

Self-Navigating Robots Use BLE

Signals and Servos

Navigating indoors is a difficult but interesting problem. Learn how these two Cornell students use Bluetooth technology to enable wheeled, mobile robots to navigate toward a stationary base station.

By
Jane Du and Jacob Glueck

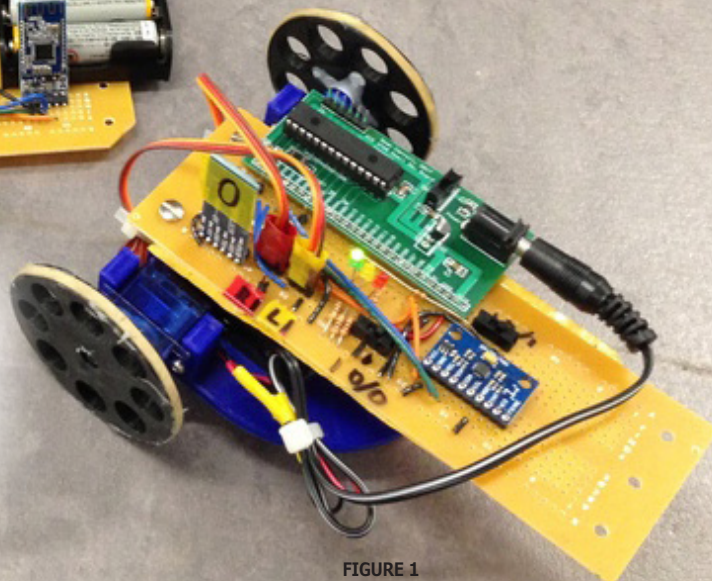


FIGURE 1

Full system with two robots and the base station

In our project, we investigate using the Received Signal Strength Indicator (RSSI) of Bluetooth Low Energy (BLE) 4.0 chips as a means for wheeled mobile robots to find their way to a stationary base station. The robots detect their proximity to the station based on the strength of the signal, and move toward what they believe to be the signal source.

Each robot was controlled by a Microchip Technology PIC32MX250 microcontroller and used a 3-axis magnetometer as compass to reliably turn and two servos to drive. Each unit was powered with three AA batteries. Finally, the chassis and wheels of each car were 3D printed. **Figure 1** shows the entire system.

In the process of designing and building the hunting robots and the beacon, we encountered an interesting challenge: counterfeit Bluetooth modules with usable hardware but incorrect firmware, obtained from unofficial vendors. We explain the process we used to find, flash and verify the correct firmware to use it for our project.

SYSTEM DESIGN

The system consists of two robots and a base station. We designed the robots to be small, light and maneuverable. Each robot has two wheels and a third, round plastic leg, serving as a

caster wheel. We 3D printed each robot's frame in Acrylonitrile Butadiene Styrene (ABS), which made them easy to assemble. After printing the parts, we mounted a three-AA battery pack to the frame to provide power and screwed in a Perfboard on standoffs. **Figure 2** shows the parts in each robot's chassis.

The electronics for each robot consist of four main components: A Microchip PIC32MC250F128B MCU, a TDK Invensense MPU-9250 IMU with a 3-axis gyroscope, accelerometer and magnetometer, an HM-10 BLE module with a Texas Instruments CC2541 chip, and two FS90R micro continuous rotation servos. The BLE module connects to the PIC through a UART connection, and the IMU connects to the PIC through an I²C bus. The base station is identical to the robots, minus the IMU and servos. The HM-10 Bluetooth module is mounted perpendicular to the breadboard, sticking up in the air, in order to, we hoped, improve the performance of the radio. **Figure 3** is a full schematic for one of the robots. Not shown is the schematic of the PIC32MC250F128B Dev Board created by Sean Carroll. A link to that is available on the *Circuit Cellar* article materials webpage.

While trying to use the Bluetooth modules, which we bought on eBay, we discovered that they didn't conform with the data sheet [2].

After doing some research, we concluded they were counterfeit! Although the boards still had Texas Instruments CC2541 chips, the firmware they were running was not genuine firmware from Jnhuamao Technology. Luckily, the only difference between the fake boards and the genuine boards was that the genuine boards had an external crystal. The genuine firmware checks for the presence of the crystal and works even without it. Because of that, we were able to salvage the counterfeit chips by reprogramming them with genuine firmware, according to an Arduino Forum post [3]. Essentially, we connected the chips to an Arduino Teensy (which is 3.3 V, and won't damage the 3.3-V CC2541), as indicated the table and schematic (Figure 4).

Then, we uploaded the CCLoader.ino [4] sketch to the Arduino. This sketch bit bangs the programming signals for the CC2541. Finally, we ran CCLoader.exe [5] in a Windows virtual machine, due to the dubious origin of the software:

```
CCLoader.exe <COM Port>
<Firmware.bin> 0
```

The firmware file came from the Arduino Forum post. [6]

Next, we updated the chip's firmware from 540, the version we had just flashed onto it, to the most recently released version at the time, 603. [7] We connected the HM-10 module to a computer using a 3.3-V FTDI-to-USB adapter. Then, using PuTTY—an SSH client—we established a serial connection (9,600 baud, 8N1) and sent the chip AT. If it is connected properly, it will respond with OK.

The rest of the procedure is as follows. Send the chip an AT+SBLUP command to put it in firmware update mode. It will respond with OK+SBLUP. Terminate the PuTTY session. Run the HMSoft.exe program distributed in the firmware update download. Again, this was done in a Windows virtual machine,

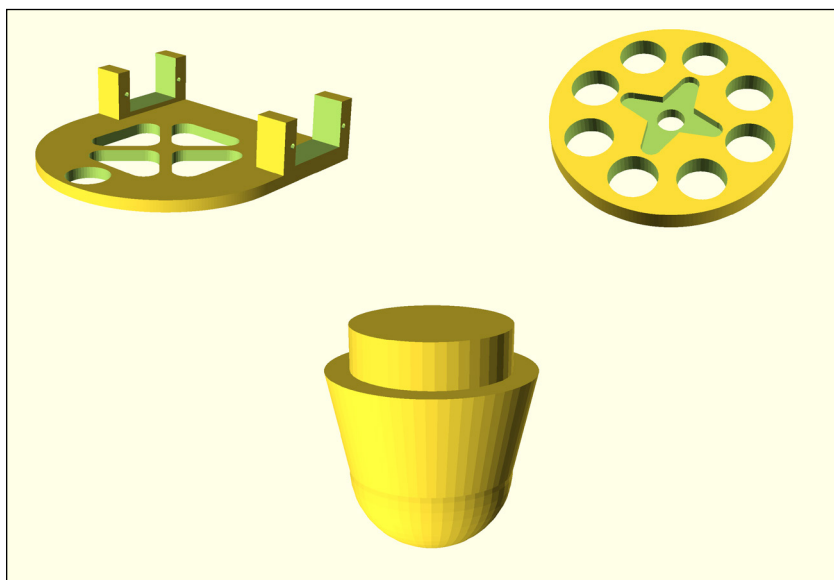


FIGURE 2 Components of robot chassis

because of the suspect software. Select the proper port and firmware file using the software and hit "Load Image." The software should handle the rest! To make sure it works, establish a serial connection again using PUTTY. Send the AT+VERS? command to query the chip for version information.

At this point, we had the Bluetooth chips in a working state. To have them measure signal strength, we sent them the sequence of commands shown in Table 1. One interesting thing we noted about the chips was that commands do not have to end with newlines or carriage returns. However, if sent, the chips will ignore them.

Once we had the Bluetooth signal strength working, we configured the IMU, a QFN MPU-9250 module [8] [9]. It has two dies: one is the AK8963 3-axis magnetometer [10], and the other contains the 3-axis gyroscope and 3-axis accelerometer, which we did not use.

The microcontroller communicates with the IMU via I²C, and the compass is connected to the rest of the MPU module

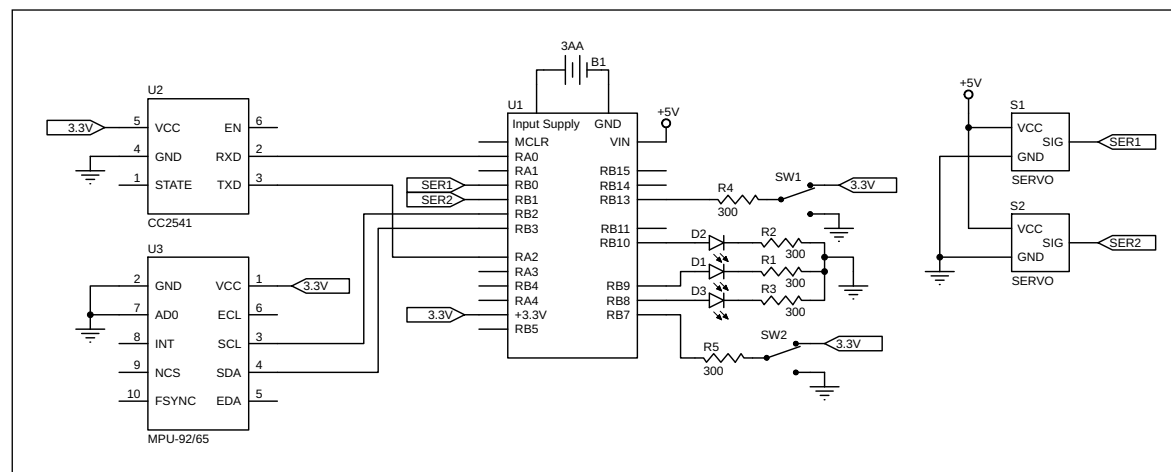


FIGURE 3 Robot schematics

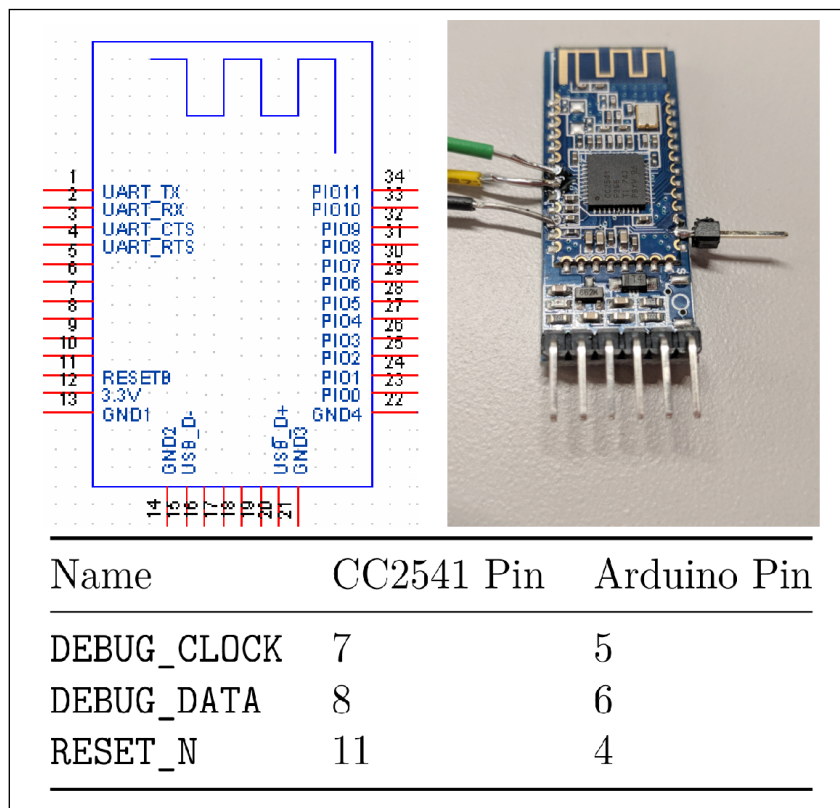


FIGURE 4

This shows the pin layout, physical setup and pin assignment required to program HM-10 module.

by an auxiliary I²C bus. For communication between the microcontroller and the AK8963's main die, we based our work on basic I²C functions from another Cornell ECE 4760 class project, the Self-Balancing Robot [11]. Their `i2c_helper.h` was very helpful. For communication with the actual magnetometer, we needed to configure the IMU to set the compass as a slave on the I²C bus. For this, it was necessary to enable pass-through mode on the IMU during its configuration.

The AK8963 has several modes of operation, and the chip must be set to power-down mode before switching to other modes. We read compass values with single measurement mode [8]. A single compass read involves setting the compass to single-measurement mode in 14-bit resolution, reading the six data registers (X low, X high, Y low, Y high, Z low, Z high), reading the Status 2 register to check for magnetic sensor overflow, and finally waiting to ensure that the IMU is not read too frequently. Without reading the Status 2 register, the read is not considered complete and further reads will fail. If the IMU is read too often, it will not have enough time to take measurements.

To calibrate the compass, the robots spun in place when powered on. They recorded the maximum and minimum values for each axis and used that data to scale and center the magnetometer readings. Once we had the robots working, we turned to the algorithm. Because they don't have much data—only

Command	Description
AT+RESET	Resets the chip to ensure it is in a clean state before receiving other commands.
AT+IBEA1	Sets the iBeacon functionality of the chip (1 on, 0 off). This allows the chip to be found with an RSSI scan.
AT+ROLE{0 1}	Sets the role to either peripheral (0) or central (1). The base station is set to peripheral, and the robots are set to central. Peripheral means the device will respond in inquiries from a central device. This allows it to be discovered during an RSSI scan.
AT+IMME{0 1}	Sets the work state of the device to either actively listening for Bluetooth signals (0), or only acting when it receives a serial command (1). Once again, the base station is set to 0: it needs to listen for signals and respond. The robots are set to 1, as the chips need to initiate scan requests when they receive the command over serial.
AT+NAME{str}	Sets the name of the chip (which is visible when scanning) to the string <code>str</code> (For example, <code>AT+NAMEPIRATE</code> names the chip <code>PIRATE</code>). We gave all the chips unique names to make debugging easier.
AT+SHOW3	Configures the device to advertise both its name and RSSI when scanning.
AT+ADDR?	Queries the device for its hardware address. We recorded the hardware device of each chip, as when doing RSSI scans, the results are reported by hardware address.
AT+DISI?	Performed only on the robots, causing a discovery scan. The result of the scan is a series of lines of the form: OK+DISC:00000000:00000000000000000000000000000000: 000000000:6832A3801EBE:-080 The next to last token, <code>6832A3801EBE</code> , is the hardware address of the discovered device, and the last token, <code>-080</code> , is the measured RSSI. The chip will transmit a line for each device it finds ("line" is a misnomer as it does not separate them with any characters), followed by <code>OK+DISCE</code> .

TABLE 1

HM-10 Bluetooth module signal strength measurement commands

the last few signal strength measurements—we decided to use a simple gradient descent algorithm (**Figure 5**).

RESULTS

When starting from 1 m away, the robots successfully made it to the base station an average of 80% of the time, in an average of 126 seconds. One robot performed better than the other, reaching the base station 100% of the time, compared to 60%, and it arrived on average 8 cm closer. Although we believe the worse-performing robot could be tuned, neither robot was as fast as we would have liked, and each made several wrong turns. One of the main reasons for this was the noise in RSSI measurements.

We expected that RSSI would vary with distance according to:

$$RSSI = A - 10n\log(d)$$

where A and n are RF propagation parameters in dBm, d is distance in meters, and RSSI is the measured RSSI in dBm [12]. We experimented with RSSI measurements to determine how well they worked by measuring the RSSI while moving two Bluetooth modules apart. One remained stationary on the floor, and the other was moved away from it 1 foot at a time. At each point, we took three RSSI measurements and averaged them. The results are shown in **Figure 6**, and the original data are in **Table 2**.

We fit the data using $A=-49$ and $n=2.9$, resulting in the blue curve above (Expected RSSI). While the general shape of the curves match, there is significant noise in the averaged RSSI data. Furthermore, when we tried to reproduce the measurements, we could not do so precisely; it seemed to depend on the position of our feet! Even though it was noisy, in general, RSSI increased as the robots got closer together, which is all our system required.

Despite the noise in the RSSI readings, the robots performed surprisingly well. We tested the robots by starting them both 1 m from the base station, one on the north side, and one on the south side. After they started, we measured the time it took them to arrive at the base station and flash their LEDs. We set a timeout of 5 minutes, at which point we would measure the distance between the robot and the base station. If a robot ran into a wall and couldn't recover, we stopped it. We also measured the distance when the robot arrived. All distances were measured between the Bluetooth modules. The results are displayed in **Table 3** with averages in **Table 4**.

We observed some interesting patterns in the data. An obvious one is that Robot 1

always reached the base station, whereas Robot 0 had only a 60% success rate. In addition, when Robot 1 arrived, it was on average 8 cm closer than Robot 0 and converged about 30 seconds faster. We surmise that the antennae on each robot had different sensitivities, as both used the same RSSI threshold for stopping once they reached the goal. Another interesting result was that when Robot 0 failed, it was on average 3 m away—far off in the land of

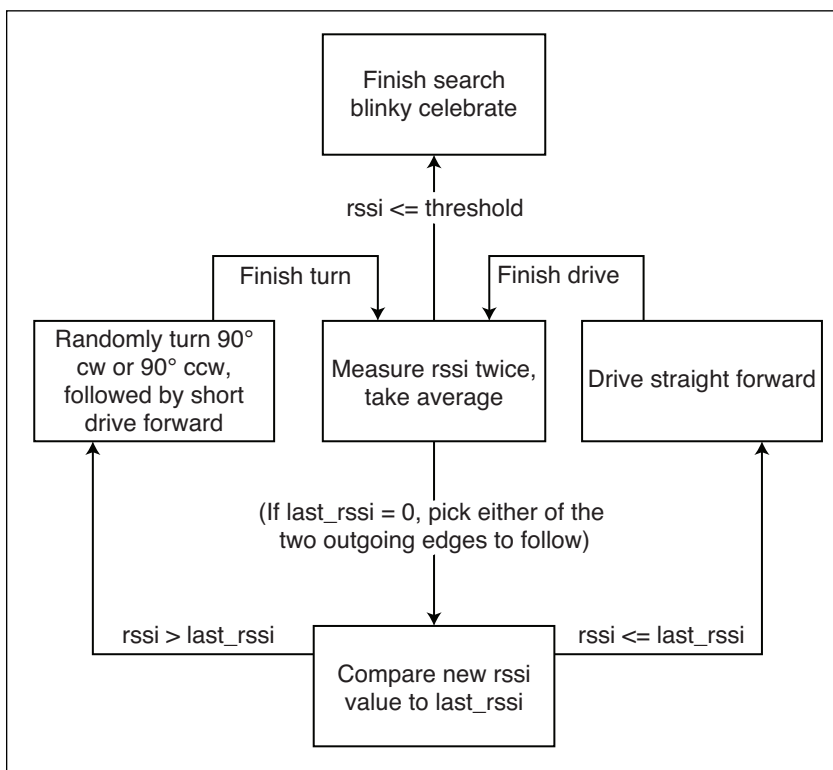


FIGURE 5 The gradient descent algorithm. The initial state is measure RSSI twice and take the average.

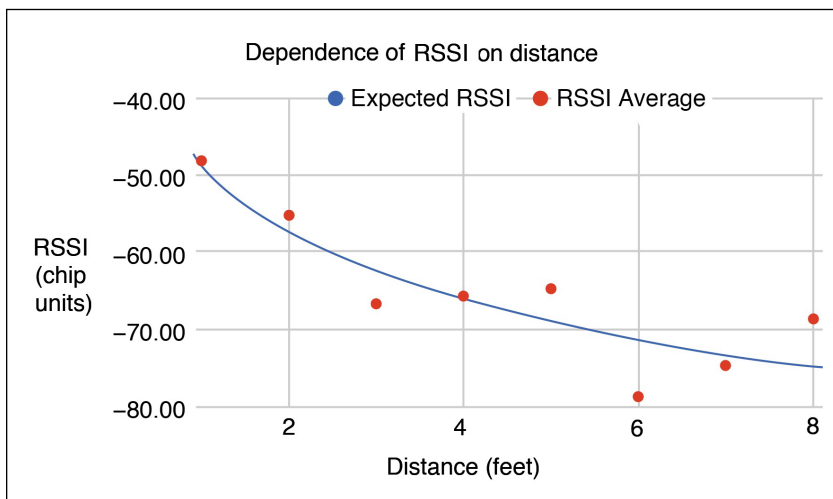


FIGURE 6 Plot showing Bluetooth RSSI as a function of distance

Distance (feet)	RSSI A	RSSI B	RSSI C	RSSI Average	Expected RSSI
1	-48	-48	-48	-48.00	-48.00
2	-56	-58	-52	-55.33	-57.03
3	-66	-66	-68	-66.67	-62.31
4	-64	-68	-65	-65.67	-66.06
5	-60	-70	-64	-64.67	-68.97
6	-80	-80	-76	-78.67	-71.34
7	-80	-79	-65	-74.67	-73.35
8	-68	-66	-72	-68.67	-75.09

TABLE 2
RSSI and distance data

shallow gradients. The only way to recover from that was dumb luck, as the signal would have been dominated by noise that far away.


CONCLUSIONS

This project was an interesting exploration into short-range distance determination using Bluetooth, a generally unconventional approach. We knew that Bluetooth RSSI would be noisy, mostly due to multipath interference and the presence of multiple transmitters in testing environments. The robots worked reliably when they stayed within roughly 1 m of the beacon. After this, they entered the

land of shallow gradients. The signal strength from the beacon—already noisy—would not change very much, and often only from noise. They normally could never recover from this.

There are two opportunities for improvements or further development:

Communication between the two robots: Although Bluetooth might not offer good distance measurement via RSSI, it can be used for reliable communication between modules. It would be straightforward for one hunting robot to inform the other whether or not it believes it is approaching the beacon. In the simplest case—a robot that is approaching, or already has arrived—the beacon can provide a second point of reference for a currently hunting robot.

More complete usage of IMU: Additional usage of the accelerometer and gyroscope, coupled with feedback from the servos, would allow the robots to maintain a dead-reckoning position estimate. This, paired with inter-swarm communication, would make for a more sophisticated and likely more efficient system. Of course, this does not resolve the shallow gradients problem, but it would allow the approach to the beacon to be much faster. 

AUTHORS' NOTE: We'd like to thank Justin Cray for his contributions to the design and construction of the Bluetooth robots.

TABLE 3
Convergence performance of the robots

(I)2-4 (I)5-7	Time (s)	Distance (cm)	Start	Time (s)	Distance (cm)	Start
1	213	53	South	94	22	North
2	Timeout	420	South	153	36	North
3	308	19	South	105	36	North
4	Collision	340	South	80	20	North
5	Collision	240	South	92	30	North
6	179	46	North	117	20	South
7	56	45	North	162	24	South
8	59	27	North	197	26	South
9	59	19	North	79	30	South
10	Collision	231	North	67	23	South

For detailed article references and additional resources go to: www.circuitcellar.com/article-materials

References [1] through [12] as marked in the article can be found there.

RESOURCES

Microchip Technology | www.microchip.com

Pololu | www.pololu.com

TDK Invensense | www.invensense.com

Texas Instruments | www.ti.com

	Robot 0	Robot 1
Convergence time (seconds)	146	115
Convergence rate	60%	100%
Final distance (cm)	35	27
Failure distance (cm)	308	-

TABLE 4
Average convergence behavior