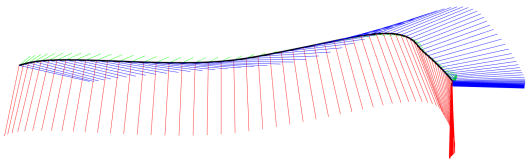


Navigation Solutions LLC

RT-BLE-001

Release 4

Wireless Foot Tracking System

10/15/2023

Ann Arbor, MI

www.imuwear.com

Tel: +1 734 223 5904

Email: lojeda@imuwear.com

Navigation Solutions LLC

RT-BLE-001

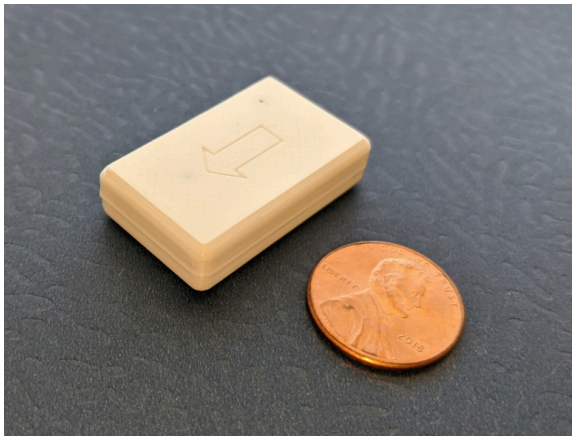
Product Overview

The RT-BLE-001 is a completely self-sustaining wireless foot-tracking solution. Consisting of inertial sensors, an onboard computer, a Bluetooth Low Energy (BLE) interface, and a rechargeable battery, this RT-BLE-001 device is highly efficient. It calculates the user's foot position and orientation within a three-dimensional space, providing real-time updates through the BLE interface. Moreover, a device, such as a smartphone, PC, or Raspberry Pi, equipped with BLE can capture the data stream using our open communication protocol. The system requires no external references and is capable of operating in almost any environment.

Product Components

The RT-BLE-001 system is delivered with the following components:

- Foot sensor
- Micro-USB charging cable
- Shoe mounting clip mechanism
- *Binary program for the Android platform
- **Sample Python programs:



* Available for download. This requires a user-provided Android device running on OS version 10.0 or higher.
**Also available for download.

Device Charging

The unit typically achieves a full charge after approximately one and a half hours when plugged in using the USB cable. After this period, it is advisable to unplug the IMU to prevent overcharging, which over time can diminish the battery's lifespan. When the unit is not in use, it is strongly recommended to charge the battery at least once a month; a failure to do so could result in the battery reaching dangerously low levels that could cause permanent damage.

Device Mounting

There are several options for securing the device to the shoe, such as using a sensor clip (provided), a shoe pouch, or tape. The device should be fastened securely against the shoe so that it doesn't wiggle or bounce during walking or running. Irrespective of the securing method employed for the device on the shoe/foot, in the ultimate setup, the LED and device arrow are situated atop, pointing forward.

The RT-BLE-001 adheres to the axis convention used in aeronautics, with X denoting forward, Y indicating right, and Z representing down.

Android Software Installation

We have developed a straightforward Android app for direct interaction with the RT-BLE-001 device. This app is primarily designed for data retrieval, on-screen result plotting, and data logging. It does not conduct additional processing, so the data is presented as received. Currently, the app isn't hosted on the Google Play Store and must be manually installed. Follow the procedure outlined below directly on your Android device:

- Through the Google Drive app, navigate to the ImuWear shared directory and tap on the app installation file (Android App/app-viewer-release.apk). This action will download the app to your local directory.
- Google Drive might require permissions to install apps. If this is the case, a message will alert you that an app has been detected and needs installation permission. When this prompt appears, select "Settings", then go to "Installing unknown apps" and enable "Allow from this source".
- Once the installation is complete, the app can be launched directly from the apps menu by selecting the "RT Foot" icon.
- The first time the application is launched, it will request access to various resources necessary for the program's operation. It needs access to location services, which are required for Bluetooth

communication, along with access to storage (photos, media, and files) for data logging purposes.

- Each time the app is launched, it checks to ensure Bluetooth is enabled. If it isn't, the user will be prompted to enable it. Do not manually pair the Android device with the foot sensor using the operating system's options, as this can prevent the app from recognizing the device.

Using the App

- Make sure the device is fully charged, a process that usually takes around 1.5 hours. The LED should display a steady green light.
- As soon as the app is launched, it will start scanning for RT-BLE-001 devices. All IMUwear available devices will be displayed on the screen. The user can press the SCAN button to initiate a new device search if needed.
- By selecting the device from the list, a foot tracking session will automatically begin and the device's LED will turn blue.
- The system will display the user's location on the screen. The map can be resized using typical pinch-to-zoom screen gestures. Users can double-tap at any point to recenter the trajectory back to its origin.

SCAN

- The system scans and lists all the nearby RT-BLE-001 devices.

DATA

- By selecting the "DATA" button in the main window, users can access the stored foot kinematic data and transfer it to another device using any available sharing service, such as Google Drive, Dropbox, or email.
- The file names adhere to the following convention:

  NS_SNSNSNSNSNSN_YYYYMMDDhhmmss.csv.

  SNSNSNSNSNSN corresponds to the 12-character serial name
  YYYY represents the year
  MM the month
  DD the day
  hh the hours in a 24-hour format
  mm the minutes
  ss the seconds

The date and time indicate when the initialization of the foot-tracker sample number occurred.

Log Data

- This feature enables the recording of position and orientation updates on the Android device. This recorded data can later be accessed via the "DATA" button.
- The logged data is formatted in comma-separated-values (CSV) and contains the following fields: Sample Number, Status, Positions (X, Y, Z), and Quaternions (Qa, Qb, Qc, Qd). Processing this file is straightforward (refer to the CSV Parsing section for guidance).
- The "Sample Number" represents when the sample was taken since the last reset of the RT-BLE-001 device (initiated via the 'Reset Location" option on the main screen). Given that the base sampling frequency is 100 Hz, this number denotes the number of hundreds of seconds that have passed since the last device reset. The actual time for any sample can be calculated by adding the "Sample Number" (first column in the CSV file) to the starting time (as indicated by the file name). Here's how to do it:

  Time(n) = (Sample_Number(n) - Sample_Number(1)) / 100 + YYYY:MM:DD:hh:mm:ss

  "Sample_Number(1)" refers to the first Sample Number and is only used if the device isn't reset during the session. If the device is reset, use zero instead.

Reset Location

- This option is utilized to initialize positioning and orientation. Upon using it, users should remain still immediately after the connection is established and until the dot indicating their location appears on the screen; this should occur within one to two seconds.

Continuous Mode

- Use this option to stream the complete foot trajectory during the session. When operating in this mode, you can choose from three sampling frequency options: 25Hz, 50Hz, or 100Hz. However, be aware that this mode may occasionally lead to data loss, as the device will attempt to stream the complete foot trajectory at the end of each stride. The risk of data loss may increase with higher frequencies.

CSV Parsing

Parsing the CSV log files collected using the Android app can be done with any spreadsheet application. As an example, you can print the foot/subject trajectory using the following Python program:

```
#plot_csv.py
import csv
import matplotlib.pyplot as plt
FILE = 'NS_DATETIME.csv'; # Replace with appropriate name
# Initialize variables
sample = []
status = []
x, y, z = ([] for i in range(3))
q0,q1, q2, q3 = ([] for i in range(4))
# Read CSV file
with open(FILE, mode='r') as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader) # Skips header
        for row in csv_reader:
        sample.append(int(row[0]))
        status.append(int(row[1]))
        x.append(float(row[2]))
        y.append(float(row[3]))
        z.append(float(row[4]))
        q0.append(float(row[5]))
        q1.append(float(row[6]))
        q2.append(float(row[7]))
        q3.append(float(row[8]))
# Display results
plt.ion()
plt.figure()
plt.plot(x, y)
plt.ylabel('Y (m)')
plt.xlabel('X (m)')
plt.figure()
plt.plot(q0)
plt.plot(q1)
plt.plot(q2)
plt.plot(q3)
plt.ylabel('Quaternions')
plt.xlabel('Sample #')
input("Press any key to terminate the program")
```

BLE Communication Protocol

This section elaborates on the BLE protocol employed for communication between any device (PC, smart device, microcontroller) and the RT-BLE-001 device. This information targets users familiar with the BLE communication interface.

The following are the UUIDs used by our device:

- Primary Service UUID = 6d071523-e3b8-4428-b7b6-b3f59c38b7bb

The primary service encompasses two characteristics:

- Data Characteristic UUID = 6d071524-e3b8-4428-b7b6-b3f59c38b7bb
- Control Characteristic UUID = 6d071525-e3b8-4428-b7b6-b3f59c38b7bb

Data Characteristic

The data characteristic is responsible for transmitting the estimated position and orientation values from the RT-BLE-001. To fit all the information into a single packet (maximum of 20 bytes with BLE4.0), we utilize 3-byte integers (a non-standard format) to convey the positional information, while the orientation (quaternions) are conveyed using 2 bytes. All information is presented in little-endian format (with the most significant bytes displayed last). Each packet is arranged as follows:

Packet Structure

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
C	C	S	X	X	X	Y	Y	Y	Z	Z	Z	Qa	Qa	Qb	Qb	Qc	Qc	Qd	Qd

The different fields in the packet structure correspond to:

- C = Counter
- S = Status
- X, Y, Z = Foot position X-forward, Y-right, and Z-down
- Qa, Qb, Qc, Qd = Foot orientation in Quaternion representation

The internal device counter (C) is provided as a 16-bit unsigned integer number and can be converted to time using a sampling frequency of approximately 100Hz.

The status (S) is non-zero when the foot is in contact with the ground.

Position (P) is given in millimeters, using a signed 3-byte format. Due to the 3-byte limit (2^23), the upper and lower limits roll over approximately every 8.3 km. This is a distance that most typical applications will not likely reach.

The Quaternions (Q*) are conveyed as signed 16-bit values. To convert them to the correct value, users must multiply the values by the scale factor, which is either 1/32768 or 1/2^15.

Control Characteristic

The control characteristics provide options for configuring the device. Currently, there are only three control commands available:

- **RESET COMMAND** (100 in decimal or 0x64 in hexadecimal): Upon receiving this command, the device resets the position and orientations to the coordinate origin.
- **FOOT FALL COMM MODE** (101 in decimal or 0x65 in hexadecimal): This is the default mode, and the system provides updates approximately once per second, or whenever the foot comes into contact with the ground during the stance phase.
- **CONTINUOUS COMM MODE** (102, 103, 104 in decimal or 0x66, 0x67, 0x68 in hexadecimal): In this mode, the device continuously transmits position and orientation at a sampling rate of approximately 25Hz (0x66), 50Hz (0x67), or 100Hz (0x68).

BLE Communication Sample Program

The Python program depicted below represents the bare minimum needed to establish BLE communication and initiate streaming. This program requires Python 3 (or a newer version) and the Bleak library. Bleak is a cross-platform BLE interface compatible with Windows, MAC, and Linux operating systems. You can find information on installing this library on the Bleak website (https://pypi.org/project/bleak/). Use the following instructions in a terminal to install the library:

Linux:

```
sudo pip install bleak
```

Windows 10

```
py -m pip install bleak
```

The program needs the Bluetooth MAC address (not to be confused with the WiFi MAC address) of the device as an argument. This address is equivalent to the serial number, as found on the device's label. The Bluetooth MAC address must have a colon ":" after every two characters. For instance, for a device labeled with the serial number "SN 123456789ABC", use the following instruction:

Linux:

```
python3 getdata_rtfoot.py 12:34:56:78:9A:BC
```

Windows 10

```
py getdata_rtfoot.py 12:34:56:78:9A:BC
```

The source code for this minimalistic program is provided below. You can use the CTRL-C key combination to exit the program at any time. A more comprehensive program example is available for download from our shared drive. This version can handle two additional optional arguments: "r" to reset the system and "cFREQ" to switch the update rates from "foot-fall" (default) to "continuous" mode, with a specified frequency (FRQ: 25, 50, or 100).

```
# getdata_rtfoot.py
from bleak import BleakClient
import asyncio
import struct
import signal
import sys
import time
DATA_CHARACTERISTIC_UUID = "6d071524-e3b8-4428-b7b6-b3f59c38b7bb"
g_quit = False
def signal_handler(sig, frame): # if CTRL-C, quit program
        global g_quit
        print('Quitting program!')
        g_quit = True
def notification_handler(sender, data): # Parse data packet
        # Packet structure (20 bytes): Sample(2), Status(1), X(3), Y(3), Z(3), Q0(2), Q1(2), Q2(2), Q3(2)
        # Sample number = data[0 ..1]
        sample = struct.unpack('<H', data[0:2])[0]
        print("Sample:", sample)
        # Status byte = data[2]
        status = struct.unpack('B', data[2:3])[0]
        print("Status:", status)
        # Positions (3-byte int): X=data[3..5], Y=data[6...8], Z=data[9...11]
        x = struct.unpack('<i',data[2:6])[0] / 256
        y = struct.unpack('<i',data[5:9])[0] / 256
        z = struct.unpack('<i',data[8:12])[0] / 256
        print ("Pos:", x, y, z)
        # Quaternions (short int): Q0=data[12..13], Q1=data[14..15], Q2=data[16..17], Q3=data[18..19]
        q0 = float(struct.unpack('<h', data[12:14])[0]) / 32768
        q1 = float(struct.unpack('<h', data[14:16])[0]) / 32768
        q2 = float(struct.unpack('<h', data[16:18])[0]) / 32768
        q3 = float(struct.unpack('<h', data[18:20])[0]) / 32768
        print("Qua:", q0, q1, q2, q3)
async def run(argv): # BLE comms
        async with BleakClient(argv[1], timeout = 4.0) as client:
        # Establish connection
        await client.is_connected()
        print("Connected with RT-BLE device")
        # Reset device
         await client.write_gatt_char(CRTL_CHARACTERISTIC_UUID,  bytearray([CRTL_RESET]))
        # Set stream mode
         await client.write_gatt_char(CRTL_CHARACTERISTIC_UUID,  bytearray([CRTL_FOOTFALL_MODE]))
        # Stream
        await client.start_notify(DATA_CHARACTERISTIC_UUID, notification_handler)
        while (not g_quit): # quit with CRTL-C
                await asyncio.sleep(1.0)
# Main program
print('Usage:\n  python3 getdata_rtfoot.py MAC_ADDRESS')
signal.signal(signal.SIGINT, signal_handler)
loop = asyncio.get_event_loop()
loop.run_until_complete(run(sys.argv))
```