We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

154
Countries delivered to

Our authors are among the

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS
BOOK CITATION INDEX
INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Bluetooth Low Energy Applications in MATLAB

Septimiu Mischie

Abstract

This chapter presents Bluetooth Low Energy (BLE) applications in MATLAB. Through these applications we acquire measurement data from BLE compatible sensors to PC. The sensors are CC2541 Keyfob and CC2650 Sensor Tag. The first one contains an accelerometer and a temperature sensor while the second one contains more sensors, but inertial sensors and magnetometer are invoked. The PC should be equipped with a general USB BLE adapter. The most important steps for implementing a BLE application are presented: scanning, connecting, configuring and data reading. Following this, more detailed applications are presented: a wireless sensor network for temperature measurement with three Keyfob-based nodes, an application that displays in real time accelerometer data and a heading computed method using either the gyroscope or the magnetometer of CC2650 Sensor Tag. The most important MATLAB elements that are used to implement these applications are different types of variables such as structure, table and object, methods to implement endless loops and real-time display of acquired data and using quaternions to handle 3D orientation of a device.

Keywords: MATLAB, Bluetooth low energy, temperature sensors, movement sensors, callback function, quaternion, 3D orientation

1. Introduction

MATLAB represents a programming language that is used for designing, simulating and testing of different technical systems [1]. This chapter provides examples of Bluetooth low energy (BLE) applications implemented in MATLAB. In this section, the main aspects regarding developing a BLE MATLAB application are presented. First of all, basics about BLE technology are presented [2–5]. BLE means exchange data between two or more devices by radio waves over short distances. Mainly, a BLE device can be scanner or advertiser. The advertiser signals its presence by sending its name and address. The scanner finds advertiser devices and can connect to one or more of them. Then the advertiser becomes a server and can send data to the scanner which is now a client. According to BLE architecture, the server can offer services to the client. Some examples of services are battery service, accelerometer service and heart rate measurements. Each service contains more characteristics. The most important attribute of a characteristic is its value, which in general represents sensor data. In addition, a characteristic has one or more of the following properties: read, write and notify.

Starting with 2019b release, MATLAB has introduced a set of functions that allow a simple implementation of BLE application [6]. The minimum setup involves a laptop having an embedded BLE adaptor or a desktop having an USB BLE adapter and some BLE compatible devices.

Scanning for BLE devices can be done using *blelist* function. Connecting to one device can then be done by *ble()* function, as in the examples of **Figure 1**.

It can be seen that three BLE devices have been discovered. The connection with Keyfobde99 is achieved, and b1 is a *ble* object having 5 fields, the last two being Services and Characteristics so in previous paragraph was presented. Then, in order to get data from a BLE device, the characteristics have to be accessed. **Figure 2** presents the last part of the Characteristics variable.

A characteristic can be accessed either by service name and characteristic name or by service universal unique identifier (UUID) and characteristic (UUID) [6]. According to the information from **Figure 2** it can be seen that only the second option can be used because there are more Custom services or characteristics.

```
>> list=blelist('timeout',15)

list =

  3×5 table

    Index          Name              Address        RSSI    Advertisement
    _____    _____    _____   ____    _____

      1      "CC2650 SensorTag"   "B0B448BD7E05"     -50     [1×1 struct]
      2      "Multi-Sensor"       "1804EDBEF596"     -56     [1×1 struct]
      3      "Keyfobde99"         "84DD20C50B29"     -71     [1×1 struct]

>> b1=ble("84DD20C50B29")

b1 =

  ble with properties:

               Name: "Keyfobde99"
            Address: "84DD20C50B29"
          Connected: 1
           Services: [10×2 table]
    Characteristics: [29×5 table]

  Show services and characteristics

>>
```

Figure 1.
Using blelist *and* ble() *functions.*

```
>> table=b1.Characteristics(18:26,1:4)

table =

  9×4 table

      ServiceName          ServiceUUID      CharacteristicName      CharacteristicUUID
    _____    _____    _____    _____

    "Tx Power"           "1804"           "Tx Power Level"         "2A07"
    "Battery Service"    "180F"           "Battery Level"          "2A19"
    "Custom"             "FFA0"           "Custom"                 "FFA1"
    "Custom"             "FFA0"           "Custom"                 "FFA2"
    "Custom"             "FFA0"           "Custom"                 "FFA3"
    "Custom"             "FFA0"           "Custom"                 "FFA4"
    "Custom"             "FFA0"           "Custom"                 "FFA5"
    "Custom"             "FFA0"           "Custom"                 "FFA6"
    "Custom"             "FFA0"           "Custom"                 "FFA7"

>>
```

Figure 2.
A part of services and characteristics of a ble *variable.*

Furthermore, there is no information about the functionalities of these services or characteristics. Therefore, to start the application, some information is necessary that can be obtained using another application such as BLE Device Monitor [7] or just Wikipedia [8]. Thus, **Table 1** presents service name and characteristic name among to UUID for a part of the positions of **Figure 2**.

In order to access a characteristic, the *characteristic* () function can be used, as in **Figure 3**. Then, after defining x variable, the returned value can be read by function *read*(x).

Another powerful features is *DataAvailbleFcn* than can be assigned to a characteristics that has the Notify attribute. This can be done as in **Figure 4**. When a new data is available, this callback function is called.

Service Name	UUID	Characteristic Name	UUID
Accelerom. service	FFA0	Accelerometer enable	FFA1
		Accelerometer range	FFA2
		Accelerometer X coordinate	FFA3
		Accelerometer Y coordinate	FFA4
		Accelerometer Z coordinate	FFA5
		Period of Reading accelerometer data	FFA6
		Temperature	FFA7

Table 1.
UUID for a service and its characteristics.

```
>> enable=characteristic(b1,"FFA0","FFA1");
>> write(enable,1); %enable accelerometer
>> x=characteristic(b1,"FFA0","FFA3"); %x axis
>> data_x=read(x)

data_x =

     66

>>
```

Figure 3.
Using the characteristic*() function.*

```
Command Window
>> x.DataAvailableFcn=@displayCharacteristicData_x

x =

  Characteristic with properties:

              Name: "Custom"
              UUID: "FFA3"
        Attributes: "Notify"
       Descriptors: [2x3 table]
 DataAvailableFcn: displayCharacteristicData_x

Show descriptors

fx >>
```
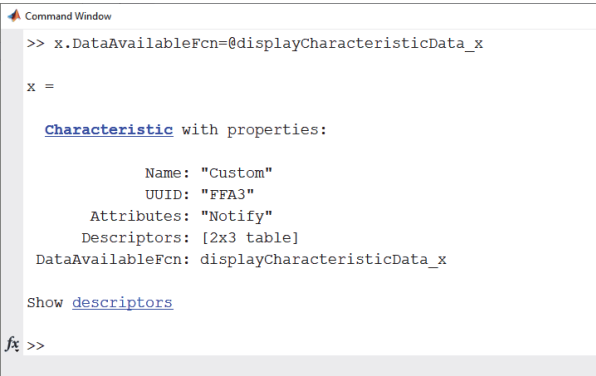
Figure 4.
Create the callback function.

Each of the following sections contains an introduction where the basic function of the program is presented, which is then followed by the program and the results, mainly in graphical form.

2. BLE network sensors for temperature monitoring

This section presents a MATLAB application that uses three temperature sensors. CC2541 Keyfob [9] is a BLE compatible device that contains an accelerometer. Among its basic function, the accelerometer contains an 8-bit temperature sensor. To access the temperature sensor the accelerometer must be enabled first and then the temperature characteristic can be accessed according to **Table 1**. The period of reading temperature is 3 sec. according to the author publication [10].

At the beginning of the program a general scanning is executed and if none of the desired sensors are discovered the application is stopped through a suitable message on the screen. To do this, accessing the elements of a variable of table type, *list*, is performed.

Then, depending on the discovered number of sensors, which can be from one to any number (three in this application) the application gets temperature from them and displays it on a graphic. For this purpose, two structures, *s_enable* and *s_x*, having a variable number of fields have been created. Number of fields will be equal with the discovered number of sensors. The structure *s_x* is used to assign a callback function for each discovered sensor, too. Furthermore, the number of matrix of small axes which are generated by subplot function is equal with the number of discovered sensors.

The structure of the program is presented below.

```
%%%%%Measure the temperature by one, two or three
%%%%%Key-fob devices%%%%%%%%%%%%%%%%%%%%
clear; close all;
global V_temp1;global V_temp2;global V_temp3
global V_time1;global V_time2;global V_time3
global N % the last N temperatures of each sensor
N=20;
V_temp1=zeros(1,N);V_temp2=zeros(1,N);V_temp3=zeros(1,N);
d=datetime;
V_time1=repmat(d,N,1);V_time2=repmat(d,N,1);V_time3=repmat(d,N,1);
list=blelist; %scan
L=size(list);
Nr=L(1);
disp(['Total number of BLE devices: ' num2str(Nr)])
if Nr==0
return
end
j=1;
for i=1:Nr
if (list.Address(i)=="84DD20C50B29" | list.Address(i)=="544A165E18AB" |
list.Address(i)=="20C38FD12605")
b(j)=ble(list.Address(i));
j=j+1;
end
end
Ng=j-1; % Number of Keyfobb devices
```

```
disp(['Number of Keyfob devices: ' num2str(Ng)])
if Ng==0
disp('No devices')
return
end
s_enable=struct; %initialize the structure
tab='abcdef'; %the fields of the structure
for i=1:Ng
s_enable.(tab(i))=characteristic(b(i),"FFA0","FFA1");
write( s_enable.(tab(i)),1);
end
s_x=struct; %initialize the structure
for i=1:Ng
s_x.(tab(i))=characteristic(b(i),"FFA0","FFA7");
s_x.(tab(i)).DataAvailableFcn = eval(['@displayCharacteristicData_temp'
num2str(i)]);
end
h=figure(1);
while (ishandle(h))
for i=1:Ng
var =eval(['V_temp' num2str(i)]);
timp =eval(['V_time' num2str(i)]);
subplot(Ng,1,i);plot(timp,var,'*');grid;title(['sensor' num2str(i)]);ylim([0
30])
xlabel('time')
ylabel('temperature, C')
drawnow;
end
pause(3)
end
clear b
%One of the three callback functions:
function displayCharacteristicData_temp1(src,evt)
global V_temp1
global V_time1
```
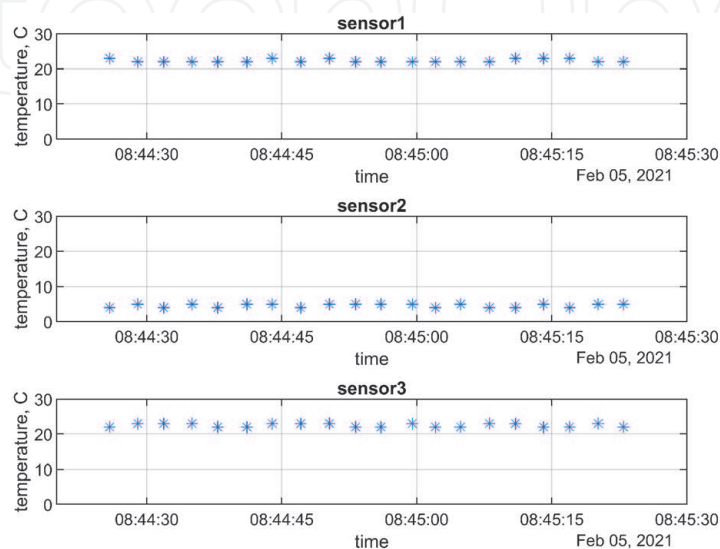

Figure 5.
The temperatures from the three sensors.

```
global N
[temp1] = read(src,'oldest');
time1=datetime(datestr(now,'HH:MM:SS.FFF'));
%update the last N=20 samples of temperature and time
V_temp1(1:N-1)=V_temp1(2:N);
V_temp1(N)=temp1;
V_time1(1:N-1)=V_time1(2:N);
V_time1(N)=time1;
end
```

The program runs in an endless loop and displays the last 20 values of the three temperatures in a MATLAB figure as in **Figure 5**. To stop the program, simply close the figure. In addition, the current date and time is displayed on the figure. One of the CC2541 Keyfob was on the outside sill of the window and therefore the resulting temperature was about 5 degree Celsius.

3. Using the accelerometer of CC2541Keyfob

This section presents a MATLAB application that accesses the accelerometer of the CC2541 Keyfob to read the 8-bit accelerations corresponding to the three axes. The program is similar to that of the previous section. There are also three callback functions, one for each axes. The period of reading data is set to 100 ms.

```
%%%%%%%%Display in real-time the last samples of three accelerations
%%%%%%%%of each axis %%%%%%%%%%%%%%%%%%%%%
clear all; close all;
b=ble("84DD20C50B29"); %BLE connection with 84DD20C50B29
enable=characteristic(b,"FFA0","FFA1")
write(enable,1) %enable accelerometer
x=characteristic(b,"FFA0","FFA3")
y=characteristic(b,"FFA0","FFA4")
z=characteristic(b,"FFA0","FFA5")
per=characteristic(b,"FFA0","FFA6")
read_per=read(per);
write(per,10);%establish the reading period to 10*10ms
read_per1=read(per);
global vector_x;global vector_y;global vector_z
global N; %number of samples
N=200;
vector_x=zeros(1,N;vector_y=zeros(1,N);vector_z=zeros(1,N);
axa =1:N;
x.DataAvailableFcn = @displayCharacteristicData_x; %functii callback,
they are %executed when sensor data are available
y.DataAvailableFcn = @displayCharacteristicData_y;
z.DataAvailableFcn = @displayCharacteristicData_z;
h=figure(1); %create the figure
while(ishandle(h)) %while the figure does exist data is displayed plot(axa,
vector_x*19.62/128,axa,vector_y*19.62/128,axa,vector_z*19.62/128);grid;leg-
end('ax','ay','az')
ylabel('accel., m/s2')
xlabel(' The most recent samples')
drawnow;
```
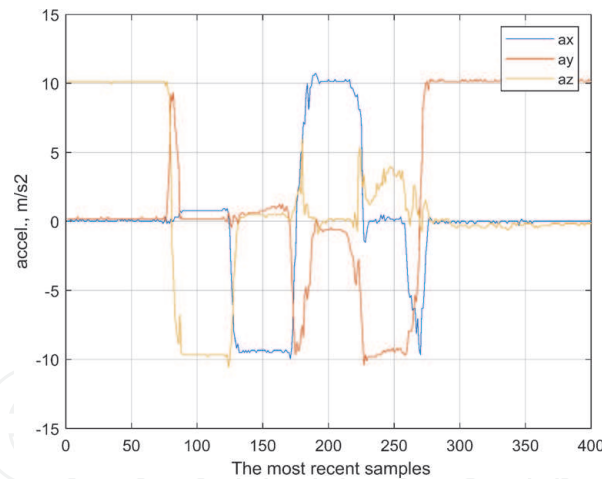
Figure 6.
The variation of the three accelerations.

```
end
clear b %clear the variable that represents the BLE connection or disconnect
%One of the three callback functions:
function displayCharacteristicData_x(src,evt)
global vector_x
global N
[data,timestamp] = read(src,'oldest');
fprintf('Time1 %s\n', datestr(now,'HH:MM:SS.FFF'))
if data>128
data=data-256;
end
%update the last N=200 samples of acceleration
vector_x(1:N-1)=vector_x(2:N);
vector_x(N)=data;
end
```

The program runs in an endless loop and displays the last N=200 samples of each the three axes. **Figure 6** presents a screenshot during the running of the program. During this time CC2541 Keyfob was moved such as one of the three axes was on the direction of gravitational force. Thus, most of the time one of the three axes has the absolute value close to g=9.81 m/s^2 while the other two are close to zero.

4. Using the movement sensor of CC2650 Sensor Tag

This section presents a MATLAB application that accesses the movement sensor of the device called CC2650 Sensor Tag. This movement sensor contains an accelerometer, a gyroscope and a magnetometer. If the accelerometer of the CC2541 Keyfob which was presented in the third section generates 8-bit data, all of the three sensors of CC2650 Sensor Tag generates 16-bit data.

The gyroscope is a three axis sensor that measures the angular rate, $\omega(t)$. By integrating the angular rate, the angular position $\alpha(t)$ is obtained as

$$\alpha(t) = \int \omega(t)dt \qquad (1)$$

Thus, Eq. (1) can be implemented by trapezoidal method by using samples of $\omega(t)$ by

$$\alpha(t) = \alpha(t-1) + \frac{\omega(t) - \omega(t-1)}{2}dt \tag{2}$$

where dt is reciprocal of sample rate.

This angle is considered in comparison with the initial position of the gyroscope which is unknown. Using the integration, it generates an error because the gyroscope has an offset. That means its output is different to zero when the gyroscope is still. Thus, by integration it follows that the angle is changed. Therefore this offset must be removed [11].

The magnetometer measures the magnetic field. Thus, if there are no other fields, it measures the magnetic field of the earth. When the magnetometer is placed horizontally, it can measure the angle from the north, h, by

$$h = atan \frac{m_y}{m_x} \tag{3}$$

where m_x and m_y are its readings. Thus, both gyroscope and magnetometer can measure the same angle but the magnetometer has a reference which is the north. For this reason this angle is also called heading. Similar to the gyroscope, the magnetometer has a drawback too. Thus, its reading must be corrected by a process called calibration [12]. Basically, this implies a rotation of 360 degrees around its z axis in both senses followed by computation of calibrated data m_{xcal} and m_{ycal},

$$m_{xcal} = X_g \cdot m_x - X_{off} \tag{4}$$

and

$$m_{ycal} = Y_g \cdot m_y - Y_{off} \tag{5}$$

where

$$X_g = \frac{max\left\{(m_{x\ max} - m_{x\ min}), (m_{y\ max} - m_{y\ min})\right\}}{m_{x\ max} - m_{x\ min}} \tag{6}$$

and

$$X_{off} = X_g \left(\frac{m_{x\ max} + m_{x\ min}}{2}\right) \tag{7}$$

while Y_g and Y_{off} have similar expressions. Also the magnetometer is very sensitive to the magnetic perturbations that can be generated by other materials from its proximity.

Regarding BLE, CC2650 Sensor Tag offers more services. The service that allows accessing the accelerometer, the gyroscope and the magnetometer has three characteristics, as shown in **Table 2**, where also the UUID can be seen. The first one is

Service Name	UUID	Char Name	UUID
Movement	F000AA80	Data	F000AA81
		Configure sensors	F000AA82
		Period	F000AA83

Table 2.
UUID for the movement service and its characteristics.

used to read data. The second one is used to enable the sensors. Each axis of the gyroscope and accelerometer can individually be enabled while the magnetometer can be enabled only for all axes. The third characteristic allows to establish the period of data reading. The data is presented as an 18 bytes string, where each sensor has a field of 6 bytes, two bytes for each axis, in the order: gyroscope, accelerometer and magnetometer. Actually UUID contain more digits but only the different part is presented in **Table 2** [8].

Because in this case the MATLAB programs are much longer than previous ones only some parts of the achieved programs are presented. Mainly, such a program has three parts:

- the first part when the sensor is still for a time while gyroscope data are gathered to compensate its offset; generally in this part 200 samples are acquired;

- the second part when the sensor is rotated with 360 degrees in both senses around z axis while the magnetometer data are gathered to compute the calibration; in this part also 200 samples are acquired;

- the last part has an indefinite duration when the sensor is moved while real-time data are displayed on the different figures.

Two programs are achieved, depending of the content of the last part. In this case there is a script and only a callback function. The period of data reading is 200 ms.

The first program computes and displays basic results from gyroscope and magnetometer.

Mainly it computes:

```
om=[om_x om_y om_z]; %%the vector with gyroscope readings
A=[a_x a_y a_z]; %%the vector with accelerometer readings
M=[m_x m_y m_z]; %%the vector with magnetometer readings
ang_z_g %the angular position (angle) around z axis, by gyroscope
ang_z_m %the angular position (angle) around z axis, by magnetometer
```

All the time the last N=200 samples of these measurements are available. A part of the script is presented in the following.

```
clear;
close all;
global i
i=0; %%this index is used inside the callback function
global ang_z_g; global om_z_prev; global ang_z_m
global offset_x; global offset_y; global offset_z;
ang_z_g=0; om_z_prev=0; ang_z_m=0
offset_x=0; offset_y=0; offset_z=0 ;
bb=ble("B0B448BD7E05");% the address of CC2650 Sensor Tag
%%%%the scaling constants, according to the range
sca_a=8*9.81/32768; %%the accelerometer range is +-8g
sca_m=4800/32768; %%the magnetometer range is +-4800 uT
sca_om=250/32768; %%the gyroscope range is +-250 deg/s
conf=characteristic(bb, "F000AA80-0451-4000-B000-000000000000" ,
"F000AA82-0451-4000-B000-000000000000" );
write(conf,[127 02]) %%enable all 9 axes, 01 111 111
```

```
per=characteristic(bb, "F000AA80-0451-4000-B000-000000000000" ,
"F000AA83-0451-4000-B000-000000000000" );
coef=20;T=10*coef*0.001; %%period of reading in seconds
write(per,coef) %%coef can be minimum 10
data=characteristic(bb, "F000AA80-0451-4000-B000-000000000000" ,
"F000AA81-0451-4000-B000-000000000000" );
data.DataAvailableFcn = @displayCharacteristicData_STag;
%A part of the callback function is presented in the following:
function displayCharacteristicData_STag(src,evt)
global i; global ang_z_g; global om_z_prev; global ang_z_m
global offset_x; global offset_y; global offset_z;
[data,timestamp] = read(src,'oldest'); %%read the current data
%%the gyroscope data
if data(2)>128
om_x=data(2)*256+data(1)-2^16;
else
om_x=data(2)*256+data(1);
end
if data(4)>128
om_y=data(4)*256+data(3)-2^16;
else
om_y=data(4)*256+data(3);
end
if data(6)>128
om_z=data(6)*256+data(5)-2^16;
else
om_z=data(6)*256+data(5);
end
om_x=om_x*sca_om;
om_y=om_y*sca_om;
om_z=om_z*sca_om;
%%correct the gyroscope's offset
if i<=200
offset_x=offset_x+om_x;
offset_y=offset_y+om_y;
offset_z=offset_z+om_z;
i=i+1;
end
om_x=om_x-offset_x/200;
om_y=om_y-offset_y/200;
om_z=om_z-offset_z/200;
om=[om_x om_y om_z]; %%the vector with gyroscope readings
if i>200
%compute the current angle by integration
ang_z_g=ang_z_g+(om_z+om_z_prev)*T/2; %T is reading period
om_z_prev=om_z; %previous value becomes current value
%%the accelerometer data
if data(8)>128
a_x=data(8)*256+data(7)-2^16;
else
a_x=data(8)*256+data(7);
end
if data(10)>128
```

```
a_y=data(10)*256+data(9)-2^16;
else
a_y=data(10)*256+data(9);
end
if data(12)>128
a_z=data(12)*256+data(11)-2^16;
else
a_z=data(12)*256+data(11);
end
a_x=a_x*sca_a;
a_y=a_y*sca_a;
a_z=a_z*sca_a;
A=[a_x a_y a_z]; %%the vector with accelerometer readings
%%%%the magnetometer data
if data(14)>128
m_x=data(14)*256+data(13)-2^16;
else
m_x=data(14)*256+data(13);
end
if data(16)>128
m_y=data(16)*256+data(15)-2^16;
else
m_y=data(16)*256+data(15);
end
if data(18)>128
m_z=data(18)*256+data(17)-2^16;
else
m_z=data(18)*256+data(17);
end
m_x=m_x*sca_m;
m_y=m_y*sca_m;
m_z=m_z*sca_m;
M=[m_x m_y m_z]; %%the vector with magnetometer readings
if i>400
%%compute the heading angle using magnetometer
AA= m_y*Ysf+Yoff; %Ysf and Yoff are constants for calibration
BB= m_x*Xsf+Xoff; %Xsf and Xoff are constants for calibration
%the part of the program that computes the above constants is not %
presented
ang_z_m=180/pi*atan2(AA,BB) %%the magnetometer-based heading %angle
end
end %%end if i>200
```

By running the previous program, the last 200 samples of some of the measurements obtained from the gyroscope and magnetometer are displayed in real-time.

Thus, the two waveforms of the top of **Figure 7** are achieved using the gyroscope while the other two from the bottom part are achieved using the magnetometer. In each case the heading is computed. The gyroscope-based angle around z axis or heading is computed using the angular rate around the z axis, see the second wave-form. The heading computed by the magnetometer presented in the third waveform is based on its x and y reading which are presented in the last waveform. It can be seen that the two waveforms that represent the heading have the same variation, except at
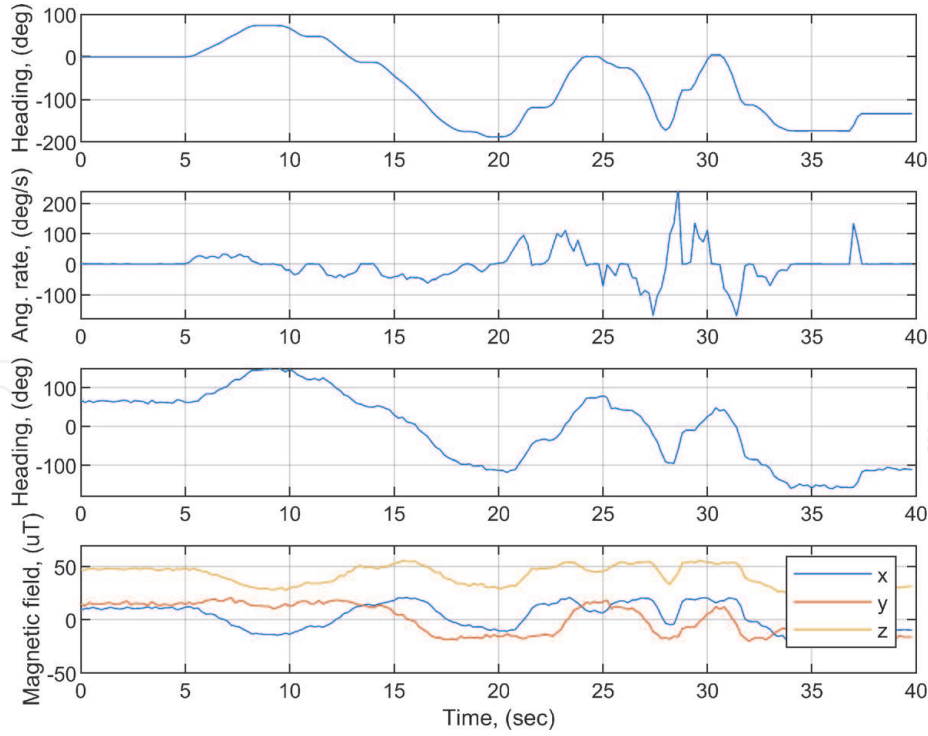
Figure 7.
Some measurements obtained from gyroscope (the two waveforms at the top) and magnetometer (the two waveforms at the bottom).

the start. Thus the gyroscope-based heading starts from zero while magnetometer-based heading starts from about 60 degrees because it indicates the north.

By using the movement sensors, 3D orientation of a device can be computed [13–17]. This can be represented in three ways: quaternion, Direction Cosine Matrix (DCM) and Euler angles. The last representation means the rotational angles around the three axes, called pitch, roll and yaw but has a disadvantage because can reach in a singularity state. DCM does not have a singularity state but needs 3x3 elements. Thus the best representation is quaternion which represents a complex number having four components [13],

$$q = \left[cos \frac{\alpha}{2} \quad e_x \, sin \frac{\alpha}{2} \quad e_y \, sin \frac{\alpha}{2} \quad e_z \, sin \frac{\alpha}{2} \right] \tag{8}$$

where α is the rotation angle and e represents the rotation axis.

Using the accelerometer allows only computing pitch and roll angles because a rotation around z axis does not change any of the three outputs. Thus the four elements of the quaternion, denoted q_{acc} can be computed by [13]

$$q_{acc} = \begin{cases} \left[\sqrt{\dfrac{a_z + 1}{2}} \quad -\dfrac{a_y}{\sqrt{2(a_z + 1)}} \quad \dfrac{a_x}{\sqrt{2(a_z + 1)}} \quad 0 \right], & a_z \geq 0 \\[3ex] \left[-\dfrac{a_y}{\sqrt{2(1 - a_z)}} \quad \sqrt{\dfrac{1 - a_z}{2}} \quad 0 \quad \dfrac{a_x}{\sqrt{2(1 - a_z)}} \right], & a_z < 0 \end{cases} \tag{9}$$

Eq. (9) can be very easily implemented in MATLAB and then the quaternion can be generated by the function *quaternion*(). However, in order to compute the third angle (yaw) the magnetometer or gyroscope readings are necessary and more difficult equations are generated [13]. Thus, a better solution is using the functions from Sensor Fusion and Tracking Toolbox of MATLAB [18] that allows estimation of 3D orientation. The function *imufilter*() uses only accelerometer and gyroscope

while the function *complementaryFilter*() uses all the three sensors. Thus the **Figure 8** presents in the top panel the angles obtained by MATLAB implementation of Eq. (9) while in the bottom panel are presented the results obtained using the *imufilter*() function. It can be seen that the waveforms are very similar and also the angle around z axis is zero for the first or close to zero for the second as expected.

Both *imufilter*() and *complementaryFilter*() return the result as an object. Then the object can be called as a function having the sensor measurements as arguments and returns a quaternion. If this quaternion becomes the argument of the *viewer*() function [16] it follows the display of a cube that moves in real-time that imitates the moving of the CC2650 Sensor Tag, as in **Figure 9**. However, two files,
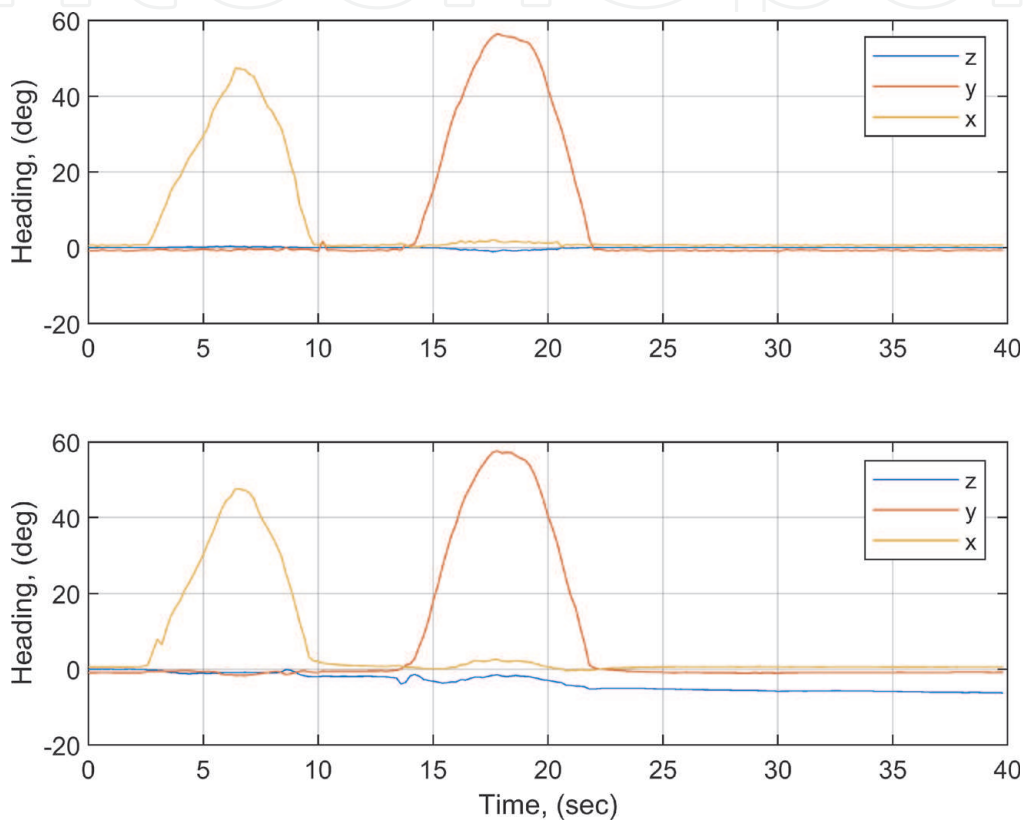
Figure 8.
The Euler angles computed by Eq. (9), top, and by imufilter(), bottom; the sensor was moved around x and y axis.
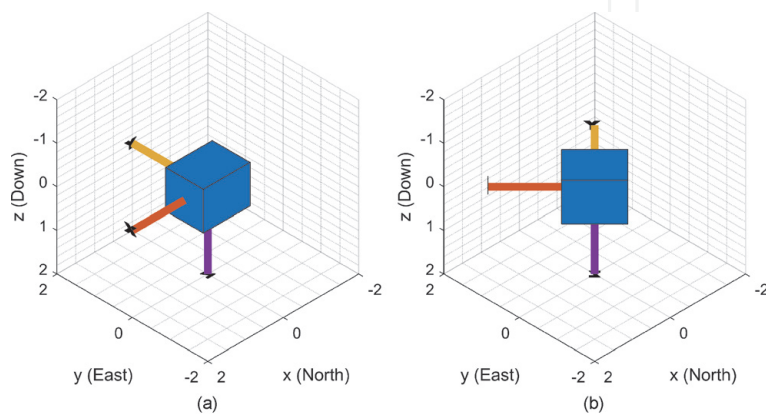

Figure 9.
The cube that imitates the moving of the CC2650 Sensor Tag, the initial position-left, the position after a 45 degrees rotation around z axis-right.
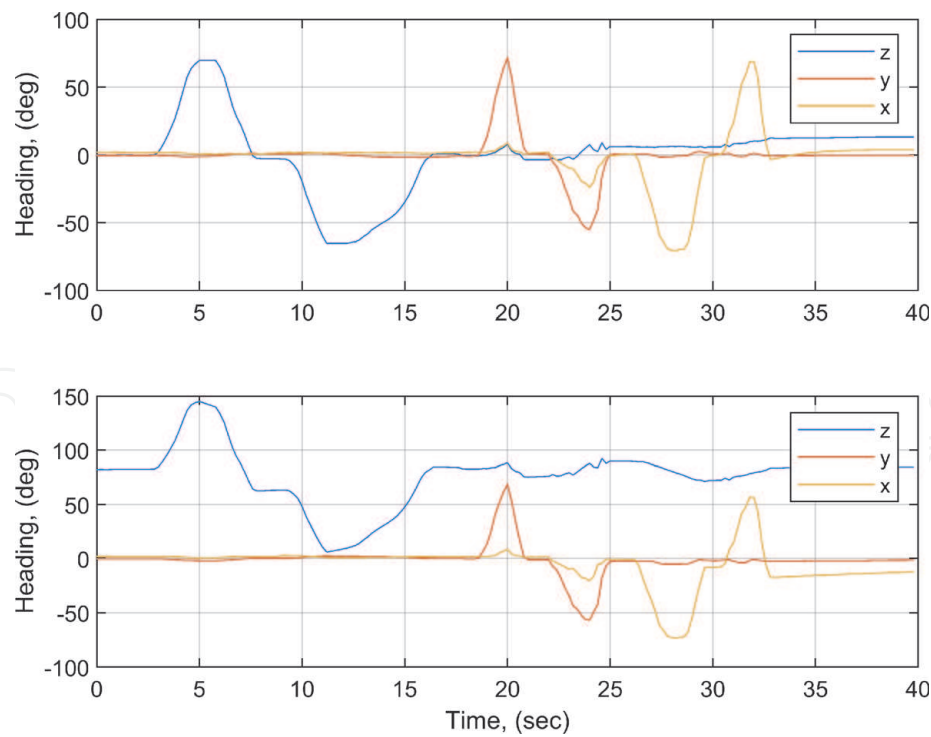
Figure 10.
The Euler angles computed by imufilter*(), top, and by* complementaryFilter*(), bottom; the sensor was moved around all the three axes.*

HelperOrientationViewer.m and HelperBox.m, must be in the current folder in order to use the function *viewer*() [18].

On the other hand, using the function *eulerd*(), the quaternion form can be converted in pitch, roll and yaw angles (Euler angles) that can be represented as waveforms. This method was used to obtain the results in **Figure 8**. Now **Figure 10** presents the three angles computed using *imufilter*(), in the top part and *complementaryFilter*(), in the bottom part. It can be seen that the angles have the same variation except that the heading around the z axis starts with zero at the top and by about 80 degrees in the bottom. This is because the *complementaryFilter*() uses the magnetometer and thus indicates the angle with respect to the north.

The program that implements these facilities is very similar to the previous one. In the following sections, only the new elements are presented.

First the new elements of the script are presented.

```
viewer = HelperOrientationViewer;
SRate=1/T;
ifilt_imu = imufilter('SampleRate', SRate);
ifilt_com = complementaryFilter('SampleRate', SRate);
h=figure(1);
while(ishandle(h))
plot(v_om');grid;legend('x','y','z') %%display the gyroscope readings
drawnow;
if i>200
viewer(qahrs_imu); %%imu quaternions are used to move the cube
end
end

%Then it follows the new elements of the callback function.
%%%%%%%%%%%%%%%%%%%%%quaternion%%%%%%%%%%%%
```

```
qahrs_imu=ifilt_imu(A,om*pi/180); %% for imufilter
qahrs_com=ifilt_com(A,om*pi/180,M); %%for complementaryFilter
eulerAnglesDegrees_imu=eulerd(qahrs_imu, 'ZYX','frame')
eulerAnglesDegrees_com=eulerd(qahrs_com, 'ZYX','frame')
```

5. Conclusions

The main MATLAB contributions of this chapter are:

- using the new introduced MATLAB functions to access BLE devices and to implement a BLE sensors network

- using the table type MATLAB to check if the desired sensors are among the discovered BLE devices

- using the structure type MATLAB having a variable number of fields to handle the discovered number of BLE devices

- retaining and updating the most recent samples of different measurements corresponding to BLE sensors and display them in real-time

- using the quaternions to handle the 3D orientation of an object

- using the new introduced MATLAB functions from Sensor Fusion and Tracking Toolbox to determine the parameters that describes 3D orientation

- displaying the cube that imitates in real-time the moving of CC2650 Sensor Tag

- as a future work, the MATLAB can be used to estimate the position of an object along with 3D orientation; in this way the tracking of an object can be completed.

Author details

Septimiu Mischie
Politehnica University Timisoara, Timisoara, Romania

*Address all correspondence to: septimiu.mischie@upt.ro

IntechOpen

References

[1] Katsikis V, editor. MATLAB. A fundamental tool for scientific computing and engineering applications. Volume1. IntechOpen; 2012. DOI: 10.5772/2557, Volume 2. IntechOpen; 2012. DOI: 10.5772/3338, Volume 3 IntechOpen; 2012. DOI: 10.5772/3339

[2] Mohamad Omar Al Kalaa, Refai H. Bluetooth Standard v4.1: Simulation the Bluetooth Low Energy Data Channel Selection Algorithm. In: Proceedings of Globecom 2014 Workshop – Telecommunication Standards – From Research to Standards. P. 729-733

[3] Afaneh M. Bluetooth5 & Bluetooth Low Energy. A Developer's Guide, e-book, 2018. Available from: https://www.novelbits.io/bluetooth-5-developers-e-book/ [Accessed: 2020-12-02]

[4] Bluetooth. Available from https://www.bluetooth.com/ [Accessed: 2020-12-02]

[5] Wu Taiyang, Redoute J.M, Yuce M. A wireless Implantable Sensor Design with Subcutaneous Energy Harvesting for Long-Term IoT Healthcare Applications. In IEEE Access, vol. 6, 2018, p.35801-35808.

[6] MATLAB Bluetooth Low Energy. Available from https://www.math works.com/help/matlab/bluetooth-low-energy-communication.html [Accessed: 2020-12-02]

[7] BLE Device Monitor. Available from https://processors.wiki.ti.com/index. php/BLE_Device_Monitor_User_Guide [Accessed: 2020-12-02]

[8] CC2650 Sensor Tag. Available from https://processors.wiki.ti.com/index. php/CC2650_SensorTag_User%27s_ Guide [Accessed: 2020-12-02]

[9] CC2541 Keyfob, Available from https://www.ti.com/tool/ CC2541KEYFOB-RD [Accessed: 2020-12-02]

[10] Mischie S. On the Development of Bluetooth Low Energy Devices. In: Proceedings of COMM 2018, Bucharest. p.339-344

[11] Mischie S. A MATLAB Graphical Interface to evaluate CC2650 Sensor Tag. In: Proceedings of 22nd IMEKO-TC4 International Symposium, Iasi, Romania, 2017

[12] Fang J, Sun H, Zhang X, Tao Y. A novel Calibration Method of Magnetic Compass Based on Ellipsoid Fitting. IEEE Trans. on Instrumentation and Measurement, vol. 60, no.6, June 2011, p. 2053-2061

[13] Valenti R, Dryanovski I, Xiao J. A Linear Kalman Filter for MARG Orientation Estimation Using the Algebraic Quaternion Algorithm. In IEEE Trans. On Instrumentation and Measurement, vol. 65, no.2 2016, p. 467-481, DOI:10.1109/ TIM.2015.2498998

[14] Yadav R.H, Bhattarai B., Gang H.S. Pyiun J.Y. Trusted K Nearest Batesian Estimation for Indoor Positioning System. In IEEE Access, vo. 7 2019, p.51484-51498. DOI: 10.1109/ ACCESS.2019.2910314

[15] Manos A., Kleian I., Hazan T. Gravity-Based Methods for Heading Computation in Pedestrian Dead Reckoning. In Sensors 2019, 19(5), 1170. DOI: 10.3390/s19051170

[16] Li G., Geng E. Ye Z., Xu Y., Lin J., Pang Y. Indoor Positioning Algorithm Based on the Improved RSSI Distance Model, In Sensors 2018, 18(9), 2820. DOI:10.3390/s18092820

[17] Thomas C, editor. Sensor Fusion. Foundation and Applications. IntechOpen 2011. DOI: 10.5772/680

[18] MATLAB. Sensor Fusion. Tracking Toolbox, 2020. Available from https://www.mathworks.com/products/sensor-fusion-and-tracking.html [Accessed: 2020-12-02]