Preprints of the 9th International Symposium on Robot Control (SYROCO'09)
The International Federation of Automatic Control
Nagaragawa Convention Center, Gifu, Japan, September 9-12, 2009

A Library Tailored for Real-Time Implementation of Model Predictive Control

Marcos Rosendo Dalte * **Walter Fetter Lages** ** **Jorge Augusto Vasconcelos Alves** ***

* *Electrical Engineering Department, Federal University of Rio Grande do Sul, Porto Alegre, RS 90035-190 BRAZIL (e-mail: marcosdalte@ece.ufrgs.br)*
** *Electrical Engineering Department, Federal University of Rio Grande do Sul, Porto Alegre, RS 90035-190 BRAZIL (e-mail: w.fetter@ieee.org)*
*** *Electrical Engineering Department, Federal University of Rio Grande do Sul, Porto Alegre, RS 90035-190 BRAZIL (e-mail: jorge@ece.ufrgs.br)*

Abstract: This paper presents a software library developed to implement model predictive control (MPC) strategies in real-time. By using MPC, an appropriate optimal control law is implicitly obtained. Furthermore, the system physical constraints on state and inputs are dealt with in a straightforward way. The library supports MPC for linear, nonlinear and linearized systems. For linear systems, the optimization problem is solved by using quadratic programming, while for the nonlinear case, sequential quadratic programming (SQP) is used. The linearized version enables the use of the faster quadratic programming algorithm for the nonlinear case. Experimental results show that the control signal computation can effectively be performed under the real-time requirements.

Keywords: Model Predictive Control, Mobile Robot, Brachiating Robot, Software Library, Nonlinear control.

1. INTRODUCTION

Real-time control of mobile robots is a challenging proposition, even for simple robots. Despite the apparent simplicity of the kinematic model of a mobile robot, the design of stabilizing control laws for those systems is a considerable challenge. Due to Brockett conditions (Brockett, 1982), a continuously differentiable, smooth stabilizing feedback control law can not be obtained. To overcome these limitations non-smooth and time-varying control laws have been proposed (Bloch and McClamroch, 1989; Samson and Ait-Abderrahim, 1991; Canudas de Wit and Sørdalen, 1992; McCloskey and Murray, 1997). Recent works dealing with robust and adaptive control of mobile robots can be found in Oya et al. (2003) and Dixon et al. (2004).

However, in realistic implementations it is difficult to obtain the desired performance, due to constraints on inputs or states that naturally arise. Usually, the controller is designed under the assumption that there are no limitations on inputs or state and, at most, the actual restrictions are taken into account afterwards. By using model predictive control (MPC), restrictions can be handled in a straightforward way while generating the control signal. For a mobile robot this is an important issue, since the position of the robot can be restricted to belong to a safe region of operation and control actions that respect actuators limits can be generated.

The use of MPC has also been proposed for more complex robots, such as brachiating robots (Fukuda et al., 1991), which is an underactuated system, i. e., the robot has two degree of freedom and only one control input, so it can move only dynamically. From the point of view of the control theory, underactuated mechanical systems present second order non-integrable differential constraints, commonly referred to as nonholonomic second order systems (Oriolo and Nakamura, 1991; Luca and Iannitti, 2002). In de Oliveira and Lages (2007) it is shown that MPC can effectively deal with a system with such properties.

In this paper, we propose a strategy to overcome some of the problems related to the use of model predictive control for mobile robots. A sequential quadratic programming (SQP) algorithm was chosen due to its ability of reaching a minimum with few evaluations of the problem-related functions (Barclay et al., 1997).

In order to enable the use of the controller implementation developed in this work for other robots and dynamic systems in general, the MPC algorithm was implemented as a general purpose C++ class library. Hence, by redefining the object that represents the system model, it is possible to use the very same software for applying MPC control to any system that can be described in space state.

⋆ The authors would like to thank to Coordenação de Aperfeiçoamento do Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Companhia Estadual de Energia Elétrica (CEEE) for their financial support.

2. MODEL PREDICTIVE CONTROL

Model predictive control is an optimal control strategy that uses the model of the system to obtain an optimal control sequence by minimizing an objective function. At each sampling interval, the model is used to predict the behavior of the system over a prediction horizon. Based on these predictions, an objective function is minimized with respect to the future sequence of inputs, thus requiring the solution of a constrained optimization problem for each sampling interval.

Although prediction and optimization are performed over a future horizon, only the values of the inputs for the current sampling interval are used and the same procedure is repeated at the next sampling time. This mechanism is known as *moving or receding horizon* strategy, in reference to the way in which the time window shifts forward from one sampling time to the next one.

In this section we present the equations of the nonlinear model predictive control (NMPC) and then a variation called linearized model predictive control (LMPC), that is useful for some classes of nonlinear systems that can be linearized in each sampling time. The linearization of the system enables the use of a quadratic programming optimization, which is a convex problem and thus much more efficient than a generic constrained optimization used for nonlinear systems.

2.1 NMPC Problem Formulation

The basic elements present in all model-based predictive controller are: prediction model, objective function, calculation of the control action. The prediction model is the central part of the MPC, because it is important to predict the future outputs of the system. In this scheme, the state space model is used as prediction model, but in different MPC schemes, other models could be used Camacho and Bordons (1999). The objective function defines the criteria to be optimized in order to force the generation of a control sequence that drives the system as desired.

Consider a general nonlinear model, expressed as:

$$\dot{x}(t) = f(x(t), u(t)) \tag{1}$$

where $x(t)$ is the state vector and $u(t)$ is the control input vector. The same nonlinear model, now described in discrete time, is given by:

$$x(k+1) = f(x(k), u(k)) \tag{2}$$

The objective function to be minimized assumes, in general, the following form:

$$\begin{aligned}\Phi(k) = &\sum_{j=1}^{N} x^T(k+j|k)Qx(k+j|k)\\ &+\sum_{j=1}^{N} u^T(k+j-1|k)Ru(k+j-1|k)\end{aligned} \tag{3}$$

where N is the prediction horizon, which here is the same as the control horizon and $Q \geq 0$ and $R \geq 0$ are weighting matrices that penalize the state error and the control effort, respectively.

By considering the fact that every real system is in practice subjected to some constraint (for example physical limits), we define the following general constraint expressions:

$$\begin{aligned} x(k+j|k) &\in \mathcal{X}, \quad j \in [1, N]\\ u(k+j|k) &\in \mathcal{U}, \quad j \in [0, N]\end{aligned}$$

where $\mathcal{X}$ is the closed and convex set of all possible values for x and $\mathcal{U}$ is the closed and convex set for all possible values for u. By supposing that such constraints are linear with respect to x and u, we can write:

$$Cx(k+j|k) \leq c, \quad j \in [1, N] \tag{4}$$

$$Du(k+j|k) \leq d, \quad j \in [0, N] \tag{5}$$

Thus, the optimization problem, to be solved at each sample time k, is to find a control sequence $u^\star$ and a state sequence $x^\star$ such that minimize the objective function $\Phi(k)$ under imposed constraints, that is:

$$u^*, x^* = arg\ \min_{u,x} \{\Phi(k)\} \tag{6}$$

subjected to:

$$x(k|k) = x_0 \tag{7}$$

$$x(k+j|k) = f(x(k+j-1|k), u(k+j-1|k)),\quad j \in [1, N] \tag{8}$$

$$Cx(k+j|k) \leq c, \quad j \in [1, N] \tag{9}$$

$$Du(k+j|k) \leq d, \quad j \in [0, N] \tag{10}$$

where x_0 is the value of x in instant k.

The problem of minimizing (6) is solved for each sampling time, resulting in the optimal control sequence:

$$u^* = \{u^*(k|k), u^*(k+1|k), \ldots, u^*(k+N|k)\} \tag{11}$$

and the optimal state sequence is given by:

$$x^* = \{x^*(k|k), x^*(k+1|k), \ldots, x^*(k+N|k)\} \tag{12}$$

with an optimal cost $\Phi^*(k)$. Thus, the control law defined by NMPC is implicitly given by the first term of the optimal control sequence:

$$h(\delta) = u^*(k|k) \tag{13}$$

where $h(\delta)$ is a continuous variable whose value is held constant during the sampling interval T.

Model predictive control is based on the assumption that for a small time horizon plant and model behavior are the same. For this assumption to hold the plant/model mismatch should be kept small. Obviously, for any real world plant, control inputs are subject to physical limitations. Hence, to avoid large plant/model mismatch those limitations should be considered while computing control inputs. This can be done in a straightforward way by defining upper and lower bounds on the control input

3. LMPC PROBLEM FORMULATION

Although many NMPC techniques has been proposed in the literature Chen and Allgöwer (1998); Allgöwer et al. (1999), it should be noticed that the computational effort necessary in this case is much higher than in the linear version. In NMPC there is a nonlinear programming problem to be solved on-line, which is nonconvex, has a larger number of decision variables and a global minimum is in general impossible to find Henson (1998). In this section, a strategy to reduce the computational burden is proposed. The fundamental idea consists in using a successive linearization approach, yielding a linear, time-varying description of the system. Then, considering the control inputs as the decision variables, it is possible to transform the optimization problem to be solved at each sampling time in a quadratic programming problem. Since they are convex, quadratic programming problems can be easily solved by numerically robust algorithms which lead to global optimal solutions.

A linear model of the system dynamics can be obtained by computing an error model with respect to a pre-specified reference trajectory, x_r. It is usual in this case to associate to this reference trajectory a *virtual* reference robot, as shown in Fig. 1, which has the same model than the robot to be controlled. Thus, we have:

$$\dot{x_r} = f(x_r, u_r), \tag{14}$$

or, in discrete-time, $x_r(k+1) = f_d(x_r(k), u_r(k))$.

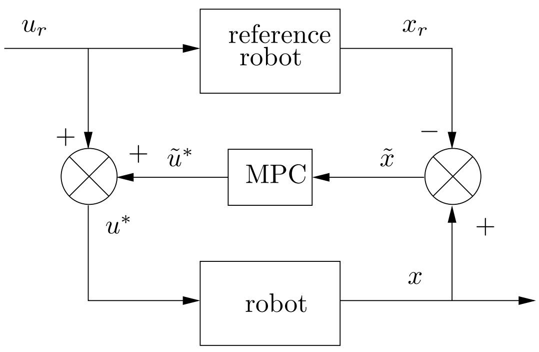

Fig. 1. Block diagram of the linearized model predictive control.

By expanding the right side of (1) in Taylor series around the point (x_r, u_r) and discarding the high order terms it follows that:

$$\dot{x} = f(x_r, u_r) + \left.\frac{\partial f(x,u)}{\partial x}\right|_{\substack{x=x_r\\u=u_r}} (x - x_r) + \\ + \left.\frac{\partial f(x,u)}{\partial u}\right|_{\substack{x=x_r\\u=u_r}} (u - u_r),$$

or,

$$\dot{x} = f(x_r, u_r) + f_x(x - x_r) + f_u(u - u_r), \tag{15}$$

where f_x and f_u are the jacobians of f with respect to x and u, respectively, evaluated around the reference point (x_r, u_r). Then, the subtraction of (14) from (15) results in:

$$\dot{\tilde{x}} = f_x\tilde{x} + f_u\tilde{u}, \tag{16}$$

where, $\tilde{x} = x - x_r$ represents the error with respect to the reference car and $\tilde{u} = u - u_r$ is its associated error control input.

The approximation of $\dot{x}$ by using forward differences gives the following discrete-time system model:

$$\tilde{x}(k+1) = A(k)\tilde{x}(k) + B(k)\tilde{u}(k), \tag{17}$$

with

$$A(k) = \begin{bmatrix} 1 & 0 & -v_r(k)\sin\theta_r(k)T \\ 0 & 1 & v_r(k)\cos\theta_r(k)T \\ 0 & 0 & 1 \end{bmatrix}$$

$$B(k) = \begin{bmatrix} \cos\theta_r(k)T & 0 \\ \sin\theta_r(k)T & 0 \\ 0 & T \end{bmatrix}$$

$$\bar{A}(k) = \begin{bmatrix} A(k|k) \\ A(k+1|k)A(k|k) \\ \vdots \\ \alpha(k,2,0) \\ \alpha(k,1,0) \end{bmatrix} \tag{18}$$

$$\bar{B}(k) = \begin{bmatrix} B(k|k) & 0 & \cdots & 0 \\ A(k+1|k)B(k|k) & B(k+1|k) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha(k,2,1)B(k|k) & \alpha(k,2,2)B(k+1|k) & \cdots & 0 \\ \alpha(k,1,1)B(k|k) & \alpha(k,1,2)B(k+1|k) & \cdots & B(k+N-1|k) \end{bmatrix} \tag{19}$$

Now it is possible to recast the optimization problem in an usual quadratic programming form. Hence, we introduce the following vectors:

$$\bar{x}(k+1) = \begin{bmatrix} \tilde{x}(k+1|k) \\ \tilde{x}(k+2|k) \\ \vdots \\ \tilde{x}(k+N|k) \end{bmatrix}, \quad \bar{u}(k) = \begin{bmatrix} \tilde{u}(k|k) \\ \tilde{u}(k+1|k) \\ \vdots \\ \tilde{u}(k+N-1|k) \end{bmatrix}$$

Thus, with $\bar{Q} = \text{diag}(Q; \ldots; Q)$ and $\bar{R} = \text{diag}(R; \ldots; R)$, the cost function (3) can be rewritten as:

$$\Phi(k) = \bar{x}^T(k+1)\bar{Q}\bar{x}(k+1) + \bar{u}^T(k)\bar{R}\bar{u}(k), \tag{20}$$

Therefore, it is possible from (17) to write $\bar{x}(k+1)$ as:

$$\bar{x}(k+1) = \bar{A}(k)\tilde{x}(k|k) + \bar{B}(k)\bar{u}(k), \tag{21}$$

where $\bar{A}$ and $\bar{B}$ are defined in (18) with $\alpha(k,j,l)$ given by:

$$\alpha(k,j,l) = \prod_{i=N-j}^{l} A(k+i|k)$$

From (20), (21) and after some algebraic manipulations, we can rewrite the objective function (3) in a standard quadratic form:

$$\bar{\Phi}(k) = \frac{1}{2}\bar{u}^T(k)H(k)\bar{u}(k) + f^T(k)\bar{u}(k) + d(k) \tag{22}$$

with

$$H(k) = 2\left(\bar{B}(k)^T\bar{Q}\bar{B}(k) + \bar{R}\right)$$
$$f(k) = 2\bar{B}^T(k)\bar{Q}\bar{A}(k)\tilde{x}(k|k)$$
$$d(k) = \tilde{x}^T(k|k)\bar{A}^T(k)\bar{Q}\bar{A}(k)\tilde{x}(k|k)$$

The matrix $H(k)$ is a *Hessian* matrix, and it is always positive definite. It describes the quadratic part of the objective function, and the vector f describes the linear part. d is independent of $\tilde{u}$ and has no influence in the determination of u^*. Thus, by defining

$$\bar{\Phi}'(k) = \frac{1}{2}\bar{u}^T(k)H(k)\bar{u}(k) + f^T(k)\bar{u}(k),$$

which is a standard expression used in quadratic programming problems and the optimization problem to be solved at each sampling time is stated as follows:

$$\tilde{u}^* = \arg\min_{\tilde{u}} \left\{ \bar{\Phi}'(k) \right\} \tag{23}$$

s. a.

$$D\tilde{u}(k+j|k) \leq d, \quad j \in [0, N-1] \tag{24}$$

Note that now only the control variables are used as decision variables. Furthermore, constraints for the initial condition and model dynamics are not necessary anymore, since now these informations are implicit in the cost function (22), and any constraint must be written with respect to the decision variables (24). In this case, the amplitude constraints in the control variables of (10) can be rewritten as

$$u_{\min} - u_r(k+j) \leq \tilde{u}(k+j|k) \leq u_{\max} - u_r(k+j)$$

and

$$D = \begin{bmatrix} I \\ -I \end{bmatrix}, \qquad d = \begin{bmatrix} u_{\max} - u_r(k+j) \\ u_{\min} + u_r(k+j) \end{bmatrix}$$

Since the state prediction is a function of the optimal sequence to be computed, it is easy to show that state constraints can also be cast in the generic form given by (24). Furthermore, constraints on the control rate and states can also be formulated in a similar way.

4. MPC LIBRARY FOR REAL-TIME IMPLEMENTATION

The MPC algorithm was implemented as a software library written in the C++ language. In the current implementation, the library is based on the donlp2 solver (Spellucci, 1998). However, there are hooks that permits an easily change of the solver for other one. To ensure accurate timing, a hard real-time extension to the Linux kernel called RTAI (Dozio and Mantegazza, 2003) was used.

The library was developed with generality in mind, so that it is useful not only for implementing MPC for mobile robots, but for any system that has a model in state space. The library supports a number of models in state space: linear, non-linear and linearized, continuous, sampled and discrete. A class library was implemented to support all combinations of those model characteristics. Some of them can be derived from others, for example, a linearized sampled model can be derived from a nonlinear continuous model. This way, in order to implement a linearized MPC such as used in Lages and Alves (2006), one can just define the continuous non-linear model of the system and let the library to obtain the linearized sample model through derivation of classes. The class hierarchy used to define system models is shown in Fig. 2.

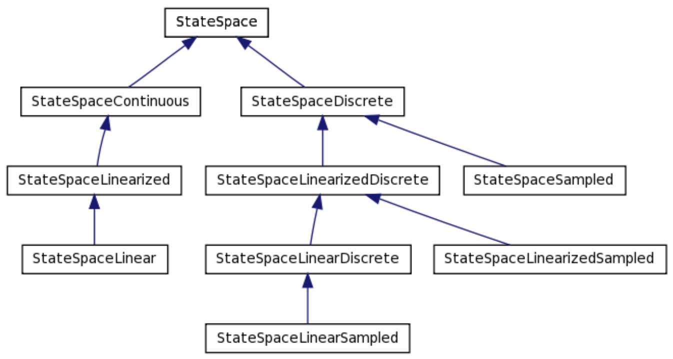

Fig. 2. Hierarchy of classes used for model description.

`StateSpace`: represents an abstract nonlinear model in state space, which can be continuous or discrete.

`StateSpaceContinuous`: represents a continuous nonlinear model in state space, as described by (1).

`StateSpaceLinearized`: represents a continuous linearized model in state space, as described by (16). This class is derived from `StateSpaceContinuous` and hence, the linearized model can be automatically obtained from the nonlinear one.

`StateSpacelinear`: represents the a continuous linear model in state space. This is a particular case of (16) where f_x and f_u are constant matrices. This class is derived from `StateSpaceLinearized`, because the linear model can be viewed as a particular case of a linearized (and nonlinear) model.

`StateSpaceDiscrete`: represents a nonlinear discrete model in state space, as described by (2).

`StateSpaceLinearizedDiscrete`: represents a linearized discrete model in state space, as described by (17). This class is derived from `StateSpaceDiscrete` and hence, the linearized model can be automatically obtained from the nonlinear one.

`StateSpaceLinearDiscrete`: represents the a discrete linear model in state space. This is a particular case of (17) where $A(k)$ and $B(k)$ are constant matrices. This class is derived from `StateSpaceLinearizedDiscrete`, because the linear model can be viewed as a particular case of a linearized (and nonlinear) model.

`StateSpaceSampled`: represents a nonlinear sampled model in state space, as described by (2). However, this class differs from `StateSpaceDiscrete` in that the discrete model is automatically obtained from a previous defined continuous model. This model can be automatically obtained from the nonlinear one though sampling of the nonlinear model. This class is also derived from `StateSpaceDiscrete` since a sampled system is a particular case of a discrete system.

`StateSpaceLinearizedSampled`: represents a linearized sampled model in state space, as described by (17). However, this class differs from `StateSpaceLinearizedDiscrete` in that the discrete model is automatically obtained from a previous defined continuous model. This class is derived from `StateSpaceLinearizedDiscrete` and hence, the linearized model can be automatically obtained from the nonlinear one.

`StateSpaceLinearSampled`: represents the a linear sampled model in state space. This is a particular case of (17) where $A(k)$ and $B(k)$ are constant matrices. However, this class differs from `StateSpaceLinearDiscrete` in that the discrete model is automatically obtained from a previous defined continuous model.This class is derived from `StateSpaceLinearDiscrete`, because the linear model can be viewed as a particular case of a linearized (and nonlinear) model.

Each of the system models described above generates an associated MPC problem, which is solved by an associated class in the library, thus generating a class hierarchy shown in Fig. 3. There is not MPC classes associated with continuous models, since it is not possible to implement a continuous MPC (which is different from MPC for a continues system) by using a digital computer. However, note that the continues `StateSpace` classes are still useful since they can be used to represent a continuous model that will be used to generate a sampled model through class derivation. Furthermore, since the sampled models are derived from discrete models (and initialized from continuous models), the MPC problem for discrete or sampled models are just the same.

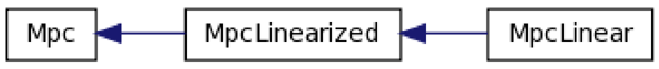

Fig. 3. Class hierarchy used for implementing MPC.

`Mpc`: represents the MPC controller for a system described by a `StateSpaceDiscrete` class. This class uses a general nonlinear programming solver (Spellucci, 1998) to solve the problem described in section 2.1.

`MpcLinearized`: represents the MPC controller for a system described by a `StateSpaceLinearizedDiscrete` class. This class is derived from `MpcDiscrete` and hence, the linearized model can be automatically obtained from the nonlinear one. However, due to the linearized model, this class uses a quadratic programming solver (Gertz and Wright, 2003), instead of a general nonlinear programming solver (Spellucci, 1998). This class solves the problem describe in section 3.

`MpcLinear`: represents the MPC controller for a system described by a `StateSpaceLinearDiscrete` class. Due to the linear model, this class uses a quadratic programming solver (Gertz and Wright, 2003), instead of a general nonlinear programming solver (Spellucci, 1998). This class solves the a particular case of the problem describe in section 3 where $A(k)$ and $B(k)$ are constant matrices.

The library was documented by using the Doxygen (van Heesch, 2004) system. Doxygen automatically generates a documentation of the library API from the code itself and from special comments tags inserted by the programmer. The documentation is generated in HTML format and automatically includes class hierarchy diagrams and documentation of function prototypes.

5. RESULTS

The library presented in previous sections was used to implement MPC for a 3-link underactuated brachiating mobile robot shown in Fig. 4. This robot has two arms and one link acting like a body where payload can located. Each arm has a gripper that can attach firmly to the supporting line, allowing the robot to execute its movement in a way similar to a manipulator robot. It moves by releasing one arm from the supporting line and grasping it again in a forward point. It is important to keep in mind that although the robot has three joints, only two of them (joint 2 and joint 3) are actuated.

By considering this brachiating robot as a serial open-chain robotic manipulator, its dynamics can be generally given by (see de Oliveira and Lages (2006) for details):

$$D(\theta)\ddot{\theta}(t) + H(\theta, \dot{\theta}) + G(\theta) = \tau - F_v \qquad (25)$$

which can be rewritten in a more suitable state space form:

$$\dot{q} = f(q) + g(q)u \qquad (26)$$

where $q = [\theta \ \dot{\theta}]^T$ is the state vector.

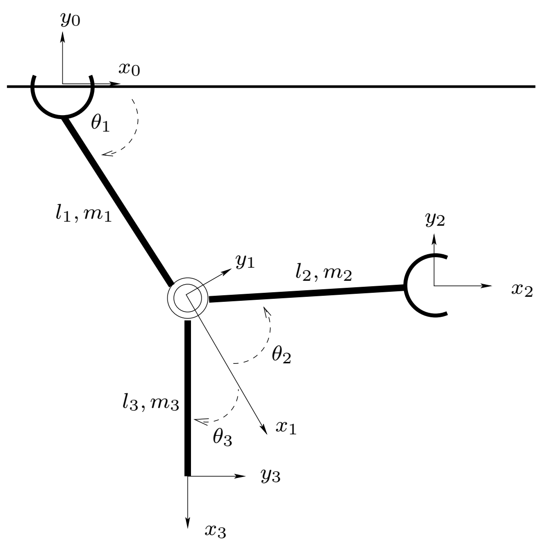

Fig. 4. Three-link brachiating robot.

The vector of the applied torque is given by $u = P\tau$ with $P = [0 \ 1 \ 1]$, which indicates that only joints 2 and 3 are actuated.

We have the initial position for the robot with the two arms horizontally stretched ($q_0 = [-\pi \ 0 \ \frac{\pi}{2} \ 0 \ 0 \ 0]^T$).

The parameters of the controller used are:

- prediction horizon $N = 5$;
- $Q = diag(5, 15)$;
- $R = diag(0.01, 0.01)$;
- $S = 0.01$;

and the constraints values are:

- maximum joint displacement $q_{max} = \pi \ (rd)$;
- maximum joint velocity $\dot{q} = 20 \ (rd/s)$
- maximum torque $\tau_{max} = 30 \ (Nm)$.

Figure 5 shows the angular position of each joint. The end-effector trajectory of the robot is shown in figure 6.

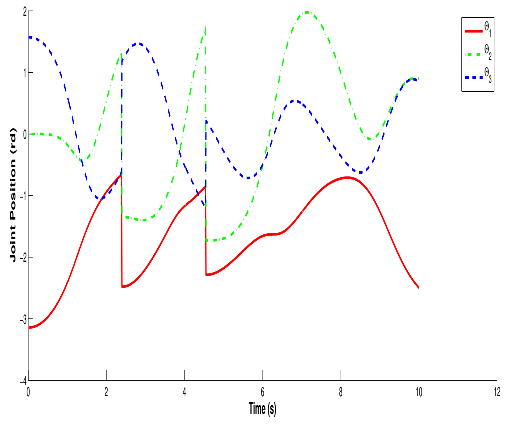

Fig. 5. Joint coordinates.

6. CONCLUSION

As shown above, the choice of MPC for the current application is well justified by some advantages: the straightfor-

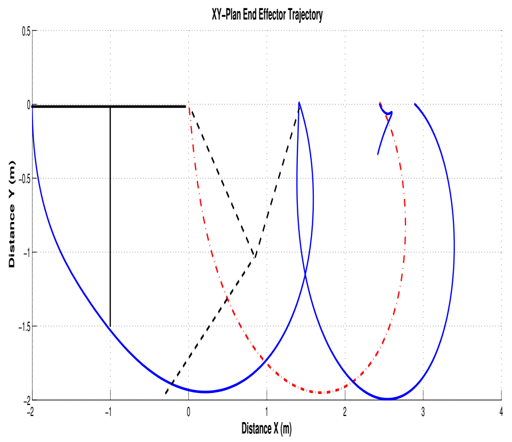

Fig. 6. Cartesian position of the end effector.

ward way in which state/input constraints can be handled; coordinate transformations to a chained or power form are not necessary; the MPC implicitly generates a control law that deals with Brockett conditions.

The viability of the proposed scheme was demonstrated by a development of software library for real-time implementation of model predictive control. This library was implemented with generality in mind, so that it can be used for implementing MPC for any system that has a model in state space. Furthermore, real-time performance data showed that recent commercially available processors have enough processing power to meet the timing requirements.

Finally, the software is available at `http://www.ece.ufrgs.br/~fetter/mpc` under the GNU General Public License (Free Software Foundation, Inc, 2007).

REFERENCES

Allgöwer, F., Badgwell, T.A., Qin, J.S., Rawlings, J.B., and Wright, S.J. (1999). Nonlinear predictive control and moving horizon estimation: an introductory overview. In P.M. Frank (ed.), *Advances in Control: Highlights of ECC'99*, 391–449. Springer-Verlag, New York.

Barclay, A., Gill, P.E., and Rosen, B. (1997). Sqp methods and their application to numerical optimal control. Technical Report NA 97-3, Dept of Mathematics, University of California, San Diego.

Bloch, A.M. and McClamroch, N.H. (1989). Control of mechanical systems with classical nonholonomic constraints. In *Proceedings of the 28th IEEE American Conference on Decision and Control*, 201–205. Tampa, FL.

Brockett, R.W. (1982). *New Directions in Applied Mathematics*. Springer-Verlag, New York.

Camacho, E.F. and Bordons, C. (1999). *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer-Verlag, London.

Canudas de Wit, C. and Sørdalen, O.J. (1992). Exponential stabilization of mobile robots with nonholonomic constraints. *IEEE Transactions on Automatic Control*, 37(11), 1791–1797.

Chen, C.C. and Allgöwer, F. (1998). A quasi-infinite horizon nonlinear nodel predictive control with guaranteed stability. *Automatica*, 34(10), 1205–1217.

de Oliveira, V.M. and Lages, W.F. (2006). Predictive control of an underactuated brachiation robot. In *Proceedings of the 8th IFAC Symposium on Robot Control*. Elsevier, Bologna.

de Oliveira, V.M. and Lages, W.F. (2007). Comparison between two actuation schemes for underctuated brachiation robots. In *Proceedings of the 2007 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Zürich.

Dixon, W.E., de Queiroz, M.S., Dawson, D.M., and Flynn, T.J. (2004). Adaptive tracking and regulation of a wheeled mobile robot with controller/update law modularity. *IEEE Control Systems Technology*, 12(1), 138–147.

Dozio, L. and Mantegazza, P. (2003). Linux real time application interface (RTAI) in low cost high performance motion control. In *Proceedings of the Motion Control 2003 Conference*. Associazione Nazionale Italiana per l'Automazione, ANIPLA, Milano, Italy.

Free Software Foundation, Inc (2007). GNU general public license. `<http://www.gnu.org/licenses/gpl.txt>`.

Fukuda, T., Saito, F., and Arai, F. (1991). A study on the brachiation type of mobile robots (heuristic creation of driving input and control using cmac). In *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, volume 2, 478–483.

Gertz, E.M. and Wright, S.J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1), 58–81.

Henson, M.A. (1998). Nonlinear model predictive control: current status and future directions. *Computers and Chemical Engineering*, 23(2), 187–202.

Lages, W.F. and Alves, J.A.V. (2006). Real-time control of a mobile robot using linearized model predictive control. In *4th IFAC Symposium on Mechatronic Systems*, 968–973. Elsevier, Heidelberg, Germany.

Luca, A.D. and Iannitti, S. (2002). A simple stlc test for mechanical systems underactuated by one control. In *Proceedings of the IEEE International Conference on Robotics & Automation*, 1735–1740.

McCloskey, R.T. and Murray, R.M. (1997). Exponential stabilization of driftless control systems using homogeneous feedback. *IEEE Transactions on Automatic Control*, 42(5), 614–628.

Oriolo, G. and Nakamura, Y. (1991). Control of mechanical systems with second-order nonholonomic constraints: Underactuated manipulators. In *Proceedings of the 30th Conference on Decision and Control*, 2398–2403.

Oya, M., Yu, C.H., and Katoh, R. (2003). Robust adaptive motion/force tracking control of uncertain nonholonomic mechanical systems. *IEEE Transactions on Robotics and Automation*, 19(1), 175–181.

Samson, C. and Ait-Abderrahim, K. (1991). Feedback control of a nonholonomic wheeled cart in cartesian space. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1136–1141. Sacramento, CA.

Spellucci, P. (1998). A new technique for inconsistent qp problems in the sqp method. *Mathematical Methods of Operations Research*, 47(3), 355–400.

van Heesch, D. (2004). *Doxygen Manual for version 1.3.7*. `<http://www.doxygen.org>`.