2013

Provable Run Time Safety Assurance for a Non-Linear System

Cory Firmin Snyder
Wright State University

PROVABLE RUN TIME SAFETY ASSURANCE FOR A NON-LINEAR SYSTEM

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Engineering

By

CORY FIRMIN SNYDER
B.S., Wright State University, 2012

2013
Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

May 10, 2013

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Cory Firmin Snyder ENTITLED Provable Run Time Safety Assurance for a Non-linear System BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

Matthew Clark, M.S.Egr.
Thesis Co-Director

Kuldip S. Rattan, Ph.D.
Thesis Co-Director

Kefu Xue, Ph.D.
Chair, Department of Electrical Engineering

Committee on
Final Examination

Kuldip S. Rattan, Ph.D.

Xiaodong Zhang, Ph.D.

Matthew Clark, M.S.Egr.

R. William Ayres, Ph.D.
Interim Dean, Graduate School

ABSTRACT

Snyder, Cory Firmin. M.S.Egr. Department of Electrical Engineering, Wright State University, 2013. Provable Run Time Safety Assurance for a Non-linear System.

Systems that are modeled by non-linear continuous-time differential equations with uncertain parameters have proven to be exceptionally difficult to formally verify. The past few decades have produced a number of useful verification tools which can be applied to such systems but each is applicable to only a subset of possible verification scenarios. The Level Sets Toolbox (LST) is one such tool which is directly applicable to non-linear systems, however, it is limited to systems of relatively small continuous state space dimension. Other tools such as PHAVer and the SpaceEx invariant of the Le Guernic-Girard (LGG) support function algorithm are specifically designed for hybrid systems with linear dynamics and linear constraints but can accommodate hundreds of continuous states. The application of these linear reachability tools to non-linear models has been achieved by approximating non-linear systems as linear hybrid automata (LHA). Unfortunately, the practical applicability and limitations of this approach are not yet well documented. The purpose of this thesis is to evaluate the performance and dimensionality limitations of PHAVer and the LGG support function algorithm when applied to a LHA approximation of a particular non-linear system. A collision avoidance scenario with autonomous differential drive robots is used as a case study to demonstrate that an over-approximated reachable set boundary can be generated and implemented as a run time safety assurance mechanism.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

In many applications, there are serious consequences when a system fails to meet its design requirements. For example, the failure of a safety critical system such as an aircraft or medical device has the potential to threaten human lives. Regardless of the particular application, though, it is always desirable to design systems which perform as intended. It is therefore useful to have the capability of proving whether systems meet applicable specifications, over an entire range of possible operating conditions. Unfortunately, the growth in the complexity of systems has out-paced the development of techniques for verification and validation. Today there is no standard tool or standard approach which can be applied to the task of verifying arbitrary systems. Instead, there are a diverse collection of tools and methods which are each applicable for certain types of verification problems, with certain types of models, with certain limitations on accuracy, and with certain sets of assumptions. Moreover, the cumulative capability of these tools still falls well short of that needed to formally verify many systems which would be considered simple in a modern scope.

There are a wide array of models that are used to describe systems, ranging in levels of abstraction, types of dynamics, and degrees of complexity. This thesis focuses on the verification of systems with dynamics that are modeled by ordinary differential equations (ODEs). At the lowest level of abstraction, many physical systems fit this niche. These systems are particularly interesting because of mathematical properties which make their analysis exceptionally difficult. For example, it is only possible to find explicit solutions for a very limited subset of possible ODEs. Even when an explicit solution can be obtained, it is usually not possible to analytically derive general properties of an ODE for ranges of inputs and initial conditions.

Control systems which are modeled at the level of ODEs are almost always designed for inner-loop tracking or regulation. Since inherent lag between system input and output makes it impossible to achieve perfect tracking and regulating performance for causal systems, specifications are used to define thresholds of acceptable performance. For example, specifications on rise-time, settling-time, percentage overshoot, and steady-state error are often used to define the minimum requirements that an inner-loop control system must satisfy in order to effectively interface with outer-loop systems. In certain restricted cases, closed-form equations can be used to calculate these metrics but generally specifications are verified by simulating a trajectory of the system for particular inputs and initial conditions. For systems that are modeled by linear ODEs, individual simulations scale linearly between input and output and therefore a single simulation reveals properties of a system's output for all inputs that can be expressed as a linear function of the simulated input. In other words, a single simulation can reveal global properties of a system's response when the system is linear. Unfortunately, the same is not generally true for systems with dynamics that are modeled by non-linear ODEs or for systems which are modeled with parameter uncertainties.

In order to verify that a system modeled with uncertain parameters meets specifications, the output of the system would need to be simulated over the entire range of possible operating conditions. Even for systems of relatively low state dimension, the number of simulations that would need to be performed in order to make reasonable verification conclusions could be beyond practical. If a large number of simulations were able to be executed within a reasonable time, it would still be impossible to generate an absolute verification proof with a finite number of simulations, each having a different set of parameters. The most popular approach that has been conceived for dealing with this problem is the reachability computation. Reachability computation produces essentially the same result as exhaustive simulation but instead of computing many individual simulations to cover the entire range of operating conditions, a reachability computation generates the set of all possible states that are reachable within each time step under any possible combinations of inputs and initial conditions. In a concise interpretation, reachability computation is essentially set-based simulation. A forwards reachable set (FRS) is a set generated from a reachability computation that is carried out as time propagates forward and a backwards reachable set (BRS) is a set which is generated from a reachability computation in which time propagates backwards. Whereas individual simulations can only provide verification proofs for a finite set of possible operating conditions, with this approach it is possible to generate a definitive proof as to whether all conceivable trajectories meet a set of specifications.

Although reachability is unarguably a convenient concept, the concept is useless without knowledge of efficient algorithms to actually calculate reachable sets for arbitrary systems, or at least for subsets of possible systems. For time-invariant linear systems, a number of different algorithms have been proposed and for all but systems of extreme dimension, the problem can be considered solved. In fact, when implemented on modern personal computers, the best of these algorithms are capable of generating tight over-approximations of reachable sets for linear systems with hundreds of states, over thousands of time steps, in only a few seconds. These algorithms will be discussed in more detail in this thesis. Unfortunately, the same algorithms are not directly applicable to non-linear systems. The most popular approach that has been proposed for extending the linear reachability framework to the application of non-linear systems requires approximating a non-linear ODE model by a piecewise continuous linear system, or more generally a linear hybrid automata (LHA).

A LHA is a modeling formalism for systems which have both continuous and discrete states. That is, each discrete mode of an automata is associated with a particular linear ODE which dictates how the system's continuous states evolve while the system is in that particular discrete mode. Each discrete mode is also associated with transition guards which are defined by inequalities over the continuous states and which govern when the discrete mode of the system changes. The utility of the LHA formalism for the reachability analysis of non-linear systems is that it allows them to be expressed as piecewise-continuous linear systems with bounded modeling error in each discrete mode. Since efficient reachability algorithms already exist for linear time invariant systems, the same algorithms can be applied to the LHA model within each discrete mode. The modeling error can be treated as an uncertain input. When the reachable set of a LHA is generated in this manner, it must be intersected with the transition guards of the current discrete mode at each iteration of the simulation. If the reachable set is found to intersect a transition guard, special accommodations must be made in order to transition the reachable set from one discrete mode to the next while maintaining a conservative, tight over approximation of the actual reachable set for the original non-linear system model. A few approaches to handling the transition problem have been developed but none are suitably efficient to handle the verification of general non-linear models.

The main contribution of this thesis is a case study which provides a detailed explanation as to how a non-linear system can be hybridized and how reachability tools can be used to calculate an over-approximation of the original non-linear system's reachable set. Additionally, the results provide insight into the applicability and limitations of applying tools designed for hybrid automata to the problem of generating reachable sets for non-linear systems modeled with uncertain parameters. A collision avoidance scenario designed for two differential drive robots was chosen for the case study because it provides a relatively simple non-linear model which has practical application. PHAVer and the LGG support function algorithm were the verification tools of choice because of claims that they are capable of calculating reachable sets for systems with hundreds of continuous states. PHAVer is also of particular interest because of its use of exact integer arithmetic to guarantee that calculated reachable sets are an over-approximation of the actual reachable set of the non-linear system. A run time assurance simulation was implemented with the Simplex architecture to verify that the results generated with PHAVer were indeed an over-approximation. The simulation also demonstrates a practical application for reachable sets generated by these reachability tools.

The outline of the rest of this thesis is as follows: Chapter 2 provides a review of recent literature which is closely tied to the research presented in this thesis; Chapter 3 gives a limited introduction to mathematical concepts that are critical to understanding the work; Chapter 4 provides a high-level overview of the case study scenario, details of the Simplex architecture implementation, and the derivation of the differential drive robot model; Chapter 5 gives details as to how the non-linear ODE model is hybridized, explains how the reachability computation is configured for each tool, and presents the reachable sets that were successfully generated with each approach; Chapter 6 discusses how the resulting reachable set is integrated with the Simplex architecture and the results of the run time assurance simulation; finally, Chapter 7 summarizes the work and presents potential areas of future research.

II. LITERATURE REVIEW

The problem of computing reachable sets has been a hot research topic for nearly two decades. In that time, numerous tools and techniques have been developed to push the envelope of capabilities within this domain. A subset of the developed tools which seem to mark major milestones in the evolution of reachability calculation are outlined here.

The earliest research that focused on the verification of hybrid systems was developed from a computer science perspective by Henzinger, et al. [1]. This research resulted in the creation of HyTech, a software tool capable of verifying reachability properties as well as temporal properties of systems modeled as LHA. A LHA is a hybrid automata model with linear constraints on the variables and their derivatives, including constant differential equations, rectangular differential inclusions, or rate comparisons. The original version of HyTech was implemented within Mathematica and used exact computation without numerical restrictions, however, the execution speed of the Mathematica version was prohibitively slow for reachability calculations of relative complexity. Later versions of the tool were implemented in c++ to increase the speed but these versions suffered from numerical restrictions. Without exact computations, the presence of precision errors makes it impossible to ensure that calculated reachable sets are indeed an over-approximation of the actual reachable set of a system. Note that an over-approximation of a reachable set is guaranteed to contain all possible trajectories of the system and may contain trajectories that do not belong to the actual reachable set of a system. For verification, an over-approximation is always more desirable than an under-approximation because it will not lead to a conclusion that the system is safe when it is actually not.

A tool named *d/dt* followed HyTech and provided the capability to generate reachable sets for a wider class of hybrid automata [2]. Specifically, *d/dt* can work with hybrid automata that have continuous linear dynamics with uncertain bounded inputs of the form $\dot{x} = Ax + Bu$, where u ranges inside a convex polyhedron. The tool computes transition successors using polytopes and stores reachable sets as non-convex orthogonal polyhedra. The ability to consider systems with affine dynamics and uncertain inputs makes it possible to more accurately over-approximate the dynamics of non-linear systems. Unfortunately *d/dt* does not use exact computations, which means that the generated approximations of the reachable set are not necessarily an over-approximation of the actual reachable set.

The first tool to implement an efficient reachability calculation with exact and robust arithmetic was PHAVer [5]. That is, the results generated by PHAVer are formally sound due to its use of unbounded integer arithmetic and guaranteed over approximations. If analysis with PHAVer terminates, the output is guaranteed to be an over-approximation of the actual reachable set of the system. Although PHAVer's baseline reachability algorithm is mechanized for LHA, it is also capable of automatically over-approximating piecewise affine dynamics as LHA and subsequently computing the reachable set. In order to avoid the crippling affects of increasing set representation size and complexity, PHAVer uses conservative limiting of bits and constraints in its polyhedral computations. The computation uses a half-space polytope representation.

The creators of PHAVer subsequently developed another tool that is referred to here as the *Le Guernic-Girard (LGG) support function algorithm* [6]. As its name suggests, the primary advantage of this algorithm is its use of support function set representations to increase the efficiency of calculations. The tool actually uses a combination of support functions and polytopes to support all necessary set operations. Another advantage in regards to both efficiency and accuracy is its ability to use variable time steps in its calculations, which allows it to maintain desired error margins. Like *d/dt*, the *LGG support function algorithm* is applicable to hybrid automata with affine dynamics within each discrete mode and can accommodate bounded uncertainty in the system's inputs. Unfortunately, the *LGG support function algorithm* is not formally sound since it uses double precision floating point computations to represent sets of states and to solve differential equations. Although the tool does not guarantee an over-approximation of the reachable set, its results can be useful in verifying and debugging general properties of a system's behaviour.

With the release of the *LGG support function algorithm*, the authors also released an extensible verification platform for hybrid systems known as *SpaceEx*. The platform is robust and user-friendly. For example, it provides a web-based interface for configuring reachability calculations and viewing generated reachable sets. *SpaceEx* permits users to perform reachability calculations with either *PHAVer* or the *LGG support function algorithm*, or to simulate individual trajectories of a system.

An approach to reachability calculation which is entirely different from the others described here uses Level Set Methods to generate reachable sets [12][14][11]. The Level Sets Toolbox is a tool which is applicable to a wide range of surface propagation problems and which has all necessary features to completely automate approximation of reachable sets [13]. The key feature of this approach is that it encodes the avoid set or the initial set as a surface in a space with dimension one greater than the system's state space dimension. When the system is not modeled with uncertainty, the level sets approach resolves to simply propagating the surface over a grid, under a constant vector field. When the system does have modeled uncertainty, a Hamilton-Jacobi Isaacs formulation is used to determine the most conservative vector field for the particular scenario, at each time step. This approach is directly applicable to both linear and non-linear systems but is limited by exponential growth in calculation time with increasing state space dimension. Reachability calculations for only three to six continuous states are feasible with current computing resources.

KeYmaera is one of the newest tools available for verifying hybrid automata [15]. The verification approach implemented in *KeYmaera* is vastly different from the others described here since it uses theorem proving to verify requirements rather than direct reachability computation. *KeYmaera* has already demonstrated significant capabilities, but details of its implementation and discussion of its applications are beyond the scope of this thesis.

Although reachability tools have come a long way, they are still far from being applicable to many real-world systems. The Simplex architecture provides an approach for ensuring that a system will not violate critical safety or performance criteria when the system itself cannot be directly verified [16]. The Simplex architecture can be defined as a formal approach for implementing and testing experimental controllers while bounding their operation with a safety controller to guarantee safety or performance. The decision as to when the system should switch between controllers is often implemented by using the reachable set of the safety controller to determine the limits of the recovery envelope. Because the switching decisions are made while the system is operational, this scheme can be classified as a run-time assurance mechanism. The negative consequence of this approach is that it adds overhead to the system's processor during operation.

The contribution of this thesis is not the development of a new reachability tool or the extension of an existing tool. Instead, this work fills a perceived void in the literature by documenting details of how reachability tools can be used to generate reachable sets for non-linear systems and how reachable sets can be used with the Simplex architecture to verify relatively complex scenarios. Although not necessarily the most recent or advanced reachability tools, PHAVer and the LGG support function algorithm were the tools of choice for this application due to the claims that they can be applied to systems with hundreds of continuous state variables. PHAVer is of particular interest because of its ability to generate guaranteed over-approximations of reachable sets. An additional product of this work is new insight into the applicability and limitations of applying these reachability tools to the verification of non-linear systems via hybridization.

III. MATHEMATICAL BACKGROUND

A decade of collaborative research has yielded multiple effective approachs to calculating reachable sets for linear systems [1][2][5][6][14]. Although each tool has unique features for improving efficiency and accuracy, most of the popular tools have similar implementations. This chapter explains a general version of the baseline reachability calculation for linear systems and how the linear reachability algorithms are extended for application to LHA and hybrid automata with affine dynamics in each discrete mode. Although this chapter does not provide a complete mathematical background on this subject, it does provide enough detail to understand the work in this thesis. Those who wish to develop a more granular understanding are referred to [8],[10], from which much of this background was derived.

3.1 The Concept of Reachability

Before diving into details of how reachable sets are calculated, the concept of a reachable set should be clearly defined. In a concise, non-mathematical definition, the reachable set of a system is the set of all states that the system could reach at some time within a certain range of time, subject to bounds on inputs, initial conditions, and any uncertain parameters. For an autonomous system with no uncertainty, the reachable set is a single trajectory and can be found by simply simulating the system's dynamics.

Let the state space of a system be given by $X \subseteq \mathbb{R}^n$, the input space by $U \subseteq \mathbb{R}^m$, and the time domain by $T = \mathbb{R}_+$. The dynamics of a system $S = (X, U, f)$ are given by the differential equation

$$\dot{\mathbf{x}} = f(X, U). \tag{3.1.1}$$

Let the function $\varepsilon : T \to X$ represent a trajectory of the system. Trajectories may be defined for all T or for $I \subset T$. Likewise, let input signals be represented by the function $\zeta : T \to U$. The set of all possible system trajectories is denoted by X^T and the set of all possible input signals is denoted by U^T. It can then be stated that ε is the response of S to ζ from x_0, when ε is the solution of equation (3.1.1) for an input of $\zeta = u(t)$ and initial state x_0. If $\varepsilon(\tau) = x'$, where $\tau \in T$ and $x' \in X$, then it can be said that S reaches x' from x_0, subject to ζ, at time τ.

Let the function $R : f, X_0, U^T, I \to X^T$ be defined as the reachability operator. The reachability operator can be applied as

$$R(f, x_0, \zeta, \tau) = x' \tag{3.1.2}$$

to a single initial condition, a single input signal, and a single time instant. As a result, the reachability operator in equation (3.1.2) maps the system parameters to a single state along the system's trajectory.

When the reachability operator is applied to a set of initial conditions, a set of inputs, and a range of time as

$$R(f, X_0, U^T, I) = \bigcup_{x_0 \in X_0} \bigcup_{\zeta \in U^T} \bigcup_{\tau \in I} R(f, x_0, \zeta, \tau), \tag{3.1.3}$$

the reachability operator produces a set of reachable states, or a reachable set.

More precisely, this mathematical definition of reachability defines the notion of a forwards reachable set (FRS). The formalism of a BRS is identical, with the exception of the fact that it's time domain is $T = \mathbb{R}-$. A more thorough discussion on the concept of reachability can be found in [10].

3.2 Representing Sets

As acknowledged earlier, the concept of a reachable set is convenient but it is useless if we cannot find an expression for the reachable set which can be directly applied to solving verification problems. For example, the reachable set of a system can be defined with respect to X_0, U, f, and T as it was in the previous section but utilizing this representation of the set resolves to solving the reachability problem itself. Therefore, it is important to specify that the computation of reachable sets involves finding a representation for the reachable set which is closed under operations which are useful for verification. Any class of syntactic set representations C that is to be used in the reachability calculation must satisfy at least the following three properties which are defined in [10]:

1. Every set $S \in C$ must have a finite representation.

2. For any set $S \in C$ and any point $x \in \mathbb{R}^n$, it must be possible to determine whether $x \in S$.

3. Each set operation $\circ$ that is required for reachability calculation must be computable and closed over the class of sets; i.e if $S_1, S_2 \in C$ then $S_1 \circ S_2 \in C$.

Also important to the actual implementation of the reachability algorithms is the accuracy with which an arbitrary set can be over-approximated by a set $S \in C$ and the efficiency that can be achieved in the computation of necessary set operations. The following section discusses the set operations which are necessary for reachability computation and also introduces and compares different classes of sets which have proven to be useful in the implementation of reachability computations.

3.2.1 Set Operations

Affine Transformation

Consider the state equation of a discrete-time linear system

$$x_{k+1} = Ax_k + Bu_k, \tag{3.2.4}$$

where $A \in \mathbb{R}^n \times \mathbb{R}^n$ is the discrete state matrix, x_k is the discrete state, $B \in \mathbb{R}^n \times \mathbb{R}^m$ is the input matrix, and $u_k \in \mathbb{R}^m$ is the input vector. We say that x_{k+1} is an affine transformation of x_k.

Definition 3.2.1. *(affine transformation) An affine transformation is equivalent to a linear transformation followed by a translation.*

To simulate the discrete trajectory of the system, we would simply apply the affine transformation at each time step, including particular inputs and initial conditions. For set-based simulation, or reachability calculation, the basic approach is essentially the same, except the transformation is performed on sets rather than single states. Therefore, in order to perform reachability calculations we must have a set representation which is closed under affine transformations.

Minkowski Sum

The Minkowski sum is an operation which defines geometric, or set-based, addition.

Definition 3.2.2. *(Minkowski sum) The Minkowski sum of two sets, $S_1, S_2 \subseteq \mathbb{R}^n$ is defined as*

$$S_1 \oplus S_2 = \bigcup_{x_1 \in S_1} \bigcup_{x_2 \in S_2} x_1 + x_2. \tag{3.2.5}$$

In words, the Minkowski sum of two sets is the set of states that results from the vector addition of each point in one set with each point in another set. In reachability calculations, the Minkowski sum is useful for adding error balls to approximated reachable sets and also in accounting for the effect of parameter uncertainty. A simple geometric representation of the concept of the Minkowski sum is illustrated in Figure 3.2.1.

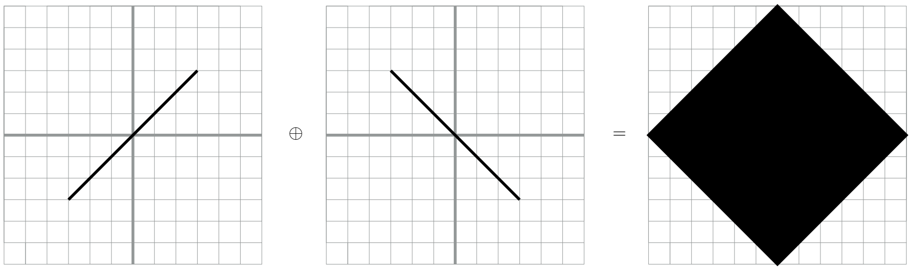


Figure 3.2.1: Illustration of a simple two-dimensional Minkowski sum.

Convex Union

It is generally desirable to work with convex sets because they are usually easier to parametrize and to manipulate than non-convex sets. Unfortunately, the union of two sets may be non-convex even if the operands are convex. For this reason, it is more practical to search for set representations that are closed under the convex hull of a union operation than to search for representations which are closed under a union operation.

Definition 3.2.3. *(convex set) A set S is convex iff all points along a line segment between any two points in the set are also contained in the set. That is, S is convex iff*

$$(1-\lambda)x_1 + \lambda x_2 \in S \qquad \forall x_1, x_2 \in S, \quad \forall \lambda \in [0,1] \tag{3.2.6}$$

Definition 3.2.4. *(convex hull) The convex hull C of a non-convex set S is the smallest possible convex set such that $x \in S \implies x \in C$.*

The concept of a convex hull is illustrated in Figure 3.2.2. The convex hull is useful for approximating non-convex sets by convex sets that are minimal over-approximations. The convex hull of a union operation is particularly useful for combining results of reachable set calculations from multiple simulation time steps and from multiple flow pipes of a hybrid reachability calculation.

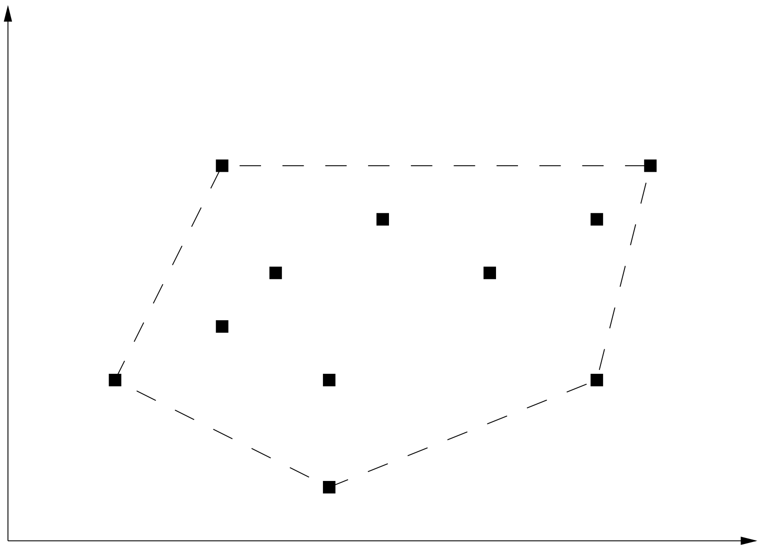


Figure 3.2.2: The convex hull of a finite set of points.

Intersection with a Hyperplane

When calculating the reachable set of a LHA, a necessary operation is the intersection of the reachable set with transition guards. Transition guards are defined as a grid of hyperplanes over the state space and are used to dictate the discrete transitions of the automata. It is important that the set representation that is chosen for calculating the reachable set of a hybrid automata be closed under the operation of intersection with a hyperplane.

3.2.2 Set Representations

Boxes

A box is perhaps the simplest possible representations of a set.

Definition 3.2.5. *(box) A set β is a box iff its membership can be defined by intervals over each dimension of $\mathbb{R}^n$, that is*

$$\beta = [\underline{x_1}, \overline{x_1}] \times ... \times [\underline{x_n}, \overline{x_n}]. \tag{3.2.7}$$

A point x belongs to β iff $\underline{x_i} \geq x_i \leq \overline{x_i}$.

The box representation has the advantage of having a small parametrization size of just $2n$. Unfortunately, the small representation size comes at the cost of a severely limited capability for representing over-approximations of arbitrary sets. Of the set operations mentioned, the box representation is only closed under the Minkowski sum operation.

Definition 3.2.6. *(interval hull) The interval hull of a set S, denoted by $\square(S)$ is the smallest possible box such that $x \in S \implies x \in \beta$. That is,*

$$\square(S) = [\underline{x_1}, \overline{x_1}] \times ... \times [\underline{x_n}, \overline{x_n}], \tag{3.2.8}$$

where $\underline{x_i} = inf\{x_i : x_i \in S\}$ and $\overline{x_i} = sup\{x_i : x_i \in X\}$.

A two dimensional illustration for the concepts of the interval hull and the box representation are given in Figure 3.2.3.

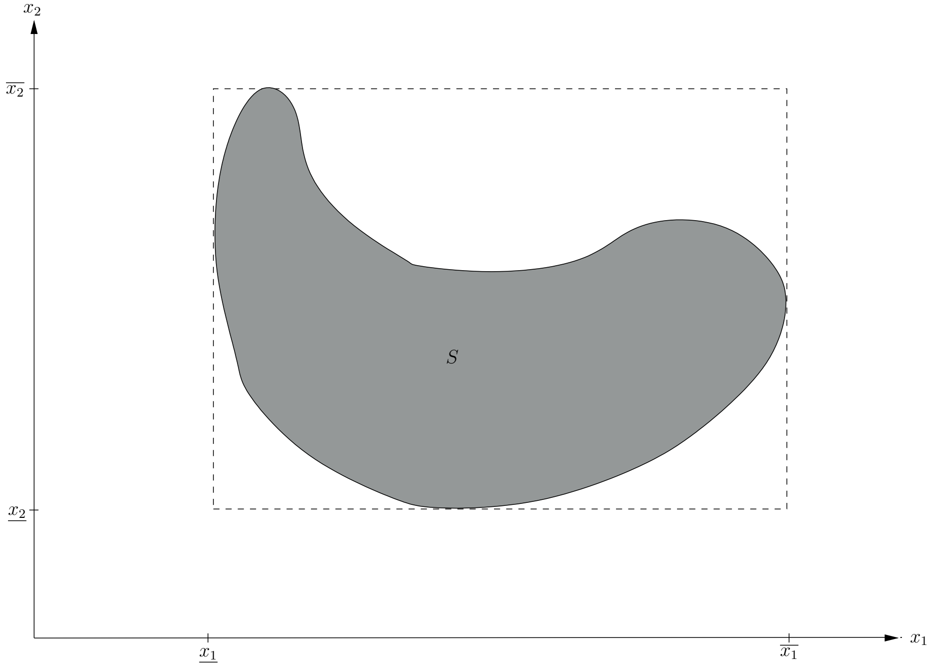


Figure 3.2.3: Illustration of a box as the interval hull of a set S.

Ellipsoids

An ellipsoid is a convex, symmetric set which can be defined by its center c and a positive definite shape matrix Q.

Definition 3.2.7. *(ellipsoid) An ellipsoid $E(c, Q)$ with center c and shape matrix Q is defined by the equation*

$$E(c, Q) = \{x : (x - c)Q^{-1}(x - c) \leq 1\}. \tag{3.2.9}$$

The ellipsoid representation is closed under affine transformations and intersection with a hyperplane but is not closed under the Minkowski sum or the convex hull of union operations. The representation size of an ellipsoid is $n^2 + n$.

Polytopes

A polytope is a convex, bounded geometric object of arbitrary dimension which has planar sides. A polygon is a 2-dimensional polytope and a polyhedron is a 3-dimensional polytope.

Definition 3.2.8. *(polytope) A polytope $\mathcal{P}$ can be defined by the bounded intersection of a finite number of half-spaces, that is*

$$\mathcal{P} = \bigcap_{h \in \mathcal{H}} h. \tag{3.2.10}$$

Definition 3.2.9. *(half-space) A half-space h is defined as a linear inequality*

$$h = \{x : x \cdot v \leq c\} \tag{3.2.11}$$

for some normal vector v and some constant c.

Notice that in order to be bounded, it is required that a polytope of dimension n be defined as the intersection of a minimum of $n+1$ half-spaces. Figure 3.2.4 illustrates the concept of a half-space polytope in 2 dimensions with 3 constraints.

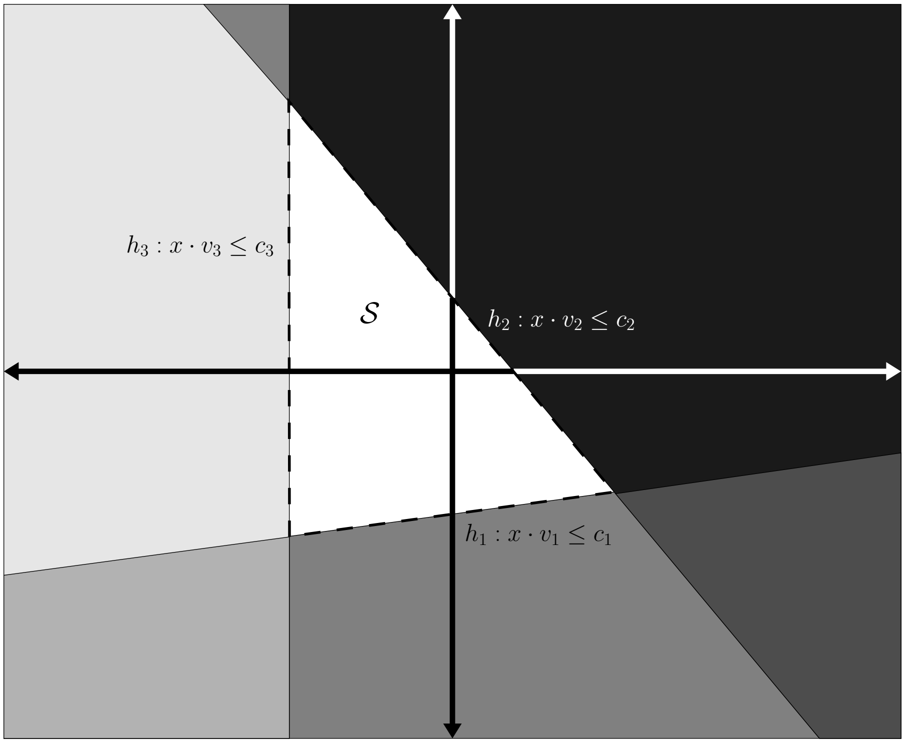

$h_3 : x \cdot v_3 \leq c_3$
$\mathcal{S}$
$h_2 : x \cdot v_2 \leq c_2$
$h_1 : x \cdot v_1 \leq c_1$

Figure 3.2.4: Illustration of a set $\mathcal{S}$ represented as a half-space polytope.

Alternative to the half-space representation, it is also possible to represent a polytope as the convex hull of a set of vertex points $\mathcal{V}$. Mathematically, the vertex polytope is defined by

$$\mathcal{P} = \{\sum_{v \in \mathcal{V}} a_v v : \forall v \in \mathcal{V}, a_v \geq 0 \ and \sum_{v \in \mathcal{V}} a_v = 1\}. \tag{3.2.12}$$

Figure 3.2.2 illustrates the concept of a polytope defined as the convex hull of a set of vertices.

Although both representations describe the same class of geometric objects, it turns out that each has its own advantages and disadvantages in regards to the efficiency of operations that can be directly computed. The half-space representation easily permits intersection with a hyperplane and in most cases affine transformations are also simple. For the vertices representation, an affine transformation amounts to simply applying the transformation to each of the vertices. The convex union of a polytope defined in vertices representation can be performed by taking the union of the vertex points. Unfortunately, there are no known algorithms for efficiently computing the Minkowski sum of polytopes for either representation. The representation size for half-space polytopes is $kd + k$ and kd for polytopes defined by their vertices, where k is the number of constraints. The major disadvantage of the polytope representations is that the representation size grows very quickly as a result of union operations and Minkowski sum operations.

Zonotopes

Zonotopes are a special sub-class of symmetric polytopes which are defined by a Minkowski sum of line segments.

Definition 3.2.10. *(zonotope) A zonotope is defined as the Minkowski sum of a set $\mathcal{G}$ of line segments known as generators and its center vector c. That is,*

$$\mathcal{Z} = \{c + \bigoplus_{g \in \mathcal{G}} a_g g : \forall g \in \mathcal{G}, -1 \leq a_g \leq 1\}. \tag{3.2.13}$$

Zonotopes are a useful class of polytopes for reachability calculations due to the fact that they are closed under the Minkowski sum; a property which is not satisfied by either half-space or vertices polytopes. In fact, the combinatorial explosion of the Minkowski sum is eliminated since the zonotope definition restricts possible sets to those that can be composed by a Minkowski sum of generating vectors. The size of a zonotope resulting from a Minkowski sum is just the sum of the sizes of the input zonotopes. Zonotopes are also closed under affine transformation but not under intersection with hyperplanes or convex hull of union operations. The representation size for a zonotope is $kn + k$.

Support Functions

All set representations that have been described thus far have been defined by geometric objects. The support function representation is unique in that set membership is determined by a function.

Definition 3.2.11. *(support function) The support function $\rho_{\mathcal{S}}$ of a set $\mathcal{S}$ is defined as*

$$\rho_{\mathcal{S}} : \gamma \to \sup_{x \in \mathcal{S}} x \cdot \gamma. \tag{3.2.14}$$

That is, the support function maps a unit direction vector γ to the the maximum value of $x \cdot \gamma$ for all $x \in \mathcal{S}$.

A support function can also be thought of as a half-space polytope defined by an infinite number of half-spaces, where each half space is given by

$$h = \{x : x \cdot \gamma \leq \rho_{\mathcal{S}}(\gamma)\}. \tag{3.2.15}$$

On a computer, it would be impossible to store and manipulate a support function represented as a half-space polytope with infinite constraints. Instead, the representation of a support function is maintained in a computer as an algorithm which computes the values of the function. It is generally very difficult to find a support function algorithm for an arbitrary set, however given a support function it is relatively easy to evolve the representation under the common set operations. For all matrices A, positive scalars λ, and vectors $\gamma \in \mathbb{R}^n$, the support functions resulting from affine transformation, scaling, Minkowski sum, and convex hull of union operations can be computed using the following properties which are derived in [8]:

$$\begin{aligned} A^T\rho_{\mathcal{S}}(\gamma) &= \rho_{\mathcal{S}}(A^T\gamma), & (3.2.16)\\ \lambda\rho_{\mathcal{S}}(\gamma) &= \rho_{\mathcal{S}}(\lambda\gamma), & (3.2.17)\\ \rho_{\mathcal{S}_1}(\gamma)\oplus\rho_{\mathcal{S}_2}(\gamma) &= \rho_{\mathcal{S}_1}(\gamma)+\rho_{\mathcal{S}_2}(\gamma), & (3.2.18)\\ CH\left(\rho_{\mathcal{S}_1}(\gamma)\bigcup\rho_{\mathcal{S}_2}(\gamma)\right) &= \max\left(\rho_{\mathcal{S}_1}(\gamma),\rho_{\mathcal{S}_2}(\gamma)\right). & (3.2.19)\end{aligned}$$

One class of sets for which support functions can be computed efficiently is the set of unit balls. When a unit ball is used to define initial or final sets for a reachability calculation, support functions for subsequent reachable sets can be computed efficiently using the aforementioned properties.

Definition 3.2.12. *(unit ball) The unit ball associated with the* k*-norm in dimension* n *is defined by*

$$\mathcal{B}=\left\{x\in\mathbb{R}^n : ||x||_k=\left(\sum_{i=0}^{n-1}|x_i|^n\right)^{1/n}\leq 1\right\}. \quad (3.2.20)$$

The support function of the unit ball associated with the k*-norm is given by*

$$\rho_{\mathcal{B}_k}(\gamma)=||\gamma||_{\frac{k}{k-1}}. \quad (3.2.21)$$

Level Sets

A level set is an implicit set representation that turns out to be very useful for reachability computations. To be precise, a level set is used to encode the boundary of a reachable set and a zero sub-level set is used to encode the membership of the set.

Definition 3.2.13. *(level set) For some function* $f:\mathbb{R}^n\to\mathbb{R}$ *and some constant* $c\in\mathbb{R}$*, a level set is defined as*

$$\mathcal{S}=\{x\in\mathbb{R}^n : f(x)=c\}. \quad (3.2.22)$$

Definition 3.2.14. *(zero sub-level set) For some function $f : \mathbb{R}^n \to \mathbb{R}$, a zero sub-level set is defined as*

$$\mathcal{S} = \{x \in \mathbb{R}^n : f(x) \leq 0\}. \tag{3.2.23}$$

A zero sub-level set representation of an arbitrary set can be designed by finding a function $f(x)$ such that the following implications are true: $f(x) \leq 0 \implies x \in \mathcal{S}$ and $f(x) > 0 \implies x \notin \mathcal{S}$. A common approach to designing the function $f(x)$ for the preceding scenario is by allowing $f(x)$ to represent the signed shortest distance between the point x and the boundary of the surface, $\partial\mathcal{S}$. Figure 3.2.5 illustrates the concept of a one-dimensional set $\mathcal{S}$ defined as the zero sub-level set of a signed distance function $f(x)$.

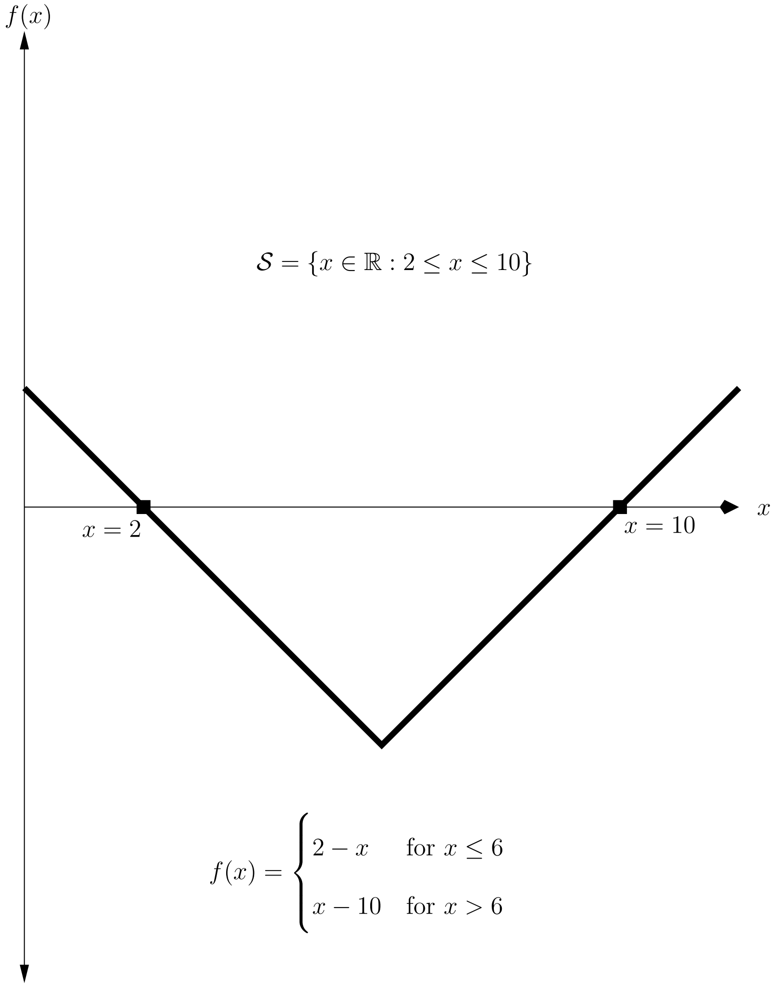


Figure 3.2.5: Illustration of a set $\mathcal{S}$ represented as a zero sub-level set of a function $f(x)$.

Although the level set representation is closed under all of the necessary operations, it is very difficult to compute a resultant closed-form function for any of them. The utility of this representation is different from those that have already been discussed because it has been applied to the reachability problem in a completely different way. Instead of reducing non-linear dynamics to piecewise linear approximations and using the Minkowski sum to account for error and uncertainty, the level set function is numerically propagated over a finite grid of points with non-linear or linear state dynamics and a Hamilton-Jacobi optimization is used to account for uncertainty. Level set reachability methods are not the focus of this thesis but the associated reachability computation is discussed in Section 3.5.

3.2.3 Comparison of Set Representations

None of the set representations that have been introduced permit easy computation of all of the operations that are useful for reachability computations. One approach that has been used to overcome this challenge is to convert between representations (by over-approximation) in order to take advantage of complementary properties of each. Other algorithms deal with this challenge by using restricted subsets of these representations in order to avoid complex calculations. Table 3.2.1, from [10], summarizes the difficulty of performing each operation on each type of set representation. A (+) indicates that the operation is easy, a (-) indicates that it is hard, a (*) indicates that it is usually easy, and an empty cell indicates that the operation is not possible. The representation size is given with respect to the dimension of the set n and the number of defining constraints k.

	Size	$A\cdot$	$\cdot \oplus \cdot$	$\mathrm{CH}(\cdot \cup \cdot)$	$\cdot \cap \mathcal{H}$
Boxes	$2n$		+		
Ellipsoids	$n^2 + n$	+			+
Halfspace Polytopes	$kn + k$	*	-	-	+
Vertice Polytopes	kn	+	-	+	-
Zonotopes	$kn + n$	+	+		
Support Functions	NA	+	+	+	-

Table 3.2.1: Comparison of set operations for each set representation [10].

3.3 Reachability Analysis for Linear Systems

The reachability calculation for autonomous discrete-time systems will be discussed first and later will be modified for autonomous continuous-time systems and for systems with inputs. As stated in the introduction, reachability analysis can be intuitively understood as set-based simulation or set-based integration. As with ordinary integrals, the additivity of integration on intervals is also applicable to set based integration and is utilized in reachability calculations to produce reachable sets at discrete time steps. In terms of the reachability operator, this property is expressed as

$$R(f, X_0, U^T, [0, t_1 + t_2]) = R(f, R(f, X_0, U^T, [0, t_1]), U^T, [t_1, t_2]). \tag{3.3.24}$$

Using this important property, a reachability calculation can be decomposed into many reachability calculations over subintervals of time. The following high-level algorithm demonstrates the utility of this principle for decomposing discrete-time reachability calculations into the union of multiple reachability calculations, each calculated over the sampling period of the system, r.

High-Level Reachability Algorithm

```
inputs: $(X_0, I)$
output: $N = R(X_0, I)$
begin
    $M := N := X_0$
    while $i = 1 : I/r$
        $M = R(M, [0, r])$
        $N = N \cup M$
    end
return $N$
```

The reachability calculation at each discrete time step is simply an affine transformation of the set. Recall that all of the set representations that were discussed were closed under affine transformation operations, with the exception of boxes. The details of performing an affine transformation for each set representation are outside the scope of this thesis but can be found in [8].

An autonomous discrete-time reachability calculation is relatively simple but solving the same problem with continuous-time dynamics adds significant complexity. The outstanding difference between continuous and discrete reachability calculations is the problem of computing the reachable states for the infinite number of time instances between discrete steps. This problem has been addressed in at least three ways:

1. The simplest approach is to make the discrete time step r sufficiently small so that subsequent discrete-time reachable sets overlap.

2. Another approach solves the problem by bloating the calculated sets. With the bloating approach, the convex hull of the initial set X_0 and the reachable set $X_r = R(f, X_0, [0, r])$ is bloated in order to account for the curvature of trajectories over the interval. Several different methods have been used to determine how much the convex hull must be bloated in order to guarantee inclusion of all possible trajectories over the interval. One method that can be applied to half-space polytope representations is pushing the constraining hyperplanes outward by a factor derived from a Taylor series approximation of the trajectories [3].

3. A third alternative for addressing this issue is to estimate the maximum distance between $R(X_0, [0, r])$ and any trajectory of the system over the discrete interval, and then add an error ball of appropriate size to $R(X_0, [0, r])$ using the Minkowski sum.

The last issue of complexity that remains to be discussed on the topic of calculating reachable sets for linear systems is the consideration of non-autonomous systems. Discrete time systems with input are of the form:

$$x_{k+1} = Ax_k + Bu_k, \tag{3.3.25}$$

where $x_0 \in X_0$ and $u_k \in U$ and X_0 and U are bounded convex sets. The modifications made here for discrete time systems with input can easily be extended to continuous-time systems using similar approaches to those used for systems without inputs. Only the discrete time version will be discussed here.

The set reachable from X_0 in a single discrete time step can be expressed as

$$R(f, X_0, U, r) = \{Ax + Bu : x \in \mathbb{R}^n, u \in U\} = AX \oplus BU. \tag{3.3.26}$$

In other words, the Minkowski sum can be used in a straightforward manner to accommodate for the effect of bounded inputs. It is worth noting here that the Minkowski sum operation introduces an effect known as the ”wrapping” effect when calculated for polytopes. The wrapping effect describes the trade-off between accuracy and representation size that is inherent with polytope Minkowski sum operations. The effect is caused by the fact that a Minkowski sum operation increases the number of constraints needed to represent the resulting set. Over many iterative steps, the representation size can become prohibitively large. The problem can be addressed by over-approximating the result of a Minkowski sum operation by a polytope with less constraints. Even this fix is not perfect though, since after many iterative steps the successive over approximation can make the result so conservative that it is useless. Balancing the wrapping effect and the effects of successive over-approximation is one of the major challenges to implementing efficient, scalable reachability calculations. The zonotope and support function representations are particularly appealing because they avoid the wrapping effect.

3.4 Reachability Analysis for Hybrid Automata

3.4.1 Defining the Hybrid Automaton

A hybrid system is a system with dynamics that are dependent upon both continuous and discrete state variables. An example that is commonly used to introduce this concept is a temperature control system that consists of a heater controlled by a thermostat [7]. The temperature of the room is a continuous state of the system and the on/off status of the heater is a discrete state of the system. Each discrete mode of the system is associated with a particular set of equations which model the continuous-time evolution of the system. For example, we would expect the temperature to decrease when the heater is off and to increase when the heater is on. Each discrete mode of the system is also associated with transition guards which dictate when the discrete mode of the system changes. For example, the heater turns on when the temperature is below the low temperature threshold of the thermostat and turns off when it is above the high temperature threshold of the thermostat.

A common modeling formalism for hybrid systems is the hybrid automaton.

Definition 3.4.1. *(hybrid automaton) A hybrid automaton is a modeling formalism for hybrid systems which is defined in [9] by the octuple:* $H = (Q, X, f, Init, D, E, G, R)$*, where*

- Q *is a finite set of discrete variables;*
- X *is a finite set of continuous variables;*
- $f : Q \times X \to X$ *is a vector field defining the dynamics of the continuous variables;*
- $Init \subset Q \times X$ *is the set of initial states;*
- $D : Q \to P(X)$ *defines the domain of the discrete modes*
- $E \subset Q \times Q$ *is a set of edges;*
- $G : E \to P(X)$ *defines guard conditions for discrete transitions;*
- $R : E \times X \to P(X)$ *is a reset map defining discontinuities in the continuous state of the system during discrete transitions.*

The state of the system is the collection of continuous and discrete states and will be denoted by $(x, q) \in X \times Q$. The formal hybrid automaton which defines the thermostat system is

- $Q = \{q_{\text{on}}, q_{\text{off}}\}$;
- $X = \mathbb{R}$;
- $f(q_{\text{on}}, x) = c_1 x$,
  $f(q_{\text{off}}, x) = -c_2 x$;
- $Init = q_{\text{off}} \times \{x \in X : x \leq T_{\text{high}}\}$;
- $D(q_{\text{on}}) = \{x \in X : x \leq T_{\text{high}}\}$,
  $D(q_{\text{off}}) = \{x \in X : x \leq T_{\text{low}}\}$;
- $E = \{(q_{\text{on}}, q_{\text{off}}), (q_{\text{off}}, q_{\text{on}})\}$;

- $G(q_{\text{on}}, q_{\text{off}}) = \{x \in \mathbb{R} : x \geq T_{\text{high}}\}$,
  $G(q_{\text{off}}, q_{\text{on}}) = \{x \in \mathbb{R} : x \leq T_{\text{low}}\}$;
- $R(q_{\text{on}}, q_{\text{off}}, x) = R(q_{\text{off}}, q_{\text{on}}, x) = x$.

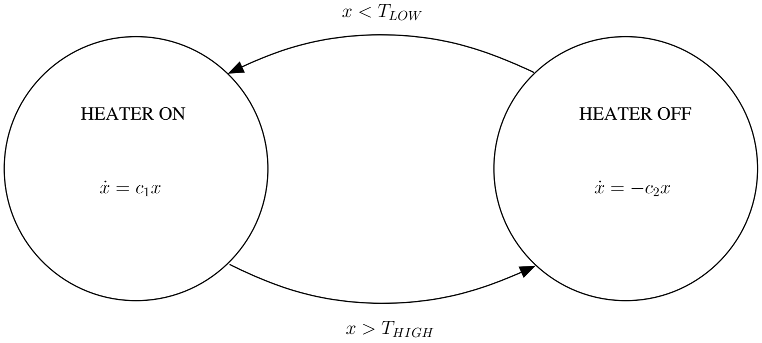


Figure 3.4.6: Illustration of simple heater/thermostat hybrid automaton.

Figure 3.4.6 shows the directed graph (Q, E) of the hybrid automaton. This hybrid automaton happens to belong to a special class of hybrid automata which have affine continuous dynamics within each discrete mode and transition guards which are defined by linear predicates. This special class of automata is hereafter referred to as affine hybrid automata (AHA).

Definition 3.4.2. *(affine hybrid automaton) An affine hybrid automata is a hybrid automaton for which the vector field f is of the form*

$$f(q, x) = A_q x \tag{3.4.27}$$

and where Init, D, *and* G *are expressed as boolean combinations of linear inequalities.*

Another special class of hybrid automata are linear hybrid automata (LHA). Note that the term LHA is used in different ways in the literature, it is sometimes used to refer to the concept which has been defined here as an affine hybrid automata.

Definition 3.4.3. *(linear hybrid automaton) A linear hybrid automaton is a hybrid automaton for which the vector field f is constant in each discrete mode and where* Init, D, *and G are expressed as boolean combinations of linear inequalities.*

3.4.2 Reachability Computation for Affine and Linear Hybrid Automata

Within each discrete mode of a AHA or LHA, the continuous states of the system evolve in exactly the same manner as the states of a linear ODE. Methods for calculating the reachable set of a linear ODE were discussed in Section 3.3 and those same procedures can be used to calculate the reachable set of hybrid formulations, with some extra logic to account for discrete transitions of the system.

Consider a AHA with discrete state space $Q = \{q_1, q_2, q_3\}$ and a continuous state space $X = \mathbb{R}^2$. The initial state of the system is given by $Init = q_1 \times P_0$ where $P_0 \subset D(q_1)$ is some arbitrary convex polytope. Let the successive reachable sets calculated over the discrete sampling interval be represented by P_i. At each simulation step i, the set P_i is intersected with the transition guards of the current discrete mode, q_i. If the intersection is null, the simulation continues in q_i. If the intersection is not null, the result of the intersection is used as an initial set for a new reachability computation in the post transition mode, hereafter denoted by q'. Figure 3.4.7a illustrates the intersection of the reachable set in mode q_1 with the transition guard $G(q_1, q_2)$. The progression of the new reachability computation in mode q_2 is illustrated in Figure 3.4.7b. New iterations of the reachability algorithm which are spawned in the post transition mode are referred to as flow pipes.

Unfortunately, the example illustrated in Figure 3.4.7 is an ideal scenario and does not account for difficulties that are manifested in the realization of this hybrid reachability scheme. For example, consider a calculation which originates in q_1 and which intersects the transition guard $G(q_1, q_3)$ at multiple simulation steps, as in Figure 3.4.8a. There are

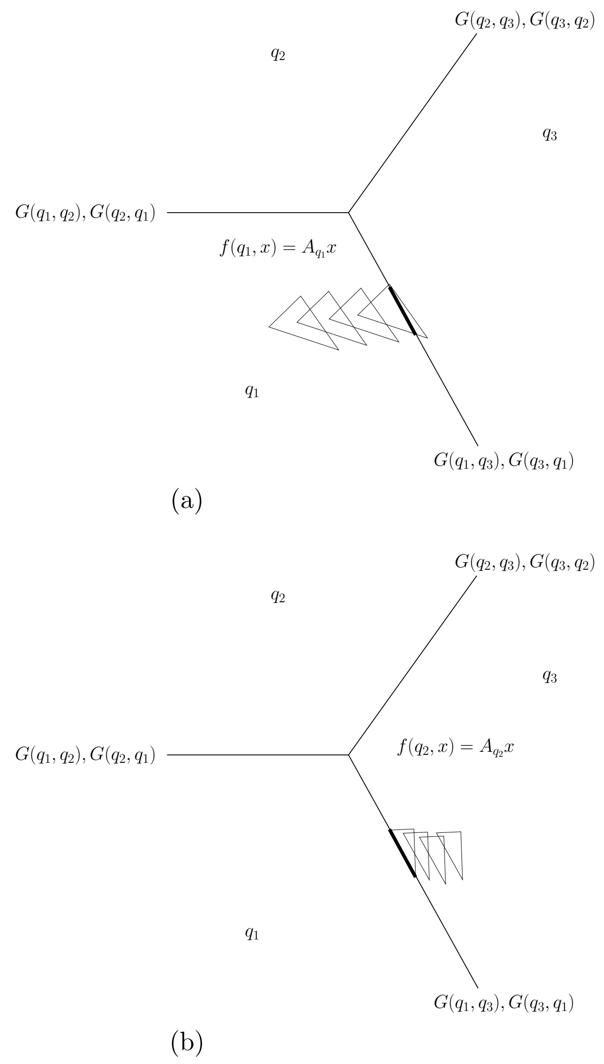

$G(q_2, q_3), G(q_3, q_2)$
q_2
q_3
$G(q_1, q_2), G(q_2, q_1)$
$f(q_1, x) = A_{q_1}x$
q_1
$G(q_1, q_3), G(q_3, q_1)$
(a)
$G(q_2, q_3), G(q_3, q_2)$
q_2
q_3
$G(q_1, q_2), G(q_2, q_1)$
$f(q_2, x) = A_{q_2}x$
q_1
$G(q_1, q_3), G(q_3, q_1)$
(b)

Figure 3.4.7: (a) The linear reachability calculation is applied with $f(q_1, x)$ until it intersects with the transition guard $G(q_1, q_3)$; (b) The intersection with the guard is used as an initial set for a new flow pipe reachability calculation with $f(q_3, x)$.

at least two ways to address this scenario, however both suffer from their own difficulties. With one method, a new flow pipe in q' could be created for each intersection at each simulation step. When this approach is implemented, the number of flow pipe computations could explode as the simulation continues to traverse discrete modes. The alternative is to continue computing the reachable sets within q_i until $P_i \cap D(q_i) = \emptyset$, at which point the set defined by $\mathcal{CH}\left\{\bigcup_i P_i \cap G(q_i, q')\right\}$ can be used as the initial set for a new flow pipe in q'. This approach could potentially lead to the propagation of large over-approximation errors. It is important to note that within each discrete mode, the over approximation at each discrete time step does not propagate to the next time step. At transition guard boundaries however, an over approximation that is intersected with the guard hyperplane does propagate into the new flow pipe computation that is spawned in q'. When some P_i intersects multiple transition guards, the combinatorial explosion problem becomes even more significant, as in Figure 3.4.9. In most implementations of hybrid reachability algorithms, a tight convex hull of successive intersections is used to address the issue while minimizing propagated over approximation error [6]. Dynamic hybridization has also been proposed, which relies on overlapping linearization domains rather than a fixed partitioning of the state space [4]. Although workable solutions have been realized, finding better, more efficient, solutions to this problem is still an open area for further research.

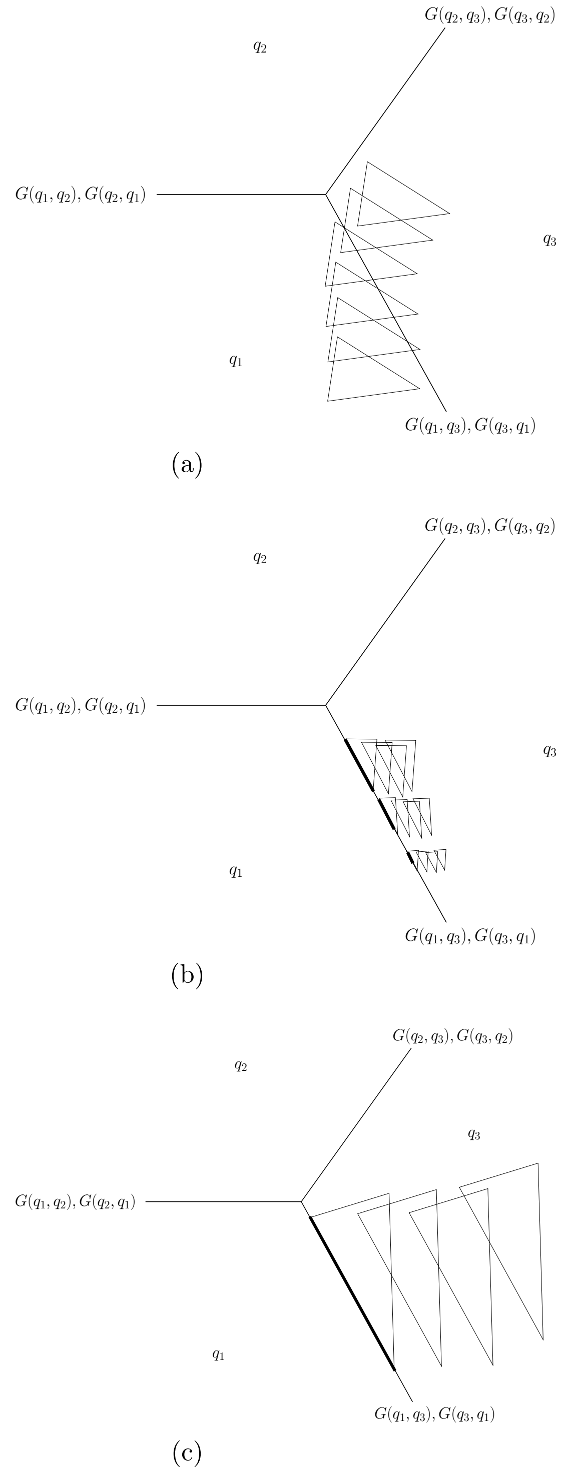

$G(q_2, q_3), G(q_3, q_2)$
q_2
$G(q_1, q_2), G(q_2, q_1)$
q_3
q_1
$G(q_1, q_3), G(q_3, q_1)$
(a)
$G(q_2, q_3), G(q_3, q_2)$
q_2
$G(q_1, q_2), G(q_2, q_1)$
q_3
q_1
$G(q_1, q_3), G(q_3, q_1)$
(b)
$G(q_2, q_3), G(q_3, q_2)$
q_2
q_3
$G(q_1, q_2), G(q_2, q_1)$
q_1
$G(q_1, q_3), G(q_3, q_1)$
(c)

Figure 3.4.8: (a) The linear reachability calculation is applied with $f(q_1, x)$ until intersection with the transition guard $G(q_1, q_3)$; (b) The intersection with the transition guard is used as an initial set for a new flow pipe reachability calculation with $f(q_3, x)$.

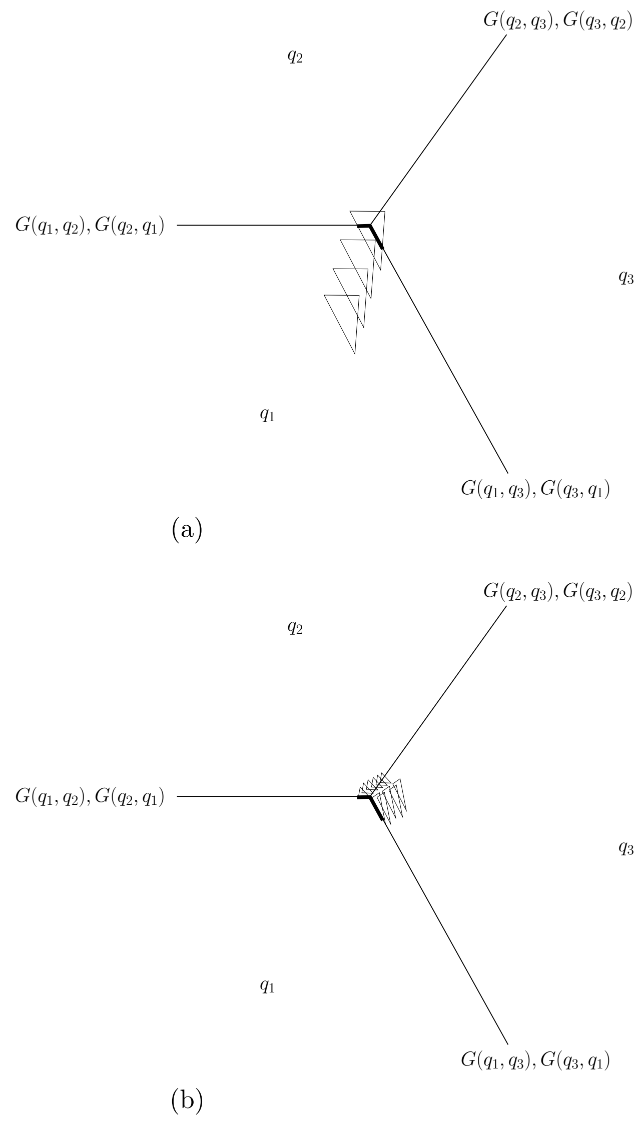

$G(q_2, q_3), G(q_3, q_2)$
q_2
$G(q_1, q_2), G(q_2, q_1)$
q_3
q_1
$G(q_1, q_3), G(q_3, q_1)$
(a)
$G(q_2, q_3), G(q_3, q_2)$
q_2
$G(q_1, q_2), G(q_2, q_1)$
q_3
q_1
$G(q_1, q_3), G(q_3, q_1)$
(b)

Figure 3.4.9: (a) The linear reachability calculation is applied with $f(q_1, x)$ until intersection with the transition guards $G(q_1, q_2)$ and $G(q_1, q_3)$; (b) After intersection with the transition guards, multiple flow pipes originate in both q_2 and q_3.

3.4.3 Modeling Non-linear Systems as Hybrid Automata

Non-linear ODE models are very useful, especially for modeling biological systems and adaptive controls. Therefore, it is useful to extend the reachability algorithms that have been developed for linear systems to non-linear systems. Unfortunately the simulation of a non-linear system cannot be represented as the successive application of an affine transformation, which was a key assumption in the development of the linear algorithms. Instead of developing reachability algorithms that apply directly to non-linear systems, the approach to approximating reachable sets for linear systems can be modified slightly for finding reachable sets of piecewise-linear approximations of the non-linear system. A piecewise linear model is actually a special case of the AHA and LHA models, for which reachability algorithms have already been developed. The term *hybridization* is used to describe the process of approximating a non-linear model as a hybrid automata. Let us consider the hybridization of the autonomous, one-dimensional non-linear system with dynamics given by

$$\dot{x} = f(x) = -x^3 \tag{3.4.28}$$

The graph of $f(x)$ on the domain $x \in [-10, 10]$ is given in Figure 3.4.10.

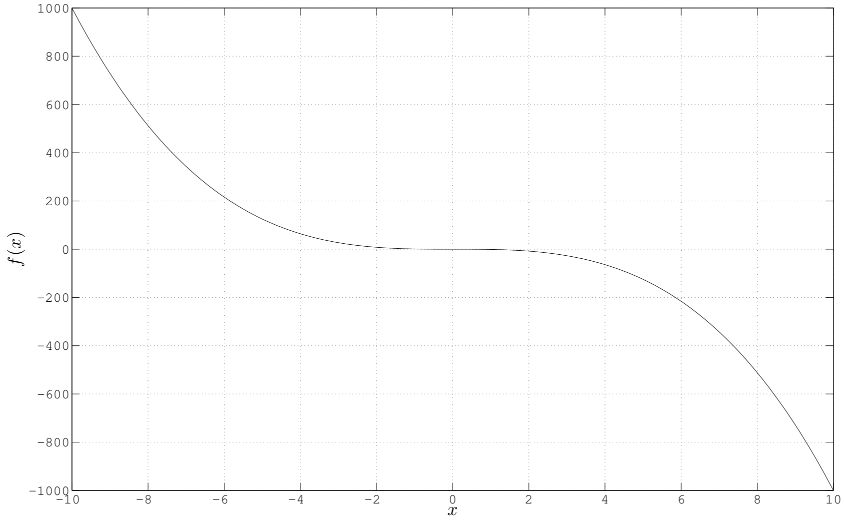


Figure 3.4.10: Graph of the function $f(x)$ over the domain $x \in [-10, 10]$

The first step in the hybridization process is partitioning the continuous state space $X = \mathbb{R}^n$ into discrete modes. For this hybridization, let five discrete modes are defined,

$\{q_1, q_2, q_3, q_4, q_5\}$, with continuous domains defined as

$$\begin{aligned}
D(q_1) &= \{x \in X : x \leq -6\}; && (3.4.29)\\
D(q_2) &= \{x \in X : -6 \leq x \leq -2\}; && (3.4.30)\\
D(q_3) &= \{x \in X : -2 \leq x \leq 2\}; && (3.4.31)\\
D(q_4) &= \{x \in X : 2 \leq x \leq 6\}; && (3.4.32)\\
D(q_5) &= \{x \in X : x \geq 6\}. && (3.4.33)
\end{aligned}$$

The collection of edges is $E = \{(q_1, q_2), (q_2, q_1), (q_2, q_3), (q_3, q_2), (q_3, q_4), (q_4, q_3), (q_4, q_5), (q_5, q_4)\}$ and the transition guards associated with each edge are

$$\begin{aligned}
G(q_1, q_2) &= \{x \in X : x \geq -6\}; && (3.4.34)\\
G(q_2, q_1) &= \{x \in X : x \leq -6\}; && (3.4.35)\\
G(q_2, q_3) &= \{x \in X : x \geq -2\}; && (3.4.36)\\
G(q_3, q_2) &= \{x \in X : x \leq -2\}; && (3.4.37)\\
G(q_3, q_4) &= \{x \in X : x \geq 2\}; && (3.4.38)\\
G(q_4, q_3) &= \{x \in X : x \leq 2\}; && (3.4.39)\\
G(q_4, q_5) &= \{x \in X : x \geq 6\}; && (3.4.40)\\
G(q_5, q_4) &= \{x \in X : x \leq 6\}. && (3.4.41)
\end{aligned}$$

The simplest way to design the continuous vector field within mode q_i is by linearizing $f(x)$ around the around the center point p_i of $D(q_i)$. Note that other linearization approaches such as least squares often give better results, the tangent approach is used here only to avoid unnecessary complexity. The linearized vector field for mode q_i is given by

$$F(q_i, x) = \left(\left. \frac{df}{dx} \right|_{x=p_i} (x - p_i) \right) + f(p_i) \qquad (3.4.42)$$

The calculated values of the vector field for each discrete mode are

$$F(q_1, x) = -192x - 1024 \tag{3.4.43}$$

$$F(q_2, x) = -48x - 128 \tag{3.4.44}$$

$$F(q_3, x) = 0 \tag{3.4.45}$$

$$F(q_4, x) = -48x + 128 \tag{3.4.46}$$

$$F(q_5, x) = -192x + 1024 \tag{3.4.47}$$

The reset relation is simply $R(Q, X) = x$. Figure 3.4.11 illustrates the hybridization of the system. The directed graph (Q, E) is shown in Figure 3.4.12.

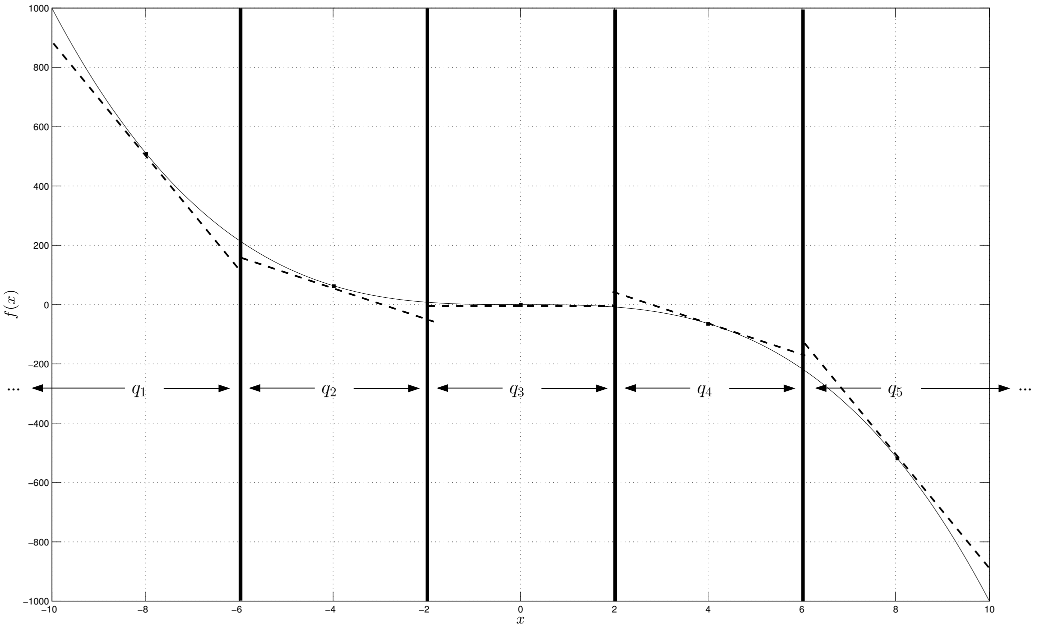


Figure 3.4.11: The discrete modes of the hybrid automaton model and their linear vector fields.

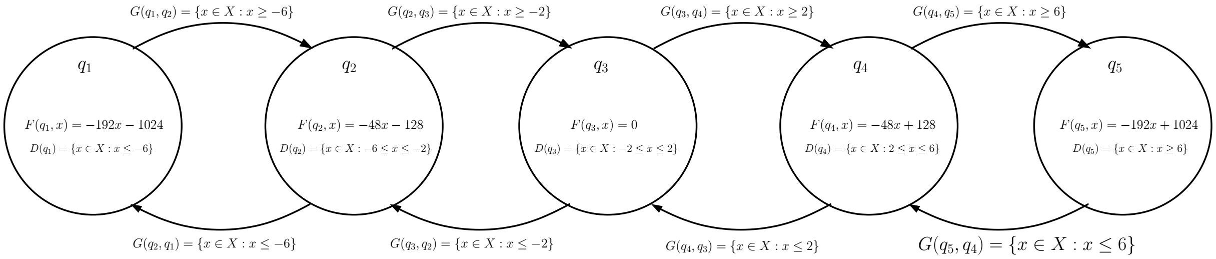

$G(q_1, q_2) = \{x \in X : x \geq -6\}$
$G(q_2, q_3) = \{x \in X : x \geq -2\}$
$G(q_3, q_4) = \{x \in X : x \geq 2\}$
$G(q_4, q_5) = \{x \in X : x \geq 6\}$
q_1
$F(q_1, x) = -192x - 1024$
$D(q_1) = \{x \in X : x \leq -6\}$
q_2
$F(q_2, x) = -48x - 128$
$D(q_2) = \{x \in X : -6 \leq x \leq -2\}$
q_3
$F(q_3, x) = 0$
$D(q_3) = \{x \in X : -2 \leq x \leq 2\}$
q_4
$F(q_4, x) = -48x + 128$
$D(q_4) = \{x \in X : 2 \leq x \leq 6\}$
q_5
$F(q_5, x) = -192x + 1024$
$D(q_5) = \{x \in X : x \geq 6\}$
$G(q_2, q_1) = \{x \in X : x \leq -6\}$
$G(q_3, q_2) = \{x \in X : x \leq -2\}$
$G(q_4, q_3) = \{x \in X : x \leq 2\}$
$G(q_5, q_4) = \{x \in X : x \leq 6\}$

Figure 3.4.12: The directed graph (Q, E) of the hybridization of $f(x)$.

This hybridization may be useful in deriving general properties of the original non-linear system but the result of reachability calculations will not be an over-approximation of the actual reachable set. Therefore, the results of such a reachability calculation are useless for generating a proof about the behaviours of the original non-linear system. In order to ensure that the generated reachable set is an over-approximation of the actual reachable set, the model must be modified to account for the linearization error. This is accomplished by adding an error polytope V_{q_i} to the linearized dynamic equations in each node. The size of the error polytope is determined by solving an optimization problem for the minimum and maximum linearization error within each mode. That is, for every $x \in D(q_i)$, the dynamics of the system are bounded by $f(x) \in f'(q_i, x) \oplus V_{q_i}$. The linearization error is treated in exactly the same way as an input for a non-autonomous system.

3.5 Reachability Analysis Using Level Set Methods

A level set representation of a target set may be propagated over a finite grid of points to approximate the forwards or BRS of a system. When the system does not have any unknown inputs or unknown parameters, the propagation of the set is defined by the solution of a simple partial differential equation at each discrete time step, for each grid point. For example, consider the simple one dimensional system

$$\dot{x} = -x + 5 \tag{3.5.48}$$

The BRS of the system will be computed for an avoid set given by

$$x^2 \leq 4. \tag{3.5.49}$$

This avoid set can be encoded as the zero sub-level set of the function

$$y = x^2 - 4. \tag{3.5.50}$$

A graph of the level set function which defines the avoid set is provided in Figure 3.5.13.

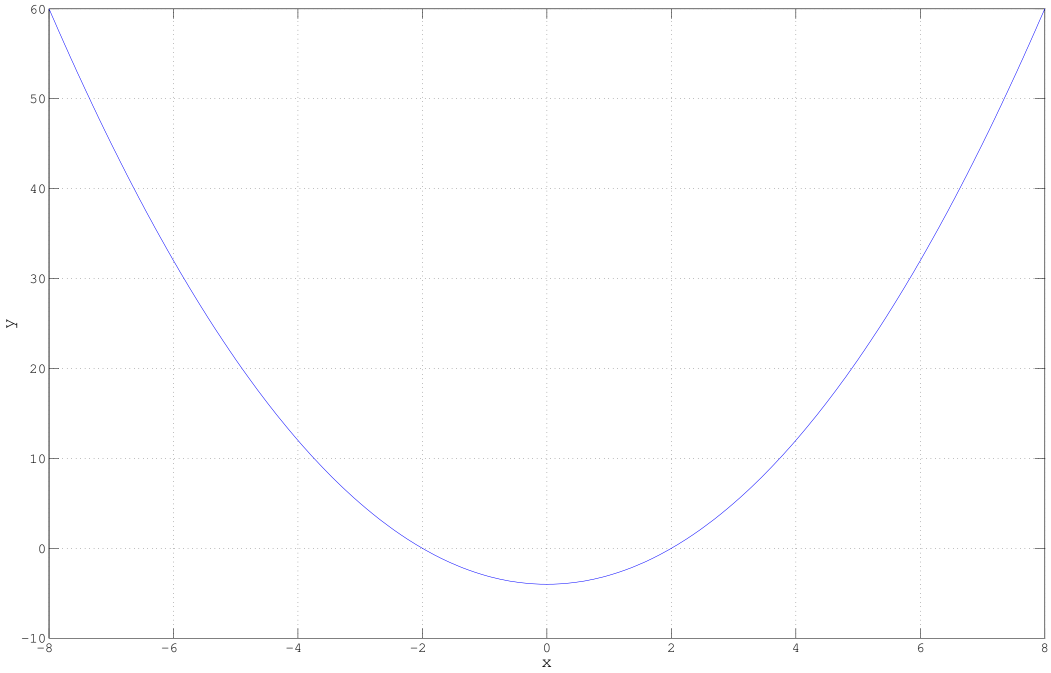


Figure 3.5.13: Graph of $y = x^2 - 4$.

It is trivial to derive the time derivative of the level set function relative to the time derivative of x, the result is

$$\dot{y} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial t}. \tag{3.5.51}$$

Initially the time derivative of the level set function can be computed analytically but generally after the first propagation step, the value of $\frac{\partial y}{\partial x}$ must be determined numerically via up-winding finite difference schemes or other approaches. Irrespective of whether the value of $\dot{y}$ is determined analytically or numerically, the evolution of the BRS at each grid point i and each discrete time index j is defined by

$$y_i^{j-1} = y_i(t_j - \tau) = \dot{y}_i(t_j)\tau + y_i(t_j). \tag{3.5.52}$$

The value of τ represents the discrete sampling period of the propagation and is assumed to be small. A graphical depiction of the example set after a single iteration of the evolution process is given in Figure 3.5.14. The parameters that were used for this calculation were $\tau = 0.2$ and a grid spacing of 0.5.

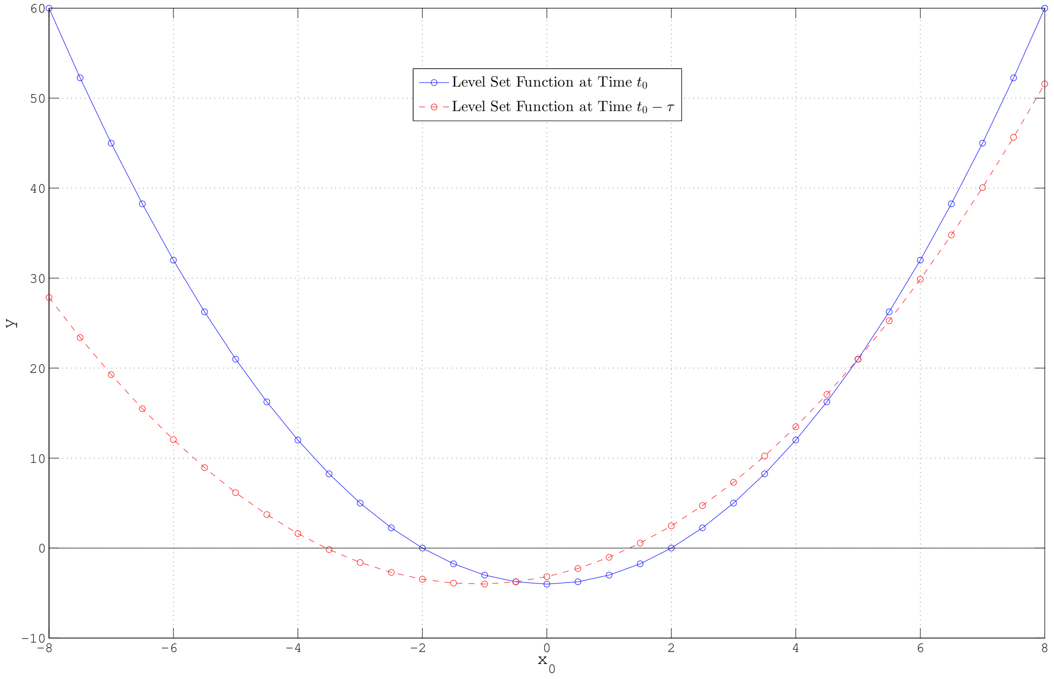


Figure 3.5.14: Graph of the level set at time t_0 and time $t_0 - \tau$.

Notice that the zero-sub-level set of $y(t_0)$ includes the grid points defined at $x_0 = 1.5$ and $x_0 = 2.0$ but the zero-sub-level set of $y(t_0 - \tau)$ does not. In other words, the current formulation of the calculation only includes states from which the avoid set is reachable within exactly time τ and does not include states from which the avoid set is reachable in time less than τ. To insure that states which pass through the avoid set are included in the BRS, the time derivative of the avoid set can be modified as in [13] by the following minimization operation

$$\dot{y} = \min\left(0, \frac{\partial y}{\partial t}\right). \tag{3.5.53}$$

This modification ensures that the value of y only decreases with time and, therefore, prevents points from leaving the BRS after having entered it. Figure 3.5.15 depicts the modified level set after a single iteration of the modified propagation algorithm. The BRS for a time horizon of 0.2 seconds is approximated by the zero-sub-level set of this numerically calculated level set propagation. The BRS for this example can be calculated for any time horizon by iteratively applying the propagation scheme defined in Equation (3.5.52). A FRS can also be calculated by simply changing the sign of the derivative term.

The preceding example considered a simple autonomous system. When a reachable set must be calculated for a system with uncertain inputs or uncertain parameters, the problem becomes more difficult. Fortunately, the Hamilton Jacobi Isaacs (HJI) equation provides an elegant solution to handling uncertainty in reachability calculations [14]. This framework transforms the problems of uncertainty into an optimization problem. Consider a scenario where a backwards reachability calculation must be performed to determine whether a system will enter an avoid set defined as a signed distance function. Minimizing the time derivative of the avoid set with respect to unknowns is equivalent to considering the worst case scenario, while maximizing the time derivative of the avoid set with respect to unknowns is equivalent to considering the best case scenario. Combining maximization and minimization operations over multiple inputs allows us to consider adversarial scenarios. The HJI equation provides a method of optimizing the time derivative at each discrete

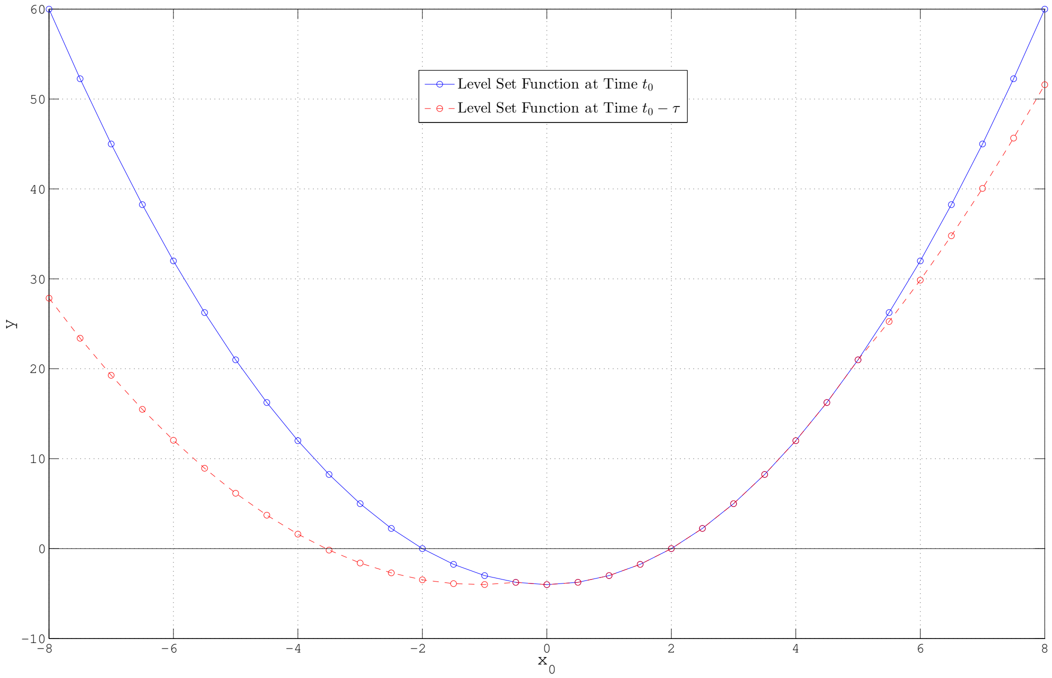


Figure 3.5.15: Graph of the level set at time t_0 and time $t_0 - \tau$, with restrictions for set shrinking.

time step by using dynamic programming. The form of the HJI equation that is used for reachability analysis is

$$\dot{y} = \min\left(0, H(x, \frac{\partial y}{\partial x})\right) \tag{3.5.54}$$

where the Hamiltonian is defined by

$$H(x,p) = \max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} p^T f(x,a,b), \tag{3.5.55}$$

such that a is an unknown parameter which is optimized so that the system avoids the avoid set and b is an unknown parameter which is optimized so that the system is driven to the avoid set.

The Level Set Toolbox [13] is an open source, MATLAB based set of tools that can be used to calculate reachable sets using this level set approach, among other applications. The utility of the toolbox is primarily its implementation of robust numeric algorithms for approximating derivatives and calculating Hamiltonian functions. Note that the toolbox cannot guarantee that generated reachable sets are an over-approximation of actual reachable sets. More detailed information about level set reachability calculations and the Level Set Toolbox can be found in [14][12][11][13].

IV. CASE STUDY SCENARIO

The scenario that was chosen for measuring the capability of the hybrid automata reachability tools was a collision avoidance scenario for differential drive robots. The original intent was to consider the verification problem for two or more independently controlled robots, each with incomplete knowledge of the other's states. Direct verification of the system's dynamics was known to be beyond the capability of these tools so the complexity of the reachability calculations was reduced by computing only the subset of reachable sets necessary to implement run-time assurance. Initial research revealed that this original intent might not be achievable with either PHAVer or the LGG support function algorithm, and the complexity of the simulation was reduced further to consider only uncertainty in the system's initial states. A reachability proof was to be generated to ensure that the robots could never come within a pre-defined distance of one another.

4.1 Run Time Assurance

Many practical systems are far too complex for modern reachability tools. For such systems, the entire range of possible operating conditions cannot be verified in reasonable time with even the fastest computers available today. The underlying concept of run-time assurance is to compute reachable sets at run time with respect to the system's present state. The advantage of this approach is that it avoids the necessity of having to consider all possible configurations of the system, which becomes an impossible problem for systems with more than a few continuous states. The particular run time assurance approach that was implemented here is the Simplex Architecture.

The Simplex Architecture is a run-time assurance implementation which uses simple, verified safety controllers to bound the functionality of a more complex experimental controller which is beyond the reach of modern verification tools [17]. Consider the block diagram of the Simplex Architecture in Figure 4.1.1.

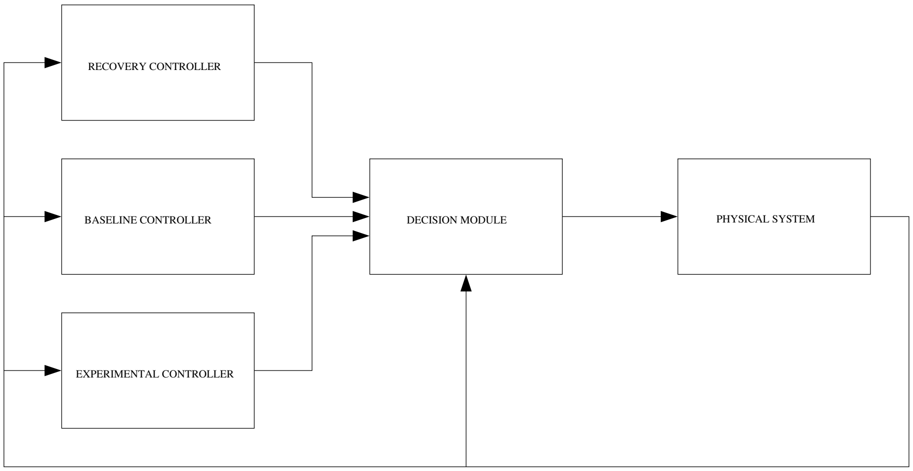


Figure 4.1.1: The Simplex Architecture

The basic architecture defines three controllers: a baseline controller, a safety controller, and an experimental controller. The central component of the Simplex Architecture is the decision module. The decision module determines which of the three controllers to use in order to maintain safety while maximizing performance. The reachable sets of the safety controllers and the baseline controller are used within the decision module to make safety decisions. When an unsafe state is within the reachable set of all of the safety controllers, the system is bound to fail regardless of the switch state of the decision module. Therefore, the reachable states of the recovery controllers must be bloated so that the decision module switches before failure is inevitable. An appropriate bloating margin can be determined by calculating the maximum rate of change of the system's states. When it is impossible or impractical to compute the maximum rate of change of the states, a conservative estimate can be used. It is notable, though, that a conservative estimate sacrifices the size of the performance envelope.

The experimental controller is a complex controller that is designed for performance.

For example, the experimental controller in the collision avoidance scenario could be a trajectory controller with path planning logic. This controller is too complex to be verified and, therefore, not suitable for safety-critical applications. The purpose of the Simplex Architecture is to use less complex controllers to provide a recovery mechanism for the experimental controller. As long as the decision module determines that the safety controllers can maintain safe operation of the system, the experimental controller is permitted to control the system.

The baseline controller is a simple controller that can be verified and which is capable of basic control that is necessary to maintain stable, safe operation of the system. In the collision avoidance scenario presented in this thesis, the baseline controller is omitted from the architecture because it is not necessary. That is, after the safety controller is initiated and the collision is avoided, the decision module switches back to the experimental controller. If the experimental controller is consistently malfunctioning and driving the system towards the unsafe states, the decision module would repeatedly switch between the two controllers in order to maintain safe operation. In some situations, this would obviously be undesirable but in the collision avoidance scenario it could make sense. As a supplementary example, the baseline controller for an aircraft might be a controller designed with just ”return to base” capabilities.

The purpose of the safety controller is to allow the system to move to a safe state when it is close to violating a safety condition. The idea is that the safety controller extends the safety envelope of the baseline controller so as to permit a larger envelope of bounded safe operation for the experimental controller. There may actually be several safety controllers for a particular application, each suited for recovering the system from a different scenario. The safety controllers must be simple enough to be verified. In the collision avoidance scenario with differential drive robots, the safety controllers are designed at the limits of the robots’ control authority in order to exert maximum recovery effort.

4.2 Modeling Differential Drive Robots

This section explains how the differential drive robot is modeled with state space equations. The derivation of the kinematic state-space model of a differential drive robot is discussed first and then extended for the development of a two-robot model defined by relative states.

A differential drive robot has two parallel, independently-controlled drive wheels. Let L represent the perpendicular distance between the drive wheels, let θ represent the heading of the robot with respect to a fixed reference frame, the coordinate pairs (x_L, y_L) and (x_R, y_R) represent the two-dimensional symmetrical center points of the drive wheels, (x_c, y_c) represent the coordinates of the point halfway between the robot's drive wheels, and v_R and v_L represent the magnitudes of the right and left wheel velocities relative to a fixed reference frame, respectively, as shown in Figure 4.2.2.

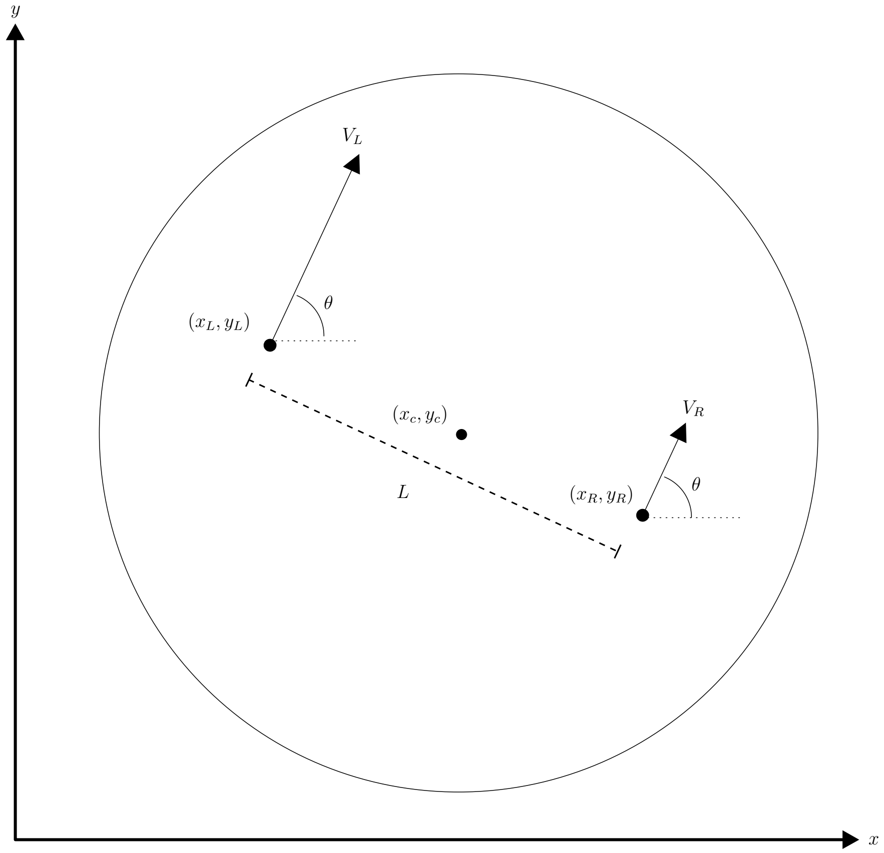


Figure 4.2.2: Parameters of a differential drive robot in a fixed reference frame.

Using the principles of rigid body physics, it is straightforward to derive general kinematic equations for the motion of a differential drive robot in terms of the defined parameters. The motion of a single point on a rigid body in two dimensions can be completely characterized by two parameters: linear velocity and angular velocity. For simplicity, the motion of the point (x_c, y_c) will be considered. Note that the following derivation is valid only under the assumption that the robot's terrain is perfectly flat and that the robot's drive wheels do not slip. The first step in the derivation is to consider the motion of the robot with respect to a moving reference frame that is defined relative to the center of either drive wheel. In this case, let the point (x_L, y_L) be at the origin of the moving reference frame and let the axes of the moving reference frame be aligned with the axes of the fixed reference frame. Parameters defined with respect to the moving reference frame will be accented with a tilde. The transformed parameters are

$$\begin{aligned} \tilde{x}_L &= x_L - x_L = 0; \\ \tilde{y}_L &= y_L - y_L = 0; \\ \tilde{v}_L &= v_L - v_L = 0; \\ \tilde{x}_R &= x_R - x_L; \\ \tilde{y}_R &= y_R - y_L; \\ \tilde{v}_R &= v_R - v_L. \end{aligned} \tag{4.2.1}$$

Note that the velocities of the left and right drive wheels can be added and subtracted as scalar quantities only because their vectors are always parallel.

The formula for the angular velocity of a point is given by

$$\omega = \frac{v}{r}, \tag{4.2.2}$$

where ω is the angular velocity, r is the radius of rotation, and v is the linear velocity. When calculating the angular velocity, it is convenient to take advantage of the principle that all points on the same rigid body share the same angular velocity. In this case, it is less work to derive the angular velocity at the point (x_R, y_R) than it would be to directly derive the angular velocity of the point of interest, (x_c, y_c). Since the origin of the moving reference frame is defined as the center of rotation, the radius of rotation at the point (x_R, y_R) is equivalent to the distance between the point and the origin; i.e. $r = L$. The angular velocity is, therefore

$$\omega = \tilde{\omega} = \frac{\tilde{v}_R}{L} = \frac{v_R - v_L}{L}. \tag{4.2.3}$$

Finding the linear velocity of the point (x_c, y_c) is now simply a matter of solving equation (4.2.2) for v. In this case, r is the radius of rotation of the point (x_c, y_c); i.e. $r = \frac{L}{2}$. The linear velocity of the robot's center point is therefore

$$\begin{aligned} \tilde{v}_c &= \frac{\omega L}{2} = \frac{v_R - v_L}{2}; & (4.2.4) \\ v_c &= \tilde{v}_c + v_L = \frac{v_R + v_L}{2}. & (4.2.5) \end{aligned}$$

These closed-form expressions for the linear and angular velocity completely characterize the motion of the point (x_c, y_c) but ultimately a state space formulation of the differential drive robot is desired. Let v_L and v_R be the inputs to the system. The dynamics of the drive motors will be ignored; that is, the existence of an inner loop velocity controller is assumed for each drive motor and the tracking errors of the controllers is assumed to be sufficiently small so that their effects on the evolution of system states can be ignored. A practical set of generalized coordinates for a differential drive robot is the triple (x_c, y_c, θ). The time derivative of the heading of the robot is simply the angular velocity, which has already been derived. The time-derivative of the x-coordinate of the robot is equivalent to the component of the robot's velocity that is parallel to the x-axis and the time-derivative of the y-coordinate of the robot is equivalent to the component of the robot's velocity that is parallel to the y-axis. The continuous-time kinematic state space equations for a differential drive robot are therefore

$$
\begin{aligned}
\dot{x_c} &= v_c \cos\theta; \\
\dot{y_c} &= v_c \sin\theta; \qquad (4.2.6) \\
\dot{\theta} &= \omega.
\end{aligned}
$$

It is evident from the state equations that the differential drive robot is a non-linear system. A closer look at the state equations also reveals that the states are nonholonomically constrained. A nonholonomic constraint is a constraint on the time-derivatives of the system states that cannot be integrated and expressed as an explicit function of the system states. The significance of the nonholonomic constraint is that it prevents the system from traversing arbitrary trajectories in the state space. Both the non-linearity and the nonholonomic properties of the system are important since they make controller design difficult and also make the problem of verifying properties of controlled states particularly interesting.

For the purpose of ensuring that two differential drive robots do not collide, a state space model must be developed which incorporates the states of both robots. One way to create this model is to simply concatenate the independent states of all robots into a single conglomerate system. This approach, however, preserves more information than is really necessary. That is, the relative states of the robots are sufficient for the purposes of collision avoidance, the actual locations and orientations of each robot are irrelevant.

First, consider a model of the two robot system in which the the x and y coordinates of the robots are defined with respect to a reference frame that moves with the center of one robot. The axes of the moving reference frame remain parallel with the axes of the fixed reference frame. When the states of the robot are translated to this reference frame, they become

$$\begin{aligned}
x_{11} &= x_1 - x_1 = 0; \\
y_{11} &= y_1 - y_1 = 0; \\
\theta_1 &= \theta_1; \\
x_{21} &= x_2 - x_1; \\
y_{21} &= y_2 - y_1; \\
\theta_2 &= \theta_2.
\end{aligned} \tag{4.2.7}$$

The coordinate pairs (x_1, y_1) and (x_2, y_2) in equation (4.2.7) represent the locations of the centers of the two robots with respect to the fixed reference frame. Likewise, the values of θ_1 and θ_2 represent the headings of the two robots. Notice that the states of the two robot system can be condensed to the quadruple $(x_{21}, y_{21}, \theta_1, \theta_2)$ when defined with respect to the moving reference frame. This transformation preserves the information which is necessary to verify collision avoidance; that is, the distances between the robots. To define the kinematic state space model with this newly defined set of states, it is necessary to derive the time-derivatives of the states with respect to the moving reference frame. The derivatives of these translated states are

$$\dot{x}_{21} = \dot{x}_2 - \dot{x}_1 = v_{c2}\cos\theta_2 - v_{c1}\cos\theta_1; \tag{4.2.8}$$

$$\dot{y}_{21} = \dot{y}_2 - \dot{y}_1 = v_{c2}\sin\theta_2 - v_{c1}\sin\theta_1; \tag{4.2.9}$$

$$\dot{\theta}_1 = \omega_1; \tag{4.2.10}$$

$$\dot{\theta}_2 = \omega_2. \tag{4.2.11}$$

It turns out that this condensed model can be condensed further into just three states by allowing the moving reference frame to rotate so that its horizontal axis is always aligned with the heading of the robot at the origin. Effectively, the previously defined relative states must be rotated by an angle $-\theta_1$. The rotated states are modified as

$$
\begin{aligned}
x'_{11} &= x_{11} = 0; \\
y'_{11} &= y_{11} = 0; \\
\theta'_{11} &= \theta_1 - \theta_1 = 0; \\
x'_{21} &= x_{21} \cos(-\theta_1) - y_{21} \sin(-\theta_1); \\
y'_{21} &= x_{21} \sin(-\theta_1) + y_{21} \cos(-\theta_1); \\
\theta'_{21} &= \theta_2 - \theta_1.
\end{aligned}
\tag{4.2.12}
$$

Refer to Figure 4.2.3 for a graphical depiction of the states of the two-robot system relative to the moving reference frame.

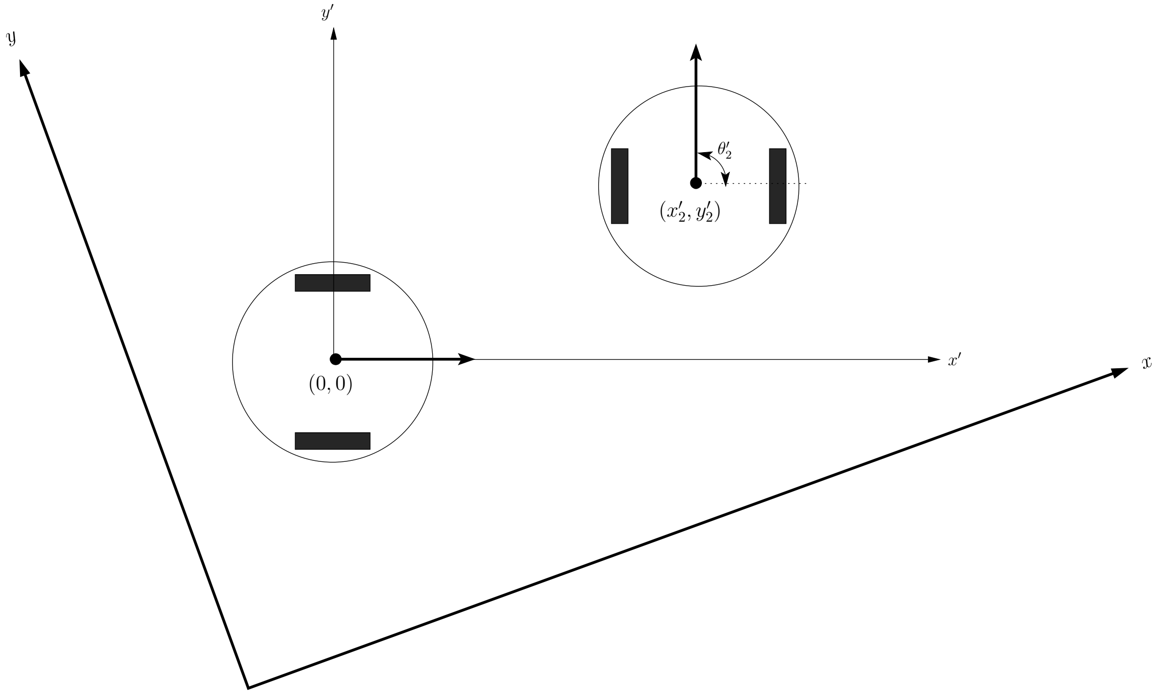


Figure 4.2.3: States of a two-robot system with respect to the moving reference frame.

The triple $(x'_{21}, y'_{21}, \theta'_{21})$ is therefore sufficient to completely characterize the two robot system for the collision avoidance scenario. The time-derivatives of these states are

$$
\begin{aligned}
\dot{x}'_{21} &= -v_{c1} + v_{c2}\cos\theta'_{21} + \omega_1 y'_{21}; \\
\dot{y}'_{21} &= v_{c2}\sin\theta'_{21} - \omega_1 x'_{21}; \\
\dot{\theta}'_{21} &= \omega_2 - \omega_1;
\end{aligned}
\tag{4.2.13}
$$

4.3 Defining Safety Controllers

Since the verification scenario that is being considered is utilizing the Simplex Architecture, reachable sets will be generated specifically for the safety controllers rather than for the experimental controller of for the entire range of possible controls that could be applied to the system. Before discussing the design of the safety controllers, several constraints are added to the scenario in order to make calculation of the reachable sets practical and the simulated scenario more interesting. In this case, constraints are implemented by defining limits for the control inputs of the system. First, let $v_{c1} = v_{c2} = 50$, so that the velocity of the robots is a constant parameter of the system rather than a control input. Additionally, the angular velocity inputs for both robots are constrained such that $\omega_1, \omega_2 \in [-1, 1]$. The modified scenario essentially assumes that the two robots are controllable only in their steering and that the rate of steering is constrained so that the robots are not capable of a zero turn radius.

Ideally, the recovery controller(s) are optimized for the purpose of preventing the system from entering the avoid set. That is, they should only be activated when the experimental controller will imminently cause failure unless it also implements the optimal control strategy for avoiding the unsafe condition. In practice though, the optimal safety controller is often either too complex for modern reachability algorithms or it is too difficult to find an optimal control strategy. For this reason, use of the Simplex architecture almost always involves some sacrifice in the performance of the experimental controller in favor of the system's safety.

Rather than attempting to find the optimal recovery controllers for this scenario, they are defined here based upon an intuitive perception of the scenario. That is, two recovery controllers are defined such that both robots make the sharpest turn within the constraints to avoid one another. The control inputs associated with each of the recovery controllers are defined as follows:

$$\begin{aligned} \mathcal{C}_1 &: \{\omega_1 = -1, \omega_2 = -1\} && (4.3.14) \\ \mathcal{C}_2 &: \{\omega_1 = 1, \omega_2 = 1\} && (4.3.15) \end{aligned}$$

Since each of the controllers defines control inputs that are constant, the reachability calculations for these controllers are as simple as possible. In fact, the model of the system can be greatly simplified for a pair of robots with the same linear and angular velocities. Consider a different formulation of the system with a new state θ_{21} representing the heading of the differential location of the robot pair in the differential coordinate system. The new state replaces both θ_1 and θ_2 in the state space model of equation (4.2.12). The modified formulation of the system is given below and the derivation is provided in Appendix VIII.

$$\begin{aligned} \dot{x}_{21} &= v_{21} \cos \theta_{21}; \\ \dot{y}_{21} &= v_{21} \sin \theta_{21}; \\ \dot{\theta}_{21} &= \omega, \end{aligned} \qquad (4.3.16)$$

where the value of the differential velocity v_{21} is given by

$$v_{21} = -2v \sin \left(\frac{\theta_2 - \theta_1}{2} \right). \qquad (4.3.17)$$

Note that the value of v_{21} is constant since v is defined as a constant and $\theta_2 - \theta_1$ must be constant since $\omega = \omega_1 = \omega_2$ is defined as a constant. This representation of the system's dynamics is used to develop the maximum necessary time evolution of the reachable set for provable safety. It is also used to determine the minimum bloating margin that is required to ensure that the actual avoid set is not violated between sampling instants of the decision module.

V. CALCULATING REACHABLE SETS

5.1 Approximating the System as a Hybrid Automata

In order to compute the reachable states of the differential drive robots with either PHAVer or the LGG algorithm, the state space model of the two robot system must be approximated as a hybrid automata. Although these tools are being applied to the same system, it turns out that different hybrid automata formulations are better suited for each tool. The design of a LHA approximation for implementation with PHAVer will be discussed first.

Although PHAVer can accept a LHA model with affine continuous dynamics in each discrete mode, it cannot handle an affine model with the unknown bounded inputs that are required to compensate for the linearization error. Instead, the dynamics of the system must be approximated by piecewise constant bounds on the derivatives. It turns out that different LHA formulations of the system can be created from the three state and four state non-linear models that have been derived. Even though the two models represent the same system, there are significant differences in efficiency, accuracy, and probability of termination between the two when each are used for reachability calculations with PHAVer. Both models will be hybridized here and the performance of PHAVer with each model will be compared. The model with three continuous states in equation (4.2.13) combines the headings of the two robots as a single differential state. In order to develop piecewise constant bounds on the derivatives within each discrete state, the discretization of this model must be defined in three dimensions, since the derivatives of the states are dependent on all three states. On the other hand, the model in equation (4.2.12) has four continuous states but the derivatives depend only on the headings of the robots. This model can therefore be discretized in just two dimensions.

The hybridization of the four state model is developed with a piecewise continuous approximation of the non-linear model. To make this hybridization example as simple as possible, the non-linear model is discretized with just two discrete states in each dimension. To do this, the continuous domain is divided into four equally sized rectangular regions over the states θ_1 and θ_2. The bounds on the derivatives of the states within each discrete mode are determined by finding the minimum and maximum values of the respective derivatives of the original non-linear system model within the domain of the particular discrete mode. A reset relation is defined in both dimensions to account for the discontinuity between 0 and 2π. All of the parameters of the hybrid system are defined below and a directed graph illustrating the structure of this LHA is given in Figure 5.1.1.

- $Q = \{q_1, q_2, q_3, q_4\}$;
- $X = \{\mathbb{R}^n \times \mathbb{R}^n \times [0, 2\pi) \times [0, 2\pi)\}$;
- $D(q_1) = \{[x, y, \theta_1, \theta_2] \in X : 0 \leq \theta_1 < \pi, 0 \leq \theta_2 < \pi\}$,
  $D(q_2) = \{[x, y, \theta_1, \theta_2] \in X : \pi \leq \theta_1 < 2\pi, 0 \leq \theta_2 < \pi\}$,
  $D(q_3) = \{[x, y, \theta_1, \theta_2] \in X : 0 \leq \theta_1 < \pi, \pi \leq \theta_2 < 2\pi\}$,
  $D(q_4) = \{[x, y, \theta_1, \theta_2] \in X : \pi \leq \theta_1 < 2\pi, \pi \leq \theta_2 < 2\pi\}$;
- $f(q_i) \in \left(\begin{array}{c} \left[\min\limits_{[x,y,\theta_1,\theta_2] \in D(q_i)} (v_{c2} \cos\theta_2 - v_{c1} \cos\theta_1), \max\limits_{[x,y,\theta_1,\theta_2] \in D(q_i)} (v_{c2} \cos\theta_2 - v_{c1} \cos\theta_1) \right] \\ \left[\min\limits_{[x,y,\theta_1,\theta_2] \in D(q_i)} (v_{c2} \sin\theta_2 - v_{c1} \sin\theta_1), \max\limits_{[x,y,\theta_1,\theta_2] \in D(q_i)} (v_{c2} \sin\theta_2 - v_{c1} \sin\theta_1) \right] \\ \omega_1 \\ \omega_2 \end{array} \right)$;
- $E = \{(q_1, q_2), (q_2, q_1), (q_1, q_3), (q_3, q_1), (q_2, q_4), (q_4, q_2), (q_3, q_4), (q_4, q_3)\}$;
- $G_1(q_1, q_2) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \geq \pi\}$, $G_2(q_1, q_2) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \leq 0\}$,
  $G_1(q_2, q_1) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \leq \pi\}$, $G_2(q_2, q_1) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \geq 2\pi\}$,
  $G_1(q_1, q_3) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \geq \pi\}$, $G_2(q_1, q_3) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \leq 0\}$,
  $G_1(q_3, q_1) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \leq \pi\}$, $G_2(q_3, q_1) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \geq 2\pi\}$,
  $G_1(q_2, q_4) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \geq \pi\}$, $G_2(q_2, q_4) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \leq 0\}$,

$G_1(q_4, q_2) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \leq \pi\}$, $G_2(q_4, q_2) = \{[x, y, \theta_1, \theta_2] \in X : \theta_2 \geq 2\pi\}$,

$G_1(q_3, q_4) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \geq \pi\}$, $G_2(q_3, q_4) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \leq 0\}$,

$G_1(q_4, q_3) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \leq \pi\}$, $G_2(q_4, q_3) = \{[x, y, \theta_1, \theta_2] \in X : \theta_1 \geq 2\pi\}$;

- $R_1(q_1, q_2, [x, y, \theta_1, \theta_2]) = R_1(q_2, q_1, [x, y, \theta_1, \theta_2]) = R_1(q_1, q_3, [x, y, \theta_1, \theta_2]) = R_1(q_3, q_1, [x, y, \theta_1, \theta_2]) = R_1(q_2, q_4, [x, y, \theta_1, \theta_2]) = R_1(q_4, q_2, [x, y, \theta_1, \theta_2]) = R_1(q_3, q_4, [x, y, \theta_1, \theta_2]) = R_1(q_4, q_3, [x, y, \theta_1, \theta_2]) = [x, y, \theta_1, \theta_2]$,

  $R_2(q_1, q_2, [x, y, \theta_1, \theta_2]) = R_2(q_3, q_4, [x, y, \theta_1, \theta_2]) = [x, y, \theta_1 + 2\pi, \theta_2]$,

  $R_2(q_1, q_3, [x, y, \theta_1, \theta_2]) = R_2(q_2, q_4, [x, y, \theta_1, \theta_2]) = [x, y, \theta_1, \theta_2 + 2\pi]$,

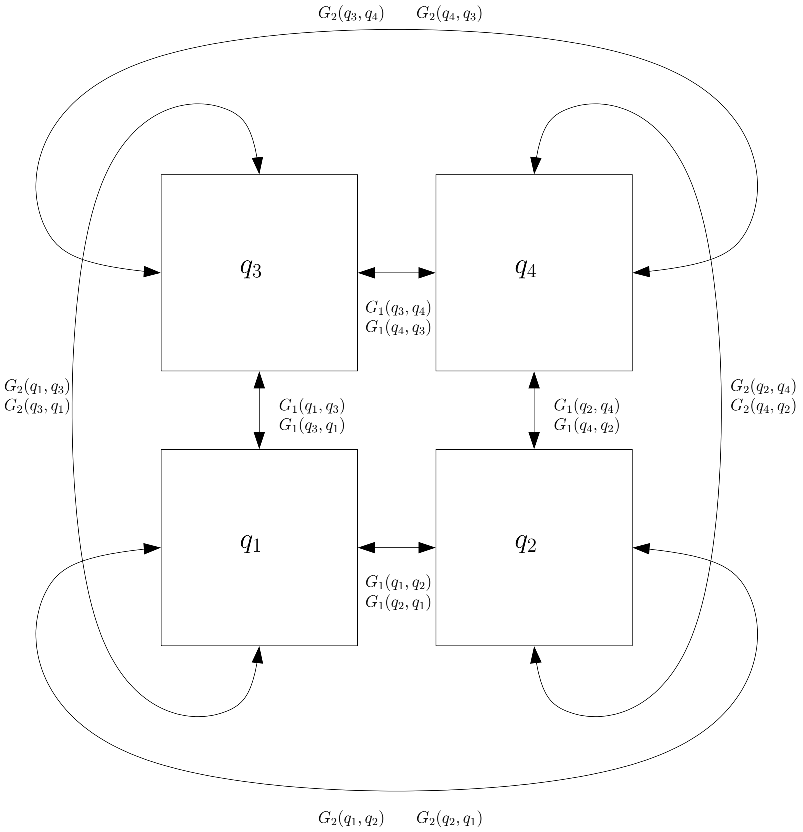
  $R_2(q_2, q_1, [x, y, \theta_1, \theta_2]) = R_2(q_4, q_3, [x, y, \theta_1, \theta_2]) = [x, y, \theta_1 - 2\pi, \theta_2]$,

  $R_2(q_3, q_1, [x, y, \theta_1, \theta_2]) = R_2(q_4, q_2, [x, y, \theta_1, \theta_2]) = [x, y, \theta_1, \theta_2 - 2\pi]$;

- *Init* : The initial conditions depend on the specific verification scenario.

Extending this hybridization to one with more discrete states in each dimension is trivial and has been automated by a Java program. The program has been created to automatically synthesize an XML file defining a hybrid automata of arbitrary discrete dimension. The model files created with this program can be directly loaded into PHAVer.

The hybridization for the three state model in equation (4.2.13) is similar to that of the four state model but must be discretized in three dimensions since the state dynamics are dependent on all three states. Again, only two discrete states are defined in each dimension in order to make the example as simple as possible. In this case the discrete modes are defined by equally sized hyper-rectangles. The bounds on the derivatives within each discrete mode are determined in the same way as before but this time using the three state model. The parameters of the LHA model are defined below and Figure 5.1.2 shows a directed graph representing the hybridized system.



$$D(q_1) = \{[x, y, \theta_1, \theta_2] \in X : 0 \leq \theta_1 \leq \pi, 0 \leq \theta_2 \leq \pi\}$$

$$D(q_2) = \{[x, y, \theta_1, \theta_2] \in X : \pi \leq \theta_1 \leq 2\pi, 0 \leq \theta_2 \leq \pi\}$$

$$D(q_3) = \{[x, y, \theta_1, \theta_2] \in X : 0 \leq \theta_1 \leq \pi, \pi \leq \theta_2 \leq 2\pi\}$$

$$D(q_4) = \{[x, y, \theta_1, \theta_2] \in X : \pi \leq \theta_1 \leq 2\pi, \pi \leq \theta_2 \leq 2\pi\}$$

Figure 5.1.1: The directed graph of the four state LHA model.

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$;

- $X = \{[x_{low}, x_{hi}] \times [y_{low}, y_{hi}] \times [0, 2\pi)\}$;

- $D(q_1) = \{[x, y, \theta] \in X : 0 \leq \theta < \pi, x_{low} \leq x < x_{mid}, y_{low} \leq y < y_{mid}\}$,

  $D(q_2) = \{[x, y, \theta] \in X : \pi \leq \theta < 2\pi, x_{low} \leq x < x_{mid}, y_{low} \leq y < y_{mid}\}$,

  $D(q_3) = \{[x, y, \theta] \in X : 0 \leq \theta < \pi, x_{mid} \leq x < x_{hi}, y_{low} \leq y < y_{mid}\}$,

  $D(q_4) = \{[x, y, \theta] \in X : \pi \leq \theta < 2\pi, x_{mid} \leq x < x_{hi}, y_{low} \leq y < y_{mid}\}$,

  $D(q_5) = \{[x, y, \theta] \in X : 0 \leq \theta < \pi, x_{low} \leq x < x_{mid}, y_{mid} \leq y < y_{hi}\}$,

  $D(q_6) = \{[x, y, \theta] \in X : \pi \leq \theta < 2\pi, x_{low} \leq x < x_{mid}, y_{mid} \leq y < y_{hi}\}$,

$D(q_7) = \{[x, y, \theta] \in X : 0 \leq \theta < \pi, x_{mid} \leq x < x_{hi}, y_{mid} \leq y < y_{hi}\}$,

$D(q_8) = \{[x, y, \theta] \in X : \pi \leq \theta < 2\pi, x_{mid} \leq x < x_{hi}, y_{mid} \leq y < y_{hi}\}$;

- $$f(q_i) \in \begin{pmatrix} \left[\min\limits_{[x,y,\theta]\in D(q_i)}(-v_{c1} + v_{c2}\cos\theta'_{21} + \omega_1 y'_{21}), \max\limits_{[x,y,\theta]\in D(q_i)}(-v_{c1} + v_{c2}\cos\theta'_{21} + \omega_1 y'_{21})\right] \\ \left[\min\limits_{[x,y,\theta]\in D(q_i)}(v_{c2}\sin\theta'_{21} - \omega_1 x'_{21}), \max\limits_{[x,y,\theta]\in D(q_i)}(v_{c2}\sin\theta'_{21} - \omega_1 x'_{21})\right] \\ \omega_2 - \omega_1 \end{pmatrix};$$

- $E = \{(q_1, q_2), (q_2, q_1), (q_1, q_3), (q_3, q_1), (q_2, q_4), (q_4, q_2), (q_3, q_4), (q_4, q_3),$
$(q_5, q_6), (q_6, q_5), (q_5, q_7), (q_7, q_5), (q_6, q_8), (q_8, q_6), (q_7, q_8), (q_8, q_7),$
$(q_1, q_5), (q_5, q_1), (q_2, q_6), (q_6, q_2), (q_3, q_7), (q_7, q_3), (q_4, q_8), (q_8, q_4)\}$;

- $G_1(q_1, q_2) = \{[x, y, \theta] \in X : \theta \geq \pi\}$, $G_2(q_1, q_2) = \{[x, y, \theta] \in X : \theta \leq 0\}$,
$G_1(q_2, q_1) = \{[x, y, \theta] \in X : \theta \leq \pi\}$, $G_2(q_2, q_1) = \{[x, y, \theta] \in X : \theta \geq 2\pi\}$,
$G(q_1, q_3) = \{[x, y, \theta] \in X : x \geq x_{mid}\}$,
$G(q_3, q_1) = \{[x, y, \theta] \in X : x \leq x_{mid}\}$,
$G(q_2, q_4) = \{[x, y, \theta] \in X : x \geq x_{mid}\}$,
$G(q_4, q_2) = \{[x, y, \theta] \in X : x \leq x_{mid}\}$,
$G_1(q_3, q_4) = \{[x, y, \theta] \in X : \theta \geq \pi\}$, $G_2(q_3, q_4) = \{[x, y, \theta] \in X : \theta \leq 0\}$,
$G_1(q_4, q_3) = \{[x, y, \theta] \in X : \theta \leq \pi\}$, $G_2(q_4, q_3) = \{[x, y, \theta] \in X : \theta \geq 2\pi\}$
$G_1(q_5, q_6) = \{[x, y, \theta] \in X : \theta \geq \pi\}$, $G_2(q_5, q_6) = \{[x, y, \theta] \in X : \theta \leq 0\}$,
$G_1(q_6, q_5) = \{[x, y, \theta] \in X : \theta \leq \pi\}$, $G_2(q_6, q_5) = \{[x, y, \theta] \in X : \theta \geq 2\pi\}$,
$G(q_5, q_7) = \{[x, y, \theta] \in X : x \geq x_{mid}\}$,
$G(q_7, q_5) = \{[x, y, \theta] \in X : x \leq x_{mid}\}$,
$G(q_6, q_8) = \{[x, y, \theta] \in X : x \geq x_{mid}\}$,
$G(q_8, q_6) = \{[x, y, \theta] \in X : x \leq x_{mid}\}$,
$G_1(q_7, q_8) = \{[x, y, \theta] \in X : \theta \geq \pi\}$, $G_2(q_7, q_8) = \{[x, y, \theta] \in X : \theta \leq 0\}$,
$G_1(q_8, q_7) = \{[x, y, \theta] \in X : \theta \leq \pi\}$, $G_2(q_8, q_7) = \{[x, y, \theta] \in X : \theta \geq 2\pi\}$
$G(q_1, q_5) = \{[x, y, \theta] \in X : y \geq y_{mid}\}$,
$G(q_5, q_1) = \{[x, y, \theta] \in X : y \geq y_{mid}\}$,
$G(q_2, q_6) = \{[x, y, \theta] \in X : y \leq y_{mid}\}$,

$G(q_6, q_2) = \{[x, y, \theta] \in X : y \leq y_{mid}\}$,

$G(q_3, q_7) = \{[x, y, \theta] \in X : y \geq y_{mid}\}$,

$G(q_7, q_3) = \{[x, y, \theta] \in X : y \leq y_{mid}\}$,

$G(q_4, q_8) = \{[x, y, \theta] \in X : y \geq y_{mid}\}$,

$G(q_8, q_4) = \{[x, y, \theta] \in X : y \leq y_{mid}\}$;

- $R_1(q_1, q_2, [x, y, \theta]) = R_1(q_2, q_1, [x, y, \theta]) = R_1(q_3, q_4, [x, y, \theta]) = R_1(q_4, q_3, [x, y, \theta]) = R_1(q_5, q_6, [x, y, \theta]) = R_1(q_7, q_8, [x, y, \theta]) = R_1(q_8, q_7, [x, y, \theta]) = R_1(q_4, q_3, [x, y, \theta]) = R(q_1, q_3, [x, y, \theta]) = R(q_3, q_1, [x, y, \theta]) = R(q_2, q_4, [x, y, \theta]) = R(q_4, q_2, [x, y, \theta]) = R(q_5, q_7, [x, y, \theta]) = R(q_7, q_5, [x, y, \theta]) = R(q_6, q_8, [x, y, \theta]) = R(q_8, q_6, [x, y, \theta]) = R(q_1, q_5, [x, y, \theta]) = R(q_5, q_1, [x, y, \theta]) = R(q_2, q_6, [x, y, \theta]) = R(q_6, q_2, [x, y, \theta]) = R(q_3, q_7, [x, y, \theta]) = R(q_7, q_3, [x, y, \theta]) = R(q_4, q_8, [x, y, \theta]) = R(q_8, q_4, [x, y, \theta]) = [x, y, \theta]$,

  $R_2(q_1, q_2, [x, y, \theta]) = R_2(q_3, q_4, [x, y, \theta]) = R_2(q_5, q_6, [x, y, \theta]) = R_2(q_7, q_8, [x, y, \theta]) = [x, y, \theta + 2\pi]$,

  $R_2(q_2, q_1, [x, y, \theta]) = R_2(q_4, q_3, [x, y, \theta]) = R_2(q_6, q_5, [x, y, \theta]) = R_2(q_8, q_7, [x, y, \theta]) = [x, y, \theta - 2\pi]$;

- *Init* : The initial conditions depend on the specific verification scenario.

As with the four state model, a Java program has been developed for the three state model to automatically generate XML files for PHAVer with arbitrary numbers of states in each discrete dimension.

The LGG support function algorithm is capable of calculating reachable sets for hybrid automata defined with affine dynamics and bounded inputs within each discrete mode. It makes sense to develop an AHA model since a more accurate hybridization of the system can be created by using affine dynamics rather than just constant bounds on the derivatives. The AHA model is created from the three-state model of the system to minimize the number of continuous states. Since the derivatives of the states are linear with respect to x'_{21} and y'_{21}, the system only needs to be linearized about θ'_{21}. As with the previous examples, only two discrete states are defined here in order to maintain simplicity. To develop the affine

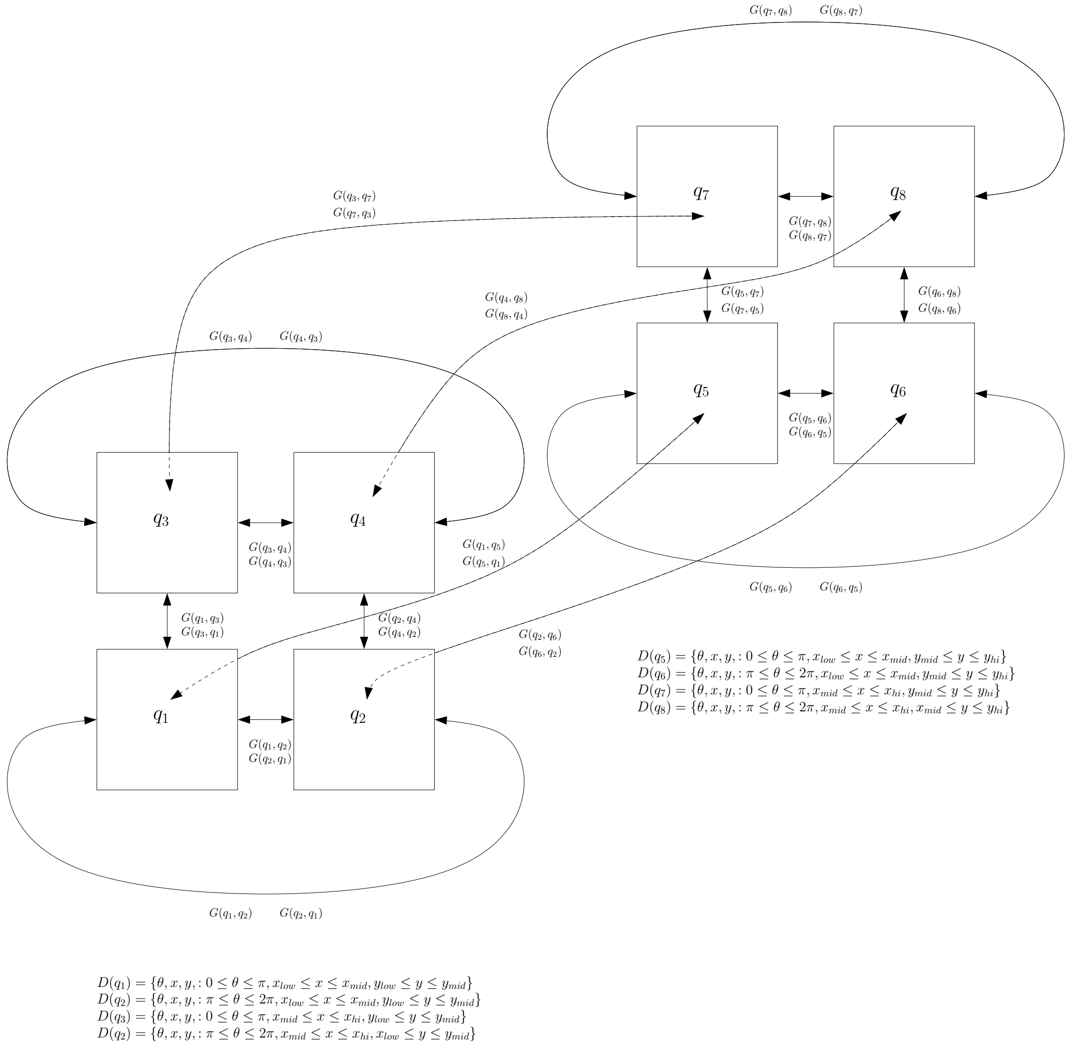

$G(q_7, q_8)$ $G(q_8, q_7)$
q_7
q_8
$G(q_3, q_7)$
$G(q_7, q_3)$
$G(q_7, q_8)$
$G(q_8, q_7)$
$G(q_4, q_8)$
$G(q_8, q_4)$
$G(q_5, q_7)$
$G(q_7, q_5)$
$G(q_6, q_8)$
$G(q_8, q_6)$
$G(q_3, q_4)$ $G(q_4, q_3)$
q_5
q_6
$G(q_5, q_6)$
$G(q_6, q_5)$
q_3
q_4
$G(q_3, q_4)$
$G(q_4, q_3)$
$G(q_1, q_5)$
$G(q_5, q_1)$
$G(q_5, q_6)$ $G(q_6, q_5)$
$G(q_1, q_3)$
$G(q_3, q_1)$
$G(q_2, q_4)$
$G(q_4, q_2)$
$G(q_2, q_6)$
$G(q_6, q_2)$
$D(q_5) = \{\theta, x, y, : 0 \leq \theta \leq \pi, x_{low} \leq x \leq x_{mid}, y_{mid} \leq y \leq y_{hi}\}$
$D(q_6) = \{\theta, x, y, : \pi \leq \theta \leq 2\pi, x_{low} \leq x \leq x_{mid}, y_{mid} \leq y \leq y_{hi}\}$
$D(q_7) = \{\theta, x, y, : 0 \leq \theta \leq \pi, x_{mid} \leq x \leq x_{hi}, y_{mid} \leq y \leq y_{hi}\}$
$D(q_8) = \{\theta, x, y, : \pi \leq \theta \leq 2\pi, x_{mid} \leq x \leq x_{hi}, x_{mid} \leq y \leq y_{hi}\}$
q_1
q_2
$G(q_1, q_2)$
$G(q_2, q_1)$
$G(q_1, q_2)$ $G(q_2, q_1)$
$D(q_1) = \{\theta, x, y, : 0 \leq \theta \leq \pi, x_{low} \leq x \leq x_{mid}, y_{low} \leq y \leq y_{mid}\}$
$D(q_2) = \{\theta, x, y, : \pi \leq \theta \leq 2\pi, x_{low} \leq x \leq x_{mid}, y_{low} \leq y \leq y_{mid}\}$
$D(q_3) = \{\theta, x, y, : 0 \leq \theta \leq \pi, x_{mid} \leq x \leq x_{hi}, y_{low} \leq y \leq y_{mid}\}$
$D(q_2) = \{\theta, x, y, : \pi \leq \theta \leq 2\pi, x_{mid} \leq x \leq x_{hi}, x_{low} \leq y \leq y_{mid}\}$

Figure 5.1.2: The directed graph of the three state LHA model.

hybridization of the model, the dynamics must be linearized in each discrete mode. In this example, the derivatives are linearized about an operating point at the center of the domain in each discrete mode; other linearization techniques would have likely yielded less error but this approach is the simplest for the discussion of the hybridization process.

The expression for $\dot{x}$ in equation (4.2.13) contains the non-linear term $\cos \theta'_{21}$. Let the linearization of this term in discrete mode q_i be represented by $a_i \theta'_{21} + b_i$. With respect to the operating point θ_{m_i}, which lies at the center of the domain $D(q_i)$, the linearization is given by the expression

$$a_i\theta'_{21} + b_i = -\sin(\theta_{m_i})(\theta'_{21} - \theta_{m_i}) + \cos(\theta_{m_i}). \tag{5.1.1}$$

To account for linearization errors and ensure that the hybrid automata model over-approximates the behavior of the actual non-linear system, an error term must be added. For the LGG support function algorithm, the error term can be added to the dynamics in the same manner as a bounded input. Let the error term of the affine expression for $\dot{x}$ in mode q_i be represented by ϵ_{x_i}. The value of ϵ_{x_i} is given by

$$\epsilon_{x_i} = \cos\theta'_{21} - (a_i\theta'_{21} + b_i). \tag{5.1.2}$$

Likewise, the non-linear term $\sin\theta'_{21}$ appears in the expression for $\dot{y}$ in equation (4.2.12). Let the linearization of this term in discrete mode q_i be represented by $c_i\theta'_{21} + d_i$ and the error term by ϵ_{y_i}. These terms are given by

$$c_i\theta'_{21} + d_i = \cos(\theta_{m_i})(\theta'_{21} - \theta_{m_i}) + \sin(\theta_{m_i}); \tag{5.1.3}$$

$$\epsilon_{y_i} = \sin\theta'_{21} - (c_i\theta'_{21} + d_i). \tag{5.1.4}$$

With these preliminary terms defined, the complete parametrization of the hybrid automata model is given below. Figure 5.1.3 shows a directed graph representing the hybridized system.

- $Q = \{q_1, q_2\}$;
- $X = \{\mathbb{R}^n \times \mathbb{R}^n \times [0, 2\pi)\}$;
- $D(q_1) = \{[x'_{21}, y'_{21}, \theta'_{21}] \in X : 0 \leq \theta'_{21} < \pi\}$,
  $D(q_2) = \{[x'_{21}, y'_{21}, \theta'_{21}] \in X : \pi \leq \theta'_{21} < 2\pi\}$;

- $f(q_i) \in \begin{pmatrix} -v_{c1} + v_{c2}(a_i\theta'_{21} + b_i + \epsilon_{x_i}) + \omega_1 y'_{21} \\ -v_{c1} + v_{c2}(c_i\theta'_{21} + d_i + \epsilon_{y_i}) + \omega_1 y'_{21} \\ \omega_2 - \omega_1 \end{pmatrix};$

- $E = \{(q_1, q_2), (q_2, q_1)\};$

- $G_1(q_1, q_2) = \{[x, y'_{21}, \theta'_{21}] \in X : \theta'_{21} \geq \pi\},$
  $G_2(q_1, q_2) = \{[x'_{21}, y'_{21}, \theta'_{21}] \in X : \theta'_{21} \leq 0\},$
  $G_1(q_2, q_1) = \{[x'_{21}, y'_{21}, \theta'_{21}] \in X : \theta'_{21} \leq \pi\},$
  $G_2(q_2, q_1) = \{[x'_{21}, y'_{21}, \theta'_{21}] \in X : \theta'_{21} \geq 2\pi\};$

- $R_1(q_1, q_2, [x'_{21}, y'_{21}, \theta'_{21}]) = R_1(q_2, q_1, [x'_{21}, y'_{21}, \theta'_{21}]) = [x'_{21}, y'_{21}, \theta'_{21}],$
  $R_2(q_1, q_2, [x'_{21}, y'_{21}, \theta'_{21}]) = [x'_{21}, y'_{21}, \theta'_{21} + 2\pi],$
  $R_2(q_2, q_1, [x'_{21}, y'_{21}, \theta'_{21}]) = [x'_{21}, y'_{21}, \theta'_{21} - 2\pi];$

- *Init* : The initial conditions depend on the specific verification scenario.

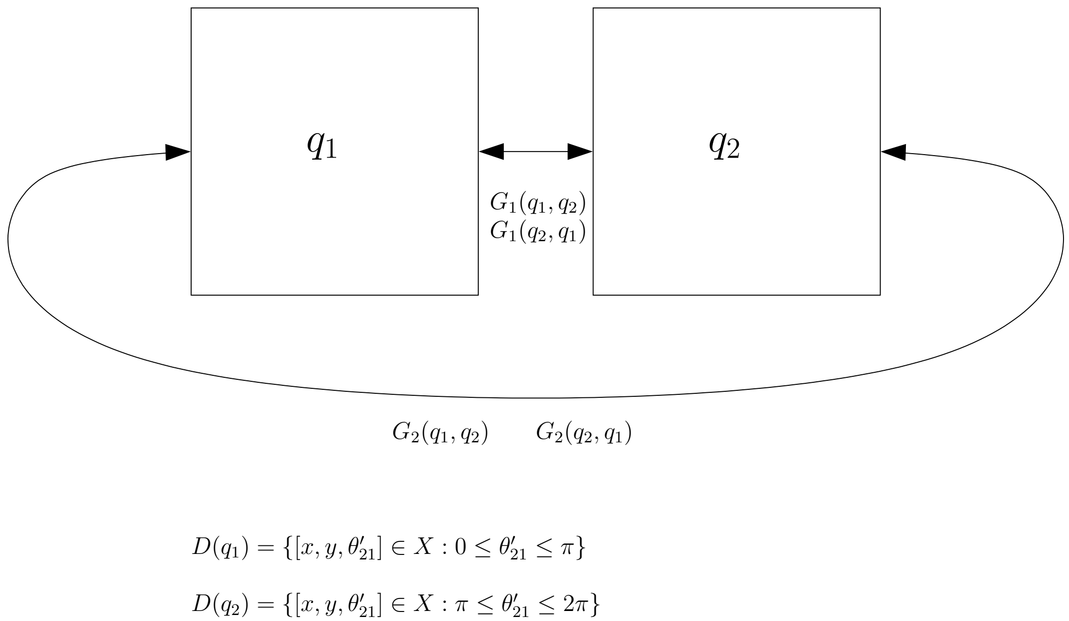

q_1
q_2
$G_1(q_1, q_2)$
$G_1(q_2, q_1)$
$G_2(q_1, q_2)$
$G_2(q_2, q_1)$
$D(q_1) = \{[x, y, \theta'_{21}] \in X : 0 \leq \theta'_{21} \leq \pi\}$
$D(q_2) = \{[x, y, \theta'_{21}] \in X : \pi \leq \theta'_{21} \leq 2\pi\}$

Figure 5.1.3: The directed graph of the affine hybrid automata model.

5.2 Setting Up the Reachability Calculations

Neither PHAVer nor the LGG support function algorithm can be directly applied to calculating reachable sets of non-linear systems. In order for these tools to be applied to the non-linear model of the differential drive robots, LHA and AHA approximations have been created. The specific LHA and AHA models were defined in the previous chapter with a minimum number of discrete states to clearly communicate the concept. However,when these hybrid automata are actually implemented, the number of discrete states must be much larger to yield reasonably accurate approximations. Unfortunately, neither of these reachability tools is particularly well suited for handling hybrid automata with large discrete dimension. The limitations on the discrete dimensions are a result of the fact that each discrete transition spawns a new flow pipe in the subsequent discrete mode. Calculating the BRS of the system with either of the tools requires the terminal conditions to include the entire range of possible values for θ_1 and θ_2. This means that the tools would initially create separate simulations within every discrete mode of the hybrid automata, each of which would subsequently spawn additional flow pipe simulations at each discrete transition. Although the calculation might terminate eventually on a system with enough memory and sufficient processor speed, calculating the reachable set in this manner is beyond the capability of the system that was used for this work. Calculating the FRS of the system happens to require much less intensive processing capabilities for this scenario, which is the reason that it was selected for this application.

There are at least two ways that the FRS could conceptually be integrated with the Simplex architecture to yield provable run time safety assurance. The first approach is to calculate the FRS of the system in real time at each discrete time step and then subsequently check whether the calculated reachable set intersects the avoid set. The safety controller would be enabled if the bloated reachable set for either safety controller were found to intersect the avoid set. Although conceptually valid, this approach is not yet generally feasible because current reachability algorithms are too slow to calculate reachable sets in real time. The alternative approach is to compute the FRS for a particular set of initial conditions and then apply a transformation during runtime to find the reachable set of the system for the actual initial conditions at each time step. This approach is not applicable to all systems because it is not always possible to find a simple transformation between trajectories of the system for different initial conditions.

It turns out that there is a transformation which relates trajectories of the relative states of the differential drive robot scenario. Consider the four state model with two possible sets of initial states, $\mathcal{X}_0 = \{x_{21_0}, y_{21_0}, \theta_{1_0}, \theta_{2_0}\}$ and $\tilde{\mathcal{X}_0} = \{\tilde{x}_{21_0}, \tilde{y}_{21_0}, \tilde{\theta}_{1_0}, \tilde{\theta}_{2_0}\}$, such that $\mathcal{X}_0$ and $\tilde{\mathcal{X}_0}$ satisfy

$$\tilde{\mathcal{X}_0} = \begin{bmatrix} \tilde{x}_{21_0} \\ \tilde{y}_{21_0} \\ \tilde{\theta}_{1_0} \\ \tilde{\theta}_{2_0} \end{bmatrix} = \begin{bmatrix} x_{21_0} + x_t \\ y_{21_0} + y_t \\ \theta_{1_0} + \theta_t \\ \theta_{2_0} + \theta_t \end{bmatrix}, \tag{5.2.5}$$

where x_t, y_t, θ_t are constant parameters. The trajectories of the system for each set of initial conditions are then related by

$$\tilde{\mathcal{X}}(t) = \begin{bmatrix} \tilde{x}_{21}(t) \\ \tilde{y}_{21}(t) \\ \tilde{\theta}_1(t) \\ \tilde{\theta}_2(t) \end{bmatrix} = \begin{bmatrix} (x_{21}(t) - x_{21_0}) \cos\theta_t - (y_{21}(t) - y_{21_0}) \sin\theta_t + x_{21_0} + x_t \\ (x_{21}(t) - x_{21_0}) \sin\theta_t + (y_{21}(t) - y_{21_0}) \cos\theta_t + y_{21_0} + y_t \\ \theta_1(t) + \theta_t \\ \theta_2(t) + \theta_t \end{bmatrix}. \tag{5.2.6}$$

The implication of this convenient property of the system is that knowledge of just one trajectory with $\theta_2 - \theta_1 = c$, where c is an arbitrary constant, produces any trajectory of the system with $\theta_2 - \theta_1 = c$ by simply applying the transformation in equation (5.2.6). A proof of this proposition is provided in Appendix IX. Clearly, the property is also applicable to the reachable sets of the system. If this transformation is utilized in the decision module of the Simplex architecture, then it is only necessary to pre-compute the reachable set for a particular value of x_{21_0}, a particular value of y_{21_0}, and all possible values of the term $\theta_{2_0} - \theta_{1_0}$. In order to make the transformation as simple as possible, the initial states that have been chosen are $x_{21_0} = 0, y_{21_0} = 0, \theta_{1_0} = 0$, and $\theta_{2_0} \in [0, 2\pi)$.

To perform the reachability calculation, an appropriate time horizon for the reachable set must first be determined. This derivation of the appropriate time horizon assumes that $\mathcal{A}$ is a convex shape parametrized in two dimensions with respect to the pair $\{x_{21}, y_{21}\}$. In the implemented scenario, the avoid set is specifically formulated as a circular set. A circular set is indeed convex and therefore satisfies the preceding assumption. Consider the three state modified model of the system that is expressed in equation (4.3.16). The minimum exterior angle of a convex avoid set is 180°. Since $\omega \in [-1, 1]$, the robot can turn either clockwise or anti-clockwise and the maximum necessary change in heading θ_{21} to not collide with $\mathcal{A}$ is $\frac{\pi}{2}$. Refer to Figure 5.2.4 for an illustration of this concept. Stated another way, the smaller angle formed by the intersection of two lines is never more than 90°. The time that it takes the the system to change heading by $\frac{\pi}{2}$ radians in either direction is given by $T_f = \frac{\pi}{2\omega}$. Therefore, this is the maximum time horizon for which the reachable set needs to be calculated. Note that the actual avoid set for this scenario does not have any straight edges, which means that the FRS will actually be over-calculated for this particular case.

One way to force the reachability tools to terminate the calculation at time T_f is to introduce a continuous clock variable t and a discrete termination state q_{T_f}. The clock variable is configured so that its initial state is zero and its derivative is one within all of the discrete states, with the exception of the termination state. The termination state is reachable from any discrete mode, with the transition guard for all modes defined by $G(q_i, q_{T_f}) \geq \{t \geq T_f\}$. Termination of the calculation is forced within the termination state by setting the derivatives of all the continuous states to zero.

Although both tools may be capable of directly calculating the FRS for the chosen initial states, it turned out to be faster and more convenient to calculate subsets of the range over θ_{2_0} in individual calculations. A program was created in c++ to automatically configure and run each tool for the necessary subsets of the initial conditions. The tool is also used to automatically concatenate the results into individual polygons for each subset. A MATLAB script was created to produce a visualization of the reachable set generated by the c++ program.

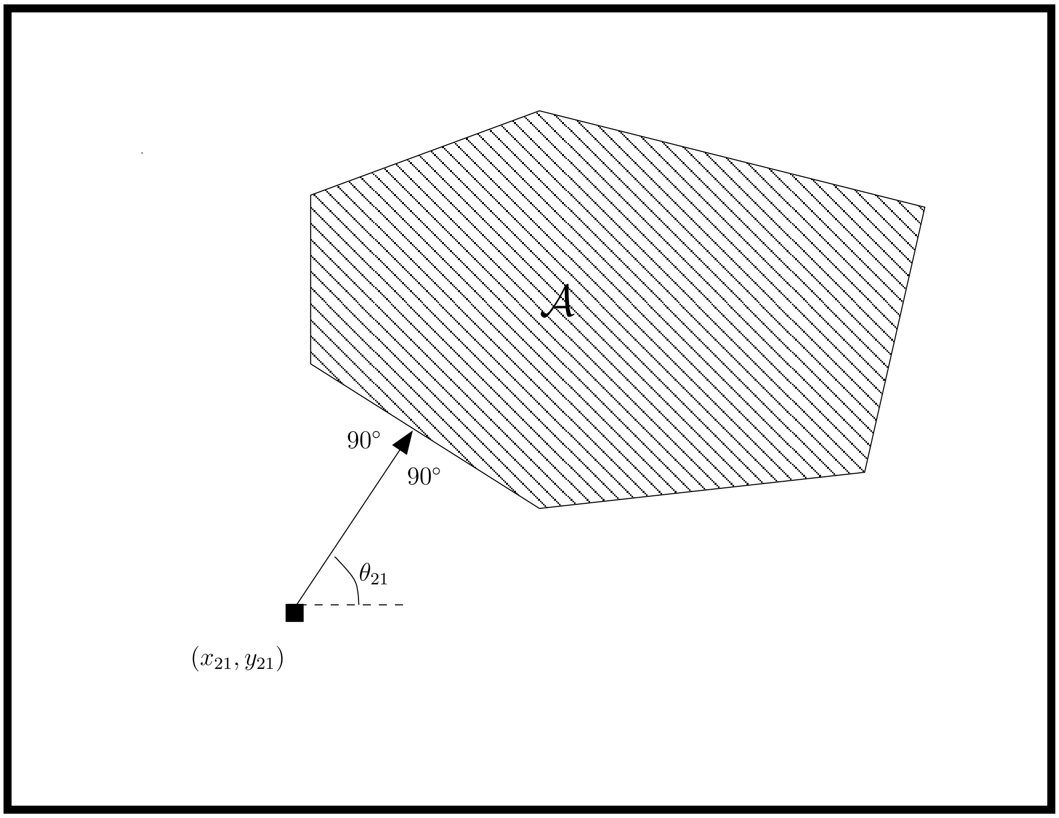


Figure 5.2.4: Illustration of the minimum time-horizon of the reachability computation.

5.3 Results Generated by the Reachability Tools

The reachable set generated with PHAVer using the hybridization of the four state model, an initial value range of $2.75 \leq \theta_{2_0} \leq 2.95$, and the dynamics of the safety controller defined by equation (4.3.15) is shown in Figure 5.3.5. The blue curved lines in the figure are actual trajectories spanning the range of possible trajectories within the defined range of initial conditions. Note that the generated reachable set completely contains all of these individual trajectories and is therefore an over-approximation of the actual reachable set. Figure 5.3.6 gives a three dimensional depiction of the reachable set for all possible ranges of θ_{2_0}. In this case, the range of intervals of θ_{2_0} was split into 32 equal sized regions covering the entire domain.

Likewise, the reachable set that was generated with PHAVer using the three state model, an initial value range of $2.75 \leq \theta'_{2_0} \leq 2.95$, and the dynamics of the safety controller defined

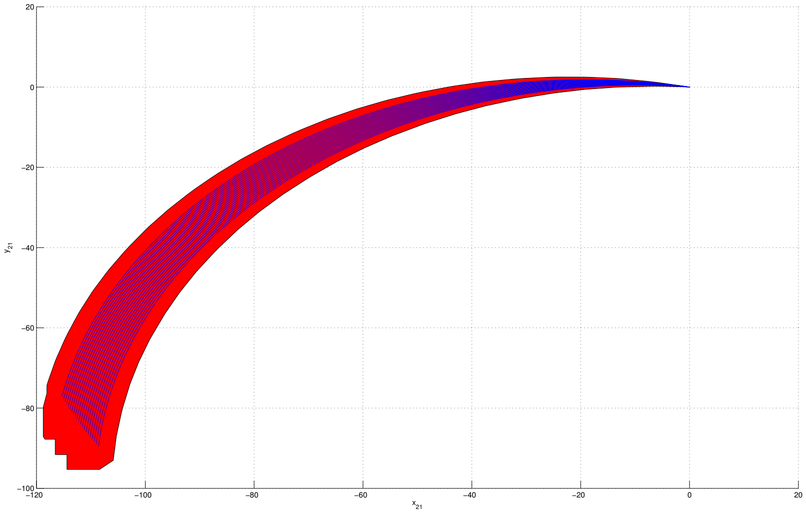


Figure 5.3.5: Subset of reachable set generated by PHAVer with exemplary trajectories plotted for the 4 state model.

by equation (4.3.15) is shown in Figure 5.3.7. Again, the blue curved lines represent actual trajectories of the system which span the range of possible trajectories defined within the range of possible initial conditions. Notice that this forwards reachable set contains all of the example trajectories and is therefore also an over-approximation. In fact, this reachable set over-approximates the actual reachable set by a much larger margin than the reachable set calculated with the four state model. The time that it took PHAVer to calculate the reachable set for the three state model was several times more than the time that it took for the four state model. Figure 5.3.8 gives a three dimensional depiction of the reachable set for all possible ranges of θ'_{2_0}. The range of intervals of θ'_{2_0} was again split into 32 equal sized regions covering the entire domain. The reachable sets generated with the two models are not directly comparable since they are defined with respect to different coordinate systems.

An attempt was made to calculate the FRS of the system with the LGG support function algorithm, the AHA model derived from equation (4.2.12), an initial value range of $2.75 \leq \theta'_{2_0} \leq 2.95$, and the dynamics of the safety controller defined by equation (4.3.15). The

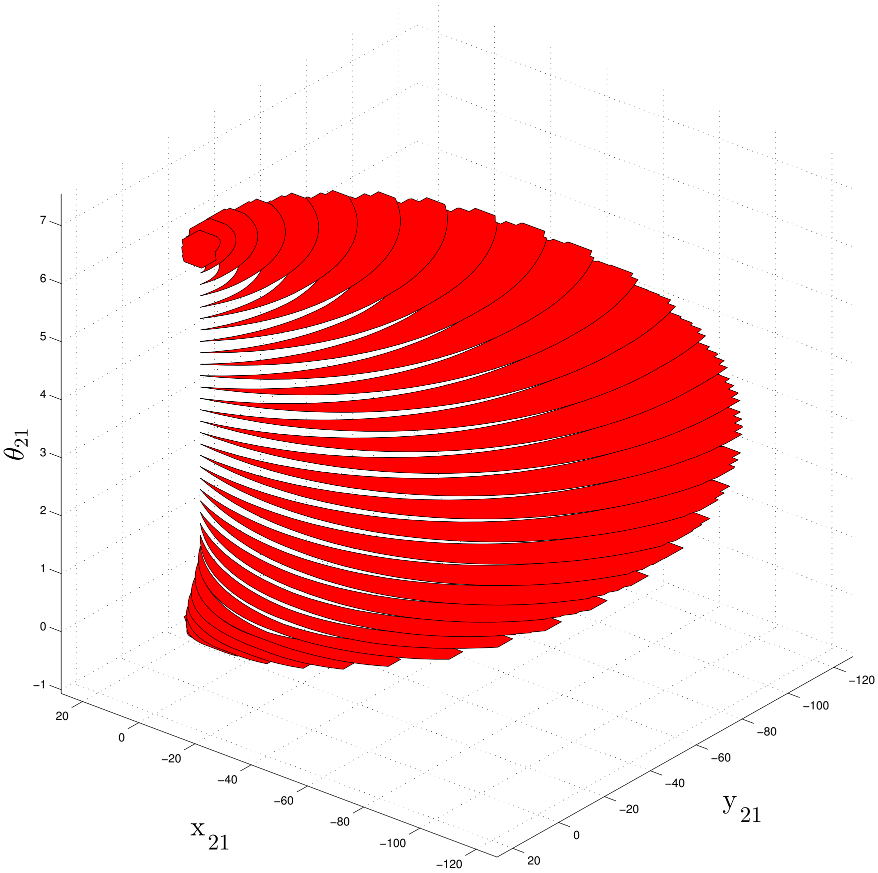


Figure 5.3.6: Three dimensional depiction of reachable set generated by PHAVer using the 4 state model.

FRS that was calculated for these conditions is shown in Figure 5.3.9. Notice that some of the example trajectories plotted in blue are not entirely contained within the reachable set. The presumed reason that the results are not an over-approximation of the actual reachable set is that the LGG support function algorithm is implemented without compensation for numerical errors. Moreover, the LGG tool was not capable of calculating the FRS for the entire range of intervals over θ'_{21}. This is not to say that the algorithm could not have completed the computation on a computer with more memory and superior processing speed.

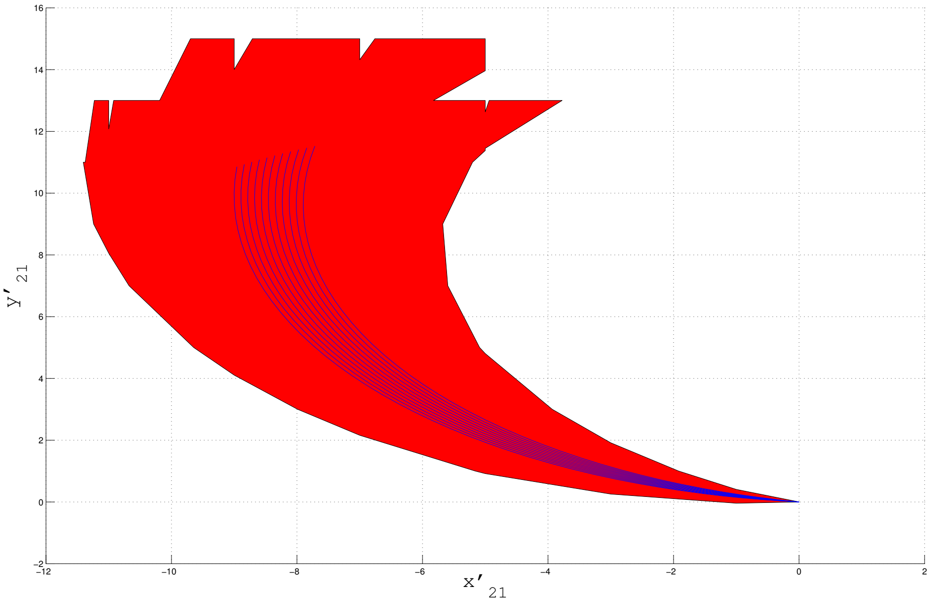


Figure 5.3.7: Subset of reachable set generated by PHAVer with exemplary trajectories plotted for the 3 state model.

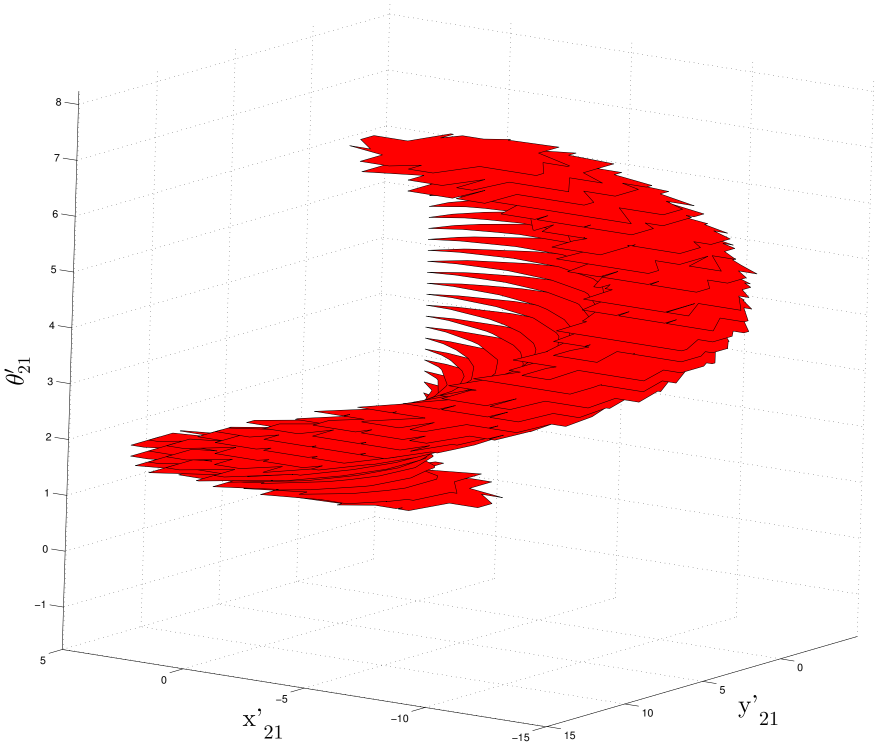


Figure 5.3.8: Three dimensional depiction of reachable set generated by PHAVer using the 3 state model.

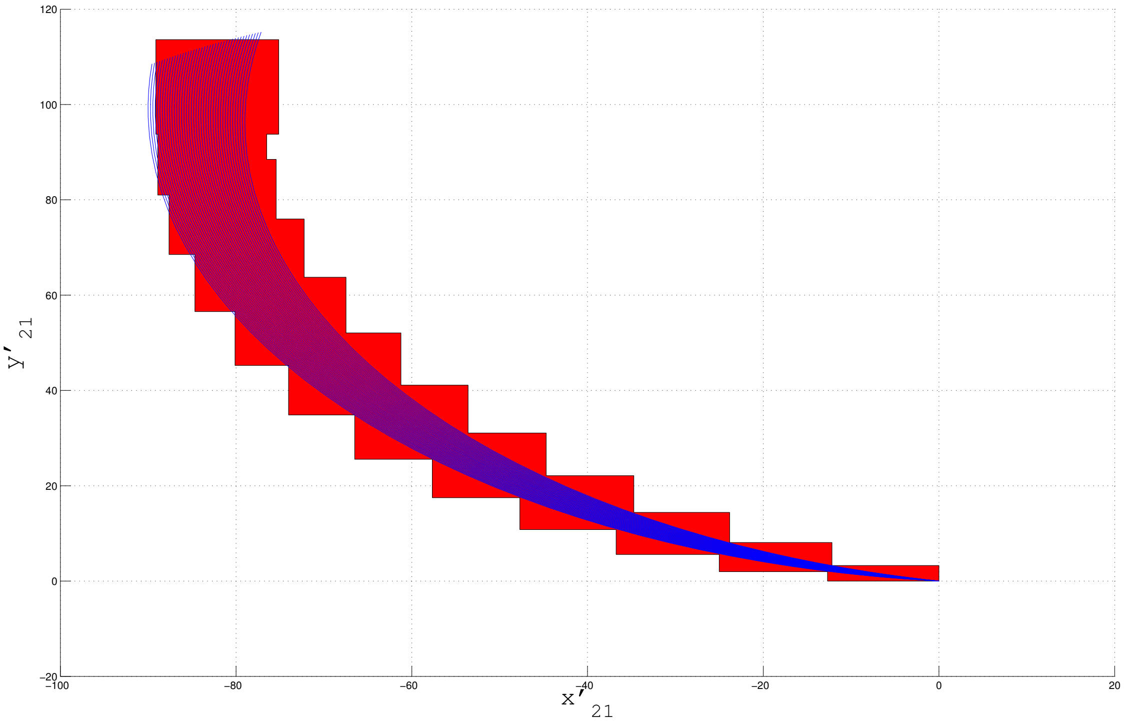


Figure 5.3.9: Subset of reachable set generated by the LGG support function algorithm plotted with exemplary trajectories for the 3 state model.

VI. SIMULATION AND RESULTS

A simulation of the differential drive robot collision avoidance scenario was programmed in Java to measure effectiveness and provide a visualization of an implementation of the generated reachable sets with the Simplex architecture. The simulation uses the reachable sets calculated with PHAVer for the four state model of the differential drive robots to implement the decision module. The other reachable sets that were generated were not integrated with the simulation because this model clearly provided the closest approximation of the actual reachable set and was also computed most efficiently.

The states of the robots are simulated using their actual kinematic model from equation (4.2.6). For an experimental controller, the line following controller that is described in [18] was used. The initial conditions and reference trajectories of the robots are random but are constrained in such a way that it is highly likely that the robots will collide when properly tracking their reference trajectories with the experimental controller. Note that the experimental controller makes no attempt to avoid collisions. Although the decision module could theoretically be implemented in a decentralized manner for identical robots, in practical applications imperfect knowledge of the states and numerical inaccuracies force the implementation to be centralized. That is, the decision of when to activate a recovery controller is shared by both robots and is implemented simultaneously for both. The avoid set $\mathcal{A}$ for this scenario is the region defined by

$$\mathcal{A} = \{(x_{21}, y_{21}, \theta_{21}) \in \mathcal{X} : x_{21}^2 + y_{21}^2 \leq d_{min}^2\}, \tag{6.0.1}$$

where d_{min} is the minimum distance between the robots that is considered safe. For these simulations, $d_{min} = 100$.

Recall that the format of the FRS that was calculated with PHAVer was a collection of polygons, each representing the FRS of the system for a particular value of $\theta_2 - \theta_1$ and for $\theta_{1_0} = 0$. As proven in Appendix IX, it is possible to find the FRS for any value of θ_{1_0} by applying a transformation to the the vertices of the polygon representations of the calculated FRS. Therefore, the actual reachable set at each time step is determined by transforming the FRS polygon associated with the current value of $\theta_2 - \theta_1$ with the transformation in equation 5.2.6, using the current values of θ_{1_0}, x_{21}, and y_{21}.

The two safety controllers that were defined in equation (4.3.14) and equation (4.3.15) allow the decision module to recover by making a hard turn in either the clockwise or counter-clockwise direction. When the FRS of just one of these controllers intersects $\mathcal{A}$, the decision module permits the experimental controller to maintain control of the system because it has at least one safety controller that can still be used to avoid $\mathcal{A}$. However, when the forwards reachable sets of both safety controllers intersect $\mathcal{A}$, the system is bound to violate the safety condition. Therefore, when only one of the safety controllers can still recover, the decision module will ideally enable that recovery controller one discrete time step before the the FRS of that controller could potentially enter $\mathcal{A}$. In order to ensure that the decision module does activate the recovery controller before it is possible for the FRS to enter $\mathcal{A}$ within one time step, the avoid set is bloated by the maximum distance that the system can travel in one discrete time step. The maximum distance Δ that can be traversed in one time step is determined by finding the maximum velocity in the differential state space and multiplying by the sampling period of the system. That is,

$$\Delta = \max_{\theta_1, \theta_2 \in [0, 2\pi]} \left[2v \sin\left(\frac{\theta_2 - \theta_1}{2}\right) T \right] = 2vT \tag{6.0.2}$$

In this case, the scenario was simulated with velocity $v = 50$ and sampling period $T = 0.01s$, so $\Delta = 1$.

One-thousand iterations of the scenario were simulated and the minimum distance between the two robots over all of the simulations was 100.963. Note that the reason the robots don't come closer to the actual minimum distance is because the avoid set is circular and the worst case value of Δ was calculated assuming that the avoid set could have straight edges.

The following images show the graphics that were developed to visualize the simulation. In each of the images, the robots are tracking the trajectory line of their associated color. A circle around the location of each robot indicates the minimum distance that the other robot should ever reach. The dotted lines provide a short history of the robots' trajectories. Finally, the diagram in the upper left-hand corner of the visualization shows the position of the robots and the reachable sets of the recovery controllers with respect to the relative coordinate system of the four state model. Figure 6.0.1 shows a screen shot of the simulation as the robots are moving towards each other. Figure 6.0.3 shows a screen shot of the simulation while the recovery controllers are actively being used to avoid a collision. Figure 6.0.5 shows a screen shot of the robots after the collision has been avoided and the experimental controller is reactivated. Finally, Figures 6.0.2,6.0.4, and 6.0.6 show larger views of the relative coordinate system depiction in the upper left hand corners of Figures 6.0.1, 6.0.3, and 6.0.5, respectively.

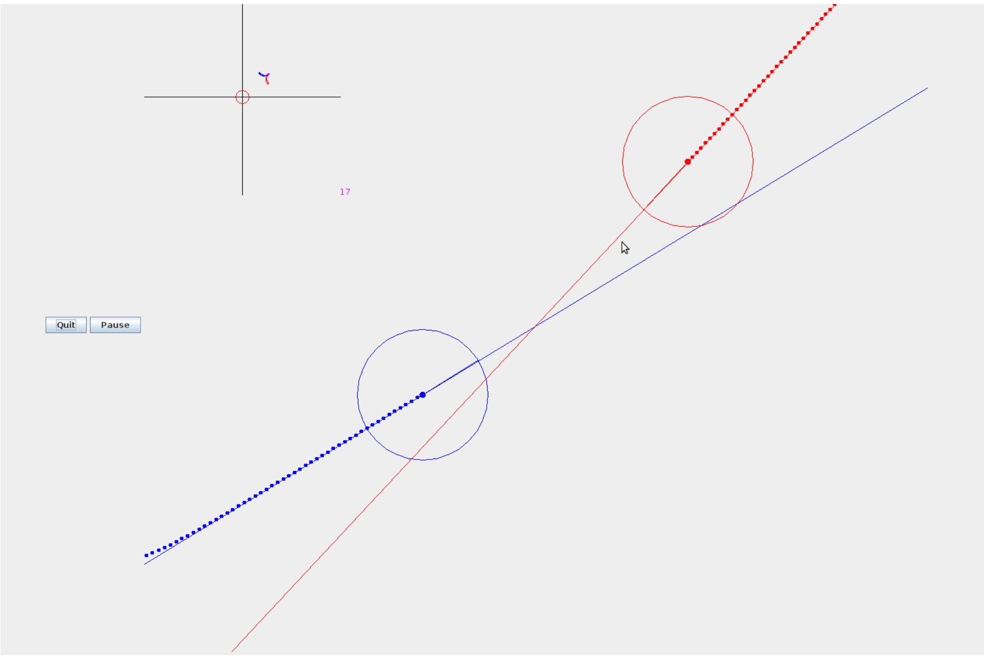


Figure 6.0.1: Screen shot of the simulation as the robots are moving towards each other.

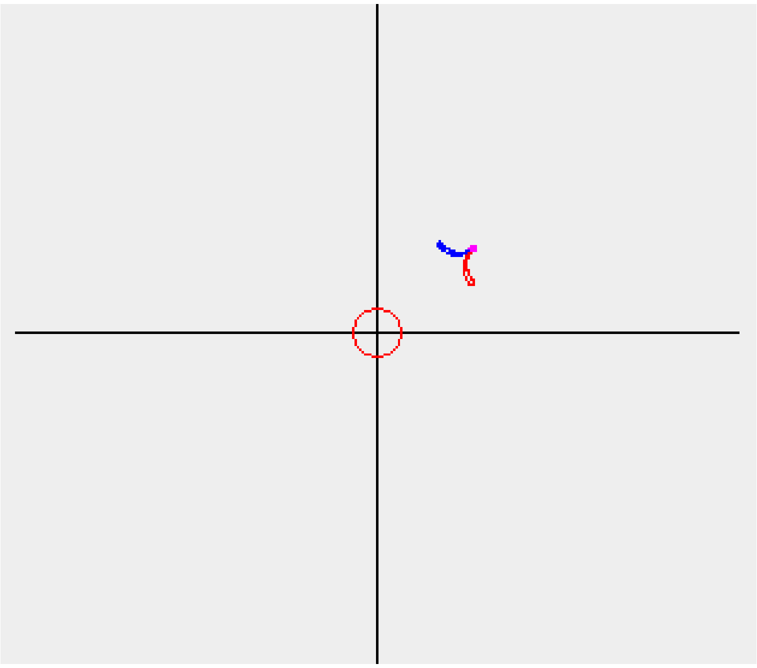


Figure 6.0.2: Relative coordinate system and reachable sets as the robots are moving towards each other.

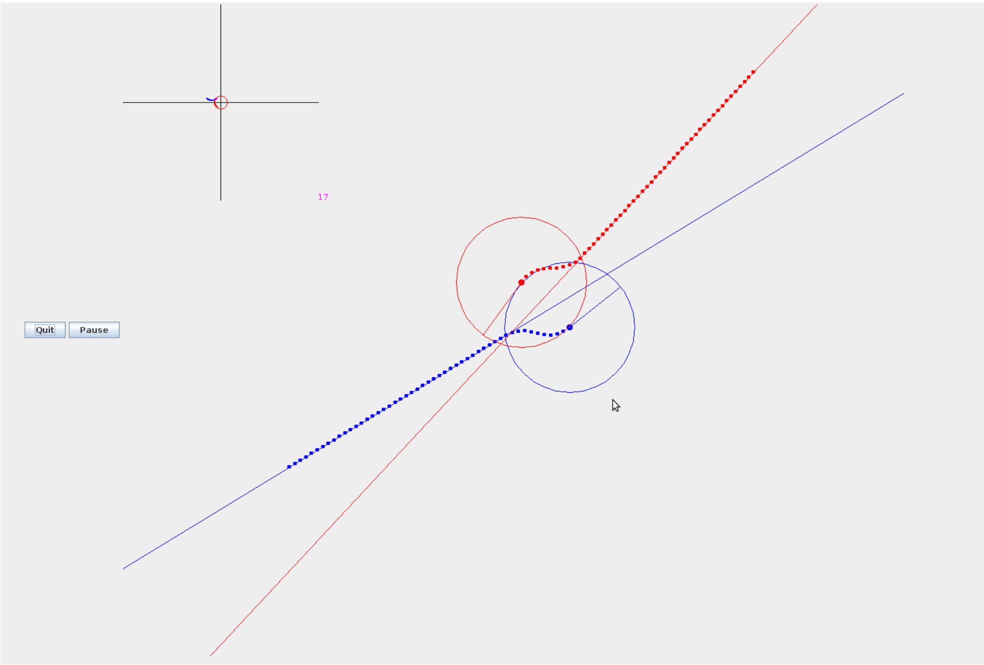


Figure 6.0.3: Screen shot of the simulation while the recovery controllers are actively being used to avoid a collision.

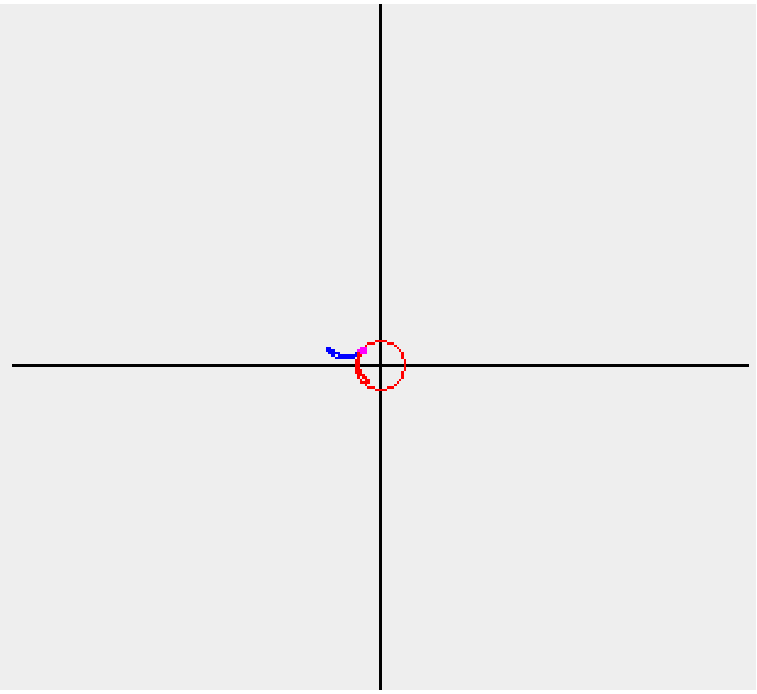


Figure 6.0.4: Relative coordinate system and reachable sets as the recovery controllers are actively being used to avoid a collision.

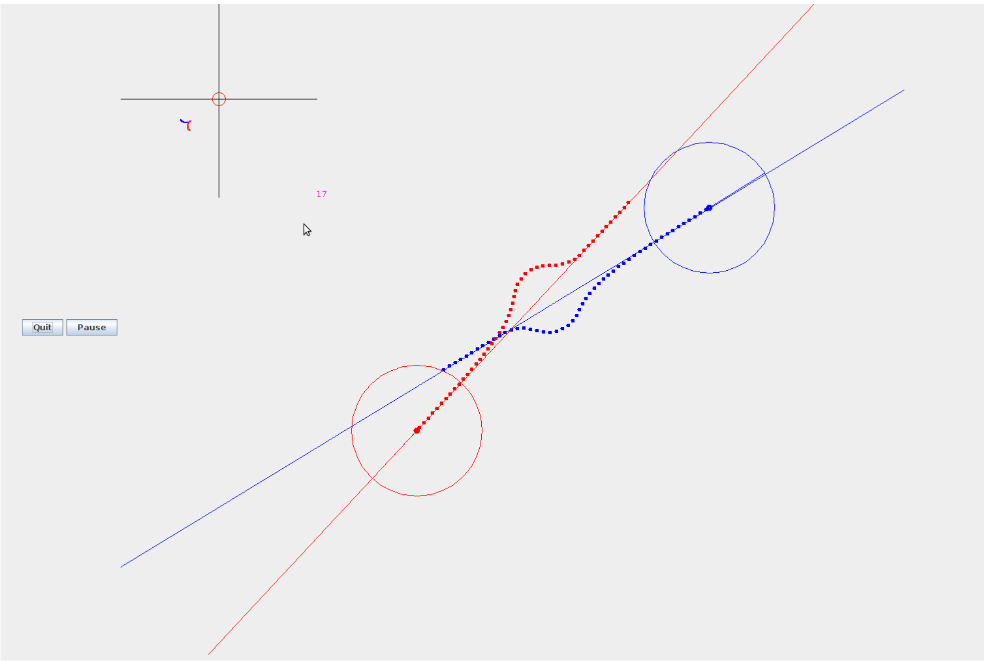


Figure 6.0.5: Screen shot of the robots after the collision has been avoided and the experimental controller is reactivated.

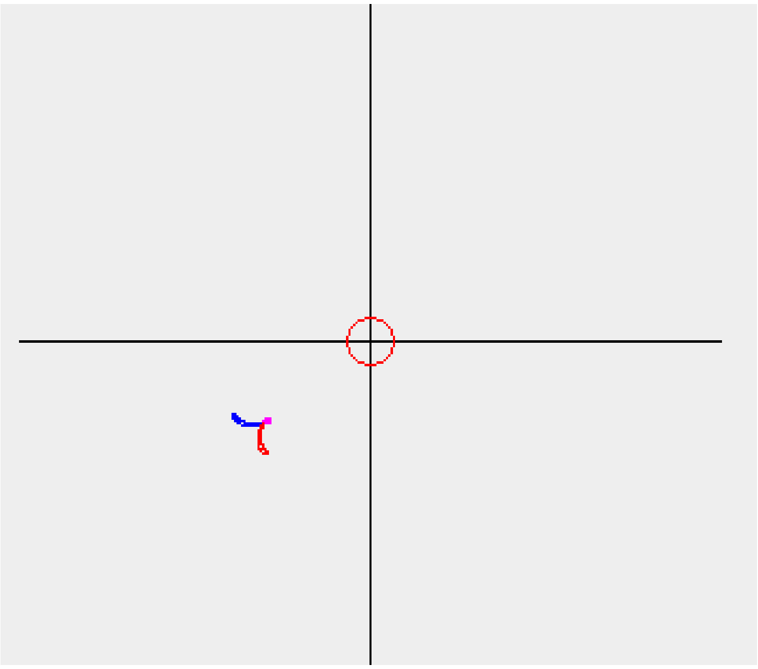


Figure 6.0.6: Relative coordinate system and reachable sets after the collision has been avoided.

VII. CONCLUSIONS

Systems that are modeled by non-linear continuous-time differential equations with uncertain parameters have proven to be exceptionally difficult to formally verify. The past few decades have produced a number of useful verification tools which can be applied to such systems but each is applicable to only a subset of possible verification scenarios. The LST is one such tool which is directly applicable to non-linear systems, however, it is limited to systems of relatively small continuous state space dimension. Other tools such as PHAVer and the SpaceEx invariant of the LGG support function algorithm are specifically designed for hybrid systems with linear dynamics and linear constraints but can accommodate hundreds of continuous states. The application of these linear reachability tools to non-linear models has been achieved by approximating non-linear systems as LHA. Unfortunately, the practical applicability and limitations of this approach are not yet well documented. The purpose of this thesis was to evaluate the performance and dimensionality limitations of PHAVer and the LGG support function algorithm when applied to a LHA approximation of a particular non-linear system. A collision avoidance scenario with autonomous differential drive robots was used as a case study to demonstrate that an over-approximated reachable set boundary can be generated and implemented as a run time safety assurance mechanism. The major contribution of this work is that it fills a void in the literature by providing a specific example that describes how both PHAVer and the LGG support function algorithm can be used to calculate reachable sets for a non-linear system. Additionally, this thesis provides insight into the applicability of both tools for the verification of non-linear systems.

One area that could be explored in future research is the problem of ensuring that a run-time assurance mechanism not only avoids unsafe conditions, but that it is also capable of eventually satisfying an objective or reaching some acceptable termination mode. In the differential drive robot line following scenario, this could be a proof that the robots are able to resume their goal of following the lines after the collision is avoided. Another area for future research could be to determine how a decision module can be designed to ensure safety when there are multiple safety criteria, either static or dynamic. Calculating optimal recovery controllers so as to minimize limitations on the experimental controller is yet another potential area for future work. Additional topics of future research and development in this field include: the development of a packaged tool for hybridizing any system, modifying the LGG support function algorithm so that numerical errors are not propagated in its computations, and creating a packaged tool which implements a dynamic hybridization algorithm similar to the one described in [4].

To conclude, the work done for this thesis has demonstrated the process of hybridizing a non-linear system model and the use of both PHAVer and the LGG support function algorithm for generating the FRS of the hybridized system. Moreover, an investigation of the limitations of these tools and a comparison of their performance has provided insight into the current capabilities of modern reachability tools. Although neither PHAVer nor the LGG support function algorithm could be directly applied to the scenario without reducing the complexity of the problem, the documentation of scenarios where the tools are not applicable is certainly also of value. The source code of all tools that were developed for this thesis can be downloaded at: `http://sourceforge.net/projects/provableruntime/files/`.

BIBLIOGRAPHY

[1] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *Software Engineering, IEEE Transactions on*, 22(3):181–201, 1996.

[2] Euene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid system, 2002.

[3] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control*, pages 20–31. Springer, 2000.

[4] Thao Dang, Colas Le Guernic, and Oded Maler. Computing reachable states for nonlinear biological models. In *Computational Methods in Systems Biology*, pages 126–141. Springer, 2009.

[5] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. *Hybrid Systems: Computation and Control*, pages 258–273, 2005.

[6] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

[7] Thomas A Henzinger. *The theory of hybrid automata.* Springer, 2000.

[8] Colas Le Guernic. Reachability analysis of hybrid systems with linear continuous dynamics. *Univerit Joseph Fourier*, 2009.

[9] John Lygeros, Karl Henrik Johansson, Slobodan N Simic, Jun Zhang, and Shankar Sastry. Continuity and invariance in hybrid automata. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 1, pages 340–345. IEEE, 2001.

[10] Oded Maler. Computing reachable sets : An introduction.

[11] Ian Mitchell, Alexandre Bayen, and Claire Tomlin. Validating a hamilton-jacobi approximation to hybrid system reachable sets. *Hybrid Systems: Computation and Control*, pages 418–432, 2001.

[12] Ian Mitchell and Claire Tomlin. Level set methods for computation in hybrid systems. *Hybrid Systems: Computation and Control*, pages 310–323, 2000.

[13] Ian M Mitchell. A toolbox of level set methods. *Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, http://www. cs. ubc. ca/˜ mitchell/ToolboxLS/toolboxLS. pdf, Tech. Rep. TR-2004-09*, 2004.

[14] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *Automatic Control, IEEE Transactions on*, 50(7):947–957, 2005.

[15] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). *Automated Reasoning*, pages 171–178, 2008.

[16] Jose G Rivera, Alejandro A Danylyszyn, Charles B Weinstock, Lui R Sha, and Michael J Gagliardi. An architectural description of the simplex architecture. Technical report, DTIC Document, 1996.

[17] D Seto, Bruce H Krogh, Lui Sha, and A Chutinan. Dynamic control system upgrade using the simplex architecture. *Control Systems, IEEE*, 18(4):72–80, 1998.

[18] Cory Snyder. An intuitive approach to formation control of differential drive robots, 2012.

APPENDIX A : PROOF OF THEOREM 1

Theorem 1. *Let the relative states of a pair of differential drive robots be defined by*

$$\begin{aligned} x_{21} &= x_2 - x_1; && (8.0.1) \\ y_{21} &= y_2 - y_1; && (8.0.2) \\ \theta_{21} &= \tan^{-1}\left(\frac{\dot{y}_{21}}{\dot{x}_{21}}\right), && (8.0.3) \end{aligned}$$

where (x_1, y_1) and (x_2, y_2) are the absolute coordinates of each robot. The dynamics of the states are then given by

$$\begin{aligned} \dot{x}_{21} &= v_{21}\cos\theta_{21}; && (8.0.4) \\ \dot{y}_{21} &= v_{21}\sin\theta_{21}; && (8.0.5) \\ \dot{\theta}_{21} &= \omega, && (8.0.6) \end{aligned}$$

where

$$v_{21} = -2v\sin\left(\frac{\theta_2 - \theta_1}{2}\right). \qquad (8.0.7)$$

Such that v is the linear velocity of both robots and ω is the angular velocity of both robots. The absolute headings of each robot are defined by θ_1 and θ_2.

Proof. Substituting v for v_{c1} and v_{c2} in equations (4.2.8) and (4.2.9) yields

$$\begin{aligned} \dot{x}_{21} &= v(\cos\theta_2 - \cos\theta_1); && (8.0.8) \\ \dot{y}_{21} &= v(\sin\theta_1 - \sin\theta_1). && (8.0.9) \end{aligned}$$

The dynamics in equations (8.0.8) and (8.0.9) are equivalent to those in equations (8.0.12) and (8.0.13). Substituting the identities from Lemmas (1 and 2 proves the equivalence.

Lemma 1.

$$\cos a - \cos b = -2\sin\left(\frac{a+b}{2}\right)\sin\left(\frac{a-b}{2}\right) \tag{8.0.10}$$

Lemma 2.

$$\sin a - \sin b = 2\cos\left(\frac{a+b}{2}\right)\sin\left(\frac{a-b}{2}\right) \tag{8.0.11}$$

$$\begin{aligned} \dot{x}_{21} &= -2v\sin\left(\frac{\theta_2+\theta_1}{2}\right)\sin\left(\frac{\theta_2-\theta_1}{2}\right) = v_{21}\sin\left(\frac{\theta_2+\theta_1}{2}\right); && (8.0.12) \\ \dot{y}_{21} &= 2v\cos\left(\frac{\theta_2+\theta_1}{2}\right)\sin\left(\frac{\theta_2-\theta_1}{2}\right) = -v_{21}\cos\left(\frac{\theta_2+\theta_1}{2}\right). && (8.0.13) \end{aligned}$$

Equations (8.0.12) and (8.0.13) can be used to simplify the expression for θ_{21} from equation (8.0.3). Lemma 3 is used in the simplification to yield the result in equation (8.0.15).

Lemma 3.

$$-\cot(a) = \tan\left(a - \frac{\pi}{2}\right) \tag{8.0.14}$$

$$\theta_{21} = \tan^{-1}\left(\frac{\dot{y}_{21}}{\dot{x}_{21}}\right) = \tan^{-1}\left(-\cot\left(\frac{\theta_2+\theta_1}{2}\right)\right) = \frac{\theta_2+\theta_1}{2} - \frac{\pi}{2} \tag{8.0.15}$$

The value of $\dot{\theta_{21}}$ is determined by taking the derivative of equation (8.0.15).

$$\dot{\theta}_{21} = \frac{\omega}{2} + \frac{\omega}{2} = \omega \tag{8.0.16}$$

Equations (8.0.12) and (8.0.13) can now be simplified by substituting the derived expression for θ_{21} from equation (8.0.15).

$$\begin{aligned} \dot{x}_{21} &= v_{21}\sin\left(\frac{\theta_2+\theta_1}{2}\right) = v_{21}\sin\left(\theta_{21} + \frac{\pi}{2}\right) = v_{21}\cos(\theta_{21}); && (8.0.17) \\ \dot{y}_{21} &= -v_{21}\cos\left(\frac{\theta_2+\theta_1}{2}\right) = -v_{21}\cos\left(\theta_{21} + \frac{\pi}{2}\right) = v_{21}\sin(\theta_{21}). && (8.0.18) \end{aligned}$$

Notice that the expressions derived in equations (8.0.17), (8.0.18), (8.0.16) are identical to those in equations (8.0.4), (8.0.5), and (8.0.6), respectively. This constitutes a proof of Theorem 1. □

APPENDIX B : PROOF OF THEOREM 2

Theorem 2. *Consider two possible sets of initial states for the system in equation (4.2.12),* $\mathcal{X}_0 = \{x_{21_0}, y_{21_0}, \theta_{1_0}, \theta_{2_0}\}$ *and* $\tilde{\mathcal{X}}_0 = \{\tilde{x}_{21_0}, \tilde{y}_{21_0}, \tilde{\theta}_{1_0}, \tilde{\theta}_{2_0}\}$*, such that* $\mathcal{X}_0$ *and* $\tilde{\mathcal{X}}_0$ *satisfy*

$$\tilde{\mathcal{X}}_0 = \begin{bmatrix} \tilde{x}_{21_0} \\ \tilde{y}_{21_0} \\ \tilde{\theta}_{1_0} \\ \tilde{\theta}_{2_0} \end{bmatrix} = \begin{bmatrix} x_{21_0} + x_t \\ y_{21_0} + y_t \\ \theta_{1_0} + \theta_t \\ \theta_{2_0} + \theta_t \end{bmatrix}, \tag{9.0.1}$$

where x_t, y_t, θ_t *are constant parameters. The following relationship exists between the trajectories of the system for these two possible sets of initial conditions:*

$$\tilde{\mathcal{X}}(t) = \begin{bmatrix} \tilde{x}_{21}(t) \\ \tilde{y}_{21}(t) \\ \tilde{\theta}_1(t) \\ \tilde{\theta}_2(t) \end{bmatrix} = \begin{bmatrix} (x_{21}(t) - x_{21_0}) \cos\theta_t - (y_{21}(t) - y_{21_0}) \sin\theta_t + x_{21_0} + x_t \\ (x_{21}(t) - x_{21_0}) \sin\theta_t + (y_{21}(t) - y_{21_0}) \cos\theta_t + y_{21_0} + y_t \\ \theta_1(t) + \theta_t \\ \theta_2(t) + \theta_t \end{bmatrix}. \tag{9.0.2}$$

Proof.

Lemma 4. *Integrating the state dynamics in equation (4.2.12) yields the following expressions* *for the states of the system as functions of time*

$$
\begin{aligned}
x_{21}(t) &= \frac{v}{\omega}\left(\sin(\omega t + \theta_{2_0}) - \sin(\theta 2_0) - \sin(\omega t + \theta_{1_0}) + \sin(\theta_{1_0})\right) + x_0; && (9.0.3)\\
y_{21}(t) &= \frac{v}{\omega}\left(\cos(\omega t + \theta_{2_0}) - \cos(\theta_{2_0}) - \cos(\omega t + \theta_{1_0}) + \cos(\theta_{1_0})\right) + y_0; && (9.0.4)\\
\theta_1(t) &= \omega t + \theta_{1_0}; && (9.0.5)\\
\theta_2(t) &= \omega t + \theta_{2_0}. && (9.0.6)
\end{aligned}
$$

The trajectories of the system for the initial conditions in the set $\tilde{\mathcal{X}}_0$ can be found by substituting for θ_{1_0} and θ_{2_0} in Lemma 4. The resulting trajectory equations are

$$
\begin{aligned}
\hat{x}_{21}(t) &= \frac{v}{\omega}(\sin(\omega t + \theta_{2_0} + \theta_t) - \sin(\theta_{2_0} + \theta_t)\\
&\quad - \sin(\omega t + \theta_{1_0} + \theta_t) + \sin(\theta_{1_0} + \theta_t)) + x_0 + x_t; && (9.0.7)\\
\hat{y}_{21}(t) &= \frac{v}{\omega}(\cos(\omega t + \theta_{2_0} + \theta_t) - \cos(\theta_{2_0} + \theta_t)\\
&\quad - \cos(\omega t + \theta_{1_0} + \theta_t) + \cos(\theta_{1_0} + \theta_t)) + y_0 + y_t; && (9.0.8)\\
\hat{\theta}_1(t) &= \omega t + \theta_{1_0} + \theta_t; && (9.0.9)\\
\hat{\theta}_2(t) &= \omega t + \theta_{2_0} + \theta_t. && (9.0.10)
\end{aligned}
$$

The identities from Lemma 5 can be used to simplify the dynamics in equations 9.0.7 to 9.0.10 to yield equation 9.0.2..

Lemma 5.

$$
\begin{aligned}
\sin u + v &= \sin u \cos v + \cos u \sin v; && (9.0.11)\\
\cos u + v &= \cos u cos v - \sin u \sin v. && (9.0.12)
\end{aligned}
$$

□