# LOWER BOUNDS FOR GEOMETRICAL AND PHYSICAL PROBLEMS

JÜRGEN SELLEN *

**Abstract.** Motion planning involving arbitrarily many degrees of freedom is known to be PSPACE-hard. In this paper, we examine the complexity of generalized motion planning problems for planar mechanisms consisting of independently movable objects.

Our constructions constitute a general framework for reducing problems in information processing to motion planning, leading to easy proofs of known PSPACE-hardness results and to exponential lower bounds for geometrical problems related to motion planning. Especially, we show that the problem of deciding whether a given mechanism $A$ can always avoid a collision with another mechanism $B$ is EXPSPACE-hard.

New lower bounds are also obtained for the problem of planning under given physical side conditions. We consider the case that certain motions require forces, e.g., to subdue friction, and ask for motions that stay under a given energy limit. Within our framework, we show that such shortest path problems are EXPTIME-hard if we use number representations by mantissa and exponent, and even undecidable if we allow that some motions require no force or an infinite amount. The proof consists of a simulation of Turing-machines with infinite tape, and shows that the notion of Turing-computability can be interpreted in purely geometrical terms. The obtained geometrical model is capable of expressing a variety of physical planning problems.

**Key words.** motion planning, lower bounds, kinematic problems, dynamic problems, geometrical problems, collision avoidability problem.

**AMS subject classifications.** 68Q05, 03D10, 68Q25, 03D15, 68U05, 65Y25

**1. Introduction.** This paper is concerned with motion planning and related problems with and without physical (resp., dynamic) constraints. In our terminology we use the notion *kinematic scene* to describe the physical world which constitutes the input of a kinematic problem: a kinematic scene is a collection of rigid objects, each object being a semi-algebraic, connected subset of the Euclidean space $\mathbf{R}^d$ $(d = 2, 3)$, described by a boolean expression of polynomial inequalities of the form $p(x) > 0$ or $p(x) \geq 0$ with integer coefficients. Each object is specified either as an obstacle, having no freedom of motion, or as a movable object, the possible motions corresponding to the Euclidean group of rigid transformations. The complexity of a kinematic scene is the length of this description, the integers being coded in binary if not further specified.

With this definition, the classical motion planning problem can be stated as follows: given a kinematic scene, and initial and final placements of the objects, find a motion connecting these placements or decide that no such motion exists.

A standard formalism for specifying positions and motions is based on the notions *configuration space* and *free space*. With configuration space $CS$ we denote the topological space which is spanned by all the parameters that uniquely determine the positions of the movable objects. The subspace consisting of the physically legal configurations is called the free space $FS$ of the kinematic scene. We define the *legal configurations* to be those, in which the transformed objects, considered as point sets in $\mathbf{R}^d$, have a pairwise empty intersection. An allowed motion in a scene corresponds to a path in its free space, i.e., a continuous function $w : [0, 1] \rightarrow FS$. For motion planning, we are only interested in the connected component of the free space of a

* Fachbereich 14 Informatik, Universität des Saarlandes, Postfach 151150, 66041 Saarbrücken, Germany (`sellen@cs.uni-sb.de`). Supported by a Graduiertenkolleg-fellowship of Deutsche Forschungsgemeinschaft.

given scene $S$ that contains an initial configuration $C_0$, and we shall further denote this subspace with $FS_{C_0}(S)$ (we use 'connected' short for 'path-connected').

General solutions to the motion planning problem are based on the fact that the free space can be described as a semi-algebraic subset of a $k$-dimensional Euclidean space, $k$ being proportional to the total number of the degrees of freedom (i.e., summarized over all independently movable objects). By decomposing the free space into connected cells and interpreting these cells as nodes of an adjacency graph, motion planning can be reduced to a simple path search in a graph. Using Collins' algorithm of cylindrical algebraic cell decomposition, Schwartz and Sharir developed an algorithm whose running time is doubly exponential in the dimension $k$ [1, 2]. This was improved by Canny, who proposed a single exponential algorithm based on a retraction technique [3]. In [4], Canny shows that the motion planning problem can be solved in polynomial space.

These algorithms seem to be worst-case optimal, as there are kinematic scenes of complexity $n$ with $k$ movable objects, whose free space consists of $n^k$ connected components. It should be noted that $n^k$ also forms an upper bound for the complexity of the free space: combinatorial arguments show that the number of all faces comprising the boundary of the free space is at most that high [5].

In fact, the decision problem for motion planning involving arbitrarily many degrees of freedom can be shown to be PSPACE-hard for even very restricted subclasses of the problem:

Reif showed that motion planning is PSPACE-hard if we restrict our considerations to the motion of one object consisting of many linked parts in a 3D tunnel system [6]. Hopcroft, Joseph and Whitesides investigated planar linkages as a simple model for robots, i.e., systems of rods whose endpoints are connected by joints or attached to the plane. Using the classical result that any multivariate polynomial $p(x_1, \ldots, x_n)$ can be 'represented' by a planar linkage, they showed that it is possible to construct for any space-bounded Turing machine $T$ a planar linkage capable of simulating $T$ [7]. Searching for a moving system whose geometry is as simple as possible but for which motion planning is still intractable, Hopcroft, Schwartz and Sharir could prove motion planning for multiple independent rectangular boxes sliding inside a rectangular box to be PSPACE-hard [8]. The slightly easier Problem of moving many discs inside a planar polygonal world yet could only be proven to be strong NP-hard [9].

In this paper, we present a general framework for performing sound and transparent translations of known problems in information processing to motion planning, which will provide us with exponential lower bounds for some geometrical problems related to motion planning, as well as for planning problems in extended models involving physical side conditions.

This framework is based on the construction of mechanisms that are able to process information in the context of motion planning. Planar kinematic scenes consisting of polygonal objects with purely translational freedom of motion turned out to be powerful enough to provide us with most of the necessary tools: in section 2, we construct scenes for elementary tasks such as storage of information, the boolean operations or arithmetic on real numbers, and show how these 'basic mechanisms' can be combined to obtain 'devices' for more complicated tasks, e.g., binary counters.

The generality of this framework and the clearness of reductions follows from the close relationship to circuit design. Nevertheless, we do not build 'mechanical computers', as Zuse did in the first half of this century: to express information processing by motion planning, the existence of some motion has to correspond to some condition, but we do not consider static states or acting forces to generate these motions deterministically.

In section 3, we prove exponential lower bounds for some kinematic problems in the field of motion planning. These problems also possess interesting geometrical interpretations. We show that the problem of deciding, whether a quantified query for allowed configurations has a positive or negative answer, is EXPTIME-hard, and that the problem of deciding, if a given mechanism $A$ can always avoid a collision with another given mechanism $B$, is EXPSPACE-hard. The proof of the first result is based on the fact that a restricted arithmetic on real numbers can be simulated by motion planning.

In practical applications motions usually have to fulfill criteria other than simply being free of collisions. As such restrictions seem to add a new quality to the problem, the robotics community increasingly pays attention to problems involving physical side conditions. These problems belong to the general framework of non-holonomic motion planning. In [10], Fortune and Wilfong describe an algorithm for planning motions such that the corresponding path in free space is continuous differentiable and of bounded curvature (see also Jacobs and Canny [11]). In [12], motions are required to be time-optimal and to have bounded velocity and acceleration ('kinodynamic motion planning'). For both cases, only exponential, exact solutions are known, though the hardness is still open. However, already the problem of finding a shortest path between two points in a 3-dimensional polygonal environment is NP-hard [13]. A survey of non-holonomic motion planning can be found in [14].

In section 4, we examine the case that motions consume energy and that some motions may be more expensive in this sense than others (e.g., due to friction). We use the notion *dynamic scene* for a kinematic scene, in which a cost function is associated to the possible motions of each object depending on its contact situation. To have a sound definition of the input of our problems, we shall further assume that the cost functions are given implicitly by assigning integer friction coefficients to the sides of the objects. The interesting problem here is to find motions with minimal costs or to decide if there exists a motion that stays under a given cost limit. For these problems we prove exponential lower bounds. If we allow that some motions are forbidden or cost nothing, we even get undecidability (this was first discovered by Reif, see [6]). Undecidability is also obtained for the similar problem of deciding if there exists a specific motion in a kinematic scene during which not more than a given constant number of objects is moved simultaneously.

These results follow from a simulation of Turing machines with unbounded or exponentially bounded tape by motion planning in dynamic scenes. The inferences of this simulation technique are of interest from both a complexity-theoretic as well as a practical point of view. A translation of the dynamic scenes underlying the simulation into the formalism of configuration space provides us with a geometrical interpretation of Turing-computability. At the same time we obtain a geometrical model, in which a variety of physical problems can be expressed.

Although the lower bounds that we obtain are only valid for problem instances that involve arbitrarily many degrees of freedom, our work should cast some light on the common structure and the complexity of physical and geometrical problems, and should provide a framework for further research.

**2. Information processing by planar mechanisms.** In this section, we construct tools for processing information in planar kinematic scenes. We show how information can be coded, stored and transported, and explore which operations can

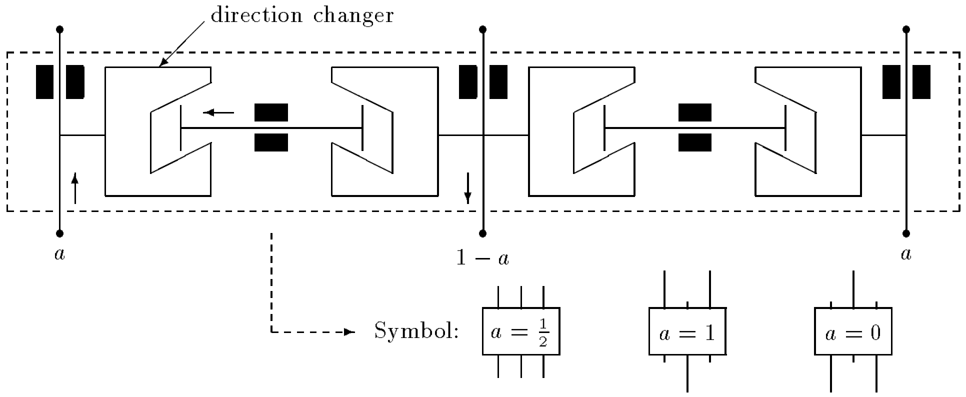



FIG. 1. *A variable cell: the position of the vertical rod at the left corresponds to the value of the variable a. By moving this rod up, the horizontal rod is pulled to the left, and the vertical rod in the middle, representing* $1-a$*, is pushed downwards.*

be realized.

Our constructions are illustrated by figure 1 – 9. The filled boxes correspond to obstacles. All other objects are movable. For clarity, rigid connections between rods are sometimes marked by fat points. We recall that the sides of our objects may be 'closed' or 'open' depending on if they are described by an inequality $p(x) > 0$ or $p(x) \geq 0$. Although this is not marked in the figures, we make use of both possibilities and leave it to the reader to complete the drafts.

**2.1. Coding of information.** To represent logical and numerical information, we code a number $a \in [0, 1]$ as the vertical position of a rod whose freedom of motion is restricted by obstacles to an interval of length 1 in the vertical direction. The logical values *true* and *false* correspond to the values 1 and 0.

For processing purposes, it is useful to have a second rod attached to the first representing the inverse value $1 - a$, and, to get symmetry, a third one again representing $a$. Thus we get a small mechanism, whose configuration corresponds to the value of a variable $a$, and which we shall denote as *variable cell*, symbolized by a box with label $a$ (see figure 1). The coupling of the three rods is realized by subassemblies transforming vertical movements to horizontal movements and vice versa. These subassemblies constitute a basic element of our constructions, and we shall further denote them as *direction changers*.

**2.2. Transportation of information.** In order to perform operations on variables, we need the possibility to transport the actual value of a variable from any place in the plane to any other place. Figure 2 shows how this can be achieved by a mechanism forcing equality between the values of two arbitrarily positioned variable cells $a$ and $a'$: the cells are connected by a sequence of rods and direction changers. Analogous to circuit design, we denote such sequences as *wires*, but we remark that they consist of movable rods carrying information by their position.

In order to get planar constructions, we need the possibility to cross wires. This can be achieved by exploiting that in planar scenes objects have two degrees of translational freedom, and thus can carry two independent information values at the same time. Figure 2 shows how two horizontal wires representing $a$ and $1-a$ can be crossed with two vertical wires representing $b$ and $1-b$. This is done by four boxes that carry

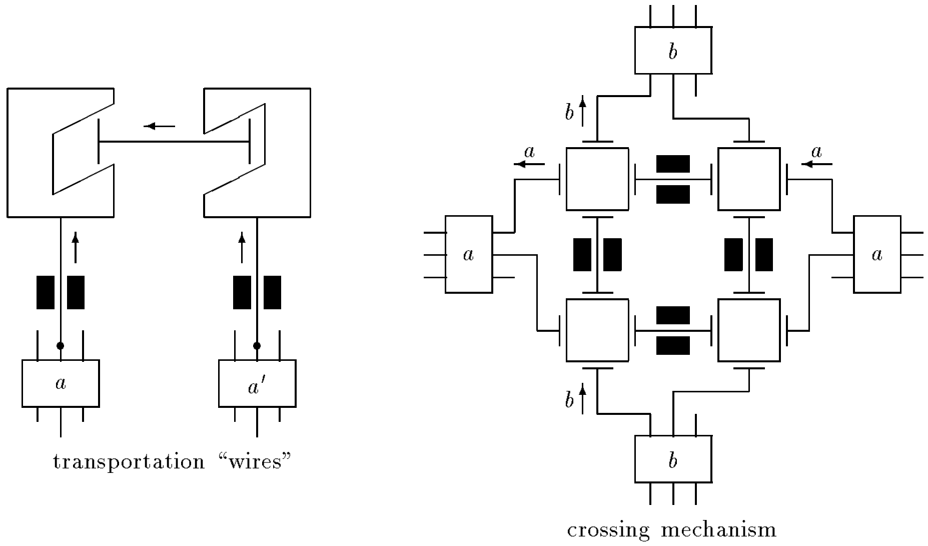



FIG. 2. *Wires and crossings: if in the crossing mechanism $b$ is increased, then the two left boxes are pulled upwards, and simultaneously the two right boxes are pushed downwards (analogous for the variable $a$, whose cells are rotated by $90°$).*

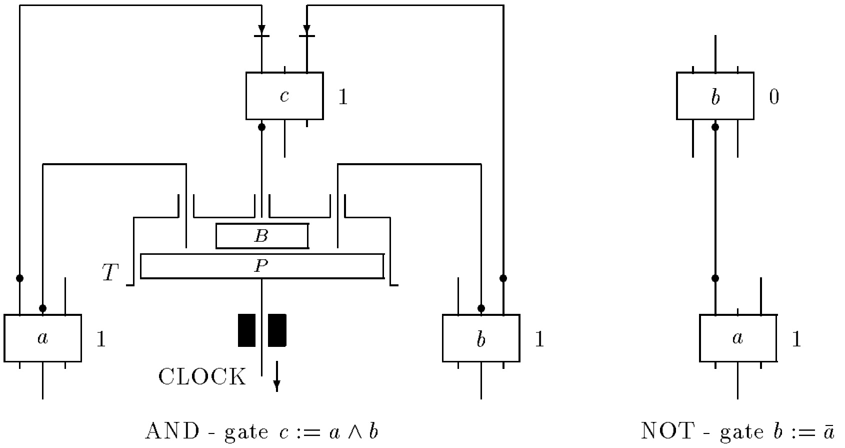



FIG. 3. *Mechanisms realizing the boolean operations NOT and AND: in the AND-gate, the shown position of $P$ corresponds to $CLOCK = 1$.*

both informations: the horizontal positions of the boxes are uniquely determined by the value of $a$, the vertical positions by the value of $b$. Changes in the value of variable $a$ (resp., $b$) only affect the horizontal (resp., vertical) positions of the boxes and thus leave the value of variable $b$ (resp., $a$) unchanged.

**2.3. Logical operations.** Planar kinematic scenes are capable of evaluating arbitrary boolean expressions in the sense that, given an expression $b(x_1, \ldots, x_n)$, there is a scene with variable cells for $x_1, \ldots, x_n$ and $x$, in which some specified motion forces the value of $x$ to be $b(x_1, \ldots, x_n)$ if $x_1, \ldots, x_n$ carry fixed logical values. We obtain this by constructing *mechanical gates*, shown in figure 3, for the boolean operations

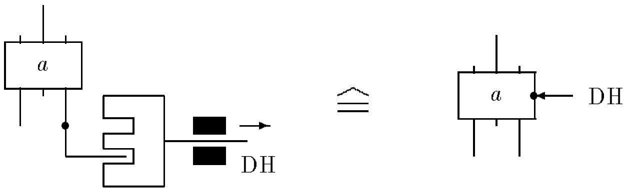



FIG. 4. *A binary memory cell: if the comb is in its leftmost position corresponding to $DH = 1$, the value of $a$ is fixed at either* 0 *or* 1.

NOT and AND.

The negation $a := \bar{b}$ can be achieved by connecting a rod coding $a$ with a rod coding $1-b$. To realize $c := a \wedge b$, we need a clock signal, i.e., a motion of an additional object forcing $c$ to be $a \wedge b$. The position of this object, a vertically movable 'plunger' $P$, is further denoted as CLOCK.

To check the correctness of the construction in figure 3, let us assume the plunger is in its upmost position corresponding to CLOCK = 1. The arrows over the rods coding $c$ are connected to the rods coding $a$ and $b$ and may press the rods coding $c$ downwards. If $a = 1$ and $b = 1$, then these arrows impose no restriction on the value of $c$, but the small box $B$ must be in a middle position forcing $c$ to be 1. If $a = 0$ or $b = 0$, then the box $B$ can be in the left or right side of the 'tub' $T$ and hence does not affect $c$, but at least one of the arrows forces $c$ to be 0.

We need the CLOCK, as the values of $a$ and $b$ can not be changed while the plunger stays in its upmost position. Only if the plunger is in its downmost position corresponding to CLOCK = 0, the variables $a$ and $b$ can be 'adjusted' arbitrarily.

**2.4. Storage and copying of information.** The constructions of the previous subsections show that combinational logic circuits can be simulated by motion planning problems in kinematic scenes. But to simulate sequential circuits, we still have to introduce a possibility to store information.

Figure 4 shows how a memory cell able to store the logical values 0 and 1 can be realized. Again we use a clock signal, further denoted as DATA HOLD (DH), to force the cell to keep its information. The value of DH corresponds to the position of a horizontally movable 'comb' which can hold a rod of a variable cell in a position coding 0 or 1.

It is possible to construct cells of polynomial complexity that are able to store a single exponential number of different values. Nevertheless, the upper bound on the combinatorial complexity of free space shows that storing a doubly exponential amount of information can not be achieved in the model of kinematic scenes.

We are now able to store information. To process information like in sequential circuits, we still need to realize loops or feed-backs: we need the possibility to copy information, i.e., to force equality between the values of two variable cells according to some clock signal.

A possible realization of such a copy operation is shown in figure 5: by moving two rods between the rods coding $a$ / $1-a$ and $b$ / $1-b$, the values of the variables $a$ and $b$ are coupled. The two rods are connected in the horizontal direction, such that their relative position is constant, but such that they are free to slide independently from each other in the vertical direction. Their absolute horizontal position is determined by the value of the clock, which we shall further denote as COPY signal.

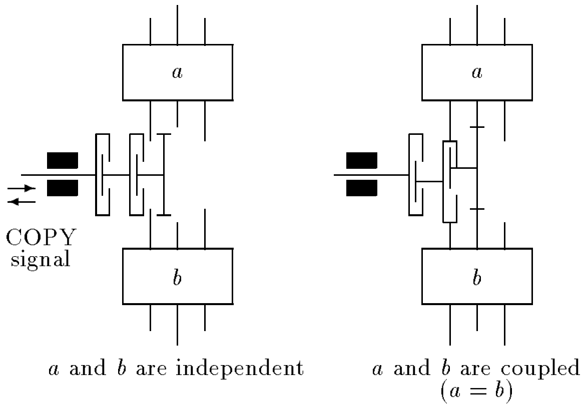



FIG. 5. *A copy mechanism: the COPY signal determines the horizontal position of three parts, which can be vertically shifted independently from each other.*

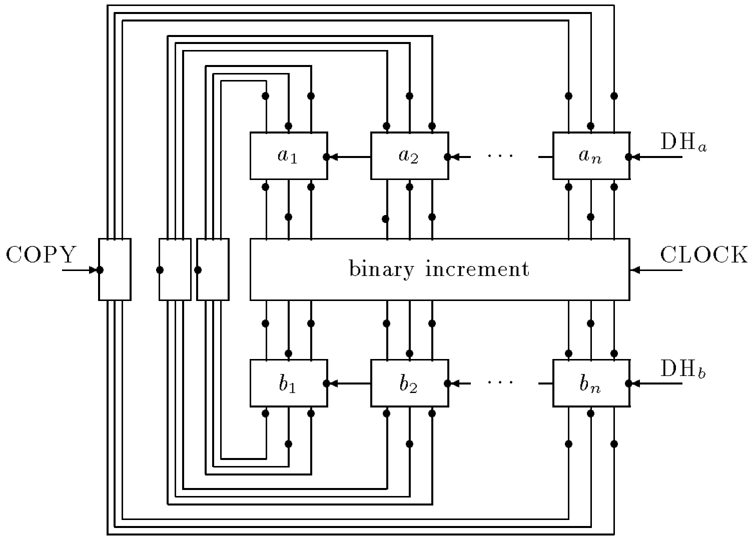
**2.5. Sequential circuits and control sequences.** The constructions of the previous subsections provide us with most of the tools that are necessary to simulate sequential circuits. The only problem that remains to be solved is to generate adequate sequences of clock signals.

We demonstrate the construction of a scene simulating a sequential circuit by the example of a binary counter: the mechanism is drafted in figure 6 and consists of two chains of binary memory cells for storing the numbers $a$ and $b$ according to the data hold signals $\mathrm{DH}_a$ and $\mathrm{DH}_b$, of a logic circuit forcing $b$ to be $a+1$ if the CLOCK signal is set to 1, and of a chain of copy mechanisms forcing $a$ and $b$ to be equal if the COPY signal is set to 1. Counting can then be forced by cyclically repeating the listed signal sequence.

Figure 7 shows how a cyclic repetition of a signal sequence can be generated. A box can be moved cyclically inside a 'ring' consisting of 8 rectangular 'chambers'. Each chamber can be forced to be empty ('closed') individually by pressing a plunger inside. To achieve that only two chambers are open at the same time, there are mechanisms connecting the plungers, which force that, if one chamber is open, all other chambers except the two neighboring ones are closed. Each chamber corresponds to a row in the signal table: the plunger for opening and closing chamber $i$ is connected with the signal wires of the signals that are registered in row $i$ as 1, such that these signals are set to 1 by opening the chamber. When moving circularly inside the ring, the box thus generates a cyclical repetition of the signal sequence (as two chambers may be open at the same time, the 'active' signals of the two corresponding rows may be forced to be 1 simultaneously, but this does not affect the simulation process for the chosen signal table).

We conclude that planar kinematic scenes are capable of simulating arbitrary sequential circuits in the context of motion planning. Especially, we can simulate PSPACE-bounded Turing machines by realizing tape and state as chains of binary memory cells. As the construction of the adequate kinematic scene can obviously be performed in polynomial time, we get a new proof of the PSPACE-hardness of the classical motion planning problem.

The reader should note that it is only possible to simulate symmetric Turing machines: as motions can always be reversed (in our construction, the 'direction' of the cyclical signal sequence can always change), the operations of the simulated




| | $DH_a$ | CLOCK | $DH_b$ | COPY |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 1 | 0 | 1 | 1 |
| 7 | 1 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 |

FIG. 6. *A binary counter: counting is forced by cyclically repeating the given signal sequence.*

machines can also be carried out in reverse order. This is captured by the notion of symmetric Turing machines and does not affect our complexity results [15].

**2.6. Arithmetic on real numbers.** Although we are able to code real numbers, we only introduced mechanisms for dealing with logical values. In this subsection, we construct kinematic scenes able to perform arithmetic operations, providing us with the possibility to describe problems in the theory of real numbers by motion planning problems.

In figure 8, mechanisms realizing the operations multiplication with constants ($b := c \cdot a$) and addition ($c := a + b$) are shown. Multiplication with constants can be achieved by using specially designed direction changers: the slope in the notch of the changer corresponds to the constant $c$. The mechanism for the addition of the two numbers $a$ and $b$ is based on our mechanism for realizing crossings. We again cross the wires carrying the values of $a$ / $1 - a$ and $b$ / $1 - b$ by four boxes. The box whose horizontal position corresponds to $a$ and whose vertical position corresponds to $b$ (i.e., the box $G$) contains a notch, in which a rod coding the result $c$ grips. This rod is pulled horizontally to the left both if the wire for $a$ is moved to the left (i.e., $a$ is increased) and if the wire for $b$ is moved upwards (i.e., $b$ is increased). The position of this rod thus corresponds to the value of the addition result. Obviously, also subtraction can be realized by this mechanism if we render the notch of $G$.

The free spaces of kinematic scenes that consist of polygonal objects with only

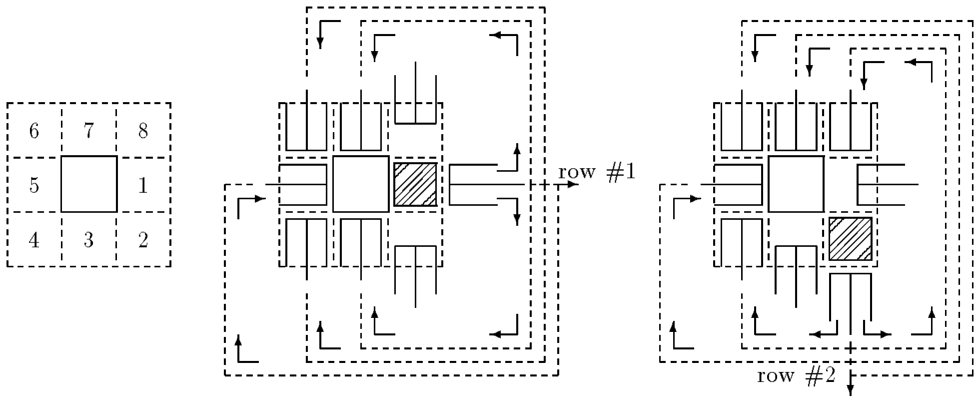



FIG. 7. *A mechanism for producing a cyclical repetition of a given signal sequence: the dashed lines represent the mechanisms connected to the plungers that open or close the chambers (only the mechanisms connected to the plungers* 1 *and* 2 *are indicated).*

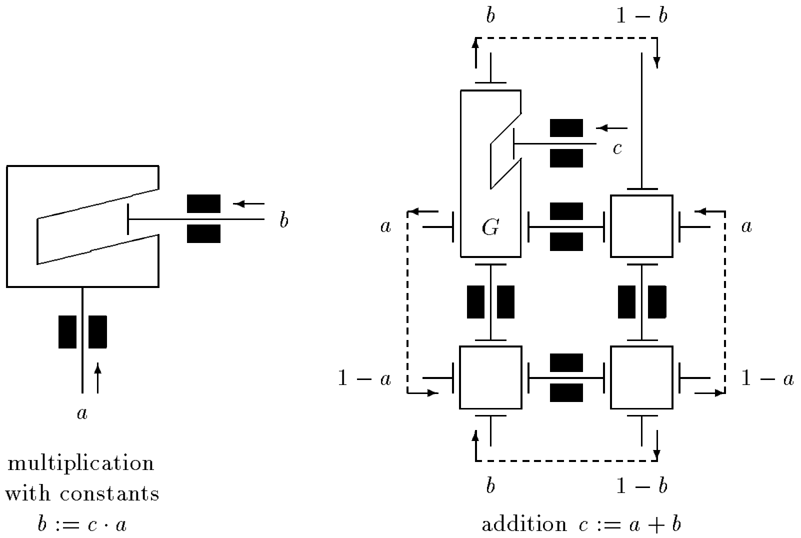



FIG. 8. *Mechanisms realizing the arithmetic operations multiplication with constants and addition.*

translational freedom of motion are multi-dimensional polyhedra. Thus it is not possible to realize 'non-linear' operations like multiplication of two real numbers by such scenes. But we remark without proof that multiplication can be realized by planar linkages and thus by scenes in 3-dimensional space that consist of objects with rotational degrees of freedom [7, 16].

To combine the capabilities to deal with logical and real values, e.g., to evaluate boolean expressions of polynomial inequalities, we must be able to perform comparisons with costants. Figure 9 shows a clocked mechanism that realizes the comparison "$a := (x < c)$?": if the cell $x$ carries a fixed real value from the domain $[0, 1]$, then the value of the cell $a$ is forced to be 1 if $x < c$ and to be 0 if $x \geq c$ by moving the object corresponding to the CLOCK signal into the position for CLOCK = 1. The adjustment of $a$ is forced by the boxes $L$ and $R$, whose shape is determined by the

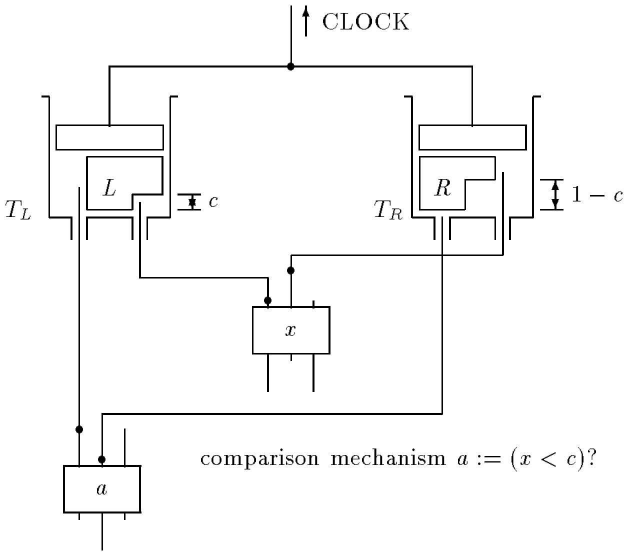



FIG. 9. *A mechanism realizing the comparison "$a := (x < c)$?".*

constant $c$: if CLOCK $= 1$ and $x \geq c$, then the box $L$ must be in the left area of the tub $T_L$, forcing $a$ to be 1. If $x < c$, then $1 - x > 1 - c$ and the box $R$ must be in the left area of the tub $T_R$, forcing $a$ to be 0. The CLOCK is again needed to have the possibility to readjust $x$.

**3. Exponential lower bounds for kinematic problems.** In this section, we present two decision problems within the model of kinematic scenes, for which we can prove exponential lower bounds using the constructions of the previous section.

The first problem concerns the avoidability of collisions between two given mechanisms $A$ and $B$: decide, if for any possible motion of $B$ (the 'attacker') there exists a simultaneous motion of $A$ (the 'defender') such that both mechanisms do not collide. A practical application is to decide, whether the motions of two given autonomous robot systems can always be coordinated such that one system can perform all its tasks in the presence of the other. We formulate the problem more geometrically as a *path lifting problem* in the configuration space of a kinematic scene:

PROBLEM 1 (COLLISION AVOIDABILITY PROBLEM). *Given a planar kinematic scene $S$ consisting of two disjoint subscenes $S_A$ and $S_B$ (i.e., the object set of $S$ is the disjoint union of the object sets of $S_A$ and $S_B$), and an initial configuration $C_0 \in FS(S)$, decide if for any path $w$ in the free space $FS(S_B)$ starting at $C_0|_{S_B}$ there exists a path $w'$ in $FS(S)$ starting at $C_0$ whose projection to $FS(S_B)$ is equal to $w$.*

We shall prove that this problem can not be solved in polynomial time or space by reducing to it the following problem $L_{\mathrm{rex}}$, which is known to be EXPSPACE-hard [17]:
given a regular expression $r$ with exponentiation (exponents are coded in binary), decide if the generated language $L(r)$ is equal to $X^*$, $X$ denoting the alphabet. (Example: $L((\mathrm{a}|\mathrm{b}^3)^2) = \{\mathrm{aa,abbb,bbba,bbbbbb}\}$.)

THEOREM 3.1. *The problem CAP is EXPSPACE-hard.*

*Proof.* ( $L_{\mathrm{rex}} \leq_{\mathrm{POLT}}$ CAP )

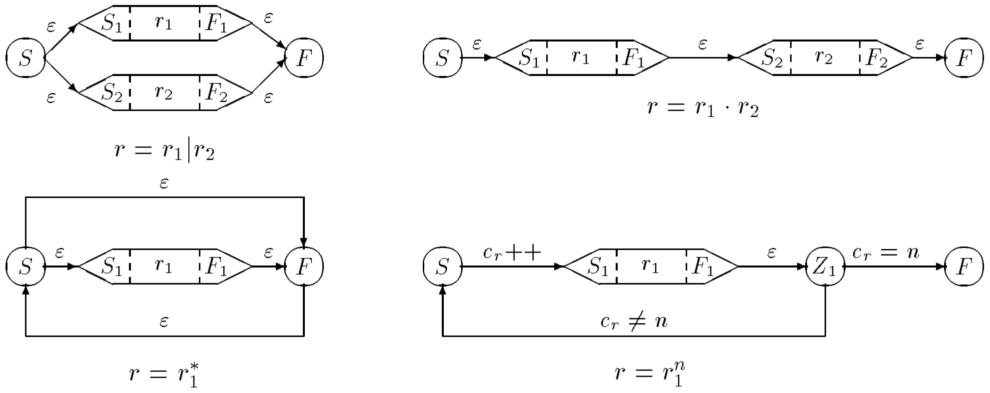



FIG. 10. *A recursive construction technique for transition graphs of automata with counters. For each occurrence of an exponent in the underlying regular expression, a new counter has to be introduced. Initially, all counters are set to* 0. *After a successfull test $c_r = n$, the counter $c_r$ has to be reset to* 0 *(this is omitted in the draft).*

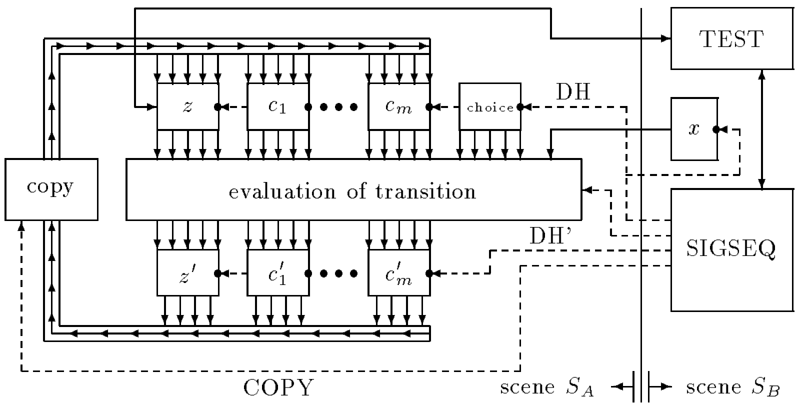



FIG. 11. *Assembly of a scene for the collision avoidability problem. Control signals are indicated by dashed lines.*

For any given regular expression $r$ with exponentiation, we construct in polynomial time a kinematic scene $S$ consisting of two subscenes $S_A$ and $S_B$ such that $S_A$ simulates an automaton accepting $L(r)$ and $S_B$ produces arbitrary input words for $S_A$. The structure of $S$ is drafted in figure 11.

**Step 1:** Construction of an automaton $\mathcal{A}$ accepting $L(r)$.

The number of states of a finite automaton accepting $L(r)$ may grow exponentially in the length of $r$, and even double exponentially if we require the automaton to be deterministic. But we can construct a non-deterministic automaton with polynomial size accepting $L(r)$ by providing the state set with a finite memory consisting of counters, one for each exponent: we get an automaton with state set $Q = Z \times C_1 \times \ldots \times C_m$, where $Z$ denotes a usual finite state set and $C_i = \{0, 1, \ldots, n_i\}$ the domain of the $i$-th counter. The only operations which are carried out on the counter values are increments "$c_i{+}{+}$", resets "$c_i := 0$", "$c_i := n_i$", and the comparisons "$c_i = n_i$?", "$c_i \neq n_i$?".

This automaton can be described by a transition graph, whose nodes correspond to the states of $Z$ and whose edges are labeled each with an input symbol (possibly $\varepsilon$), or with input $\varepsilon$ and one of the allowed counter actions. The transition graph can be obtained from a given regular expression $r$ by recursively applying the rules shown in figure 10. The resulting graph representing $\mathcal{A}$ has a description of size polynomial in the length of $r$.

Additionally, we require that $\mathcal{A}$ has the following properties:

(i) There exists a special final state $F_{\text{rev}}$, which has a transition to each state of $\mathcal{A}$ for each input symbol (including $\varepsilon$), but which can not be reached from any other state.

(ii) There are $\varepsilon$-transitions from each state of $\mathcal{A}$ to itself.

(iii) There exists a bound $c(r)$, polynomial in the size of $r$, such that the following holds: if there exists a sequence of $\varepsilon$-transitions from a state $Z_1$ to a state $Z_2$, then $Z_2$ can be reached from $Z_1$ by $\leq c(r)$ $\varepsilon$-transitions.

To obtain property (iii), we may render the construction of the automaton for $r = r_1^n$ in figure 10 as follows: if $\varepsilon \in L(r_1)$ (which can be decided in polynomial time), then we add an $\varepsilon$-transition from $S$ to $Z_1$, during which the counter $c_r$ is set to $n$. This allows to 'exit' an $\varepsilon$-loop by a constant (i.e., independent on $n$) number of transitions. (This property holds for the other construction elements by default.)

**Step 2:** Construction of a scene $S_A$ simulating $\mathcal{A}$.

By using additional inputs to code the possible non-deterministic choices of $\mathcal{A}$, it is possible to build a combinational logic circuit with $O(\log(|Z| \cdot n_1 \ldots n_m))$ inputs and outputs which 'evaluates' the transition diagram of $\mathcal{A}$, and whose size is polynomial in the size of $\mathcal{A}$. This circuit can be extended to a sequential logic circuit with binary registers for storing the state $(z, c_1, \ldots, c_m)$ of $\mathcal{A}$, and with external inputs for the next input symbol $x \in X \cup \{\varepsilon\}$ and the actual choice, such that $\mathcal{A}$ is simulated step by step according to the chosen inputs when the circuit is clocked.

Using the constructions of the previous section, we can transform this circuit to a kinematic scene $S_A$ which is able to simulate the working of $\mathcal{A}$ if a specific sequence of DH, COPY, DH' and CLOCK is provided.

Non-determinism can be captured by imposing no restriction on the value of the input coding the choice of $\mathcal{A}$: we provide this input with a chain of binary memory cells for storing its value, such that it can be adjusted arbitrarily when the adequate DH signal is set to 0.

**Step 3:** Construction of a scene $S_B$ producing arbitrary inputs for $S_A$.

The scene $S_B$ is assembled by three parts:

(i) A chain of binary memory cells for storing the value of the actual input $x$.

(ii) A mechanism SIGSEQ similar to the mechanism in figure 7, which produces a cyclical signal sequence that forces $S_A$ to simulate the working of $\mathcal{A}$.

(iii) A mechanism TEST which can be forced to test at any time if the actual state $z$ of $S_A$ is a final state, and which allows $S_A$ to return to its initial configuration after this test process.

In order to simulate the working of $\mathcal{A}$, we have to ensure that $S_A$ can make 'arbitrary' $\varepsilon$-transitions between two 'real' inputs generated by $S_B$. To achieve this, SIGSEQ produces a sequence of $c(r) + 1 + c(r)$ repetitions of the signal sequence necessary to simulate a single transition. For the first $c(r)$ transitions, the input $x$ is forced to code $\varepsilon$. For the next transition, the input $x$ may be arbitrary (i.e., the value of $x$ is not restricted). For the final $c(r)$ transitions, the input $x$ is again forced to code $\varepsilon$.

In order to test if $z$ is a final state, we simultaneously have to test if $z$ contains

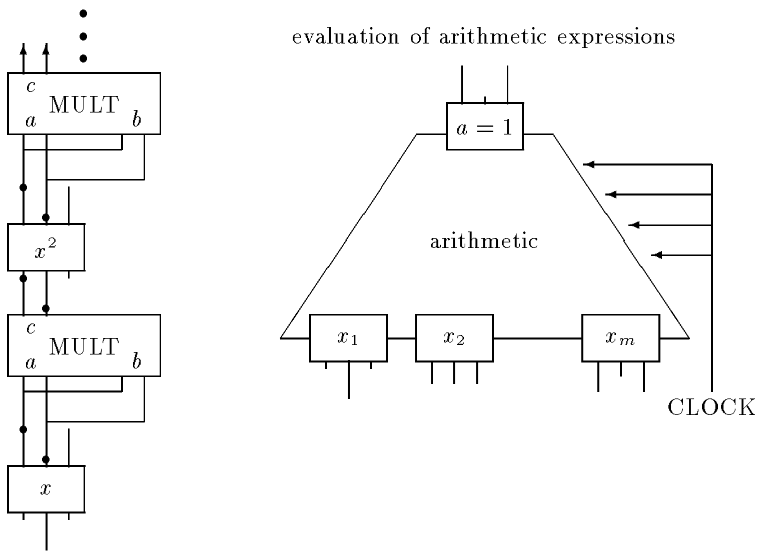



FIG. 12. *A scene for evaluating expressions in the theory of real numbers. The method of generating small numbers by squaring is drafted on the left.*

an actual state (i.e., the test can only be performed if we are at the end of a clock sequence with $DH = 1$). If $z$ is an actual state but not final, then there will be a collision during the test. In the last phase of the test, $S_A$ is allowed to return to its initial configuration, e.g. by 'decoupling' the DH signal from SIGSEQ with the help of a copy mechanism.

The scene $S_B$ can produce any input $w = x_1 \dots x_n$ of arbitrary length, and simultaneously can force $S_A$ by the generated signal sequence to simulate some computation of the non-deterministic automaton $\mathcal{A}$ on $w$. By the TEST mechanism, we can ask if the simulated computation accepts $w$. If $L(r) = X^*$ then, given any motion of $S_B$ containing a test, there always exists a motion of $S_A$ such that $S_A$ does not collide with $S_B$ during the test ($S_A$ has to simulate an accepting computation of $\mathcal{A}$ for the input word generated by $S_B$ before the test is performed).

On the other hand, if there is a word $w \notin L(r)$, and we force $S_A$ to simulate $\mathcal{A}$, then we get a collision by producing $w$ as input and then performing the test, no matter which non-deterministic computation is chosen by $S_A$.

We also have to take care about motions of $S_B$ which do not generate a valid computation of the automaton $\mathcal{A}$. The only way to produce such 'irregular' motions is by reversing the clock sequence. By construction of the automaton, we can however always get to the final state $F_{\mathrm{rev}}$ by a reversed transition, and then stay there. Thus $S_A$ can avoid a collision during a test after any irregular motion of $S_B$.

Concluding, $S_A$ can avoid a collision with $S_B$ for any given (i.e., predefined) motion of $S_B$ if and only if $L_{\mathrm{rex}} = X^*$. This finishes the proof. □

As the TEST mechanism resets the scene $S_A$ to an initial state after testing, this proof also shows that the following problem is EXPSPACE-hard: decide, if $S_A$ can avoid the collision with one recursively predefined infinite motion of $S_B$.

The second problem which we investigate in this section concerns quantified queries on allowed configurations, and can be stated as follows:

PROBLEM 2 (QUANTIFIED QUERY PROBLEM). *Given a kinematic scene $S$ with configuration space $CS(S) \subseteq \mathbf{R}^k$, an initial configuration $C_0 \in FS(S)$ and quantifiers $Q_i \in \{\exists, \forall\}$, $1 \leq i \leq k$, decide if the formula*

$$Q_1 x_1 \in [0,1] \ldots Q_k x_k \in [0,1]: \ (x_1, \ldots, x_k) \in FS_{C_0}(S) \tag{1}$$

*is true.*

This problem is similar to the EXPTIME-hard problem of deciding if a given formula in the theory of the real numbers $(\mathbf{R}, +, *, =, <, 0, 1)$ is true. But in the problem QQP, we are restricted to formulas that describe the free space of some kinematic scene. Further, we require that the given formula has to be satisfied by values belonging to a specific connected component of free space.

Our goal is to get a reduction of the decision problem in the theory of the real numbers to the problem QQP by constructing for any given formula a scene evaluating this formula. By the planar mechanisms introduced in the previous section, we can not realize the arithmetic operation of multiplication, but already the decision problem in the theory of the reals $(\mathbf{R}, +, =, <, 0, 1)$ without multiplication is known to be EXPTIME-hard [17]. We thus base our considerations on this theory, and denote the corresponding satisfiability problem of quantified formulas as TH($\mathbf{R}$,+). To model this problem in the context of kinematic scenes, we have to solve the following: in our constructions, we can only represent real numbers in the interval $[0,1]$, or at least in a bounded interval, as all objects have a bounded size. Here we can use that there exists a general decision algorithm for the considered satisfiability problem, which is based on the fact that it is sufficient to test the correctness of the formula for a finite subset of some domain $D = [-2^{2^{cn}}, 2^{2^{cn}}]$, $n$ corresponding to the length of the formula and $c$ denoting a fixed constant.

Thus we can confine our investigations to a bounded domain, but there still remains the problem how to code this domain: using mechanisms of size $2^{2^{cn}}$ is not possible, as this would lead to scenes with descriptions of exponential size. We shall code the domain $D$ by compressing it to $[-1/2, 1/2]$, and then shift this interval to $[0,1]$. But this leads to the problem of how to represent the numbers 0 and 1, i.e. the allowed coefficients in the input formulas, which then correspond to $\hat{0} = 1/2$ and $\hat{1} = 1/2 + 1/2 \cdot 1/2^{2^{cn}}$. To get the small offset $1/2^{2^{cn}}$, or more generally to get a coding of the large domain $D$, we have to use multiplications and thus planar linkages (i.e., arbitrary scenes with rotational degrees of freedom). We build a mechanism which squares a given number $x \in [0,1]$ $cn$ times, specified in a configuration corresponding to $x = 1$ (i.e., with a polynomial description). Then we have access to the number $1/2^{2^{cn}}$ in the context of a motion planning problem by requiring $x$ to move to a position coding $1/2$ (see figure 12). By multiplying the result with $1/2$ and then adding $1/2$, we can construct a mechanism which generates the number $\hat{1} = 1/2 + 1/2 \cdot 1/2^{2^{cn}}$ if some clock signal is set to 1.

Because of the necessity of multiplications to code the domain $D$, we can prove the following theorem only for arbitrary kinematic scenes with rotational degrees of freedom.

THEOREM 3.2. *The problem QQP is EXPTIME-hard.*

*Proof.* ( TH($\mathbf{R}$,+) $\leq_{\mathrm{POLT}}$ QQP )
We have to construct for any given formula $F'$ of length $n$ in the theory $(\mathbf{R}, +, =, <, 0, 1)$ a kinematic scene $S$ with an initial configuration $C_0$ such that the formula (1) holds if and only if the given formula is true.

W.l.o.g. we can assume that the formula $F'$ is given in prenex normal form

$F' =$ "$Q'_1 x'_1 \ldots Q'_m x'_m : E'(x'_1, \ldots, x'_m)$", $E'$ denoting a boolean expression with linear inequalities as atoms.

By the tools presented in the previous section, it is easy to construct a scene $S$ with $m$ variable cells coding $\hat{x'_1} = (1/2^{1+2^{cn}})x'_1 + 1/2, \ldots, \hat{x'_m} = (1/2^{1+2^{cn}})x'_m + 1/2$, which evaluates $E'$ into a logical variable $a$ if a CLOCK signal is set to 1 (see figure 12).

The atomic inequalities in $E'$ can be written as additions and subtractions of variables and the number 1, followed by a comparison with 0. Thus we need the value $\hat{1}$ only as a constant which we can add to or substract from a variable. For each occurrence of 1 in $E'$, the scene $S$ contains a mechanism which generates $\hat{1}$ as adequate input if the CLOCK signal is set to 1. As we are interested in values satisfying the expression $E'$, we fix the evaluation result $a$ in the position coding the logical value 1.

The scene $S$ and some initial configuration $C_0$ with CLOCK = 0 can be constructed in polynomial time. We assume that the parameters $x_1, \ldots, x_k$ spanning configuration space begin with the positions of the rods used to code $\hat{x'_1}, \ldots, \hat{x'_m}$, and that the last parameter $x_k$ corresponds to the value of the CLOCK. The quantifiers $Q_1, \ldots, Q_m$ in the formula (1) are defined by $Q_1 = Q'_1$ , ... , $Q_m = Q'_m$ , $Q_{m+1} = \exists$ , ... , $Q_{k-1} = \exists$ , $Q_k = \forall$.

We define $E(x_1, \ldots, x_k)$ to be the formula "$Q_{m+1}x_{m+1} \ldots Q_k x_k : (x_1, \ldots, x_k) \in FS_{C_0}(S)$". Then (1) can be written as "$Q_1 x_1 \ldots Q_m x_m : E(x_1, \ldots, x_k)$".

Assume the expression $E'$ is true for specific values of $x'_1, \ldots, x'_m$. As $E'$ can be evaluated to 1, there exists a motion in $S$ starting at the initial configuration, during which first the objects coding $\hat{x_1}, \ldots, \hat{x_m}$ are set to the corresponding values, and then all other objects are positioned such that the CLOCK can switch from 0 to 1. Thus also the expression $E$ is true for the corresponding values (i.e., the same values respective to the linear transformation of the domain).

If, on the other hand, $E$ is true for specific values of $x_1, \ldots, x_m$, then there exists a motion in $S$ during which $E'$ is evaluated to 1. Thus also $E'$ is true for the corresponding values.

As all values in $[0, 1]$ can be adjusted in $S$ respective to the given initial configuration, we get that (1) is true if and only if $F'$ is true (recall that the satisfiability of $F'$ is determined by its satisfiability in $D$). This finishes the reduction. □

We remark that the problems CAP and QQP are both problems concerning classes of motions or configurations. Exponential lower bounds for problems concerning the existence of single motions seem to be impossible to prove within the framework of kinematic scenes: coding computations of exponentially time-bounded Turing machines by motions requires the possibility to distinguish between double exponentially many configurations. This however is impossible in a free space of roughly single exponential combinatorial complexity.

In the next section, we investigate extended models of scenes involving physical constraints, in which we have the possibility to process more information than in kinematic scenes, and return to the classical existence problem of motions.

## 4. Motion planning under physical constraints.

**4.1. Undecidability results.** We first investigate motion planning in the model of dynamic scenes for the idealized case that motions may be forbidden (i.e., have infinite costs) if specific sides of two objects are sliding along each other tangentially during this motion. According to the introduced model of dynamic scenes, we assume that this effect occurs due to friction, and code it in the input by marking adequate

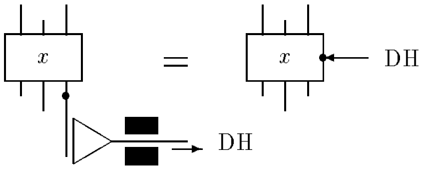



FIG. 13. *A memory cell in the dynamic model.*

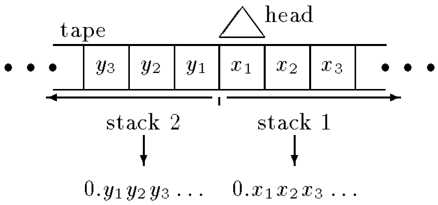



FIG. 14. *Coding of the tape of a Turing machine by real numbers.*

sides of the objects with the symbol '$\infty$'. We consider tangential movements of two objects along touching sides as illegal if both concerned sides are marked with '$\infty$'.

The essential difference between this model and the model of kinematic scenes is that we now have the possibility to fix objects in arbitrarily many positions and thus to store an infinite amount of information in the position of one object. Figure 13 shows a memory cell able to store any real number $x \in [0, 1]$ by setting the value of a data hold signal DH to 1. The storage state is established by 'pressing' a side of a triangle against the rod of a variable cell $x$, the position of this triangle corresponding to the value of DH (the touching sides of the triangle and of the rod are marked with '$\infty$').

As the considered constraints only concern the existence of motions, the allowed configurations of a dynamic scene are the same as the allowed configurations of the underlying kinematic scene. Thus motion planning in this model can not be expressed as a simple path search in some semi-algebraic subspace of the Euclidean space. In fact, the dynamic motion planning problem as stated below is undecidable, as it opens the possibility to process an infinite amount of information and thus to simulate arbitrary Turing machines with infinite tape.

PROBLEM 3 (DYNAMIC MOTION PLANNING PROBLEM). *Given a dynamic scene $S$ with forbidden movements, and initial and final configurations $C_i$ and $C_f$, decide if there exists an allowed motion connecting these configurations.*

THEOREM 4.1. *The problem DMPP is undecidable.*

*Proof.* (Simulation of arbitrary Turing machines)
We show that we can construct for any given, deterministic Turing machine $M$ a dynamic scene $S$, and for any input $w$ of $M$ an initial configuration $C_i \in FS(S)$, such that there exists a motion in $S$ connecting $C_i$ with some final configuration $C_f$ if and only if $M$ accepts $w$.

We assume that $M$ has only one tape and an alphabet with $p$ symbols, including the blank. Obviously, $M$ can be simulated by a push-down-automaton with two stacks, one corresponding to the part of the tape left to the actual position of the head, the other corresponding to the part below and right to the head. The actions of $M$, especially movements of the head, can be expressed by stack operations. We interpret

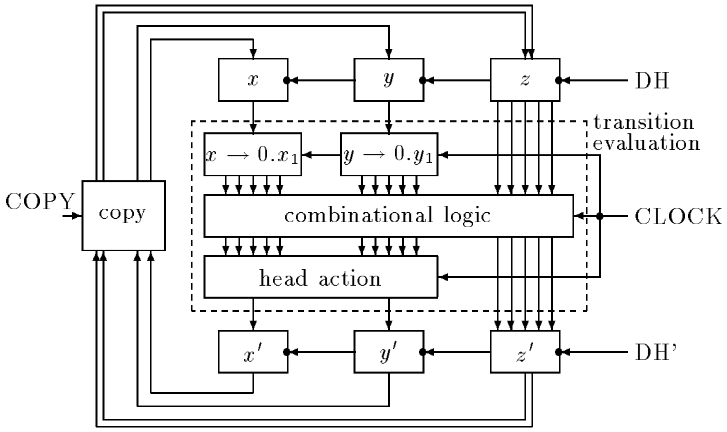



FIG. 15. *Assembly of a scene for simulating arbitrary Turing machines. The cells labeled with $x, y, x'$ and $y'$ denote memory cells for real values, $z$ and $z'$ can be stored in chains of binary memory cells.*

the contents of these stacks as $p$-adic numbers $x = 0.x_1x_2x_3\ldots$ and $y = 0.y_1y_2y_3\ldots$ in $[0, 1]$, as shown in figure 14, stored in adequate memory cells. The actions of the Turing machine $M$ can now be realized by the logical and arithmetic operations which we introduced in section 2:

The transition table of $M$, with binary coded inputs and outputs, can be 'evaluated' by a combinational logic circuit and thus by a kinematic scene. As the actual state can be stored in a chain of binary memory cells, it only remains the problem to get a binary representation of the actual input symbol $x_1$ from $x$, to write a new symbol $x_1'$ given by a binary representation to $x$, and then to simulate the movement of the head. In order to get access to $x_1$, we can use our comparison mechanism to test simultaneously for each integer $0 \leq i \leq p-1$ if $\frac{i}{p} \leq x < \frac{i+1}{p}$, and calculate a binary representation of $x_1$ from the results. In the opposite direction, constructing the real number $0.x_1'$ from a binary representation of $x_1'$ can trivially be attained by additions and multiplications with constants. Substituting $x_1$ by $x_1'$ is done by adding $0.x_1' - 0.x_1$ to $x$. To realize a movement of the head to the right, we have to 'pop' $x_1'$ from $x$ and to 'push' it to $y$. This translates to $x := (x - 0.x_1')\cdot p$ and $y := (\frac{1}{p}\cdot y) + 0.x_1'$. Movements to the left also require to analyze $y$ by comparisons, in order to calculate a logical representation of $y_1$.

The simulation principle is shown in figure 15: we need 3 memory elements $x, y, z$ for the actual configuration of $M$, and 3 elements $x', y', z'$ for storing the transition result until it is copied to the actual configuration. The whole mechanism is controlled by the signals DH, COPY, DH' and CLOCK, which are generated analogous to the case of the binary counter (see figure 6,7).

We remark that motions in the constructed scene need not correspond to regular computations of $M$, as transitions of $M$ can also be carried out in reverse order. Nevertheless there exists a motion from a given initial configuration corresponding to an input $w$ to some final configuration corresponding to the (w.l.o.g. unique) accepting state of $M$ if and only if $M$ accepts $w$. □

The essential fact that makes this proof work is that we can fix objects in arbitrary positions, achieved by restricting the set of allowed motions. Dynamic scenes are a powerful tool for defining such restrictions, and there are other practical problems in

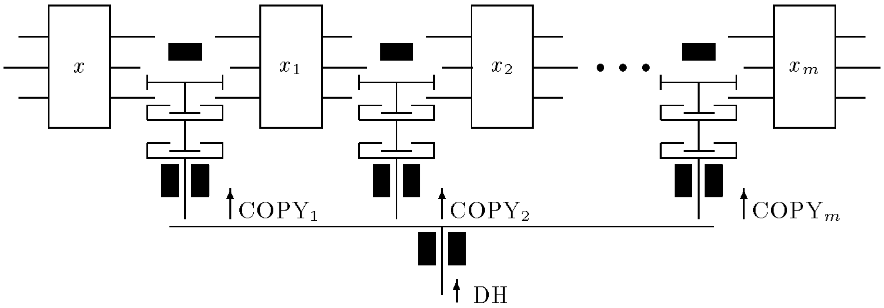



FIG. 16. *A memory cell for storing the real value $x$ in the context of passive motion planning.*

which we have weaker possibilities of defining restrictions on motions, but in which we nevertheless get a similar situation. We introduce one such problem, which we call *passive motion planning*, and show that this problem is also undecidable.

In the standard formulation of motion planning, we assume that the objects can move independently on their own, as by a magic hand. But in practical applications, the objects often have to be moved by an external robot system able to move only a bounded number of objects at the same time. We thus only allow motions during which not more than a given number $m$ of objects are moved simultaneously. Further idealized by considering only translational movements in one direction and translated into the formalism of configuration space, we define passive motion planning as follows:

PROBLEM 4 (PASSIVE MOTION PLANNING PROBLEM). *Given a kinematic scene $S$, initial and final configurations $C_i$ and $C_f$, and an integer $m$, decide if there exists a path in $FS(S) \subseteq \mathbf{R}^k$ connecting $C_i$ with $C_f$, which consists of a finite number of straight line segments, each segment being orthogonal to at least $k - m$ of the $k$ coordinate axes of $\mathbf{R}^k$.*

THEOREM 4.2. *The problem PMPP is undecidable.*

*Proof.* Passive motion planning allows us to fix the contents of a variable cell according to a data hold signal by connecting the cell to $m$ further variable cells via $m$ copy mechanisms (see figure 16). As establishing the connection must be able by moving at most $m$ objects at the same time, we force all copy mechanisms to get to their connecting position by setting a data hold signal DH to 1, but we do not connect them to this signal rigidly.

Given an arbitrary Turing machine $M$ and an input $w$, we first construct a dynamic scene $S'$ able to simulate $M$, as in the proof of theorem 4.1. Let $m'$ be the number of movable objects in $S'$. We substitute the memory cells for real-valued variables by the mechanisms described above, with $m = m'$. The input for PMPP consists of the resulting kinematic scene $S$, an initial configuration $C_i$ corresponding to $w$, a final configuration $C_f$ corresponding to the accepting state of $M$, and the bound $m = m'$.

As any two configurations in $FS_{C_i}(S')$ which can be connected by some finite motion can also be connected by a motion consisting of a finite sequence of translations of the $m$ movable objects, the restrictions which are imposed on allowed motions by passive motion planning only affect the memory cells in $S$. These are forced to store information by setting the data hold signals to 1. Thus the allowed motions in $S$ and $S'$ are essentially the same. $\square$

In this subsection, we have investigated motion planning problems involving constraints by which motions could be forbidden. In the next subsection, we weaken this restriction by assigning costs to motions, such that we only have the possibility to make some motions more expensive than others.

**4.2. Exponential lower bounds.** In this subsection, we investigate arbitrary dynamic scenes in which finite costs are implicitly defined for motions by assigning integers, denoted as friction coefficients, to the sides of the objects. We do not need to specify a sound physical model of how to define friction (in such a model we would also have to deal with forces or masses). We only shall need the fact that some motions can be made expensive in comparison to others. The goal in dynamic motion planning is to find a motion which stays under a given cost limit.

If we allow that some motions cost nothing (i.e., consume no energy), we have a situation which is symmetric to the idealization that some motions have infinite costs. To translate a given problem involving infinite costs to an equivalent problem involving costs 0, we only have to assign to all motions with infinite cost (i.e., forbidden motions) some finite cost and to all other motions cost 0. The existence of an allowed motion then corresponds to the existence of a motion with cost 0, making it possible to carry over the undecidability results of the previous subsection to this problem.

We thus require that during any motion some sides are tangentially sliding along each other. For the case of planar scenes, this can be achieved by assuming that all objects are sliding on some ground floor. For the sake of completeness, we define the cost of a path (resp., motion) as follows: if two sides with friction coefficients $a$ and $b$ slide along each other during parts of the motion, and if $l$ denotes the Euclidean length of the relative movement of the two involved objects, then this pair of sides causes the cost $a \cdot b \cdot l$. The cost of the entire path results from summarizing these costs.

PROBLEM 5 (CHEAPEST MOTION PLANNING PROBLEM). *Given a dynamic scene $S$ with friction coefficients $\in \mathbf{N}$ such that any motion of positive length has some cost $> 0$, initial and final configurations $C_i$ and $C_f$, and a cost limit $m \in \mathbf{N}$, decide if there exists a motion with cost $< m$ connecting these configurations.*

THEOREM 4.3. *The problem CMPP is EXPTIME-hard, if the friction coefficients are coded by mantissa and exponent.*

*Proof.* Based on the construction introduced in the proof of theorem 4.1, we build dynamic scenes able to simulate exponentially time-bounded Turing machines in the context of cheapest motion planning. As we can not fix objects like in this proof, we lose the capability of storing arbitrary information. But we still keep the possibility to code a finite amount of information, which depends not only on the underlying kinematic scene, but also on the friction coefficients and the cost limit $m$.

Assume we are given a $c2^n$ time-bounded, deterministic Turing machine $M$ and an input string $w$ of length $n$. Let $S'$ be the dynamic scene constructed for $M$ in the proof of theorem 4.1, and let $C_i$ and $C_f$ be the initial and final configurations corresponding to the initial state of $M$ with input $w$ and the accepting state of $M$.

We first assign the friction coefficient 1 to all object sides of $S'$ with the exception of those sides that are marked with '$\infty$'. Then, if there exists an allowed motion in $S'$ connecting $C_i$ and $C_f$, there also exists an adequate motion in the resulting scene with cost $< c'c2^n$, $c'$ being a constant only depending on $S'$ (or $M$, respectively). We set the cost limit to $m = c'c2^n$ and define $S$ by substituting the '$\infty$' marks by the friction coefficient $c_f = mp^{2c2^n}$, $p$ denoting the size of the alphabet.

The cost limit has been chosen such that, if $M$ accepts $w$, there exists a motion in $S$ connecting $C_i$ and $C_f$ with cost $< m$. It remains to show that there exists no such motion if $M$ rejects $w$. In this case, there exists no allowed motion in $S'$ connecting $C_i$ with $C_f$ though the underlying kinematic scenes are the same. Thus a connecting motion in $S$ must include some movements which are forbidden in $S'$, i.e., movements of the rods of the memory cells coding the tape while their data hold signal is 1. But these movements are expensive in $S$, and in order to stay under the given limit $m$, such a movement can not change the position of a rod by a distance of more than $m/c_f^2 < 1/p^{2c2^n}$. Hence, the only 'actions', which can be 'performed' by a connecting motion in $S$ in addition to what is possible by allowed motions in $S'$, correspond to changes of the tape at positions, which are more than $2c2^n$ tape cells away from the actual head position. These positions can never be reached during a computation of $M$ started with input $w$ ($M$ uses at most $c2^n$ cells of the tape !). As the possible changes in the contents of a memory cell during a storage state are so small that they do not affect the simulation process, we can translate a motion in $S$ connecting $C_i$ and $C_f$ with cost $< m$ to an accepting computation of $M$ (or, more strict, of the symmetric Turing machine corresponding to $M$), a contradiction to the presupposition that $M$ rejects $w$.

The size of the underlying kinematic scene does not depend on $n$. $C_i$, $C_f$ and the cost limit $m$ have a binary description of size polynomial in $n$, and the friction coefficients (especially $c_f$) can be coded in a length polynomial in $n$ by a number representation using mantissa and exponent. The input to CMPP can be computed for given $M$ and $w$ in time polynomial in $n$. This completes the proof. □

In the proofs of theorems 4.1 – 4.3, we used planar scenes consisting of polygonal objects with purely translational freedom of motion. Thus the considered motion planning problems correspond to path search problems in multi-dimensional polyhedra, specified by boolean expressions of linear inequalities of size polynomial in the size of the scenes.

The problem CMPP can be interpreted as the problem of finding a path with cost $< m$ in a polyhedron, in which specific cost functions are associated to the sides. We shall focus on this aspect in the next subsection.

**4.3. A geometrical formulation of Turing-computability.** In the proof of theorem 4.1, we introduced a technique for simulating Turing machines by dynamic motion planning problems. Translated into the formalism of configuration space, our constructions also provide us with an interesting geometrical interpretation of the notion of Turing-computability.

We consider again dynamic scenes with forbidden motions, defined by marking some sides in a kinematic scene by the friction coefficient '$\infty$'. To capture the restrictions on motions in dynamic motion planning, we define the *free space of a dynamic scene* as the free space $FS(S) \subseteq \mathbf{R}^k$ of the underlying kinematic scene $S$ supplemented by a function $f : FS(S) \to \mathrm{Power}(\mathbf{R}^k)$, which assigns to each point $p \in FS(S)$ a set of vectors such that a path $w : [0,1] \to FS(S)$ corresponds to an allowed motion in the dynamic scene if and only if the tangent vector at any point $p \in w([0,1])$ is in the set $f(p)$. As $f$ defines a structure on $FS(S)$ which no legal path is allowed to 'leave', we shall further denote the tupel $(FS(S), f)$ a *structured space*.

This notion provides us with a powerful tool for describing physical problems. By allowing only vectors of specific lengths or directions as tangents of allowed paths, problems involving constraints on object velocities or path curvature can be represented. To plan 'smooth motions' of a rectangular object (e.g., a car) with translational and rotational freedom in some given planar environment, the allowed tangent vectors at each point $p = (x, y, \theta)$ of free space may be defined as $f(p) =$

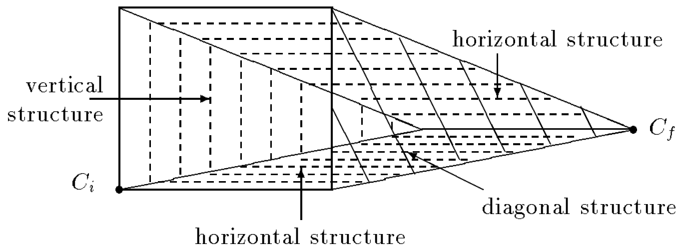



FIG. 17. *A structured polyhedron: the polyhedron consists of* 4 *structured sides. E.g., the triangle on the right can only be crossed in diagonal direction.*

$\{\ (\alpha \sin(\theta), \alpha \cos(\theta), \theta')\ ;\ \alpha \in [a, b]\ ,\ \theta' \in [\theta - \delta, \theta + \delta]\ \}$, with given constants $a, b$ and $\delta$.

Compared to the general model, the structured spaces that occured in this section constitute a very restricted subclass. The free space of a kinematic scene is a semi-algebraic space, which can be decomposed into disjoint, open, semi-algebraic cells, each cell being homeomorphic to the Euclidean space $\mathbf{R}^m$ for some $m \in \mathbf{N}_0$ [1]. In the special case of polygonal objects with purely translational freedom of motion, the free space is a polyhedron, and the cells can be triangulated into simplices. The function $f$, which defines the structure of the free space of the scenes constructed in the proof of theorem 4.1, is constant on each open simplex $s$ of the triangulation, i.e., $f(p) = f(p')\ \forall p, p' \in s$. As the constraints do only force tangent vectors to be zero in some coordinates, the value of $f$ corresponds to some linear subspace, or hyperplane, of the Euclidean space $\mathbf{R}^k$. We talk about *linearly structured polyhedra*, if the set of allowed vectors $f(p)$ for $p \in s$, $s$ an open simplex of the polyhedron, is defined as the space spanned by a given set of vectors.

As the dynamic scenes constructed in the proof of theorem 4.1 for given Turing machines and input strings only depend on the transition functions, we can interpret Turing machines as linearly structured polyhedra. The points of a linearly structured polyhedron can thus be compared with machine configurations, and allowed paths with computations.

Figure 17 provides us with some intuition about structured polyhedra: there are situations such that there exist paths starting at a given initial configuration which come arbitrarily close to a given final configuration though there is no allowed path of finite length connecting these configurations. The described situation corresponds to a Turing machine, which interprets the contents of its tape as the real number 1, successively divides this number by 2, and thus approaches but never reaches 0.

We conclude by the following theorem, which provides an alternative, geometrical definition of Turing-computability:

THEOREM 4.4. *A language $L \subseteq \{0, 1\}^*$ is Turing-computable if and only if there exists a linearly structured polyhedron $P$, an edge $e \subseteq P$, a linear function $h$ embedding $\{0, 1\}^*$ as binary number '$0.w$' in $e$, and a point $q \in P$, such that*

$$w \in L \quad \Leftrightarrow \quad \exists \text{ allowed path from } h(w) \text{ to } q \text{ in } P. \tag{2}$$

*Proof.* The construction of dynamic scenes for arbitrary Turing machines shows that a structured polyhedron $P$ (together with $e$, $h$ and $q$) satisfying condition (2) exists if $L$ is Turing-computable.

The opposite direction, that for any given $P$, $e$, $h$ and $q$, the language $L$ satisfying (2) is Turing-computable, follows from the fact that we can enumerate all points of $P$ that are reachable from a given point $\in e$ (the algorithm has to follow the structure of $P$ inductively, simplex for simplex, by marking the already detected reachable areas). $\square$

**5. Future work.** We think that the ideas, which we developed in this paper, are interesting from both a complexity theoretic as well as from a physical point of view.

Because of the intuitive power of geometrical reasoning, and also because of the close relationship to physical problems, it seems to be a challenging subject of research to develop a complexity theory based on the geometrical machine model introduced in theorem 4.4.

Interesting questions in this context are, how known characteristics of Turing machines (number of tapes, nondeterminism etc.) are reflected in the geometrical model, and how characteristics of the geometrical model could help to classify problems in complexity theory. We state the following concrete problems:

1. The number of tapes of a Turing machine corresponds to the number of different structures on the simplices of $P$. Which restrictions on the possible structures lead to interesting subclasses of the class of Turing-computable languages ?

2. Can two structured polyhedra accepting the same language be transformed to each other by some reasonable set of given transformation rules ?

3. The dimension of the polyhedron that we get by taking the roundabout way via dynamic scenes depends on the Turing machine $M$. Is it possible to find for each structured polyhedron an 'equivalent' 3-dimensional polyhedron ?

The latter of these problems is related to the probably most interesting question in practice, namely to detect the complexity of motion planning under constraints for lower-dimensional problem instances, especially instances involving only one movable object.

**Acknowledgements.** The author likes to thank Prof. G. Hotz for his support, helpful discussions and suggestions.

REFERENCES

[1] J. T. Schwartz and M. Sharir, *On the piano movers problem II: General techniques for computing topological properties of real algebraic manifolds*, Advances in Applied Mathematics, 4 (1983), pp. 298–351.
[2] C.-K. Yap, *Algorithmic Motion Planning*, Algorithmic and Geometric Aspects of Robotics Volume 1, ed. J.T. Schwartz, C.-K. Yap, Lawrence Erlbaum Associates, 1987, pp. 95–143.
[3] J. Canny, *A New Algebraic Method for Robot Motion Planning and Real Geometry*, Proc. 28th IEEE Symp. on Foundations of Computer Science (1987), pp. 39-48.
[4] J. Canny, *Some Algebraic and Geometric Computations in PSPACE*, Proc. 20th ACM Symp. on the Theory of Computing (1988), pp. 460–467.
[5] M. Sharir, *Algorithmic Motion Planning in Robotics*, Technical Report No. 392, New York University, August 1988.
[6] J. Reif, *Complexity of the Mover's Problem and Generalizations*, Proc. 20th IEEE Symp. on Foundations of Computer Science (1979), pp. 421–427.
[7] J. E. Hopcroft, D. A. Joseph, and S. H. Whitesides, *Movement Problems for 2-Dimensional Linkages*, SIAM Journal on Computing, 13 (1984), pp. 610–629.
[8] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, *On the complexity of motion planning for multiple independant objects: PSPACE-hardness of the warehouseman's problem*, Int. Journal of Robotics Research, Vol. 3, No. 4 (1984), pp. 76–88.
[9] P. Spirakis and C.-K. Yap, *Strong NP-hardness of moving many discs*, Information Processing Letters, 19 (1984), pp. 55–59.

[10] S. Fortune and G. Wilfong, *Planning Constrained Motion*, 20th ACM Symp. on the Theory of Computing (1988), pp. 445–459.
[11] P. Jacobs and J. Canny, *Planning smooth paths for mobile robots*, IEEE Int. Conf. on Robotics and Automation (1989), pp. 2–7.
[12] J. Canny, A. Rege, and J. Reif, *An Exact Algorithm for Kinodynamic Planning in the Plane*, 6th Symp. on Computational Complexity (1990), pp. 271–280.
[13] J. Canny and J. Reif, *New Lower Bound Techniques for Robot Motion Planning*, 28th IEEE Symp. on Foundations of Computer Science (1988), pp. 306–318.
[14] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
[15] H. R. Lewis and C. H. Papadimitriou, *Symmetric space-bounded computation*, Theoretical Computer Science 19 (1982), pp. 161–187.
[16] A. B. Kempe, *On a general method of describing plane curves of the nth degree by linkwork*, Proc. London Mathematical Society, 7 (1876), pp. 213–216.
[17] J. E. Hopcroft and D. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley Publishing Company, 1979.