LQR and Output Feedback Control of a quadcopter through a Genetic Algorithm

Michele Viscione, Emanuele Ciccarelli

July 19, 2021

Abstract

The choice for a cost functional for the LQR altitude control problem of an UAV influences heavily the optimal solution found. We will achieve the design of a cost functional with a Genetic Algorithm, thanks to which we could optimize its choice also with respect to new specifications. Finally we will compare this approach with an heuristic choice of the functional and with an Output Feedback Controller, also generated by a Genetic Algorithm.

1 Introduction

In the context of an UAV (specifically, quadcopters) altitude control, the Linear Quadratic Regulator (LQR) control scheme is a good technique which permits us to obtain optimal state feedback controllers for the linearized model, once the cost functional to be minimized is defined. Although optimal, the design of such a functional is not a straightforward task and is often made by heuristics and trial and error procedures. Moreover, the LQR scheme has as intrinsic specific the necessity of a state measure, which is not always possible (even if some state estimates techniques such as Kalman Filter could accomplish that). To deal with this problems we will use a Genetic Algorithm (GA) to strengthen our design of a cost functional which could also satisfies other specifications such as the overshoot or the rise time. Finally, we will compare this Genetic Linear Quadratic Regulator (GLQR) approach with the classical LQR and an Output Feedback Control generated by a GA.

2 Dynamic Model

2.1 Introduction to quadcopters

The use of Unmanned Aerial Vehicles (UAV) has become very popular in the last years. A great contribution of this increase is, for sure, the development and improvement of control systems but, nonetheless their high manoeuvrability, simplicity of construction, low maintenance costs, and low noise. UAVs have become very useful not only for military purposes but in many areas such as; aerial photo and video shooting, farm irrigation and crop monitoring, border patrol and rescue missions, electric power line and gas pipeline monitoring and many others.

Quadcopters are rotary wing machines that make use of four propellers in achieving vertical take-off and landing and other flight manoeuvrings like roll, pitch and yaw. Within UAV hardware, quadcopters are being widely used for different purposes, such as educational, commercial or entertainment. This choice can be justified by the fact that this model presents a very low moment of inertia and six degrees of freedom, which results in great stability of the quadcopter. Moreover, usually quadcopters are small in size and use a variety of sensors to achieve a high level of stability and control, allowing them to navigate even in narrow spaces. Additionally, because each rotor is small, they require less power during flight, which makes quadcopters much safer both to human operators and to the flight environment. Finally, quadcopters are generally low cost and easy to construct.

In the light of this consideration, the purpose of this work is to develop a LQR controller for a quadcopter, using a genetic algorithm in order to tune this kind of controller.

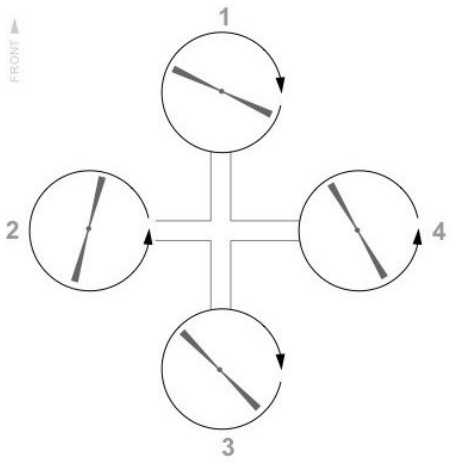

Figure 1: Plant view of the Quadcopter

2.2 Notations

A quadcopter consists of four propellers located orthogonally along the body frame. Figure 1 shows this configuration.

The state of the quadcopter can be represented by its vertical position z, its linear vertical velocity $w = \dot{z}$, and the four angular velocity of the propellers $\Omega_i, i = 1...4$. The input u_i is the voltage V_i applied to the i-th motor. In general a control problem of this kind have 4 inputs, however since we are dealing with an hovering control, each propeller must provides the same thrust, thus we can consider the problem with only one input u which represents the voltage of each motor.

2.3 Vertical attitude equations

This work assumes the physical model presented in [4] as described below. The equations that describe the vertical attitudes related to the Z axis of the aircraft are:

$$\dot{x} = \begin{bmatrix} \dot{z} \\ \dot{w} \\ \dot{\Omega_1} \\ \dot{\Omega_2} \\ \dot{\Omega_3} \\ \dot{\Omega_4} \end{bmatrix} = Ax + Bu \qquad (1)$$
$$y = Cx$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.0106 & 0.0106 & -0.0106 & 0.0106 \\ 0 & 0 & -10 & 0 & 0 & 0 \\ 0 & 0 & 0 & -10 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10 & 0 \\ 0 & 0 & 0 & 0 & 0 & -10 \end{bmatrix}$$
$$B = \begin{bmatrix} 0 & 0 & 1 & -1 & 1 & -1 \end{bmatrix}^T$$
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3 LQR Controller

In LQR designs, the system's performance index is characterised by a cost function (J) for which the controller seeks to minimise. This cost function is given by the formula:

$$J = \int_0^\infty [x^T Q x + u^T R u] dt \qquad (2)$$

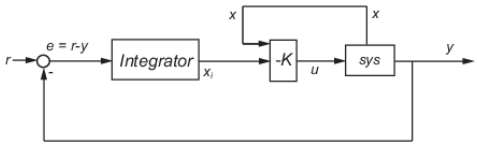

Figure 2: Control scheme

where Q is the state weighting matrix, symmetric and positive semi-definite. R is the control weighting matrix, symmetric and positive definite. These weighting matrices play a paramount role in the determination of the relative importance of the existing error as well as the energy expenditure of the system. Therefore, the parameters of these matrices must be chosen accurately in order to achieve a satisfactory LQR controller.

In literature, there exist several methods for the choice of these matrices. In this work, instead of using a more standard approach, we will use a genetic algorithm to tune the parameters of this LQR.

3.1 Obtaining the feedback gain matrix through Riccati equation

After obtaining the Q and R matrices above, we need to solve the steady-state of the Riccati equation associated with this LQR problem:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{3}$$

Then, the computation of the optimal gains[1] is performed through:

$$K = R^{-1}B^T P \tag{4}$$

Moreover, as a way of dealing with the effect of perturbations and with the steady-state error, an integrator was embedded in the control structure. This inclusion leads to additional robustness of the control system and eliminates the steady-state errors due to the presence of an eventual constant disturbances. The whole control scheme is represented in figure 2. By adding this integrative action we obtain the following new system:

$$\bar{A} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{5}$$

The optimal gains are computed by directly applying the LQR gain computation presented formerly. The optimal gain matrix obtained is then:

$$\bar{K} = \begin{bmatrix} K & -k_1 \end{bmatrix} \tag{6}$$

where K is the vector of gains for the state-variables, obtained from the classical LQR problem and k_1 is the gain of the integrative action. A convenient way for solving this $\bar{K}$ is provided by MATLAB through the following command:

$$\bar{K} = lqi(A, B, Q, R) \tag{7}$$

3.2 Example of "classical" LQR

Before to use the genetic algorithm in order to tuning the LQR, we will briefly see some standard method to choose the matrices Q and R. A classical approach for this problem is for example, according to the Bryson's Rule [5], to choose Q and R diagonal and whose diagonal elements are, respectively, expressed as the reciprocals of the squares of the maximum acceptable values of the state variable (X) and the input control variable (u). Thus we have:

$$Q_{ii} = \frac{1}{maximum\ acceptable\ value\ of\ X_i^2} \tag{8}$$

$$R_{jj} = \frac{1}{maximum\ acceptable\ value\ of\ u_i^2} \tag{9}$$

Moreover, if we are not satisfy from the matrices thus obtained, there are general guidelines that we can follow to obtain the desired values for K [6]:

[1] Notice that this optimal gain consists in a full state-feedback. This requires that the system is observable. If this is not the case, that is a very common situation in real contest, we need something to estimate the state. A good choice for this purpose is the use of an extended kalman filter directly on the nonlinear model

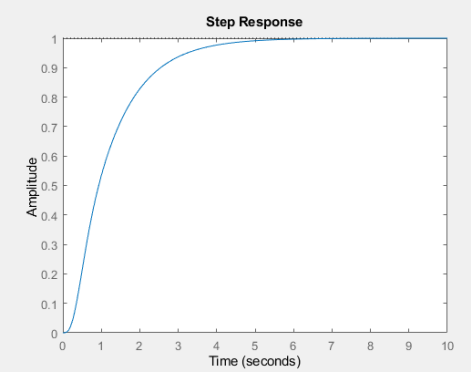

Figure 3: Step response of a LQR controller obtained using the matrices given in [1]

- The larger the values of R, the lower K becomes and the slower the state variables approach zero.
- The lower the values of R, the higher K becomes and the faster the state variables approach zero.
- The larger the values of Q, the higher K becomes and the faster the state variables approach zero.
- The lower the values of Q, the lower K becomes and the slower the state variables approach zero.

For our purpose, we directly take Q and R from [1], indeed the aim of this work is not to derive a classical LQR controller but instead perform an analysis on the difference perforce between the latter and a LQR controller tuned using a genetic algorithm. The matrices that we used are shown below:

$$Q = \begin{bmatrix} 10^7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^7 \end{bmatrix}, R = 1 \tag{10}$$

with these matrices we obtain the step-response shown in Fig. 3.

4 Genetic Algorithm

Although LQR is optimal once Q and R are given, choosing them is not an easy task. In fact, these are based entirely on the trade-off of control effort ($u^T Ru$) and convergence intensity ($x^T Qx$), where the latter do not allow us an immediate verification with respect to specifications such as bandwidth, transient et similia. In this regard, our next goal is to find Q and R that "optimize" the LQR problem with respect to the effort-convergence trade-off or with respect to new specifications as we will see. To do so, we will use a Genetic Algorithm (GA), which is a useful tool for the research of locally almost optimal solutions.

4.1 Introduction to GA and notation

The Genetic Algorithm is a biologically inspired algorithm, where distinct possible solutions are seen as distinct individuals, making up a population. The final solution is obtained through a cyclic process of selection, mating, mutation, and the addition of random genetic material, that is a new possible configuration for a solution. More in detail, we start with the initialization of a population, generating individuals with random characteristics (genes), which identify their solution. Any of these is considered more or less fit based on the evaluation of a fitness function computed over their genes, in such a way that the more a solution (genes) is far from our goal, the more this value is low. At each algorithm step t we select the fittest individuals, those with the higher fit value, to preserve them and add their offspring. The offspring is generated with a mating function

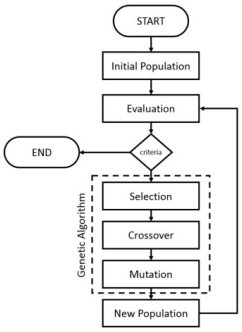

Figure 4: Left (a): After 8000 generations with 10 individuals per population. Right (b): After 16000 generations.

which implement some kind of crossover procedure between the genes of the parents. After this we proceed to apply a mutation to our new population with a certain probability and intensity. Although mutations may seem counterproductive, and often brings to sub-optimal solutions, those are a must for the search of other local optima. As a last step we fill the ranks with new individuals with random genes and start again from the selection phase. We will use the following notation to distinguish between the best solution among our population at time $t = T$, b_T, and the best solution so far b°. In Fig.4 is shown a sketched flux diagram of the algorithm.

4.2 Fitness function

The design of an effective fitness function is a crucial task for the quality of the solutions that we want to find. So, we have to spot first the properties to highlight for a solution, and then weight them in order of importance. For what concern the maximization, in our case it is congenial to design the fitness function to be minimized instead, so that the better is a solution, the lower will be its fit value. This has no impact on the overall architecture of the algorithm. As a first attempt, given the context of a control problem, an example of a fitness function could be:

$$\int_0^T \|r(t) - y(t)\|^2 \, dt$$

where $r(t)$ is the reference signal, $y(t)$ the system output and T a certain time value until which evaluate our controller. This of course does not take in account appropriately some properties such as bandwidth and overshoot. We can then design our fitness function as

$$\int_0^T \frac{At}{t+1} \|e(t)\|^2 \, dt + B \max_{t>s} \|e(t)\| + C\frac{s}{T} \tag{11}$$

where s is the rise time and A, B, C scalar weights. Note that this fitness function also weights differently the tracking error as time goes on. As a further step one might add weights on control effort, phase margin and so on. Of course, as we increase the number of specifications it becomes harder to find "good" solutions, especially if some of them are conflictual.

4.3 Elitism

Elitism is the possibility to be carried on unchanged for the best individual of the population in order to ensure the monotony of the fitness value of the best solution as the algorithm iterate. This may have the counter effect of heighten the possibility of been stuck for many generations in a local optimum, even if not good. In our context this is implemented through the presence of a "resurrected" individual, which may appear with a certain probability (usually 1 in 200 to 1000 generations). Such element is the absolute best solution so far obtained b°. Its primary purpose is not to maintain quality of the solutions but to compare and crossover the genes of different local optima. So, this probability has to be taken such that from one resurrection to another there is a possibility to move from a local to another through mutations, crossover and the new genetic material. This could of course be implemented more systematically by ensuring the "resurrection" at a certain rate of generations.

4.4 Feedback Variance Genetic Algorithm

Sometimes, the various parameters and intervals for the random generation of our genes may be a real guess. For the mutations case, one could resolve applying, with a certain probability, a increment/decrement with variance relative to a percentage of gene value to be altered. So that, for example, we have a mutation with 5% of the value of the gene. Note that a gene is generally not a number, but in our context we can assume that. But what can we say for the random genes generation? This problem implies the knowledge of an interval, at most empiric, for the values of the single genes to be draw from. In some contexts, such as the LQR case, we have not such a clue, so it could be useful to introduce an upgraded scheme of our GA.

Assume the generation of the i-th gene value obtained through a gaussian distribution with variance ${\sigma_i}^2$ and null mean. As the algorithm goes on the absolute best solution b° has i-th gene value $\bar{g_i}$. At this point we can feedback this value as the new standard deviation of the gaussian distribution which will generates the future random i-th genes. In this way, for the empirical rule (68-95-99.7 rule) there is a 68% of probability that the new gene's value will end up in the interval $(-\bar{g_i}, \bar{g_i})$. This means that if the gene value of the best solutions keeps getting bigger, the distribution will automatically grow its variance and so generate more spreaded values, vice versa a shrinking value of the gene will reduce the variance to permit more precision in the interval. This feedback finds its steady state when the distribution generates, almost certainly, values that perform worse than its standard deviation, so when for any algorithm step we have $\sigma_i = \bar{g_i}$. This feedback has of course to be actuated only once a valid b° is found. We name this scheme Feedback Variance Genetic Algorithm (FVGA) to distinguish it from the classic GA.

A similar scheme could be implemented with the use of the best solution for the actual population b_T, but this may stuck our individuals in bad solutions in some circumstances. An effective approach could be the hybrid one, part of the generated individuals (or genes) are subjected to FVGA and other are generate as in the common GA, with the initial guessed interval for the gene value draw.

5 LQR and Output Feedback Control with GA

5.1 Output Feedback Control

Our first goal is to generate a controller capable of regulate the output of our system (1) at a constant reference r through an output feedback, performing so a vertical attitude control. The GA will generate linear controllers with random dimension (mean 2), random structure and parameters. The genes of any individual will be the zeros and poles of this transfer functions. This time, the fitness function we want to minimize will be

$$\int_0^T \frac{At}{t+1} \|e(t)\|^2 \, dt \cdot (W_p(N+1) + W_z(Z+1)) + B \max_{t>s} \|e(t)\| + C\frac{s}{T} \tag{12}$$

where, as only difference from (11), N and Z are the number of poles and zeros and W_p, W_z are their weights. In this way we are endorsing smaller controllers. But, as we can see in Fig.5 (b) and (c), this fitness function do not consider the control effort. In (c), after 40000 generations, the controller (2 poles and zeros) reaches the reference after 0.04 seconds with an overshoot of $3 \cdot 10^{-8}$ meters, completely careless of the actuation effort needed. Another critical problem is the weight balance, for example, if we pick C too big the GA will minimize the rise time and almost ignore the overshoot (Fig.5 (b)). If we add some specifications on the control effort we obtain the controller

$$G(s) = -780.86 \cdot \frac{s}{s+13}$$

which, as we can see in Fig.5 (d), has a satisfying reference tracking. This effort weight has been implemented through a switch state coefficient which activates only after the reaching of a threshold for the control signal.

In this instance, the crossover procedure may consist in inherit the poles from the dominant (most fit) parent and the zeros from the other. If this is not possible due to dimensional problems, the parent roles are swapped. Another possibility is that of randomly pick one by one the offspring's poles and zeros (monomials, binomials, and complex trinomials) from those of the parents.

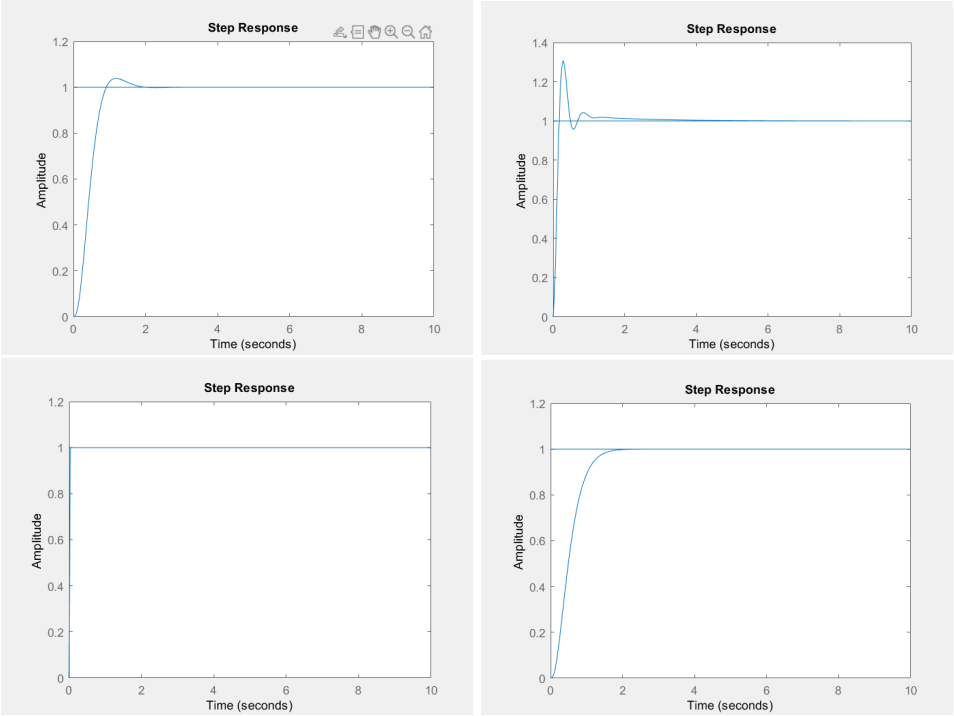

Figure 5: Upper-left (a): 1-pole controller, 670 generations with 10 individuals per population. Upper right (b): step response of a controller made with weight C too big, prominent overshoot. Down-Left (c): control effort not considered, y(t) is almost a rect signal after 40000 generations. Down-right (d): best 1-pole, 1-zero controller, 4360 generations, control effort considered.

5.2 Definition of genes in the LQR context

As for the LQR controller made with GA (GLQR), we have to rephrase our problem. This time, instead of the controller poles and zeros, we could take as our new genes the $\frac{(n+1)(n)}{2}$ parameters of the $(n+1) \times (n+1)$ symmetric Q, n the dimension of (1), and the scalar R value. Such an implementation presents some difficulties, since mutations and random generation of Q and R has to be maid so that $Q \geq 0$ and $R > 0$. This could be easily achieved if one assumes Q as diagonal, as we did, since the problem is reduced to the maintaining of the positivity of $n+2$ scalar values. If this is not the case, we may implement an algorithm to generates valid $Q \geq 0$ matrices (this could be applied in the same way to R when is not scalar). From the Sylvester's criterion we know that the upper-left minors Q_i , $i = 1, ..., n+1$ must be positive definite (or semidefinite if one would accept a singular Q). On the other hand we know that for the block matrix

$$W = \begin{bmatrix} S & P \\ P^T & U \end{bmatrix}$$

the determinant of W can be computed as follows

$$\det(W) = \det(S) \det(U - P^T S^{-1} P) \tag{13}$$

Suppose U as scalar and P a column vector with elements p_i. Inductively, we can prove from (13) that if S is positive definite then W is positive definite if and only if

$$U > \sum_{i}^{n} \sum_{j}^{n} p_i p j S_{ij}^{-1} \tag{14}$$

Such an argument could be extended for induction (or geometrically) in the non scalar U case, or to the semidefinite case with some slight variations.

So, to generate $Q \geq 0$, we could be to pick first a value for Q_{11}, then randomize Q_{12} and bound a new random Q_{22} to be $Q_{22} > \frac{Q_{12}^2}{Q_{11}}$ in order to guarantee the positive (semi)definiteness. Then we will obtain the final Q applying this procedure recursively. As a further step we could also provide some specifications on the non-diagonal values to lighten up some bounds on the diagonal elements.

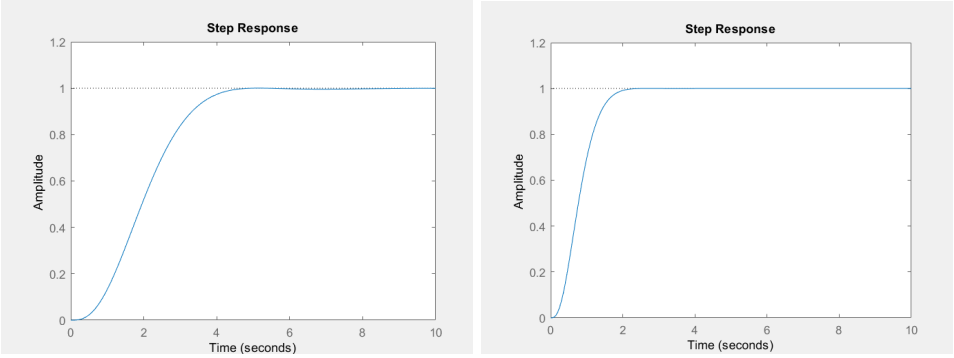

Figure 6: Left (a): After 8000 generations with 10 individuals per population. Right (b): After 16000 generations.

5.3 GLQR

Now that we defined a new genes representation for the LQR case, we are ready to implement the GA. To achieve our goal we will use the FVGA scheme described in section 4.4, since the values for Q and R may vary considerably and we don't want to make the GA solution convergence depends only on our interval guesses. So, initially the GA will generates Q and R based on our guessed intervals, and after a certain number of generations (1000) the FVGA will modify the variance of our distributions to generate more spreaded or precise values according to genes values of the current absolute best solution b°. As for the fitness function, we will use a discretized version of (11) with the addition of a weight for the R value to account the control effort. For what concern crossover, the offspring will inherit randomly the $[Q, R]$ parameters from their parents. In Fig. 6 is shown the results of our simulations, where the solution of (b) has genes:

$$Q = \begin{bmatrix} 390418.04 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 30064.987 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.902951 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.877231 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.150631 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.653851 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2300654.7 \end{bmatrix} \tag{15}$$

$$R = 0.0209932$$

which brings to a state feedback and integral action with

$$\bar{K} = [K, k_1] = \begin{bmatrix} -8649 & -2685 & 3 & -3 & 4 & -3 & 10469 \end{bmatrix} \tag{16}$$

As shown in Fig. 7, relative to the LQR control scheme we used in Simulink.

6 Conclusions

As we have seen so far, LQR provides an optimal state feedback according to the matrices Q and R chosen to tune. However, specifications about performance that the controlled system must have are not directly related to these matrices. Indeed, in the ideal case we have at most some general guidelines already provided in 3.2. A GA is an efficient approach to overcame this drawback, moreover, through a GA we are able not only to choose Q and R in order to met the requirements but we can also obtain more easily better performance with respect to the classical trial-and-error approach. A straight-forward analysis on the different LQR obtained shows the improvements from a classical LQR tuning, showed in 3.2, and LQR tuning using a GA, showed in 5.3.

It is known that the LQR controllers are robust and produce a very low, or null in the LQI case, steady state error, but with a big transition delay and using six feedback gains, that makes them a bad choice whenever the system needs fast parameters update and has no direct access to all states of the plant. If this is the case, the wide range of applications of the GAs could be helpful in the creation of an output feedback control, as shown in 5.1.

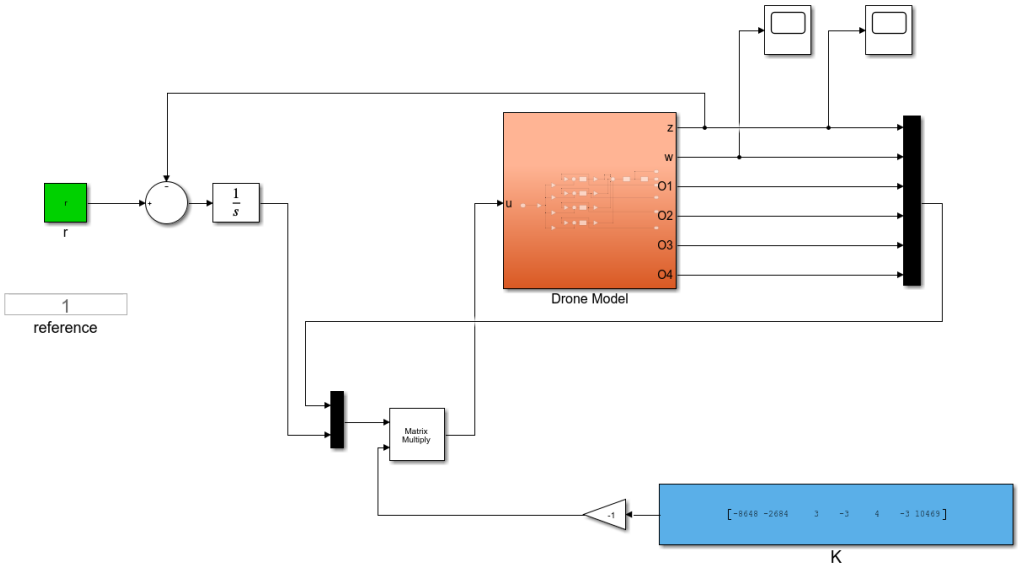

Figure 7: Simulink model of the LQR control.

An interesting feature found during the development of this work is the different time of convergence of the GA toward a satisfactory solution, in the LQR tuning and, respectively, in the output feedback control. Indeed, although in the former we have almost "good"[2] solutions, the convergence is very slow and only after a very large number of generations we have found a satisfactory LQR controller. In the latter instead we lost this property of obtaining "good" solutions at each step but the convergence to a performing one is faster.

References

[1] Lucas M. Argentim, Willian C. Rezende, Paulo E. Santos, Renato A. Aguiar. PID, LQR and LQR-PID on a quadcopter platform, Informatics, Eectronics & Vision (ICIEV), 2013.
[2] Emmanuel Okyere1, Amar Bousbaine1, Gwangtim T. Poyi1, Ajay K. Joseph1, Jose M. Andrade1. LQR controller design for quad-rotor helicopters, The Journal of Engineering 2019.
[3] Luís Martins, Carlos Cardeira1, Paulo Oliveira. Linear Quadratic Regulator for Trajectory Tracking of a Quadrotor, IFAC-PapersOnLine 52(12):176-181, 2019.
[4] T. Jirinec, "Stabilization and control of unmanned quadcopter," Master's thesis, CZECH TECHNICAL UNIVERSITY IN PRAGUE, 2011.
[5] Bryson, A.E.: 'Control of spacecraft and aircraft' (Princeton University Press, Princeton, NJ, 2015)
[6] Rahman, A., Ali, S.M.: 'Design And analysis Of A quadratic optimal control system For A type One plant model', Department of Electronics and Communication Engineering, National University of Singapore, Singapore; School of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore, 2013

[2]good in the sense that even if they do not match the desired requirements they guarantee at least an hovering control