UNIVERSITY OF TRENTO

Master of Science in Mechatronics Engineering

Gait recognition basing on IMU sensor and machine learning methods

Manual

Professor:
De Cecco Mariolino

Students:

Albi Matteo	232277
Cardone Andrea	224836
Oselin Pierfrancesco	229564

Department of Industrial Engineering
Academic Year 2021 - 2022

Contents

1 LPMSB2 communication library

1.1 Info

Based on the library: lpms.m
Author: H.E.Yap
Date: 2016/07/19
Revision: 0.1
Copyright: LP-Research Inc. 2016

FILE: include/LPMScommunicationBT/LpmsBT.m
FULL DOCUMENTATION: LPMS Operator's Manual
IMPORTANT: To enable the connection the bluetooth must be activated on the computer. You must pair the device with your computer first before connecting to it: the LPMSB2 device should be shown as "Associated" in your bluetooth device list (Windows 11).

1.2 General

1.2.1 Communication mode

The sensor has two communication modes:

- COMMAND MODE: send a data packet only on request
- STREAMING MODE: send data packets at a defined streaming frequency

1.2.2 Read process

To read an answer an interrupt-like approach is used. To enable an interrupt the function `configureCallback` is used. To set the interrupt is passed the number of bytes the program must receive to trigger the interrupt and the function to call when the interrupt is triggered. The function called is `readCallbackFcn`: it reads all the data available on the input buffer and parses it by calling the `parse` function. The parse function reads each byte until a full packet is read, extracting all the needed info like packet function, data length and raw data, saving them in object variables. Then the `parseFunction` method is called, checking the packet function code (aka command identifier) and executing the right instructions according to it (parsing data in case of get instruction, check ACK in case of set instruction).
Both sent and received packets have a command identifier. When using a get function, tx and rx packets have the same identifier, when using a set function the tx packet code defines the set action, instead the device answers with an ACK or NACK packet code, depending on whether the set action succeeds or not. Command identifiers are listed in rows 72 to 83 (not all commands are implemented).
After all data in the input buffer has been read, the program proceeds with normal execution. Data is stored in a dedicated struct, saved in a queue (object attribute), of max capacity determined by the constant attribute `DATA_QUEUE_SIZE` (row 140).

NOTE: in case of streaming mode, the interrupt is triggered every time a packet is received.

1.2.3 Data mode

Sensors' data can be delivered in two different formats:

- 32bit mode: using four bytes for each value;
- 16bit mode: using two bytes for each value (data parsing not implemented for this mode, see full documentation of the sensor for more details).

1.3 Methods

- `bool connect()`
  Establishes a connection with the LPMSB2 device and retrieves the actual configuration to update the object variables, calling the method `bool getConfig()`.
  This step is crucial because the device doesn't transmit all available data, but only the enabled one (printed to the command window after a successful connection). The number of enabled sensors' data affects the number of bytes for which the interrupt is triggered and the parse data function.
  The device is set in command mode by default calling the method `setCommandMode()`; the set streaming frequency is printed to the command window as well.
  By default it tries to connect to a device containing the name `DEV_NAME` (constant attribute set at row 54). In case the program should connect to a specific LPMSB2 sensor, specify the partial/full name (it can be shown using the `bluetoothlist` command).
  PARAMS: none.
  RETURN: true if connection is successful, false otherwise.

- `bool disconnect()`
  Disconnect from the device and reset the object variable `isSensorConnected` (boolean attribute that tracks the connection status).
  PARAMS: none.
  RETURN: true if disconnection is successful, false otherwise (in case no device is connected).

- `bool setCommand Mode()`
  Set the sensor communication mode to command. It sends a packet with the proper command identifier and waits for the ACK. The procedure described in the General section is automatically executed switching the interrupt on. If successful, it switches the interrupt off.
  PARAMS: none.
  RETURN: true if set is successful, false otherwise.

- `bool setStreamingMode()`
  Set the sensor communication mode to streaming, same approach used in the `setCommandMode()` method. If successful, set the interrupt with the right amount of bytes (based on the number of enabled sensors' data).
  PARAMS: none.
  RETURN: true if set is successful, false otherwise.

- `bool getConfig()`
  Get the configuration of the device, where it is listed: streaming frequency, enabled sensors' data, data mode. It sends a packet with the proper command

identifier and waits for the answer of the device (having the same function code). These data will also be printed to the command window. All received data will be saved in internal object attributes.
PARAMS: none.
RETURN: false if no device is connected.

- `void dispConfig()`
  Displays actual config looking at the internal object attributes. In case no device is connected, nothing is printed.
  PARAMS: none.
  RETURN: none.

- `bool setTransmitData(boolVector)`
  Sets the enabled transmit data, building a packet with the correct command identifier and a four bytes data field, where the 32 bits available are used as flags to define which sensor's data are enabled. The correct bits are set in the function based on the input parameter.
  NB: If the passed vector is empty, default value `DEFAULT_TRANSMIT_DATA` is used, defined at row 88.
  PARAMS:

  – boolVector: defines, using flags, which sensors' data are enabled, using the following logic:

Data	Vector position	bit position
Barometer	1	9
Angular velocity	7	16
Magnetometer	2	10
Euler angles	8	17
Accelerometer	3	11
Quaternions	9	18
Gyroscope	4	12
Altimeter	10	19
Thermometer	5	13
Linear accelerations	11	21
Heave motion	6	14

Table 1: Relation between flag index, enabled sensor and bit position in the `setTransmitData` function

  RETURN: true if set is successful, false otherwise.

- `bool setStreamFreq(freq)`
  Sets the streaming frequency, building a packet with the correct command identifier and a four bytes data field, where the frequency is specified. The input parameter is firstly checked if it is acceptable (only particular frequency values are admitted) and then converted into a four bytes number to ensure proper transmission.
  PARAMS:

  – `freq`: defines streaming frequency. Accepted values are: 5, 10, 30, 50, 100, 200, 300, 500 [Hz].

RETURN: true if set is successful, false otherwise.

- `sensorData_struct getCurrentSensorData()`
  Retrieves the newest data from the sensor. In case the device is in streaming mode, return the struct where last data read is saved; in case the device is in command mode, send a `getData` request packet to it and return the struct where data is saved.
  PARAMS: none
  RETURN: struct with last sensors' data if the command is successful, empty struct otherwise

- `sensorData_struct getQueueSensorData()` Pop and return the oldest data stored in the queue.
  PARAMS: none.
  RETURN: struct with last sensors' data if the queue isn't empty, empty struct otherwise.

2 Detect Phases function

This function is used to label the acquired data in offline mode.

2.1 Info

FILE: include/detectPhases_3.m
FUNCTION: `table detectPhases_3(IMU_data)`
PARAMS:

- `IMU_data`: table containing data registered with an IMU sensor (obtained using `readtable` MATLAB function on a csv file).

RETURN: the same table given as param with one more column corrisponding to the label

2.2 Threshold Algorithm

1. Filter with a low-pass filter to find only one maximum peak per period, the cutoff frequency must be not too small to preserve a good signal.
2. Filter with a low-pass filter with lower cutoff frequency to find good MSt.
3. Calculate an automatic threshold: weighted average between the positive part of the signal and its maximum and assign threshold_2.
4. Create a matrix to store the conditioned signals and phases.
5. Start for loop on all the signal.
6. Find maximum peaks indicating (MSw).
7. Once found MSw backward for loop to find the nearest minimum (TO).
8. Once found TO continue backward for loop to find the nearest maximum (MSt).

9. Once found MSt go on with forward loop from MSw to find the nearest minimum (IC).
10. Repeat from point 6.

A label is assigned to every sample of the signal, the phases are:

1. from IC to MSt
2. from MSt to TO
3. from TO to MSw
4. from MSw to IC

2.3 Parameters Tuning

1. The data must be related only to the walking part, if necessary cut the signal before the `detect_phases` function (probable problems with the automatic threshold).
2. Be sure that the GyroZ_deg_s_ signal is related to the rotation of the leg on the sagittal plane (side view), the signal must qualitatively have the following aspect:

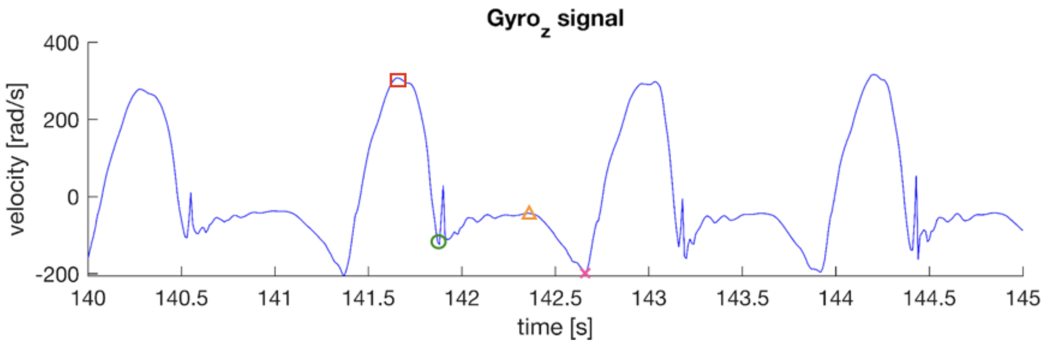

Figure 1: Signal of the gyroscope with highlighted the four notable points

3. Choose `f_1` (cutoff relative frequency) such that there is (possibly) only one peak detected for MSw point, IC point must be detectable (green circle previous image). The resulting signal must look like the red signal in the following image. The aim is to obtain a good signal for the following processing eliminating noise. `f_1` value should be around 0.3.

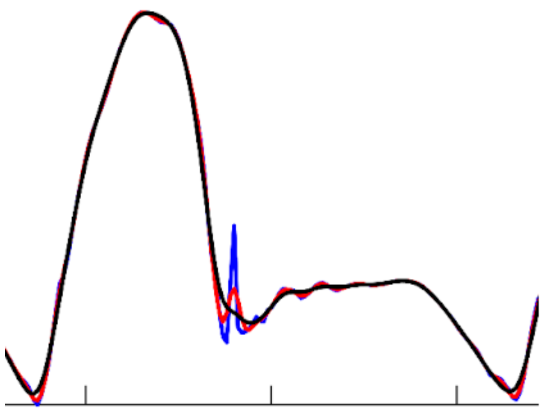

Figure 2: Detail of the three signals for one gait cycle

4. Choose `f_2` such that the signal is smooth enough for detection of MSt event (orange triangle in first image of `detect_phases`). The signal must not have ripples in the high part going backwards from the TO event (green rectangle). The aim is to obtain only one peak while preserving its position; the resulting signal must look like the black signal in the following image. There is no problem if there are ripples in the part near the IC point (start of the arrow). The algorithm uses by default `f_2` = 0.15.

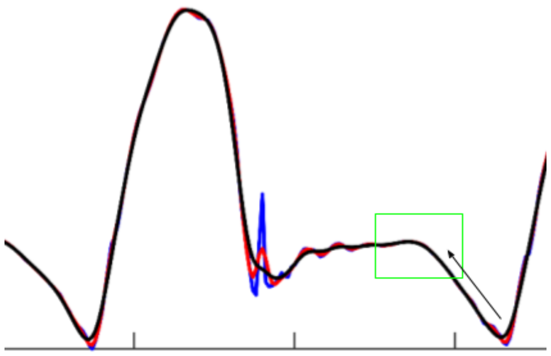

Figure 3: Signals with highlighted the location of MSt point and the direction of search

5. Choose `k` such that under there is a good weighting between the maximum value of the signal and the average of the positive part of the signal. Increase `k` results in an increase of the automatic threshold.

$$threshold_{auto} = \frac{1}{2}(\max(gyro_z)k + mean(gyro_z > 0)) \qquad (1)$$

   With the right `k` the threshold is high enough to avoid unwanted high peaks for MSw detection, like the positive peak after IC (green circle in the first image). In the following image the purple line represents the resulting threshold. The `k` parameter is set equal to 0.3 by default.

6. Choose `threshold_2` such that under this threshold there are the ripples near the IC point. This is a safety threshold for a correct detection of MSt point. Sometimes near IC point there are big ripples that cannot be filtered out with

`f_2`, so we use this threshold. Be sure that `threshold_2` isn't above the MSt point. In the following image is reported an example (`threshold_2` is the green one). The algorithm uses by default `threshold_2` = -60.

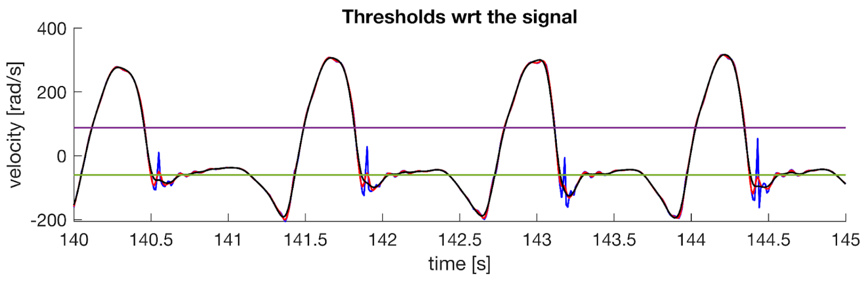

Figure 4: Signals with the two threshold

7. Be sure that the algorithm correctly detects the gait phases. Otherwise tune the parameters as illustrated before. The phases must present a step-like shape as shown in the following image (in the image the label of the gait phases is multiplied by 100 and lowered by 200 to be comparable with the gyro signal, 100*phase - 200).

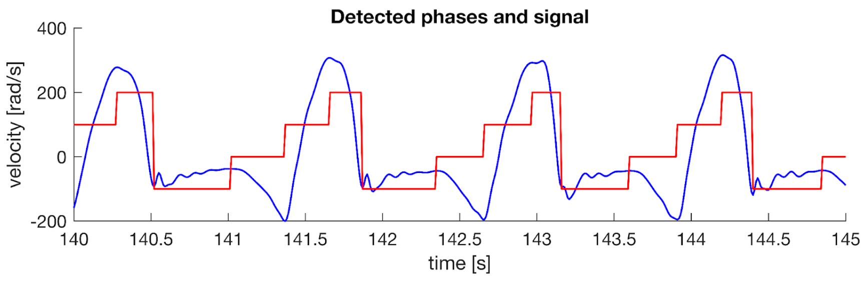

Figure 5: Signal with the detected phases

8. These parameters should not be modified for every set of data, the algorithm is made to work well with the differences of gait between different people walking at different speeds. Nonetheless you must be sure that the labeling algorithm works fine if you want to create a new set of labeled data. This is true in particular if the new conditions are quite well distant from healthy people who walk with a speed between 4 and 6 km/h.

3 Simulate stream function

This function simulates a real-time datastream of incoming data and its classification.

3.1 Info

FILE: include/simulateStream.m
FUNCTION: `vector simulateStream(network, testDataset, reset_label, graphicsEnabled)`
PARAMS:

- `network`: properly trained network by which classifying data.

- `testDataset`: a dataset on which simulating the stream flux. The preprocessed dataset will be classified with the `detectPhases_3` in order to further estimate the accuracy.
- `reset_label`: boolean variable; if it's false it updates the very last new incoming data with its label, otherwise the whole batch (k+1,n+1) will be classified once again.
- `graphicsEnabled`: boolean variable; if it's true, a graphical representation of the flux simulation is provided.

RETURN: stream accuracy, computed as ratio between correct labeled timestamps on total timestamps.

3.2 Functioning

This classification process is different from a normal prediction because only a partial time frame is passed to the network.
In this function the default batch of data on which applying the classification at each iteration is 350. Since the sampling frequency is 100 Hz, this corresponds to a latency of 3.5 seconds for setup (in order to fill the vector). The batch size should be properly chosen according to the number of cell units of the RNN and to the number of past timeframes we want to evaluate. This affects the accuracy of the evaluation and also its speed. In this case the network has been trained with 50 units, so 350 provides an high accuracy.
For optimal performances it would be better to simulate different batch sizes in order to find the best trade off.
In figure 6 it's shown a portion of the labeled stream plotting.

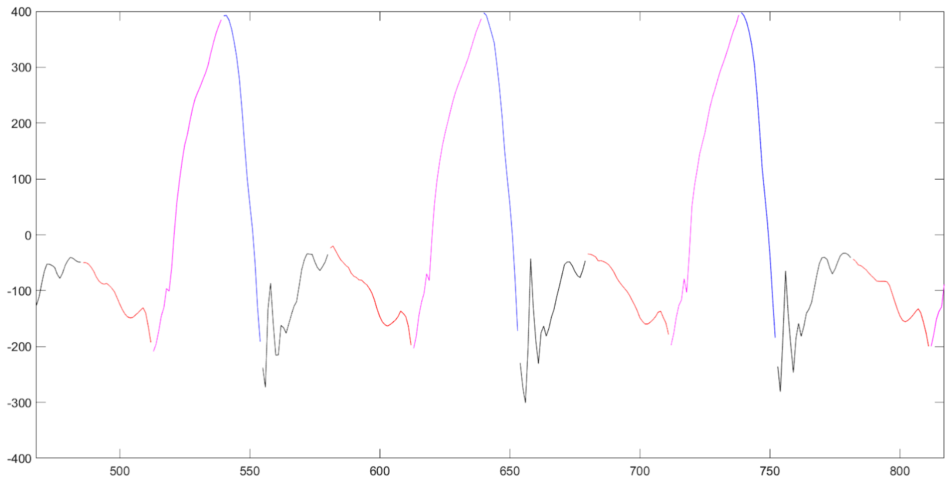

Figure 6: Example of real-time classification. Z-axis signal is plotted. Data belongning to different classes are displayed with different colours.

4 Labeled plot function

Plot data belonging to different classes with different colors

4.1 Info

FILE: include/plotLabeledData.m
FUNCTION: `void plotLabeledData(data, time)`
PARAMS:

- `data`: vector of data to be displayed. In a Cartesian space, it corresponds to the Y-value vector.
- `time`: vector of data to be used as x values in the Cartesian plot. In a time-plot, it corresponds to the time.

4.2 Functioning

Goal of this function is to plot a set of data, providing the x-axis interval, which could be for example the time frame.
In particular, this function allows to plot data belonging to different classes with different colors. The maximum number of different classes, and therefore colors, is 5. If you want a higher span, you just have to add color values to the colors array. If the number of classes is higher than 30, the function will return an error because for common projects, classification is performed among a small number of classes. It is basically a warning workaround to underline data could not have been labeled yet.
By providing the x-axis span, you can iteratively recall the function in order to produce an animated plot. This has been exploited to produce the graphical representation in the simulateStream function.

5 Pre-process data for clustering function

This function properly pre-processes data before feeding them into an unsupervised learning clustering.

5.1 Info

FILE: include/dataPreprocessingUnsupervised.m
FUNCTION: `void dataPreprocessingUnsupervised(varargin)`
PARAMS: varargin

1. `dataset`
   The very first parameter is the set of data, which must be in the following form dataset={file_log1,..., file_logN
2. `'features'`
   Optional parameter to be inserted if feature mapping has to be applied
3. `minibatch_size`
   Optional parameter to be inserted if feature mapping has to be applied. In particular, it is an integer and corresponds to the mini batch size on which applying the mapping.

5.2 Functioning

The function is pretty similar to dataPreprocessing, however at the end the table is converted to a matrix and, if requested, feature mapping is applied. The list of the features can be found in the report or directly into the code.
NOTE: in order to properly apply feature mapping, data are sorted by classes for avoiding spur images.

6 Unsupervised learning script

This script runs unsupervised learning classification methods, in particular k-Means algorithm. It trains the system and evaluates its performances.

6.1 Info

FILE: unsupervisedLearning.m

6.2 Functioning

First of all, data are imported into the workspace and they are split into training and test set just after.
Two arrays for storing the accuracy values are initialized, namely `acc_train` and `acc_test`.
A for cycle is introduced, in order to execute the classification 2 times: in the first one, data are preprocessed, dropping useless information and classified with the `detectPhases_3` algorithm in order to provide further estimation for the accuracy.
In the second one, data are preprocessed with the feature mapping. In particular, the size for the mini batch has been set to 150. It's up to you choosing that value.
Regarding the k-Means algorithm, the number of clusters has been set obviously to 4 and the number of epochs to 60.
Eventually, a visual representation of the classification is displayed as well as the estimated accuracy.

7 Pre-process data for RNN function

This function properly pre-processes data before feeding them into an RNN.

7.1 Info

FILE: include/dataPreprocessing.m
FUNCTION: `[X,Y] dataPreprocessing(dataset, useful_data)`
PARAMS: varargin

- `dataset`: List of tables containing the data to process, each table is a different file. The columns must have the same name for every table.
- `useful_data`: Vector of string listing the name of the columns the program must not delete from the table, used to keep only useful information.
  ATTENTION: can't be empty otherwise the returned matrices will be empty as well.

RETURN:

- `X`: cell array of matrices, each matrix containing the input features for the RNN in the correct shape.
- `Y`: cell array of vectors, each vector containing the timestamp associated label for the RNN in the correct shape.

7.2 Functioning

Collected data are imported from files in a table data format. However an RNN needs the input to be of a different shape. More specifically, every timestamp must occupy a different column, while in the rows must list the input features. For the expected result, the same shape is needed, with a single row listing the expected label for each timestamp.
EXAMPLE:

```
file1 = readtable('filename1');
file2 = readtable('filename2');
tables= {file1, file2};
useful_data = ['column_1_name', 'column_4_name'];
[X,Y] = dataPreprocessing(tables ,useful_data);
```

8 RNN script

This script runs deep learning classification methods, in particular an RNN. It trains the system and evaluates its performances.

8.1 Info

FILE: RNN.m
PARAMS: The parameters used for the network definition are listed from row 63 to 79, the training options are defined from row 82 to 94.

8.2 Functioning

This script loads all the dataset and trains a single network with parameters defined in the script itself. Then compute the three KPIs used to evaluate a network: test accuracy, phase accuracy and stream accuracy, printing them to the command window.
This script it's been used mainly in the development process of all the different functions and for networks first evaluations. As the last step, when evaluating the stream accuracy, the `simulateStream` function is also used to display the simulation of a real time classification.

9 trainMultipleRNN script

9.1 Info

FILE: trainMultipleRNN.m
PARAMS (at rows 72 to 74; don't change the net type definition):

- `Net type`: it can be GRU or LSTM
- `NHiddenLayers`: number of hidden layers of the net
- `MaxEpochs`: maximum number of epochs of the training process
- `GradientThreshold`: gradient threshold used in training

9.2 Functioning

The structure that contains a trained network saves:

- All the parameters of that network (type, number of hidden layers, number of epochs and gradient threshold)
- The train net (as result of the training function of MATLAB)
- The three KPIs to evaluate it: phase, test and streaming accuracy

Then, using a cascade of for-cycles, the creation, training and evaluation of the different networks occurs, and at the end the results are saved.
ATTENTION: the save function might fail (the error will be shown at the end of the training process). Please check it and in case of error manually save the variable `results`.

10 Results analysis script

Using the result of the script trainMultipleRNN, it performs a simple automated analysis, finding the nets with best KPIs, one net for each, and plotting how these KPI behave when varying a net or training parameter.

10.1 Info

FILE: resultsAnalysis.m

10.2 Functioning

The overall analysis consists in calculating the mean test and phase accuracy for each net and finding the net with the best performance, one net for each KPI.
Then, when analyzing a parameter, for example the number of epochs, calculates the mean KPIs for all nets having a certain value (e.g. when evaluating the 150 epochs case, calculate the mean test, phase and stream accuracy of all networks having that number of training epochs). At last it plots the results.
This analysis was conducted in order to exploit trends in the net performances when varying a specific parameter. In the last rows (after row 191) the results are printed to the command window.