Executive Summary of the Thesis

Development and enhancement of an IMU-based system for real-time monitoring of astronauts training

Laurea Magistrale in Biomedical Engineering - Biotechnologies for Electronics (BTE)

Author: Federica Camarda

Advisor: Prof. Giancarlo Ferrigno

Co-advisor: Francesco Jamal Sheiban

Academic year: 2021-2022

1. Introduction

This thesis is part of a research project funded by the Italian Space Agency (ASI) called MARS-PRE (Biological and functional MARkers for PREcision astronomical bio-medicine), aimed at investigating the effects of microgravity (i.e, the absence of gravity) on the human body through the analysis of biological markers.

As worldwide space agencies are now planning long-duration missions (LDM) to the Moon and Mars, it became fundamental to thoroughly investigate the physiological effects that the microgravity, the space radiations exposure and the acceleration forces can cause to the human body for the safety of the astronauts involved. Indeed, it has been observed that the aforementioned factors lead to a deconditioning of multiple physiological systems (such as musculoskeletal, cardiovascular, respiratory and visual), which might actually not fully recover once back to the Earth. Furthermore, long missions can increase the risk of carcinogenesis or/and degenerative diseases, due to the link between the effects of the harsh external stimuli exposure and the individual genomic map.

Regarding the musculoskeletal system in particular, it has been observed that microgravity induces a decrease in bone density 1% per month approximately due to mineral loss, a value close to the one of elderly men and women on Earth, suggesting a possible risk of the body to be unable to properly repair a fracture during space flights [3]. Further, it has also been shown that the absence of gravity favors muscle atrophy, with reported lower limb muscles mass decreases of 35-40% of their intial value after a period of 90-180 days [1]. Therefore, the current project deals with the need to design and develop proper countermeasures against these physiological changes to allow astronauts to carry out future LDMs without permanent damage to their organism, as well as to prevent the emergence of critical situations in a context lacking appropriate facilities for medical support.

2. State of the Art and aims

In order to prevent the deconditioning of the musculoskeletal systems, precise countermeasures (deriving from a combination of physical exercises, diet and sometimes pharmacological therapy, such as the use of bisphosphonates) have been designed to be adopted not only during mission time, but also before and after the flight.

Astronauts aboard the ISS devote 2.5 hours 6 days a week to physical exercise, performing both aerobic and resistive trainings.

Specific devices have been planned for inflight training programs for astronauts, such as the Advance Resistive Exercise Device (ARED), the T2 treadmill and the Vibration Isolation and Stabilization Cycle Ergometer (CEVIS). The installation of the ARED on the ISS only took place in 2008; the device is designed to simulate free weight exercise providing constant resistance to movements through a bar coupled to two vacuum cylinders. When a subject lifts the bar on its shoulders, the vacuum inside the cylinders opposes to the movement of the piston inside them, thus creating concentric resistance. Conversely, when the bar is lowered, the resistance to the vacuum creates eccentric resistance [4]. The ARED features an adjustable load from 0 to over 2,675 N, providing concentric resistance up to 272 kg and constant force over the entire range of motion. Thanks to these features, ARED allows to reduce loss of strength in the thigh muscles, going from a range of -9 / -20% (pre-ARED) to a range of -4 / -15% ([6]) and an improvements in the BMD values of the pelvis and hip.

Many astronauts, however, continue to lose more than 20% of muscle strength due indistinctive countermeasures [5]. Constant monitoring of exercise execution is also required to let astronauts train correctly and effectively. However, due to the long distance from the Earth, a mission to Mars potentially entails transmission delays from 6 to over 40 minutes [2], thus being prohibitive for real-time telemonitoring. Hence the aim of this research project was to develop and enhance an IMU-based system (SpaceSens), based on a previous work, to provide a real-time feedback to users about the goodness of the physical exercises being executed. In particular:

- an application was developed in order to acquire, process and visualize the human motion signals recorded through six inertial sensors, collecting data of linear acceleration, angular velocity and magnetic field (only for test purposes[1]) along the three axes;
- a customized machine learning classifier was implemented to classify resistive exercise carried out by different subjects.

[1]Data acquired from magnetometers were not included in the later implementation of the classifier, because there is no need to simulate the microgravity condition.

3. Materials and Methods

Although the hardware and the firmware parts of the setup are beyond the scope of the current project, an overview of the principal components and their internal characteristics is deemed necessary.

The motion tracking system had the requirements to be small, light, with low energy consumption, so its hardware has been implemented using a magnetic-inertial sensor, a microcontroller, a Bluetooth communication system and an integrated power supply. The magnetic-inertial sensor used is the SparkFun MPU-9250 IMU Breakout, which comprises InvenSense latest 9-axis MEMS sensor, featuring an MPU-6500 chip (containing a 3-axis accelerometer and gyroscope) and an AK8963 chip (that instead contains a 3-axis magnetometer). The microcontroller used is an Arduino Pro Mini 328 - 3.3V/8MHz, due to its very small footprint that fits well the needs and requests of the project. Regarding the Bluetooth module, HC-05 SPP (Serial Port Protocol) module was chosen, not only because of its compact size, but also to allow a greater operating distance range compared to other modules and provide continuous data transmission. Finally, a 750mAh 3.7V 30C Li-Po battery (i.e., a lithium polymer battery) powers all the components aforementioned.

The SpaceSens firmware, instead, is composed by the set of instructions loaded into the microcontroller in order to read data from the sensors and send it to a processing unit (i.e., a computer) via Bluetooth. It was implemented using the Arduino IDE (Integrated Design Environment) software and written in C and C ++ languages. The firmware contains serial communication and sensors calibration functions. The serial communication is responsible for sending the data to a Python-based graphical user interface (GUI) allowing to display, store and elaborate the quantities read by the different MPU-9250 sensors; the data streamed through these firmware instructions are composed both by a percentage number (representing the sensors' battery level) and the measured quantities, encoded in packets of 3 and 28 bytes respec-

tively.
The accelerometer and gyroscope calibration, instead, is carried out entirely on the firmware side when turning on the sensors and placing them on a plane, with the z-axis facing upwards. During the initial calibration phase, the firmware records the sensors' biases (comparing the quantities read in this configuration to their ideal, fixed values) and stores the offsets to be removed before sending the data through the serial channel during the data streaming phase.

The main contribution of the present work regards the application implementing the GUI to let the user control the sensors, followed by a phase of data acquisition and processing aimed at creating the classifier that will be inserted into the GUI for real-time feedback to astronauts.

GUI implementation Concerning the software, two architectural patterns were exploited to facilitate readability, streamline and ease of code maintenance: the *Model-View-Controller* (MVC) and the *Publisher-Subscriber* pattern.

The first pattern is deeply used to implement graphical applications, and its main role is to allow developers to separate the program's logic from the implementation of the actual interface. The MVC consists of three software classes: a Model, a View and a Controller class - hence, its name.

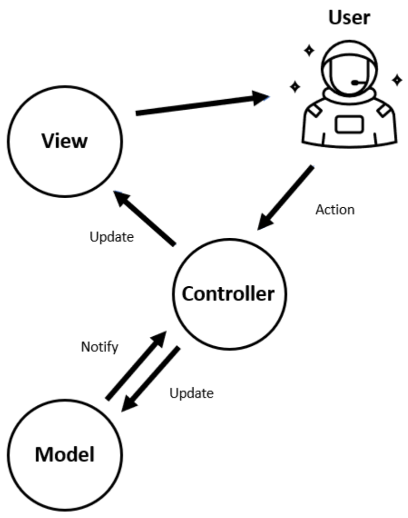


Figure 1: *Representation of the Model-View-Controller Design Pattern.*

In this project's specific implementation, the Model class contains all the streaming data sent from the microcontroller to the serial port of the computer, the View contains all the graphic components of the interface to be displayed on the computer's screen, and the Controller collects user inputs from the View and communicates them to the Model, which in turn updates the data visualized by the View itself.

The second architectural pattern, the *Publisher-Subscriber*, is a widely used messaging pattern in which a class (Publisher) is responsible to update messages in a common channel from which specific classes (Subscribers) pick up notifications about the topic they are subscribed to. As for SpaceSens implementation, each serial port involved in the firmware communication doubles as a Publisher that notifies its Subscriber whenever a new packet of data is received from the sensor's microcontroller. Thus, the Subscriber class stores a data structure of the packets received (and continuously updated) by the Publisher, and the data contained in this structure can be either visualized or saved in a file for offline analyses.

To favor code readability, the implementation has been divided into four Python scripts linked to each other. One of them manages the creation of the application's main window, one is responsible for post-processing data acquired during the exercise execution, one stores all the instruction related to the graphical interface (implementing the MVC pattern) and finally, the last one is responsible for the serial communication (based on the Publisher-Subscriber pattern).

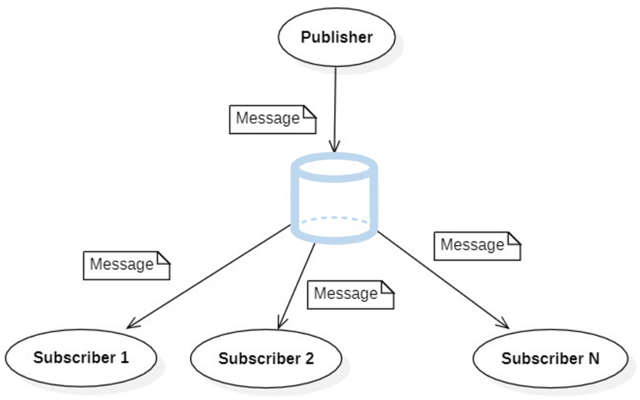


Figure 2: *Publisher-Subscriber Design Pattern scheme.*

As the software was required to read and store data from six different IMUs simultaneously, its implementation entailed the use of the multiprocessing Python module to allow different instances of the program (running sequentially) to be executed in parallel. Python processes are functions dispatched from within a script to be executed in separate set of activities: as processes do not have a shared memory, their use within the SpaceSens application required the implementation of a process Manager and Proxy class, in order to allow the interaction with and data retrieval from the different processes memory.

An important step is then the magnetometer calibration, which allows on the user request through the GUI to calculate the bias and scale values caused by distortions of the involved sensors in order to send them to the microcontrollers to be used for the gravity removal from the acceleration signal.

The user interface has been designed in order to be easy-to-use, so every element is shown in one main page with a clear layout that decreases the learning curve of the program; the user-friendliness is also enforced with the implementation of pop-up windows that communicate the outcome of user interactions involving elements outside of the application (i.e., file saving), that would otherwise be hidden.

Classifier The aforementioned application was tested in a real-life scenario and used to collect data from different subject to develop a custom machine learning classifier to be then embedded into the new application.

The data were acquired in early 2022 at the Giuriati Sport Center (Politecnico di Milano) from 9 different subjects (4 females and 5 males), with distinct weights and heights. Each subject performed 10 repetitions for the three types of exercises (normal squat, wide squat and deadlift), both correctly and with the most frequent types of errors: 5 for squats and wide squats (knee over toes, valgus knees, rounded back, raised heels, shallow squat) and 3 for the deadlift (rounded back, hyperextended back, bar over the shoulders). The 6 IMU sensors were placed in the mid-legs, mid-thighs, sternum and pelvis. The data collected were then processed to extract the gravity-free acceleration and angular velocity signals. These quantities were subsequently filtered with a low-pass Butterworth filter and divided into individual repetitions of the exercises, by means of a search algorithm of peaks and valleys of the recorded signals. Then, a total of 2250 characteristics (6 sensors x 375 characteristics for each sensor) were calculated from the signals in both time and frequency domains (such as mean, standard deviation, interquartile range, entropy). After a data augmentation process through the addition of Gaussian noise with variable mean and variance, with subsequent outliers removal, the most representative features were extracted and used to train and test a multi-layer perceptron (MLP) for multi-class classification.

4. Results and Conclusions

The program described in the previous section successfully allowed to collect data from the different subject throughout the data acquisition session, lasting a full working day. As its software components have been designed following the object-oriented programming (OOP) paradigm, the Python-based application shows significant improvements in terms of both timing and architecture compared to the software application implemented within the previous work, designed as a state machine.

The main visible advantages are the swiftness of the sensors connection, the reduction of the code base dimension and improved readability and scalability of the code thanks to the two architectural patterns used. Moreover, the use of pop-up windows and many different widgets greatly improved the user experience (UX), promoting the interaction and interest of the users towards the application.

Swiftness of sensors connection Using the previous application, whenever the program was unable to establish an immediate connection between an IMU-based device and its computer virtual port, the user had to wait the end of the sensors search loop over the remaining ports (as registered by the computer's operative system) for the application to start over and try to reconnect to missing sensors.

The application developed in this project avoids such problem, as it allows a targeted recognition of the ports. Each available device is visible in a list within the main page with its name, so to be connected on demand only when necessary for data acquisition. The color of the label displaying the names in the devices list is set to gray by default. When the pairing between the computer and the device is successful, the color of its text label turns white, notifying the user that the device is ready to be used. Additionally, the presence of real time plots in the right part of the window allows a quick peek on the state of the connection: when the plots related to the connected sensor are empty, the serial connection has most probably encountered some hick-ups and the user can press the button again to connect that particular device and possibly close the faulty plot tab using a specific button.

Use of architectural patterns As stated before, another great advantage of the new software implementation is the use of the architectural patterns. While in this work a custom *Publisher-Subscriber* parallel pattern allows to continuously stream data from the different microcontrollers to the PC, the previous application made use of Python multiprocessing's module built-in *Pipes* classes. As the name suggest, a *Pipe* is a process that allows bidirectional (read/write) flow of data. However, due to its high-level implementation, it required a fixed syntax so that every time there was the need to visualize data (i.e., draw real-time plots) or perform actions requiring access to the measured data (such as the magnetometer calibration), one of the two ends of a new pipe had to be defined and activated, resulting in a burdening of the computational power required by the application.

Classifier Concerning the development of the classifier, the Multi-Layer Perceptron was able to classify squat executions into six error classes, with an accuracy of 97.6% on the training and 98.6% on the test data.

best estimator
'hidden_layer_size': (650, 40)
'activation': 'relu'
'solver': 'adam'
'alpha': 1e-07
'momentum': 0.8
'learning_rate': 'constant'

Table 1: *Table reporting the hyperparameters of the best estimator.*

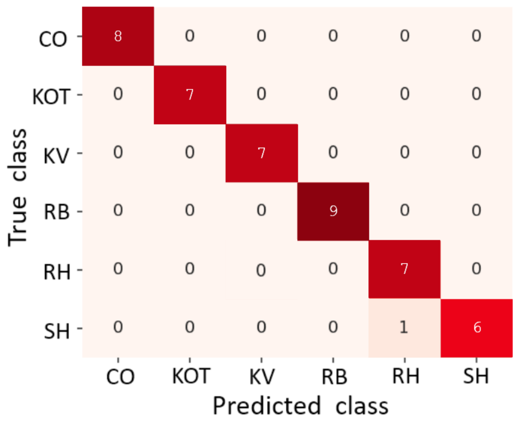
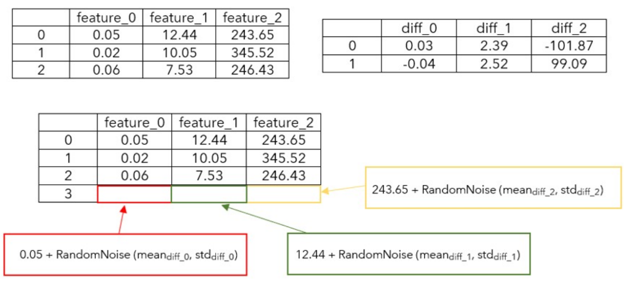


Figure 3: *Confusion matrix of observations included in the test set of one specific subject.*

This result was obtained through a data augmentation process which was suggested by the limited size of the dataset, consisting of only 60 observations (derived from the 10 repetitions performed for each type of squat). Indeed, the classifier used on the starting dataset performed a stable accuracy of 100% both on the train and on the test sets, an unrealistic result. For example, if we consider that the positioning of the IMUs in a subsequent training session could vary (even slightly), the prediction on the correctness of the squat could be wrong. Data augmentation helps reduce overfitting when training a model, making it more robust to the transformations.
The data augmentation was achieved through a cloning of the starting dataset contaminated by Gaussian noise with variable mean and variance. In this context, different mean and variance values were examined, in order to give the right variability to the data and find the right trade-off between overfitting and underfitting of the model.
The test accuracy value of 98.6% is the result

of a data augmentation with two different types of Gaussian noise with mean 0.5 and -0.5 and higher standard deviations of 10 (arbitrary measuring units). The reason for using two noises with non-zero averages is due precisely to the possible variations that a new acquisition session could cause. It was preferred to allow some variability to the data which, however, did not affect the average of the starting data. Although theoretically the value of the variance may seem high when compared to the actual values of the different time series, it empirically proved to prevent overfitting. This could be at least partially explained by a low probability of extracting values close to the tail of the bell (and therefore too high).
Nonetheless, different strategies were also investigated to perform data augmentation and subsequent MLP training. Among these, the most promising approach involved computing the pairwise difference between each row of the original dataset resulting in a new dataset with 59 rows. For each column of this new dataset, mean and standard deviation were subsequently extracted and used as the coefficients of the noise generation function (see Figure 4). Despite it showed lower accuracy values than the ones provided by the previous solution (ranging from 88 to 90%), it can still be considered a valid alternative.



	feature_0	feature_1	feature_2
0	0.05	12.44	243.65
1	0.02	10.05	345.52
2	0.06	7.53	246.43

	diff_0	diff_1	diff_2
0	0.03	2.39	-101.87
1	-0.04	2.52	99.09

	feature_0	feature_1	feature_2
0	0.05	12.44	243.65
1	0.02	10.05	345.52
2	0.06	7.53	246.43
3			

243.65 + RandomNoise ($mean_{diff\_2}$, std_{diff_2})
0.05 + RandomNoise ($mean_{diff\_0}$, std_{diff_0})
12.44 + RandomNoise ($mean_{diff\_1}$, std_{diff_1})

Figure 4: *Methodology explanation concerning the noise addition related to the mean and variance of each feature of the new dataset given by the pairwise difference between each row of the original one.*

A removal of the outliers and the subdivision into train and test set were then performed.

Finally, although the present work greatly improved the usability of the SpaceSens system, there are still some improvements that could be done: first, an emergency shutdown mechanism would help to prevent the risk of battery damages when almost empty (as typical of lithium polymer batteries).
Then, a custom classifier for both wide-stance squat and deadlift could be implemented allowing the application to monitor the execution of the exercises more precisely and thus allowing the astronauts to acquire data for the algorithm directly in-flight or in the pre-flight phase on the ISS.

References

[1] Gian C. Demontis, Marco M. Germani, Enrico G. Caiani, Ivana Barravecchia, Claudio Passino, and Debora Angeloni. Human pathophysiological adaptations to the space environment. *Frontiers in Physiology*, 8(AUG):1–17, 2017.

[2] Tamas Haidegger and Zoltan Benyo. Surgical robotic support for long duration space missions. *Acta Astronautica*, 63(7-10):996–1005, 2008.

[3] Laurie J. Abadie, Charles W. Lloyd, Mark J. Shelhamer, NASA Human Research Program. Gravity, Who Needs It? NASA Studies Your Body in Space. pages 1–4, 2015.

[4] James A. Loehr, Stuart M.C. Lee, Kirk L. English, Jean Sibonga, Scott M. Smith, Barry A. Spiering, and R. Donald Hagan. Musculoskeletal adaptations to training with the advanced resistive exercise device. *Medicine and Science in Sports and Exercise*, 43(1):146–156, 2011.

[5] Maria Stokes, Tobias Weber, European Space Agency, Nick Caplan, and Lieven Danneels. ' Post - mission Exercise (Reconditioning)' Topical Team FINAL REPORT Recommendations for Future Post-mission Neuro-musculoskeletal Reconditioning Research and Practice. (June):1–96, 2016.

[6] Kunihiko Tanaka, Naoki Nishimura, and Yasuaki Kawai. Adaptation to microgravity, deconditioning, and countermeasures. *Journal of Physiological Sciences*, 67(2):271–281, 2017.