Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Symbolic Computation 37 (2004) 761–775
Journal of Symbolic Computation
www.elsevier.com/locate/jsc

# Symbolic computation for mobile robot path planning

Nick T. Koussoulas*, Photis Skiadas

*Department of Electrical and Computer Engineering, University of Patras, Greece*

Received 21 October 1998; accepted 11 May 1999

**Abstract**

Motion planning for mobile robots is an arduous task. Among the various methods that have been proposed for the solution of this problem in its open loop version is the Lafferriere–Sussmann method, which is based on differential geometry and employs piecewise constant inputs. This paper gives a succinct description of the method and of a freely available software tool, called the Lie Algebraic Motion Planner—LAMP and written in Mathematica™, which automates motion planning based on this technique.


*Keywords:* Mobile robotics; Motion planning; Differential geometry

## 1. Introduction

Mobile robotics is an actively researched area of modern industrial technology. Mobile robots support effectively the operations in many modern factories and warehouses, while additional applications, such as in consumer products (the autonomous vacuum cleaner), in space (NASA's Mars Pathfinder/Rover; multi-body spacecraft), and in underwater activities, are being energetically pursued. Getting a mobile robot to move in a desired fashion stumbles at difficulties arising from the necessarily intricate modeling. Given that classical (Jacobian) linearization is either ineffective or inconclusive for such systems, other directions have been investigated. Thus, the modeling basis is that mobile robots belong in the category of finite dimensional mechanical systems known as nonholonomic systems. In the case of mobile robots, nonholonomy signifies that the velocity constraints are linear but they turn out to be not integrable, indicating that some configurations in

* Corresponding author. Tel.: +30-2610-997296; fax: +30-2610-997309.
*E-mail address:* ntk@ee.upatras.gr (N.T. Koussoulas).

0747-7171/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.1999.05.001

the velocity space, i.e. directions of movement, are not possible. Having established a modeling framework does not mean that things become easier: one is still faced with a hard nonlinear problem.

Typical examples of real-life nonholonomic systems include the rolling disk and the rolling sphere, while controlled nonholonomic systems include the unicycle, the front wheel directed car, and cars with trailers. Attention is concentrated mostly on the kinematics problem, however; including dynamic effects is not particularly difficult. Motion planning and relevant control issues for nonholonomic control systems have been given considerable attention in recent years. The result has been a large number of approaches and techniques.

Robot path planning refers to the derivation of feasible paths the robot can follow that will take it from a given starting configuration to any other. Path planning is one instance of the more general motion planning problem that also includes the point stabilization problem and the trajectory tracking problem. The motion planning problem that we consider here is thought of in the sense of deriving an *open loop* control law that given an initial and a final state will drive the robot from the former to the latter. Notice that we do not consider any feedback control laws that rely on sensor information to drive the robot.

In contrast to path planning for holonomic systems (i.e., those with integrable velocity constraints), there is no one-to-one correspondence between available paths and feasible trajectories. Only the motions that satisfy the nonholonomic constraints are allowed. Despite this, by introducing the realistic assumption that the system is completely controllable we make sure that at least one feasible path exists for every pair of initial and final states. This can be done even when there are obstacles to be avoided (obstacles are usually modeled as forbidden areas of the state space).

The motion planning problem has been tackled in a variety of ways (Latombe, 1991; Lafferriere, 1991; Murray et al., 1994). There exist additional methods based on geometric phases and on parametrization of the input (which may lead to the use of neural networks as learning devices). However, it seems that differential geometry is the basis of the more elaborate methods. Among the differential geometric methods, one can find those relying on Lie algebra, within which a further distinction is made concerning the type of control used (piecewise constant or sinusoidal), and other techniques based on averaging theory, exterior differential forms, and flatness. For a very informative overview see Kolmanovsky and McClamroch (1995). Lafferriere and Sussmann (Lafferriere and Sussmann, 1991, 1992; Lafferriere, 1991) have developed a particularly elegant Lie algebraic technique employing (basically) piecewise constant controls. The application range of the method is general, i.e. it can treat any form of configuration space. The avoidance of obstacles with known positions can also be implemented but in a non-automatic fashion through the exclusion of appropriate areas of the configuration space, i.e. the method currently cannot handle inequality constraints.

The rest of the paper contains the following. In Section 2 we present an overview of the Lafferriere–Sussmann method, treating concurrently a simple but important example involving a car-like robot. In an effort to automate the task of motion planning, we describe in Section 3 a tool for symbolic implementation of that method which is called LAMP

(Lie Algebraic Motion Planner). The code has been written in Mathematica[1] 3.0 and contains routines that may prove useful outside the context of mobile robotics. Finally, in Section 4, we comment on the use of the new tool and the motion planning method.

## 2. The Lafferriere–Sussmann method for motion planning

The object of control is a nonholonomic system represented by the following differential equation:

$$\dot{\mathbf{x}}(t) = \mathbf{g}_1(\mathbf{x})u_1 + \mathbf{g}_2(\mathbf{x})u_2 + \cdots + \mathbf{g}_m(\mathbf{x})u_m \tag{2.1}$$

where the state $\mathbf{x}(t)$ belongs to an open set $N$ of $\Re^n$ and $\mathbf{g}_i$, $i = 1, \ldots, m$, are real analytic vector fields defined on $N$ and linearly independent, $\forall\, \mathbf{x} \in N$. We note that we deal only with kinematic aspects and that the system is driftless, that is, motion is possible only when some input is being applied. The motion planning problem (MPP) consists in finding reasonable algorithms producing for every pair of points $\mathbf{x}_0$ and $\mathbf{x}_f$ an open loop control

$$t \to \mathbf{u}(t) = (\, u_1(t) \quad u_2(t) \quad \cdots \quad u_m(t)\,)^{\mathrm{T}} \tag{2.2}$$

that steers $\mathbf{x}_0$ to $\mathbf{x}_f$.

The basic idea of the Lafferriere–Sussmann (LS) method is to solve the MPP for a "milder" system, which is created by suitably extending the given one. The extension consists in adding a number of vector fields, which are formed as Lie brackets of the original vector fields, to the dynamics of the system. Then, an MPP for the extended system is solved for nominal trajectories that are identical or close to the desired path. The resulting control law for the extended system can now be used as a basis for determining the sought control law for the original system. First, one has to determine the coordinates of the nominal trajectory with respect to the basis used for the original system. This action leads to the formation of a system of differential equations, which, when solved, provides the coordinates of the target point in a special basis. Finally, by equating the motions of the original and extended system, we get the desired control law in the following way. We equate the formal representations of the trajectories considering them as flows of the corresponding differential equations. Then, by solving an algebraic system for the unknown coefficients that correspond to the moves made by the original system, we form the final control law.

At this point two issues must be further clarified. First, the special basis on which we describe the system is much simpler when a condition called nilpotency is present. A system is called nilpotent when the iterated Lie brackets are zero beyond a certain order, which we call the nilpotency degree. In this way, the representation of the system is guaranteed to be finite. The basis for the relevant algebra when nilpotency is present is called the Philip Hall basis and contains those elements that are free of any relations (skew symmetry and Jacobi identity) among themselves. Second, the formal representation of the flows implies that all calculations are carried out at the formal level on the corresponding (nilpotent) Lie group and the results are translated into a control policy. These calculations involve

[1] Mathematica is a registered trademark of Wolfram Research Inc.

the Baker–Campbell–Hausdorff formula, which gives a precise expression for the composition of two moves resulting from a specific selection of vector fields. More details can be found in Lafferriere and Sussmann (1991, 1992) and Lafferriere (1991). The detailed sequence of necessary steps can be better understood via the following detailed example.

Let us consider a car described by the following driftless model, which has been already converted to its nilpotentized form:

$$\dot{\mathbf{x}}(t) = \mathbf{g}_1(\mathbf{x})u_1 + \mathbf{g}_2(\mathbf{x})u_2 \tag{2.3}$$

where

$$\mathbf{x}(t) = (\,x_1 \quad x_2 \quad x_3 \quad x_4\,)^{\mathrm{T}} \tag{2.4}$$

and

$$\mathbf{g}_1(\mathbf{x}) = (\,1 \quad 0 \quad x_2 \quad x_3\,)^{\mathrm{T}} \qquad \text{and} \qquad \mathbf{g}_2(\mathbf{x}) = (\,0 \quad 1 \quad 0 \quad 0\,)^{\mathrm{T}}. \tag{2.5}$$

Our task is to compute the open loop control law $t \to \mathbf{u}(t) = (\,u_1 \quad u_2\,)^{\mathrm{T}}$, which moves the system from the initial state $\mathbf{x}_0 = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}}$ to the final state $\mathbf{x}_f = (\,0 \quad 0 \quad 0 \quad -1\,)^{\mathrm{T}}$.

First, we check the nilpotency degree of the given system (implicitly, we verify that the system is indeed nilpotent). We discover that this degree is equal to three:

$$\mathbf{g}_3 = [\mathbf{g}_1, \mathbf{g}_2] = (\,0 \quad 0 \quad -1 \quad 0\,)^{\mathrm{T}} \tag{2.6}$$
$$\mathbf{g}_4 = [\mathbf{g}_1, [\mathbf{g}_1, \mathbf{g}_2]] = (\,0 \quad 0 \quad 0 \quad 1\,)^{\mathrm{T}} \tag{2.7}$$
$$\mathbf{g}_5 = [\mathbf{g}_2, [\mathbf{g}_1, \mathbf{g}_2]] = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}} \tag{2.8}$$
$$\mathbf{g}_6 = [\mathbf{g}_1, [\mathbf{g}_1, [\mathbf{g}_1, \mathbf{g}_2]]] = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}} \tag{2.9}$$
$$\mathbf{g}_7 = [\mathbf{g}_2, [\mathbf{g}_1, [\mathbf{g}_1, \mathbf{g}_2]]] = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}} \tag{2.10}$$
$$\mathbf{g}_8 = [\mathbf{g}_1, [\mathbf{g}_2, [\mathbf{g}_1, \mathbf{g}_2]]] = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}} \tag{2.11}$$
$$\mathbf{g}_9 = [\mathbf{g}_2, [\mathbf{g}_2, [\mathbf{g}_1, \mathbf{g}_2]]] = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}}. \tag{2.12}$$

The Philip Hall basis will have the form

$$\mathcal{B} = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4, \mathbf{g}_5\}. \tag{2.13}$$

The corresponding extended system will be

$$\dot{\mathbf{x}}(t) = \mathbf{g}_1(\mathbf{x})w_1 + \mathbf{g}_2(\mathbf{x})w_2 + \mathbf{g}_3(\mathbf{x})w_3 + \mathbf{g}_4(\mathbf{x})w_4 + \mathbf{g}_5(\mathbf{x})w_5 \tag{2.14}$$

where $\mathbf{w}(t) = (\,w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5\,)^{\mathrm{T}}$ is its control vector.

Now we can proceed with the actual calculations of the LS method.

We calculate the control for the extended system $\mathbf{w}(t)$, which drives the extended system from $\mathbf{x}_0$ to $\mathbf{x}_f$. This control law will be constructed in such a way that the trajectory of the extended system from $\mathbf{x}_0$ to $\mathbf{x}_f$ will have the form $\gamma(t) : [0, 1] \to \Re^4$, where

$$\gamma(t) = \begin{pmatrix} \gamma_1(t) \\ \gamma_2(t) \\ \gamma_3(t) \\ \gamma_4(t) \end{pmatrix} = \begin{pmatrix} A_1 t + B_1 \\ A_2 t + B_2 \\ A_3 t + B_3 \\ A_4 t + B_4 \end{pmatrix} \tag{2.15}$$

and $A_i$, $B_i \in \Re$, $i = 1, 2, 3, 4$. The curve $\gamma(t)$ is such that

$$\gamma(t=0) = \mathbf{x}_0 = (\,0 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}} \tag{2.16}$$

and

$$\gamma(t=1) = \mathbf{x}_f = (\,0 \quad 0 \quad 0 \quad -1\,)^{\mathrm{T}}. \tag{2.17}$$

Therefore, the real numbers $A_i$, $B_i$ can be determined as

$$A_1 = B_1 = A_2 = B_2 = A_3 = B_3 = B_4 = 0, \qquad A_4 = -1 \tag{2.18}$$

yielding finally

$$\gamma(t) = \begin{pmatrix} \gamma_1(t) \\ \gamma_2(t) \\ \gamma_3(t) \\ \gamma_4(t) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -t \end{pmatrix}. \tag{2.19}$$

The extended system will have eventually the following model:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ x_2 \\ x_3 \end{pmatrix} w_1 + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} w_2 + \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix} w_3 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} w_4 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} w_5. \tag{2.20}$$

or

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ x_2 w_1 - w_3 \\ x_3 w_1 + w_4 \end{pmatrix}. \tag{2.21}$$

Solving the above system for $w_1$, $w_2$, $w_3$, and $w_4$ we will get

$$(\,w_1 \quad w_2 \quad w_3 \quad w_4\,)^{\mathrm{T}} = (\,0 \quad 0 \quad 0 \quad -1\,)^{\mathrm{T}}. \tag{2.22}$$

Applying this control law to the extended system, we will observe that its trajectory starts from $\mathbf{x}_0$, follows the curve $\gamma(t)$, and finally rests on $\mathbf{x}_f$.

With the control law $\mathbf{w}(t)$ known, we will calculate next the control law $\mathbf{u}(t)$ that drives the original system from $\mathbf{x}_0$ to $\mathbf{x}_f$. To begin with, we will determine the coordinates of a point $\mathbf{x}(t)$ that lies on the curve $\gamma(t)$ in the P. Hall basis. This can be achieved by solving the system of differential equations

$$\begin{aligned} &w_1\mathbf{g}_1 + w_2\mathbf{g}_2 + w_3\mathbf{g}_3 + w_4\mathbf{g}_4 + w_5\mathbf{g}_5 \\ &\quad = c_1(h_i, \mathbf{g}_i)\dot{h}_1 + c_2(h_i, \mathbf{g}_i)\dot{h}_2 + c_3(h_i, \mathbf{g}_i)\dot{h}_3 + c_4(h_i, \mathbf{g}_i)\dot{h}_4 + c_5(h_i, \mathbf{g}_i)\dot{h}_5 \end{aligned}$$

with initial conditions

$$h_1(0) = h_2(0) = h_3(0) = h_4(0) = h_5(0) = 0. \tag{2.23}$$

The coefficients $c_i(h_i, \mathbf{g}_i)$, $i = 1, 2, \ldots, 5$, have the following form:

$$c_1(h_i, \mathbf{g}_i) = \mathbf{g}_1 \tag{2.24}$$

$$
\begin{aligned}
c_2(h_i, \mathbf{g}_i) &= Ad_{\mathrm{e}^{-h_1\mathbf{g}_1}}\mathbf{g}_2 \\
&= \mathrm{e}^{-h_1\mathbf{g}_1}\mathbf{g}_2\mathrm{e}^{h_1\mathbf{g}_1} \\
&= \mathbf{g}_2 - h_1\mathbf{g}_3 + \tfrac{1}{2}h_1^2\mathbf{g}_4 \qquad (2.25)\\
c_3(h_i, \mathbf{g}_i) &= Ad_{\mathrm{e}^{-h_1\mathbf{g}_1}\mathrm{e}^{-h_2\mathbf{g}_2}}\mathbf{g}_3 \\
&= \mathrm{e}^{-h_1\mathbf{g}_1}\mathrm{e}^{-h_2\mathbf{g}_2}\mathbf{g}_3\mathrm{e}^{h_2\mathbf{g}_2}\mathrm{e}^{h_1\mathbf{g}_1} \\
&= \mathbf{g}_3 - h_2\mathbf{g}_5 - h_1\mathbf{g}_4 \qquad (2.26)\\
c_4(h_i, \mathbf{g}_i) &= Ad_{\mathrm{e}^{-h_1\mathbf{g}_1}\mathrm{e}^{-h_2\mathbf{g}_2}\mathrm{e}^{-h_3\mathbf{g}_3}}\mathbf{g}_4 \\
&= \mathrm{e}^{-h_1\mathbf{g}_1}\mathrm{e}^{-h_2\mathbf{g}_2}\mathrm{e}^{-h_3\mathbf{g}_3}\mathbf{g}_4\mathrm{e}^{h_3\mathbf{g}_3}\mathrm{e}^{h_2\mathbf{g}_2}\mathrm{e}^{h_1\mathbf{g}_1} \\
&= \mathbf{g}_4 \qquad (2.27)\\
c_5(h_i, \mathbf{g}_i) &= Ad_{\mathrm{e}^{-h_1\mathbf{g}_1}\mathrm{e}^{-h_2\mathbf{g}_2}\mathrm{e}^{-h_3\mathbf{g}_3}\mathrm{e}^{-h_4\mathbf{g}_4}}\mathbf{g}_5 \\
&= \mathrm{e}^{-h_1\mathbf{g}_1}\mathrm{e}^{-h_2\mathbf{g}_2}\mathrm{e}^{-h_3\mathbf{g}_3}\mathrm{e}^{-h_4\mathbf{g}_4}\mathbf{g}_5\mathrm{e}^{h_4\mathbf{g}_4}\mathrm{e}^{h_3\mathbf{g}_3}\mathrm{e}^{h_2\mathbf{g}_2}\mathrm{e}^{h_1\mathbf{g}_1} \\
&= \mathbf{g}_5. \qquad (2.28)
\end{aligned}
$$

Finally, we have that

$$
\begin{aligned}
c_1(h_i, \mathbf{g}_i) &= \mathbf{g}_1 \\
c_2(h_i, \mathbf{g}_i) &= \mathbf{g}_2 - h_1\mathbf{g}_3 + \tfrac{1}{2}h_1^2\mathbf{g}_4 \\
c_3(h_i, \mathbf{g}_i) &= \mathbf{g}_3 - h_2\mathbf{g}_5 - h_1\mathbf{g}_4 \qquad (2.29)\\
c_4(h_i, \mathbf{g}_i) &= \mathbf{g}_4 \\
c_5(h_i, \mathbf{g}_i) &= \mathbf{g}_5.
\end{aligned}
$$

Replacing the coefficients in Eq. (2.21), we get

$$
\begin{aligned}
&\dot{h}_1\mathbf{g}_1 + \dot{h}_2(\mathbf{g}_2 - h_1\mathbf{g}_3 + \tfrac{1}{2}h_1^2\mathbf{g}_4) + \dot{h}_3(\mathbf{g}_3 - h_2\mathbf{g}_5 - h_1\mathbf{g}_4) + \dot{h}_4(\mathbf{g}_4) + \dot{h}_5(\mathbf{g}_5) \\
&\quad = w_1\mathbf{g}_1 + w_2\mathbf{g}_2 + w_3\mathbf{g}_3 + w_4\mathbf{g}_4 + w_5\mathbf{g}_5 \qquad (2.30)
\end{aligned}
$$

or

$$
\begin{aligned}
&(\dot{h}_1)\mathbf{g}_1 + (\dot{h}_2)\mathbf{g}_2 + (\dot{h}_3 - h_1\dot{h}_2)\mathbf{g}_3 + (-h_1\dot{h}_3 + \tfrac{1}{2}h_1^2\dot{h}_2 + \dot{h}_4)\mathbf{g}_4 + (\dot{h}_5 - h_2\dot{h}_3)\mathbf{g}_5 \\
&\quad = w_1\mathbf{g}_1 + w_2\mathbf{g}_2 + w_3\mathbf{g}_3 + w_4\mathbf{g}_4 + w_5\mathbf{g}_5 \qquad (2.31)
\end{aligned}
$$

so after some algebra we end up with the following system:

$$
\begin{aligned}
&\dot{h}_1 = 0 \\
&\dot{h}_2 = 0 \\
&\dot{h}_3 - h_1\dot{h}_2 = 0 \qquad (2.32)\\
&\tfrac{1}{2}\dot{h}_1^2\dot{h}_2 - h_1\dot{h}_3 + h_4 = -1 \\
&\dot{h}_5 - h_2\dot{h}_3 = 0
\end{aligned}
$$

which gives

$$\dot{h}_1 = 0, \qquad \dot{h}_2 = 0, \qquad \dot{h}_3 = 0, \qquad \dot{h}_4 = -1, \qquad \dot{h}_5 = 0. \qquad (2.33)$$

Solving this set of equations with initial conditions

$$h_1(0) = h_2(0) = h_3(0) = h_4(0) = h_5(0) = 0 \qquad (2.34)$$

we find that

$$\begin{aligned} h_1(t) &= 0 \\ h_2(t) &= 0 \\ h_3(t) &= 0 \\ h_4(t) &= -t \\ h_5(t) &= 0. \end{aligned} \tag{2.35}$$

Therefore, the Philip Hall coordinates of the final point $\mathbf{x}_f$ can be computed by substituting the value $t = 1$ in the solution:

$$(h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5)^{\mathrm{T}} = (0 \quad 0 \quad 0 \quad -1 \quad 0)^{\mathrm{T}}. \tag{2.36}$$

At this point, a major decision must be taken concerning the number of steps (movements) that are necessary for reaching the target state from the initial state. This problem has not been solved in its general form and thus only approximate techniques may be applied. The LS formulation proves to be helpful in this respect because it is possible to use the following relation (in the case of bang-bang moves):

$$\mathbf{x}_f = \mathbf{x}_0 \mathrm{e}^{a_1 \mathbf{g}_1} \mathrm{e}^{a_2 \mathbf{g}_2} \cdots \mathrm{e}^{a_s \mathbf{g}_1} = \mathbf{x}_0 \mathrm{e}^{h_m \mathbf{g}_m} \cdots \mathrm{e}^{h_1 \mathbf{g}_1} \tag{2.37}$$

where $s$ is the number of desired moves, $m$ is the dimension of the P. Hall basis, and e denotes the formal exponential (or, equivalently, the flow of the corresponding differential equation). Then, one must determine the unknown coefficients $a_i$, which can be done with the help of the Campbell–Baker–Hausdorff formula (Murray et al., 1994). These coefficients are not independent from each other. As explained in Varadarajan (1984), a move along the Lie bracket of two vector fields $\mathbf{g}_1$ and $\mathbf{g}_2$ can be analyzed in a series of four independent moves along $\mathbf{g}_1$ and $\mathbf{g}_2$ in turn. Note also that the coefficients of all four exponentials depend on the same coefficient. In a similar way, one can discover that movement along a Lie product (nested set of Lie brackets) of order 4 implies a series of ten moves, again with a single coefficient involved. In this way, it is possible to derive relations that connect the coefficients $a_i$ with the known quantities $h_j$.

Now, returning to our example, let us assume that the original system can move from the initial state $\mathbf{x}_0$ to the final state $\mathbf{x}_f$ in nine bang-bang moves. Then, the corresponding control vectors will be of the form

$$\begin{aligned} u_1 &= (u_1 \quad u_2)^{\mathrm{T}} = (a_1 \quad 0)^{\mathrm{T}} && \text{for time } T_1 = 1 \\ u_2 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad a_2)^{\mathrm{T}} && \text{for time } T_2 = 1 \\ u_3 &= (u_1 \quad u_2)^{\mathrm{T}} = (a_3 \quad 0)^{\mathrm{T}} && \text{for time } T_3 = 1 \\ u_4 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad a_4)^{\mathrm{T}} && \text{for time } T_4 = 1 \\ u_5 &= (u_1 \quad u_2)^{\mathrm{T}} = (a_5 \quad 0)^{\mathrm{T}} && \text{for time } T_5 = 1 \\ u_6 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad a_6)^{\mathrm{T}} && \text{for time } T_6 = 1 \\ u_7 &= (u_1 \quad u_2)^{\mathrm{T}} = (a_7 \quad 0)^{\mathrm{T}} && \text{for time } T_7 = 1 \\ u_8 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad a_8)^{\mathrm{T}} && \text{for time } T_8 = 1 \\ u_9 &= (u_1 \quad u_2)^{\mathrm{T}} = (a_9 \quad 0)^{\mathrm{T}} && \text{for time } T_9 = 1 \end{aligned}$$

where the coefficients $a_i, i = 1, \ldots, 9$, are real numbers. Thus, the original system will follow a trajectory of the form

$$\mathbf{x}_f = \mathbf{x}_0 \mathrm{e}^{a_1\mathbf{g}_1}\mathrm{e}^{a_2\mathbf{g}_2}\mathrm{e}^{a_3\mathbf{g}_1}\mathrm{e}^{a_4\mathbf{g}_2}\mathrm{e}^{a_5\mathbf{g}_1}\mathrm{e}^{a_6\mathbf{g}_2}\mathrm{e}^{a_7\mathbf{g}_1}\mathrm{e}^{a_8\mathbf{g}_2}\mathrm{e}^{a_9\mathbf{g}_1} \tag{2.38}$$

$$= \mathbf{x}_0 \mathrm{e}^{h_5\mathbf{g}_5}\mathrm{e}^{h_4\mathbf{g}_4}\mathrm{e}^{h_3\mathbf{g}_3}\mathrm{e}^{h_2\mathbf{g}_2}\mathrm{e}^{h_1\mathbf{g}_1} \tag{2.39}$$

$$= \mathbf{x}_0 \mathrm{e}^{-\mathbf{g}_4}. \tag{2.40}$$

At this point application of the Campbell–Baker–Hausdorff formula will give us a system of algebraic equations relating the known quantities $h_1, \ldots, h_5$ with the unknown coefficients $a_1, \ldots, a_9$. In principle, we can find a solution that will be parametrized by the four superfluous coefficients whose values we may fix. However, hunting for a set of values that will indeed be suitable for the prescribed move is an arduous task. It is far preferable and not at all limiting to select the kind of moves the system will make. Thus, we accept that a series of four bang-bang moves corresponds to a move along the Lie bracket (represented by $\mathbf{g}_3$ in Eq. (2.4)). Assuming that this move will be the first one, we fix the corresponding coefficient to be equal to one and then we can determine the unknown coefficients $a_5, \ldots, a_9$, finding eventually that

$$a_1 = 1, a_2 = 1, a_3 = -1, a_4 = -1, a_5 = 1, a_6 = 1, a_7 = 1,$$
$$a_8 = -1, a_9 = -2 \tag{2.41}$$

so finally

$$\begin{aligned}
u_1 &= (u_1 \quad u_2)^{\mathrm{T}} = (1 \quad 0)^{\mathrm{T}} && \text{for time } T_1 = 1\\
u_2 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad 1)^{\mathrm{T}} && \text{for time } T_2 = 1\\
u_3 &= (u_1 \quad u_2)^{\mathrm{T}} = (-1 \quad 0)^{\mathrm{T}} && \text{for time } T_3 = 1\\
u_4 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad -1)^{\mathrm{T}} && \text{for time } T_4 = 1\\
u_5 &= (u_1 \quad u_2)^{\mathrm{T}} = (1 \quad 0)^{\mathrm{T}} && \text{for time } T_5 = 1\\
u_6 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad 1)^{\mathrm{T}} && \text{for time } T_6 = 1\\
u_7 &= (u_1 \quad u_2)^{\mathrm{T}} = (1 \quad 0)^{\mathrm{T}} && \text{for time } T_7 = 1\\
u_8 &= (u_1 \quad u_2)^{\mathrm{T}} = (0 \quad -1)^{\mathrm{T}} && \text{for time } T_8 = 1\\
u_9 &= (u_1 \quad u_2)^{\mathrm{T}} = (-2 \quad 0)^{\mathrm{T}} && \text{for time } T_9 = 1.
\end{aligned}$$

We calculate now the first two movements just to illustrate what the trajectory will be.

*Move* 1

Applied control:

$$\mathbf{u}_1 = (1 \quad 0)^{\mathrm{T}}. \tag{2.42}$$

Time interval: $0 \le t \le 1$.

System model:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ x_2 \\ x_3 \end{pmatrix}. \tag{2.43}$$

Initial conditions:

$$x_1(0) = x_2(0) = x_3(0) = x_4(0) = x_5(0) = 0. \tag{2.44}$$

Solution:

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{pmatrix} = \begin{pmatrix} t \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.45}$$

Final state at $t = 1$: $(\,1 \quad 0 \quad 0 \quad 0\,)^{\mathrm{T}}$.

*Move* 2

Applied control:

$$\mathbf{u}_2 = (\,0 \quad 1\,)^{\mathrm{T}}. \tag{2.46}$$

Time interval: $1^+ \leq t \leq 2$.

System model:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}. \tag{2.47}$$

Initial conditions:

$$x_1(1) = 1, \qquad x_2(1) = x_3(1) = x_4(1) = x_5(1) = 0. \tag{2.48}$$

Solution:

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{pmatrix} = \begin{pmatrix} 1 \\ t \\ 0 \\ 0 \end{pmatrix}. \tag{2.49}$$

Final state at $t = 2$: $(\,1 \quad 1 \quad 0 \quad 0\,)^{\mathrm{T}}$.

And so on for the rest of the moves. Figs. 1–4 show the trajectory for each state. The plots have been produced by the software code described in Section 3 and the time is in s/10.

## 3. Implementation for symbolic computation

The Lafferriere–Sussmann motion planning method just described requires a large amount of tedious calculations should one wish to use it for development purposes. In the course of our research, where we investigated motion planning with piecewise constant inputs whose values are limited to a finite set (Skiadas and Koussoulas, 1997), we have been confronted with a need for an experimentation platform and thus we decided to implement the method in a symbolic manipulation environment. The result has been a set

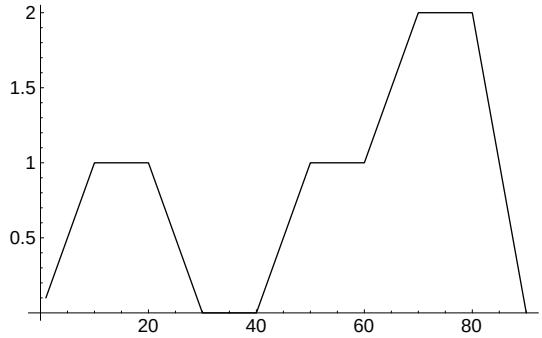


Fig. 1. Evolution of state $x_1$.

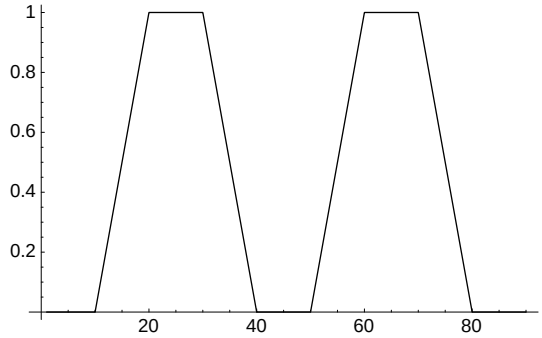


Fig. 2. Evolution of state $x_2$.

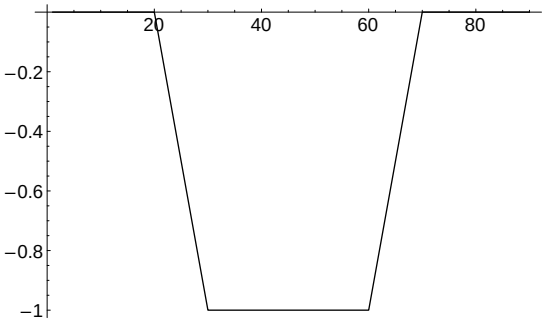


Fig. 3. Evolution of state $x_3$.

of routines that fully automates the LS motion planning method while some of the routines developed can be used for more general computations in the area of geometric nonlinear control. We do not call this set of routines a “package” since some functionalities are

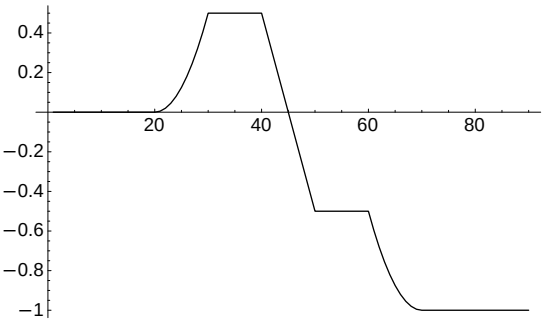


Fig. 4. Evolution of state $x_4$.

realized in a non-fancy simplistic way, especially the problem definition requested from the user. The set has been named LAMP v1.0, for Lie Algebraic Motion Planner.

When setting out to automate the motion planning method described above, the first decision had to do with the choice of a suitable symbolic manipulation platform. Finally, Mathematica was chosen on the basis of two facts: first, there has been some basic work on theoretical topics done within that environment (cf. Murray et al. (1994) and the dissertation of the first author of Murray et al. (1994)); and second, it seems that Mathematica is being established as the primary environment in the area of robotics, e.g. the Robotica package (Nethery and Spong, 1994), Screws in Murray et al. (1994). Affordability, being always a consideration, combined with the above facts led to the choice of Mathematica 3.0.1 as the software platform.

### *3.1. Structure of code*

The package contains a number of routines that implement some general differential geometric and group theoretic operations and another set of more specialized routines strictly related to the motion planning method.

The user interface is not at all fancy. Near the beginning of the code, the user keys in the definition of his/her problem that includes the specific vector fields, initial and target states, initial and final times, and the dimension of the control vector. Some examples from standard problems of the field are provided as a guide. The user is later asked about the desired number of moves and the sequence of the activated vector fields. In case of two vector fields, only the initial one must be indicated. Otherwise, full details are necessary. The above represent the total user involvement.

The code is about 1100 lines long (with comments). There are in total 17 routines or functions. Seven of those are borrowed from the dissertation of R.M. Murray and they are marked with an asterisk in the list below. Their total size is about 25 lines of pure code. Another set of small routines deals with group theoretic and differential geometric operations. Finally, there are four larger routines, which actually implement the motion planning method.

`Ad`: adjoint representation ($Ad_{X,Y} = \mathrm{e}^{X} Y \mathrm{e}^{X}$, where e denotes the formal exponential)

`fexp`: the formal exponential; the (finite because of nilpotency) order of the series expansion is an argument

`fbracket`: the formal Lie bracket ($[X, Y] = X * Y - Y * X$)

`jacobian`: calculation of the Jacobian

`SLieBracket`: the symbolic Lie bracket; calculates the Lie bracket for specific vector fields

`BCH`: implementation of the Baker–Campbell–Hausdorff formula

`Bracket` *: definition of the Lie bracket for symbolic manipulation purposes

`degree` *: calculates the degree of a Lie product

`leq` *, `lt` *: they define the ordering on the Lie algebra

`PHallCheck` *, `PHallFilter` *: they sift the basis elements according to the P. Hall criteria

`PhilipHallBasis` *: construction of the Philip Hall basis given the list of generators and the order

`NilpDegree`: given a set of vector fields, it checks and calculates their nilpotency degree up to a user-defined limit

`PHbasis`: calculates the Philip Hall basis (using PhilipHallBasis), its actual value for the given vector fields, and constructs a valid basis retaining only linearly independent elements

`ExtendedSystem`: implements the first two stages of the LS method—namely, it forms the extended system and calculates nominal controls

`MovementODE`: formulates and solves the system of ordinary differential equations (ODEs) that calculates the backward P. Hall coordinates of the target point

Naturally, some of the routines can be used independently. For example, NilpDegree can determine the degree of nilpotency for any set of vector fields while BCH can be used in other contexts (e.g. coadjoint orbits in quantum mechanics) where actual calculations are needed. It should be noted that the authors know of no other implementation of the Baker–Campbell–Hausdorff formula.

### *3.2. Remarks*

During the development of the code, several difficulties appeared due mostly to some features (or lack thereof) of Mathematica. Besides some necessary redefinition of properties for some user-provided functions and heavy use of string-to-expression conversions (`ToExpression[]` command) to facilitate flexible substitutions for example, the greatest source of difficulty proved to be noncommutative multiplication. Mathematica, just like other widely used symbolic manipulation packages, does not support calculations in a noncommutative environment very well, even though noncommutative multiply is available as an operation. In this code, noncommutative multiply is implemented through the redefinition of the command `Times[]` by removing its commutativity property. This is done instead of using `NonCommutativeMultiply[]` because it created fewer difficulties in the calculations of other things, such as identifying the coefficients of powers in polynomials, expansion of expressions, and the like. The greatest difficulty occurred in the routine MovementODE (whose most operations are done under an algebra with noncommutative multiplication), where Mathematica had to be tricked to correctly perform Collect[], which collects together terms involving the same powers in an expression. Such occurrences are clearly marked in the code.

The code works under Mathematica 3.0 as well but we believe that its overall complexity will not allow it to run under earlier versions (we did not even try). Even under these circumstances, there is a particular behavior of the Mathematica kernel that users must be aware of. When the code is first run after Mathematica starts, the execution is incorrect. The user can either wait for the useless calculations to come to an end (and ignore the results, of course) or abort the computation. Following that and for all subsequent executions, the program runs correctly. This is most probably due to the less than perfect handling by the kernel of the redefined multiplication command `Times[]`.

The execution time depends on the dimension of the problem and the complexity of the particular vector fields. Expected execution times for the typical cases of the rolling disk (dimension 3) and the car (dimension 4), such as the one in the example above, are 1 min and 24 min, respectively. However, in the case of the rolling disk, the figure can easily become three or four times as large if the number of moves is increased or the target state creates further demands. These values were realized on a Pentium PC with 64 MB of RAM running at 133 MHz under Microsoft Windows 95. Finally, it should be mentioned that the authors are not aware of any similar package implementing algorithms for motion planning that are based on differential geometry.

### *3.3. Example output*

As an example of application, the output for the example presented in Section 2 appears in Fig. 5. This output is followed by the figures shown in the last section (Figs. 1–4).

## 4. Discussion

The Lafferriere–Sussmann method for motion planning of mobile robots (and nonholonomic systems in general) is an elegant differential geometric approach to a difficult problem. The method has the usual shortcomings that are common to all currently available methods, namely, sensitivity to modeling errors, sensitivity to disturbances, and generation of open loop solutions only. In particular, the sensitivity of the procedure to small changes in the problem definition, even values of the target state, leads to unexpectedly different results. Despite the above, the method is quite general and therefore potentially useful for a wide range of motion planning problems.

The generality and applicability of the method justify an implementation that will provide an automated way for performing the large number of necessary calculations and computations. This is now available in the form of a software tool written in the symbolic manipulation language Mathematica. The tool behaves well in general but may present some minor difficulties in its use. One has to do with the execution time, which can be quite long for medium sized systems (state space dimension larger than four). The other difficulty has to do (in all appearances) with Mathematica's handling of the redefined basic multiplication routine for noncommutative multiplication. Fortunately, this problem is alleviated by a dummy execution of the code the first time it is used.

The tool is flexible enough to suit the users' needs for motion planning tasks. The code can be easily adapted to display intermediate results, do additional plotting, etc. Furthermore, some routines can be used independently in other contexts too, mainly

```
Nilpotency degree and corresponding number of basis elements:

            2: 1

            3: 2

            4: 4

            5: 10

Nilpotency degree: 3
```

$$\text{Philip Hall basis:} \quad \begin{pmatrix} g1 \\ g2 \\ [g1,\ g2] \\ [g1,\ [g1,\ g2]] \\ [g2,\ [g1,\ g2]] \end{pmatrix}$$

$$\text{Actual value of Philip Hall basis (dimension = 5):} \quad \begin{pmatrix} 1 & 0 & x2 & x3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
Dimension of extended system: 4
```

$$\text{Extended control solution:} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

$$\text{Numerical value of extended control solution:} \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

```
      FINAL ODE SOLUTION:
 h1[t] → 0      h2[t] → 0      h3[t] → 0      h4[t] → -1. t      h5[t] → 0
```

$$\text{Final point coordinates (P. Hall):} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1. \\ 0 \end{pmatrix}$$

```
    Solution for coefficients a:
 {a5 → 1., a9 → -2., a7 → 1., a1 → 1., a3 → -1., a6 → 1., a8 → -1., a2 → 1., a4 → -1.}

SOLVED a: {1. g1, 1. g2, -1. g1, -1. g2, 1. g1, 1. g2, 1. g1, -1. g2, -2. g1}

FINAL STATE: {0., 0, 0, -1.}
```

Fig. 5.

differential geometric nonlinear control. Finally, the tool is freely available electronically at the following address: http://www.lar.ee.upatras.gr/koussoulas/software/LAMP-v1.nb or by contacting the first author (ntk@ee.upatras.gr).

## References

Kolmanovsky, I., McClamroch, N.H., 1995. Developments in nonholonomic control problems. IEEE Control Systems 40, 20–36.

Lafferriere, G., 1991. A general strategy for computing steering controls of systems without drift. In: Proceedings of 30th CDC, Brighton, England.

Lafferriere, G.A., Sussmann, H.J., 1991. Motion planning for controllable systems without drift. In: Proceedings of IEEE Robotics & Automation Conference, Sacramento, CA.

Lafferriere, G.A., Sussmann, H.J., 1992. A differential geometric approach to motion planning. In: Li, Z., Canny, J.F. (Eds.), Nonholonomic Motion Planning. Kluwer, Boston.

Latombe, J.-C., 1991. Robot Motion Planning. Kluwer AP, Boston.

Murray, R.M., Li, Z., Sastry, S.S., 1994. A Mathematical Introduction to Robotic Manipulation. CRC Press, Boca Raton.

Nethery, J., Spong, M., 1994. Robotica: a Mathematica package for Robot analysis. IEEE Robotics and Automation Magazine 1 (1).

Skiadas, Ph., Koussoulas, N.T., 1997. Motion planning for drift-free nonholonomic systems under a discrete levels control constraint. In: Proc. 5th IEEE Mediterranean Symposium in Control and Automation 1997. Paphos, Cyprus.

Varadarajan, V.S., 1984. Lie Groups, Lie Algebras, and Their Representations. Springer, New York.