NON-LINEAR MODEL PREDICTIVE CONTROLLER FOR TRAJECTORY TRACKING OF AN OMNI-DIRECTIONAL ROBOT USING A SIMPLIFIED MODEL

J. Rodrigo Ferreira * **A. Paulo Moreira** **

* *University of Porto, Faculty of Engineering - DEEC, Porto*
** *University of Porto, Faculty of Engineering - DEEC, and INESC - Porto*

Abstract: In this paper we propose a non-linear model predictive controller architecture (NMPC) for trajectory tracking of an omni-directional mobile robot. The controller employs a simplified process model to predict the evolution of the state of the robot which allows for real-time minimization of the cost function using gradient descent methods. The cost function is chosen so as to penalize the position and orientation errors, as well as the variation of the control effort. Simulation tests are performed under different controller settings in order to evaluate the performance of system. The results are presented at the end of the paper.

Keywords: trajectory tracking, model predictive control, omni-directional mobile robots, gradient descent methods, process simulation

1. INTRODUCTION

Given the applications in which mobile robots are used nowadays, the demand for higher performance trajectory tracking controllers is steadily increasing. Be it military situations, surveillance or recognition missions, or even robotic soccer, these controllers are required to operate correctly at high speeds, providing precise and stable tracking of complicated trajectories in sometimes very dynamic environments. In robotic soccer, for example, each robot is required to track, at high speeds, trajectories with sudden changes in direction and orientation, in a playing field crowded with other robots. Classical control strategies are clearly not sufficient.

Trajectory tracking controllers may be divided into two major categories: reactive controllers and predictive controllers. Reactive controllers typically rely on structures with feedback loops for the state of the robot, calculating the control signals based on the error between the current state and the desired one. This means that corrective measures are taken only after the error has occurred, which is unsuitable for tracking trajectories at high velocities. Predictive controllers on the other hand use future information (both by taking advantage of having knowledge about the whole trajectory and by predicting the evolution of the state of the robot based on its model) to optimize the control signals. This allows action to be taken before the errors occur, thus making predictive control much more suitable for high speed trajectory tracking. However, the complex models used for the prediction might cause the controllers to become too computationally intensive to be useful for real time control. On the other hand, the use of simplified models might cause the predictions to become too inaccurate rendering them useless for control at high speeds. Interesting implementations of predictive controllers are presented in (Li *et al.*, 2001) or (Gu and Hu, 2000). The prime reference for the controller here designed is (Conceição *et al.*, 2008).

The controller presented in this paper was developed as part of a larger project which required high-performance trajectory tracking of a single robot

[0] Work supported by the Portuguese Science and Technology Foundation (FCT) under the project "Perception-Driven Coordinated Multi-Robot Motion Control" (PTDC/EEA-CRO/100692/2008).

for use in mobile robot formations in robotic soccer. Due to their ability to incorporating information about the trajectory into the calculation of the control signals, model predictive control techniques were used ((Camacho and Bordons, 2004), (Findeisen and Allgöwer, 2002)). A simplified process model was used, approximating the response of each motor by a first order system with an invariant step response. The steepest descent method was used to minimize the cost function and calculate the desired control signals.

This paper is organized in 5 sections. Section 1 provides an introduction to the problem and a brief presentation of the state of the art. In section 2 the robot used is described and modelled. The controller architecture is detailed in section 3 and test results are presented and discussed in section 4. Finally, conclusions on the work done are drawn in section 5.

2. THE ROBOT

As testbed for the controller, a three wheeled omnidirectional robot was used. This particular robot is part of the *5DPO* robotic soccer team from the *Faculty of Engineering of the University of Porto*. It employs a mix of a computer based vision system and odometry to track its position in the world, the control being provided by a laptop connected by a *RS232* link to the motor drivers. Each motor driver has its own PID controller to track a speed reference. The laptop runs the *Ubuntu 9.10* operating system and all the control software is programmed in the *Lazarus/FPC* language. The control loop runs with a 40 ms period, synchronized with the arrival of a new frame from the camera (running at 25 FPS). A schematic of the robot is presented in figure 1.

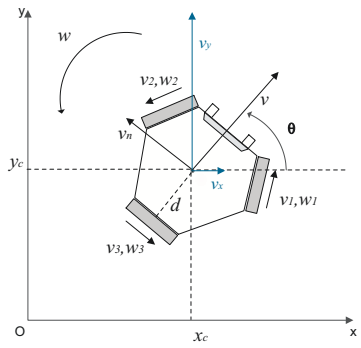


Fig. 1. Schematic of the three wheeled omnidirectional robot used

Let $X(t) = [x_c(t)\ \ y_c(t)\ \ \theta(t)]^T$ be the state of the robot, with $x_c(t)$ and $y_c(t)$ being its position and $\theta(t)$ its orientation in relation to the X axis of the world frame. Let also $V_R(t) = [v(t)\ \ v_n(t)\ \ w(t)]^T$ and $V_{XoY}(t) = [v_x(t)\ \ v_y(t)\ \ w(t)]^T$ be the speed vectors of the robot, in coordinates of the robot and the world frames, respectively. $v(t)$, $v_n(t)$, $v_x(t)$, and $v_y(t)$ are the linear speeds and $w(t)$ the angular speed. The vector $V_w(t) = [v_1(t)\ \ v_2(t)\ \ v_3(t)]^T$ contains the linear speeds of each wheel.

The equations for direct and inverse kinematics (that is, to obtain $V_w(t)$ from $V_R(t)$ and vice-versa) can be found in (Helder P. Oliveira and Costa, 2009). The speed values can be switched between world-frame coordinates and robot-frame coordinates using equation 1.

$$\begin{bmatrix} v(t) \\ v_n(t) \\ w(t) \end{bmatrix} = \begin{bmatrix} cos(\theta(t)) & sin(\theta(t)) & 0 \\ -sin(\theta(t)) & cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_x(t) \\ v_y(t) \\ w(t) \end{bmatrix} \quad (1)$$

2.1 *Evolution of the state of the robot*

Considering a simplified version of the system with a sampling time of T the state of the robot at instant $k+1$ can be determined from its speed and state at instant k by applying equation 2, where $ct = cos(\theta(k))$ and $st = sin(\theta(k))$.

$$X(k+1) = X(k) + T \cdot \begin{bmatrix} ct & -st & 0 \\ st & ct & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v(t) \\ v_n(t) \\ w(t) \end{bmatrix} \quad (2)$$

3. CONTROLLER ARCHITECTURE

The basic idea behind the NMPC controller presented in this paper is quite an intuitive one, following closely the classic MPC approach. Using a mathematical model of the robot, the evolution of its state (position and orientation) is predicted for the so called prediction horizon (T_p), using different sets of control inputs. For each set of control inputs a cost value is calculated for the simulation. The cost function penalizes differences in position and orientation of the robot in relation to the reference, as well as the variation of the control effort. The steepest descent gradient method is then used to optimize the cost function, calculating the control inputs that minimize its value for the desired control horizon (T_u). The first value of the calculated control inputs vector is applied to the robot and the process is repeated in the next cycle. The structure of the controller is presented in figure 2. R is the global reference trajectory, $|V|$ the desired speed module, U the control inputs and X the state of the robot. Also, R_{ref} represents the reference trajectory for the controller (see section 3.2) and X_{sim} the simulated state of the robot.

3.1 *Cost Function*

This being a trajectory tracking problem, the cost function naturally penalizes differences between the

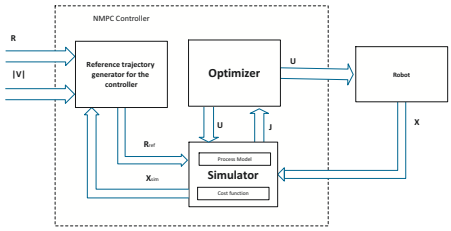


Fig. 2. Structure of the Predictive Controller

position and orientation of the robot and those of the reference trajectory, for each instant of the prediction horizon T_p (here used as its discreet equivalent, N_p). Thus, the cost function is defined, in discrete time, in equation 3 as:

$$J(N_1, N_2, N_c) = \sum_{i=N_1}^{N_2} (\lambda_1([x_{sim}(i) - x_{ref}(i)]^2 + [y_{sim}(i) - y_{ref}(i)]^2)) + \sum_{i=N_1}^{N_2} (\lambda_2[\theta_{sim}(i) - \theta_{ref}(i)]^2) + \sum_{i=1}^{N_c} \lambda_3(\Delta U(i)) \quad (3)$$

- N_1, N_2 - prediction horizon limits, in discreet time, so that $N_1 > 0$ e $N_2 \leq N_p$.
- N_c - control horizon.
- λ_1, λ_2, λ_3 - weights for each component of the cost function
- $x_{sim}(k)$, $y_{sim}(k)$, $\theta_{sim}(k)$ - predicted position and orientation of the robot at instant i.
- $x_{ref}(k)$, $y_{ref}(k)$, $\theta_{ref}(k)$ reference position and orientation values for the robot at instant i .
- $\Delta U(k) = [v_r(k) - v_r(k-1)]^2 + [vn_r(k) - vn_r(k-1)]^2 + [w_r(k) - w_r(k-1)]^2$ - variation of the control signals, with $U(i)$ being the reference velocities vector.

3.2 *Reference Trajectory*

The global reference trajectory is simply an ordered set of points mapped in the XoY frame, with an orientation reference associated to each of these points. It contains no time parametrization whatsoever, so each time the cost function is calculated a reference trajectory for the desired prediction horizon must be created, taking into account the desired speed and the global reference trajectory. Figure 3 depicts the process of obtaining this reference trajectory.

The first point of the new trajectory R_{ref} is the point from the segments that compose the global reference trajectory R that's closest to the robot. The remaining points are calculated as having a distance equal to

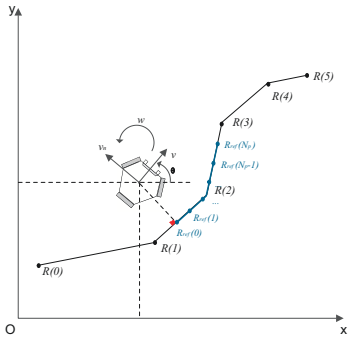


Fig. 3. Obtaining the reference trajectory for the controller

$|V_{ref}| * T$ from the previous point, marked over the main reference trajectory. $|V_{ref}|$ is the module of the desired speed and T the period of the control algorithm, here 40 ms. For each point of the new trajectory, the θ reference is linearly interpolated between the θ references of the global reference trajectory.

3.3 *Process Model and Simulation*

The initial process model was a purely cinematic model of the robot, containing no dynamics. Elements where then progressively added to the model, in an attempt to improve performance of the controller by making the model closer to reality.

First the saturation effect of the motor speeds was added to the model. Using the process described in (Conceição *et al.*, 2006) a function was added to detect wheel speed reference saturation and scale the remaining motor speeds in order to maintain movement direction.

Then the step response of each motor plus *PID* controller set was approximated by a first order LTI (Linear Time Invariant) system. Let $U(t)$ be the speed reference of the motor and $Y(t)$ the actual speed. Applying the Laplace transform, the transfer function of the first order system will be defined as $G(s) = \frac{Y(s)}{U(s)} = \frac{K}{s+r}$, where K is the DC gain of the system and $r = 1/\tau$, τ being the time constant of the system. This type of LTI system has an exponential response to a step input. An initial value of τ was estimated by plotting the step response of the motors in simulation. This time constant was later fine-tuned by testing several values and analysing the controller performance for each. The results found in (Neuman and Baradello, 1979) were used to implement a digital transfer function to calculate this response.

The final model of the process consists of a cinematic model with added motor speed saturation and first order dynamics for motor speed step response. Given this model the simulation proceeds as follows. The linear speed references $V_R(t)$ are transformed in wheel speed references V_w using inverse kinematics. The wheel speed saturation detection is applied to

these wheel speed reference values, scaling them if needed. Then, for each motor the real motor speeds $V_{w.Real}$ are predicted according to the model. Using the kinematics equations the real motor speeds are turned back into real linear speeds $V_{R.Real}$. Finally, equation 2 is used to calculate the new state of the robot. This process is repeated for every time step - with different control signals if $Nu > 1$ - for which the simulation is desired.

3.4 *Minimization of the Cost Function*

Because the model is non-linear, iterative numerical methods were used to minimize the cost function J. The steepest descent method (presented in (Fletcher, 2001)), described in algorithm 1 to minimize $f(x)$, was adapted and used to minimize J in relation to U. From the minimization of the cost function results the optimal control input.

Algorithm 1 - Steepest Descent	
1.	Given x_0, set $k = 0$
2.	Compute $d_k = -\nabla f(x_k)$, where ∇f denotes the gradient of f.
3.	If $d_k < \epsilon$ stop. Else, find the new point $x_{k+1} = x_k + \alpha \cdot d_k$.
4.	Increment $k = k + 1$, go to step 2.

In the algorithm definition, α is the size of the step in the direction of the travel, x_k and x_{k+1} are the variable values in k and $k + 1$, and ϵ is the stop criterion.

4. EXPERIMENTAL RESULTS

All of the results presented in this paper were obtained from simulation tests performed with the SimTwo software (Costa, 2010), using a very precise model of the robot determined previously in (Caldas, 2009). SimTwo uses the ODE physics simulation engine (Smith, 2010), which enables a very precise simulation of rigid body dynamics. Due to the quality of the model used, the simulation exhibits a high degree of realism and may be considered a very good approximation of the real robot's behaviour.

4.1 *Quality Measures*

Three quality measures were defined to qualify each test result: *Maximum Overshoot* (maximum overshoot the robot's trajectory exhibits when performing the corners), *Maximum Settling Time* (the maximum time the robot takes to get back within 5 cm of the reference trajectory after overshooting), and *Total Quadratic Error* (sum of the squares of the differences between the robot's position and the reference trajectory at every instant). Initial tests have shown that the *Total Quadratic Error* is not a good quality measure, as sometimes results with large overshoots exhibited the same values for this measure as results with much smaller overshoots, due to the behaviour during the rest of the trajectory. Therefore, *Maximum Overshoot* and *Maximum Settling Time* are considered the primary quality measures.

4.2 *Test Trajectory*

In order to simulate common situations in robotic soccer, a test trajectory containing abrupt changes in direction and orientation is needed. Taking this into account, a "pulse" type trajectory was used, containing different reference orientations for each segment. The trajectory is defined in table 1 and can be seen in the XY plots in section 4.4.

index (i)	x_{ref} (m)	y_{ref} (m)	$\theta_{ref\ segment\ i\ to\ i+1}$ (rad)
1	0	0	0
2	3	0	$\pi/2$
3	3	3	0
4	6	3	$-\pi/2$
5	6	0	0
6	9	0	-

Table 1. Parameters for the reference trajectory used in the controller tests

4.3 *Optimizer Parameters*

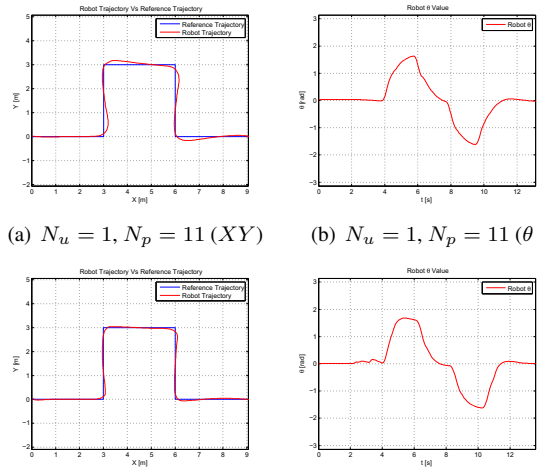
The time constraints on the execution of the control algorithm are rather severe, so the minimization algorithm to minimize J must be parametrized accordingly. Namely, a correct iteration limit IT_{max} and stop criterion ϵ must be determined so that the control loop is executed in under 15-20 ms, given the desired N_p and N_u values. The algorithm execution stops and the best values obtained so far are considered optimal when either the iteration count goes over IT_{max} iterations or the cost value goes under ϵ.

Several tests of the optimization algorithm were conducted using $\epsilon = 0$ and $IT_{max} = \infty$, with $N_u = 2$ and $N_p = 10$. Analysis of the logs of the optimization algorithm have shown that for values of $J < 0.1$ the changes in the control inputs are not very significant. Therefore, a slightly lower value of $\epsilon = 0.05$ was defined as a stop criterion. Choosing IT_{max} is a matter of balancing algorithm execution time and optimizer precision. When the robot has been following a straight line for a while, the optimization usually converges in around 4 or 5 iterations (taking around 3 *ms* of processing time on a laptop running with an Intel Core 2 Duo processor at 2GHz). When a corner is performed and the robot overshoots, the cost values rise abruptly and the cost function cannot be minimized in useful time. A value of $IT_{max} = 30$ was chosen because it allowed for a certain degree of optimization of the control signals in these situations while keeping the algorithm execution time within the defined limits.

4.4 *Effect of Controller Parameters*

In order to find the effect on performance of the controller parameters, tests were conducted for values of N_p ranging from 8 to 12 and values of N_u ranging from 1 to 3. The optimizer parameters were the ones described in 4.3 and the weights used for the cost function were $\lambda_1 = 2$, $\lambda_2 = 1$, $\lambda_3 = 0$. The reference speed was $2m/s$.

Results for *Maximum Overshoot* and *Maximum Settling Time* are presented in tables 2 and 3. Several plots are also presented in figure 4 in order do illustrate the most pertinent results.



(a) $N_u = 1$, $N_p = 11$ (XY) (b) $N_u = 1$, $N_p = 11$ $(\theta$

(c) $N_u = 2$, $N_p = 10$ (XY) (d) $N_u = 2$, $N_p = 10$ (θ)

Fig. 4. Controller test results at $V_{ref} = 2m/s$

Maximum Overshoot [m] Vs N_p, N_u					
N_u	N_p				
	8	9	10	11	12
1	0.3720	0.3400	0.2153	0.1958	0.2000
2	0.2474	0.1857	0.0966	0.1006	0.0770
3	0.2749	0.1406	0.0703	0.0791	0.0733

Table 2. Maximum Overshoot Values

Maximum Settling Time [s] Vs N_p, N_u					
N_u	N_p				
	8	9	10	11	12
1	1.2000	1.0800	1.0400	1.2800	1.3600
2	0.8000	0.9600	0.7600	1.1600	0.7200
3	1.0400	0.6800	1.2400	1.1200	0.9200

Table 3. Maximum Settling Time Values

The average and maximum execution times of the control algorithm for each of the test parameters are also presented in table 4.

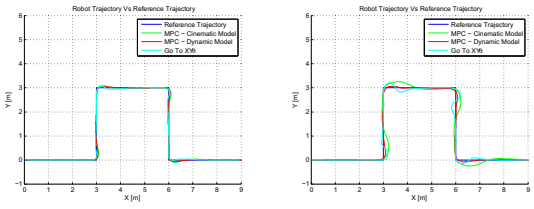
Let us consider the *Maximum Overshoot* as the main quality measure. The results clearly show that, in a general way, the performance of the controller improves with the increase of N_u. However, each unitary increase in N_u results in a large increase in computation time for the control algorithm, due to the way the *Steepest Descent* algorithm works. Because the control loop time is limited to $40ms$ this sometimes results in loss of synchronism and degradation of performance. Also, the improvement when switching from $N_u = 1$ to $N_u = 2$ is bigger than when switching from $N_u = 2$ to $N_u = 3$. The performance improvement at $N_u = 3$, when compared to $N_u = 2$, does not make up for the increase in execution time. The effect of N_p is more complex. Values much lower than 10 result in large overshoots due to the corners not being predicted soon enough. With values much higher than 10 the robot tends to miss the control points and do the corners "on the inside". There is also an effect on computational time of the algorithm, albeit not as severe as the one caused by N_u. In general terms, the computational time also rises with the increase of N_p, as expected. However, for values of $N_p = 8$, the average computational time of the algorithm appears larger than for $N_p = 9$. This is probably due to the *Steepest Descent* algorithm needing, on average, more iterations to converge when really short prediction horizons are used. The ideal balance between performance and acceptable computational time appears to be $N_u = 2$ and $N_p = 10$.

Max. and Avg. Control Loop Execution Time [s] Vs N_p, N_u					
N_u	N_p				
	8	9	10	11	12
1	7.41 16	7.28 15	7.38 16	7.87 17	9.29 18
2	13.10 23	12.280 24	13.30 25	13.60 26	16.77 29
3	18.63 30	18.09 34	17.78 34	21.14 35	23.54 36

Table 4. Maximum and Average Control Loop Execution Times

4.5 *Effect of the Process Model*

The tests in figure 5 were aimed at comparing the performance of the present controller with an identical predictive controller using a purely cinematic model and with the reactive controller currently used in the 5DPO robotic soccer team.



(a) $V_{ref} = 1m/s$ (b) $V_{ref} = 1.5m/s$

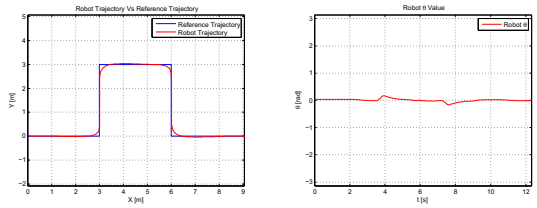
Fig. 5. Controller performance comparison tests

The results clearly show the advantage of predictive control over purely reactive one. Whereas the reactive controller causes severe overshoots an undershoots at sharp corners, its performance severely degrading with the increase in reference speed, the predictive controller results in much smoother trajectories, with smaller overshoots. The reactive controller performs

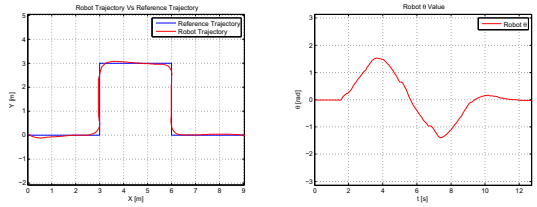
a go to XY operation to some points ahead in the trajectory and not to the next immediate point, which accounts for the fact that it starts turning slightly before the corner. Also depicted is the importance of correctly simulating the dynamics of the robot. At low speeds, the controller with the purely cinematic model presents a performance similar to that of the one with the simplified dynamic model, with only a slightly larger overshoot. When the reference velocity is increased, the performance of the controller with the purely cinematic model drops significantly below that of the other. The higher the reference speed desired, the more important it is to correctly simulate the dynamics of the robot.

4.6 *Trajectories with smooth and fixed* θ_{ref}

Finally, results are shown in figure 6 for the tracking of trajectories with either fixed θ_{ref} or smooth variations of θ_{ref} along the segments. The same trajectory as previously was used, but the θ references have been changed to the two situations described.



(a) XY plot ($\theta_{ref} = 0$) (b) θ plot ($\theta_{ref} = 0$)



(c) XY plot (smooth θ_{ref}) (d) θ plot (smooth θ_{ref})

Fig. 6. Results for trajectories with smooth and fixed θ_{ref}, $V_{ref} = 2m/s$

With a fixed value of θ_{ref} the trajectory is tracked with very low overshoot. As for the trajectory with a smooth θ_{ref} variation, some errors are discernible but not very significant. For both situations, the performance of the controller is significantly better than in trajectories with abrupt θ_{ref} variations at corners. These results also attest to the quality of the controller.

5. CONCLUSION

A strategy for trajectory tracking of an omni-directional mobile robot with a non-linear model predictive controller using iterative numeric methods for minimization of the cost function was proposed. The main difference from the works upon which this controller is based on lies on the use of the simplified process model, allowing for real-time execution of the algorithm on rather modest computers using large (> 10) prediction horizons. Both the parameters of the controller (N_u, N_p) and the optimizer (IT_{max}, ϵ, α) have a high impact on the performance and should be tuned to the trajectories and robot at hand. The importance of correct simulation of the robot's dynamics as opposed to simply using a cinematic model has been demonstrated. The higher the reference speed desired, the more important it is to correctly simulate dynamics. The experimental results obtained attest to the quality of the controller when precisely tuned to the robot used. This performance is even better when the reference trajectories have a fixed θ reference or this value is allowed to vary smoothly along the trajectory.

6. REFERENCES

Caldas, Renato Miguel dos Santos (2009). Modelação e Simulação de um Robot Omnidireccional de 3 rodas. Msc thesis.

Camacho, E. F. and C. Bordons (2004). *Model Predictive Control*. second ed.. Springer.

Conceição, André Scolari, A. Paulo Moreira and Paulo J. Costa (2006). Trajectory tracking for omni-directional mobile robots based on restrictions of the motor's velocities. *Department of Electrical and Computer Enginnering, University of Porto - Porto - Portugal*.

Conceição, André Scolari, A. Paulo Moreira and Paulo J. Costa (2008). A nonlinear model predictive control strategy for trajectory tracking of a four-wheeled omnidirectional mobile robot. *Optim. Control Appl. Meth. 2008; 335-352*.

Costa, Paulo J. (2010). Simtwo. Available from http://paginas.fe.up.pt/ paco/wiki/index.php?n= Main.SimTwo.

Findeisen, Rolf and Frank Allgöwer (2002). An introduction to nonlinear model predictive control. *21st Benelux Meeting on Systems and Control*.

Fletcher, R. (2001). *Practical Methods of Optimization*. second ed.. Wiley.

Gu, Dongbing and Huosheng Hu (2000). Wavelet neural network based predictive control for mobile robots. Vol. 5. pp. 3544 –3549 vol.5.

Helder P. Oliveira, Armando J. Sousa, A. Paulo Moreira and Paulo J. Costa (2009). Modeling and assessing of omni-directional robots with three and four wheels. In: *Challenges and Solutions, A D Rodi (Ed.), ISBN: 978-953-307-038-4, INTECH*.

Li, T.-H.S., Sheng-Sung Jian and Ming-Che Tsai (2001). Design and implementation of fuzzy trajectory following and planning control for mobile robots. Vol. 1. pp. 453 –458 vol.1.

Neuman, C. P. and C. S. Baradello (1979). Digital transfer functions for microcomputer control. *Systems, Man and Cybernetics, IEEE Transactions on* **9**(12), 856 –860.

Smith, Russel (2010). Open dynamics engine. Available from http://www.ode.org.