Università degli studi di Trento

Department of Industrial Engineering

Course of Robotic Perception and Action

Project report

EMG Controler

Students:

Dal Mas Massimiliano
De Martini Alessandro
Koszoru Kristóf

Professor:

Prof. De Cecco Mariolino
Dr. Covre Nicola

Academic year 2019/2020

Contents

Abstract

Exploiting an EMG sensored bracelet, it is possible to create a program of clinical rehabilitation addressed to arm muscles.
Using machine learning tools, it should be possible to perform evaluations on how a person performs gestures in specific cases for medical evaluation.

In this paper is presented a first research in the field in which a virtual model of helicopter is controlled using an EMG bracelet connected to a neural net for gesture classification, focusing on the net training and the connection between MATLAB® and Unity®.

1 Introduction

In the field of rehabilitation new technologies are always studied and a lot of research is spent. One field focuses on the possibility to track the muscle signals in real time, to perform analysis on the activation and find a proper rehabilitation program. The idea is to unify an EMG bracelet, which is composed by eight sensors able to capture the superficial signals of muscle activation with a game; so a potential patient is invited to perform specific gestures to command specific events in the software.

As a first step of the research, the concept is to create a set of predetermined hand-gestures and train a neural network model to classify them. In a second phase, exploiting the ZMQ interface, a connection between MATLAB® and Unity® is established to control a virtual model of the helicopter based on the classification output of the neural net.

The final use of this project consists of a helicopter drive simulation controlled by an EMG bracelet. The aim of the project is to build a real-time interface that rely on EMG data as input to control the virtual 3D model. The gesture recognition is performed using a Deep Neural Network. In the later section, a detailed explanation on how the data has been acquired and how the neural net has been implemented in MATLAB® will be given.
The Publisher-subscriber interface, which allows the communication between MATLAB and the virtual environment, is implemented exploiting a ZMQ interface, while the game is carried out using Unity®.

This project mainly focuses on formalising a reliable and natural relation between the user and the simulated object. It can have tangible applications in the real world, such as rehabilitation purposes, as a main research field, but also applications like drone or robot driving.

2 Machine learning basics

To be able to recognise an arm action from the signal provided by eight EMG sensors, a deep neural net is needed as classifier.

2.1 Deep Neural Network concepts

A neural net is an algorithm designed to behave similarly to a human brain. It is composed of units (perceptrons) which have weighted input connections, a net input function, an activation function, and an output connection. These are designed to be stacked in multiple layers. Each layer contains several neurons connected to the following series. The nodes the net is fed from are called *input nodes*, while the ones the outcome is returned are named as *output nodes*. The perceptrons inserted in the middle layers are, instead, *hidden nodes*.
The input values are numeric as the output ones, since the Neural Net is a math model.
Deep neural nets are distinguished by their complexity, as they usually have more than three hidden layers. Each layer computes the outputs based on the values provided by the previous one, and their training is based on the same characteristics. The more we go in depth, the more complex the characteristics a node can recognize, since it aggregates and recombines the features of the previous layer.

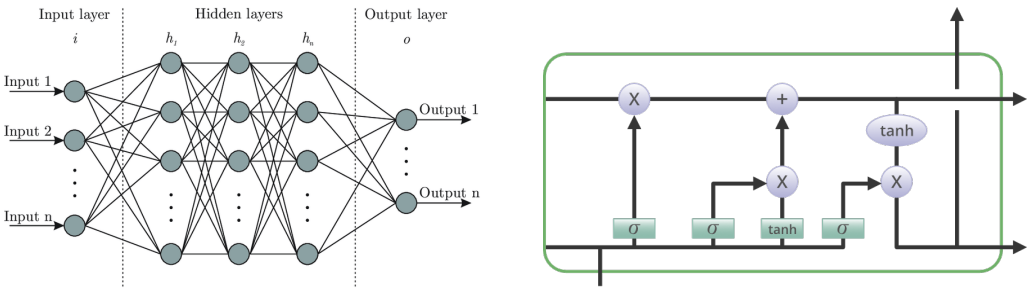
2.2 LSTM model

In some situations, a neural net can be useful to predict a certain kind of output based on the previous ones, especially with sequential data. A first approach is to use Recurrent Neural Nets (RNN) which are able, through a loop connection of nodes, to remember a past information for a short term of time. This system is limited when it comes across problems whose solution depends on a large amount of previous data, or when some information is more worth remembering than some else.
LSTM model, that is Long Short Term Memory, is able to face and solve these problems using cell states. A LSTM unit can selectively remember or forget things according to three dependencies:

- The previous cell state.
- The previous hidden state.
- The input at the current time step.

The LSTM cells have different components called input gate, forget gate and output gate. The input gate controls how a new input value flows in the cell; the forget gate regulates the way a value is remembered in the cell, while the output gate is responsible to control how the number stored in the cell is used to compute the output activation of the LSTM unit.

In this way the gates regulate the flow of information coming in and going out of the cell, and the latter is able to remember values for an arbitrary time interval.



(a) *Neural network layers.* (b) *LSTM cells components.*

Figure 1: Machine learning structure.

3 EMG sensor

To acquire muscles signals, a bracelet with EMG (electromyographic) sensors was used. It is composed by eight different sensors which cover the circumference of the arm, and each of them provides a distinct signal. The device is also able to localise itself and provide its orientation using the built-in three-dimensional accelerometer and gyroscope sensors.
The connection with the computer, and so with its software, happens via Bluetooth thanks to a USB pairing device. On the configuration panel of the Myo software, it is possible to set the EMG bracelet up, configure it, and power it off when no longer needed.

When it is worn, it is important to respect the orientation of the sensors, and to position the main one on the correct muscle, so that meaningful data can be acquired, and ensure that the experiment is reproducible.

Figure 2: Myo bracelet.

This device normally provides two types of signals, that are one for the eight sensors, and another for the accelerometers installed in the bracelet. In this specific application, only the first category of signals is used.

4 Matlab side

As previously seen, in paragraph 3, the Myo armband has different sensors. Only the signal derived from the electric activity of the muscle is processed to recognize the gestures acquired made in the laboratory and used to control the game.

The data acquisition and processing is done in MATLAB through the customization of three different scripts provided by the MiRo laboratory (University of Trento).

Each of them is designed to run a specific task of signal processing:

- Row data collection.
- Neural network training
- Real time action recognition.

The connection between the sensor and the MATLAB software is done using a C++ library called MyoMex [2], which permits to obtain access to the electrical data collects by the Myo.

4.1 Raw data collection

In order to develop a system which allows the analysis of different gestures, it is necessary to access the raw data of each single EMG sensor of the Myo armband. MATLAB offers a good tool to work with raw data collected from the Myo, especially thanks to the fact that behind this task there is a supportive community who is working on it. Once the bracelet is connected to the computer via Myo Connect, a connection with MATLAB is established installing the MyoMex library which allows to access the current raw data of the armband.

The first difficult part of these works was to select the features to run our Unity program described in paragraph 5, that is, choosing the gestures that permit to have different sensor diagram. For this purpose, several gestures were analyzed gestures using a site provided by Myo [1].
To understand muscle activation a kinesiology book [3] was used as reference. As a result, six different gesture to perform six different actions were found:

RX : *relax* means the arm is kept in a relaxed position along the side, so that no muscle activation is captured. In Unity it is considered "OK".

F : this action expects a fist with the palm facing left.In Unity is called using the same label, and it means forward.

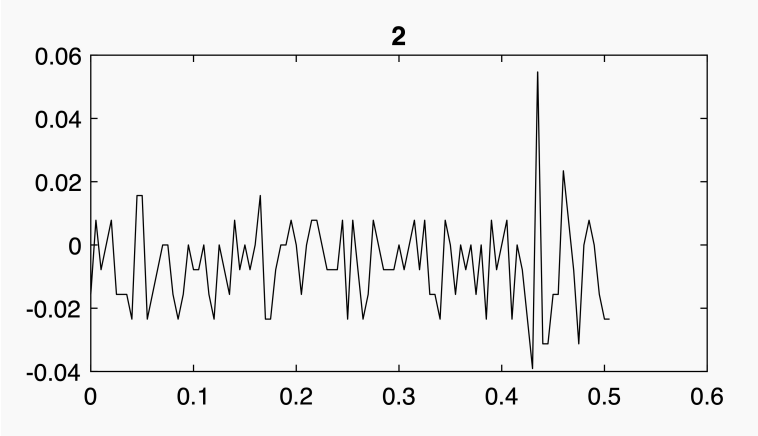
OH : open hand; as the previous palm faces left. It is used as "UP" in Unity

R : the right gesture performed with a wrist extension. It has the same meaning in Unity.

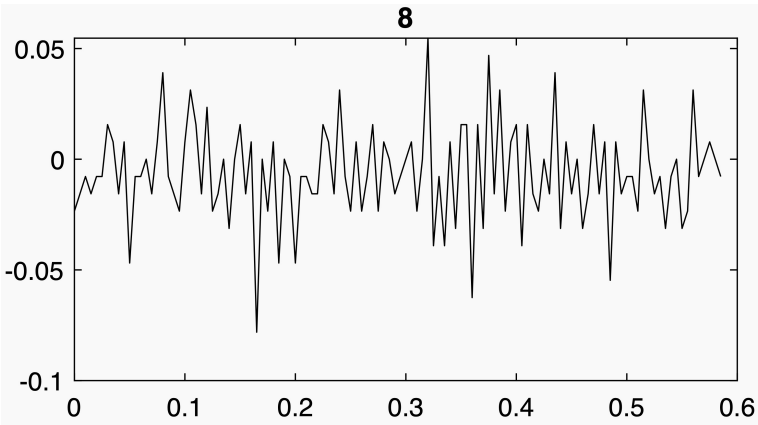
L : left gesture performed with a wrist extension. It has the same meaning in Unity.

DN : it is performed with a ulnar deviation from the first. It means *down* in Unity.

For each gesture, the code enables the collection of Myo data for about T seconds and, eventually, it plots every value of a single sensor. The EMG data are logged in a matrix with nine columns. Every sensor value is saved in a line, and the last one is a number that represents the action. The data range is between -1 and 1 without any unit. The Figure 3 shows some data during the acquisition. They come from a eight-figure table where each frame shows a sensor value.



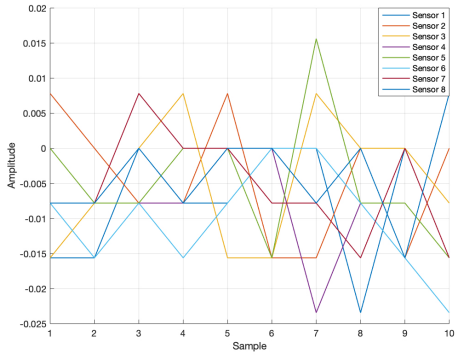
(a) *Sensor number two.*



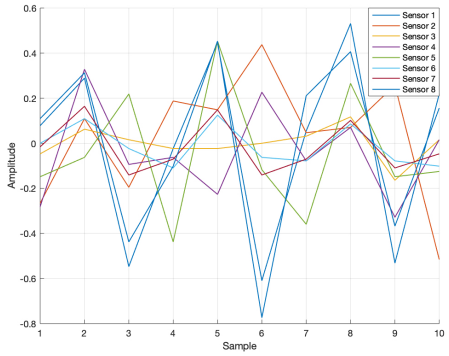
(b) *Sensor number eight.*

Figure 3: Sensors data in a cell describing two different actions.

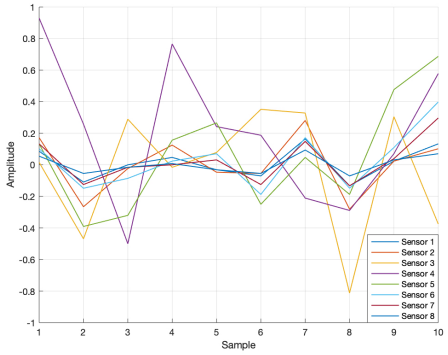
Moreover, working with the data, it is possible to display the images shown in figure 4. The following ones show the plot of ten sample raw EMG data for each of the eight sensors using different colors. The frequency of the acquired samples is about 200 Hz.



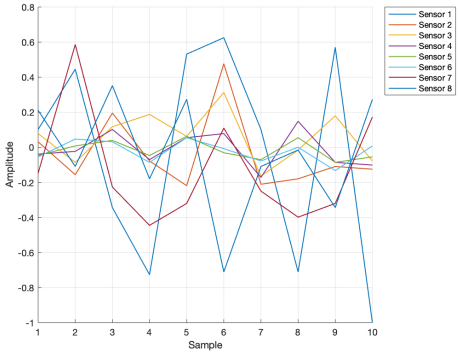
(a) *Relax.*



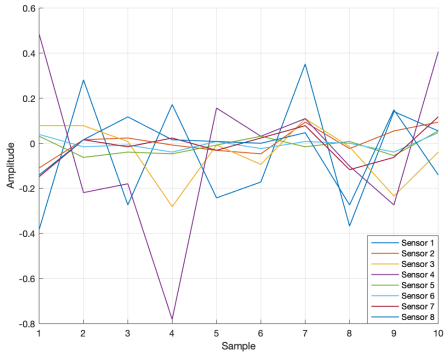
(b) *Fist.*



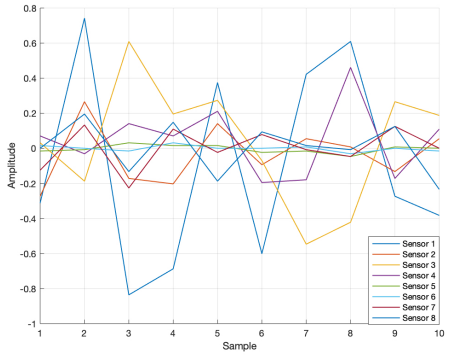
(c) *Right*



(d) *Left.*



(e) *Open hand.*

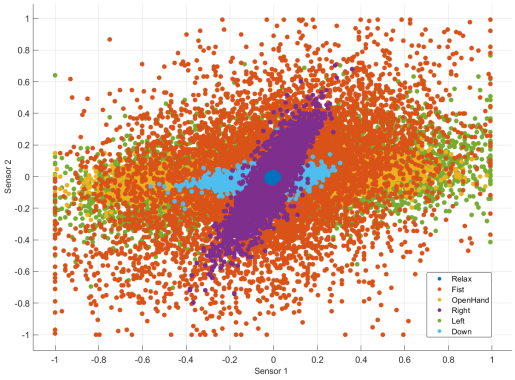


(f) *Down.*

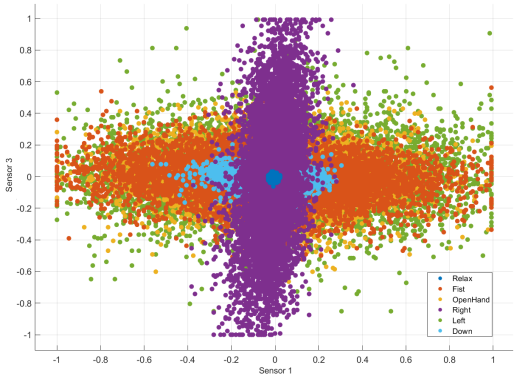
Figure 4: Sensors data in a cell describing two different actions.

The data can also be printed in two dimensions as shown in Figure 5. This kind of graphic is very meaningful to understand the correlations between sensors; furthermore they also show if the data was correctly acquired. In fact, as it is possible to see in Figure 6, for any problem related to the bracelet position, the data pattern wouldn't be well shaped as in this case.

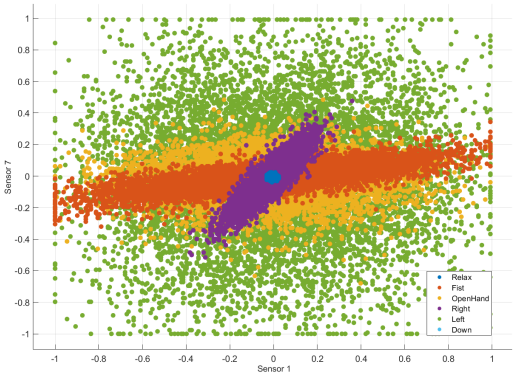
If points are near to the origin (i.e. relax), it means the values are close to zero. If a line can be found, as in figure 5a, it denotes a correlation. The more the correlation shown, the more it is easy to recognize the actions using a machine learning algorithm.



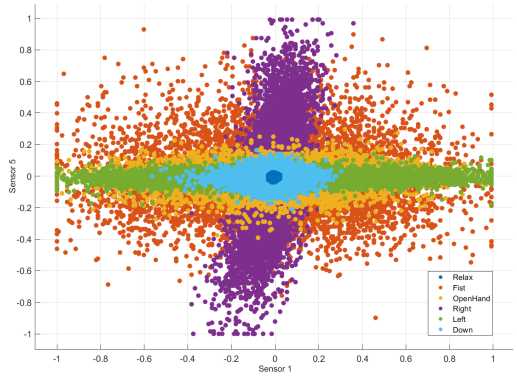
(a) *Sensor 1 and sensor 2.*



(b) *Sensor 1 and sensor 3.*

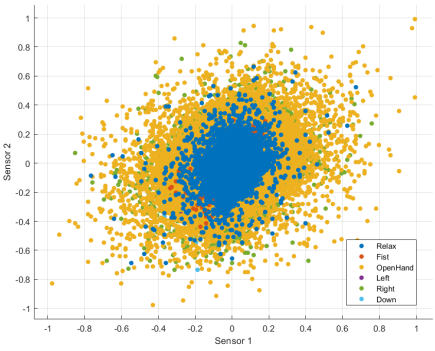


(c) *Sensor 1 and sensor 7.*

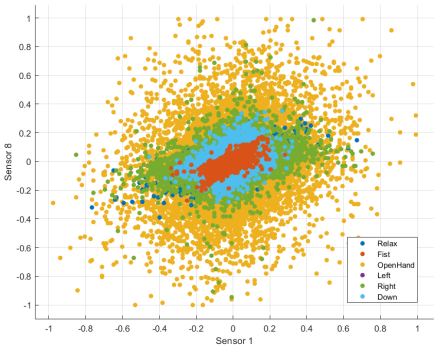


(d) *Sensor 1 and sensor 8.*

Figure 5: Data acquired correctly. Correlations between sensor one and other.



(a) *Sensor 1 and sensor 2.*



(b) *Sensor 1 and sensor 8.*

Figure 6: Data wrongly acquired.

Once enough data was taken, also considering different people in the acquisition, the network training can be performed. In this step, a large amount of data can guarantee a more precise classification can be made by the trained neural net.

4.2 Neural network training

A neural network, as already mentioned, is essential in order to classify the signals into the different actions in real time. Different algorithms have been tried to find the most suitable for this case.

MATLAB has a good and easy to use Machine learning extension. It provides different network models which can be customized in many ways. The model which revealed to be the most performing is LSTM, as it reached the highest accuracy. Moreover, also the Neural Network structure was selected in order to get better performances. In the appendix it is shown that the neural network structure has different layers.

- Input Layer: it's the first layer; the number of neurons is equal to the number of sensors of the EMG ambard.
- LSTM Layer: it's the second layer which apply the LSTM method; it has a 150 hidden units (neurons).
- FullyConnectedLayer: it is the layer that maps the output of the LSTM layer into a desired output size (a hundred times the number of classes).
- DropoutLayer: this layer permits to ignore all the neurons with a probability, in this case, lower than $p < 0.3$. It is useful in order to prevent the data over fitting.
- FullyConnectedLayer: it maps the output of the previous layer into a desired output size (number of classes).
- SoftmaxLayer: this layer permits to properly run a multi-class function; it is placed just before the Classification Layer and it is used to determine the probability of multiple classes at once.
- ClassificationLayer: it is the last layer and it allows to classify the input of the eight sensor into a probability. The highest probability correspond to the gesture made.

The used neural network required a specific data organization. All the data was organized in Cells and divided in Train and Test data. Each cell contains n vectors with 10 sample each of the data acquired. The number n is equal to the total amount of data acquired divided by the number of samples per vector (i.e. 10 in our case). In this way action classification is not done sample by sample (quite a difficult task - more sensitive to noise) but vector by vector.

Once all the network parameters are set, and data is split in cells, the Neural Network can be trained. As shown in the Figure 7, MATLAB provide a visual the step by step training performance monitoring interface.

As already mentioned in section 2, the training is divided in different epochs. At the end of each epoch, the software modifies some parameters in the network structure. Epochs can be seen in the picture as numbers and as vertical lines.

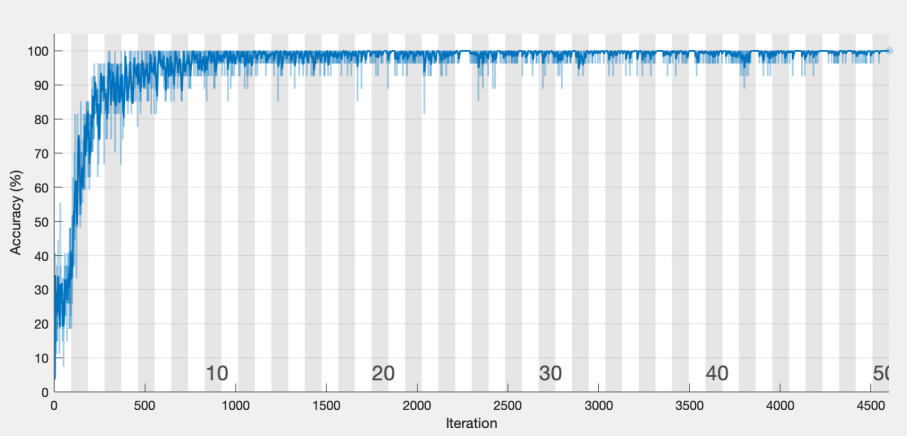


Figure 7: Training graph.

The accuracy reached is (98.4%) and, as shown in the confusion matrix in figure 8, meaning the network works correctly.

$$\begin{bmatrix} 81 & 0 & 0 & 3 & 0 & 0 \\ 0 & 81 & 0 & 2 & 0 & 0 \\ 0 & 0 & 82 & 1 & 0 & 0 \\ 0 & 0 & 0 & 83 & 0 & 1 \\ 1 & 0 & 0 & 0 & 82 & 0 \\ 0 & 0 & 0 & 0 & 0 & 83 \end{bmatrix}$$

Figure 8: Confusion matrix, $accuracy = 98.4\%$.

4.3 Real time action recognition

The last script used in this research permits to recognize actions in real time, and then, through a ZMQ protocol, delivers the output to Unity.

Once the ZMQ protocol is started, we enter an infinite loop in which the real time script acquired data as already explained. In fact, signals are acquired for 200 ms from the sensors; this time allows to have a minimum of forty samples. Like in the training, data is split in cells, each with ten samples. Then the neural network is fed.

With a minimum of four cell of feed, four different classification are given: the action selected will be the one associated with the mode.

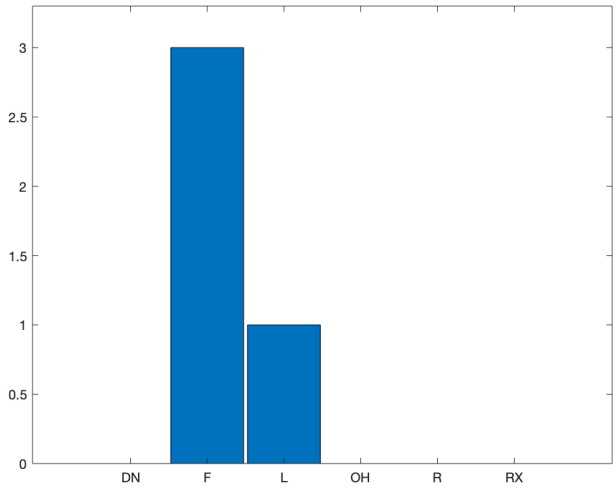


Figure 9: Action = Fist, Prediction = Fist

4.4 Real time simulation

In order to test the MATLAB unity communication and to see how the Neural networks works a fourth script was created.

It randomly selects n samples from a data collection not used for the network training. The latter is fed with this amount of data, and the outputs give the possibility to check the real performance of the classification system.

5 Unity Game Development

The aim of the Unity project is to create a safe environment where the different actions predicted by the ML network can be simulated and tested. The test scenario contains a 3D helicopter model surrounded by mountains, trees and other common objects. The control of the vehicle can be chosen from the options of a menu by selecting or the ZMQ messaging protocol or the embedded virtual joystick.

5.1 Overall description of the Game

The task, from a user point of view, is to navigate through a predefined path in the 3D space by touching red rings during the flight. All the rings are in a predefined position at the beginning, and the game ends when all of this objects are collected by the player. Therefore, the software can be used for muscle rehabilitation purposes, where the patient can practise the movements by playing without even thinking on the actual monotonous work they have to perform.

The game uses a publicly available free asset called 'Base Helicopter Controller' which can be found in the Unity Asset Store [5]. The source code is available on Github [4]. The package contains a couple of different Helicopter models with basic control options, therefore one of the main goals was to extend the functionality of the controller in order to be able to receive commands from other programs requested over TCP protocol.

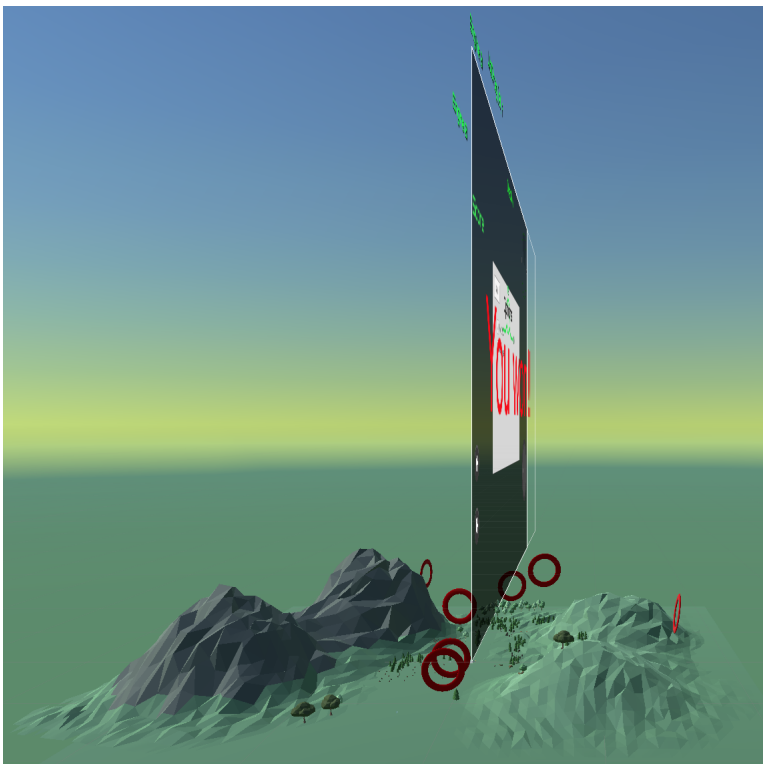


Figure 10: Unity environment.

5.2 The communication interface

The ZMQ Subscriber listens on the tcp://127.0.0.1:5000 port by default and tries to parse the information received. The structure of the message is predefined and can only take six different formats: "R", "L", "F", "U", "D" and "OK"; the explanation of the actions are Right, Left, Forward, Up, Down and OK - no action - respectively. After receiving an unclassified command the controller does not change its state, remaining in the last valid one. The controller also holds the last action until another command comes and overwrites it. The Up and Down commands are affecting the main engine speed generating vertical movement, while the *Right* and *Left* commands give a certain amount of rotation of the vertical axis around the horizontal one. The forward action brings the model ahead, but while the action goes on, a little loss in height can be observed, since the helicopter tilts its front downwards to reach a higher speed.

```
private float Evaluate_EMG_Input(string input)
{
    if(input == horizontalAxis && lastAction == "R") {
        return 0.5f;

    }
    else if(input == horizontalAxis && lastAction == "L") {
        return -0.5f;

    }
    else if(input == verticalAxis && lastAction == "F") {
        return 1.0f;
    }
    else if (input == jumpButton && lastAction == "U") {
        return 1.0f;
    }
    else if (input == jumpButton && lastAction == "D") {
        return -1.0f;
    }

    else if (lastAction == "OK")  {
        return 0.0f;
    }
    return 0.0f;
}
```

5.3 Go Back Home - Emergency feature

The game can also simulate the behaviour of the helicopter in case of emergency situations e.g. if the commanding signal is lost due to a malfunction in the MATLAB part or if we are getting out of the range of the radio based vehicle remote control. Since we know the position and orientation of both the model and the base station, we can create a basic path control method which will be executed after a certain timeout (5 seconds) in the ZMQ receiver.

Since this is not the main focus of the project, the implemented solution is quite trivial, and can be used only in similar environments, that is where the endpoint is visible from the starting point. The motion can be divided into two steps: in the first one the helicopter aligns with the target orientation, pointing exactly towards the base station, while in the second phase the model starts moving forward decreasing its height until it finally lands on the landing station.

Figure 11: Go back home function in action.

Figure 12: Pause panel.

6 ZMQ - Connection between MATLAB and Unity

So far this project has been illustrated in its two main parts: the MATLAB side, which is responsible of classifying the signals of the EMG sensors into predefined labeled actions, and the Unity game, that is a simple model of a helicopter that can be controlled with specific commands.

To unify the two sides, so that the Unity model can be controlled by arm gestures, a communication between MATLAB and Unity software is needed. The way implemented in this project exploits the ZMQ library protocol.

The latter is a high performance tool for asynchronous messaging, suitable for distributed applications. In this specific case the communication is mono-directional, that is from MATLAB to Unity. Every time a new action is recognized, the deep learning code sends a message with the label to the game.

The protocol is based on a publish - subscribe architecture: the sender, called publisher, sends a message, through a message queue, without any knowledge of which receiver - subscriber - is listening. On the other hand, the receiver subscribes to a publisher and listens to the queue, without being aware of which publisher is posting the message. Since the two parts are not interacting directly with each other, the protocol is asynchronous, that is a published message isn't necessarily received in real time.

In this project, according to our needs, the MATLAB is the publisher, and it sends the label of the classified action, while Unity is the subscriber which receives and processes the messages.

7 Results

After a couple of months of research, a working prototype took shape, in which the deep learning classifier processes the signals of the bracelet quasi real-time ($delay < 1\,\text{sec}$) and sends the predicted gestures through the ZMQ interface to the running helicopter simulation according to the predefined protocol of accepted commands. The Figure 14 shows the system in action: the left gesture performed by the user cause the same directional change in the simulation.

Since the visual response of a correctly performed action is not instantaneous, the precise control of the model may seem challenging without practise, compared to a traditional keyboard or touch input, but the capability to recognise natural hand movements are truly promising and can be the goal of a lot of future products to formulate a better human-machine interface.

Figure 13: Unity game with vitrual control inputs.

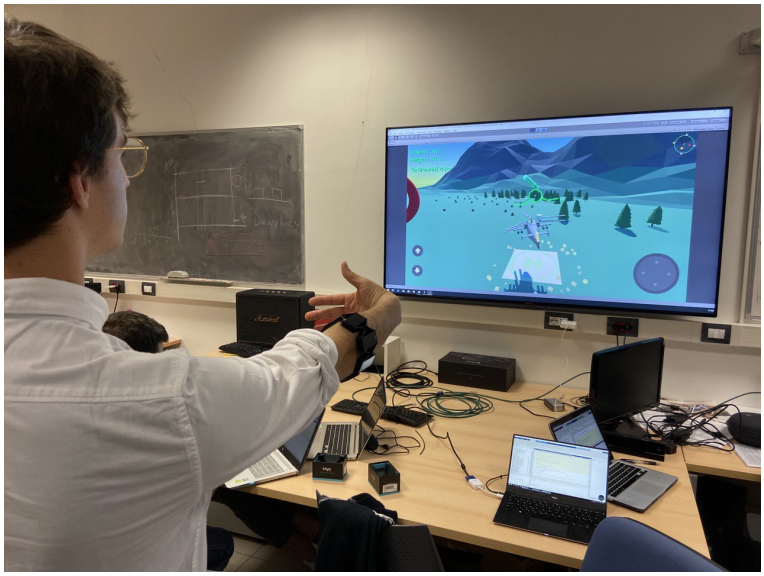


Figure 14: Test scenario.

8 Conclusions and considerations

On the technological side, during the project development serious issues were faced, especially with the reliability of the Myo Matlab wrapper in the long period usage, therefore a possible future goal could be to use the official C++ or python libraries for a more stable and efficient behaviour.

Regarding to the game functionality, the unity part could be extended with additional maps, quests or missions in order to provide a more immersive gaming experience. This project only focused on making a connection between the human and the virtual helicopter in the most natural way.

The Deep learning module can also be further improved, for instance, with other approaches in the construction of the model. The LSTM layer could be substituted with convolutional layers and the input can be changed from the discrete sensor values to their frequency spectrum. These options may lead to significant improvements in the accuracy of the system.
Collecting further data for longer time from a larger number of people can also decrease the system sensitivity to unwanted noises, and, possibly, it would result in a more precise gesture predicting system.

References

[1] Myo Armband. *MyoDiagnostics*. http://diagnostics.myo.com.

[2] mark-toma. *MyoMex*. https://github.com/mark-toma/MyoMex. 2017.

[3] Donald A Neumann. *Kinesiology of the musculoskeletal system-e-book: foundations for rehabilitation*. Elsevier Health Sciences, 2013, pag. 225.

[4] SunCube. *Base-Helicopter-Controller*. https://github.com/suncube/Base-Helicopter-Controller. 2016.

[5] SunCube. *Base-Helicopter-Controller Unity Asset Store*. https://assetstore.unity.com/packages/tools/physics/base-helicopter-controller-40107. 2015.

A Matlab scripts used for the computations

Here we have reported the Matlab launch scripts and all the function used during the project. We often use the same function for more then one script.

A.1 Row data collection

```
clc
clear all
close all

Data = [];

%%

% Relax, Fist, Open Hand, Right, Left, Down
labels = ["RX", "F", "OH", "R", "L", "DN",  "QQ"]

for k = 1:length(labels) - 1

    labels(k)
    disp('START ACQUISITION')
    pause(0.5)

    %% myo test

    MyoData =[];
    Time_   =[];

    %% create myo mex (ONLY FIRST TIME!!!!)
    % adds directories to MATLAB search path
    install_myo_mex;
    dataset=[];

    act=0;
    close all
    Features = [];

    %% generate myo instance

    install_myo_mex;
    mm = MyoMex(1);
    pause(0.1);

    %% collect about T seconds of data
```

```
disp('Start recording');
T = 25;
m1 = mm.myoData(1);
m1.clearLogs();
m1.startStreaming();
pause(T);
initialTime = m1.timeEMG;

%%

% Plot of sensor voltage
while m1.timeEMG-initialTime < T
    if ~isempty(m1.timeEMG_log)
    for i=1:8
        subplot(3,3,i);
        plot(m1.timeEMG_log - m1.timeEMG_log(1),m1.emg_log(:,i));
        title(sprintf("%s%s%d", Action,"sensor", i));
    end
    end
    pause(0.001);
end

%%

    clear initialTime;
    m1.stopStreaming();

    MYOdata = [];
    time = m1.timeEMG_log - m1.timeEMG_log(1);

    MyoData = m1.emg_log(:,:);

    mm.delete;
    clear mm;

    % Last element is k, ones create an array of all ones
    newData = [MyoData'; ones(1,length(MyoData(:,1)))*k];
    Data    = [Data, newData];

    labels(k)

% Data beckup, in case of crashing

filename = append('EMG_1ac25sec_ALE1-06', labels(k), '.mat')
save(filename,'Data')
```

```
    disp(' NEXT GESTURE WILL COME, ATTENTION ! WAIT')
    disp(labels(k+1))
    pause(10)

end


%% Save data in a table .csv

Dataset = table();
Dataset{:,:} = Data;
writetable(Dataset,'EMG3_1ac25sec_MAX1.csv','WriteRowNames',true)
```

A.2 Neural network training

```
clc
close all
clear all

% Which class we got in the data file we upload
class = ["RX", "F", "OH", "R", "L", "DN"];

Data = [];

% Starting from 0 the number of acquisition, if you don't load files
numberacquFILE = 0;

%% Load files

% Alessandro data

load('data/EMG2_ALE_DN_1_DN.mat');
Data2 = Data;
Data2(9,:) = 6;

load('data/EMG2_ALE_complessivi_1_RXFOHRLSM.mat');
Data1 = Data;

% We want to remove an action from the second file adding a new one.
% We acquire the down later

interest_actions = [1, 2, 3, 4, 5];
FinalData = select(numberacquFILE, 25, interest_actions, Data1);

Data1 = FinalData;

% Kristof data

load('data/EMG3_KK_DN_1.mat');
Data4 = Data;
Data4(9,:) = 6;

load('data/EMG2_KK_complessivi_1_RXFOHRLSM1.mat');
Data3 = Data;
position = [];

interest_actions = [1, 2, 3, 4, 5];
```

```
FinalData = select(numberacquFILE, 25, interest_actions, Data3);
Data3 = FinalData; % chande data wich we are working with

% Massimiliano data

load('data/EMG3_Max_RxFOhRLDn1.mat');
Data5 = Data;

% Put all the data together

Data = [Data1 Data2 Data3 Data4 Data5];

%% If necessary Load Dataset from .csv file

[Data, numberacquFILE] = writedata(25, 1, 1, ...
    'EMG2_1ac25sec_ALE1-06', 1, Data, numberacquFILE);

filename = append('data/EMG2_ALE_DN_1_DN', ...
    string(numberacquFILE),'.mat');

save(filename,'Data');

%%

interest_actions = [1, 2, 3, 4, 5, 6]; % Number of the action

n_of_classes = length(interest_actions);

% If you want to select some actions from the file.
FinalData = select(numberacquFILE, 25, interest_actions, Data);
Data = FinalData;

%% Create some cells from arrays

temp = cellaF(Data, interest_actions);
Data = temp;

% Define X/Y -> sensor aplitude / action

X = {};
Y = [];

for ii = 1:length(Data)
    temp = Data{ii,1};
    X{ii,1} = temp(1:8, 1:end);
    Y(ii,1) = temp(9, 1)';
```

```
end

X_ = {};
Y_ = [];
X_fin = {};
Y_fin = [];

for ii = 1:length(X)
    % how many elements each cell
    temp = X{ii,1};
    n_acquisition = 10;
    leng = round(length(X{ii,1})/ ...
        (n_of_classes * n_acquisition) - 0.5);
    n_cells = round(leng*n_of_classes - 0.5);
    for jj = 0:(leng-1)
        num = jj*ii+1;
        X_{jj+1,1} = temp(1:8, 1 + ...
            n_acquisition*(jj):n_acquisition*(jj+1));
        Y_(jj+1) = Y(ii);
    end
    X_fin = {X_fin{:,:} X_{:,1}};
    Y_fin = [Y_fin Y_];
end

X_ = X_fin';
Y_ = Y_fin';

%% Train and Test

% Define training and test part
X_test = {};
Y_test = [];
test = [];

% How many test elements
tot = leng * n_of_classes;
jump = 6; % every six elements a test
num_test = round(tot/jump - 0.5);
jj = 1;

for ii = 1:(num_test-1)
        X_test{jj,1} = X_{jump*(ii), 1}; % X_test{jj,1} = ...
            X_(salto*(ii), 1);
    test = [test jump*(ii)]; % Save the position of the test cell
        Y_test(jj) = Y_(jump*(ii));
    jj = jj + 1;
```

```
end

Y_test = Y_test';

% Create Train cell

% Invert test vector
%In order to delete the X elements starting from the end.

temp = [];
for i = 1:length(test)
    temp = [test(1,i) temp];
end
test = temp;

% Delete X elements already present in test cell. Be careful, X change
for i = 1:length(test)
    X_(test(1,i)) = [];
    Y_(test(1,i)) = [];
end

X_train = X_;
Y_train = Y_;

%% Define Labels as categorical Variables

temp1 = "";
temp2 = "";

for i = 1:length(Y_train(:,1))
    temp1(i) = class(Y_train(i));
end

for i = 1:length(Y_test(:,1))
    temp2(i) = class(Y_test(i)); % With cells
end

Y_train = temp1';
Y_test = temp2';

Y_train = categorical(Y_train);
Y_test  = categorical(Y_test);

%% Clear Workspace and plot Observation s

figure
```

```
hold on
plot(X_train{1}', 'linestyle','-','linewidth',1)
grid on
xlabel('Sample')
ylabel('Amplitude')
numFeatures = size(X_train{1},1);
legend("Sensor " + string(1:numFeatures),...
    'Location','northeastoutside')
hold off

%% Define Training algorithm

% Train the net with that part

% Layers
inputSize = 8;
numHiddenUnits = 150;
numClasses = n_of_classes;

layers = [ ...
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(n_of_classes*100)
    dropoutLayer(0.3) % Dropout layer
    fullyConnectedLayer(n_of_classes)
    softmaxLayer
    classificationLayer]

% Options

maxEpochs = 50;
miniBatchSize = 27;

options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',1, ...
    'Verbose',false, ...
    'Plots','training-progress');


net = trainNetwork(X_train,Y_train,layers,options);

%% Save training datas
```

```
save('training/training_ALE_KK_MAX_RFOHRLDW_2layer.mat','net')


%% Test the net and plot the confusion matrix

load("training/training_ALE_KK_MAX_RFOHRLDW_2layer.mat")

miniBatchSize = 27;

YPred = classify(net, X_test,'MiniBatchSize',miniBatchSize);

acc = sum(YPred == Y_test)./numel(Y_test)

C = confusionmat(Y_test,YPred)

trace(C)
```

A.2.1 Select function

This function is used in order to select some actions from a package of data which contains more that the actions we need. For example we acquired also the Spider-man action giving its the number 7. We can discard it just using the select function.

```
function DataNew = select(numberacqu, time, interest_actions, Data)

DataNew  = []; % Array
position = [];


for ii = 1:length(interest_actions)

        newidx = find(Data(9,:) == interest_actions(ii));
        position = [position newidx];

end

DataNew = zeros(9,length(position));

for jj = 1:length(position)
   DataNew(:,jj) = Data(:,position(jj)); % Arrays, cell
end
```

A.2.2 Writedata function

Writedata function permit to save data converting them from a .csv file to a matix in a .mat file.

```
function [Data, numberacquTOT] = writedata(time, numberacqu, ...
    numberactions, firstName, numberoffiles, Data, numberacquTOT)

% lengthData = numberactions * numberacqu * time * 50 * 8;
% 50 acq second (0.02 freq.), 8 sensors,

numberacqu_temp = numberacquTOT + (numberoffiles * ...
    time * numberacqu)/25;
numberacquTOT = numberacqu_temp;
% Data = zeros(9,numberoffiles * lengthData);

for i = 1:numberoffiles

  filename = append(firstName, string(i),'-0', ...
    string(numberactions),'.csv')

  T = readtable(filename);

  newData = table2array(T); ;

  Data = [Data, newData];

end


end
```

A.2.3 cellaF

This function permit to create cells from a matrix of data. From instance the number of cells is equal to the number of actions.

```
function cell = cellaF(Data, interest_actions)

cell = {};
position = [];
last = [0];
```

```
for ii = 1:length(interest_actions)
     newidx = find(Data(9,:) == interest_actions(ii));
     position = [position newidx];
     last = [last position(end)];
end

for jj = 1:(length(last)-1)
        cell{jj,1} = Data(:,last(jj)+1:last(jj+1));
end

end
```

A.3 Real time action recognition

```
close all;
clear;
clc;

labels = ["RX", "F", "OH", "R", "L", "DN"];

%% Add the path necessary for zmq

run ./addZmqUtility;

topic = '';
ip    ='tcp://127.0.0.1:5000';

[pub,ok] = Publisher(ip);

if not(ok)
  errormsg('Publisher not initialized correctly');
end

%% Load the trained model

load('training/training_ALE-KK-MAX-RFOHRLDW-2layer.mat');

%% Start acquisition

trial = 1;

while trial == 1

    % myo test
    MyoData =[];
```

```
Time_    =[];
install_myo_mex; % adds directories to MATLAB search path
dataset = [];
% act = 0;

%%
for labeIndex = 1:1  % one acquisition for real time

    close all
    Features=[];

    disp('START ACQUISITION')
    pause(0.5)

    % generate myo instance
    install_myo_mex;
    mm = MyoMex(1);
    pause(0.1);

    disp('Start recording');
    T = 0.2; % Time of acquisition
    m1 = mm.myoData(1);
    m1.clearLogs();
    m1.startStreaming();
    pause(T);

    initialTime = m1.timeEMG;

    clear initialTime;

    m1.stopStreaming();

    MYOdata=[];

    time = m1.timeEMG_log - m1.timeEMG_log(1);

    mu = mean(m1.emg_log(:,:))';
    MyoData = m1.emg_log(:,:);

    mm.delete;
    clear mm;

end

clc
```

```
    result  = [];
    MyoData = MyoData';

    % End acquisition

%% Convert datas to cells

X = {MyoData};

X_fin = {};

n_acquisition = 10;

for ii = 1:length(X)

    % how many elements each cell
    temp = X{ii,1};
    leng = round(length(X{ii,1})/(n_acquisition) - 0.5);

    for jj = 0:(leng-1)
        num = jj*ii+1;
        X_{jj+1,1} = temp(1:8, 1 + ...
            n_acquisition*(jj):n_acquisition*(jj+1));
    end

    X_fin = {X_fin{:,:} X_{:,1}};

end

X = X_fin';

%% Predict action

    miniBatchSize = 27;

    YPred = classify(net, X,'MiniBatchSize',miniBatchSize);

    Prediction = mode(YPred);

    Prediction = string(Prediction);

%% Init console

   msg_labels = ["OK", "F", "U", "R", "L", "D"];

   index = find(labels(1,:) == Prediction);
```

```
    msg = char(msg_labels(index));

    if (strcmp(msg,''))
        break;
    end

    %fprintf(['Publish: '+ msg + ' \n']);
    bool = pub.publish(topic, msg);


end
```

A.4 Real time simulation

```
% close all;
clear;
clc;

class = ["RX", "F", "OH", "R", "L", "DN"];

%% Add the path necessary for zmq

% test publisher-subscriber procedure

run ./addZmqUtility;

topic = '';
ip    ='tcp://127.0.0.1:5000';

[pub,ok] = Publisher(ip);

if not(ok)
  errormsg('Publisher not initialized correctly');
end

%% Load the trained model and the data

load('training/training_ALE-KK-MAX-RFOHRLDW-2layer');

load('Data/TOT-SIMULAZIONE.mat');

%%

interest_actions = [1, 2, 3, 4, 5, 6];
```

```
n_of_classes = length(interest_actions);

% change data wich we are working with
FinalData = select(1, 25, interest_actions, Data);
Data = FinalData;

% Transform to cell
temp = cellaF(Data, interest_actions);
Data = temp;

%%

while(1)

    n = round(9000*rand(1));

    MyoDataRX = Data{1,1}(:,n:n+40);
    MyoDataF = Data{2,1}(:,n:n+40);
    MyoDataOH = Data{3,1}(:,n:n+40);
    MyoDataR = Data{4,1}(:,n:n+40);
    MyoDataL = Data{5,1}(:,n:n+40);
    MyoDataDN = Data{6,1}(:,n:n+40);

    l = 30*rand(1);

    if(l<=5)
        MyoData = MyoDataRX;
        action = "RELAX"
    elseif(l>5 && l<=10)
        MyoData = MyoDataF;
        action = "FIST"
    elseif(l>10 && l <=15)
        MyoData = MyoDataOH;
        action = "OPEN_HEND"
    elseif(l>15 && l <=20)
        MyoData = MyoDataR;
        action = "RIGHT"
    elseif(l>20 && l <=25)
        MyoData = MyoDataL;
        action = "LEFT"
    elseif(l>25 && l <=30)
        MyoData = MyoDataDN;
        action = "DOWN"
    end

    % Convert datas to cells
```

```
    X = {MyoData};

    X_fin = {};

    n_acquisition = 10;

    for ii = 1:length(X)

        % how many elements each cell
        temp = X{ii,1};
        leng = round(length(X{ii,1})/(n_acquisition) - 0.5);

        for jj = 0:(leng-1)
            num = jj*ii+1;
            X_{jj+1,1} = temp(1:8, 1 + ...
            n_acquisition*(jj):n_acquisition*(jj+1));
        end

        X_fin = {X_fin{:,:} X_{:,1}};

    end

    X = X_fin';

    % Predict action

    miniBatchSize = 27;

    YPred = classify(net, X,'MiniBatchSize',miniBatchSize);

    Prediction = mode(YPred);

    % Plot histogram
    % hist(YPred);

    Prediction = string(Prediction)

    pause(1);

end
```

B Come back home function made in Unity script

Here we have reported the part of the Unity script made for the *Back home function* function. As we said in paragraph 5 when the helicopter is not receiving other commands (possible crash) it come back to the start position alone.

The variable count count the number of time we receive the same input, timeSinceMessage if the time from the last input. Whenever we receive an action timeSinceMessage is set equal to zero. Whenever we receive an action different from the previous count is set equal to zero.

B.1 Back home function

```
void FixedUpdate()
{
    ProcessingInputs();
    LiftProcess();
    MoveProcess();
    TiltProcess();
    Visualize();


    timeSinceMessage += Time.fixedDeltaTime;

    if(timeSinceMessage <= 5f)
    {
        EngineForceRef = EngineForce;
        refer = Vector3.Distance(transform.position, ...
        targetFin.position);
    }


    if ((timeSinceMessage > 5f && Vector3.Distance(transform.position,
       targetFin.position) > 2f) || ...
                (count > 5f && Vector3.Distance(transform.position,
                   targetFin.position) > 2f))
    {
        // Go home
        Debug.Log("Going home");
        transform.position = Vector3.MoveTowards(transform.position,
           targetFin.position, 0.15f);

        if (transform.position.x - targetFin.position.x > 2f || transform.
           position.z - targetFin.position.z > 2f)
        {
            if (EngineForce > 10)
            {
                myVector = new Vector3(0.0f, transform.position.y, 0.0f);
                transform.LookAt(targetFin.position + myVector);
            }
            if (EngineForce > 10)
            {
```

```
                // Decrise velocity
                EngineForce = Vector3.Distance(transform.position, targetFin
                    .position) / refer * EngineForceRef;
            }
            else if (distanceToGround > 7)
            {
                // Velocity shouldn't be less then 10
                EngineForce = 10;
            }
            else
            {
                // If you are near to the ground decrease velocity
                EngineForce = 8;
            }
        }
        else
        {
            // If you are very near to the Target decrease velocity
            EngineForce = 8;
        }
    }
}
```