Nonlinear control for the vehicle
by numerical inversion of its behavioral model

Huu Phuc Nguyen, Jérôme De Miras, Stéphane Bonnet, Ali Charara
Sorbonne University, Université de technologie de Compiègne, CNRS, UMR 7253 Heudiasyc
CS 60 319 - 60 203 Compiègne cedex
Email: {huu-phuc.nguyen, demiras, bonnetst, ali.charara}@hds.utc.fr

Abstract—This paper deals with the nonlinear longitudinal and lateral combined control for the vehicle based on a simple, fast, nonlinear discrete time control approach. An approximate numerical one-step time discretization of the nonlinear plant model behavior is used to find out the outputs of controller that minimize the distance between the plant output and a linear system as closed loop reference, leading the system to adopt its dynamical behavior. Since that approximation is obtained from offline simulations and the prediction horizon is limited to one time-step, the execution time of the algorithm can be completely bounded. Experimental results obtained from some simulations show the performance and robustness of the proposed controller.

I. INTRODUCTION

The single-track vehicle model depicted on the Fig. 1 is usually used for studying the vehicle’s dynamics. It’s a nonlinear underactuated system that presents complex properties interesting for studying control problems. The control problem for vehicle is usually considered by separating into two parts, the one for cruise control and the other for lateral control. The aim of longitudinal control is to keep the velocity of the vehicle stable at a desired speed while the lateral control keeps the vehicle in the right lane and the correct direction. Many works and control applications about the vehicle can be found in the literature. For example, in [1], the author used the sliding mode control with high order for lateral control. The authors in [2] considered the system like a flat system and build the control law. In [3], the authors proposed an acceleration slip regulation strategy for longitudinal control. A robust longitudinal velocity tracking of vehicles using traction control and brake control was proposed [4]. In [5] and [6], a learning control was used to tune the parameters of a PID control. An piecewise affine controller was used in [7] for control a fast unmanned ground vehicles. A combined lateral and longitudinal control of vehicles was considered in [8]. The authors in [9] and [10] also proposed a combined longitudinal and lateral vehicle control employing algebraic estimation techniques coupled with flatness-based control or model-free control.

In this paper, we focus on the design of a discrete-time, state-space, nonlinear controller to design a combined control for the vehicle system. This controller has been studied and applied on a laboratory real-time magnetic levitation system [11], and on the PVTOL aircraft [12]. Since the computational burden gets heavier as the prediction horizon increases, the smallest possible horizon will be used: one time step. By doing so, the ability of the controller is foreclosed to handle explicit constraints in favor of getting a fast controller. However, it still has to be robust towards both prediction and modeling errors and external perturbation forces. Firstly, the nonlinear model used will be described, then the control algorithm, and finally simulation results obtained from MATLAB/SIMULINK and MAPLE will be presented, showing the validation of the proposed model and the robustness of the proposed approach.

II. VEHICLE MODEL

A. Single-track vehicle model

The simplified model of the vehicle (Fig. 1) has been famously represented by a single-track vehicle model for studying vehicle dynamics or tire’s characteristics [13]. Using the equation of Newton-Euler, the system is given by the following equations in the body frame:

\[
\begin{align*}
V_x &= \dot{V}_x + \frac{F_{xf}}{m} \cos \delta - \frac{F_{yf}}{m} \sin \delta - k_1 V_x^2 \\
V_y &= \dot{V}_y + \frac{F_{yr}}{m} \frac{m}{V_y} - k_2 V_y^2 \\
I_x \dot{\bar{\theta}} &= -F_{yr} L_r + \left( F_{xf} \sin \delta + F_{yf} \cos \delta \right) L_f + M_{xf} + M_{xf} \\
\end{align*}
\]

(1)

The dynamics of rotation for the wheels are described by:

\[
\begin{align*}
I_{xf} \ddot{\phi}_f &= -F_{xf} R + M_f - M_{bf} \\
I_{yr} \ddot{\phi}_r &= -F_{xr} R + M_r - M_{br} \\
\end{align*}
\]

(2)
where the subscripts $f$ and $r$ are related to the front and rear wheels, the variables and the parameters are described in the table:

| $G$ | Center of mass |
| $[x, y, \theta]$ | Pose of vehicle |
| $[V_x, V_y]$ | Velocity in local body frame |
| $L_f, L_r$ | Wheel base |
| $m$ | Mass of vehicle |
| $I_z$ | Moment of inertia |
| $\delta$ | Steering angle |
| $F_{zf}, F_{yz}, M_{zf}$ | Front wheel’s force and moment |
| $F_{yr}, F_{yr}, M_{yr}$ | Rear wheel’s force and moment |
| $I_{gf}$ | Inertia of front wheel |
| $I_{yr}$ | Inertia of rear wheel |
| $\phi_f, \phi_r$ | Wheels’ angular velocity |
| $M_f, M_r$ | Wheels’ torque |
| $M_{bf}, M_{br}$ | Brake torque |
| $R$ | Loaded radius of wheel |
| $k_1, k_2$ | Drag forces’ coefficient |
| $d$ | Distance to road center line |
| $\theta_c$ | Angle respect to road |

B. Full vehicle model

There are many toolkits and drag-drop environments for modeling the vehicle dynamics using multibody dynamics theory, for example SimMechanics from Mathworks, MapleSim from MapleSoft. In this paper, we used a 4-wheels vehicle model built with MapleSim depicted on Fig. 2 with traction on the rear wheels. This model uses Tire Library Component and Drive Line Component for using Pacejka tire model and gear box. It contains four simple suspensions and brake for each wheel. This model also includes a differential gear box to transmit the torque from transmission to wheel. The inputs of the system are: steering signal, braking signal and torque from motor to transmission. Some sensors were used to obtain the vehicle’s state: position, velocity, orientation and angular velocity. All parameters from this model are taken out to build the single-track model in previous subsection and this model is exported to SIMULINK’s environment.

III. PROPOSED CONTROL SCHEME

A. Problem formulation

The basic idea behind the proposed discrete-time control strategy is to use the state of a given stable linear system as a closed loop reference for the output of the plant. That way, if the plant output trajectory is able at each time step to match the reference system state, it will have the same dynamics and converge to the origin. In the following, vectors and matrices are denoted in bold font.

Let a nonlinear, time-invariant model of the plant of the form:

\[
\begin{align*}
\dot{x}(t) &= f(x(t), u(t)) \\
y(t) &= h(x(t))
\end{align*}
\]

where $x \in S \subset \mathbb{R}^n$ is the plant state vector, $u \in U \subset \mathbb{R}^d$ its input vector control, and $y \in Y \subset \mathbb{R}^m$ its output. Suppose now the following discrete-time prediction map obtained from that model exists:

\[
x_{k+1} = p_x(x_k, u_k)
\]

where the function $p_x$ is simply the sample and hold discretization of the state function $f$ in (3) using a time-step of duration $\Delta t$ such as $t_k = k \Delta t$. In the general case, obtaining an exact analytic expression for $p_x$ is difficult. However, its approximate values for a given state and input vector can usually be computed through numerical integration of $f$.

The aim of the controller is to make the plant output $y$ to converge to a given set point $y_c$. Equivalently, the error $\hat{y} = y - y_c$ must be driven to zero. Suppose now the following stable, homogeneous linear system:

\[
\dot{\hat{y}}(t) = A \cdot \hat{y}(t), \quad \hat{y} \in \mathbb{R}^m, A \in \mathcal{M}^{m \times m}.
\]

The stable dynamic defined by $A$ selected according to the known dynamics of the plant and is Hurwitz matrix. From (5) the trajectory of $y(t)$ will also converge towards that point. The objective of the controller is to find at each time-step $k$ a vector control $u_k$ that will make the next-step plant output $y_{k+1}$ be as close of $y_{k+1}^g$ as possible, with $y_{k+1}^g$ obtained from the discretization of (5) as follows:

\[
y_{k+1}^g = e^{A \Delta t} (y_{k} - y_c) + y_c.
\]

The reference $y_{k+1}^g$ can be selected as a nonlinear stable function of a given set point $y_c$ and the current state $y_k$. Then a lookup table can be used to calculate its value $y_{k+1}^g$. It is essential that the reference state trajectories and the output trajectory share dynamics of the same order. At each time step, the plant output vector $y_{k+1}$ has to be computed. That means that an output prediction map $p_y$ is required such as:

\[
y_{k+1} = p_y(x_k, u_k).
\]

By defining a vector $v = [x, y]^T$, $x$ and $y$ are obtained as follow:

\[
x = P_x \cdot v, \quad y = P_y \cdot v
\]

At step $k + 1$ this vector is

\[
v_{k+1} = [p_x(x_k, u_k), p_y(x_k, u_k)]^T = p(x_k, u_k).
\]
Finally, computing the control input at each time step can be formally written as solving the following Euclidean norm minimization problem:

$$u_k = \arg \min_{u \in \mathcal{U}} \| y^2_{k+1} - P_y \cdot p(x_k, u) \|_2.$$  \hfill (9)

B. Online control algorithm

Suppose that at step $k$, an estimation $\hat{x}_k$ of the plant state is available, and that its output $y_k$ is measured, allowing to compute the vector $v_k$. The control input computation is instantaneous, and that the state estimation does not introduce any delays: from the measurements at step $k$, the control input $u_k$ can be readily computed and immediately applied to the plant. With real time system, some methods were proposed to compensate for delay between the output control and the measurement, for example [11].

The aim of the control algorithm is to compute an approximate solution to (9), as analytical forms for $p$ (i.e $p_x$ and $p_y$) is not known in general. Suppose also that specific approximate values of theses functions can be computed at any point within the continuous set $S \times U$. The algorithm can be split into four main parts:

1) State estimation and output vector computation, supposed available, or from sensors,
2) Prediction error estimation,
3) Reference point calculation, as shown above in (6)
4) Control input approximation.

The prediction maps are based on a model of the system. As precise as it may be, it necessarily contains approximations that make the prediction maps inherently unreliable. In addition to modeling approximations, external perturbations that are not modeled at all also lead to incorrect predictions: they have to be corrected. Fundamentally, the prediction error is simply the difference between the predicted plant state and outputs for step $k$ obtained from data at step $k-1$ and their estimations obtained from measurements at step $k$. The error vectors $\varepsilon_k$ is defined such as:

$$\varepsilon_k = p(v_{k-1}, u_{k-1}) - v_k.$$  

Further predictions used in the algorithm to compute the control input are corrected using those two vectors, under the hypothesis that the error obtained at step $k$ is close enough to that obtained at step $k-1$. In other words, the error dynamics is supposed slow relative to the controller. Since it may not be the case in noisy conditions for instance, where the measurements can vary a lot between two steps, the error signal is not used as is, but filtered to remove high frequency content, the main objective being to compensate for static errors or slowly varying perturbations. Trying to compensate for high frequency errors can indeed lead to oscillations and controller instability. The error signal for the state prediction used at step $k$ can thus be written as:

$$\varepsilon_k = \varepsilon_{k-1} + \alpha (p(v_{k-1}, u_{k-1}) - v_k)$$  \hfill (10)

with $\alpha$ a damping coefficient in $(0,1]$, forming a numerical low-pass filter. The error signal for the output predictions is handled by the same method.

Since the expression for $p$ is supposed unknown but we are able to perform numerical evaluations for each one, an iterative algorithm will be used to approximate the solution of (9). This algorithms starts by uniformly discretizing the input interval $\mathcal{U} = [u, \bar{u}]$ to yield the input sets

$$\mathcal{U}_{i=1,...,d} = \left\{ u_j | u_j = u_i + \frac{\bar{u}_i - u_i}{N_i}, j = 0, \ldots, N_i \right\}$$

If $p$ is not a convex function, the solution of the minimization problem can be found using an exhaustive search algorithm. If $p$ is a convex function, the following more efficient algorithm, based upon the work presented in [14], [12], can be used:

1: function CALCULATEACTION(x, y$^g$)
2: $d \leftarrow \text{dim}(\mathcal{U})$  \hfill \triangleright Inputs’ dimension
3: $s_i \leftarrow |\mathcal{U}_i|, i = 1, \ldots, d$  \hfill \triangleright Number of points in $\mathcal{U}_i$
4: $\sigma = \text{conv} \{\xi_0, \ldots, \xi_d\}, \sigma' = \text{conv} \{\Gamma_0, \ldots, \Gamma_d\}$
5: $\xi_0 \leftarrow \left[ \frac{1}{2} \right]$  \hfill \triangleright Initial vertex at center point for inputs
6: $\Gamma_0 \leftarrow \mathcal{P}_y \cdot p(x, \xi_0)$  \hfill \triangleright Initial vertex for outputs $\mathcal{Y}$
7: for $i = 1, \ldots, d$ do
8: $\xi_i \leftarrow \xi_{i-1} + e_i$  \hfill \triangleright $e$ is canonical basis of $\mathbb{R}^d$
9: end for
10: repeat
11: for $i = 1, \ldots, d$ do
12: $\gamma_i \leftarrow \Gamma_i - \Gamma_0$  \hfill \triangleright Calculate frame of $\text{aff}(\sigma')$
13: end for
14: Calculate projection $y_{\perp}$ of $y^g$ onto $\text{aff}(\sigma')$
15: Solve $\alpha$ from $\Gamma_0 + \sum_{i=1}^{d} \gamma_i \cdot \alpha_i = y_{\perp}$
16: $\alpha' \leftarrow T \cdot \alpha$  \hfill \triangleright Transform to $\text{aff}(\sigma')$
17: $r \leftarrow \text{TESTSIMPLEX}(\alpha')$
18: $n \leftarrow 0$  \hfill \triangleright Number of pivots
19: for $i : r_i = false$ do
20: $\tilde{\xi}_i \leftarrow \text{PIVOT VERTEX} (\xi_i)$
21: if $1 \leq \tilde{\xi}_i \leq s$ then  \hfill \triangleright Check in $\mathcal{U}$
22: $\xi_i \leftarrow \tilde{\xi}_i$  \hfill \triangleright Recalculate $\sigma$ and $\sigma'$
23: $\Gamma_i \leftarrow \mathcal{P}_y \cdot p(x, \xi_i)$
24: $n \leftarrow n + 1$
25: end if
26: end for
27: until $n = 0$
28: $y \leftarrow \text{PROJECTONSIMPLEX}(\sigma', y^g)$
29: Calculate $u$ as orthogonal projection of $y$ onto $\text{aff}(\sigma)$
30: return $u$
31: end function

In this algorithm, the simplex and the affine-envelope of a set of vector are defined as:

$$\text{conv}(\mathcal{X}) = \left\{ \sum_{i=1}^{k} \omega_i \cdot x_i | x_i \in \mathcal{X}, \omega_i \in \mathbb{R}^+ \right\}, \sum_{i=1}^{k} \omega_i = 1 \right\}$$

$$\text{aff}(\mathcal{X}) = \left\{ \sum_{i=1}^{k} \omega_i \cdot x_i | x_i \in \mathcal{X}, \omega_i \in \mathbb{R} \right\}, \sum_{i=1}^{k} \omega_i = 1 \right\}$$

At current time, the state vector of the system is $x$. This algorithm starts by searching a simplex in the inputs space.
that minimizes the distance to the reference point \( y^g \). At the first step, one input simplex is arbitrarily chosen for generation of the output simplex at lines 3 – 9 using linear interpolation. During the next step, the reference point is projected onto \( \text{aff}(\sigma') \). Using the coordinates of this point, we can detect if \( y \) is within the simplex \( \sigma' \) or not (line 17) and return a vector of booleans. If it contains a false element at index \( i \), that means the solution must be in the sub-simplex not contained in this vertex. The input simplex and the output simplex are recalculated by performing a pivot operation (line 20). The algorithm is repeated. If we have no vertex to pivot, we obtain the simplex expected. In other words, the final simplex is our result. Another algorithm in [14] is used to find a point on the simplex that is closest to the reference (line 28). Finally, the control input is calculated as an orthogonal projection of this point onto \( \text{aff}(\sigma) \).

Once control input minimizing the distance has been computed, it can be applied to the plant, ending the current step. While it only finds an approximate value, the execution time of this algorithm can be bounded, and depends only on the number of input discretization points used. However, the error due to the linear approximations is proportional to the square of the discretization grid size; a trade-off has to be found between the minimization error and execution time. Another important factor is the time taken by the numerous evaluations of the prediction maps \( p \) \((p_c \text{ and } p_v)\).

The complete algorithm for closed-loop system is [14]:

In order to build that piecewise approximation, a regular rectangular grid \( G \) is constructed on the joint state and input spaces \( S \times U \). Each point of \( G \) represent a possible initial state and a constant control input that can be applied to the model. For each point, the equation of the system is solved over one time-step with the given initial state and using the associated control input, using simulation tools such as Simulink. The solution vectors \( s \) are then stored in a table. The values of \( p \) at unknown points can be interpolated from that table, using barycentric linear interpolation as described in [15] for instance. While linear interpolation leads to errors proportional to the square of the grid size, its main advantage is the low computing effort needed to get interpolated values, hence the choice of that approach instead of more precise but less efficient methods such as spline interpolation.

D. Implementation of the algorithm for the vehicle

Consider now the single-track system as defined by (1) and (2). The state vector and the output of this system are \( x = [V_x, Vy, \theta, \phi_f, \phi_r]^T \) and \( y = [V_x, Vy, \theta, \dot{\theta}]^T \) respectively. The input vector is \( u = [\delta, M_r]^T \). To calculate the tire force and moment of the wheels, the Magic formula [13] has been used as function of vertical force, longitudinal slip, slip angle, camber angle and the road’s condition. The tracking reference depends on time and is given by \( y_c = [V_{xd}, V_{yd}, \dot{\theta}]^T \). The desired lateral velocity \( V_{yd} \) and angular velocity of the vehicle \( \dot{\theta} \) are calculated as an internal reference by using the following system:

\[
\begin{align*}
\dot{d} &= V_x \sin \theta + V_y \cos \theta \\
\dot{\theta} &= \dot{\theta} - \frac{\cos \theta \dot{\theta}_e - V_y \sin \theta}{R_e - \dot{d}}
\end{align*}
\]

where the vector \( R_e \) represents the curvature of the road, for example \( R_e \) is equal to the radius for circle and \( R_e \) is very big for a line road. The state vector and the output vector are \( x = [d, \theta, \dot{\theta}]^T \) and \( y = [d, \theta, \dot{\theta}]^T \) respectively, with the input vector is \( u = [V_x, \dot{\theta}]^T \) and \( V_x \) used as the internal input. Using the prediction map for this system, the value of \( V_{yd} \) and \( \dot{\theta} \) can be predicted knowing the current state and the target at the next step. The target point for bicycle system is rebuilt as \( y_c = [V_{xd}, V_{yd}, \dot{\theta}]^T \). The algorithm is modified as Fig. 4.

---

![Diagram](image-url)
compute the desired vector $[V_{yd}, \dot{\theta}_d]^T$. Block $\text{Predictor}_\text{traj}$ contains exactly the algorithm as Fig. 3 applying for system (11). The desired state is bounded in one range due to the use of a prediction map to search the optimal input. As the same meaning, when the output vector and the state vector exceed their ranges, they will be trimmed.

**IV. SIMULATION RESULTS**

In this section some simulation results obtained using MATLAB and SIMULINK are presented. The vehicle started from the initial condition $V_x(0) = 0$, $V_y(0) = 0$, $\theta(0) = 0$, $\dot{\theta}(0) = 0$. The angle and the distance to the road is 0 and $3m$ respectively. After $t_0 = 10$ the controller will be triggered. The internal reference linear state-space system for each axis $V_x$ and $V_y$ has an equivalent canonical first-order transfer function with a time constant of $\tau_x = 10$. For the angle $\theta$, a second-order transfer function with a frequency of $\omega_0 = 15Hz$ and a damping ratio $\zeta = 0.5$ has been used. The torque is limited in the range $|\tau| \leq 400$ Nm and the steering angle saturation bound $\pm \pi / 18$ rad. When the computed torque is negative, the brakes will be activated. The inputs are bounded by $|V_x| \leq 4.0, |V_y| \leq 1.0$ and $|\theta| \leq 1.0$. To validate the model and test the robustness of the proposed controller, the prediction map is generated using a single-track model and test for a 4-wheels vehicle modeled from MAPLESIM with the initial equivalent condition. The parameters used for simulation are given in the table following:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>1408 kg</td>
</tr>
<tr>
<td>$I_z$</td>
<td>1950 kg.m$^2$</td>
</tr>
<tr>
<td>$I_{yf}, I_{yr}$</td>
<td>0.78 kg.m$^2$</td>
</tr>
<tr>
<td>$L_r$</td>
<td>1.490 m</td>
</tr>
<tr>
<td>$L_f$</td>
<td>1.482 m</td>
</tr>
<tr>
<td>$R_{eff}$</td>
<td>0.344 m</td>
</tr>
<tr>
<td>$k_1$</td>
<td>0.001 m$^{-1}$</td>
</tr>
</tbody>
</table>

In the first simulation, the tracking trajectory is a circle with $R_e = 20m$ while keeping velocity $V_x = 10m/s$ from a straight line. Fig. 5 shows the temporal response trajectory. These references are followed and reached without any static error, showing the robustness of the controller to external perturbations from curvature. Fig. 6 shows the distances between the single-track model and the vehicle to the road. Fig. 7, 8 and 9 show the velocity of the vehicle, the steering angle and the torque applied to the wheels. They also show that the brakes have to be activated when the velocity exceeds the desired value. Due to the error between the prediction map built from a single-track model and the vehicle was slowly compensated using (10), Fig.8 shows some oscillations on the steering angle. In future work, the vehicle model will be corrected by using a learning method.

The second simulation makes the vehicle go in a complex path with more perturbation, for example the obstacle and the curvature of the road. Fig. 10 shows the temporal response trajectory and the distance to the road is shown by the Fig. 11. The vehicle with $\pm 20\%$ of mass was also tested. When $x$ limited in the range of $[150, 230]$, the vehicle has been driven to the distance $d = 3m$ in order to avoid the obstacle. When $t = 47s$ the vehicle enters a half-circle and the curvature changed as a step function then Fig. 11 shows the response of the controller. All pictures show that the dynamics of vehicle is more complex than its single-track model and then the time response is longer.

**V. CONCLUSION**

Since the real experiments on a laboratory magnetic levitation shaft with the same novel predictive nonlinear controller applied [11], and for the PVTOL [12], the results of vehicle control presented in this paper shows that this controller has good convergence and robustness properties. The laboratory has two electric robotised cars (Renault ZOE) on which it will soon be possible to test this control law. However,
the error correction step is essential to the algorithm and needs further refinement. Moreover, whether a well-chosen reference system could be used to constrain the state and/or outputs within specified bounds remains to be investigated.

REFERENCES