Verification and Validation of Convex Optimization Algorithms for Model Predictive Control

Raphaël Cohen, Eric Féron, Pierre-Loïc Garoche

Verification and Validation of Convex Optimization Algorithms for Model Predictive Control

Raphael Cohen* and Eric Feron †
Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA 30332

Pierre-Loïc Garoche‡
Onera – The French Aerospace Lab, Toulouse, 31000 France

Advanced embedded algorithms are growing in complexity and they are an essential contributor to the growth of autonomy in many areas. However, the promise held by these algorithms cannot be kept without proper attention to the considerably stronger design constraints that arise when the applications of interest, such as aerospace systems, are safety-critical. Formal verification is the process of proving or disproving the "correctness" of an algorithm with respect to a certain mathematical description of it by means of a computer. This article discusses the formal verification of the Ellipsoid method, a convex optimization algorithm, and its code implementation as it applies to receding horizon control. Options for encoding code properties and their proofs are detailed. The applicability and limitations of those code properties and proofs are presented as well. Finally, floating-point errors are taken into account in a numerical analysis of the Ellipsoid algorithm. Modifications to the algorithm are presented which can be used to control its numerical stability.

Nomenclature

$\mathbb{R}$	=	The set of all real numbers.
$\mathbb{F}$	=	Set of floating-point numbers.
$\mathbb{R}^+$	=	The set of all positive real numbers.
$\mathbb{R}^*_+$	=	The set of all strictly positive real numbers.
$\mathbb{R}^n$	=	The set of real vectors of length n.
$\mathbb{R}^{m\times n}$	=	The set of real matrices of size $m \times n$.
$\|A\|_F$	=	Frobenius norm of a matrix A.
$\|A\|$	=	Two norm of a matrix A.
$\|x\|$	=	Two norm of a vector x.
B_n	=	n-dimensional unit Euclidean ball. $B_n = \{z \in \mathbb{R}^n : \|z\| \leq 1\}$.
$B_r(x)$	=	Ball of radius r centered on x. $B_r(x) = \{z \in \mathbb{R}^n : \|z - x\| \leq r\}$.
$\text{Ell}(B, c)$	=	Ellipsoid set defined by: $\text{Ell}(B, c) = \{Bu + c : u \in B_n\}$.
fl()	=	Floating-point rounding of a given real number.
H	=	Model Predictive Control Horizon.
$k(A)$	=	Condition number of a matrix A.
N	=	Number of iterations.
u	=	Plant Input.
$\mathbf{u}$	=	Collection of input vectors to horizon: $\mathbf{u} = [u_1 \dots u_{H-1}]$.
Vol()	=	Volume of a given set.
X	=	Original Decision Vector for an Optimization problem.
x	=	Plant State Vector.
$\mathbf{x}$	=	Collection of state vectors to horizon: $\mathbf{x} = [x_1 \dots x_H]$.
X_f	=	Feasible set of an optimization problem.
X_ϵ	=	Epsilon optimal set of an optimization problem.

*Ph.D. Student, School of Aerospace Engineering, raphael.cohen@gatech.edu.
†Professor of Aerospace Engineering, School of Aerospace Engineering, feron@gatech.edu.
‡Research Scientist, Onera, pierre-loic.garoche@onera.fr.

Z	=	Projected Decision Vector for an Optimization problem.
γ	=	Upper bound of Reduction Ratio.
θ	=	Elevation Angle in Radians for the three degrees of freedom Helicopter.
λ	=	Ellipsoids Widening Coefficient.
$\sigma_{\max}(A)$	=	Largest singular value of a matrix A.
$\sigma_{\min}(A)$	=	Smallest singular value of a matrix A.
ϕ	=	Travel Angle in Radians for the three degrees of freedom Helicopter.
ψ	=	Pitch Angle in Radians for the three degrees of freedom Helicopter.

I. Introduction

Formal verification of optimization algorithms used online within control systems is the sole focus of this research. Recently, such algorithms have been used online with great success for the guidance of safety-critical applications, including, autonomous cars [1] and reusable rockets [2]. The latter case has resulted in technology demonstrations such as the landings of SpaceX's Falcon 9 [3] and BlueOrigin's New Shepard. Thus, algorithms solving optimization problems are already used online, have been embedded on board, and yet still lack the level of qualification required by civil aircraft or manned rocket flight. Automatic code generation for solving convex optimization problems has already been done [4, 5], but does not include the use of formal methods. Likewise, work within the field of model predictive control already exists where numerical properties of algorithms are being evaluated [6]. Nevertheless, this work is only valid for Quadratic Programming (See Section II) and using fixed-point arithmetic. As well, no formal verification is performed. On the other hand, some contributions have been made concerning formal verification of control systems [7–10], but they mainly focus on formal verification and code generation for linear control systems. Research has also been made toward the verification of numerical optimization algorithms [11, 12], yet it remains purely theoretical and no proof was obtained using formal verification tools. Contributions on formal verification of optimization algorithms have already been made [13], but this work focuses on a single optimization problem, where closed-loop behaviors are not being addressed, which does not meet the level of guarantees needed for receding horizon controllers. As well, the formal proof was not complete and no numerical analysis was presented.

The need for enhanced safety and better performance is currently pushing for the introduction of advanced numerical methods into the next generation of cyber-physical systems. While most of the algorithms described in this article have been established for a long time, their online use within embedded systems is relatively new and introduces issues that have to be addressed. Among these methods, this study focuses on numerical optimization algorithms.

The following scientific contributions are presented:

- axiomatization of optimization problems and formalization of algorithm proof (Ellipsoid method) as code annotation
- extraction of guarantees of convergence for sequential optimization problems, representing closed-loop management
- modification of the original algorithm to account for floating-point errors
- generation of C code implementations via credible autocoders of receding horizon controllers along with ANSI/ISO C Specification Language (ACSL) annotations

The choice of ellipsoid method here seems unconventional as current state of art solvers typically use some variant of the interior-point method. However it has been shown in [13] that guaranteeing the numerical accuracy of second-order methods are very challenging. This paper is a first attempt at providing methods and tools to formally verify convex optimization code for solving online receding-horizon control problems. The article is structured as follows: Section II presents backgrounds for convex optimization, model predictive control, and axiomatic semantics using Hoare triples. Section III focuses on the axiomatization of second-order cone programs and the formal verification of the Ellipsoid Method. Furthermore, the closed-loop management and the online aspect of the developed algorithm is discussed in Section IV. A modified version of the original Ellipsoid Method used to control floating-point errors, is presented in Section V. A floating-point analysis of the ellipsoid method is presented in Section VI, while Section VII presents how this framework can be automated and applied to a system, the three degrees of freedom (DOF) Helicopter. Finally, Section VIII concludes this article.

II. Preliminaries

A. Second-Order Cone Programming

Optimization algorithms solve a constrained optimization problem, defined by an objective function and a set of constraints to be satisfied:

$$\begin{aligned} \min \quad & f_o(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i \text{ for } i \in [1, m] \end{aligned} \tag{1}$$

This problem searches for $x \in \mathbb{R}^n$, the optimization variable, minimizing $f_o \in \mathbb{R}^n \to \mathbb{R}$, the objective function, while satisfying constraints $f_i \in \mathbb{R}^n \to \mathbb{R}$, with associated bounds b_i. An element of $\mathbb{R}^n$ is feasible when it satisfies all the constraints f_i. An optimal point is defined by the element having the smallest cost value among all feasible points. An optimization algorithm computes an exact or approximated estimate of the optimal cost value, together with one or more feasible points achieving this value. A subclass of these problems that can be efficiently solved are convex optimization problems. In these cases, the functions f_o and f_i are required to be convex [4], with one of the consequences being that a local minimizer is also a global minimizer. Furthermore, when the constraints are linear the problem is either called a Linear Program (LP) if the cost is also linear or called a Quadratic Program (QP) if the cost is quadratic. Optimization problems where both the cost and the constraints are quadratic are called Quadratically Constrained Quadratic Program (QCQP). Semi-Definite Programs (SDP) represent problems that have constraints which can be formulated as a Linear Matrix Inequality (LMI). Here, only a specific subset of convex optimization problems are presented in details: Second-Order Cone Programs (SOCPs). For $x \in \mathbb{R}^n$, a SOCP in standard form can be written as:

$$\begin{aligned} \min \quad & f^T x \\ \text{s.t.} \quad & \|A_i x + b_i\|_2 \ \leq \ c_i^T x + d_i \text{ for } i \in [1\ ,\ m] \\ \text{With: } & f \in \mathbb{R}^n,\ A_i \in \mathbb{R}^{n_i \times n},\ b_i \in \mathbb{R}^{n_i},\ c_i \in \mathbb{R}^n,\ d_i \in \mathbb{R}. \end{aligned} \tag{2}$$

A classification of the most common convex optimization problems is presented in Fig. 1. Frequently, optimization problems that are used online for control systems can be formulated as a SOCP. Furthermore, extensions to SDPs are possible with little additional work. The algorithm used and the proof are still valid for any convex problem.

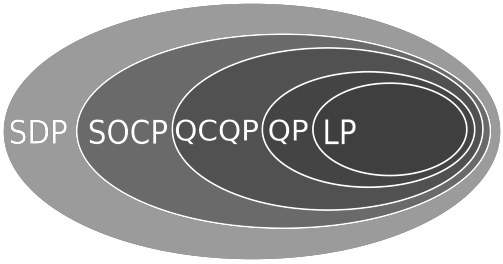


Figure 1 Classification of Some Convex Optimization Problems

B. Model Predictive Control (MPC)

Model predictive control (also known as receding horizon control) is an optimal control strategy based on numerical optimization. In this technique, a discrete-time dynamical model of the system is being used to predict potential future trajectories. As well, a cost function, that depends on the future control inputs and states, is being considered over the receding prediction horizon H, with the objective being to minimize this cost. At each time t, a convex optimization problem is being solved and the corresponding input is sent to the system. A time step later, the exact same process occurs and is repeated until a final time. We refer the reader to [14, 15] for more details on MPC.

C. Axiomatic Semantics and Hoare Logic

Semantics of programs express their behavior. Using axiomatic semantics, the program's semantics can be defined in an incomplete way, as a set of projective statements, i.e., observations. This idea was formalized by [16] and then [17] as a way to specify the expected behavior of a program through pre- and post-conditions.

Hoare Logic. A piece of code C is axiomatically described by a pair of formulas (P, Q) such that when P holds before executing C, then Q should be valid after its execution. This pair acts as a contract for the function and (P, C, Q) is called a Hoare triple. In most uses, P and Q are expressed in first order formulas over the variables of the program. Depending on the level of precision of these annotations, the behavior can be fully or partially specified. In our case we are interested in specifying, at code level, algorithm specific properties such as the algorithm convergence or preservation of feasibility for intermediate iterates. Software analyzers, such as the Frama-C platform [18], provide means to annotate source code with these contracts, and tools to reason about these formal specifications. For the C language, ACSL [19] (ANSI C Specification Language) can be used to write source comments.

Figure 2 shows an example of a function contract expressed in ACSL. The "ensures" keyword expresses all the properties that will be true after the execution of the function, assuming that all the properties listed within the "requires" keywords were true before the execution (similar to a Hoare triple). As well, it is possible to annotate and check the part of the memory assigned by a function using the keyword "assigns". In the case shown in Fig. 2, the function is not assigning anything during its execution, and therefore no global variable were changed. Throughout this article, the verification is performed using the software analyzer Frama-C and the SMT solver (Satisfiability Modulo Theories) Alt-Ergo [20], via the Weakest Precondition (WP) plug-in. The role of the WP plug-in is to implement a weakest precondition calculus for ACSL annotations present at code level. For each annotation, the WP plug-in generates proof obligations (mathematical first-order logic formulas) that are then submitted to Alt-Ergo. Further information about the WP plug-in can be found in [21].

ACSL + C

```
/*@
  @ requires -2 <= x <= 2;
  @ ensures \result == x*x;
  @ ensures 0 <= \result <= 4;
  @ assigns \nothing;
*/
double square(double x){
  return x*x;
}
```

Figure 2 ACSL Function Contract

III. Formal Verification of an Ellipsoid Method C code Implementation

Our goal is to build a framework that is capable of compiling the high-level requirements of online MPC solvers into ACSL augmented C code which can then be automatically verified using existing formal methods tools for C programs such as Frama-C and Alt-ergo. The MPC solver shall take a parameterized SOCP problem as inputs and always outputs a solution that is both feasible and epsilon-optimal within a predefined number of iterations.

This kind of requirement has never been formalized before. Hence it also has never been verified by the state of art automatic formal methods techniques and it is not really possible to do so without going into some manual proofs. To formalize these high-level requirements, one has to:

- formalize in ACSL the low-level types, such as vectors and matrices
- formalize in ACSL second-order cone problems
- formalize the solver (from algorithm and input problem to the generated C code)
- formalize the properties of the ellipsoid method in a way such that they can be expressed as axiomatic semantic of the C program (ACSL types, axioms, functions to express feasability and optimality).

For example, mathematical types from linear algebra can be defined axiomatically. The input problem and solver choice is automatically transformed into a C program. The high-level properties (feasability and optimality) of the chosen solver together with the input problem are compiled into an ACSL form (expressing the axiomatics semantics of the generated C program), and then inserted into the C program as comments. The various artifacts (ACSL types, functions, predicates, axioms, lemmas, theorems), that were manually written to support the compilation of the high-level requirements (HLR) into C+ACSL and its automatic verification, are packaged into libraries. Examples of these artifacts include types like matrix, vector, optim, predicates like "isFeasible" and functions such as "twoNorm", etc.

A. Semantics of an Optimization Problem

The first work to be done is the formal definition of an optimization problem. In order to do so, new mathematical types, objects, axioms and theorems are created. Our goal is to axiomatize optimization problems with enough properties allowing us to state all the needed optimization-level properties at code level. Let us consider the second-order cone program, described in Eq. (2).

Encoding an SOCP. In order to fully describe an SOCP, we use the variables:

$$f \in \mathbb{R}^n, \quad A = \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}, \quad C = \begin{bmatrix} c_1^T \\ \vdots \\ c_m^T \end{bmatrix}, \quad d = \begin{bmatrix} d_1 \\ \vdots \\ d_m \end{bmatrix} \text{ and also the vector } m = \begin{bmatrix} n_1 & \dots & n_m \end{bmatrix}.$$

The vector m is used to collect the sizes of the vectors $A_i \cdot x + b_i$. Furthermore, if $\sum_{i=1}^{m} n_i = 0$, then the SOCP (2) is an LP. Using ACSL, a new type and a high level function are defined, providing the possibility to create objects of the type "*optim*". Figure 3 represents an extract of the ACSL optimization theory. First, a new ACSL theory is created using the keyword "axiomatic". The keyword "logic" is used to define a new function and its signature. Information about the part of memory used by a function is provided using the keyword "reads". Figure 3 presents the definition of 2 functions. The function $socp\_of\_size\_2\_6\_0$ is used to instantiate objects representing an optimization problem of appropriate sizes.

The function *constraints* returns a vector collecting the values of all the constraint functions for a given problem and point. When applying a method to solve an actual optimization problem, many concepts are crucial. The work here is to highlight the parts of the HLR that need to be formalized (via axiomatization) and packaged into a library to support the automatic compilation of the HLR into ACSL augmented C code. The concepts of feasibility and optimality are being axiomatized. For this, given a second-order cone program, an axiomatic definition is given for the vector constraint, the gradient of a constraint, the cost, optimal point (making the assumption that it exists and is unique), etc. For instance, Fig. 4 illustrates the axiomatization of a constraint calculation and the feasibility predicate definition.

ACSL

```
/*@
axiomatic OptimSOCP {
type optim;
logic optim socp_of_size_2_6_0(
   matrix A,vector b,matrix C,
   vector d, vector f, int* m)
   reads m[0..5];
logic vector constraints(optim OPT,
   vector x);
*/
```

Figure 3 ACSL Optim Type Definition

For the constraint calculation, two axioms are defined representing two different cases: The case where the constraint is linear and the case where it is not. The predicate shown in Fig. 4 defines that a point is feasible if all the components of its constraint vector are negative. When instantiating an object of type vector or matrix, the size of the considered

ACSL

```
/*@
  axiom constraint_linear_axiom:
    \forall optim OPT, vector x, integer i;
      getm(OPT)[i] == 0 ==>
        constraint(OPT, x, i) ==
          -scalarProduct(getci(OPT,i),x,size_n(OPT))-getdi(OPT,i);
  axiom constraint_socp_axiom:
    \forall optim OPT, vector x, integer i;
      getm(OPT)[i] != 0 ==>
        constraint(OPT, x, i) ==
          twoNorm(vector_affine(getAi(OPT,i),x,getbi(OPT,i))) -
          scalarProduct(getci(OPT,i),x,size_n(OPT))-getdi(OPT,i);
  ...
  predicate
    isFeasible(optim OPT,vector x) = isNegative(constraints(OPT,x));
*/
```

Figure 4 ACSL Feasible Predicate Definition

object needs to be known since it is hard-coded in the ACSL axiomatization. This is not an issue since at this time (post parsing), all the sizes of the variables used are already defined. The objects sizes only depend on the plant's order and the horizon. Thus, as long as the order of the plant and the horizon do not change dynamically, the variables sizes can be predicted. Also, working with predefined and hard-coded size objects will help the analyzers proving the goals.

B. The Ellipsoid Method

Despite its modest efficiency with respect to interior point methods, the Ellipsoid Method [22–24] benefits from concrete proof elements and could be considered a viable option for critical embedded systems where safety is more important than performance. This section presents a way to annotate a C code implementation of the Ellipsoid Method. Before recalling the main steps of the algorithm, some mathematical preliminaries are presented.

Ellipsoids in $\mathbb{R}^n$. An ellipsoid can be characterized as an affine transformation of an Euclidean Ball. Before defining an Ellipsoid set, the definition of an Euclidean ball is first recalled.

Definition 1 (Euclidean ball) *Let $n \in \mathbb{N}$, V_n denotes the unit Euclidean ball in $\mathbb{R}^n$. $\mathrm{Vol}(V_n)$ represents its volume. As well, $B_R(x)$ is defined as the ball of radius R centered on x* $\Big($ *i.e* $\{z \in \mathbb{R}^n : (z-x)^T(z-x) \leq R\}$ $\Big)$.

Definition 2 (Ellipsoid Sets) *Let $c \in \mathbb{R}^n$ and $B \in \mathbb{R}^{n\times n}$, be a non-singular matrix ($det(B) \neq 0$). The Ellipsoid* $\mathrm{Ell}(B,c)$ *is the set :*

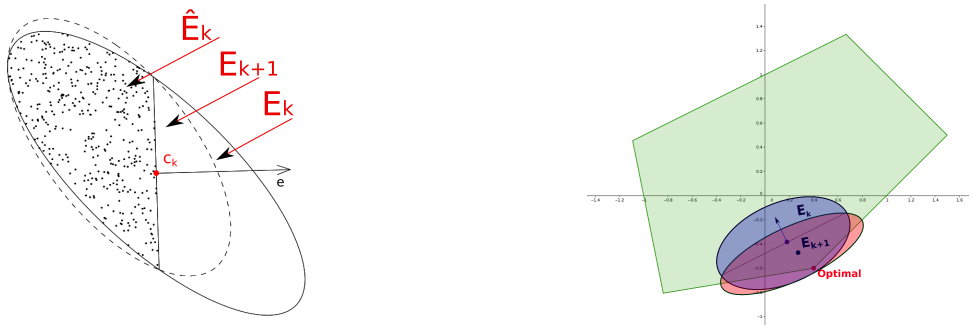
$$\mathrm{Ell}(B,c) = \{Bu + c : u^T u \leq 1\} \tag{3}$$

Definition 3 (Volume of Ellipsoids) *Let* $\mathrm{Ell}(B,c)$ *be an ellipsoid set in $\mathbb{R}^n$. $Vol(\mathrm{Ell}(B,c))$ denotes its volume and is defined as :*

$$\mathrm{Vol}(\mathrm{Ell}(B,c)) = |det(B)| \cdot \mathrm{Vol}(V_n) \tag{4}$$

Algorithm. The main steps of the algorithm detailed in [23–25] are now presented. In the following, $E_k = \text{Ell}(B_k, c_k)$ denotes the ellipsoid computed by the algorithm at the k^{th} iteration.

Ellipsoid cut. The algorithm starts with an ellipsoid containing the feasible set X, and therefore the optimal point x^*. An iteration consists of transforming the current ellipsoid E_k into a smaller volume ellipsoid E_{k+1} that also contains x^*. Given an ellipsoid E_k of center c_k, the objective is to find a hyperplane containing c_k that cuts E_k in half, such that one half is known not to contain x^*. Finding such a hyperplane is called the *oracle separation* step, *cf.* [24]. Within the SOCP setting, this cutting hyperplane is obtained by taking the gradient of either a violated constraint or the cost function. Then, the ellipsoid E_{k+1} is defined by the minimal volume ellipsoid containing the half ellipsoid $\hat{E}_k$ that is known to contain x^*. The Fig. 5a and 5b illustrate such ellipsoids cuts.



(a) Ellipsoid Cut **(b) Ellipsoid Cut In a LP Setting**

Figure 5 Ellipsoids Cut Illustration

Ellipsoid transformation. From the oracle separation step, a separating hyperplane, e, that cuts E_k in half with the guarantee that x^* is localized in $\hat{E}_k$ has been computed. The following step is the *Ellipsoid transformation*. Using this hyperplane e, one can update the ellipsoid E_k to its next iterate E_{k+1} according to Eqs. (5),(6) and (7). In addition to that, an upper bound, γ, of the ratio of $\text{Vol}(E_{k+1})$ to $\text{Vol}(E_k)$ is known.

$$c_{k+1} = c_k - 1/(n+1) \cdot B_k p \ , \tag{5}$$

$$B_{k+1} = \frac{n}{\sqrt{n^2-1}} B_k + \left(\frac{n}{n+1} - \frac{n}{\sqrt{n^2-1}} \right) (B_k p) p^T \tag{6}$$

with:

$$p = B_k^T e / \sqrt{e^T B_k B_k^T e}. \tag{7}$$

Termination. The search points are the successive centers of the ellipsoids. Throughout the execution of the algorithm, the best point so far, $\hat{x}$ is being stored in memory. A point x is better than a point y if it is feasible and has a smaller cost. When the program reaches the number of iteration needed, the best point so far, $\hat{x}$, which is known to be feasible and ϵ-optimal, is returned by the algorithm. A volume related property is now stated, at the origin of the algorithm convergence, followed by the main theorem of the method. Both properties can be found in [24, 26].

Property 1 *[Reduction ratio.] Let $k \geq 0$, by construction:*

$$\text{Vol}(E_{k+1}) \leq \exp\{-1/(2 \cdot (n+1))\} \cdot \text{Vol}(E_k) \tag{8}$$

Please find below the the proof of this property.

Proof 1 *Let us put the update formula (6) into the form:*

$$B_{k+1} = \alpha B_k + \beta (B_k p) p^T$$

With: $\alpha = n/\sqrt{n^2-1}$ and $\beta = n/(n+1) - n/\sqrt{n^2-1}$. Let us now take the determinant of both sides.

$$\det(B_{k+1}) = \det\left(B_k \cdot \left(\alpha I_n + \beta p p^T \right) \right) = \det\left(B_k \right) \det\left(\alpha I_n + \beta p p^T \right) = \det\left(B_k \right) \alpha^n \det\left(I_n + \frac{\beta}{\alpha} p p^T \right)$$

Using Sylvester's determinant identity: $\det(I_n + AB) = \det(I_m + BA) \quad \forall A \in \mathbb{R}^{n\times m}, B \in \mathbb{R}^{m\times n}$, *the determinant on the right side of the equality can be put into the form:*

$$\det(B_{k+1}) = \alpha^n \det\left(B_k\right) \cdot \left(1 + \frac{\beta}{\alpha}\|p\|\right)$$

But, from Eg. (7), one can see that $\|p\| = 1$. *Therefore:*

$$\frac{\text{Vol}(E_{k+1})}{\text{Vol}(E_k)} = \frac{|\det(B_{k+1})|}{|\det(B_k)|} = \alpha^n \cdot \left(1 + \frac{\beta}{\alpha}\right) \leq \exp\left(\frac{-1}{2(n+1)}\right)$$

□

Necessary Geometric Characteristics. In order to know the number of steps required for the algorithm to return an ϵ-optimal solution, three scalars and a point $x_c \in \mathbb{R}^n$ are needed:

- a radius R such that
$$X_f \subset B_R(x_c) \tag{9}$$
- a scalar r such that there exists a point $\bar{x}$ such that
$$B_r(\bar{x}) \subset X_f \tag{10}$$
- and another scalar V such that
$$\max_{x\in X_f} f_o - \min_{x\in X_f} f_o \leq V. \tag{11}$$

Figure 6 illustrates the scalars R and r. The Feasible set (assumed to be bounded) is shown in green. The main result can be stated as:

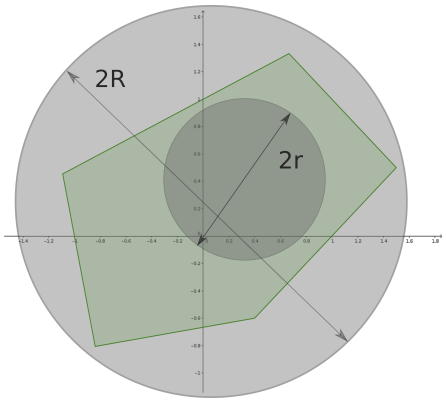


Figure 6 Included and Including Balls for the feasible set of linear constraints (shown in green)

Theorem 1 *Assuming that X is bounded, non-empty and that scalars R, r and V satisfying Eqs. (9), (10) and (11) are known. Then, for all* $\epsilon \in \mathbb{R}_+^*$, *the algorithm, using N iterations, will return* $\hat{x}$, *satisfying:*

$$f_o(\hat{x}) \leq f_o(x^*) + \epsilon \text{ and } \hat{x} \in X$$

With $N = 2n(n+1)\log\left(\frac{R}{r}\frac{V}{\epsilon}\right)$, *n being the dimension of the optimization problem.*

This result, when applied to LP, is the first proof of the polynomial solvability of linear programs. This proof can be found in [24, 26].

C. ACSL Theory

This section would be to describe all the manually written ACSL artifacts to support the automatic formalization and then verification of the HLR. Indeed, the software analyzer takes as an input the annotated C code plus ACSL theories that define new abstract types, functions, as well as axioms, lemmas and theorems. The lemmas and theorems need to be proven but the axioms are taken to be true. For the SMT solver, properties within C code as annotations are usually harder to prove than lemmas within an ACSL theory. On top on checking the mathematical correctness of an ACSL annotation, the SMT solver needs to check the soundness of the code itself (e.g., memory allocation, function calls, ACSL contracts on the function called, etc). Thus, the approach chosen here is to develop as much as possible the ACSL theories to express and prove the main results used by the algorithm. That way, the Hoare triples at code level are only an instantiation of those lemmas and are relatively simple to prove.

ACSL

```
/*@ axiomatic LinAlg {
  type vector;
  type matrix;
  logic vector vec_of_16_scalar(double * x)
    reads x[0..15];
  logic vector vec_of_36_scalar(double * x)
    reads x[0..35];
  ...
  logic vector vector_add(vector A, vector B);
  axiom vector_add_length:
    \forall vector x, y;
      vector_length(x) == vector_length(y) ==>
      vector_length(vector_add(x,y)) ==
        vector_length(x);
  axiom vector_add_select:
    \forall vector x, y, integer i;
      vector_length(x) == vector_length(y) ==>
      0 <= i < vector_length(x) ==>
      vector_select(vector_add(x,y),i) ==
      vector_select(x,i)+vector_select(y,i);
  ...
}
*/
```

Figure 7 ACSL Linear Algebra Theory

Linear Algebra Based ACSL Theory. In this ACSL theory, new abstract types for vectors and matrices are defined. Functions allowing the instantiation of those types of objects are also defined. As well, all of the well-known operations have also been axiomatized, such as vector-scalar multiplication, scalar product, norm etc. This ACSL theory is automatically generated during the autocoding process of the project and all the sizes of the vectors and matrices are known. Thus, within this theory, only functions that will create objects, from a C code pointer, of appropriate sizes (as illustrated in Fig. 7) are being defined. ACSL code is printed in green and its keywords in red. The C code keywords are printed in blue and the actual C code is printed in black. Using the keyword "type", two new abstract types, matrix and vector are being defined. Furthermore, the ACSL constructors for those types and the axiomatization of vector addition is shown in Fig. 7. The first axiom states that the length of the addition of two vectors, of same length is equal to this same length. The second axiom states that the elements of the addition of two vectors is the addition of the elements of the two separate vectors. Figure 7 shows a partial sample of the autocoded ACSL linear algebra theory.

Optimization and Ellipsoid Method Based ACSL Theory. The axiomatization of optimization problems has already been briefly discussed in Section III.A. Additionally, work has been dedicated to axiomatize the calculation of the vector constraint, feasibility, epsilon-optimality, etc. As well, Ellipsoids and related properties are formally defined, such as those presented in Fig. 8. Figure 8 presents the definition of another theory called "Ellipsoid" and within it, it shows the creation of a new abstract type "ellipsoid" and the definition of the functions "Ell" and "inEllipsoid". The function "Ell" returns the Ellipsoid formed by the matrix P and vector x as defined in Eq. (3). The function "inEllipsoid" returns true if the vector z is in the Ellipsoid E and returns false otherwise. As it was explained earlier, Theorem 1 was translated to ACSL and auto generated as ACSL lemmas. All the autocoded lemmas are proven using the software analyzer Frama-C and the SMT solver Alt-Ergo. One of the autocoded lemmas can be found in Fig. 9 and can be expressed using common mathematical notations as:

ACSL

```
#include "axiom_linalg.h"
/*@ axiomatic Ellipsoid {
  type ellipsoid;
  logic ellipsoid Ell(matrix P, vector x);
  logic boolean inEllipsoid(ellipsoid E,
      vector z);
  ...
}
*/
```

Figure 8 Ellipsoid Type Definition

$$\text{Assuming: } 0 < \epsilon/V < 1 \ ; \ 0 < \epsilon \ ; \ \forall x, y \in X, \ f_o(x) - f_o(y) \leq V \ ;$$

$$\forall z \in \mathbb{R}^n, \ z \notin \text{Ell}(P, x), \implies x_{best} \text{ is better than } z \text{ and } \text{Vol}(\text{Ell}(P, x)) < X_{\epsilon/V} \quad \text{Then: } x_{best} \text{ is } \epsilon - \text{optimal.}$$

A formal definition is also given for a point x being better than another point y. The set $X_{\epsilon/V}$ represents the set obtained by shrinking the feasible set X_f by a factor ϵ/V centered on the optimal point x^*. Further details about this set can be found in [24].

ACSL Lemma

```
/*@
 lemma epsilon_solution_lemma:
   \forall optim OPT, real r,V,epsilon, matrix P, vector x, x_best;
      (0 < epsilon/V < 1) ==> 0 < r ==> 0 < V ==> 0 < epsilon ==>
    size_n(OPT) > 0 ==>
    ( \forall vector x1, x2; isFeasible(OPT, x1) ==>  isFeasible(OPT, x2) ==>
      cost(OPT,x1) - cost(OPT,x2) <= V  ) ==>
    ( \forall vector z; !inEllipsoid(Ell(P,x), z) ==> isBetter(OPT, z, x_best) ) ==>
    ( \exists vector x; include(tomyset(Ell(mat_mult_scalar(ident(size_n(OPT)),r), x)) ,
            feasible_set(OPT)) ) ==>
     volume(tomyset(Ell(P,x))) < pow(epsilon/V*r, size_n(OPT)) ==>
     isEpsilonSolution(OPT, x_best, epsilon);
*/
```

Figure 9 Ellipsoid Method Main Lemma

D. Annotating C Code

Details are now given about how the C code is annotated and the type of Hoare triples used. For this, a specific technique was adopted. Every C code function is implemented in a separated file. That way, for every function, a corresponding C code body (.c) file and header file (.h) are automatically generated. The body file contains the implementation of the

function along with annotations and loop invariants. The header file contains the declaration of the function with its ACSL contract. The first kind of Hoare triples and function contracts added to the code were to check basic mathematical operations. Figure 10 shows an ACSL contract relative to the C code function computing the 2-norm of a vector of a size two. The ACSL contract specifies that the variable returned is positive and equal to the 2-norm of the vector of size two described by the input pointer. Using the keywords "behavior", "disjoint" and "complete" one can specify the different scenarios possible and treat them separately. Furthermore, it is proved that the result is always positive or null, and assuming the corresponding input vector not equal to zero, the output is strictly greater than zero. This last property becomes important when one must prove there are no divisions by zero (normalizing vectors). The implementation of the function is presented in Fig. 11. Once all the C functions implementing elementary mathematical operations have been annotated and proven, the next step consists of annotating the higher level C functions such as constraint and gradient calculations, matrix and vector updates, etc. Please find in Fig. 12 as an example, the annotated C function for the function "getp" that computes the vector p as described in Eq. (7), needed to perform the ellipsoid update. In

C Code + ACSL

```
#ifndef getNorm_2_lib
#define getNorm_2_lib
#include "axiom_linalg.h"
#include "my_sqrt.h"
#include "scalarProduct_2.h"
/*@
  @ requires \valid(Ain+(0..1));
  @ ensures \result ==
      twoNorm(vec_of_2_scalar(Ain));
  @ ensures \result >= 0;
  @ assigns \nothing;
  @ behavior Ain_non_null:
    @ assumes nonnull(vec_of_2_scalar(Ain));
    @ ensures \result > 0;
  @ behavior Ain_null:
    @ assumes !nonnull(vec_of_2_scalar(Ain));
    @ ensures \result == 0;
  @ complete behaviors Ain_non_null, Ain_null;
  @ disjoint behaviors Ain_non_null, Ain_null;
*/
double getNorm_2(double *Ain);
#endif
```

Figure 10 getNorm_2 Header C Code File

C Code + ACSL

```
#include "getNorm_2.h"
double getNorm_2(double *Ain) {
  double sum;
  sum = scalarProduct_2(Ain, Ain);
  return my_sqrt(sum); }
```

Figure 11 getNorm_2 Body C Code File

C Code + ACSL

```
#include "getp.h"
void getp() {
  double norm;
  double norm_inv;
  getTranspose();
  /*@ assert mat_of_2x2_scalar(&temp_matrix[0])==
    transpose(mat_of_2x2_scalar(&P_minus[0])); */
  changeAxis();
  /*@ assert vec_of_2_scalar(&temp2[0]) ==  mat_mult_vector(
    mat_of_2x2_scalar(&temp_matrix[0]), vec_of_2_scalar(&grad[0])); */
  /*@ assert vec_of_2_scalar(&temp2[0]) == mat_mult_vector(transpose(
    mat_of_2x2_scalar(&P_minus[0]) ), vec_of_2_scalar(&grad[0])); */
  norm = getNorm_2(temp2);
  /*@ assert 1/norm == 1/twoNorm( mat_mult_vector(
    transpose( mat_of_2x2_scalar(&P_minus[0]) ), vec_of_2_scalar(&grad[0]))); */
  /*@ assert vec_of_2_scalar(&temp2[0]) == mat_mult_vector(transpose(
    mat_of_2x2_scalar(&P_minus[0])),vec_of_2_scalar(&grad[0]));  */
  norm_inv = 1.0 / (norm);
  scaleAxis(norm_inv);
  /*@ assert 1/norm == 1/twoNorm( mat_mult_vector( transpose(
    mat_of_2x2_scalar(&P_minus[0])),vec_of_2_scalar(&grad[0]))); */
  /*@ assert vec_of_2_scalar(&temp2[0]) == mat_mult_vector(transpose(
    mat_of_2x2_scalar(&P_minus[0])), vec_of_2_scalar(&grad[0])); */
  /*@ assert vec_of_2_scalar(&p[0])==vec_mult_scalar(vec_of_2_scalar(&temp2[0]),1/norm); */
}
```

Figure 12 getp.c Body C Code File

Fig. 12 one can note the presence of several function calls followed by ACSL annotations, encoding the corresponding specifications. The first function call refers to the function "getTranspose" which computes the transpose of the matrix "P_minus" and stores it into the variable "temp_matrix". The function "changeAxis" multiplies the matrix "temp_matrix"

by the vector "grad" and stores the result into the vector "temp2". The current state of the memory is specified at each line of code using ACSL annotations. Then, after computing the norm and scaling the vector "temp2", the annotations specify that the resulting vector stored in the variable p has indeed been calculated as stated in Eq. (7).

IV. Sequential Optimization Problems

In this section, convergence guarantees are provided for a class of optimization problems used online. For this, an optimization problem with parameterized constraints and cost will be considered. This section concerns the study of how this parameter affects the optimization problem at each iteration and how to find the condition that the parameter needs to satisfy in order to prove convergence for every point along the trajectory.
First, the study focuses on the special case of linear constraints. Following this, another section will be dedicated to SOCP constraints.

A. Parameterized Linear Constraints

The objective is to solve in real-time the optimization problem described in Eq. (12). The vector $X \in \mathbb{R}^{n_x}$ denotes the decision vector.

$$\begin{array}{ll} \underset{X}{\text{minimize}} & f_o(X) \\ \text{subject to} & AX \leq b \end{array} \tag{12}$$

The initialization of this optimization problem is done using Eq. (13), where S is a full rank matrix. Usually, S represents a selector matrix and is of the form: $[I_p \; O_{p\times(n_x-p)}]$. A selector matrix is a matrix that selects one or more component from X. If S is a selector matrix then SX returns certain components of X. In that case it is obviously full rank. x_o denotes the input of the controller and it is written $\hat{x}_o$ to account for the fact that x_o changes from one optimization problem to another.

$$SX = \hat{x}_o \tag{13}$$

One can decompose and separate the equality and inequality constraints hidden behind the matrix A and vector b. That way, Eq. (12) can be written as:

$$\begin{array}{ll} \underset{X}{\text{minimize}} & f_o(X) \\ \text{subject to} & A_{\text{eq}}X = b_{\text{eq}} \\ & A_{\text{ineq}}X \leq b_{\text{ineq}} \\ & SX = \hat{x}_o \end{array} \tag{14}$$

The idea here is to project all the equality constraints in order to eliminate them while keeping track of the variable parameter, $\hat{x}_o$. There exist matrices M, A_1 and A_2 such that for all vectors X satisfying the equality constraints of the problem defined by Eq. (14), there exists a vector Z such that:

$$X = A_1 b_{\text{eq}} + A_2 \hat{x}_o + MZ \tag{15}$$

Equation (15) is a direct implication that the set of solutions of a linear system is an affine set. In this same equation, M is a matrix formed by an orthonormal basis of the null space of the matrix $\begin{bmatrix} A_{\text{eq}} \\ S \end{bmatrix}$ and d denotes the total number of equality constraints (number of rows of A_{eq} + number of rows of S). The vector Z then belongs to $\mathbb{R}^{n_z}$ with $n_z = n_x - d$. Thus, the original optimization problem described by Eq. (12) is equivalent to the projected problem:

$$\begin{array}{ll} \underset{Z}{\text{minimize}} & f_o(A_1 b_{\text{eq}} + A_2 \hat{x}_o + MZ) \\ \text{subject to} & A_{\text{f}} Z \leq b_o + Q\hat{x}_o \end{array} \tag{16}$$

With: $$A_{\text{f}} = A_{\text{ineq}}M \quad ; \quad b_o = b_{\text{ineq}} - A_{\text{ineq}}A_1 b_{\text{eq}} \quad \text{and} \quad Q = -A_{\text{ineq}}A_2. \tag{17}$$

More details and references about equality constraints elimination can be found in [4]. Using the Ellipsoid Method online to solve this optimization problem requires the computation of geometric characteristics on the feasible set of this parametric optimization problem for every possible x_o. For that reason, for now, let us assume: $\|x_o\|_2 \leq r_o$.
The parameterized polyhedral set below $P_{\hat{x}_o}$ is defined as:

$$P_{\hat{x}_o} = \{z \in \mathbb{R}^{n_z} : A_{\text{f}} z \leq b_o + Q\hat{x}_o\} \tag{18}$$

Having this collection of polyhedral sets, the goal is to compute ball radii that will tell us about the volume of the feasible set that would be true for every x_o. The operator $\nu(.)$ returns for a matrix A the vector $\nu(A)$ whose coordinates

are the 2-norm of the rows of A. Thus, a way of computing those geometric characteristics is to consider the two extreme polyhedral sets below:

$$P_{\min} = \{z \in \mathbb{R}^{n_z} : A_f z \leq b_o - r_o v(Q)\}, \tag{19}$$

$$P_{\max} = \{z \in \mathbb{R}^{n_z} : A_f z \leq b_o + r_o v(Q)\}. \tag{20}$$

In order to give an example of this concept, please find in Fig. 13 an illustration of such polyhedral sets (the illustrated sets have no physical meaning and do not represent any MPC problem). The two extreme polyhedral sets are drawn with solid lines and the actual feasible polyhedral sets are drawn using dotted lines. The values used are:

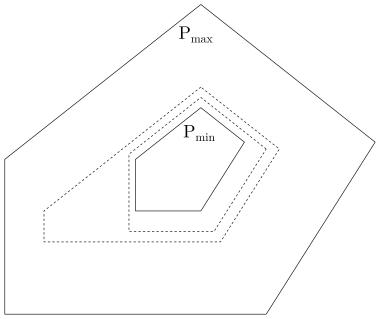


Figure 13 $P_{\max}$ **and** $P_{\min}$ **Polyhedral Sets**

$$A_f = \begin{bmatrix} -1 & 1 \\ 1 & 1 \\ 1 & -0.5 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \; v(Q) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \; b_o = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1.5 \\ 0.5 \\ 0.5 \end{bmatrix} \text{ and } \; r_o = 0.5.$$

Fact 1 *[Extreme Polyhedral Sets]*

$$\forall x_o \in \mathbb{R}^n \; s.t. \; \|x_o\|_2 \leq r_o \;, \; P_{\min} \subset P_{\hat{x}_o} \subset P_{\max}.$$

Proof 2 *Take x_o such that $\|x_o\|_2 \leq r_o$.*
First, using the Cauchy-Schwarz inequality, one can write:

$$|(Qx_o)(i)| = |row(Q,i)^T x_o| \leq \|row(Q,i)\|_2 \|x_o\|_2 \leq v(Q)(i) r_o \quad \forall i \tag{21}$$

$$\implies -v(Q) r_o \leq Q x_o \leq v(Q) r_o.$$

Then, if $x \in P_{\min}$, one can conclude that $A_f x \leq b_o - r_o v(Q)$. Using the inequality above, it is clear that it implies $x \in P_{x_o}$. Similarly, assuming that $x \in P_{x_o}$ and using the inequality below, it is clear that $x \in P_{\max}$. □

Next, one needs to find three scalars r, R and V such that:

$$\exists \, \bar{z}_1 \;\; \text{such that} \;\; B(\bar{z}_1, r) \subset P_{\min} \;, \tag{22}$$

$$\exists \, \bar{z}_2 \;\; \text{such that} \;\; P_{\max} \subset B(\bar{z}_2, R) \;, \tag{23}$$

$$V \geq \max_{z \in P_{\hat{x}_o}} f_o(z) - \min_{z \in P_{\hat{x}_o}} f_o(z) \;, \; \forall \;\; \|\hat{x}_o\| \leq r_o. \tag{24}$$

For the first scalar, r, one can compute a numerical value by running an off-line optimization problem finding the largest ball inside $P_{\min}$. If no solution can be found, the value of r_o needs to be decreased, and the process is repeated until acceptable values for r_o and r are found. Further information about finding the largest ball in a polytope can be found in [4].
As a consequence of equality constraint elimination, and seen in Eq. (15), the relation between the original decision vector X and the projected vector Z can be written as:

$$X = A_{\text{proj}} \begin{bmatrix} b_{\text{eq}} \\ \hat{x}_o \end{bmatrix} + MZ. \tag{25}$$

The decision vector X can now be decomposed into two parts, $\mathbf{x}$ and $\mathbf{u}$. The part $\mathbf{u}$ is bounded due to constraints in the original optimization problem (described by Eq. (12)). If no constraints on $\mathbf{u}$ were originally present, one can construct bounds on the projected vector Z to make the problem bounded and simpler to analyze. The point here being that from bounded variables within the vector X, one can infer bounds on the projected vector Z. There is no need to have original

bounds specifically on the collection of future inputs. Rewriting Eq. (25) yields:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} b_{\text{eq}} \\ \hat{x}_o \end{bmatrix} + \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} Z. \tag{26}$$

Following this, one can conclude:

$$Z = M_2^{-1}\left(\mathbf{u} - A_{21} b_{\text{eq}} - A_{22}\hat{x}_o\right). \tag{27}$$

Therefore assuming again that $\|\hat{x}_o\| \leq r_o$, one can compute a value of R such that:

$$\|Z\| \leq \left\|M_2^{-1}\right\| \left(\|\mathbf{u}\| + \left\|A_{21} b_{\text{eq}}\right\| + \|A_{22}\| \, r_o \right) = R. \tag{28}$$

On the other hand, from the physical meaning of the variables and the constraints of the optimization problem, one can construct bounds in which the variables should live, and therefore find a lower bound for V.
With the values R and r, one can now guarantee the convergence of a family of optimization problems parameterized by x_o.

Theorem 2 *[MPC Ellipsoid Method Convergence]*
The problem given by Eq. (14) is run in an online fashion in order to implement receding horizon control.
Using the Ellipsoid Method and initializing the first Ellipsoid by $B(\bar{z}_2, R)$, the method will find an ϵ-solution using N iterations for all $\hat{x}_o$ such that $\|\hat{x}_o\| \leq r_o$, with:

$$N = 2n(n+1)\log\left(\frac{R}{r}\frac{V}{\epsilon}\right)$$

with the variables $\bar{z}_2$, r, R and V satisfying the Eqs. (22), (23) and (24). The variable n denotes the dimension of the parameterized optimization problem.

Proof 3 . *Let us assume, the problem given by Eq. (14) is run online. For each iteration, the input parameter x_o is assumed to satisfy $\|x_o\| \leq r_o$. X_{f} denotes the feasible set of the current optimization problem to solve. That way, Eq. (22) is satisfied. Thus, by definition of r, we know that:*

$$\exists \bar{z}_1 \;\; \textit{such that} \;\; B(\bar{z}_1, r) \subset P_{\min} \subset X_f.$$

Similarly, because it is assumed that R satisfies Eq. (23), the following property holds:

$$\exists \bar{z}_2 \;\; \textit{such that} \;\; X_f \subset B(\bar{z}_1, r).$$

As well, the scalar V satisfies Eq. (11). Finally, One can conclude that the returned point will indeed be ϵ-optimal using N iterations thanks to Theorem 1. □

Linear Programs: In the case of Linear Programs, the method developed is identical and having a linear cost, the scalar V is easily found by running the two optimization problems below.

$$\begin{array}{ll} \underset{\hat{x}_o, z}{\text{minimize}} & c^T(A_1 b_{\text{eq}} + A_2 \hat{x}_o + M z) \\ & A_{\text{f}} z \leq b_o + Q\hat{x}_o \\ & \|\hat{x}_o\|_2 \;\leq\; r_o \end{array} \qquad \begin{array}{ll} \underset{x_o, z}{\text{maximize}} & c^T(A_1 b_{\text{eq}} + A_2 x_o + M z) \\ & A_{\text{f}} z \leq b_o + Q x_o \\ & \|x_o\|_2 \;\leq\; r_o \end{array}$$

B. Second-Order Conic Constraints
In the last section, the fact that only linear constraints was present was used to find the radius r. Therefore, in this section we aim to give ways of computing these constants for second-order constraints. Consider that one wants to solve a model predictive control optimization problem given by Eq. (29).

$$\begin{array}{ll} \underset{X}{\text{minimize}} & f_o(X) \\ & \|A_i X + b_i\|_2 \;\leq\; c_i^T X + d_i \;\; , \; i = 1 \ldots m \\ & A_{\text{eq}} X = b_{\text{eq}} \\ & SX = \hat{x}_o \end{array} \tag{29}$$

We assume that no equality constraints are hidden in the second-order constraints (if not, a very simple analysis will confirm this and one can extract those equality constraints and put it into the couple $(A_{\text{eq}}, b_{\text{eq}})$ from equation (29)).

Performing again an equality constraint elimination, we end up with an equivalent optimization problem, of smaller dimension, that contains no equality constraint. We want to find the radius of the biggest ball inside the feasible set of this latter problem. Unfortunately, second-order constraints are still present and finding the largest balls inside second-order cones is not an easy task. For this, we use the equivalence of norms in finite dimensions, noting that $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are linear, to perform a linear relaxation on the second-order constraints and end up with a polyhedral set as the feasible set. For instance, the problem described by Eq. (30) represents a linear relaxation of the original problem stated in Eq. (29) after having eliminated the equality constraints.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & f_o^{'}(Z) \\ & \sqrt{n}\left\|A_i^{'}Z + b_i^{'}\right\|_\infty \;\leq\; c_i^{'T}Z + d_i^{'} \;,\; i = 1 \ldots m \end{aligned} \tag{30}$$

Thus, by finding the largest ball inside the resulting feasible set (which is a polyhedral set), a ball is finally found inside the original second-order cone.

V. Bounding the Condition Number

The use of the Ellipsoid Method was justified by arguing that it represents a trade off between performance and safety. As well, it also attracted our attention for its numerical properties. Although at first, the Ellipsoid Method appears to be numerically stable, finding *a priori* bounds on the program's variables is challenging. The worst case occurred when the separating hyperplane has the same direction at each iteration. In this case, the condition number of the successive ellipsoids increases exponentially, and so do the program's variables. This worst case is very unlikely to happen in practice. Nevertheless, a mathematical way to get around this case is needed, along with a way to compute *a priori* bounds on the variables before the execution of the program. In this section, a method for controlling the condition number of the successive ellipsoids and a way to compute those bounds by adding a correcting step in the original algorithm is presented. This section also includes how the formal proof of the corresponding software is modified to support the correctness of the resulting modified ellipsoid algorithm.

A. Bounding the Singular Values of the Successive Ellipsoids

When updating B_i by the usual formulas of the ellipsoid algorithm (Eq. (6)), B_i evolves according to $B_{i+1} = B_i D_i$, where D_i has $n-1$ singular values equal to $n/\sqrt{n^2-1}$, and has one singular value equal to $n/(n+1)$. It follows that at a single step the largest and the smallest singular values of B_i can change by a factor from $[1/2,\ 2]$. The objective is to prove that one can modify the algorithm to bound the singular values of the matrix B_i throughout the execution of the program.
Minimum Half Axis: First, we claim that if $\sigma_{min}(B_i)$ is less than than $r\epsilon/V$ then the algorithm has already found an ϵ-solution. Note that the scalar ϵ is the desired precision and the scalars r and V are defined in Section III. Please find below a proof of this statement.

Proof 4 *Let us assume $\sigma_{min}(B) < r\epsilon/V$. In this case, E_i is contained in the strip between two parallel hyperplanes, the width of the strip being less than $2 \cdot r\epsilon/V$ and consequently E_i does not contain $X_\epsilon = \theta X_f + (1-\theta)x_*$, where x_* is the minimizer of f_o and $\theta = \epsilon/V$ (because X_ϵ contains a ball of radius $r\epsilon/V$). Consequently, there exists $z \in X_f$ such that $y = \theta z + (1-\theta)x_* \in X_\epsilon$ but $\notin E_i$, implying by the standard argument (developed in [24]) that the best value f^+ of f processed so far for feasible solutions satisfies $f^+ \leq f(y) \leq f(x_*) + \theta f(z - x_*)$ which implies that $f^+ \leq f(x_*) + \epsilon$. We can thus stop the algorithm and return the current best point found (feasible and smallest cost).* □

Maximum Half Axis: We argue now that one can modify the original ellipsoid algorithm in order to bound the value of the maximum singular value of B_i. When the largest singular value of B_i is less than $2R\sqrt{n+1}$, we carry out a step as in the basic ellipsoid method. When this singular value is greater than $2R\sqrt{n+1}$, a corrective step is applied to B_i, which transforms E_i into E_i^+. Under this corrective step, E_i^+ is a localizer along with E_i, specifically $E_i \cap X_f \subset E_i^+ \cap X_f$, and additionally:

(a) The volume of E_i^+ is at most γ (upper bound of reduction ratio from Eq. (8)) times the volume of E_i ;

(b) The largest singular value of B_i^+ is at most $2R\sqrt{n+1}$.

The corrective step is described as follows. First, define $\sigma = \sigma_{max}(B_k) > 2R\sqrt{n+1}$ and let e_i be a unit vector corresponding to the singular direction. We then consider:

$$G = \text{diag}\left(\sqrt{n/(n+1)},\ \sqrt{n+1}/\sigma,\ \ldots,\ \sqrt{n+1}/\sigma\right).$$

This case is concluded by performing the update:

$$B_{i+1} = B_i \cdot G \quad \text{and} \quad c_{i+1} = c_i - (e_i^T c_i) \cdot e_i. \tag{31}$$

Please find in Fig. 14 an illustration of this corrective step. The ellipsoid E_i shown in Fig. 14 appears to have a condition number too high due to its large semi-major axis. The correction step as detailed previously has been performed and the corrected ellipsoid E_i^+ is shown with dotted line. As one can see, the volume as been decreased and the area of interest $E_i \cap X_f$ (dotted area) still lies within the ellipsoid E_i^+. Hence, one can conclude that throughout the execution of the code:

$$\sigma_{min}(B_i) \geq \frac{1}{2}\frac{r\epsilon}{V} = \frac{r\epsilon}{2V} \quad \text{and} \quad \sigma_{max}(B_i) \leq 2 \times 2R\sqrt{n+1} = 4R\sqrt{n+1}.$$

B. Corresponding Condition Number

From the definition of the condition number, one can write:

$$k(B) = \|B\| \cdot \left\|B^{-1}\right\| = \sigma_{max}(B)/\sigma_{min}(B).$$

Thus, by bounding the singular values of B, a bound on its condition number can be constructed. The factors $1/2$ and 2 that could affect the singular values of B at each iteration are also taken into account to get:

$$k(B) \leq \left(\frac{2}{1/2} \cdot \frac{2R\sqrt{n+1}}{r\epsilon/V}\right) = \left(\frac{8R\sqrt{n+1}}{r\epsilon/V}\right)$$

and

$$\|B\| = \sigma_{max}(B) \leq 4R\sqrt{n+1}.$$

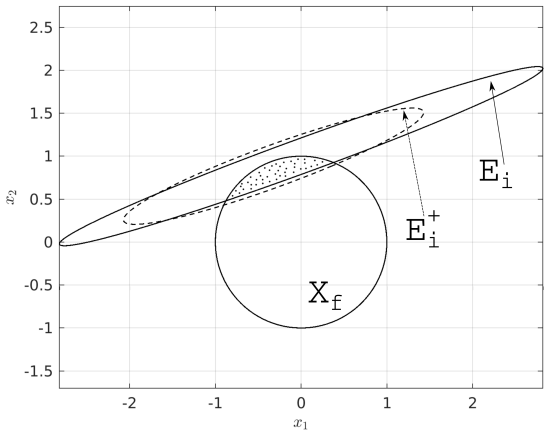


Figure 14 Corrective step for Ellipsoid E_i according to Eq. (31). The feasible set X_f is illustrated as the unit ball.

C. Corresponding norm on c

At each iteration it is known that the optimal point belongs to the current ellipsoid. Thus:

$$\|x^* - c_k\| \;=\; \|B_k u\| \;\leq\; \|B_k\| \cdot \|u\| \;\leq\; \|B_k\|, \quad \text{for some } u \in B_1(0).$$

Finally,

$$\|c_k\| \;\leq\; R + \|x_c\| + \|B_k\|.$$

D. Consequences on the Code

In this section, we explain how to implement this correcting step and give the tools to verify it. In order to detect an ellipsoid with large semi-major axis, the largest singular value σ_{max} of the current matrix B_k needs to be computed. However, performing a singular value decomposition would be far too expensive and slow (this decomposition being performed at each iteration). The Frobenius norm, defined in Def. 4, which is a well-known upper bound on the maximum singular value, is computed instead.

Definition 4 *The Frobenius norm of a matrix $A \in \mathbb{R}^{n\times n}$ is:*

$$\|A\|_F \;=\; \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}^2}$$

The Frobenius norm of a vector has been axiomatized by setting it equal to the vector 2-norm of the "vectorized" matrix. A matrix is "vectorized" by concatenating all its rows in a single vector. In the case of an overly large semi-major axis, the direction e in which this axis lies is needed as well. Therefore, the power iteration algorithm is performed in order to compute this information.

VI. Floating-Point Considerations

Standard notation is used for rounding error analysis [27–29], fl() being the result of the expression within the parenthesis rounded to the nearest floating-point number. The relative rounding error unit is written **u** and **eta** denotes the underflow

unit. For IEEE 754 double precision (binary64) we have **u**= 2^{-53} and **eta**= 2^{-1074}.
This section presents an analysis targeting the numerical properties of the ellipsoid algorithm. Contributions already have been made concerning finite-precision calculations within the ellipsoid method [26]. However, this work only shows that it is possible to compute approximate solutions without giving exact bounds, and is only for Linear Programming (LP). Also, the analysis performed considers abstract finite-precision numbers and the actual machine floating-point types are not mentioned. Thanks to the analysis performed in this section, using the IEEE standard for floating-point arithmetic and knowing exactly how the errors are being propagated, it is possible to check *a posteriori* the correctness of the analysis using static analyzers [30, 31].

A. Preliminaries

Within this algorithm, we focus our attention on the update formulas (5), (6) and (7), allowing us to update the current ellipsoid into the next one. In order to propagate the errors due to rounding through the code, we state now a useful theorem dealing with matrix perturbations and inverses.

Theorem 3 *[32][Matrix perturbations and Inverse] Let A be a non-singular matrix of* $\mathbb{R}^{n\times n}$ *and* ΔA *a small perturbation of A. Then,*

$$\frac{\left\|(A+\Delta A)^{-1}-A^{-1}\right\|}{\left\|A^{-1}\right\|} \leq k(A)\frac{\|\Delta A\|}{\|A\|} \tag{32}$$

B. Norms and Bounds

To successfully perform the algorithm's numerical stability analysis, we need to know how "big" the variables can grow within the execution of the algorithm. Indeed, for a given computer instruction, the errors due to floating-point arithmetic are usually proportional to the variables values. As it was explained in Section V, we slightly modified the ellipsoid algorithm to keep the condition numbers of the ellipsoid iterates under control. Therefore, implementing this corrected algorithm, we can use the following results:

$$\|p\| = \left\|B^T e\right\| / \sqrt{e^T B B^T e} = 1, \tag{33}$$

$$\|B\| \leq 4R\sqrt{n+1}, \tag{34}$$

$$k(B) \leq (8RV\sqrt{n+1})/(r\epsilon), \tag{35}$$

$$\|c\| \leq R + \|x_c\| + \|B\|, \tag{36}$$

where n, R, r, V, x_c and ϵ are the variables described in Section III.

C. Problem Formulation and Results

In order to take into account the uncertainties on the variables due to floating-point rounding, the algorithm is modified to make it more robust. Those uncertainties are first evaluated and a coefficient λ is then computed. This coefficient represents how much the ellipsoid E_k is being widened at each iteration (see Fig. 15). Let us assume we have $B \in \mathbb{F}^{n\times n}$, $p \in \mathbb{F}^n$, $c \in \mathbb{F}^n$. We want to find $\lambda \geq 1 \in \mathbb{R}$ such that:

$$\mathrm{Ell}\Big(B, c\Big) \subset \mathrm{Ell}\Big(\lambda \cdot \mathrm{fl}(B),\ \mathrm{fl}(c)\Big). \tag{37}$$

Before evaluating any of those rounding errors, Lemma 1 is stated, which gives a sufficient condition for the coefficient λ to have $\mathrm{Ell}\big(\lambda \cdot \mathrm{fl}(B), \mathrm{fl}(c)\big)$ covering $\mathrm{Ell}(B, c)$. If the calculations of B and c were perfect, using Lemma 1, $\lambda = 1$ would be a solution; no correction is indeed necessary.

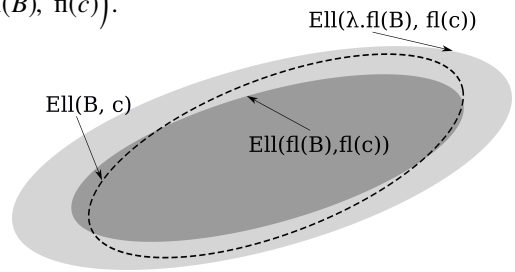


Figure 15 Ellipsoid Widening

Lemma 1 *[Widening - Sufficient Condition]*

$$\left\|\mathrm{fl}\big(B\big)^{-1}B\right\| + \left\|\mathrm{fl}\big(B\big)^{-1}\right\| \cdot \|c - \mathrm{fl}(c)\| \leq \lambda \implies \mathrm{Ell}\Big(B, c\Big) \subset \mathrm{Ell}\Big(\lambda \cdot \mathrm{fl}(B),\ \mathrm{fl}(c)\Big).$$

Proof 5 *The starting hypothesis is:* $\left\|\mathrm{fl}\big(B\big)^{-1}B\right\| + \left\|\mathrm{fl}\big(B\big)^{-1}\right\| \cdot \|c - \mathrm{fl}(c)\| \leq \lambda$. *Using the fact that the two norm is a consistent norm, we have:*

$$\forall\, u \in B_1(0)\,,\ \left\|\text{fl}(B)^{-1}Bu\right\| \leq \left\|\text{fl}(B)^{-1}B\right\| \quad \textit{and} \quad \left\|\text{fl}(B)^{-1}\Big(c - \text{fl}(c)\Big)\right\| \leq \left\|\text{fl}(B)^{-1}\right\| \cdot \|c - \text{fl}(c)\|\,.$$

Therefore using these properties and the assumed starting property, we have:

$$\forall\, u \in B_1(0)\,,\ \left\|\text{fl}(B)^{-1}Bu\right\| + \left\|\text{fl}(B)^{-1} \cdot \Big(c - \text{fl}(c)\Big)\right\| \leq \lambda.$$

Then, using the triangle Inequality ($\|x + y\| \leq \|x\| + \|y\|$), we finally have:

$$\forall\, u \in B_1(0)\,,\ \left\|\text{fl}(B)^{-1} \cdot \Big(Bu + c - \text{fl}(c)\Big)\right\| \leq \lambda.$$

Reformulating the second part of the above statement (we are assuming that $\lambda > 0$):

$$\forall\, u_1 \in B_1(0)\,,\ z = Bu_1 + c \rightarrow \exists\, u_2 \in B_1(0)\,,\ \Big(\lambda \cdot \text{fl}(B)\Big)^{-1} \cdot \Big(z - \text{fl}(c)\Big) = u_2.$$

Using the definition of Ellipsoids described in Eq. (3), one can write the equivalent statement:

$$\forall\, z \in \text{Ell}(B, c), \exists\, u_2 \in B_1(0)\,,\ \Big(\lambda \cdot \text{fl}(B)\Big)^{-1} \cdot \Big(z - \text{fl}(c)\Big) = u_2.$$

Rearranging the equation implies:

$$\forall\, z \in \text{Ell}(B, c), \exists\, u_2 \in B_1(0)\,,\ z = \lambda \cdot \text{fl}(B)u_2 + \text{fl}(c).$$

Again, this is equivalent to:

$$\forall\, z \in \text{Ell}(B, c) \rightarrow z \in \text{Ell}\Big(\lambda \cdot \text{fl}(B), \text{fl}(c)\Big).$$

which is the desired property:

$$\text{Ell}\Big(B, c\Big) \subset \text{Ell}\Big(\lambda \cdot \text{fl}(B),\ \text{fl}(c)\Big).$$

□

The floating-point errors Δ_B, $\Delta_{B^{-1}}$ and Δ_c are defined as:

$$\Delta_B = \text{fl}(B) - B \quad , \quad \Delta_c = \text{fl}(c) - c \quad , \quad \Delta_{B^{-1}} = (\text{fl}(B))^{-1} - (B)^{-1} \quad . \tag{38}$$

For now, it is assumed that after performing the floating-point analysis, $\mathcal{E}_B$ and $\mathcal{E}_c$ have been found such that:

$$|(\Delta_B)_{i,j}| \leq \mathcal{E}_B \quad \forall i, j \in [1, n], \qquad \text{and} \qquad |(\Delta_c)_i| \leq \mathcal{E}_c \quad \forall i \in [1, n].$$

From Lemma 1, one can see that the calculation of a widening coefficient λ highly depends on the accuracy of the matrix $\text{fl}(B)^{-1}$. Therefore, $\mathcal{E}_{B^{-1}}$ is also needed such that: $|(\Delta_{B^{-1}})_{i,j}| \leq \mathcal{E}_{B^{-1}} \quad \forall i, j \in [1, n]$.
The quantity $(B)^{-1}$ is not used explicitly in the algorithm and its floating-point error could not be evaluated by numerically analyzing the method. Instead, perturbation matrix theory [32] and Theorem 3 will be used, which gives a lower bound on $\mathcal{E}_{B^{-1}}$ given $\mathcal{E}_B$, the norm of B and its condition number. The result is stated as follows.

Lemma 2 *[Widening - Analytical Sufficient Condition]*

$$1 + \frac{k(B)}{\|B\|}\sqrt{n} \cdot \left(\sqrt{n} \cdot k(B)\mathcal{E}_B + \mathcal{E}_c + \frac{k(B)}{\|B\|}n\mathcal{E}_B\mathcal{E}_c\right) \leq \lambda \implies \text{Ell}\Big(B, c\Big) \subset \text{Ell}\Big(\lambda \cdot \text{fl}(B),\ \text{fl}(c)\Big).$$

Proof 6 *To prove this lemma, $\|\Delta_{B^{-1}}\|$ is first evaluated. Using Eq. (32):*

$$\|\Delta_{B^{-1}}\| \leq k(B)\frac{\left\|B^{-1}\right\|}{\|B\|}\|\Delta_B\| = \frac{k^2(B)}{\|B\|^2}\|\Delta_B\|\,.$$

But, $\|\Delta_B\| \leq \|\Delta_B\|_F \leq n\mathcal{E}_B$, which implies: $\|\Delta_{B^{-1}}\| \leq \frac{k^2(B)}{\|B\|^2}n\mathcal{E}_B$. The three constants I, J and K are now defined: $I = \left\|\text{fl}(B)^{-1}B\right\|$, $J = \left\|\text{fl}(B)^{-1}\right\|$ *and* $K = \|c - \text{fl}(c)\|$.
The next step consists of computing an upper bound for each of those constants.

$$I = \|I_n + \Delta_{B^{-1}}B\| \implies I \leq \|I_n\| + \|\Delta_{B^{-1}}\|\,\|B\| = 1 + \frac{k^2(B)}{\|B\|}n\mathcal{E}_B,$$
$$J \leq \left\|(B)^{-1}\right\| + \|\Delta_{B^{-1}}\| = \frac{k(B)}{\|B\|} \cdot \left(1 + \frac{k(B)}{\|B\|}n\mathcal{E}_B\right),$$
$$K \leq \sqrt{n}\mathcal{E}_c.$$

So, if

$$1 + \frac{k(B)}{\|B\|}\sqrt{n}\cdot\left(\sqrt{n}\cdot k(B)\mathcal{E}_B + \mathcal{E}_c + \frac{k(B)}{\|B\|}n\mathcal{E}_B\mathcal{E}_c\right) \leq \lambda \implies I + J\cdot K \leq \lambda,$$

then

$$\left\|\mathrm{fl}(B)^{-1}B\right\| + \left\|\mathrm{fl}(B)^{-1}\right\|\cdot\|c - \mathrm{fl}(c)\| \leq \lambda.$$

Using the result of Lemma 1, one can conclude on the inclusion property (37). □

Thus, due to Lemma 2, following the floating-point analysis of the algorithm, a coefficient λ such that Eq. (37) is valid can now be computed. After finding such a λ, using over-approximation schemes, we consider how the algorithm's convergence changes. Because the method's proof lies with the fact that the final ellipsoid has a small enough volume, this correction has an impact on the guaranteed number of iterations. Lemma 3 addresses those issues.

Lemma 3 *[Convergent Widening Coefficient] Let $n \in \mathbb{N}, n \geq 2$.*
The algorithm implementing the widened ellipsoids, with coefficient λ converges if:

$$\lambda < \exp\big(1/(n(n+1))\big). \tag{39}$$

In that case, if N denotes the original number of iteration needed, the algorithm implementing the widened ellipsoids will require:

$$N_\lambda = N/\big(1 - n(n+1)\cdot\log(\lambda)\big) \quad \textit{iterations} \tag{40}$$

Proof 7 *We recall that the Ellipsoid algorithm implementing the widened ellipsoids, with coefficient λ converges if:*

$$\mathrm{Vol}\big(\lambda\cdot\mathrm{Ell}(B_{k+1}, c_{k+1})\big) < \mathrm{Vol}\big(\mathrm{Ell}(B_k, c_k)\big).$$

When using the Ellipsoid algorithm update process (with or without condition number correction), we have:

$$\mathrm{Vol}\big(\mathrm{Ell}(B_{k+1}, c_{k+1})\big) \leq \gamma\cdot\mathrm{Vol}\big(\mathrm{Ell}(B_k, c_k)\big)$$

With $\gamma = \exp(-1/(2(n+1)))$, and Eq. (4), we know that:

$$\mathrm{Vol}\big(\mathrm{Ell}(\lambda\cdot B_{k+1}, c_{k+1})\big) = \lambda^{n/2}\cdot\mathrm{Vol}\big(\mathrm{Ell}(B_{k+1}, c_{k+1})\big)$$

Therefore, the algorithm implementing the widened ellipsoids, with widening coefficient λ will converge if:

$$\lambda^{n/2}\cdot\gamma < 1 \quad \textit{which is equivalent to} \quad \lambda < \exp\big(1/(n(n+1))\big)$$ □

Proof 8 *The second statement of Lemma 3 is now proved, which shows how to compute the updated number of iterations for the algorithm implementing widened ellipsoids. We introduce $B'_{k+1} = \lambda B_{k+1}$. Therefore using Eq. (4), and $\mathrm{Vol}(E'_{k+1}) = \lambda^{n/2}\cdot\mathrm{Vol}(E_{k+1})$, we get:*

$$\gamma_\lambda = \frac{\mathrm{Vol}(E'_{k+1})}{\mathrm{Vol}(E_k)} = \frac{\mathrm{Vol}(E'_{k+1})}{\mathrm{Vol}(E_{k+1})}\frac{\mathrm{Vol}(E_{k+1})}{\mathrm{Vol}(E_k)} = \lambda^{n/2}\cdot\gamma.$$

To end up at the final step with an ellipsoid of the same volume, we need: $\mathrm{Vol}(E_o)\cdot\gamma_\lambda^{N_\lambda} = \mathrm{Vol}(E_o)\cdot\gamma^N$. Which implies

$$N_\lambda\big(\log(\gamma) + n/2\cdot\log(\lambda)\big) = N\log(\gamma).$$

Replacing γ by its value, using property 1 from Section III.B:

$$N_\lambda\Big(n/2\cdot\log(\lambda) - 1/(2(n+1))\Big) = -N/(2(n+1))$$

And thus, we arrive at the formula: $$N_\lambda = N/\{1 - n(n+1)\log(\lambda)\}.$$ □

D. Computing $\mathcal{E}_B$ and $\mathcal{E}_c$

This section introduces the evaluation of the floating-point errors taking place when performing the update formulas (5) and (6) (represented by $\mathcal{E}_c$ and $\mathcal{E}_B$). For this, numerical properties for basic operations appearing in the algorithm are first presented.

Rounding of a Real. Let $z \in \mathbb{R}$, $\quad \tilde{z} = \mathrm{fl}(z) = z + \delta + \eta \quad$ with $|\delta| < \mathbf{u}$ and $|\eta| < \mathbf{eta}/2$

Product and Addition of Floating-Points. Let $a, b \in \mathbb{F}$.

$$\begin{aligned} \mathrm{fl}(a+b) &= (a+b)(1+\epsilon_1) && \text{with} \quad |\epsilon_1| < \mathbf{u} \\ \mathrm{fl}(a\times b) &= (a\times b)(1+\epsilon_2) + \eta_2 && \text{with} \quad |\epsilon_2| < \mathbf{u},\ |\eta_2| < \mathbf{eta} \ \text{ and } \ \epsilon_2\cdot\eta_2 = 0 \end{aligned}$$

Reals-Floats Product. Let $z \in \mathbb{R}$ and $a \in \mathbb{F}$, $\quad |\mathrm{fl}\big(\mathrm{fl}(z)\cdot a\big) - z\cdot a| \leq |z||a|\cdot\mathbf{u} + |a|\cdot 2\mathbf{u}(1+\mathbf{u})$

Scalar Product. Let $a, b \in \mathbb{F}^n$. We define, $\langle a, b\rangle = \sum_{i=1}^{n} a_i b_i$ and $|a, b| = \sum_{i=1}^{n} |a_i b_i|$. We have then:

$$|\mathrm{fl}\langle a, b\rangle - \langle a, b\rangle| \leq A_n |a, b| + \Gamma_n$$
$$\text{With:} \quad A_n = n \cdot \mathbf{u}/(1 - n \cdot \mathbf{u}) \quad \text{and} \quad \Gamma_n = A_{2n} \cdot \mathbf{eta}/\mathbf{u}$$

Rounding Error on c. Knowing how the errors are being propagated through elementary transformations, the goal is to compute the error for a transformation similar to the vector c's update. For each component of c, we have:

$$c_i = c_i - \Big(1/\{n+1\}\Big) \cdot \langle \mathrm{Row}_i(B), p\rangle.$$

Therefore, the operation performed, in floating-point arithmetic is:

$$\mathrm{fl}\big(c + \mathrm{fl}\big(\mathrm{fl}(z) \cdot \mathrm{fl}\langle a, b\rangle\big) \qquad \text{with: } a, b \in \mathbb{F}^n,\ c \in \mathbb{F} \text{ and } z \in \mathbb{R}. \tag{41}$$

Using this and neglecting all terms in **eta** and powers of **u** greater than two, we get:

$$\mathcal{E}_c \leq u \cdot \Big(\big(16n^2 + 16n + 3\big) \cdot \|B\| + \|c\|\Big). \tag{42}$$

Error on B. Similarly, for each component of B, the operation below is performed:

$$B_{i,j} = \alpha \cdot B_{i,j} + \beta \cdot \langle \mathrm{Row}_i(B), p\rangle \cdot p_j.$$

In floating-point arithmetic, this last equation can be put into form:

$$\mathrm{fl}\Big(\mathrm{fl}\Big(\mathrm{fl}(z_1) \cdot d\Big) + \mathrm{fl}\Big(\mathrm{fl}(z_2) \cdot \mathrm{fl}\big(\mathrm{fl}\langle a, b\rangle \cdot c\big)\Big)\Big).$$

Similarly, propagating the errors using elementary transformations implies that:

$$\mathcal{E}_B \leq u \cdot \|B\| \cdot \Big(\big(n^2/\{1 - nu\} + 2\big)|\beta| + n + 2|\alpha| + 1\Big). \tag{43}$$

VII. Automatic Code Generation and Examples

A. Credible Autocoding

Credible autocoding, is a process by which an implementation of a certain input model in a given programming language is being generated along with formally verifiable evidence that the output source code correctly implements the input model. The goal of the work presented in this article is to automatically generate, formally verifiable C code implementations of a given receding horizon controller. Thus, an autocoder that we call a "Credible Autocoder" (see Fig. 16) has been built that generates an ACSL annotated C code implementation of a given MPC controller. This autocoder takes as an input a formulation of a MPC controller written by the user in a text file. Once the output code is generated, it can be checked using the software analyzer Frama-c and the plugin WP. If the verification terminates positively, the code correctly implements the wanted receding horizon control. The controller can then be compiled and the binary file embedded in a feedback control loop. The specification and the requirements are automatically generated from the input text file written by the user. The verification taking place in this work is applied to one high-level requirement (of the MPC solver). The other high-level requirements such as those involving control-related issues are not examined. Nevertheless, given that the input text file is written in a high-level language, specifically designed for MPC formulations, it is relatively simple to read. The use of autocoders for automated MPC code generation makes the formulation easier to check and add traceability to the algorithms.

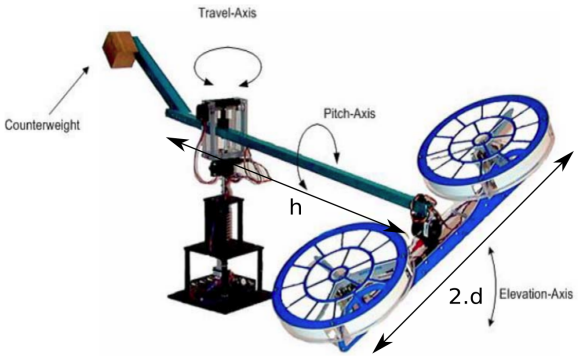


Figure 17 Quanser – 3 DOF Helicopter

B. Example: the three degree-of-freedom helicopter

The three degree-of-freedom (3 DOF) helicopter shown in Fig. 17 was used to illustrate the framework developed in this article. The vector state of the system collects the 3 axis angles and rates and it is denoted by $x = [\theta\ \psi\ \phi\ \dot{\theta}\ \dot{\psi}\ \dot{\phi}]$. The inputs are the voltages of the front and back DC motors.

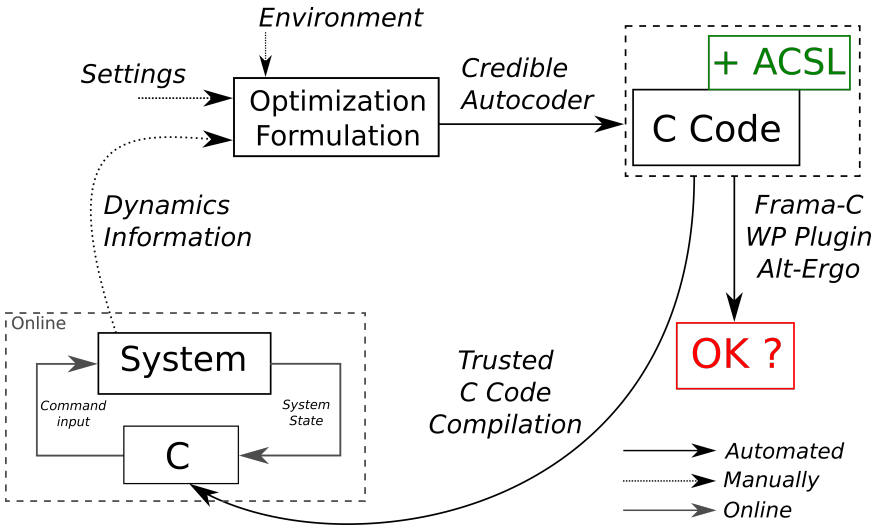


Figure 16 Credible Autocoding Framework

Further information about the 3 DOF helicopter can be found in [33]. Given an inner feedback controller and a discretization step of $T = 0.5\ sec$, the resulting system is a stable linear system. The problem we are trying to solve is a landing of the 3 DOF helicopter. Starting with an angle of 25 deg in elevation, 15 deg in travel (see Fig. 17 for axis) and all the other states being zero, the objective is to design a controller that can drive the system back to the origin while avoiding the ground. The ground is the area below $\theta = 0$. Thus, the constraint for enforcing ground avoidance is a formula on the elevation and the pitch angles and can be formulated as: $h\sin(\theta) \pm d\sin(\phi) >= 0$. By linearizing this constraint over small angles, one can obtain linear inequalities of the form: $A_{obs}x \leq b_{obs}$. We want to implement the following MPC controller.

$$\begin{aligned} \underset{X=[\mathbf{x},\mathbf{u}]}{\text{minimize}} \quad & \sum_{k=1}^{H} \|x_k\| \\ & x_{k+1} = Ax_k + Bu_k\,,\ k = 1..H-1 \\ & \|u_k\| \leq 60\,,\ k = 1..H-1 \\ & 0 \leq x_k(1) \ \text{ and } \ A_{obs}x_k \leq b_{obs}\,,\ k = 2..H \\ & x_1 = \hat{x}_o \end{aligned} \tag{44}$$

Assuming that $\|\hat{x}_o\| \leq 27$, we found, using the method developed in Section IV a radius 8.0612 (running an off-line optimization problem that finds the largest ball inside P_{min}). Similarly, $R = 322$, was obtained using Eq. (28). From the problem formulation, one can see that the norms of the successive x's along the trajectory constructed are supposed to be minimized. Given a starting point, an upper bound on the objective function over the feasible set can be computed. The objective function is maximal when x_o has the largest norm, and when the system stays at this point throughout the trajectory. The constant V can therefore be computed as:

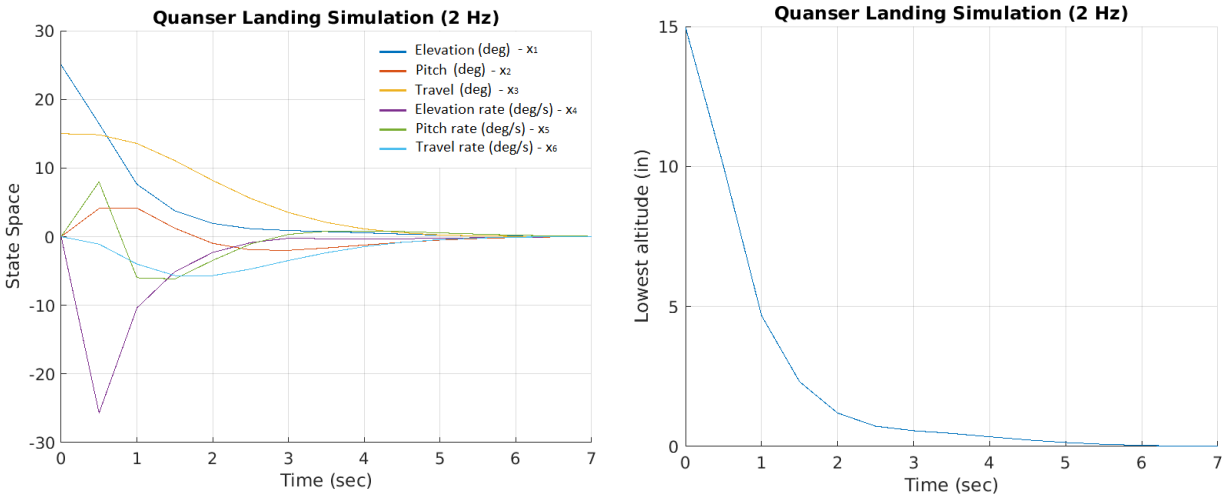
$$V = H \cdot \|x_o\| \ \leq \ H \cdot 27 = 6 \times 27 = 162. \tag{45}$$

Following those calculations, a number of step of $N = 5528$ is found. In order to control floating-point errors, a widening coefficient λ can be constructed. Performing the steps described in Section VI, using double precision floating-points and an accuracy of $\epsilon = 0.25$ result in:

$$\lambda = 1.000695409372118.$$

As it was explained in Section VI, the ellispoid widening increased the number of iterations needed for the convergence to $N_\lambda = 6817$. The simulation was ran on a Intel Core i5-3450 CPU @ 3.10GHz × 4 processor and the running time was approximately 0.2 sec for a single point. The results of the simulation can be found in Fig. 18a and 18b. Figure 18a presents the closed-loop response for the state vector x and Fig. 18b shows the lowest altitude point with time for the same simulation. The text file presented in Fig. 19 has been used to generate the C code in order to perform the simulation. The whole simulation was executed using the autocoded C code and a Simulink model. The full autocoder

source code, input file (Fig. 19), Simulink model and a user guide for the autocoder can be found online*.



(a) State Vector versus time

(b) Lowest Altitude versus time

Figure 18 Simulation Data

Input File to the Autocoder

```
Input
xo(6)
Output
u(:,1)
Constants
H = 6; M = H-1; l = 90; r = 40;
A = [0.7101    0.0000   -0.0000    0.2331    0.0000    0.0000;
     0.0000    0.2105    0.4023    0.0000    0.0977    0.7390;
    -0.0000   -0.1272    0.9846   -0.0000   -0.0134    0.4733;
    -0.8721    0.0000   -0.0000    0.0724    0.0000    0.0000;
    -0.0000   -2.0777    0.7830    0.0000   -0.2674    1.6711;
    -0.0000   -0.4224   -0.1072   -0.0000   -0.0618    0.8109];
B = [0.2899    0.0000;    -0.0000   -0.4023; 0.0000    0.0154;
     0.8721    0.0000;     0.0000   -0.7830; 0.0000    0.1072];
Aobs = [-l  -r  0  0  0  0; -l  r  0  0  0  0]; bosbt = [0;0];
Variables
x(6,H) u(2,M)
Minimize
sum( || x(:,k) || , k = 1..H )
SubjectTo
constraint1: x(:,1) = xo;
constraint2: x(:,k+1) = A*x(:,k) + B*u(:,k)  ,k=1..H-1;
constraint3: -30 <= u(1,k)                   ,k=1..H-1;
constraint4: u(1,k) <= 30                    ,k=1..H-1;
constraint5: -30 <= u(2,k)                   ,k=1..H-1;
constraint6: u(2,k) <= 30                    ,k=1..H-1;
constraint8:  0 <= x(1,k)                    ,k=2..H;
constraint9: -40 <= x(2,k)                   ,k=2..H;
constraint10: x(2,k) <= 40                   ,k=2..H;
constraint11: Aobs*x(:, k) <= bosbt          ,k=2..H;
Information
r = 8.06; R = 322; V = 162; eps = 0.25; lambda = 1.000695409372118;
```

Figure 19 3 DOF Helicopter Landing Problem: Autocoder Input File

*The source code for the autocoder is available at: https://cavale.enseeiht.fr/quanser_mpc/

VIII. Conclusion

In this article, we presented a formal framework for the automatic generation and verification of optimization code for solving second-order cone programs. We focused on the ellipsoid method due to its good numerical characteristics. We built a framework capable of compiling the high-level requirements of online receding horizon solvers into C code programs which can then be automatically verified using existing formal methods tools. The credible autocoding framework developed is targeting a certain type of convex optimization problems and the high-level requirements formalized are appropriately chosen. However, if additional high-level requirements are needed, some manual formalization are needed. Although high-level requirements formalization can be complex, the struggle during this task is to formalize the low-level mathematical types and predicates needed. Hence, this task being already done, the same mathematical foundations can be used and the formalization of additional high-level requirements within the credible autocoder is highly simplified.

A numerical analysis of the method has been presented, showing how to propagate the errors due to floating-point calculations through the operations performed by the program. A modified version of the algorithm was presented, allowing us to compute "reasonable" *a priori* bounds on floating-points errors. However, the numerical analysis performed remains for now purely manual. Its correctness depends on the exactitude of equations obtained manually and no verification tools was used for this part. Future work shall include the use of formal methods to validate the numerical analysis.

Acknowledgments

This work was partially supported by projects ANR ASTRID VORACE, ANR FEANICSES ANR-17-CE25-0018, and NSF CPS SORTIES under grant 1446758. The authors would also like to thank Pierre Roux from ONERA for reviewing this work, and Arkadi Nemirovski for sharing how the Ellipsoid method can be modified to keep the ellipsoid's condition number bounded.

References

[1] Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., and Morari, M., "Embedded Online Optimization for Model Predictive Control at Megahertz Rates," *IEEE Trans. Automat. Contr.*, Vol. 59, No. 12, 2014, pp. 3238–3251. doi:10.1109/TAC.2014.2351991, URL http://dx.doi.org/10.1109/TAC.2014.2351991.

[2] Açikmese, B., III, J. M. C., and Blackmore, L., "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Trans. Contr. Sys. Techn.*, Vol. 21, No. 6, 2013, pp. 2104–2113. doi:10.1109/TCST.2012.2237346, URL http://dx.doi.org/10.1109/TCST.2012.2237346.

[3] Blackmore, L., *The Bridge*, Liveright New York, 2016. URL https://www.nae.edu/File.aspx?id=164381.

[4] Boyd, S., and Vandenberghe, L., *Convex optimization*, Cambridge University Press, New York, NY, USA, 2004.

[5] Mattingley, J., and Boyd, S., "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, Vol. 13, No. 1, 2012, pp. 1–27.

[6] Patrinos, P., Guiggiani, A., and Bemporad, A., "A dual gradient-projection algorithm for model predictive control in fixed-point arithmetic," *Automatica*, Vol. 55, 2015, pp. 226–235.

[7] Feron, E., "From Control Systems to Control Software," *Control Systems, IEEE*, Vol. 30, No. 6, 2010, pp. 50 –71. doi: 10.1109/MCS.2010.938196.

[8] Champion, A., Delmas, R., Dierkes, M., Garoche, P.-L., Jobredeaux, R., and Roux, P., "Formal methods for the analysis of critical control systems models: Combining non-linear and linear analyses," *International Workshop on Formal Methods for Industrial Critical Systems*, Springer, 2013, pp. 1–16.

[9] Herencia-Zapana, H., Jobredeaux, R., Owre, S., Garoche, P.-L., Feron, E., Perez, G., and Ascariz, P., "PVS linear algebra libraries for verification of control software algorithms in C/ACSL," *NASA Formal Methods Symposium*, Springer, 2012, pp. 147–161.

[10] Roux, P., Jobredeaux, R., Garoche, P.-L., and Féron, É., "A generic ellipsoid abstract domain for linear time invariant systems," *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, ACM, 2012, pp. 105–114.

[11] Wang, T., Jobredeaux, R., Pantel, M., Garoche, P.-L., Feron, E., and Henrion, D., "Credible autocoding of convex optimization algorithms," *Optimization and Engineering*, Vol. 17, No. 4, 2016, pp. 781–812.

[12] Wei, H., "Numerical stability in linear programming and semidefinite programming," 2006.

[13] Cohen, R., Davy, G., Feron, E., and Garoche, P.-L., "Formal Verification for Embedded Implementation of Convex Optimization Algorithms," *IFAC-PapersOnLine*, Vol. 50, No. 1, 2017, pp. 5867–5874.

[14] Blackmore, L., Acikmese, B., and Scharf, D. P., "Minimum-landing-error powered-descent guidance for Mars landing using convex optimization," *Journal of guidance, control, and dynamics*, Vol. 33, No. 4, 2010, pp. 1161–1171.

[15] Borrelli, F., Falcone, P., Keviczky, T., Asgari, J., and Hrovat, D., "MPC-based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Autonomous Systems*, Vol. 3, No. 2, 2005, pp. 265–291.
[16] Floyd, R. W., "Assigning Meanings to Programs," *Proceedings of Symposium on Applied Mathematics*, Vol. 19, 1967, pp. 19–32.
[17] Hoare, C. A. R., "An axiomatic basis for computer programming," *Commun. ACM*, Vol. 12, 1969, pp. 576–580.
[18] Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., and Yakobowski, B., "Frama-C: a software analysis perspective," Springer, 2012, pp. 233–247.
[19] Baudin, P., Filliâtre, J.-C., Marché, C., Monate, B., Moy, Y., and Prevosto, V., "ACSL: ANSI/ISO C Specification Language. Version 1.11." http://frama-c.com/download/acsl.pdf, 2016.
[20] Conchon, S., Contejean, E., and Iguernelala, M., "Canonized Rewriting and Ground AC Completion Modulo Shostak Theories : Design and Implementation," *Logical Methods in Computer Science*, Vol. 8, No. 3, 2012.
[21] Frama-C, 2019. URL https://frama-c.com/wp.html.
[22] Grötschel, M., Lovász, L., and Schrijver, A., "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, Vol. 1, No. 2, 1981, pp. 169–197.
[23] Bland, R. G., Goldfarb, D., and Todd, M. J., "The ellipsoid method: A survey," *Operations research*, Vol. 29, No. 6, 1981, pp. 1039–1091.
[24] Nemirovski, A., *Introduction to Linear Optimization*, Lecture notes, Georgia Institute of Technology, 2012.
[25] Boyd, S. P., and Barratt, C. H., *Linear controller design: limits of performance*, Prentice Hall Englewood Cliffs, NJ, 1991.
[26] Khachiyan, L. G., "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics*, Vol. 20, No. 1, 1980, pp. 53–72.
[27] Rump, S. M., "Verification of positive definiteness," *BIT Numerical Mathematics*, Vol. 46, No. 2, 2006, pp. 433–452.
[28] Rump, S. M., "Error estimation of floating-point summation and dot product," *BIT Numerical Mathematics*, Vol. 52, No. 1, 2012, pp. 201–220.
[29] Roux, P., Jobredeaux, R., and Garoche, P., "Closed loop analysis of control command software," *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, 2015, pp. 108–117. doi:10.1145/2728606.2728623, URL http://doi.acm.org/10.1145/2728606.2728623.
[30] Goubault, E., and Putot, S., *Static Analysis of Finite Precision Computations*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 232–247. doi:10.1007/978-3-642-18275-4_17, URL https://doi.org/10.1007/978-3-642-18275-4_17.
[31] Putot, S., Goubault, E., and Martel, M., *Static Analysis-Based Validation of Floating-Point Computations*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 306–313. doi:10.1007/978-3-540-24738-8_18, URL https://doi.org/10.1007/978-3-540-24738-8_18.
[32] El Ghaoui, L., "Inversion error, condition number, and approximate inverse of structured matrices," *Linear Algebra and its Applications*, Vol. 342, No. 1–3, 2002.
[33] Quanser, "3-DOF Helicopter Reference Manual," , n.d. URL https://www.lehigh.edu/~inconsy/lab/frames/experiments/QUANSER-3DOFHelicopter_Reference_Manual.pdf.