Control of a Robot Differential Platform Using Time Scaling

DANILO RAIRÁN
Universidad Distrital Francisco José de Caldas
Technological Faculty
Carrera 7 No. 40 B - 53
COLOMBIA
drairan@udistrital.edu.co

Abstract: - A research area in control studies the human capabilities to control dynamical systems, because the brain could be the most powerful control centre. However, there are many limitations for the application of a human in the loop. For instance, the time response of a system could be too slow or too fast for a human. Thus, losing attention or not having enough time to make decisions becomes the main challenge in this control field. This paper proposes the use of time scaling plus the learning in a Neural Network to overcome those time constraints. This new control strategy starts by scaling the system in time until a comfortable value for a human, then a Neural Network learns the control actions from the human, and finally that Network runs at different time rates, which will be applied to control a robotic differential platform. The new control procedure improves the control performance carried out by a human by properly changing the time constant of the robot model. We also consider the problem of possible variations of the robot platform after the training stage by using a dynamic version of the back propagation algorithm.

Key-Words: - Time Scaling, Differential Platform, Motion Control, Neural Networks

1 Introduction

Researchers in control have used the concept of time scaling for years. Scaling consists in changing the time variable in such a way that the speed and acceleration profile of a reference signal vary, while its position coordinates remains unchanged, which facilitates manoeuvers to track the reference [1], [2], [3]. A similar application keeps the speed, thus scaling position and acceleration, as in [4]. However, the use of time scaling is not limited to the redefinition of the reference signal, for instance authors in [5] propose to speed up the plant of a system so that its dynamic approaches an integrator in the controller, in which case the control system approximates a simple degree one case. Scaling the reference facilitates the work of a human in the loop, because among many other limits we cannot properly handle changes faster that 0.5 Hz, as stayed in [6] and [7]. Even when that limit may sound low, it is enough to control several applications, especially when the human is part of the system, as in [8], or when the human in the loop has responsibilities only during an initial learning stage, such as the proposals in [9] and [10]. The main contribution in this paper corresponds to the combination of time scaling and learning in the generation of a powerful new tool to control dynamic systems. The learning could be implemented in a number of ways, for instance using Support Vector Machines, but given the popularity of the Neural Networks we use them in this paper. Neural Networks are good candidates to classify, as shown in [11-15], as well as to approximate functions, in [16] and [17], or to predict given time series, as shown in [18], [19] and [20], which correspond to some of the tasks required during the learning stage for the new control strategy.

In brief, control using time scaling requires three steps: Scale, Train, and Run (STR). In the first step, Scale, a person takes the model of a plant and changes the speed of its transient behavior by defining a scaling factor. This stage aims to find the best scaling factor, or alternatively a set of them that makes the control of the system comfortable for a human. Thus, fast systems, such as magnetic levitation reduces its speed until a person properly control it. On the other hand, slow systems, such as temperature control, increases its speed. Thus, passing milliseconds or hours, respectively, into seconds. In the second step, Train, the data captured during the first step serves to train a Neural Network, so that instead of having a human in the loop, the system runs automatically using the Neural Network as the controller. This second step still uses scaled time, and only in the third step, Run, the system runs at the original time rate. Not only in simulations, but also with the real plant. This last step may require changes in the Neural

Network due to differences between the model and the plant in the real world. Given that one of the main contribution of this paper regards the use of time scaling in control, the first five, out of the seven sections, details every aspect of the scaling process. First, showing a general formula to apply time scaling, secondly modeling a differential platform, used as application, thirdly applying the general formula to scale dynamical systems in time, next capturing data using a computer interface, and finally selecting the best scaling factor. Seventh Section regards the training stage, but also include some trials at the original scale of time. Last Section, details the implementation stage, emphasizing on how to handle with differences between model and plant in the real world.

2 Time Scaling Formula

Scaling a linear system comprises adequately changing its poles and zeros location while keeping their angle in the S-plane. Thus, the variation corresponds to a change in the distance from poles and zeros to the origin by a factor called the scaling factor k_t. Given the inverse relationship between the pole location and its respective time constant, $k_t > 1$ implies faster systems, whereas $k_t < 1$ corresponds to slower systems. For instance, given $H = \frac{1}{S^2+2S+2}$, then $DC\ gain = k = \frac{1}{2}$, and poles $p_{1,2} = -1 \pm j$, if $k_t = 2$, then $k' = k = \frac{1}{2}$, $p_{1',2'} = 2 \pm 2j$, thus $H' = \frac{4}{S^2+4S+8}$, where H' correspond to a new system two times faster than H.

The time scaling used in this paper requires an additional condition. The actuating signal, provided by the human (and recorded in a computer,) as well as any other discrete observation of the system, should also use the scaling factor k_t. Thus, a sampling rate, t_s, changes into $t_s' = t_s/k_t$. This last step in the scaling process guarantees having identical time series for original and scaled systems.

The definition of a general formula to scale systems starts with the analysis of a characteristic polynomial, as defined in equation (1), with poles placed at $S = -p_i$, which correspond to a polynomial with real coefficients a_i.

$$(S+p_1)(S+p_2)\cdots(S+p_{n-1})(S+p_n) = S^n + a_1S + \cdots + a_{n-1}S + a_n \quad (1)$$

Time scaling requires scaling poles by a factor k_t, as shown in equation (2).

$$\frac{(S+p_1k_t)(S+p_2k_t)\cdots(S+p_{n-1}k_t)(S+p_nk_t)}{S^n + a_1Sk_t + \cdots + a_{n-1}S{k_t}^{n-1} + a_n{k_t}^n} = (2)$$

The analysis for the numerator of the transfer function starts with the analysis of a constant numerator, as defined in equation (3).

$$H = \frac{b_n}{S^n + a_1S^{n-1} + \cdots + a_{n-1}S + a_n}$$
$$H' = \frac{b_nG_n}{S^n + a_1S^{n-1}k_t + \cdots + a_{n-1}S{k_t}^{n-1} + a_n{k_t}^n} \quad (3)$$

The constant G_n in equation (3) was included to guarantee equal DC gain for H and H', thus $\frac{b_n}{a_n} = \frac{b_nG_n}{a_n{k_t}^n}$, then $G_n = {k_t}^n$, as shown in equation (4).

$$H' = \frac{b_n{k_t}^n}{S^n + a_1S^{n-1}k_t + \cdots + a_{n-1}S{k_t}^{n-1} + a_n{k_t}^n} \quad (4)$$

This new system H' shares both shape and amplitude with H, and the only difference corresponds to the time scale. Now, a numerator of order one, as shown in equation (5).

$$H = \frac{b_{n-1}S}{S^n + a_1S^{n-1} + \cdots + b_1S + b_0} \quad (5)$$

The derivative in the numerator does not allow the comparison made with a constant numerator. The solution consists of integrating H and H', as shown in equation (6) and equation (7).

$$H_1 = H\frac{1}{S} = \frac{b_{n-1}}{S^n + a_1S^{n-1} + \cdots + a_{n-1}S + a_n} \quad (6)$$

The integral for H' is multiplied by a factor k_t in order to make both integrals comparable, and then the equivalent DC gain may be computed.

$$H_1' = \frac{b_{n-1}G_{n-1}k_t}{S^n + a_1S^{n-1}k_t + \cdots + a_{n-1}S{k_t}^{n-1} + a_n{k_t}^n} \quad (7)$$

The constant G_{n-1} in (7) guarantees equal DC gain for H_1 and H_1', thus $\frac{b_{n-1}}{a_n} = \frac{b_{n-1}G_{n-1}k_t}{a_n{k_t}^n}$, then $G_{n-1} = k_t^{n-1}$. Thus:

$$H_1' = \frac{b_{n-1}Sk_t^{n-1}}{S^n + a_1S^{n-1}k_t + \cdots + a_{n-1}S{k_t}^{n-1} + a_n{k_t}^n} \quad (8)$$

The procedure to compute the formula for scaling a linear system in time concludes by following the same procedure for a constant and a derivative, as was already made, but for higher orders in the numerator, as written in equation (9).

$$H' = \frac{b_0 S^n + b_1 S^{n-1} k_t + \cdots + b_{n-1} S k_t^{n-1} + b_n {k_t}^n}{S^n + a_1 S^{n-1} k_t + \cdots + a_{n-1} S {k_t}^{n-1} + a_n {k_t}^n} \quad (9)$$

The formula in equation (9) works well for an arbitrary number of poles of a proper transfer function. For example, if $H = \frac{S+2}{S^3+3S^2}$, and using a scaling factor $k_t = 2$, then $H' = \frac{4S+16}{S^3+6S^2}$.

3 Mathematical Model of the Differential Platform

The definition of the mathematical model for the differential platform DaNI 2.0 begins by analysing its kinematic relations. These relations describe the position by the x and y coordinates, as well as its direction, θ, as a function of time, as indicated in equation (10). These coordinates depend on parameters such as the distance between the wheels, b (which equals 0.37 m), the velocity of the right wheel, v_R, the velocity of the left wheel, v_L, and the coordinates of the initial position, x_0, y_0, θ_0. Values x_0 and y_0 correspond to the initial position in the xy-plane, while θ_0 describes the initial direction.

$$\begin{aligned} \theta &= \theta_0 + \frac{1}{b}\int_0^t (v_R - v_L)dt \\ x &= x_0 + \frac{1}{2}\int_0^t (v_R + v_L)\cos(\theta)\, dt \\ y &= y_0 + \frac{1}{2}\int_0^t (v_R + v_L)\sin(\theta) dt \end{aligned} \quad (10)$$

In addition to the kinematic in equation (10), a more precise model includes the dynamic of the pair motor-wheel. That dynamic relates the linear velocity $v(s)$ in the wheel with the voltage in the motor $Vref(s)$ as shown in equation (11). An identification process looks for a model that matches better the experimental outputs, $v(s)$, given the input data, $Vref(s)$. This identification compresses the dynamic into a linear and second order system, with settling time close to 1.5 s and a damping ratio of 0.95, which means that the system is slightly underdamped.

$$\frac{v(s)}{Vref(s)} = \frac{6.9}{s^2+5.3s+6.9} \quad (11)$$

The simulation of the robot motion uses the block diagram in Fig. 1.a, which includes two main parts: the kinematic and the Right and Left wheels. See that each wheel in that figure has a different set point. That discrepancy causes the robot to rotate, and the proper definition of those references constitutes the main concept behind motion control. Thus, to describe a linear path both references remain equal, to turn left $v_L < v_R$, and to the right $v_L > v_R$.

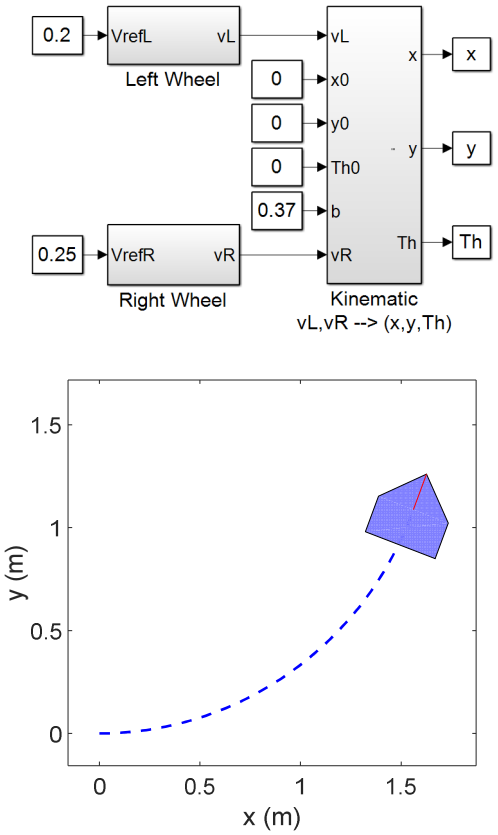

Fig. 1. Model of the robot platform. a) Block diagram of the robot, b) motion simulation.

4 Time Scaling of the Model

The control process using time scaling starts with the modelling stage and continuous by scaling that model in time. This scaling uses the procedure in equation (9) applied to the relations in equations (10) and (11), both in the complex frequency domain. Time scaling changes the transient behaviour of the robot (not its steady state), in such a way that a person leads the robot better than using the original time rate. The steady speed of each wheel remains equal, what vary is the transient duration to go from one speed to another. Thus, the equation (12), which presents the scaling of the dynamic, preserves the DC-gain (steady state) compare with the same gain in equation (11); see procedure in equation (9). To clarify the difference between steady and transient behaviour we present an example: the temperature control of an oven may require certain final temperature (its steady state), while time scaling will define how long a simulation of the process takes to get to that final temperature.

$$\frac{Vel(s)}{V(s)} = \frac{6.9k_t^2}{s^2+5.3sk_t+6.9k_t^2} \quad (12)$$

The kinematic component in equation (10) integrates velocities to obtain positions. Thus, the scaled component, according to the procedure in equation (9), is shown in equation (13).

$$\begin{aligned} \theta(s) &= \frac{1}{b}\frac{k_t}{s}(v_R - v_L) \\ X(s) &= \frac{1}{2}\frac{k_t}{s}(v_R + v_L)\cos(\theta) \\ Y(s) &= \frac{1}{2}\frac{k_t}{s}(v_R + v_L)\sin(\theta) \end{aligned} \tag{13}$$

Time scaling also requires redefining the sampling time t_s. For instance, if a system runs five times slower than normal, $k_t = 1/5$, then the sampling time should also change as follows: $t_s' = t_s/k_t = 5t_s$. This requirement has a direct relation with the work of a neural network, as explained later.

5 Control Interface

An important advantage of time scaling regards the acquisition of the human capability to control, especially when the system is too fast or too slow for a person. This acquisition requires an interface between the person and the scaled system. In this case, that interface presents the robot in an xy-plane, and records the commands from the person through the position of the mouse pointer. The central block of the interface, in Fig. 2.a, contains the code for equations (12) and (13). Its three lower inputs correspond to the initial conditions (x_0, y_0, θ_0), and the other two are the set points for the wheels. On the other hand, the three upper outputs are the robot position (x, y, θ), whereas the remaining outputs present a vector in the direction of the robot motion, θ. This vector helps the person to predict the result of continuing the control action that currently holds through the mouse.

In addition to the central block in the Fig. 2.a there are three blocks. One of them, Real-Time Sync, guarantees running the simulation in real time at a sampling time equal to t_s', as explained in the previous section, where $t_s = 50\,ms$. The second block, Graphical Interface, generates a figure as presented in Fig. 2.b. This figure provides feedback to the person controlling the robot, by presenting the result of the control actions. The third block, Mouse Position, will be explained later.

The left part of Fig. 2.b shows the reference path for the robot, as well as its ideal position at any time, marked with a square. The reference path corresponds to a Lissajous curve as defined in equation (14). This closed curve implies high and low curvature ratios, to the right and to the left, with changing acceleration across the path. Thus, this type of curve presents ideal challenges to test a motion controller.

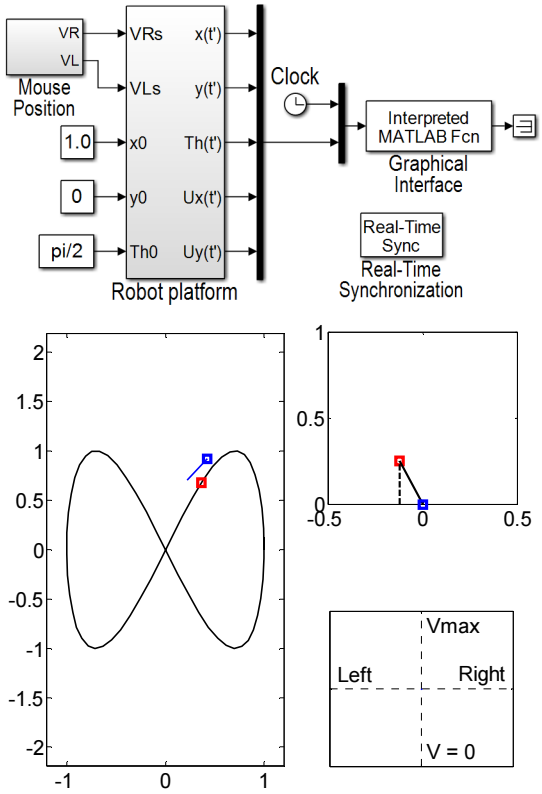

Fig. 2. Control interface. a) Block diagram, b) graphical interface.

$$\begin{aligned} x_r(t) &= \cos(wt) \\ y_r(t) &= sin(2wt) \end{aligned} \tag{14}$$

The definition of the angular frequency in equation (14) starts by considering an average situation where the robot runs at 0.2 m/s during one round, with length approaching 3π m. Thus, and given that $w = 2\pi/(3\pi/0.2)$, then $w = 2/15\ rad/s$. If the final controlled variable were the speed, the value of w would be 2/15, but because it is the position (and time scaling affects it, as shown in equation (13), then $w = \frac{2}{15}k_t$.

The upper right part of the Fig. 2.b presents the motion relative to the robot. Therefore, the square representing the robot keeps still during the simulation, while the square indicating the reference moves at the distance. The ideal control situation requires the robot to stay exactly over the reference, but in a more realistic situation there is always a distance. The final goal of a controller consists in eliminating that distance. Thus, for instance in the scenery presented in the Fig. 2 the robot should turn left, at medium speed to align the robot with the reference.

The control process presented so far requires the definition of the set point for each wheel, but humans make it different. For instance, a person control a vehicle through pedals and a steering wheel, in other words by controlling the linear speed, v (using pedals), and the angular speed, ω (by turning the steering wheel). Thus, the position of the mouse in the Y-axis of the lower right part of the Fig. 2.b indicates the linear velocity, v. Defined from 0 to 0.4 m/s, because $0 \leq Y \leq 1$, according to the expression in equation (15). Similarly, the position over the X-axis represents ω, where $-1 \leq X \leq 1$. The block label Pointer position in Fig. 2.a includes both expressions in the equation (15), where two additional saturation blocks guarantee that the velocities never overcome the natural limits of ± 0.4 m/s.

$$\begin{aligned} v_R &= v + \tfrac{b}{2}\omega = (0{,}2Y + 0{,}2) + \tfrac{b}{2}(-X) \\ v_L &= v - \tfrac{b}{2}\omega = (0{,}2Y + 0{,}2) - \tfrac{b}{2}(-X) \end{aligned} \quad (15)$$

6 Selection of the Scaling Factor

The best scaling factor maximizes the performance of the control process. In other words, minimizes the error, which considers two variables: 1) the distance between robot and reference, R, and 2) the angle between ideal and actual directions for the robot, $\Delta\theta$. By ideal direction, we understand the line of sight between the robot and its reference, θ_r, because that is the shortest distance between them. In addition, (x_r, y_r) in equation (16) corresponds to the coordinates of the reference position.

$$\begin{aligned} R &= \sqrt[2]{(x_r - x)^2 + (y_r - y)^2} \\ \theta_r &= tg^{-1}\left({}^{y_r - y}/_{x_r - x}\right) \\ \Delta\theta &= \theta_r - \theta \end{aligned} \quad (16)$$

The final definition of error comprises the effect of distance and direction in a single index, as shown in equation (17). Those two variables, different in nature, can be combined given the similarity in amplitudes for the curve used as reference. In this index, tf corresponds to the simulation time, which equals $60\pi/k_t$ given that the trajectory covers four turns around the reference path. Thus, and in order to make the errors at different scaling factors comparable, the constant k_t multiplies both integrals.

$$Error = \left(\int_0^{tf} R dt + \int_0^{tf} |\Delta\theta| dt\right) kt \quad (17)$$

The selection of the best scaling factor starts by running a simulation of the motion control at $k_t = 1$. This experiment shows that it is possible to control the robot, but we expected to have a better performance using lower scaling factors, because the person would have more time to react to the changes in the reference path. Then, the next test sets k_t in ½. That change decreases the Error index, as expected. Thus, the search continuous looking around $kt =$ ½, as follows $k_t = \frac{1}{4}, \frac{1}{3}, \frac{5}{12}, \frac{1}{2}, \frac{7}{12}, \frac{2}{3}, \frac{3}{4}$. Values under ¼ makes tf longer than 12 minutes, which elevates the Error index, because the attention decrease. On the other hand, k_t does not pass ¾ because it is evident that greater factors increase the Error index.

Now, with the rank of k_t defined, the procedure to capture data used 20 simulations per factor from two participants: 10 simulations per person, however only the best 5 out the 20 are shown in Fig. 3. This reduction looks for consistency in the data, avoiding possible variations during the learning stage. Each box in Fig. 3 shows the maximum, minimum, median, and first and third quartiles per scaling factor. Thus, and looking at the lowest median, the best performance happens at $k_t = 5/12$. On the other hand, the global minimum is somewhere between $k_t = 1/3$ and $k_t =$ ½. Factors at the left of $1/4$ have large errors, given the attention problems caused by long simulations. Similarly, factors at the right of $2/3$ also increases the Error index due to the relation between the reaction time of a person and the speed of the system.

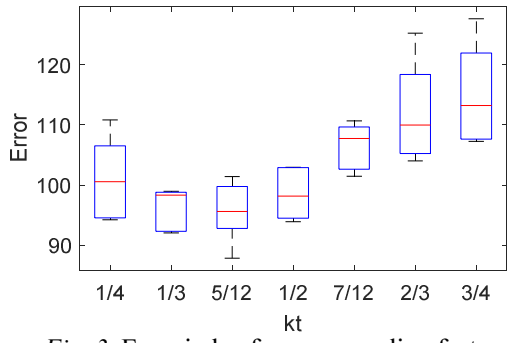

Fig. 3. Error index for seven scaling factors.

Both participants produce simulations with similar Error indexes, however they follow different control strategies. One of them jumps from maximum to null velocity continuously, whereas the other generates smooth curves (see v_R and v_L in Fig. 4), as a traditional control algorithm does. Thus, future work may study how personal differences ends in different control strategies, as well as the determination of the number of those control styles. Finally, we should remark that prolonging the trial stage, beyond 10 simulations, might result in decreasing the Error in Fig. 3, due to the training, as happens in gamming

where the player becomes an expert, but then expertise would play a bigger role than the effect of the scaling factor.

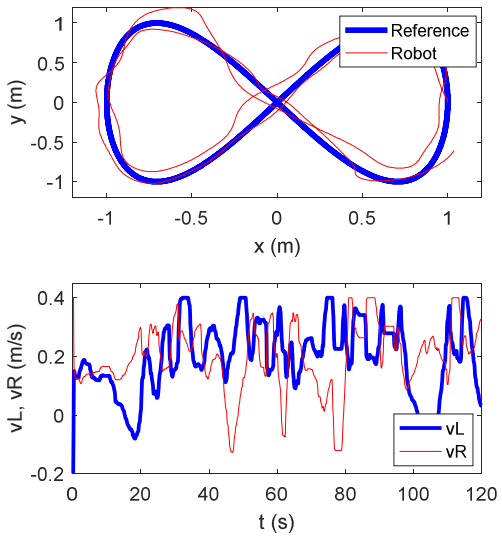

Fig. 4. Two turns of the human control of the scaled systems at $k_t = 5/12$. a) Robot motion and reference path, b) velocities.

7 Control Using Neural Networks

An algorithm replaces the human in this new step of the control process using time scaling. That algorithm should 1) learn what the human does to control the robot, and 2) perform that control at different scaling factors. A traditional candidate for the learning stage is an Artificial Neural Network. On the other hand, and concerning the second constraint, handling different factors implies having the same outputs at the same inputs, regardless the sampling time, and some neural network architectures do precisely that. For instance, a feedforward multilayer perceptron processes input data (passing it through weights, transfer functions, and sums) almost instantly, due to the short time to compute those operations. In practice, an output instantly follows a corresponding input. Thus, we use a feedforward multilayer perceptron to replace the human, because that network fulfils both requirements to automate the control process.

A person should control two variables during each simulation: the linear and the angular velocities, which carry out by properly locating the mouse pointer in the control interface. However, in the case of a vehicle, the driver may in addition listen to the radio, talk, and many other things at the same time, which show the computational power of a human brain. In contrast, we should divide the control task in two parts aiming for small and thus fast enough networks. Hence, a network, NNR, controls the linear velocity, while other, $NN\theta$, controls the angular velocity, as shown in Fig. 5. The input data for NNR is the radius, R, given the proportionality between the linear velocity and the distance to a target. Additionally, the input of $NN\theta$ is $\Delta\theta$, due to the relation between angular velocity and changes in direction.

Those two networks use data from the experiment with best performance ($k_t = 5/12$), such as robot position $\langle x, y, \theta \rangle$, reference position $\langle x_r, y_r \rangle$, and set-points $\langle v_R, v_L \rangle$. These data cover 4 turns around the reference path (about 940 samples), as shown in Fig. 4, and serve to compute the input of the networks (R and $\Delta\theta$), as well as their targets (v, ω). Computing those inputs and outputs requires the expressions in equations (16) and (18), respectively, were definitions for v and ω comes from the connections in Fig. 5.

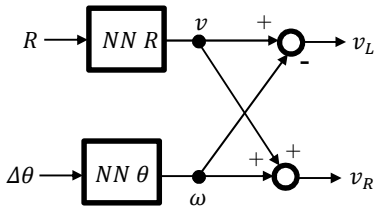

Fig. 5. Networks configuration to control the robot.

$$v = \frac{v_L + v_R}{2}$$
$$\omega = \frac{v_L - v_R}{2} \tag{18}$$

Each network has two inputs, p_1 and p_2, as shown in Fig. 6. The first input, p_1, corresponds to the current value: $R(k)$ or $\Delta\theta(k)$, whereas the second one, p_2, corresponds to the difference between $p_1(k)$ and its value two steps back, $p_1(k-2)$, it is $R(k) - R(k-2)$ or $\Delta\theta(k) - \Delta\theta(k-2)$. The first input serves as an indication of how much effort the control system should make to move the robot to its ideal position ($R \to 0, \Delta\theta \to 0$), whereas the second input shows how fast the robot has changed its position, which may be associated with a basic form of prediction. Those inputs pass through a single hidden layer with two neurons and hyperbolic tangents as transfer functions, as shown in Fig. 6, given that this structure works as a universal approximator. The training of the two networks, NNR and $NN\theta$, uses the algorithm called Levenberg-Marquardt backpropagation, and data divided in three subsets: 70% to train, 15% to validate, and 15% to test. Thus, the best network comes from training 20 networks and choosing the network with the lowest mean square error (mse). Table 1 reports the weights for the best networks.

Now, the already trained networks replace the human in the control loop. Those networks receive data from the simulated robot (R or $\Delta\theta$) and come up with appropriate set points for the wheels (v_L and v_R), aiming to reduce the linear and angular distance to the reference path. First, at the scale factor $k_t = 5/12$, and later at $k_t = 1$. The first test with this new and automatic control system consists in following the same reference path used to get the training data. If this test shows at least the same human performance, the control system is ready for experiments at different paths and speeds, as well as the unitary scaling factor. Changing the scale factor leaves the weights and structure of the networks untouched, and only requires the variation of the discrete blocks $1/z$ (in the case of the Simulink application), which allow the reading of the inputs only every sampling time. Thus, that delay goes from $t_s' = 120\ ms$ to $t_s = 50\ ms$.

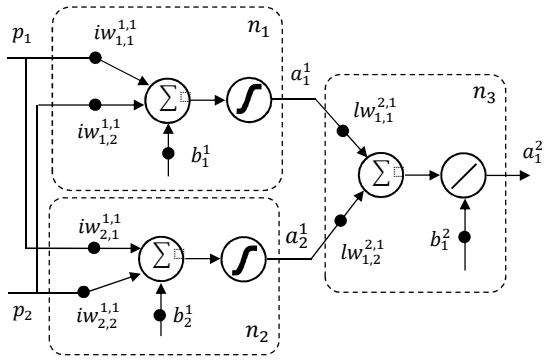

Fig. 6. Feedforward multilayer perceptron structure.

Table 1. Final weights for NNR and $NN\theta$.

NNR		
Hidden layer, n1	Hidden layer, n2	Output layer, n3
$iw_{1,1}^{1,1} = -2.139$	$iw_{2,1}^{1,1} = 2.381$	$lw_{1,1}^{2,1} = -0.712$
$iw_{1,2}^{1,1} = 0.782$	$iw_{2,2}^{1,1} = 6.999$	$lw_{1,2}^{2,1} = -0.721$
$b_1^1 = -0.138$	$b_2^1 = -6.161$	$b_1^2 = -0.537$
$NN\theta$		
$iw_{1,1}^{1,1} = 1.766$	$iw_{2,1}^{1,1} = -0.165$	$lw_{1,1}^{2,1} = 0.371$
$iw_{1,2}^{1,1} = 4.018$	$iw_{2,2}^{1,1} = 2.358$	$lw_{1,2}^{2,1} = -1.319$
$b_1^1 = -0.312$	$b_2^1 = 0.423$	$b_1^2 = 0.074$

A possible variation of the reference path to test the control system implies changing y_r in equation 5. For instance, from two times w to one or three times. If it is one time, the robot follows a circular path, as shown in the upper part of Fig. 7, at a speed defined by the value of w. The maximum speed (experimentally defined) permits a single turn in a minute, with some oscillations at the beginning of the motion caused by the initial position of the robot. On the other hand, if $y_r = sin(3wt)$ the path has three internal loops, then the robot cannot go at the same speed as before, but completes a whole turn in 90 seconds, as shown in the lower part of Fig. 7. It is important to emphasize that the networks were trained at $t_s' = 120\ ms$, and now they run at $t_s = 50\ ms$; nevertheless, they adequately control the robot platform. The simulations of the control system with these new paths show that the neural networks control the robot at new scenarios and at different scaling factors. The remaining question is how a real platform will react to the new controller, as shown in the next section.

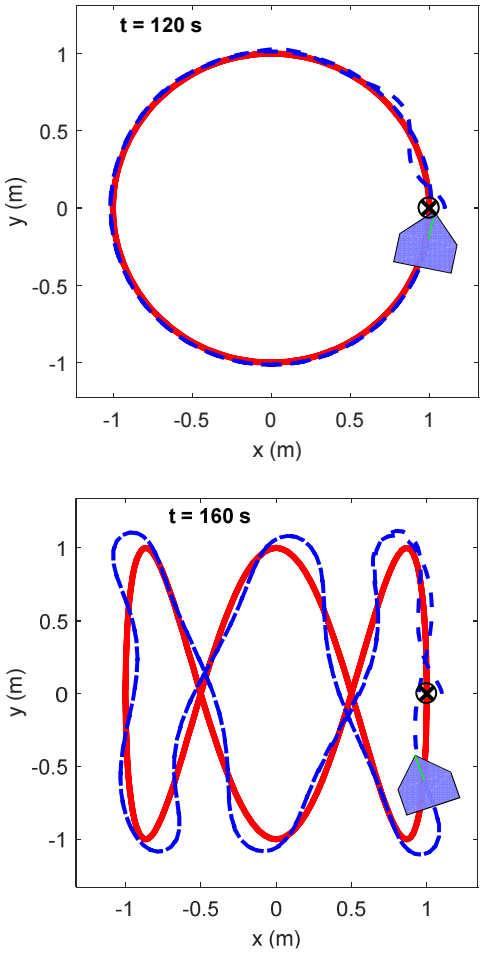

Fig. 7. Performance of the system at $k_t = 1$, $x_r = cos(wt)$ a) $y_r = sin(wt)$, b) $y_r = sin(3wt)$.

8 Final Implementation

Any difference between the behaviour of the model and the plant may lead to losses of the controller performance, because the control procedure highly depends on the quality of the model. Thus, this section proposes a solution for this problem. Instead of leaving the controller fixed, a learning algorithm could continuously change the weights of the neural networks to adapt, not only to the environment, but also to changes in the platform

itself. The selected learning algorithm, in [21], defines an updating rule for the output layer, and another for the hidden layer, which permits the definition of different learning rates for those layers.

Learning comes up as a solution for the mismatch between model and platform, but it also could move the control system into instability. Thus, in addition to define the learning rates for each layer, we should define when to learn. For instance, the experimentation process shows that activating the learning for the radius network first and then for the angle network presents better results than a simultaneous approach. Results in Fig. 8 corresponds to the learning from zero to two minutes for the radius, and from two to eight minutes for the angle.

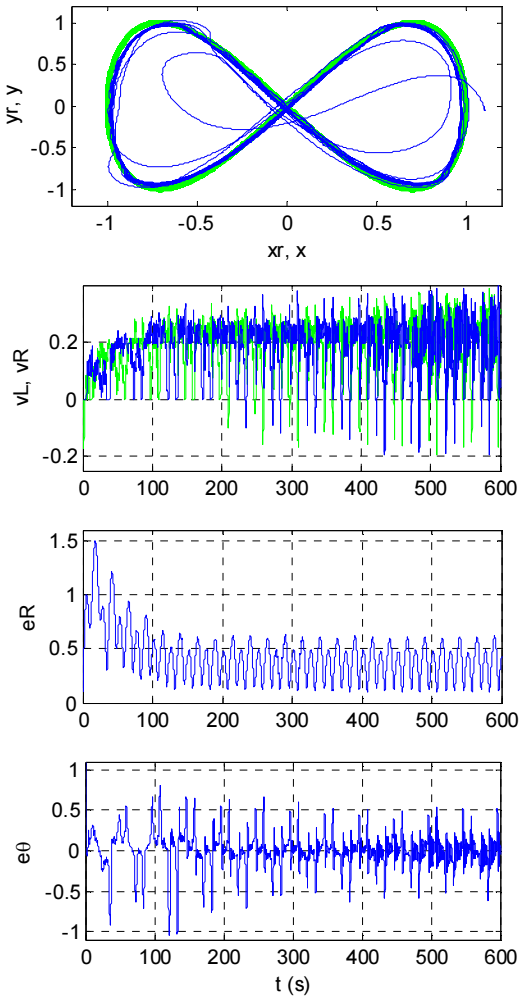

Fig. 8. Application of the learning algorithm for radius and angle networks.

Initial error for radius and angle seems too large in Fig. 8 compare with errors after the learning stage, but this is only because one of the weights for each neuron was deliberately set to zero, as shown in Table 2. This change aims for evaluating the learning process, emulating a drastic change or discrepancy between simulation and final implementation. On the other hand, see that weights in the output layer changes more than in the hidden layer, which may mean two things: 1) weights in the hidden layers are closer to local minima, or 2) the learning rate affect differently each layer, because the learning rate for the hidden layers was 0.01, while 0.001 for the output layer.

Table 2. Changes in the weights values due to the learning stage.

NNR		
Hidden layer, n1	Hidden layer, n2	Output layer, n3
$iw_{1,1}^{1,1}$ $-2.139 \rightarrow -2.232$	$iw_{2,1}^{1,1}$ $2.381 \rightarrow 2.380$	$lw_{1,1}^{2,1}$ $-0.712 \rightarrow -0.864$
$iw_{1,2}^{1,1}$ $0.782 \rightarrow 0.782$	$iw_{2,2}^{1,1}$ $6.999 \rightarrow 6.999$	$lw_{1,2}^{2,1}$ $\mathbf{0 \rightarrow -0.164}$
b_1^1 $-0.138 \rightarrow -0.309$	b_2^1 $-6.161 \rightarrow -6.161$	b_1^2 $-0.537 \rightarrow -0.372$
$NN\theta$		
$iw_{1,1}^{1,1}$ $1.766 \rightarrow 2.426$	$iw_{2,1}^{1,1}$ $-0.165 \rightarrow -0.228$	$lw_{1,1}^{2,1}$ $0.371 \rightarrow 0.768$
$iw_{1,2}^{1,1}$ $4.018 \rightarrow 3.999$	$iw_{2,2}^{1,1}$ $2.358 \rightarrow 2.358$	$lw_{1,2}^{2,1}$ $\mathbf{0 \rightarrow -0.054}$
b_1^1 $-0.312 \rightarrow -0.005$	b_2^1 $0.423 \rightarrow 0.416$	b_1^2 $0.074 \rightarrow 0.039$

9 Conclusions

This paper presented the control process of a robotic platform based on time scaling. This process changes the time constants of a robot model, given a scaling factor, facilitating the control actions of a person over the scaled system. Once the person controls the system, two neural networks replace the person and run at the original time rate. Selecting the best scaling factor implies maximizing the performance of the control system, using several trials with different persons at variable transient behaviours. In this case, the best factor slows the transients to about half the normal speed. The data collected at that factor serve to train the neural networks that replaces the person, which is tested by trying paths different from the training path. On the other hand, the differences between model and real platform may cause losses in the control performance, thus in the last section, an online training algorithm continuously tunes the neural networks to accommodate them to the real plant, preserving or even improving the performance of the control system with a human in the loop. Finally, we consider that is important to test more plants to show the advantage of this new control process, also to enhance the control interface to facilitate the control actions, for instance looking for a more realistic

scenario using virtual reality, instead of presenting the robot as a single square.

References:

[1] A. K. Singh, K. M. Krishna and S. Saripalli, "Planning trajectories on uneven terrain using optimization and non-linear time scaling techniques," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, 2012, pp. 3538-3545. doi: 10.1109/IROS.2012.6385662

[2] V. Duggal *et al*., "Overtaking maneuvers by non linear time scaling over reduced set of learned motion primitives," *2015 IEEE Intelligent Vehicles Symposium (IV)*, Seoul, 2015, pp. 115-120. doi: 10.1109/IVS.2015.7225672

[3] Y. Hu, G. Yan, Z. Lin and M. Liu, "Time-scaling control and passive walking of bipeds with underactuation degree one," *49th IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, 2010, pp. 1116-1121. doi: 10.1109/CDC.2010.5717814

[4] N. Motoi, T. Shimono and R. Kubo, "Point-to-point motion control based on reproduction of recorded human motions with time scaling," *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, Dallas, TX, 2014, pp. 2834-2839. doi: 10.1109/IECON.2014.7048910

[5] L. Hsu, T. R. Oliveira and J. P. V. S. Cunha, "Extremum seeking control via monitoring function and time-scaling for plants of arbitrary relative degree," *2014 13th International Workshop on Variable Structure Systems (VSS)*, Nantes, 2014, pp. 1-6. doi: 10.1109/VSS.2014.6881131

[6] S. A. S. Mousavi, X. Zhang, T. M. Seigler and J. B. Hoagg, "Characteristics that make dynamic systems difficult for a human to control," *2016 American Control Conference (ACC)*, Boston, MA, 2016, pp. 4391-4396. doi: 10.1109/ACC.2016.7525613

[7] B. Yu, R. B. Gillespie, J. S. Freudenberg and J. A. Cook, "Human control strategies in pursuit tracking with a disturbance input," *53rd IEEE Conference on Decision and Control*, Los Angeles, CA, 2014, pp. 3795-3800. doi: 10.1109/CDC.2014.7039980

[8] M. Corno, P. Giani, M. Tanelli and S. M. Savaresi, "Human-in-the-Loop Bicycle Control via Active Heart Rate Regulation," in *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 1029-1040, May 2015. doi: 10.1109/TCST.2014.2360912

[9] L. Peternel, E. Oztop and J. Babič, "A shared control method for online human-in-the-loop robot learning based on Locally Weighted Regression," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, 2016, pp. 3900-3906. doi: 10.1109/IROS.2016.7759574

[10] R. B. Warrier and S. Devasia, "Iterative Learning From Novice Human Demonstrations for Output Tracking," in *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 4, pp. 510-521, Aug. 2016. doi: 10.1109/THMS.2016.2545243

[11] F. Xie, H. Fan, Y. Li, Z. Jiang, R. Meng and A. Bovik, "Melanoma Classification on Dermoscopy Images Using a Neural Network Ensemble Model," in *IEEE Transactions on Medical Imaging*, vol. 36, no. 3, pp. 849-858, March 2017. doi: 10.1109/TMI.2016.2633551

[12] A. Kumar, J. Kim, D. Lyndon, M. Fulham and D. Feng, "An Ensemble of Fine-Tuned Convolutional Neural Networks for Medical Image Classification," in *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 1, pp. 31-40, Jan. 2017. doi: 10.1109/JBHI.2016.2635663

[13] X. Y. Zhang, G. S. Xie, C. L. Liu and Y. Bengio, "End-to-End Online Writer Identification With Recurrent Neural Network," in *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 2, pp. 285-292, April 2017. doi: 10.1109/THMS.2016.2634921

[14] H. Liu; N. Shu; Q. Tang; W. Zhang, "Computational Model Based on Neural Network of Visual Cortex for Human Action Recognition," in *IEEE Transactions on Neural Networks and Learning Systems* , vol. PP, no. 99, pp. 1-14 doi: 10.1109/TNNLS.2017.2669522

[15] T. E. Chen *et al*., "S1 and S2 Heart Sound Recognition Using Deep Neural Networks," in *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 2, pp. 372-380, Feb. 2017. doi: 10.1109/TBME.2016.2559800

[16] P. Andras, "High-Dimensional Function Approximation With Neural Networks for Large Volumes of Data," in *IEEE Transactions on Neural Networks and Learning Systems* , vol. PP, no. 99, pp.1-9 doi: 10.1109/TNNLS.2017.2651985

[17] H. Liu, Y. Zhang and W. Wu, "Saturated adaptive back-stepping control for robot

manipulators with RBF neural network approximation," *2016 IEEE International Conference on Information and Automation (ICIA)*, Ningbo, 2016, pp. 1550-1555. doi: 10.1109/ICInfA.2016.7832065

[18] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer and K. Funaya, "Robust Online Time Series Prediction with Recurrent Neural Networks," *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Montreal, QC, 2016, pp. 816-825. doi: 10.1109/DSAA.2016.92

[19] S. Hussein, R. Chandra and A. Sharma, "Multi-step-ahead chaotic time series prediction using coevolutionary recurrent neural networks," *2016 IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC, 2016, pp. 3084-3091. doi: 10.1109/CEC.2016.7744179

[20] C. Zhao, M. van Heeswijk and J. Karhunen, "Air quality forecasting using neural networks," *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Athens, 2016, pp. 1-7. doi: 10.1109/SSCI.2016.7850128

[21] S.G. Kadwane, Amit kumar, B.M. Karan, T. Ghose "Online Trained Simulation and DSP Implementation of Dynamic Back Propagation Neural Network for Buck converter," *ACSE Journal*, vol. 6, no. 1, pp. 27-34, Jan. 2006.