Sensing Mobile Android-assisted Real Time (SMART) Car

EE554 Real-time Computer Systems
Prof. Monte Ung
Spring 2012

Austen Hagio, hagio@usc.edu, (808) 398-8516
Hieu Nguyen, hmnguyen@usc.edu, (808) 383-2408
Lance Sakamoto, lsakamot@usc.edu, (808) 772-1943
Zachary Slavis, slavis@usc.edu, (702) 280-0825

Table of Contents

I. Introduction/Motivation

Traffic accidents are one of the leading causes of death among humans. It is predicted that by 2020, traffic accidents will exceed HIV/AIDS as a burden of death and disability. By 2030, it will become the fifth leading cause of death. Accidents occur because of humans' inability to react quickly in driving situations that require fast responses. Humans' reaction time is crippled as a result of drug usage such as drunk driving, physical impairment such as poor eyesight, sleep deprivation, poor road conditions caused by bad weather, inexperience, speeding, or distractions (both from inside the vehicle or from the surrounding environment). The occurrence of accidents is highly dependant on factors relating to human behavior, sensory perception, decision making, reaction speed, awareness, and alertness.

Our solution to the rising problem of road traffic accidents is to reduce the number of accidents that are caused by human error. We will build a car that can automatically avoid head-on collisions, maintain a constant distance while following another car, and allow manual control override by a remote user through a graphical user interface.

II. Approach

Avoiding traffic collisions is a life-critical task that depends on a real-time system to operate reliably. This system must meet its deadlines in a predictable manner but also process sensor data correctly to avoid false positive results. Our project encompasses the following aspects of a real-time system:

- analog data sampling from proximity sensors and phone sensors
- accurate motor control of the R/C car for collision avoidance
- wireless communication (synchronization, queueing)

- multithreaded Android application

III. Hardware Implementation

A. List of Parts

- R/C Corvette (1:10 scale)
- Arduino Uno R3
- BlueSMiRF Silver Bluetooth Modem
- Android Smart Phones (“Controller” and “Camera”)
- Sharp Infrared Proximity Sensor (short and long range)
- L293NE H-bridge
- 2-cell Lithium-Polymer Battery

B. System Diagrams

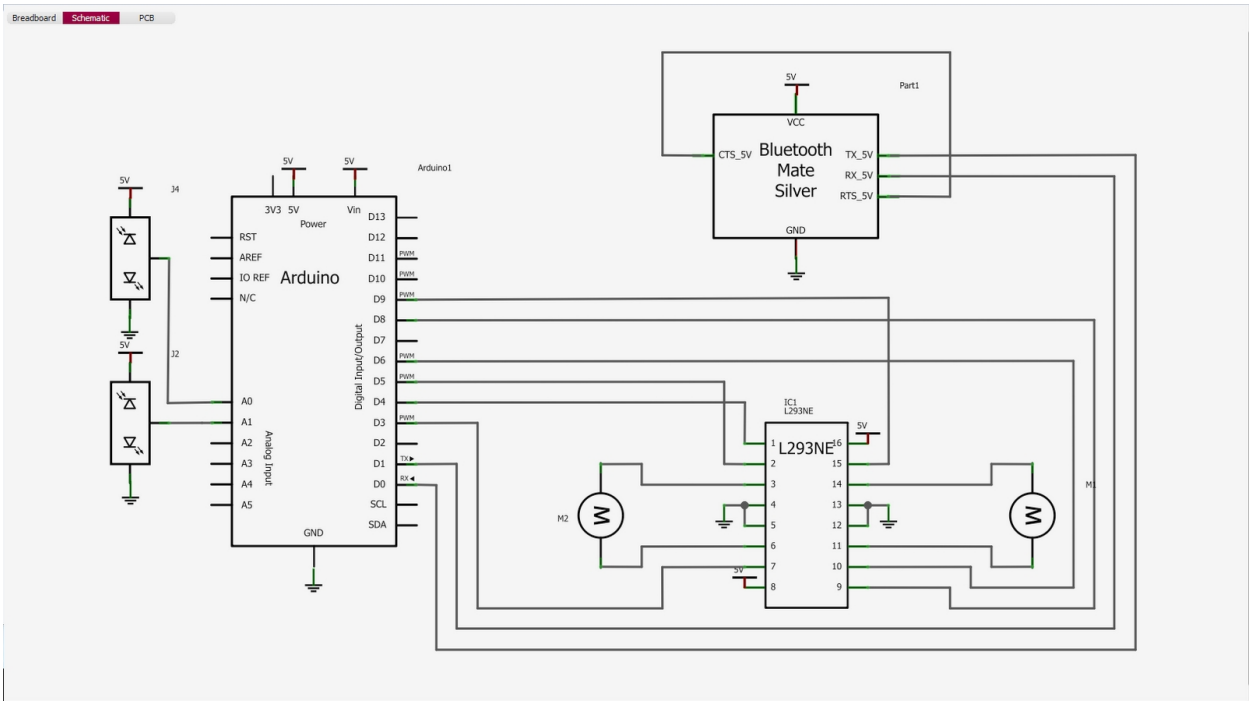


Figure 1. SMART Car On-board Schematic

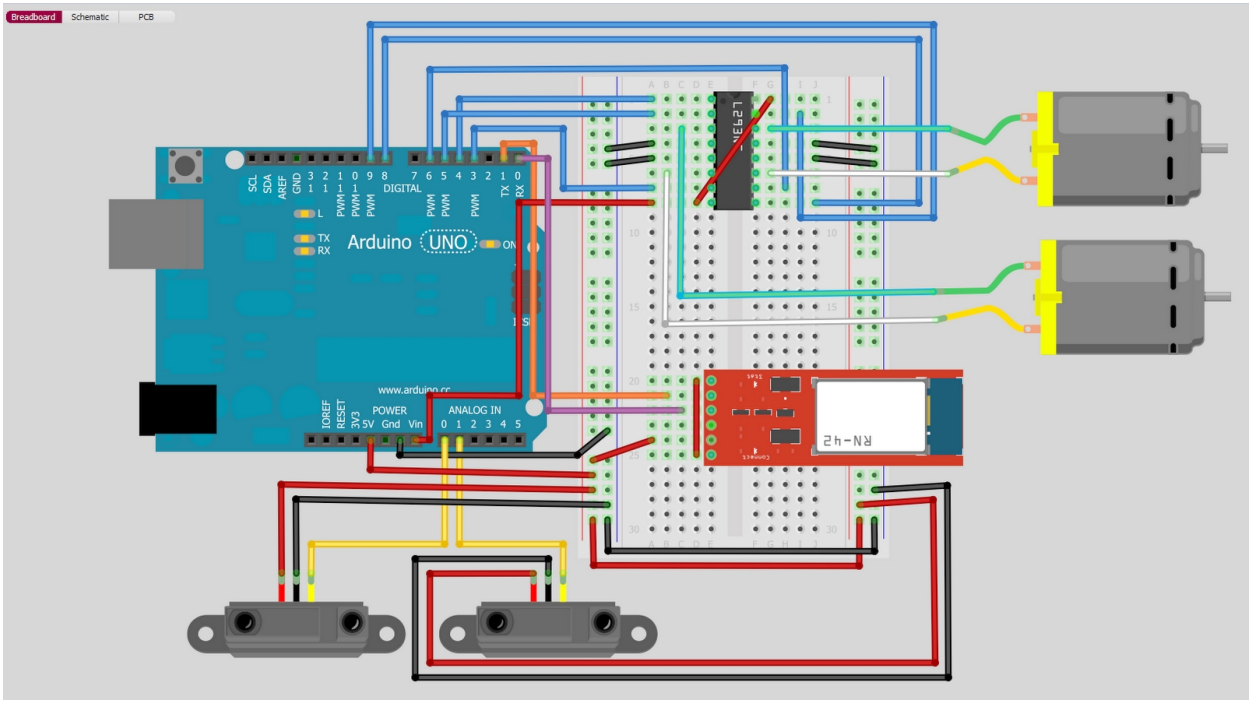


Figure 2. SMART Car Pictorial Diagram

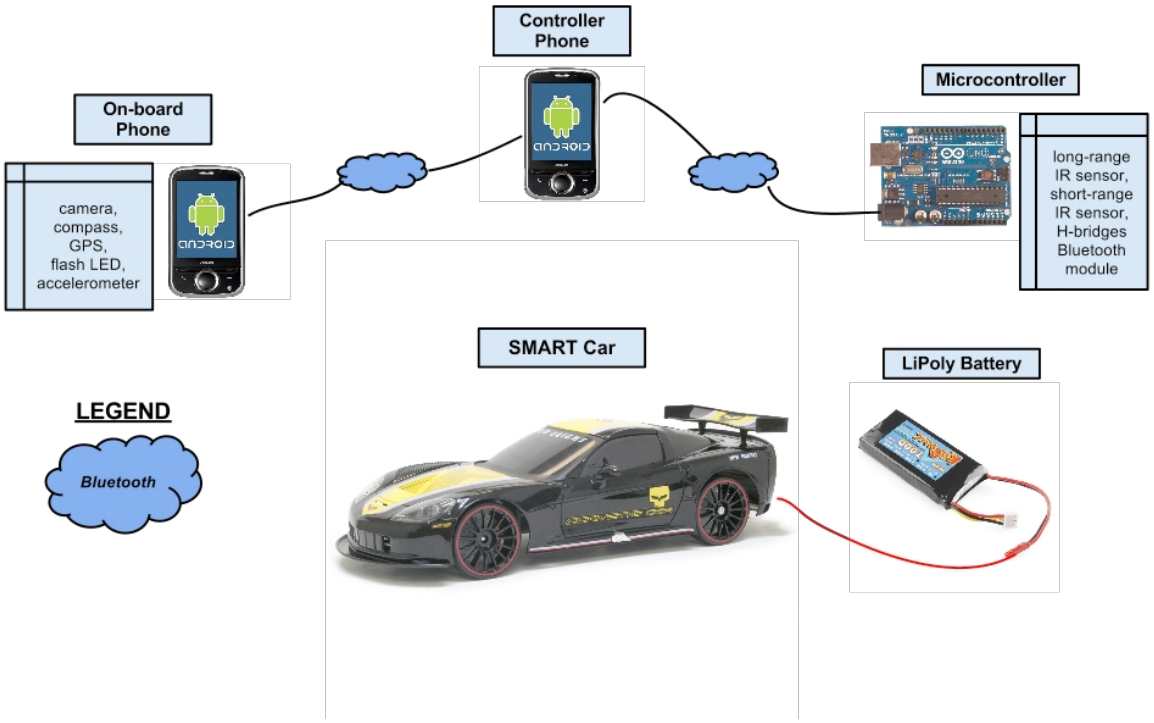


Figure 3. System Diagram

IV. Component Description

A. Arduino Uno Microcontroller

To process the data to and from the various components, we used an Arduino Uno microcontroller board. The Arduino Uno utilizes an ATmega16 microcontroller and has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a SDA and SCL pin, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

For uploading our code to the microcontroller, we used the open-source software Arduino Uno 1.0. All the code was written in C and used the Arduino's libraries.

B. Sensors

The SMART Car employs two different infrared sensors on its front and back with which it can judge its distance from obstacles. On the front of the vehicle, we mounted a Sharp GP2Y0S02YK0F Long Range Infrared Proximity Sensor, which can measure distances in the range of 20 to 150 cm. On the back of the vehicle, we mounted a Sharp GP2D120XJ00F Short Range Infrared Proximity Sensor, which can measure distances in the range of 4 to 30 cm. Because we believed that the vehicle would most likely spend most of its traveling time in the forward direction and at higher speeds than in the backward direction, we decided that the front of the vehicle had to be able to recognize obstacles that were farther away. However, the long range sensor seemed to perform better than the short range sensor at recognizing obstacles at a closer distance and could have been used on the back of the vehicle as well.

The analog voltage output of the infrared sensors varied exponentially with the measured distance.To calibrate the sensors we first tried to derive a function to define the relationship between analog voltage and distance. However, we soon discovered that the sensors we possessed did not accurately follow the behavior specified in the specification sheets. In addition, the sensors were prone to glitches due to slight vibrations of the sensors or changes in ambient light. We realized that we did not need an accurate calibration method and instead could implement collision avoidance functionality with approximations of voltage to distance mapping. We figured out a range of analog voltage values that mapped to distances we deemed

as “safe” and a range of analog voltage values that mapped to distances we deemed as “unsafe.” These ranges changed

dynamically with speed. The range of analog voltage values mapped to distances we deemed as “unsafe” were greater when the vehicle was traveling at a higher speed to allow for a greater stopping distance.

The Arduino Uno microcontroller reads the output voltage values from its analog pins every cycle. However, because the raw data obtained every cycle from the pins is likely to contain glitches, we used a running median of 9 cycles of values to give us a less glitch-affected analog output voltage from the sensors. The running median library can be found at: http://arduino.cc/playground/Main/RunningMedian. We realized, however, that even this median value was prone to glitches. Therefore, we accounted for this by enforcing the requirement that only three consecutive median values in state agreement could decide the distance state the vehicle was in—either safe or unsafe. When three consecutive median values fell in the range of “unsafe” values, a danger flag was set and passed to the system. The danger flag indicated to the system, specifically the motors, that action needed to be taken to avoid collision. The back and front sensors had separate danger flags.

To implement “cruise control” we used a similar error handling system to read sensor values. In cruise control mode, the sensors set one of three state flags to indicate to the system whether the vehicle is too far, at a safe distance, or too far from an object that it wants to follow.

The implementation of the running median and the requirement that three consecutive median distance values be used to set the distance state ensures reliability and redundancy. The sensors output real-time data to the Arduino Microcontroller for processing. The value of the danger flag is a result of that real-time processing.

C. Bluetooth

i. Hardware Interface

The Arduino Uno does not contain a built-in transceiver for wireless communication. Instead, an external module can be interfaced to its UART serial port, and communication with the TTL serial device can occur across the digital RX/TX pins. This was done using the BlueSMiRF Silver Bluetooth modem to send and receive data streams for motor control and mode configuration from a remote controller smart phone. By utilizing the Amarino "MeetAndroid" library, we implemented an effective protocol using flags and callback functions.

The BlueSMiRF device is a Roving Networks RN-42 FCC-approved Class 2 Bluetooth radio modem that delivers a data rate of 3 Mbps over a 20-meter range. It has a built-in antenna capable of transmitting the 2.4GHz signal at 4dBm output, with a typical receive sensitivity of -80dBm. The BlueSMiRF can operate in harsh RF environments by using frequency hopping spread spectrum (FHSS) and Gaussian frequency shift keying (GFSK) modulation over 79 channels at 1MHz intervals. With a 1152000bps baud rate, the connection between the BlueSMiRF and the Arduino is secure with 128-bit encryption and error correction. The BlueSMiRF runs off the 5VCC supply from the Arduino, and once powered, an on-board red LED begins flashing. Once a Bluetooth connection is made with the device, the red LED turns off and a solid green LED turns on to signify a secure connection. For pairing the BlueSMiRF with another Bluetooth device, the id is RN42-5D94 and the pin# is '1234'.

Once the BlueSMiRF is interfaced with the Arduino, data can be streamed across the UART port and viewed using the serial monitor. The communication protocol between the Arduino and the controller smart phone over Bluetooth, is implemented using the Amarino's

powerful communication infrastructure. The Amarino Toolkit (http://www.amarino-toolkit.net/) is a toolkit to connect Android-driven mobile devices with Arduino microcontrollers via Bluetooth. It provides easy access to internal phone events which can be further processed on the Arduino linked to the MeetAndroid library. Essentially, the Arduino registers a callback function to perform on receipt of a unique flag from the smart phone. Within the callback functions, we can then read the rest of the Bluetooth message that contains motor commands or operation mode configuration instructions. From here, our motor control library processes the motor commands and allows the car to move full speed ahead!

ii. Software Interface (Android Application)

In order to implement the unmanned vehicle feature, we will use an Android-powered phone to remotely control the vehicle. The Android-powered phone is equipped with a Bluetooth device, which will be used to send and receive pertinent data for controlling the unmanned vehicle. In order to control the vehicle, we have two requirements of the Android application:

- Motor Control – Send data to control the vehicle motors (steering motor and throttle motor).
- Video Stream – Receive a video stream from a camera mounted aboard the vehicle.

To complete the first requirement, the Android-powered phone will connect via Bluetooth to the Arduino microcontroller, which is capable of adjusting the motor inputs. By sending data via this Bluetooth connection to the Arduino, we can control the vehicle motors.

To complete the second requirement, the Android-powered phone will connect via Bluetooth to another Android-powered phone (with a built-in camera) which will be mounted on the vehicle, camera facing the forward path of the vehicle. From this point forward, we will refer

to the Android-powered phone used for controlling the vehicle as the “Controller Android” and the Android-powered phone mounted aboard the vehicle as the “Camera Android.” By having the Camera Android send a stream of images to the Controller Android, we can effectively “watch” where the vehicle is being moved.

An Android-powered smart phone is mounted to the front of the car, providing the vehicle with a rich collection of high-performance sensors and essentially, a global communications endpoint. The on-board smart phone runs a mobile application to read data from the various sensors and communicates this information to the controller phone over a Bluetooth connection. The sensors used are a camera for obtaining visual data, an orientation sensor to measure the car's heading, a 3-axis accelerometer to measure the car's acceleration, and a GPS for determining the car's geolocation, speed, and altitude. The mobile application also provides access to the camera's dual-LED flash, which can be used as car headlights in low-light conditions.

Utilizing the on-board phone's sensors, the car becomes “smarter” by knowing its position, velocity, acceleration, and orientation. This information is extremely useful for making precise movements in applications such as autonomous navigation. The sensors have variable accuracy and sampling rates, making them extremely flexible for application requirements. Pressing the “Camera” button on the UI will command a picture to be taken and transmitted at 2Hz. This crude form of video streaming can be used for remote surveillance or visual confirmation by transmitting the image to the controller phone for display.

Below is a figure of the Controller Android’s User Interface (Note that vehicle would be moving slightly forward and turning moderately left. Image shown is simulated; during operation, actual video would be shown from camera aboard vehicle):

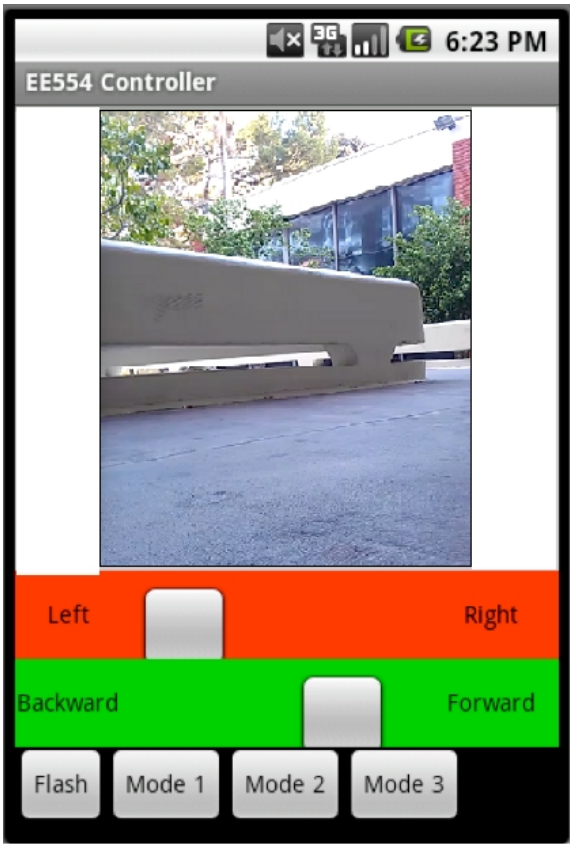


Figure 4. User Interface on Controller Android

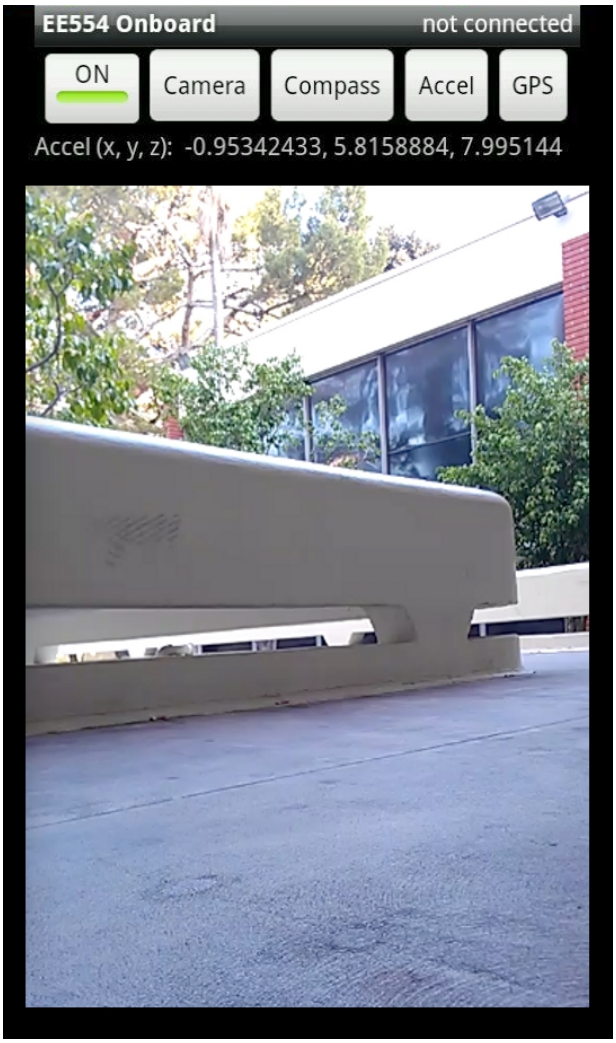


Figure 5. Application on Camera Android

On the user interface (Figure 4), there are also four buttons at the bottom of the interface:

- Flash – Toggles the flash on the Camera Android for enhanced visibility during nighttime operations.
- Mode 1 – Place vehicle into unmanned vehicle mode (Drive vehicle using Controller Android)
- Mode 2 – Place vehicle into obstacle avoidance mode (Vehicle will continue to drive forward until an obstacle obstructs vehicle's path.)

- Mode 3 – Drafting mode (Vehicle will maintain a fixed distance to other vehicle directly in front.)

This section primarily deals with Mode 1 – unmanned vehicle mode. These buttons are included as part of the Controller Android user interface in order to switch modes, and the protocol for telling the vehicle to switch modes is described below, however, the operation of Mode 2 and Mode 3 are described elsewhere in this report. Note that the vehicle only responds to motor controls from Controller Android when the vehicle is in Mode 1 or Mode 2 Mode 3 will not respond to motor commands from Controller Android's user interface.

Below we go into further details of Requirement 1 and 2, including data stream protocol and source code.

Requirement 1 – Motor Control (and Switching Between Modes 1, 2, and 3)

In order to remotely drive the vehicle through the Controller Android, we must have access to the motor controls aboard the vehicle. Since the Arduino microcontroller has access to the motor controls, we must send commands from the Controller Android to the Arduino microcontroller. To do this, we use a Bluetooth serial connection between the Controller Android and the Arduino microcontroller. For all this to work properly, the Controller Android and Arduino microcontroller must have an agreed upon protocol for communication motor control states, which are described next.

The vehicle has two motors, one for moving forward/backward and one for steering left/right. We represent the state of each motor on an integer scale of 0 to 10. In other words, the state will be represented by two integers (one for each motor), and each can be in a state of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, or 10.

For the forward/backward motor:

- A value of 0 through 4 represents motor is in reverse. (0 being full speed reverse, 4 being slowest speed in reverse)
- A value of 5 represents motor is stopped.
- A value of 6 through 10 represents motor is in forward. (6 being slowest speed forward, 10 being full speed forward)

For the steering motor:

- A value of 0 through 4 represents steering to the left. (0 being farthest left, 4 being slightly
- A value of 5 represents steering is dead center.
- A value of 6 through 10 represents steering is to the right. (6 being slightly right, 10 being farthest right)

When the user of the Controller Android wishes to change the state of the motors, such as steer more left or increase forward speed, the user simply drags the appropriate slidebar (see picture of user interface above). The Controller Android application will sense the user's change, and update the internal state within the Android's memory. The Controller Android will then communicate the updated state to the Arduino microcontroller via the Bluetooth connection, which will then update the motor controls appropriately.

The state will be sent over the Bluetooth serial stream using the following protocol: 'a' followed by a byte for the steering state (integer from 0 to 10), 'd' followed by a bye for throttle state (integer from 0 to 10). For example, if we were currently in steering state 5 (dead center) and throttle state 6 (slightly forward), we would send the following over the serial stream:

{'a', 0x05, 'd', 0x06}

Note: Since we are sending a serial stream of bytes upon each update, we must ensure that we do not reach an overrun. In this case, we have a human making updates to the state through the application's user interface. We can reasonably estimate that a human will make a state update not more than one time per second. However, to be completely sure, let's assume that the human user can make a state change every millisecond, or 1000 state updates per second.

The output stream is through a Bluetooth socket at a rate of 57,600bps. Each status update is 4 bytes, or 32 bits. This means that we can send 57,600 bps / 32 bits per state = 1800 states per second.

Our maximum number of user state updates is 1000 per second, while our Bluetooth socket can handle up to 1800 state updates per second. **No overrun is possible.**

To switch between Mode 1, Mode 2, and Mode 3, the user would simply press the appropriate button. Controller Android would then send a message over the Bluetooth serial stream to instruct the Arduino to switch modes. The following message would be sent: 'm' followed by the requested mode. For example, if we press the Mode 2 button, the following message would be sent:

{'m', 0x02}

To help incorporate Bluetooth communication between an Android device and an Arduino microcontroller, we used the Amarino library. The Amarino toolkit is an open source project used to easily integrate Android devices and Arduino microcontrollers. More can be found at: http://www.amarino-toolkit.net/index.php/home.html

We used the following functions from the Amarino library:

- Amarino.connect – Used to initiate connection between Android and Arduino.
- Amarino.sendDataToArduino – Used to send data from Android to Arduino.

- Amarino.disconnect – Used to close connection between Android and Arduino

Our source code can be found in Appendix A.

Requirement 2 – Video Stream

Aside from motor control (which allows us to drive the vehicle), we also require a video stream in order to see where the vehicle is heading. To do this, we mount the Camera Android phone aboard the front of the vehicle, and we have the camera face the forward path of the vehicle. The Camera Android, upon connecting to the Controller Android via a serial Bluetooth socket, will begin sending a video stream. The video stream will consist of JPEG images sent one after another at a predetermined rate.

For the Camera Android and Controller Android to communicate properly over the serial Bluetooth stream, we must have a common protocol that each follows. We have decided upon the following protocol:

- There are two control bytes:
    - START (0x12) – used to indicate start of image data
    - END (0x23) – used to indicate end of image data.
- Each control byte will be headed with the **C**ONTROL **B**YTE **H**EADER (CBH -- 0x00)
- Each data byte will be headed with the **D**ATA **B**YTE **H**EADER (DBH – 0xFE)

For example, if the image data consists of the following three bytes: {0x44, 0x55, 0x66}, then the actual message sent would be:

{CBH,START,DBH,0x44,DBH,0x55,DBH,0x66,CBH,END}

{0x00,0x12,0xFE,0x44,0xFE,0x55,0xFE,0x66,0x00,0x23}

Notice that this protocol has to send (2n + 4) bytes total, where n is the number of bytes in the our Android applications. More about this project can be found at:

http://developer.android.com/resources/samples/BluetoothChat/index.html

We used the following functions from the BluetoothChatService library:

- BluetoothChatService.connect – Used to initiate connection between two Android devices.
- BluetoothChatService.write – Used to send data from Android to other Android device.
- BluetoothChatService.ConnectedThread – Thread within BluetoothChatService which reads received data into a buffer.

Our source code can be found in Appendix A.

Note: Since we are sending a serial stream of bytes upon each image frame, we must ensure that we do not reach an overrun. For example, if we try and send images at a rate faster than the Bluetooth socket can handle, we will reach an overrun. This will cause a jittery and unreliable video stream as the buffer overruns and drops frames.

Our video stream Bluetooth socket functions at a maximum throughput rate of 0.7 Mbps. Each image is roughly 100 KB, or 800 Kb. This means that we can send 0.7 Mbps / 800 Kb per image = 0.875 images per second. In other words, we can send one image roughly every 1.15 seconds. Another feature we integrated is the "Flash" button, which will toggle the flash on the Camera Android. When the user presses the "Flash" button, the Controller Android program simply sends the bytes {0x01, 0x02} to the Camera Android. Upon receiving these bytes, the Camera Android application simply toggles the flash between on and off.

D. Motor Control

The motor control of the SMART car is a critical real-time aspect of the system. Its accurate control is essential to SMART car features such as collision avoidance and drafting.

The first step of developing the motor control for the SMART car was to allow bidirectional current flow for multi-directional driving e.g. forward, reverse, left, and right. A L293NE quadruple half H-bridge IC was used to allow bidirectional current flow as well as the disabling of motors to come to a rolling stop. The schematic and diagram of connecting the motors, L293, and arduino together is shown in the figures below.

After establishing reverse current through the various motors, the speed of the motors was refined and processing of commands over bluetooth was added. In order to control the speed of the motors, pulse width modulation was used to vary the voltage supplied to the motors. Although there are 256 possible values that the Arduino library allows for pulse width modulation, not all values could correspond to a motor value. The flag 'a' and 'd' correspond to the front and rear motor respectively, while the motor values could range from 1-10 to indicate speed and direction. For the front motor, values less than 5 correspond to turning left while values greater than 5 correspond to turning right. However, because of the mechanics of the car, pulse width modulation of the front motor could not be refined and controlled as much as the rear motor to allow varying degrees of turning. The same motor value concept applies to the rear motor where values less than 5 correspond to reversing while values greater than 5 correspond to moving forward.

The most real-time critical aspect of the motor control is the ability of the car to brake in time to avoid objects. Initially for collision avoidance, the motors were simply disabled when the sensor detected an object. However, disabling the motors didn't meet the car's deadline to come to a complete stop to avoid hitting the object. Rather, to come to a complete stop quicker, the

maximum voltage, VCC, would be applied in the reverse direction for 1 millisecond. Although this solved our initial problems, certain conditions were not met, such as when the car is running at full speed. As a result, the braking function was modified to vary the duration of reverse voltage as a function of the previous direction and speed. Since the motors we used were not digital motors and their state could not be read, a flag was used to indicate which direction the car was previously moving in, forward or reverse. This flag was based on the last motor value received from the bluetooth command stream. By determining the previous direction of the car, the direction that the reverse voltage needed to be applied was known. In addition, from this flag, the duration that the reverse voltage needed to be applied was also known. By increasing the duration 125 milliseconds for every step up in speed the car could come to stop within the distance the sensor sets as its danger threshold.

This concept of collision avoidance was carried over to the SMART car's drafting functionality. Based on new flags set by the front sensor, the car accelerates, decelerates, or comes to a stop in order to maintain a constant distance from an object in front of it. When the object is at the specified distance from the front of the car, the car accelerates given that the object in front moves in unison with it. If the car became was closer than the specified distance, the car would decelerate. Finally, if the object was too close, the car would come to a complete stop. By implementing this function, the SMART car can demonstrate drafting which is useful for saving fuel in commercial vehicles.

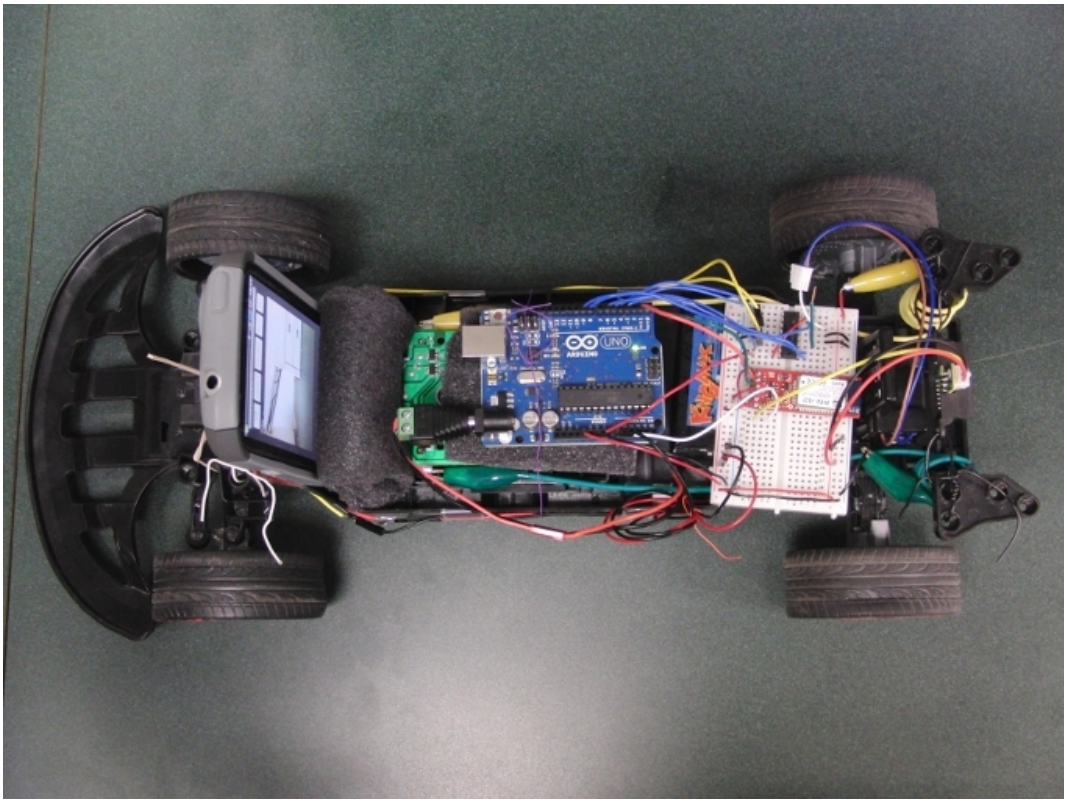
V. Results

We were able to successfully implement a car that can automatically avoid collisions from the front or back, maintain a constant distance while following another car, and allow

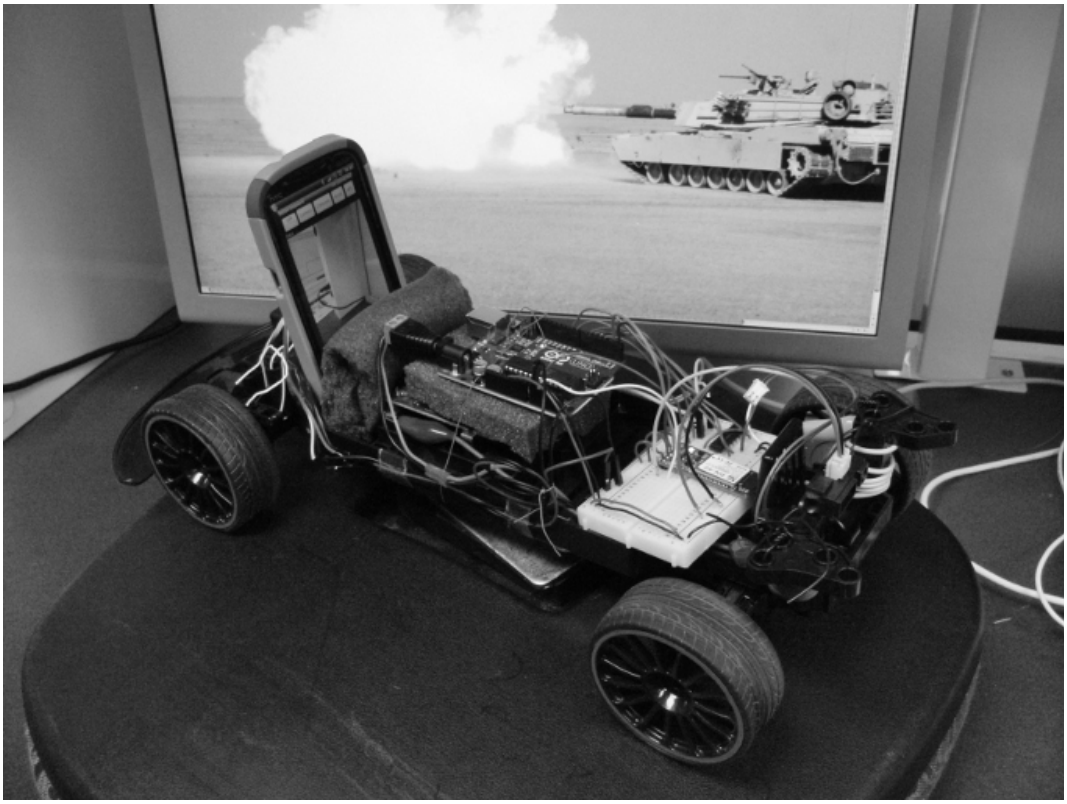
manual control override by a remote user through a graphical user interface giving real-time image data from an on-board camera.

A real-life application of automatic collision avoidance is illustrated in certain vehicles that brake automatically for pedestrians while reversing. In addition, by implementing the drafting feature into commercially available vehicles, traffic congestion can be reduced by allowing better traffic flow through normally congested highways.

This is the final result of our car:



This is our car in action:



VI. Future Work

Because of time and resource constraints, we could not implement all the features we had thought of. However, our project establishes a base of features that can be further developed for various applications. By implementing a more complex array of sensors on the car, we could have developed a “lane changing assistance” feature that could alert the driver if a vehicle is in a blindspot. Furthermore, by utilizing the various sensor data from the camera android in conjunction with a navigation software like Google Maps, an autonomously driving vehicle could also be implemented. Finally, remote operation of the vehicle with higher frame-rate video streaming from the car can have a military application as an unmanned vehicle in combat zones.

VII. References

1. http://news.drive.com.au/drive/motor-news/new-technology-could-end-driveway-crashes-20100802-111um.html
2. http://www.amarino-toolkit.net/tl_files/doc/index.htm
3. http://arduino.cc/en/Reference/HomePage
4. http://www.sparkfun.com/tutorials/67
5. http://developer.android.com/guide/topics/wireless/bluetooth.html

VIII. Appendices

Appendix A - Source Code for Controller Android Application

EE554.java (main program)

```
package ee554;

import java.io.File;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
```

```
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import at.abraxas.amarino.Amarino;
import ee554.controller.R;

public class EE554 extends Activity implements OnSeekBarChangeListener, OnClickListener {


        private static final String TAG = "EE554";

        private static final String DEVICE_ADDRESS_MICRO = "00:06:66:46:5D:94";

        private static final String DEVICE_ADDRESS_PHONE = "A8:26:D9:01:34:B1";

        private BluetoothChatService mChatService = null;

        SeekBar forwardbackward, leftright;
        ImageView image;
        Button buttonFlash, buttonMode1, buttonMode2, buttonMode3;

        BluetoothAdapter mBluetoothAdapter = null;
        BluetoothDevice mBluetoothDevice;

        int stateLeftRight, stateForwardBackward;

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

        forwardbackward = (SeekBar) findViewById(R.id.SeekBarForwardBackward);
        leftright = (SeekBar) findViewById(R.id.SeekBarLeftRight);
        image = (ImageView) findViewById(R.id.imageView1);
        buttonFlash = (Button) findViewById(R.id.flash);
        buttonMode1 = (Button) findViewById(R.id.mode1);
        buttonMode2 = (Button) findViewById(R.id.mode2);
        buttonMode3 = (Button) findViewById(R.id.mode3);

        // register listeners
                forwardbackward.setOnSeekBarChangeListener(this);
                leftright.setOnSeekBarChangeListener(this);
                buttonFlash.setOnClickListener(this);
                buttonMode1.setOnClickListener(this);
                buttonMode2.setOnClickListener(this);
                buttonMode3.setOnClickListener(this);

                image.setImageResource(R.drawable.btn_square_overlay_normal);

        // connect to Amarino
                Amarino.connect(this, DEVICE_ADDRESS_MICRO);

                // connect to phone
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
                mBluetoothDevice = mBluetoothAdapter.getRemoteDevice(DEVICE_ADDRESS_PHONE);
```

```
}

public void onClick(View v)
{
	if (v == buttonFlash)
	{
		byte[] out = {0x01, 0x02};
		mChatService.write(out);
		Log.d(TAG, "toggle flash" + out[0] + out[1]);
	}
	if (v == buttonMode1)
	{
		Amarino.sendDataToArduino(this, DEVICE_ADDRESS_MICRO, 'm', 1);
		Log.d(TAG, "Mode 1");
	}
	if (v == buttonMode2)
	{
		Amarino.sendDataToArduino(this, DEVICE_ADDRESS_MICRO, 'm', 2);
		Log.d(TAG, "Mode 2");
	}
	if (v == buttonMode3)
	{
		Amarino.sendDataToArduino(this, DEVICE_ADDRESS_MICRO, 'm', 3);
		Log.d(TAG, "Mode 3");
	}
}


	@Override
	protected void onStart() {
		super.onStart();

		stateForwardBackward = 5;
		forwardbackward.setProgress(stateForwardBackward);

		stateLeftRight = 5;
		leftright.setProgress(stateLeftRight);

  new Thread(){
	public void run(){
		try {
					Thread.sleep(6000);
				} catch (InterruptedException e) {}
				Log.d(TAG, "init state");
				updateState();
	}
  }.start();

  // Initialize the BluetoothService to perform bluetooth connections
  mChatService = new BluetoothChatService(this);
  mChatService.connect(mBluetoothDevice);

  new Thread(){
	File imgFile = new File ("/sdcard/image.jpg");
	public void run(){
		Bitmap bm = BitmapFactory.decodeFile(imgFile.getAbsolutePath());
		image.setImageBitmap(bm);
	}
  }.start();
```

```
    }


        @Override
        protected void onStop() {
                super.onStop();

                // stop Amarino's background service, we don't need it any more
                Amarino.disconnect(this, DEVICE_ADDRESS_MICRO);
        }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Stop the Bluetooth chat services
        if (mChatService != null) mChatService.stop();
        Log.d(TAG, "--- ON DESTROY ---");
    }


        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
                updateState();
        }



        private void updateState() {

                stateForwardBackward = forwardbackward.getProgress();
                stateLeftRight = leftright.getProgress();

                Log.d(TAG, "update state");
                Log.d(TAG, String.format("    lr: %d",stateLeftRight));
                Log.d(TAG, String.format("    fb: %d",stateForwardBackward));

                // Send left/right
                Amarino.sendDataToArduino(this, DEVICE_ADDRESS_MICRO, 'a', stateLeftRight);

                // Send forward/backward
                Amarino.sendDataToArduino(this, DEVICE_ADDRESS_MICRO, 'd', stateForwardBackward);

        }

        public void onStartTrackingTouch(SeekBar seekBar) {

        }

        public void onStopTrackingTouch(SeekBar seekBar) {

        }

}
```

BluetoothService.java (Bluetooth Support Library, Library based off of Android Open Source Project – BluetoothChat Sample Program)

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

```
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package ee554;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.util.Log;
import android.widget.ImageView;

/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 * incoming connections, a thread for connecting with a device, and a
 * thread for performing data transmissions when connected.
 */
class BluetoothChatService {
    // Debugging
    private static final String TAG = "BluetoothChatService";
    private static final boolean D = true;

    // Name for the SDP record when creating server socket
    private static final String NAME = "EE554";

    // Unique UUID for this application
    private static final UUID MY_UUID = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");

    // Member fields
    private final BluetoothAdapter mAdapter;
    private AcceptThread mAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0;       // we're doing nothing
    public static final int STATE_LISTEN = 1;     // now listening for incoming connections
    public static final int STATE_CONNECTING = 2; // now initiating an outgoing connection
    public static final int STATE_CONNECTED = 3;  // now connected to a remote device

    /**
```

```
 * Constructor. Prepares a new BluetoothChat session.
 * @param context  The UI Activity Context
 * @param handler  A Handler to send messages back to the UI Activity
 */
public BluetoothChatService(Context context) {
    mAdapter = BluetoothAdapter.getDefaultAdapter();
    mState = STATE_NONE;

}

/**
 * Set the current state of the chat connection
 * @param state  An integer defining the current connection state
 */
private synchronized void setState(int state) {
    if (D) Log.d(TAG, "setState() " + mState + " -> " + state);
    mState = state;

}

/**
 * Return the current connection state. */
public synchronized int getState() {
    return mState;
}

/**
 * Start the chat service. Specifically start AcceptThread to begin a
 * session in listening (server) mode. Called by the Activity onResume() */
public synchronized void start() {
    if (D) Log.d(TAG, "start");

    // Cancel any thread attempting to make a connection
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Start the thread to listen on a BluetoothServerSocket
    if (mAcceptThread == null) {
        mAcceptThread = new AcceptThread();
        mAcceptThread.start();
    }
    setState(STATE_LISTEN);
}

/**
 * Start the ConnectThread to initiate a connection to a remote device.
 * @param device  The BluetoothDevice to connect
 */
public synchronized void connect(BluetoothDevice device) {
    if (D) Log.d(TAG, "connect to: " + device);

    // Cancel any thread attempting to make a connection
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    }

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
```

```
    // Start the thread to connect with the given device
    mConnectThread = new ConnectThread(device);
    mConnectThread.start();
    setState(STATE_CONNECTING);
}

/**
 * Start the ConnectedThread to begin managing a Bluetooth connection
 * @param socket  The BluetoothSocket on which the connection was made
 * @param device  The BluetoothDevice that has been connected
 */
public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {
    if (D) Log.d(TAG, "connected");

    // Cancel the thread that completed the connection
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Cancel any thread currently running a connection
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Cancel the accept thread because we only want to connect to one device
    if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}

    // Start the thread to manage the connection and perform transmissions
    mConnectedThread = new ConnectedThread(socket);
    mConnectedThread.start();

    setState(STATE_CONNECTED);
}

/**
 * Stop all threads
 */
public synchronized void stop() {
    if (D) Log.d(TAG, "stop");
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}
    setState(STATE_NONE);
}

/**
 * Write to the ConnectedThread in an unsynchronized manner
 * @param out The bytes to write
 * @see ConnectedThread#write(byte[])
 */
public void write(byte[] out) {
    // Create temporary object
    ConnectedThread r;
    // Synchronize a copy of the ConnectedThread
    synchronized (this) {
        if (mState != STATE_CONNECTED) return;
        r = mConnectedThread;
    }
    // Perform the write unsynchronized
    r.write(out);
}

/**
 * Indicate that the connection attempt failed and notify the UI Activity.
 */
```

```
private void connectionFailed() {
    setState(STATE_LISTEN);
}

/**
 * Indicate that the connection was lost and notify the UI Activity.
 */
private void connectionLost() {
    setState(STATE_LISTEN);

}

/**
 * This thread runs while listening for incoming connections. It behaves
 * like a server-side client. It runs until a connection is accepted
 * (or until cancelled).
 */
private class AcceptThread extends Thread {
    // The local server socket
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        BluetoothServerSocket tmp = null;

        // Create a new listening server socket
        try {
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "listen() failed", e);
        }
        mmServerSocket = tmp;
    }

    public void run() {
        if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
        setName("AcceptThread");
        BluetoothSocket socket = null;

        // Listen to the server socket if we're not connected
        while (mState != STATE_CONNECTED) {
            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                Log.e(TAG, "accept() failed", e);
                break;
            }

            // If a connection was accepted
            if (socket != null) {
                synchronized (BluetoothChatService.this) {
                    switch (mState) {
                    case STATE_LISTEN:
                    case STATE_CONNECTING:
                        // Situation normal. Start the connected thread.
                        connected(socket, socket.getRemoteDevice());
                        break;
                    case STATE_NONE:
                    case STATE_CONNECTED:
                        // Either not ready or already connected. Terminate new socket.
```

```
                    try {
                        socket.close();
                    } catch (IOException e) {
                        Log.e(TAG, "Could not close unwanted socket", e);
                    }
                    break;
                }
            }
        }
    }
    if (D) Log.i(TAG, "END mAcceptThread");
  }

  public void cancel() {
    if (D) Log.d(TAG, "cancel " + this);
    try {
      mmServerSocket.close();
    } catch (IOException e) {
      Log.e(TAG, "close() of server failed", e);
    }
  }
}


/**
 * This thread runs while attempting to make an outgoing connection
 * with a device. It runs straight through; the connection either
 * succeeds or fails.
 */
private class ConnectThread extends Thread {
  private final BluetoothSocket mmSocket;
  private final BluetoothDevice mmDevice;

  public ConnectThread(BluetoothDevice device) {
    mmDevice = device;
    BluetoothSocket tmp = null;

    // Get a BluetoothSocket for a connection with the
    // given BluetoothDevice
    try {
      tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
    } catch (IOException e) {
      Log.e(TAG, "create() failed", e);
    }
    mmSocket = tmp;
  }

  public void run() {
    Log.i(TAG, "BEGIN mConnectThread");
    setName("ConnectThread");

    // Always cancel discovery because it will slow down a connection
    mAdapter.cancelDiscovery();

    // Make a connection to the BluetoothSocket
    try {
      // This is a blocking call and will only return on a
      // successful connection or an exception
      mmSocket.connect();
    } catch (IOException e) {
      connectionFailed();
```

```
            // Close the socket
            try {
                mmSocket.close();
            } catch (IOException e2) {
                Log.e(TAG, "unable to close() socket during connection failure", e2);
            }
            // Start the service over to restart listening mode
            BluetoothChatService.this.start();
            return;
        }

        // Reset the ConnectThread because we're done
        synchronized (BluetoothChatService.this) {
            mConnectThread = null;
        }

        // Start the connected thread
        connected(mmSocket, mmDevice);
    }

    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "close() of connect socket failed", e);
        }
    }
}

/**
 * This thread runs during a connection with a remote device.
 * It handles all incoming and outgoing transmissions.
 */
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        Log.d(TAG, "create ConnectedThread");
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e(TAG, "temp sockets not created", e);
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        Log.i(TAG, "BEGIN mConnectedThread");
        byte[] buffer = new byte[1024000];
             int count = 0;
        int bytes;
```

```
    FileOutputStream imageOut;

char nextCh;
boolean currentImage = false;

// Keep listening to the InputStream while connected
while (true) {
   try {
      // Read from the InputStream
      //bytes = mmInStream.read(buffer);
     // Read from the InputStream
                        nextCh = (char) mmInStream.read();
                                //Log.d("BYTE", "Byte received");
                                //for(int i=0; i<count; i++)
                                //{
                                        //Log.d("BYTE", Byte.toString((byte)nextCh));
                                //}

                        //Check if nextCh is control byte or data byte
                        if (nextCh == 0x00)
                        {
                                //Control Byte
                                // Check if next byte is START or END
                                nextCh = (char) mmInStream.read();
                                Log.d("BYTE", "Control byte received");
                                Log.d("BYTE", Byte.toString((byte)nextCh));
                                if (nextCh == 0x12)
                                {
                                        //Start byte
                                        // Start tracking image
                                        currentImage = true;
                                        // Reset number of bytes in buffer
                                        count = 0;

                                }
                                else if (nextCh == 0x23)
                                {

                                        //End byte
                                        // Save current image buffer as output file
                                        //imageOut = openFileOutput("image.jpg", 1);
                                        //imageOut.write(buffer, 0, count);
                                        //imageOut.close();
                                        currentImage = false;

                                        Log.d("PIC", "Pic received");
                                        Log.d("PIC", "Count = " + count);
                                        for(int i=0; i<count; i++)
                                        {
                                                Log.d("PIC", Byte.toString(buffer[i]));
                                        }

                                        //Set new picture
                                try {
                                                File myFile = new File("/sdcard/image.jpg");
                                                myFile.createNewFile();
                                                FileOutputStream fOut = new FileOutputStream(myFile);
                                                OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
                                                for (int i =0; i<buffer.length; i++)
                                                        myOutWriter.write((char)buffer[i]);
```

```
                                        myOutWriter.flush();
                                        myOutWriter.close();
                                        fOut.close();
                                } catch (Exception e) {}


                        }
                        else
                        {
                                //Unknown byte, stop tracking image
                                currentImage = false;
                        }

                }
                else if (nextCh == 0xFE)
                {
                        //Data Byte
                        nextCh = (char) mmInStream.read();
                        Log.d("BYTE", "Data byte received");
                        Log.d("BYTE", Byte.toString((byte)nextCh));
                        // If tracking current image, save data byte to current image file buffer
                        if (currentImage == true)
                        {
                                // Add character to buffer
                                buffer[count] = (byte) nextCh;
                                // Increase number of character count
                                count++;
                        }

                }
                else
                {
                        // Unknown byte, stop tracking image
                        currentImage = false;
                }

            // Send the obtained bytes to the UI Activity
            // mHandler.obtainMessage(BluetoothChat.MESSAGE_READ, bytes, -1, buffer)
            //       .sendToTarget();
        } catch (IOException e) {
            Log.e(TAG, "disconnected", e);
            connectionLost();
            break;
        }
    }
}

                /**
 * Write to the connected OutStream.
 * @param buffer  The bytes to write
 */
public void write(byte[] buffer) {
    try {
        mmOutStream.write(buffer);

        // Share the sent message back to the UI Activity
        //mHandler.obtainMessage(BluetoothChat.MESSAGE_WRITE, -1, -1, buffer)
        //       .sendToTarget();
    } catch (IOException e) {
        Log.e(TAG, "Exception during write", e);
```

```
            }
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }
}
```

Appendix B - Source Code for Camera Android Application

```
package com.example.android.BluetoothChat;

import java.io.IOException;

import android.content.Context;
import android.hardware.Camera;
import android.hardware.Camera.PreviewCallback;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

class Preview extends SurfaceView implements SurfaceHolder.Callback {
  private static final String TAG = "Preview";

  SurfaceHolder mHolder;
  public Camera camera;

  Preview(Context context) {
    super(context);

    // Install a SurfaceHolder.Callback so we get notified when the
    // underlying surface is created and destroyed.
    mHolder = getHolder();
    mHolder.addCallback(this);
    mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
  }

  // Called once the holder is ready
  public void surfaceCreated(SurfaceHolder holder) {
    // The Surface has been created, acquire the camera and tell it where
    // to draw.
    camera = Camera.open();
    camera.setDisplayOrientation(90);      //set preview orientation to portrait

    try {
      camera.setPreviewDisplay(holder);

      camera.setPreviewCallback(new PreviewCallback() {
        // Called for each frame previewed
        public void onPreviewFrame(byte[] data, Camera camera) {
          //Log.d(TAG, "onPreviewFrame called at: " + System.currentTimeMillis());
          Preview.this.invalidate();
        }
      });
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  // Called when the holder is destroyed
  public void surfaceDestroyed(SurfaceHolder holder) {
              if (camera != null) {
                        camera.stopPreview();
                        camera.release();
                        camera = null;
    }
  }

  // Called when holder has changed
  public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
```

```
            camera.startPreview();
    }

}
```

```
/*              ONBOARD CAMERA ANDROID BLUETOOTH JAVA CODE
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.example.android.BluetoothChat;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.hardware.SensorEventListener;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.content.Context;
import android.hardware.Camera;
import android.hardware.Camera.AutoFocusCallback;
import android.hardware.Camera.Parameters;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.widget.FrameLayout;
import android.widget.ToggleButton;

/**
```

```
 * This is the main Activity that displays the current chat session.
 */
public class BluetoothChat extends Activity implements SensorEventListener {
	// Debugging
	private static final String TAG = "BluetoothChat";
	private static final boolean D = true;

	// Message types sent from the BluetoothChatService Handler
	public static final int MESSAGE_STATE_CHANGE = 1;
	public static final int MESSAGE_READ = 2;
	public static final int MESSAGE_WRITE = 3;
	public static final int MESSAGE_DEVICE_NAME = 4;
	public static final int MESSAGE_TOAST = 5;

	// Key names received from the BluetoothChatService Handler
	public static final String DEVICE_NAME = "device_name";
	public static final String TOAST = "toast";

	// Intent request codes
	private static final int REQUEST_CONNECT_DEVICE = 1;
	private static final int REQUEST_ENABLE_BT = 2;

	// Layout Views
	private TextView mTitle;
	private ListView mConversationView;
	private EditText mOutEditText;
	private Button mSendButton;

	// Name of the connected device
	private String mConnectedDeviceName = null;
	// Array adapter for the conversation thread
	private ArrayAdapter<String> mConversationArrayAdapter;
	// String buffer for outgoing messages
	private StringBuffer mOutStringBuffer;
	// Local Bluetooth adapter
	private BluetoothAdapter mBluetoothAdapter = null;
	// Member object for the chat services
	private BluetoothChatService mChatService = null;

	// Globals
	Preview preview;
	ToggleButton lightBtn;
	Button cameraBtn;
	Button compassBtn;
	Button accelBtn;
	Button gpsBtn;
	TextView postText;
	SensorManager mSensorManager = null;
	LocationManager locationManager = null;
	float acc_X, acc_Y, acc_Z, com_X, com_Y, com_Z;
	int camera_flag = 0, compass_flag = 0, accel_flag = 0, gps_flag = 0, camera_init = 0, light_flag = 0;
	Thread camera_thread = null;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		if (D)
			Log.e(TAG, "+++ ON CREATE +++");

		// Set up the window layout
		requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
```

```
		setContentView(R.layout.main);
		getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
				R.layout.custom_title);

		// Set up the custom title
		mTitle = (TextView) findViewById(R.id.title_left_text);
		mTitle.setText(R.string.app_name);
		mTitle = (TextView) findViewById(R.id.title_right_text);

		// Get local Bluetooth adapter
		mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

		// If the adapter is null, then Bluetooth is not supported
		if (mBluetoothAdapter == null) {
			Toast.makeText(this, "Bluetooth is not available",
					Toast.LENGTH_LONG).show();
			finish();
			return;
		}
	}

	public void buttonActions() {
		lightBtn.setOnClickListener(new OnClickListener() {
			public void onClick(View v) {
				if (lightBtn.isChecked()) {
					lightBtn.setChecked(true);
					Camera.Parameters camParam = preview.camera.getParameters();
					camParam.setFlashMode("torch");
					preview.camera.setParameters(camParam);
					light_flag = 1;
				} else {
					Camera.Parameters camParam = preview.camera.getParameters();
					camParam.setFlashMode("off");
					preview.camera.setParameters(camParam);
					light_flag = 0;
				}
			}
		});
		cameraBtn.setOnClickListener(new OnClickListener() {
			public void onClick(View v) {
				camera_flag = (camera_flag == 0) ? 1 : 0;
/*
				if (camera_flag == 1 && camera_thread == null && camera_init == 0) {
					camera_thread = new Thread() {
					   @Override
					   public void run() {
						   while (true) {
							   preview.camera.autoFocus(new AutoFocusCallback() {
								   @Override
								   public void onAutoFocus(boolean
success, Camera camera) {
								   Camera.Parameters camParam =
camera.getParameters();
								   camParam.setZoom(0);

camParam.setFocusMode(Parameters.FOCUS_MODE_AUTO);
								   camera.setParameters(camParam);
								   }
							   });
							   preview.camera.takePicture(shutterCallback,
rawCallback, jpegCallback);
```

```
                                                                try { Thread.sleep(1000); }
                                                                catch (Exception e) {}
                                                        }
                                                    }
                                                };
                                                camera_thread.start();
                                                camera_init = 1;
                                        }
*/
/*
                                        if (camera_flag == 0) {// && camera_thread != null) {
                                                Thread temp_thread = camera_thread;
                                           camera_thread = null;
                                           temp_thread.interrupt();
                                        }
*/
                                        preview.camera.autoFocus(new AutoFocusCallback() {
                                                @Override
                                                public void onAutoFocus(boolean success, Camera camera) {
                                                Camera.Parameters camParam = camera.getParameters();
                                                camParam.setZoom(0);
                                                camParam.setPictureSize(150, 150);
                                                camParam.setFocusMode(Parameters.FOCUS_MODE_AUTO);
                                                camera.setParameters(camParam);
                                                }
                                         });
                                         preview.camera.takePicture(shutterCallback, rawCallback, jpegCallback);
                        }
                });
                compassBtn.setOnClickListener(new OnClickListener() {
                        public void onClick(View v) {
                                compass_flag = 1;
                                accel_flag = 0;
                                gps_flag = 0;
                        }
                });
                accelBtn.setOnClickListener(new OnClickListener() {
                        public void onClick(View v) {
                                accel_flag = 1;
                                compass_flag = 0;
                                gps_flag = 0;
                        }
                });
                gpsBtn.setOnClickListener(new OnClickListener() {
                        public void onClick(View v) {
                                gps_flag = 1;
                                compass_flag = 0;
                                accel_flag = 0;
                                LocationListener ll = new mylocationlistener();
                                locationManager.requestLocationUpdates(
                                                LocationManager.GPS_PROVIDER, 0, 0, ll);
                                locationManager.requestLocationUpdates(
                                                LocationManager.NETWORK_PROVIDER, 0, 0, ll);
                        }
                });
        }

        // Handles data for jpeg picture
        PictureCallback jpegCallback = new PictureCallback() {
                public void onPictureTaken(byte[] data, Camera camera) {
                        byte[] dataStream = new byte[150000];
```

```
                int counter = 0;
                try {
                        /*
                        data [0] = 0x01;
                        data [1] = 0x02;
                        data [2] = 0x03;
                        data [3] = 0x04;
                        data [4] = 0x05;
                        */
                        // add start bytes 0x0012
                        dataStream[counter++] = 0x00;
                        dataStream[counter++] = 0x12;

                        // for each data byte
                        for (int i = 0; i < data.length; i++) {
                                dataStream[counter++] = (byte) 0xfe; // add 0xFF
                                dataStream[counter++] = data[i]; // add data byte
                        }

                        // add end bytes 0x0023
                        dataStream[counter++] = 0x00;
                        dataStream[counter++] = 0x23;


                        // send data
                        sendMessage(dataStream);

                        // Write to SD Card
                        // outStream = new FileOutputStream(String.format("/sdcard/RC-image.jpg"));
                        // outStream.write(data);
                } finally {
                }
                Log.d(TAG, "onPictureTaken - jpeg");
        }
};

// Called when shutter is opened
ShutterCallback shutterCallback = new ShutterCallback() {
        public void onShutter() {
                Log.d(TAG, "onShutter'd");
        }
};

// Handles data for raw picture
PictureCallback rawCallback = new PictureCallback() {
        public void onPictureTaken(byte[] data, Camera camera) {
                Log.d(TAG, "onPictureTaken - raw");
        }
};

class mylocationlistener implements LocationListener {
        @Override
        public void onLocationChanged(Location location) {
                if (location != null) {
                        double pLong = location.getLongitude();
                        double pLat = location.getLatitude();
                        if (gps_flag == 1) {
                                postText.setText("Lat, Long:  " + Double.toString(pLat)
                                                + ",  " + Double.toString(pLong));
                        }
                }
```

```
		}

		@Override
		public void onProviderDisabled(String provider) {
		}

		@Override
		public void onProviderEnabled(String provider) {
		}

		@Override
		public void onStatusChanged(String provider, int status, Bundle extras) {
		}
}

public void onSensorChanged(SensorEvent event) {
		synchronized (this) {
			switch (event.sensor.getType()) {
			case Sensor.TYPE_ACCELEROMETER:
				acc_X = event.values[0];
				acc_Y = event.values[1];
				acc_Z = event.values[2];
				if (accel_flag == 1) {
					postText.setText("Accel (x, y, z):  " + acc_X + ", "
							+ acc_Y + ", " + acc_Z);
				}
				break;
			case Sensor.TYPE_ORIENTATION:
				com_X = event.values[0];
				com_Y = event.values[1];
				com_Z = event.values[2];
				if (compass_flag == 1) {
					postText.setText("Orientation (degrees):  " + com_X);
				}
				break;

			}
		}
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) { }

@Override
public void onStart() {
		super.onStart();
		if (D)
			Log.e(TAG, "++ ON START ++");

		// If BT is not on, request that it be enabled.
		// setupChat() will then be called during onActivityResult
		if (!mBluetoothAdapter.isEnabled()) {
			Intent enableIntent = new Intent(
					BluetoothAdapter.ACTION_REQUEST_ENABLE);
			startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
			// Otherwise, setup the chat session
		} else {
			if (mChatService == null)
				setupChat();
		}
}
```

```
@Override
public synchronized void onResume() {
	super.onResume();
	if (D)
		Log.e(TAG, "+ ON RESUME +");

	// Performing this check in onResume() covers the case in which BT was
	// not enabled during onStart(), so we were paused to enable it...
	// onResume() will be called when ACTION_REQUEST_ENABLE activity
	// returns.
	if (mChatService != null) {
		// Only if the state is STATE_NONE, do we know that we haven't
		// started already
		if (mChatService.getState() == BluetoothChatService.STATE_NONE) {
			// Start the Bluetooth chat services
			mChatService.start();
		}
	}

	mSensorManager.registerListener(this,
			mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
			mSensorManager.SENSOR_DELAY_UI); // SENSOR_DELAY_FASTEST
	mSensorManager.registerListener(this,
			mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
			mSensorManager.SENSOR_DELAY_UI);
}

private void setupChat() {
	Log.d(TAG, "setupChat()");

	// Initialize the array adapter for the conversation thread
	mConversationArrayAdapter = new ArrayAdapter<String>(this,
			R.layout.message);
	mConversationView = (ListView) findViewById(R.id.in);
	mConversationView.setAdapter(mConversationArrayAdapter);

	// Initialize the compose field with a listener for the return key
	mOutEditText = (EditText) findViewById(R.id.edit_text_out);
	mOutEditText.setOnEditorActionListener(mWriteListener);

	// Initialize the send button with a listener that for click events
	mSendButton = (Button) findViewById(R.id.button_send);
	mSendButton.setOnClickListener(new OnClickListener() {
		public void onClick(View v) {
			// Send a message using content of the edit text widget
			TextView view = (TextView) findViewById(R.id.edit_text_out);
			String message = view.getText().toString();
			//sendMessage(message);
		}
	});

	// Initialize the BluetoothChatService to perform bluetooth connections
	mChatService = new BluetoothChatService(this, mHandler);

	// Initialize the buffer for outgoing messages
	mOutStringBuffer = new StringBuffer("");

	// Setup camera and sensors
	preview = new Preview(this);
	((FrameLayout) findViewById(R.id.preview)).addView(preview);
```

```
		lightBtn = (ToggleButton) findViewById(R.id.light_button);
		cameraBtn = (Button) findViewById(R.id.camera_button);
		compassBtn = (Button) findViewById(R.id.compass_button);
		accelBtn = (Button) findViewById(R.id.accel_button);
		gpsBtn = (Button) findViewById(R.id.gps_button);
		postText = (TextView) findViewById(R.id.data_text);
		mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
		locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

		buttonActions();
	}

	@Override
	public synchronized void onPause() {
		super.onPause();
		if (D)
			Log.e(TAG, "- ON PAUSE -");
		mSensorManager.unregisterListener(this,
				mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
		mSensorManager.unregisterListener(this,
				mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION));
		locationManager.removeUpdates((LocationListener) this);
		preview.camera.release();
		if (mChatService != null) mChatService.stop();
	}

	@Override
	public void onStop() {
		super.onStop();
		mSensorManager.unregisterListener(this,
				mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
		mSensorManager.unregisterListener(this,
				mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION));
		locationManager.removeUpdates((LocationListener) this);
		preview.camera.release();
		if (mChatService != null) mChatService.stop();
		if (D)
			Log.e(TAG, "-- ON STOP --");
	}

	@Override
	public void onDestroy() {
		super.onDestroy();
		// Stop the Bluetooth chat services
		if (mChatService != null)
			mChatService.stop();
		if (D)
			Log.e(TAG, "--- ON DESTROY ---");
	}

	private void ensureDiscoverable() {
		if (D)
			Log.d(TAG, "ensure discoverable");
		if (mBluetoothAdapter.getScanMode() != BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)
{
			Intent discoverableIntent = new Intent(
					BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
			discoverableIntent.putExtra(
					BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
			startActivity(discoverableIntent);
		}
```

```
	}

	/**
	 * Sends a message.
	 *
	 * @param message
	 *            A string of text to send.
	 */
	private void sendMessage(String message) {
		// Check that we're actually connected before trying anything
		if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
			Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT)
					.show();
			return;
		}

		// Check that there's actually something to send
		if (message.length() > 0) {
			// Get the message bytes and tell the BluetoothChatService to write
			byte[] send = message.getBytes();
			mChatService.write(send);

			// Reset out string buffer to zero and clear the edit text field
			mOutStringBuffer.setLength(0);
			mOutEditText.setText(mOutStringBuffer);
		}
	}
*/
	private void sendMessage(byte[] message) {
		// Check that we're actually connected before trying anything
		if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
			Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT)
					.show();
			return;
		}

		// Check that there's actually something to send
		//if (message.length() > 0) {
			// Get the message bytes and tell the BluetoothChatService to write
			//byte[] send = message.getBytes();
			mChatService.write(message);

			// Reset out string buffer to zero and clear the edit text field
			//mOutStringBuffer.setLength(0);
			//mOutEditText.setText(mOutStringBuffer);
		//}
	}

	// The action listener for the EditText widget, to listen for the return key
	private TextView.OnEditorActionListener mWriteListener = new TextView.OnEditorActionListener() {
		public boolean onEditorAction(TextView view, int actionId,
				KeyEvent event) {
			// If the action is a key-up event on the return key, send the
			// message
			if (actionId == EditorInfo.IME_NULL
					&& event.getAction() == KeyEvent.ACTION_UP) {
				String message = view.getText().toString();
				//sendMessage(message);
			}
			if (D)
				Log.i(TAG, "END onEditorAction");
```

```
                return true;
        }
};

// The Handler that gets information back from the BluetoothChatService
private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
                switch (msg.what) {
                case MESSAGE_STATE_CHANGE:
                        if (D)
                                Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
                        switch (msg.arg1) {
                        case BluetoothChatService.STATE_CONNECTED:
                                mTitle.setText(R.string.title_connected_to);
                                mTitle.append(mConnectedDeviceName);
                                mConversationArrayAdapter.clear();
                                break;
                        case BluetoothChatService.STATE_CONNECTING:
                                mTitle.setText(R.string.title_connecting);
                                break;
                        case BluetoothChatService.STATE_LISTEN:
                        case BluetoothChatService.STATE_NONE:
                                mTitle.setText(R.string.title_not_connected);
                                break;
                        }
                        break;
                case MESSAGE_WRITE:
                        byte[] writeBuf = (byte[]) msg.obj;
                        // construct a string from the buffer
                        String writeMessage = new String(writeBuf);
                        mConversationArrayAdapter.add("Me:  " + writeMessage);
                        break;
                case MESSAGE_READ:
                        byte[] readBuf = (byte[]) msg.obj;

                        //Toggle flash light
                        if(readBuf[0] == 0x02 && readBuf[1] == 0x01) {
                                if (light_flag == 0) {
                                        Camera.Parameters camParam = preview.camera.getParameters();
                                        camParam.setFlashMode("torch");
                                        preview.camera.setParameters(camParam);
                                        light_flag = 1;
                                } else {
                                        Camera.Parameters camParam = preview.camera.getParameters();
                                        camParam.setFlashMode("off");
                                        preview.camera.setParameters(camParam);
                                        light_flag = 0;
                                }
                        }

                        // construct a string from the valid bytes in the buffer
                        String readMessage = new String(readBuf, 0, msg.arg1);
                        mConversationArrayAdapter.add(mConnectedDeviceName + ":  "
                                        + readMessage);
                        break;
                case MESSAGE_DEVICE_NAME:
                        // save the connected device's name
                        mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
                        Toast.makeText(getApplicationContext(),
                                        "Connected to " + mConnectedDeviceName,
```

```
                                        Toast.LENGTH_SHORT).show();
                        break;
                case MESSAGE_TOAST:
                        Toast.makeText(getApplicationContext(),
                                        msg.getData().getString(TOAST), Toast.LENGTH_SHORT)
                                        .show();
                        break;
                }
        }
};

public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (D)
                Log.d(TAG, "onActivityResult " + resultCode);
        switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
                // When DeviceListActivity returns with a device to connect
                if (resultCode == Activity.RESULT_OK) {
                        // Get the device MAC address
                        String address = data.getExtras().getString(
                                        DeviceListActivity.EXTRA_DEVICE_ADDRESS);
                        // Get the BLuetoothDevice object
                        BluetoothDevice device = mBluetoothAdapter
                                        .getRemoteDevice(address);
                        // Attempt to connect to the device
                        mChatService.connect(device);
                }
                break;
        case REQUEST_ENABLE_BT:
                // When the request to enable Bluetooth returns
                if (resultCode == Activity.RESULT_OK) {
                        // Bluetooth is now enabled, so set up a chat session
                        setupChat();
                } else {
                        // User did not enable Bluetooth or an error occured
                        Log.d(TAG, "BT not enabled");
                        Toast.makeText(this, R.string.bt_not_enabled_leaving,
                                        Toast.LENGTH_SHORT).show();
                        finish();
                }
        }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.option_menu, menu);
        return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.scan:
                // Launch the DeviceListActivity to see devices and do scan
                Intent serverIntent = new Intent(this, DeviceListActivity.class);
                startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
                return true;
        case R.id.discoverable:
                // Ensure this device is discoverable by others
                ensureDiscoverable();
```

```
                        return true;
                }
                return false;
        }

}
```

Appendix C - Source Code for Sensor Control

```
#ifndef ReadSensor_h
#define ReadSensor_h
//
//    FILE: ReadSensor.h
// PURPOSE: ReadSensor library for Arduino
// HISTORY: See ReadSensor.cpp

#include <C:\Users\Lance\Desktop\arduino-1.0-windows\arduino-1.0\libraries\RunningMedian\RunningMedian.h>


class ReadSensor
{
        public:
        RunningMedian samples_f;
        RunningMedian samples_b;
        void SensorRead(int &value_f, int &value_b);
        void AnalyzeDanger(int value_f, int value_b, bool &prev_f, bool &prev_b, int &danger_f, int &danger_b, bool &dflag_f, bool
&dflag_b, unsigned char &lastcom);
        void DangerLoop(int value_f, int value_b, bool &prevl_f, bool &prevl_b, int &dangerl_f, int &dangerl_b, bool &dflag_f, bool
&dflag_b, unsigned char &lastcom);
        void Cruise(int value_f, int value_b, bool &pf, bool &pr, bool &pc, int &cf, int &cr, int &cc, unsigned char &cstatus);


        protected:
};

#endif
// END OF FILE


//    FILE: ReadSensor.cpp
// PURPOSE: ReadSensor library for Arduino
#include "Arduino.h"
#include "ReadSensor.h"
#include "RunningMedian.h"



// Reads Sensor Value, returns running median
void ReadSensor::SensorRead(int &value_f, int &value_b)
{
        void clear();
        int counter = 0;
        while(counter!=9){ //collect 9 readings from both sensors
                int x = analogRead(A0); //Read front sensor
                int y = analogRead(A1); //Read back sensor
                samples_f.add(x);
                //Serial.println(x);
                //delay(70);
                samples_b.add(y);
                counter++;
                delay(10);
        }
        value_f = samples_f.getMedian(); //take median of readings from front sensor
        value_b = samples_b.getMedian(); //take median of readings from back sensor
        Serial.println(value_f); //print median value from front sensor
        //Serial.println(value_b); //print median value from back sensor
        void clear();
   counter=0;
}

// returns distance condition flag
void ReadSensor::AnalyzeDanger(int value_f, int value_b, bool &prev_f, bool &prev_b, int &danger_f, int &danger_b, bool &dflag_f, bool
&dflag_b, unsigned char &lastcom)
{
        int dis_f = 121;
        if(lastcom == 0x09)
        {
                dis_f = 93;
```

```
		}
		else if(lastcom == 0x0a)
		{
			dis_f = 70;
		}
		if(value_f>=dis_f){ //median distance value too close on front sensor
    danger_f++;
    prev_f = true;
    if(danger_f>2 && prev_f==true){
				danger_f = 0;
				prev_f = false;
      //Serial.println("danger_f");
				dflag_f=true;
    }
   }
   else if(value_f<dis_f){ //safe median distance value reading on front sensor
    prev_f = false;
    danger_f = 0;
   }
		if(value_b>=30){ //median distance value too close on back sensor
    danger_b++;
    prev_b = true;
    if(danger_b>2 && prev_b==true){
				danger_b = 0;
				prev_b = false;
      //Serial.println("danger_b");
				dflag_b=true;
    }
   }
   else if(value_b<30){ //safe median distance value reading on back sensor
    prev_b = false;
    danger_b = 0;
   }
}

void ReadSensor::DangerLoop(int value_f, int value_b, bool &prevl_f, bool &prevl_b, int &dangerl_f, int &dangerl_b, bool &dflag_f, bool
&dflag_b, unsigned char &lastcom)
{
		int dis_f = 121;
		if(lastcom == 0x09)
		{
			dis_f = 93;
		}
		else if(lastcom == 0x0a)
		{
			dis_f = 70;
		}
		if(value_f<dis_f){ //safe median distance value reading on front sensor
    dangerl_f++;
    prevl_f = true;
    if(dangerl_f>2 && prevl_f==true){
				dangerl_f = 0;
				prevl_f=false;
				dflag_f=false;
				//Serial.println("good_f");
    }
   }
   else if(value_f>=dis_f){ //median distance value too close on front sensor
    prevl_f = false;
    dangerl_f = 0;
   }
		if(value_b<30){ //median distance value too close on back sensor
    dangerl_b++;
    prevl_b = true;
    if(dangerl_b>2 && prevl_b==true){
				dangerl_b = 0;
				prevl_b = false;
				dflag_b=false;
				//Serial.println("good_b");
    }
```

```
    }
    else if(value_b>=30){ //safe median distance value reading on back sensor
      prevl_b = false;
      dangerl_b = 0;
    }
}

void ReadSensor::Cruise(int value_f, int value_b, bool &pf, bool &pr, bool &pc, int &cf, int &cr, int &cc, unsigned char &cstatus)
{
        //Serial.println("value of front sensor");
        //Serial.println(value_f);
        if(20<=value_f && value_f<=170){
                pf = true;
                pr = false;
                pc = false;
                cf++;
                cr = 0;
                cc = 0;
                if(cf>2 && pf==true){
                        cstatus = 0x00;
                        Serial.println("cstatus is");
                        Serial.print(cstatus);
                }
        }
        else if(171<=value_f && value_f<=271){
                pf = false;
                pr = true;
                pc = false;
                cf = 0;
                cr++;
                cc = 0;
                if(cr>2 && pr==true){
                        cstatus = 0x05;
                        Serial.println("cstatus is");
                        Serial.print(cstatus);
                }
        }
        else if(272<=value_f && value_f<=600){
                pf = false;
                pr = false;
                pc = true;
                cf = 0;
                cr = 0;
                cc++;
                if(cc>2 && pc==true){
                        cstatus = 0x0a;
                        Serial.println("cstatus is");
                        Serial.println(cstatus);
                }
        else {
                cstatus = 0x0a;
                }
        }
}


// END OF FILE
```

Appendix D - Source Code for Motor Control

```
#ifndef Motor_h
#define Motor_h
//    FILE: Motor.h
// PURPOSE: Motor Control  library for Arduino
// HISTORY: See Motor.cpp

class Motor
{
        public:
                //void MotorOn(const int &enable, const int &high, const int &low);
                void MotorOn(unsigned char &flag, unsigned char &motorvalue);
                void killMotors(bool &flag_f, bool &flag_b, unsigned char &last_com);
                //void stopMotor(unsigned char &lastcom);

        protected:
                /*typedef struct way {
                        int enable;
                        int high;
                        int low;
                };
                const int motor1Pin1 = 5;    // H-bridge leg 1 (pin 2, 1A)
                const int motor1Pin2 = 3;    // H-bridge leg 2 (pin 7, 2A)
                const int enablePin1 = 4;    // H-bridge enable pin
                const int motor2Pin1 = 6;    // H-bridge leg 1 (pin 2, 1A)
                const int motor2Pin2 = 9;    // H-bridge leg 2 (pin 7, 2A)
                const int enablePin2 = 8;    // H-bridge enable pin
                */
};

#endif
// END OF FILE


//    FILE: Motor.cpp
// PURPOSE: Motor Control library for Arduino
#include "Arduino.h"
#include "Motor.h"

void Motor::MotorOn(unsigned char &flag, unsigned char &motorvalue)
{
        int speed;
        if (flag == 'a') {
                if (motorvalue<5) {
                //LEFT (a)
                        digitalWrite(4, HIGH);
                        //digitalWrite(5, HIGH);
                        speed = (5-motorvalue)*51;
                        analogWrite(5, speed);
                        digitalWrite(3, LOW);

                        //delay(200);
                        digitalWrite(4, LOW);
                } else if (motorvalue>5) {
                //RIGHT (d)
                        digitalWrite(4, HIGH);
                        //digitalWrite(3, HIGH);
                        speed = (motorvalue-5)*51;
                        analogWrite(3, speed);
                        digitalWrite(5, LOW);

                        //delay(200);
                        digitalWrite(4, LOW);
                }
                else{
                        digitalWrite(3, LOW);
                        digitalWrite(5, LOW);
                        digitalWrite(4, LOW);
                }
```

```
	} else if (flag == 'd') {
		if (motorvalue>5) {
		//FORWARD (w)
			digitalWrite(8, HIGH);
			//digitalWrite(9, HIGH);
			speed = (motorvalue-5)*51;
			analogWrite(9, speed);
			digitalWrite(6, LOW);

			//delay(200);
			digitalWrite(8, LOW);
		} else if (motorvalue<5) {
		//BACKWARD (s)
			digitalWrite(8, HIGH);
			//digitalWrite(6, HIGH);
			speed = (5-motorvalue)*51;
			analogWrite(6, speed);
			digitalWrite(9, LOW);

		//	delay(200);
			digitalWrite(8, LOW);
		} else{
			digitalWrite(6, LOW);
			digitalWrite(9, LOW);
			digitalWrite(8, LOW);
		}
	} else {
		digitalWrite(4, LOW);
		digitalWrite(8, LOW);
	}
}

void Motor::killMotors(bool &flag_f, bool &flag_b, unsigned char &lastcom)
{
	int delay_t;
	if (flag_f) {
		digitalWrite(8, HIGH);
		digitalWrite(6, HIGH);
		digitalWrite(9, LOW);
		if (lastcom<5) {
			delay_t = (5-lastcom);
			delay(delay_t*125);
		} else if (lastcom>5) {
			delay_t = (lastcom-5);
			delay(delay_t*125);
		}
		//Serial.print("delay for ");
		//Serial.println(delay_t, HEX);
		//delay(100);

		digitalWrite(3, LOW);
		digitalWrite(5, LOW);
		digitalWrite(4, LOW);
		digitalWrite(6, LOW);
		digitalWrite(8, LOW);
		digitalWrite(9, LOW);

	} else if (flag_b) {
		digitalWrite(8, HIGH);
		digitalWrite(9, HIGH);
		digitalWrite(6, LOW);
		if (lastcom<5) {
			delay_t = (5-lastcom);
			delay(delay_t*125);
		} else if (lastcom>5) {
			delay_t = (lastcom-5);
			delay(delay_t*125);
		}
		//delay(100);
```

```
                digitalWrite(3, LOW);
                digitalWrite(5, LOW);
                digitalWrite(4, LOW);
                digitalWrite(6, LOW);
                digitalWrite(8, LOW);
                digitalWrite(9, LOW);
        }
        else {
                digitalWrite(3, LOW);
                digitalWrite(5, LOW);
                digitalWrite(4, LOW);
                digitalWrite(6, LOW);
                digitalWrite(8, LOW);
                digitalWrite(9, LOW);
        }
}
/*
void stopMotor(unsigned char &lastcom);
{
        int delay_t;
        if (lastcom<5) {
                        delay_t = (5-lastcommand);
                        delay(delay_t*50);
        } else if (lastcom>5) {
                        speed = (lastcom-5)*51;
                        delay(delay_t*50);
        }
}
*/
// END OF FILE
```

Appendix E - Source Code for Arduino Uno Microcontroller

```
#include <MeetAndroid.h>
#include <RunningMedian.h>
#include <ReadSensor.h>
#include <Motor.h>

MeetAndroid meetAndroid;
Motor motor;
byte initFlag = 0x64, initCom1 = 0x07, initCom2 = 0x03, initCom3 = 0x05;
byte lastcom = 0x05;
byte cspeed = 0x07;
byte cstop = 0x08;
byte lastmotor = 0x05;
byte forward = 0x64;
bool stopped = true;
bool tflag = true;
bool fflag = false;
byte mode = 0x01;
int mode3_flag = 0;


// Sensor Setup
  int dangercounter_f = 0;
  int dangercounter_b = 0;
  int dangercounterl_f = 0;
  int dangercounterl_b = 0;
  int sensorvalue_f = 0;
  int sensorvalue_b = 0;
  bool previouscounter_f = false;
  bool previouscounter_b = false;
  bool previouscounterl_f = false;
  bool previouscounterl_b = false;
  bool dangerflag_f = false;
  bool dangerflag_b = false;

  bool p_f = false;
  bool p_r = false;
  bool p_c = false;
  int c_f = 0;
  int c_r = 0;
  int c_c = 0;
  byte c_status = 0x07;

  ReadSensor readsensor;

void setup()
{
    Serial.begin(115200);
    pinMode(13, OUTPUT);  //onboard LED
    meetAndroid.registerFunction(motorCommand, 'a');
    meetAndroid.registerFunction(motorCommand, 'd');
    meetAndroid.registerFunction(modeConfig, 'm');
    motor.MotorOn(initFlag, initCom1);
    motor.MotorOn(initFlag, initCom2);
    motor.MotorOn(initFlag, initCom3);
}

void loop()
{
  switch(mode)
  {
      case 0x01:  //Mode 1: Normal Operation
      meetAndroid.receive();
      break;

      case 0x02:  //Mode 2: Head-on collision avoidance
      // Read proximity sensors, specify minimum distance, and calculate difference from threshold
      Serial.print(lastcom, HEX);
      readsensor.SensorRead(sensorvalue_f, sensorvalue_b);
```

```
    readsensor.AnalyzeDanger(sensorvalue_f, sensorvalue_b, previouscounter_f, previouscounter_b, dangercounter_f,
dangercounter_b, dangerflag_f, dangerflag_b, lastcom);
    if (dangerflag_f || dangerflag_b == true)
    {
      digitalWrite(13, HIGH);
      // PID control loops to maintain distance and heading
      // Motor control from Android smartphone over Bluetooth
      motor.killMotors(dangerflag_f, dangerflag_b, lastcom);
      Serial.println("disable motors!");
      while(dangerflag_f || dangerflag_b == true)
      {
        readsensor.SensorRead(sensorvalue_f, sensorvalue_b);
        readsensor.DangerLoop(sensorvalue_f, sensorvalue_b, previouscounterl_f, previouscounterl_b, dangercounterl_f,
dangercounterl_b, dangerflag_f, dangerflag_b, lastcom);
      }
      lastcom = 0x05;
    }
    else
    {
      digitalWrite(13, LOW);
      meetAndroid.receive();
    }
    break;

    case 0x03:  //Mode 3: Cruise control
    readsensor.SensorRead(sensorvalue_f, sensorvalue_b);
    readsensor.Cruise(sensorvalue_f, sensorvalue_b, p_f, p_r, p_c, c_f, c_r, c_c, c_status);
    //Serial.print("Status is  = ");
    //Serial.println(c_status, HEX);
    //Serial.print("Last motor speed is = ");
    //Serial.println(lastmotor, HEX);
    if (c_status == 0x00) { //too far
      digitalWrite(13, LOW);
      if (lastmotor<10) {
        lastmotor++;
      }
      stopped = true;
      motor.MotorOn(forward, cspeed);
    }
    else if (c_status == 0x05) { // ok
      digitalWrite(13, LOW);
      if (lastmotor>5) {
        lastmotor--;
      }
      stopped = true;
      motor.MotorOn(forward, cspeed);
    }
    else if (c_status == 0x0A) { //too close
      digitalWrite(13, HIGH);
      if (stopped) {
        motor.killMotors(tflag, fflag, cstop);
        delay(5000);
      }
      stopped = false;
      lastmotor = 0x05;
    }
    else {
        if (stopped) {
          motor.killMotors(tflag, fflag, cstop);
          delay(5000);
        }
    }
    Serial.println("stopped flag is");
    Serial.println(stopped);
    mode3_flag = 1;
    meetAndroid.receive();
    break;

  default:
    Serial.println("State = DEFAULT");
```

```
    meetAndroid.receive();
  }
}

void motorCommand(byte flag, byte numOfValues)
{
  byte command = meetAndroid.getInt();
  if (mode3_flag == 0)
  {
    motor.MotorOn(flag, command);
    //Serial.print("Flag = ");
    //Serial.println(flag, HEX);
    //Serial.print("Command = ");
    //Serial.println(command, HEX);
  }

  lastcom = command;
}

void modeConfig(byte flag, byte numOfValues)
{
  mode = meetAndroid.getInt();
  if (mode3_flag == 1)
  {
    mode3_flag = 0;
  }
}
```