Formal Methods in Computer-Aided Design 2022

Automating Geometric Proofs of Collision Avoidance with Active Corners

Nishant Kheterpal*
Robotics Institute
University of Michigan
Ann Arbor, MI, USA
nskh@umich.edu

Elanor Tang
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
elanor@umich.edu

Jean-Baptiste Jeannin
Aerospace Engineering
University of Michigan
Ann Arbor, MI, USA
jeannin@umich.edu

***Abstract*—Avoiding collisions between obstacles and vehicles such as cars, robots, or aircraft is essential to the development of autonomy. To simplify the problem, many collision avoidance algorithms and proofs consider vehicles to be a point mass, though the actual vehicles are not points. In this paper, we consider a convex polygonal vehicle with nonzero area traveling along a 2-dimensional trajectory. We derive an easily-checkable, quantifier-free formula to check whether a given obstacle will collide with the vehicle moving on the planned trajectory. We apply our active corner method to two case studies of aircraft collision avoidance and benchmark its performance.**

I. Introduction

Preventing collisions with obstacles or foreign objects is crucial when developing autonomous capabilities for robots, cars, aircraft, and many other vehicles. As such, collision avoidance remains a major research theme of the autonomy, robotics, and formal methods communities. In particular, for safety-critical tasks such as vehicles interacting with humans or animals, it is imperative to provide *formal* proofs that the vehicle will not collide with agents in its environment.

In many papers studying trajectory planning or collision avoidance, e.g. [34], [3], [20], the vehicle is modeled as a point, and the volume — or surface area — occupied by the vehicle is ignored. In reality, land and air vehicles are not points but have a certain volume, and contact of any external object with any part of the vehicle would constitute a collision. In this paper, we present a novel, automated, and general technique to *transform* a planned trajectory of a vehicle with volume into explicit boundaries of the region in which an obstacle will not be at risk of a collision. This transformation provides an efficient, runtime-checkable test to determine whether a given obstacle will collide with a vehicle on the planned trajectory, even when the vehicle has volume.

Given a part of a trajectory $\mathcal{T}$, a vehicle occupying the volume $v(x_\mathcal{T}, y_\mathcal{T})$ when centered on position $(x_\mathcal{T}, y_\mathcal{T})$ along the trajectory, and a point-obstacle (x_O, y_O), the vehicle will not collide with the obstacle if and only if:

$$\forall (x_\mathcal{T}, y_\mathcal{T}) \in \mathcal{T}, (x_O, y_O) \notin v(x_\mathcal{T}, y_\mathcal{T}) \qquad (1)$$

In the rest of the paper, we will call this formulation the *implicit* formulation of collision avoidance. This implicit formulation is a correct definition, but it has one major drawback: because of the universal quantifier on $(x_\mathcal{T}, y_\mathcal{T})$, it is not easy to check systematically or at runtime whether an obstacle is indeed at risk of a collision. Ideally, we would want to obtain a quantifier-free, easily checkable formula that is equivalent to (1); in the rest of this paper we will call that formula, which represents a region in the plane, the *explicit* formulation. In theory, one could use quantifier elimination, but for trajectories containing more than a few symbolic parameters, the algorithm does not finish in a reasonable time due to its doubly-exponential time complexity in the number of variables [14].

This issue arose before, notably in the verification of the Next-Generation Aircraft Collision Avoidance System ACAS X [22], [23]. In that work, the formal proof of correctness was divided into: (i) establishing the trajectory of the aircraft from its equations of motion, leading to a formula of the form of (1); and (ii) establishing an equivalent quantifier-free formula that can be checked efficiently at runtime. Both tasks required a proof in the KeYmaera X theorem prover, with significant manual effort [35]. A similar approach was used in the verification of collision avoidance for ACAS Xu, the unmanned version of ACAS X, with horizontal maneuvers [1]. The object of this paper is to automate and generalize task (ii) of this process.

In order to automate task (ii), we propose a different approach based on geometric intuition. Let us examine an example of a rectangular vehicle performing a simple maneuver (Figure 1). The central idea of the method presented in this paper is that the boundaries of the explicit formulation are either *trajectories of a corner* of the vehicle or *sides of the vehicle* at a few particular points.

The corners to consider at every point depend on the slope of the trajectory: for a rectangular vehicle, the boundaries follow the top-right and bottom-left corners when the vehicle's velocity is directed "northwest" (towards the top left) or southeast; and, symmetrically, the boundaries follow the top-left and bottom-right corners when the vehicle's velocity is directed towards the northeast or southwest of the plane. We call these corners *active corners*. But this is not enough: at points where the trajectory switches from following one set of corners to another, the boundary may follow a side of the vehicle at that point, e.g., its bottom boundary at the lowest point of the trajectory on Figure 1. We call these points

https://doi.org/10.34727/2021/isbn.978-3-85448-053-2_43

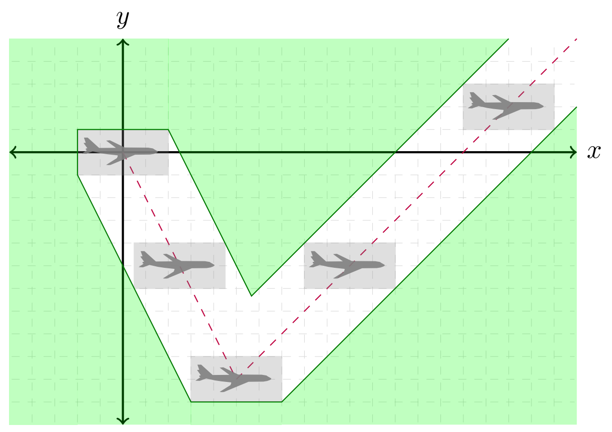


Fig. 1: Safe region for a rectangle with $w = 2, h = 1$ and its center following Equation (2). The trajectory is dashed in purple, safe region shaded in green, and unsafe region is unshaded.

transition points. By capturing the motion of these boundaries — both corners depending on the slope or the sides of the vehicle at certain points — we can construct a quantifier-free formula equivalent to (1), corresponding to its equivalent explicit formulation. Our approach is fully symbolic with no approximation.

In this paper, we formalize and generalize our approach to different trajectories and polygons and show how to find active corners and transitions symbolically and how to form a quantifier free explicit formulation equivalent to the input implicit formulation. We carefully prove that our transformation is both sound (any obstacle shown safe using the explicit formulation is safe using the implicit formulation) and complete (no obstacles that are actually safe appear unsafe using the explicit formulation). Finally, we detail a fully symbolic Python implementation of our work and present an evaluation of its performance on two applications from previous papers (where we fully automate results that had required significant manual proof effort) and a third non-polynomial example.

II. Overview

This section provides an overview of our approach, by walking through a simple example constructing a geometric safe region used to verify obstacle avoidance for aircraft. At present, our method applies only to two-dimensional planar motion due to the increased complexity of three-dimensional motion when analyzing trajectories and polyhedra. The example uses linear motion and a rectangle, though the method generalizes to other planar motion and convex polygons, as detailed in Section IV.

A. Trajectory

Consider the planar side-view of an airplane flying, initially descending at constant velocity and then ascending with constant velocity. For simplicity, assume the aircraft has infinite acceleration. In this example, we represent the bounds of the aircraft as an axis-aligned rectangle with width $2w$ and height $2h$ that moves in the (x, y) plane. The airplane begins at the origin and moves in the plane with piecewise trajectory $\mathcal{T}$:

$$\mathcal{T} = \begin{cases} y = -2x & x \in [0, 5] \\ y = x - 15 & x \in [5, \infty) \end{cases} \tag{2}$$

B. Implicit Formulation

Suppose the rectangle translates with its center moving along this piecewise trajectory. Additionally, assume there is a point obstacle at (x_O, y_O) to be avoided; that is, the rectangle never intersects the obstacle. Then we can state an quantified (or *implicit*) formulation of obstacle avoidance:

$$\forall (x_{\mathcal{T}}, y_{\mathcal{T}}) \in \mathcal{T}, (|x_O - x_{\mathcal{T}}| > w \vee |y_O - y_{\mathcal{T}}| > h) \tag{3}$$

The implicit formulation of the safe region (3) straightforwardly represents a safety property of obstacle avoidance — if an object moving with its center fixed along the nominal trajectory $\mathcal{T}$ is far enough away (either width w in the x-axis or height h in the y-axis) from a point obstacle at (x_O, y_O), then the object is safe. Here we use *safe region* to mean the set of all obstacle locations for which collision is avoided.

C. Explicit Formulation

The need for a quantifier-free equivalent of Equation (3) motivates an *explicit formulation* of the safe region for the obstacle. The goal of this work is to automate the generation of such a formulation. We compute the reachable set of the object as it moves along a trajectory in order to compute the complement of the safe region: the *unsafe region*. We can express the *unsafe region* (the set of all locations for which an obstacle will collide with the object as it moves along a given trajectory) directly as a union of regions in the plane, each defined (in this case) by an intersection of linear inequalities (Equation (4)) bounding the region, plotted in Figure 1. The *safe region* is simply the negation of (4).

$$\begin{aligned} &\Big((x_O \geq -w) \ \wedge \ (y_O \leq h) \ \wedge (y_O \geq -2x_O - 2w - h) \\ &\wedge \ (x_O \leq 5 + w) \ \wedge \ (y_O \leq -2x_O + 2w + h) \\ &\wedge \ (y_O \geq -10 - h)\Big) \bigvee \Big((x_O \geq 5 - w) \\ &\wedge \ (y_O \geq -10 - h) \ \wedge (y_O \leq x_O + w + h - 15) \\ &\wedge \ (y_O \geq x_O - w - h - 15)\Big) \end{aligned} \tag{4}$$

Note the first disjunction in Equation (4) corresponds to the motion of the aircraft on the left side of Figure 1 as it descends; the second corresponds to the right side as the aircraft ascends.

III. Algorithm

A. Preliminaries

In this work, we define the *safe region* as the set of obstacle locations where, given a polygon's trajectory, a collision will not occur. Correspondingly, the *unsafe region* will be unsafe if an obstacle invades its area. As such, the *unsafe region* corresponds to the reachable set of the polygon as it moves along a trajectory. We can define a quantified representation of the safe region: the *implicit formulation* from (1). We also use the term *explicit formulation* in this work; we use that to mean an equivalent to the *implicit formulation*, but without quantifiers like $\forall$ and $\exists$. Our method primarily applies to convex polygons. In this paper, we discuss polygons with central symmetry for ease of exposition, though the method straightforwardly extends to irregular and asymmetric convex polygons, and can be extended to concave polygons (Section III-F).

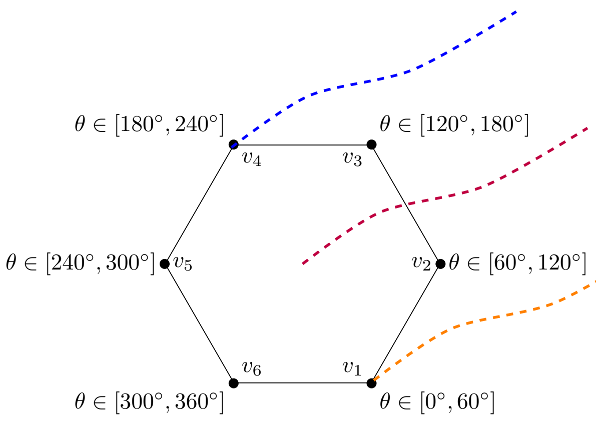
We consider two-dimensional planar trajectories defined piecewise, with each piece a function $y = f(x)$ or $x = f(y)$ and f a C^1 function (differentiable and having a continuous derivative). Trajectories must have a *finite* number of these C^1 pieces. The pieces themselves need not be continuous, though the applications we study do include continuous piecewise trajectories. The subdomains for the piecewise trajectory must be non-overlapping and exhaustive, meaning their union should cover the entire domain of the trajectory. Polygons move along the trajectory without rotating. Since the polygons translate along the trajectory, there is a constant vector offset $\begin{bmatrix}\Delta x_i \\ \Delta y_i\end{bmatrix}$ from the center to the i-th vertex, and there are n vertices for an n-sided polygon. Thus, the trajectory for vertex i is $y - \Delta y_i = f(x - \Delta x_i)$ or $x - \Delta x_i = f(y - \Delta y_i)$. We consider the trajectories of all vertices of the polygon in an attempt to bound its motion and compute the reachable set of the object as it moves along the trajectory.

B. Active Corners

Throughout this section we consider a trajectory of the form $y = f(x)$; the case for $x = f(y)$ is symmetric. The boundaries of the safe region are (for centrally symmetric polygons) formed by the trajectories of a pair of opposite corners of the vehicle (Figure 2) — we call this pair of corners *active corners*. For asymmetric polygons, the corners may not directly oppose each other.

We choose the active corners to represent the outermost extent of the object along the trajectory; as such, their motion bounds the safe region. Which corners are active depends on the slope of the trajectory (which can be computed from the derivative of f) and the shape of the convex object. A corner v_i is active when the slope θ of the trajectory is between the slopes of the sides adjacent to v_i; when a corner is active, its opposite corner is also active based on the symmetry of the polygon. More precisely, if we number the corners v_1 through v_n counter-clockwise (with $v_{n+1} = v_0$ and $v_{-1} = v_n$), corner v_i is active if and only if the slope θ of the trajectory is in the angle interval $[\angle\overrightarrow{v_{i-1}v_i}, \angle\overrightarrow{v_i v_{i+1}}]$, or symmetrically in the angle interval $[\angle\overrightarrow{v_{i+1}v_i}, \angle\overrightarrow{v_i v_{i-1}}]$. Because the direction of the trajectory is inconsequential for our purpose, θ is modulo 180°.

For example, on the hexagon in Figure 2, v_1 and v_4 are active when $\theta \in [0°, 60°] \cup [180°, 240°]$; v_2 and v_5 are active when $\theta \in [60°, 120°] \cup [240°, 300°]$; and v_3 and v_6 are active when $\theta \in [120°, 180°] \cup [300°, 360°]$.



$\theta \in [180°, 240°]$
$\theta \in [120°, 180°]$
v_4
v_3
$\theta \in [240°, 300°]$
v_5
v_2
$\theta \in [60°, 120°]$
v_6
v_1
$\theta \in [300°, 360°]$
$\theta \in [0°, 60°]$

Fig. 2: A hexagon, the angles of its sides, and shifted active corner-trajectories

At transition points (where the active corners change), the boundary of the safe region may not follow an active corner. We detail what happens then in Section III-C. Note that when a linear trajectory is parallel to a side of the polygon (e.g. $\theta = 60°$ for the hexagon in Figure 2), two adjacent corners may both be active and either can be classified as such. For such trajectories, the active corners technically do not change for the length of the linear path, so there would be no transition points as long as the trajectory parallels a polygon edge.

Given an obstacle at point (x_O, y_O), we can check if it is inside the unsafe region (or reachable set) in a computationally efficient fashion. If an obstacle lies outside the unsafe region, it would be either above both corner-trajectories or below both corner-trajectories for whichever corners are active. We can express the location of the obstacle with respect to a corner-trajectory in a single equation by considering the value of $y_O - f(x_O - \Delta x_i) - \Delta y_i$ for active corner (vertex) v_i. This term will be positive for both vertices v_i, v_j if the obstacle is above both corner-trajectories, and similarly negative if the obstacle is below both. Therefore, any point (x_O, y_O) in the safe region has a positive value for the product of the two expressions above, and any point in the unsafe region has a negative or zero value for this product. This yields the following test to check if an object lies in (part of) the *unsafe region*:

$$(y_O - f(x_O - \Delta x_i) - \Delta y_i) \cdot (y_O - f(x_O - \Delta x_j) - \Delta y_j) \le 0 \tag{5}$$

where $\Delta x_i, \Delta y_i, \Delta x_j, \Delta y_j$ are the (constant) offsets from the center of the polygon to the active corners (vertices) v_i, v_j.

When implementing this algorithm, the trajectories of all other vertices lie within the trajectories of the active corners, so to check whether an obstacle lies in the portion of the unsafe region defined by the active corners, it suffices to check over all pairs of vertices (v_i, v_j) with $i, j \in \{1, 2, ...n\}$. This check can be made more efficient by considering only all possible pairs of active corners based on the polygon shape and discarding, say, pairs of adjacent vertices. If the test in (5) indicates that an object lies within the unsafe region, trajectory $y = f(x)$ or $x = f(y)$ is clearly unsafe.

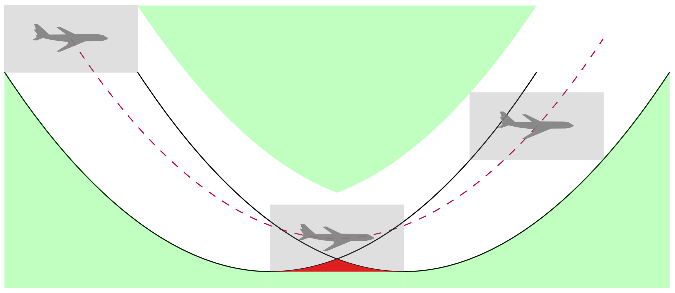


Fig. 3: A rectangular airplane moving along a planar trajectory. At the transition point at the parabola's vertex, the "notch" is visible and shaded in red; part of the object lies outside the corner-trajectories at this point.

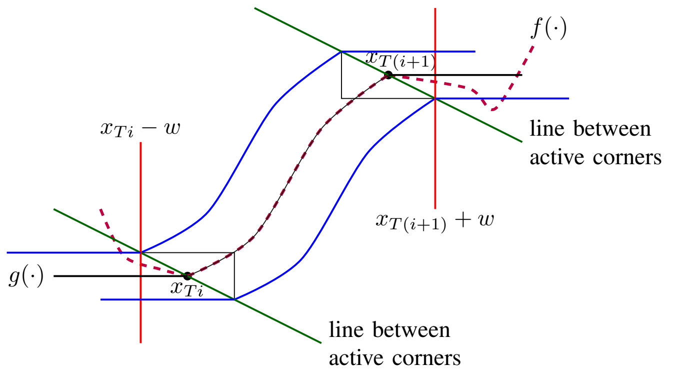


Fig. 4: For piecewise functions, between transition points and/or piecewise boundaries, this figure shows the difference between $f(\cdot)$ and $g(\cdot)$ and the two additional checks on (x_O, y_O) described in Section III-D.

C. Notches at Transition Points Between Active Corners

It turns out that using only active corners would yield an underestimate of the reachable set, which would be unsound for verifying safety. Figure 3 illustrates why: the white area bounded by the trajectories of the corners does not contain the red "notch," even though a collision would occur with an obstacle in this notch. Therefore, if the test in (5) yields a value > 0, the trajectory is not necessarily safe; we **additionally** check safety at all *transition points* (x_T, y_T) to see whether the obstacle at (x_O, y_O) lies within the polygon centered at (x_T, y_T). Recall that transition points are defined as points on the trajectory where the active corners switch. In the full test for safety (Equation 7), this check is represented as **in_polygon()** and can leverage one of many point-in-polygon implementations, which generally run in linear time on the order of number of vertices. As the slope of the function may change at the boundary between piecewise subfunctions, we also add a notch check at each subdomain boundary.

D. Handling Piecewise Functions

In order to account for piecewise functions, we modify our method in two ways to avoid using a subfunction outside the subdomain over which it holds. The first is a modification to hold subfunctions constant outside of the subdomain over which they're defined; and the second is an additional

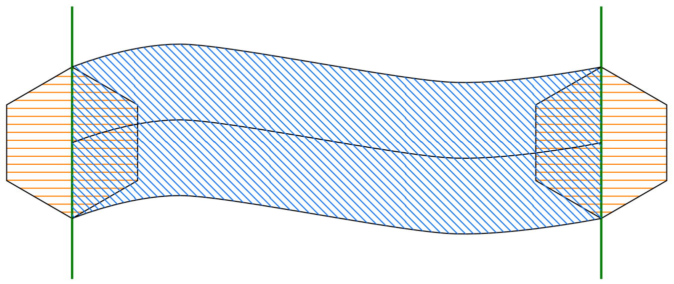


Fig. 5: Terms in Equation (7), illustrated. Green terms restrict test to relevant piecewise subdomains, blue, diagonally-hatched terms check if obstacles are between active corner pairs, and orange, horizontally-hatched terms check if obstacles are in the notch at transition points and subdomain bounds.

boolean clause to the safety test in (5) so it only applies over a valid subdomain. In this case, the subdomain is an interval $[x_{Tk}, x_{T(k+1)}]$, where $x_{Tk}, x_{T(k+1)}$ may be piecewise boundaries *or* transition points. Because of this, there may be many subdomains for a single piecewise case in which there happen to be many transition points.

First, we define a function $g(x)$ (or $g(y)$ symmetrically) that holds constant the value of each subfunction outside of its subdomain $[x_{Tk}, x_{T(k+1)}]$. The function $g(\cdot)$ is used in place of $f(\cdot)$ in (5) above. Let $y_{Tk} = f(x_{Tk})$.

$$g(x) = \begin{cases} y_{Tk} & \text{if } x \le x_{Tk} \\ f(x) & \text{if } x_{Tk} < x < x_{T(k+1)} \\ y_{T(k+1)} & \text{if } x \ge x_{T(k+1)} \end{cases} \qquad (6)$$

Additionally, we add a clause to ensure the modified subfunction $g(\cdot)$ is only used over the correct subdomain. First, we check $x_{Tk} - w < x_O < x_{T(k+1)} + w$, where w is the half-width of the object. We also construct a line between the two active corners of the object in each of the two piecewise boundary locations (x_{Tk}, y_{Tk}) and $(x_{T(k+1)}, y_{T(k+1)})$ and check (x_O, y_O) is between the two lines. This way, we ensure the test for being unsafe holds only for the region on which each subfunction applies. Figure 4 illustrates the function $g(\cdot)$ and the additional subdomain-related clauses.

E. Generic Explicit Formulation

This leads to a generic quantifier-free explicit formulation to test whether an obstacle is in the safe region, where $\{(x_T, y_T)_k\}$ represents the set of all transition *and* boundary points on the trajectory between piecewise subdomains. $g(\cdot)$ is used as defined previously in Section III-D. As defined previously, $\Delta x_i, \Delta y_i, \Delta x_j, \Delta y_j$ are the (constant) offsets from the center of the polygon to active corners v_i and v_j. Our algorithm generates a test for whether an obstacle is unsafe (if a collision will occur); negating the boolean formula or its result allows testing whether an obstacle lies in the safe region.

Equation (7) is color-coded in correspondence with Figure 5. The first, third, and fourth lines ensure the test applies only over the correct piecewise domain and are in green; the

$$\textbf{unsafe?} = \bigvee_{\{(g_k, x_{Tk}, x_{T(k+1)})\}} \Bigg(x_{Tk} - w < x_O < x_{T(k+1)} + w \ \wedge \ (x_O, y_O) \text{ between}$$
$$\Big(\textbf{Line}\big((x_{Tk} + \Delta x_i, y_{Tk} + \Delta y_i), (x_{Tk} + \Delta x_j, y_{Tk} + \Delta y_j)\big), \textbf{Line}\big((x_{T(k+1)} + \Delta x_i, y_{T(k+1)} + \Delta y_i), (x_{T(k+1)} + \Delta x_j, y_{Tj} + \Delta y_j)\big)\Big) \wedge$$
$$\bigvee_{\{(v_i, v_j)\}} \big(y_O - g(x_O + \Delta x_i) - \Delta y_i\big)\big(y_O - g(x_O + \Delta x_j) - \Delta y_j\big) \leq 0 \Bigg) \vee \Big(\bigvee_{\{(x_T, y_T)_i\}} \textbf{in\_polygon}(x_{Ti}, y_{Ti}, x_O, y_O) \Big) \tag{7}$$

second line checks the obstacle is between the active corners and is in blue; the fifth line is in orange and checks for the notch at transition points and piecewise subdomain boundaries.

F. Extensions

Thus far, we have considered point-mass obstacles, but the reasoning extends to obstacles that have the same properties as the object (convex and centrally symmetric). This is achieved through a reduction where the shape of the obstacle is incorporated into the shape of the object. For example, in the simple case where both are horizontal rectangles, with the object of height $2h$ and width $2w$, and the obstacle of height $2h_O$ and width $2w_O$, the object and obstacle intersect if and only if the center of the obstacle is contained in a virtual object with the same center as the initial object, but of height $2(h + h_O)$ and width $2(w + w_O)$. We have thus reduced the problem of collision avoidance with a convex object to a problem of avoidance with a point-mass object. A similar reasoning — albeit a little more complicated — can be applied to any convex, centrally symmetric obstacle.

For ease of presentation, and because they appear in most practical applications, we have focused on objects that are convex and centrally symmetric. We can extend the reasoning to non-centrally symmetric objects: the only difference in that case is that pairs of active corners do not change together, but rather one active corner may change on one side, and another active corner may change on the other side later. Pairs of active corners are thus not opposite corners of the object anymore. The convexity of the object (and obstacles) is essential for active corners; however, we can extend our reasoning to non-convex, polygonal objects by seeing them as unions of convex sub-objects and ensuring collision avoidance with each sub-object. Finally, due to its reliance on corners, our method cannot handle circles or ellipses, but they can be approximated by polygons.

IV. Proof of Equivalence

We prove the equivalence of the safe regions represented by 1) the implicit formulation and 2) the explicit output of our active-corner method for trajectories of form $y = f(x)$. The proof of soundness follows; the proof of completeness is in our full paper on arXiv. The proof structure considers segments of the trajectory in which no active corner switch occurs; that is, where the angle of the tangent to the trajectory is bounded. In these segments, the bounds on the trajectory tangent angle allow us to bound the location of points in the interior of the polygon and show they lie between the two active corners. The two endpoints of a segment represent locations at which 1) the notch exists or 2) the trajectory switches to a new piecewise subfunction. In our method, these cases are handled by testing if obstacle (x_O, y_O) is inside the polygon at various transition points $\{(x_T, y_T)\}_i$.

A. Proof Preliminaries

Consider a segment of the motion along trajectory $y = f(x)$ or $x = f(y)$ in which no active corner switches or piecewise trajectory segment switches occur. We can arbitrarily rotate this segment of motion and the proof will hold, since the object translates along the trajectory without rotation. Assume, then, that a rotation is made by an angle θ such that the active corners are oriented along a vertical line. This rotation is an invertible transformation, so the logic of this proof holds through the entire trajectory. Because of this coordinate rotation, we consider only trajectories $y = f(x)$ for the proof; any trajectory $x = f(y)$ can be rotated into the form $y = f(x)$ invertibly, so our results hold for these forms as well. Let v_i, v_j denote the active corners for this segment, with corresponding offsets $\Delta x_i, \Delta y_i, \Delta x_j, \Delta y_j$ in the rotated coordinate system.

Since no active corner switch occurs, then we know the slope of function $y = f(x)$ is limited by the shape of the polygon itself — let these bounds be $\pm m$, with m representing the slope of the relevant sides of the polygon. Slopes of f beyond this range cannot occur over the trajectory segment in consideration, due to our assumption that the no active corner switches occur. Because the polygon is symmetric, the lower bound on slope is the negative of the upper bound (illustrated further in Figure 7). This proof is presented considering regular, symmetric polygons for simplicity, but extends to asymmetric polygons as discussed in our full paper on arXiv. To prove the soundness of our method, we must prove that all obstacles shown safe using our method (safe$_{\text{expl}}$) are also safe using the input implicit formulation (safe$_{\text{impl}}$). To prove safe$_{\text{expl}} \implies$ safe$_{\text{impl}}$, we prove the contrapositive unsafe$_{\text{impl}} \implies$ unsafe$_{\text{expl}}$.

Specifically, unsafe$_{\text{impl}}$ means that an obstacle at (x_O, y_O) is inside a polygon centered at some coordinates (x, y); unsafe$_{\text{expl}}$ means that the below holds from (5):

$$(y_O - f(x_O - \Delta x_i) - \Delta y_i) \cdot (y_O - f(x_O - \Delta x_j) - \Delta y_j) \leq 0$$

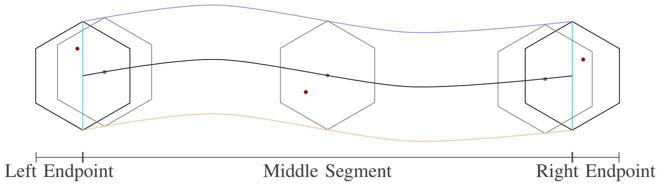


Fig. 6: Sections of proof

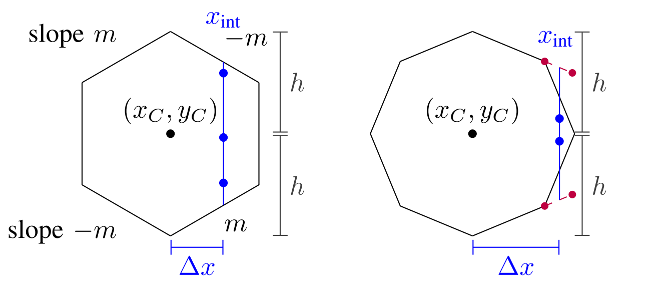


Fig. 7: Figure of the slope of the sides of a regular hexagon and octagon.

This proof has three sections: one holds for the majority of the trajectory segment, one for the beginning of the segment, and one of the end of the segment. The beginning- and end-of-segment proofs follow the form of the main proof but consider polygons fixed at the trajectory segment endpoints. They are included in our full paper available on arXiv.

B. Middle Segment Proof

Consider a (symmetric) polygon P, with half-height h and half-width w, centered at (x_C, y_C), where $y_C = f(x_C)$. Let $(x_{\text{int}}, y_{\text{int}})$ be a point inside or on the edges of P. We prove that interior point $(x_{\text{int}}, y_{\text{int}})$ lies between the active corners of an identical polygon $\overline{P}$ located at $(x_{\text{int}}, f(x_{\text{int}}))$. We do this by bounding three terms: 1) $f(x_{\text{int}})$, 2) y_{int}, and 3) the active corners of $\overline{P}$ to prove that y_{int} lies between them.

First, we bound $f(x_{\text{int}})$ (the center of $\overline{P}$). Let $x_{\text{int}} = x_C + \Delta x$, for $\Delta x \in [-w, w]$. Let $f(x_{\text{int}}) = f(x_C + \Delta x) = y_C + \Delta y$, for some Δy which we will bound. The slope of the trajectory $\frac{dy}{dx}$ is bounded by $(-m, m)$, because this proof considers a segment of motion with no changes in active corner. Hence Δy is bounded proportionally to Δx, with $\Delta y \in (-|m\Delta x|, |m\Delta x|)$. Therefore, $f(x_{\text{int}}) \in (y_C - |m\Delta x|, y_C + |m\Delta x|)$. Our proof proceeds assuming $\Delta x \neq 0$, since if $\Delta x = 0, x_{\text{int}}$ will lie on the vertical centerline of P. In that case, it is trivial to show x_{int} lies between the active corners.

Recall $x_{\text{int}} = x_C + \Delta x$, for $\Delta x \in [-w, w]$. Let $y_{\text{int}} = y_C + \Delta y_{\text{int}}$, for some Δy_{int} which we will bound. Given that the slopes of the sides on the top and bottom of P are $\pm m$, we assert that any $(x_{\text{int}}, y_{\text{int}})$ with $x_{\text{int}} = x_C + \Delta x$ has a corresponding $\Delta y_{\text{int}} \in \big[-h + |m\Delta x|, h - |m\Delta x|\big]$. This is illustrated in Figure 7 with a hexagon, but it generalizes to any symmetric convex polygon. Given this, we can bound the x and y interior coordinates as below:

$$(x_{\text{int}}, y_{\text{int}}) = \begin{bmatrix} x_C + \Delta x \\ [y_C - h + |m\Delta x|, y_C + h - |m\Delta x|] \end{bmatrix} \tag{8}$$

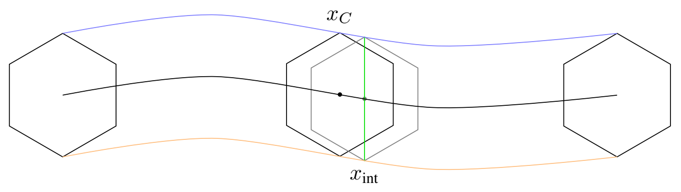


Fig. 8: Shifted polygon illustration

Finally, we show interior point y-coordinate y_{int} lies within the active corners of $\overline{P}$. Because we consider a rotated coordinate frame such that the active corners are oriented along the vertical axis, the top and bottom active corners are located at $(\overline{x_{\text{top}}}, \overline{y_{\text{top}}}) = (x_{\text{int}}, f(x_{\text{int}}) + h)$ and $(\overline{x_{\text{bot}}}, \overline{y_{\text{bot}}}) = (x_{\text{int}}, f(x_{\text{int}}) - h)$, respectively. The bounds on $\overline{y_{\text{top}}}$ and $\overline{y_{\text{bot}}}$ are given by the following:

$$\begin{aligned} y_C - |m\Delta x| + h < \overline{y_{\text{top}}} < y_C + |m\Delta x| + h \\ y_C - |m\Delta x| - h < \overline{y_{\text{bot}}} < y_C + |m\Delta x| - h \end{aligned} \tag{9}$$

Then $y_{\text{int}} \le y_C - |m\Delta x| + h < \overline{y_{\text{top}}}$ and $y_{\text{int}} \ge y_C + |m\Delta x| - h > \overline{y_{\text{bot}}}$.

The top active corner trajectory is given by $f_{\text{top}}(x) = f(x) + h$ and the bottom active corner trajectory is given by $f_{\text{bot}}(x) = f(x) - h$. By definition, $f_{\text{top}}(x_{\text{int}}) = \overline{y_{\text{top}}}$ and $f_{\text{bot}}(x_{\text{int}}) = \overline{y_{\text{bot}}}$, or equivalently, $\overline{y_{\text{top}}} - f_{\text{top}}(x_{\text{int}}) = 0$ and $\overline{y_{\text{bot}}} - f_{\text{bot}}(x_{\text{int}}) = 0$. Since $y_{\text{int}} < \overline{y_{\text{top}}}$ and $y_{\text{int}} > \overline{y_{\text{bot}}}$,

$$y_{\text{int}} - f_{\text{top}}(x_{\text{int}}) < 0 \qquad y_{\text{int}} - f_{\text{bot}}(x_{\text{int}}) > 0 \tag{10}$$

By multiplying the equations in (10), we get

$$(y_{\text{int}} - f_{\text{top}}(x_{\text{int}})) \cdot (y_{\text{int}} - f_{\text{bot}}(x_{\text{int}})) < 0 \tag{11}$$

This is an equivalent test for whether an object lies in the unsafe region from (5). Therefore we have shown that for all (x_C, y_C) points satisfying $y_C = f(x_C)$, all points $(x_{\text{int}}, y_{\text{int}})$ inside and on the boundary of a polygon centered at (x_C, y_C) also have $(f_{\text{top}}(x_{\text{int}}) - y_{\text{int}}) \cdot (f_{\text{bot}}(x_{\text{int}}) - y_{\text{int}}) \le 0$. These are exactly the definitions of unsafe$_{\text{impl}}$ and unsafe$_{\text{expl}}$ from IV-A earlier. Therefore, we have shown unsafe$_{\text{impl}}$ $\Longrightarrow$ unsafe$_{\text{expl}}$ and the contrapositive safe$_{\text{expl}}$ $\Longrightarrow$ safe$_{\text{impl}}$ holds as well.

V. Implementation

We have implemented our automated method in Python using SymPy, a symbolic math library [30]. The code implementing our algorithm and the applications in Section VI is available on GitHub at https://github.com/nskh/automatic-safety-proofs.

Given a fully symbolic trajectory and object, we first identify the angles corresponding to sides of the object. Then, following Section III-B, we identify points on the trajectory corresponding to the angles θ_i of each side of the object. To avoid discontinuities in the arctan function, the implementation solves a reformulation: $\frac{\partial f}{\partial x}\sin(\theta_i) = \frac{\partial f}{\partial y}\cos(\theta_i)$. Solving this equation may yield either y in terms of x or the reverse. In this case, we substitute the implicit solution for x or y into the trajectory equation, eliminate the remaining variable, and identify transition points. Given transition points,

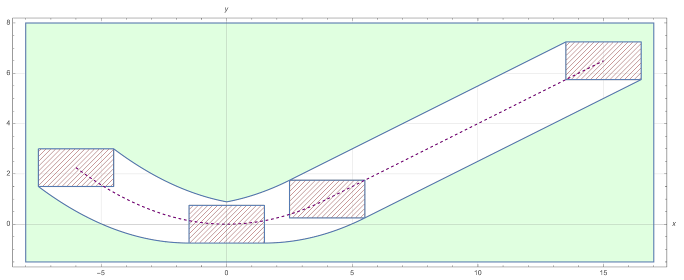


Fig. 9: Safe region for an instance of [22]. The notches are the red-hatched rectangles and the trajectory is dashed in purple.

we can implement the test from (7). SymPy includes a "point-in-polygon" method, which we use to identify if an obstacle (x_O, y_O) lies in the "notch" at any transition point. The output explicit formulation can be expressed either in LaTeX or in Mathematica format; output in Mathematica also supports generating code to copy-paste directly and plot the safe region using Mathematica's `RegionPlot[]` functionality. Examples can be found in Figures 9 and 10.

In order to implement our method in a fully symbolic fashion, we must account for the potential values of symbols when instantiated. We can leverage SymPy's built-in "assumptions" to specify that certain symbols representing, say, trajectory parameters or object dimensions are real, positive, and/or nonzero, but these assumptions may not suffice to construct a fully symbolic safe region. In that case, our fully symbolic implementation computes a number of potential valid safe regions. As detailed in Section III-D, we construct the explicit formulation using many clauses defined on intervals between transition points and/or piecewise boundaries. In the symbolic case, the order of these terms may differ, depending on, say, the sign of a variable in the trajectory. Additionally, symbolic piecewise cases for, say, $x < b$ may mean that certain transition points do not occur at all if b lies in some range. Correspondingly, our fully symbolic implementation computes all valid orderings of piecewise boundaries and transition points; it additionally considers all valid combinations of transition points to account for "notches" that may not exist when piecewise bounds and/or trajectory parameters are instantiated. In order to check if orderings are valid, we attempt to sort using the SymPy assumptions: if we know b is positive, no returned ordering will place b before a transition point at 0, for example. Additionally, we enforce that adjacent points in the ordering "come from" the same functions: we will not return an ordering where a transition point from piecewise subfunction f_1 lies between the piecewise boundaries for f_2. Doing so ensures that we often generate relatively few (~ 10) potential orderings despite considering many combinations, though examples with intractably many orderings do exist.

VI. Applications and Evaluation

A. Verification of vertical maneuvers in ACAS X

A collision avoidance system intended to prevent near mid-air collisions, ACAS X, was verified in [22]. The KeYmaera X proof presented in [22] required a significant amount of human interaction (on the order of hundreds of hours), while the method presented in this paper generates an explicit formulation from the trajectory fully automatically. ACAS X prevents collisions between aircraft by issuing advisories (control commands) to one aircraft, the *ownship*. The bounds of aircraft in this work are shaped like hockey pucks (cylinders wider than they are tall) of a radius r_p and half-height h_p. From a side perspective of an encounter between aircraft, the bounds are rectangular. In [22], verification was performed in a side-view perspective, assuming two aircraft approach each other in a vertically-oriented planar slice of three dimensions. A careful choice of reference frame can reduce a three-dimensional encounter between aircraft into a two-dimensional system, by modeling the encounter as a 1-dimensional vertical encounter and the distance of a horizontal encounter [22, Section 6].

To simplify calculations, [22] used the relative horizontal speed r_v of the two aircraft and assumed it constant; the vertical velocity of the oncoming aircraft $\dot{h}$ is also assumed constant. Advisories consist of climb and descent speed advisories, yielding ownship trajectories that are piecewise combinations of parabolas and straight lines. One example trajectory is below in (12), which assumes the advisory issued is for the ownship to climb at a rate $\dot{h}_f$ greater than its current vertical velocity $\dot{h}_0$. (r_t, h_t) are the (x, y) coordinates for trajectory $\mathcal{T}$ in this example, and a_r is the acceleration.

$$(r_t, h_t) = \begin{cases} \left(r_v t,\, \frac{a_r}{2}t^2 + \dot{h}_0 t\right) & \text{for } 0 \le t < \frac{\dot{h}_f - \dot{h}_0}{a_r} \\ \left(r_v t,\, \dot{h}_f t - \frac{(\dot{h}_f - \dot{h}_0)^2}{2a_r}\right) & \text{for } \frac{\dot{h}_f - \dot{h}_0}{a_r} \le t \end{cases} \tag{12}$$

The implicit formulation of the safe region is below, for an oncoming aircraft at relative coordinates r_O, h_O.

$$\forall t. \forall r_t. \forall h_t. \big((r_t, h_t) \in \mathcal{T} \implies |r_O - r_t| > r_p \vee |h_O - h_t| > h_p\big) \tag{13}$$

In [22], the authors eliminate the parametrization over t, which yields an initial parabolic section and then straight-line motion after. We use this t-free trajectory to compute the unsafe region, which is displayed in Figure 9. A boolean formulation of the unsafe region is available in our full paper on arXiv.

B. Verified Turning Maneuvers for Unmanned Aerial Vehicles

Turning maneuvers for unmanned aerial vehicles (UAVs) have been verified as safe in [1], where the UAV was represented as a circular safety buffer around a point object fixed along the trajectory. The KeYmaera X proof presented in [1] required a significant amount of human interaction (on the order of hundreds of hours); in contrast, the method presented in this paper generates an explicit formulation from the trajectory fully automatically. This work represents motion in a two-dimension plane viewed top-down, with the buffer "puck" taking the form of a circle. The turning maneuver trajectory moves along a circular arc then in a straight line:

$$(x_{\mathcal{T}}, y_{\mathcal{T}}) = \begin{cases} x_{\mathcal{T}}^2 + y_{\mathcal{T}}^2 = R^2 & y_{\mathcal{T}} < x_{\mathcal{T}} \tan\theta \\ y_{\mathcal{T}} = \frac{R\cos\theta - x_{\mathcal{T}}}{\tan\theta} + R\sin\theta & y_{\mathcal{T}} \ge x_{\mathcal{T}} \tan\theta \end{cases} \tag{14}$$

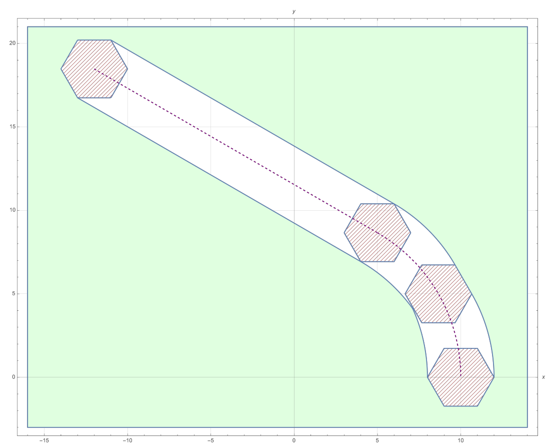


Fig. 10: Approximated safe region for an instance of [1]. The notches are the red-hatched hexagons, the trajectory is dashed in purple.

With a circular safety buffer of radius r_p, the implicit formulation ensures for all points along the trajectory, the obstacle (x_O, y_O) is at least r_p away.

$$\forall x_{\mathcal{T}}.\forall y_{\mathcal{T}}.\big(\text{traj}(x_{\mathcal{T}}, y_{\mathcal{T}}) \implies (x_O - x_{\mathcal{T}})^2 + (y_O - y_{\mathcal{T}})^2 \geq r_p^2\big) \tag{15}$$

Note that our method does not support circular objects, only polygons, so we overapproximate the circular safety buffer as a regular hexagon inscribing a circle. This approximation allows a valid overapproximation of the unsafe region, since the hexagon contains the original circle in [1]. Note that the approximation of a circle can be made arbitrarily precise by increasing the number of sides of a polygon used. A plot of the unsafe region is in Figure 10 and a boolean formulation of the unsafe region is in our full paper on arXiv.

C. Runtime Evaluation

This section presents a comparison of our method to quantifier elimination via cylindrical algebraic decomposition (CAD) [12]. We consider a variety of cases, from fully numeric to fully symbolic. Fully symbolic cases use trajectories without real constants, like $ax^2 + bx + c = d$, and polygons with variable dimensions like rectangles of width w and height h. Fully numeric cases instantiate all parameters with reals to yield, say, trajectory $4x^2 + 2x + 1$ and a rectangle of width 2 and height 1. In Table I, our "Numeric Trajectory" examples instantiate only the trajectory with reals but leave the polygon symbolic, and the "Numeric Hexagon/Rectangle" examples leave the trajectory symbolic but use reals for the polygon dimensions.

Results were generated using a 2017 iMac Pro workstation with 128 GB of RAM, with CAD results using Mathematica's `Resolve` implementation. A table of results is shown in Table I. We use the examples from VI-A (ACAS X) and VI-B (UAV). The Dubins path example is inspired by common path planners and takes the form of two circular arcs connected by a straight line and ending with a line; its symbolic trajectory equation is included in our full paper on arXiv.

Our findings in Table I demonstrate the advantages and disadvantages of our method relative to quantifier elimination using CAD. For non-polynomial examples like a rectangle moving along the Dubins path described above or the UAV example from [1], the active corner method is able to compute fully symbolic formulations of the safe region when CAD fails to return an answer when run overnight (8+ hours). We do note that due to the complexity of a symbolic *hexagon* moving along the Dubins path, the number of transition points means our method cannot compute an answer, though neither can CAD. For a fully numeric example from [1], CAD took 2381 seconds to run but returned `False` incorrectly in place of a region. Additionally, memory is often a constraint for symbolic computation given the CAD algorithm's doubly-exponential runtime [14]; many examples consumed 100+ GB of RAM and one case grew to consume 350 GB of RAM without returning an answer. In the worst case, however, our method consumes under 100MB of RAM. On the other hand, for strictly polynomial examples like that in [22], CAD runs quickly and efficiently, though our method remains competitive.

VII. Related Work

Reachability computation is a vital question in safety-critical cases where users seek to guarantee properties or behavior. One method of constructing reachable sets for safety is zonotope reachability [3]. Reachability computation using zonotopes offers efficient algorithmic methods and supports analysis of dynamical systems with uncertainty. Zonotopes have been used in verification of automated vehicles [2], the design of safe trajectories for quadrotor aircraft [24], and the analysis of power systems [15], among other applications. Zonotope reachability methods discretize a dynamical system and iteratively propagate an estimate of the reachable set forward in time. Their input is a differential equation, while our method requires an explicit closed-form trajectory. For the purpose of checking safety, the estimate of the reachable set must be either exact or an overestimate; in order to deal with discretization error, zonotope methods repeatedly overestimate the reachable interval. Zonotope methods for nonlinear systems rely on linearization and again account for error that may occur by expanding the reachable set [4]. Our method yields exact reachable sets. While it is possible to model convex object reachability with zonotopes, the reachable set expands with the time horizon because the dimensions of the object are treated not as constant dimensions but as uncertainty in initial conditions that is propagated forward through time [20].

Interval-based reachability methods share similarities to zonotope methods but do not aim for the tightest approximations possible; instead simpler axis-aligned sets (hyper-rectangles in high dimensions) are used for computation [32], [33]. These representations simplify storage in memory and operations like intersections but do not compute exact estimates in the way our method does. However, they do support both continuous [31], [25] and discrete [13] dynamic systems with uncertainty. Set-valued constraint solving may be used but similarly relies on inexact discretization [21]. Other reach-

Example	Instance	Active Corners Time	Active Corners RAM	CAD Time	CAD RAM
UAV	Fully Numeric	**0.48 sec**	**7.1 MB**	*2381* sec*	30.89 MB
UAV	Numeric Trajectory	**0.82 sec**	**8.4 MB**	DNF	50+ GB
UAV	Numeric Hexagon	**38 sec**	**22 MB**	DNF	100+ GB
UAV	Fully Symbolic	**45 sec**	**24 MB**	DNF	100+ GB
Dubins	Fully Numeric	**1.2 sec**	**9.0 MB**	DNF	11+ GB
Dubins	Fully Symbolic: Rectangle	**4505 sec**	**91 MB**	DNF	4+ GB
Dubins	Fully Symbolic: Hexagon	DNF	N/A	DNF	8+ GB
ACAS X	Fully Numeric	0.13 sec	5.9 MB	**0.04 sec**	**160 KB**
ACAS X	Numeric Trajectory	0.48 sec	6.6 MB	**0.04 sec**	**188 KB**
ACAS X	Numeric Rectangle	0.51 sec	6.6 MB	**0.2 sec**	**325 KB**
ACAS X	Fully Symbolic	**0.57 sec**	6.6 MB	1.1 sec	**1.8 MB**

TABLE I: Evaluation results, with better results **bolded**. DNF: example did not finish in 8+ hours. **: incorrect answer.*

ability methods for differential equations include Hamilton-Jacobi reachability for systems with complex, nonlinear, high-dimensional dynamics [7], and control barrier functions, which enable the construction of safe optimization-based controllers [5].

A counterpart to reachability is automatic invariant generation for hybrid systems, in which a formal statement showing a system never evolves into an unsafe state is proved. In [17], the authors proved a polynomial and its Lie derivatives can represent algebraic sets of polynomial vector fields. A procedure to check invariance of polynomial equalities was proposed in [18]. Semi-algebraic invariants for polynomial ODEs were studied in [19], [40], [28]. Invariants for hybrid systems were studied in [37] and [29]. Relational abstractions bridge the gap between continuous and discrete modes by over-approximating continuous system evolution to summarize the system as a purely discrete one using invariant generation [38]. Barrier certificates have also been used as invariants for safety verification in hybrid systems [36].

Our work has similar aims to swept-volume collision checking, from path planning and graphics, in which approximate, efficient collision-checking is performed as a volume is moved along a path. A convex over-approximation swept-volume approach was presented in [16]. Swept-volume checking in four dimensions was performed using an intersection test in space-time in [10]. An efficient algorithm computing distances between convex polytopes, the Lin-Canny algorithm, was proposed for this task in [26], [27]. Methods are typically discrete and approximate for performance in online applications. That said, there are some exact methods such as collision checking for straight-line segments like those on robotic arms [39] and an algorithm for large-scale environments [11]. However, these methods operate on individual collision checking instances, such as graphics simulations or video game environments, and their results cannot be used repeatedly. Our method yields provably correct, fully symbolic, and exact safe regions for continuous trajectories and supports, for example, quantifier-free and efficient testing in runtime or in large-scale settings once a desired safe region formulation has been generated.

Another alternative to this work is quantifier elimination, a general algorithm for converting formulas with quantified variables into equivalent statements that are quantifier-free [41], [12]. Quantifier elimination can be performed using Cylindrical Algebraic Decomposition (CAD), an algorithm that operates on semialgebraic sets [12], [6]. QEPCAD is one notable software tool implementing CAD that could be used in this work [8]. The runtime of the CAD algorithm is doubly exponential in the number of total variables (not the number of quantified variables) [14], [9]; we offer a detailed comparison to CAD in Mathematica in Section VI-C.

VIII. Conclusion and Future Work

We have presented an automated approach to construct explicit safe regions for convex polygons moving in the plane with piecewise equations of motion of the form $y = f(x)$ and $x = f(y)$. We have also proved the equivalence of the implicit and explicit formulations of the safe region, discussed an automated implementation of our method, and benchmarked the performance of our method compared to quantifier elimination using cylindrical algebraic decomposition.

We would like to study how our method extends to objects translating in 3 or n dimensions; we conjecture that active corners will become active edges in three dimensions (and $(n-2)$-polyhedra in n dimensions). Additionally, we would like to expand to handle trajectories in the form of inequalities; rotating objects; and invariants of differential equations of the form $f(x, y) = 0$ rather than explicit trajectories. On the implementation side, we are currently exploring how to automatically output a machine-checkable proof of equivalence between the implicit and explicit formulations, using the PVS theorem prover.

Acknowledgements

The authors would like to thank Nikos Aréchiga and Gabor Orosz for their feedback during the development of this research, as well as Jiawei Chen, Necmiye Ozay, and Mohit Tekriwal for comments on earlier versions of this paper. This work was partially funded by a NASA Fellowship. Toyota Research Institute ("TRI") provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. Thanks also to the SISL lab at Stanford for their `aircraftshapes` TikZ library.

References

[1] Eytan Adler and Jean-Baptiste Jeannin. Formal verification of collision avoidance for turning maneuvers in uavs. In *AIAA Aviation 2019 Forum*, page 2845, 2019.

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

[3] Matthias Althoff, Goran Frehse, and Antoine Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:369–395, 2021.

[4] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048, 2008.

[5] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.

[6] Dennis S Arnon, George E Collins, and Scott McCallum. Cylindrical algebraic decomposition i: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.

[7] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.

[8] Christopher W. Brown. Qepcad b: A program for computing with semi-algebraic sets using cads. *SIGSAM Bull.*, 37(4):97–108, December 2003.

[9] Christopher W Brown and James H Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 54–60, 2007.

[10] Stephen Cameron. Collision detection by four-dimensional intersection testing. 1990.

[11] Jonathan D Cohen, Ming C Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–ff, 1995.

[12] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In H. Brakhage, editor, *Automata Theory and Formal Languages*, pages 134–183, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[13] Samuel Coogan and Murat Arcak. Efficient finite abstraction of mixed monotone systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 58–67, 2015.

[14] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1):29–35, 1988.

[15] Ahmed El-Guindy, Yu Christine Chen, and Matthias Althoff. Compositional transient stability analysis of power systems via the computation of reachable sets. In *2017 American Control Conference (ACC)*, pages 2536–2543. IEEE, 2017.

[16] A Foisy and V Hayward. A safe swept volume method for collision detection. In *The Sixth International Symposium of Robotics Research*, pages 61–68, 1993.

[17] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 279–294. Springer, 2014.

[18] Khalil Ghorbal, Andrew Sogokon, and André Platzer. Invariance of conjunctions of polynomial equalities for algebraic differential equations. In *International Static Analysis Symposium*, pages 151–167. Springer, 2014.

[19] Khalil Ghorbal, Andrew Sogokon, and André Platzer. A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets. *Computer Languages, Systems & Structures*, 47:19–43, 2017.

[20] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[21] Luc Jaulin. Solving set-valued constraint satisfaction problems. *Computing*, 94(2):297–311, 2012.

[22] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. A formally verified hybrid system for the next-generation airborne collision avoidance system. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 21–36. Springer, 2015.

[23] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Aurora Schmidt, Ryan W. Gardner, Stefan Mitsch, and André Platzer. A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. *Int. J. Softw. Tools Technol. Transf.*, 19(6):717–741, 2017.

[24] Shreyas Kousik, Patrick Holmes, and Ram Vasudevan. Safe, aggressive quadrotor flight via reachability-based trajectory design. In *Dynamic Systems and Control Conference*, volume 59162, page V003T19A010. American Society of Mechanical Engineers, 2019.

[25] Thomas Le Mézo, Luc Jaulin, and Benoit Zerr. An interval approach to compute invariant sets. *IEEE Transactions on Automatic Control*, 62(8):4236–4242, 2017.

[26] Ming C Lin. Efficient collision detection for animation. In *Proceedings of the Third Euro-graphics Workshop on Animation Cambridge*, 1992.

[27] Ming C Lin. *Efficient collision detection for animation and robotics*. PhD thesis, PhD thesis, Department of Electrical Engineering and Computer Science, UC Berkeley, 1993.

[28] Jiang Liu, Naijun Zhan, and Hengjun Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 97–106, 2011.

[29] Nadir Matringe, Arnaldo Vieira Moura, and Rachid Rebiha. Generating invariants for non-linear hybrid systems by linear algebraic methods. In *International Static Analysis Symposium*, pages 373–389. Springer, 2010.

[30] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[31] Pierre-Jean Meyer, Samuel Coogan, and Murat Arcak. Sampled-data reachability analysis using sensitivity and mixed-monotonicity. *IEEE control systems letters*, 2(4):761–766, 2018.

[32] Pierre-Jean Meyer, Alex Devonport, and Murat Arcak. Tira: Toolbox for interval reachability analysis. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 224–229, 2019.

[33] Pierre-Jean Meyer, Alex Devonport, and Murat Arcak. *Interval Reachability Analysis: Bounding Trajectories of Uncertain Systems with Boxes for Control and Verification*. Springer Nature, 2021.

[34] Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24-June 28, 2013*, 2013.

[35] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2):219–265, 2017.

[36] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

[37] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Constructing invariants for hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 539–554. Springer, 2004.

[38] Sriram Sankaranarayanan and Ashish Tiwari. Relational abstractions for continuous and hybrid systems. In *International Conference on Computer Aided Verification*, pages 686–702. Springer, 2011.

[39] Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. Exact collision checking of robot paths. In *Algorithmic foundations of robotics V*, pages 25–41. Springer, 2004.

[40] Andrew Sogokon, Khalil Ghorbal, Paul B Jackson, and André Platzer. A method for invariant generation for polynomial continuous systems. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 268–288. Springer, 2016.

[41] Alfred Tarski. A decision method for elementary algebra and geometry. University of California Press, Berkeley, 1951.