EE 474 Lab 5: Final Project

University of Washington
by Brad Marquez, Joseph Rothlin, Aigerim Shintemirova
EE 474

June 8, 2016

1. Introduction

The goal of this project was to utilize the tools and skills we learned in the previous labs, e.g. GPIO pins functionality. motors and sensors control, Bluetooth. We were able to achieve control of the motors through a one-way bluetooth transmission from a Windows Surface to the BlueSMiRF. This program is ran through on bootup through a service that we set up.

We also designed a system that scans the space around it with 4 sensors mounted on a servo. We are then able to send this data back to a terminal emulator, where the data can be picked up and read to map out the Robo-Tank's surroundings.

Basic Functionality:

Drive Program

- Q- Forward
- A- Backward
- W- Right
- S- Left

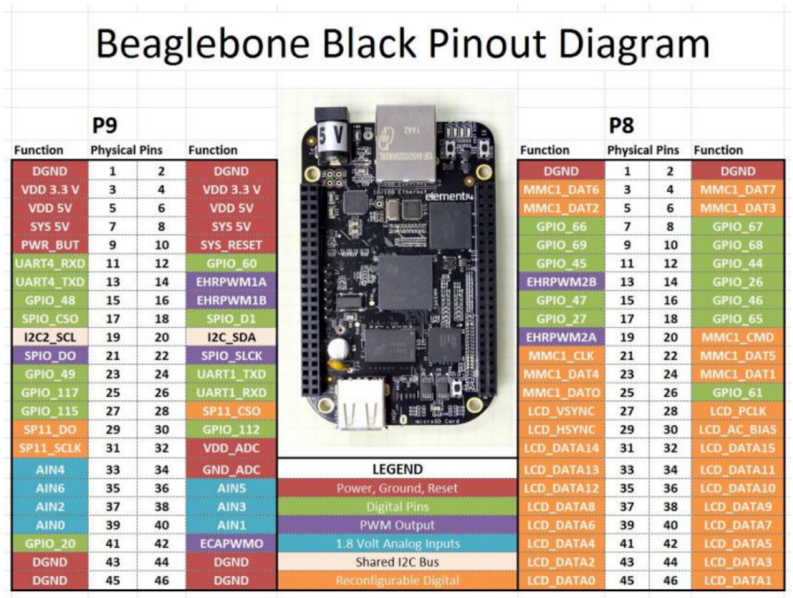
Sensing Program

- W- Forward
- S- Backward
- A- Left
- D- Right
- E- Scan

2. Hardware Description:

2.1. Connecting to the Board

Establishing the proper connection to the board is essential to accomplish the goal of this lab. It is done using SSH networking protocol commands. The command used was *ssh root@192.168.7.2*. Once, the connection is established, programs could be loaded onto the board.



Beaglebone Black Pinout Diagram

P9

Function	Physical Pins		Function
DGND	1	2	DGND
VDD 3.3 V	3	4	VDD 3.3 V
VDD 5V	5	6	VDD 5V
SYS 5V	7	8	SYS 5V
PWR_BUT	9	10	SYS_RESET
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPIO_CSO	17	18	SPIO_D1
I2C2_SCL	19	20	I2C_SDA
SPIO_DO	21	22	SPIO_SLCK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SP11_CSO
SP11_DO	29	30	GPIO_112
SP11_SCLK	31	32	VDD_ADC
AIN4	33	34	GND_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPWMO
DGND	43	44	DGND
DGND	45	46	DGND

P8

Function	Physical Pins		Function
DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DATO	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

LEGEND
Power, Ground, Reset
Digital Pins
PWM Output
1.8 Volt Analog Inputs
Shared I2C Bus
Reconfigurable Digital

Figure 1. Pinout for the Beaglebone black board. Notice that the export reference to the GPIO pins is denoted by the number within the Function name. For example, the number used to denote Physical Pin 15, P9 is 48 (GPIO_48).

2.2 Connecting the hardware

Table 1: The Pin Description for the Beaglebone Pins

Pin No.	Symbol	External Connection	Function Description	Attached Beaglebone Pin
1	VSS	Power Supply	Ground	DGND
2	VDD	Power Supply	Supplies Voltage for logical operations (+5 V)	SYS 5V
3	V0	Adj Power Supply	Power supply for screen contrast	N/A
4	SER_DATA	Shift Register	Serial Data Input for H-bridge	GPIO_PIN_112
5	SR_CLOCK	Shift Register	Clock signal for the Shift Register to the H-bridge	GPIO_PIN_49
6	LATCH_	Shift Register	Clock signal for the Shift Register to the H-bridge	GPIO_PIN_115

7	FRONT	Front Sensor	Pin for the front sensor	AIN6
8	BACK	Back Sensor	Pin for the back sensor	AIN0
9	LEFT	Left Sensor	Pin for the left sensor	AIN2
10	RIGHT	Right Sensor	Pin for the right sensor	AIN4
11	PWM	H-Bridge	Supplies PWM to H-bridge for motors	PWM P9_14
12	PWM-Servo	Servo	Supplies PWM to the Servo	PWM P8_13
13	RX_	BlueSMiRF	TX of Beaglebone	UART1_RX
14	TX_	BlueSMiRF	RX of Beaglebone	UART1_TX

2.3 Transferring cross-compiled files to the board

Files are transferred and cross-compiled directly to the board using the given Makefile. We were able to set the Makefile so that it compiles the hBridge and its included files. The compiler used in the makefile is the arm-linux-gnueabihf- compiler. Note that there are separate programs for driving and sensing. The sensing program has the ability to drive but transmission from the Surface is glitchy and slow.

Table 2. The targets of the Driving Program Makefile and their individual effects

Target	Function Description
all	Creates the drivers for the H-bridge Driver
hBridgeDrive	Compiles H-bridge Driver
hBridgeDrive.o	Creates object file for the Drive control
transfer	Transfer the compiled files to the Beaglebone
clean	Removes the transferred file from the Beaglebone

Table 3. The targets of the Sensing Program Makefile and their individual effects

Target	Function Description
all	Creates the drivers for the H-bridge

hBridge	Compiles the master hBridge program
hBridgeDriver.o	Creates object file from the hBridgeDriver.c
ScannerControl.o	Creates object file for scanning the surroundings
SensorControl.o	Creates object file for each individual sensor
ServoControl.o	Creates object file for the servo control
transfer	Transfer the compiled files to the Beaglebone
clean	Removes the transferred file from the Beaglebone

2.4. Overall System Structure

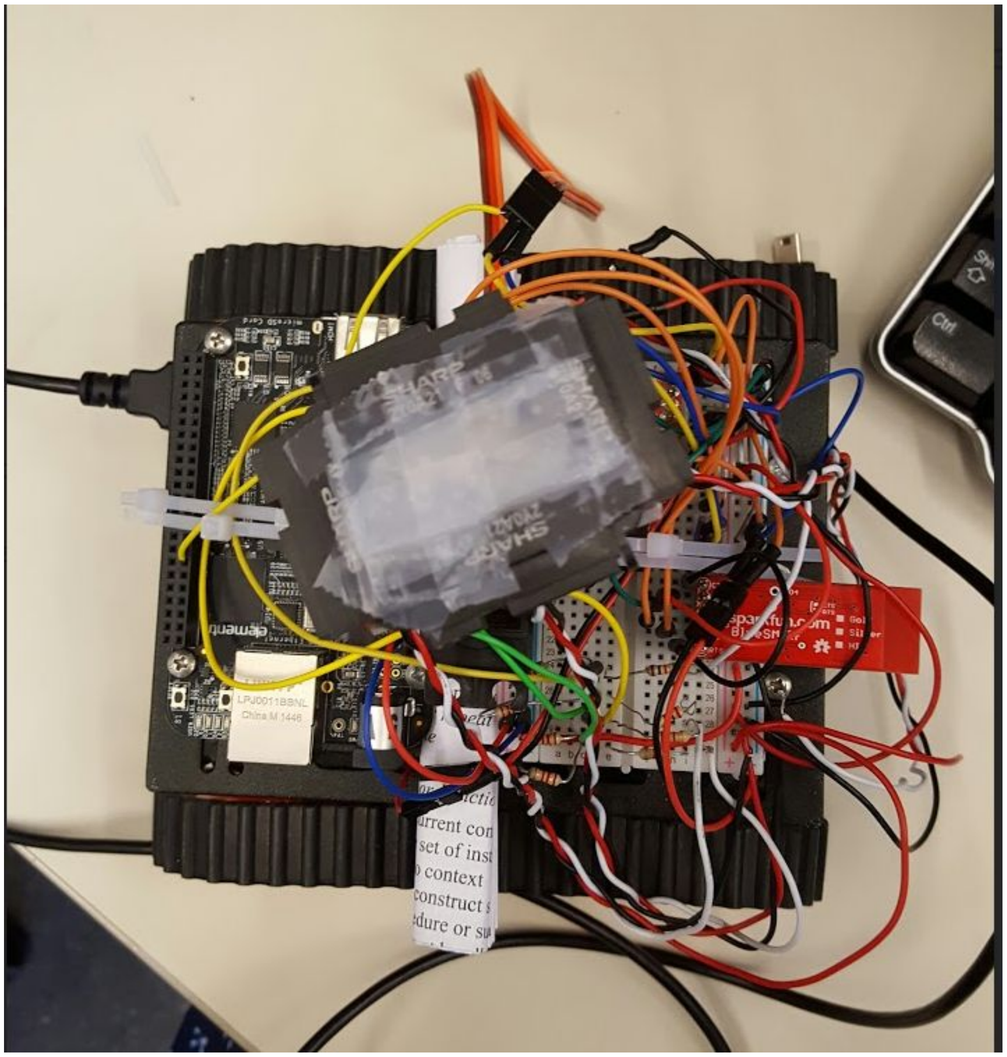


Figure 2. Snapshot of the connections between the pins on the BeagleBone, sensors, and H-bridge driver, and Bluetooth.

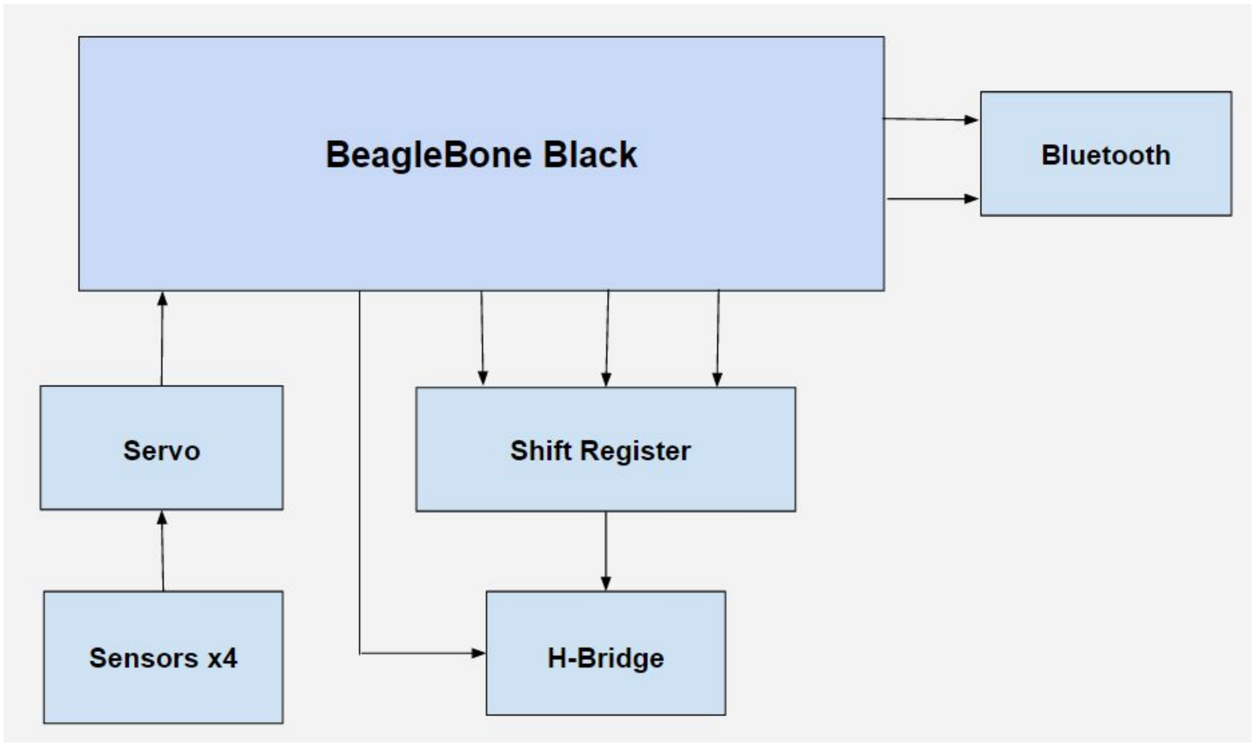


Figure 3. Block Diagram of the connections between the pins on the BeagleBone, sensors, H-bridge driver, and Bluetooth. Note that the sensors are physically attached to the servo, and send read signals straight to the Beaglebone.

2.5. H-bridge

H-bridge (TB6612FNG) is used as a motor driver. We wrote a program in user-space that acts as a Master and moves the RoboTank based on inputted data over Bluetooth. The program written for the H-bridge is based on the serial-in/parallel-out shift register that allows to reduce the number of GPIO pins used. The parallel register on the shift register correspond to the H-bridge inputs as follows: (QA: Standby, QB: AIN1, QC: AIN2, QD: BIN1, QE, BIN2, QF: N/A, QG: N/A, QE: N/A). Note that A corresponds to the left motor and B corresponds to the right motor.

Table 4: The Pin Descriptions for the H-bridge

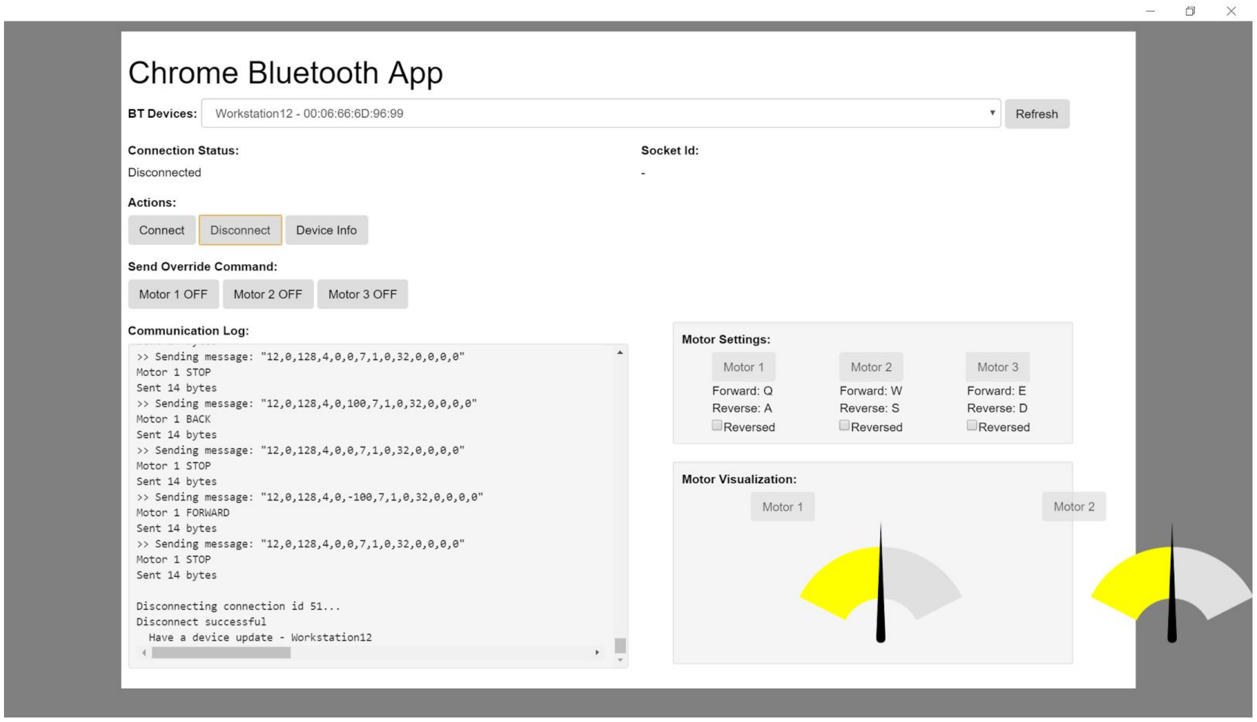
Pin No.	**Symbol**	**External Connection**	**Function Description**	**Notes**
1	PWMA	PWM P9_16	PWM for the left motor	Is tied to PWMB
2	AIN2	Shift	Logic 2 for the	From Shift Register

		Register: QC	left motor	
3	AIN1	Shift Register: QB	Logic 1 for the left motor	From Shift Register
4	STBY	Shift Register: QA	Standby	From Shift Register
5	BIN1	Shift Register: QD	Logic 1 for the right motor	From Shift Register
6	BIN2	Shift Register: QE	Logic 2 for the right motor	From Shift Register
7	PWMB	PWM P9_16	PWM for the right motor	Is tied to PWMA
8	GND	Ground	Ground	Ground #1/3
9	VM	~6 V from battery pack	Voltage for the motors	Motors can take up to ~15 V
10	VCC	5 V from Beaglebone	Voltage for logic	For H-bridge logic
11	GND	Ground	Ground	Ground #2/3
12	AO1	Left Motor - Red	Controls left motor	Connected to Red of left motor
13	AO2	Left Motor - Black	Controls left motor	Connected to Black of left motor
14	BO2	Right Motor - Black	Controls right motor	Connected to Black of right motor
15	BO1	Right Motor - Red	Controls right motor	Connected to Red of right motor
16	GND	Ground	Ground	Ground #3/3

Table 3: Inputs and corresponding outputs for the H-bridge

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

In order to input data, we used a program called NXT Bluetooth Controller from the Chrome Web Store which was able to communicate and send data packets on pressed keys. We chose to command the tank with: Q- Forward, A- Backward, W- Right, S- Left, and Default- Stop.


Chrome Bluetooth App

BT Devices: Workstation12 - 00:06:66:6D:96:99

Connection Status: Disconnected

Socket Id: -

Actions:

Send Override Command:

Communication Log:

```
>> Sending message: "12,0,128,4,0,0,7,1,0,32,0,0,0,0"
Motor 1 STOP
Sent 14 bytes
>> Sending message: "12,0,128,4,0,100,7,1,0,32,0,0,0,0"
Motor 1 BACK
Sent 14 bytes
>> Sending message: "12,0,128,4,0,0,7,1,0,32,0,0,0,0"
Motor 1 STOP
Sent 14 bytes
>> Sending message: "12,0,128,4,0,-100,7,1,0,32,0,0,0,0"
Motor 1 FORWARD
Sent 14 bytes
>> Sending message: "12,0,128,4,0,0,7,1,0,32,0,0,0,0"
Motor 1 STOP
Sent 14 bytes

Disconnecting connection id 51...
Disconnect successful
  Have a device update - Workstation12
```

Motor Settings:

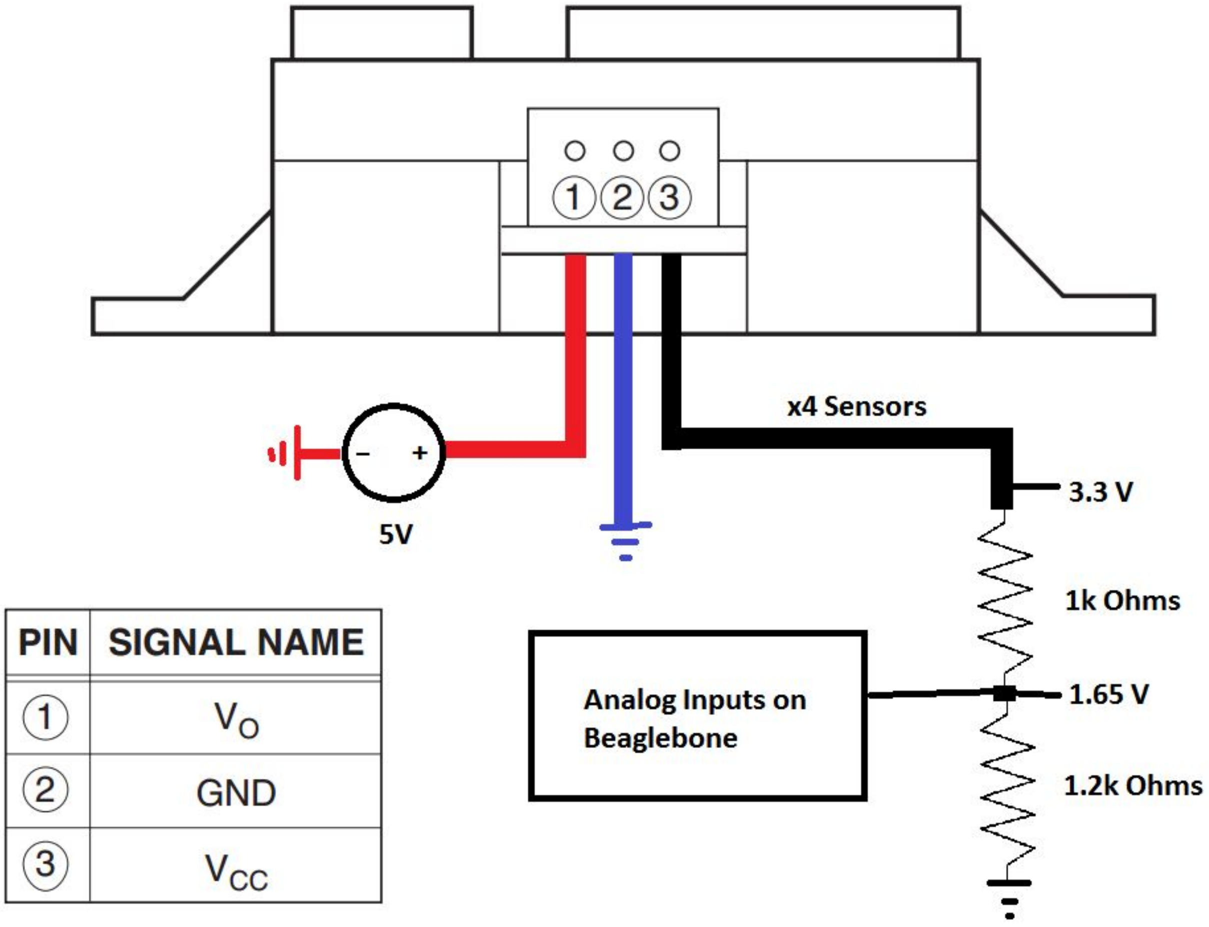
Motor 1	Motor 2	Motor 3
Forward: Q	Forward: W	Forward: E
Reverse: A	Reverse: S	Reverse: D
☐ Reversed	☐ Reversed	☐ Reversed

Motor Visualization:

Figure 4. GUI of the Bluetooth Controller

2.6. Sensors

Four sensors were installed on top of a servo that is controlled by the RoboTank when it receives an input of 'e' over Bluetooth. On an 'e' input, our program commands the servo to spin 90 degrees in 10 degree increments, taking 300 samples at each increment. The average is then calculated from the sensor readings at each degree, for each motor, and those set of values are returned as a block of data. Note that the sensors output maximum of 3.3 V and the reading ADC pins used read a maximum of 1.8 V. If the ADC pins are fed more than 1.8 V, the readings are very unstable and inaccurate. To avoid this problem, the 3.3 V output of the sensor is voltage divided down to ~1.8 V using R1 = 1k Ohms and R2 = 1.2k Ohms. A preset threshold value sets the distance at which the sensors can recognize an obstacle and communicate the information to the H-bridge. This program writes what side has a sensor that is detecting an obstacle to the TX of the bluetooth, which then can be picked up by a terminal emulator. From the terminal emulator, we were able to map out the surroundings around the sensor in a text based way.



PIN	SIGNAL NAME
1	V_O
2	GND
3	V_{CC}

Figure 4. Block Diagram of the connections between the sensors and the Beaglebone

3. Software Description

3.1 Drive

3.1.1 hBridgeDrive.c

Function: Sets up motors, takes in inputs to drive motors.
This program establishes the connection with the Bluetooth and allows to send/receive serial data. We used a NXT Bluetooth Controller from the Chrome Web Store, which is able to send data bytes to the bluetooth. We then use these inputs to drive the RoboTank forward, left, right, and backwards.

3.1.2 Arduino-serial-lib

Function: Library used to set up and use the serial port.
This library allows us to initialize the serial port, write to the serial port, read from the serial port, close the serial port, and flush the serial port.

3.2 Sensing

3.1.1 hBridgeDriver.c

Function: Sets up motors, takes in inputs to drive motors or scan.
This program first sets up the pointers and configures the attached bluetooth as needed. Then the program starts a timer interrupt which tells the system to read in data from the serial port. From a Surface, which is paired with the BlueSMiRF, we are able to transmit commands through a terminal emulator (PuTTy). We then used these command to drive the tank forward, left, right, and backwards and start scanning.

3.1.2 Arduino-serial-lib.c

Function: Library used to set up and use the serial port.
This library allows us to initialize the serial port, write to the serial port, read from the serial port, close the serial port, and flush the serial port.

3.1.3 SensorControl.c

Function: Takes in values from attached sensor.
This program contains functions for a high level program to easily control the four sensors attached to the servo providing concise encapsulation. A function is provided to initialize the files associated with the sensors, read values from specific sensors, and close all the files associated with the sensors.

3.1.4 ServoControl.c

Function: Controls the attached servo motor.

This program contains functions for a high level program to easily control the servo motor within a concise encapsulation. A function is provided to initialize the PWM pin files associated with the servo, set the servo to a specific angle, and close the PWM pin file.

3.1.5 ScannerControl.c

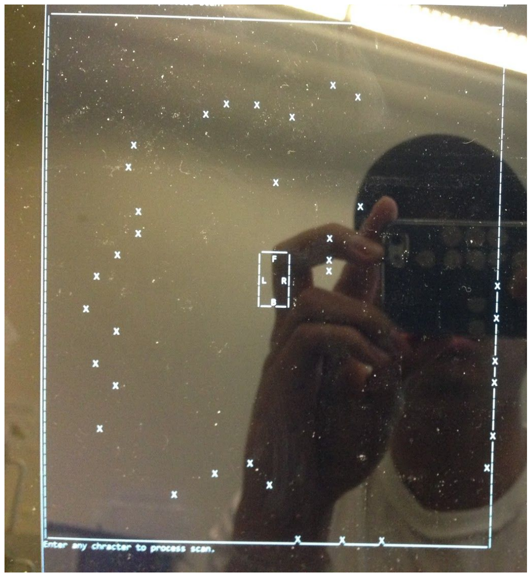
Function: Controls the servo and sensor to spin and take data.

This program contains functions for a higher level program to easily control the servo motor along with the distance sensors (this entire collection is referred to as the scanner) in a concise encapsulated manner. This program allows for files initialization, files closure, the ability to initiated a scan with various sample rate settings, and functions which will print a set of scanner readings in both as both a set of values and a visual text display (as seen in figure 5).

3.1.6 mapBuild.c (on Surface)

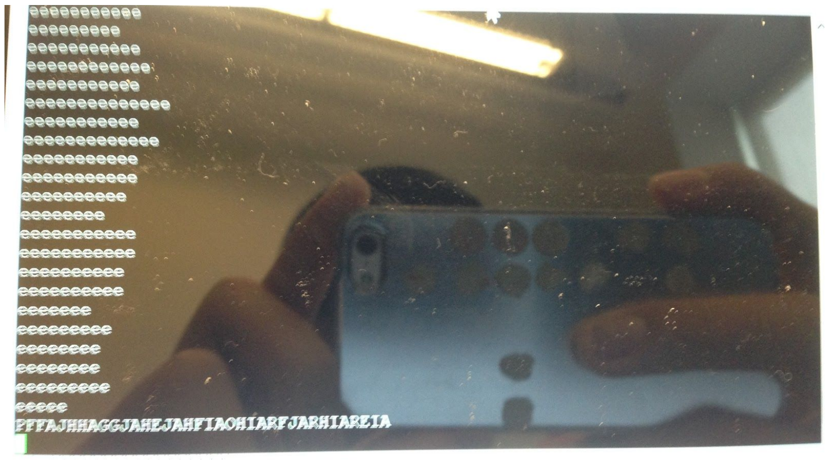
Function: Reads data from a txt file logged to by the PuTTy Terminal Emulator and outputs a map of the surrounding of the Tank on the terminal.

When the RoboTank is told to scan, after it is done, it sends back data through the bluetooth. However, this data are non-displayable ASCII characters, so we made the transmitted data have a range of A-Z so that it can be displayed on the terminal emulator and printed to a txt file. This program can then parse the txt file, take in the eligible data, and print out a map based on the data to the terminal.



```
Enter any chracter to process scan.
```

Figure 5. Screenshot of a map made with data sent from the Bluetooth. Excuse the reflection.



```
eeeeeeeeeee
eeeeeeeee
eeeeeeeeeee
eeeeeeeeeeee
eeeeeeeeeee
eeeeeeeeeeeeee
eeeeeeeeeee
eeeeeeeeeeeee
eeeeeeeeeee
eeeeeeeeeee
eeeeeeeeee
eeeeeeee
eeeeeeeeeee
eeeeeeeeeee
eeeeeeeeee
eeeeeeeeee
eeeeeee
eeeeeeeee
eeeeeeee
eeeeeeee
eeeeeeeee
eeeee
PFFAJHHAGGJAHEJAHFIAOHIARFJARHIAREIA
```

Figure 6. Screenshot of the PuTTy terminal used to send/receive data. Excuse the reflection.

4. Discussion of Errors/Problems

4.1 Motor Errors

The motors did not always work properly and required a certain amount of mechanical push. We tried adding extra battery alongside the already connected battery pack, but the motors still operated relatively slow. As a result, turn left and turn right commands are not as powerful as forward and backward. We believe that the error is caused by the connections within the RoboTank.

4.2 Sensing Errors

Due to the quality of the sensors there is found the be a large margin of error in the accuracy of the readings on the sensors. Even while taking a large number of sensor readings and averaging them out, we still find a large amount of error within the accuracy of the reading. Using the sensors in different environments also had an effect on the accuracy. For example, using the sensors to measure long distances in a well lit area results in poor accuracy while measuring short distances provides more consistent values. The quality of the sensors was the most significant reason for error when measuring distances. When creating a display of the distance readings there are also slight rounding errors that come up in the software due to rounding in calculations. Floating point values are used in most of the calculations to prevent this as much as possible. The servo motor also introduces some error in the scan readings because it's ability to move to a specified angle is not without slight error.

4.3 Bluetooth Errors

When we tried to send and receive data at the same time through the PuTTy terminal, data sometimes is not sent from the Surface to the BlueSMiRF. We suspect this has to do with a lack of flow control, as the sending and receiving both compete with each other to go through the one COM port. Solutions that we thought might work are using a mutex, semaphore, or implementing hardware/software flow control. Unfortunately, we were not able to implement these.

5. Conclusion

The main purpose of this lab was to build a design that encompasses the projects developed in labs throughout the quarter. As a result, we built a system where 4 sensors are connected to a servo that spins 90 degrees, senses the space around it, and commands the RoboTank (H-Bridge) to move accordingly. Another way to operate the RoboTank is to connect through Bluetooth and send directions to the motors through a remote application.