Using Lie Group Symmetries for Fast Corrective Motion Planning

Konstantin Seiler, Surya P.N. Singh, and Hugh Durrant-Whyte

Abstract. For a mechanical system it often arises that its planned motion will need to be corrected either to refine an approximate plan or to deal with disturbances. This paper develops an algorithmic framework allowing for fast and elegant path correction for nonholonomic underactuated systems with Lie group symmetries, which operates without the explicit need for control strategies. These systems occur frequently in robotics, particularly in locomotion, be it ground, underwater, airborne, or surgical domains. Instead of reintegrating an entire trajectory, the method alters small segments of an initial trajectory in a consistent way so as to transform it via symmetry operations. This approach is demonstrated for the cases of a kinematic car and for flexible bevel tip needle steering, showing a prudent and simple, yet computationally tractable, trajectory correction.

1 Introduction

In practice, mechanical systems drift. Be it due to unexpected disturbances or in order to refine a coarse plan, corrective motion planning seeks to efficiently adapt a given trajectory in an elegant way. This is of particular interest in the agile control of underactuated nonholonomic systems. The nature of these systems is that certain degrees of freedom can only be controlled in a coupled manner (if at all). This makes it computationally hard to determine simple and valid trajectories [3, 11], thus it is preferable to efficiently adapt a given trajectory in an elegant way without having to start anew. Even in cases where explicit control laws are available, pure pursuit tracking is likely to produce unwanted artefacts due to its myopic nature [5,13]. Taking a larger horizon into account increases algorithmic and computational complexity, but enables alterations to the path in an elegant way. An example of such corrections is illustrated in Fig. 1.

Konstantin Seiler · Surya P.N. Singh · Hugh Durrant-Whyte
Australian Centre for Field Robotics, University of Sydney
e-mail: {k.seiler,spns,hugh}@acfr.usyd.edu.au

D. Hsu et al. (Eds.): Algorithmic Foundations of Robotics IX, STAR 68, pp. 37–52.
springerlink.com © Springer-Verlag Berlin Heidelberg 2010

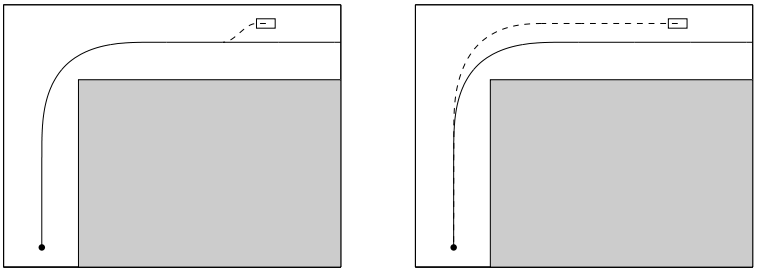

Fig. 1 A car (*small rectangle*) is following a previously planned path (*solid line*) to the goal (*dot*), but got off track due to disturbances. The left image shows a pure pursuit controller trying to get back on track as quickly as possible, resulting in unnecessary turns (*dashed line*). A more natural solution is shown in the second picture where the available space is used to elegantly correct the path during the upcoming turn.

Mechanical systems frequently exhibit symmetries that can be represented as Lie groups of translation or rotation [8, 14, 17]. Exploiting this can allow for elegant trajectory corrections in a computationally tractable way. This is valuable as the degrees of freedom represented within this symmetry group are often the ones that are only indirectly modifiable, and thus hard to control. For example, for most vehicles (be it submersible, ground, or airborne) properties such as thrust, speed and turning rate can be easily influenced; whereas, the position and heading are hard to control. As the latter often exhibits aforementioned symmetries, these methods allow for efficient planning and control of this subset.

Towards this, an algorithmic framework is introduced that allows for elegant planning and control systems that exhibit symmetries but are hard to control due to nonholonomic constraints. The method works without prior knowledge of control strategies specific to the system at hand. Further, it can be used either as an aid within an existing planning technique such as *rapidly exploring random tree* (RRT) or *probabilistic roadmap* (PRM) algorithms [11]; or, as presented here, on its own in order to adapt an existing trajectory and partly replace a classical controller.

This approach generalises on the use of Lie group actions for gap reduction during RRT planning. Cheng [4], for example, introduced a method to insert *coasting trajectories* into an existing trajectory in order to reduce gaps that arise during sampling based planning. That approach is likely to perform well for twisted paths but it comes short for less twisted ones as there is no possibility to shorten any part of an initial trajectory to recover from overshooting. The algorithm presented overcomes this problem by actually altering existing segments of the initial trajectory in a consistent manner.

The following discussion is framed on the assumption that an initial path has been obtained, but needs to be corrected as it does not reach the desired goal. Such corrections might be necessary due to gaps arising from sampling based

planners, dynamic changes in the environment, or due to disturbances. Furthermore the algorithm is designed under the assumption that the initial path is up to some degree surrounded by free space, but may still contain narrow doorway situations. For ease of presentation, this work concentrates on altering degrees of freedom represented by aforementioned symmetry groups. It is understood that the remaining degrees are dealt with via a classical planning or control methods [11].

The remainder of this paper is structured as follows. Section 2 develops the mathematical model and introduces the basic concepts for trajectory alteration. Section 3 presents the algorithmic framework for situations without obstacles and shows results of its application to the (kinematic) car tracking and flexible needle steering problems. This is extended to the cases with obstacles in Section 4. Finally, Section 5 summarises the ideas presented and discusses future applications.

2 Mathematical Model

2.1 Basic Definitions

Kinodynamics can be defined on a state space X, which itself is a differentiable manifold with a metric [11]. The subset $X_{\text{obs}} \subseteq X$ denotes the states that have obstacles, and its complement $X_{\text{free}} := X \setminus X_{\text{obs}}$ is the viable free space. For clarity of presentation, an obstacle free setting ($X_{\text{obs}} = \emptyset$) is assumed (cases with obstacles will be tackled in Section 4). The space $U \subseteq \mathbb{R}^n$ represents the the system's control inputs. System progress is modelled via a set of ordinary differential equations (ODE)

$$\dot{x} = F(x,u) \tag{1}$$

for $x \in X$ and $u \in U$.

It is assumed that the reader is familiar with the notion of Lie groups. An introduction to the topic can be found, for example, in [12,1]. Let G be a Lie group acting on X such that $F(.,u)$ are left-invariant vector fields under the action of G. That is, there exists a multiplication law for elements $g \in G$ and $x \in X$, such that $gx \in X$, and for every trajectory $x(t) : I \subseteq \mathbb{R} \to X$ and control input $u(t) : I \to U$ fulfilling Eq. (1), the product $gx(t)$ also fulfils (1) for the same $u(t)$. This setting often allows for a decomposition of the state space X in the form

$$X = Z \times G$$

where the manifold Z is the *base space* and the Lie group G is denoted the *fibre component*.[1] The projections from X onto its components Z and G are denoted π_Z and π_G respectively. Common examples of such invariantly acting Lie groups arising from the system's symmetry group, are translations ($\mathbb{R}^n$), rotations ($\mathrm{SO}(2)$, $\mathrm{SO}(3)$) or combinations thereof ($\mathrm{SE}(2)$, $\mathrm{SE}(3)$, $\mathbb{R}^3 \times \mathrm{SO}(2)$, ...).

[1] For the decomposition to exist, the Lie group's action has to be *free*. That is, for all $x \in X$ and $g,h \in G$ it has to be true that $gx = hx$ implies $g = h$. If G is a symmetry group of the system, this is usually the case.

Applying this framework to the example of a kinematic car yields a state space X containing five dimensions, denoted speed v, turning rate ω, position x and y, and heading θ. The control inputs U contain two dimensions, acceleration a and change in turning rate ρ. The equations of motion are

$$\begin{aligned}\dot{v} &= a \\ \dot{\omega} &= \rho \\ \dot{x} &= \cos(\theta)v \\ \dot{y} &= \sin(\theta)v \\ \dot{\theta} &= \omega v\,.\end{aligned}$$

Since the car's behaviour is independent of position and heading in the sense that, if a valid path is translated or rotated, the resulting path is equally valid, as illustrated in Fig. 2, these dimensions form a symmetry Lie group to the system. Thus G should be set to be the group of Euler transformations, $\mathrm{SE}(2)$, representing x, y and θ. The remaining base space Z is spanned by v and ω. Thus

$$X = \mathbb{R}^2 \times \mathrm{SE}(2)\,.$$

Introducing some notation simplifies matters. Let $I \subseteq \mathbb{R}$ be a closed finite interval. Then I^- and I^+ denote the lower and upper boundary values respectively, such that

$$I = [I^-, I^+]\,.$$

A *time dependent control input* is considered to be a function $u : I_u \to U$ that maps from a closed finite interval $I_u \subseteq \mathbb{R}$ into the control space U. Integrating such a control input over time via the ODE (1) gives rise to a path in state space X that is dependent on an initial state x_0 and time t. Such integrated paths can be written as functions $\Phi_u(x_0, t) : X \times I_u \to X$ with the properties

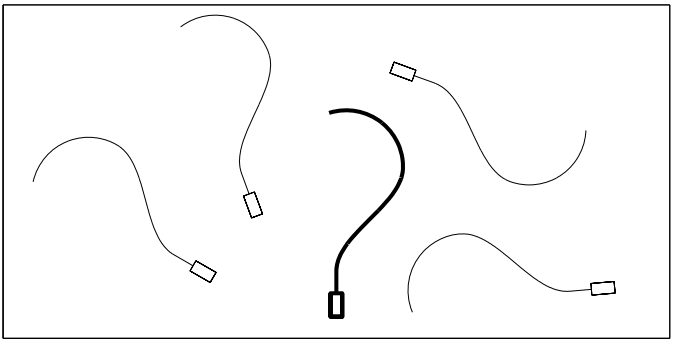

Fig. 2 The car (*rectangle*) has a valid initial state and path depicted in *bold*. It follows that the translated and rotated initial states and paths are equally valid.

$$\dot{\Phi}_u(x_0,t) = F(\Phi_u(x_0,t),u(t))$$

and

$$\Phi(x_0,I_u^-) = x_0 \ .$$

Given two time dependent control inputs $u : I_u \to U$ and $v : I_v \to U$ with $I_u^+ = I_v^-$, $u * v : [I_u^-, I_v^+] \to X$ is defined as the concatenation of the two functions u and v such that

$$(u*v)(t) = \begin{cases} u(t), & \text{if } t \in [I_u^-, I_u^+) \\ v(t), & \text{otherwise.} \end{cases}$$

This notation may also be used in cases where $I_u^+ \neq I_v^-$. In these cases a suitable reparameterisation of I_v is performed implicitly. Note that when using this notation for two integrated paths in state space X, the concatenation results in a single continuous path if and only if the final state of the first path coincides with the initial state of the second path. When this is the case, the resulting path is equivalent to integrating the concatenated control inputs directly, thus

$$\Phi_{u*v}(x_0,t) = \Phi_u(x_0,t) * \Phi_v(\Phi_u(x_0,I_u^+),t) \ .$$

2.2 Trajectory Transformations

It is hard to find a solution for the planning problem of connecting two predefined points x_{start} and x_{goal} in X [3, 10]. In the general case, this leads to running a search over all time varying control inputs. As the space of all possible control inputs can be too big to search exhaustively, many algorithms focus on relatively small subsets and either run a search over a discrete path set [6] or run a non-linear optimisation algorithm or search over a continuous path set [7, 9]. The former, by its very nature, can only reach a discrete subset of X, where as the latter typically involves reintegrating the whole trajectory $\Phi_u(x_{\text{start}},t)$ in each step of the optimisation process.

Using operations given by a Lie group to transform a valid trajectory allows for the reuse of large parts of a previously calculated $\Phi_u(x_{\text{start}},t)$ as long as changes to the trajectory happen in a compatible way. Thus searching a continuum can be done without complete reintegration.

Let u and v be two time dependent control inputs that differ in some region, but coincide otherwise. They can be split up as

$$u = u_1 * u_2 * u_3$$

and

$$v = u_1 * v_2 * u_3$$

where u_1 and u_3 represent the parts that are common to both. Note that the lengths of the middle segments I_{u_2} and I_{v_2} do not necessarily have to be equal. Starting both trajectories at a common initial state $x_0 \in X$ yields

$$\Phi_u(x_0,t) = \Phi_v(x_0,t) \quad \text{for } t \in I_{u_1}.$$

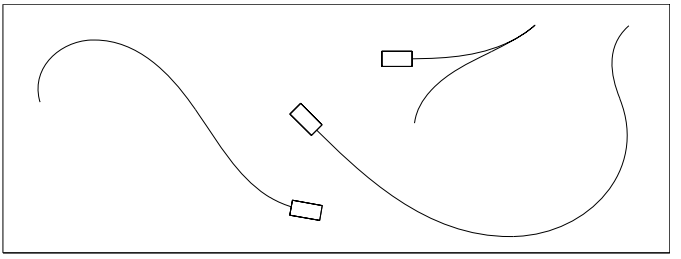

Fig. 3 Three trajectories for a car (*rectangle*), all resulting from the same control input. The behaviour is sensitive to initial conditions (speed and turning rate), causing different trajectories.

In general, equality of the third part of the control inputs, u_3, can not be used, as the final states of the middle segments, $\Phi_u(x_0, I^+_{u_2})$ and $\Phi_u(x_0, I^+_{v_2})$, need not coincide. Using different states as initial states for the third part of the path can result in a variety of different trajectories as illustrated in Fig. 3. If however it is assumed that the final states of the middle segments u_2 and v_2 only differ on the fibre component G but coincide on the base space Z, the similarity of the third parts of the trajectory can be exploited. Having equality on the base space as in

$$\pi_Z(\Phi_u(x_0, I^+_{u_2})) = \pi_Z(\Phi_v(x_0, I^+_{v_2})) \tag{2}$$

implies there exists a transformation $g \in G$ such that

$$\Phi_v(x_0, I^+_{v_2}) = g\Phi_u(x_0, I^+_{u_2}) \,. \tag{3}$$

In the case of the kinematic car, Eq. (2) can be interpreted as having identical speed v and turning rate ω. Then Eq. (3) yields the translation and rotation necessary to transform one state into the other. Because the equations of motion are invariant under translation and rotation, the resulting third parts of the paths will be translated and rotated versions of each other as illustrated in Fig. 4. In the general case, the same line of reasoning on invariance yields

$$\Phi_{u_3}(\Phi_v(x_0, I^+_{v_2}), t) = \Phi_{u_3}(g\Phi_u(x_0, I^+_{u_2}), t) = g\Phi_{u_3}(\Phi_u(x_0, I^+_{u_2}), t) \,. \tag{4}$$

Looking at this result from a viewpoint of computational complexity, Eq. (4) can save calculation time. Given Φ_u, the computational cost of Φ_v is mainly the cost of integrating the second segment given by v_2. The third segment defined by u_3 can be calculated directly by the use of group operations. In particular, during non-linear optimisation, the final state $\Phi_v(x_0, I^+_v)$ is typically the only one of interest. Thus, there is no need to actually transform the whole third segment of the path. Instead one can determine the trajectory's final state directly. As a result, the cost for $\Phi_v(x_0, I^+_v)$ is linear in the size of I_{v_2}.

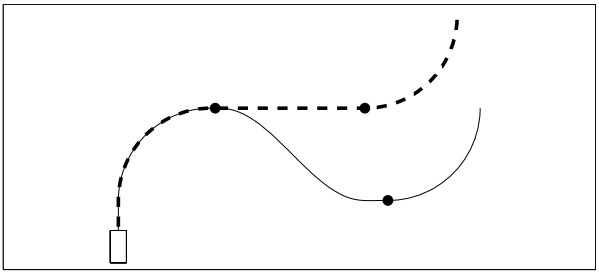

Fig. 4 A car (*rectangle*) follows two different paths resulting from control inputs that only differ on a region in the middle but are identical otherwise. The paths coincide up to the first marker (*dot*). After that, the paths differ. However at the respective second markers, speed and turning rate are identical for both paths and thus the remaining parts of the path are the same, just transformed.

2.3 Optimising a Trajectory

Given a time dependent control input u and a corresponding trajectory $\Phi_u(x_0,t)$, one might be able to find an alteration u^c that stretches (or compresses) the trajectory's behaviour on the base space Z over time. That is

$$\pi_Z(\Phi_{u^c}(x_0,t)) = \pi_Z(\Phi_u(x_0,ct)) \tag{5}$$

for a stretch factor $0 < c \in \mathbb{R}$. In particular, this yields identical final states on Z,

$$\pi_Z(\Phi_{u^c}(x_0,I^+_{u^c})) = \pi_Z(\Phi_u(x_0,I^+_u)) \ .$$

In the case of the car, for instance, this could map to reduced accelerator commands resulting in a longer distance travelled by the time the target speed is reached. While the stretching operation does not change the end result on the base space Z, it does alter the fibre G, thus emphasising or weakening features of the trajectory. For the car, the stretching operation can be calculated by dividing the control inputs a and ρ by c while multiplying the time they are applied by c.

Combining the results obtained so far, an efficient tool for altering a trajectory during a non-linear optimisation process can be built. Let $\Phi_u(x_0,t)$ be a trajectory given by a split control input

$$u = u_1 * \ldots * u_n$$

and starting point $x_0 \in X$. Changing a single u_i to $u_i^{c_i} =: v_i$ results in a Lie group operation $g_i \in G$ as of Eq. (3). Repeating this, one is able to alter several or even all segments of the path at once in order to get a new control input

$$v := v_1 * \ldots * v_n$$

where all v_i result from some $u_i^{c_i}$. In cases where $c_i = 1$, and thus the segment is unaltered, the corresponding g_i is set to the identity element $\mathbf{1} \in G$ without further

calculation. Assuming $\Phi_u(x_0,t)$ is given and the changed segments $\Phi_{v_i}(\Phi_u(x_0,I^+_{u_{i-1}}),t)$ and transformations g_i have been calculated, the new trajectory $\Phi_v(x_0,t)$ is computed efficiently using group operations only. Iteratively applying (4) yields

$$\Phi_{v_i}(\Phi_v(x_0,I^+_{v_{i-1}}),t) = g_{i-1}\dots g_1\Phi_{v_i}(\Phi_u(x_0,I^+_{u_{i-1}}),t)$$

and thus

$$\Phi_v(x_0,t)=\Phi_{v_1}(x_0,t)*g_1\Phi_{v_2}(\Phi_{u_1}(x_0,I^+_{u_1}),t)*\dots*g_{n-1}\dots g_1\Phi_{v_n}(\Phi_{u_{n-1}}(x_0,I^+_{u_{n-1}}),t)\,.$$

In particular, one can write the final state of the trajectory as

$$\Phi_v(x_0,I^+_v) = g_n\dots g_1\Phi_u(x_0,I^+_u)\,. \tag{6}$$

Clearly not much is saved in cases where all segments of the trajectory have been changed (i.e., all $c_i \neq 1$). However, if only a small fraction of the control input has been altered, then it is only necessary to reintegrate the fibre component of those altered segments. Thus the computational cost for calculating the new trajectory, or directly its end point, is linear in the length of the changed segments plus the cost of a few additional group operations.

Note that it is possible to perform the calculations of g_i and $\Phi_{v_i}(\Phi_u(x_0,I^+_{u_{i-1}}),t)$ separately for each segment, independent of what is done to other segments. Thus, for another transformation using some c'_i, all results where $c'_i = c_i$ can be reused. This speeds up things significantly for gradient calculations as will be detailed later and also allows for parallel computation.

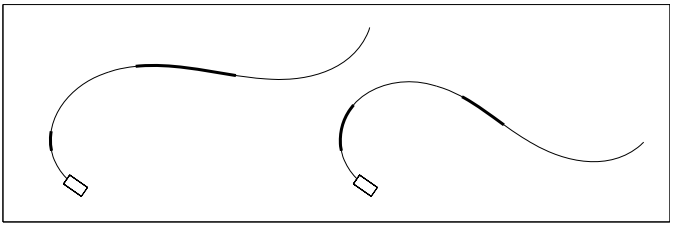

Fig. 5 A scaling operation has been applied to the *bold* segments of the *left hand* path to derive the *right hand* trajectory. Only the *bold* segments had to be reintegrated, the remainder is identical.

3 Path Correction Algorithm without Obstacles

For path correction, it will be assumed that an initial path $\Phi_u(x_0,t)$ as well as its control input u and initial state x_0 have been given. Furthermore, the path's final state $\Phi_u(x_0,I^+_u)$ does not coincide with the goal x_{goal}, but is somewhat in the vicinity of it. The objective is to alter the trajectory Φ_u in such a way that its final state matches x_{goal}. It will be assumed that the correction needs to be done in the fibre component only and that there are no obstacles present. This will be achieved in two

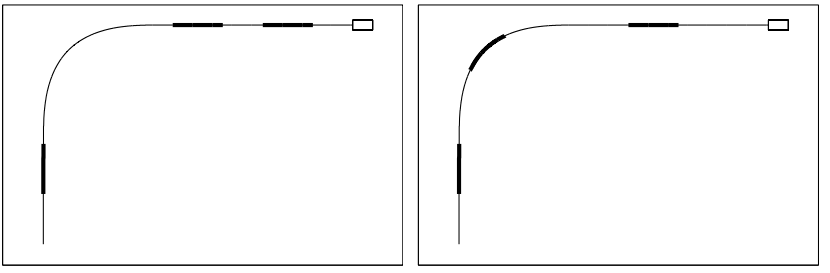

Fig. 6 The segments (*bold*) chosen for the trajectory on the *left* are unable to span the space well as they enable moving the final state horizontally and vertically, but prohibit alteration to the car's heading. The selection shown on the *right* is superior because changes in all directions including heading are possible.

steps: (1) a small and suitable set of path segments will be selected for stretching operations; (2) matching stretching factors c_i will be determined for said segments.

When selecting path segments, it is advantageous to select exactly as many segments as there are dimensions in the Lie group G. Using less segments results in too few degrees of freedom when altering the trajectory and thus failure to span a whole neighbourhood of the final state $\Phi_u(x_0, I^+)$. Using more segments than $\dim G$ leads to undesired behaviour as the solution is no longer unique. Furthermore, segments are chosen in such a way that the directions they move the trajectory's final state into have the potential to span the space well as illustrated in Fig. 6. This can be formalised by considering the derivatives

$$\frac{\partial \Phi_v(x_0, I_v^+)}{\partial c_i} = \frac{\partial g_i \Phi_u(x_0, I_u^+)}{\partial c_i}$$

evaluated at $c_i = 1$. As above, v represents the control input u with some segments u_i replaced by their scaled versions $u_i^{c_i}$ and, again, $g_i \in G$ denotes the resulting Lie group transformation. The quality of a selection of $\dim G$ segments can then be measured by analysing the condition of the resulting Jacobian

$$J = \frac{\partial \Phi_v(x_0, I_v^+)}{\partial (c_1, \ldots, c_{\dim G})} = \left(\frac{\partial g_1 \Phi_u(x_0, I_u^+)}{\partial c_1}, \ldots, \frac{\partial g_{\dim G} \Phi_u(x_0, I_u^+)}{\partial c_{\dim G}} \right) \tag{7}$$

evaluated at $c_i = 1$ for all i. The derivatives in the matrix on the right hand side are written as column vectors. If the matrix's condition is small, it has the potential to span the space well.

Since each column of J in Eq. (7) is independent of the remaining segments, the derivative has to be calculated only once. Thus, in practise, a solution is to select a larger set of non-overlapping segments and out of that then randomly draw selections of $\dim G$ elements for further testing. The selection with the smallest condition of the resulting Jacobian is then chosen. An exhaustive search for the optimal

selection is not necessary since it is sufficient to remove poor candidates. Taking a few random samples is often enough.

As optimisation algorithms typically work by minimising a target function [2], here the distance of the path's final state to the goal, it might seem tempting not to use the condition of the final state's Jacobian as presented here, but instead estimate the convergence rate of that target function directly via its second order approximation and the eigenvalues of the Hessian [16, 4]. In tests however this proved to perform poorly.

Once a set of segments is chosen, the values for the c_i need to be determined in order to actually improve the trajectory. Therefore a target function $f(c_1,\ldots,c_{\dim G})$ is defined as the distance between $\Phi_v(x_0,I_v^+)$ and x_{goal}. It is then minimised using a Conjugate Gradient method [16, 2]. Estimating the gradient of f at $(c_1,\ldots,c_{\dim G})$ is done by taking into account the function value $f(c_1,\ldots,c_{\dim G})$, as well as those resulting from going a small step into each direction, $f(c_1,\ldots,c_i+\varepsilon,\ldots,c_{\dim G})$, naively resulting in $\dim G+1$ integrations for each segment. However, since only two distinct values, c_i and $c_i+\varepsilon$, are used for each dimension of G, the calculated g_i can be recombined to obtain all function evaluations necessary. Thus, the cost to estimate a gradient is two integrations per segment plus some group operations.

Pseudocode for this algorithm is presented in Algorithm 1. It was used for the path depicted on the right hand side of Fig. 1 as well as the example presented in Fig. 7. Implementing this algorithm for more complex 3D cases, such as bevel tip needle steering, the state space X consists of eight dimensions: Insertion speed v, turning rate ω as well as six degrees of freedom representing position and orientation in three space. Thus the base space Z represents v and ω whereas G equals the group of Euler transformations $\mathrm{SE}(3)$. Following previous notation in this domain [15, 17], $\mathrm{SE}(3)$ is represented using homogeneous 4×4 matrices $g=\left(\begin{smallmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{smallmatrix}\right)$ where $R\in\mathrm{SO}(3)$ is a rotation matrix and $\mathbf{t}\in\mathbb{R}^3$ represents translation. The control

Algorithm 1. Path correction algorithm without obstacles

$u \leftarrow$ current plan
$M \leftarrow$ select set of at least $\dim G$ non overlapping segments of I_u
$C_{\min} \leftarrow \infty$
for $i = 1$ to $\min(maxSelections, \text{number of selections possible})$ **do**
  $S \leftarrow$ draw new selection of $\dim G$ random elements of M
  $J \leftarrow \left(\frac{\partial g_1 \Phi_u(x_0,I_u^+)}{\partial c_1},\ldots,\frac{\partial g_{\dim G}\Phi_u(x_0,I_u^+)}{\partial c_{\dim G}}\right)$ {calculated for the segments stored in S}
  if $\mathrm{cond}(J) < C_{\min}$ **then**
    $C_{\min} \leftarrow \mathrm{cond}(J)$
    $S_{\min} \leftarrow S$
  end if
end for
optimise $c_1,\ldots,c_{\dim G}$
  $v \leftarrow$ scale the segments of u stored in $S_{\min}$ according to values of $c_1,\ldots,c_{\dim G}$
until $\mathrm{dist}\left(x_{\text{goal}},\Phi_v(x_0,I_v^+)\right)$ **minimal**
return v

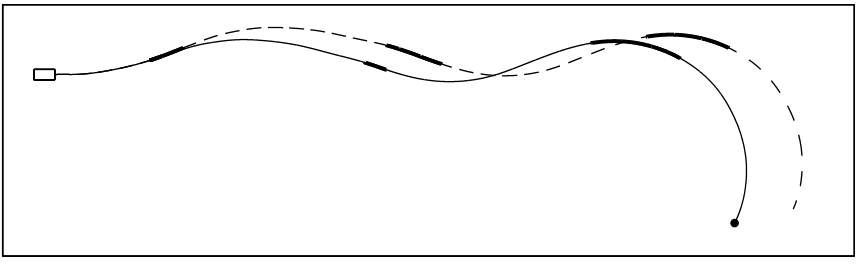

Fig. 7 A car (*rectangle*) is trying to reach the goal (*dot*). The *dashed* line shows the initial path that fails to reach the goal. By altering the segments depicted in *bold*, the *solid* path is created.

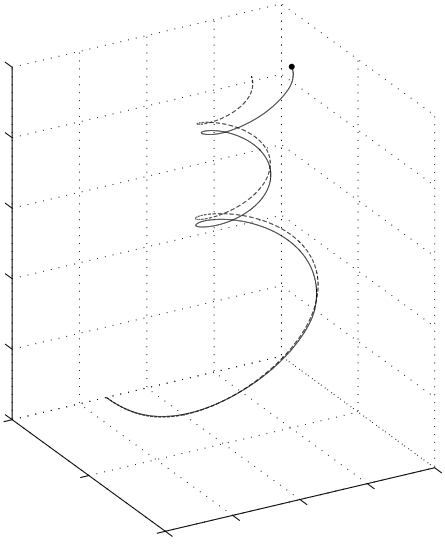

Fig. 8 Path correction for a needle steering case. The needle needs to reach the goal (*dot*), but the initial plan, depicted by the *dashed* line, misses it. The solid line is the correction made by the path correction algorithm.

space U has two dimensions, acceleration $a = \dot{v}$ and change in turning rate $\rho = \dot{\omega}$. The remaining equations of motion are given by

$$g^{-1}\dot{g} = \begin{pmatrix} 0 & -\omega & 0 & 0 \\ \omega & 0 & -\kappa v & 0 \\ 0 & \kappa v & 0 & v \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \mathfrak{se}(3)$$

where the constant κ is the curvature of the needle's trajectory and $\mathfrak{se}(3)$ is the Lie algebra of $\mathrm{SE}(3)$. An example for path correction using this system is presented in Fig. 8.

Since the algorithm works by enlarging or shrinking certain sections of the trajectory, it can not perform well in cases where the trajectory has too few features. Especially in cases where the path consists only of a straight line or a section of a circle, it is impossible to find segments that span the space well in a way discussed previously and illustrated in Fig. 6.

4 Path Correction Algorithm with Obstacles

When dealing with obstacles, an inversion of perspective is helpful. Up to now the initial path was considered to start at the robot's current state and the final state was then optimised. However, it is equally valid to anchor the initial path at the goal state x_{goal} and consider the robot to be at a state x_{curr} that does not coincide with the path's initial state $x_{\text{start}} = \Phi_u(x_{\text{start}}, I_u^-)$.

It will be assumed that the initial path is a collision free trajectory $\Phi_u(x_{\text{start}}, t)$ with $\Phi_u(x_{\text{start}}, I_u^+) = x_{\text{goal}}$. X_{obs} does not have to be empty, but it is assumed that there is

Algorithm 2. Path correction algorithm with obstacles

$u \leftarrow$ original plan
$v \leftarrow$ current plan
$x_{\text{curr}} \leftarrow$ system's current state
$S \leftarrow$ empty stack
repeat
  if $\Phi_v(x_{\text{curr}}, t)$ in collision **then**
    $v_1, v_2 \leftarrow v$ split at point of first collision
  else
    $v_1, v_2 \leftarrow v, \emptyset$
  end if
  $x_{\text{new}} \leftarrow$ find intermediate goal using u
  $v_1 \leftarrow$ run *algorithm without obstacles* (Alg. 1) for $\Phi_{v_1}(x_{\text{curr}}, t)$ in order to reach x_{new}
  if $\Phi_{v_1}(x_{\text{curr}}, t)$ collision free **then**
    $S.push \leftarrow v_1, x_{\text{curr}}$
    $v, x_{\text{curr}} \leftarrow v_2, \Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$
  else if $\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$ collision free **then**
    $v \leftarrow v_1 * v_2$
  else if S not empty **then**
    $v_0, x_{\text{curr}} \leftarrow S.pop$
    $v \leftarrow v_0 * v_1 * v_2$
  else
    return FAIL
  end if
until $v = \emptyset$
while S not empty **do**
  $v_0, x_{\text{curr}} \leftarrow S.pop$
  $v \leftarrow v_0 * v$
end while
return v

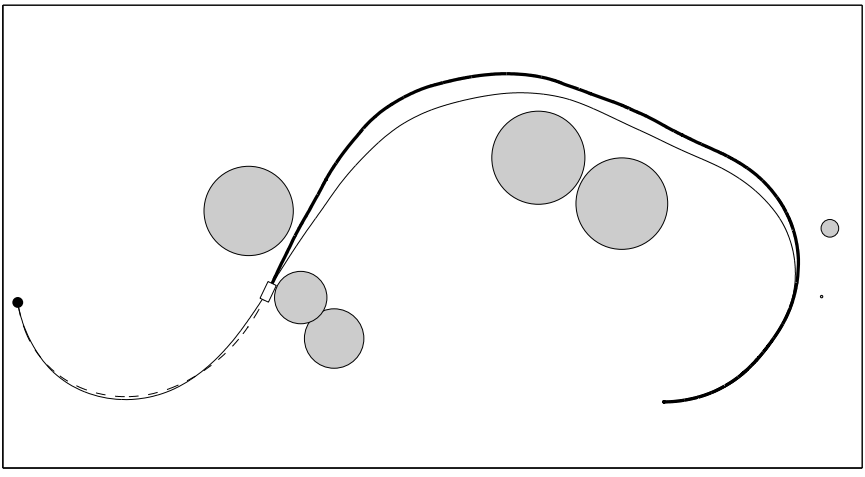

Fig. 9 A car tracking a path (*thin solid line*) through terrain with obstacles to reach the goal (*dot*). While driving, base space and control inputs (v, ω, a and ρ) are disturbed by random errors in form of a Wiener process. Errors on the base space are corrected by use of a feedforward controller with a saturation function, while resulting errors in position and heading are corrected repeatedly using the path correcting algorithm. The actual path taken is depicted by the *bold solid line* and the currently planned path that is to be followed is shown by the *dashed line*.

a certain amount of free space surrounding the trajectory most of the time that can be used for corrective actions. The robot is currently at x_{curr} and is following a path defined by v that is derived from, but that might not coincide with u. So $\Phi_v(x_{\text{curr}}, t)$ does not necessarily reach the goal x_{goal}. Allowing a discrepancy between u and v is advantageous when making multiple corrections; for example, when repeated online calculations are performed while executing a path under disturbances. In this setting, the initial path u will be kept constant during the whole process, while alterations are made to v only.

If no collisions occur in $\Phi_v(x_{\text{curr}}, t)$, alterations to v can be made directly using the *path correction algorithm without obstacles* (Algorithm 1). Otherwise, in case of collisions, the first point of collision is found as

$$t_{\text{col}} := \min\{t \in I_v \mid \Phi_v(x_{\text{curr}}, t) \in X_{\text{obs}}\}$$

and the path can be split such that $v = v_1 * v_2$ and $t_{\text{col}} = I_{v_1}^+$. To get around the obstacle, it is necessary to correct the path in two steps, first $\Phi_{v_1}(x_{\text{curr}}, t)$ using a new intermediate goal $x_{\text{new}} \in X_{\text{free}}$ and then v_2.

To define x_{new}, the colliding point $\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$ is pulled towards the corresponding point x_{orig} of $\Phi_u(x_{\text{start}}, t)$ where, due to scaling operations performed on v, the

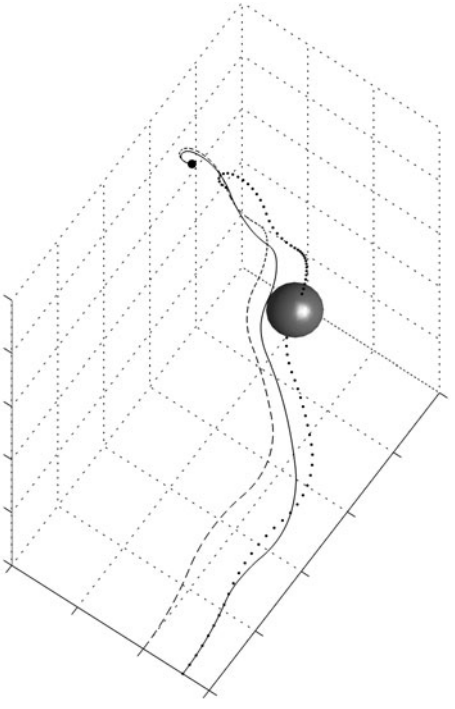

Fig. 10 A needle has to reach a goal (*dot*) and has an initial path depicted by the *dashed line*. Due to an offset in the initial position, the planned trajectory would result in the *dotted line*, colliding with an obstacle and missing the goal. The path is corrected and a valid path depicted by the *solid line* is created.

time t is not necessarily identical for u and v any more. This is done by parameterising a *straight line* connecting x_{orig} with $\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$ by $s : [0,1] \to X$ such that $s(0) = x_{\text{orig}}$ and $s(1) = \Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$.[2] Using s, the intermediate goal is defined as

$$x_{\text{new}} := s(\alpha^n) \,,$$

where n is the smallest $n \in \mathbb{N}$ such that $s(\alpha^n) \in X_{\text{free}}$ and $\alpha \in [0,1)$. The convergence rate α determines how fast the trajectory should be pulled back towards the original path and away from the obstacle. A large α results in staying closer to the obstacle

[2] Note that, as states already coincide on the base, the line s only has to be defined in G and is constant in Z. In most cases, it is intuitive what a suitable choice for a straight line within G should be and how it can be implemented easily. In less obvious cases, the exponential map $\exp : \mathfrak{g} \to G$ can be used, where $\mathfrak{g}$ is the Lie algebra of G. Let $d = \pi_G(x_{\text{orig}})^{-1} \pi_G(\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+))$ be the difference between the two states to connect. The points have to be close enough such that d lies within the identity component of G (i.e. the image of exp), as otherwise an easy connection is not possible. Then the line can be defined as $s(t) := \pi_Z(x_{\text{orig}}) \exp(t \exp^{-1}(d))$ for a suitable pre-image $\exp^{-1}(d)$.

and thus in a higher change that future corrections are necessary; a smaller α on the other hand pulls the trajectory more aggressively towards the original trajectory, preventing effective use of the available free space. The presented implementation uses $\alpha = 1/2$.

Using x_{new}, the *path correction algorithm without obstacles* is run over v_1 to get a new plan w_1 and thus $w = w_1 * v_2$. Note that it is acceptable if the new trajectory does not actually reach x_{new} itself as the main purpose of the operation is to pull the path away from the obstacle. What has to be considered tough are collisions of $\Phi_{w_1}(x_{\text{curr}}, t)$. If $\Phi_{w_1}(x_{\text{curr}}, t)$ is collision free, the process continues by recursively applying the *path correction algorithm with obstacles* on v_2. If $\Phi_{w_1}(x_{\text{curr}}, t)$ is in collision, two cases have to be considered. If the final state $\Phi_{w_1}(x_{\text{curr}}, I^+_{w_1})$ is in collision, the optimisation run did not get close enough to x_{new} because the path given by v_1 was too short or too featureless for the algorithm to perform well. In cases where v_1 is not the first part of the path due to a recursive call, w (and thus v_1) can be extended and a new attempt can be made. Otherwise the system is too close to an obstacle for suitable correction, and the algorithm is considered failed. If the final state $\Phi_{w_1}(x_{\text{curr}}, I^+_{w_1})$ is in X_{free}, the optimisation run was successful but the alteration introduced a new collision. Then a recursive call on the altered plan w and the starting point x_{curr} is necessary to get rid of the newly introduced collision. Pseudocode for the complete framework is given in Algorithm 2.

The algorithm can be used either offline within another planning framework (e.g. PRM, RRT) [4,11] or online while tracking a previously planned path. When applied in the latter approach, it is important that the initial trajectory u is kept unaltered during the whole process. Examples of how the algorithm performs are presented in Figs. 9 and 10.

5 Conclusion

An algorithmic framework was presented that allows for elegant and fast path correction while preserving the character of the initial trajectory, thus eliminating the need for expensive re-planning from scratch. The algorithm has been implemented for a kinematic car as well as for needle steering and simulations for the system's behaviour under disturbances have been performed. Future work will include implementing the system on experimental field systems currently under development.

Acknowledgements. This work is supported by the Rio Tinto Centre for Mine Automation and the ARC Centre of Excellence program funded by the Australian Research Council (ARC) and the New South Wales State Government.

References

1. Bloch, A.M.: Nonholonomic Mechanics and Control. In: Interdisciplinary Applied Mathematics, vol. 24. Springer, New York (2003)
2. Byrd, R.H., Nocedal, J., Waltz, R.A.: Knitro: An integrated package for nonlinear optimization. In: Large Scale Nonlinear Optimization, pp. 35–59. Springer, Heidelberg (2006)

3. Canny, J.F.: The complexity of robot motion planning. MIT Press, Cambridge (1988)
4. Cheng, P., Frazzoli, E., LaValle, S.M.: Improving the performance of sampling-based motion planning with symmetry-based gap reduction. IEEE Transactions on Robotics 24(2), 488–494 (2008)
5. Kanayama, Y., Kimura, Y., Miyazaki, F., Noguchi, T.: A stable tracking control method for an autonomous robot. In: Proceedings of the IEEE International Conference on Robotics and Automation, vol. 1, pp. 384–389 (1990)
6. Kelly, A., Nagy, B.: Reactive nonholonomic trajectory generation via parametric optimal control. The International Journal of Robotics Research 22(7-8), 583–601 (2003)
7. Kobilarov, M., Desbrun, M., Marsden, J.E., Sukhatme, G.S.: A discrete geometric optimal control framework for systems with symmetries. In: Proceedings of Robotics: Science and Systems, Atlanta, GA, USA (2007)
8. Koon, W.S., Marsden, J.E.: Optimal control for holonomic and nonholonomic mechanical systems with symmetry and Lagrangian reduction. SIAM Journal on Control and Optimization 35(3), 901–929 (1995)
9. Lamiraux, F., Bonnafous, D., Lefebvre, O.: Reactive path deformation for nonholonomic mobile robots. IEEE Transactions on Robotics 20(6), 967–977 (2004)
10. Latombe, J.C.: Robot Motion Planning. Kluwer, Boston (1991)
11. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
12. Murray, R.M., Li, Z., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation. CRC Press, Boca Raton (1994)
13. Ollero, A., Heredia, G.: Stability analysis of mobile robot path tracking. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 461–466 (1995)
14. Ostrowski, J.P.: Computing reduced equations for robotic systems with constraintsand symmetries. IEEE Transactions on Robotics and Automation 15(1), 111–123 (1999)
15. Park, W., Reed, K.B., Chirikjian, G.S.: Estimation of model parameters for steerable needles. In: IEEE International Conference on Robotics and Automation, pp. 3703–3708 (2010)
16. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University Pittsburgh (1994)
17. Webster III, R.J., Cowan, N.J., Chirikjian, G.S., Okamura, A.M.: Nonholonomic modeling of needle steering. In: Proc. 9th International Symposium on Experimental Robotics (2004)