Computing Funnels Using Numerical Optimization Based Falsifiers*

Jiří Fejlek\textsuperscript{1,2†} and Stefan Ratschan\textsuperscript{1‡}

\textsuperscript{1}The Czech Academy of Sciences, Institute of Computer Science
\textsuperscript{2}Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering

March 8, 2022

Abstract

In this paper, we present an algorithm that computes funnels along trajectories of systems of ordinary differential equations. A funnel is a time-varying set of states containing the given trajectory, for which the evolution from within the set at any given time stays in the funnel. Hence it generalizes the behavior of single trajectories to sets around them, which is an important task, for example, in robot motion planning.

In contrast to approaches based on sum-of-squares programming, which poorly scale to high dimensions, our approach is based on falsification and tackles the funnel computation task directly, through numerical optimization. This approach computes accurate funnel estimates far more efficiently and leaves formal verification to the end, outside all funnel size optimization loops.

1 Introduction

An important task in robotics motion planning is to follow a given trajectory into some target set \cite{26, 3, 30}. Especially, numerous path planning algorithms \cite{33, 29, 17} rely on this task. This involves first designing a controller that follows this trajectory \cite{22}, and then determining a neighbourhood of the trajectory (a funnel \cite{31}) where the controller fulfils its goal of reaching a given target set \cite{18, 28}. In this paper, we present an efficient method for this second task.

In the literature \cite{29, 31, 15}, funnel construction is usually based on sum-of-squares programming (SOS) \cite{27}, a relaxation technique for polynomial systems. However, such formulations are sensitive to numerical errors and scale poorly to high dimensions—both in theory \cite{16} and in practice \cite{23}. Moreover, SOS methods tend to underestimate the actual funnel size \cite{7, 25}.

To alleviate these drawbacks of SOS methods, we propose the use of falsifiers based on numerical optimization to compute funnel candidates directly. We leave potential formal verification to the end. This allows for an efficient funnel optimization loop, since the dimensions of subsequent nonlinear programming (NLP) problems are the same as the dimension of the system and do not further increase as it is the case of SOS methods, which increases at least quadratically in the problem dimension \cite{16}. Our computational experiments show that without the verification part, the falsifiers still provide quite accurate estimates of control funnels. As an additional advantage we note that the method is also applicable to non-polynomial systems that SOS-based methods cannot handle directly (i.e., they need the non-polynomial dynamics to be approximated by a polynomial one).

In further related work \cite{2, 24}, funnels are computed using linearization of system dynamics with conservative estimates of nonlinear effects. Unfortunately, both publications lack a direct comparison with methods based on SOS on the same benchmark problem which makes an estimation of their performance in terms of scalability and conservativeness difficult. Verification methods for ODEs \cite{19} and hybrid systems \cite{8} insist on formal verification of the results, but compute an overapproximation of all trajectories from a given initial set instead of funnels. This means that system trajectories that start within a computed overapproximation but not from the initial set are not guaranteed to stay in the overapproximation. Concerning SOS relaxations,

\footnotesize
\textsuperscript{*}This work was supported by the project GA21-09458S of the Czech Science Foundation GA ČR and institutional support RVO:67985807.
\textsuperscript{†}ORCID: 0000-0002-9498-3460
\textsuperscript{‡}ORCID: 0000-0003-1710-1513
it would also be possible to use alternatives, namely DSOS and SDSOS \(1\). This improves scalability but results in more conservative solutions which would result in smaller funnels.

The structure of the paper is as follows. In Section \(2\) we state the precise problem. In Section \(3\) we review the problem of funnel construction and describe existing approaches based on SOS programming. In Section \(4\) we introduce our algorithm and explain its implementation. In Section \(5\) we provide computational experiments. Section \(6\) concludes the paper.

## 2 Problem Statement

Consider a system

\[
\dot{x} = F(x, t)
\]

where \(F: \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n\) is a smooth function. We further assume that system \(1\) has a unique solution for any initial point \(x_0 \in \mathbb{R}^n\) and time \(t_0 \geq 0\). We denote this solution by \(\Sigma(x_0, t_0)\), which is a function in \([t_0, \infty) \to \mathbb{R}^n\). We will also simply write \(\Sigma(x_0)\) for \(\Sigma(x_0, 0)\).

Let \(x_0 \in \mathbb{R}^n\) and \(T > 0\). In this paper, we consider the problem of computing funnels \(31\). A funnel is a time-varying set of states \(F(t)\) for \(0 \leq t \leq T\) such that for all \(t \in [0, T]\) and \(x \in F(t)\), the solution from \((x, t)\) stays in the funnel, i.e., \(\Sigma(x, t) \in F(t)\) for all \(t \in [t, T]\). In addition, we require that the final part of a funnel \(F(T)\) is a subset of some chosen set of goal states \(\mathcal{G}\). Finally, we also want a funnel as large in volume as possible.

To ease the construction of funnels, we reduce our attention to funnels \(F(t)\) that are constructed around some chosen system trajectory \(\tilde{x}(t) = \Sigma(x_0)(t)\) for \(t \in [0, T]\) that ends in \(\mathcal{G}\) \(29\) \(31\). These funnels can be described using a differentiable positive definite function \(P(x, t) : \mathbb{R}^n \times [0, T] \to \mathbb{R}\), and a differentiable real function \(\rho(t) : [0, T] \to (0, +\infty)\) as sublevel sets \(F(t) = \{x \in \mathbb{R}^n \mid P(x - \tilde{x}(t), t) \leq \rho(t)\}\). Hence, funnel construction reduces to construction of functions \(P(x, t)\) and \(\rho(t)\) such that \(F(t)\) forms a funnel as large in volume as possible.

Simplifying the problem even further, we will assume that the shape \(P(x, t)\) is provided beforehand e.g. as an ellipsoid given by a solution of the corresponding Lyapunov \(11\) or Riccati equation \(12\), or an optimized one for the linearized model using matrix inequalities \(6\). Consequently, all that remains is to determine an optimal \(\rho(t)\) wrt. fixed \(P(x, t)\).

## 3 Funnel construction

In this section, we will shortly review general funnel construction and an SOS-based variant \(29\) \(31\). We assume a set \(\mathcal{F}(t) = \{x \in \mathbb{R}^n \mid P(x - \tilde{x}(t), t) \leq \rho(t)\}\) as described in the previous section. First, we explore conditions on \(P(x, t)\) and \(\rho(t)\) that make \(\mathcal{F}(t)\) a funnel. Let us define for each \(t \in [0, T]\) sublevel sets \(L_{P, \rho}^\leq(t) \equiv \{x \in \mathbb{R}^n \mid P(x, t) \leq \rho(t)\}\), and level sets \(L_{P, \rho}(t) \equiv \{x \in \mathbb{R}^n \mid P(x, t) = \rho(t)\}\). Hence, we can write \(\mathcal{F}(t) = L_{P, \rho}^\leq(t) \oplus \tilde{x}(t)\), where \(\oplus\) denotes Minkowski addition.

Assume that \(P(x, t)\) and \(\rho(t)\) are chosen in such a way that the final sublevel set is a subset of a target set, i.e., \(L_{P, \rho}^\leq(T) \oplus \tilde{x}(T) \subseteq \mathcal{G}\). Moreover, assume that for all \(t \in [0, T]\) and all \(x \in L_{P, \rho}(t) \oplus \tilde{x}(t)\) the value of \(P\) decreases faster or increases slower than \(\rho(t)\) along the system dynamics, that is

\[
\dot{P}(x - \tilde{x}(t), t) < \dot{\rho}(t),
\]

where \(\dot{P}(x, t) = \nabla_x P(x, t)^T \dot{x} + \frac{\partial}{\partial x} P(x, t)\). Due to this requirement, for all \(t \in [0, T]\), all states \(x\) in \(L_{P, \rho}^\leq(t) \oplus \tilde{x}(t)\) stay in \(L_{P, \rho}^\leq(t) \oplus \tilde{x}(t)\) for all \(t \in [t, T]\). Consequently, the set \(L_{P, \rho}^\leq(t) \oplus \tilde{x}(t)\) forms a funnel.

As we mentioned in the previous section, we reduce our attention to the case in which \(P(x, t)\) is fixed beforehand. An often suitable candidate for \(P(x, t)\) is a solution of the Lyapunov or Riccati equation. A candidate then has quadratic form \(P(x, t) = x^T S(t)x\), where \(S(t)\) is a solution to a respective equation. Still, even if we fix \(P(x, t)\), we still need to determine \(\rho(t)\). Additionally, we would like \(\rho(t)\) to be chosen in such a way that the sublevel sets are as large as possible.

In previous work \(29\) \(31\), \(\rho(t)\) is parametrized piecewise-linearly and the parameters are optimized using a line-search approach. In each iteration, \(\rho(t)\) is verified using SOS programming. Moreover, computation of \(\rho(t)\) can be approximated by performing it in finitely many time samples \(29\) \(31\). This partially alleviates

\(^1\) actually, continuous and right differentiable suffices
the needed computational burden due to ignoring polynomial dependence in \( t \). However, certain care must be taken with choosing time samples to obtain a reasonable approximation of a funnel as we will see later in Example 1 of our computational experiments.

To be more specific, let us assume that both \( P(x - \tilde{x}(t), t) \) and \( \dot{P}(x - \tilde{x}(t), t) \) are polynomials in \( x \). Choose time instants \( 0 = t_1 < t_2 < \cdots < t_N = T \), denote \( \rho(t_1), \ldots, \rho(t_N) \) as \( \rho_1, \ldots, \rho_N \), and set \( \rho(t) = \rho_i + \frac{\rho_{i+1} - \rho_i}{t_{i+1} - t_i} t \) for an interval \( t \in [t_i, t_{i+1}] \). To optimize the volume of a discrete funnel, a linear cost \( \sum_{i=1}^{N} \rho_i \) is considered for optimization \([31]\). Hence, values \( \rho_1, \ldots, \rho_N \) for which \( \rho(t) \) meets \([2] \) in all time samples are found by solving a bilinear SOS program

\[
\max_{\rho_1, \ldots, \rho_N, \varepsilon, \mu_1, \ldots, \mu_N} \sum_{i=1}^{N} \rho_i \\
\text{subject to} \\
L_{\tilde{P}, \rho}(T) + \tilde{x}(T) \subseteq \mathcal{G}, \\
\varepsilon - \dot{P}(x - \tilde{x}(t_i), t_i) + \frac{\rho_{i+1} - \rho_i}{t_{i+1} - t_i} + \\
+ \mu_i(x)(P(x - \tilde{x}(t_i), t_i)) \\
is an SOS polynomial \( \forall i = 1, \ldots, N-1 \)
\]

where \( \varepsilon > 0 \) and \( \mu_1, \ldots, \mu_N \) are real polynomials. Note that the constraint in \([3] \) is bilinear in \( \mu \) and \( \rho \), and thus an algorithm for solving \([3] \) iteratively alternates between solving SOS program \([3] \) for multipliers \( \mu \) and \( \varepsilon \) with fixed \( \rho \), and solving SOS program \([3] \) for \( \rho \) with fixed \( \mu \) and \( \varepsilon \) \([29, 31]\). This also requires a valid initial funnel as described in \([31]\).

SOS programming, while a convex optimization problem, is computationally demanding, can encounter numerical problems, and scales poorly to high dimensions \([16, 23] \). In particular, an SOS polynomial constraint in \([3] \) can be reformulated as \([20]\)

\[
\varepsilon - \dot{P}(x - \tilde{x}(t_i), t_i) + \frac{\rho_{i+1} - \rho_i}{t_{i+1} - t_i} + \\
+ \mu_i(x)(P(x - \tilde{x}(t_i), t_i)) = z(x)^T Q z(x),
\]

where \( z(x) \) is a vector of monomials up to degree \( d \), and \( Q \) is an unknown semidefinite matrix with \( \binom{n+d}{d} \times \binom{n+d}{d} \approx n^{2d} \) elements provided that polynomial on the left hand side is of degree \( 2d \) \([16]\). Since polynomials are equal only if their coefficients are equal, constraint \([4] \) can be replaced with \( \binom{n+d}{d} \times \binom{n+d}{d} \) equalities (coefficient matching conditions \([16]\)) and one semidefinite matrix constraint \( Q \geq 0 \). Hence, the states \( x \) are removed from the optimization, but a new semidefinite matrix variable \( Q \) is introduced, which causes the aforementioned scalability issues in SOS programming \([16]\).

Moreover, the approach requires repeated solving of \([3] \) to perform optimization over \( \mu \) and \( \rho \). Also note that a resulting value of \( \rho \) may not be optimal, since the problem \([3] \) is bilinear (i.e., non-convex), and the transformation to SOS is a relaxation technique. A final slight drawback of SOS relaxation is that system dynamics \( F \) \([1] \) and \( P \) must be polynomials.

## 4 Constructing \( \rho \) using numerical optimization

In this section, we describe a funnel computation algorithm that avoids the use of costly SOS programming. We propose the use of falsifiers based on numerical optimization to solve the optimization of \( \rho \) and to leave potential formal verification to the end.

Our algorithm samples the constructed funnels in time and proceeds backwards. Let us choose time instants \( 0 = t_1 < t_2 < \cdots < t_N = T \) and find for a given \( P(T) \) a value \( \rho_N \), such that \( L_{\tilde{P}(t_N), \rho_N} \dot{x}(T) \subseteq \mathcal{G} \) is as large as possible. This NLP problem can be solved for \( P(T) \) and \( \mathcal{G} \) quadratic using semidefinite programming. Next, we compute the samples \( \rho(t_1), \ldots, \rho(t_{N-1}) \) which we denote as \( \rho_1, \ldots, \rho_{N-1} \). Finally, we assume an interpolation between the samples and check condition \([2] \) for the interpolated funnel.

Three NLPs are to be solved for each time sample. The first two, NLPs \([6] \) and \([5] \), are used to provide the time sampled optimal funnel (in terms of volume). The final one \([8] \) checks condition \([2] \) that would be used
Algorithm 1 Funnel Synthesis

In: A system \( \dot{x} = F(x,t) \), a goal region \( G \), a reference trajectory \( \hat{x}(t) \) for \( t \in [0,T] \) with \( \hat{x}(T) \in G \), positive definite function \( P(x,t) \), time samples 
\[
0 = t_1 < \cdots < t_N = T,
\]
and sampling for each interval \([t_k, t_{k+1}]\).

Out: Funnel \( \mathcal{F}(t), t \in [0,T] \)

1. Let \( \rho_N \) be s.t. \( L \leq P(T) \), \( \rho_N + \hat{x}(T) \subseteq G \)

2. For \( k := N - 1, N - 2, \ldots, 1 \)
   (a) Put \( \rho_k := c \rho_{k+1} \)
   (b) Repeat until \( \tau_1 \) subsequent iterations do not change \( \rho_k \)
      • Solve \( (6) \) from a random initial point.
      • If the solution evolves outside of \( L \leq P(t_{k+1}), \rho_{k+1} + \hat{x}(t_{k+1}) \), find a solution \( x' \) to \( (5) \) and put \( \rho_k := \gamma_1 P(x', t_k) \).
   (c) Repeat until \( \tau_2 \) subsequent iterations do not change \( \rho_k \)
      • Solve \( (8) \) for the respective sampling from random initial points.
      • If the solution does not meet \( (7) \), put \( \rho_k := \gamma_2 \rho_k \)

3. return the funnel \( \mathcal{F}(t) := \{ P(x,t) \leq \rho^I(t) \} \), where \( \rho^I(t) \) is a piece-wise linear interpolation between the samples \( \rho_1, \ldots, \rho_N \)

for formal verification of the interpolated funnel. The algorithm shrinks the funnel, if any counterexample to condition \( (2) \) is found, or accepts the sampled value, if it does not.

Let us describe the algorithm more closely. Assume that we already determined the optimal value of \( \rho_{k+1} \). To determine the optimal value of \( \rho_k \), consider the NLP that seeks a point with smallest possible value \( \rho_k \) for which the system leaves \( L \leq P(t_{k+1}), \rho_{k+1} + \hat{x}(t_{k+1}) \) after evolving from \( t_k \) to \( t_{k+1} \):
\[
\begin{align*}
\min_x P(x - \hat{x}(t_k), t_k) \\
\text{subject to}
\end{align*}
\]
\[
P(\Sigma(x,t_k)(t_{k+1}) - \hat{x}(t_{k+1}), t_{k+1}) \geq \rho_{k+1}
\]
(5)

NLP \( (5) \) is non-convex, thus a local NLP solver can solve this NLP only approximately. Therefore, for reliably accepting a certain value \( \rho_k \), more needs to be done. The first step to do so is another NLP
\[
\begin{align*}
\max_x P(\Sigma(x,t_k)(t_{k+1})) \\
\text{subject to}
\end{align*}
\]
\[
P(x - \hat{x}(t_k), t_k) \leq \rho_k,
\]
(6)

that checks whether the current estimate \( \rho_k \) results in a counterexample, a state that evolves outside of \( L \leq P(t_{k+1}), \rho_{k+1} + \hat{x}(t_{k+1}) \). If the found optimum is bigger than \( \rho_{k+1} \), we found a counter-example, and hence we solve NLP \( (5) \), using the solution to NLP \( (6) \) as an initial feasible estimate. This solution gives us a new, smaller estimate for \( \rho_k \). If the found optimum is not bigger than \( \rho_{k+1} \), we cannot make a definite conclusion, since NLP \( (6) \) is again non-convex. Hence, we increase the trust in the current estimate by repeatedly solving NLP \( (6) \) from random initial points until no further counter-example is found within a certain number \( \tau_1 \) of subsequent iterations.

The use of NLP \( (6) \) has two major advantages over only iterating NLP \( (5) \) from random initial points. First, NLP \( (6) \) directly checks for the existence of a counter-example, making it more efficient for this purpose,
in our experience. And second, the result of NLP \([6]\) provides a much more useful starting point for NLP \([5]\) than random starting points.

To enforce termination of the loop between NLPs \([6]\) and \([5]\), we update \(\rho_k\) as \(\gamma_1 P(x', t_k)\), where \(x'\) is the found numerical solution of \(\[3\]\) and \(0 < \gamma_1 < 1\). The loop must terminate after finitely many iterations, since there must be a \(\rho_k\) small enough such that no counterexample exists due to continuity of solutions of ordinary differential equations wrt. their initial conditions \([11]\) and the fact that \(P(x, t)\) is positive definite. It should also be noted that, in general, we do not have \(\Sigma(x, t)\) available in explicit form, and hence we must approximate it using numerical integration.

After the end of the iteration between NLPs \([6]\) and \([5]\), we try to extend the funnel from \(t_k\) to the whole time interval \([t_k, t_{k+1}]\). For this we use linear interpolation between \(\rho_k\) and \(\rho_{k+1}\). Based on this, we would have to check condition \([2]\) for all \(t \in [t_k, t_{k+1}]\) and all \(x \in L_{p_i}(t) + \tilde{x}(t)\). However, as mentioned in \([29, 31]\), this is not convenient to check due to dependency on time \(t\). It is computationally far more efficient (for both SOS relaxation and our presented approach) to simply sample the time interval \([t_k, t_{k+1}]\) and to check the condition discretely. Moreover, continuity arguments show \([31]\) that provided that sampling is fine enough, no counterexamples to condition \([2]\) may exist.

Assume a sampling \(t_{k,1}, \ldots, t_{k,M}\) of interval \([t_k, t_{k+1}]\), and denote by \(\rho^j\) the linear interpolation between \(\rho_k\) and \(\rho_{k+1}\) used in \([29, 31]\). Then \(\rho^j(t) = \frac{\rho_{k+1} - \rho_k}{t_{k+1} - t_k} t_k + \rho_k\), and for \(j = 1, \ldots, M\), and all \(x \in L_{p_i}(t_{k,j}) \oplus \tilde{x}(t_{k,j})\), we must ensure

\[\hat{P}(x - \tilde{x}(t_{k,j}), t_{k,j}) \leq \hat{\rho}^j(t_{k,j}).\]  

Since \(\hat{P}\) tends to zero for \(x \to \tilde{x}\), we can guarantee that these conditions can be met by choosing \(\rho_k\) small enough. Thus again the resulting algorithm will succeed in finitely many iterations. However, we need to employ a line-search strategy on \(\rho_k\) to obtain an optimal funnel that meets \([7]\).

We can check condition \([7]\) by numerical optimization, namely by solving an NLP

\[
\max_x \hat{P}(x - \tilde{x}(t_{k,j}), t_{k,j})
\]

subject to

\[
P(x - \tilde{x}(t_{k,j}), t_{k,j}) = \rho^j(t_{k,j})
\]

which is again a non-convex problem. Again we ensure reliability of the check by solving the NLP repeatedly from random initial points until no more counter-examples appear. If a counter-example is found, we reduce \(\rho_k\) using a multiplier \(0 < \gamma_2 < 1\).

Algorithm \([1]\) summarizes the whole algorithm. We estimate the initial value of \(\rho_k\) as \(c \rho_{k+1}\), where \(\rho_{k+1}\) is the previous computed value. Note that \(c\) should be chosen large enough to ensure that the first estimate always contains a counterexample, and thus the first estimate is always an upper bound of the optimal funnel size.

## 5 Computational Experiments

In this section, we discuss the results of computational experiments using the method from the previous section. The implementation was done in MATLAB R2017b and ran on a PC with Intel Core i7-10700K, 3.8GHz and 32GB of RAM. We will do a comparison between the method described in the previous section and the SOS method described in \([31]\). The NLP solver used for our method and for generation of reference trajectories was implemented in CasADi \([5]\] with internal NLP solver ipopt \([32]\). The SOS method was implemented in Yalmip \([13]\] with internal SDP solver Mosek \([4]\).

### 5.1 Example 1: Inverted pendulum

We start with a simple two dimensional problem, an inverted pendulum, and continue with more involved examples later. The dynamics of the inverted pendulum are

\[
\dot{\theta} = \frac{g}{l} \sin \theta - \frac{b \dot{\theta}}{ml^2} + \frac{u}{ml^2},
\]

where we set \(m = 1, l = 0.5, g = 9.81, b = 0.1\). Assume the task of steering an inverted pendulum to its unstable equilibrium \(\tilde{x} = 0\). First, we computed a stabilizing reference trajectory of length \(T = 3\) with step
0.05 using CasADi, see Figure 1a. Next, we constructed an LQR tracking controller for the interpolated reference (piecewise cubic in states and piecewise linear in control) by solving the Riccati equation for $R = Q = I$ with a final value of a cost-to-go matrix $S(T) = Q$ using the RKF45 integrator with a maximum step 0.005 and used again cubic interpolation, and hence we obtained matrices $S(t)$ for $t \in [0, T]$. We set $\mathcal{G} = \{x \in \mathbb{R}^2 | (x - \tilde{x}(T))^T(x - \tilde{x}(T)) \leq 0.0025\}$ as our target set.

For the SOS method, we set an initial feasible funnel as $\rho(t) = 0.0025 \exp(0.6 \cdot T - t)$ using the template from [31]. Additionally, we approximated the non-polynomial dynamics with cubic Taylor polynomials and set $\mu$ to be quadratic. We did not use polynomials of higher degree due to numerical problems encountered by the SDP solver. We terminated the SOS algorithm, if the volume of the funnel increased less than 0.1% between two subsequent iterations or if the solver failed due to numerical errors. In our method, we set $\gamma_1 = 0.9999$, and $\gamma_2 = 0.999$, and the iteration bounds $\tau_1 = 10$ and $\tau_2 = 30$ and computed funnels for both the original and the polynomial model. We used just one sample for each interval ($t_{k+1}$ for the interval $[t_k, t_{k+1}]$) in evaluating derivatives for both methods.

The results can be seen in Figure 2 and Table 1. The SOS method is faster and SOS funnels are larger for the last time intervals. However, the computed funnels are incorrect in the sense that they do not relate to an actual funnel as we have tested via numerical simulations of the polynomial system. The reason for this problem is the fact that the derivatives are checked too sparsely for this example. Falsifier based funnels do not suffer from this in this example since funnel sizes are also estimated from above by numerical integration, not just by derivatives alone. Hence, sparser sampling is needed to obtain an accurate estimate of the actual funnel in comparison to the SOS method. We also tried SOS to verify the computed funnels and while the solver did not falsify them, the solver reported some solutions to not to be reliable. In terms of volume, the cubic Taylor approximation underestimates the funnel size and this example would require a significantly higher (about 7 according to our method) degree polynomial to accurately describe the actual funnel.
5.2 Example 2: Quadcopter

Let us consider a twelve dimensional problem. We assume the quadcopter model (2.30) – (2.35) in [10] for unit mass and the obstacle avoidance manoeuvre from Figure 1b computed using CasADi. We again constructed an LQR tracking controller for the interpolated discrete trajectory (piecewise cubic in states and piecewise linear in control) by solving the Riccati equation for $Q = 10I$ and $R = I$ with final value of a cost-to-go matrix $S(T) = Q$ using the RKF45 integrator with a maximum step 0.001 and again used cubic interpolation, and hence we obtained matrices $S(t)$ for $t \in [0, T]$. We set $G = \{ x \in \mathbb{R}^{12} \mid (x - \tilde{x}(T))^T (x - \tilde{x}(T)) \leq 0.1 \}$ as our target set.

For the SOS method, we set an initial feasible funnel as $\rho(t) = 0.1 \exp(-1.94 \cdot \frac{T - t}{T})$. We approximated non-polynomial dynamics with cubic Taylor polynomials, that accurately approximates the original dynamics, and set $\mu$ to be quadratic. We terminated the SOS algorithm, if the volume of the funnel increased less than 0.1% between two subsequent iterations. In our method, we again set $\gamma_1 = 0.9999$, $\gamma_2 = 0.999$, $\tau_1 = 10$ and $\tau_2 = 30$. We again considered just one sample for each interval $(t_{k+1}$ for the interval $[t_k, t_{k+1}]$) in evaluating derivatives for both methods.

The results can be seen in Figure 3 and Table 1. Notice that, the falsifier based and SOS method actually (except for $h = 0.1$) provided very similar solutions which demonstrates the accuracy of our method. However, the SOS method is significantly slower here ($20 \times - 50 \times$). This shows lower of scalability of SOS programming in the problem dimension [10]. But we can exploit the similarity of both solutions to significantly speed up the SOS method by combining both methods initializing the SOS method by the result of ours. To illustrate this, we ran the SOS method with the initial funnel $\exp(-0.15 \cdot \frac{T - t}{T}) \rho(t)$, where $\rho(t)$ was the funnel computed by our method, see Table 1.

5.3 Example 3: Pendulum revisited

Let us return to a pendulum example, where we explore our method on problems of higher dimensions parametric in $n$. We assume a model of an $n$-link pendulum with $g = 9.81$ and we set the other parameters (all weights and lengths) to 1. The derivation of equations of motion can be found in [14]. The equation can be written in manipulator form

$$M(\theta, \dot{\theta}) \ddot{\theta} + G(\theta, \dot{\theta}) = u, \quad (10)$$
Figure 3: Quadcopter: Funnels for falsifier based (full line), and SOS based (dashed line), and SOS + falsifier based (stars) methods for time steps $h = 0.1, 0.05, 0.03$ (blue, red, green).

Table 2: $n$-link pendulum:

| $n$ | Original $t$ | Linear $t$ | SOS $t$ | Original $t$ | Linear $t$ | SOS $t$ |
|-----|--------------|------------|---------|--------------|------------|---------|
| 1   | 13.3         | 13.6       | 0.3     | 11           | 274.3      | 30.8    | 12378.2 |
| 2   | 17.5         | 16.9       | 0.6     | 12           | 379.9      | 33.0    |         |
| 3   | 19.7         | 18.2       | 1.8     | 13           | 443.1      | 34.4    |         |
| 4   | 26.5         | 20.4       | 7.2     | 14           | 752.4      | 36.4    |         |
| 5   | 40.5         | 23.9       | 31.1    | 15           | 1044.6     | 38.8    |         |
| 6   | 50.1         | 23.8       | 87.4    | 16           | 1326.5     | 41.3    |         |
| 7   | 66.6         | 23.9       | 353.5   | 17           | 1819.2     | 45.4    |         |
| 8   | 90.9         | 25.4       | 840.5   | 18           | 2366.7     | 49.6    |         |
| 9   | 128.1        | 26.8       | 2302.3  | 19           | 3046.4     | 50.6    |         |
| 10  | 185.2        | 28.7       | 5635.9  | 20           | 3990.7     | 53.4    |         |

where we assume that $u \in \mathbb{R}^n$ is a control input. Next, we construct a nonlinear stabilizing controller for $n$-link pendulum

$$u(\theta, \dot{\theta}) = \left[ I, M(\theta, \dot{\theta}) \right]^T K \left( \left[ \theta, \dot{\theta} \right]^T - \left( \frac{\pi}{2}, 0, \ldots, 0 \right)^T \right),$$

where $I$ is an identity matrix and $K$ is a gain matrix of the LQR controller based on a slightly simpler model $\ddot{\theta} + M(\theta, \dot{\theta})^{-1}G(\theta, \dot{\theta}) + u = 0$ linearized around the pendulum-upwards equilibrium $(\frac{\pi}{2}, 0, \ldots, 0)$.

We compute funnels for stabilization of the $n$-link pendulum with the derived controller for $Q_n = 10nI$ and $R = 1$. We set a target $G_n = \left\{ \left[ \theta, \dot{\theta} \right]^T \in \mathbb{R}^{2n} \mid \Delta^T S_n \Delta \leq \rho_n \right\}$, where $\Delta = \left( \left[ \theta, \dot{\theta} \right]^T - \left( \frac{\pi}{2}, 0, \ldots, 0 \right)^T \right)$, and $S_n$ is a cost-to-go matrix of the LQR controller, and where $\rho_n$ is chosen in such a way that the volume of $G_n$ is the same as the volume of a hypersphere with radius of $r^2 = 0.025$ in $2n$ dimensions. Notice that the whole problem is time-invariant since the chosen system trajectory around which we will construct a funnel is constant $\hat{x}(t) = (\frac{\pi}{2}, 0, \ldots, 0)^T$ as well as the shape $S(t) = S_n$.

We again set for our method $\gamma_1 = 0.9999$, $\gamma_2 = 0.999$. Moreover, we used $\tau_1 = 10$ and $\tau_2 = 50$. And we again considered just one sample for each interval $(t_{k+1}$ for the interval $[t_k, t_{k+1}]$ in evaluating derivatives.

We computed funnels of length $T = 1$ with a time step 0.025 for $n = 1, \ldots, 20$, i.e., for state dimensions...
E is the ellipsoid of maximal value with the matrix $e^n$ up to 40. For a comparison, we tested our method on the original model and its linearization. The results can be seen in Table 2. As can be seen from the results, the funnels were successfully computed for all $n$. However, the required computational time increases steadily for the original model, which is mostly caused by the fact that system dynamics become more and more complex with each link added, which steadily increases computational time required for evaluation of system dynamics and its first and second order derivatives.

It should be noted however that the computational time remained much more reasonable for the linearized model where this increase in complexity naturally does not occur. This shows that our method can work reasonably well even in high dimensions provided that a model dynamics are not too complicated. To illustrate scalability of the SOS method, we computed one iteration of the SOS method on the linearized dynamics with $\mu$ set as quadratic. As can bee seen in Table 2, the computational time increases dramatically and becomes impractical with about 9 links.

For the linearized model, we can compare the computed values of $\rho$ with the true optimal values. These can be computed directly for linear systems with an ellipsoidal funnel using the state transition matrix. Assume a linear system $\dot{x} = Ax$ and an ellipsoid $E = \{x \in \mathbb{R}^n \mid x^TQ^{-1}x \leq 1\}$. Using and affine mapping with the matrix $e^{At}$, this ellipsoid transforms into the ellipsoid

$$E(t) = \left\{ x \in \mathbb{R}^n \mid x^T\left(e^{At}Qe^{At^T}\right)^{-1}x \leq 1 \right\}$$

after time $t$. Hence, the optimal value $\rho$ for a funnel with a given shape $S$ in time $t$ that ends in the ellipsoid $E$ is the ellipsoid of maximal value $\rho$ for which $\{x \in \mathbb{R}^n \mid x^TSx \leq \rho\} \subseteq E(-t)$.

We computed the optimal values of $\rho$ with a time step 0.025 iteratively. The comparison of the resulting values of $\rho(0)$ can be seen in Table 3. The values computed for the linearized model are slightly lower than the optimal ones. This underestimation of the funnels is largely caused by the derivative check (DC) that assumes a piece-wise linear $\rho$, which the optimal $\rho$ is not. If the derivative check is skipped, our results and the optimal values are nearly identical.

| $n$ | Original $\rho(0)$ | Linear, DC $\rho(0)$ | Linear, no DC $\rho(0)$ | Linear, optimal $\rho(0)$ |
|-----|--------------------|----------------------|-------------------------|--------------------------|
| 1   | 18.48              | 15.48                | 15.48                   | 15.54                    |
| 2   | 6.24               | 6.13                 | 6.13                    | 6.16                     |
| 3   | 1.17               | 1.17                 | 1.17                    | 1.17                     |
| 4   | 1.26               | 1.26                 | 1.31                    | 1.32                     |
| 5   | 1.70               | 1.70                 | 1.77                    | 1.78                     |
| 6   | 1.79               | 1.79                 | 1.94                    | 1.95                     |
| 7   | 1.96               | 1.97                 | 2.13                    | 2.14                     |
| 8   | 2.05               | 2.06                 | 2.32                    | 2.34                     |
| 9   | 2.24               | 2.26                 | 2.55                    | 2.56                     |
| 10  | 2.36               | 2.38                 | 2.79                    | 2.81                     |
| 11  | 2.57               | 2.59                 | 3.04                    | 3.06                     |
| 12  | 2.77               | 2.80                 | 3.30                    | 3.32                     |
| 13  | 2.87               | 2.91                 | 3.56                    | 3.58                     |
| 14  | 3.06               | 3.12                 | 3.81                    | 3.83                     |
| 15  | 3.25               | 3.33                 | 4.07                    | 4.09                     |
| 16  | 3.45               | 3.40                 | 4.32                    | 4.34                     |
| 17  | 3.50               | 3.59                 | 4.57                    | 4.59                     |
| 18  | 3.69               | 3.80                 | 4.82                    | 4.85                     |
| 19  | 3.88               | 4.00                 | 5.08                    | 5.10                     |
| 20  | 4.07               | 4.20                 | 5.34                    | 5.37                     |

Table 3: $n$-link pendulum: value $\rho(0)$ for the original model and its linearization (with and without derivative check (DC)), and the optimal value of $\rho(0)$ for the linearized model computed via the state transition matrix.
6 Conclusion

In this paper, we presented an algorithm that computes funnels along trajectories of systems of ordinary differential equations. Compared to related work based on SOS programming, in our computational experiments, the algorithm computed larger funnels in less time. The algorithm does not formally verify in itself, but its result can then be formally verified using a well-known palette of verification techniques that includes—in addition to SOS programming—computer algebra [9] or interval computation [21]. In addition, these funnels could be used to initialize the SOS method as illustrated by Example 2.

References

[1] Amir Ali Ahmadi and Anirudha Majumdar. DSOS and SDSOS optimization: More tractable alternatives to sum of squares and semidefinite optimization. SIAM Journal on Applied Algebra and Geometry, 3(2):193–230, 2019.

[2] Matthias Althoff and John M. Dolan. Online verification of automated road vehicles using reachability analysis. IEEE Transactions on Robotics, 30(4):903–918, 2014.

[3] N. H. Amer, H. Zamzuri, K. Hudha, and Z. A. Kadir. Modelling and control strategies in path tracking control for autonomous ground vehicles: A review of state of the art and challenges. Journal of Intelligent & Robotic Systems, 86(2):225–254, 2017.

[4] Erling D Andersen and Knud D Andersen. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In High performance optimization, pages 197–232. Springer, 2000.

[5] J. A. E. Andersson, J. Gillis, G. Horn, James B Rawlings, and Moritz Diehl. CasADi— A software framework for nonlinear optimization and optimal control. Mathematical Programming Computation, 11(1), 2018.

[6] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. Linear matrix inequalities in system and control theory. SIAM, 1994.

[7] Y. Chang, N. Roohi, and S. Gao. Neural Lyapunov control. In Advances in Neural Information Processing Systems, volume 32, pages 3245–3254. Curran Associates, Inc., 2019.

[8] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear optimization and optimal control. Journal of Symbolic Computation, 12:299–328, 1991.

[9] Luis Rodolfo García Carrillo, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard. Modeling the quad-rotor mini-rotorcraft. In Quad Rotorcraft Control, pages 23–34. Springer, 2013.

[10] H. K. Khalil. Nonlinear Systems. Prentice Hall, 3rd edition, 2002.

[11] D. Liberzon. Calculus of Variations and Optimal Control Theory: A Concise Introduction. Princeton University Press, New Jersey, 2011.

[12] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In Proceedings of the CACSD Conference, Taipei, Taiwan, 2004.

[13] A. M. Lopes and J. A. T. Machado. Dynamics of the n-link pendulum: a fractional perspective. International Journal of Control, 90(6):1192–1200, 2015.

[14] Anirudha Majumdar, Georgina Hall, and Amir Ali Ahmadi. Recent scalability improvements for semidefinite programming with applications in machine learning, control, and robotics. Annual Review of Control, Robotics, and Autonomous Systems, 3(1):331–360, 2020.
[17] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[18] J. Moore and R. Tedrake. Control synthesis and verification for a perching UAV using LQR-trees. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3707–3714, 2012.

[19] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. Comput.*, 105:21–68, 1999.

[20] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96:293–320, 2003.

[21] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

[22] H. Ravanbakhsh, S. Aghli, C. Heckman, and S. Sankaranarayanan. Path-following through control funnel functions. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 401–408, 2018.

[23] H. Ravanbakhsh and S. Sankaranarayanan. Learning control Lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, 43(2):275–307, 2019.

[24] Taylor Reynolds, Danylo Malyuta, Mehran Mesbahi, Behcet Acikmese, and John M Carson. Funnel synthesis for the 6-dof powered descent guidance problem. In *AIAA Scitech 2021 Forum*, page 0504, 2021.

[25] Spencer M Richards, Felix Berkenkamp, and Andreas Krause. The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, 44(1):466—476, 2018.

[26] B. Rubí, R. Pérez, and B. Morcego. A survey of path following control strategies for UAVs focused on quadrotors. *Journal of Intelligent & Robotic Systems*, 98(2):241–265, 2020.

[27] W. Tan and A. Packard. Searching for control Lyapunov functions using sums of squares programming. *Allerton conference on communication, control and computing*, pages 210–219, 2004.

[28] J. Z. Tang, A. M. Boudali, and I. R. Manchester. Invariant funnels for underactuated dynamic walking robots: New phase variable and experimental validation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3497–3504, 2017.

[29] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[30] T. G. Thuruthel, Y. Ansari, E. Falotico, and C. Laschi. Control strategies for soft robotic manipulators: A survey. *Soft Robotics*, 5(2):149–163, 2018.

[31] Mark M. Tobenkin, Ian R. Manchester, and Russ Tedrake. Invariant funnels around trajectories using sum-of-squares programming. *IFAC Proceedings Volumes*, 44(1):9218–9223, 2011.

[32] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[33] A. Weiss, C. Danielson, K. Berntorp, I. Kolmanovsky, and S. Di Cairano. Motion planning with invariant set trees. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1625–1630, 2017.