applied sciences

MDPI

Article

Development of Task-Space Nonholonomic Motion Planning Algorithm Based on Lie-Algebraic Method

Arkadiusz Mielczarek and Ignacy Dulęba *

Department of Cybernetics and Robotics, Wroclaw University of Science and Technology, Janiszewski St. 11/17, 50-372 Wrocław, Poland; arkadiusz.mielczarek@pwr.edu.pl
* Correspondence: ignacy.duleba@pwr.edu.pl

Abstract: In this paper, a Lie-algebraic nonholonomic motion planning technique, originally designed to work in a configuration space, was extended to plan a motion within a task-space resulting from an output function considered. In both planning spaces, a generalized Campbell–Baker–Hausdorff–Dynkin formula was utilized to transform a motion planning into an inverse kinematic task known for serial manipulators. A complete, general-purpose Lie-algebraic algorithm is provided for a local motion planning of nonholonomic systems with or without output functions. Similarities and differences in motion planning within configuration and task spaces were highlighted. It appears that motion planning in a task-space can simplify a planning task and also gives an opportunity to optimize a motion of nonholonomic systems. Unfortunately, in this planning there is no way to avoid working in a configuration space. The auxiliary objective of the paper is to verify, through simulations, an impact of initial parameters on the efficiency of the planning algorithm, and to provide some hints on how to set the parameters correctly.

Keywords: nonholonomic systems; motion planning; Lie algebraic method; configuration space; task-space

Citation: Mielczarek, A.; Dulęba, I. Development of Task-Space Nonholonomic Motion Planning Algorithm Based on Lie-Algebraic Method. *Appl. Sci.* **2021**, *11*, 10245. https://doi.org/10.3390/app112110245

Academic Editor: Alessandro Gasparetto

Received: 4 October 2021
Accepted: 27 October 2021
Published: 1 November 2021

1. Introduction

A large number of contemporary and practically important robots can be described as nonholonomic systems. Despite different sources of nonholonomy and nonholonomic manipulators, wheel mobile robots [1] and free-floating space robots [2] belong to this class. Even when modeled at the kinematic level, nonholonomic systems are difficult to control because the number of controls is smaller than the dimension of their configuration spaces. Thus, sophisticated maneuvers have to be performed to get a desired location, clearly exemplified by the task of parking a car (especially towing trailers). For some special cases of nonholonomic systems (like Dubbins [3] and Reeds-Shepp cars [4] or systems in a chain form [5]), dedicated methods were proposed to plan their motions. However, there is a constant need to develop general purpose methods applicable to any nonholonomic system. Two main classes of general purpose methods can be distinguished: global [6,7] and local ones [8]. Global methods try to get a whole output trajectory at once (although in an iterative manner), while local ones construct an output trajectory from sub-trajectories derived from local planning. Global methods are rooted in various paradigms. Using a linearization of a nonholonomic system along the trajectory corresponding to current controls, Tchon and coworkers [7] reformulated a nonholonomic motion planning task into an inverse task solved with classical methods. Jakubczyk [9] applied the Volterra series expansion of a nonholonomic system with an output function to get highly oscillatory controls that were able to trace a desired trajectory with an arbitrary precision. Arismendi and coworkers [10] adapted a variant of a wave front technique to design the fast marching square path planning method of nonholonomic motion planning.

Generally, algorithms derived from global methods are computationally involved and very sensitive to geometric constraints due to obstacles. Consequently, they cannot be

applied in dynamic or partially unknown environments. Local methods do not suffer from these drawbacks.

In this paper, a Lie-algebraic local method of motion planning will be studied. There are three main goals of the paper:

1. to extend applicability of the method to plan a motion also within a task-space, and to provide a complete algorithm for implementing the method,
2. to show through simulations how data parameterizing the algorithm impacts its efficiency,
3. to compare planning within a configuration and a task-space, pointing out similarities and differences.

It is worth noticing that local and global planning tasks belong to a classic triple: geometrical planning, motion planning, and control. Recently, some attempts have been made to combine the last two tasks into one planning-control loop [11,12]. A vast majority of motion planning algorithms are based on discretization of control/configuration spaces to transform continuous tasks, described by differential equations and static output functions, into a graph domain. In this discrete space, graph-searching algorithms, like RRT or its numerous variants [13], can be used to solve the planning task.

The paper is organized as follows: In Section 2 a model of nonholonomic systems is introduced supplemented with an output function, then a motion planning task is defined and, finally, a Lie algebraic algorithm to solve the task is presented. In Section 3 an extensive simulation study is provided to evaluate performance of the algorithm for various data that impact its behavior. Then, some observations are formulated based on the simulation results. Section 4 concludes the paper.

2. Model and Algorithm

Nonholonomic systems modeled at a kinematic level are described by the control-affine equation [14]

$$\dot{\boldsymbol{q}} = \sum_{i=1}^{m} \boldsymbol{g}_i(\boldsymbol{q}) u_i, \quad \dim(\boldsymbol{q}) = n > m, \tag{1}$$

where $\mathbb{Q} \ni \boldsymbol{q}$ is a configuration, $\boldsymbol{u} = (u_1, \ldots, u_m)^T$ are controls, and $\boldsymbol{g}_i(\boldsymbol{q})$ are (smooth) vector fields (generators) that span a null space of nonholonomic constraints. By definition, nonholonomic systems satisfy the Lie algebra rank condition. Thus, according to the Chow's theorem [15], they are small time locally controllable.

In order to shorten notations, two-input systems, $m = 2$, will be considered with $\boldsymbol{g}_1 = \boldsymbol{X}$ and $\boldsymbol{g}_2 = \boldsymbol{Y}$. Two input systems are the most challenging as for the systems it is quite easy to generate difficult motion planning tasks by increasing a configuration space dimension.

In many practical applications not all components of a configuration vector are important (for example wheel angles of mobile robots do not impact obstacle collisions), thus an output map is introduced

$$\boldsymbol{x} = \boldsymbol{k}(\boldsymbol{q}), \quad \dim(\boldsymbol{x}) = r \tag{2}$$

and usually $r < n$.

Now, a motion planning task can be stated as follows: given an initial configuration $\boldsymbol{q}_0$ and a goal point, $\boldsymbol{x}_f$ find controls $\boldsymbol{u}(\cdot)$ which, applied to the system (1), generates an end-point of the resulting trajectory mapped with Equation (2) into the desired location $\boldsymbol{x}_f$.

In Lie-algebraic methods, locally, around a current configuration $\boldsymbol{q}_c$, possible directions of motion of the system (1) are described by vector fields derived from generators $\boldsymbol{X}, \boldsymbol{Y}$. To avoid redundancy, it is a common practice to take only a basis of the Lie algebra spanned by the generators. There are at least three such bases, but probably the most frequently used is the Ph. Hall basis. It is composed of an infinite number of Lie monomials, interpreted as

vector fields within the scope of the control system (1). The very first elements of the basis are given below

$$H = (\boldsymbol{X}, \boldsymbol{Y}, [\boldsymbol{X}, \boldsymbol{Y}], [\boldsymbol{X}, [\boldsymbol{X}, \boldsymbol{Y}]], [\boldsymbol{Y}, [\boldsymbol{X}, \boldsymbol{Y}]], \ldots) \tag{3}$$

where $[\boldsymbol{A}, \boldsymbol{B}]$ denotes a Lie bracket of Lie monomials (for vector fields $[\boldsymbol{A}, \boldsymbol{B}] = (\partial \boldsymbol{B}/\partial \boldsymbol{q})\boldsymbol{A} - (\partial \boldsymbol{A}/\partial \boldsymbol{q})\boldsymbol{B}$). The basis is composed of generators (degree 1 Lie monomials) and higher degree monomials derived from generators and their descendants. The system (1) is a nonholonomic one, thus a finite (and usually small) number of initial Ph. Hall basis elements, $\boldsymbol{H}_{trunc}$, can be taken to satisfy the Lie algebra rank condition [15]

$$\forall \boldsymbol{q}_c \in \mathbb{Q} \quad \operatorname{rank} \boldsymbol{H}_{trunc}(\boldsymbol{q}_c) = n. \tag{4}$$

Consequently, the system (1) is a small time locally controllable and it can evolve at any configuration in any direction. The condition (4) is quite easy to check in off-line mode with the assistance of symbolic computation packages. However, a very demanding problem is how to generate a motion in any direction with a small number of controls. Fortunately, the generalized Campbell–Baker–Hausdorff–Dynkin formula [16] answers this question constructively. It states that a configuration shift $\boldsymbol{z}(t)(\boldsymbol{q}_c)$ at a current configuration $\boldsymbol{q}_c$, depends on controls $\boldsymbol{u}$ [17] as follows

$$\begin{aligned} \boldsymbol{z}(t)(\boldsymbol{q}_c, \boldsymbol{u}) =& \alpha_1(t)\boldsymbol{X}(\boldsymbol{q}_c) + \alpha_2(t)\boldsymbol{Y}(\boldsymbol{q}_c) + \alpha_3(t)[\boldsymbol{X}, \boldsymbol{Y}](\boldsymbol{q}_c) + \\ &+ \alpha_4(t)[\boldsymbol{X}, [\boldsymbol{X}, \boldsymbol{Y}]](\boldsymbol{q}_c) + \alpha_5(t)[\boldsymbol{Y}, [\boldsymbol{X}, \boldsymbol{Y}]](\boldsymbol{q}_c) + \ldots = \boldsymbol{H}_{trunc}(\boldsymbol{q}_c)^T \boldsymbol{\alpha}(\boldsymbol{u}), \end{aligned} \tag{5}$$

where control-dependent coefficients $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots)^T$ pre-multiplying Lie monomials (vector fields) are expressed as

$$\begin{aligned} &\alpha_1(t) = \textstyle\int_0^t u_1(s_1) ds_1, \quad \alpha_2(t) = \int_0^t u_2(s_1) ds_1, \\ &\alpha_3(t) = \tfrac{1}{2} \textstyle\int_0^t \int_0^{s_2} (u_1(s_1)u_2(s_2) - u_2(s_1)u_1(s_2)) ds_1 ds_2, \quad \ldots \end{aligned} \tag{6}$$

and they become more and more complex as the integration is performed over the k-dimensional simplex, where k is the degree of a Lie monomial the coefficient corresponds to. It is a common practice to express controls as linear combinations of time-dependent functions [7]

$$u_i(t) = \sum_{j=0}^{N_i - 1} \phi_j(t) p_i^j, \quad i = 1, \ldots, m, \quad t \in [0, T], \tag{7}$$

where $\phi_j(t)$ are elements of any functional basis (polynomials, harmonic functions) and N_i is the number of variables required to describe the i-th control. Thus, a vector $\boldsymbol{p}$ collects all variables p_i^j and uniquely describes controls $\boldsymbol{u}$ and also energy of controls. Moreover, the shift (5) can be expressed as a function of $\boldsymbol{p}$ and resembles forward kinematics (but at a velocity level when considered on a fixed time horizon)

$$\boldsymbol{z}(t)(\boldsymbol{q}_c, \boldsymbol{p}) = \boldsymbol{F}(\boldsymbol{p}). \tag{8}$$

The shift (8) from the configuration space is mapped into the task-space, and within this space a desired motion towards the goal $\boldsymbol{x}_f$ can be computed using the Newton algorithm. A complete algorithm to plan a motion of the system (1) with the output function (2), either in a configuration or a task-space, is presented in Algorithm 1.

Algorithm 1 nonholonomic motion planning within a task-space

Step 1. Read-in initial data:
the system (1) together with the output function (2), the initial configuration $\boldsymbol{q}_0$ and the goal point $\boldsymbol{x}_f$, desired accuracy of reaching the goal ϵ, the initial value of $\boldsymbol{p}_c$.
Step 2. Initialize empty resulting trajectory $\boldsymbol{q}_{res}(\cdot)$, and set the current configuration $\boldsymbol{q}_c \leftarrow \boldsymbol{q}_0$.
Step 3. Check the stop condition: iff

$$\|\boldsymbol{x}_f - \boldsymbol{k}(\boldsymbol{q}_c)\| < \epsilon, \tag{9}$$

then stop computations and output resulting trajectory $\boldsymbol{q}_{res}(\cdot)$. Otherwise, progress with Step 4.
Step 4. Select a parameterization of controls (7), a time horizon T, and derive mapping (8).
Step 5. At a current point in the task-space $\boldsymbol{x}_c = \boldsymbol{k}(\boldsymbol{q}_c)$, select the planned shift

$$\Delta\boldsymbol{x} \leftarrow \xi(\boldsymbol{x}_f - \boldsymbol{x}_c) \tag{10}$$

by setting a positive coefficient ξ.
Step 6. Using the Newton algorithm, Ref. [18] find $\boldsymbol{p}^\star$ that solves the inverse kinematic task

$$\Delta\boldsymbol{x} = \boldsymbol{J}(\boldsymbol{q}_c)\boldsymbol{F}(\boldsymbol{p}), \tag{11}$$

where $\boldsymbol{J}(\boldsymbol{q}) = \partial\boldsymbol{k}(\boldsymbol{q})/\partial\boldsymbol{q}$ is the Jacobi matrix of the mapping (2).
Step 7. Check whether the solution is acceptable:

$$\|\boldsymbol{x}_f - \boldsymbol{k}(\boldsymbol{q}^\star(T))\| < \|\boldsymbol{x}_f - \boldsymbol{x}_c\|, \tag{12}$$

where $\boldsymbol{q}^\star(T)$ is the end-point of the trajectory $\boldsymbol{q}^\star(\cdot)$ initialized at $\boldsymbol{q}_c$ and generated by the system (1) with controls $\boldsymbol{u}^\star(t)$, $t \in [0, T]$, corresponding to the vector of parameters $\boldsymbol{p}^\star$, cf. Equation (7).
Step 8. If Condition (12) is satisfied, then:

1. append the trajectory $\boldsymbol{q}^\star(\cdot)$ to the resulting trajectory $\boldsymbol{q}_{res}(\cdot) \leftarrow \boldsymbol{q}_{res}(\cdot) \cup \boldsymbol{q}^\star(\cdot)$,
2. update the current configuration $\boldsymbol{q}_c \leftarrow \boldsymbol{q}^\star(T)$
3. and go back to Step 4.

Otherwise, decrease the coefficient ξ and go to Step 5.

Comments on Algorithm 1:

- A nonholonomic motion planning in a configuration space is achieved by setting the identity output function (2).
- A selected parameterization (7), Step 4, can vary from one iteration to another. Usually, one or at most a few parameterizations can be exploited, and vectors $\alpha(\boldsymbol{u}(\boldsymbol{p}))$ from Equation (5) can be computed in an off-line mode.
- In Step 7, a minimal requirement was formulated, i.e., a new end-point of trajectory in the task-space should be closer to the goal than it was in the previous iteration. More restrictive requirement assumes that it should also be close enough to the planned sub-goal $\boldsymbol{x}_c + \Delta\boldsymbol{x}$.
- Singularities in solving (11) result from a rank deficiency of the Jacobi matrix, derived from the right hand side of (11)

$$\hat{\boldsymbol{J}}(\boldsymbol{p}) = \frac{\partial(\boldsymbol{J}(\boldsymbol{q}_c)\boldsymbol{F}(\boldsymbol{p}))}{\partial\boldsymbol{p}} = \boldsymbol{J}(\boldsymbol{q}_c)\frac{\partial\boldsymbol{F}(\boldsymbol{p})}{\partial\boldsymbol{p}}. \tag{13}$$

Thus, singularities may arise

1. either due to a rank deficiency of $\boldsymbol{J}(\boldsymbol{q}_c)$ (they are avoidable by a small variation of controls which results in a small shift of $\boldsymbol{q}_c$)

2. or a rank deficiency of the Jacobi matrix $\partial \boldsymbol{F}(\boldsymbol{p})/\partial \boldsymbol{p}$ at the current $\boldsymbol{p}_c$. Equivalent to the matrix $\partial \alpha(\boldsymbol{u}(\boldsymbol{p}))/\partial \boldsymbol{p}$, cf. Equations (5), (7), and (8) as the full rank matrix $\boldsymbol{H}_{trunc}(\boldsymbol{q}_c)$ cannot introduce singularities at all, cf. Equation (4).

- In Step 5, the positive coefficient ξ should not be too large, as values of matrices $\boldsymbol{J}(\boldsymbol{q}_c)$, $\boldsymbol{H}_{trunc}(\boldsymbol{q}_c)^T$ are computed at the current configuration $\boldsymbol{q}_c$ and neglecting some vector fields, while truncation $\boldsymbol{H} \to \boldsymbol{H}_{trunc}$, is justified only locally. Its value cannot be too small either, as many iterations might be required to complete the algorithm.
- In motion planning within a task-space with $r < n$, it is impossible to avoid planning within a configuration space, as the mapping (2) is not unique in this case.

3. Simulations

In order to evaluate Algorithm 1, some tests were performed using two models of mobile robots depicted in Figure 1. The unicycle is described by equations

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \begin{bmatrix} \cos(q_3) & 0 \\ \sin(q_3) & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u} = \begin{bmatrix} \cos(q_3) \\ \sin(q_3) \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_2 = \boldsymbol{X}(\boldsymbol{q})u_1 + \boldsymbol{Y}(\boldsymbol{q})u_2. \tag{14}$$

The configuration vector $\boldsymbol{q}$ is composed of position (q_1, q_2) and orientation q_3 of the vehicle on a plane, while u_1, u_2 denote controls interpreted as angular and linear velocities.

The model of a kinematic car is given as

$$\dot{\boldsymbol{q}} = \begin{bmatrix} L\cos(q_3)\cos(q_4) & 0 \\ L\sin(q_3)\cos(q_4) & 0 \\ \sin(q_4) & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u} = \begin{bmatrix} L\cos(q_3)\cos(q_4) \\ L\sin(q_3)\cos(q_4) \\ \sin(q_4) \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 = \boldsymbol{X}(\boldsymbol{q})\, u_1 + \boldsymbol{Y}(\boldsymbol{q})\, u_2, \tag{15}$$

where configuration $\boldsymbol{q}$ has got an extra component q_4 describing the steering wheel angle. For simplicity, it was assumed that $L = 1$. Both models are nonholonomic ones because generators $\boldsymbol{X}, \boldsymbol{Y}$ supplemented with the Lie bracket $[\boldsymbol{X}, \boldsymbol{Y}]$ for the unicycle

$$[\boldsymbol{X}, \boldsymbol{Y}] = \begin{bmatrix} \sin(\boldsymbol{q}_3) \\ -\cos(\boldsymbol{q}_3) \\ 0 \end{bmatrix}, \tag{16}$$

and

$$[\boldsymbol{X}, \boldsymbol{Y}] = \begin{bmatrix} L\cos(\boldsymbol{q}_3)\sin(\boldsymbol{q}_4) \\ L\sin(\boldsymbol{q}_3)\sin(\boldsymbol{q}_4) \\ -\cos(\boldsymbol{q}_4) \\ 0 \end{bmatrix}, \quad [\boldsymbol{X}, [\boldsymbol{X}, \boldsymbol{Y}]] = \begin{bmatrix} -L\sin(\boldsymbol{q}_3) \\ L\cos(\boldsymbol{q}_3) \\ 0 \\ 0 \end{bmatrix}, \quad [\boldsymbol{Y}, [\boldsymbol{X}, \boldsymbol{Y}]] = \boldsymbol{X}. \tag{17}$$

for the kinematic car, satisfy Condition (4).

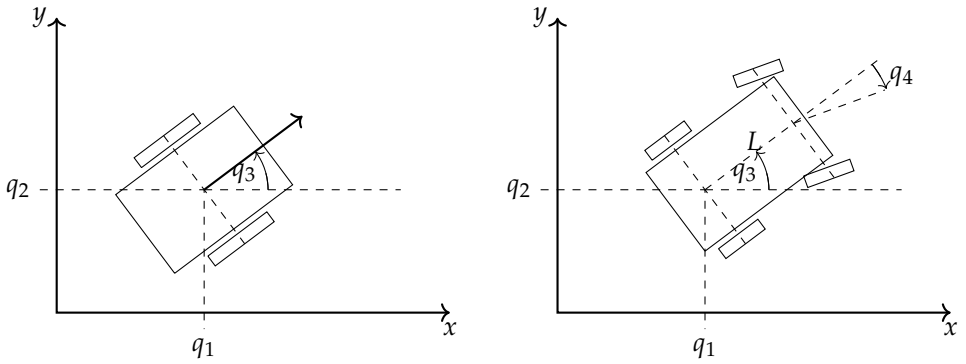

Figure 1. The unicycle (**left** panel) and the kinematic car (**right** panel).

For both models, the position on the xy plane was selected as the output function

$$\boldsymbol{k}(\boldsymbol{q}) = (q_1, q_2)^T. \tag{18}$$

If a robot is inscribed in a circle of sufficient radius, its orientation does not influence the possibility of collisions with obstacles. For the kinematic car another output function was used, coupling position on the xy plane and orientation of the vehicle

$$\boldsymbol{k}(\boldsymbol{q}) = (q_1, q_2, q_3)^T, \tag{19}$$

In practice, the orientation of the steering axle is not relevant. In order to check the motion planning Algorithm 1 within a configuration space, the identity output function was selected for both mobile robots.

In all tested cases, an initial configuration was set to the vector composed of zeroes only

$$\boldsymbol{q}_0 = \boldsymbol{0}. \tag{20}$$

In the tests, a side-way motion maneuver on the xy-plane was planned for both models and all output functions. The maneuver is challenging as it requires a motion along higher (than generators $\boldsymbol{X}, \boldsymbol{Y}$) degree vector fields. Depending on a dimension of a task-space, goal points were selected as

$$\boldsymbol{x}_f = (0, \delta), \quad \boldsymbol{x}_f = (0, \delta, 0), \quad \delta \in \{0.05, 0.1, 0.2, 0.5, 0.7, 1\}. \tag{21}$$

Any single task was run using three parameterizations of controls. Each control was composed of a few very first items of the orthonormal Fourier basis

$$\begin{aligned} u_1 &= p_1^1\rho + p_1^2\sqrt{2}\rho\sin(\omega t) + p_1^3\sqrt{2}\rho\cos(\omega t), \\ u_2 &= p_2^1\rho + p_2^2\sqrt{2}\rho\sin(\omega t) + p_2^3\sqrt{2}\rho\cos(\omega t) \end{aligned} \tag{22}$$

$$u_1 = p_1^1\rho + p_1^2\sqrt{2}\rho\sin(\omega t), \quad u_2 = p_2^1\rho + p_2^3\sqrt{2}\rho\cos(\omega t) \tag{23}$$

$$u_1 = p_1^1\rho + p_1^3\sqrt{2}\rho\cos(\omega t), \quad u_2 = p_2^1\rho + p_2^2\sqrt{2}\rho\sin(\omega t) \tag{24}$$

where $\rho = \sqrt{1/T}$ and $\omega = 2\pi/T$. Later on, it was assumed that $T = 1$. The total energy expenditure on controls is equal to

$$energy(\boldsymbol{p}) = \sum_{i=1}^{2}\sum_{j=1}^{\#N_i}(p_i^j)^2, \tag{25}$$

where $\#N_i$ is the number of the Fourier basis elements contributing to the i-th control. All tasks, with the data collected in Table 1, were solved using Algorithm 1. In Step 6 of the algorithm, two versions of the Newton algorithm were tested: the basic one and with

the energy (25) optimization within the null-space of the Jacobi matrix (13). In order to obtain statistically significant results, Step 6 was repeated 100 times for each task, with initial values of $\boldsymbol{p}$ varied (each component of $\boldsymbol{p}$ was generated randomly with the uniform distribution on the interval $[-1, 1]$). Programs used in simulations were written in Wolfram Mathematica, version 11.3, and run on a computer with Intel® Core™ i5-8400 CPU and 24 GiB RAM memory. In Figures 2 and 3, some characteristic paths (projected on the xy-plane and with a target point marked with an asterisk) were visualized for the unicycle and the kinematic car, respectively.

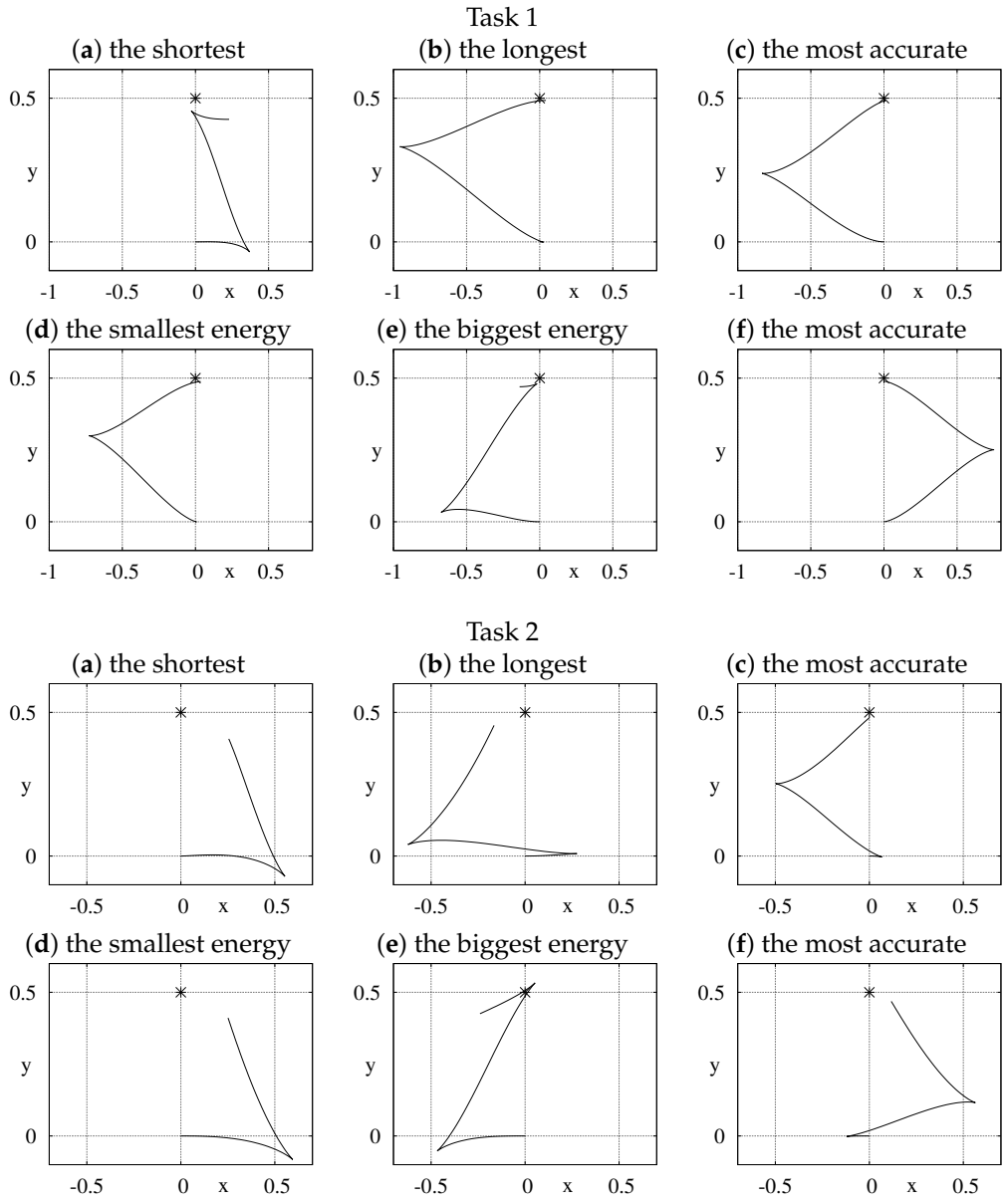

Figure 2. *Cont.*

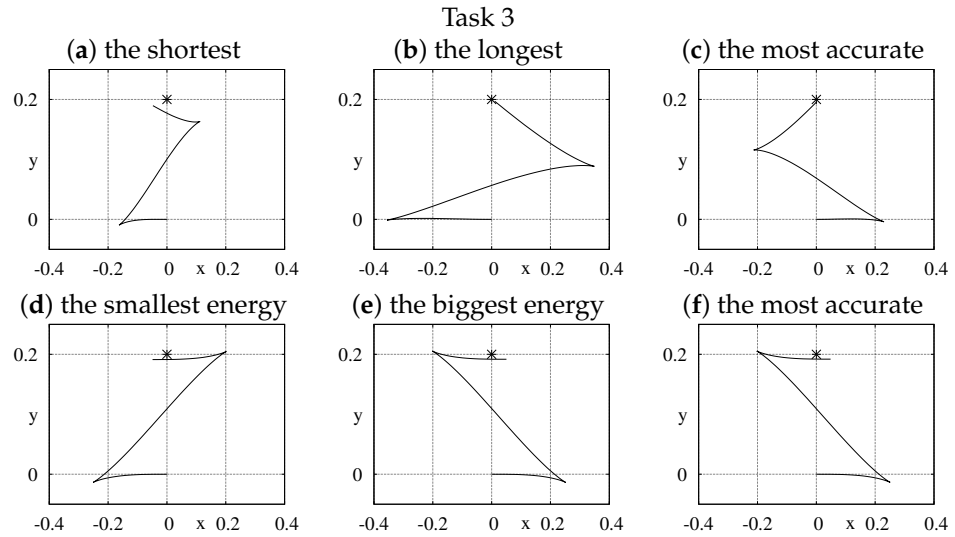

Figure 2. *xy*-plane projections of some paths generated with basic (**a**–**c**) and null-space optimization (**d**–**f**) versions of the Newton algorithm run for the unicycle robot.

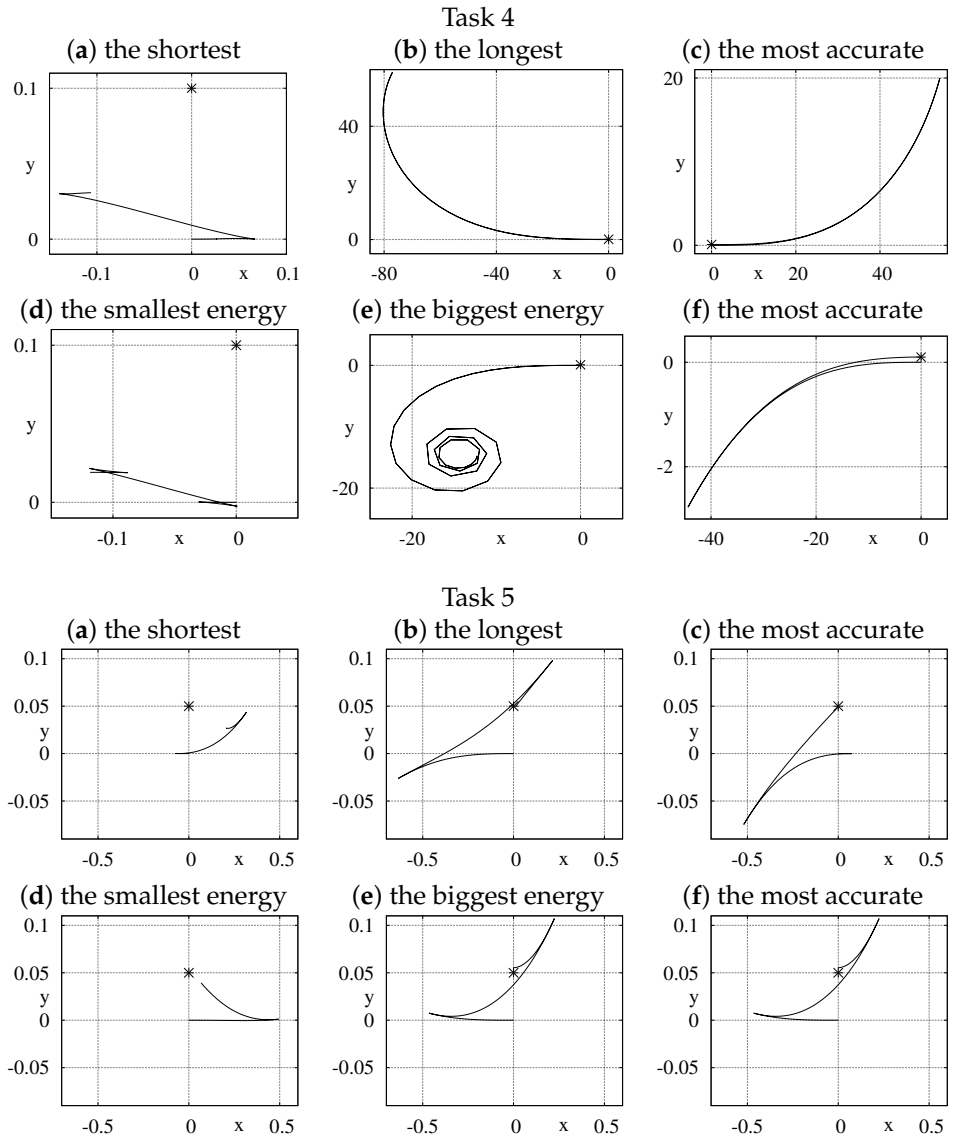

Figure 3. *Cont.*

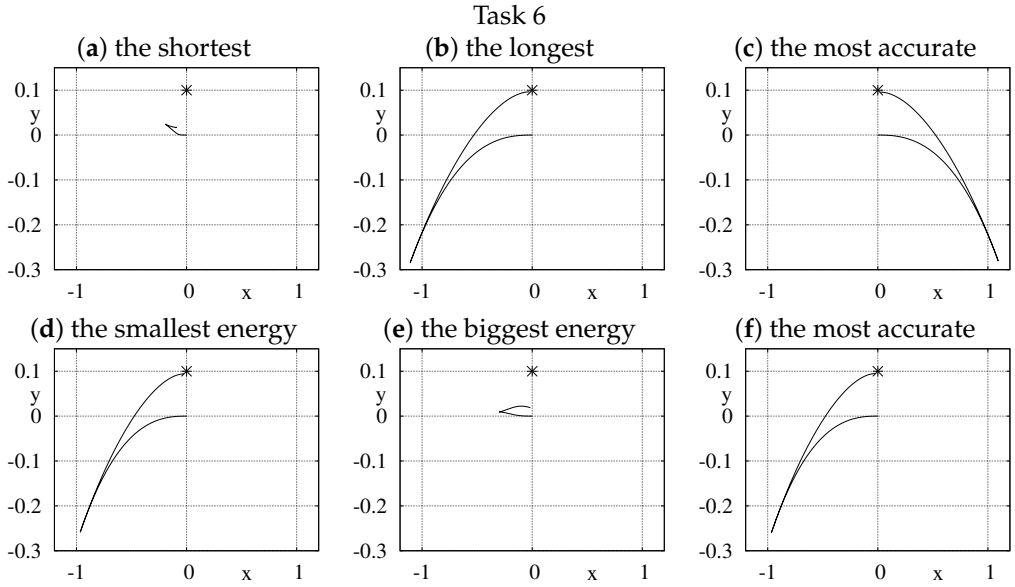

Figure 3. *xy*-plane projections of some paths generated with basic (**a**–**c**) and null-space optimization (**d**–**f**) versions of the Newton algorithm run for the kinematic car.

Table 1. Data for tasks.

Task No.	Model	Output Function	Controls	δ
1	unicycle	identity	Equation (22)	0.5
2	unicycle	Equation (18)	Equation (22)	0.5
3	unicycle	Equation (18)	Equation (24)	0.2
4	kinematic car	identity	Equation (22)	0.1
5	kinematic car	Equation (18)	Equation (22)	0.05
6	kinematic car	Equation (19)	Equation (23)	0.1

Some numeric data describing the quality of solutions were collected in Tables 2 and 3. The abbreviations used:

b.a. —the best end-point accuracy obtained;

a. —how many generated paths were accurate enough,

f. —the percentage of the Newton algorithm runs which caused numerical problems due to an inversion of near-singularity matrices,

ctime —the average computation time. It is worth noticing that the time includes all stages of running each task, i.e., a symbolic generation of the Ph. Hall basis, kinematic-like mapping, and Jacobians and, finally, a truly numeric trajectory generation.

A path is considered to be accurate if an end-point of generated trajectory $\boldsymbol{x}_{freal}$ in the task-space is close enough to the target point $\boldsymbol{x}_f$, i.e.,

$$\|\boldsymbol{x}_f - \boldsymbol{x}_{freal}\| < \eta \|\boldsymbol{x}_f - \boldsymbol{x}_0\|, \tag{26}$$

where $\boldsymbol{x}_0 = \boldsymbol{k}(\boldsymbol{q}_0)$—the starting point and η— an accuracy coefficient. In Tables 2 and 3, data were collected for $\eta = 0.3$. The most accurate path satisfies

$$\min \|\boldsymbol{x}_f - \boldsymbol{x}_{freal}\|. \tag{27}$$

In Tables 4 and 5, statistical data (length and energy ranges for paths satisfying Condition (26)) were collected. The case when no path meets Condition (26) is marked with a dash (or not included at all when both versions of the Newton algorithm failed).

Table 2. Accuracy, numeric failures, and computing time for the unicycle tasks.

			The Newton Algorithm							
			Basic				Null-Space Optimization			
Output Function	**Control Param.**	δ	**b.a.**	**a. [%]**	**f. [%]**	**Ctime [s]**	**b.a.**	**a. [%]**	**f. [%]**	**Ctime [s]**
ident.	(22)	0.05	0.000	100	0	1.7	0.001	100	0	1.7
		0.1	0.001	100	0	1.6	0.001	100	0	1.6
		0.2	0.002	98	0	1.5	0.003	100	0	1.5
		0.5	0.009	49	0	1.5	0.011	62	0	1.5
		0.7	0.017	43	0	1.4	0.020	42	0	1.4
		1	0.037	29	0	1.2	0.040	31	0	1.3
	(23)	0.05	0.000	100	0	0.6	0.000	100	0	0.6
		0.1	0.000	100	0	0.5	0.001	100	0	0.5
		0.2	0.000	100	0	0.4	0.002	100	0	0.5
		0.5	0.006	100	0	0.4	0.010	100	0	0.5
		0.7	0.012	100	0	0.4	0.020	100	0	0.4
		1	0.028	100	0	0.4	0.040	100	0	0.4
	(24)	0.05	0.002	100	0	0.5	0.005	100	0	0.6
		0.1	0.010	99	0	0.5	0.017	100	0	0.5
		0.2	0.032	82	0	0.4	0.049	100	0	0.6
		0.5	0.157	0	0	0.4	0.195	0	0	0.5
		0.7	0.262	0	0	0.4	0.322	0	0	0.5
		1	0.454	0	0	0.4	0.546	0	0	0.4
(18)	(22)	0.05	0.001	6	5	42.3	0.006	1	0	42.4
		0.1	0.004	7	3	43.1	0.018	1	0	43.2
		0.2	0.007	4	0	38.5	0.051	1	1	38.5
		0.5	0.136	1	0	42.7	0.172	0	0	42.8
		0.7	0.155	2	2	40.3	0.257	0	0	40.3
		1	0.328	0	3	39.2	0.381	0	0	39.2
	(23)	0.05	0.005	10	15	13.7	0.042	0	0	13.8
		0.1	0.016	2	20	13.0	0.083	0	1	13.2
		0.2	0.095	0	21	13.0	0.164	0	0	13.4
		0.5	0.388	0	5	12.6	0.406	0	1	12.8
		0.7	0.546	0	3	12.4	0.567	0	2	12.5
		1	0.783	0	2	12.6	0.809	0	1	12.7
	(24)	0.05	0.009	7	11	15.6	0.005	68	2	16.6
		0.1	0.033	0	13	15.0	0.017	62	4	15.2
		0.2	0.069	0	3	15.0	0.037	43	7	15.1
		0.5	0.103	1	5	15.3	0.207	0	8	15.4
		0.7	0.296	0	2	15.0	0.329	0	17	15.1
		1	0.553	0	4	14.7	0.539	0	11	14.8

Table 3. Accuracy, numeric failures, and computing time for the kinematic car tasks.

			The Newton Algorithm							
			Basic				Null-Space Optimization			
Output Function	**Control Param.**	δ	**b.a.**	**a. [%]**	**f. [%]**	**Ctime [s]**	**b.a.**	**a. [%]**	**f. [%]**	**Ctime [s]**
ident.	(22)	0.05	0.001	8	14	31.8	0.001	6	8	32.4
		0.1	0.021	1	18	30.8	0.004	9	8	31.5
		0.2	0.077	0	18	30.5	0.022	3	9	31.3
		0.5	0.017	3	14	28.3	0.005	7	4	29.4
		0.7	0.292	0	19	30.6	0.104	4	3	31.5
		1	0.446	0	25	30.6	0.030	7	4	31.7
	(23)	0.05	0.001	7	0	10.0	0.001	12	3	10.4
		0.1	0.001	4	8	9.2	0.001	12	31	9.5
		0.2	0.244	0	10	9.5	0.000	11	27	9.9
		0.5	0.507	0	22	8.4	0.003	7	40	8.8
		0.7	0.679	0	23	10.0	0.011	7	36	10.3
		1	0.940	0	34	10.0	0.015	7	35	10.5
	(24)	0.05	0.001	47	53	12.0	0.001	48	52	12.8
		0.1	0.001	52	48	11.1	0.001	53	47	12.3
		0.2	—	0	100	—	0.000	46	54	14.7
		0.5	—	0	100	—	0.002	40	60	15.0
		0.7	—	0	100	—	0.003	36	64	14.8
		1	—	0	100	—	0.008	42	58	14.1
(18)	(22)	0.05	0.001	6	5	42.3	0.006	1	0	42.4
		0.1	0.004	7	3	43.1	0.018	1	0	43.2
		0.2	0.007	4	0	38.5	0.051	1	1	38.5
		0.5	0.136	1	0	42.7	0.172	0	0	42.8
		0.7	0.155	2	2	40.3	0.257	0	0	40.3
		1	0.328	0	3	39.2	0.381	0	0	39.2
	(23)	0.05	0.005	10	15	13.7	0.042	0	0	13.8
		0.1	0.016	2	20	13.0	0.083	0	1	13.2
		0.2	0.095	0	21	13.0	0.164	0	0	13.4
		0.5	0.388	0	5	12.6	0.406	0	1	12.8
		0.7	0.546	0	3	12.4	0.567	0	2	12.5
		1	0.783	0	2	12.6	0.809	0	1	12.7
	(24)	0.05	0.009	7	11	15.6	0.005	68	2	16.6
		0.1	0.033	0	13	15.0	0.017	62	4	15.2
		0.2	0.069	0	3	15.0	0.037	43	7	15.1
		0.5	0.103	1	5	15.3	0.207	0	8	15.4
		0.7	0.296	0	2	15.0	0.329	0	17	15.1
		1	0.553	0	4	14.7	0.539	0	11	14.8

Table 3. *Cont.*

			The Newton Algorithm							
			Basic				Null-Space Optimization			
Output Function	**Control Param.**	δ	**b.a.**	**a. [%]**	**f. [%]**	**Ctime [s]**	**b.a.**	**a. [%]**	**f. [%]**	**Ctime [s]**
(19)	(22)	0.05	0.002	58	7	43.4	0.005	94	0	43.5
		0.1	0.005	53	5	43.2	0.018	93	1	43.3
		0.2	0.015	32	4	43.2	0.052	63	3	43.3
		0.5	0.070	30	1	42.2	0.175	0	1	42.3
		0.7	0.142	17	1	42.4	0.260	0	0	42.5
		1	0.210	6	2	43.0	0.384	0	0	43.2
	(23)	0.05	0.007	24	0	13.9	0.007	29	0	14.0
		0.1	0.024	19	0	13.8	0.024	28	0	13.9
		0.2	0.073	0	0	13.7	0.074	0	0	13.8
		0.5	0.303	0	0	13.5	0.304	0	1	13.6
		0.7	0.496	0	2	13.7	0.496	0	0	13.8
		1	0.788	0	0	13.4	0.815	0	0	13.4
	(24)	0.05	0.005	46	1	16.5	0.005	59	0	17.1
		0.1	0.020	10	0	15.1	0.019	61	0	15.2
		0.2	0.074	0	0	14.2	0.056	63	1	14.2
		0.5	0.228	0	0	13.9	0.208	0	0	13.9
		0.7	0.349	0	0	13.7	0.330	0	0	13.8
		1	0.555	0	0	13.7	0.540	0	3	13.8

Table 4. Path length and energy of motion of the unicycle.

			The Newton Algorithm			
Output Function	**Control Param.**	δ	**Basic**		**Null-Space Optimization**	
			Length	**Energy**	**Length**	**Energy**
ident.	(22)	0.05	0.62 ÷ 2.63	0.27 ÷ 1.24	0.62 ÷ 0.69	0.42 ÷ 0.54
		0.1	1.26 ÷ 3.02	0.51 ÷ 1.35	1.24 ÷ 1.25	0.71 ÷ 0.71
		0.2	2.52 ÷ 3.95	0.79 ÷ 1.68	2.50 ÷ 2.51	1.01 ÷ 1.01
		0.5	6.30 ÷ 7.55	1.29 ÷ 2.08	6.27 ÷ 6.28	1.59 ÷ 1.60
		0.7	8.80 ÷ 9.70	1.57 ÷ 2.31	8.78 ÷ 8.79	1.89 ÷ 1.89
		1	12.60 ÷ 13.58	2.14 ÷ 2.73	12.55 ÷ 12.55	2.26 ÷ 2.26
	(23)	0.05	0.62 ÷ 1.66	0.23 ÷ 1.14	0.62 ÷ 0.63	0.47 ÷ 0.51
		0.1	1.26 ÷ 2.67	0.36 ÷ 1.23	1.24 ÷ 1.26	0.71 ÷ 0.71
		0.2	2.51 ÷ 3.86	0.61 ÷ 1.59	2.50 ÷ 2.51	1.01 ÷ 1.01
		0.5	6.28 ÷ 7.26	1.22 ÷ 2.10	6.27 ÷ 6.28	1.59 ÷ 1.60
		0.7	8.80 ÷ 9.75	1.50 ÷ 2.35	8.78 ÷ 8.79	1.89 ÷ 1.89
		1	12.57 ÷ 13.61	1.84 ÷ 2.69	12.55 ÷ 12.56	2.26 ÷ 2.26
	(24)	0.05	0.62 ÷ 2.92	0.23 ÷ 1.53	0.62 ÷ 0.63	0.50 ÷ 0.50
		0.1	1.25 ÷ 2.27	0.43 ÷ 1.30	1.24 ÷ 1.26	0.71 ÷ 0.71
		0.2	2.51 ÷ 3.67	0.84 ÷ 1.60	2.50 ÷ 2.51	1.01 ÷ 1.01

Table 4. *Cont.*

Output Function	Control Param.	δ	The Newton Algorithm			
			Basic		Null-Space Optimization	
			Length	Energy	Length	Energy
(18)	(22)	0.05	0.41 ÷ 3.03	0.18 ÷ 1.04	0.36 ÷ 0.62	0.35 ÷ 0.50
		0.1	0.76 ÷ 3.01	0.36 ÷ 1.18	0.72 ÷ 1.28	0.50 ÷ 0.70
		0.2	1.65 ÷ 4.07	0.60 ÷ 1.27	1.83 ÷ 1.99	0.86 ÷ 0.90
		0.5	4.35 ÷ 6.28	1.21 ÷ 1.63	4.80 ÷ 4.80	1.39 ÷ 1.39
		0.7	6.45 ÷ 9.79	1.40 ÷ 1.88	—	—
		1	9.00 ÷ 13.46	1.72 ÷ 2.23	—	—
	(23)	0.05	0.38 ÷ 1.91	0.18 ÷ 0.90	0.36 ÷ 0.41	0.34 ÷ 0.41
		0.1	0.73 ÷ 2.07	0.33 ÷ 0.91	0.72 ÷ 0.73	0.54 ÷ 0.54
		0.2	1.51 ÷ 2.89	0.61 ÷ 0.98	—	—
		0.5	4.31 ÷ 5.12	1.13 ÷ 1.30	—	—
		0.7	6.51 ÷ 6.78	1.44 ÷ 1.47	—	—
	(24)	0.05	0.63 ÷ 2.18	0.22 ÷ 1.13	0.62 ÷ 0.67	0.48 ÷ 0.60
		0.1	1.26 ÷ 2.88	0.45 ÷ 1.39	1.24 ÷ 1.28	0.68 ÷ 0.71
		0.2	2.52 ÷ 4.33	0.65 ÷ 1.42	2.50 ÷ 2.51	1.01 ÷ 1.01
		0.5	6.37 ÷ 7.91	1.26 ÷ 1.94	—	—
		0.7	9.04 ÷ 10.42	1.56 ÷ 2.32	—	—
		1	12.88 ÷ 14.18	1.90 ÷ 2.65	—	—

Table 5. Path length and energy of motion for the kinematic car.

Output Function	Control Param.	δ	The Newton Algorithm			
			Basic		Null-Space Optimization	
			Length	Energy	Length	Energy
ident.	(22)	0.05	6912 ÷ 49,083	74 ÷ 199	5166 ÷ 5392	64 ÷ 66
		0.1	18,281 ÷ 18,281	121 ÷ 121	9788÷ 10,735	89 ÷ 93
		0.2	—	—	18,057 ÷ 19,637	120 ÷ 126
		0.5	44,864 ÷ 135,750	190 ÷ 331	41,005 ÷ 45,333	182 ÷ 191
		0.7	—	—	59,707 ÷ 75,423	219 ÷ 247
		1	—	—	82,748 ÷ 87,747	258 ÷ 266
	(23)	0.05	21,332 ÷ 45,418	131 ÷ 191	4049 ÷ 24,218	57 ÷ 140
		0.1	76,600 ÷ 79,107	249 ÷ 253	8310 ÷ 22,968	82 ÷ 136
		0.2	—	—	18,926 ÷ 137,973	123 ÷ 334
		0.5	—	—	40,082 ÷ 162,380	180 ÷ 362
		0.7	—	—	55,316 ÷ 113,002	211 ÷ 302
		1	—	—	80,542 ÷ 197,181	255 ÷ 399
	(24)	0.05	21,146 ÷ 123,282	130 ÷ 316	4020 ÷ 86,003	57 ÷ 264
		0.1	86,087 ÷ 163,986	264 ÷ 364	8229 ÷ 105,744	81 ÷ 292
		0.2	—	—	17,406 ÷ 176,427	118 ÷ 378
		0.5	—	—	54,373 ÷ 142,871	209 ÷ 340
		0.7	—	—	76,363 ÷ 211,192	248 ÷ 413
		1	—	—	106,666 ÷ 219,503	294 ÷ 421
(18)	(22)	0.05	2.63 ÷ 5.96	0.94 ÷ 1.70	5.28 ÷ 5.28	1.40 ÷ 1.40
		0.1	4.43 ÷ 10.33	1.20 ÷ 1.61	9.00 ÷ 9.00	1.71 ÷ 1.71
		0.2	8.17 ÷ 17.87	1.55 ÷ 1.93	15.87 ÷ 15.87	2.11 ÷ 2.11
		0.5	37.16 ÷ 37.16	2.55 ÷ 2.55	—	—
		0.7	39.00 ÷ 49.05	1.08 ÷ 2.93	—	—
	(23)	0.05	2.20 ÷ 3.39	0.88 ÷ 1.09	—	—
		0.1	4.19 ÷ 4.80	1.07 ÷ 1.12	—	—
	(24)	0.05	5.31 ÷ 5.65	1.10 ÷ 1.34	5.22 ÷ 5.29	1.40 ÷ 1.40
		0.1	—	—	8.95 ÷ 9.01	1.70 ÷ 1.71
		0.2	—	—	15.90 ÷ 15.94	2.11 ÷ 2.19
		0.5	35.86 ÷ 35.86	3.19 ÷ 3.19	—	—

Table 5. *Cont.*

Output Function	Control Param.	δ	The Newton Algorithm			
			Basic		Null-Space Optimization	
			Length	Energy	Length	Energy
(19)	(22)	0.05	5.38 ÷ 13.11	0.98 ÷ 2.50	5.23 ÷ 6.80	1.40 ÷ 1.73
		0.1	9.44 ÷ 27.03	1.35 ÷ 3.79	8.95 ÷ 10.80	1.70 ÷ 2.04
		0.2	16.93 ÷ 29.52	1.71 ÷ 3.57	15.83 ÷ 15.87	2.11 ÷ 2.11
		0.5	36.59 ÷ 54.22	2.22 ÷ 3.36	—	—
		0.7	49.38 ÷ 57.04	2.51 ÷ 3.03	—	—
		1	66.52 ÷ 76.19	2.92 ÷ 3.62	—	—
	(23)	0.05	6.77 ÷ 7.24	1.43 ÷ 1.95	6.75 ÷ 6.80	1.72 ÷ 1.73
		0.1	10.81 ÷ 11.14	1.95 ÷ 2.33	10.73 ÷ 10.79	2.04 ÷ 2.04
	(24)	0.05	5.30 ÷ 5.93	1.01 ÷ 1.62	5.22 ÷ 5.29	1.40 ÷ 1.41
		0.1	9.04 ÷ 9.36	1.40 ÷ 1.68	8.96 ÷ 9.01	1.70 ÷ 1.71
		0.2	—	—	15.90 ÷ 15.94	2.12 ÷ 2.12

Based on results of the tests, some observations can be formulated:

1. Length and energy of resulting paths strongly depend on initial values of $\boldsymbol{p}_c$, especially when the basic Newton algorithm is used. The Newton algorithm, being local and iterative, has to generate trajectory-dependent results (the next iteration output depends on the output of the current iteration) and it tends to get into a desired location as fast as possible. It may happen that the basic algorithm provides better results than the optimized one. However, a smaller variance of results was observed in the energy-optimized version. In this case, the search space is searched more thoroughly.
2. Trajectories generated with the optimized version of the Newton algorithm usually form a few clusters corresponding to local minima. This feature was not observed in the basic algorithm, as trajectories are not constrained by the energy minimization.
3. Although the null-space optimization Newton algorithm does not necessarily guarantee the best accuracy, still much more often it promptly reaches the target.
4. The failure ratio, cf. Equation (26), is much bigger for non-redundant parameterizations. What is interesting, in some tasks, parameterizations with the same number of parameters, cf. (23) and (24), generate significantly different results. Sometimes a task is solvable in one parameterization, while it is not in the other.
5. The optimization within the null space significantly increases computational costs. In order to decrease the costs, this version of the Newton algorithm might be run after the target point was reached using the basic algorithm.
6. For the unicycle robot and the identity output function, the quality of the solutions was generally better than that for tasks with smaller dimensional task-spaces, likely due to simplicity of the model and a small number of local minima. However, the length of a path and an energy of motion is smaller for tasks with low-dimensional output functions as it is less restricted.
7. Obviously, the higher dimensional output space is, the more difficult task is generated. Additionally, a high redundancy in a parameter space is more computationally costly, but gives more flexibility as well. Thus, a compromise between flexibility and costs has to be found.
8. Computational costs are smaller for the identity output function than for other output functions, as there is no need to transfer data between configuration and task spaces.
9. Short length steps, characterized by small values of δ, are easier to control.
10. There are a few sources of inaccuracies in motion planning using Lie-algebraic methods: the truncation of the Lie series (5), evaluation of vector fields at q_c although a trajectory surrounds the point (quite well visible for long-range motions) numeric properties of the Newton algorithm. All of them can be controlled by the right choice of parameters.

11. A planning accuracy varies substantially with the current configuration q_c, as vector fields in the Lie-algebraic methods are evaluated at the point, cf. Equation (5), and vector fields exhibit different variability around different $\boldsymbol{q}_c$.
12. In almost all xy-projection plots one can notice characteristic cusps, which are typical in Lie-algebraic motion planning. At a close vicinity of a cusp a tangent line to the trajectory remains almost the same, while direction of motion is changed. It means that due to particular values of controls and generators, some components of velocity $\dot{q}$ in Equation (1) change their signs while passing through zero. An illustrative example of such behavior can be constructed by setting: $\boldsymbol{X}$ long, $\boldsymbol{Y}$ short, u_2 small, and u_1 control passing though zero. Therefore, one can expect that the number of cusps strongly and positively correlates with frequencies of controls. For mobile robots tested the explanation is even simpler, as the cusps result from linear velocity sign changes.
13. Many trajectory cusps result in a zigzag motion, which is especially useful in obstacle-cluttered environments as the volume of maneuvers can be controlled and kept small. However, it is also visible that generating a relatively short motion along high degree vector fields requires a relatively long path to follow.
14. Results for the kinematic car are the most interesting, especially for the identity output function. In this case, a hard to meet condition $q_4 = 0$ forces extensive movement. This condition is particularly difficult to satisfy for non-redundant parameterizations. Therefore, in some sub-figures of Task 4, cf. Figure 3, scales of axes were extended to present the whole paths. Nevertheless, the point $\boldsymbol{x}_{freal}$ (cf. Figure 3 b,c,e,f) is close to the desired point $\boldsymbol{x}_f$ as required.
15. It should be noted that planning a motion of the kinematic car is qualitatively more difficult than of the unicycle, as a high degree (equal to 3) of vector fields have to be involved, comparing to degree 2 for the unicycle, cf. (16), (17).
16. It may look strange that in Task 6, cf. Figure 3e, the projected path with the biggest energy is relatively short. In this case, the major contribution to the energy consumption was caused by control $\boldsymbol{u}_2$, responsible for reorientation of the robot.

4. Conclusions

In this paper, the Lie-algebraic method of motion planning for nonholonomic systems, originally designed to plan within the configuration space, has been extended to also work in a task-space. There are some advantages of planning within the task-space:

- a smaller than the configuration space dimension either increases redundancy and facilitates optimizing a motion or allows to decrease the number of parameters of controls,
- it allows to define and solve many practical tasks, especially in obstacle-cluttered environments when some components of the configuration vector are not important.

However, there are also some disadvantages:

- a new source of singularities introduced by the output mapping,
- the impossibility to plan only within the task-space, without referring explicitly to the configuration space (usually the task-space dimension is smaller than the configuration space dimension, thus no unique mapping between the two spaces exists).

Some general observations based on simulation results:

- Depending on the planning purpose, the Newton algorithm, extensively used in the proposed algorithm, can be applied with or without optimization. The first version (low computational costs) should be used for tasks with no extra requirements or as a first try solution.
- A small redundancy in setting the search space size ($= \dim(\boldsymbol{p})$) is advised not to cause a significant increase of computational costs, while preserving flexibility and solvability of planning tasks.

- An initial value of the parameter vector $\boldsymbol{p}$ highly impacts solvability of the planning task and also the resulting trajectory shape. Therefore, the multi-start technique is recommended, especially when planning is performed in the off-line mode.

Our future research will concentrate on applications of the Lie-algebraic method in difficult environments (narrow passages, non-convex obstacles) in the task-space. In such environments, the planning should be performed through selection of sub-goals on the way to the target position. The main difficulty is the sub-goal selection is caused by the fact that nonholonomic systems are not governed by the Euclidean geometry, but rather the sub-Riemannian one [19]. The Lie-algebraic method is local and general-purpose, thus it is particularly well suited for solving tasks in partially known or dynamic environments.

Author Contributions: Conceptualization, A.M. and I.D.; methodology, A.M. and I.D.; software, A.M.; validation, A.M. and I.D.; formal analysis, A.M. and I.D.; resources, A.M. and I.D.; writing—original draft preparation, I.D.; writing—review and editing, I.D.; visualization, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Wroclaw University of Science and Technology within Grant No. 821 110 41 60.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Duleba, I. Kinematic models of doubly generalized N-trailer systems. *J. Intell. Robot. Syst.* **2019**, *94*, 135–142. [CrossRef]
2. Vafa, Z.; Dubowsky, S. The kinematics and dynamics of space manipulators: The virtual manipulator approach. *Int. J. Robot. Res.* **1990**, *9*, 3–21. [CrossRef]
3. Dubins, L.E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *Am. J. Math.* **1957**, *79*, 497–516. [CrossRef]
4. Reeds, J.A.; Shepp, L.A. Optimal paths for a car that goes both forwards and backwards. *Pac. J. Math.* **1990**, *145*, 367–393. [CrossRef]
5. Murray, R.M.; Sastry, S.S. Nonholonomic Motion Planning: Steering Using Sinusoids. *IEEE Trans. Autom. Control* **1993**, *38*, 201–215. [CrossRef]
6. Dulęba, I.; Sąsiadek, J. Nonholonomic motion planning based on Newton algorithm with energy optimization. *IEEE Trans. Control. Syst. Technol.* **2003**, *11*, 355–363. [CrossRef]
7. Tchoń, K.; Jakubiak, J. Endogenous configuration space approach to mobile manipulators: A derivation and performance assessment of Jacobian inverse kinematics algorithms. *Int. J. Control* **2003**, *26*, 1387–1419. [CrossRef]
8. Duleba, I.; Wissem Khefifi, W.; Karcz-Duleba, I. Layer, Lie algebraic method of motion planning for nonholonomic systems. *J. Frankl. Inst.* **2012**, *349*, 201–215. [CrossRef]
9. Jakubczyk, B. Nonholonomic path following with fastly oscillating Controls. In Proceedings of the 9th Workshop on Robot Motion and Control, Wąsowo, Poland, 3–5 July 2013; pp. 99–103.
10. Arismendi, C.; Alvarez, D.; Garrido, S.; Moreno, L. Nonholonomic Motion Planning Using the Fast Marching Square Method. *Int. J. Adv. Robot. Syst.* **2015**, *12*, 1–14. [CrossRef]
11. Gawron, T.; Michałek, M.M. The VFO-driven motion planning and feedback control in polygonal worlds for a unicycle with bounded curvature of motion. *J. Intell. Robot. Syst.* **2018**, *89*, 265–297. [CrossRef]
12. Łakomy, K.; Michałek, M.M. Robust output-feedback VFO-ADR control of underactuated spatial vehicles in the task of following non-parametrized paths. *Eur. J. Control* **2021**, *58*, 258–277. [CrossRef]
13. LaValle, S. *Planning Algorithms*; Cambridge University Press: Cambridge, UK, 2006.
14. Lynch, K.M.; Park, F.C. *Modern Robotics: Mechanics, Planning, Control*; Cambridge University Press: Cambridge, UK, 2017.
15. Chow, W.L. Über Systeme von linearen partiellen Differentialgleichungen erster Ordnung. *Math. Ann.* **1939**, *117*, 98–105.
16. Strichartz, R.S. The Campbell-Baker-Hausdorff-Dynkin Formula and Solutions of Differential Equations. *J. Funct. Anal.* **1987**, *72*, 320–345. [CrossRef]
17. Duleba, I.; Khefifi, W. Pre-control from of the gCBHD formula and its application to nonholonomic motion planning. In Proceedings of the IFAC Symposium on Robot Control, Wroclaw, Poland, 1–3 September 2003; Dulęba, I., Sąsiadek, J.Z. Eds.; Elsevier: Oxford, UK, 2004; Volume 1, pp. 123–128.
18. Nakamura, Y. *Advanced Robotics: Redundancy and Optimization*; Addison-Wesley: Boston, MA, USA, 1991.
19. Agrachev, A.; Barilari, D.; Boscain, U. *A Comprehensive Introduction to Sub-Riemannian Geometry*; Cambridge University Press: Cambridge, UK, 2019.