NONLINEAR TRAJECTORY GENERATION FOR THE CALTECH MULTI-VEHICLE WIRELESS TESTBED

Jonathan Chauvin[†], **Laure Sinègre**[†], **Richard M. Murray**[‡]

[†] École Nationale Supérieure des Mines de Paris, 60 bd St Michel, 75272 Paris, France
{jonathan.chauvin,laure.sinegre}@ensmp.fr
[‡] Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA
murray@cds.caltech.edu

Keywords: Trajectory planning, Optimal control, Nonlinear control, Model Predictive Control, Real-time

Abstract

The Caltech Multi-Vehicle Wireless Testbed (MVWT) is a platform designed to explore theoretical advances in multi-vehicle coordination and control, networked control systems and high confidence distributed computation. The contribution of this report is to present simulation and experimental results on the generation and implementation of optimal trajectories for the MVWT vehicles. The vehicles are nonlinear, spatially constrained and their input controls are bounded. The trajectories are generated using the NTG software package developed at Caltech. Minimum time trajectories and the application of Model Predictive Control (MPC) are investigated.

1 Introduction

In this article, the problem we are interested in is relative trajectory generation for flight control systems. As the vehicles have second order dynamics and are underactuated we cannot rely on AI style planning. We have to call on optimization based trajectory generation and advanced control techniques. Like in the case of the Caltech Ducted Fan experiment [9] we use the Nonlinear Trajectory (NTG) software package. Our goal is also the same: being able to generate and to implement trajectories as aggressive as possible that the vehicles can actually follow i.e. trajectories that satisfy every constraint of the testbed. Those constraints can either be linear, like the boundaries of the testbed or nonlinear like the constraints on the input. The main difference and also difficulty in our case is that the system is not linearly controllable around its equilibrium.
In Section 2 we will give a quick description of the system properties and in Section 3 and 4 we will describe the progression which led us from the optimization problem to the implementation on the real vehicles. In Section 5 other optimization problems such as minimum time trajectory generation and model predictive control are investigated.

2 Description of the System's Properties

The vehicle we are working on consists of a laptop mounted on three low-friction, omni directional caster [1]. Two ducted fans are mounted on the top of the vehicle, each of them capable of producing from 0 to 4.5 N of continuous thrust, as shown on Figure 1. The vehicle has three degrees of freedom, two of translation: x and y and one of rotation: θ, as shown on Figure 2. The second order dynamics, assuming viscous friction, is:

$$\begin{cases} m\ddot{x} = \eta\dot{x} + (F_s + F_p)\cos\theta \\ m\ddot{y} = \eta\dot{y} + (F_s + F_p)\sin\theta \\ J\ddot{\theta} = \psi\dot{\theta} + (F_s - F_p)r_f \end{cases}$$

Notice that the system is not linearly controllable

Figure 1: picture of the vehicle

around any equilibrium point, corresponding to $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}) = (c_1, c_2, c_3, 0, 0, 0)$ for any $(c_1, c_2, c_3) \in \mathbf{R}^2 \times S^1$ with equilibrium inputs $(F_s, F_p) = (0, 0)$. Besides the system is differentially flat [2, 3], i.e the whole state and the inputs can be parameterized by two flat outputs. Let z_1 be x and z_2 be

y, we have:

$$\left(z_1, z_1^{(1)}, z_1^{(2)}, z_1^{(3)}, z_1^{(4)}, z_2, z_2^{(1)}, z_2^{(2)}, z_2^{(3)}, z_2^{(4)}\right) = \phi(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}, F_s, F_p)$$

As we would like to have those two sets of coordinates

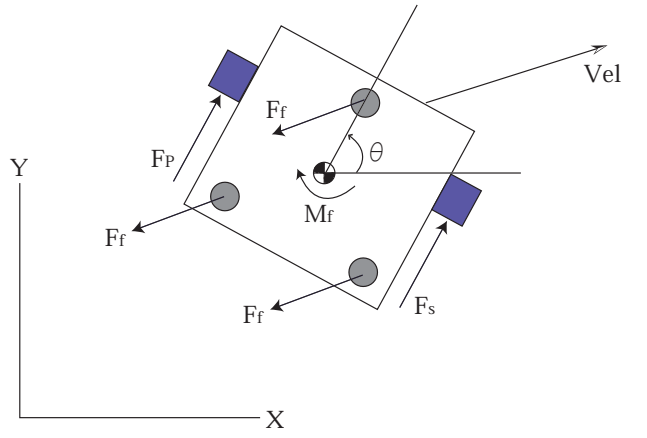

Figure 2: schematic of the vehicle

linked by a diffeomorphism, we have to add two more parameters in the first set, for examples $d(F_s + F_p)/dt$ and $d^2(F_s + F_p)/dt^2$. Let $\tilde{\phi}$ denote this linking function. The determinant of its jacobian is: $\det(\text{Jac}(\tilde{\phi})) = -2\frac{(F_s+F_p)^3 r_f}{m^6 J}$. So $\tilde{\phi}$ defines a diffeomorphism as long as F_s or F_p remain stricly positive; equilibrium points are singularities. Given that $(F_s, F_p) \in [0, 4.5]^2$, instead of trying to generate a trajectory from an equilibrium point to an another equilibrium point, we generate a trajectory starting from a point on a given circular trajectory and ending at a given point on an another circular trajectory [1]. This makes sense because we already know how to stabilize the error dynamics around a circular trajectory with an LQR controller and also because in some contexts it is more useful to have a vehicle circling instead of staying in one position.

3 Trajectory Generation

The goal of this section is to describe the optimization problem whose resolution will give us an admissible trajectory that will be applied to the vehicle. We would like to generate a trajectory that satisfies the environmental contraints: the limits of the testbed, the presence of an obstacle in the middle of the testbed linked to an unrelated experiment and also the physical constraints due to the fact that the thrust produced by the fans is limited. So we will describe two optimization problems, one which is using the flat coordinates and one which is not and we will discuss the advantages and disadvantages of each parameterization.

3.1 Optimization problem in usual coordinates

We would like to minimize:

$$\min_{\bar{X}} \int_0^T q(X(t), u(\bar{X}(t))dt + V(X(T), u(\bar{X}(T))$$

under the constraints:

$$\begin{cases} X(0) = X_0 & \text{initial constraints} \\ t_c(X(t), u(t)) = 0, \forall t \in [0, T] & \text{trajectory constraints} \\ X(T) = X_T & \text{final constraints} \end{cases}$$

Where X is the state $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$, $\bar{X}$ an extension of the state $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}, \ddot{x}, \ddot{y}, \ddot{\theta})$ and u is the input, $u = (F_s, F_p)$. The cost functions in this case are :

$$q(X, u) = (X - X_T)^t Q (X - X_T) + (u - u_T)^t R (u - u_T)$$
$$V(X, u) = (X - X_T)^t Q' (X - X_T)$$

With Q, Q', R some definite positive diagonal matrices and:

$$u(X) = \frac{1}{2} \begin{pmatrix} \Omega(x)\cos\theta + \Omega(y)\sin\theta + (J\ddot{\theta} + \psi\dot{\theta})r_f \\ \Omega(x)\cos\theta + \Omega(y)\sin\theta - (J\ddot{\theta} + \psi\dot{\theta})r_f \end{pmatrix}$$

where $\Omega(x) = m\ddot{x} + \eta\dot{x}$.
Explicitly, the trajectory constraints can be divided into two categories. A linear constraint that corresponds to staying on the testbed means that we have to satisfy :

$$\forall t \in [0, T] \quad 0 < x(t) < x_{max} \quad \text{and} \quad 0 < y(t) < y_{max}$$

The nonlinear constraints are:

$$\begin{cases} (m\ddot{x} + \eta\dot{x})\sin\theta - (m\ddot{y} + \eta\dot{y})\cos\theta = 0 & \text{dynamics} \\ (x - x_{post})^2 + (y - y_{post})^2 > R_{post}^2 & \text{post} \\ 0 \leq F_s \leq F_{max} \quad \text{and} \quad 0 \leq F_p \leq F_{max} & \text{forces} \end{cases}$$

3.2 Optimization problem in flat coordinates

In this case what we would like to minimize is:

$$\min_{Z_1, Z_2} \int_0^T q(Z_1(t), Z_2(t))dt + V(Z_1(T), Z_2(T))$$

under the constraints :

$$\begin{cases} Z(0) = Z_0 & \text{initial constraints} \\ t_c(Z_1(t), Z_2(t)) = 0, \forall t \in [0, T] & \text{trajectory constraints} \\ Z(T) = Z_T & \text{final constraints} \end{cases}$$

Where $Z = (Z_1, Z_2)$, $Z_1 = (z_1, z_1^{(1)}, z_1^{(2)}, z_1^{(3)}, z_1^{(4)})$ and $Z_2 = (z_2, z_2^{(1)}, z_2^{(2)}, z_2^{(3)}, z_2^{(4)})$. The cost functions in this case are :

$$q(Z_1, Z_2) = (X(Z_1, Z_2) - X(Z_T)^t Q(X(Z_1, Z_2) - X(Z_T)) + (u(Z_1, Z_2) - u(Z_T))^t R(u(Z_1, Z_2) - u(Z_T))$$
$$V(Z_1, Z_2) = 0$$

With Q and R some definite positive diagonal matrix. The trajectory constraints are given by:

$$0 < z_1(t) < x_{max} \quad \text{and} \quad 0 < z_2(t) < y_{max}, \forall t \in [0, T]$$

And the nonlinear constraints are :

$$\begin{cases} (z_1 - x_{post})^2 + (z_2 - y_{post})^2 > R_{post}^2 & \text{post} \\ 0 < F_s < F_{max} \quad \text{and} \quad 0 < F_p < F_{max} & \text{forces} \end{cases}$$

The fact that we are now using a system of coordinates with no obvious physical meaning makes the equations, and particularly the expression of the forces much more complex than previously. That is why we have not explicitly written the link between the forces and Z in this paragraph. Moreover, losing some of our physical intuition sounds like an argument in favor of the usual coordiantes. Nevertheless this method has some considerable advantages: there is no more dynamical constraint and the trajectory generated will necessarily match the dynamics of the system automatically. As a result, computation times are generally shorter, which explains why we chose to use the flat coordinates.

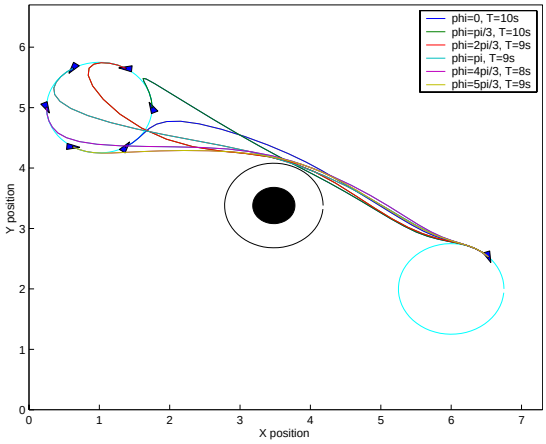

Figure 3: Six of the twelve trajectories we run on the vehicle -to go-

3.3 Problem set-up

We are using NTG (Nonlinear Trajectory Generation) [4, 5], which is a software package used for trajectory generation that combines elements of geometric control and B-splines A complete treatment of these functions can be found in (*de Boor, 1978*). The two inputs, Z_1 and Z_2, are projected on a basis of $26\,7^{th}$ order B-splines and we are looking for their coefficients. What remains to define is the time T, the initial and final position of the vehicle and the initial guess for the coefficients.

We note that some of those parameters are very sensitive, particularly the time horizon T. If for some T NTG is not able to find a solution it can be because the horizon is to short but also because it is to long. We want to reach a non equilibrium point, so if it is possible to reach the final point in a certain amount of time t_1, NTG may not converge for $t_1 + \delta$ as going to the final point in t_1 and staying at this position during δ is not a valid solution. A straight line linking the starting and the ending point is a good enough initial guess for the problem. So, what we decided to implement on the real vehicles is a program which lets the user choose between twelve different trajectories to go from one predefined circle to another predefined circle. These trajectories are not precomputed. NTG is generating the coefficients in real time. The only thing NTG already knows is the amount time a specific trajectory can take.

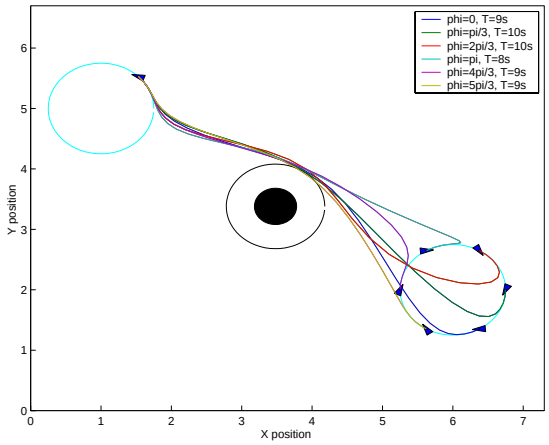

Figure 4: Six of the twelve trajectories we run on the vehicle -return-

4 Implementation on the Experimental Testbed

4.1 Finding an appropriate controller

As the dynamic equations we are using cannot fully describe the real vehicle, it is important to find a way to stabilize the system around the trajectory generated by NTG. The first idea was to approximate the trajectory by the tangent circle at each point and to stabilize the system with an LQR controller around the corresponding circular trajectory. But what if the portion of the trajectory we are trying to approximate by a circle is almost a straight line? Also, following a circular trajectory means having one's acceleration orthogonal to one's velocity which is not necessarily the case for any point on the nonlinear trajectory. Let us rather have a look at the error dynamics:

$$\dot{e} = Ae + B(F_s^{ref} + F_p^{ref}, \theta_{ref})(u - u_{ref})$$

where $e = X - X_{ref}$
Moreover, $\forall (\theta_{ref}, F_s^{ref} + F_p^{ref}) \in S^1 \times \mathbf{R}^*$ the pair (A, B) is controllable. So assuming that $F_s^{ref} + F_p^{ref}$ and θ_{ref} are known, which is the case since these parameters are given by NTG, it is possible to design an LQR controller for this problem. On the real system it is not possible to compute the

gain matrix K on-line, so we made a gain scheduling for 16 values of θ_{ref} (linearly spaced between 0 and $\frac{\pi}{2}$) and for 24 values of $F_s^{ref} + F_p^{ref}$ (linearly spaced between 0 and 12), where the other values for θ_{ref}, in $[\frac{\pi}{2}, 2\pi]$, are obtained by symmetries.

4.2 Implementation

The program is written in C++ and is running under the QNX real time operating system. It is also using the RhexLib robot programming suite [6] which facilitates a modular programming style. This has been very useful since this has let us use a controller that had already been programmed in order to stabilize a vehicle around a circular trajectory. The structure we set up consists of one thread running NTG in real time and two controller modules: `modulecontroller`, one stabilizing the vehicle around a circular trajectory and one around the trajectory generated by NTG. On top of that we have one module called `SwitchController` which switches between the two controller modules, by activating one and deactivating the other. So by default, when the programm is launched, the vehicle tries to follow a certain predefined circular trajectory. It keeps on following this trajectory until a key is pressed. At this moment we know the grid number which corresponds to the vehicle position on the circle, NTG computes the trajectory starting two grid numbers ahead, so that when the vehicle reaches the starting point the nonlinear trajectory has already been generated. The circular controller is deactivated and the reference trajectory becomes the nonlinear one. After following the trajectory generated by NTG during a certain time, we switch back to the first controller, which has now been set up so that the vehicle aims at following the second predefined circle. We could have defined the second condition differently, for example as the first one, as a space condition, $X - X_{final} < \varepsilon$, but as the controller stabilizing around the nonlinear trajectory is gain scheduled we fear it may not bring the vehicle close enough to the final point. In fact we tried both solutions and even if both worked, the first one seems to be more efficient.

4.3 Experimental results

Figure 5 and Figure 6 show data that has been taken from an experiment conducted on the MVWT testbed. What we can notice is the fact that the controller is tracking θ very well. This is due to the fact that the dynamics of θ is linear and independant from the other outputs, so the controller that we apply on θ is a normal LQR one without any gain scheduling. This is a first step but this is still far from what is needed. Actually it is not acceptable to have to know the time that the trajectory will take before computing it. We should be able to compute trajectories in reconfigurable environments with the minimum knowledge of what will happen in the future, so we should not have to make assumption for the duration of the trajectory. Furthermore, what we have implemented yet does not take into account the fact that the environnement can change as soon as the trajectory has been computed. So the next section presents an overview of what could be the future research directions.

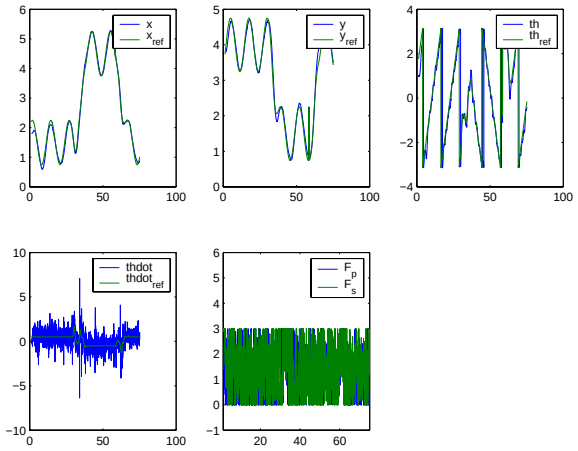

Figure 5: data from the experiment

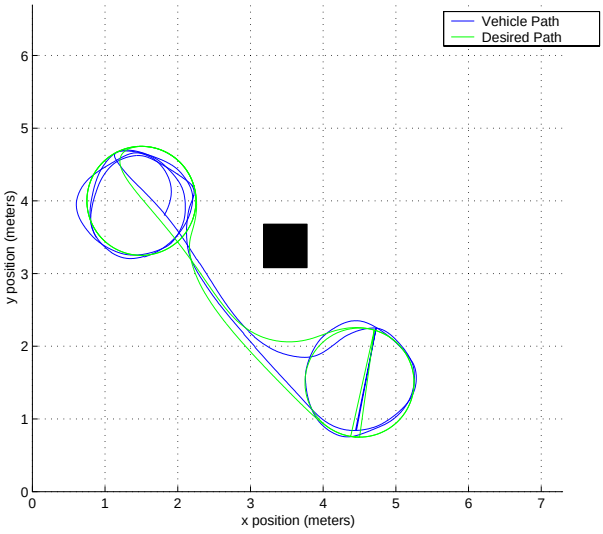

Figure 6: comparison between the trajectory followed by the real vehicle and its reference

5 Other Trajectory Generation

5.1 Minimum time trajectory generation

A way to fix the time problem is to incorporate it in the optimal problem, not as a constant but as a new variable. Let us pose: $\tau = t/T$, $(\tilde{x}(\tau), \tilde{y}(\tau), \tilde{\theta}(\tau)) = (x(t), y(t), \theta(t))$ and $(\tilde{F}_s(\tau), \tilde{F}_p(\tau)) = (F_s(t), F_p(t))$. As τ is the new time variable the dynamics become:

$$\begin{cases} m\frac{\ddot{\tilde{x}}(\tau)}{T^2} = \eta\frac{\dot{\tilde{x}}(\tau)}{T} + (\tilde{F}_s + \tilde{F}_p)\cos\tilde{\theta}(\tau) \\ m\frac{\ddot{\tilde{y}}(\tau)}{T^2} = \eta\frac{\dot{\tilde{y}}(\tau)}{T} + (\tilde{F}_s + \tilde{F}_p)\sin\tilde{\theta}(\tau) \\ J\frac{\ddot{\tilde{\theta}}(\tau)}{T^2} = \psi\frac{\dot{\tilde{\theta}}(\tau)}{T} + (\tilde{F}_s - \tilde{F}_p)r_f \end{cases}$$

Notice that the problem is still flat with the 3 outputs $\tilde{x}$, $\tilde{y}$ and T.

5.1.1 Optimization problem

We would like to minimize:

$$\int_0^1 T d\tau$$

under the constraints:

$$\begin{cases} \forall t \in [0,T] \quad 0 < z_1(t) < x_{max} \quad \text{and} \quad 0 < z_2(t) < y_{max} \\ (z_1 - x_{post})^2 + (z_2 - y_{post})^2 > R_{post}^2 \qquad \text{post} \\ 0 < \tilde{F}_s < F_{max} \quad \text{and} \quad 0 < \tilde{F}_p < F_{max} \qquad \text{forces} \end{cases}$$

The main difference between this problem and the one we previously solved is that, in this case, 16 of the 20 initial and final position constraints are nonlinear which make the problem much more difficult.

5.1.2 Problem set-up in NTG

We solved this problem using NTG with 9^{th} order B-splines for $\tilde{x}$ and $\tilde{y}$ to be able to have a very aggressive trajectory and we set a 1^{st} order B-splines to have T as a constant. The first idea was to set the coefficients for $\tilde{x}$ and $\tilde{y}$ linearly spaced between their initial and final point. These initial guesses for the coefficients worked sometimes depending on the initial guess on the final time we gave to NTG. It seems that there are a lot of local minima with relatively small basin of attraction. This problem comes from the fact that, as the vehicle are running on the horizontal plane, there is no gravity term in the dynamics. As the two directions x and y are equivalent,there are a lot of local minima which do not necessarly satisfy the constraints. Therefore different initial guesses for the time can lead to very different trajectories that are not always admissible. Since, a priori, we have no idea of what the optimal time should be, we decided to give a better initial trajectory. Instead of taking the straight line between the initial and final x we compute a 9^{th} order polynomial which verifies :

$$\begin{array}{l} P(0) = x_i, P^{(1)}(0) = x_i^{(1)}, P^{(2)}(0) = x_i^{(2)}, P^{(3)}(0) = x_i^{(3)}, \\ P^{(4)}(0) = x_i^{(4)}, P(T_{poly}) = x_f, P^{(1)}(T_{poly}) = x_f^{(1)}, \\ P^{(2)}(T_{poly}) = x_f^{(2)}, P^{(3)}(T_{poly}) = x_f^{(3)}, P^{(4)}(T_{poly}) = x_f^{(4)} \end{array}$$

Where T_{poly} is a constant and $[0, T_{poly}]$ the domain of definition of the polynomial. We make the projection of our polynomial on the B-splines basis. So we have now the coefficients of x and $\tilde{x}$, as they have the same projection on the B-splines basis.

5.2 Simulation results

Figure 7 and Figure 8 show the new trajectories computed by NTG. They satisfy the same constraints as previously. In order to find the polynomial that is used as initial guess, we have set T_{poly} to $10s$. The initial guess for the last coefficient, which determined the value of the duration of the trajectory has been also set to $10s$. Let us call this initial guess T_{ntg} and the final value of the coefficient T_{opt}. In fact it is not always as easy as in this case to find a solution. The different times, T_{opt}, T_{poly} and T_{ntg} are not necessarly equal. Figure 9 and

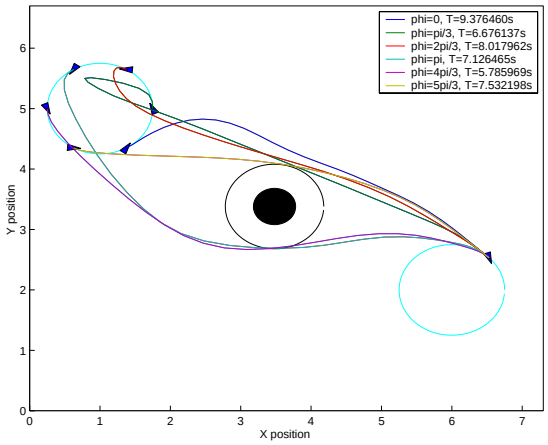

Figure 7: Six of the twelve trajectories NTG computes with minimum time -to go-

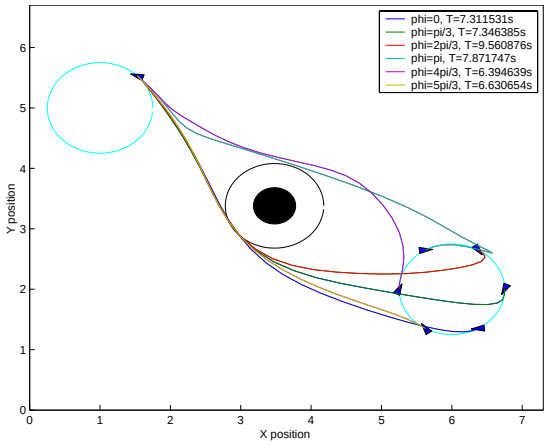

Figure 8: Six of the twelve trajectories NTG computes with minimum time -return-

Figure 10 show the influence of these parameters on the final trajectory. In the case of Figure 9 just T_{poly} is varying. There are two sets, one for each value of T_{poly} of two curves, one is the initial guess (magenta and cyan) and the other is the computed trajectory (respectively red and blue). On this figure little crosses can also been seen, these are the coefficients that are actually given to NTG, they can be blue, which means that at these points the constraints on the forces are verified or red which means the contrary. What is interesting to notice on this figure is that the computed trajectory stays very close to the initial guess, this is an illustration of the important number of minima in this problem. Figure 10 shows the influence of T_{ntg}. The trajectories that are in blue are those which satisfy the constraints, those which are in red do not satisfy the constraints. The green one is the initial guess. The good value of T_{ntg} may not be T_{poly} since the initial guess may not satisfy the constraints at all. As previously we notice that whatever the value of T_{ntg} is the general shape of the computed trajectory is never far from the shape of the initial guess.

As the vehicle is running on the horizontal plan there is no

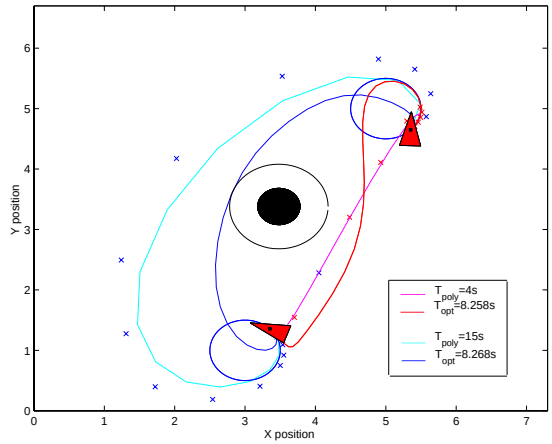

Figure 9: computed trajectories for different values of T_{poly}

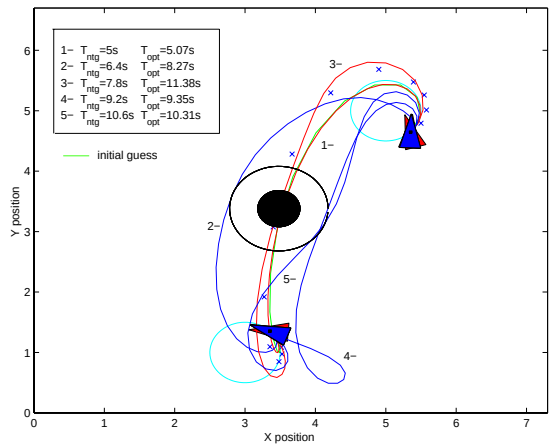

Figure 10: computed trajectories for different values of T_{ntg}

gravity term in his dynamics. Therefore the x and y axis are equivalent, which explains partially the existence of a lot of local minima. In order to be sure to converge to a minimum which satisfies the constraints we could either find a way to figure out a better initial guess or set the problem differently. One of the only parameter left to play on is T_{poly}, and we have seen the importance it can have. Experimentally we have noticed that for $T_{poly} = 10s$ NTG was converging to a proper solution almost every time (cf Figure 11, $T_{opt} = -1$ means that NTG has not been able to find a proper solution). This sounds logical since $10s$ can be considered as a characteristic time for the system and the testbed. So a solution could be to figure out what this characteristic time is for a given situation. But another one could also be to add a term to the cost function so that the system is not symmetric anymore. Anyway in both cases it lacks the capacity of facing changing environments, the MPC will fill this lack.

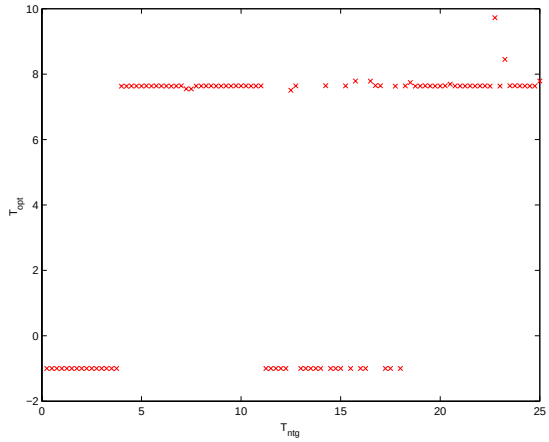

Figure 11: $T_{opt} = f(T_{ntg})$ for $T_{poly} = 10s$

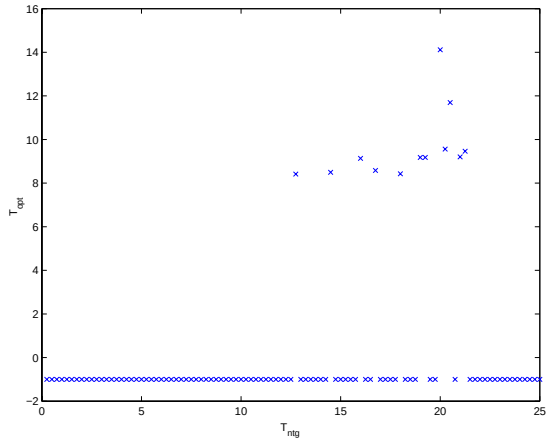

Figure 12: $T_{opt} = f(T_{ntg})$ for linearly spaced coefficients

5.3 Model Predictive Control

In Model Predictive Control, the current control action is determined by solving a finite horizon open-loop optimal control problem on-line ([9] and [10]). Each optimization yields a control law that is applied to the plant until the next sampling instant. MPC is traditionally applied to plants with dynamics slow enough to allow computations between samples. It is also one of few suitable methods in applications that can impose constraints on the states and or inputs, as the constraints are directly enforced in the on-line optimal control problem. With the advent of faster modern computers, it has become possible to extend MPC to systems governed by faster dynamics that warrant this type of solution. An example of such a system was the Caltech ducted fan, a thrust-vectored flight control experiment where actuation and spatial constraints are present[8].

5.3.1 MPC used as a controller

The first controller used to follow a circle was a LQR controller, as the error system is controllable around zero and is easy to

implement in polar coordinates. Before trying to use the MPC to go from one circle to an other circle, let us try to use this technique as a simple controller to follow a circle. Besides when we will be able to generate a trajectory from one point on a circle to an other point on another circle, we will only have to switch controller as we made for the LQR controllers.

Optimization problem Stabilizing the vehicle around a circular trajectory is equivalent to stabilizing the error of the state around zero. So we pose the problem as :

$$\min_{\bar{X}_e} \int_{t_0}^{t_0+T} (\bar{X}_e(t)^t Q \bar{X}_e(t))dt + V(\bar{X}_e(T+t_0))$$

Under the constraints :

$$\bar{X}(t_0) = \bar{X}_{t_0}$$

where $\bar{X}_e = \bar{X} - \bar{X}_{ref}$,with $\bar{X}$ the extended state, $\bar{X}_{ref}$ the extended reference's state and $\bar{X}_{t_0}$ the real state of the vehicle.

Resolution We solve this problem using the semi-flat system (3 outputs: x,y and θ). For that we use 57 7^{th} B-splines whith 21 breakpoints. The semi-flat system seems to be the simplest and fastest way to implement it. The semi-flat system has simpler expression than the flat one, because we want quadratic costs (unintegrated and final) to assure the stability around the circle. Indeed, the error dynamics is C^∞ and controllable around any equilibrium point, the unintegrated cost is C^∞ and convex for both X and u, and we use a final cost such that :

$$\forall X \neq X_f \min_{u \in \mathcal{U}_{adm}} [\dot{V} + q](X,u) < 0$$

By using the Riccatti solution for the error dynamic, we are guaranteed to converge([9] and [10]).
With the flat system, we would have 4 nonlinear initial constraints (θ, $\dot{\theta}$,F_s and F_p) and the two last one are very difficult to compute. Instead we have only 2 nonlinear constraints (F_s and F_p), and one nonlinear constraint (the dynamic constraint) which seems to be easier.
Moreover, the MPC controller is in theory better than the LQR controller because the MPC controller looks at the future of the trajctory and tries to minimize the error along the trajectory. The main advantage of MPC over LQR is that it handles nonlinear plants and constraints.

5.3.2 Generation and control of nonlinear trajectories with MPC

The quality of the MPC is not only to be a very good controller. We can use it as a way to generate a trajectory from one point to an other one, without adding a controller. A feasible trajectory is computed from the current position to the desired position over a finite time horizon T, used for a short period of time $\delta < T$, and then recomputed based on the new position. The main advantage of this technique is that the system can react to new situations (presence of an obstacle,movement of this obstacle, changement of the capacity of a vehicle (a fan can be dammaged and have a lost of thrust, ...). However to be able to use that, we must garantee the convergence of the algorithm at each computation, and garantee the fastness of the convergence (How can we face a varying environment if we need 5s to compute a trajectory ?).

Optimization problem In MPC, the current optimal control

$$\min_{Z_1,Z_2} \int_{t_0}^{T+t_0} q(Z_1(t),Z_2(t))dt + V(Z_1(T+t_0),Z_2(T+t_0))$$

under the constraints:

$$\begin{cases} Z(t_0) = Z_{t_0} & \text{initial} \\ \forall t \in [t_0, T+t_0] \quad t_c(Z_1(t),Z_2(t)) = 0 & \text{trajectory} \end{cases}$$

No terminal constraint is enforced in this study. In theory, the resulting control $\overset{*}{T}(\cdot)$ is instantaneously applied until a new state update occurs, usually at a prespecified sampling interval of time δ seconds. Repeating these computations yields a feedback control law.

Resolution We solve this problem using flat coordinates with 26 7^{th} order B-splines and with 31 breakpoints. The horizon time is $T = 12s$, and we apply it during $\delta = .5s$. Unfortunately we cannot garantee the convergence of this technique in a finite time because the system's dynamics is not controllable around any equilibrium point.
Moreover, if the algorithm does not converge to an optimal solution (it can find a local minima, or no admissible solution), we stay on the previous trajectory.

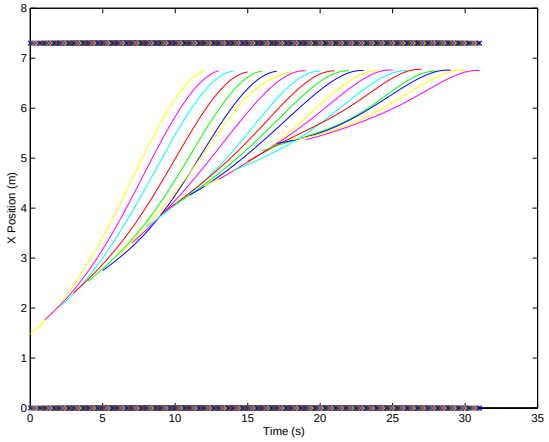

Figure 13: evolution of x with the MPC method

Simulation Results Figure 13, Figure 14 and Figure 15 represent all the results of the computations (a new color represents a new computation). One important thing to notice is the continuity of x and the discontinuity of F_p. These come from the initial constraint we impose. We only impose the new trajectory to have the same position (x and y) and the same

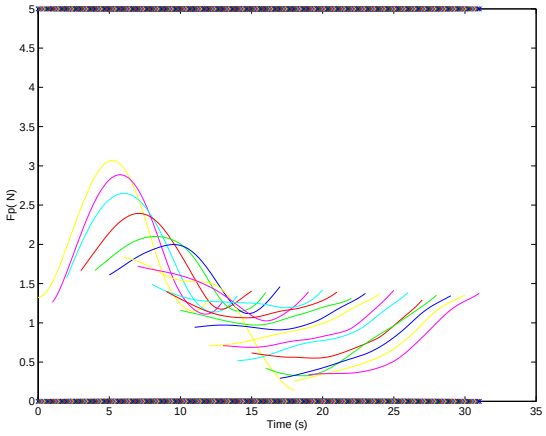

Figure 14: evolution of F_p with the MPC method

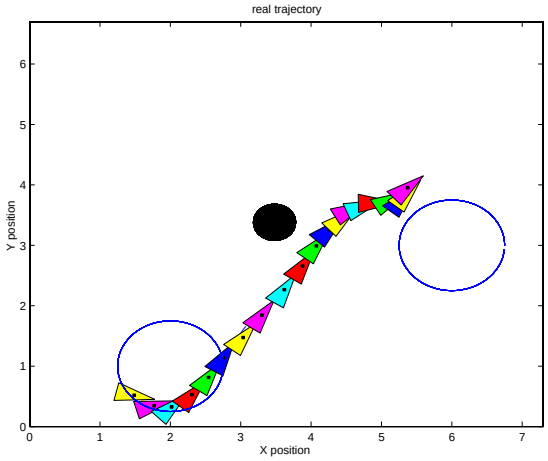

Figure 16: trajectory of the vehicle with the MPC method

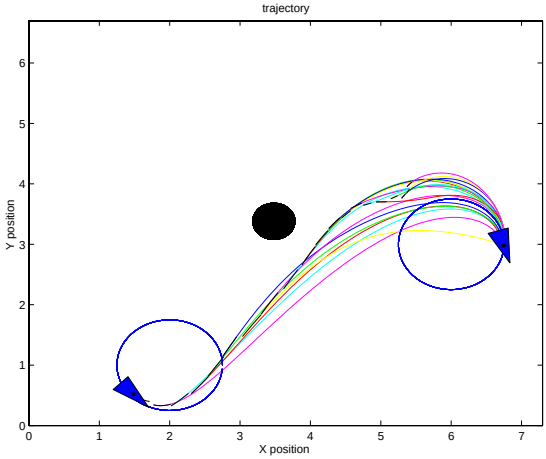

Figure 15: evolution of trajectories with the MPC method

velocity ($\dot{x}$ and $\dot{y}$) as the previous one after δ. This allow the vehicle to have a smooth trajectory, but this does not guarantee the continuity of the forces. That is why we want to recompute it as often as we can in order to have admissible jumps on the forces. Besides we can see on Figure 16 (each arrow represents the position where we begin of a new computation) that we do not converge on the circle. Even if we try a longer simulation time, we will not go to the circle, but we will get closer. As we keep the same time horizon, the problem is that there are two many solutions for the problem and we are only sure that we get closer to the circle. We have the asymptotic convergence of the vehicle. As we showed previously that we have a robust controller to track a circle, the idea is to switch the two MPC as soon as we are close enough to the circle.

With this method, we are sure that we will get closer and closer to the circle, but we have no guarantee of the fastness of the convergence. We would have better results if we had a very small time horizon T (2s for example), so that even if the vehicle is close the optimization problem still makes sense. A way to solve this problem can be to have a varying time horizon. Thanks to the optimal principle,we know that if we have a solution for a time horizon T, by following this trajectory during δs, we will have a solution for the new problem with a time horizon $T_1 = T - \delta$ which will be the same trajectory on $[\delta, T]$. As we are fixing the time horizon, we do not take the distance we must do into account. It seems more logical to have a time horizon proportionnal to to distance to the final point.

6 Conclusion and Future work

The contents of this report summarize the evolution of the trajectory generation on one vehicle. Due to the nonlinear controllability, we realized in Section 3 trajectories from one circle to an other circle. We implemented this type of trajectories in Section 4 but for each trajectory, we need to fix by hand a parameter which needs to be solved by an other formulation of the problem. Moreover we can not face a varying environment. Section 5 exposes how we tried to solve this problem. We first tried to implement minimum time trajectories in order not to have to fix the time's parameter, and then Model Predictive Control to be able to face a varying environment.

Distributed systems that are dynamic, particulary multi-vehicle coordination problems, are becoming more important in engineering applications. Many of these systems are governd by constraints, e.g. network band-width, control input saturation, and spatial limitations on the state of the system. The MPC theory is very attractive, thanks to improvements of computing power, and seems to be a bright way to face a varying space. The Caltech Ducted Fan showed us that it is possible to recompute trajectories on-line for a dynamical fast system. A part of the challenge of the MVWT project is the extention of this success to a nonlinearly controllable system.

The future of the project seems clear. It is the most natural to continue the research on both themes that are minimum time and MPC. Future work on minimum time will consist in still searching to understand the problem of the situation. The problem is to know precisely the influence of each B-spline's coefficients on the trajectory and to have a better numerical

conditionment for the problem. This will answer the question: "Do we have to break the symmetry by using a final cost?". Future work on MPC consists in finding a good final cost for the optimization problem in order to be able to have the fastest solution we can. This will be linked to an improvement on the theory on Model Predictive Control when the system is not linearly controllable.

References

[1] L. Cremean and al. The Caltech multi-vehicle wireless testbed. In *Submitted: 2002 Conference on Decision and Control*, Las Vegas, NV, 2002

[2] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6):1327-1360, 1995.

[3] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. A Lie-Backlund approach to equivalence and flatness of nonlinear systems.*IEEE Trans. Auto. Cont.*, 44(5), 928-937.

[4] Mark B. Milam, Kudah Mushambi, and Richard M. Murray. A computational approach to real-time trajectory generation for constrained mechanical systems. In *Proceedings of the 2000 Conference on Decision and Control*, Sydney, Australia, 2000.

[5] Nicolas Petit, Mark B. Milam, Richard M. Murray. Inversion based constrained trajectory optimization. *IFAC Symposium on Nonlinear Control Systems Design*, (NOLCOS),2001.

[6] E. Klavins and U. Saranli. Object orient state machines. *Embedded Systems Programming Magazine*, 2002. In Press.

[7] U. Saranli, M. Buehler, and D. E. Koditschek. *RHex: A simple and highly mobile hexapod robot. The International Journal of robotics Research*, 20(7):616-631, July 2001.

[8] Ryan Franz, Mark B. Milam and John Hauser. Applied receding horizon control of the Caltech ducted fan.

[9] William B. Dunbar, Mark B. Milam, Ryan Franz, and Richard M. Murray. Model predictive control of a thrust-vectored flight control experiment. In *2002 IFAC World Congress*, Barcelona, Spain, 2002.

[10] William B. Dunbar and Richard M. Murray. Model predictive control of coordinated multi-vehicleformations. In *2002 Conference on Decision and Control*, Las Vegas, NV, 2002.