The Synchronized Holonomic Model: A Framework for Efficient Generation of Motion

Marcell Missura[1], Daniel D. Lee[2], Oskar von Stryk[3], and Maren Bennewitz[1]

Abstract— **We present a simple and efficient mathematical framework suitable for generating motion in the context of a variety of robotic motion tasks ranging from low-level motor control up to high-level locomotion planning. Our concept is based on a one-dimensional second-order model that allows analytic computation of its inverse dynamics while respecting physical constraints. This makes it a particularly useful tool for tasks that are expressed only as a start and goal state, such as animation key frames or way points in path planning. By means of time synchronization, the model extends easily to an arbitrary number of dimensions in a way that the target is reached in all dimensions at the same time. The framework excels in terms of execution time, which lies in the microsecond range even for high-dimensional trajectory generation tasks. We demonstrate our method in two different settings—full-body trajectory generation and path planning—and show its benefits in comparison with current state-of-the-art algorithms.**

I. Introduction

Modern research in robotics is strongly focused on the development of autonomous robots that move about in indoor and outdoor environments to aid us in our every day lives. However, human-populated areas are vibrant with motion and full of unforeseen events that require immediate reaction. Fast and predictive control of motion is crucial for autonomous agents to operate in dynamic environments. Unfortunately, the high complexity of robotic systems and their nonlinear dynamics are rarely analytically tractable and make the development of such controllers a challenging task. Abstract models are needed that are fast to compute, even at the cost of being only approximately correct.

We propose a simple, decoupled holonomic model as a generic concept to approximate and generate dynamic motion. The mathematical simplicity of the embedded equations gives rise to very efficient computations of forward and inverse dynamics in closed form. The computational efficiency of the model allows it to scale up into the high-dimensional spaces of complex bodies of many degrees of freedom, whereby a time synchronization is possible such that the target state is reached in all dimensions at the same time. This can be a crucial feature when for example the target state is a whole-body pose of a robot, or a way point where all coordinates have to be reached at the same time. The low complexity of the output of the model, which is a small set of quadratic polynomials, provides a convenient basis for further processing, e.g., collision checking or smoothing. Possible applications for this framework are the computation of whole-body trajectories for servo motors or within search-based motion planning algorithms where a fast model is needed to evaluate a vast number of candidate trajectories in a short time.

We argue that using a second-order formulation comes with a computational advantage over higher degrees and enables analytic access to important concepts such as root finding and inverse dynamics. Yet, a second-order system approximates physically feasible motion with a continuous velocity profile. We show experiments that highlight the computational performance of our system when it comes to computing inverse dynamics and compare it with the Reflexxes Motion Library [1]. Furthermore, we leverage the analytic capabilities of our formulation to implement a basic motion planning algorithm that outperforms the state-of-the-art Global Dynamic Window approach [2].

II. Related Work

Cubic splines [3] and Bezier curves [4] are standard mathematical methods of key frame interpolation. They are fast to compute and produce C^2 motion trajectories. However, it is in general not possible to impose bounds on their derivatives in order to guarantee physically feasible motion.

The Reflexxes Motion Library [1], which we compare our method with in Section IV, addresses this problem. It is a successful implementation of a third-order system that has matured to industrial application and is commercially available as a product. It produces velocity-, acceleration-, and jerk-bounded trajectories that connect a start state and a target state, for both of which the position and velocity can be specified. The output of the library is represented as a sequence of up to seven pieces of third-order polynomials. Despite using numerical methods, the Reflexxes Motion Library guarantees a finite number of computation steps in the worst case and executes in the sub-millisecond range.

Beul et al. [5] presented a similar approach to computing jerk-bounded splines of third-order to control aerial vehicles. The authors formulate the equations of motion for a set of eight different motion templates to cover a wide range of motion tasks, and find an analytical solution for each case using a Matlab toolbox. The most generic case is a seven piece spline. In comparison to our framework, this work shows very clearly the increase in complexity from second-order to third-order systems for motion generation.

Mellinger et al. [6] used C^4 splines and minimized the integral of the squared fourth derivative in order to compute

[1]Marcell Missura and Maren Bennewitz are with Humanoid Robots Lab, University of Bonn, Germany

[2]Daniel D. Lee is with the GRASP Laboratory, University of Pennsylvania, USA

[3]Oskar von Stryk is with SIM, Department of Computer Science, Technische Universität Darmstadt, Germany

minimum snap trajectories for aggressive maneuvers for quadrotors. Their QP formulation is real-time capable as long as the number of degrees of freedom is relatively low.

Lau et al. [7] used quintic Bézier splines to plan a curvature continuous trajectory for a synchrodrive robot. A high-level straight line planner provides a heuristical guess for the curve parameters, which are then optimized along with the velocity profile of the vehicle by an anytime optimization algorithm. The choice of the C^4 quintic splines was motivated by a number of requirements such as continuous curvature even in the control points where two spline pieces are joined, and locality that would allow the effect of a parameter to be limited to a few neighbouring segments rather than the entire curve.

Purwin et al. [8] have already attempted to formulate a second-order motion generator similar to ours for wheeled omnidirectional vehicles. However, their formulation is more complex in terms of case differentiation and less complete, as it restricts the velocity in the target state to be zero.

Classic feedback controllers such as PID-controllers and Linear Quadratic Regulators [11] from optimal control theory are an alternative method to generate motion. They produce good results and they are widely used in industrial applications. However, the trajectories they produce are not time-optimal and it is impossible to specify a time to reach a target, which renders the time synchronization of multiple dimensions impossible.

Dynamic Motion Primitives [9], [10] are in principle PD-controllers which are augmented with a forcing function that is formulated as a weighted sum of radial basis functions. The basis functions are well suitable for machine learning and this is the context in which this approach has been investigated. However, since a large number of exponential functions may need to be evaluated, the method is not well suited for execution in the sub-millisecond range. It also lacks the ability to specify a target time or velocity.

III. The Synchronized Holonomic Model

We introduce the mathematical formulation of our model in a one dimensional setting. We discuss the implementation of forward dynamics and inverse dynamics functions, and then extend the formalism to two dimensions using a time synchronization technique.

Let $\mathbf{x} = [x, \dot{x}]^T$ describe the position and the velocity of a dynamic system in a one-dimensional space. We assume a linear second-order model given by

$$\ddot{x} = a, \tag{1}$$

meaning that the motion of the system is characterized by constant acceleration. We control this system with a bang-bang control signal $C = \{(t_k, \ddot{x}_k)_{k=0,\ldots,N-1}\}$, which is a set of N control tuples, where t_k are time intervals and $-A \leq \ddot{x}_k \leq A$ are piecewise constant accelerations bounded by parameter A. Furthermore, we also assume the velocity $-V \leq \dot{x} \leq V$ to be bounded by parameter V at all times.

Let $\mathbf{x}_0 = [x_0, \dot{x}_0]^T$ be the initial state. The future states at the discrete times by the ends of each bang are given by the recursive relation

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & t_k \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} t_k^2/2 \\ t_k \end{bmatrix} \ddot{x}_k. \tag{2}$$

A continuous forward dynamics function for a real-valued time $0 \leq T \leq \sum_{i=0}^{N} t_i$ is computed as

$$\mathcal{F}(\mathbf{x}_0, C, T) = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \mathbf{x}_m + \begin{bmatrix} t^2/2 \\ t \end{bmatrix} \ddot{x}_m, \tag{3}$$

where m is such that $\sum_{i=0}^{m} t_i \leq T < \sum_{i=0}^{m+1} t_i$, and $t = T - \sum_{i=0}^{m} t_i$.

A. Time-Optimal Inverse Dynamics

We are typically interested in the inverse setting and want to compute the set of controls that steer the point mass from its initial state $\mathbf{x}_0$ to a goal state $\mathbf{x}_1 = [x_1, \dot{x}_1]^T$. We define the inverse dynamics function $C_o = \mathcal{G}(\mathbf{x}_0, \mathbf{x}_1)$ which computes the control sequence C_o that accomplishes the motion from $\mathbf{x}_0$ to $\mathbf{x}_1$ in a time-optimal fashion. As postulated by Pontryagin's Maximum Principle, a time-optimal motion would consist of accelerating towards the target at the acceleration limit up to a switching time where the sign of the acceleration is flipped, and the point mass would decelerate and reach the target position exactly with the right velocity. An example of such a motion sequence is depicted in Figure 1 on the left. In hard cases when either the point mass cannot possibly accelerate to the desired velocity before it reaches the target location (undershoot), or the point mass cannot decelerate to the desired velocity over the available distance (overshoot), negative acceleration must be used first before switching to a positive one at the right time in order to accelerate and reach the target precisely with the desired velocity. Examples of both hard cases are shown in Figure 1. Hence, we assume the control sequence $C_o = \{(t_0, \sigma A), (t_1, -\sigma A)\}$ to consist of two bangs of opposite signs, both performed at the acceleration limit. The unknowns are the time intervals t_0 and t_1—the durations of the two bangs—and the approach sign σ.

In general, the sign of the approach σ is determined by the direction of the target relative to the start state given by $\sigma_d = \mathrm{sgn}(x_1 - x_0)$. In case of an undershoot or overshoot, this sign needs to be flipped. The hard cases can be detected efficiently using

$$\sigma_h = \begin{cases} -1, & \text{if } \sigma_d \dot{x}_1 > 0 \wedge \dot{x}_1^2 - \dot{x}_0^2 > 2A|x_1 - x_0| \\ -1, & \text{if } \sigma_d \dot{x}_0 > 0 \wedge \dot{x}_0^2 - \dot{x}_1^2 > 2A|x_1 - x_0| \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

Then, the approach sign is given by $\sigma = \sigma_d \sigma_h$.

Now for computing the bang times t_0 and t_1, we expand equation (2) using C_o and obtain

$$x_1 = \left(x_0 + \dot{x}_0 t_0 + \frac{1}{2}\sigma A t_0^2 \right) + (\dot{x}_0 + \sigma A t_0)\, t_1 - \frac{1}{2}\sigma A t_1^2, \tag{5}$$

$$\dot{x}_1 = \dot{x}_0 + \sigma A t_0 - \sigma A t_1. \tag{6}$$

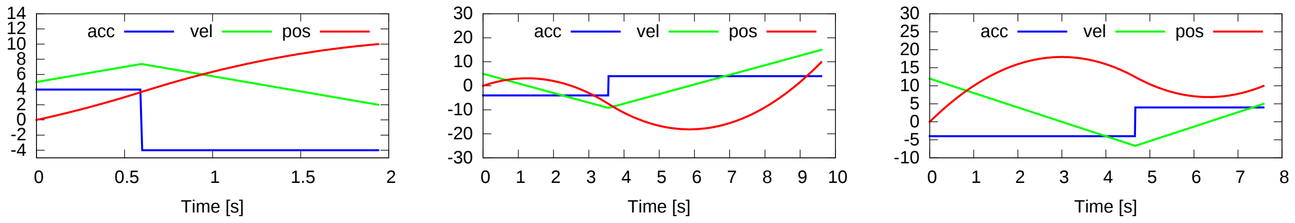

Fig. 1: Time-optimal motions generated by our bang-bang controller for a system with acceleration and velocity bounds $A = 4\,m/s^2$ and $V = 20\,m/s$. Left: Ideal motion between the states $\mathbf{x}_0 = [0,5]^T$ and $\mathbf{x}_1 = [10,2]^T$. The system accelerates towards the target at the acceleration limit up until the switching time at $0.6\,s$ where it flips the sign of the acceleration and brakes to reach the target location with the target velocity. Center: When the target velocity is changed to $15\,m/s$, the available distance $(10\,m)$ is not enough to accelerate to the target speed and a run up motion is computed that begins with a negative bang. Right: When the start velocity is changed to $12\,m/s$, the maximal deceleration is not sufficient to prevent the system from overshooting.

Solving for t_0 and t_1 yields

$$K = \sigma\sqrt{\frac{1}{2}\left(\dot{x}_0^2 + \dot{x}_1^2\right) + \sigma A(x_1 - x_0)} \tag{7}$$

$$t_0 = \frac{1}{\sigma A}\left(K - \dot{x}_0\right), \tag{8}$$

$$t_1 = \frac{1}{\sigma A}\left(K - \dot{x}_1\right). \tag{9}$$

The velocity limit V is imposed after the bang-bang sequence has been computed. We require that the input velocities are already within bounds such that $-V \leq \dot{x}_0 \leq V$ and $-V \leq \dot{x}_1 \leq V$. Then, if the absolute value of the velocity at the end of the first bang exceeds V, the solution has to be replaced with a bang-coast-bang sequence $C'_o = \{(t_0, \sigma A), (t_1, 0), (t_2, -\sigma A)\}$, where after accelerating to maximum velocity, the point mass would travel some amount of time at the velocity limit before decelerating to reach the target state. The approach sign σ remains the same as before. The bang times t_0, t_1, and t_2 can be computed directly as

$$t_0 = \frac{1}{\sigma A}\left(\sigma_d V - v_0\right), \tag{10}$$

$$t_2 = \frac{1}{\sigma A}\left(\sigma_d V - v_1\right), \tag{11}$$

$$t_1 = \frac{(x_1 - x_0) - \left(\dot{x}_0 t_0 + \frac{1}{2}\sigma A t_0^2\right) - \left(\sigma_d V t_2 - \frac{1}{2}\sigma A t_2^2\right)}{\sigma_d V}. \tag{12}$$

By handling the initially mentioned hard cases and the velocity constraint explicitly, the time-optimal function $\mathcal{G}$ guarantees a precise solution for any given motion task without violating the physical constraints.

B. *Timed Inverse Dynamics*

Aside from the time-optimal motion, we also define a timed inverse dynamics function $C_T = \mathcal{H}(\mathbf{x}_0, \mathbf{x}_1, T)$ which computes a control sequence C_T that steers the point mass in a way that the target state $\mathbf{x}_1$ is reached exactly at time $T \geq 0$ unless this would violate the acceleration and the velocity constraints. This is the case if and only if the requested motion time T is less than the optimal time computed by the $\mathcal{G}$ function.

The control sequence $C_T = \{(t_0, a), (t_1, -a)\}$ consists again of two bangs, but this time with a variable magnitude a of the acceleration. We set

$$T = t_0 + t_1, \tag{13}$$

$$x_1 = (x_0 + \dot{x}_0 t_0 + \frac{1}{2}a t_0^2) + (\dot{x}_0 + a t_0)t_1 - \frac{1}{2}a t_1^2, \tag{14}$$

$$\dot{x}_1 = \dot{x}_0 + a t_0 - a t_1, \tag{15}$$

and eliminate t_0 and t_1 to solve for a

$$a = \frac{\pm\sqrt{4\left(\delta^2 - T\delta\gamma\right) + 2T^2\left(\dot{x}_0^2 + \dot{x}_1^2\right)} + 2\delta - T\gamma}{T^2}, \tag{16}$$

using $\delta = x_1 - x_0$, and $\gamma = \dot{x}_0 + \dot{x}_1$. When computing a, both the negative and the positive result of the square root in (16) must be computed and the one with the larger absolute value is used. This results in the acceleration a with the correct sign. In the exceptional case that $\dot{x}_0 = \dot{x}_1 = (x_1 - x_0)/T$, a evaluates to zero and the solution is $C_T = \{(0, T)\}$. Otherwise the acceleration bound $-A \leq a \leq A$ is applied *before* the bang times t_0 and t_1 are computed.

The next step is the computation of t_0 and t_1 with the now known and bounded a using

$$t_0 = \begin{cases} T - \sqrt{\frac{T\dot{x}_0 - \delta}{a} + \frac{T^2}{2}}, & \text{if } \frac{T\dot{x}_0 - \delta}{a} + \frac{T^2}{2} > 0 \\ T, & \text{otherwise} \end{cases}, \tag{17}$$

$$t_1 = T - t_0. \tag{18}$$

Note that (14) should be solved for t_0 rather than (15). This makes sure that if an infeasible motion time T is given, an error will first occur in the target velocity, but the position will still be reached at the right time. Only if t_0 turns out to be larger than T—this is what the case differentiation in (17) protects against—the target position can no longer be reached. In this case, the entire motion time T is used for the first bang. This way, even if an infeasible motion time was requested, the equations robustly produce a sensible motion that attempts to reach the target by giving the target position priority over the target velocity.

The velocity limit V is imposed in the same manner as with the time-optimal function $\mathcal{G}$. Assuming valid input parameters $-V \leq \dot{x}_0 \leq V$ and $-V \leq \dot{x}_1 \leq V$, if the absolute value of the velocity at the end of the first bang exceeds V, the solution is a bang-coast-bang sequence $C'_T = \{(t_0, a), (t_1, 0), (t_2, -a)\}$. Using the acceleration a

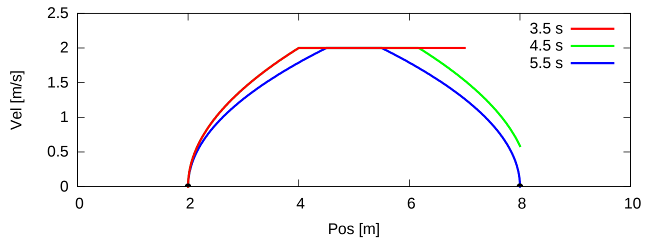

Fig. 2: Timed motions generated by our bang-bang controller from the start state $\mathbf{x}_0 = [2, 0]^T$ to the target state $\mathbf{x}_1 = [8, 0]^T$ at desired times $T = 3.5\,s$ (red), $T = 4.5\,s$ (green), and $T = 5.5\,s$ (blue), respectively. The acceleration and velocity constraints were set to $A = 1\,m/s^2$ and $V = 2\,m/s$. In the first case, 3.5 s are not sufficient to reach the target position. Accelerating at the limit, the system hits the velocity bound after 2.0 s and keeps moving towards the target until the motion time runs out. In the second case, the correct position is reached, but with a higher than desired velocity. In the third case, $5.5\,s$ are long enough for all target conditions to be fulfilled without having to use maximum acceleration.

and the first switching time t_0 that we determined for the bang-bang sequence, we compute the peak velocity $v^* = \dot{x}_0 + at_0$ at the end of the first bang. The sign of this velocity $\sigma_d = \text{sgn}(v^*)$ is needed for the following computations, in particular because the velocity limit V is given as a positive parameter. We set

$$T = t_0 + t_1 + t_2, \tag{19}$$

$$t_0 = \frac{V - v_0}{a}, \tag{20}$$

$$t_2 = \frac{V - v_1}{a}, \tag{21}$$

$$x_1 = (x_0 + \dot{x}_0 t_0 + \frac{1}{2}at_0^2) + V\,(t_1 + t_2) - \frac{1}{2}at_2^2, \tag{22}$$

and solve for a with the correct sign

$$a = \sigma_d \left| \frac{2\,V\,(V - \dot{x}_1 - \dot{x}_0) + \dot{x}_0^2 + \dot{x}_1^2}{2\,(T\,V - x_1 + x_0)} \right|. \tag{23}$$

Then, after applying the acceleration bound $-A \leq a \leq A$, we compute the switching times

$$t_0 = \min\{\frac{\sigma_d\,V - v_0}{a}, T\}, \tag{24}$$

$$t_2 = \min\{\frac{\sqrt{-\left(\sigma_d\,V - \dot{x}_0\right)^2 + 2a\,(T\sigma_d\,V - \delta)}}{a}, T - t_0\}, \tag{25}$$

$$t_1 = T - t_0 - t_2. \tag{26}$$

Similar to the bang-bang case when the velocity limit has not been reached, we use the technique of bounding the switching times in order to ensure that the motion stops at time T. For obtaining equation (25), we solve the spatial equation (22) in order to give the target position priority over the velocity. This failure mode behavior is visualized in Figure 2.

C. Two-Dimensional Model

When modelling a two-dimensional system, we simply regard two decoupled, one-dimensional systems $\mathbf{X}_0 = [\mathbf{x}_0, \mathbf{y}_0]^T$. The control input is extended to two sets of tuples $\mathbf{C}_2 = (C_x, C_y)$, one for each respective dimension such that $C_x = \{(t_{x_k}, \ddot{x}_k)_{k=0,\ldots,N_x-1}\}$ and $C_y = \{(t_{y_k}, \ddot{y}_k)_{k=0,\ldots,N_y-1}\}$. The two dimensions are synchronized in the sense that the target states are reached at the same time, i. e., $\sum_{i=0}^{N_x-1} t_{x_i} = \sum_{j=0}^{N_y-1} t_{y_j}$. Note that only the total time is synchronized. C_x and C_y can still contain different switching times and even a different number of bangs, if one of the dimensions reaches the velocity limit and the other does not.

The extension of the forward dynamics to the two-dimensional case is trivially given by

$$\mathcal{F}_2\left(\mathbf{X}_0, \mathbf{C}_2, T\right) = \begin{bmatrix} \mathcal{F}\left(\mathbf{x}_0, C_x, T\right) \\ \mathcal{F}\left(\mathbf{y}_0, C_y, T\right) \end{bmatrix}. \tag{27}$$

The two-dimensional time-optimal inverse dynamics function $\mathcal{G}_2$ warrants further attention. We implement the two-dimensional function by computing both time-optimal control sequences

$$C_x = \mathcal{G}\left(\mathbf{x}_0, \mathbf{x}_1\right), \tag{28}$$

$$C_y = \mathcal{G}\left(\mathbf{y}_0, \mathbf{y}_1\right), \tag{29}$$

independently for each dimension. Then, we determine the greater of the two optimal motion times

$$T_{max} = \max\{\sum_{i=0}^{N_x-1} t_{x_i}, \sum_{j=0}^{N_y-1} t_{y_j}\} \tag{30}$$

and use the timed inverse dynamics function

$$C_x = \mathcal{H}\left(\mathbf{x}_0, \mathbf{x}_1, T_{max}\right) \tag{31}$$

to recompute the motion of the faster dimension to match the time T_{max} of the dimension that takes longer time to complete. Here we used x as an example for the faster dimension. Since the time T_{max} is always greater than or equal to the time needed for the time-optimal motion, it is guaranteed that the motion task is feasible in time T_{max} and $\mathcal{H}$ produces a control signal that reaches the target precisely at the right time.

The two-dimensional timed inverse dynamic function

$$\mathcal{H}_2\left(\mathbf{X}_0, \mathbf{X}_1, T\right) = \begin{bmatrix} \mathcal{H}\left(\mathbf{x}_0, \mathbf{x}_1, T\right) \\ \mathcal{H}\left(\mathbf{y}_0, \mathbf{y}_1, T\right) \end{bmatrix} \tag{32}$$

is again a trivial composition of two one-dimensional ones.

The Synchronized Holonomic Model can easily be extended to an arbitrary number of dimensions by composing one-dimensional functions and synchronizing with the dimension that requires the most time. The computation time scales linearly with the number of dimensions. The concatenation of multiple way points $\mathbf{X}_k$ to compose complex motions arises naturally by computing the pairwise control sequences $\mathbf{C}_{k=0,\ldots,N-1} = \mathcal{G}\left(\mathbf{X}_k, \mathbf{X}_{k+1}\right)$. The computation time scales linearly in the number of way points as well.

IV. Experimental Results

We present experiments performed in the context of two distinct settings of robotic motion. The first context is whole-body motion generation, where motion tasks typically take place in the high-dimensional spaces of many degrees of freedom. We compare the performance of our Synchronized Holonomic Model with the Reflexxes Motion Library [1] and common cubic splines. The second context is locomotion planning, where we embed our method in a custom motion planner and evaluate it with respect to the Global Dynamic Window algorithm [2].

A. *Whole-Body Motion Generation*

In the whole-body motion generation context the task is the generation of motor commands with a high frequency on a low level of abstraction. Especially when it comes to humanoid robots, a large number of actuators (>20) must be provided with a set point at a frequency of roughly 100 Hz. A common strategy is to express the motion in terms of key frames, i. e., snapshots of positions, velocities, and sometimes accelerations, at discrete times in the future. Then, a key frame interpolation technique is used to obtain fine-grained intermediate motion states in the gaps in between the frames. In most cases such motion controllers are feedback controllers and the measured position is fed back during the motion as the first key frame while he target key frame stays fixed. This way the system can automatically account for drift and disturbances, and the target can potentially change during the motion, but it requires recomputing the entire motion trajectory in every iteration. This is when computational efficiency becomes particularly important.

Perhaps the most commonly known such interpolation technique is the cubic spline. A spline, however, does not respect physical constraints. The Reflexxes Motion Library Type IV [1] computes a third order "spline of splines" that adheres to bounds on the first, second, and third derivatives. Our Synchronized Holonomic Model computes second order polynomials and enforces bounds on the first and second derivatives. We measured the performance of all three algorithms on a 2.16 GHz Intel CPU by presenting each of them a large number (10,000) of key frame pairs with an increasing number of dimensions and having them compute their output. We averaged the computation times over all key frame pairs and recorded the data in Table I. The cubic spline is a baseline for computation times as it does away with a small number of multiplications, additions, and divisions. However, it entirely ignores the velocity and acceleration constraints. The computation time of our method is dominated by the square root function appearing in equations (8), (9) and (16), (17), which is a more costly operation than for example division. Still, while fully respecting the velocity and acceleration constraints, the Synchronized Holonomic Model manages to achieve computation times that lie clearly in the range of a few microseconds and are competitive with the cubic spline. The data in Table I reflects the fact that computing in more than one dimension incurs the additional cost of time synchronization, meaning that the time-optimal inverse dynamics function is computed for all dimensions first, and then the timed inverse dynamics function is computed for all but the slowest dimension. The Reflexxes Motion Library includes numerical methods and takes about two orders of magnitude more time to finish. The

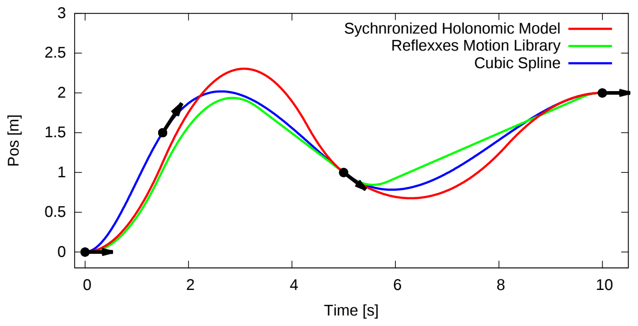

Fig. 3: Position trajectories over four key frames generated by the Synchronized Holonomic Motion Model, the Reflexxes Motion Library, and a cubic spline. The second key frame has been placed intentionally out of reach of the acceleration and velocity constraints. The cubic splines ignores the constraints and touches the key frame nonetheless while the other two methods respect the constraints and miss the second key frame.

results of our performance measurement match the numbers given in the documentation of the library. All three methods can potentially be used as feedback controllers, but of course, a shorter execution time allows a higher control frequency.

Furthermore, the nature of the output of these trajectory generators has an influence on the processing times of algorithms that post-process their result. Collision checking, for example, is an integral part of motion planning algorithms that can make good use of an efficient representation. All three methods output the coefficients of polynomials. A cubic spline is exactly one cubic polynomial between two key frames. Our method computes two or three quadratic polynomials between two key frames, depending on whether the velocity constraint has to be enforced or not. The Reflexxes Motion Library computes in most cases seven cubic polynomials. Collision checking in a polygonal or polyhedral environment means intersecting these polynomials with lines or surfaces, which boils down to computing the roots of such polynomials. It is much faster to compute the roots of quadratic polynomials than cubic ones, even more so if there are few polynomials to begin with.

Figure 3 shows a plot of the output of all three methods after receiving the same four key frames as input, the second of which has been intentionally placed out of reach of the physical constraints. The cubic spline is not aware of these constraints and touches each key frame precisely, thereby producing potentially infeasible motion. Our algorithm and the Reflexxes Motion Library both generate a sensible and feasible motion that attempts to, but does not reach the second key frame. Both methods manage to recover by the third key frame and continue on to the fourth. Notably, the position curve produced by our method has a larger spatial footprint

	1D	2D	6D	20D
Cubic Spline	0.1μs	0.21μs	0.63μs	2.08μs
Synch. Holonomic Model	0.2μs	0.54μs	2.02μs	7.16μs
Reflexxes Motion Library	35.4μs	77.6μs	220.3μs	778.8μs

TABLE I: Computation times of a cubic spline, our Synchronized Holonomic Model, and the Reflexxes Motion Library for a motion signal between two key frames with an increasing number of dimensions.

than the one produced by the Reflexxes Motion Library. The reason for this is a key difference in the implementation of these algorithms. While the Synchronized Holonomic Model computes the minimal peak acceleration in equation (16) that is needed to reach the next key frame at the desired time and accelerates as little as possible over the entire motion, the Reflexxes Motion Library ramps up the acceleration to the limit at first and then spends a considerable amount of time travelling with constant velocity. This effect can be seen for example between 6 and 10 seconds in Figure 3. This way, the peak velocity is minimized rather than the peak acceleration. The same behavior can easily be implemented in our framework by always using a bang-coast-bang sequence when computing the timed inverse function $\mathcal{H}$ in Section III-B, but so far it is unclear which type of behavior is more advantageous. Last but not least, the authors of the Reflexxes Motion Library report on the existence of gaps in time where the synchronization of multiple dimensions fails due to the nature of the jerk bounded third-order polynomials. With our acceleration bounded second-order method there is only a minimal time below which a motion is not feasible. Otherwise a solution always exists.

B. *Motion Planning*

We demonstrate the potential of the Synchronized Holonomic Model in the context of locomotion planing in a two-dimensional scenario. We consider a simple box world as shown in Figure 4. A holonomic vehicle referred to as the "taxi" has the task of collecting a passenger waiting in drop zone 2 without colliding with the obstacle in the center of the scene. When the passenger is collected, the taxi delivers the passenger to drop zone 1. Then, a new passenger is spawned and asks to be picked up. We implemented this game with the help of the Box2D rigid body simulation engine [12].

We created a simple motion planning algorithm using the Synchronized Holonomic Model by computing the time-optimal trajectory directly from the taxi to the passenger, and also the three-point trajectories from the taxi over each way point associated with the corners of the obstacle to the passenger. We sort these trajectory candidates by their total motion time and check them for collision in ascending order. The first collision-free trajectory is the fastest solution. The acceleration in the starting point of the best trajectory is then converted to a force which is applied in the Box2D simulation to move the taxi. The entire motion planning algorithm averages at about $220\,\mu$s computation time, which allows us to use it as a feedback controller with fast re-planning capabilities. In this simple setting, this algorithm is sufficient to reliably maneuver the taxi between the drop zones without colliding with the obstacle. Figure 4 shows a screen shot of our simulation and illustrates the computed trajectory candidates. The fastest collision-free trajectory is highlighted in blue. In fact, we can be sure that this algorithm drives the taxi in a time-optimal fashion through the given way point.

When using the Synchronized Holonomic Model to generate a trajectory, collision checking is an easy task. First we expand the obstacle by the size of the taxi using the Minkowski sum of the two rectangles. This way, the taxi can be considered to be a point. Now we consider a vertical edge of the expanded obstacle given by its end point coordinates (x_r, y_u, x_r, y_l). Given the motion state of our robot in the x-dimension $\mathbf{x}_0 = [x_0, \dot{x}_0]^T$ and a bang $(t, \ddot{x})$, a collision between the taxi and the edge can be computed with the equation $x_0 + \dot{x}_0 t + \frac{1}{2}\ddot{x}t^2 = x_r$. Solving for t will provide up to two possible solutions in time, out of which we are only interested in the positive one with the smaller value. If such a solution exists, and $y_u \geq y(t) \geq y_l$, we have a collision between the taxi and the edge. Performing this computation for every bang in the control sequence and for every edge in the scene yields a complete and efficient collision check. If the edge is not vertical, a rotation is applied in order to transform the taxi and the edge into a vertical edge situation. Consequently, this collision checking method can be applied to arbitrary complex polygons with slanted edges.

We compare our solution with an implementation of the Global Dynamic Window Approach (DWA) [2], one of the best performing state-of-the-art algorithms. We tuned the parameters and profiled the code of our DWA implementation to get the most of its performance. It does drive the taxi remarkably well. It executes in $650\,\mu$s after a preprocessing time of 6.5 ms for a navigation table that only needs to be computed once. Figure 5 shows the DWA algorithm performing in our box world simulation. Neither the DWA, nor our way point based algorithm ever collide with the obstacle. However, while our algorithm delivers 36 passengers per minute, DWA only manages to deliver 29. Figure 6 visualizes the trajectories produced by these algorithms. The near optimal trajectories of our Synchronized Holonomic Model-based algorithm are shown in red. The DWA trajectories are shown in blue. The DWA trajectories are less consistent and less ideal than the ones produced by our method. Another not so obvious place where DWA loses time is the stopping phase in the drop-off zone where DWA decelerates and stops a bit too smoothly, while our method performs near perfect landings exploiting the full dynamics of the vehicle. Furthermore, a noteworthy advantage of our method is that it is predictable in the sense that it computes a complete plan to the target. The DWA algorithm only follows local rules and the only way to predict the future trajectory of the taxi is to simulate it.

Certainly, if the complexity of the environment increases, the computation time of our algorithm will increase with the number of edges and way points present in the scene, while the computation time of the DWA will always remain constant. However, when moving obstacles come into play that cannot be included in its navigation table, the DWA will have to rely on its limited preview horizon and produce suboptimal behavior, or even collide. Our algorithm has the potential to compute three-point trajectories among moving obstacles, but this is future work.

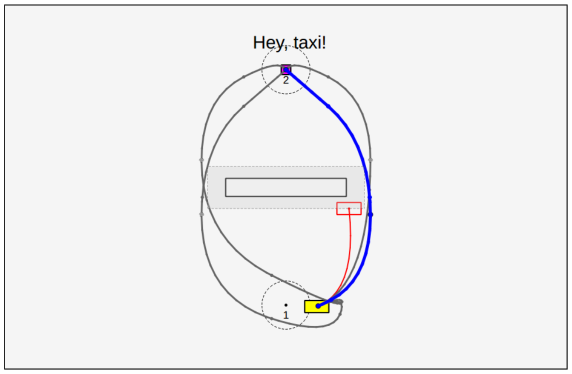

Fig. 4: Visualization of our Synchronized Holonomic Model-based motion planner. Trajectory candidates originate from the “taxi” (the yellow box) that is already in motion towards the passenger. The set of trajectory candidates includes a direct way from the taxi to the goal and three-point trajectories over each way point associated with the corners of the obstacle. The direct trajectory collides with the obstacle and is marked in red. The grey trajectories did not need to be collision checked as the blue trajectory is collision-free and has the shortest travelling time among all candidates.

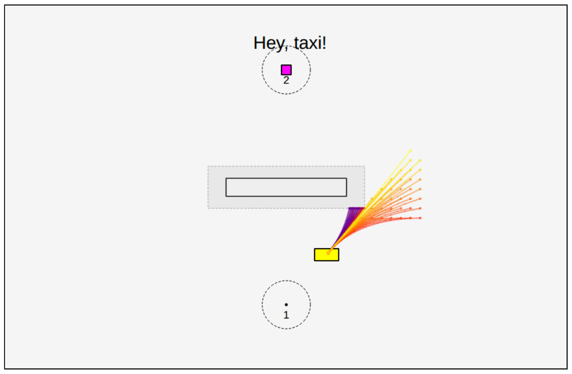

Fig. 5: Visualization of the Dynamic Window Algorithm performing in our simulation. The color coded trajectories are sampled in acceleration space and evaluated according to an objective function that includes information from the NF1 table in the background and the collisions with the obstacle. The best trajectories are marked in yellow.

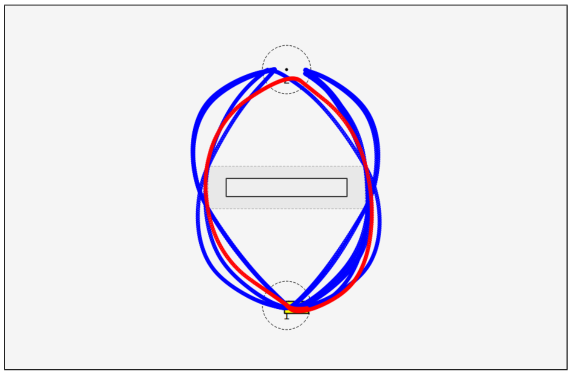

Fig. 6: Trajectories as they happened in simulation. The red trajectories were produced by our Synchronized Holonomic Model-based algorithm, the blue trajectories by the Dynamic Window Approach. Our method consistently follows one of two symmetrical time-optimal solutions. The Dynamic Window Approach is less consistent and follows suboptimal paths.

V. Conclusions

We presented a mathematical framework that computes the inverse dynamics of a holonomic system in order to generate a smooth motion trajectory from a given initial state to a goal state. Time-optimal and timed motions are both available. Apart from motion tasks that are infeasible due to acceleration and velocity bounds, the formalism is guaranteed to find a solution. It can accommodate an arbitrary number of dimensions by synchronizing them in a way that all dimensions reach the target state at the same time. We argue that using a second-order model offers the benefits of high computational efficiency and analytic convenience in both calculating the inverse dynamics and post-processing the output. We demonstrated these advantages in experiments where our method outperformed the state-of-the-art solutions in two distinct areas of motion planning: whole-body motion generation and locomotion planning. In the former we highlighted the computational edge of our equations in comparison with the Reflexxes Motion Library, while in the latter we used our method to implement a simple motion planning algorithm that outperformed the Dynamic Window Approach in terms of task efficiency and computation time.

Perhaps the most compelling question that arises when comparing our method to other works using third or higher degree polynomials as basis functions is whether the steep increase of complexity and computation time is a necessary cost in order to circumvent the discontinuities in the acceleration profile of second-order systems.

References

[1] Torsten Kröger and Friedrich M Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26(1):94–111, 2010.

[2] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *In IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 341–346, 1999.

[3] Michiel Hazewinkel. Spline interpolation. In *Encyclopedia of Mathematics*, 2001.

[4] Michiel Hazewinkel. Bézier curves. In *Encyclopedia of Mathematics*, 2001.

[5] Marius Beul and Sven Behnke. Analytical time-optimal trajectory generation and control for multirotors. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016.

[6] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *In IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2011.

[7] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Kinodynamic motion planning for mobile robots using splines. In *In IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[8] Oliver Purwin and Raffaello D’Andrea. Trajectory generation and control for four wheeled omnidirectional vehicles. *Robotics and Autonomous Systems*, 54(1):13–22, 2006.

[9] Stefan Schaal. Dynamic movement primitives: A framework for motor control in humans and humanoid robotics. In *The International Symposium on Adaptive Motion of Animals and Machines*, Kyoto, Japan, March 2003.

[10] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[11] Karl Johan Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, USA, 2008.

[12] Erin Catto. Box2d: A 2d physics engine for games. `http://box2d.org`, 2010.