth or on a virtual robot using augmented reality.

4.1 Control Flow

Refer Figure 4.1

4.2 User Interface Design

After the title screen is shown, user is navigated to a screen where he can choose the robot whether real or virtual. This screen is shown in Figure 4.2

Once the user selects an option, he is taken to the programming area. This is where

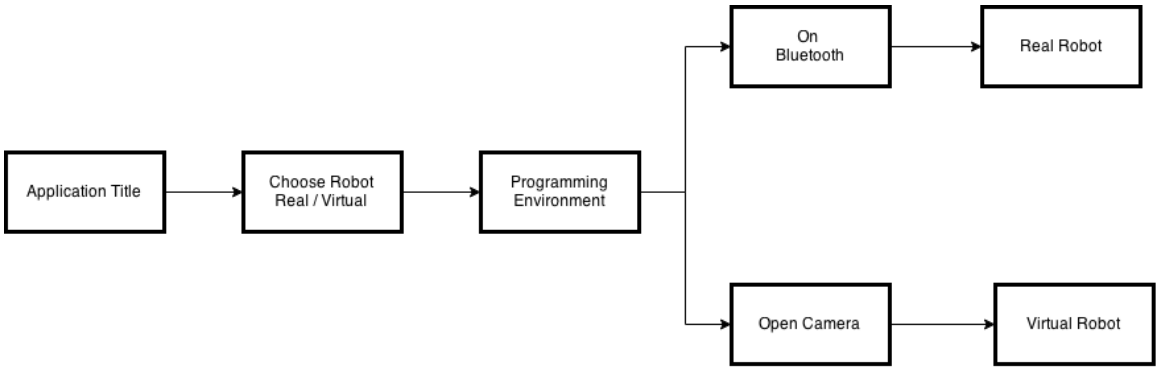

Figure 4.1: Application Control Flow

the user creates programs using drag-and-drop blocks which can be later ported to the medium selected in the previous screen. This programming area is developed using the Google's open source Blockly framework.

4.3 Google's Blockly framework

Blockly is an open source library released in 2012 for building visual programming editors for the web. The core graphical user interface of this environment is similar to Scratch, but also quite simplified and less cluttered. Users are presented with a Blocks pane in the left hand-side of the screen, where they can choose between different categories of block functions. The center of the screen, and the majority of the interface is devoted to the work area. This is a blank canvas where users can drag and drop blocks and connect them to form the structure of their code. In the lower right-hand corner of the screen is the trashbin, where users can drop chunks of blocks that they wish to delete. The upper side offers buttons to run the program.

Google's Blockly is programmed entirely in Javascript. This makes it easy to

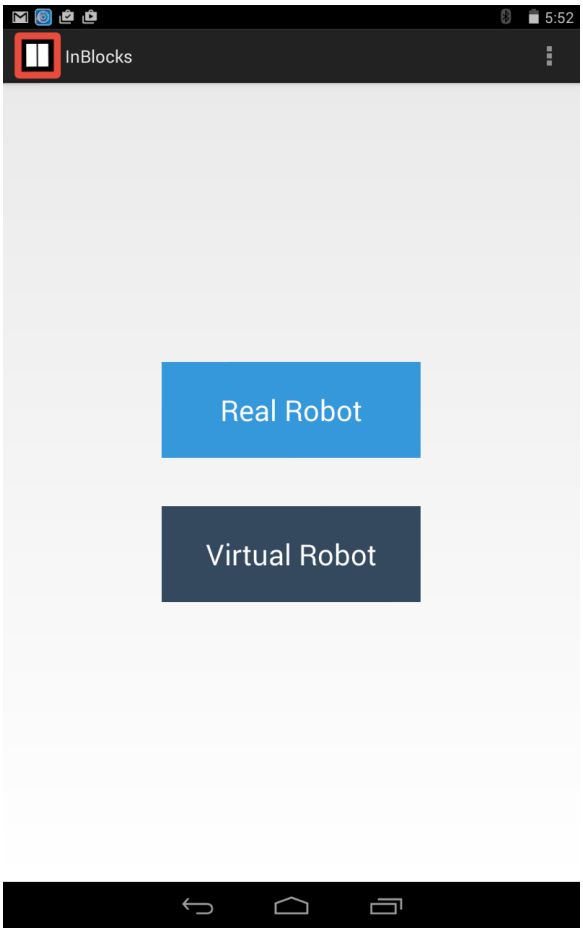

Figure 4.2: Screen to choose the robot

integrate with a webpage. Furthermore, Blockly has integrated interpreters that translate the output of the block strucutures to Javascript, Python, or XML. Finally, there is a provision to customize the new blocks available to the user and add new blocks using another web application, Block Factory. For this research, author has enhanced Google's Blockly framework so that it can be integrated to mobile devices running on Android platform.

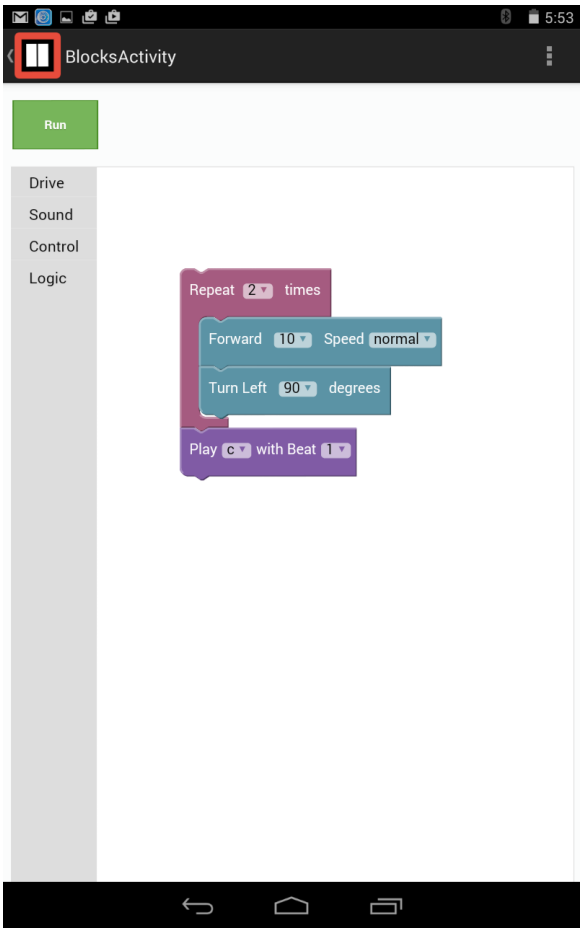

Figure 4.3: Programming Area

4.3.1 Adding Blockly to a page

- Include the core Blockly script and the core blocks set to the HTML page.

```
<script src="blockly_compressed.js"></script>

<script src="blocks_compressed.js"></script>
```

- Then include the messages for the user's language (in this case English):

```
<script src="msg/js/en.js"></script>
```

- Add an empty div somewhere in the page's body and set its size:

```
<div id="blocklyDiv"></div>
```

- Add the structure of the toolbox anywhere on the page:

Listing 4.1: Code snippet to include Blockly toolbox

```
<xml id="toolbox" style="display: none">
  <category name="Drive">
    <block type="drive_forward"></block>
    <block type="drive_backward"></block>
    <block type="drive_left"></block>
    <block type="drive_right"></block>
    <block type="drive_stop"></block>
    <block type="drive_obstacle"></block>
  </category>
  <sep></sep>
  <category name="Sound">
    <block type="sound_play"></block>
  </category>
  <sep></sep>
  <category name="Control">
    <block type="control_repeat"></block>
  </category>
  <sep></sep>
  <category name="Logic">
    <block type="logic_if"></block>
  </category>
```

```
</xml>
```

- Call the following to inject Blockly into an empty div. This script should be at the bottom of the page, or called by the onload event.

```
<script>
  var workspace = Blockly.inject('blocklyDiv',
      {toolbox: document.getElementById('toolbox')});
</script>
```

- Finally, place the html and javascript in the assets folder of the Android project directory and add the below lines to include the page to the Android Activity.

Listing 4.2: Code snippet to include a static html in Android

```
WebView myWebView = (WebView) findViewById(R.id.webView);
myWebView.loadUrl("file:///android_asset/index.html");
myWebView.getSettings().setJavaScriptEnabled(true);
myWebView.addJavascriptInterface(new WebInterface(this), "Android");
```

4.3.2 Creating new custom blocks

New blocks can be developed either using Block Factory at `https://blockly-demo.appspot.com/static/demos/blockfactory/index.html` or by using the API. A block developed for the forward movement of the robot is shown in source 4.3.

Listing 4.3: Code snippet to create a custom block

```
Blockly.Blocks['drive_forward'] = {
  init: function() {
```

```
    this.setHelpUrl('http://www.example.com/');
    this.setColour(195);
    this.appendDummyInput()
        .appendField("Forward ");
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown([["10", "10"]]), "DISTANCE");
    this.appendDummyInput()
        .appendField(" Speed");
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown([["normal", "200"]]), "SPEED");
    this.setInputsInline(true);
    this.setPreviousStatement(true, "null");
    this.setNextStatement(true, "null");
    this.setTooltip('');
  }
};
```

4.3.3 Generating Code

Blockly applications need to turn blocks into code for execution. For this a new JavaScript file needs to be included in the script tag of the HTML file. The Javascript file containes the code to be generated for a specific block. The code generated for the forward block is show in source 4.4

Listing 4.4: Code snippet to generate code for a custom block

```
Blockly.JavaScript['drive_forward'] = function(block) {
```

```
  var dropdown_time = block.getFieldValue('DISTANCE');
  var dropdown_speed = block.getFieldValue('SPEED');
  var code = 'forward("' + dropdown_time + '","' + dropdown_speed + '");\n';
  return code;
};
```

4.3.4 Interpreting the code

The code generated was interpreted using an open source sandboxed JavaScript interpreter in JavaScript downloaded from `https://github.com/NeilFraser/JS-Interpreter`. The interpreted commands are then passed to the Android platform and processed based on whether a real robot or virtual robot was selected.

4.4 Summary

InBlocks has been designed in such a way that the core of the algorithm gets evaluated on the mobile device and then it calls the API provided on the robot when required. This is the major difference in the architecture between InBlocks and MiniBloq. In MiniBloq, the entire code is compiled and transferred to the robot. The current version of InBlocks supports basic blocks to design algorithms and to call the APIs of the motor and the buzzer. Any number of new blocks can be designed to create programs with complicated logic. A limitation of the current version is that it doesn't support running infinte loops on the robot like the MiniBloq. InBlocks requires the program to contain finite number of steps. This limitation can be overcome by adding a "Stop" button to the application which would stop executing the running program

when necessary. The complexity of experiments conducted in InBlocks were medium in nature. The longest program executed had 20 blocks which was enclosed in a while loop repeating 4 times. Further complex experiments were the program contains conditionals and sensor events returning values back to the mobile device are planned to be conducted in future.

Chapter 5

Augmented Reality

Augmented Reality is artificial computer generated stimuli which is overlaid onto physical world. It differs from Virtual Reality in a way that it does not suppress perception of physical world but mixes physical reality with virtual content. Refer Figure 5.1

A general procedure to generate augmented reality content is described below.

1. **Initialize camera** - This is to capture the real world content.

2. **Capture** - The captured real world content is processed to identify pre-defined patterns.

3. **Identify Marker** - Marker is a predefined pattern available within applications database. Image Processing techniques are used to recognize the presence of marker in the captured video feed.

4. **Retrieving Position/Orientation** - Once the marker is recognized, the position and orientation of the marker is identified and given to the application.

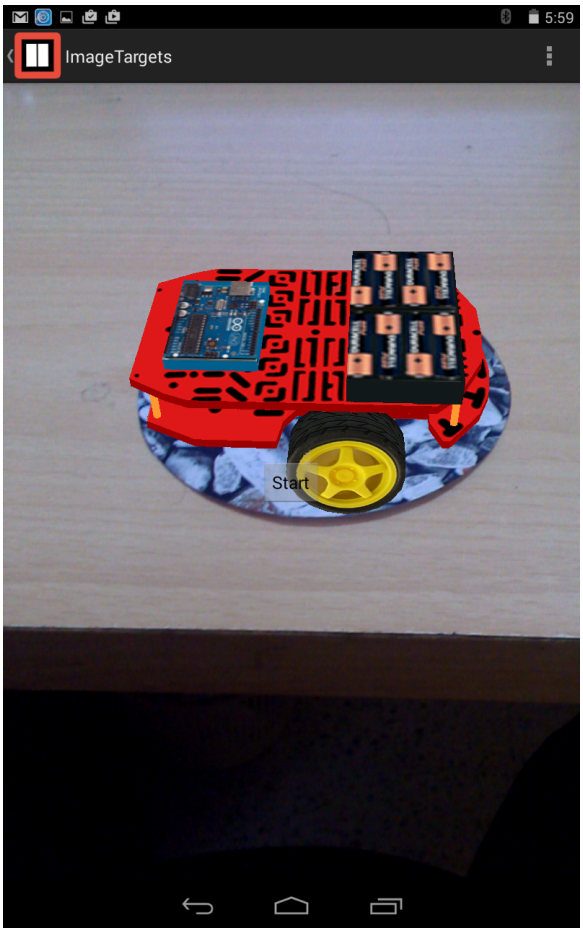

Figure 5.1: Augmented Reality Robot

5. **Render Object** - The application renders the virtual content using the position and orientation information got from the above step.

6. **Augment** - In this step the virtual content is layered over the real content.

Figure 5.2 illustrates these steps.

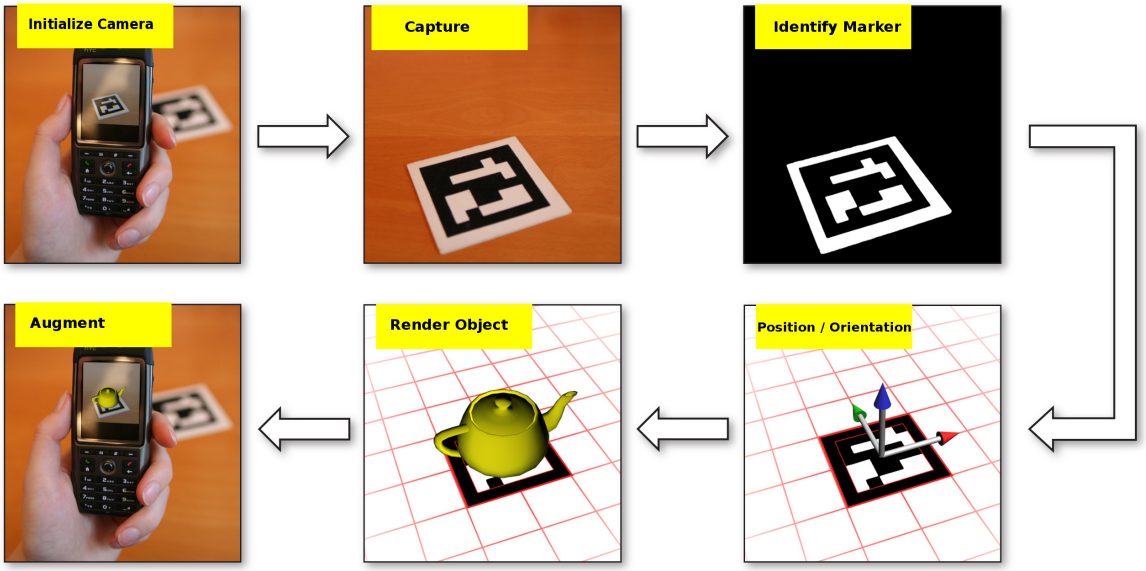

Figure 5.2: Steps to generated AR content

5.1 Vuforia Augmented Reality SDK

Vuforia is an Augmented Reality Software Development Kit (SDK) for mobile devices that enables the creation of Augmented Reality applications. It uses Computer Vision technology to recognize and track planar images (Image Targets) and simple 3D objects, such as boxes, in real-time. This image registration capability enables developers to position and orient virtual objects, such as 3D models and other media, in relation to real world images when these are viewed through the camera of a mobile device. The virtual object then tracks the position and orientation of the image in real-time so that the viewers perspective on the object corresponds with their perspective on the Image Target, so that it appears that the virtual object is a part of the real world scene.

5.2 Vuforia SDK Configuration

Vuforia SDK can be downloaded from https://developer.vuforia.com/downloads/sdk. To configure the SDK with Android Studio below steps were followed.

- Launch Android Studio
- Select File - Import Project and browse to the root directory of the Vuforia project you want to open
- Proceed in the Import Wizard dialog (click Next, Next) until you reach a page with this message "Alternatively, you can fill in the actual path map in the table below":
- Click to edit
- Enter the actual path to the Vuforia.jar library
- In the Project view, right-click on the Project and expand the view hierarchy so to locate the Vuforia.jar under "app/src/main"
- Right-click on Vuforia.jar to open the context menu
- Click on the "Add as library..." option in the context menu
- Alternatively, if you cannot locate the Vuforia.jar in your project hierarchy right-click on the Project and select "Open Module Settings", select "App", then select the "Dependencies" tab and click on the "+" button to Add a File Dependency and browse to the Vuforia.jar file

- Create a folder called "jniLibs" under the "app/src/main" folder under your Android Studio project directory
- Copy the "armeabi-v7a" folder (including the libVuforia.so file located inside it) from the "installdir/build/lib" to the "app/src/main/jniLibs" folder
- the resulting directory structure under your project root should be "/app/src/main/jniLibs/v7a/libVuforia.so"
- Clean and rebuild the project
- Run the app on your device

5.3 Creating the virtual content

The virtual content was created using a 3D modeling tool, Autodesk Maya. The process involved 3 steps -

1. 3D Modeling
2. Texture Mapping
3. Exporting the 3D Model

5.3.1 3D Modeling

3D modeling (or modelling) is the process of developing a mathematical representation of any three-dimensional surface of an object via specialized software. The outcome is called a 3D model. It can be displayed as a two-dimensional image through a process

called 3D rendering or it can be used for simulation purposes. The 3D model of the robot created is shown in Figure 5.3

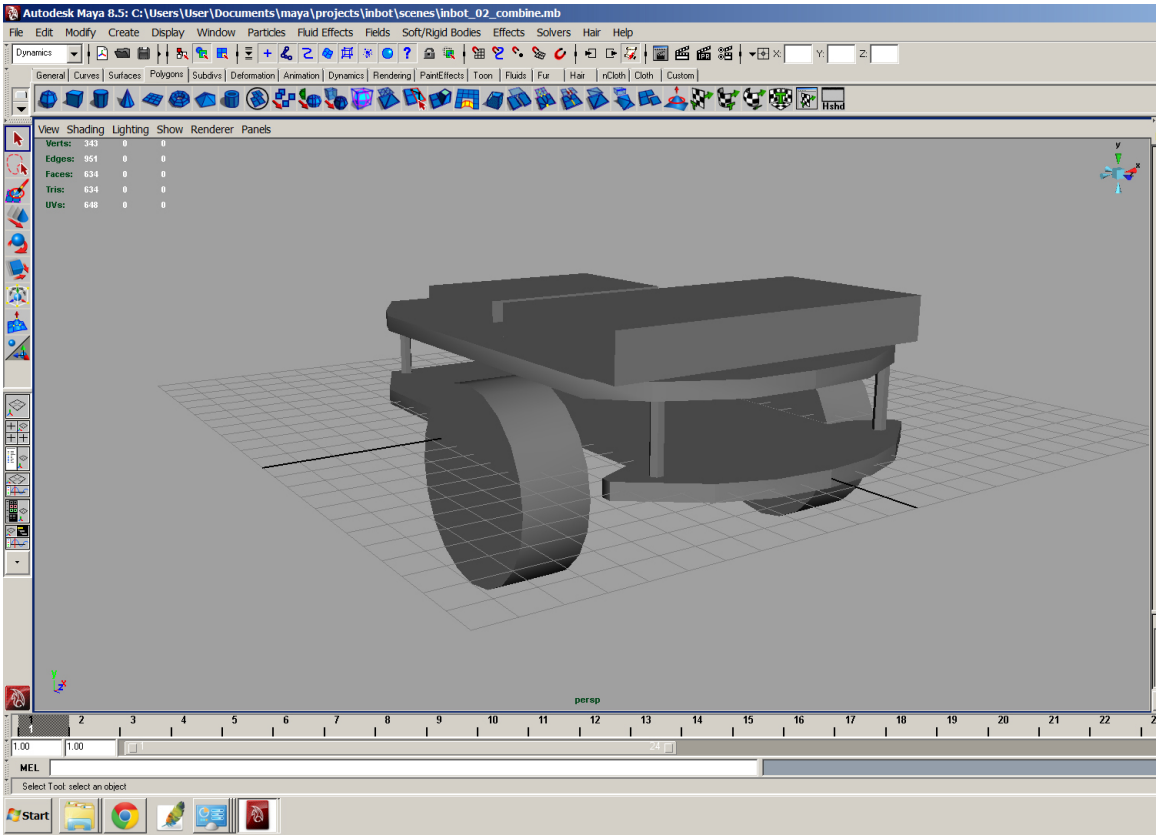

Figure 5.3: 3D model created in Maya

5.3.2 Texture Mapping

Texture mapping is a method for adding detail, surface texture (a bitmap or raster image), or color to a computer-generated graphic or 3D model. A texture map is applied (mapped) to the surface of a shape or polygon. This process is akin to applying patterned paper to a plain white box. Every vertex in a polygon is assigned

a texture coordinate which in the 2d case is also known as a UV coordinate. Image sampling locations are then interpolated across the face of a polygon to produce a visual result that seems to have more richness than could otherwise be achieved with a limited number of polygons.

UV snapshot of the 3D model was exported using Maya. This is shown in Figure-??.

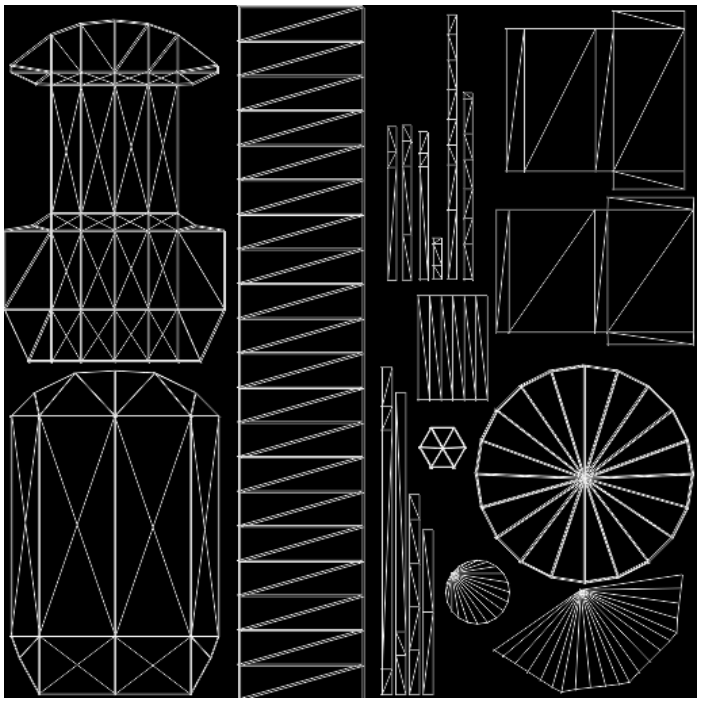

Figure 5.4: UV snapshot

This was then painted in Adobe Photoshop to generate the texture map used for the 3D model. The texture map and the texture mapped 3D model is shown in Figure-5.5 and Figure 5.6 respectively.

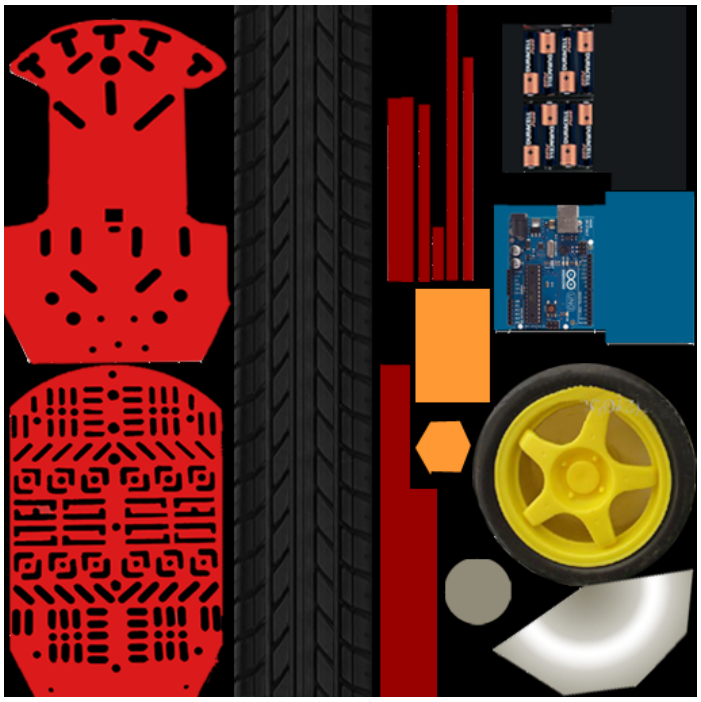

Figure 5.5: Texture Map painted using Photoshop

5.3.3 Exporting the 3D Model

Once the model was textured mapped, it was exported to OBJ file format. OBJ file format is a simple data-format that represents 3D geometry using the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. This information is used in Android platform as a class "InbotMesh" to import the virtual content.

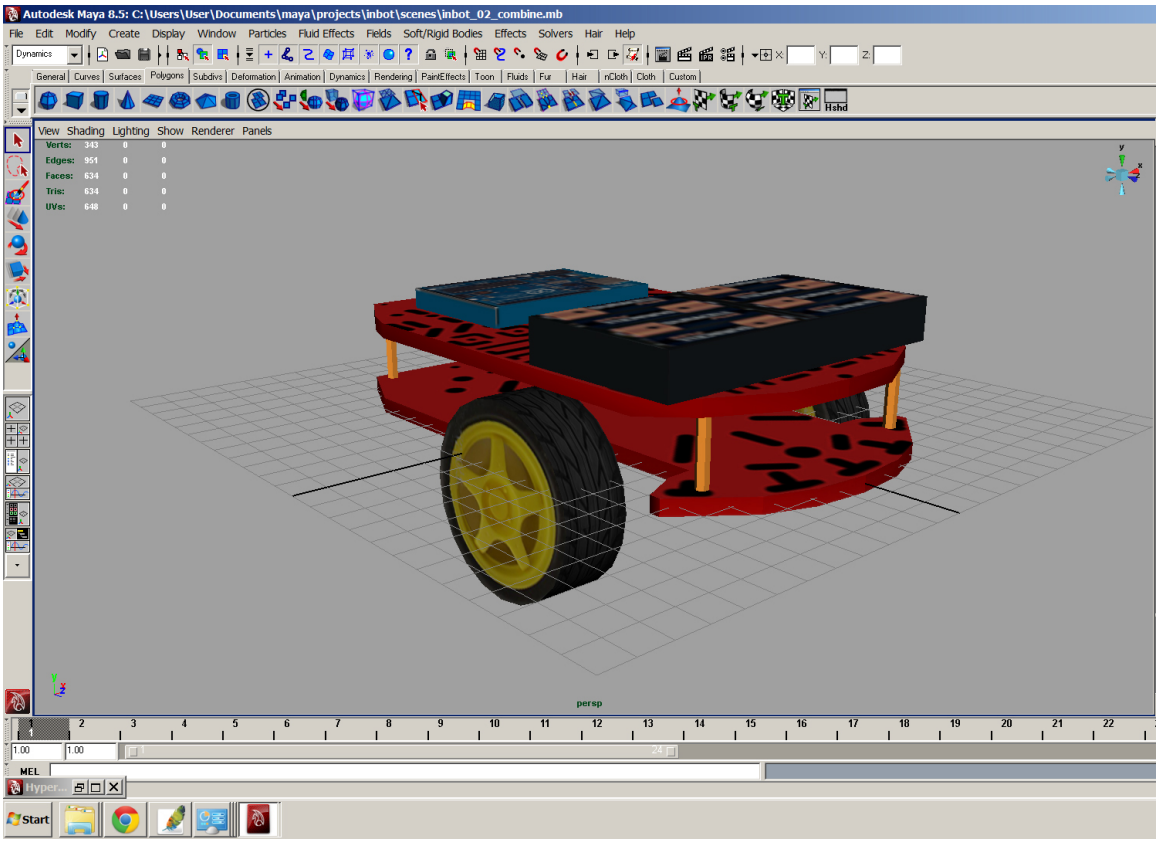

Figure 5.6: Texture Mapped model in Maya

5.4 Development with Vuforia

5.4.1 Image Targets

Image Targets represent images that the Vuforia SDK can detect and track. The SDK detects and tracks the features that are naturally found in the image itself by comparing these natural features against a known target resource database. This is shown in source 5.1. Once the Image Target is detected, the SDK will track the image as long as it is at least partially in the cameras field of view.

Listing 5.1: Mention the target resource database

```
mDatasetStrings.add("StonesAndChips.xml");
```

5.4.2 Positioning the virtual content using OpenGL

The 3D virtual content is positioned and scaled to sit correctly on the target. By default, the pose matrix places content in the center of the target with X to the right, Z to the top, and y coming out of the target.

The code for rendering the 3D model is located in the renderFrame() method. The renderFrame() method is called at every frame and can be used to update the OpenGL view. The following code snippets 5.2 show the part of the sample code that builds the MVP matrix from the trackable Pose.

Listing 5.2: Constructing the MVP matrix

```
for (int tIdx = 0; tIdx < state.getNumTrackableResults(); tIdx++)
{
  TrackableResult result = state.getTrackableResult(tIdx);
  Trackable trackable = result.getTrackable();
  ImageTarget itarget = (ImageTarget)trackable;
  Matrix44F modelViewMatrix_Vuforia = Tool.convertPose2GLMatrix(result.
     getPose());
  float[] modelViewMatrix = modelViewMatrix_Vuforia.getData();
  float[] modelViewProjection = new float[16];
  Matrix.scaleM(modelViewMatrix, 0, OBJECT_SCALE, OBJECT_SCALE,
     OBJECT_SCALE);
  Matrix.translateM(modelViewMatrix, 0, 0.0f, 0.0f, 0.0f);
```

```
  Vec2F targetSize = itarget.getSize();
  Matrix.scaleM(modelViewMatrix, 0, targetSize.getData()[0],
  targetSize.getData()[1], 1.0f);
  Matrix.multiplyMM(modelViewProjection, 0, vuforiaAppSession.
     getProjectionMatrix().getData(),
   0, modelViewMatrix, 0);
}
```

The model mesh vertex arrays (vertices, normals, and texture coordinates) are fed to the OpenGL rendering pipeline. This task is achieved by doing the following:

- Bind the shader.
- Assign the vertex arrays to the attribute fields in the shader.
- Enable these attribute arrays.

This is shown in the code snippet 5.3

Listing 5.3: Enabling the attribute arrays

```
GLES20.glVertexAttribPointer(vertexHandle, 3, GLES20.GL_FLOAT,false, 0,
    mInBot.getVertices());
GLES20.glVertexAttribPointer(normalHandle, 3, GLES20.GL_FLOAT, false, 0,
    mInBot.getNormals());
GLES20.glVertexAttribPointer(textureCoordHandle, 2, GLES20.GL_FLOAT, false,
     0, mInBot.getTexCoords());
GLES20.glEnableVertexAttribArray(vertexHandle);
GLES20.glEnableVertexAttribArray(normalHandle);
GLES20.glEnableVertexAttribArray(textureCoordHandle);
```

Finally, activate the correct texture unit, binds the desired texture, and then renders the model using the glDrawElements call. In case the model doesn't have index information, the glDrawArrays method is used. Refer source 5.4.

Listing 5.4: Binding the texture and rendering

```
GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, mTextures.get(textureIndex).
   mTextureID[0]);
GLES20.glUniform1i(texSampler2DHandle, 0);
// pass the model view matrix to the shader
GLES20.glUniformMatrix4fv(mvpMatrixHandle, 1, false, modelViewProjection,
   0);
GLES20.glDrawElements(GLES20.GL_TRIANGLES, mInBot.getNumObjectIndex(),
   GLES20.GL_UNSIGNED_SHORT, mInBot.getIndices());
```

5.4.3 Controlling the virtual content

To control the virtual content, commands from the programming environment are parsed and converted to values for the translation and rotation matrix which then acts on the modelview matrix. For example if the command is to move forward by 10 units from the current position, the dx and dz values can be calculated as shown in Fig 5.7. These values are then passed to translate the modelViewMatrix. Refer Source 5.5

Listing 5.5: Moving the virtual robot

```
float dx = (float) ( Math.sin(Math.toRadians(headingAngle)));
```

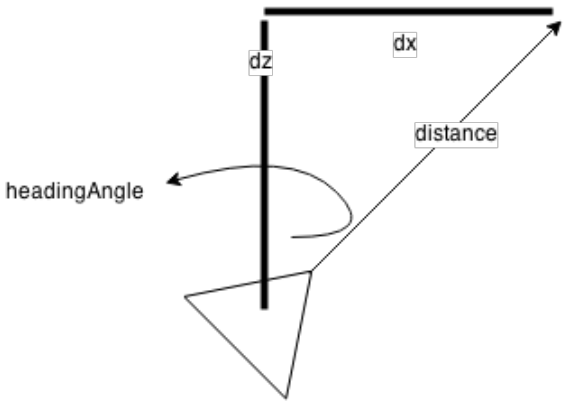

Figure 5.7: Translating the 3D model

```
float dz = (float) ( Math.cos(Math.toRadians(headingAngle)));

Matrix.translateM(modelViewMatrix, 0, dx, 0.0f, dz);
```

Chapter 6

InBot

Innovator's Bot also know as InBot is a differential drive robotic platform capable of teaching basic robotics and sensor integration. It is designed in such a way that the parts can be purchased online and can be easily assembled. A comprehensive Arduino library was written to support the existing peripherals. Once the library is installed, you can program the InBot using InBlocks.

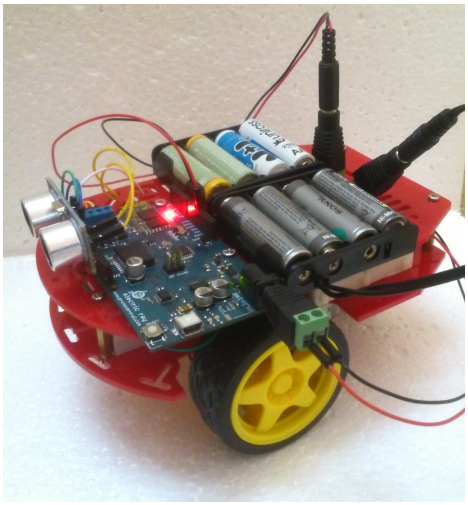

Figure 6.1: InBot

6.1 InBot Hardware

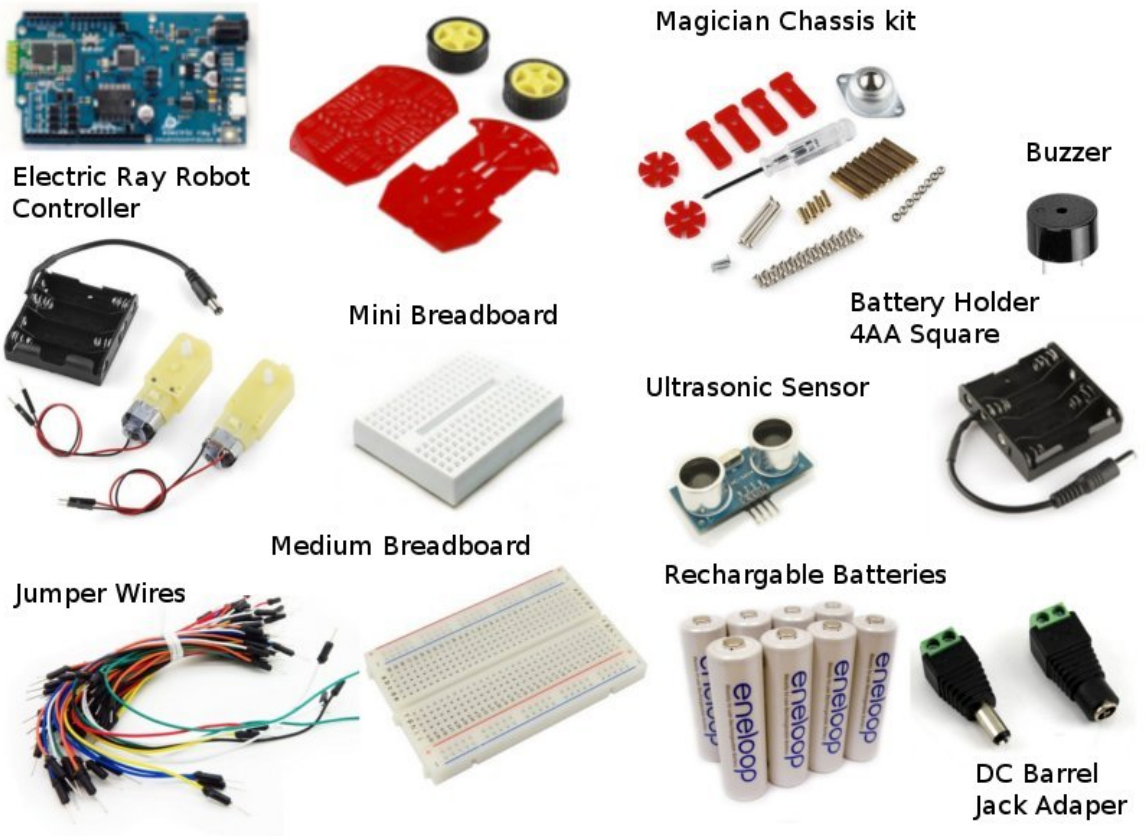

Figure 6.2: InBot hardware components

The Magician Chassis is an economical robot platform which features two gearmotors with 65mm wheels and a caster. Electric Ray is a completely integrated, easy-to-use robot controller fully compatible with the Arduino Uno and programmable with the Arduino environment. With an Arduino-compatible Atmega328P microcontroller, dual motor controllers and a Bluetooth module it can be easily assembled to magician chassis.

Table 6.1: Component Cost and Vendor details

No	Name	Quantity	Price(Rs.)	Purchased from
1	Magician Chassis kit	1	1150.00	Tenet Technetronics
2	Electric Ray Robot Controller	1	3499.00	ProtoCentral
3	Mini Breadboard (Self Adhesive)	1	89.00	ProtoCentral
4	Medium Breadboard (Self Adhesive)	1	159.00	ProtoCentral
5	Ultrasonic Sensor	1	99.00	eBay
6	Buzzer	1	75.00	Tenet Technetronics
7	Battery Holder - 4AA Square	1	165.00	Tenet Technetronics
8	DC Barrel Jack Adaper	2	186.00	Tenet Technetronics
9	Rechargable Batteries	8	849.00	Amazon
10	Jumper Wires (65 piece pack)	1	199.00	ProtoCentral
	Total		6470.00	

6.2 InBot Assembly

A step by step illustration of assembling the InBot is shown starting from Figure 6.4

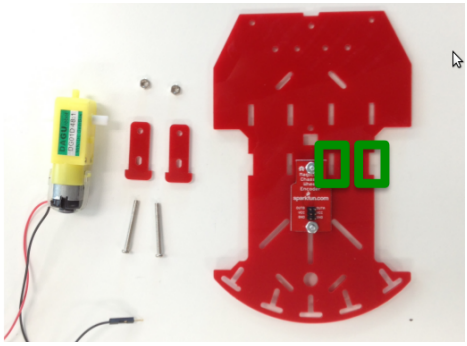

Figure 6.3: Before attaching right motor

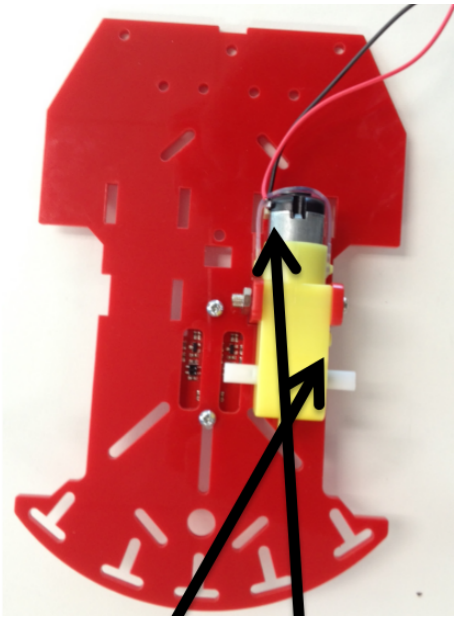

Figure 6.4: After attaching right motor make sure small knob on the side of the motor faces the outside of the robot.

6.3 Summary

During this research, special care has been taken to model the InBot robotic platform appealing, affordable and also similar to the Electric Ray robotic platform in terms of the hardware specifications. It also has an additional support for bluetooth communication. Bright colors were selected for chassiss to make it appealing for children. The ultrasonic sensors at the front were used to make it look like the eyes of the robot. The assembly and wiring of the robot is also carefully designed so that a beginner in electronics can easily assemble it.

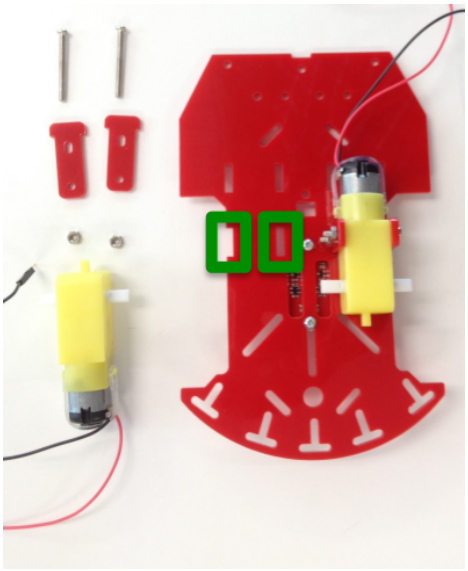

Figure 6.5: Before attaching left motor

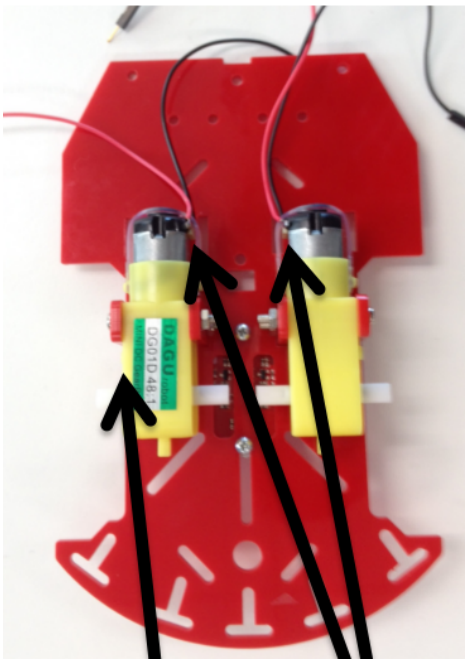

Figure 6.6: After attaching left motor make sure small knob on the side of the motor faces the outside of the robot.

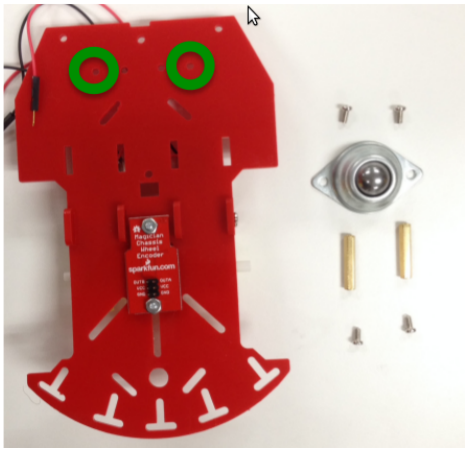

Figure 6.7: Before attaching omni wheels

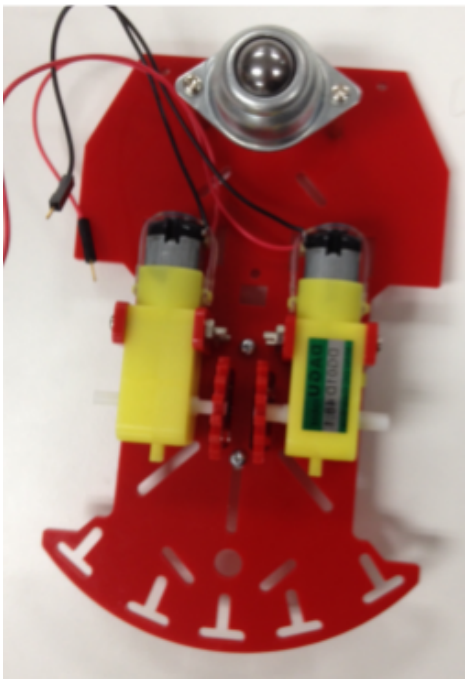

Figure 6.8: After attaching omni wheels

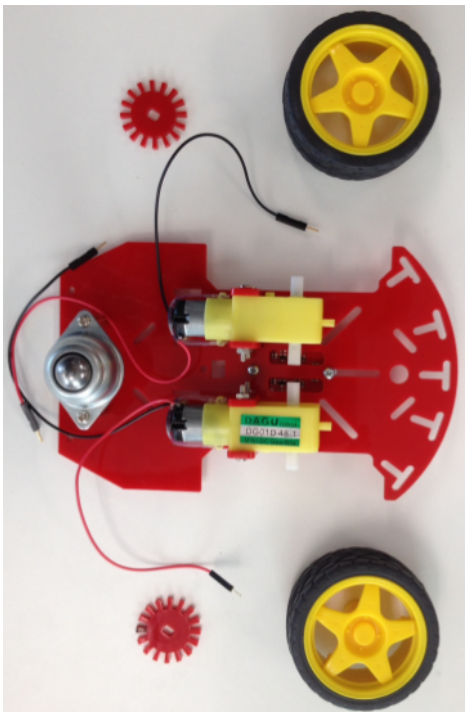

Figure 6.9: Before attaching wheels

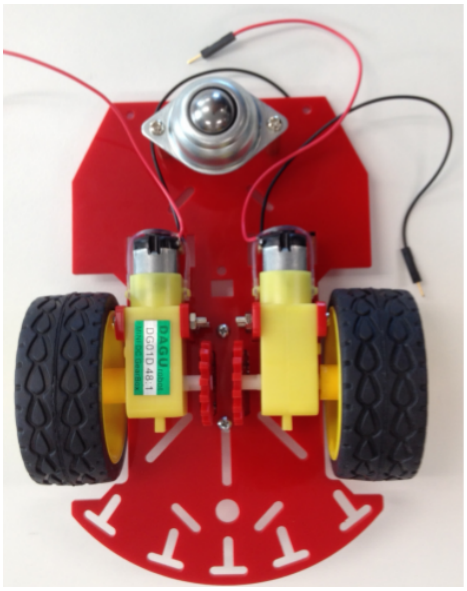

Figure 6.10: After attaching wheels

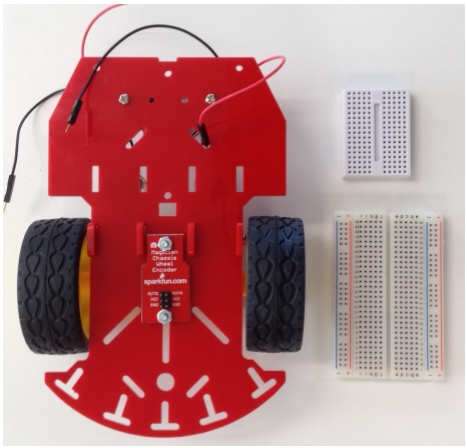

Figure 6.11: Before attaching bread boards

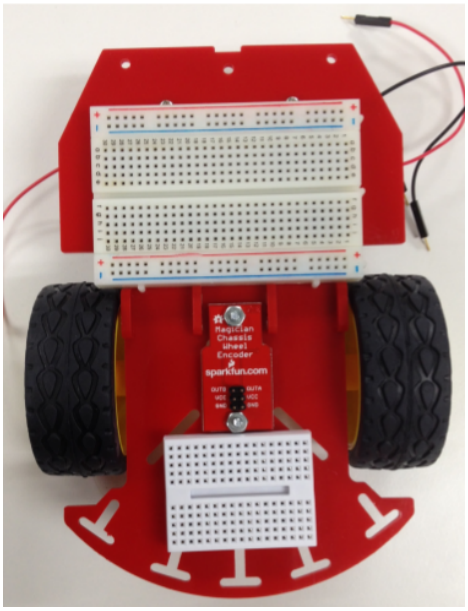

Figure 6.12: After attaching bread boards

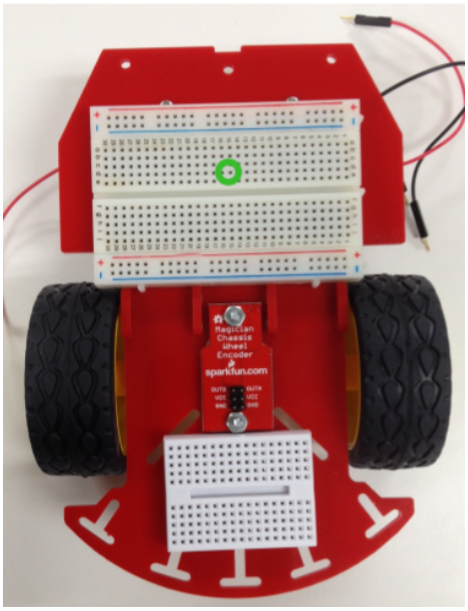

Figure 6.13: Before attaching buzzer

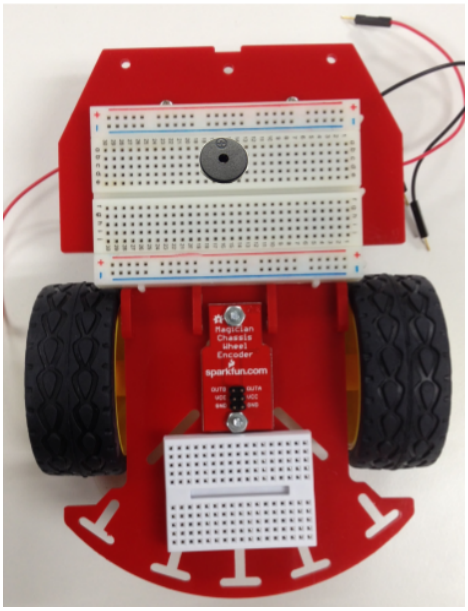

Figure 6.14: After attaching buzzer

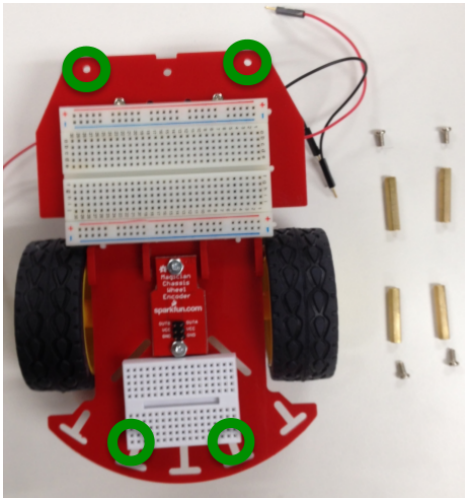

Figure 6.15: Before attaching chassis standoffs

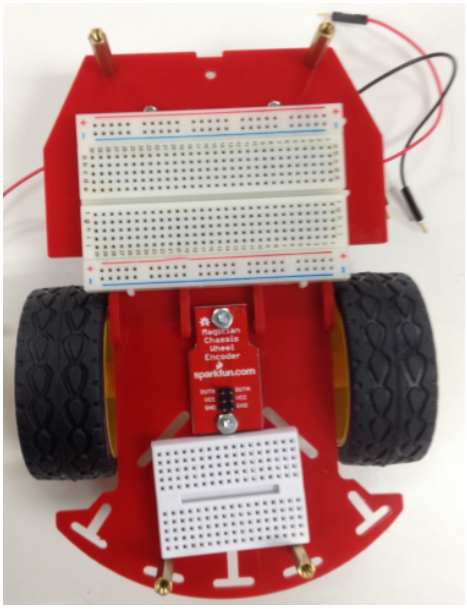

Figure 6.16: After attaching chassis standoffs

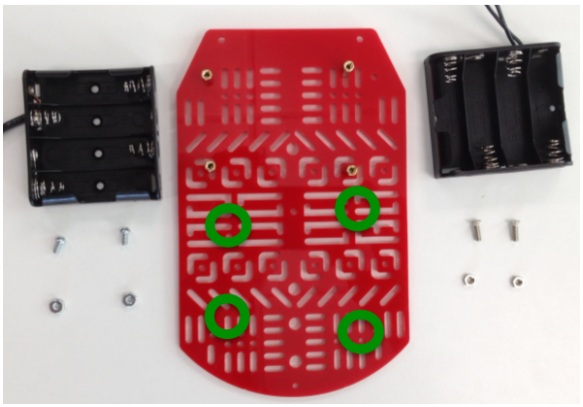

Figure 6.17: Before attaching battery holders

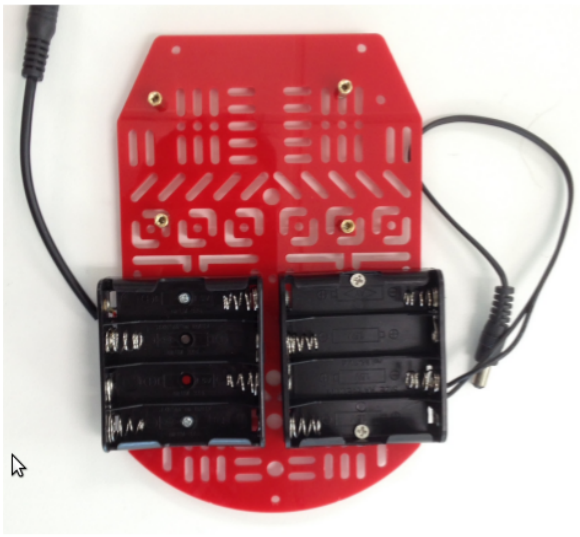

Figure 6.18: After attaching battery holders

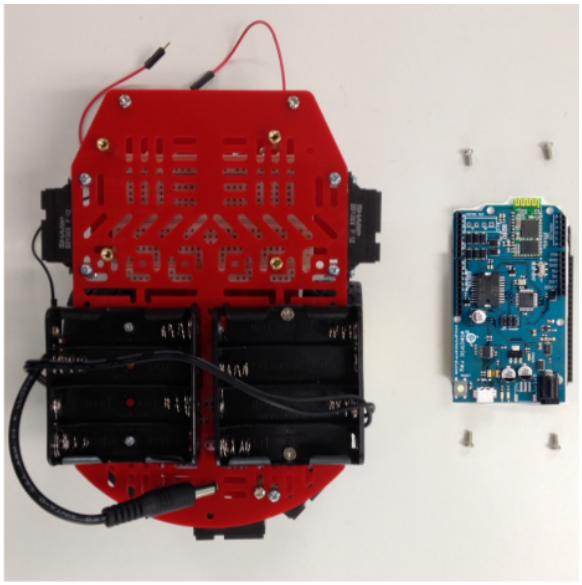

Figure 6.19: Before attaching Electric Ray Robot Controller

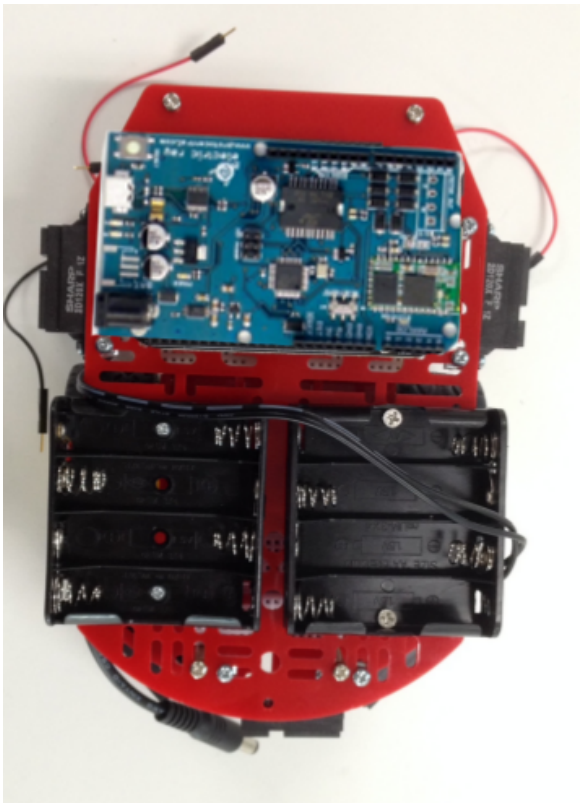

Figure 6.20: After attaching Electric Ray Robot Controller

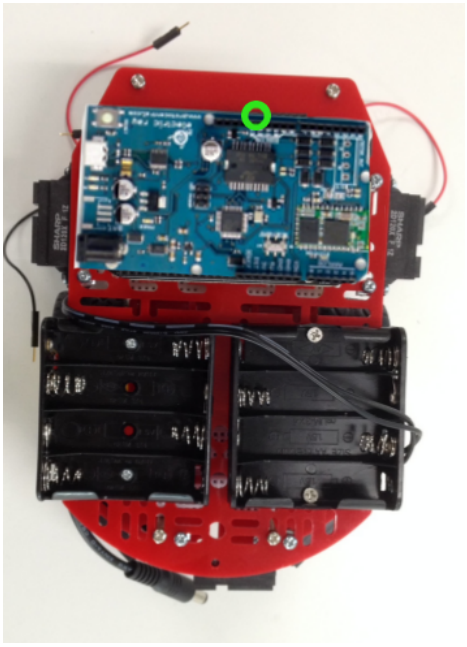

Figure 6.21: Before attaching Ultrasonic sensor

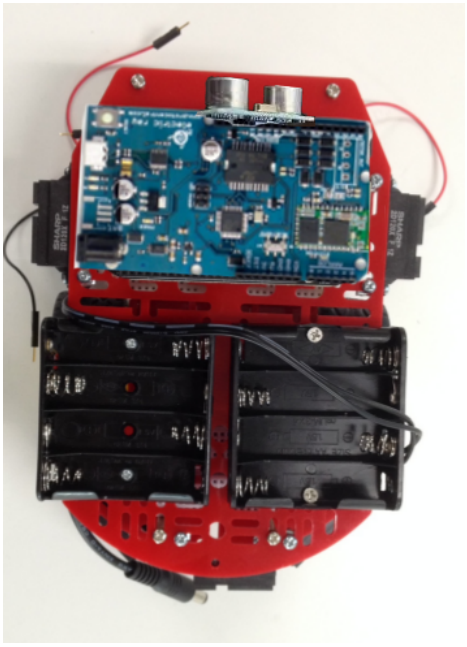

Figure 6.22: After attaching Ultrasonic sensor

Chapter 7

Bluetooth Communication

7.1 Introduction

Bluetooth is a popular method of communication between devices. Most of the smartphones today have the capability to communicate using Bluetooth. This is useful to mobile application developers whose apps require a wireless communication protocol. The objective of this chapter is to explain how the Bluetooth tools available to an Android developer were used in order to send and receive data to and from the robot wirelessly.

7.2 Development

There are several issues that must be overcome before the Android device can successfully transmit and receive data via Bluetooth. First, the Android must determine if it supports Bluetooth, and if it does, if Bluetooth is turned on. Then, it must pair and connect with the Bluetooth module on the robot. Finally, the Android must

actually send and receive data.

The first thing the program should do is determine if the Android device supports Bluetooth. To do this, create a BluetoothAdapter object using the function getDefaultAdapter(). If this returns null, then the Android device does not support Bluetooth. This is shown in source 7.1

Listing 7.1: Determine if Android supports Bluetooth

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
        // Device does not support Bluetooth
}
```

If getDefaultAdapter does not return null, then the Android supports Bluetooth. The next step is to determine if Bluetooth is enabled, and if it is not enabled, to enable it. Source 7.2 accomplishes this task, checking if the BluetoothAdapter is enabled and reacting accordingly.

Listing 7.2: Turn on Bluetooth if disabled

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, 1);
}
```

Next, the program has to retrieve the actual Bluetooth device it will communicate with, in this case the robot's Bluetooth module. The BluetoothAdapters getBonded-Devices() function will do this. This function puts all of the Androids currently-paired

devices into a storage structure called a 'Set'. Since only the robots Bluetooth module is paired with the Android, only this device will be in the Set. Assign this device to a BluetoothDevice variable. Source 7.3 demonstrates these steps.

Listing 7.3: Get the Bluetooth module device

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
        for (BluetoothDevice device : pairedDevices) {
                mDevice = device;
        }
}
```

At this point, the Android has the robots Bluetooth module stored in a BluetoothDevice object. The next objective is to form the connection between the Android and the robot. This work should take place in separate thread. This is because forming a connection can block a thread for a significant amount of time. Up until now, all of the programs code has been written in the main thread, or 'user interface thread' (UI thread). The UI thread should never be blocked. Therefore, create a new thread class where the connection will form. Source 7.4 shows the code to accomplish this.

Listing 7.4: Thread used for connecting Bluetooth devices

```
private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;
        private static final UUID MY_UUID = UUID.fromString("<ID>");
```

```
        public ConnectThread(BluetoothDevice device) {
                BluetoothSocket tmp = null;
                mmDevice = device;
                try {
                        tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
                } catch (IOException e) { }
                        mmSocket = tmp;
                }
        public void run() {
                mBluetoothAdapter.cancelDiscovery();
                try {
                        mmSocket.connect();
                } catch (IOException connectException) {
                        try {
                                mSocket.close();
                        } catch (IOException closeException) { }
                        return;
                }
        }
        public void cancel() {
                try {
                        mmSocket.close();
                } catch (IOException e) { }
        }
}
```

This thread requires a BluetoothDevice as a parameter and uses it to create a BluetoothSocket. This socket is what Bluetooth uses to transfer data between devices. The UUID used in Source 7.4 tells the socket that data will be transferred serially, which means one byte at a time. To start this thread, add the code in Source 7.5.

Listing 7.5: Starting the connection thread

```
mConnectThread = new ConnectThread(mDevice);
mConnectThread.start();
```

The code will now connect the robots Bluetooth module with the Android. The last objective is to send and receive data using this connection. Like connecting, transferring data is time-intensive and can block the thread, so this work should also take place in a separate thread. Create another inner thread class like that shown in Source 7.4. The thread requires a BluetoothSocket as a parameter and uses it to create an InputStream and OutputStream. The InputStream is used for reading data coming from the robot, and the OutputStream is used for sending data to the robot.

The command from android is sent in the format "TYPE ARGUMENT1 ARGUMENT2". E.g. "PLAY c 1". This implies that the robot's buzzer should play the note 'c' for 1 second. The program which runs at the robot, parses this command and directs the command to the required peripheral.

While reading the InputStream a "#" terminating character is required. This is because multiple samples of data can arrive in the InputStream between readings. When data is read in the InputStream, it is moved to the end of a buffer. After this happens, the buffer is iterated through, beginning at the location after the last "#" was found. If another "#" is found, the program 'handles' the data between the

previously-found "#" and this new "#". This process continues until the end of the buffer. If the last character in the buffer is a "#", the buffer is cleared after the data between the last 2 "#"s is handled.

Chapter 8

Conclusions and Future Work

The graphical programming tool along with the Electric ray robot were introduced to 42 High School students. A set of experiments including hello world program, robot following a square path, and obstacle avoidance were demonstrated to students. The students were asked to modify the existing program to make robot follow a triangle path. The time given to complete this activity was set to 2 hours. All 42 students were able to modify the program and demonstrate triangle path made by the robot, within the given time duration. Later IRobot which is programmable by Player-Stage enviroment were introduced to all students and similar programs were shown in player-stage environment [5, 12]. Player-stage environment was configured to write C programs. Three students were able to complete and demonstrate the triangle path made by robot. This suggests that students found the graphical enviroment more intuitive and syntax independent. In a survey, 34 students liked to program on our graphical programming tool. The remaining students wanted to learn programming in a traditional way. However these students were still interested to learn more about

the customized Electric Ray robot.

Figure 8.1: Students programming with MiniBloq

Dr. Monica Anderson, Associate Professor, Computer Science Department of University of Alabama has acknowledged this platform and would be using it in their Introductory Programming Course for students.

An advisable model of learning robotics in my observation is listed below.

1. The InBlocks app with the augmented reality robot can be used first to learn the building blocks of programming. This can be started by children even at the age of 9. Once mastered the programs with the augmented reality platform a student can choose to buy and program the real robot.

Figure 8.2: Program ported to Electric Ray

2. The MiniBloq and Electric Ray platform is recommended for students with age 13 and above. The platform can also be used in schools and colleges as an introductory course to programming and robotics. Once mastered the visual programming platform they can move to a textual programming platform like C, Java or Python.

3. The plug and play InBot platform is recommended for students of the age 16 and above. This can be used as a learning tool for students interested in embeded systems and building robots. They can use the InBlocks platform or the MiniBloq platform to program the robots. They can also program the robot using an Arduino IDE provided they having sufficient experience with textual

programming.

The current implementation of the augmented reality robot limits its ability to react with the real world. In future, a 3D scanning technique using the Smart Terrain library of Vuforia is planned. With this technique the augmented reality robot will have the intelligence to interact with objects and surfaces. The camera present in the mobile device should scan the world near the virtual robot and create a virtual mesh using depth sensing algorithms. This virtual mesh of the real world should be then used by the augmented reality robot for interaction.

Also, in future an improvement on the firmware which runs on the InBot is planned. The current architecture doesn't store the bluetooth commands it receives. The commands are processed real-time. This can lead to loss of packets if the commands are received at a fast rate. This limitation can be improved by using a datastructure to store the commands in InBot's memory and then process the commands one after the another.

Bibliography

[1] Electric ray. http://www.protocentral.com/robotics-kits/475-electric-ray-robot.html. [Online; accessed 11-Feb-2015].

[2] irobot. http://www.irobot.com/. [Online; accessed 21-Jan-2015].

[3] memory. http://www.arduino.cc/en/Tutorial/Memory. [Online; accessed 11-Jun-2015].

[4] Minibloq. http://blog.minibloq.org/. [Online; accessed 10-Feb-2015].

[5] Player stage. http://playerstage.sourceforge.net/. [Online; accessed 04-Apr-2015].

[6] S. Brigandi, J. Field, and Y. Wang. A LEGO Mindstorms NXT based multirobot system. *Advanced Intelligent Mechatronics (AIM)*, 2010.

[7] K. Cannon, M. LaPoint, N. Bird, K. Panciera, H. Veeraraghavan, N. Papanikolopoulos, and M. Gini. No Fear: University of Minnesota Robotics Day Camp Introduces Local Youth to Hands on Technologies. *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2006.

[8] J. Davies, B. L. Wellman, M. Anderson, and M. Raines. Providing robotic experiences through object-based programming (PREOP). *Proceedings of the 2009 Alice Symposium*, 2009.

[9] B. S. Fagin, L. D. Merkle, and T. W. Eggers. Teaching computer science with robotics using Ada/Mindstorms 2.0. *ACM SIGAda Ada Letters*, 2001.

[10] C. Massey, G. Weaver, J. Ostrowski, and T. Amos. Agents for change: Robotics for girls. *National Science Foundation, Division of Resource Development. Award*, 2000.

[11] R. Rahul, A. Whitchurch, and M. Rao. An open source graphical robot programming environment in introductory programming curriculum for undergraduates. *2014 IEEE International Conference on MOOC, Innovation and Technology in Education (MITE)*, 2014.

[12] R. T. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2008.

[13] S. Wagner. Robotics and children: Science achievement and problem solving. *Journal of Computing in Childhood Education*, 1998.

[14] B. L. Wellman, J. Davis, and M. Anderson. Alice and robotics in introductory cs courses. *ACM New York*, 2009.

Appendix A

MinBloq Programming Manual

A.1 Hello World

Goal: To display "Hello World" on the OLED display of the robot.

Click on the OLED display block. On clicking the red triangle to the right side of the block, a new pop-up expands which shows all the blocks related to display block. Click on the first block which is used to display a string. Click on the red triangle of the string block and select the last block "abc" which will help you type a string. Type "Hello World" as shown below. This completes the program to display "Hello World". Finally port your program on to the robot. The program is shown in Figure A.1

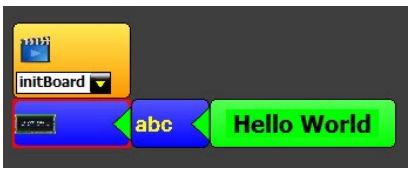

Figure A.1: Program to display Hello World

A.2 Delays

Goal: Blinking "Hello World" display.

Delay block pauses the program for the amount of time specified as parameter. There are two types of delay blocks - the millisecond delay block and the micro second delay block. Click on the millisecond delay block. Then click on the red arrow on the right hand side of the block that just appeared. Now, on the the second row of buttons that just popped up, click on the hash symbol . This is to send a constant value as a parameter. A box with a zero should have attached itself to the right side of the delay box. Type 1000 into it. This will tell the robot to wait one second (1000 ms) before executing any other operations. The below program will blink the "Hello World" display 2 times with a delay of 1 second in between. Complete program is shown below in Figure A.2

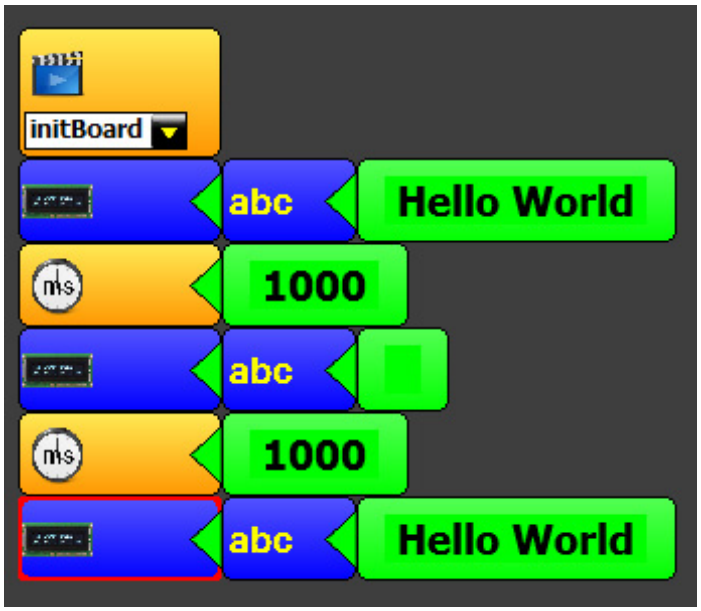

Figure A.2: Program to demonstrate delays

A.3 Variables

Goal: Pass the value for the delay block in previous program as a variable.

Variables are used when we want to store the data in the program. You can use them to hold values and then come back to them when you want to use that value. For example in the previous program, we have used 1000 ms for the delay block at two places. Instead of passing the value directly, we can store the value in a variable and then pass the variable to the delay block.

On the action window in Minibloq, there is a button that looks like a can. An empty can will define a new variable, and can only be used at the top of the program. A full can will reassign a variable's value. Click the empty can to initialize a new variable. Give it a name "delay". Assign a value 1000. Now in the previous program pass the delay variable for the delay block. Complete program is shown below in Figure A.3

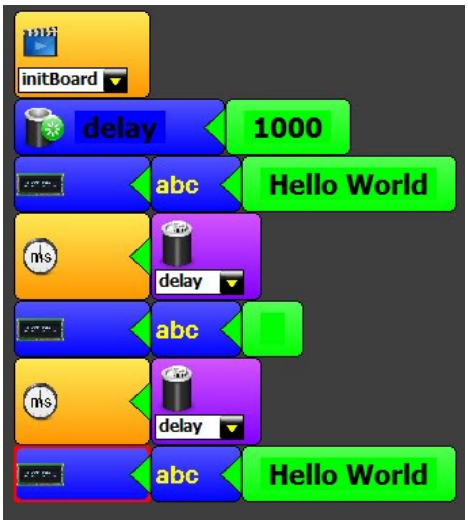

Figure A.3: Program to demonstrate variables

A.4 If Statements

Goal: Display "Hello World" if the value of a variable named "option" is 1 else display "Welcome".

An If statement allows the flow of the program to be changed, which leads to more interesting code.

On the second row of the actions box, there is a "?" block. Click on that to implement an if statement. Three blocks should appear on your screen. The top indicates the start of the if. The second is the else. The bottom indicates the end of the if statement. Clicking on the red arrow on the right side of the top block will open up a boolean menu. If the expression evaluates to true, any blocks between it and the middle "if" block will execute. Otherwise any blocks between the middle and the bottom will execute. Click the equal to sign. This block can accept two values. Select the variable as one value and 1 in the other value as shown below. Complete program is shown below in Figure A.4

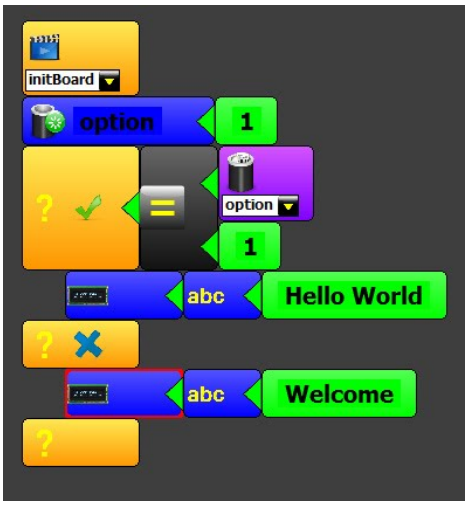

Figure A.4: Program to demonstrate if statement

A.5 While loop

Goal: Blinking "Hello World" continuously.

A while loop is a statement that allows code to be executed repeatedly based on a given boolean condition. The while construct consists of a block of statements and a condition. The condition is evaluated, and if the condition is true, the statements are executed. This repeats until the condition becomes false.

On the actions window, there is a button on the upper left hand corner with a "?" and a cycle on it. This is a while loop block. Click on that to get two blocks on the screen. The top one receives a condition and everything between it and the bottom block executes as long as it is true. Click on the red arrow on the while block. Click on the Tick mark. It means the condition is true. Here we have created a never ending loop. This is necessary as we want the "Hello World" blinking continuously on the display. Complete program is shown below in Figure A.5

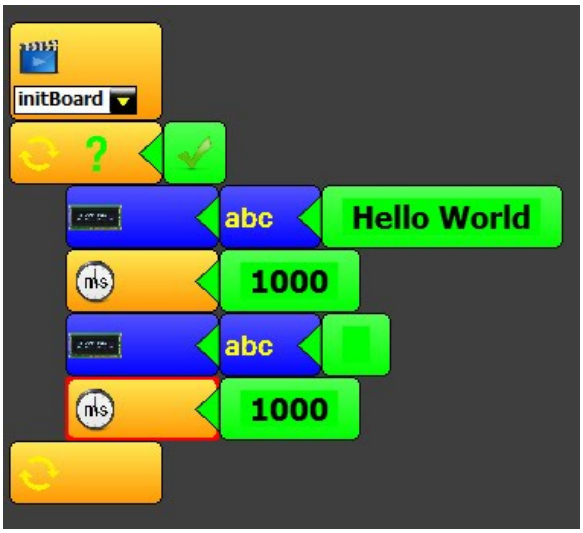

Figure A.5: Program to demonstrate while loop

A.6 Repeat Loop

Goal: To blink "Hello World" 10 times.

A repeat loop is a statement that allows code to be executed for given number of times. The repeat construct consists of a block of statements and a counter value. The counter value is incremented from zero for each block of execution. The statements are executed till the counter value is reached.

On the actions window, there is a button on the upper right hand corner with a hash symbol and a circle on it. This is a repeat block. Click on that to get two blocks on the screen. The top one receives a counter value and everything between it and the bottom block executes till it counter value is reached. Click on the red arrow on the while block. Click on the hash symbol block to give counter a numeric value 10. Add blocks to blink "Hello World" between the two blocks for repeat. The complete program is shown below in Figure A.6

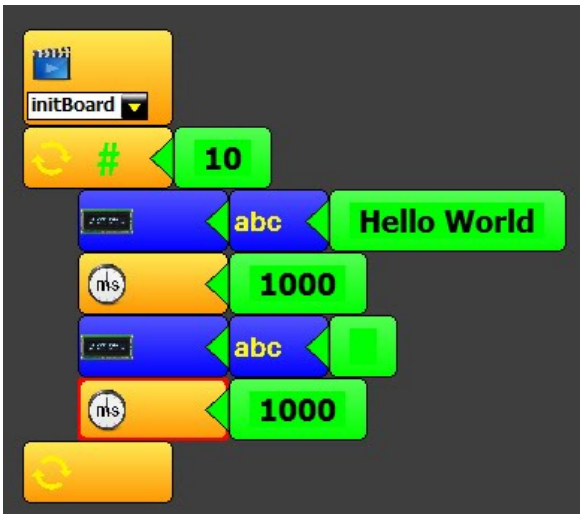

Figure A.6: Program to demonstrate repeat loop

A.7 Movement

Goal: To move the robot forward.

The movement block in the action window helps you to control the movement of the robot. This contains 5 blocks.

1. Forward block
2. Backward block
3. Turn Left block
4. Turn Right block
5. Stop block

To move the robot forward use the forward block. It takes in a parameter speed. Give a constant numeric value between 0 and 255. Add a delay block, to program for how much time the robot should move forward. If we give a value of 250, it means the robot keeps moving forward for 250 ms and then stops. Put the logic in a while loop so that, it keeps on moving forward every 250 ms. Complete program is shown below in Figure A.7

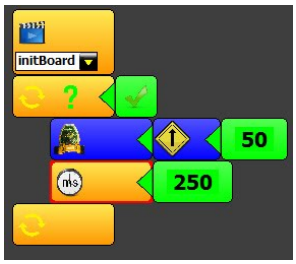

Figure A.7: Program to demonstrate robot movement

A.8 Obstacle Sensor

Goal: To stop the robot when it senses an obstacle ahead.

The obstacle sensor block, positioned in the number window, returns a numeric value (in cm) on how close the robot is to the obstacle. With this value you can set your own threshold, telling the robot on what to do when the threshold is reached. Check if the value returned by the obstacle sensor block is less than 20 cm. If yes, then stop the robot. Else keep moving forward. The complete program is shown below in Figure A.8.

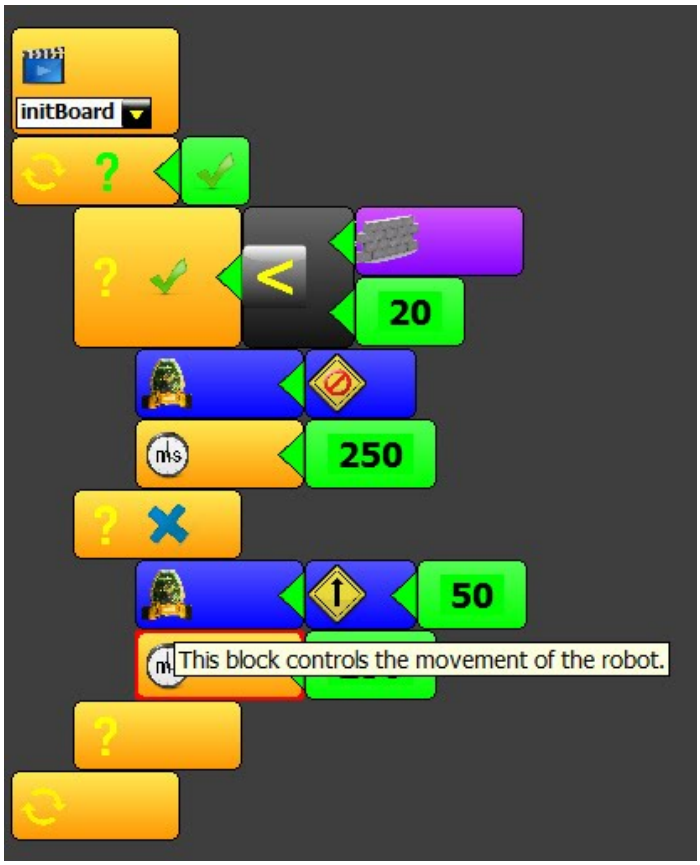

Figure A.8: Program to demonstrate obstacle sensor

A.9 IR Sensor

Goal: Make a floor tile black in color. When the robot enters the black tile, its speed decreases and when it comes out the tile, its speed returns back to normal.

IR sensors use Infra Red (IR) rays to emit and detect the amount of IR light that returns. Its usually used to detect between light and dark surfaces as light colored surfaces reflect more IR light than dark colored surfaces.

The IR sensor block is situated in the number window as it returns a number. Check if each sensor block is returning a value less than 50. That means its passing through a white tile. For white tiles, maintain a speed of 60 for the robot. If the value is greater than 50, its passing through a black tile. For black tiles, reduce the speed of the robot to 30. Complete program is shown below in Figure A.9.

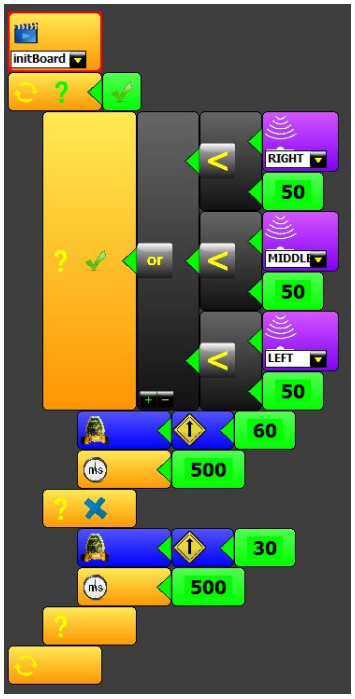

Figure A.9: Program to demonstrate IR sensor

A.10 Buzzer

Goal: To play the notes "c-d-e-f-g-a-b-c".

This lesson will help you use the buzzer to make musical notes. Before you start, make sure that the buzzer is activated with knob for the buzzer is in the right position.

Click on the "Buzzer" block in the action bar. he block takes three parameters -

1. Note - Also referred as the pitch of the sound. This sets the frequency of the sound. By clicking on the red triangle button on the right of the block, user can select from 8 available frequencies (c, d, e, f, g, a, b, C). These are similar to the notes available in music.

2. Beat - This parameter sets the basic unit of time. For example, a note with beat "2" plays for a longer time than a note with a beat "1". Click on the red triangle button to get a number window pop-up. Select hash symbol to give a constant number 1 as input.

3. Tempo - This sets the speed or pace of a given piece of music. So, if its a piece of music, its ideal to have the same tempo for all the notes. For the current task, set the value to a constant number "300".

Repeat the same steps for the notes d-e-f-g-a-b-C. Complete program is shown below in Figure A.10.

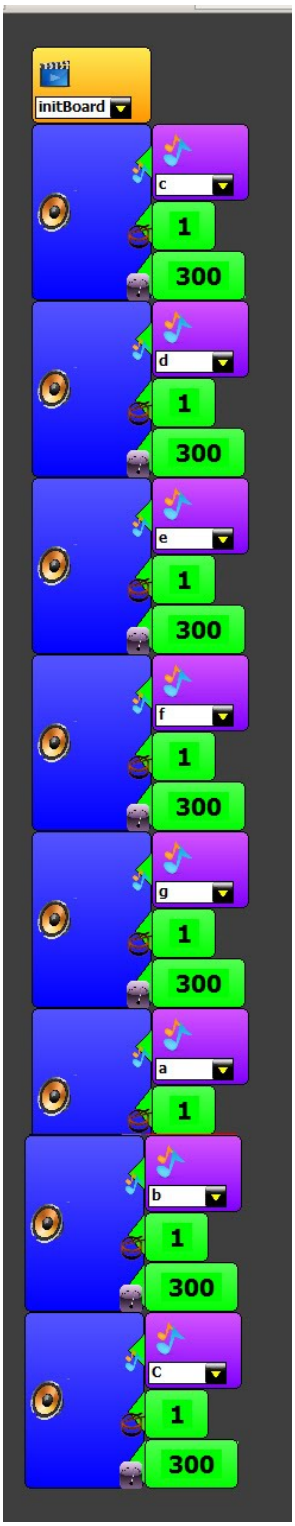

Figure A.10: Program to demonstrate Buzzer

Appendix B

Arduino sketch program

Listing B.1: Arduino program to move the robot forward

```
int pwm_a = 3; //PWM control for motor outputs 1 and 2 is on digital pin 3
int pwm_b = 6; //PWM control for motor outputs 3 and 4 is on digital pin 11
int dir_a1 = 4;
int dir_a2 = 2;
int dir_b1 = 5;
int dir_b2 = 7;

void setup(){
  Serial.begin(9600);
  pinMode(pwm_a, OUTPUT); //Set control pins to be outputs
  pinMode(pwm_b, OUTPUT);
  pinMode(dir_a1, OUTPUT); //set motor direction output
  pinMode(dir_a2, OUTPUT);
```

```
  pinMode(dir_b1, OUTPUT);
  pinMode(dir_b2, OUTPUT);
}

void loop(){
    forward();
    delay(2000);
}

void forward() {
  digitalWrite(dir_a1, LOW); //Reverse both motors direction, 1 high
  digitalWrite(dir_a2, HIGH);
  digitalWrite(dir_b1, LOW); //Reverse both motors direction, 2 high
  digitalWrite(dir_b2, HIGH);
  analogWrite(pwm_a, 255); //set motor A to run at 100% duty cycle (fast)
  analogWrite(pwm_b, 255); //set motor B to run at 100% duty cycle (fast)
}
```