Capstone Project

Voice-Controlled Mobile Robotic Arm

Lab Team:	**Tianshu Bao**
	Shawn Stern
	Dan Tran
Lab Section:	**EE 478**
Date:	**June 11, 2013**

TABLE OF CONTENTS

1 TABLE OF FIGURES

2 GROUP MEMBER CONTRIBUTIONS AND TIME ESTIMATE

Tianshu Bao:

- Bluetooth serial interface between Arduino and Android
- Android application and hyper-threading for robot control
- Arduino code to drive motors and arm
- Motor assembly and chassis fabrication
- Java and Arduino code debug

Shawn Stern:

- Arduino code to control distance sensor, SPI library for digital potentiometer control, and LED driver. Contributions to motor and arm control code.
- Artwork design and editing for interface
- System wiring and breadboard layout for all components
- Power system
- Arduino code debug

Dan Tran:

- Assembly of arm and servos
- Mounting of distance sensor, battery packs, and Arduino onto chassis

ESTIMATE OF HOURS SPENT

Design and coding: 70 man-hours

Breadboard and chassis assembly: 30 man-hours

Testing and debug: 60 man-hours

Documentation: 20 man-hours

3 ABSTRACT

The system comprises a Bluetooth-connected, Android application driven robot with the ability to move around its environment with a tank-style chassis, and grab items in the environment with a small arm assembly mounted on its rear. The system can detect objects in its forward path and to its immediate left and right, to avoid collisions when being operated manually. In addition, the system can navigate around simple obstacles with no direct input from the user. Finally, the system also supports an array of voice commands in lieu of physical button presses. Robot direction is indicated both on the user interface and visually using colored LEDs, and the output of the distance sensor is proportional to the brightness of an additional range indication LED on the robot chassis. While the intended variable-speed drive system could not be successfully implemented, the hardware and software was adapted to work with the distance sensor LED to provide variable brightness instead. All systems have been tested and verified for proper operation, and the robot meets all design goals.

4 INTRODUCTION

The intent of this capstone project was to be a culmination of all skills gathered in the course of previous classes such as EE 472, 471, and 271, in addition to any other non-embedded classes the group members may have been exposed to in the course of their studies. The system was to be developed over a short timeframe of approximately four weeks, with the system being an original idea created by the group members and approved by the instructor. The project was to be sufficiently complex, both in terms of its software design and the necessary hardware systems that drive it. This served as an excellent opportunity for the group members to test their engineering skills and design experience prior to entering their careers in electrical engineering.

For this project, the group elected to design a robotic arm that would be mounted on a mobile chassis and controllable by means of an Android application and voice commands over Bluetooth. In order to make the project even more complex, an array of additional features were added such as object and collision avoidance systems, robot status annunciation systems, and a system for autonomous control of the robot as opposed to a purely manual control scheme similar to a RC car.

Development for the Android application was done using the Java language and the Android Development SDK provided by Google. Voice recognition was accomplished by researching the Google API and developing code for the application to send data to Google servers for translation into strings, which could then be parsed by the app to control the robot. The hardware itself on the robot is all driven using the Arduino Uno, a microcontroller based on the 8-bit ATmega328 by Amtel. Using Arduino's proprietary language which has a mix of Java and C elements, code was developed to control the distance sensor, digital potentiometer, motors, servos, LEDs, and Bluetooth module that are necessary for the robot to function.

At each stage of development, the system components were tested heavily for reliability and functionality. This applies both to the Android application and the array of hardware controlled by the Arduino, and resulted in a solid, error-free system that is robust and performs as designed.

5 REQUIREMENT SPECIFICATION

5.1 System Description

This specification describes a user-controlled, multi-wheeled robot with a rotating arm for grabbing nearby objects. The arm will have 135 degrees of freedom horizontally, and up to 120 degrees of freedom vertically. The robot is to be able to move at reasonable speeds of up to 0.5m/s through its environment, with the ability to quickly turn in any direction, and to move forward and backwards in addition to braking. The robot is to be able to detect immediate hazards and obstacles in its forward path up to 3 meters away and halt, and provide feedback to the user informing them of the event. The robot is to be able to grasp small, light objects on the ground or in the robot's immediate vicinity for relocation. All robot functions are to be controllable using an Android phone application with controls for the motion of the robot, in addition to controls for the arm mechanism itself. The robot is to also be able to accept voice commands as a substitute for direct, physical interaction with the android interface. In addition, the robot is to support autonomous navigation from its starting location to a point determined by the user. The robot is to be low cost, flexible, easily reproducible, and intuitive to use.

5.2 Specification of External Environment

The system is to operate in an environment such as a home, ideally with minimal clutter on the floor surfaces on which it will be operating. Room temperature and typical in-house lighting are the expected environment.

The unit will run off of a 9V battery pack attached to the robot chassis for easy swapping in the event of a dead battery. Uptime will be up to 5 hours.

5.3 System Input and Output Specification

5.3.1 System Inputs

The system will accept the following external inputs:

- Analog pulse from the distance sensor, 5V with a pulse width of 0uS to 38ms

The system will accept the following commands by means of button presses on the Android application screen:

• Move forward	(move continuously while button is pressed)
• Move backward	(move continuously while button is pressed)
• Turn right	(move continuously while button is pressed)
• Turn left	(move continuously while button is pressed)
• Brake	(Halts ALL robot movement)
• Rotate arm left	(move continuously while button is pressed)
• Rotate arm right	(move continuously while button is pressed)
• Rotate arm up	(move continuously while button is pressed)
• Rotate arm down	(move continuously while button is pressed)

- Grasp/Release (toggle button)
- Receive Voice Command

The following voice commands will be usable after pressing the “Receive Voice Command” button on the application:

- Go forward (robot moves until told otherwise)
- Go back (robot moves until told otherwise)
- Turn left (robot moves until told otherwise)
- Turn right (robot moves until told otherwise)
- Brake (robot halts ALL motion)
- Turn left/right ‘x’ degrees
- Self Control mode
- Grasp
- Release

5.3.2 System Outputs

The system will report the following information on the user interface via serial interface over Bluetooth connection:

- Current direction of travel (forward, back, turning, braked)
- Arm status (rising, lowering, grasped, open)
- Distance to nearest large object in front of the robot in cm
    - Range from 0 to 4000cm
- DC 0v or DC 5V to drive motors
- The following PWM signals:
    - 5V signal to claw servo, open/closed
    - 5V signal to arm elevation servo, various degrees of being lowered or elevated
    - 5V signal to arm rotation servo, range of rotation between left and right of center
    - 5V signal to distance sensor mount servo, left, right, and default forward state.

5.4 User Interface

The user will be able to continuously control the robot and query its sensors for data using a simple android touch interface or vocal commands. The following buttons are available:

Robot Direction:

- Directional arrows. Robot moves (relative to its current orientation) in accordance with the held arrow until it is released.
- Brake. Robot halts all motion and ceases any autonomous functions.

Arm Control:

- Directional arrows to control the orientation and elevation of the grasper at the end of the arm. Arm moves as long as the appropriate button is depressed or until the arm reaches the limits of its rotation. Robot will only be able to move one discrete direction at a time.
- Grasp/Release toggle. Pressing the button causes the grasper to switch to the opposite state (from grasping to releasing the object, and vice versa)

Activate Voice Control:

- Press button to launch voice prompt to interpret voice commands.

Measurements and robot status information will be displayed in the top left corner of the android interface as text through a serial interface over Bluetooth.

The android interface will appear as follows: upon loading the application, the user will be presented with a splash screen while the Bluetooth pairing and initialization of the app occur. The application interface itself will have an upper set of buttons for directional control. The silver button with a microphone on it, when pressed, will prompt for a voice command. The red button emblazoned with a brake pedal, positioned between the sets of controls and off to the right for ergonomic reasons, serves as the 'Brake' command that will halt all robot movement until a new input is given to the system. The lower set of buttons will operate the arm assembly, with the center button serving as the grasp/release toggle.

Figure 1 - User App Splash Screen

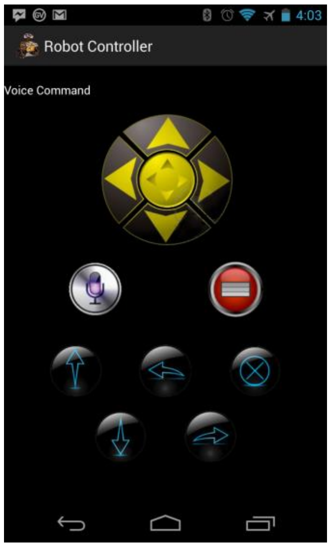


Figure 2 - User App Interface

The field where "Voice command" is listed in the location where textual status information will be displayed.

5.5 Use Cases

The cases are given in the below diagram:

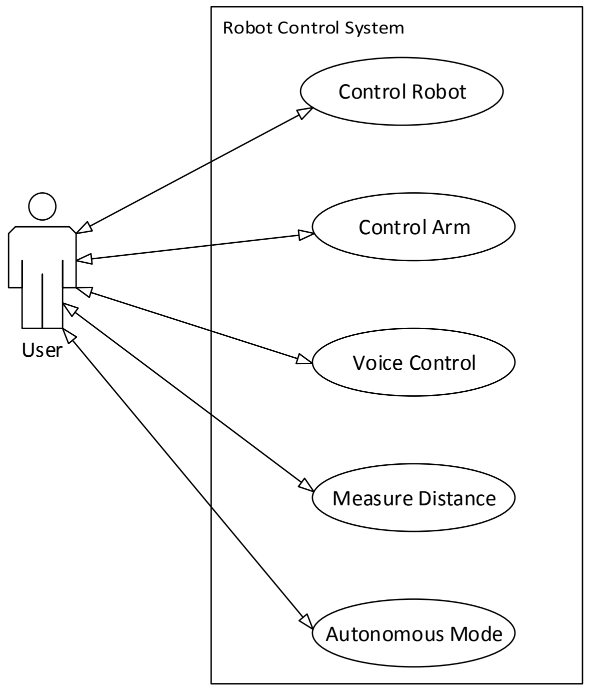


Figure 3 - Use Case Diagram

Control Robot

The robot chassis will move in accordance with user input. If the Forward button is held down, the robot will continue to move forward in a straight line until the button is released or if an object is detected as too close (robot will automatically brake and the only available movements will be rotation or moving backwards). This system is controlled by the upper set of arrows in the top half of the android interface. In addition, speed control is an option to adjust how quickly the robot chassis may move.

Exceptions: If terrain is too cluttered, the robot may not be able to maintain true, straight motion as its path will be disrupted by debris. If Bluetooth connection is lost, error will occur.

Control Arm

The arm assembly will move in accordance with user input. If the user holds the Down button, the grasping end of the arm will lower itself closer to the ground until either the user releases the button, or if the grasping end reaches the limit of its lowering range. The grasping end of the arm can be commanded to either close its grip or release it. This system is controlled by the lower set of arrows and central button in the bottom half of the android interface.

Exceptions: If the object is too heavy or too large, the arm assembly is not guaranteed to be able to properly grasp the object, or to be able to lift it without potentially destabilizing the robot. If Bluetooth connection is lost, error will occur.

Voice Control

The user interface will support the conversion from spoken words in English to their respective commands as needed to control the robot and/or arm assembly as appropriate. For example, the user may instruct the arm to “Arm up 20 degrees” at which point the robot arm will rise such that the difference from its previous position and new position is 20 degrees, within tolerances. The mode may be activated by pressing the speaker icon in the center of the user display.

Exceptions: If user does not speak clearly with proper enunciation, their statement may be misinterpreted. If Bluetooth connection is lost, error will occur.

Measure Distance

The robot will return the distance from the front section of the robot to the nearest object in front of it that is at least .5sq ft in dimension. The button for this function is still under design consideration.

Exceptions: If an object is smaller than .5sq ft. there may be an error in detecting the object correctly. If Bluetooth connection is lost, error will occur.

Autonomous Mode

The robot will attempt to return to the user, avoiding obstacles by judging the least-obstructed path and turning in that direction before continuing forward. User can stop the robot once it is in range of the user for object retrieval.

Exceptions: If the path is blocked with debris that blocks the path of the sensor, error may occur in position tracking. System is heavily dependent on the user to specify the correct destination as the robot lacks means of identifying location other than distance and orientation.

5.6 System Functional Specification

The system is intended to operate and move around in the environment based on user input, with the input from the distance sensor as a safeguard from collisions. A measurement system will be implemented to convert the output pulse of the distance sensor into a user-accessible distance value, selectable as either inches or cm. This distance value will also be used directly by the robot for safety reasons to halt robot movement if it is in danger of a head-on collision.

The user will be able to select either continuous motion of the robot (moves until stopped) or motion that only lasts as long as the respective button is being held down.

5.7 Operating Specifications

The system shall operate in a typical household environment:

Temperature Range 40 – 120F

Humidity up to 90%

Power 9V battery

The system shall operate for a minimum of 5 hours on a fully charged 9V battery

5.8 Reliability and Safety Specification

MTBF: Minimum 2,500 hours

The system will follow the safety standards for household electronics outlined by the standard UL 22.

6 DESIGN SPECIFICATION

6.1 System Description

This specification describes a user-controlled, tank-treaded robot with a rotating arm for grabbing nearby objects. The arm will have 135 degrees of freedom horizontally, and up to 120 degrees of freedom vertically. The robot is to be able to move at reasonable speeds of up to 0.5m/s through its environment, with the ability to turn in any direction, and to move forward and backwards in addition to braking. The robot is to be able to detect immediate hazards and obstacles in its forward path up to 3 meters away and halt, and provide feedback to the user informing them of the event. The robot is to be able to grasp small, light objects on the ground or in the robot's immediate vicinity for relocation. All robot functions are to be controllable using an Android phone application with controls for the motion of the robot, in addition to controls for the arm mechanism itself. The robot is to also be able to accept voice commands as a substitute for direct, physical interaction with the android interface. In addition, the robot is to support autonomous navigation from its starting location to a point determined by the user. The robot is to be low cost, flexible, easily reproducible, and intuitive to use.

6.2 Specification of External Environment

The system is to operate in an environment such as a home, ideally with minimal clutter on the floor surfaces on which it will be operating. Room temperature and typical in-house lighting are the expected environment.

The unit will run off of a 9V battery pack attached to the robot chassis for easy swapping in the event of a dead battery. Uptime will be up to 5 hours.

6.3 System Input and Output Specification

6.3.1 System Inputs

The system will accept the following external inputs:

- Analog pulse from the distance sensor, 5V with a pulse width of 0uS to 38ms

The system will accept the following commands by means of button presses on the Android application screen. Each command is sent over Bluetooth on the 9600 baud serial connection as a specific character or string specifying an action for the Arduino to take:

• "1", Move forward	(move continuously while button is pressed)
• "0", Move backward	(move continuously while button is pressed)
• "3", Turn right	(move continuously while button is pressed)
• "2", Turn left	(move continuously while button is pressed)
• "4", Brake	(Halts ALL robot movement)
• "B", Rotate arm left	(move continuously while button is pressed)
• "C", Rotate arm right	(move continuously while button is pressed)
• "A", Rotate arm up	(move continuously while button is pressed)

- “9”, Rotate arm down (move continuously while button is pressed)
- “8”, Grasp/Release (toggle button)
- Receive Voice Command (activates API to convert speech to text, which is then converted to one of the following serial commands)

The following voice commands will be usable after pressing the “Receive Voice Command” button on the application:

- “1”, Go forward (robot moves until told otherwise)
- “0”, Go back (robot moves until told otherwise)
- “2”, Turn left (robot moves until told otherwise)
- “3”, Turn right (robot moves until told otherwise)
- “4”, Brake (robot halts ALL motion)
- “E, F or H, I”, Turn left/right ‘x’ degrees (Sent repeatedly until motion complete)
- “7”, Self Control mode
- “8”, Grasp/Release

6.3.2 System Outputs

The system will report the following information on the user interface via 9600 baud serial interface over Bluetooth connection:

- Current direction of travel as a string (“forward”, “back”, “turning”, “braked”)
- Arm status as a string (“rising”, “lowering”, “grasped”, “open”)
- Distance to nearest large object in front of the robot in cm
    - Range from 0 to 4000 ± 1cm
- DC 0V or DC 5V to drive motors ± 0.1V
- The following PWM signals:
    - 5V signal to claw servo, duty cycle of either 30% or 80% for open/closed
    - 5V signal to arm elevation servo, between 10% and 60% for various degrees of being lowered or elevated
    - 5V signal to arm rotation servo, between 30% and 60% for range of rotation between left and right of center
    - 5V signal to distance sensor mount servo, 5% for left, 95% for right, and 50% for default forward state.
    - All PWM signals have step size of 1%

6.4 User Interface

The user will be able to continuously control the robot and query its sensors for data using a simple android touch interface or vocal commands. The following buttons are available:

Robot Direction:

- Directional arrows. Robot moves (relative to its current orientation) in accordance with the held arrow until it is released.

- Brake. Robot halts all motion and ceases any autonomous functions.

Arm Control:

- Directional arrows to control the orientation and elevation of the grasper at the end of the arm. Arm moves as long as the appropriate button is depressed or until the arm reaches the limits of its rotation. Robot will only be able to move one discrete direction at a time.
- Grasp/Release toggle. Pressing the button causes the grasper to switch to the opposite state (from grasping to releasing the object, and vice versa)

Activate Voice Control:

- Press button to launch voice prompt to interpret voice commands.

Measurements and robot status information will be displayed in the top left corner of the android interface as text through a serial interface over Bluetooth.

The android interface will appear as follows: upon loading the application, the user will be presented with a splash screen while the Bluetooth pairing and initialization of the app occur. The application interface itself will have an upper set of buttons for directional control. The silver button with a microphone on it, when pressed, will prompt for a voice command. The red button emblazoned with a brake pedal, positioned between the sets of controls and off to the right for ergonomic reasons, serves as the ‘Brake’ command that will halt all robot movement until a new input is given to the system. The lower set of buttons will operate the arm assembly, with the center button serving as the grasp/release toggle. Refer to **Figure 1** and **Figure 2** for the respective diagrams of these interface elements.

The field where “Voice command” is listed in the location where textual status information will be displayed.

6.5 Use Cases

Refer to **Figure 3** for the Use Case Diagram.

Control Robot

The robot chassis will move in accordance with user input. If the Forward button is held down, the robot will continue to move forward in a straight line until the button is released or if an object is detected as too close (robot will automatically brake and the only available movements will be rotation or moving backwards). This system is controlled by the upper set of arrows in the top half of the android interface. In addition, speed control is an option to adjust how quickly the robot chassis may move.

Exceptions: If terrain is too cluttered, the robot may not be able to maintain true, straight motion as its path will be disrupted by debris. If Bluetooth connection is lost, error will occur.

Control Arm

The arm assembly will move in accordance with user input. If the user holds the Down button, the grasping end of the arm will lower itself closer to the ground until either the user releases the button, or if the grasping end reaches the limit of its lowering range. The grasping end of the arm can be commanded to either close its grip or release it. This system is controlled by the lower set of arrows and central button in the bottom half of the android interface.

Exceptions: If the object is too heavy or too large, the arm assembly is not guaranteed to be able to properly grasp the object, or to be able to lift it without potentially destabilizing the robot. If Bluetooth connection is lost, error will occur.

Voice Control

The user interface will support the conversion from spoken words in English to their respective commands as needed to control the robot and/or arm assembly as appropriate. For example, the user may instruct the arm to "Arm up 20 degrees" at which point the robot arm will rise such that the difference from its previous position and new position is 20 degrees, within tolerances. The mode may be activated by pressing the speaker icon in the center of the user display.

Exceptions: If user does not speak clearly with proper enunciation, their statement may be misinterpreted. If Bluetooth connection is lost, error will occur.

Measure Distance

The robot will return the distance from the front section of the robot to the nearest object in front of it that is at least .5sq ft in dimension. The button for this function is still under design consideration.

Exceptions: If an object is smaller than .5sq ft. there may be an error in detecting the object correctly. If Bluetooth connection is lost, error will occur.

Autonomous Mode

The robot will attempt to return to the user, avoiding obstacles by judging the least-obstructed path and turning in that direction before continuing forward. User can stop the robot once it is in range of the user for object retrieval.

Exceptions: If the path is blocked with debris that blocks the path of the sensor, error may occur in position tracking. System is heavily dependent on the user to specify the correct destination as the robot lacks means of identifying location other than distance and orientation.

6.6 System Functional Specification

The system is intended to operate and move around in the environment based on user input, with the input from the distance sensor as a safeguard from collisions. A measurement system will be implemented to convert the output pulse of the distance sensor into a user-accessible distance value, selectable as either inches or cm. This distance value will also be used directly by the robot for safety reasons to halt robot movement if it is in danger of a head-on collision.

The user will be able to select either continuous motion of the robot (moves until stopped) or motion that only lasts as long as the respective button is being held down.

The system comprises of several major blocks, as given in the following system block diagram.

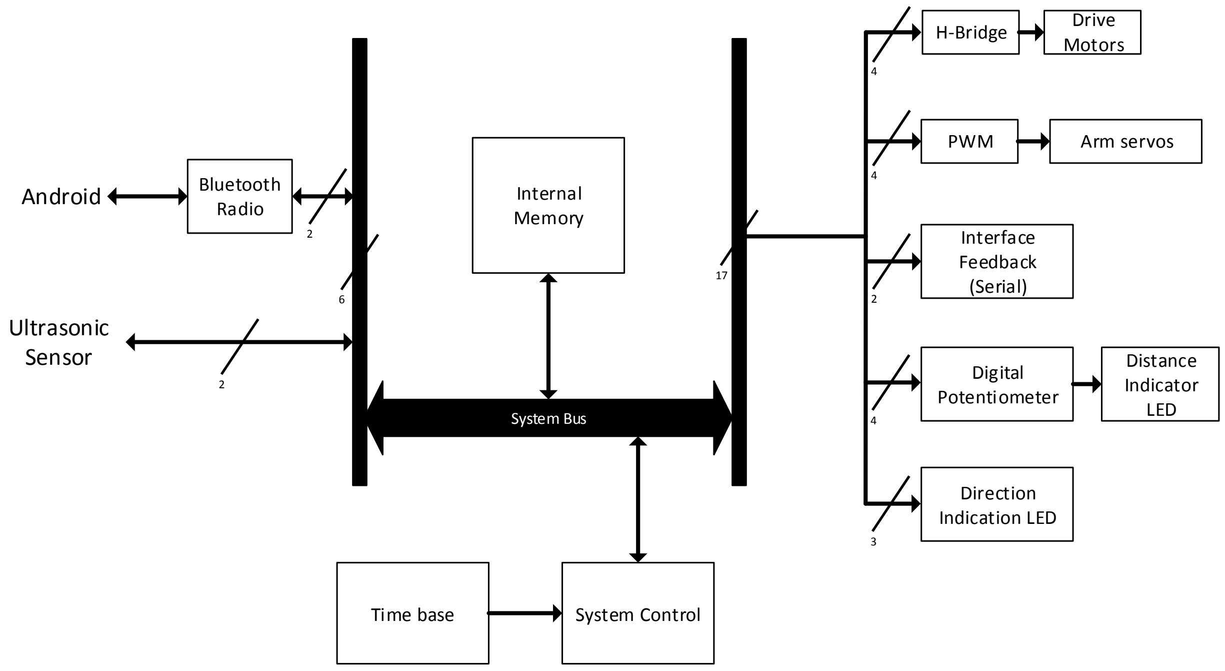


Figure 4 – System/Hardware Block Diagram

Bluetooth Subsystem – the Bluetooth subsystem facilitates communication between the Android user interface and the Arduino. The system controls serial communication between the two systems as necessary to allow system control and user interaction with the system.

Ultrasonic Sensor – upon being sent a minimum 10μs pulse, the sensor will send a burst of sound waves out and record their return times as a pulse whose length is proportional to the distance traveled. Because of the measurement being done via pulse length as opposed to a voltage level, an A/D converter is not required.

Time Base – the time base in this case is the internal, 16MHz oscillator on the Arduino Uno board.

Digital Potentiometer – using SPI, a digital potentiometer and current control circuit determine how much power is used to drive an LED. The LED grows brighter the closer the robot is to an obstacle, and dimmer as it moves away.

Drive and Arm Motors – The arm servos are the simple motors that will be controlled using PWM signals based on user input. They operate on 2.7 to 5V DC. The drive motors are controlled by DC signals fed into an H-Bridge.

Interface Feedback – this is a system that controls what output is sent from the Arduino to the user interface on the Android device, such as status messages about robot movement, arm movement, and object detection.

The activities required to perform a distance measurement are as given in the following activities diagram.

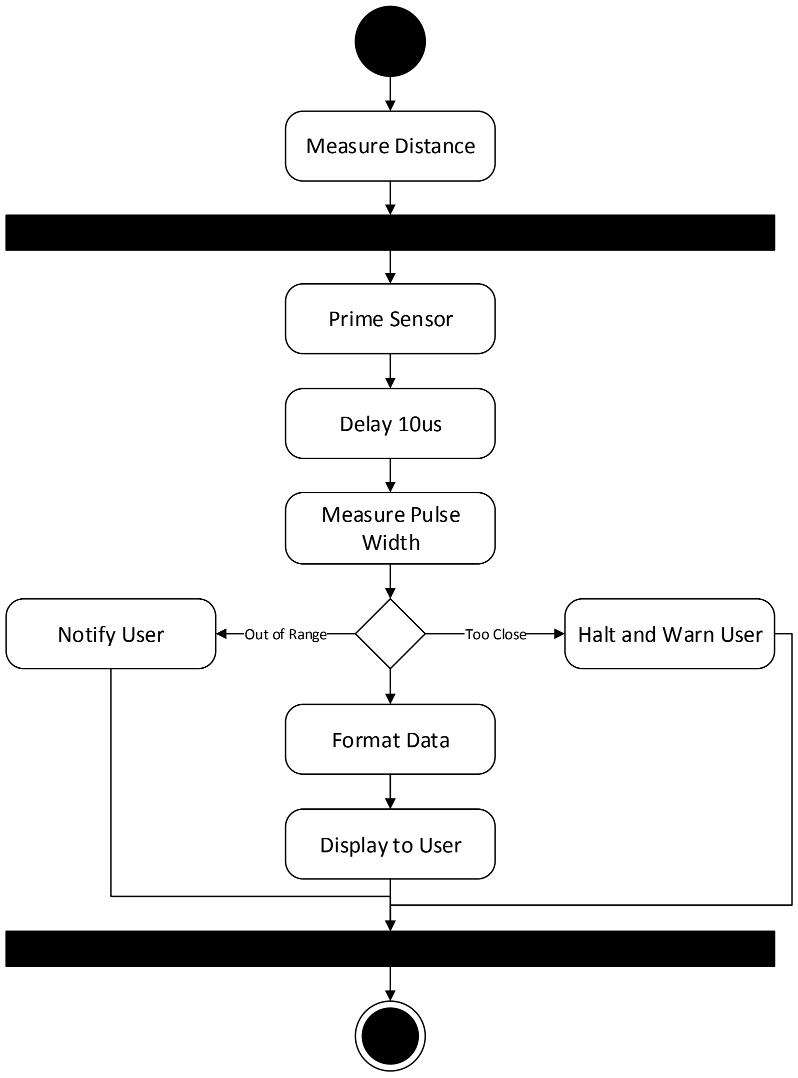


Figure 5 - Activity Diagram for Distance Measurement

6.7 Operating Specifications

The system shall operate in a typical household environment:

Temperature Range 40 – 120F

Humidity up to 90%

Power 9V battery

The system shall operate for a minimum of 5 hours on a fully charged 9V battery

6.8 Reliability and Safety Specification

MTBF: Minimum 2,500 hours

The system will follow the safety standards for household electronics outlined by the standard UL 22.

7 DESIGN PROCEDURE

7.1 Bluetooth and Android Application with Voice Commands

The communication between Android device and robot is established through a Bluetooth serial module. The Android app is designed to connect with specified Bluetooth via its mac address. The data transfer rate between Android device and Arduino is set to be 9600 bit/s based on the I/O port configuration of Arduino. On the Android application, the input data stream is handled with the Java Handler library, which creates strings from a bytes array when there is a received message from Arduino. The Arduino handles the data input and output stream with TX and RX ports which are connected to the Bluetooth serial module.

The voice command feature is accomplished by incorporating Google Voice Recognition with Android built-in libraries. The voice recognition activity requires an internet connection through either Wi-Fi or 3/4G (Wi-Fi is preferred to reduce latency and the time before a command is sent to the robot). The language that the voice recognizer can handle is restricted to English at this time due to the limitations of the Google API functions. There are sets of default voice commands hardcoded into the Android application, which will respond to user's voice command input for functions such as "go forward" or "brake."

7.2 Motor Control System

The motor control system is designed to control the rotation of the left and right motors in the double gearbox. In order to obtain different rotation directions based on analog signals from Arduino device, the SN754410 H-Bridge Motor Driver was used. The h-bridge takes in four analog signals from Arduino, which specifies the motor's rotation direction, and a power input that indicates the power applied to the motor. When both motors rotate the same direction, the robot moves either forward or backward, and when motors rotate the opposite direction, the robot turns left or right based on the order of rotation. A certain, small time delay is hardcoded into the motor control function which specifies the turning degree of the motor. Currently, the motor can precisely turn 30, 60, 90 and 180 degrees when the corresponding functions are executed, or simply rotate or move forward continuously while told to do so.

7.3 Arm Control System

The arm control system is to precisely control the motion of the robot arm in order to pick up certain objects. The arm parts are connected with servomotors at the joints so that it has 180 degrees of freedom both in the horizontal and vertical directions. The servomotors are controlled via the Arduino through the built-in Servo library, which contains functions to specify the degree and direction of the servomotor in a straightforward manner.

7.4 Distance Sensing System

The distance sensing system of the robot is to detect the distance from the front of the robot to the nearest obstacle in front of the sensor. The sensor is planned to use ultrasonic sound waves combined with a microphone to detect the time taken for a an emitted pulse to return to the speaker. The speed of sound is well known, so distance is calculated simply dividing the travel time by half (round trip there and back is double the true distance) and then by a factor to convert time in microseconds to a unit of distance such as centimeters.

In addition to this data being stored internally and used by the robot drive system, this data will also be relayed to the user by means of an illuminated LED that grows in intensity as the robot nears an object, and dims as it moves away. This will be controlled in by a digital potentiometer driven with SPI.

7.5 Robot Motion Annunciation System

The robot will visually announce its planned and current direction of travel using a tri-color LED. Because of the three LEDs embedded in the system, a total of 8 colors (red, green, blue, cyan, magenta, yellow, white, and off/black) can be displayed. When the robot is idle or in motion, a respective LED color is displayed to make those in the environment aware of the robot's operation and direction at a glance. Green corresponds to forward movement, red to backwards, cyan to right, yellow to left, white to system initialized, and black/off to idle/still. This system is driven using Arduino digital output pins.

7.6 Autonomous Mode

Autonomous (self-control) mode is programmed into the robot so that the robot can avoid obstacles and find its way out of an environment when there are difficulties for direct control via the Android device. During the autonomous mode, the robot uses the distance sensor to avoid collision with objects on its path. When the environment in front of the robot is blocked and the distance to the obstacle falls within a threshold, the robot will stop and compare the left and right distance from its current position. The robot will pick the direction with fewer obstacles and turn toward that direction before continuing forward. If both left and right are of the same distance, the robot will turn around completely and attempt another route.

8 SYSTEM DESCRIPTION

8.1 Bluetooth Connection

8.1.1 Inputs

- Input data stream from Android device and Arduino

8.1.2 Outputs

- Output data stream to Android device and Arduino

8.1.3 Data Transfer Constraints

The baud rate for the data transfer between Android and Arduino via Bluetooth is limited to 9600 bits/s, which is due to the input data stream limitations on the Arduino device.

8.1.4 Error Handling

If connection is lost with the Bluetooth module, the system will become unresponsive until the connection is reestablished.

8.2 Motor Control System

8.2.1 Inputs

- Analog signal from Arduino
- 5V power supply signal and ground

8.2.2 Outputs

- DC voltage to left motor and right motor, voltage is proportional to the input power supply from 3-5V

8.2.3 Side Effects

When the h-bridge is not well balanced, the voltage supply to the left and right motors might be slight different, which results in a different rotation speed between left and right motors, which cause the robot to move in a slightly curly path when it is supposed to go straight.

8.2.4 Pseudo English description of algorithms, functions, or procedures

```
Set up
    Set pin 2 and 4 to control the left motor
    Set pin 7 and 8 to control the right motor
Motor stop
```

```
        digitalWrite(motor_left[0], LOW)
        digitalWrite(motor_left[1], LOW)
        digitalWrite(motor_right[0], LOW)
        digitalWrite(motor_right[1], LOW)
        delay 25ms
Drive backward
        digitalWrite(motor_left[0], HIGH)
        digitalWrite(motor_left[1], LOW)
        digitalWrite(motor_right[0], HIGH)
        digitalWrite(motor_right[1], LOW)
Drive forward
        digitalWrite(motor_left[0], LOW)
        digitalWrite(motor_left[1], HIGH)
        digitalWrite(motor_right[0], LOW)
        digitalWrite(motor_right[1], HIGH)
Turn left
        digitalWrite(motor_left[0], LOW)
        digitalWrite(motor_left[1], HIGH)
        digitalWrite(motor_right[0], HIGH)
        digitalWrite(motor_right[1], LOW)
Turn right
        digitalWrite(motor_left[0], HIGH)
        digitalWrite(motor_left[1], LOW)
        digitalWrite(motor_right[0], LOW)
        digitalWrite(motor_right[1], HIGH)
Turn left 30 degrees
        turn_left()
        delay 500ms
        motor_stop()
Turn left 60 degrees
        turn_left()
        delay 700ms
        motor_stop()
```

```
Turn left 90 degrees
    turn_left()
    delay 1100ms
    motor_stop()
Turn 180 degrees
    turn_left()
    delay 2700ms
    motor_stop()
Turn right 30 degrees
    turn_right()
    delay 500ms
    motor_stop()
Turn right 60 degrees
    turn_right()
    delay 700ms
    motor_stop()
Turn right 90 degrees
    turn_right()
    delay 1100ms
    motor_stop()
```

8.2.5 Error handling

There are no observed errors associated with the motor control system, and if there were the robot would either need to be repaired or shut down to prevent random motion.

8.3 Arm Control System

8.3.1 Inputs

- 6V power supply signal and ground, provided by AA batteries

8.3.2 Outputs

- 4.5V TTL signal for each pin 3, 6, 9 on the Arduino

8.3.3 Side Effects

- This system has no potential for causing other problems in the systems directly

8.3.4 Pseudo English description of algorithms, functions, or procedures

```
Setup
    Set pin 3 to output for the shoulder
    Set pin 6 to output for the elbow
    Set pin 9 to output for the gripper
    Drive pin 3 to output 90 degrees
    Drive pin 6 to output 30 degrees
    Drive pin 9 to output 120 degrees
System Loop
    Grasp – drive pin 9 to output 65 degrees
    Drop – drive pin 9 to output 120 degrees
    Arm turn – drive pin 6 to output from 10 to 170 degrees
    Arm rotate – drive pin 3 to output from 30 to 120 degrees
```

8.3.5 Timing Constraints

There are no critical timing constraints to meet for this subsystem.

8.3.6 Error Handling

If the object is too heavy or too large, either the servomotors will not provide strong enough output to lift the object, or the fragile mount for the arm has the potential to break.

8.4 Distance Sensing System

8.4.1 Inputs

- Trigger – 5V TTL signal, length of 10μs
- 5V power supply signal and ground

8.4.2 Outputs

- Echo – 5V TTL signal, length is proportional to measured distance from 150μs to 25ms, 38ms if no object detected.
- DC voltage to the blue LED ranging from 1.4V to 5V proportional to distance

8.4.3 Side Effects

- The pulseIn function used by Arduino will hang for 1 second waiting for an input if nothing is received. This will delay the entire system by 1 second, due to the nature of Arduino delays.

8.4.4 Pseudo English description of algorithms, functions, or procedures

```
Setup
    Set pin 18 to output
    Set pin 19 to input
    Instantiate MCP4131 w/ pin 10 as CS
System Loop
    Drive pin 18 low
    Drive pin 18 high for 10us
    Drive pin 18 low
    Distance = pulseIn(pin 19)
    Distance = Distance / 2 / 29
    Pot_value = Map Distance from 0 to 31
    Set the MCP4131 to Pot_Value over SPI
```

8.4.5 Timing Constraints

The trigger pulse to the sensor must be 10us or longer, otherwise the sensor will not fire the necessary sounds for a measurement.

8.4.6 Error Handling

If no distance is measured, the system will hang for 1 second while it waits for input, then continue as normal, saving distance as 0.

8.5 Robot Motion Annunciation System

8.5.1 Inputs

- 5V power supply and ground
- 3, 5V TTL signals one each corresponding to red, green, and blue

8.5.2 Outputs

- LED colors: red, green, blue, cyan, magenta, yellow, white, or off

8.5.3 Side Effects

- This system has no potential for causing other problems in the systems directly

8.5.4 Pseudo English description of algorithms, functions, or procedures

```
System loop
    If the robot is going forward
        Set pins low to display green
    If the robot is going backwards
        Set pins low to display red
    If the robot is going left
        Set pins low to display yellow
    If the robot is going right
        Set pins low to display cyan
    If the robot is still
        Set all pins high to disable the LED
    If the robot is moving autonomously
        Set all pins low to display white
```

8.5.5 Timing Constraints

There are no critical timing constraints to meet for this subsystem.

8.5.6 Error Handling

Any errors in this system will simply result in a particular color not being displayed, or a damaged LED, but otherwise no errors in the system itself that can be deemed as critical.

8.6 Autonomous Mode

8.6.1 Inputs

- Distance variable, internal to the Arduino

8.6.2 Outputs

- Motor control signals to drive the robot in the correct direction

8.6.3 Side effect

If the robot is trapped in a circular environment, its very difficult for it to find its way out due to the limitation of the distance that it can only detects the distance in front of the robot.

8.6.4 Pseudo English description of algorithms, functions, or procedures

```
Autonomous Mode
```

```
distanceFwd = distance from sensor
if (distanceFwd > dangerThresh)
    go forward
else
    stop
    compare left and right distance
    if (left distance > right distance)
        turn left
    else if (right distance > left distance)
        turn right
    else
        turn around
```

8.6.1 Timing Constraints

There are no critical timing constraints to meet for this subsystem.

8.6.2 Error Handling

There are no specific error handling countermeasures other than those in the code itself which attempts to account for the situation where the robot is stuck in a circle.

9 SOFTWARE IMPLEMENTATION

9.1 System Decomposition

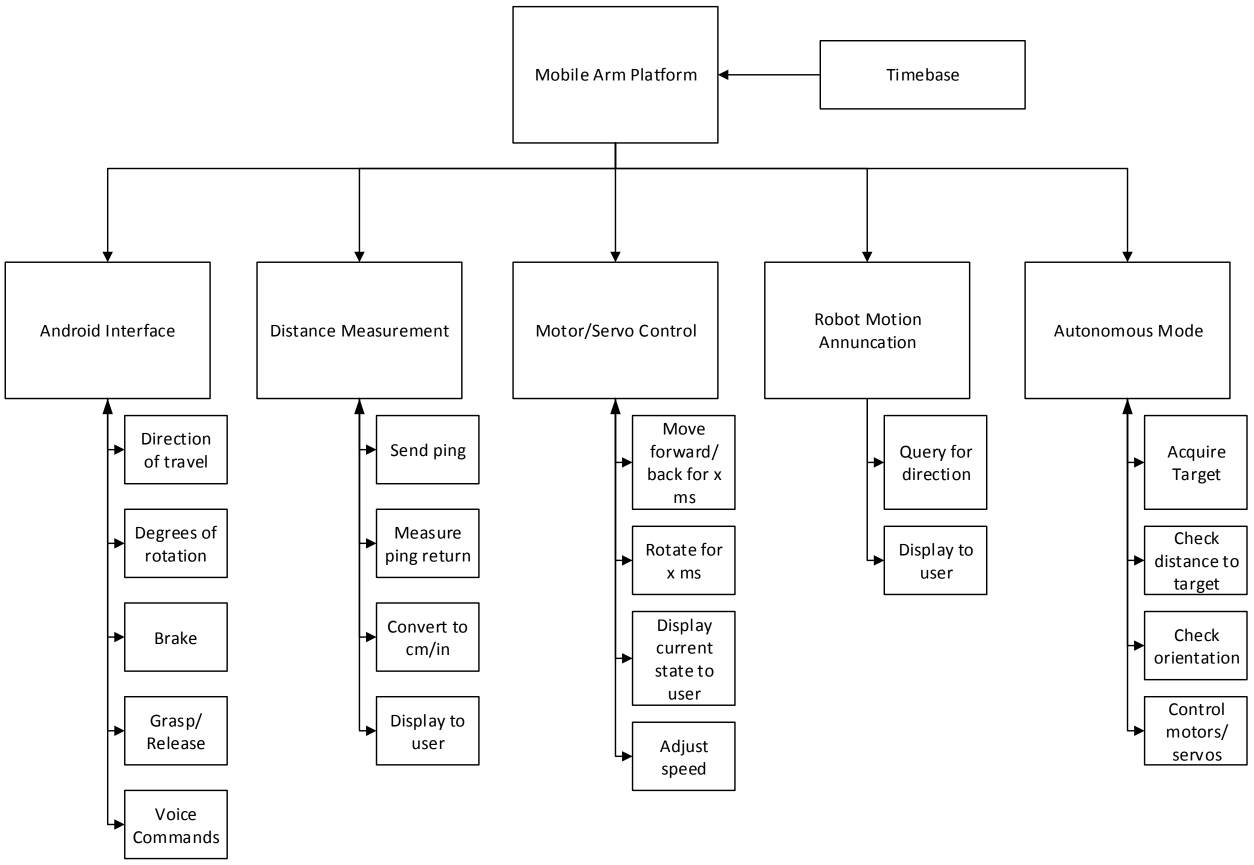


Figure 6 - System Functional Decomposition

The software of the system was developed based on the five primary software blocks shown above, and described in detail below.

9.1.1 Bluetooth Connection and Android Interface (with voice controls)

The software for robot control Android application is developed in Java ADT. The Bluetooth connection between Android device and Arduino is established at the onCreate and onResume stages in the Android application life cycle. In onCreate stage, a Bluetooth adapter is acquired and the Bluetooth interface on the Android device is checked (since the Bluetooth connection needs the Android device to have its Bluetooth interface turned on). If the Android device does not support Bluetooth interface, the program will throw a fatal error and exit. In the onResume stage, the software tries to set up a pointer to the remote device using its mac address, which is hardcoded into the function. After it successfully obtained a Bluetooth device, it creates a Bluetooth socket, which uses an insecure RF communication connection. Since the discovering of remote device is recourse intensive, the software forces to cancel the discovery and setup the Bluetooth connection. After the connection is established, the Android device can transfer data with the Arduino device.

The control buttons on the Android user interface are made with ImageViews and certain on-touch effect is assigned to each button. The direction control button contains a single onTouch function which response to user's click event. In order to have the correct response based on four directions, Java canvas library is being used to identify the correct region of a user click and the button is assigned to behave according the user's on-click region.

Voice command on the Android application is developed with Google's voice recognition library, and a voice command button is assigned to start the voice recognition activity when user chooses to input a voice command. The onActivityResult function for the voice command takes in user's voice input data, parse the data and stores it in an array. Then based on different input cases, the voice recognition activity calls its helper method to handle user input.

The data transfer between Android device and Arduino device is handled with a separate thread in the Android program, which protects the performance of the main activity, thus the interrupts generated with input and output data stream will not affect the Bluetooth connection and other parts of the main program.

9.1.2 Distance Measurement

The software for the Arduino control of the sensor was developed in accordance with the waveform diagrams for the sensor, as detailed in **Figure 18**. The digitalWrite function is used to first set the trigger pin low, then set high for a delay of 10 microseconds before being sent low again. The output of the sensor on the echo pin is the saved using the pulseIn function provided by the Arduino library, which returns the length of the measured pulse in microseconds, and delays for a full second if no signal is received. This value is saved into a long variable named distance, before being divided by 2 and 29 to convert it into centimeters.

Once the value is received, it is converted on a linear scale using the Arduino map function to convert it from a distance value to a value from 0 to 31 (0 being close distance, 31 far). This value is then sent over SPI to the digital potentiometer by first setting the CS pin low, transmitting an address of 0 to the pot, followed by the desired resistance step, and finally setting the CS pin back to high. All SPI functions for the digital pot were turned into an Arduino library for ease of use in the project, and also include useful increment and decrement functions. However, given that the motor control use of the pot was abandoned and replaced with this LED driving function that is better suited to direct resistance mapping, those functions were not used in this project. The functions can still be found in the libraries in the appendix, **section 17.5.8**.

9.1.3 Motor and Arm Control

The behavior of the motors is controlled by the Arduino. When both of the output analog signals to the left motor and right motor are low, the motor stops. When the both the analog signals to the positive side of the motor are high, the motor rotates forward and vice versa when both the analog signals are low. The turning of the robot is accomplished by having left motor and right motor rotates in different directions, with opposite analog control signals applied to the motors. When the user chooses to use voice command to control the robot, there are four, hardcoded degrees by which the robot can turn based on the user's voice input. The default is 90 degrees, which is done by keeping the robot turning for a certain period of time before stopping, in this case the delay is 1100ms. The user can also tell the robot to turn between 30 degrees, 60 degrees and 180 degrees, the time delays for which were calculated and tested for.

Three servomotors were used to facilitate arm control in this system. Using the Arduino Servo built in library code, the code begins by creating a Servo object for each servo to be controlled. The function servo.attach(3) was used to attach the servo on pin 3, and likewise for servo.attach(6) and servo.attach(9) to attach the servos on pins 6 and 9. The Arduino function servo.write(value) writes a value in to the servo, physically setting the angle of the shaft in degrees equal to the given value, thus orienting the shaft of the servo as desired.

The HS-55 Micro Servo can rotate up to 180 degrees by use of a pulse signal (PWM) ranging from 600µs to 2400 µs in period. Starting from 600µs for a rotation of 90 degrees left up to 2400µs for 90 degrees to the right, and with the neutral 0 degree position controlled by a pulse of period 1500µs as shown in **Figure 7**.

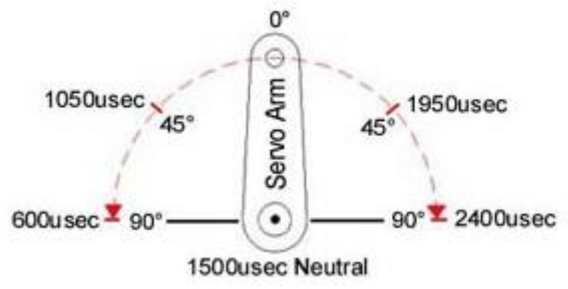


Figure 7 - Servo Motor Rotation

9.1.4 Robot Motion Annunciation System

In order to annunciate to the user what direction it is traveling via LEDs, a set of functions and variables were developed to make the process of lighting LEDs more intuitive when called in the system code. To accomplish this, since the LEDs are common anode and connected to the Arduino by the cathodes, the first step was to define ON for and LED as digital LOW, and OFF as digital HIGH. The next step was to define each color as an array of booleans for the colors necessary to be lit in order to display that color. For example, with the LED pins as red, green, and blue in that order (Pins 14, 15, and 16 on the Arduino, respectively), red is stored as {ON, OFF, OFF}, and a color such as cyan which is a mix of green and blue is stored as {OFF, ON, ON}. Lastly, for even more simplification, all of these colors are then stored in another array of colors, so that any color can be accessed only by knowing its index in that array. In this case, it was decided to store the colors as {RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, WHITE, BLACK} where in this case black is the state where the LED is inactive.

In order to set a color to the LED, two functions are used. The first setColor function accepts an integer specifying the color to be displayed, and the const boolean pointer to one of the colors in the large array of individual colors. This function retrieves the three ON or OFF booleans from the given color, then calls another version of setColor that accepts the same integer, and the new array of three booleans. It then uses a simple for loop to write the three digital values to the respective three pins.

This code to provide a particular color of LED is then called inside the case statements that control if the robot is moving forward, back, left, right, and so forth so that the appropriate color is displayed at the right time.

9.1.5 Autonomous Mode

In order for the robot to find its way out of an obstacle rich environment when difficult for the user to control the robot directly via mobile phone, an autonomous (self-control) mode is programmed with a simple artificial intelligence method. The self-control mode uses the distance sensor to avoid obstacles while it travels. When the environment in front of the robot is clear, the robot keeps on moving forward. The threshold distance before running into any obstacles is set to be 6cm from the distance sensor. When the distance in front of the robot falls within the threshold value, the robot stops and turns the distance sensor to both the left and right of the robot to acquire distance information. The robot then compares the left and right distance returned from the sensor, and turns to the direction that has fewer obstacles. If the returned distances are the same and both are within the threshold, the robot turns around and will try to find another way out of its predicament.

9.2 System Activity, Software Block, and Control/Data Flow Diagrams

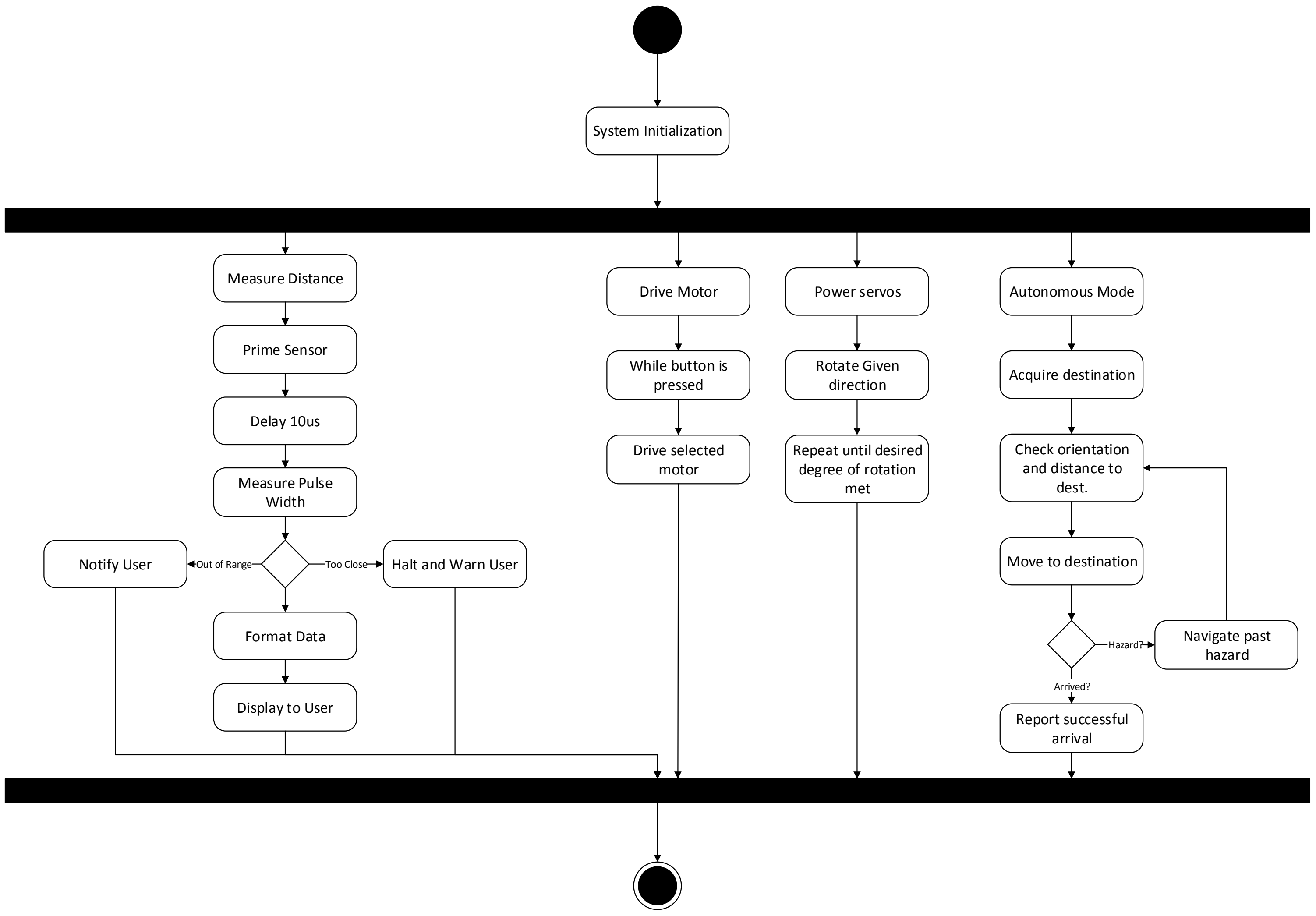


Figure 8 - System Activity Diagram

The diagram above, **Figure 8**, highlights the overall structure of the program running on the Arduino, which is constantly monitoring for input from the Android to tell it to either drive a motor, control the arm, or move autonomously. At the same time, it is constantly monitoring for distance to halt the robot if it is in danger of collision, and in the case of autonomous mode, distance polling is also used to check for the shortest path around an object, be it by turning left or right.

The software block diagram below in **Figure 9** shows this structure in a different format, more similar to the internals of the Arduino. The system initializes, and then constantly idles until given either a manual command for motion, or a command to move autonomously. Once that occurs, the robot then must parse the user input and determine whether to move itself, move the arm, or attempt to move autonomously. Once that action is complete, the system stores data about its current state and waits for the next command.

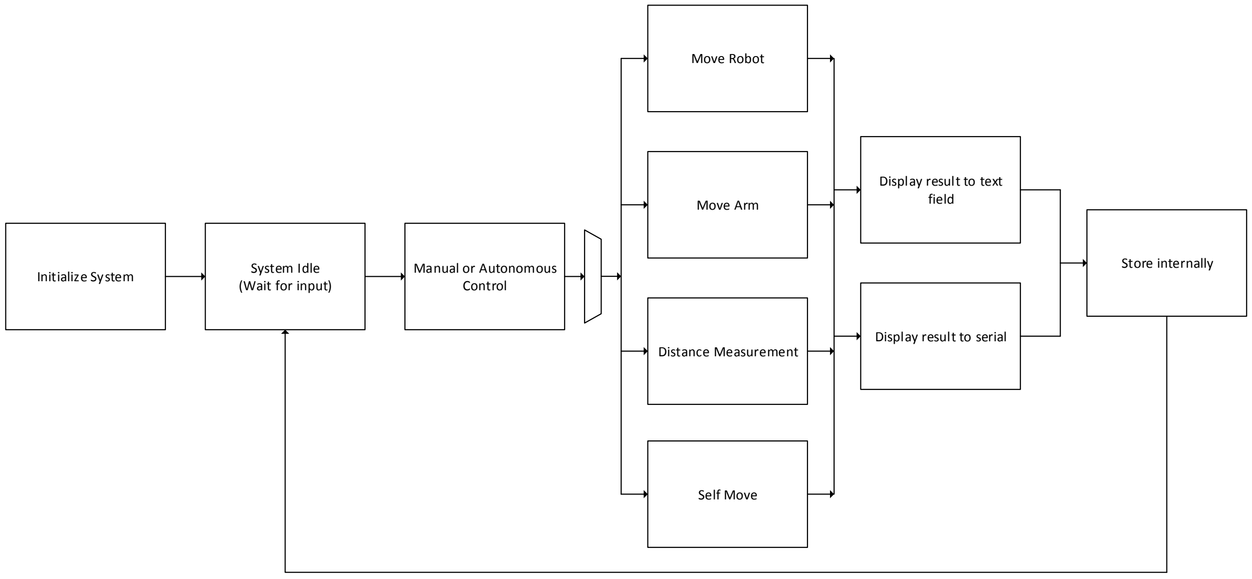


Figure 9 - Software Block Diagram

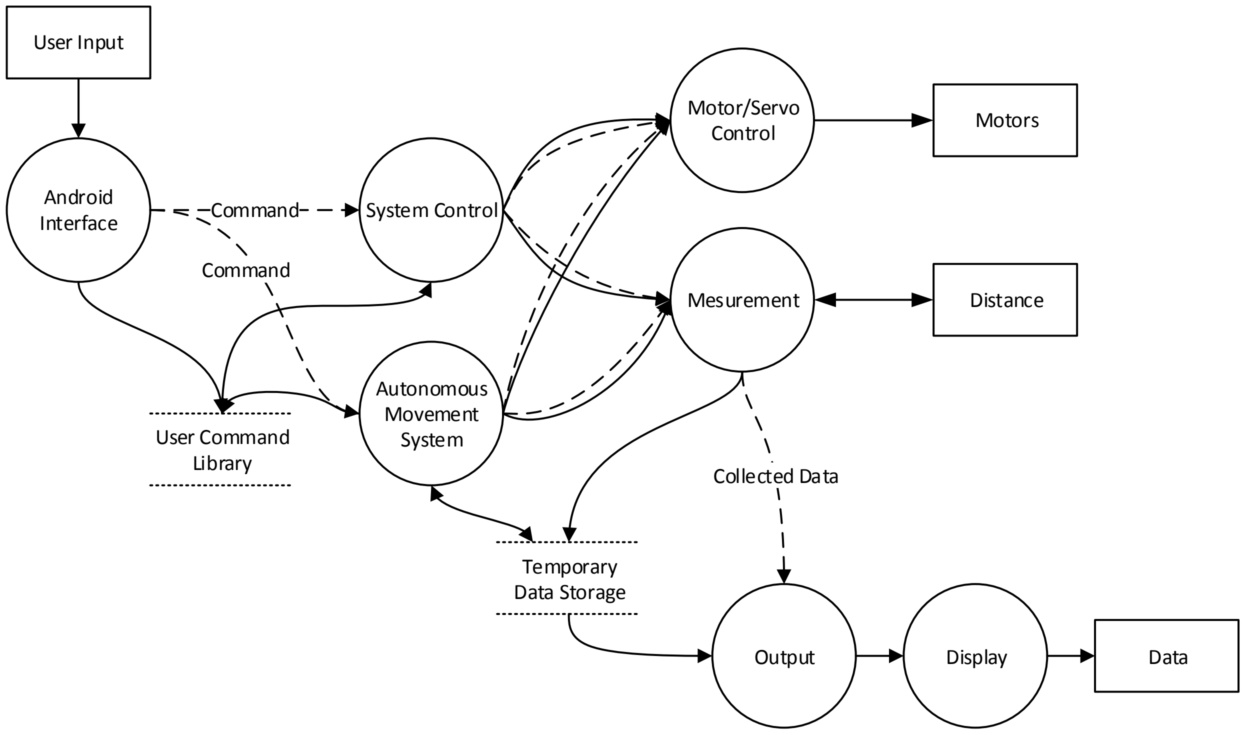


Figure 10 - Control and Data Flow Diagram

In **Figure 10** the control flow and data management of the system is modeled. User input is first interpreted by the android interface, converted by the application into a series of control bytes, and sent over Bluetooth to the serial port on the Arduino. At this point, the Arduino reads the command, interprets it, and translates it into either a direct control action that directly drives a motor/servo or the measurement system, or a command to move autonomously. Both cases control the motors, servos, and measurement systems, gather data from them, and store them temporarily for transmission back to the user. This data is also used to drive, for example, the display systems on the robot such as the motion LED, distance LED, and serial display on the user app interface.

10 HARDWARE IMPLEMENTATION

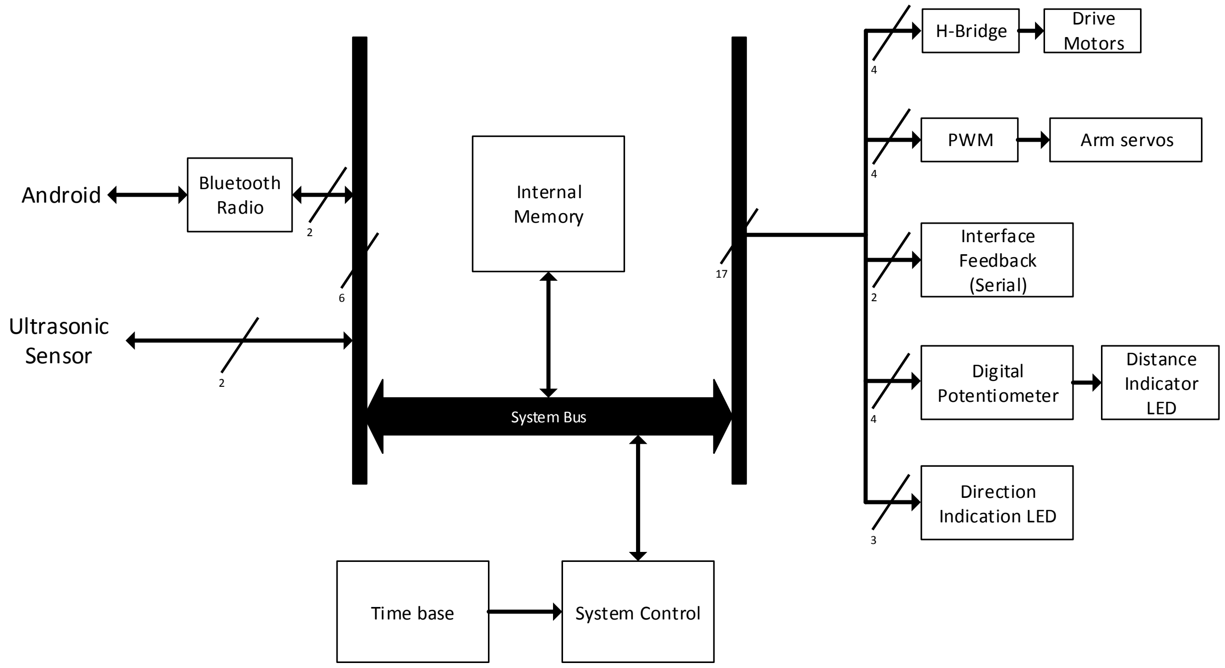


Figure 11 - System Hardware Block Diagram

10.1 Bluetooth

The Bluetooth is connected to Arduino device via two data transmission lines, TX and RX which are swapped between the RX and TX of the Arduino (pins 0 and 1) to enable serial communication. Communication takes place at 9600 baud, as described in the software implementation section. The wiring connection for the Bluetooth module is as shown below.

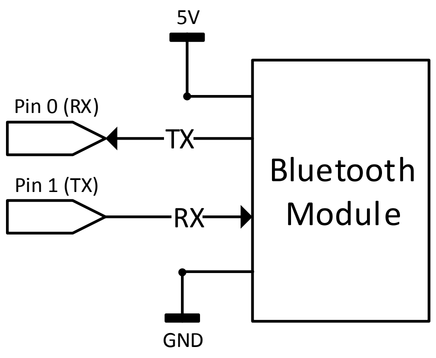


Figure 12 - Wiring Connections for Bluetooth Module

10.2 Motor Control

A H-Bridge is used to supply the control signals to the positive and negatives terminals of each motor. Connections are shown as in **Figure 13** below, and the code governing the actions of the circuit are described as in the software implementation section. There is no additional hardware to describe for this system, and the digital potentiometer had to be migrated to the distance sensing system.

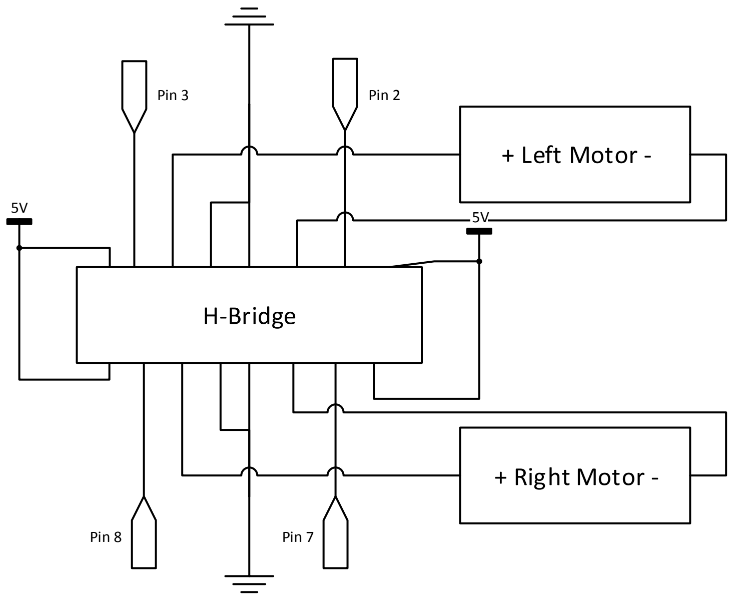


Figure 13 - Wiring Connections for the H-Bridge

10.3 Arm Control

Figure 14 - Hitec HS-55 Micro Servo Motor

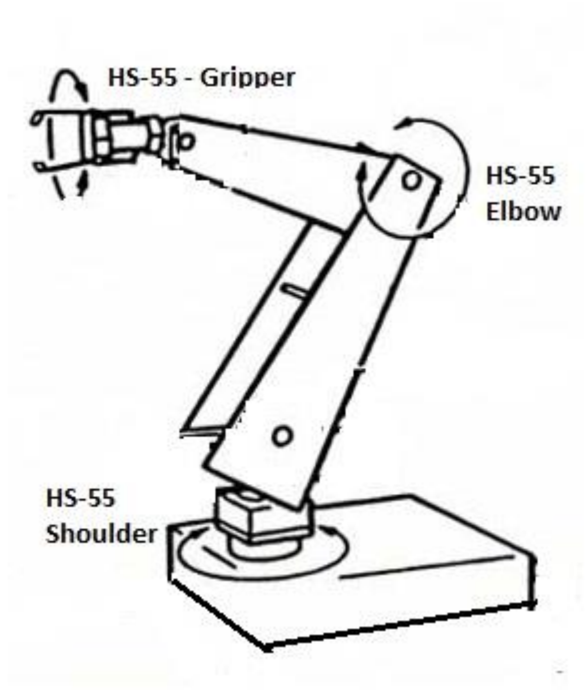


Figure 15 - Robot Arm Design

Figure 15 depicts the robot arm design in its final iteration. The HS-55 micro servomotor was selected to control the shoulder, elbow, and gripper components of the arm. The arm operates using three pins driven by the Arduino, with pin 3 for the shoulder servo, pin 6 for the elbow servo, and pin 9 for the gripper servo.

The servomotors each have three wires: power, ground, and signal. The power wire connects to the 4.5V output from the AA batteries, and the ground wire connect to the common ground between the Arduino and the batteries. The signal pin is connected to pin 3, 6, and 9 on the Arduino board respectively for each servo.

The diagram below in **Figure 16** demonstrates the hookup of a single servomotor.

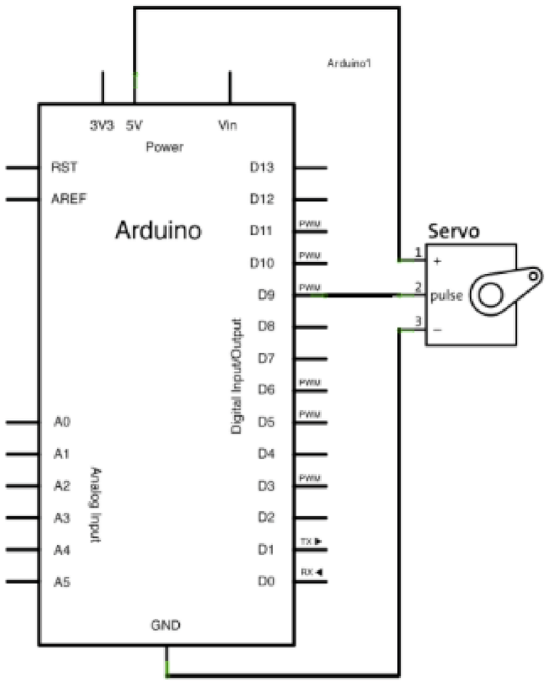


Figure 16 - Servo Motor Schematic

10.4 Distance Sensing System

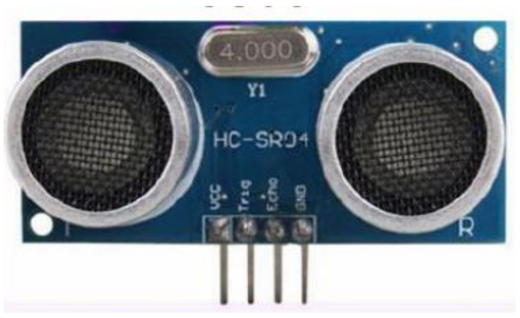


Figure 17 - HC-SR04 Sensor

The HC-SR04 ultrasonic distance sensor shown in **Figure 17** was selected for use in this project. The sensor operates by emitting a series of eight, 40 kHz pulses then measuring the delay before the pulses are received by the embedded microphone. The sensor operates using four pins driven by the Arduino: one 5V power supply pin, ground, a trigger pin to drive the sensor, and an echo pin whose output must be measured by the Arduino to get distance.

The sensor is operated by sending a logical high signal to the sensor's trigger pin for at least 10µs. Once the signal is returned to low, the sensor sends the eight pings, and returns a logical high pulse on the echo pin for the duration the system measured. The length of this pulse is in µs with a range of about 150µs-25ms, with 38ms reported if no obstacle is detected. The Arduino can be used to measure the length of the pulse in microseconds, which is then divided by 2 to account for the signals traveling out and back, then by a factor of 29 to convert into cm based on the speed of sound. This process in hardware is as shown below in **Figure 18**.

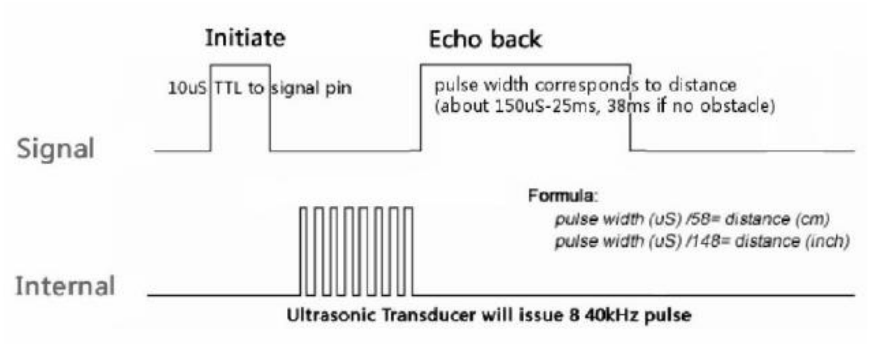

Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

Figure 18 - Distance Measurement Procedure

The sensor is currently wired to pins 18 and 19 of the Arduino, with power and ground supplied by the Arduino as well. The connections are as shown below in **Figure 19**.

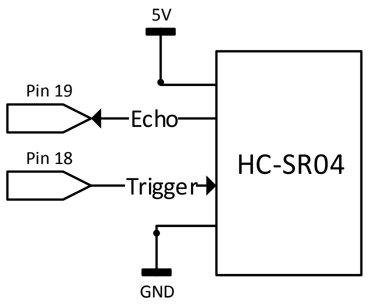


Figure 19 - HC-SR04 Wiring

While the distance is used by the system software, the distance also has an effect on system hardware as well. The distance value is, after being measured, mapped linearly from a scale of 0 to 31 to be fed into a digital potentiometer that drives the brightness of the attached distance-indicating LED. The pinout for the digital potentiometer is as shown below in **Figure 20**.

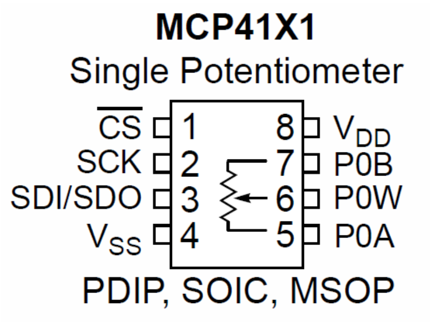


Figure 20 - Digital Potentiometer Pinout

As can be observed from the pinout, this potentiometer is controlled using SPI, not a physical knob. This has the benefit of allowing the pot to be adjusted by hardware signals from the Arduino on the fly by software, instead of requiring manual adjustment as with a traditional analog pot. This carries the downside of having only discrete resistance values, not a full range of analog values as the pot internally is actually a resistor ladder as shown below.

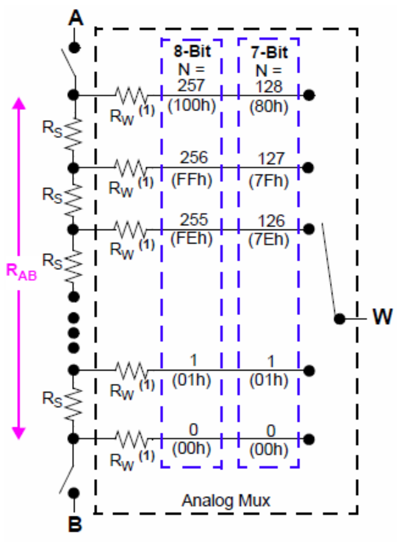


Figure 21 - Digital Potentiometer Resistance Ladder

This particular model of potentiometer is 7-bit, meaning there are only 128 discrete resistance values that can be modeled by the pot. In testing, only the lowest 32 resistance values gave an appreciable difference in LED brightness, and increasing it beyond those first 32 steps left the LED just as dim as at the 32nd step. Thus, it was decided to do the mapping of the distance value from 0 to 31 instead of from 0 to 127, which would have been unnecessary and not very useful as there is no visible difference between brightness at step 31 and 127 with the current LED.

There was an additional hardware consideration to be made. Given the 8-pin limitation on the potentiometer package, the SDI and SDO pins are internally multiplexed to a single external pin. This means that the SDI and SDO pin connections to the host (the Arduino) must also be multiplexed. Research led to the following solution, which was proven to work in testing. In this situation, the resistance of R1 as shown in **Figure 22** is 4.7kΩ.

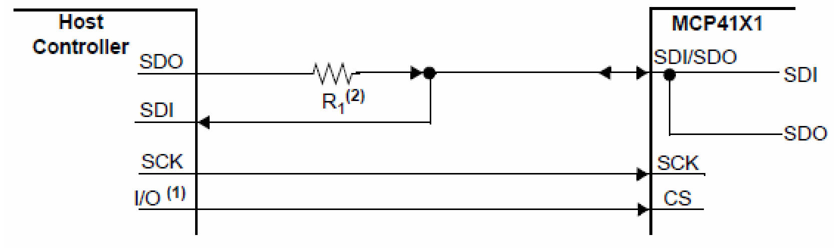


Figure 22 - SPI Interface between Arduino and Digital Pot

The hardware for this distance display system, including the Arduino pins used, the MCP4131, and the blue LED being driven, is as shown below in **Figure 23**.

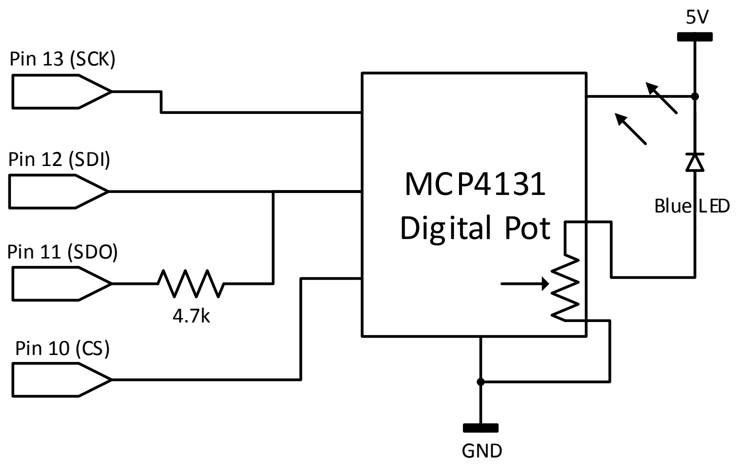


Figure 23 - Distance Display Wiring

10.5 Robot Motion Annunciation System

The main components of this system are a tri-color LED package, resistors to limit current, the 5V power rail from the Arduino, and connections to pins 14-16. In order to light an LED, the respective pin is driven low using the digitalWrite command in the Arduino library, thus driving current across the LED and lighting it. An LED is turned off by sending the connected pin high, putting 5V at both ends of the LED, therefore no current, and no light. Each LED has its current limited by a connected 330-ohm resistor, leftovers from the lab 2 LED driver. The connections for the Motion Annunciation System are as shown in **Figure 24**.

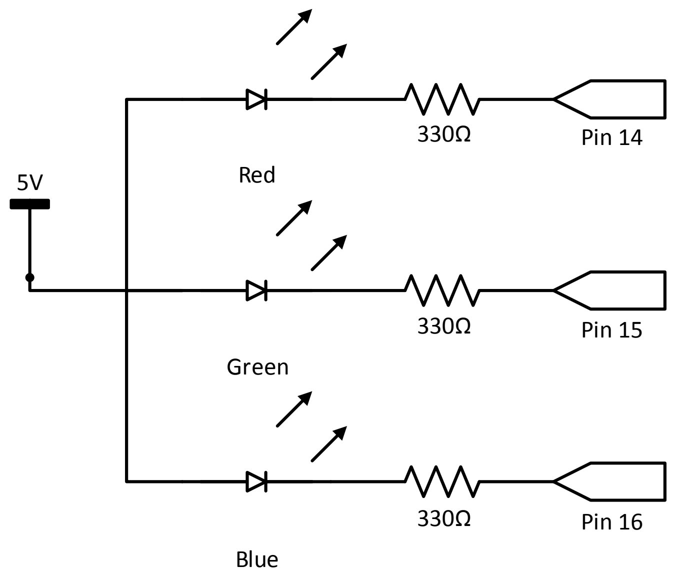


Figure 24 - Wiring Diagram for the Tri-Color LED

11 TESTING

11.1 Test Plan

To ensure the design meets the specified requirements, a number of tests must be performed on the system. The following sub-systems must be tested for correct performance and reliability: Bluetooth connection, serial connection via Bluetooth, Android interface, distance sensor, motor control, arm control, and autonomous motion.

The Bluetooth connection must be checked that it can correctly pair with the user's Android device quickly and reliably with the ability to send and receive serial data to and from the paired Arduino at a baud rate of 9600. The quality of data must also be checked, to see that the proper data is arriving and being transmitted at both ends of the connection.

Secondly, the interface developed for the Android device must be tested for correct operation. The interface buttons are to appear depressed when touched by the user finger, and the proper user feedback shall be displayed in the text field of the interface as necessary.

User speech interpretation must also be verified for correctness based on the user's accent and ability to speak clear English.

Lastly, the motor control system must be checked for consistency and reliability. The motor must rotate in the correct direction for the given PWM, at the correct speed for the given voltage, amperage, and gear ratio with which the motor is configured. This also applies to the servos, which are expected to rotate a specific number of degrees when told to do so.

The autonomous algorithm must correctly display the ability to drive past obstacles.

11.2 Test Specification

11.2.1 Bluetooth Connection and Android Interface

- Test for successful handshaking between Android and Arduino Bluetooth module
- Test that proper character is sent from Anroid to Arduino serial input
- Button presses result in respective character output to serial. Commands must match exactly:
    - Up = '1'
    - Down = '0'
    - Left = '2'
    - Right = '3'
    - Stop = '4'
    - Enter autonomous mode = '7'
    - Grasp/Release arm = '8'
    - Arm down = '9'
    - Arm up = 'A'
    - Arm left = 'B'
    - Arm right = 'C'
    - Arm reset = 'D'

    - Rotate left 30 degrees = ‘E’
    - Rotate left 60 degrees = ‘F’
    - Rotate right 30 degrees = ‘H’
    - Rotate right 60 degrees = ‘I’
    - Rotate 180 degrees ‘ G’
- Check that all voice commands for the above commands also work (with the exception of arm controls, which are button-press only). In addition, check that voice command variations are also recognized, such as “move forward” and “go forward” being synonymous to the robot
- Check that robot motion also sends correct status message over serial back to the android interface
    - Forward motion = “Moving Forward”
    - Backward motion = “Moving Backward”
    - Turning Left = “Moving Left”
    - Turning Right = “Moving Right”
    - Stop = “Brake”
    - Autonomous motion = “Enable Autonomous Mode”

11.2.2 Arm and Motor Control

- Check that motors operate correctly:
    - Left and right motors both driving forward for forward motion
    - Left and right motors both driving backwards for backward motion
    - Left motor driving forward and right back for right rotation
    - Left motor driving backward and right forward for left rotation
    - Both motors stopped for idle/braked
- Check that arm motion is correct and also correctly constrained
    - Arm horizontal rotation restricted to frontal cone from 10 degrees (left) to 150 degrees (right) rotation
    - Arm vertical rotation restricted to a cone from 120 degrees (full up) to 30 degrees (full down)
    - Grasping motion closes claw to 120 degrees, and opens to 65 degrees
    - Up to ±5 degrees of error acceptable due to nature of the claw vs. the size of the objects it’s intended to grab

11.2.3 Distance Sensing

- Verify correct operation from 3 cm up to 3.5 meters with error of less than 1cm due to rounding and signal noise
- Test that robot correctly stops within 7cm of a hazard, with error of up to 1cm
- Test that LED luminosity varies properly in relation to distance, range of 1.4V to 5V proportional to distance

11.2.4 Robot Motion Annunciation System

- Verify correlation between displayed color and robot motion

    - Green for forward motion
    - Red for backwards motion
    - Cyan for right turns
    - Yellow for left turns
    - Off for idle
    - White when recently reset

11.2.5 Autonomous Mode

- Verify correct pattern of obstacle navigation
    - Robot moves straight until present with obstacle
    - Distance sensor turns left 90 degrees, measures distance, then repeats for 90 degrees right of the chassis
    - Robot turns in direction of greatest distance and continues until next obstacle

11.3 Test Cases

11.3.1 Bluetooth Connection and Android Interface

The cell phone being used for testing, an LG Nexus 4, is to have its Bluetooth system activated. From there the Bluetooth device, named Linvor, is to be paired with. The user application will boot correctly, and the Bluetooth module will display a solid red light once the handshaking is complete and the connection made. Commands will be sent to the Arduino by means of pressing the buttons on the Android application, and the robot will be monitored for correct operation of the respective actions. All actions should only be taken when the user is pressing the respective button, with a delay time of less than 25ms depending on the quality of the Bluetooth connection and the phone's connection to Wi-Fi. All buttons should display their pressed version of each graphic when pressed, to visually identify themselves from the other buttons. In addition, voice commands should also show correct operation.

11.3.2 Arm and Motor Control

Using the android application, the robot motor will be testing using the four directional buttons and the brake pedal buttons on the interface. Correction direction of travel for the corresponding button will be checked with no error other than a delay between button press and the command being sent over Bluetooth (less than 25ms).

Likewise, the arm will be tested manually using the Android interface. Motion of the arm should correspond exactly with the currently pressed button, until the arm reaches the limit of rotation for the current motion as outlined in the test specification. The exception is the arm grasping motion, which is limited to two discrete states (not counting the short period of rotation).

11.3.3 Distance Sensing

The robot will be controlled to move forward in a straight line, with a 7cm long segment of ruler attached to the front. When a hand or other object is placed in front of the sensor at the 7cm distance, the robot will be checked if it stopped or not in reaction to the obstacle. Likewise, objects will be placed closer to the sensor to verify that it works for closer ranges. In addition, it should be tested that the robot will not move forward if an obstacle is already within the collision range of the sensor.

Lastly, the blue LED connected to the digital potentiometer will be checked that it correctly varies its brightness. The LED should be dim if its path is clear, and gradually grow in brightness until almost blinding by the time the users hand reaches 3cm from the sensor.

11.3.4 Robot Motion Annunciation System

This system will be tested by remotely operating the robot and checking for the correct colors to be displayed by the LED as the robot moves. Forward motion should result in a green glow, right turns should illuminate the LED a cyan color, and so on as described in the test specification.

11.3.5 Autonomous Mode

The robot will be placed in front of a simple set of obstacles similar to a hallway with some sharp 90-degree turns. The robot will be given the voice command to enter autonomous mode, at which point it is expected to go forward, stop in front of the first wall, scan, turn left, go forward, scan again, and turn right before continuing on to the end of the course. This is depicted in **Figure 25**.

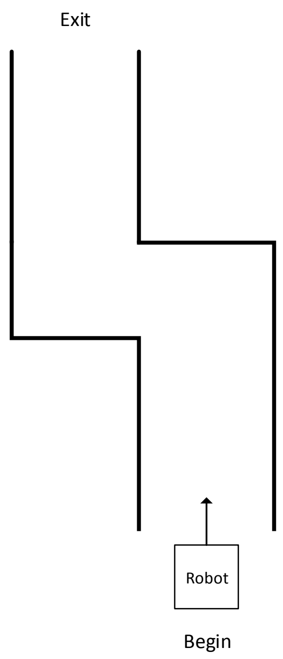


Figure 25 - Robot Autonomous Motion Course Example

12 PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS

The Bluetooth system demonstrates the ability to quickly and securely connect with the Android device, in addition to clearly sending commands. **Figure 26** demonstrates one example of the interface being used. In the upper left corner, the serial data from the Arduino is displayed. In this case, the robot has been given a voice command to move forward continuously, and the Arduino is likewise sending the status message "move forward" to the Android's display.

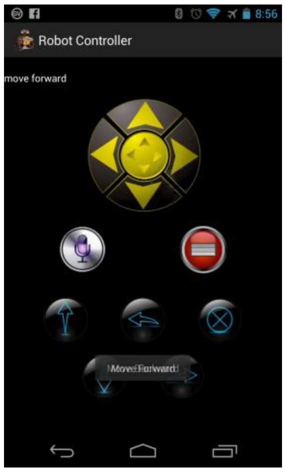


Figure 26 - Android Interface in Operation

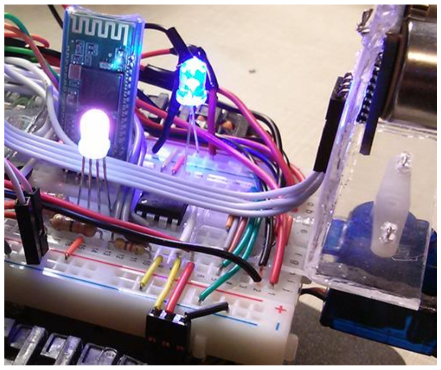


Figure 27 - LED at Max Brightness

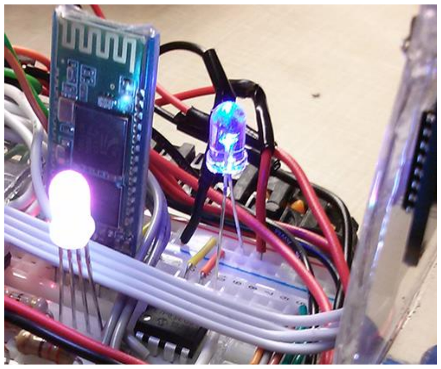


Figure 28 - LED at Min Brightness

As shown in **Figure 27** and **Figure 28**, the distance-sensing unit correctly controls the brightness of the LED as a hand is placed close to or far from the sensor, as expected. In addition, the robot demonstrates the ability to detect and halt when presented with an obstacle that is within the 7cm threshold of the robot.

Figure 29 - Various Tri-Color LED States

The above images in **Figure 29** illustrate the different colors displayed using the Tri-Color LED during robot operation. The LED is clearly shown as green when moving forward, red when backing up, cyan when turning right, yellow when moving left, white when a system reset is sent, and off when the robot is idle.

In addition, the robot clearly demonstrates the ability to control its arm and reach a variety of angles during operation. The figures below show the robot arm deployed at several angles to demonstrate its range of motion.

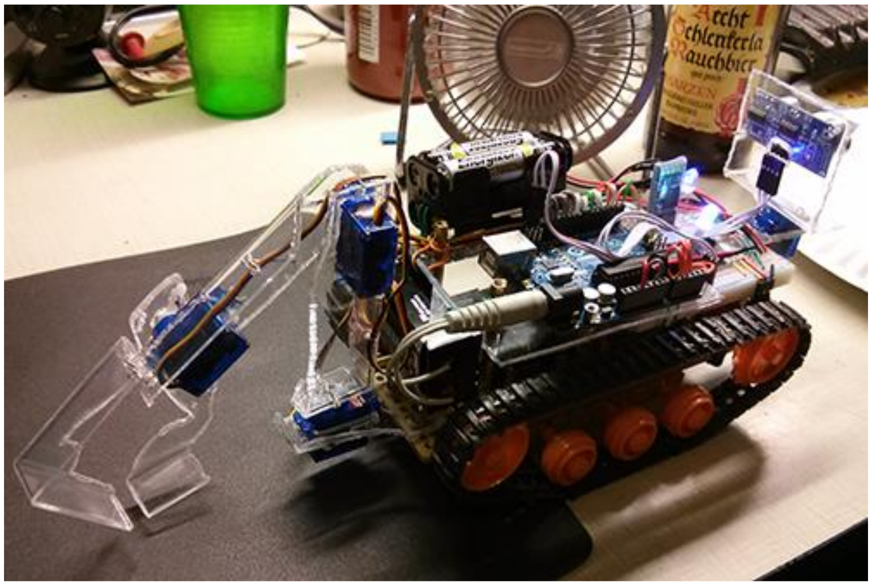


Figure 30 - Robot Arm Centered, Completely Down, Claw Closed

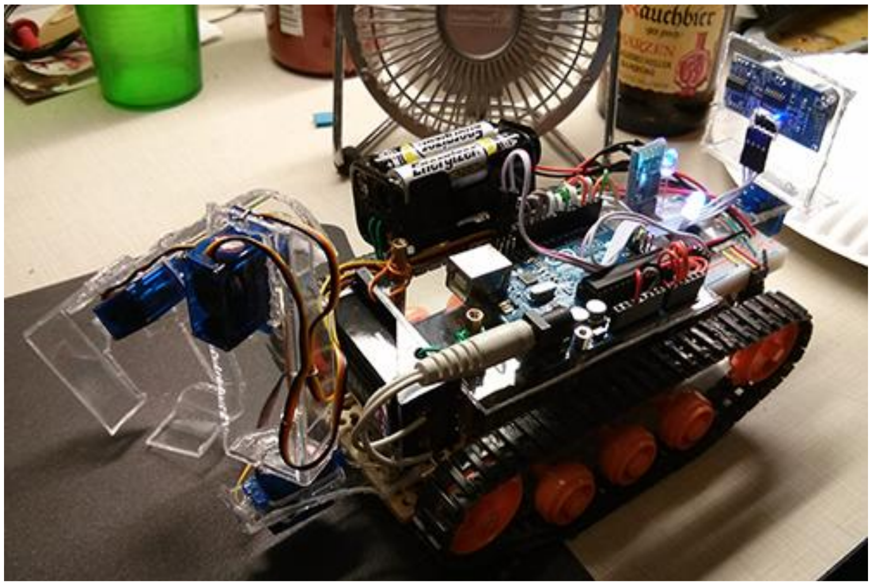


Figure 31 - Robot Arm Completely Left, Down, and Claw Closed

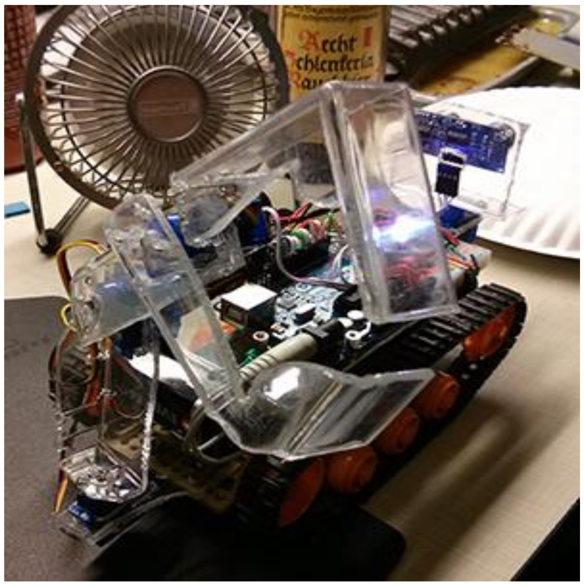


Figure 32 - Robot Arm Completely Right, Up, and Claw Closed

In addition to simple motor control, the robot also demonstrates the ability to navigate a simple set of obstacles, as described in the test cases. This was proven during testing and during the final project demo to the instructor.

In summary, all subsystems work properly and the robot operation meets all expectations. The minor errors in the system are as described in the following section.

13 ANALYSIS OF ERRORS

There are no remarkable errors in the operation of the system. The only error that interferes with the operation of the system is related to the voice recognition: users with heavy foreign accents are difficult for the Google servers to interpret, and generally, what they say comes out with enough of a difference from their intent that it changes from a recognized command to something the application is not programmed to recognize. An expansion on the project would be to include far more cases for how the command words could be mispronounced or misinterpreted by Google, but that was out of the scope of this project in its current state.

14 ANALYSIS OF FAILED SYSTEMS AND EFFORTS TO RECTIFY THEM

There was one major system that failed to work correctly and could not be implemented. One goal for the robot was to add variable speed control, so that the user could tell the robot to travel at higher or lower speeds. This was to be done using a simple voltage divider to drive the H-bridge instead of a direct connection to the system 5V rail. In order to add complexity, and of course practicality, a digital potentiometer was selected to serve as one part of the voltage divider. The potentiometer is controlled using SPI, which necessitated a review of SPI from Lab 2 and in particular the changes to it between the PIC and the Arduino. Through research and testing, an SPI library was developed that supplements the default Arduino library, and allows simple control of the pot with simple functions such as pot.increment, pot.decrement, and pot.setTap (set the value of the potentiometer). The voltage divider circuit worked flawlessly during testing; however, when the output of the divider was then connected to the input of the H-bridge, the voltage would reliably sink and never surpass approximately 1.2V.

A number of approaches were taken to try to solve the issue, which seemed to be the large input impedance of the H-bridge affecting the operation of the voltage divider. The first was to use an LM741 Op Amp to amplify the output voltage before sending it into the H-bridge. This was ineffective for the most part, only raising the max voltage level to about 2.1V in testing. Another idea was to use an additional voltage divider to offset the voltage output of the voltage divider, as was done in Lab 2 to boost the output of the thermocouple. This had worse results compared to the Op Amp approach, netting a max input voltage of about 1.8V. The final and most effective attempt came through research and some recollection of a concept covered in EE 233 where Op Amps can be used to isolate parts of a circuit from each other and eliminate loading effects, such as those when connecting a low impedance device to one of high impedance. A unity buffer was made using the op amp and this resulted in a maximum output voltage of 2.7V, which while the best of all approaches, ultimately was not enough to drive the motors, which need at least 3V to move at their slowest speed in practice.

Unable to proceed with that part of the project and with time wasting, it was decided to make use of the SPI libraries and hardware design that had already been invested in, and instead use the digital potentiometer to drive the brightness of the distance indication LED, as previously described in the hardware and software implementation sections of the report.

15 SUMMARY

In summary, the lab project is a complete success and all features were implemented based on the expectations made as the project progressed. While it is a shame that the motor speed control functionality was not able to be implemented, the code and resources for it were able to be put to good use in the other sections of the robot so minimal time and effort was lost to that endeavor. The robot functions as desired, with the ability to move in all directions on the intended surfaces, can grasp reasonably small objects with the girth necessary by the claw design, and properly reacts to obstacles in the environment both when manually controlled and when operating autonomously. Robot status is correctly relayed by the colored LEDs, and distance is visually indicated by a variable-brightness LED. Coding was primarily done in the Eclipse IDE for the Android application, and the Arduino IDE for the Arduino .ino files.

16 CONCLUSION

The main objective of this capstone project was to allow the group to collaborate on and execute an independent engineering project tying together all the skills that each member brings to the team’s collective knowledge base. Students were tasked with developing a project that was equally complex in both software and hardware aspects over a short time span. In addition, students had to refine their project management skills, set weekly goals and deadlines, and delegate tasks based on the skills that each member had to offer. This served as an excellent exercise of engineering in industry.

All major system components were thoroughly tested and meet all project specifications. The robot moves correctly based on user input, reacts perfectly to voice commands, annunciates its actions to the user appropriately, avoids collisions with obstacles, and can even navigate simple sets of obstacles autonomously.

In conclusion, this was a great simulation of engineering in the real world and a successful lab project to wrap up the embedded concentration. Scheduling, design, and testing were essential, and along the way a number of tough design decisions had to be made, including making design trade-offs and finding alternative methods to solve certain problems or ways to repurpose previously failed designs.

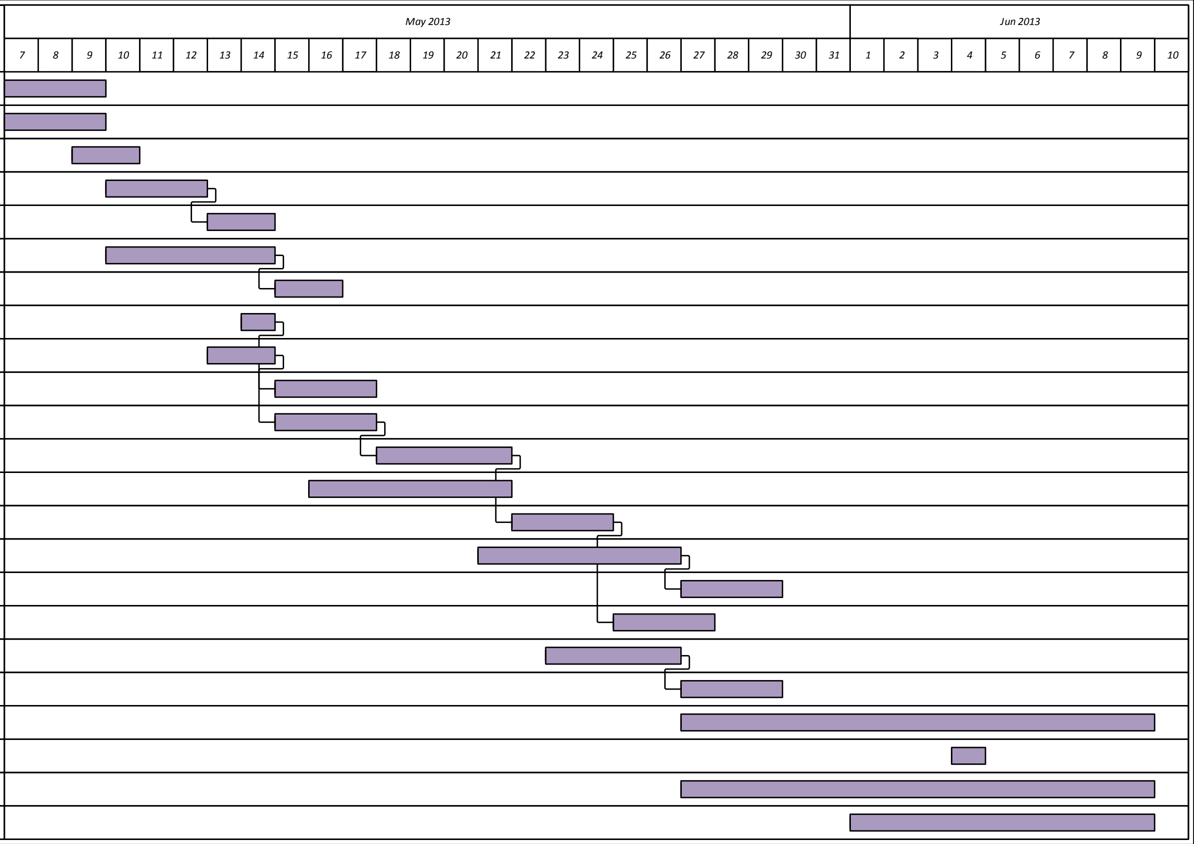
17 APPENDICIES

17.1 Bill of Materials

Name	Quantity	Price/unit	Total
Arduino Uno	1	$21.95	$21.95
HS-422 Servo Motor	4	$9.99	$39.96
Bluetooth Module	1	$9.49	$9.49
Vehicle chassis + wheels	1	$20.04	$20.04
H-bridge	1	$6.00	$6.00
Standoffs (Bulk)	1	$10.00	$10.00
Ultrasonic Sensor	1	$5.19	$5.19
Acrylic sheet (wire kit box, on hand)	1	$0.00	$0.00
Jumper wire set	1	$10.00	$10.00
Breadboard	1	$13.00	$13.00
Tamiya Double Gearbox	1	$10.19	$10.19
10k Digital Potentiometer	1	$1.50	$1.50
Tri-color LED	1	$4.00	$4.00
Blue LED	1	$0.30	$0.30
330 Ohm Resistors	3	$0.10	$0.30
9V Battery	1	$3.96	$3.96
AA Batteries	8	$0.60	$4.80
9V Battery Mount	1	$3.96	$3.96
AA Battery mount (holds 4)	2	$4.00	$8.00
Glue	1	$5.00	$5.00
		Grand Total	**$177.64**

17.2 Schedule

ID	Task Name	Designer	Start	Finish	Duration
1	Preliminary Interface Design	Tianshu	5/7/2013	5/9/2013	3d
2	Preliminary Bluetooth Testing	Tianshu	5/7/2013	5/9/2013	3d
3	Distance Sensor Design	Shawn	5/9/2013	5/10/2013	2d
4	Distance sensor control code	Shawn	5/10/2013	5/12/2013	3d
5	Distance sensor testing	Shawn	5/13/2013	5/14/2013	2d
6	Interface: Robot control	Tianshu	5/10/2013	5/14/2013	5d
7	Interface testing stage 1	Shawn	5/15/2013	5/16/2013	2d
8	Motor assembly	Tianshu	5/14/2013	5/14/2013	1d
9	Motor control code	Tianshu	5/13/2013	5/14/2013	2d
10	Motor performance testing	Shawn and Tianshu	5/15/2013	5/17/2013	3d
11	Robot Chassis design and build	Shawn	5/15/2013	5/17/2013	3d
12	Arm Design and build	Dan	5/18/2013	5/21/2013	4d
13	Arm actuation control code	Shawn and Tianshu	5/16/2013	5/21/2013	6d
14	Grasper attachment design	Dan	5/22/2013	5/24/2013	3d
15	Grasper control code	Shawn and Tianshu	5/21/2013	5/26/2013	6d
16	Arm movement and grasper testing	Shawn and Tianshu	5/27/2013	5/29/2013	3d
17	Arm mounting system design	Shawn	5/25/2013	5/27/2013	3d
18	Arm rotation control code	Tianshu	5/23/2013	5/26/2013	4d
19	Arm mount testing	Shawn and Tianshu	5/27/2013	5/29/2013	3d
20	System debug and interface polishing	Shawn and Tianshu	5/27/2013	6/9/2013	14d
21	Deadline to determine sensor for location sensing	All	6/4/2013	6/4/2013	1d
22	Final system performance verification	Shawn and Tianshu	5/27/2013	6/9/2013	14d
23	Write and edit Final documentation	Shawn	6/1/2013	6/9/2013	9d



17.3 Failure Modes Analysis

Item/ Function	**Potential Failure Mode**	**Potential Cause**	**Local Effects**	**Probability**	**Severity**	**Mitigation Requirements**
Bluetooth adapter	Loss of pairing with android	Distance from paired device, current overload	Loss of control of robot	(B) Remote	(V) Critical	System will hang and attempt to regain paired connection. Robot will not operate until repaired to prevent unwanted movement.
Ultrasonic Sensor	Failure to return pulse	Current damage to circuitry, insufficient pulse from Arduino	Distance data unavailable	(B) Remote	(III) Minor	Robot will operate as normal without the collision detection function available.
Servo	Motor failure	Insufficient power, mechanical failure	Inability to move related section of arm	(A) Extremely Unlikely	(IV) Moderate	Arm will move based on still available servos, degrees of freedom from damaged servo will not be available
Voice interpreter	Improper recognition	User speaks unclearly, with heavy foreign accent	Command values may be misread, command type misread	(D) Reasonably Possible	(II) Very minor	User must speak clearly or be able to recognize the improper command and use the halt command to cancel movement.

17.4 Sequence Diagram

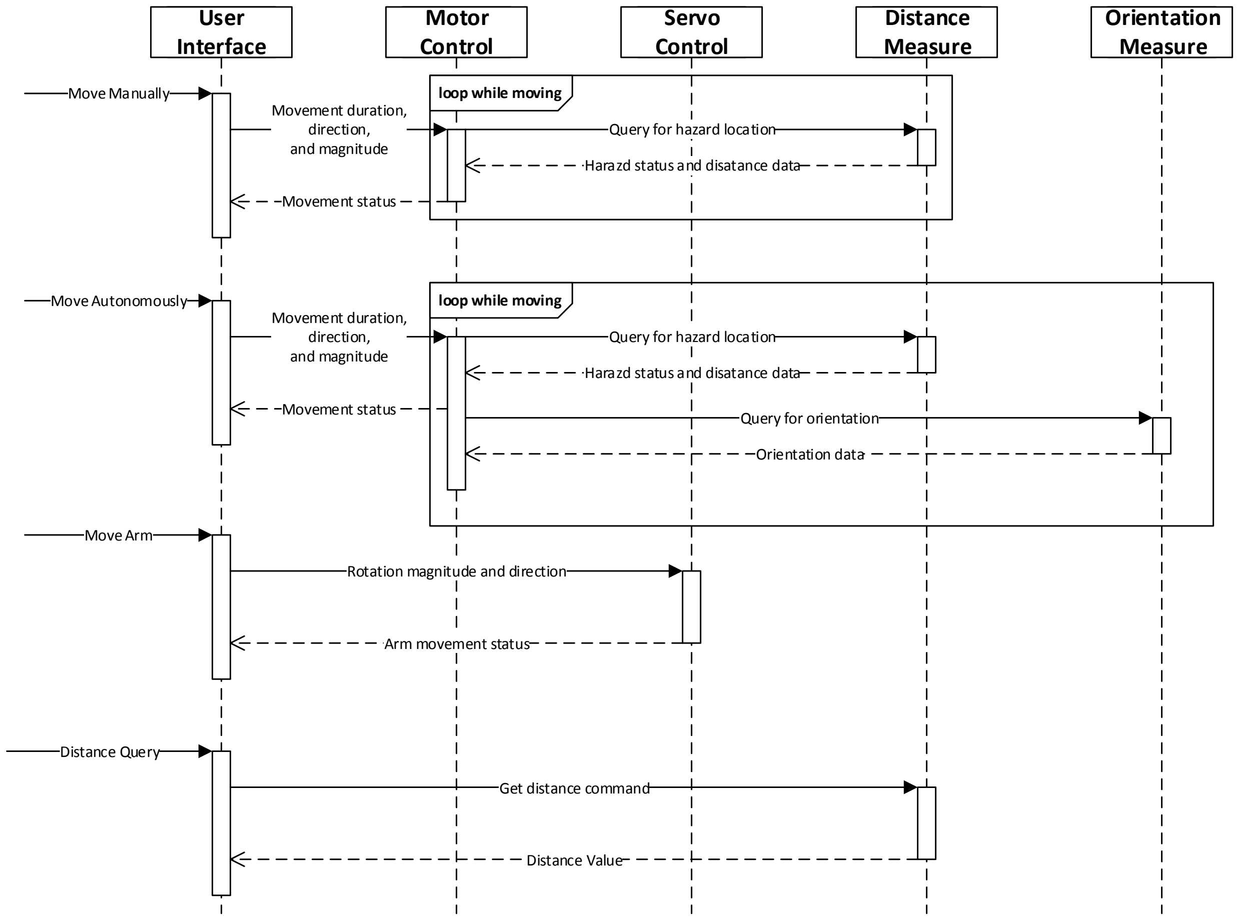


Figure 33 - System sequence diagram

17.5 Code

17.5.1 MainActivity.java (Android App)

```java
package com.example.robotController;

//Tianshu Bao
//Robot Control Project

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.UUID;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.graphics.Path;
import android.graphics.Point;
import android.graphics.RectF;
import android.graphics.Region;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.speech.RecognizerIntent;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnTouchListener;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
    private static final String TAG = "Robot Control";

    ImageView btnUp, btnDown, btnLeft, btnRight, btnVoice, btnAccel,
btnBreak;  //Robot Control buttons
    ImageView armUp, armDown, armLeft, armRight, armGrasp;  //Arm Control
buttons
    TextView txtArduino;
    Handler h;
    TextView myTextView;

    final int RECIEVE_MESSAGE = 1;       // Status  for Handler
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private StringBuilder sb = new StringBuilder();

    private ConnectedThread mConnectedThread;

    // SPP UUID service
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");

    // MAC-address for Bluetooth module
    private static String address = "00:13:03:27:00:09";

    private static final int VOICE_RECOGNITION_REQUEST_CODE = 1001;
    private boolean needtoConnect = true;

    private String previous_state = "4";

    private Region r_up;
    private Region r_down;
    private Region r_left;
    private Region r_right;

    private boolean moveLeft = false;
    private boolean moveRight = false;
    private boolean moveForward = false;
    private boolean moveBackward = false;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        txtArduino = (TextView) findViewById(R.id.txtArduino);      // for
display the received data from the Arduino
        myTextView = (TextView)findViewById(R.id.TextView1);

        btnUp = (ImageView) findViewById(R.id.btnUp);
        //btnDown = (ImageView) findViewById(R.id.btnDown);
        //btnLeft = (ImageView) findViewById(R.id.btnLeft);
        //btnRight = (ImageView) findViewById(R.id.btnRight);
        btnVoice = (ImageView) findViewById(R.id.VoiceControl);
        //btnAccel = (ImageView) findViewById(R.id.btnAccel);
        btnBreak = (ImageView) findViewById(R.id.btnBrake);

        armUp = (ImageView) findViewById(R.id.armUp);
        armDown = (ImageView) findViewById(R.id.armDown);
        armLeft = (ImageView) findViewById(R.id.armLeft);
        armRight = (ImageView) findViewById(R.id.armRight);
        armGrasp = (ImageView) findViewById(R.id.armGrasp);

        h = new Handler() {
            public void handleMessage(android.os.Message msg) {
                switch (msg.what) {
                case RECIEVE_MESSAGE:
// if receive massage
                    byte[] readBuf = (byte[]) msg.obj;
                    String strIncom = new String(readBuf, 0, msg.arg1);
// create string from bytes array
                    sb.append(strIncom);
// append string
```

```
                    int endOfLineIndex = sb.indexOf("\r\n");
// determine the end-of-line
                    if (endOfLineIndex > 0)
{                                                   // if end-of-line,
                        String sbprint = sb.substring(0, endOfLineIndex);
// extract string
                        sb.delete(0, sb.length());
// and clear
                        txtArduino.setText("Data from Arduino: " + sbprint);
// update TextView
                    }
                    //Log.d(TAG, "...String:"+ sb.toString() +  "Byte:" +
msg.arg1 + "...");
                    break;
                }
            };
        };

        btAdapter = BluetoothAdapter.getDefaultAdapter();       // get
Bluetooth adapter
        checkBTState();

        Draw();
        btnInitial();
    }

    //Initialize buttons
    private void btnInitial() {
        //Direction Control Button
        btnUp.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;

                Point point = new Point();
                point.x = (int)event.getX();
                point.y = (int)event.getY();
                Log.d(TAG, "point: " + point);

                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    if(r_up.contains((int)point.x,(int) point.y)) {
                        iv.setImageResource(R.drawable.forward);
                        mConnectedThread.write("1");
                        previous_state = "1";
                        moveForward = true;
                        Toast.makeText(getBaseContext(), "Move Forward",
Toast.LENGTH_SHORT).show();
                    } else if (r_down.contains((int)point.x,(int) point.y)) {
                        iv.setImageResource(R.drawable.backward);
                        mConnectedThread.write("0");
                        previous_state = "0";
                        moveBackward = true;
                        Toast.makeText(getBaseContext(), "Move Backward",
Toast.LENGTH_SHORT).show();
                    } else if (r_left.contains((int)point.x,(int) point.y)) {
                        iv.setImageResource(R.drawable.left);
                        mConnectedThread.write("2");
                        moveLeft = true;
                        Toast.makeText(getBaseContext(), "Move Left",
Toast.LENGTH_SHORT).show();
                    } else if (r_right.contains((int)point.x,(int) point.y))
{
                        iv.setImageResource(R.drawable.right);
                        mConnectedThread.write("3");
                        moveRight = true;
                        Toast.makeText(getBaseContext(), "Move Right",
Toast.LENGTH_SHORT).show();
                    }
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    if (moveForward) {
                        iv.setImageResource(R.drawable.normal);
                        mConnectedThread.write("4");
                        moveForward = false;
                    } else if (moveBackward) {
                        iv.setImageResource(R.drawable.normal);
                        mConnectedThread.write("4");
                        moveBackward = false;
                    } else if (moveLeft) {
                        iv.setImageResource(R.drawable.normal);
                        //mConnectedThread.write(previous_state);
                        mConnectedThread.write("4");
                        moveLeft = false;
                    } else if (moveRight) {
                        iv.setImageResource(R.drawable.normal);
                        //mConnectedThread.write(previous_state);
                        mConnectedThread.write("4");
                        moveRight = false;
                    }
                }

                return true;
            }
        });

        /*
        //Button Down
        btnDown.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    iv.setImageResource(R.drawable.down_pressed);
                    mConnectedThread.write("0");
                    previous_state = "0";
                    Toast.makeText(getBaseContext(), "Move Forward",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.down);
                    //mConnectedThread.write("4");
                }
                return true;
            }
        });
        //Button Left
        btnLeft.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    iv.setImageResource(R.drawable.left_pressed);
                    mConnectedThread.write("2");
                    Toast.makeText(getBaseContext(), "Move Right",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.left);
                    mConnectedThread.write(previous_state);
                }
                return true;
            }
        });

        //Button Right
        btnRight.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    iv.setImageResource(R.drawable.right_pressed);
                    mConnectedThread.write("3");
                    Toast.makeText(getBaseContext(), "Move Right",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.right);
                    mConnectedThread.write(previous_state);
                }
                return true;
            }
        });
        */

        //Voice Recognition
        btnVoice.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                speak();
            }
        });


        /*
        //Accelerate button
        btnAccel.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    iv.setImageResource(R.drawable.faster_button_pressed);
                    mConnectedThread.write("5");
                    Toast.makeText(getBaseContext(), "Accelerate",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.faster_button);
                    mConnectedThread.write("6");
                }
                return true;
            }
        });
        */

        //Break Button
        btnBreak.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    iv.setImageResource(R.drawable.brake_button_pressed);
                    mConnectedThread.write("4");
                    previous_state = "4";
                    Toast.makeText(getBaseContext(), "Break",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.brake_button);
                    mConnectedThread.write("4");
                }
                return true;
            }
        });

        //Arm Grasp
        armGrasp.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    mConnectedThread.write("8");
                    iv.setImageResource(R.drawable.arm_grasp_press);
                    Toast.makeText(getBaseContext(), "Arm Grasp",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.arm_grasp);
                }
                return true;
            }
        });

        //Arm Up
        armUp.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    mConnectedThread.write("9");
                    iv.setImageResource(R.drawable.arm_up_press);
                    Toast.makeText(getBaseContext(), "Arm Up",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    iv.setImageResource(R.drawable.arm_up);
                    mConnectedThread.write("D");
                }
                return true;
            }
        });

        //Arm Down
        armDown.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    mConnectedThread.write("A");
                    iv.setImageResource(R.drawable.arm_down_press);
                    Toast.makeText(getBaseContext(), "Arm Down",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    mConnectedThread.write("D");
                    iv.setImageResource(R.drawable.arm_down);
                }
                return true;
            }
        });

        //Arm Left
        armLeft.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    mConnectedThread.write("B");
                    iv.setImageResource(R.drawable.arm_left_press);
                    Toast.makeText(getBaseContext(), "Arm Left",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    mConnectedThread.write("D");
                    iv.setImageResource(R.drawable.arm_left);
                }
                return true;
            }
        });

        //Arm Right
        armRight.setOnTouchListener(new OnTouchListener() {
            public boolean onTouch(View v, MotionEvent event) {
                ImageView iv = (ImageView) v;
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    mConnectedThread.write("C");
                    iv.setImageResource(R.drawable.arm_right_press);
                    Toast.makeText(getBaseContext(), "Arm Right",
Toast.LENGTH_SHORT).show();
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    mConnectedThread.write("D");
                    iv.setImageResource(R.drawable.arm_right);
                }
                return true;
            }
        });
    }

    //Get positions of the direction control button
    public void Draw() {
        Path p_up = new Path();
        Path p_down = new Path();
        Path p_left = new Path();
        Path p_right = new Path();

        p_up.moveTo(78, 20);
        p_up.lineTo(154, 130);
        p_up.lineTo(270, 130);
        p_up.lineTo(330, 20);
        p_up.close();

        p_down.moveTo(78, 364);
        p_down.lineTo(158, 270);
        p_down.lineTo(250, 270);
        p_down.lineTo(330, 364);
        p_down.close();

        p_left.moveTo(20, 74);
        p_left.lineTo(134, 152);
        p_left.lineTo(134, 272);
        p_left.lineTo(20, 340);
        p_left.close();

        p_right.moveTo(384, 74);
        p_right.lineTo(276, 146);
        p_right.lineTo(276, 256);
        p_right.lineTo(370, 330);
        p_right.close();


        RectF rectUP = new RectF();
        p_up.computeBounds(rectUP, true);
        r_up = new Region();
        r_up.setPath(p_up, new Region((int) rectUP.left, (int) rectUP.top,
(int) rectUP.right, (int) rectUP.bottom));

        RectF rectDOWN = new RectF();
        p_down.computeBounds(rectDOWN, true);
        r_down = new Region();
        r_down.setPath(p_down, new Region((int) rectDOWN.left, (int)
rectDOWN.top, (int) rectDOWN.right, (int) rectDOWN.bottom));

        RectF rectLEFT = new RectF();
        p_left.computeBounds(rectLEFT, true);
        r_left = new Region();
        r_left.setPath(p_left, new Region((int) rectLEFT.left, (int)
rectLEFT.top, (int) rectLEFT.right, (int) rectLEFT.bottom));

        RectF rectRIGHT = new RectF();
        p_right.computeBounds(rectRIGHT, true);
        r_right = new Region();
        r_right.setPath(p_right, new Region((int) rectRIGHT.left, (int)
rectRIGHT.top, (int) rectRIGHT.right, (int) rectRIGHT.bottom));

    }

    //Create bluetooth socket
    private BluetoothSocket createBluetoothSocket(BluetoothDevice device)
throws IOException {
        if(Build.VERSION.SDK_INT >= 10){
            try {
                final Method  m =
device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord", new
Class[] { UUID.class });
                return (BluetoothSocket) m.invoke(device, MY_UUID);
            } catch (Exception e) {
                Log.e(TAG, "Could not create Insecure RFComm Connection",e);
            }
        }
        return  device.createRfcommSocketToServiceRecord(MY_UUID);
    }

    @Override
    public void onResume() {
        super.onResume();


        if (btSocket != null && !btSocket.isConnected()) {
            needtoConnect = true;
        }


        if (needtoConnect) {
            Log.d(TAG, "...onResume - try connect...");

            // Set up a pointer to the remote device using it's mac address.
            BluetoothDevice device = btAdapter.getRemoteDevice(address);

            //Create bluetooth socket
            try {
                btSocket = createBluetoothSocket(device);
            } catch (IOException e) {
                errorExit("Fatal Error", "In onResume() and socket create
failed: " + e.getMessage() + ".");
            }

            // Discovery is resource intensive.
            btAdapter.cancelDiscovery();

            // Establish the connection.  This will block until it connects.
            Log.d(TAG, "...Connecting...");
            try {
                btSocket.connect();
                Log.d(TAG, "....Connection ok...");
                Toast.makeText(getBaseContext(), "Bluetooth Connected",
Toast.LENGTH_SHORT).show();
                needtoConnect = false;
            } catch (IOException e) {
                try {
                btSocket.close();
                needtoConnect = true;
                } catch (IOException e2) {
                    errorExit("Fatal Error", "In onResume() and unable to
close socket during connection failure" + e2.getMessage() + ".");
                }
            }

            Log.d(TAG, "...Create Socket...");

            // Create a data stream so we can talk to server.
            mConnectedThread = new ConnectedThread(btSocket);
            mConnectedThread.start();
        }
    }

    @Override
    public void onPause() {
        super.onPause();

        Log.d(TAG, "...In onPause()...");

        /*
        try     {
          btSocket.close();
        } catch (IOException e2) {
          errorExit("Fatal Error", "In onPause() and failed to close socket."
+ e2.getMessage() + ".");
        }
        */
    }

    private void checkBTState() {
        // Check for Bluetooth support and then check to make sure it is
turned on
        if(btAdapter==null) {
            errorExit("Fatal Error", "Bluetooth not support");
        } else {
            if (btAdapter.isEnabled()) {
                Log.d(TAG, "...Bluetooth ON...");
            } else {
                //Prompt user to turn on Bluetooth
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);
            }
        }
    }

    private void errorExit(String title, String message){
        Toast.makeText(getBaseContext(), title + " - " + message,
Toast.LENGTH_LONG).show();
        finish();
    }

    private class ConnectedThread extends Thread {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the input and output streams, using temp objects because
            // member streams are final
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            byte[] buffer = new byte[256];  // buffer store for the stream
            int bytes; // bytes returned from read()

            // Keep listening to the InputStream until an exception occurs
            while (true) {
                try {
                    // Read from the InputStream
                    bytes = mmInStream.read(buffer);
                    h.obtainMessage(RECIEVE_MESSAGE, bytes, -1,
buffer).sendToTarget();     // Send to message queue Handler
                } catch (IOException e) {
                    break;
                }
            }
        }

        /* Call this from the main activity to send data to the remote device
*/
        public void write(String message) {
            Log.d(TAG, "...Data to send: " + message + "...");
            byte[] msgBuffer = message.getBytes();
            try {
                mmOutStream.write(msgBuffer);
            } catch (IOException e) {
                Log.d(TAG, "...Error data send: " + e.getMessage() + "...");
              }
        }
    }

    //Voice Recognition
    public void speak() {
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

        // Specify the calling package to identify your application
        intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,
getClass().getPackage().getName());

        // Display an hint to the user about what he should say.
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Control Command");

        // Given an hint to the recognizer about what the user is going to
say
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);

        startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if (requestCode == VOICE_RECOGNITION_REQUEST_CODE) {
            if(resultCode == RESULT_OK) {
                ArrayList<String> voiceInput =
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                String command = voiceInput.get(0);
                directionControl(command);
                myTextView.setText(command);
            }
        }else if(resultCode == RecognizerIntent.RESULT_AUDIO_ERROR){
            showToastMessage("Audio Error");
        }else if(resultCode == RecognizerIntent.RESULT_CLIENT_ERROR){
            showToastMessage("Client Error");
        }else if(resultCode == RecognizerIntent.RESULT_NETWORK_ERROR){
            showToastMessage("Network Error");
        }else if(resultCode == RecognizerIntent.RESULT_NO_MATCH){
            showToastMessage("No Match");
        }else if(resultCode == RecognizerIntent.RESULT_SERVER_ERROR){
            showToastMessage("Server Error");
        }
        super.onActivityResult(requestCode, resultCode, data);
    }

    private String[] forward = {"go forward", "move forward", "go up", "go
front"};
    private String[] backward = {"go backward", "move backward", "go back",
"move back"};
    private String[] left = {"turn left", "move left", "go left"};
    private String[] right = {"turn right", "move right", "go right",
"alright", "can write", "goal right"};
    private String[] stop = {"break", "stop", "hold on"};
    private String[] back = {"turn around", "turn back"};
    private String[] auto = {"self control", "autonomous", "come back", "auto
control"};

    //Control the direction of the robot based on voice input
    private void directionControl(String command) {
        if (contains(forward, command)) {
            mConnectedThread.write("1");
        } else if (contains(backward, command)) {
            mConnectedThread.write("0");
        } else if (contains(left, command)) {
            controlLeft(command);
        } else if (contains(right, command)) {
            controlRight(command);
        } else if (contains(stop, command)) {
            mConnectedThread.write("4");
        } else if (contains(auto, command)) {
            mConnectedThread.write("7");
        } else if (contains(back, command)) {
            mConnectedThread.write("G");
        }
    }

    //Check if the input command is in default
    private boolean contains(String[] array, String command) {
        for (String s: array) {
            if (command.contains(s)) {
                return true;
            }
        }
        return false;
    }

    //Control left turn
    private void controlLeft(String command) {
        if (command.contains("turn left 60")) {
            mConnectedThread.write("F");
        } else if(command.contains("turn left 30")) {
            mConnectedThread.write("E");
        } else {
            mConnectedThread.write("2"); //Default turn left 90 degrees
            delay(1300);
            mConnectedThread.write("4");
        }
    }

    //Control right turn
    private void controlRight(String command) {
        if (command.contains("turn right 60")) {
            mConnectedThread.write("I");
        } else if(command.contains("turn right 30")) {
            mConnectedThread.write("H");
        } else {
            mConnectedThread.write("3"); //Default turn right 90 degrees
            delay(1300);
            mConnectedThread.write("4");
        }
    }

    private void delay(int t) {
        try {
            Thread.sleep(t);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    //Show toast message on screen
    void showToastMessage(String message){
        Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
    }
```

```
}
```

17.5.2 Splash.java (Splash screen for App)

```
package com.example.robotController;

//Tianshu Bao
//Robot Control Project Splash Screen

import java.util.Timer;
import java.util.TimerTask;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Window;

public class Splash extends Activity {

    // Set Duration of the Splash Screen
    long Delay = 3000;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Remove the Title Bar
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        // Get the view from splash_screen.xml
        setContentView(R.layout.splash);

        // Create a Timer
        Timer RunSplash = new Timer();

        // Task to do when the timer ends
        TimerTask ShowSplash = new TimerTask() {
            @Override
            public void run() {
                // Close SplashScreenActivity.class
                finish();

                // Start MainActivity.class
                Intent myIntent = new Intent(Splash.this,
                        MainActivity.class);
                startActivity(myIntent);
            }
        };

        // Start the timer
        RunSplash.schedule(ShowSplash, Delay);
    }

}
```

17.5.3 robot_control.ino (Primary Arduino file)

```
#include <SPI.h>
```

```
#include <digi_pot.h>
#include <Servo.h>

char incomingByte;  // incoming data

int autonomous = 0;  //Robot in self-control mode or not

digi_pot MCP4131(10); // with a CS pin of 10
int speedup = 0;

Servo sensorMotor;  //Pin 5

Servo armMotor1;
Servo armMotor2;
Servo armMotor3;

int ledDigitalOne[] = {14, 15, 16}; // Red, green, and blue LED pins
const boolean ON = LOW;       //Define on as LOW for LEDs
const boolean OFF = HIGH;     //Define off as HIGH

//Predefined Colors
const boolean RED[] = {ON, OFF, OFF};
const boolean GREEN[] = {OFF, ON, OFF};
const boolean BLUE[] = {OFF, OFF, ON};
const boolean YELLOW[] = {ON, ON, OFF};
const boolean CYAN[] = {OFF, ON, ON};
const boolean MAGENTA[] = {ON, OFF, ON};
const boolean WHITE[] = {ON, ON, ON};
const boolean BLACK[] = {OFF, OFF, OFF};

const boolean* COLORS[] = {RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, WHITE,
BLACK};


// Output pin
int trig = 18;
// Measurement pin
int echo = 19;

int check_distance = 0;

long distance = 0;

int grab = 0;
int up = 0;
int down = 0;
int left = 0;
int right = 0;

int brightness = 0;

void setup() {
    Serial.begin(9600); // initialization
    motor_Init();
    arm_Init();
    sensorMotor.attach(5);
    sensorMotor.write(90);
    MCP4131.setTap(127);
    pinMode(trig, OUTPUT);
    pinMode(echo, INPUT);
    for(int i = 0; i < 3; i++){
       pinMode(ledDigitalOne[i], OUTPUT); //Set the three LED pins as outputs
    }
}

void loop() {
    if (Serial.available() > 0) {  // if the data came
        incomingByte = Serial.read(); // read byte
        switch (incomingByte) {
          case '1': // 1 to move forward
            motor_stop();
            drive_forward();
            Serial.println("Moving Forward");  // print message
            setColor(ledDigitalOne, COLORS[1]);
            check_distance = 1;
            break;
          case '0':  // 0 to move backward
            motor_stop();
            drive_backward();
            Serial.println("Moving Backward");
            setColor(ledDigitalOne, COLORS[0]);
            check_distance = 0;
            break;
          case '2': // 2 to turn left
            motor_stop();
            turn_left();
            Serial.println("Moving Left");
            setColor(ledDigitalOne, COLORS[3]);
            check_distance = 0;
            break;
          case '3':  // 3 to turn right
            motor_stop();
            turn_right();
            Serial.println("Moving Right");
            setColor(ledDigitalOne, COLORS[4]);
            check_distance = 0;
            break;
          case '4':  // 4 to stop
            motor_stop();
            Serial.println("Break");
            setColor(ledDigitalOne, COLORS[7]);
            check_distance = 0;
            autonomous = 0;
            break;
          case '5':
            speedup = 1;
            break;
          case '6':
            speedup = 0;
            break;
          case '7':  //enable self-control mode
            autonomous = 1;
            Serial.println("Enable autonomous mode");
            break;
```

```
      case '8':
        //Arm Grasp and Release
        if (grab == 0) {
          grasp();
          grab = 1;
        } else {
          drop();
          grab = 0;
        }
        break;
      case '9':
        //Arm Down
        down = 1;
        break;
      case 'A':
        //Arm Up
        up = 1;
        break;
      case 'B':
        //Arm Left
        left = 1;
        break;
      case 'C':
        //Arm Right
        right = 1;
        break;
      case 'D':
        //Arm Stop
        left = 0;
        right = 0;
        up = 0;
        down = 0;
        break;
      case 'E':
        left_30();
        break;
      case 'F':
        left_60();
        break;
      case 'G':
        turn_back();
        break;
      case 'H':
        right_30();
        break;
      case 'I':
        right_60();
        break;
    }
  }

  //Self-control
  if (autonomous == 1) {
    selfControl();
  }

  //Distance in front of the robot
```

```
    distance = ping();
    brightness = map(distance, 10, 200, 0, 31);
    MCP4131.setTap(brightness);

    //Check distance
    if (check_distance == 1 && autonomous == 0) {
      Serial.println(distance);
      if (distance < 10) {  //if path is blocked, stop
        motor_stop();
        setColor(ledDigitalOne, COLORS[7]);
      }
    }

    /*
    if (speedup == 1) {
      MCP4131.decrement();
      delay(10);
    } else {
      MCP4131.increment();
      delay(10);
    }
    */
    if (up == 1) {
      arm_up();
    } else if (down == 1) {
      arm_down();
    } else if (left == 1) {
      arm_left();
    } else if (right == 1) {
      arm_right();
    }
}

/*
Sets an led to any color
*/
void setColor(int* led, boolean* color){

    for(int i = 0; i < 3; i++){
        digitalWrite(led[i], color[i]);
    }

}

void setColor(int* led, const boolean* color){
    boolean tempColor[] = {color[0], color[1], color[2]};
    setColor(led, tempColor);
}
```

17.5.4 arm_control.ino

```
int turn_degree = 90;
int rotate_degree = 30;

//Initialize arm servo motors
void arm_Init() {
  armMotor1.attach(3);
  armMotor2.attach(6);
  armMotor3.attach(9);
  armMotor1.write(90);
  delay(500);
  armMotor2.write(30);
  delay(500);
  armMotor3.write(120);
  delay(500);
}

//Grasp object
void grasp() {
  armMotor3.write(65);
  delay(500);
}

//Release object
void drop() {
  armMotor3.write(120);
  delay(500);
}

//Turn arm (Range from 10 to 170)
void arm_turn(int degree) {
  armMotor1.write(degree);
}

//Rotate arm (Range from 30 to 120)
void arm_rotate(int degree) {
  armMotor2.write(degree);
  delay(20);
}

void arm_up() {
  if (rotate_degree >= 30 && rotate_degree <= 120) {
    rotate_degree++;
    arm_rotate(rotate_degree);
    if (rotate_degree == 121) {
      rotate_degree = 120;
    }
  }
}

void arm_down() {
  if (rotate_degree >= 30 && rotate_degree <= 120) {
    rotate_degree--;
    arm_rotate(rotate_degree);
    if (rotate_degree == 29) {
      rotate_degree = 30;
    }
  }
}

void arm_left() {
  if (turn_degree >= 10 && turn_degree <= 170) {
    turn_degree--;
    arm_turn(turn_degree);
    if (turn_degree == 9) {
      turn_degree = 10;
    }
  }
}

void arm_right() {
  if (turn_degree >= 10 && turn_degree <= 170) {
    turn_degree++;
    arm_turn(turn_degree);
    if (turn_degree == 171) {
      turn_degree = 170;
    }
  }
}
```

17.5.5 motor_control.ino

```
int motor_left[] = {2, 4};
int motor_right[] = {7, 8};

void motor_Init() {
    // Setup motors
    int i;
    for(i = 0; i < 2; i++) {
        pinMode(motor_left[i], OUTPUT);
        pinMode(motor_right[i], OUTPUT);
    }
}

//Stop motor
void motor_stop() {
    digitalWrite(motor_left[0], LOW);
    digitalWrite(motor_left[1], LOW);

    digitalWrite(motor_right[0], LOW);
    digitalWrite(motor_right[1], LOW);
    delay(25);
}

//Move forward
void drive_backward() {
    digitalWrite(motor_left[0], HIGH);
    digitalWrite(motor_left[1], LOW);

    digitalWrite(motor_right[0], HIGH);
    digitalWrite(motor_right[1], LOW);
}

//Move backward
```

```
void drive_forward() {
    digitalWrite(motor_left[0], LOW);
    digitalWrite(motor_left[1], HIGH);

    digitalWrite(motor_right[0], LOW);
    digitalWrite(motor_right[1], HIGH);
}

//Turn left
void turn_left() {
    digitalWrite(motor_left[0], LOW);
    digitalWrite(motor_left[1], HIGH);

    digitalWrite(motor_right[0], HIGH);
    digitalWrite(motor_right[1], LOW);
}

//Turn right
void turn_right() {
    digitalWrite(motor_left[0], HIGH);
    digitalWrite(motor_left[1], LOW);

    digitalWrite(motor_right[0], LOW);
    digitalWrite(motor_right[1], HIGH);
}

//Turn left 30 degrees
void left_30() {
  turn_left();
  delay(500);
  motor_stop();
}

void left_60() {
  turn_left();
  delay(700);
  motor_stop();
}

void left_90() {
  turn_left();
  delay(1100);
  motor_stop();
}

void turn_back() {
  turn_left();
  delay(2700);
  motor_stop();
}

void right_30() {
  turn_right();
  delay(500);
  motor_stop();
}
```

```
void right_60() {
  turn_right();
  delay(700);
  motor_stop();
}

void right_90() {
  turn_right();
  delay(1100);
  motor_stop();
}
```

17.5.6 autonomous.ino

```
long duration;

int leftDistance, rightDistance;
const int dangerThresh = 8; //threshold for obstacles (in cm


void selfControl() {
  int distanceFwd = ping();
  if (distanceFwd > dangerThresh) {  //if path is clear
    setColor(ledDigitalOne, COLORS[1]);
    drive_forward();
  } else {
    motor_stop();
    sensorMotor.write(0);
    delay(500);
    rightDistance = ping(); //scan to the right
    delay(500);
    sensorMotor.write(180);
    delay(700);
    leftDistance = ping();
    delay(500);
    sensorMotor.write(90);
    delay(100);
    compareDistance();
  }
}

//Compare the distance of left and right
void compareDistance()
{
  if (leftDistance>rightDistance) {  //if left is less obstruct, turn left
    setColor(ledDigitalOne, COLORS[4]);
    right_90();
  } else if (rightDistance>leftDistance) {  //if right is less obstructed,
turn right
    setColor(ledDigitalOne, COLORS[3]);
    left_90();
  } else {  //if they are equally obstruct, turn around
    turn_back();
  }
}
```

```
//Send out distance sensor signals
long ping() {
  // Prime sensor by sending 10us pulse
  digitalWrite(trig,LOW);
  delayMicroseconds(2);
  digitalWrite(trig,HIGH);
  delayMicroseconds(10);
  digitalWrite(trig,LOW);
  delayMicroseconds(2);

  //Convert duration into distance
  duration = pulseIn(echo, HIGH);
  return duration / 29 / 2;
}
```

17.5.7 digi_pot.h (Header for the Digital Pot Control library)

```
/*
    Author: Shawn Stern
    Date: 5/28/2013
    EE 478 Final Lab

    Header for the MCP4131 digital pot. Uses Arduino SPI library.
*/

#ifndef MCP4131_h
#define MCP4131_h

#include "Arduino.h"
#include <SPI.h>

// SPI Pins
#define MOSI    11
#define MISO    12
#define SCK     13

// Pot min and max values
#define MCP4131_MIN 0
#define MCP4131_MAX 127

class digi_pot
{
public:
    digi_pot(int csPin);
    void increment();
    void decrement();
    void setTap(int value);


private:
    int cs;
    void enable();
    void disable();
};

#endif
```

17.5.8 digi_pot.c (Main library file for controlling the digital pot)

```
/*
    Author: Shawn Stern
    Date: 5/28/2013
    EE 478 Final Lab

    Source file for the MCP4131 digital pot control
    code. Uses Arduino SPI library.
*/

#include "digi_pot.h"

digi_pot::digi_pot(int csPin)
{
    //Set CS pin
    cs = csPin;
    pinMode(cs, OUTPUT);
    disable();
    // SPI setup
    SPI.begin();
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    SPI.setClockDivider(SPI_CLOCK_DIV128);
}

// Select the chip
void digi_pot::enable() {
    digitalWrite(cs, LOW);
}

// Deselect the chip
void digi_pot::disable() {
    digitalWrite(cs, HIGH);
}

// Increment the value of the pot by one step
void digi_pot::increment() {
    enable();
    SPI.transfer(0x06);
    disable();
}

// Decrement the value of the pot by one step
void digi_pot::decrement() {
    enable();
    SPI.transfer(0x0A);
    disable();
}

// Set potentiometer to a specific value 0-127
void digi_pot::setTap(int value) {
    enable();
    SPI.transfer(0);
    SPI.transfer(value);
    disable();
}
```