Construction of a robust pipeline to estimate 3D trajectories from 2D hand landmarks

Master's Thesis submitted to the

Faculty of Informatics of the *Università della Svizzera Italiana*

in partial fulfillment of the requirements for the degree of

Master of Science in Informatics

presented by

Umberto Cocca

under the supervision of

Dr. Alessandro Giusti

co-supervised by

Dr. Gianluigi Ciocca, Dr. Loris Roveda

February 2022

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Umberto Cocca
Lugano, 23 February 2022

-dedica-

“Gutta cavat lapidem.”
lat. «la goccia scava la pietra»

Abstract

Robotics systems are increasingly adopted for filming and photography purposes. Exploiting robots, in fact, it is possible to program complex motions for the camera, achieving high-quality video/photography.

These robots, until recent years, must be controlled through physical remote systems, nevertheless, this is not true anymore. Thanks to the latest efforts, a high-fidelity hand and finger tracking solution without specialized hardware needed (e.g. depth sensors, Kinect etc...) was built by MediaPipe machine learning solutions [Zhang et al., 2020].

Delving into the goal of capturing 3D motions taking advantage of a hand tracking system, my contribution was, in the first instance, to build a hand gestures recognition neural network, next to set up a robust pipeline that solves different problems during the process of detecting 3D trajectories obtained from 2D pixel landmarks as estimating orientation (roll, yaw and pitch) and consequently z-space as a result of orientation test algorithm and matrices transformations. The trajectories are based on a dynamic speed, interpolating and smoothing them with ridge regression.

In conclusion, the captured trajectory has been launched on a drone in simulation. It is implemented in the ROS framework, using Gazebo as a tool by reason of robust physics engine with high-quality graphics.

Since hand tracking is a vital component to provide a natural way for interaction and communication in AR/VR then the entire pipeline that has been built to detect the trajectories can be easily translated into another kind of task.

Acknowledgements

Work in progress

Contents

Figures

Tables

Equations

Listings

Introduction

Chapter 1

Literature review

This chapter discusses previous research about the topic, providing a brief introduction to the approach we adopted.

Chapter 2

Background

This chapter provides some background concepts to understand the material presented in this thesis.

2.1 Regression

The goal of regression is to predict the value of one or more continuous target variables t given the value of a D-dimensional vector x of input variables. [Bishop, 2006]
Given a training data set comprising N observations x_n, where $n = 1, ..., N$, together with corresponding target values t_n, the goal is to predict the value of t for a new value of x.
From a probabilistic perspective, we aim to model the predictive distribution $p(t|x)$ because this expresses our uncertainty about the value of t for each value of x.

2.1.1 Linear Models for Regression

The simplest linear model for regression is one that involved a linear combination of the input variables:

$$y(x, w) = w_0 + w_1 x_1 + ...w_D x_D \tag{2.1}$$

where $x = (x_1, ..., x_D)^T$. This is often simply known as linear regression. The key property of this model is that it is a linear function of the parameters w_i, but also of x_i and establishes significant limitations on the model. It is possibile extend the class of models by considering linear combinatioins of fixed nonlinear functions of the input variables:

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x) \tag{2.2}$$

where $\phi_j(x)$ are known as basis functions. The total number of parameters in this model will be M.
The parameter w_0 is called bias parameter. It is often convenient to define an additional dummy "basis function" $\phi_0(x) = 1$, so that:

$$y(x,w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x) \tag{2.3}$$

where $w = (w_0, ..., w_{M-1})$ and $\phi = (\phi_0, ..., \phi_{M-1})^T$
By using non linear basis functions, we allow the function y(x,w) to be a non linear function of the input vector x. A particle example of this model where there is a single input variable x is the polynomial regression. The basis functions take the form of powers of x so that $\phi_j(x) = x^j$.
There are other possibile choices for the basis functions as:

$$\phi_j(x) = e^{\frac{-(x-u_j)^2}{2s^2}} \tag{2.4}$$

where u_j regulates the locations of the basis functions in input space, while the parameter s is their spatial scale. These are usally reffered to as "Gaussian" basis functions.
We can use simply the identity basis functions in which the vector $\phi(x) = x$.

2.1.2 Normal equations

The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an error function that measures the misfit between the function $y(x,w)$, for any given value of w, and the training set data points. One simple choice of error function, which is widely used, is given by the sum of the squares of the errors between the predictions $y(x_n,w)$ for each data point x_n and the corresponding target values t_n, so that we minimize

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, w) - t_n]^2 \tag{2.5}$$

where the factor of 1/2 is included for mathematical convenience. It is a nonnegative quantity that would be zero if, and only if, the function $y(x, w)$ were to pass exactly through each training data point.
We can solve the curve fitting problem by choosing the value of w for which $E(w)$ is as small as possible. Because the error function is a quadratic function of the coefficients w, its derivatives with respect to the coefficients will be linear in the elements of w, and so the minimization of the error function has a unique solution w^*. The resulting polynomial is given by the function $y(x, w^*)$.
We can write **??** (2.5) in matrix notation [Wikipedia, 2021] as :

$$\frac{1}{2}(\phi w - t)^T(\phi w - t) \tag{2.6}$$

where ϕ is an NxM matrix, so that:

$$\phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_{M-1}(x_N) \end{pmatrix} \tag{2.7}$$

Equation 2.1. This is called the design matrix whose elements are given by $\phi_{nj} = \phi_j(x_n)$.

$$w = \begin{pmatrix} w_0 \\ \vdots \\ w_{M-1} \end{pmatrix} \quad t = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix} \tag{2.8}$$

Therefore, we have to solve the optimization problem finding the minimum of the cost function $E(w)$:

$$w^* = \arg\min_{w} \frac{1}{2}(\phi w - t)^T(\phi w - t) \tag{2.9}$$

So we compute the gradient and solving for w we obtain:

$$\begin{aligned} \nabla E(w) = \phi^T(\phi w - t) = 0 \\ \phi^T \phi w - \phi^T = 0 \\ \phi^T \phi w = \phi^T t \\ w^* = (\phi^T \phi)^{-1} \phi^T t \end{aligned} \tag{2.10}$$

Equation 2.2. They are known as the normal equations for the least squares problem

This is the real minimum because if we take the second derivative $\nabla^2 E(w) = \phi^T \phi$ and this is a symmetric matrix, so it's also positive definitive matrix, which means that this objective function that we try to minimize is convex. So if we find a stationary point, such that derivative is zero, we also find a global minium. Furthermore, to find a solution we need to invert this matrix $\phi^T \phi$, so we need some condition that assure this is invertible and this is the case when the columns of the matrix are linearly independent.
Once we find the solution w^* and we receive a new data point that we never seen during training, we just predict the new target t^* as:

$$t^* = w^{*^T} \phi(x) \tag{2.11}$$

2.2 Regularization

If we use a set of features that is too expressive, for example a ten polynomial grade (x^{10}), then this interpolates very close our training data, because we have a model with 10 parameters where we can perfectly represents the data points. The risk is that the model became something like this:
[picture]
We want a trade-off between fits data and able to generalize. Infact, on the other hand if our model is not to expressive and not to complex our data will be linearly

rapresentable in the feature space and this means that the performance will be very poor.

2.2.1 The Problem Of Overfitting

We want to penalize for some features with parameters with high values, if our model is overfitting is very likely that the parameters of our model will have a big magnitude, this means that also the features supply to this parameters will be higher and very low and this cause a lot of problems.
In order to control over-fitting we introduce the idea of adding a regularization term to an error fuction, so that the total error function to minimized takes the form:

$$E(w) = E_D(w) + \lambda E_W(w) \tag{2.12}$$

Where λ is the regularization coefficient and it is the trade-off between how we well fit training set and how to establish the parameters w with low values, therefore having simple hypotesys avoiding over-fitting. E_D is the error based on dataset, while E_W is based on weights.

2.2.2 Cost Function

One of the simplest forms of regularizer is given by the sum-of-squares of the weight vector elements:

$$E_W(w) = \frac{1}{2} w^T w = \frac{||w||_2^2}{2} \tag{2.13}$$

where $||w||_2$ is the euclidean norm $\sqrt{\sum_{i=1}^{n} x_i^2}$.
This is also called ridge regression, a method of estimating the coefficitents of multiple-regression models in scenarios where indipendent variables are highly correlated.

If we also consider the sum-of-squares error function given by:

$$E_D(w) = \frac{1}{2}[t_n - w^T \phi(x_n)]^2 \tag{2.14}$$

then the total error function becomes:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}[t_n - w^T\phi(x_n)]^2 + \frac{\lambda}{2}w^T w \tag{2.15}$$

This particular choice of regularizer is known in the machine learning literature as weight decay because in sequential learning algorithms, it encourages weight values to decay towards zero.
Setting the gradient of E(w) wrt w to zero, and solving for w, we obtain:

$$w^* = (\lambda I + \phi^T\phi)^{-1}\phi^T t \tag{2.16}$$

and this is an extension of the least-squares solution (2.10). This is a better version than before because is also possibile prove that $(\lambda I + \phi^T\phi)$ is always invertible if $\lambda > 0$, therefore w^* always exists.

2.2.3 (Batch) Gradient Descent

Chapter 3

Tools

Chapter 4

Methodologies

Chapter 5

Evaluation

Conclusion and perspectives

Appendix A

List of Acronyms

Bibliography

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Wikipedia. Mean squared error — wikipedia, free encyclopedia, 2021. URL `https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=1057863857`. [matrix notation| Online; 03:08, 30 November 2021].

Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking, 2020.