UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

DJI Tello 3D Hand Gesture control

Advisor: Dr. Alessandro Giusti

Coadvisor: Dr. Loris Roveda

Coadvisor: Dr. Gianluigi Ciocca

Relazione della prova finale di:
Umberto Cocca
Matricola 807191

Anno Accademico 2019-2019

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

This is a sentence for all my followers

“Gutta cavat lapidem, non vi sed saepe cadendo.”

Abstract

Hello

Acknowledgements

Hello

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.

Siccome non è mai esistito un sistema che riconosce la mano davvero efficiente e in tmepo reale allora non è mai esistito la possibilità di poter controllare le traiettorie con la mano. In seguito alla soluzione recente fatta dai ricercatori di google che ha risolto i diversi problemi di performance e precisione nel riconoscere la mano, ora è possibile ragionare su come eseguire delle traiettorie con la mano. QUesto mio studio ha l'obiettivo quindi di aggiungere un'ulteriore mattoncino al lavoro svolto dai ragazzi di google per controllare il drone usando soltanto la mano, risolvendo quindi diversi problemi.

In recent years, the application of Artificial Intelligence (AI) and Deep Learning (DL) techniques to multi-agent cooperative problems has become increasingly successful. Gasser and Huhns, in *Distributed Artificial Intelligence* [**?**], have addressed the issue of coordination and cooperation among agents with the combination of distributed systems and AI, a discipline known as Distributed Artificial Intelligence (DAI).

Similarly, Panait and Luke, in their work *Cooperative multi-agent learning: The state of the art* [**?**], report that Multi-Agent Learning (MAL), the application of machine learning to problems involving multiple agents, has become a popular approach which deals with unusually large search spaces. In addition, they mention three of the most classic techniques used to solve cooperative Multi-Agent (MA) problems, that are Supervised and Unsupervised Learning, and Reinforcement Learning (RL). These methods are distinguished according to the type of feedback provided to the agent: the target output in the first case, no feedback is provided in the second, and reward based on the learned output in the last one.

Given these three options, the vast majority of articles in this field used reward-based methods to approach MAL scenarios, in particular RL, that make they possible to achieve sophisticated goals in complex and uncertain environments [**?**]. However, this technique is notoriously known for being hard, in particular it is difficult to design a

suitable reward function for the agents to optimise, which precisely leads to the desired behaviour in all possible scenarios. This problem is further exacerbated in multi-agent settings [**??**].

Inverse Reinforcement Learning (IRL) addresses this problem by using a given set of expert trajectories to derive a reward function under which these are optimal. Nevertheless, this technique imposes often unrealistic requirements [**?**].

Imitation Learning (IL) is a class of methods that has been successfully applied to a wide range of domains in robotics, for example, autonomous driving [**??**]. They aim to overcome the issues aforementioned and, unlike reward-based methods, the model acquires skills and provides actions to the agents by observing the desired behaviour, performed by an expert [**???**]. Using this approach the models learn how to extract relevant information from the data provided to them, directly learning a mapping from observations to actions. A more challenging situation occurs when the model also has to infer the coordination among agents that is implicit in the demonstrations, using unsupervised approaches to imitation.

In this direction, literature suggests that cooperative tasks sometimes cannot be solved using only a simple distributed approach, instead it may be necessary to allow an explicit exchange of messages between the agents. In *Multi-agent reinforcement learning: Independent vs. cooperative agents* [**?**], Tan affirms that cooperating learners should use communication in a variety of ways in order to improve team performance: they can share instantaneous informations as well as episodic experience and learned knowledge. Also Pesce and Montana propose the use of inter-agent communication for situations in which the agents can only acquire partial observations and are faced with a task requiring coordination and synchronisation skills [**?**]. Their solution consists in an explicit communication system that allows agents to exchange messages that are used together with local observations to decide which actions to take.

Our work is based on *Learning distributed controllers by backpropagation* [**?**], which proposes an approach in which a distributed policy for the agents and a coordination model are learned at the same time. This method is based on an important concept introduced in *Coordinated multi-agent imitation learning* [**?**]: the network has the ability to autonomously determine the communication protocol. Likewise, we use imitation learning approaches to solve the problem of coordinating multiple agents, introducing a communication protocol, which consists in an explicit exchange of messages between the robots. The communication is not provided to the network, instead, it is a latent variable which has to be inferred. The results show the effectiveness of this communication strategy developed, as well as an illustration of the different patterns emerging from the tasks.

Chapter 2

Background

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

Nel caso la tesi riguardi un progetto di sviluppo software, è preferibile, per maggiore chiarezza, descrivere i requisiti, l'interfaccia utente e l'architettura in capitoli separati (il codice potrà essere allegato in appendice). Il capitolo finale dovrà contenere una sintesi del lavoro, e una descrizione degli eventuali problemi aperti e dei possibili sviluppi futuri.

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.

Given a training data set comprising N observations xn, where n = 1, . . . , N, together with corresponding target values tn, 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 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:

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

2.1.3 (Batch) Gradient Descent

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 Regularized Linear Regression

2.2.4 Regularized Linear Regression - Normal Equation

Chapter 3

Tools

Chapter 4

Methodologies

Chapter 5

Evaluation

Conclusion and perspectives

Appendix A

List of Acronyms

AI Artificial Intelligence

DAI Distributed Artificial Intelligence

DL Deep Learning

IL Imitation Learning

IRL Inverse Reinforcement Learning

MA Multi-Agent

MAL Multi-Agent Learning

RL Reinforcement Learning

Bibliography

Zool Hilmi Ismail and Nohaidda Sariff. A survey and analysis of cooperative multi-agent robot systems: challenges and directions. In *Applications of Mobile Robots*. IntechOpen, 2018.