Citation	<p>Van Loock, W., Bellens, S., Pipeleers, G., De Schutter, J., Swevers, J. (2013)</p> <p>Time-optimal parking and flying: Solving path following problems efficiently</p> <p>Proc. of the 2013 IEEE International Conference on Mechatronics (ICM). IEEE International Conference on Mechatronics. Vicenza, 27-28 February, 1 March 2013 (pp. 840-845)</p>
Archived version	Author manuscript: the content is identical to the content of the published paper, but without the final typesetting by the publisher
Published version	http://dx.doi.org/10.1109/ICMECH.2013.6519150
Publisher homepage	http://ieeexplore.ieee.org/servlet/opac?punumber=6516157
Author contact	<p>E-mail: wannes.vanloock@kuleuven.be</p> <p>Phone number: +32 16 32 92 64</p>
IR	https://lirias.kuleuven.be/handle/123456789/358305

(article begins on next page)

Time-optimal parking and flying: Solving path following problems efficiently

Wannes Van Loock, Steven Bellens, Goele Pipeleers, Joris De Schutter, Jan Swevers

Department of Mechanical Engineering, Division PMA, KU Leuven, Celestijnenlaan 300B, B-3001 Heverlee, Belgium
wannes.vanloock@mech.kuleuven.be

Abstract—Path following deals with the problem of following a geometric path without any preassigned timing information and constitutes an important step in solving the general motion planning problem. The current paper considers path following for differentially flat systems. In this case the dynamics of the system can be projected along the path to a single input system, resulting in a free end-time optimal control problem. We propose to rewrite the problem in terms of the velocity along the path and the path itself. This way, we arrive at a fixed end-time optimal control problem that can be solved efficiently by interior-point solvers. Two challenging examples, a truck-trailer parking simulation and a quadrotor mission, illustrate the efficiency of the problem formulation and the practicality of the developed software.

I. INTRODUCTION

As opposed to trajectory tracking where a time-varying reference is tracked, path following deals with the problem of following a geometric path without any preassigned timing information. Path following is often considered as the low level stage in a decoupled motion planning problem [1]. First, a high level planner determines a geometric path taking into account geometric constraints, e.g. obstacle avoidance. Subsequently, an optimal motion along the geometric path is determined whereby the system dynamics and limitations, such as actuator constraints, are accounted for. In fact, many industrially relevant problems are path following problems. To name but a few: robot path following [2], [3], [1] in e.g. welding or painting, control of autonomous vehicles [4], [5], but also control for chemical reactors [6] and batch crystallization [7].

Only recently system limitations are being considered in an optimal control framework for path following. Most methods employ the fact that motion along a path can be described by a single path coordinate s [1], [3], which allows projecting the dynamics of the system along the path to a single-input system. Using this reformulation, the authors of [6] consider path following for differentially flat systems [8]. However, according to the authors the path cannot be self-intersecting and the complexity of the optimization problem depends on the path geometry. In [2], the authors transform the time-optimal path following problem for the specific model structure of robotic manipulators into a convex optimization problem by reformulating the problem in terms of the velocity along the path. The present paper extends this transformation for the more general case of differentially flat systems. The approaches from [2] and [6] are unified into a framework for solving path following problems. A small dimensional fixed end-time

optimal control problem is derived, which is efficiently solved by interior-point optimizers. Contrary to [6], the problem complexity does not depend on the path parametrization.

To simplify the implementation of the optimization problem a software package is developed in Python [9] using Ipopt [10] as a solver. The purpose of this paper is to explain the theory behind the software and illustrate the use of it by means of two examples: a truck-trailer parking simulation and a quadrotor mission.

Section II recalls the notion of differentially flat systems and addresses the basic path-following problem. By projecting the system dynamics onto the path, we arrive at a small dimensional, free end-time optimal control problem. Our main theoretical contribution is stated in Section III. The free end-time optimal control problem is transformed into a fixed end-time problem of smaller dimension by rewriting the problem in terms of the velocity along the path. Finally, Section IV briefly details the developed software such that path following problems can be defined easily. Subsequently, it is illustrated by means of two challenging non-linear system examples.

In the following, we will use $\partial_\tau^k g(\tau)$ to denote the k -th derivative of $g(\tau)$ with respect to τ . For small k , we will use the shorthand notation $\dot{g}$ to denote the derivative with respect to time.

II. PROBLEM STATEMENT

A. Differential flatness

Before addressing the path-following problem, let us first recall the notion of differential flatness, introduced in [8].

Definition 1 (Differentially flat system): The system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \quad (1)$$

with $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$, $m \leq n$, is differentially flat if there exists a variable $\mathbf{z} \in \mathbb{R}^m$, the so-called flat output, such that

$$\mathbf{z}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \dots, \partial_t^\alpha \mathbf{u}(t))$$

and

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{\Phi}(\mathbf{z}(t), \dots, \partial_t^{\beta-1} \mathbf{z}(t)), \\ \mathbf{u}(t) &= \mathbf{\Psi}(\mathbf{z}(t), \dots, \partial_t^\beta \mathbf{z}(t)), \end{aligned} \quad (2)$$

where $\mathbf{\Phi}$ and $\mathbf{\Psi}$ are smooth functions and $\alpha, \beta \in \mathbb{N}$.

The flat output characterizes all the state motions and corresponding input history. It is a powerful concept in motion planning as it avoids integration of the differential equations.

In the following we consider systems of the form (1) with output $\mathbf{y} \in \mathbb{R}$ and we assume that the system is differentially flat with flat output $\mathbf{z} = \mathbf{y}$.

B. Optimal path following problem

Consider a geometric reference $\mathbf{y}_r(s) \in \mathcal{C}^\beta$, a regular, i.e. $\partial_s \mathbf{y}_r(s) \neq \mathbf{0}, \forall s \in [0, 1]$, parametrized curve as a function of a scalar path coordinate s . The time dependency follows from the relation $s(t)$. Without loss of generality it is assumed that the trajectory starts at $t = 0$, ends at $t = T$ and that $s(0) = 0 \leq s(t) \leq s(T) = 1$. Furthermore, it is assumed that the velocity along the path is non-negative, i.e. $\dot{s}(t) \geq 0$ and that the boundary conditions $\mathbf{x}_0, \mathbf{x}_T$ are consistent with $\mathbf{y}_r(s)$, such that $\mathbf{x}_0 = \Phi(\mathbf{y}_r(0), \dots, \mathbf{y}_r^{(\beta-1)}(0))$ and $\mathbf{x}_T = \Phi(\mathbf{y}_r(1), \dots, \mathbf{y}_r^{(\beta-1)}(1))$.

In this paper we want to find an input signal $\mathbf{u}(t)$ such that the geometric reference is followed exactly by the system output $\mathbf{y}(t) = \mathbf{z}(t)$:

$$\mathbf{z}(t) = \mathbf{y}_r(s(t)),$$

without violating constraints on states and inputs

$$\mathbf{u}(t) \in \mathcal{U}, \mathbf{x}(t) \in \mathcal{X},$$

in minimal time T .

For arbitrary non-linear systems this problem is challenging to solve as it requires integrating a high dimensional non-linear state space model. However when considering flat systems the system dynamics can be projected along the path onto a linear single-input system that is trivial to integrate [6]. By applying the chainrule we find the time derivatives of $\mathbf{y}_r(s)$

$$\begin{aligned} \dot{\mathbf{y}}_r(s) &= \partial_s \mathbf{y}_r(s) \dot{s}, \\ \ddot{\mathbf{y}}_r(s) &= \partial_s^2 \mathbf{y}_r(s) \dot{s}^2 + \partial_s \mathbf{y}_r(s) \ddot{s}, \end{aligned} \quad (3)$$

and so on.

By substituting (3) into (2), we can rewrite the states and inputs to the system as (with slight abuse of notation):

$$\begin{aligned} \mathbf{x} &= \Phi(s, \dot{s}, \dots, \partial_t^{\beta-1} s) \\ \mathbf{u} &= \Psi(s, \dot{s}, \dots, \partial_t^\beta s). \end{aligned}$$

The path following problem can now be formulated as the small dimensional optimal control problem with control input $\partial_t^\beta s(t)$ and states $s(t)$ up to $\partial_t^{\beta-1} s(t)$

$$\begin{aligned} &\underset{s(\cdot), T}{\text{minimize}} && T \\ &\text{subject to} && \partial_t^i s(0) = s_{0i}, \partial_t^i s(T) = s_{Ti}, i = 0, \dots, \beta - 1 \\ & && T \geq 0 \\ & && \dot{s}(t) \geq 0, \forall t \in [0, T] \\ & && \Phi(s(t), \dot{s}(t), \dots, \partial_t^{\beta-1} s(t)) \in \mathcal{X}, \forall t \in [0, T] \\ & && \Psi(s(t), \dot{s}(t), \dots, \partial_t^\beta s(t)) \in \mathcal{U}, \forall t \in [0, T], \end{aligned} \quad (4)$$

where s_{0i} and s_{Ti} are chosen such that $\mathbf{x}_0 = \Phi(s_{00}, \dots, s_{0\beta-1})$ and $\mathbf{x}_T = \Phi(s_{T0}, \dots, s_{T\beta-1})$. Although this reformulation simplifies the problem to a great extent, it suffers from three

important drawbacks: Firstly, it requires additional assumptions on the geometry of the path, such as the existence of $\mathbf{y}^{-1}(s)$ [11]. Secondly, because $s(t)$ is an optimization variable, the geometric path itself enters in the problem formulation through $\mathbf{y}_r(s(t))$. Therefore strongly non-linear paths will inherently make the problem harder to solve. Thirdly, the problem is a free end-time optimal control problem for which the solution can vary quite non-linearly with changes in the end-time. In the following section we show how to overcome these difficulties by a transformation of variables.

III. PROBLEM TRANSFORMATION

The key idea is to transform the problem such that, instead of time t , the path coordinate s becomes the independent variable. This way we can get rid of the additional assumptions on $\mathbf{y}_r(s)$. Moreover, the problem is transformed into a fixed end-time optimal control problem. To this end we follow the transformation proposed in [2] and parametrize the velocity along the path as a function of s :

$$b(s) = \dot{s}^2. \quad (5)$$

Taking the derivative with respect to time on both sides of (5), yields $2\dot{s}\ddot{s} = \partial_s b(s) \dot{s}$ or

$$\ddot{s} = \frac{\partial_s b(s)}{2}.$$

By repeatedly applying the chain-rule, we find

$$\begin{aligned} \partial_t^3 s &= \frac{\partial_s^2 b(s) \dot{s}}{2} = \frac{\partial_s^2 b(s) \sqrt{b(s)}}{2}, \\ \partial_t^4 s &= \frac{\partial_s^3 b(s) \ddot{s}}{2} + \frac{\partial_s^3 b(s) \dot{s}^2}{2} = \frac{\partial_s^3 b(s) \partial_s b(s)}{4} + \frac{\partial_s^3 b(s) b(s)}{2}, \end{aligned}$$

and so on.

States and inputs can now be reformulated as a function of $\mathbf{y}_r(s)$ and $b(s)$:

$$\begin{aligned} \mathbf{x} &= \Phi(\mathbf{y}_r(s), b(s), \dots, \partial_s^{\beta-2} b(s)) \\ \mathbf{u} &= \Psi(\mathbf{y}_r(s), b(s), \dots, \partial_s^{\beta-1} b(s)). \end{aligned}$$

Furthermore, using $dt = \frac{ds}{\dot{s}}$ the objective function is reformulated as

$$T = \int_0^T 1 d\tau = \int_0^1 \frac{1}{\sqrt{b(s)}} ds.$$

With the above transformation the optimization problem (4) is reformulated as the optimal control problem with pseudo-time s , differential states $b(s)$ up to $\partial_s^{\beta-2} b(s)$ and control $\partial_s^{\beta-1} b(s)$

$$\begin{aligned} &\underset{b(\cdot)}{\text{minimize}} && \int_0^1 \frac{1}{\sqrt{b(s)}} ds \\ &\text{subject to} && \partial_s^i b(0) = b_{0i}, \partial_s^i b(1) = b_{Ti}, i = 0, \dots, \beta - 2 \\ & && b(s) \geq 0, \forall s \in [0, 1] \\ & && \Phi(\mathbf{y}_r(s), b(s), \dots, \partial_s^{\beta-2} b(s)) \in \mathcal{X}, \forall s \in [0, 1] \\ & && \Psi(\mathbf{y}_r(s), b(s), \dots, \partial_s^{\beta-1} b(s)) \in \mathcal{U}, \forall s \in [0, 1], \end{aligned} \quad (6)$$

where b_{0_i} and b_{T_i} are chosen such that $\mathbf{x}_0 = \Phi(\mathbf{y}_r(0), b_{0_0}, \dots, b_{0_{\beta-2}})$ and $\mathbf{x}_T = \Phi(\mathbf{y}_r(1), b_{T_0}, \dots, b_{T_{\beta-2}})$.

This reformulation holds several advantages over (4). Firstly, it contains one differential state fewer, and is therefore a smaller dimensional optimal control problem. Secondly, the optimal control problem now has a fixed end-time ($s = 1$), which simplifies the problem to great extent. Thirdly, additional assumptions on $\mathbf{y}_r(s)$ can be dropped as s is the independent variable in the problem, e.g. $\mathbf{y}_r(s)$ can be self-intersecting. Finally, this also means that the geometry of the path no longer influences the complexity of the problem.

IV. EXAMPLES AND SOFTWARE

As is clear from the previous section, deriving the transformed optimization problem requires many tedious manipulations. To alleviate this burden, a software package is developed to easily model path following problems. The problems are modeled using Python and CasADi [12] and discretized using a direct transcription method. As a solver we use Ipopt [10], an interior-point solver for large non-linear programs. In the following, the software is briefly explained and is illustrated by means of two challenging non-linear systems, a truck-trailer parking simulation and a quadrotor mission. The software and examples are available for download via <http://people.mech.kuleuven.be/~u0063507/pathfollowing/>.

A. Software

The software supports the definition of differentially flat systems and path following problems. To define a flat systems with m flat outputs where the highest derivative is β is defined by

```
S = FlatSystem(m, beta ).
```

The j -th derivative of flat output i is available via the attribute `y[i, j]`. States and inputs can be defined by using the instance methods `set_state(expr, 'name')`. Already defined states are available via the attribute `x['name']`. The instance method `dt(expr)` can be used to easily calculate an expression for the time derivative of `expr`.

A pathfollowing problem for the differentially flat system S is defined with

```
P = PathFollowing(S).
```

The i -th time derivative of the path variable s is stored in the attribute `in s[i]`. The instance method `set_path(expr)`, where `expr` must be a function of `s[0]`, defines the path to be followed. Constraints can be set using the instance method `set_constraint(expr, lb, ub)` where `lb` and `ub` are values for lower and upper bounds respectively. Various solver options can be set using `set_options({'option': value})`. Aside from all supported options of Ipopt in CasADi, the following options are available

- 'N': The number of discretization steps
- 'Nr': The number of returned time points in the solution
- 'reg': A regularization factor added to the goal function to avoid singular arcs in the solution

Fig. 1. The truck-trailer: notations

Finally, the instance method `solve()`, solves the problem. The solution is stored in the attribute `P.sol`. To easily plot all states, inputs and constraints, the instance method `plot()` can be used.

B. Parking a truck with trailers

In this section we consider steering a truck with a trailer along a geometric reference. The hitch of the trailer is connected to the rear axle of the truck. The inputs to the system are the driving velocity v , and steering velocity $\dot{\phi}$. Figure 1 shows the considered setup and the notations. The flat output for this system is the Cartesian coordinate, (x_1, y_1) of the medium point of the axle of the trailer [8]. Indeed

$$\begin{aligned} x_0 &= x_1 + d_1 \cos(\theta_1) \\ y_0 &= y_1 + d_1 \sin(\theta_1) \\ \cos(\theta_0)v &= \dot{x}_0 \\ \tan(\phi)v &= d_0 \dot{\theta}_0, \end{aligned}$$

and $\tan(\theta_i) = \frac{\dot{y}_i}{\dot{x}_i}$, $i = 0, 1$.

The constraints $\theta_0 - \theta_1, \phi \in]-\frac{\pi}{2}, \frac{\pi}{2} [$ rad avoid singularities, which might occur whenever $\dot{x}_i = 0$. Furthermore to avoid slipping of the wheels $v, \dot{v}$ and $\dot{\phi}$ are constrained to ± 10 m/s, ± 20 m/s² and $\pm \frac{\pi}{3}$ rad/s respectively:

$$\begin{aligned} \mathbf{x} &= (\theta_0 - \theta_1, \phi, v)^T \in]-\frac{\pi}{2}, \frac{\pi}{2} [\text{ rad} \times \\ &\quad]-\frac{\pi}{2}, \frac{\pi}{2} [\text{ rad} \times [-10, 10] \text{ m/s} \\ \mathbf{u} &= (\dot{v}, \dot{\phi})^T \in [-20, 20] \text{ m/s}^2 \times [-\frac{\pi}{3}, \frac{\pi}{3}] \text{ rad/s}. \end{aligned}$$

Consider parking a truck of $d_0 = 3.5$ m with a trailer of $d_1 = 5.5$ m where the center of the trailer axle follows the geometric trajectory

$$\begin{pmatrix} x_1(s) \\ y_1(s) \end{pmatrix} = \begin{pmatrix} 17 - 17p(s) \\ 2p(s)^4 (35 - 84p(s) + 70p(s)^2 - 20p(s)^3) \end{pmatrix},$$

with

$$p(s) = s^2(s + 3(1 - s)),$$

such that the trailer performs a smooth motion from $(17, 0)$ m to $(0, 2)$ m and $\theta_i(0) = \theta_i(T) = \phi(0) = \phi(T) = v(0) = v(T) = \dot{\phi}(0) = \dot{\phi}(T) = 0$. The code for solving the truck-trailer problem is given in Appendix A. A solution is obtained in 8.8 s of which 0.3 s in Ipopt and 8.5 s in non-linear function evaluations on a 2.8 GHz computer with 4 GB RAM using the automatic initialization of Ipopt. Note that most time is spent in non-linear function calls. This time can be reduced to a large extent by compiling the functions. Figure 2 shows the optimal velocity, acceleration and steering velocity to accomplish the

Fig. 2. Time optimal controls for parking a truck-trailer: driving velocity v (top) and steering velocity $\dot{\phi}$ (bottom)

Fig. 3. Time optimal truck trailer parking maneuver

desired trajectory. Note that at every time instant either the velocity, acceleration or steering velocity constraint is active as expected for a time-optimal motion profile. The constraints on $\theta_0 - \theta_1$ or ϕ are never active. Figure 3 visualizes the movement of the truck and trailer in 20 equal time steps.

C. Flying a quadrotor

A second example considers flying a quadrotor time-optimally along a predetermined geometric reference. Figure 4 illustrates the used notation. The quadrotor is differentially flat with flat outputs $(x, y, x, \psi)^T$ the coordinates of the center of gravity of the quadrotor and the yaw angle. In [13], a detailed derivation of the flatness is found. The control input $\mathbf{u} = (u_1, u_2, u_3, u_4)^T$ consists of the net body force u_4 and the three body moments u_1, u_2, u_3 . The inputs are constrained to

$$\mathbf{u} \in [-8, 8] \text{ Nm} \times [-8, 8] \text{ Nm} \times [-8, 8] \text{ Nm} \times [1, 32] \text{ N}.$$

Consider flying the quadrotor along the trajectory

$$(x(s), y(s), z(s), \psi(s)) = (3, 3, 5, 0.5\pi)p(s),$$

with

$$p(s) = s^4 (35 - 84s + 70s^2 - 20s^3)$$

such that the quadrotor performs a smooth rest-to-rest movement from $(0, 0, 0, 0)$ to $(3 \text{ m}, 3 \text{ m}, 5 \text{ m}, 90^\circ)$. In appendix B

Fig. 4. The quadrotor with coordinates (x, y, z) of the center of gravity and roll, pitch, yaw angles (θ, ϕ, ψ)

Fig. 5. Time optimal controls for a rest-to-rest movement with a quadrotor

the Python code is given that is used to solve the problem. The solution is obtained in 28.7 s of which 0.4 s in Ipopt and 28.3 s in non-linear function evaluations. By compiling the code the time in non-linear function calls can be greatly reduced.

The optimal controls to accomplish the desired trajectory time-optimally are shown in figure 5. Again, note that at each time instant at least one constraint is active. Also note the aggressive flight angle ψ in figure 6 when breaking to arrive at standstill in $(3, 3, 5)$. Figure 7 shows the flight of the quadrotor in 35 equal time steps.

V. CONCLUSION

The presented paper and software deals with path following problems for (non- linear) differentially flat systems, in which a geometric path without any preassigned timing information is to be followed. The system dynamics are projected along the path onto a single input system, resulting in a small dimensional free end- time optimal control problem. The main contribution

Fig. 6. Euler angles ϕ and θ for a time optimal rest-to-rest movement with a quadrotor

Fig. 7. Time optimal quadrotor flight in 35 time steps

of the paper is a transformation of this problem in terms of the velocity along the path, which allows rewriting the problem as a fixed end-time optimal control problem that is efficiently solved by interior-point solvers.

As the transformation requires many tedious manipulations, a Python software package based on CasADi and Ipopt has been developed, which allows for simple definition of the path following problem. The use and practicality of the software is illustrated by means of two challenging test cases, a truck-trailer parking simulation and a quadrotor. Even though both examples are highly non-linear systems, optimal solutions are calculated efficiently thanks to the proposed transformation.

ACKNOWLEDGEMENT

Goele Pipeleers is a Postdoctoral Fellow of the Research Foundation – Flanders (FWO – Vlaanderen). This work benefits from KU Leuven – BOF PFV/10/002 Center-of-Excellence Optimization in Engineering (OPTEC), the Belgian Programme on Interuniversity Attraction Poles, initiated by the Belgian Federal Science Policy Office (DYSCO), G.0377.09 of the Research Foundation – Flanders (FWO – Vlaanderen), and KU Leuven’s Concerted Research Action GOA/10/11.

APPENDIX

The appendix contains the source code used to run the examples of the truck- trailer parking simulation and the quadrotor.

A. Truck-trailer

Note that a small factor ϵ is added to avoid singularities when $\tan \theta_i = 0/0$.

```
from pathfollowing import *
from casadi import *

d0, d1, e, pi = 3.5, 5.5, 1e-8, 3.14159265

# Define flat system
S = FlatSystem(2, 4)
y = S.y
S.set_state(y[0, 0], 'x1')
S.set_state(y[1, 0], 'y1')
# assures angle is between -pi/2 and pi/2
S.set_state(arctan(tan(arctan2(y[1, 1], y[0, 1] + e))), 'th1')
S.set_state(y[0, 0] + d1 * cos(S.x['th1']), 'x0')
S.set_state(y[1, 0] + d1 * sin(S.x['th1']), 'y0')
S.set_state(S.dt(S.x['x0']), 'vx0')
S.set_state(S.dt(S.x['y0']), 'vy0')
S.set_state(arctan(tan(arctan2(S.x['vy0'], S.x['vx0'] + e))), 'th0')
S.set_state(S.dt(S.x['th0']), 'dth0')
S.set_state(S.x['vx0'] / cos(S.x['th0']), 'v')
S.set_state(S.dt(S.x['v']), 'dv')
S.set_state(
    arctan(tan(arctan2(S.x['dth0'] * d0, S.x['v'] + e))),
    'phi')
S.set_state(S.dt(S.x['phi']), 'dphi')

# Path tracking problem
P = PathFollowing(S)
ps = P.s[0] ** 2 * (P.s[0] + 3 * (1 - P.s[0]))
x1 = 17 - 17 * ps
y1 = 2 * ps ** 4 (ps ** 3 + 7 * ps ** 2 (1 - ps) +
    21 * ps * (1 - ps) ** 2 + 35 * (1 - ps) ** 3)
P.set_path([x1, y1])
P.set_constraint('v', -10, 10)
P.set_constraint('dv', -20, 20)
P.set_constraint(S.x['dphi'], -np.pi / 3, np.pi / 3)
P.set_options({'N': 199, 'Nt': 499, 'reg': 1e-15})
P.solve()
P.plot()
```

B. Quadrotor

```
from pathfollowing import *
from casadi import *

G = 9.81
m = 1.35
Jx, Jy, Jz, I = 0.1325, 0.1325, 0.2651, 0.235

# Define flat system
```

```

S = FlatSystem(4, 5)
y = S.y
S.set_state (
    m * sqrt(y[0, 2] ** 2 + y[1, 2] ** 2 + (y[2, 2] - G) ** 2),
    'u4')
S.set_state (
    arcsin(-m * (y[0, 2] * sin(y[3, 0]) -
        y[1, 2] * cos(y[3, 0])) / S.x['u4']),
    'theta')
S.set_state (
    arcsin(-m * (y[0, 2] * cos(y[3, 0]) + y[1, 2] * sin(y[3, 0])) /
        (S.x['u4'] * cos(S.x['theta']))),
    'phi')
S.set_state (S.dt(S.x['theta']) - sin(S.x['phi']) * y[3, 1], 'wx')
S.set_state (
    cos(S.x['theta']) * S.dt(S.x['phi']) +
    sin(S.x['theta']) * cos(S.x['phi']) * y[3, 1],
    'wy')
S.set_state (
    -sin(S.x['theta']) * S.dt(S.x['phi']) +
    cos(S.x['theta']) * cos(S.x['phi']) * y[3, 1],
    'wz')
S.set_state (
    (Jx * S.dt(S.x['wx']) + (Jz - Jy) * S.x['wy'] * S.x['wz']) / I,
    'u2')
S.set_state (
    (Jy * S.dt(S.x['wy']) + (Jx - Jz) * S.x['wx'] * S.x['wz']) / I,
    'u1')
S.set_state (Jz * S.dt(S.x['wz']) + (Jy - Jx) * S.x['wx'] * S.x['wy'],
    'u3')

# Path tracking problem
P = PathFollowing(S)
s = P.s[0]
ps = s ** 4 * (35 - 84 * s + 70 * s ** 2 - 20 * s ** 3)
path = [3 * ps, 3 * ps, -5 * ps, 0.5 * np.pi * ps]
P.set_path(path)
P.set_constraint('u1', -8, 8)
P.set_constraint('u2', -8, 8)
P.set_constraint('u3', -8, 8)
P.set_constraint('u4', 1, 32)
P.set_options({'Nt': 499, 'N': 199, 'reg': 1e-15})
P.solve()

```

REFERENCES

- [1] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *Automatic Control, IEEE Transactions on*, vol. 30, no. 6, pp. 531–541, 1985. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1104009
- [2] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *Automatic Control, IEEE Transactions on*, vol. 54, no. 10, pp. 2318–2327, oct. 2009.
- [3] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985. [Online]. Available: <http://ijr.sagepub.com/content/4/3/3.abstract>
- [4] R. Skjetne, T. I. Fossen, and P. V. Kokotovi, "Robust output maneuvering for a class of nonlinear systems," *Automatica*, vol. 40, no. 3, pp. 373–383, 2004. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0005109803003467>
- [5] G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter trajectory tracking control," *Electrical Engineering*, vol. 44, no. August, pp. 1–14, 2008.
- [6] T. Faulwasser, V. Hagenmeyer, and R. Findeisen, "Optimal exact path-following for constrained differentially flat systems," in *Proc. of 18th IFAC World Congress, Milano, Italy*, 2011, pp. 9875–9880.
- [7] Z. K. Nagy, "Model based control approach for batch crystallization product design," in *Proceedings of the 17th IFAC World Congress*, M. P. Chung, Myung Jin, Ed., vol. 17, 2008.
- [8] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *International Journal of Control*, vol. 61, pp. 1327–1361, 1995.
- [9] Python programming language. Python software foundation. [Online]. Available: www.python.org
- [10] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006, 10.1007/s10107-004-0559-y. [Online]. Available: <http://dx.doi.org/10.1007/s10107-004-0559-y>
- [11] W. Van Loock, G. Pipeleers, M. Diehl, and J. Swevers, "Optimal path following for differentially flat systems," *Automatica*, submitted.
- [12] J. Andersson, J. Åkesson, and M. Diehl, "CasADi – A symbolic package for automatic differentiation and optimal control," in *Proc. 6th International Conference on Automatic Differentiation*, ser. Lecture Notes in Computational Science and Engineering, S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds. Berlin: Springer, 2012.
- [13] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.