Optimizing flying trajectory of quadrotors with evolutionary neuro-controllers
Elective in Robotics - Quadrotor Modeling (Prof. M. Vendittelli)

Norman Di Palo
MSc. in Artificial Intelligence and Robotics

February 16, 2018

Abstract

In this project, we developed a neurocontroller, that is a nonlinear feedback controller based on a feedforward neural network, to enhance the performance of a normal PD control of a quadrotor. The neural network is trained using evolutionary strategies, optimizing the accuracy of the tracking, as well as the enrgy consumption. Several experiments show how, after around 30 mins of real time training, the behaviour of the quadrotor is enhanced.

1 Introduction

The control of a quadrotor can be a non-trivial task, especially when the dynamical model is not completely known or there are disturbances and model errors. Furthermore, the instability of a control law can lead to potentially dangerous behaviours that can damage the vehicle and its surroundings. In this work, we propose a method for exploring new control techniques that are learned online with no supervision, while also obtaining a robust stability that stops the quadrotor from unwanted dynamics. The developed architecture is composed of two parallel parts: a modified PD controller that tracks a desired reference position, and a neural network that provides an input to the quadrotor based on the state error, thus being a non-linear feedback controller. The neural network is trained using evolutionary strategies, a family of algorithms that have a wide literature and are often adopted as a scalable and higly parallelizable class of learning algorithms to find robust policies. In the first sections, we will describe the overall architecture and the ideas behind it, then we will show a series of experiments that prove the utility of the neurocontroller.

2 Quadrotor and PD controller models

The complete dynamical model of a quadrotor is quite complex, including various aerodynamical effects. In this project, we considered a simplified model with negligible aerodynamical and gyroscopic effects. The state is composed of the x, y and z position, the Euler angles, and respective time derivatives. The inputs are not directly the propellers torques, but are the total force applied along the z axis of the drone and the torques around the three axis. It is straightforward to see how these quantities are just a linear transformation of the propeller torques.

$$\xi = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \varphi, \theta, \psi, p, q, r)$$

$$u = (T, \tau_x, \tau_y, \tau_z)$$

The dynamical model adopted is the following, where the state and the inputs are as previously described, while m and I_i are the mass and inertias of the drone. For simplicity, in software simultions they were all set to 1.

$$\begin{aligned}
\ddot{x} &= -(\cos(\psi)\sin(\vartheta)\cos(\varphi) + \sin(\psi)\sin(\varphi))\frac{T}{m} \\
\ddot{y} &= -(\sin(\psi)\sin(\vartheta)\cos(\varphi) - \sin(\varphi)\cos(\psi))\frac{T}{m} \\
\ddot{z} &= -\cos(\vartheta)\cos(\varphi)\frac{T}{m} + g \\
\ddot{\varphi} &= \frac{\tau_\varphi}{I_x} \\
\ddot{\vartheta} &= \frac{\tau_\vartheta}{I_y} \\
\ddot{\psi} &= \frac{\tau_\psi}{I_z}
\end{aligned}$$

A PD feedback controller for regulation and trajectory tracking must first control the attitude of the quadrotor, that is its orientation in space, and after that the T input that actually moves it. Hence, a inner attitude control loop must be designed inside a position control loop, with the former being faster than the latter. The desired attitude is computed based on the desired velocity of the quadrotor. This is composed by a constant gravity compensation part, and an error reduction part. To make the design of the controller easier, it will be based on the linearized model of the quadrotor around an hovering position, that is having $\varphi, \theta = 0$ and all the velocities equal to zero. Also, we consider $T = -mg$.
The PD controller has the following structure: an arbitrary desired ψ can be chosen (for this application, we consider it to be constant). Then, based on the linearized model, we can compute the desired θ, φ. The linearized model is the

following:

$$\ddot{x} = g(cos(\psi)\theta + sin(\psi)\varphi)$$

$$\ddot{y} = g(sin(\psi)\theta - cos(\psi)\varphi)$$

Thus, we can obtain, from the desired $\ddot{x}$ and $\ddot{y}$, the reference angles.

$$\varphi_{des} = \frac{1}{g}(\ddot{x}(sin(\psi_{des})) - \ddot{y}cos(\phi_{des})$$

$$\theta_{des} = \frac{1}{g}(\ddot{x}(cos(\psi_{des})) + \ddot{y}sin(\phi_{des})$$

These desired angles can easily be obtained with a PD inner loop based on the three torque commands.

$$u_{i:2,4} = \left[-k_{p,\varphi_i}(\varphi_i - \varphi_{i,des}) - k_{d,\varphi_i}(\dot{\varphi}_i)\right]$$

This controller will bring the attitude to the desired configuration while forcing a zero angular velocity. The first input is computed to minimize the z error.

$$u_1 = m(g + \ddot{z}_{des})$$

Finally, to find the desired $\ddot{x}, \ddot{y}$ to compute all the other parameters, we can define an outer PD loop based on the position error in the reference fixed frame.

$$\ddot{r} = -k_{p,r}(r - r_{des}) - k_{d,\dot{r}}(\dot{r} - \dot{r}_{des})$$

Where $r = [x, y, z]$ The desired velocity are always set to zero, since the trajectory in define as a successive and sometimes sparse set of desired positions, and also to improve the stability of the quadrotor. Furthermore, a new term is added to force the quadrotor to stay in a zero-angle configuration, thus without violating too much the linearization assumpation.

$$u_{i:2,4} += -k_{ang,\theta_i}(\theta_i)$$

These term has been proven empirically to improve stability in fast and quickly changing trajectories.

3 Evolutionary Algorithms and Neurocontroller

Evolutionary and genetic algorithms are a class of artificial intelligence algorithm, inspired by the concept of Darwinian evolution and selection, and used mainly for optimization problem, especially in cases where a classic solution is difficult to obtain. The main idea underneath these algorithms in to find a solution for and optimization problem by randomly evolving and combining the parameters of the best performing solutions, based on a fitness function $f(\theta)$, selecting the best ones and continuing to evolve them. These family of algorithms are widely used, with applications in robotics too. We designed an evolutionary

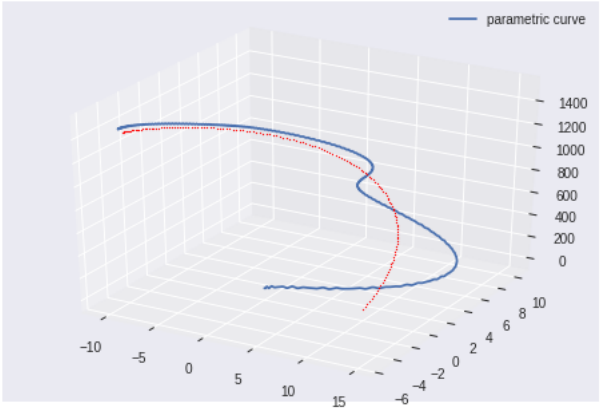

Figure 1: Trajectory of the PD controlled quadrotor (blue) and reference trajectory made of sequential reference positions. (red)

algorithm to evolve a neurocontroller that was run in parallel to a PD controller. The neurocontroller is made of a neural network that takes the errors in position and velocity. The outputs of the network are the 4 inputs of the drone, u_i. The activations functions are tanh for each layers, except for the output layer that is linear. We explored different topologies for the network, tyring to minimize as much as possible the number of parameters to make both the learning phase quicker and to make it computationally light. Our final model was made of two hidden layers, with 150 total parameters.

The evolutionary algorithm was applied to the parameters of the network, evolving the best performing solutions based on the total final error in position and later also energy consumption, integrated at each time step. These algorithms are best-fitted to cases where we can execute multiple runs in parallel, since we can quickly evaluate the best solutions, evolve them and select the next generation of solutions. In our case we designed it to run on a single drone, trying to obtain good results in as little iterations as possible. We did so by optimizing the way the evolution occurred: at each step, the best performing parameters were updated by a vector of gaussian white noise of the same dimension of the weights vector, multiplied by a learning step. The new set of weights were then evaluated on a new run. If the overall loss was lower than the lowest loss, we continued to evolve the weights towards the same direction sampled before. Otherwise, the original weights were updated by adding the dame vector, but with half the learning step. Furthermore, in the first phase we used a higher learning step to escape local minima, while we decremented it exponentially as the learning went on.

Algorithm 1 Evolutionary algorithm applied to a neurocontroller

Initialize network weights w_{try}=w_{best}, lowest error err_{low} to inf, $improved = false$, $direction = 0$
for each episode **do**
    $err_{episode}$=0
    Copy best achieving weights w_{try}=w_{best}
    if $improved == True$ **then**
        $improved = False$
        Evolve $w_{try} += direction$
    else
        Sample *direction* from White Noise Gaussian
        Evolve $w_{try} += direction$
    end if
    while Episode is not finished **do**
        $err_{tot} += |err_{position}| + |err_{velocity}|$
    end while
    if $err_{tot} < err_{low}$ **then**
        $err_{low} = err_{tot}$
        $w_{best} = w_{try}$
        $improved = True$

4 Experiments

The network, in parallel with the PD, acts as a non-linear feedback controller, and the function-apprimixator nature of neural networks allows it to create interesting and unpredictable behaviours to optimize the loss function, in our case the trajectory tracking error and the energy consumption. The training lasts for around 50 iterations, that are approximately 30 minutes of real-word training, considering the time needed to perform the trajectory and to come back to the reference initial point. After this time, the overall loss is decreased of around 15-20%. Having more real drones, this time could dramatically decrease, or at the same time cost more efficient solutions could be found. More evident improvements can be seen if we add an external disturbance to our trajectory, a helicoidal ascending trajectory: after half of the total trajectory time, a gust of wind suddenly changes the velocity of the drone, causing it to go off trajectory very rapidly. The neurocontroller finds interesting strategies to overcome this disturbance and minimize the error quickly, while also having a small energy consumption. It is evident from the experiments of these trajectories could not be obtained with a linear controller.

5 Conclusion and further work

In this work we showed how the trajectory tracking precision and energy consumption of a quadrotor with a simplified model can be enhanced by first adding stability terms to the PD controller, and then by putting in parallel a neurocontroller that is trained via evolutionary strategies. The overall loss, computed as the integral of the trajectory error and energy consumption, is decreased by around 15% after half an hour of real world experiments.
An interesting futher work topic is developing local expert networks for different behaviours (ex.: wind gust recovery, high velocity trajectories, error recovery) with evolutionary strategies, and then creating a neural network controller that can learn via imitation learning all these behaviours, thus becoming able to profoundly enhance the performance in a whole variety of situations.

6 Bibliography

R. Mahony, V. Kumar, P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," IEEE Robotics and Automation Magazine, vol.19, no.3, pp. 20-32, Sept. 2012

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, Ilya Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning", arXiv:1703.03864

S. Nolfi, D. Floreano "Evolutionary Robotics", The MIT Press

B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, "Robotics: Modelling, Planning and Control", Springer, 2009

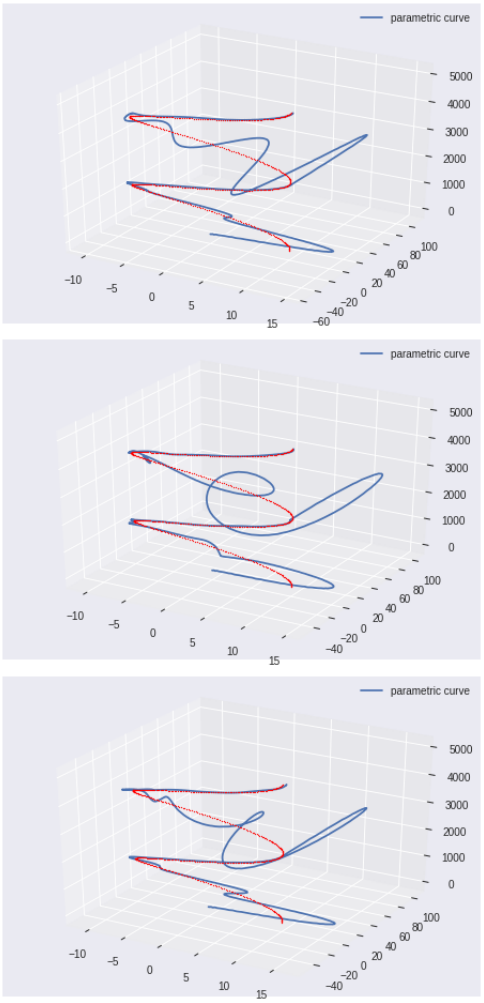

Figure 2: Up: Drone trajectory (blue) and reference trajectory (red) with PD. Notice the sudden gust of wind at half the total run. Middle: Drone trajectory (blue) and reference trajectory (red). Notice the different behaviour that rapidly converges back with a fluent motion. Down: another evolved neurocontroller trajectory