NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Department of Computer Science and Engineering

Project Report

Title: _____

_____

Submitted in Partial Fulfillment of the Requirements
For the Degree of
BACHELOR OF SCIENCE
In Computer Engineering

Submitted by: _____ Date _____

Advisor Approval: _____ Date _____

Departmental Acceptance: _____ Date _____

Senior Design Project - BluTank

TOMASZ GLAZEWSKI

RICHARD LAU

RIJVI RAJIB

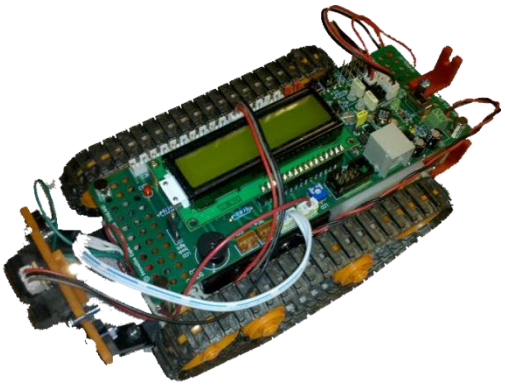
Table of Contents

Introduction

BluTank, a robotic vehicle platform powered by Bluetooth for secure wireless control with Android support for mobile ease-of-use, was the final product of a twelve week endeavor by a team of three. BluTank was designed as a proof-of-concept that open secure wireless abilities could be added to robotic platforms in use by the military. Bluetooth devices are secured against eavesdropping and hijacking through individual pairing (explained in more detail later). Bidirectional data transfer and encryption is fully supported by Bluetooth for improved responsiveness during communication.

Military robots will be improved by incorporating the open Bluetooth standard for wireless transmissions. Modern military robots are based off the *Wheelbarrow* robot which was designed by Lieutenant-Colonel Peter Miller in 1972. The *Wheelbarrow Revolution* is currently used in Iraq and Afghanistan and has an undisclosed wireless radio frequency specification. Current, controlled demolition of explosives is completed using a manual detonator wire. The use of wired detonation has caused problems before and will be solved with the implementation of Bluetooth. With Bluetooth, BluTank can be used to navigate to a bombsite and a detonator can be remotely started over Bluetooth while the robot's status is updated over a secure channel.

Our vehicle is a demo sample of what can be accomplished with a simple Bluetooth modem. The battery pack provided is a group of AAs that powered everything from the onboard LCD display to the Bluetooth modem and any other sensor peripherals added to the robot such as a distance calculator



The robot is equipped with treads, to simulate the designs of robots currently utilized. The treads are powered by a servo motor which is allowed 256 degrees of speed on each tread so the robot is precisely controlled and a more accurate response is guaranteed from the robot at different conditions. The motors are placed internally at the center of the vehicle so they are protected from external abuse. The Bluetooth controller is also placed internally, and is connected to the top of the vehicle's PIC board.

The Bluetooth modem is paired with a laptop or an Android device to send data over the UART port. The laptop would interface with the modem using HyperTerminal. The Android device interfaces with the Bluetooth modem through a custom application and the Android SDK Bluetooth API. Both interfaces will not only send data to the modem, but is also capable of receiving data from the Bluetooth modem to identify and define anything the our vehicle will send back such as distance from collision, potential target sites, road conditions, vehicle traction information, and many other things.

Data Log

Project Motivation

Our main motivation for creating a mobile Bluetooth platform was a particular scene from the 2008 film The Hurt Locker. The film is about the lives of Explosive Ordinance Disposal (EOD) technicians and early on in the film the bomb disposal crew utilizes a bomb disarming robot to approach and identify an Improvised Explosive Device (IED). After it is identified, the crew attempts to setup a controlled explosion using the same robot but failed to do so due to technical problems. Their main technician ends up having to setup the explosives himself while wearing a protective suit but is killed from another hidden explosive that is remotely activated by an insurgent. While real life bomb disposal is much more remote than the movie portrays, it seems like there is a lack of redundancy in case of

mechanical problems with most common bomb disposal robots. Therefore we proposed a Bluetooth platform enabled robot both for its built in wireless and encryption abilities. In addition, most modern Bluetooth modems have a decent tolerance in environments with lots of wireless activity through frequency mapping while consuming very little power. Bluetooth combined with a powerful microcontroller architecture creates a relatively inexpensive device with very good reliability and lots of functionality.

Bluetooth was selected because modern EOD robots use classified radio frequency schemes for transmitting data from the control unit to the robot and may be susceptible to hacking or radio interference. There also exist robots which use fiber optic lines for transmission. In either case, there are possibilities that can lead to technical problems that can delay the defusing of a suspicious device. With Bluetooth we have a good range (up to roughly 100 meters for Class 1 devices), frequency mapping to decrease sensitivity to radio interference, and built in encryption that is secure (as of Bluetooth 2.1). This adds additional layers of robustness to devices that critically need it for their job.

The military is also transitioning to Android enabled devices due to their ease of use and mobility for personnel. We figured Bluetooth would also be great for this area since many such devices also have onboard Bluetooth enabled. Since we are using Bluetooth to transmit serial data through the adapter to our platform, we can also develop an android application to control our robot. This allows us to maintain all of the benefits of Bluetooth while giving our system mobility for on-the-fly access, which could be very useful for military operations where operators may be deployed in the field without proper equipment support aside from the robot. With android, we can quickly distribute our control program for those who need it and provide instant support even if they do not have a specific dedicated device like in traditional applications.

Project Goals

- Design platform for microcontroller to Bluetooth interface featuring encryption
- Design robot for robustness and low power consumption
- Build microcontroller robotic movement system within a cost effective budget
- Develop and implement Android powered application to communicate via Bluetooth

Initial Decisions

The observed bomb disarming robot used in *The Hurt Locker* had a high profile and additional instruments for bomb disposal. The BluTank concept however is designed primarily for controlled demolition of suspicious devices. Design elements were borrowed from the Goliath tracked mine used in service by the German military in World War II. The lower profile and reinforced central chassis protects the control unit and servo motors of the robot. Tracks were chosen to provide better all terrain mobility over wheels which can get caught or stuck.

The PIC microcontroller interface board was used as opposed to the widely used Arduino programming board that many other people would use – this decision was made strictly because the PIC interface would work better with the Bluetooth modem that is utilized.

There are several sensors on our PIC board to utilize in our vehicle. At the time, we decided we'd have traction sensors on the motors, speed sensor using a distance calculator, and collision sensors to detect any contact that could occur in the immediate future. The Bluetooth controller would be controlled by an Android device through a custom application for better control over what is sent and received by the vehicle through the Bluetooth modem. HyperTerminal is used on a laptop to test the control of the vehicle and ensure that everything is working for the Android devices.

Finalized Decisions

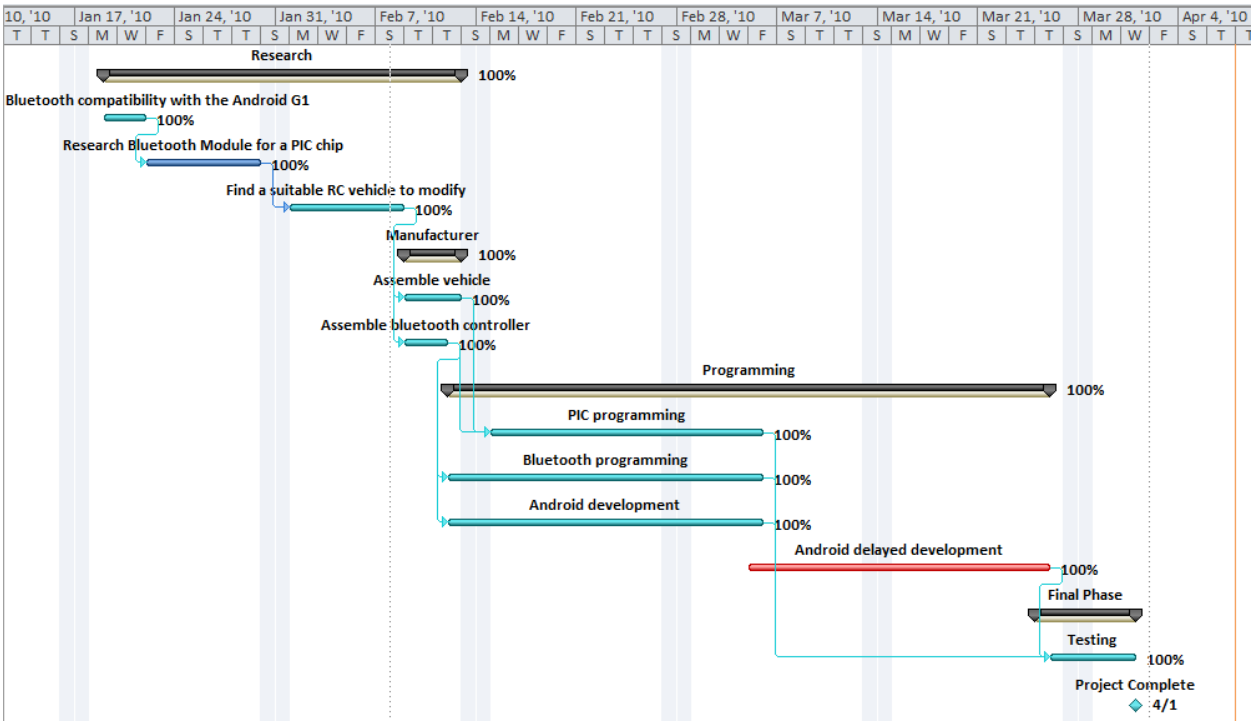
We ordered several parts for our vehicle and a Bluetooth modem to interface with it. The original PIC chip select caused issues due to its internal baud rate. After some searching, we found a PIC 16F887 set at 20MHz. The microcontroller came with a robotics kit that included an LCD screen unit that could be attached to display output and a sample C compiler for development. We also purchased a Bluetooth Class 1 modem for the sake of keeping the cost low. The Class 1 modem has an effective range of 100 meters. The Bluetooth modem contains a serial connection with supported speeds between 2400 bps and 115200 bps; this allows the Bluetooth modem to respond quickly to commands set we set up. The PIC board we chose had UART support with dedicated Tx and Rx ports which we would utilize specifically for Bluetooth.

Due to the fact that Android operating system is new, three different versions of Android were encountered during development. The T-Mobile G1 uses version 1.6 and the Verizon Droid was previously running on version 2.0.1. As of March 31, 2009, the Verizon Droid updated to version 2.1. The android application was developed under the 2.0.1 version framework which allows the usage of the Bluetooth API for the Android SDK. Luckily, the Android 2.1 version is backwards compatible and works with the 2.0.1 version of the operating system. The HyperTerminal application on laptops was used to interface with the Bluetooth modem and to send and receive data to and from the vehicle's PIC board during development testing. In order to get the T-Mobile G1 connect to our Bluetooth modem, we had to modify the application significantly; instead of creating a unique application for it, we created a simulated HyperTerminal for the Android utilizing the Console application that can be installed. There was no API usage for the 1.6 version – instead, we rooted our system to gain administration access and directly modified the operating system to utilize the Bluetooth hardware that the G1 comes with.

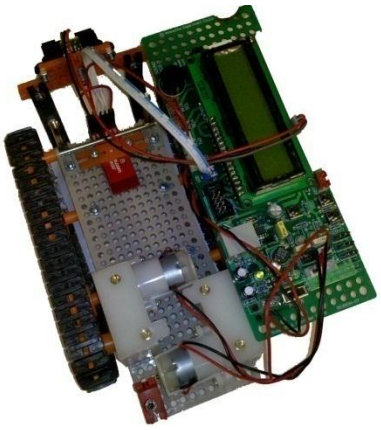
Cost Analysis

Item	Cost
RC Test Vehicle	$189.95
Bluetooth modem	$64.95
Total Cost	$254.90

Development Time Table

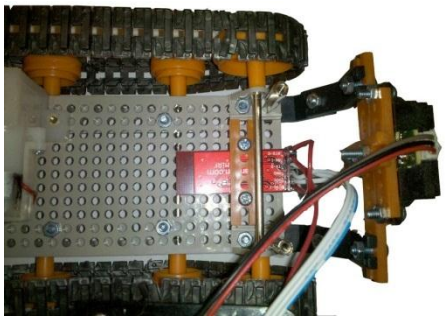


Development Process



After the general design was finalized, component parts were purchased. Tamiya modeling tracks were used along with a bracket for mounting the motors and chassis train. The PIC development board is attached to lower plate using thumb screws. An LCD display unit is placed on top of the PIC board is used for debugging and basic testing of our control program before we attempt to send data over the Bluetooth connection. An IR control was initially used for developing and checking the motor control scheme but was removed to incorporate the Bluetooth adapter later on.

The Bluetooth modem came without any attachments so we were unable to connect it to the development board. To solve this issue, pins were manually soldered to an adapter for the modem. Wires were then placed into an adapter which allows for easy plug and play connectivity. This allowed us to easily connect the Bluetooth modem to the PIC board through the designated ports. The wires were connected to the TxD and RxD port of the PIC board which also had separate +5V pin and a GND pin to power the modem.



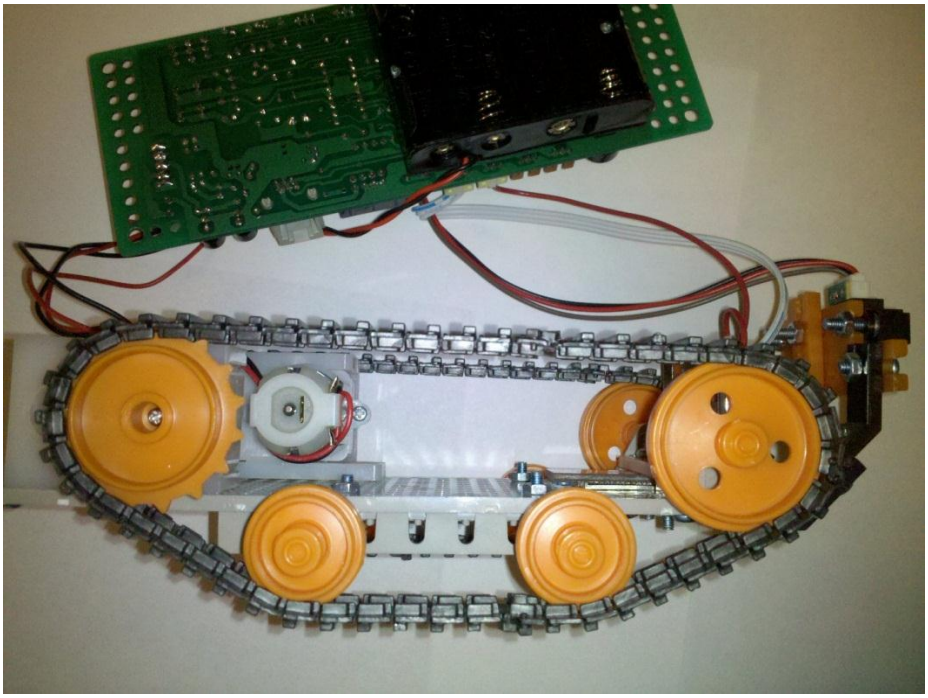
After the physical device was built, the next step was the most time consuming – programming the board and the interface for the Bluetooth modem. For that, we spent our time developing on two different platforms at once – Android operating system 2.1 and the PIC microcontroller architecture. The Android operating system utilizes Java and XML as the main language interfacing with the Bluetooth API provided by the SDK while the PIC microcontroller uses C and uses some libraries provided by MikroC.

The Android program to interface with the Bluetooth modem is programmed in Java. The Eclipse IDE with the ADT plugin was used to compile and program the application for Android 2.1. Since the Bluetooth API is new, the application development for it was tedious. Documentation on this API was scant and hard to decipher, but after a while, it was updated as more developers got a hold of it.

Three files were used for the Android program before it could be compiled fully. The BluetoothService.java stored all the methods necessary to utilize the Bluetooth modem. This file handles the sending and receiving of byte data along with the connection and authorization of the Bluetooth modem with the Android device. Since the Bluetooth modem data is encrypted, the Android device had to be set up to make sure that the data sent and received would be readable on either end of the program. The DeviceListActivity.java lists the various devices that can connect to application device and send Bluetooth data over. Finally, the BlueCar.java is the main file of the Android application – it is the class that decides what happens and runs when the XML front end is used. At first, a simple HyperTerminal-like interface was created to send and receive data with the Bluetooth modem. As the application progressed and was tested, a graphical interface was added, providing 4 buttons to control the robot's motor movement and then a 5^{th} button to send the "STOP" command to make the robot stop moving. The graphical interface is modified in the XML file and cleaned up integrating custom graphics for each button.

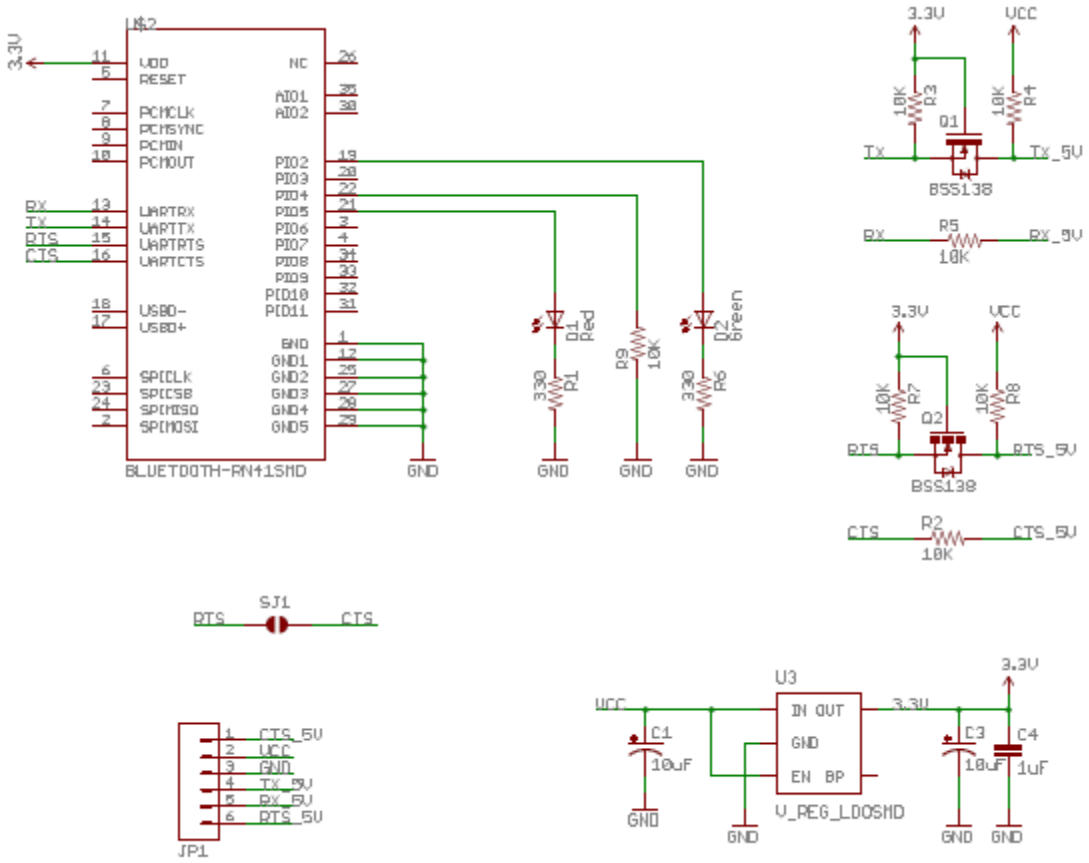
The PIC microcontroller program to interface with the Bluetooth modem's receiving data is programmed in C. The MikroC application program provided some useful libraries to handle UART connection and anything else needed to interface with the PIC board and the microcontroller. Essentially, the board chosen is irrelevant to the robot since it's simply being utilized by the provided pins on our PIC16F887. The board simply organizes the interface with the PIC16F887 and helps us connect the necessary peripherals.

The file “uart.c” is the main bulk of the code of what our robot will do with specific commands sent from the UART and how to interface with the physical attachments on the board such as the modem and any other sensors we add. There are 4 main functions to control the speed of the motors, forward and reverse, one function just for bit assignment and initialization, and finally, the main function which contains an infinite loop to listen for data being sent in.



Technical Aspects

Bluetooth Modem



- FCC Approved Class 1 Bluetooth Radio Modem
- Extremely small radio - 0.15x0.6x1.9"
- Very robust link both in integrity and transmission distance (100m) - no more buffer overruns!
- Low power consumption : 25mA avg
- Hardy frequency hopping scheme - operates in harsh RF environments like Wi-Fi, 802.11g
- Encrypted connection
- Frequency: 2.4~2.524 GHz
- Operating Voltage: 3.3V-6V
- Serial communications: 2400-115200bps
- Operating Temperature: -40 ~ +70C
- Built-in antenna

Bluetooth Overview

Bluetooth is an open wireless standard for communicating over short-range radio frequencies on the unlicensed Industrial, Scientific and Medical (ISM) 2.4 - 2.4835 GHz frequency band. Due to the public nature of the ISM band, there is considerable interference due to propagation of radio signals from other devices ranging from cordless home phones, microwaves, the IEEE 802.11 b/g WLAN standard, etc. Bluetooth deals with this by using frequency-hopping spread spectrum (FHSS) technology in order choose between 79 different 1 MHz bands in a pseudo-random selection sequence for hops. Bluetooth channels are only used for a few hundred milliseconds per hop and channels with heavy interference are avoided for fault tolerance.

Overall, Bluetooth provides an inexpensive and relatively low-power alternative for creating small wireless networks (piconets) that can be used for cable replacement, file sharing, synchronization, telephony, and tethering for internet access to name a few. In our case, we are using Bluetooth to connect and control our robot using a small encrypted piconet. Bluetooth is ideal for this purpose because it has both built in encryption features and low power consumption, also as of Bluetooth 2.1 pairing has been changed to Secure Simple Pairing (SSP) which makes pairing more transparent and quicker. SSP also improves on the previous Bluetooth framework by increasing protection against passive eaves dropping and man-in-the-middle (MITM) type attacks PIN-based systems suffer from.

Security Protocols

Since we are using Bluetooth 2.1, unlike previous iterations of Bluetooth we have SSP support. SSP has different association modes depending on the device input capability of the devices trying to pair. SSP also introduces the use of the Elliptic Curve Diffie-Hellman (ECDH) key protocol which allows two parties to establish a shared secret over an insecure channel for added security during key exchange and link key creation.

When using SSP, there are four available association models that provide significant improvements in security over previous versions of Bluetooth. The four available modes are Numeric Comparison, Just Works, Passkey Entry, and Out Of Band (OOB). In addition, there is legacy support for pre-v2.1 devices that use fixed PIN authentication.

Association Models

NUMERIC COMPARISON

In Numeric Comparison mode, both devices must be able to display a six digit number and users are asked to compare both numbers and verify yes or no for each device. Yes must be selected on both devices in order for pairing to complete. By requiring users to manually verify each device, it allows each device to be identified as who they say they are as well as protecting against man-in-the-middle (MITM) attacks. As the displayed number is not an actual input in link key generation, knowing the number does not provide a benefit for someone eavesdropping.

JUST WORKS

Just Works association is used when one device does not have display or input capability which commonly occurs when pairing handheld devices with headsets. In this case, Numeric Comparison is used but the user is not shown the number and is simply asked to accept or decline pairing. Just Works provides passive eavesdropping protection but is still vulnerable to MITM attacks.

PASSKEY ENTRY

Passkey Entry mode is used when one device has a display and another device has input capability. One device displays a six digit number as in Numeric Comparison and the other device is inputs the displayed number for validation. The number displayed is not used in link key generation so it also provides MITM protection as in Numeric Comparison.

OUT OF BAND

Out Of Band association works using an intermediary mechanism for device discovery as well as key exchange during pairing. The OOB channel serves as the main source of protection against MITM attacks so it must be chosen carefully to ensure protection.

Legacy Pairing

Pre-Bluetooth v2.1 pairing or legacy pairing refers to the use of a PIN number prior to SSP for authentication. In addition to using PIN-based pairing, there are several security modes based on the needs and services provided by Bluetooth devices.

Security Mode 1

Non-secure pairing mode with all security features disabled. In this mode, the device is discoverable to any nearby Bluetooth devices and connections made using this security mode are susceptible to eavesdropping and attacks. This security mode is supported in Bluetooth implementations v2.0 or earlier.

Security Mode 2

In this security mode, security is implemented after a link is established but before the data channel is used. Also known as service level enforced security, this security mode provides security for the data channel after it is established but not before linking. This mode also introduces authorization, which limits the available services open to paired devices. This security mode is supported in all versions of Bluetooth (for v2.1, this mode is provided only for legacy pairing/compatibility).

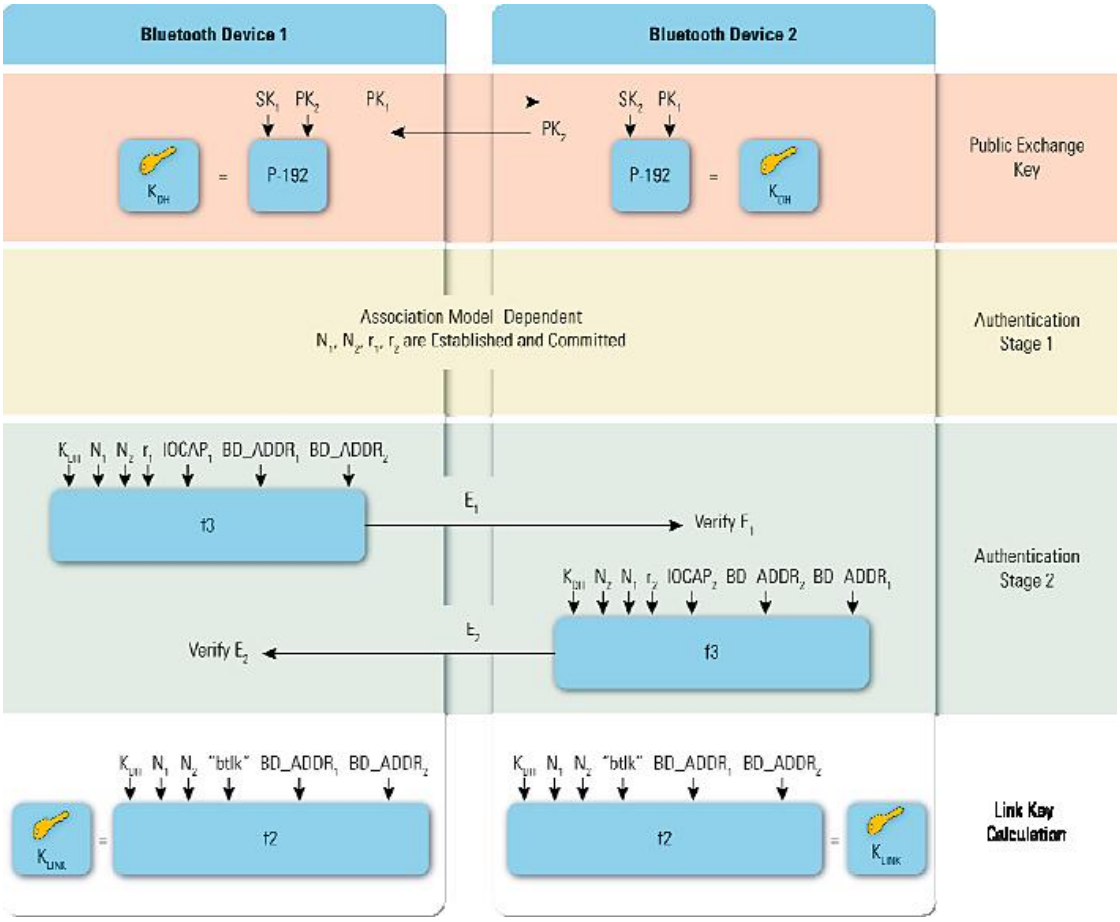
Security Mode 3

Link level-enforced security mode enables the use of security procedures before linking and authentication and encryption is mandatory on all connections made with the device.

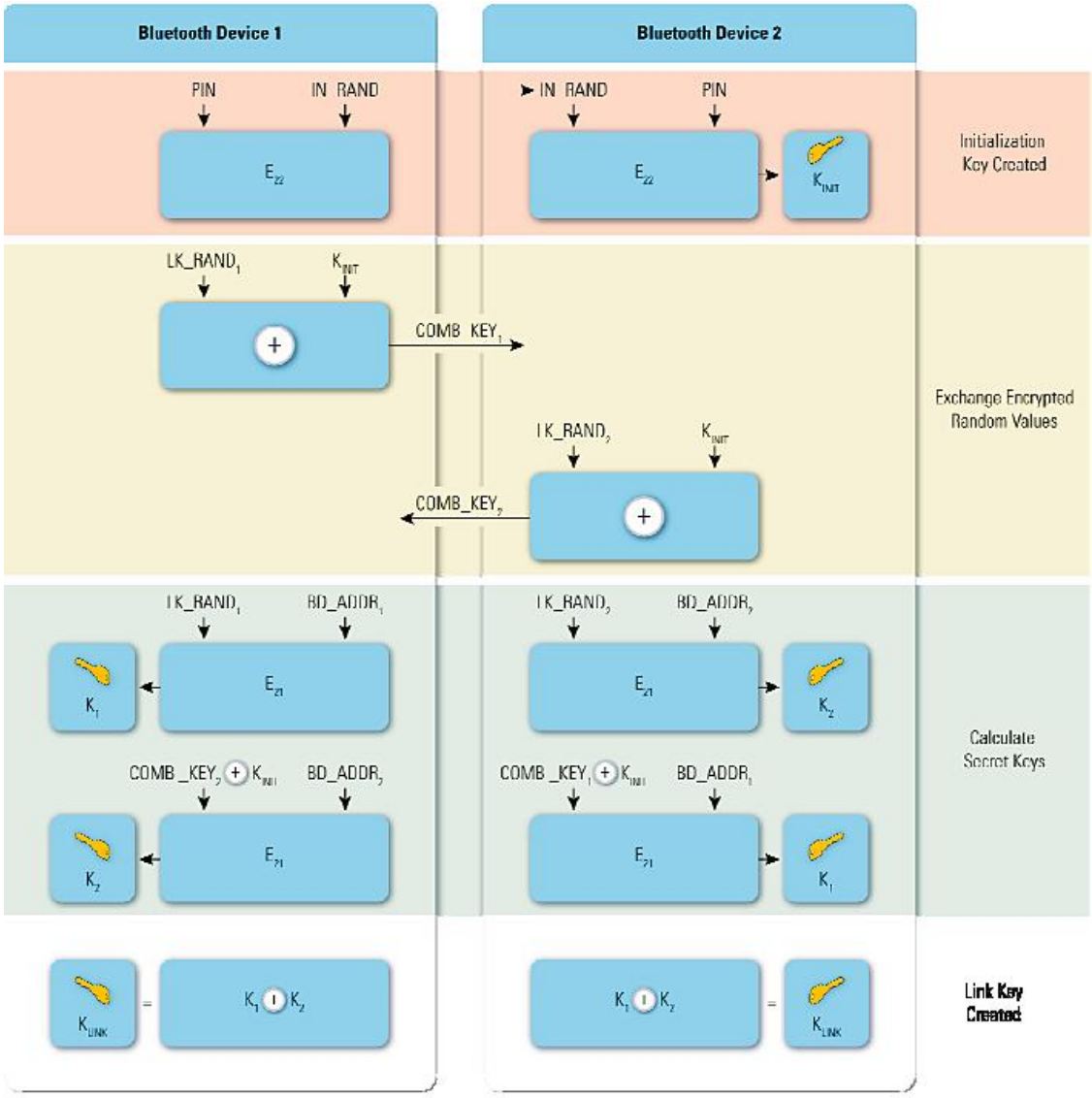
Authentication can be either unidirectional or mutual depending on the types of services shared between devices. Authentication and encryption is based on a separate secret link key after pairing. This security mode is supported in Bluetooth implementations v2.0 or earlier.

Link Key Generation

Before pairing authentication can occur, both parties need to generate a Bluetooth Link Key which will be a shared secret between the two devices during transactions. For Bluetooth 2.0 and earlier, two devices associating with each other must simultaneously derive link keys during initialization, during which users enter a PIN into one or both devices (depending on the type connections that can be made). The main improvement present in Bluetooth 2.1 and up is the introduction of ECDH public/private key pairs rather than symmetric keys generated via a PIN.



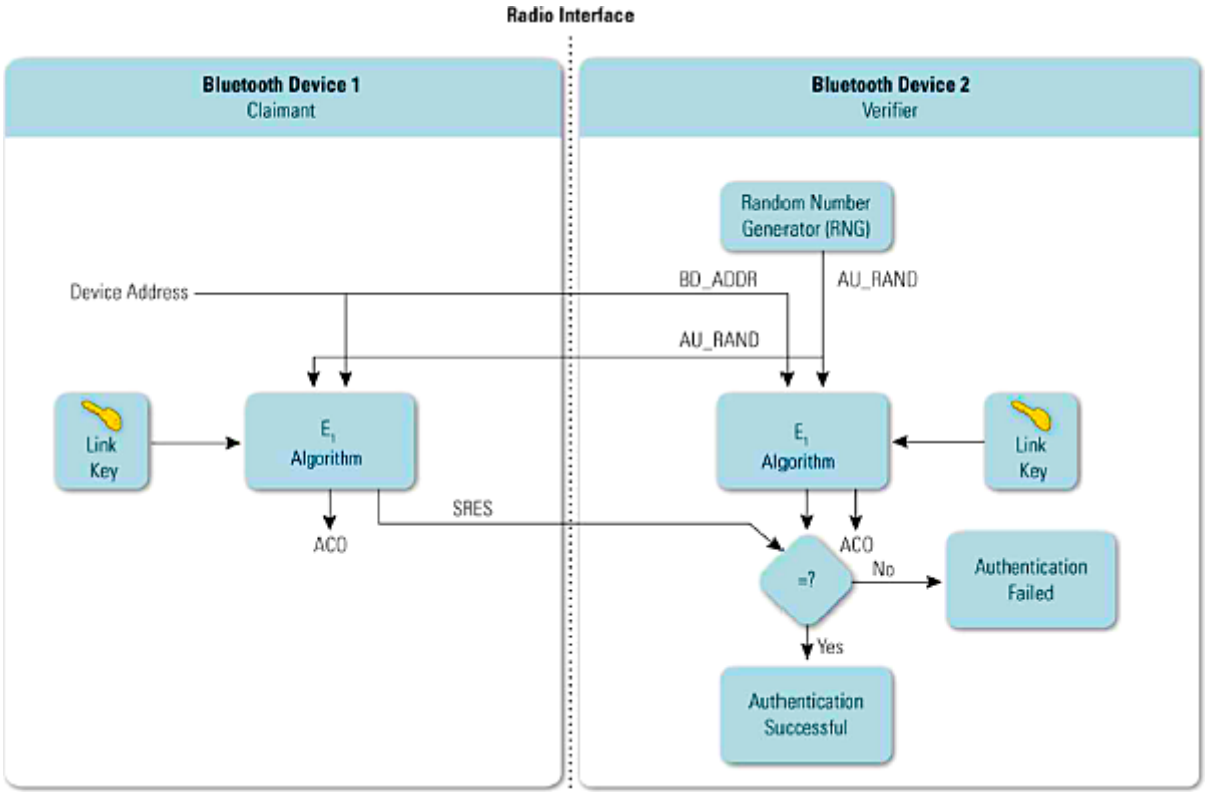
Bluetooth 2.1+ Link Key Generation



Bluetooth 2.0- Link Key Generation

Authentication

The Bluetooth device authentication process uses a challenge-response scheme. Each device attempting to pair in the authentication procedure is referred to as either the claimant or the verifier. The claimant is the device attempting to prove its identity while the verifier is the device that is validating the identity of the claimant. The challenge-response protocol validates devices based on each device's knowledge of a secret Bluetooth link key.



Bluetooth Authentication

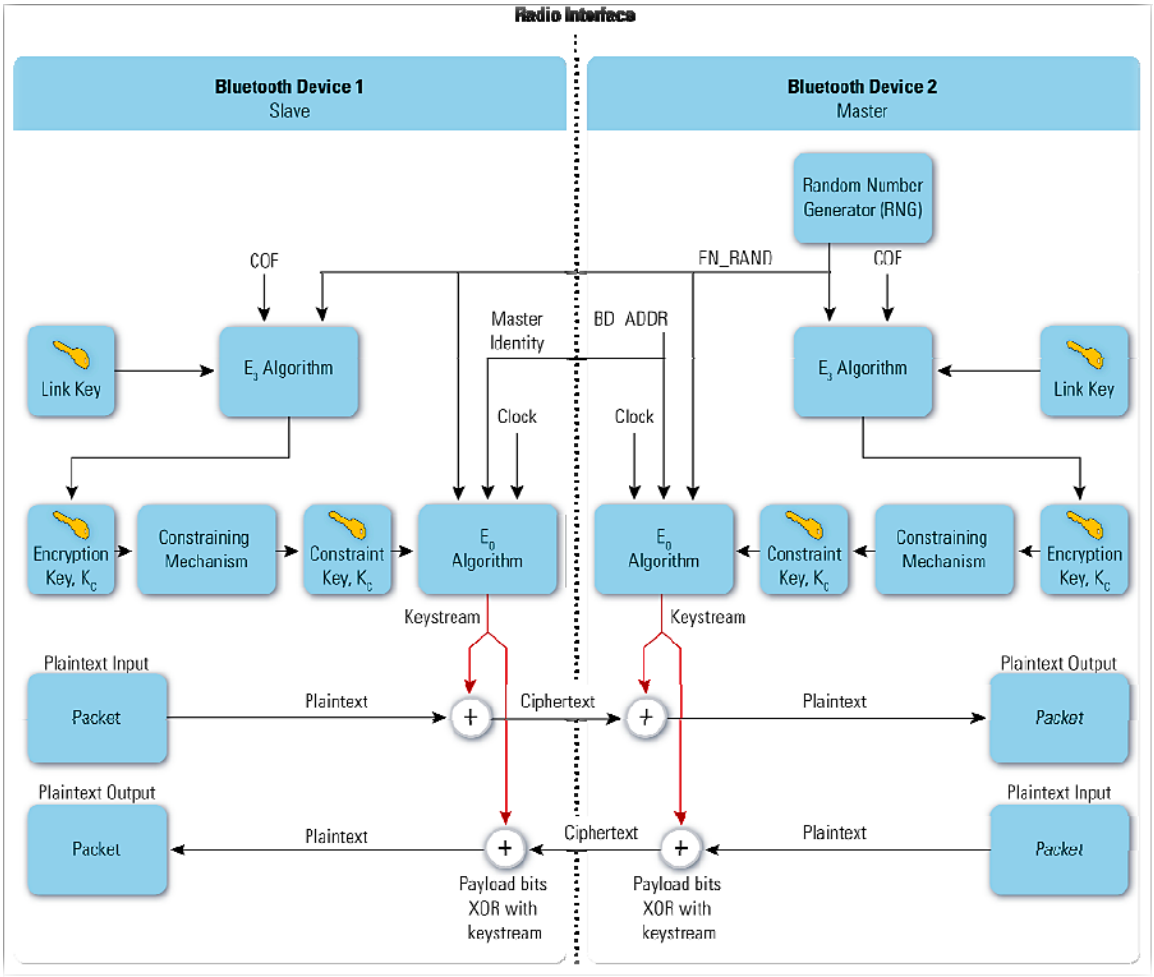
Authentication begins with the transmittance of a 128-bit random challenge (AU_RAND) to the claimant. The claimant then uses the E1 algorithm to generate a response using their own 48-bit Bluetooth device address (BD_ADDR), the link key, and the AU_RAND as inputs. The verifier also performs this computation and the 32 most significant bits from E1 are used for the authentication. The remaining 96 bits of the 128-bit output are then used as the Authenticated Ciphering Offset (ACO) value, which is later used for the encryption key. The 32-bit response (SRES) is then transmitted from the claimant to the verifier. After the verifier compares the received SRES value with its own computed value, authentication is either successful or fails. When authentication fails, there is a time delay between the two devices that increases exponentially with every failed attempt. This is done as a safety precaution to make it unlikely for authentication to succeed through brute force. After this one-way authentication is

completed, mutual authentication can be performed if necessary but having claimant and verifier exchange roles and redo the authentication process.

Encryption

In addition to authentication and authorization, Bluetooth has various encryption modes depending on the types of services and needs of the end user in order to ensure confidentiality. Encryption ranges from no encryption, encryption on individual traffic using individual link keys, to full encryption on all traffic using a master link key.

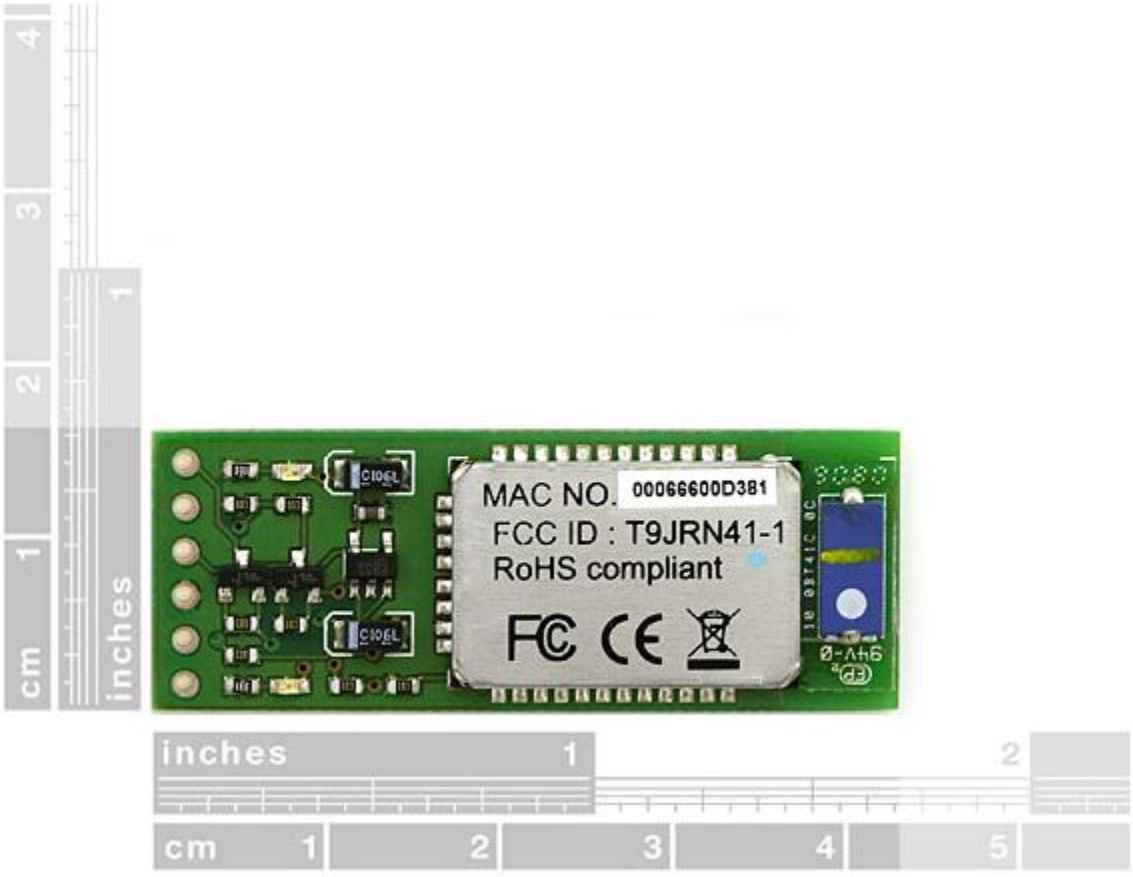
The encryption key uses an internal key generator (KG) to generate an encryption key that will be used by the encryption algorithm. The KG generates steam cipher keys based on the 128-bit Link Key, a 128-bit random number (EN_RAND), and the 96-bit ACO value carried over from the linking procedure. Using the steam cipher, E0, a key stream output is XOR-ed with the payload and sent to the receiving device. The key stream is generating using linear feedback shift registers (LFSR). For inputs, the BD_ADDR, EN_RAND, a slot number, and the encryption key are used as inputs and combined to initialize the LFSRs before transmissions. The slot number changes with each packet as well as the steam cipher.



Bluetooth Encryption

Power Management

Bluetooth has a variety of power saving features in order to improve reliability and versatility in the field. One method for reducing power usage is the use of radio link power control through the negotiation and adjustment of radio power according to the signal strength measurement between devices. Known as the Received Signal Strength Indication (RSSI), two networked devices can request each other to readjust their relative radio power level in order to conserve power as well as improved signal characteristics by limiting their radio frequencies to a certain range. Recent improvements to the data throughput through EDR also improves power consumption by reducing the duty cycle of devices even though more power is consumed from transmissions since transmissions times are significantly reduced.

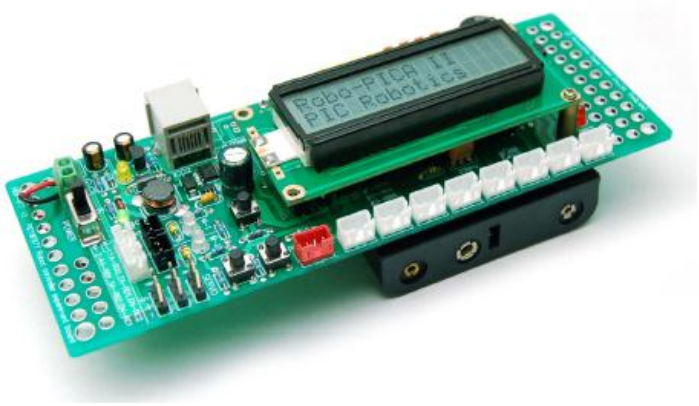


Blue Smirf Gold Module

PIC Microcontroller Board

All control, detection and communication logic is handled by integrated PIC16F887 microcontroller. BluTank utilizes development board with PIC microcontroller paired with multiple external devices. Consequently, it reduced time and cost associated with building that system from raw parts. Primary external components consist of:

- ICD2 Programming Interface
- I^2C Bus Interface
- Driving DC Motor Circuit (via H-Bridge)
- Digital and Analog I/O Ports
- Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter (EUSART)
- LCD Module Interface
- Piezo speaker
- Digital and Analog I/O Ports
- Driving Servo Motor Circuit
- 20 MHz Oscillator
- 5 VDC Switching Power Supply

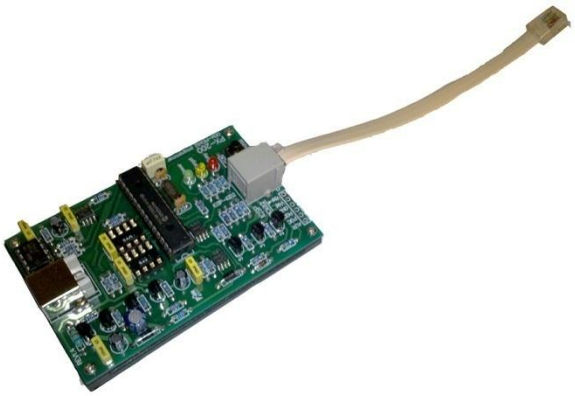


These peripherals supplement capabilities built in the microcontroller. This robot utilizes less than half of PIC potential; however, all external components were utilized except for I^2C Bus and Servo motor circuits. The compiler, MikroC Pro from MikroElectronica, provided general libraries that allowed easy programming of the chip without going too much into detail about fundamentals. Special registers, semi-direct pin access and aliasing enabled highly optimized coding and control behavior with minimal resource use. Compiler reported 38% of ROM and 13% of RAM usage.

PIC Microcontroller Board PIC16F887

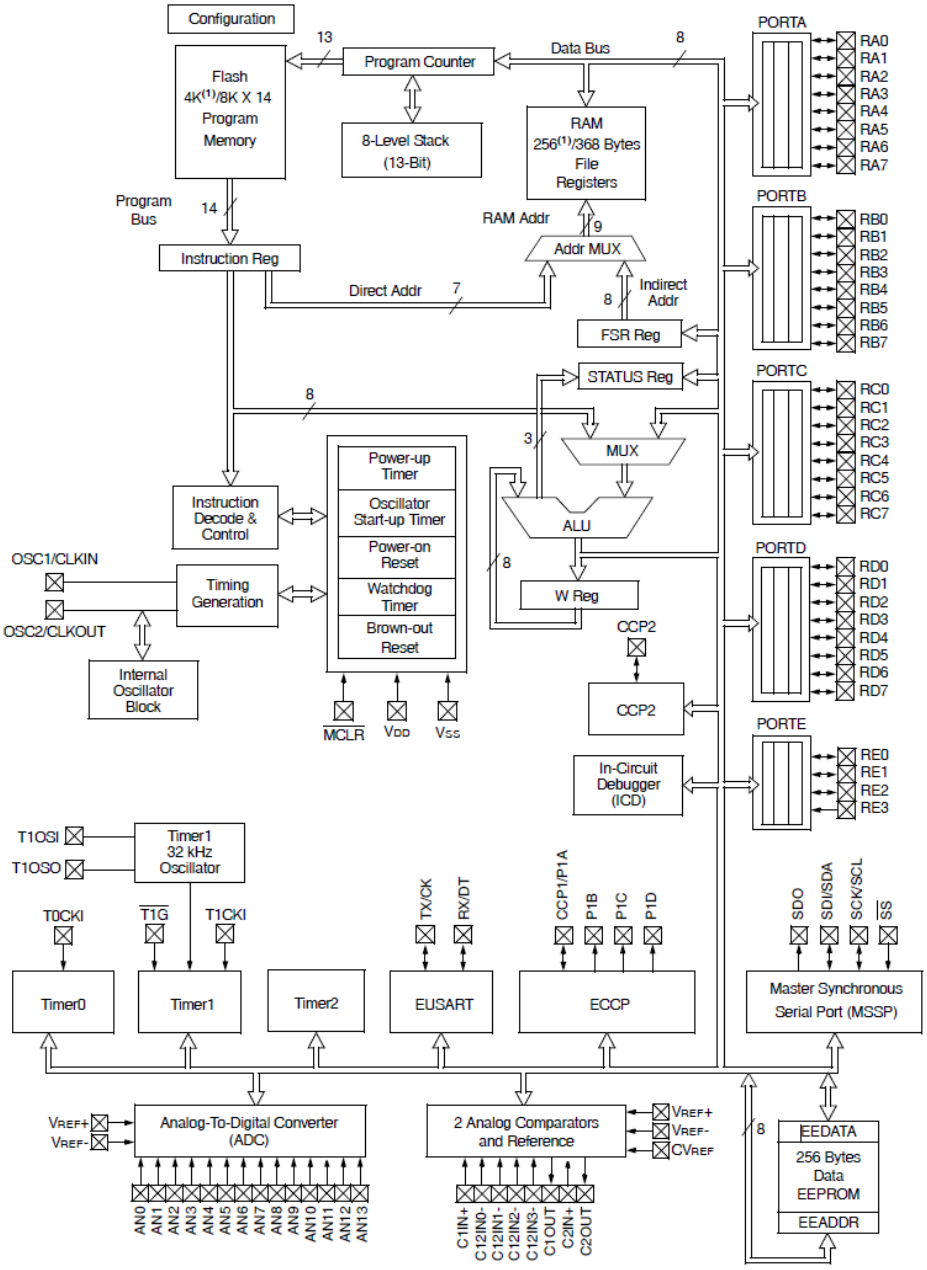
The microcontroller of choice was PIC due to its simplicity, highly developed C libraries and build-in hardware. The chip is all contained package which only required LCD display circuitry and H-bridge to operate efficiently. The microchip on our development board uses external 20 MHz clock. PIC16F887 primary features that were used in this project are outlined below:

- Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter (EUSART)
- Analog-To-Digital Converter
- Pulse-Width Modulation



The programming is performed using ICD2 Interface which allows complete chip programming but also allows reading microcontrollers memory using connected computer. The In-Circuit-Debugger together with mikroC Pro software enabled rapid programming of the chip and greatly reduced debugging time. Every change of code involved complete wipe and reload of flash memory located on board of the chip. In summary, PIC in contrary to competitors offers best all-in-one package.

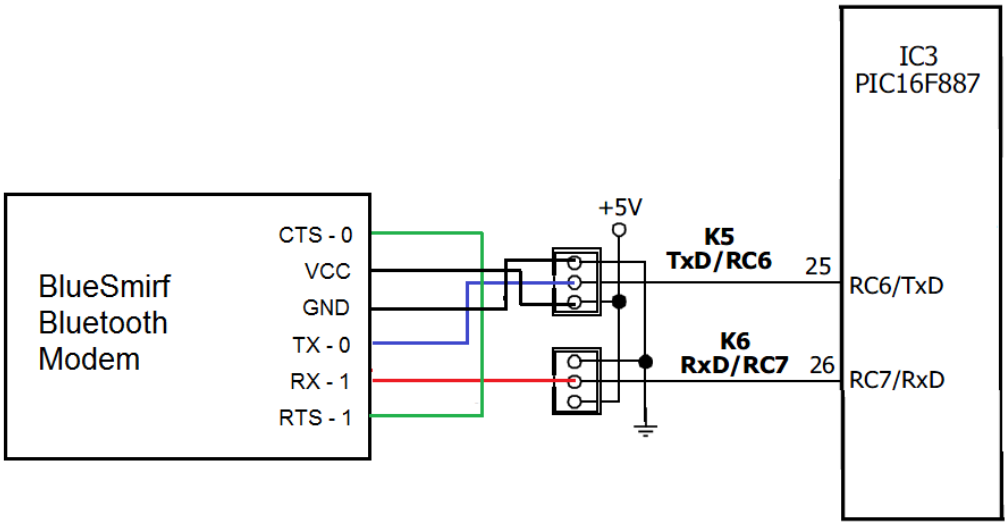
MikroC Pro compiler has extensive support for PIC libraries and allows maximum utilization of the device in the most effective way. Consequently, the entire code for microcontroller was less than 500 lines of C code which yielded around 2000 lines of assembly code. That limited number of required instructions can be attributed to extensive hardware support.



Program Memory Type	Flash
Program Memory (KB)	14
CPU Speed (MIPS)	5
RAM Bytes	368
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	1 CCP, 1 ECCP
Timers	2 x 8-bit, 1 x 16-bit
ADC	14 ch, 10-bit
Comparators	2
Temperature Range (C)	-40 to 125
Operating Voltage Range (V)	2 to 5.5
Pin Count	40

PIC Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter (EUSART)

The most essential interface of BluTank is presence of EUSART. Although PIC includes enhanced version, only the simple UART function was utilized. Universal Asynchronous Receiver/Transmitter allows PIC microcontroller to operate in purely stateless mode. In other words, connection and uplink is maintained solely by BlueSmirf modem and PIC has nothing to do with it other than to receive and send bytes. However, BT modem has capability of external control, but for simplicity, those pins are bridged as seen in schematic below. UART Interface also provides VDD and GND pins allowing connecting by utilizing single split connection, again prompting simplicity



UART is implemented using hardware registers to store TX and RX bytes, set of various readiness bits and configuration registers. EUSART implementation is greatly expanded but UART can be implemented by simply setting the baud rate and pooling TX and RX readiness bits. In this design, UART is initialized at 9600 baud rate, the lowest supported by hardware. The byte transmission is initialized by checking for idle bit. If it is set, byte to be sent is written to output register and UART protocol handles from there and blocks by unsetting idle bit, consequently, this design will be busy polling till UART is ready. Upon completed transmission, idle bit is again set and process repeats. Receiving bytes is also

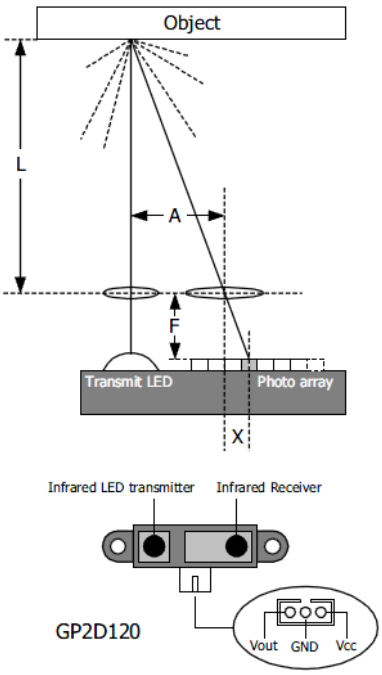
handled in similar fashion. When ready bit is set, byte can be read from input register. Viewing register also unblocks the receiver to allow reading of additional bytes. UART is highly limited in features as it doesn't have effective error detection and control; however EUSART carries excessive complexity needed to accomplish simple task. The data exchange between remote and BluTank involves only single ASCII bit and special byte. MikroC UART library allows reading and writing to UART in single line of code.

PIC Analog-To-Digital Converter and Distance Sensor

The BluTank utilizes SHARP analog IR distance sensor to avert possible front-end collisions. In addition it is the only analog circuit and as a result it doesn't contest over PIC's single 14 channel, 10 bit Analog-to-Digital converter. Consequently, one channel is permanently set for ADC. Since, PIC is an 8 bit device; the converted result is stored in two 16bit registers, prior to any use, it must be converted again to become a single variable. Then it is compared against critical value to see if collision is imminent.

Upon startup, ADC is initialized in similar fashion as all other pins. The first lines of BluTank code initialize individual pins or their groups. Most pins have two configuration bits, first sets input or output and next sets analog or digital. Every binary change to the bit condition is followed by 20 ms delay to allow full cycling. ADC needs additional configuration in special registers. It stores type of reference voltage, output format, clock sampling, and input channel and finally, it enables the ADC. Since no interrupts are used in this development, busy polling for conversion is used. However, it is accomplished quick enough to not have any functional effect on BluTank. When conversion ready bit is set, the result is combined in single variable.

Distance sensor utilizes infrared light to measure distance to facing object. The data is reported to microcontroller in form of variable voltage. That's why ADC is used to digitalize the result. The fundamental parts of distance sensor consist of IR transmitter and receiver. Transmitter sends forward infrared beam perpendicular to the senor. Adjacent receiver accepts reflected light and passes it through its lens. The array of photosensitive sensors behind those lens activate and offset from center of lens and light intensity determine the distance, as seen on pictures on the right. The shorter the distance, the higher light intensity mean higher output voltage and higher conversion value. The obvious drawback of the sensor is limited range and scope. SONAR or LIDAR would offer better detection and would be less prone to error; however they would also mean increased costs.

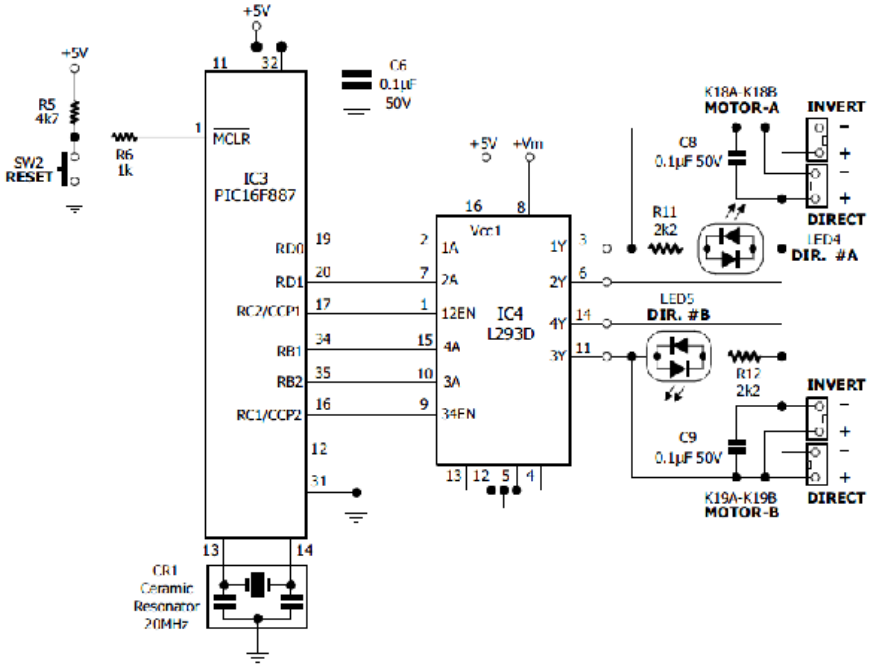


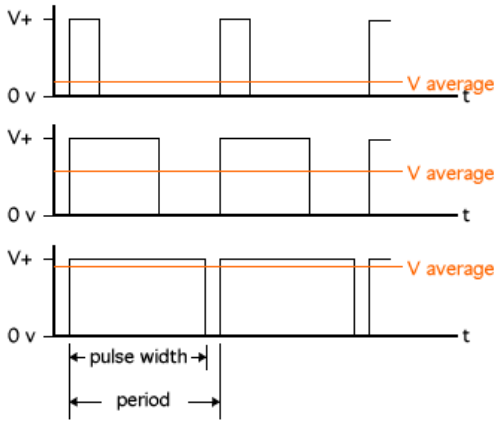
PIC LCD Circuit, Pulse-Width Modulation and H-Bridge

The BluTank offers user feedback through LCD screen mounted on the top. PIC microcontroller interfaces with this 16 character, 2 row display using standard 6 digital output pins following HD44780 compliant standards. Only four bits are used to transmit data while remaining two bits enable the module and register select. The standard usually follows eight data bits, but due to the size of the display, full array is not necessary. MikroC Pro contains comprehensive LCD library which upon initialization only requires HD44780 pin mapping and offers dozen of display functions.

The feature in PIC second to UART is PWM. It is essential to control the speed of two motors used in BluTank. Pulse-Width modulation is an effective way to provide analog output in digital systems without the need for separate digital to analog circuit. The principle of PWM operation in DC systems is

rather simple. In order to provide variable voltage output, PIC microcontroller is paired with H-Bridge to not only translate pulsating voltage but also provide current needed to drive DC motors. PWM sends logical truth as pulses of specific width. The analog voltage is obtained by H-bridge which compares the pulse against the period as average. Consequently, longer pulse yields higher analog voltage and vice





versa.

PIC microcontroller offers hardware PWM implementation that can effortlessly accessed using PWM libraries. There are two separate PWM channels. The fundamental functions are outlined below:

- PWM_Init(x) Sets period of PWM to x Hz
- PWM_Start() Begins PWM operation
- PWM_Stop() Stop PWM operation
- PWM_Set_Duty(x) Sets pulse width of PWM, if x = (0 = 0%, 255 = 100%)

H-Bridge also accepts 2 bits per motor in addition to PWM input. Those bits determine direction or state of the motors. When both bits are set or unset, then motor is locked or in brake mode. Alternating bit values determine direction. When PWM is stopped, motors move freely.

BluTank Controls

The BluTank was designed to operate continuously via predetermined path or instantaneously using remote to give it the most flexibility and minimal user input. Consequently, remote control operation can be a bit confusing at first even for experienced R/C operator.

The design for motor control revolves around a special byte, which has dual purpose. First of all, it is used to store bits regarding BluTank status, such as speed and direction and also possible collision detection. Second, this byte is sent to remote control every time a change in status is performed allowing user to see motor speeds right on his remote application in addition to LCD display on vehicle itself.

Transmission and Special Status Byte

All control, detection and communication revolve about those 8 bits. During development it became clear that effective communication between BluTank and Android application is essential. A custom protocol was proposed, but it would require extra code on both transmission ends to handle it successfully. Since PIC is 8-bit system and UART transmits in 8-bits, it became clear that single byte for receiving and transmitting is the best option.

All input to BluTank is sent as single 8-bit ASCII character. Since design is stateless, PIC only waits for byte to be ready from UART and the processes it. There are number of different “keystrokes” which are checked by switch logic statement and executed accordingly. After each process, PIC sends back a status/special byte via UART to indicate new conditions. Status byte is a key variable as it is used in BluTank programming. All changes to operation revolve about reading or modifying individual bits.

Status Byte							
0	1	2	3	4	5	6	7
Dir	Speed		Dir	Speed		Collision	Reserved
Motor A			Motor B				

The first three bits deal with Motor A, followed by similar three bits for Motor B. Seventh bit is set to indicate dangerous proximity and last bit is reserved for future feature or parity check. Three bits in motor group indicate direction and speed. Direction bit means either forward or reverse. Speed bits indicate one of three speeds: minimum, medium and maximum. The PWM duty for each is 86%, 92% and 100% respectfully. One may think that those values seem high, however, as motors are relatively low quality, any lower PWM value would prevent from motor operation. Collision bit is set when distance to obstacle is too close. When bit is on, no forward motion is allowed to prevent operator from hitting any objects. In next sections, Status Byte is referred as SB for syntax simplicity, but it is defined as global variable.

Motor Controls

In the C program, there are eleven functions dealing with motor operations. Android application executes individual functions to accomplish desired tasks. Those modes will be explored in next section.

- Motor_Init()

  This function is called upon BluTank initialization. It sets PWM duty cycle to 5 kHz and configures H-Bridge directional bits to digital output and unsets them.

- fwd_a() fwd_b()

  These functions check global Status Byte for respective motor and increments speed by one. In another words on every call it will bring motor from max reverse speed to max forward. When

maximum speed is reached and function is called. It will give audible indication. It is using bitmasks to determine previous condition and uses bitwise logic to swap bit values.

- rev_a() rev_b()

  These functions are essentially the same as previous but operate in reverse. All four functions will cycle motor speeds individually as seen below (extremes indicate beeps):

  ↔ Max_Rev ↔ Med_Rev ↔ Min_Rev ↔ STOP ↔ Min_Fwd ↔ Med_Fwd ↔ Max_Fwd ↔

- quick_fwd() quick_rev() quick_left() quick_right()

  These functions are used for instantaneous controls. Once called, they use copy of global Status Byte and set first 6 bits for desired maximum direction. Then drive function is called using modified Status Byte. Following preset delay, drive function is called again but using unmodified global Status Byte allowing BluTank to resume its previous mode of operation. Forward function is also checking proximity bit and if it is set, prevents forward movement and gives audible alert.

- stop_ab()

  This function simply sets status bits to STOP settings

- drive (SB)

  This function accepts Status Byte formatted variable and executes its status. With the exception of instantaneous motion, it is always called with global Status Byte. When executed, it checks SB using masks to determine condition and sets PWM duty together with directional H-Bridge bits. Also it updates motor speeds on LCD display.

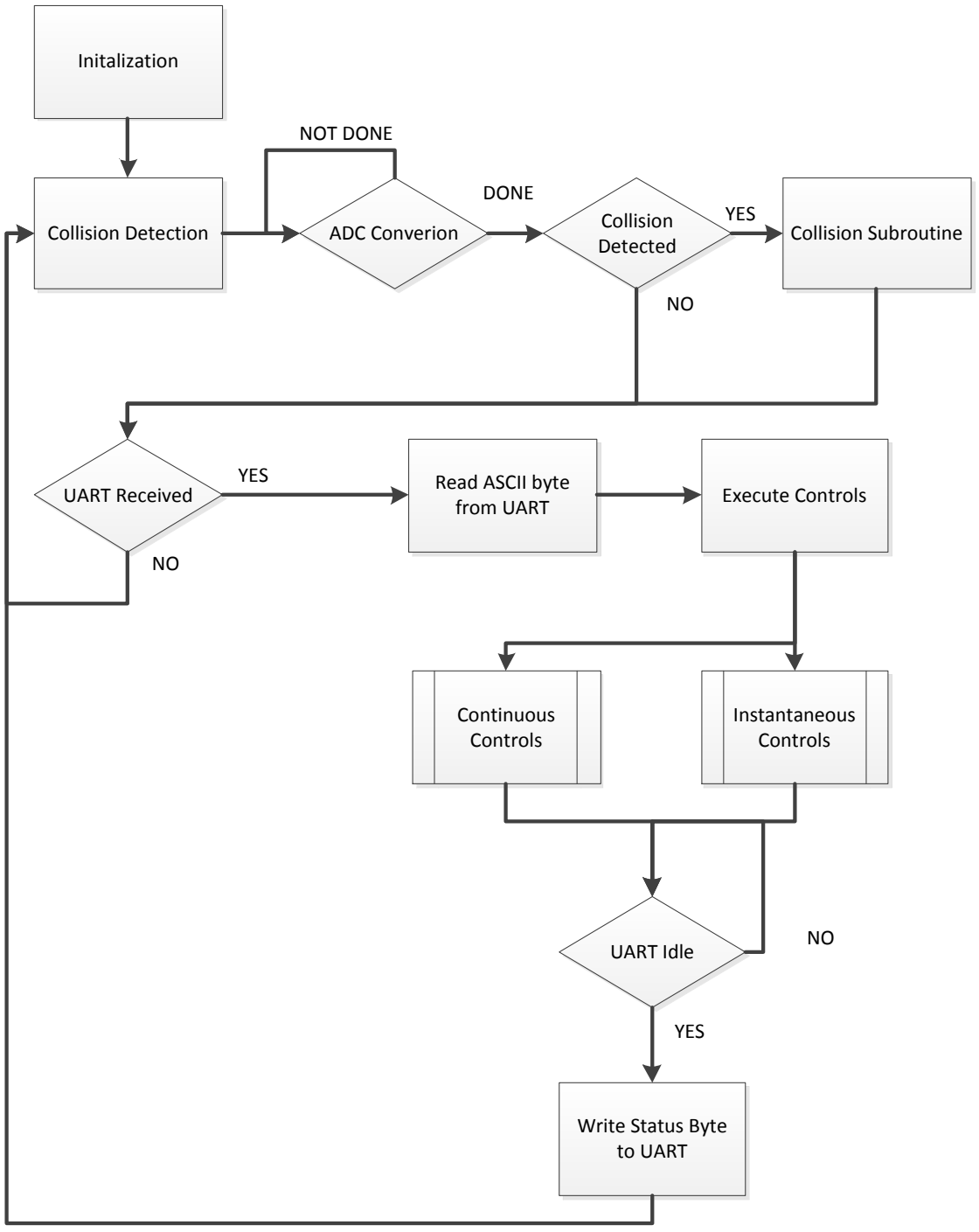
Instantaneous vs. Continuous Operation and Collision Detection

One of the greatest features of BluTank is dual steering mode. Controls present on Android application allow vehicle to be operated in any mode with the simple touch. Instantaneous operations simply represent means to execute rapid movement for short duration of time. Continuous operation involves setting particular speed of a motor and running it indefinitely without further input. When incoming ASCII byte is processed, switch statement selects different motor functions. Keys are not case sensitive. Key 'H' produces audible alert analog to car's horn.

ASCII Key	Function	Motor A	Motor B
W	Increase speed (Continuous)	fwd_a()	fwd_b()
S	Decrease speed (Continuous)	rev_a()	rev_b()
A	Bear Left (Continuous)	rev_a()	fwd_b()
D	Bear Right (Continuous)	fwd_a()	rev_b()
Q	Stop all motors	stop_ab()	
I	Max forward for short duration	quick_fwd()	
K	Max reverse for short duration	quick_rev()	
J	Turn Left	quick_left()	
L	Turn Right	quick_right()	

Since instantaneous commands end with resuming of previous state, it allows operator to quickly correct direction of moving vehicle, for example to avoid obstacles. Continuous commands allow user to set vehicle in motion without further interaction. It could be used to set BluTank to go in circles or travel in particular direction indefinitely. However, since it is programmed in stateless mode, loss of connectivity due to range would not affect the status of the vehicle and only limitation would be obstacles or insufficient power.

Another useful feature is collision detection. BluTank uses IR to detect dangerous proximity and if detected, it will execute recovery subroutine. It begins with setting the collision flag in Status Byte and stopping the vehicle. Then for brief moment is driven in reverse and again stopped. Status Byte is then sent to the user to notify him of detected collision. By that time, vehicle should be far enough to allow user any recovery maneuvers. If vehicle is stopped and collision is still imminent, LED will indicate that on the BluTank and will prevent user from going forward as safety.


Initalization
NOT DONE
Collision Detection
ADC Converion
DONE
Collision Detected
YES
Collision Subroutine
NO
UART Received
YES
Read ASCII byte from UART
Execute Controls
NO
Continuous Controls
Instantaneous Controls
UART Idle
NO
YES
Write Status Byte to UART

Android Software Stack

Android is a software stack (usually just called an operating system) for mobile devices that includes an operating system, middleware, and key applications. It uses a modified version of the Linux kernel and was initially developed by Android Inc., which was later purchased by Google, and now developed by the Open Handset Alliance. The Linux kernel used does not have a native X Window System nor does it support the full set of standard GNU libraries like the system libraries such as GNU C Library, which makes it difficult to reuse existing Linux applications or libraries on Android. Android allows developers to write managed code in Java (a programming language), controlling the device using the Google-developed Java libraries, the software development kit (SDK). The SDK provides the tools and APIs (Application Programming Interface) necessary to develop applications that run on Android-powered devices.

The Android software stack can use VGA, 2D graphics library, and 3D graphics library based on OpenGL ES 2.0 specifications. It uses SQLite as the Database Software for data storage purposes. All the software written in Java can be compiled to be executed in the Dalvik virtual machine, which is a specialized virtual machine implementation designed for mobile device use, however not technically a standard Java Virtual Machine. The Android software stack does not support J2ME (Java Platform, Micro Edition), like some other mobile operating systems. It supports various audio and video formats including H.264 (in 3GO or MP4 container) and MP3.

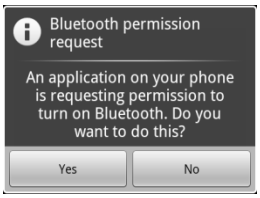
Libraries written in C and other languages can be compiled to ARM native code and installed using the Android Native Development kit. Native classes can be called from Java code running under the Dalvik VM using special calls which are a part of the standard Android Java classes. Completed applications can be compiled and installed using traditional development tools, however, the ADB tools given by Google gives a root shell under the Android Emulator which allows native ARM code to be

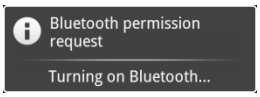
uploaded and executed easily. ARM code can be compiled using GCC on a standard computer. Running native code is quite complicated since Android uses a non-standard C library (Bionic).

Android Releases

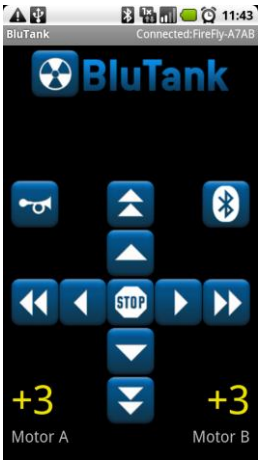
On April 30, 2009, the first official release of Android (1.5 Cupcake) was marketed on the T-Mobile G1 as Google's flagship product. The 1.5, which is based on Linux Kernel 2.6.27) release came with several new features and UI updates from the 1.0 done by Android, Inc., such as the ability to record and watch videos with the camcorder mode, uploading videos to YouTube directly from the phone, a software keyboard with an "autocomplete" features, Bluetooth A2DP support (used for high quality Bluetooth audio), auto-connect with Bluetooth headsets that are already paired, animated screens, and an expanded copy and paste that includes web pages. The 1.6 update (Donut), based on Linux Kernel 2.6.29 used currently (as of May 2010) on the T-Mobile G1 and HTC Hero, released on September 15, 2009 included more updates to make the Android devices more user friendly which included an updated voice search with faster responses and deeper integration with native applications, updated support for various technology for CMDA/EVDO, 802.1x, VPN, Gestures, and Text-to-Speech engine, support for WVGA resolutions, and overall speed improvements for searching and the camera. On October 26, 2009, Android 2.0 (Éclair) which is based on Linux Kernel 2.6.29 was released and contained a lot of changes which included the Bluetooth API used for BluTank, optimized hardware speeds, support for more screen sizes and resolutions, a new browser with HTML5 support, digital zoom, and live wallpapers. The 2.0.1 version was released on December 3, 2009 with 2.1 following on January 23, 2010, both version still based off the 2.6.29 Linux Kernel.

The Android operating system used for the BluTank project is 1.6 and 2.1. Currently, the application only works for 2.0.1 and above, which includes the 2.1 release. For 1.6, a custom Bluetooth HyperTerminal-like interface is setup to make sure everything works without issues. The icons are custom made and come in a variety of colors, but for our project, we decided that the shades of blue were the best option for the application.





When you open the BluTank application, if Bluetooth is not enabled, you will be requested to enable the Bluetooth adapter. Once the adapter is enabled, a screen will show up showing that the Bluetooth adapter is being turned on. After the Bluetooth adapter is turned on, the menu shows up. The menu is a simple one time connect method on the top right of the screen.



Once connected, the application will display the name of the connected device, in our case, FireFly-A7AB. The menu is simple with the double triangles representing quick move and the single triangles representing a constant movement in a specific direction. The horn button creates a little honk and the stop button actually stops the vehicle from moving.

Code Information

Bluetooth Service on Android

```
package Blue.Car;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
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.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

public class BluetoothService {
    private static final String TAG = "BluTank";
    private static final boolean D = true;

    private static final String NAME = "BluTank";

    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    public static final int STATE_NONE = 0;
    public static final int STATE_LISTEN = 1;
    public static final int STATE_CONNECTING = 2;
    public static final int STATE_CONNECTED = 3;

   public BluetoothService(Context context, Handler handler) {
      mAdapter = BluetoothAdapter.getDefaultAdapter();
      mState = STATE_NONE;
      mHandler = handler;
    }

    private synchronized void setState(int state) {
      if (D) Log.d(TAG, "setState() " + mState + " -> " + state);
      mState = state;

      // Give the new state to the Handler so the UI Activity can update
      mHandler.obtainMessage(BlueCar.MESSAGE_STATE_CHANGE, state, -1).sendToTarget();
    }

    public synchronized int getState() {
      return mState;
    }

    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);
  }

  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);
  }

  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();

    // Send the name of the connected device back to the UI Activity
    Message msg = mHandler.obtainMessage(BlueCar.MESSAGE_DEVICE_NAME);
    Bundle bundle = new Bundle();
    bundle.putString(BlueCar.DEVICE_NAME, device.getName());
    msg.setData(bundle);
    mHandler.sendMessage(msg);

    setState(STATE_CONNECTED);
  }

  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);
  }

  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);
    }

    private void connectionFailed() {
      setState(STATE_LISTEN);

      // Send a failure message back to the Activity
      Message msg = mHandler.obtainMessage(BlueCar.MESSAGE_TOAST);
      Bundle bundle = new Bundle();
      bundle.putString(BlueCar.TOAST, "Unable to connect device");
      msg.setData(bundle);
      mHandler.sendMessage(msg);
    }

    private void connectionLost() {
      setState(STATE_LISTEN);

      // Send a failure message back to the Activity
      Message msg = mHandler.obtainMessage(BlueCar.MESSAGE_TOAST);
      Bundle bundle = new Bundle();
      bundle.putString(BlueCar.TOAST, "Device connection was lost");
      msg.setData(bundle);
      mHandler.sendMessage(msg);
    }

    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 (BluetoothService.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);
      }
    }
}


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
            BluetoothService.this.start();
            return;
        }

        // Reset the ConnectThread because we're done
        synchronized (BluetoothService.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);
        }
    }
}

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[1024];
        int bytes;

        // Keep listening to the InputStream while connected
        while (true) {
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);

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

```
            BREAK;
          }
        }
      }

      PUBLIC VOID WRITE(BYTE[] BUFFER) {
        TRY {
          MMOUTSTREAM.WRITE(BUFFER);

          // SHARE THE SENT MESSAGE BACK TO THE UI ACTIVITY
          MHANDLER.OBTAINMESSAGE(BLUECAR.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);
        }
      }
    }
}
```

BLUETOOTH FRONT END COMMANDS

```
package Blue.Car;

import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.text.Layout;
import android.util.Log;
//import android.view.KeyEvent;
import android.view.Gravity;
import android.view.LayoutInflater;
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.View.OnLongClickListener;
//import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
//import android.widget.Button;
//import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.PopupWindow;
import android.widget.TextView;
import android.widget.Toast;
```

```
public class BlueCar extends Activity {

    // Debugging
    private static final String TAG = "BlueTank";
    private static final boolean D = true;

    // MAC Address of Firefly Modem
    private static final String FIREFLY_MAC = "00:06:66:03:A7:AB";

    // 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 ImageButton forwardButton;
    private ImageButton reverseButton;
    private ImageButton leftButton;
    private ImageButton rightButton;
    private ImageButton honkButton;
    private ImageButton stopButton;
    private ImageButton qfButton;
    private ImageButton qrButton;
    private ImageButton qlButton;
    private ImageButton qrightButton;
    private ImageButton bluetoothButton;
    private ImageButton bluetoothDCButton;
    private ImageButton logoButton;
    private ImageButton logoLONGButton;

    Drawable trollThomas;

    private TextView motorA;
    private TextView motorB;
    private TextView infoText;


    // 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 BluetoothService mChatService = 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;
    }
}

@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() == BluetoothService.STATE_NONE) {
          // Start the Bluetooth chat services
          mChatService.start();
        }
    }
}

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
honkButton = (ImageButton) findViewById(R.id.button_honk);
honkButton.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);
         if (mChatService.getState() != BluetoothService.STATE_CONNECTED) {
         Toast.makeText(BlueCar.this, R.string.not_connected, Toast.LENGTH_SHORT).show();
      }
         else
         {
                  motorA.setText(":)");
                     motorB.setText(":)");
                  String message = "h";
                  sendMessage(message);
         }
   }
});

forwardButton = (ImageButton) findViewById(R.id.button_forward);
forwardButton.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 = "w";
      sendMessage(message);
   }
});
reverseButton = (ImageButton) findViewById(R.id.button_reverse);
reverseButton.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 = "s";
      sendMessage(message);
   }
});
leftButton = (ImageButton) findViewById(R.id.button_left);
leftButton.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 = "a";
      sendMessage(message);
   }
});
rightButton = (ImageButton) findViewById(R.id.button_right);
rightButton.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 = "d";
      sendMessage(message);
   }
});

stopButton = (ImageButton) findViewById(R.id.button_stop);
stopButton.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 = "q";
      sendMessage(message);
   }
});
```

```
qfButton = (ImageButton) findViewById(R.id.button_qf);
qfButton.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 = "i";
      sendMessage(message);
   }
});

qrButton = (ImageButton) findViewById(R.id.button_qr);
qrButton.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 = "k";
      sendMessage(message);
   }
});

qlButton = (ImageButton) findViewById(R.id.button_ql);
qlButton.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 = "j";
      sendMessage(message);
   }
});

qrightButton = (ImageButton) findViewById(R.id.button_qright);
qrightButton.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 = "l";
      sendMessage(message);
   }
});

// Auto-connect to the Firefly MAC address defined above
bluetoothButton = (ImageButton) findViewById(R.id.button_bluetooth);
bluetoothButton.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
        if (mChatService.getState() != BluetoothService.STATE_CONNECTED) {
        // Get the BluetoothDevice object
                BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(FIREFLY_MAC);
                // Attempt to connect to the device
                mChatService.connect(device);
              }
        else
        {
                Toast.makeText(BlueCar.this, R.string.connected, Toast.LENGTH_SHORT).show();
        }
   }
});

bluetoothDCButton = (ImageButton) findViewById(R.id.button_bluetooth);
bluetoothDCButton.setOnLongClickListener(new OnLongClickListener(){
        public boolean onLongClick(View v) {
                //mChatService.stop();
                BlueCar.this.onDestroy();
                return true;
        }
});
```

```
logoButton = (ImageButton) findViewById(R.id.button_logo);
logoButton.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
                        AlertDialog alertDialog = new AlertDialog.Builder(BlueCar.this).create();
                        alertDialog.setTitle("About BluTank");
                        alertDialog.setMessage("" +
                                        "This Senior Design Project II project was created by:\n" +
                                        "Tomasz Glazewski\n" +
                                        "Richard Lau\n" +
                                        "Rijvi Rajib\n" +
                                        "Advisor: Ramesh Karri");
                        alertDialog.setButton("Close", new DialogInterface.OnClickListener() {
                          public void onClick(DialogInterface dialog, int which) {
                            return;
                        } });
                       alertDialog.show();
        }
});

logoLONGButton = (ImageButton) findViewById(R.id.button_logo);
logoLONGButton.setOnLongClickListener(new OnLongClickListener(){
        public boolean onLongClick(final View v) {
                AlertDialog alertDialog = new AlertDialog.Builder(BlueCar.this).create();
                alertDialog.setTitle("Thomas the Tank Engine");
                alertDialog.setMessage("" +
                                        "This Senior Design Project II project was created by:\n" +
                                        "Tomasz Glazewski\n" +
                                        "Richard Lau\n" +
                                        "Rijvi Rajib\n" +
                                        "Advisor: Ramesh Karri");
                alertDialog.setIcon(R.drawable.tomas);
                  alertDialog.setButton("Close", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
                      return;
                    } });
                    alertDialog.setButton2("Troll?", new DialogInterface.OnClickListener() {
                      public void onClick(DialogInterface dialog, int which) {
                                Toast.makeText(BlueCar.this, "THOMAS THE TANK ENGINE IS AWESOME!", Toast.LENGTH_LONG).show();
                                View mainLayout = BlueCar.this.findViewById(R.id.layout_main);
                                trollThomas = BlueCar.this.getResources().getDrawable(R.drawable.trolltomas);
                                mainLayout.setBackgroundDrawable(trollThomas);
                                View forwardB = BlueCar.this.findViewById(R.id.button_forward);
                                forwardB.setVisibility(4);
                                View qforwardB = BlueCar.this.findViewById(R.id.button_qf);
                                qforwardB.setVisibility(4);
                                View leftB = BlueCar.this.findViewById(R.id.button_left);
                                leftB.setVisibility(4);
                                View qleftB = BlueCar.this.findViewById(R.id.button_ql);
                                qleftB.setVisibility(4);
                                View stopB = BlueCar.this.findViewById(R.id.button_stop);
                                stopB.setVisibility(4);
                                View rightB = BlueCar.this.findViewById(R.id.button_right);
                                rightB.setVisibility(4);
                                View qrightB = BlueCar.this.findViewById(R.id.button_qright);
                                qrightB.setVisibility(4);
                                View reverseB = BlueCar.this.findViewById(R.id.button_reverse);
                                reverseB.setVisibility(4);
                                View qreverseB = BlueCar.this.findViewById(R.id.button_qr);
                                qreverseB.setVisibility(4);
                               return;
                    }});
                   alertDialog.show();
                return true;
        }
});
```

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

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

@Override
public synchronized void onPause() {
    super.onPause();
    if(D) Log.e(TAG, "- ON PAUSE -");
}

@Override
public void onStop() {
    super.onStop();
    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() != BluetoothService.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 BluetoothService 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 static int getBit(byte[] data, int pos) {
    int posByte = pos/8;
    int posBit = pos%8;
    byte valByte = data[posByte];
    int valInt = valByte>>(8-(posBit+1)) & 0x0001;
    return valInt;
 }
```

```
private static String getSpeed(String data) {
   if (data.equals("111")) { return "+3"; }
   else if (data.equals("110")) { return "+2"; }
   else if (data.equals("101")) { return "+1"; }
   else if (data.equals("100")) { return "0"; }
   else if (data.equals("001")) { return "-1"; }
   else if (data.equals("010")) { return "-2"; }
   else if (data.equals("011")) { return "-3"; }
   else { return data; }
 }

// The Handler that gets information back from the BluetoothService
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 BluetoothService.STATE_CONNECTED:
            mTitle.setText(R.string.title_connected_to);
            mTitle.append(mConnectedDeviceName);
            mConversationArrayAdapter.clear();

            // Initialize motor speed indicators to 0
            motorA = (TextView) findViewById(R.id.TextView03);
            motorA.setText("0");
            motorB = (TextView) findViewById(R.id.TextView04);
            motorB.setText("0");
            infoText = (TextView) findViewById(R.id.text_info);
            infoText.setText("");
            break;
         case BluetoothService.STATE_CONNECTING:
            mTitle.setText(R.string.title_connecting);
            break;
         case BluetoothService.STATE_LISTEN:
         case BluetoothService.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;
         // construct a string from the valid bytes in the buffer
         //String readMessage = new String(readBuf, 0, msg.arg1);
         //int readMessage;
         //readMessage = getBit(readBuf, 0);
         String firstHalfByte, secondHalfByte, motorAspeed, motorBspeed, motors;
         int bit0 = getBit(readBuf, 0);
         int bit1 = getBit(readBuf, 1);
         int bit2 = getBit(readBuf, 2);
         int bit3 = getBit(readBuf, 3);
         int bit4 = getBit(readBuf, 4);
         int bit5 = getBit(readBuf, 5);
         int bit6 = getBit(readBuf, 6);

         firstHalfByte = ""+bit0+bit1+bit2;
         secondHalfByte = ""+bit3+bit4+bit5;

         motorAspeed = getSpeed(firstHalfByte);
```

```
            motorBspeed = getSpeed(secondHalfByte);
            motors = "Motor A:" + motorAspeed + " Motor B:" + motorBspeed;

            mConversationArrayAdapter.add(mConnectedDeviceName+":  " + motors);
            motorA.setText(motorAspeed);
            motorB.setText(motorBspeed);

            if (bit6 == 1)
            {
               infoText.setText("Collision imminent. Vehicle stopped.");
            }
            else
            {
               infoText.setText("");
            }

            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;
    }

}
```

PIC Tank Controls

```
char *text = "BluTank";  // Define message
unsigned stats  =  0b10010000;
unsigned old_stats;
char key = 'X';
int motor_init_=0;
int adc;

//speed levels
const unsigned maximum = 255;
const unsigned medium = 235;
const unsigned minimum = 220;
//time constant
const unsigned qtime = 200;
//distance konstant
const int max_d = 555;

//=====================================================================//
//================== BIT ASSIGNMENTS==================================//
//=====================================================================//
// Lcd pinout settings
 sbit LCD_RS at RD2_bit;
 sbit LCD_EN at RD3_bit;
 sbit LCD_D7 at RD7_bit;
 sbit LCD_D6 at RD6_bit;
 sbit LCD_D5 at RD5_bit;
 sbit LCD_D4 at RD4_bit;
// Pin direction
 sbit LCD_RS_Direction at TRISD2_bit;
 sbit LCD_EN_Direction at TRISD3_bit;
 sbit LCD_D7_Direction at TRISD7_bit;
 sbit LCD_D6_Direction at TRISD6_bit;
 sbit LCD_D5_Direction at TRISD5_bit;
 sbit LCD_D4_Direction at TRISD4_bit;

 // speed levels: 255, 235, 220, 0

//=====================================================================//
//================== Speed Change ====================================//
//=====================================================================//
void Motor_Init() {
 if (motor_init_==0)         // First time ?
 {
   motor_init_=1;           // Status
```

```
  ANSELH = 0;
  TRISB1_BIT=0;            // MOTOR B 2A
  TRISB2_BIT=0;            // MOTOR B 2B
  TRISD0_BIT=0;            // MOTOR A 1A
  TRISD1_BIT=0;            // MOTOR A 1B
  TRISC1_BIT=0;
  TRISC2_BIT=0;
  RB1_BIT=0;
  RB2_BIT=0;
  RD0_BIT=0;
  RD1_BIT=0;
  PWM1_INIT(5000);         // INITAIL PWM 1E
  PWM2_INIT(5000);         // INITAIL PWM 2E
 }
}

VOID DRIVE ( UNSIGNED CONTROLS ) {

   DELAY_MS(20);
   // SET FWD/REV, MOTOR A
   IF (( CONTROLS & 0B10000000 )== 0B10000000) {
     RD0_BIT =0;

     IF (( ~CONTROLS & 0B01100000) == 0B01100000) {
       RD1_BIT =0;
       LCD_CHR(2,1,'S');
       }
     ELSE IF ((STATS & 0B00000010) == 0B00000010) {
        RD1_BIT =0;
        SOUND_PLAY(1400,100);
        DELAY_MS(50);
        SOUND_PLAY(1450,100);
        DELAY_MS(50);
        SOUND_PLAY(1500,100);
        DELAY_MS(50);
     }
     ELSE {
        RD1_BIT =1;
        LCD_CHR(2,1,'F');
        }
   }
   ELSE {
     RD0_BIT =1;
     RD1_BIT =0;
     LCD_CHR(2,1,'R');
   }
   // SET FWD/REV, MOTOR B
   IF (( CONTROLS & 0B00010000) == 0B00010000) {
     RB1_BIT =0;

     IF (( ~CONTROLS & 0B00001100) == 0B00001100) {
       RB2_BIT =0;
       LCD_CHR(2,4,'S');
       }
     ELSE IF ((STATS & 0B00000010) == 0B00000010) {
        RB2_BIT =0;
        SOUND_PLAY(1400,100);
        DELAY_MS(50);
        SOUND_PLAY(1450,100);
        DELAY_MS(50);
        SOUND_PLAY(1500,100);
        DELAY_MS(50);
     }
     ELSE {
        RB2_BIT =1;
        LCD_CHR(2,4,'F');
```

```
        }
    }
    ELSE {
      RB1_BIT =1;
      RB2_BIT =0;
      LCD_CHR(2,4,'R');
    }
    // SET SPEED, MOTOR A
    IF (( CONTROLS & 0B01100000) == 0B01100000) {
      PWM1_START();
      PWM1_SET_DUTY(MAXIMUM);
      LCD_CHR(2,2,'3');
      }
    ELSE IF (( CONTROLS & 0B01000000) == 0B01000000) {
       PWM1_START();
       PWM1_SET_DUTY(MEDIUM);
       LCD_CHR(2,2,'2');
       }
    ELSE IF (( CONTROLS & 0B00100000) == 0B00100000) {
       PWM1_START();
       PWM1_SET_DUTY(MINIMUM);
       LCD_CHR(2,2,'1');
       }
    ELSE {
       PWM1_STOP();
       PWM1_SET_DUTY(0);
       LCD_CHR(2,2,'0');
       }

    // SET SPEED, MOTOR B
    IF (( CONTROLS & 0B00001100) == 0B00001100) {
      PWM2_START();
      PWM2_SET_DUTY(MAXIMUM);
      LCD_CHR(2,5,'3');
      }
    ELSE IF (( CONTROLS & 0B00001000) == 0B00001000) {
      PWM2_START();
      PWM2_SET_DUTY(MEDIUM);
      LCD_CHR(2,5,'2');
      }
    ELSE IF (( CONTROLS & 0B00000100) == 0B00000100) {
      PWM2_START();
      PWM2_SET_DUTY(MINIMUM);
      LCD_CHR(2,5,'1');
      }
    ELSE {
       PWM2_STOP();
       PWM2_SET_DUTY(0);
       LCD_CHR(2,5,'0');
       }
}

VOID FWD_A() {
    IF (( STATS & 0B10000000) == 0B10000000 ) {  // MOTOR MOVING FORWARD
      IF (( STATS & 0B01000000) == 0B01000000 ) {
        IF (( STATS & 0B00100000) == 0B00100000 ) SOUND_PLAY(2200,100); // MAX SPEED ALREADY REACHED
        ELSE {  // MED -> MAX
           STATS |= 0B00100000;
           }
        }
      ELSE {
        IF (( STATS & 0B00100000) == 0B00100000 ) { // MIN -> MED
          STATS ^= 0B01100000;
          }
        ELSE { // STOP -> MIN
           STATS |= 0B00100000;
```

```
          }
        }
      }
      ELSE { // MOTOR MOVING BACKWARDS
       IF (( STATS & 0B01000000) == 0B01000000 ) {
         IF (( STATS & 0B00100000) == 0B00100000 ) { // MAX -> MED
           STATS ^= 0B00100000;
         }
         ELSE {  // MED -> MIN
          STATS ^= 0B01100000;
         }
       }
      ELSE { // MIN -> STOP
         STATS ^= 0B10100000;
         }
       }
}
VOID REV_A() {
   IF (( STATS & 0B10000000) == 0B10000000 ) {  // MOTOR MOVING FORWARD
     IF (( STATS & 0B01000000) == 0B01000000 ) {
       IF (( STATS & 0B00100000) == 0B00100000 ) { // MAX -> MED
           STATS ^= 0B00100000;
          }
       ELSE {  // MED -> MIN
          STATS ^= 0B01100000;
          }
       }
      ELSE {
       IF (( STATS & 0B00100000) == 0B00100000 ) { // MIN -> STOP
         STATS ^= 0B00100000;
         }
       ELSE { // STOP -> REV MIN
          STATS ^= 0B10100000;
          }
       }
      }
      ELSE { // MOTOR MOVING BACKWARDS
       IF (( STATS & 0B01000000) == 0B01000000 ) {
         IF (( STATS & 0B00100000) == 0B00100000 ) SOUND_PLAY(2200,100); // MAX REVERSE REACHED
         ELSE {  // MED -> MAX
          STATS ^= 0B00100000;
         }
       }
      ELSE { // MIN -> MED
         STATS ^= 0B01100000;
         }
       }
}
VOID FWD_B() {
   IF (( STATS & 0B00010000) == 0B00010000 ) {  // MOTOR MOVING FORWARD
     IF (( STATS & 0B00001000) == 0B00001000 ) {
       IF (( STATS & 0B00000100) == 0B00000100 ) SOUND_PLAY(2100,100); // MAX SPEED ALREADY REACHED
       ELSE {  // MED -> MAX
          STATS |= 0B00000100;
          }
       }
      ELSE {
       IF (( STATS & 0B00000100) == 0B00000100 ) { // MIN -> MED
         STATS ^= 0B00001100;
         }
       ELSE { // STOP -> MIN
          STATS |= 0B00000100;
          }
       }
      }
      ELSE { // MOTOR MOVING BACKWARDS
```

```
      IF (( STATS & 0B00001000) == 0B00001000 ) {
        IF (( STATS & 0B00000100) == 0B00000100 ) { // MAX -> MED
          STATS ^= 0B00000100;
        }
        ELSE {  // MED -> MIN
          STATS ^= 0B00001100;
         }
       }
     ELSE { // MIN -> STOP
        STATS ^= 0B00010100;
        }
      }
}
VOID REV_B() {
   IF (( STATS & 0B00010000) == 0B00010000 ) {  // MOTOR MOVING FORWARD
     IF (( STATS & 0B00001000) == 0B00001000 ) {
       IF (( STATS & 0B00000100) == 0B00000100 ) { // MAX -> MED
           STATS ^= 0B00000100;
           }
       ELSE {  // MED -> MIN
          STATS ^= 0B00001100;
          }
       }
     ELSE {
       IF (( STATS & 0B00000100) == 0B00000100 ) { // MIN -> STOP
         STATS ^= 0B00000100;
         }
       ELSE { // STOP -> MIN (REV)
          STATS ^= 0B00010100;
          }
       }
     }
     ELSE { // MOTOR MOVING BACKWARDS
      IF (( STATS & 0B00001000) == 0B00001000 ) {
        IF (( STATS & 0B00000100) == 0B00000100 ) SOUND_PLAY(2100,100); // MAX REV SPEED ALREADY REACHED
        ELSE {  // MED -> MAX
          STATS ^= 0B00000100;
         }
       }
     ELSE { // MIN -> MEDIUM
        STATS ^= 0B00001100;
        }
      }
}

VOID QUICK_FWD()  {
   IF ((STATS & 0B00000010) == 0B00000010) {
     SOUND_PLAY(1400,100);
     DELAY_MS(50);
     SOUND_PLAY(1450,100);
     DELAY_MS(50);
     SOUND_PLAY(1500,100);
     DELAY_MS(50);
   }
   ELSE {
      //SET BITS FOR MAX FWD
      OLD_STATS |= 0B11111100;
      DRIVE (OLD_STATS);
      DELAY_MS(QTIME);
      }
}
VOID QUICK_REV() {
   //SET BITS FOR MAX REV
   OLD_STATS |= 0B01101100;
   OLD_STATS &= 0B01101111;
   DRIVE (OLD_STATS);
```

```
    delay_ms(qtime);
}
void quick_left() {
    //set bits for max A rev, B fwd
    old_stats |= 0b01111100;
    old_stats &= 0b01111111;
    drive (old_stats);
    delay_ms(qtime);
}
void quick_right() {
    //set bits for max A fwd, B rev
    old_stats |= 0b11101100;
    old_stats &= 0b11101111;
    drive (old_stats);
    delay_ms(qtime);
}


void stop_ab() {
    stats |= 0b10010000;
    stats &= 0b10010011;
}

void beep() {
    Sound_play(1500,100);
    Delay_ms(150);
    Sound_play(1700,100);
    Delay_ms(100);
    }


void Read_Adc()
    {
    ADCON0=0b11001001; // Select Analog2 RC_Mode and ADON
    ADCON0.GO=1; // Start Convert
    while(ADCON0.GO); // Wait Until Convert Complete
    Adc=(ADRESH*4)+(ADRESL/64); // 10 bit Data ==> Adc
}

int detect() {
    Read_adc();
    if ( adc > max_d ) {
      stats |= 0b00000010;
      if  ((~stats & 0b01101100) == 0b01101100);
      else {
          stop_ab();
          drive (stats);
          Delay_ms(150);
          rev_a();
          rev_b();
          drive (stats);
          Delay_ms(500);
          stop_ab();
          drive (stats);
      }
      while (UART1_Tx_Idle() == 0 );
      UART1_Write(stats);
      return 1;
    }
    else {
        stats &= 0b11111101;
        return 0;
    }
}
/*
void toggle_lights() {
```

```
    STATS ^= 0B00000001;
    IF ( (STATS &= 0B00000001) == 0B00000001) { //LIGHTS ON
      LCD_CHR(2, 8,'L');
      //RA0_BIT = 1;
    }
    ELSE {
    LCD_CHR(2,8,'_');
    //RA0_BIT = 0;
    }
}
*/

//=====================================================================//
//================== MAIN PROGRAM ================================//
//=====================================================================//

VOID MAIN()
{
 TRISB3_BIT =0;            //DETECT LED
 //TRISA0_BIT =0;            //DRIVING LIGHTS, NEED TO CHANGE PORT AS IT FUCKS SHIT UP
 LCD_INIT();
 LCD_CMD(_LCD_CLEAR);       // CLEAR DISPLAY
 LCD_CMD(_LCD_CURSOR_OFF);   // TURN CURSOR OFF
 LCD_OUT(1, 1, TEXT);       // PRINT TEXT TO LCD, 2ND ROW, 1ST COLUMN
 SOUND_INIT(&PORTC,0);
 SOUND_PLAY(2500,400);
 UART1_INIT(9600);
 MOTOR_INIT();
 DELAY_MS(100);

 WHILE(1)                        // INFINITE LOOP
 {
 RB3_BIT = DETECT();
 DELAY_MS(20);

 IF (UART1_DATA_READY() == 1) {
    KEY = UART1_READ();
    OLD_STATS = STATS;
    SWITCH(KEY) {
            CASE 'W':
            CASE 'W':
               FWD_A();
               FWD_B();
               BREAK;
            CASE 'S':
            CASE 'S':
               REV_A();
               REV_B();
               BREAK;
            CASE 'A':
            CASE 'A':
               REV_A();
               FWD_B();
               BREAK;
            CASE 'D':
            CASE 'D':
               FWD_A();
               REV_B();
               BREAK;
            CASE 'Q':
            CASE 'Q':
               STOP_AB();
               BREAK;
            CASE 'I':
            CASE 'I':
               QUICK_FWD();
```

```
            BREAK;
        CASE 'K':
        CASE 'K':
            QUICK_REV();
            BREAK;
        CASE 'J':
        CASE 'J':
            QUICK_LEFT();
            BREAK;
        CASE 'L':
        CASE 'L':
            QUICK_RIGHT();
            BREAK;
        CASE 'H':
        CASE 'H':
            BEEP();
            BREAK;
            /*
        CASE 'E':
        CASE 'E':
            TOGGLE_LIGHTS();
            BREAK;
            */
        DEFAULT :
            BREAK;
        }
   DRIVE(STATS);

   WHILE (UART1_TX_IDLE() == 0 );
   UART1_WRITE(STATS);
   }
  }
 }
```

References

PIC16F882/883/884/886/887 Data Sheet - ww1.microchip.com/downloads/en/DeviceDoc/41291F.pdf

PIC16F88X Memory Programming Specification - ww1.microchip.com/downloads/en/DeviceDoc/41287D.pdf

Pica2.0 Robotics Kit User's Manual - www.sparkfun.com/datasheets/Robotics/Robo-PICA2007.pdf

MikroElektronika PIC Compiler - www.mikroe.com/eng/downloads/get/30/mikroc_pic_pro_manual_v100.pdf

Roving Networks Bluetooth module datasheet - www.rovingnetworks.com/documents/RN-41.pdf

Bluetooth SIG Specifications - www.bluetooth.com/Bluetooth/Technology/Building/Specifications

NIST SP 800-121, Guide to Bluetooth Security – csrc.nist.gov/publications/nistpubs/800-121/SP800-121.pdf

Android SDK - developer.android.com/sdk/index.html

Android Development Guide - developer.android.com/guide/basics/what-is-android.html

Android References - developer.android.com/reference/packages.html