Certifying the Safe Design of a Virtual Fixture Control Algorithm for a Surgical Robot*

Yanni Kouskoulas
Johns Hopkins University
Applied Physics Lab

David Renshaw
Carnegie Mellon University
Computer Science Dept.

André Platzer
Carnegie Mellon University
Computer Science Dept.

Peter Kazanzides
Johns Hopkins University
Dept. of Computer Science

ABSTRACT

We applied quantified differential-dynamic logic (Qd$\mathcal{L}$) to analyze a control algorithm designed to provide directional force feedback for a surgical robot. We identified problems with the algorithm, proved that it was in general unsafe, and described exactly what could go wrong. We then applied Qd$\mathcal{L}$ to guide the development of a new algorithm that provides safe operation along with directional force feedback. Using KeYmaeraD (a tool that mechanizes Qd$\mathcal{L}$), we created a machine-checked proof that guarantees the new algorithm is safe for all possible inputs.

Categories and Subject Descriptors

F.3.1 [**Logics and the Meaning of Progams**]: Specifying and Verifying and Reasoning about Programs; I.2.9 [**Artificial Intelligence**]: Robotics

Keywords

Quantified differential dynamic logic, medical robotics, formal verification

1. INTRODUCTION

Imagine an operating room in the near future where a surgeon works diligently to excise a tumor at the base of a patient's skull. The physician might employ newly developed robotic technology to more clearly visualize the surgery and more precisely guide the surgical instrument to make the incisions. The ultimate goal: surgery that is safer and more effective than before, providing better patient outcomes.

The robotic machinery to make this future a reality are slowly being developed, but the systems are complex, and can be prone to subtle, unexpected errors. It is easy to see how safety critical such systems are; a bug in the implementation or error in the algorithm that controls the surgical tool might cause it to make the wrong incision, with devastating consequences for the patient.

The usual approach today for ensuring the safety of complex systems is careful design, thoughtful examination of the algorithms, and testing. This approach was applied in [16], where the authors built the system and tested the final product with a surgical procedure on a cadaver. Testing is useful, but only shows the presence of bugs, not their absence.

This paper describes the analysis of one safety property of a skull-base surgery (SBS) robot algorithm, described in [16], to help ensure its safe and predictable operation. Rather than taking a testing approach, we apply formal methods to analyze the control algorithm of interest. This rigorous analysis ensures that the algorithm and the hardware that it controls behave predictably and safely for *all* possible inputs, rather than only for finitely many test cases. The guarantee we seek is much more comprehensive, and can lead to much safer and more predictable systems.

The contribution of this work is that it helps explore how to usefully apply newly developed formal approaches to practical systems. This has two benefits: first, it helps guide the development and refinement of logics and tools, by identifying what is necessary to put these techniques into widespread use; second, it helps the development of practical robotic systems by introducing new formal methods as a powerful and maturing set of design tools.

2. BACKGROUND

The SBS robot in [16] restricts movement of a surgical tool to be within a preoperatively defined surgical site (see Fig. 1), provides force feedback to the surgeon, to indicate when he or she is approaching these boundaries, and aids in fine control of the tool by damping small movements.

To configure this robot, a surgeon describes an operating volume in which to work by a series of planes oriented and positioned in space, called "*virtual fixtures*." Each planar boundary extends infinitely, and the intersection of the volume on the "safe" side of each plane is designed to exclude the areas of anatomy with which the surgeon does not wish to interact. We are considering planar boundaries because that is what was used in the original work.

During surgery, the surgeon holds the tool, (think of it

*This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246 and NSF EXPEDITION CNS-0926181.

HSCC'13, April 8–11, 2013, Philadelphia, Pennsylvania, USA.

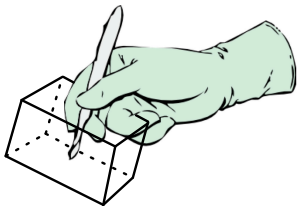


Figure 1: Cooperatively controlled robot enforcing virtual fixtures to restrict a tool-tip to moving within a small volume.

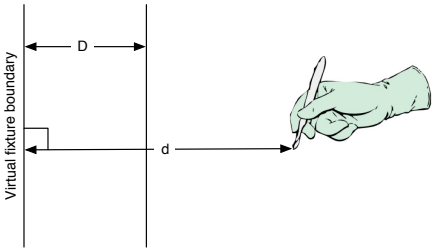


Figure 2: The robot determines which mode it is in depending on the distance from the virtual fixture boundary.

as a scalpel, even though it may be another tool with the same form factor) and applies it to the surgical task at hand, moving it about. The robot is attached to the tool through a rigid mechanical linkage, so as the surgeon exerts forces and torques on the tool, the robot can sense them, and can also exert forces on the tool opposing or aiding the surgeon's movements. This interaction between the SBS robot and surgeon is called cooperative control.

The control algorithm provides three behaviors through different modes of operation: within the center of the safe volume it allows free movement of the tool; close to the surface at the edge of the safe volume it creates an increasing resistance to movement that lets the surgeon know that he or she is close to a virtual boundary; and at the boundary it opposes the movement of the tool, preventing it from crossing the boundary.

Qualitatively, this will produce an invisible boundary that feels soft or mushy. As the tool pushes towards the boundary, the resistance will become progressively firmer; eventually the tool will slow to a stop and "stick" when it reaches the limit of its allowed movement.

The "physics" of the robot are created by an admittance control design, a circuit that converts sensed forces and torques to velocity through a multiplicative factor. If the surgeon exerts force $\bar{f}$ on the tool tip, located at Cartesian position $\bar{p}$, the control circuit translates the input force into a velocity $\bar{p}'$ at the tool tip, given by:

$$\bar{p}' = K(\bar{p})G(\bar{f})\bar{f} \tag{1}$$

where overbars indicate vectors, prime $'$ indicates a derivative with respect to time, and K and G are 3×3 matrices (when just the position is controlled).

A more detailed version of the admittance control equation can be found in [16], but for our purposes this is an adequate model. G is the scale factor, which in the actual system is a matrix with non-linear (exponential) terms described in [7]. It allows the surgeon to switch between moving rapidly over large areas, and doing precise work in a small area with fine control, without interrupting the workflow. In our study, we simplify G and consider it a constant.

$K(\bar{p})$ is a gain term, used to provide force feedback and enforce the "virtual fixtures" as described above. Its elements change form abruptly depending on the position of the tip of the tool, $\bar{p}$.

Regions of movement are defined in terms of D, a design parameter that indicates a cutoff distance from the boundary in its safe direction, as in Fig. 2. In the free region, i.e., the points whose distance d from the boundary satisfy $d > D$, K is the identity matrix. In the slow region, where $0 \leq d \leq D$, the system provides force feedback opposing movement in proportion to how close the tool is to the boundary. Past the edges of the boundary in the unsafe zone, K becomes zero; our tool should not reach past the boundary during normal operation, if our algorithm is safe.

During the experiments [16], Eqn. 1 was effective in preventing the tool from penetrating the safety boundary, but had the undesirable property that once the tool was near the boundary, motion in *all* directions was attenuated, including motions away from the boundary. This led to the development of a new control law, where in the slow region ($0 \leq d \leq D$), the velocity of the tool is attenuated by subtracting a fraction of the component in the direction normal to the boundary. The amount subtracted is proportional to how close to the boundary the tool is at any given point. So if the velocity of the tool is $\bar{p}'$, the distance from the boundary is d, and the unit normal to the boundary is $\hat{n}_1$, the final velocity $\bar{p}'_1$ is given in [5] as:

$$\bar{p}'_1 = \bar{p}' - \left(1 - \frac{d}{D}\right)(\bar{p}' \cdot \hat{n}_1)\hat{n}_1 \tag{2}$$

where $\cdot$ is the dot product of two vectors. The force feedback from each region is applied sequentially, using the attenuated velocity remaining from prior steps. The designers of this algorithm recognize that there is an irregularity in the computation for acute angles, but were not sure what practical effect it had on the system's behavior.

The goal of this research is to ensure that the algorithm safely restricts movement to stay within the virtual fixture boundaries, so that the robot will not adversely impact the safety of the overall system or otherwise unnecessarily endanger the patient.

3. RELATED WORK

There are a number of papers related to the use of virtual fixtures for surgery, such as [10, 1]. The closest to the current work are [4, 9], which take a constrained optimization approach to enforcing more complicated boundaries that are generated by anatomy, and providing an approach that can guide a tool successfully through a specific path. The current work is focused on somewhat orthogonal concerns of improving the fidelity of the model of the interface between discrete program and robotic machinery, (i.e. accurately representing the control mechanism by incorporating realistic delay), and using this improved model to prove safety for all possible inputs, helping to improve the maturity of formal methods. It would be interesting in the future to explore the benefits of applying a formal approach to proving the safety of the more complicated algorithms.

Algorithmic verification approaches such as [2, 3] exist, but they are not appropriate for this work because they could only show safety for a specific number of boundaries, n, i.e. we cannot quantify over elements in the structure in our model. We wish to use one proof to show safety for all possible configurations of any number of boundaries, i.e. $\forall n$, where n is the number of boundaries configured.

At the intersection of formal methods and surgical robotic systems, the authors of this work have also produced related work certifying surgical robot systems in [8, 6], but this prior work is different because it is focused on certifying program implementation for concurrent software, rather than algorithm design for hybrid systems.

4. FORMAL APPROACH

Like all formal methods, the approach we employ has three components: a method for creating a model of the algorithm and the physics of the system, a language for writing a precise specification of its behavior, and a strategy to rigorously prove that the model has exactly the behavior described in the specification. We used two closely related hybrid logics to help us formally verify the safe behavior of the control algorithm used to enforce virtual fixtures: differential-dynamic logic ($\mathsf{d}\mathcal{L}$) [11, 12], and quantified differential-dynamic logic [13, 14] ($\mathsf{Qd}\mathcal{L}$). Both of these logics use the same approach to modeling the system, specifying behavior, and proving properties.

Modeling a hybrid system in $\mathsf{Qd}\mathcal{L}$ and $\mathsf{d}\mathcal{L}$ is done via a hybrid program (HP). The hybrid program looks a bit like an imperative programming language, with real numbers and discrete sets comprising its primitive types. The statements available within a hybrid program are: instantaneous, simultaneous assignment of values θ_i to program state variables x_i, e.g. $x_1 := \theta_1, \ldots, x_n := \theta_n$; enforcing a logical assertion χ on program state, i.e. $?\chi$; composing program α with β, i.e. $\alpha;\beta$; nondeterministically choosing to execute one of two programs α and β, i.e. $\alpha \cup \beta$; repetition of a program α some nondeterministic number of times, i.e. α^*; and continuous evolution of a set of differential equations, i.e. $x_1' = \theta_1, \ldots, x_n' = \theta_n \& \chi$. The last type of statement is the most interesting: it represents the evolution of continuous time in the hybrid system according to the given differential equations, satisfying the first order constraint, χ. The state variables x_i and those contained in terms θ_i, evolve continuously in response to this statement and stop at any time before leaving χ.

Specifying behavior in $\mathsf{Qd}\mathcal{L}$ and $\mathsf{d}\mathcal{L}$ is done using first-order logic with the usual logical connectives, i.e. $\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$, $\phi \rightarrow \psi$, and quantifiers, i.e. $\forall x \phi$, and $\exists x \phi$. The specification language also contains box and diamond modal operators $[\alpha]\phi$ and $\langle\alpha\rangle\phi$; the former is satisfied when ϕ holds after all runs of α, and the latter is satisfied if ϕ holds after at least one run of α.

The proof strategy for these logics depends on applying a set of sound inference rules to transform pieces of the goal into tautologies in real arithmetic.

$\mathsf{Qd}\mathcal{L}$ is a generalization of $\mathsf{d}\mathcal{L}$, designed to allow quantification of variables used as arguments to other functions, and continue to ensure that the resulting arithmetic is decidable. Decidability is important because when working with these logics the leaves of proof trees are systems of equations devoid of hybrid program constructs; solving them proves or disproves that branch of the proof tree. Quantification over variables used as arguments to other functions becomes useful when we model multiple boundaries.

We are fortunate that there is a mechanization of $\mathsf{Qd}\mathcal{L}$, described in [15], that allows us to model and mechanically prove properties of a hybrid system using this new logic.

5. DEVELOPING A SPECIFICATION

We are interested in proving the following critical safety property for our system:

> For any configuration of virtual fixture boundaries, if the surgeon starts the tool at a safe place, for all possible uses of the robot, at each point in time, the tool remains in a safe place.

There are many cases where developing a specification is difficult, because the behavior we want is complicated, or the language we use to describe it is not sufficiently expressive. Neither of these is the case for our particular problem. The property is simple, and our logic is sufficiently expressive. The challenge is how to prove it.

If we call the model of our control algorithm "ctrl," and we have a single boundary i with a unit normal to the boundary $\hat{n}_i$ pointing in the safe direction, a point $\bar{r}_i$ on the boundary, then a tool at position $\bar{p}$ would be safe if $(\bar{p} - \bar{r}_i) \cdot \hat{n}_i \geq 0$. We will call the preceeding expression safe_i.

Thus for the i^{th} boundary, our safety property would be expressed as:

$$\text{safe}_i \rightarrow [\text{ctrl}]\text{safe}_i \tag{3}$$

An implication with a precondition in the implicant, and a modality in the implicand is a common idiom in differential dynamic logic used to represent safety properties. Our safety property says if we start in the safe location for boundary i, and you run the robot using ctrl, then at every point in time, you will also be in a safe location.

If the surgeon specifies 3 virtual fixtures, we could state the safety property as $(\text{safe}_1 \wedge \text{safe}_2 \wedge \text{safe}_3) \rightarrow [\text{ctrl}](\text{safe}_1 \wedge \text{safe}_2 \wedge \text{safe}_3)$. In fact, we considered and verified such properties of the SBS system with a fixed number of virtual fixtures in KeYmaera. But the surgeon can specify any number of virtual fixtures as input, so we would then have to reverify the system if the surgeon ever decided to use 4 or more fixtures. The same problem arises if we consider 10 boundaries or any other fixed number instead.

In $\mathsf{Qd}\mathcal{L}$, however, we can develop a single model for an arbitrary number of boundaries represented, e.g., as a linked list. We can traverse boundaries of interest with an uninterpreted function next(), that when applied consecutively to its result, identifies a list of boundaries, and a hybrid program that iterates over the different boundaries in that list: $[i := \text{first}; (i := \text{next}(i))^*]((i \neq \text{end}) \rightarrow \text{safe}_i)$. Unlike most hybrid programs, which represent a model of a hybrid system, this one's sole purpose for existence is to span the list provided to us by the next() function and terminated by a constant symbol end (satisfying next(end) = end).

For a finite, end-terminated list of boundaries defined by next(), we are interested in proving the following property saying that ctrl safely respects all boundaries always:

$$\begin{aligned}&[i := \text{first}; (i := \text{next}(i))^*]((i \neq \text{end}) \rightarrow \text{safe}_i) \rightarrow \\ &\quad [\text{ctrl}][i := \text{first}; (i := \text{next}(i))^*]((i \neq \text{end}) \rightarrow \text{safe}_i)\end{aligned} \tag{4}$$

What remains is developing the model ctrl and proving that it matches this specification.

6. DEVELOPING A MODEL

In this section, we describe the different steps we took to create an accurate model of the system. This means we will start with simplifying assumptions, create a model of our control algorithm, and show how we refine our model to eventually prove the safety of our system.

Why not simply jump straight to the final result? Because the process of developing and refining a model helps identify problems in the design and can lead to changes in the control algorithm we are modeling. Describing the modeling process illustrates the strategy we use to create these models, and provides general lessons that are applicable to many other hybrid system modeling and verification tasks.

Model development is an integral part of formal verification, and how to get to the right model is not always obvious. For this modeling effort we are not sure what the final model will look like, so we will step back and break the problem into two pieces: first, we must create the model of the control algorithm for the safety and force feedback of a single boundary, and second, we must add code to our single-boundary control model that applies that control to many boundaries sequentially, and produces a combined effect that ensures safety for a finite collection of boundaries.

6.1 Event-Driven Continuous Control

The first attempt at modeling our SBS system is a direct translation of the continuous equations in Eqn. 1. This equation implicitly assumes a sort of continuous control, where our system responds infinitely fast and infinitely often; it will lead to what is sometimes called an *event-driven* system, since it responds instantaneously to events.

There are many analog control circuits that do exhibit behavior that is nearly continuous control. In fact, the underlying control circuit that implements admittance control for the SBS robot is one example, assuming its step response is sharp and its settling time is short compared to the other time scales in the system.

Because the admittance control circuit can be modeled by continuous control, it may be tempting to think that a continuously controlled, event-driven model is a good representation for the virtual fixture control algorithm. It is not. The virtual fixture control algorithm cannot be practically implemented as an analog circuit, because K must be set in a manner consistent with both the current geometric state, but also the currently configured set of virtual fixture boundaries. This means the part of our system that computes K will be digital, and can introduce a significant delay in our response.

This is a very general lesson for anyone who is developing complex control algorithms. The lower level control circuits may be analog, and often can be modeled and analyzed very well by continuous control, and the control and circuit theory that we have available in our toolbox. But when building up more complicated control behaviors at higher levels of system abstraction, those behaviors may require more computation, and it may only be practical to implement them using digital components. In this case, modeling the control algorithm using continuous, event-driven control is a poor approximation that may lead to inaccurate conclusions.

Regardless, this sort of approximation is useful because it provides a quick sanity check of our basic concept under ideal conditions; if our control algorithm does not work correctly under continuous control, we cannot expect it to work under a more realistic model that incorporates delay. We will illustrate the utility of this initial step for model development by creating a model of control for a single boundary. Attempting to apply formal methods to this model will force us to refine the model, and eventually identify a problem with our initial control system design.

Our initial modeling attempt begins by writing pseudo code for a hybrid program that represents our control algorithm. There will be a section of code that assigns values to state variables in the system, and we can call this block "input." It is followed by a non-deterministic choice of different statements describing continuous evolution according to a system of differential equations directly taken from the admittance control equation. Each statement corresponds to a mode of operation, describing that mode's continuous behavior, as well as constraints that must hold in that case. Our pseudo-code looks like this:

ctrl ≡ (input;
((mode 1 diff eqn & mode 1 constraints) ∪
(mode 2 diff eqn & mode 2 constraints) ∪
(mode 3 diff eqn & mode 3 constraints) ∪
...
(mode n diff eqn & mode n constraints)))*

This defines an event-driven controller, because it will switch from one mode to the other as described by the event of moving from one evolution domain constraint to another. The semantics of QdL are nondeterministic, such that the system can switch between overlapping evolution domain constraints at any time. In our system, all evolution domain constraints are disjoint except for their overlapping mode boundaries, so that the system switches over to the other modes exactly at those boundary events.

We can now convert this pseudo-code into an algorithm by adding details. For simplicity, we will first model the controller in two dimensions, and fix a single virtual fixture boundary to the x-axis, with the safe side being quadrants one and two of our cartesian coordinate system, where y is positive. Next, we will fill in the details for "input." From the physician's perspective, the interaction with the system is via simple hand motions, exerting force on the tool to move it around. We model this motion using the nondeterministic assignment to the derivative of the force, creating a piecewise linear model of the force. We use $\bar{f}_p = (f_{xp}, f_{yp})$ to represent the derivative of force with respect to time, and write $f_{xp} := *$ and $f_{yp} := *$ to indicate non-deterministic assignment. (Because our mechanization does not have any way to represent vector quantities, we have to decompose our system into scalar equations.) We relate these quantities to the force by including the appropriate differential equations $f_x' = f_{xp}, f_y' = f_{yp}$ during continuous evolution of our system. Finally, we fill in the differential equations using the basic physics of our admittance control circuit and damping mechanism, given in Eqs. 1 and 2. As in Eqn. 1, $\bar{p}$ represents the position of the tooltip, $\bar{p}'$ is its derivative with respect to time, and $\bar{f}$ represents the force that the surgeon exerts on the system at a given point in time. The logical constraints that are provided during continuous evolution distinguish the different input cases. The refined hybrid program is given in Table 1.

In this simple model, we can see the different cases that our robot controls for, to make a single boundary safe. For

$$
\begin{array}{l}
\text{ctrl}_1 \equiv \big(f_{xp} := *; f_{yp} := *; \\
\text{/* free zone */} \\
(p_x' = Gf_x, p_y' = Gf_y, f_x' = f_{xp}, f_y' = f_{yp} \& (p_y \geq D)) \cup \\
\text{/* slow zone */} \\
(p_x' = Gf_x, p_y' = G\frac{p_y}{D} f_y, f_x' = f_{xp}, f_y' = f_{yp} \& \\
\quad ((0 \leq p_y \leq D) \wedge (f_y \leq 0))) \cup \\
(p_x' = Gf_x, p_y' = Gf_y, f_x' = f_{xp}, f_y' = f_{yp} \& \\
\quad ((0 \leq p_y \leq D) \wedge (f_y \geq 0))) \cup \\
\text{/* past boundary */} \\
(p_x' = 0, p_y' = 0, f_x' = f_{xp}, f_y' = f_{yp} \& ((p_y \leq 0) \wedge (f_y \leq 0))) \cup \\
(p_x' = 0, p_y' = Gf_y, f_x' = f_{xp}, f_y' = f_{yp} \& ((p_y \leq 0) \wedge (f_y \geq 0)))\big)^*
\end{array}
$$

Table 1: Simple, event-driven model exhibiting continuous control for a single virtual fixture in two dimensions.

example, there are two cases in the slow zone, one for moving towards the boundary, and one for moving away from it.

For this system, safety means that we do not move past the virtual fixture boundary. We state our safety property as ensuring that the tool tip, located at $\bar{p}$, will never go into the bad region below the x-axis, so $\text{safe}_1 = (p_y \geq 0)$. We also define some sanity requirements for our system, $\text{sane} = (G > 0) \wedge (D > 0)$ to formalize assumptions about the different parameters.

THEOREM 1. *The event-driven SBS control system in Table 1 is safe for a* single *boundary in two dimensions, i.e., the following* dL *formula is provable:*

$$safe_1 \wedge sane \rightarrow [ctrl_1] safe_1 \tag{5}$$

We were able to mechanize and prove the safety of this system, using KeYmaera, a mechanization of dL.

It is tempting to stop our modeling exercise here, because it seems obvious that this algorithm will be safe for multiple boundaries. Here, we ended up reusing a continuous model appropriate for the design of linear, time-invariant systems, for formal verification of a high level control algorithm that is a hybrid system. For this system, the assumption of continuous control is not appropriate.

Generalizing to Multiple Boundaries.

When we try to extend our formal model to include an arbitrary but finite number of multiple boundaries, we fail; this modeling approach can (with difficulty) be scaled to a higher number of boundaries, but it cannot represent the general case of any number of boundaries in a single model.

To use this modeling approach for an arbitrary number of boundaries, we would have to embed a looping program within different modes describing the continuous dynamics of the system, in order to control for all of the boundaries and their overlapping regions. The semantics of QdL and dL do not allow this, because it would represent the execution of a discrete program instantaneously at every instant of time; this is more than physically impossible to implement.

In the process of formalizing an accurate model of our system, we discover something inaccurate about our model that is dangerous to our conclusions of its safety; our modeling language tells us so by prohibiting what we are trying to do. As discussed above, we have implicitly extended the assumption of continuous control to the implementation of the gain term $K(\bar{p})$, which contains discrete aspects of the system that switch its form, giving the system its hybrid flavor. This system needs a more accurate modeling approach to correctly ensure its safety.

6.2 Improved Delayed-Response Control

To relax the idealized assumptions of the event-driven model, and allow us to represent an arbitrary but unspecified number of boundaries in our system, we will model the control algorithm with delay in the control loop. When we create the software that implements this control algorithm, it will contain a loop that senses its state and the current inputs, makes a decision, and then sends commands to the robotic machinery to respond. This cycle of decision-making introduces some finite (possibly variable) delay, with an upper bound we call ϵ, in the response of the robot. Modeling this delay is appropriate because it more closely matches the system behavior, in that the robot takes some action, and then cannot react again to the external world for some period of time up to ϵ; see Figure 3. For the robot system used in [16], $\epsilon = 18.2$ msec.

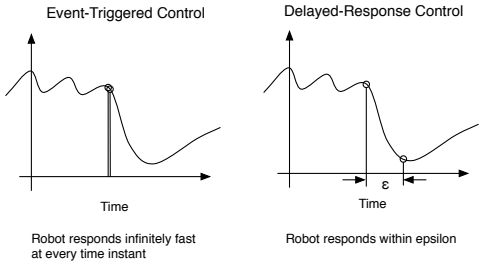


Figure 3: Event-triggered vs. Delayed-Response Control

Converting our event-triggered model to one with a delayed response requires the opposite of what we contemplated doing in the previous section. Rather than embedding a looping program into the continuous dynamics to represent discrete behavior, we must refactor the hybrid program, rewriting it so that the differences in the modes are expressed in the discrete program at the beginning of each continuous evolution. This refactoring is illustrated in Fig. 4.

Once this is done, there will only be one set of differential equations that describes the continuous evolution of our system, and it will simply represent the physics of the system

Event-triggered	**Delayed-Response**
ctrl ≡	ctrl ≡
(discrete;	(discrete;
(mode1 dyn) ∪	(mode1discrete ∪
(mode2 dyn) ∪	mode2discrete ∪
(mode3 dyn))*	mode3discrete);
	$t := 0$; (dyn , $t' = 1 \,\&\, t \leq \epsilon$))*

Figure 4: Converting a model that uses event-triggered control into a model that has a delayed response. Refactoring differences between different modes into a discrete program component, leaving our model with a single term describing the continuous evolution of the system with an extra clock t bounded by ϵ.

and the behavior of our lower-level admittance controller, which should not change regardless of the mode the system is in or the damping decision made by the control algorithm.

6.2.1 First Iteration: Immediate Control

We rewrite Eqn. 1 to match Eqn. 2, the implementation described in [5], for which we wish to prove safety:

$$\bar{p}'(t) = G(\bar{f}(t) - k\hat{n}) \tag{6}$$

In doing so, we define a state variable k that will be used to describe the amount of force-feedback in the direction normal to the boundary. The value of k is computed and set at each step by the discrete program, and used as a constant in the ODE during continuous evolution. Incidentally, this rewriting of Eqn. 1 will become a problem, as it changes the functional form of our equation, and prevents us from generalizing to multiple boundaries.

For our first iteration, we write a discrete program to set the value of g according to the current value of the force and position at the beginning of each step, implement the differential equation above, and use the approach described in Fig. 4 to allow up to ϵ time to elapse before our next control decision. This approach runs into immediate problems, which are evident even without doing a formal proof: Since the system may now be delayed by up to ϵ, we have introduced the possibility that the tool moves so quickly that it crosses the buffer of size D in some time less than ϵ. Whether this occurs in practice will depend on the values of D, ϵ, and the maximum velocity the robot allows. We can prove this if it is not sufficiently obvious.

6.2.2 Second Iteration: Predictive Control

To solve this problem, we must redesign the algorithm so that it anticipates the motion in the time step of duration $\leq\epsilon$, so that it can avoid violating the safety conditions. Rather than reacting only in the fixed forced-feedback zone, we will anticipate, based on the tool's position, its velocity, and its acceleration, whether we need to oppose its motion to prevent it from exceeding the virtual fixture we have defined. Force feedback can be applied if and when the tool is safe, in the slow (force-feedback) zone, and moving slowly enough to provide useful feedback to the user. At this point, our redesign is focused on safety only, since we need to get this right before we add force feedback.

The rest of this section describes the details of this design, which requires: computing the characteristics of the path the tool takes during the time step that represents the robot's delay; enumerating input cases, and computing k so that the robot's behavior remains safe; and proving that the design does in fact ensure the safety we expect.

Calculating the Tool's Path.

In order to create a predictive version of the algorithm, we must calculate the path of the tool with respect to the boundary. The force exerted on the tool at a point in the time step can be written as the initial force at the beginning of a step (subscript 0), plus the derivative of the force times time, and each of these can further be broken into x and y coordinates:

$$\bar{f}(t) = \bar{f}_0 + \bar{f}_p t = (f_{x0} + f_{xp}t)\hat{x} + (f_{y0} + f_{yp}t)\hat{y} \tag{7}$$

where $\hat{x}$ and $\hat{y}$ are unit normals in the x and y directions.

By calculating force components normal to the boundary, we can relate these quantities to the frame of reference of the boundary, and more easily model a boundary at an arbitrary location and an arbitrary orientation. We can write f_n, the force normal to the boundary, as the initial normal force f_{n0} plus the derivative f_{np} of the normal force times time

$$f_n(t) = \bar{f}(t)\cdot\hat{n} = (\bar{f}_0\cdot\hat{n}) + (\bar{f}_p\cdot\hat{n})t = f_{n0} + f_{np}t \tag{8}$$

By solving the ODE in Eqn. 6, we find the position $\bar{p}$ of the tooltip at any time t during the step, and then we can use this to find the distance $d(t)$ from the boundary:

$$\begin{array}{rcl} \bar{p}(t) & = & \bar{p}_0 + G(\bar{f}_0 - k\hat{n})t + \frac{1}{2}G\bar{f}_p t^2 \\ d(t) & = & (\bar{p}(t) - \bar{r})\cdot\hat{n} \\ & = & d_0 + G(f_{n0} - k)t + \frac{1}{2}Gf_{np}t^2. \end{array} \tag{9}$$

We need to look at the undamped case to know whether we need the robot to apply damping. The discriminant of the quadratic describing the distance will tell us whether our parabolic trajectory without damping intersects the boundary we are interested in; if $\text{disc} = (Gf_{n0})^2 - 2Gf_{np}d_0$ is positive, we know that we may intersect the virtual fixture during the next ϵ time step.

Calculating Safe k for Different Input Conditions.

Now we need to use Eqn. 9 to calculate a k that ensures safety throughout the time step of duration $\leq\epsilon$. We want to damp it safely, but not unnecessarily, depending on the specifics of the motion that is being made. For example, if the tool's motion is away from a boundary, we don't need any damping, unless the acceleration eventually reverses its direction and has it intersecting the boundary during this time step, in which case we do. There are various cases for this type of motion, and we have to compute a safe value of k for each case, and then use the results during the execution of our hybrid program.

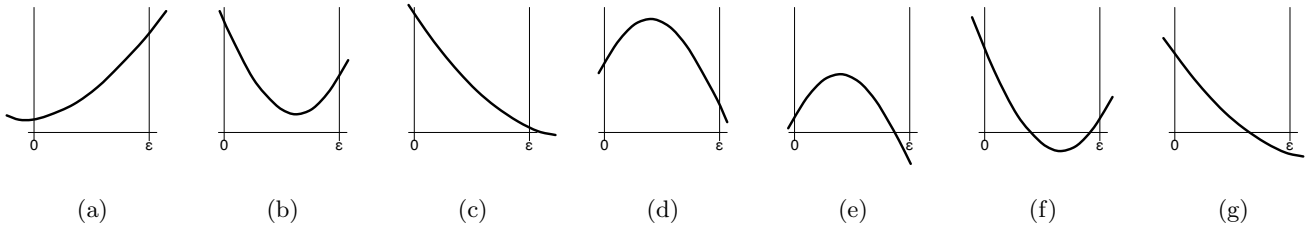
The cases are shown in Fig. 5, and the calculations for k used in the model for each case follow. How do we know that we have safely addressed all of the cases? This is one of the benefits of a formal approach. We think carefully about the cases, enumerate them, and then calculate a safe k depending on the behavior we seek. Then, we attempt to prove the safety property. If we have missed any cases, or miscalculated k, the proof will fail and lead to a counterexample of whatever cases we have missed. In this way, verification can support the design process for this control algorithm.

There are four cases where the robot should not do anything, because we will not hit the boundary during this time step. These four cases correspond with the subfigures a–d in Fig. 5, and for these cases, the robot will not add any damping; we represent this in our model by setting $k := 0$ for these cases.

(a) We are moving away from the boundary ($f_{n0} \geq 0$), and accelerating away from the boundary ($f_{np} \geq 0$).

(b) We are moving towards the boundary ($f_{n0} \leq 0$), but accelerating away from the boundary ($f_{np} \geq 0$), so that we turn around before we reach it ($\text{disc} \leq 0$).

(c) We are moving towards the boundary ($f_{n0} \leq 0$), but accelerating away from the boundary ($f_{np} \geq 0$) so that we will intersect it ($\text{disc} \geq 0$), but not during this time step ($d(\epsilon) \geq 0$ and $f_n(\epsilon) \leq 0$), so no action is necessary yet.



(a) (b) (c) (d) (e) (f) (g)

Figure 5: Each subfigure represents a movement scenario in which the skull-base surgery robot must enforce safety. The y-axis of each subfigure represents the distance of the closest approach of the tool to a single virtual fixture boundary, i.e. $d(t)$. The x-axis of each subfigure represents the progression of time, with a zero-reference being assigned to the beginning of a step, progressing to the maximum delay, ϵ. The final, safe version of the realistic controller contains each of these scenarios as an explicitly controlled case.

(d) We are accelerating towards the boundary ($f_{np} \leq 0$) but we never reach it during our step ($d(\epsilon) \geq 0$).

There are three cases where the robot must interfere, to prevent the tool-tip from crossing the virtual fixture. These remaining three cases correspond with the subfigures e–g in Fig. 5. The logic and calculations for k follow for each case:

(e) We are accelerating towards the boundary ($f_{np} \leq 0$), and we would reach it or exceed it at the end of our time step (i.e. $[k := 0]d(\epsilon) \leq 0$[1]) unless additional damping is applied.

Since we are accelerating towards the boundary, we know the furthest we can reach past the boundary will be at time ϵ. To ensure that we do not cross the boundary, we find a damping value that just slows us enough so that we touch but do not cross the boundary at this time, by solving $d(\epsilon) = 0$ for k. We find $k := f_{n0} + \frac{(d_0 + \frac{1}{2} G f_{np} \epsilon^2)}{(G\epsilon)}$ to ensure safety, so we set that value as k at the beginning of the step for this case.

(f) We are moving towards the bounday ($f_{n0} \leq 0$) but accelerating away from the boundary ($f_{np} \geq 0$), and we will still intersect it (disc ≥ 0) during this time step. We know this because the trajectory has turned around ($f_n(\epsilon) \geq 0$) and reached its minimum.

Here, we need to ensure that the minimum value just touches the boundary, so that no part of the path actually crosses it. We compute the time where our parabolic trajectory reaches its minimum, $t_m = \frac{-f_{n0}}{f_{np}}$; this is the furthest past the boundary that our path reaches. Now we can require that the discriminant of the resulting quadratic equation be zero, and solve for the damping that would be required to create that path. We find $k := f_{n0} - \sqrt{\frac{2 f_{np} d_0}{G}}$ would be the damping required to keep this safe.

(g) We are moving towards the bounday ($f_{n0} \leq 0$) but accelerating away from the boundary ($f_{np} \geq 0$), but we will still intersect it (disc ≥ 0) during this time step. We know this because even though the trajectory is still downward, ($f_n(\epsilon) \leq 0$), we have exceeded the boundary by the end of the step ($[k := 0]d(\epsilon) \leq 0$).

This trajectory is monotonic during our step and moving towards the boundary. The position $[k := 0]d(\epsilon)$ thus represents the furthest possible distance we can go past the boundary during the step, without damping.

By applying damping and opposing the downward trajectory, the minimum will move to an earlier time. Without damping, the time at which our parabolic trajectory reaches a minimum is beyond our time step, $t_m \geq \epsilon$, but by changing the damping k, we may find it has moved so that $t_m < \epsilon$. If the t_m occurs during our ϵ step, we need to produce a force that ensures that the minimum point is raised so that our path does not intersect the boundary. We must solve $d(t_m) = 0$ for k, and in this case, we find that $k := f_{n0} - \sqrt{\frac{2 f_{np} d_0}{G}}$ is a safe damping coefficient, as in case f. If the k computed above still leaves $t_m \geq \epsilon$, then this correction is more than is necessary. We can solve $d(\epsilon) = 0$ for k, and exactly as in case e we find that a damping of $k := f_{n0} + \frac{(d_0 + \frac{1}{2} G f_{np} \epsilon^2)}{(G\epsilon)}$ keeps our trajectory safe.

Proving (Un)Safety.

We have been able to quickly produce a design that accurately represents lag in the controller and the linear damping described in [16, 5], and can be applied to multiple boundaries. The modeling effort and our formal approach has led us to make some modifications to the control algorithm to improve safety, namely we redesigned the algorithm so that it damps its movements predictively, taking into account time lag in the system.

We are able to use KeYmaera to prove the safety of this design for a single boundary, but when we implement multiple boundaries in KeYmaeraD, this algorithm is in general unsafe, and we can prove it with a counterexample. We will present an informal proof by constructing a description of an algorithm that depends upon sequential application of Eqn. 2 for each boundary, and pointing out that any algorithm that makes these assumptions produces a counterexample when one tries to prove its safety.

Consider a set of boundaries, C. For each boundary $i \in C$, there is a a normal $\hat{n}_i$, and a distance of the tool to the closest point on that boundary d_i. We write an assertion,

[1]We use notation from differential dynamic logic to indicate the setting the state variable k to a value in the following calculations. So $[k := 0]d(\epsilon) \leq 0$ is the assertion that with the value of k set to zero, evaluating the equation $d(\epsilon)$ yields a value less than zero.

Q_i, that encodes what it means to be safe for boundary i:

$$Q_i(\bar{w},\bar{v}) \equiv \left(-\bar{w}\cdot\hat{n}_i \le -\bar{v}\cdot\hat{n}_i\frac{d_i}{D}\right) \vee (\bar{w}\cdot\hat{n}_i \ge 0) \tag{10}$$

In this assertion, the quantity $\bar{v}$ represents the velocity of the tool tip before we exert any control, and $\bar{w}$ represents the velocity of the tool tip, after we have exerted control. The satisfaction of $Q_i(\bar{w},\bar{v})$ describes the exerting of safe control for boundary i, when the initial velocity without control would have been $\bar{v}$, by modifying the final velocity to be $\bar{w}$. In other words, $Q_i(\bar{w},\bar{v})$ is satisfied when both $\bar{w}$ and $\bar{v}$, have components normal to the boundary i that are towards the boundary, and $\bar{w}$'s normal component is attenuated in proportion to its distance from the boundary, *or* when $\bar{w}$ has its component normal to the boundary going away from the boundary.

We assume C_s is a proper subset of C, and $\bar{w}$ is a velocity such that $\forall i \in C_s, Q_i(\bar{w},\bar{v})$. This means that all forces have been attenuated properly for the boundaries indexed in C_s.

We wish to construct an algorithm which when given $\bar{v}$, and $\bar{w}$, computes a new velocity $\bar{x}$ for each index $j \in C \setminus C_s$ such that,

$$\forall i \in C_s, Q_i(\bar{x},\bar{v}) \wedge Q_j(\bar{x},\bar{v}). \tag{11}$$

By induction, such an algorithm could then be applied to each successive boundary to compute the resulting, force-feedback velocity that is safe for all virtual fixture boundaries.

An example of an algorithm that makes these assumptions follows:

$$\bar{x} = \begin{cases} \bar{w} + \left[\left(\frac{d_j}{D}\bar{v} - \bar{w}\right)\cdot\hat{n}_j\right]\hat{n}_j & \text{if } \bar{w}\cdot\hat{n}_j \le 0 \wedge \bar{v}\cdot\hat{n}_j \le 0 \wedge \\ & \quad \left(-\bar{w}\cdot\hat{n}_j \ge -\bar{v}\cdot\hat{n}_j\frac{d_j}{D}\right) \\ \bar{w} & \text{if } \bar{w}\cdot\hat{n}_j \le 0 \wedge \bar{v}\cdot\hat{n}_j \le 0 \wedge \\ & \quad \left(-\bar{w}\cdot\hat{n}_j \le -\bar{v}\cdot\hat{n}_j\frac{d_j}{D}\right) \\ \bar{w} & \text{if } \bar{w}\cdot\hat{n}_j \ge 0 \\ \bar{w} - (\bar{w}\cdot\hat{n}_j)\hat{n}_j & \text{if } \bar{w}\cdot\hat{n}_j \le 0 \wedge \bar{v}\cdot\hat{n}_j \ge 0 \end{cases} \tag{12}$$

The first case simply subtracts out the component of tool velocity normal to the boundary that would be necessary to attenuate it proportional to how close the tool is to the boundary. The second and third cases do nothing, as the normal is already attenuated due to transformations from other boundaries. The fourth case represents conditions where other boundaries are producing attenuating velocity in opposition to what is necessary to attenuate the normal component of velocity with respect to the current boundary.

When attempting a safety proof for this algorithm, we have to prove Eqn. 11 by showing the safety of different cases in Eqn. 12. We ended up disproving one of the cases by finding a counterexample during an attempt to complete the safety proof using KeYmaera. In particular, we found a counterexample in the fourth case where:

$$\begin{aligned} &\{Q_i(\bar{w},\bar{v}), \bar{w}\cdot\hat{n}_j \le 0, \bar{v}\cdot\hat{n}_j \ge 0, \hat{n}_i\cdot\hat{n}_j \le 0\} \nvdash \\ &\quad Q_i(\bar{x},\bar{v}) \wedge Q_j(\bar{x},\bar{v}), \end{aligned} \tag{13}$$

violating our specification. The term $\hat{n}_i\cdot\hat{n}_j \le 0$ describes the subset of the fourth case where the counterexample occurs. The counterexample occurs when boundaries intersect each other at acute angles, and the dot product of their normals is a negative quantity, i.e., they partially face each other.

The existence of this counterexample means that there are geometric configurations of multiple boundaries where the correction introduced by the system for one boundary can push the tool past another virtual fixture boundary into an unsafe location. This means that the tool can "slip" past a virtual fixture boundary, through the edge formed by the intersection of two different virtual fixtures. This also means that for large velocity movements, the process of enforcing one virtual fixture boundary can violate another, sufficiently closely spaced boundary.

For multiple, arbitrarily configured boundaries, an algorithm based on these assumptions is not in general safe.

6.2.3 Third Iteration: Non-linear Damping

We wish to make our system safe, yet produce a directional damping so that motions away from the boundary are not attenuated. Because of the problems described above, we revert back to damping the system with a multiplicative factor as in Eqn. 1. Multiplicative damping ensures that damping from one boundary will not be reversed by further damping from another. Still using the notation from Eqns. 7–8, we will revisit the design using the formal approach we have adopted, which will help us ensure safety as we design the directional damping we seek. As before, we start with

$$\bar{p}' = K(\bar{p})G(\bar{f})\bar{f} \tag{14}$$

but we choose a form for K appropriate to non-linear damping, $K = \left(\begin{smallmatrix} K_x & 0 \\ 0 & K_y \end{smallmatrix}\right)$.

The rest of this section follows the structure of Section 6.2.2, computing the characteristics of the path the tool takes during the time step, enumerating input cases, and computing safe damping coefficient, and finally proving that the design does in fact ensure the safety we expect.

Calculating the Tool's Path.

We can now compute the position of the tool-tip during each step and its distance from a boundary, as before, first solving Eqn. 14, and then computing the distance during the time step as before.

$$\begin{aligned} \bar{p}(t) &= \bar{p}_0 + G(K_x f_{x0}\hat{x} + K_y f_{y0}\hat{y})t + \\ &\quad \tfrac{1}{2}G(K_x f_{xp}\hat{x} + K_y f_{yp}\hat{y})t^2 \\ d(t) &= (\bar{p}(t) - \bar{r})\cdot\hat{n} \\ &= d_0 + G(K_x f_{x0}n_x + K_y f_{y0}n_y)t + \\ &\quad \tfrac{1}{2}G(K_x f_{xp}n_x + K_y f_{yp}n_y)t^2 \end{aligned} \tag{15}$$

In this case, the discriminant will tell us whether our parabolic trajectory intersects the boundary we are interested in. The discriminant is given by $\text{disc} = G^2(K_x f_{x0}n_x + K_y f_{y0}n_y)^2 - 2G(K_x f_{xp}n_x + K_y f_{yp}n_y)d_0$. The turn-around point for this trajectory is at $t_m = \frac{-(K_x f_{x0}n_x + K_y f_{y0}n_y)}{(K_x f_{xp}n_x + K_y f_{yp}n_y)}$ as long as $(K_x f_{xp}n_x + K_y f_{yp}n_y) \neq 0$.

Calculating Safe k for Different Input Conditions.

We go through the exercise of computing the control for each input scenario in Fig. 5, as in Sec. 6.2.2. We assume $k = K_x = K_y$. The only differences are in cases e, f, and g.

(e) Solving for $d(\epsilon) = 0$, we find $k := -\frac{d_0}{G}\left(f_n\epsilon + \frac{1}{2}f_{np}\epsilon^2\right)^{-1}$.

(f) For our system, the minimum point is thus at $t_m = \frac{-(K_x f_{x0}n_x + K_y f_{y0}n_y)}{(K_x f_{xp}n_x + K_y f_{yp}n_y)}$. With $K_x = K_y$, $t_m = \frac{-f_{n0}}{f_{np}}$. We can require that the the discriminant of the resulting quadratic equation be zero, and solve it, finding that $k := \frac{2f_{np}d_0}{Gf_{n0}^2}$ provides appropriate damping to guarantee a safe trajectory.

(g) In general, the trajectory reaches its minimum point at $t_m = \frac{-f_{n0}}{f_{np}}$. It is sufficient to ensure $d(\epsilon) = 0$ exactly as in case e. We find $k := -\frac{d_0}{G}\left(f_n\epsilon + \frac{1}{2}f_{np}\epsilon^2\right)^{-1}$ safely damps the system, as before.

We also add force feedback into this design, as described in [16], defining a slow zone that is between the virtual fixture boundary, and up to a distance D away. We attenuate the tool's movement by a factor proportional to how far into the slow zone the tool has progressed, measured by its distance from the boundary divided by the total thickness of the slow zone, i.e. $\frac{d(t)}{D}$. The tool will slow down more and more as it approaches the boundary. We apply force feedback only when the tool's path primarily moves towards the virtual fixture, and when its starting point is within the slow zone. If after applying attenuation that provides force feedback, we would cross a virtual fixture boundary, we can add further attenuation to those cases, as described above, to ensure that the system remains safe.

Careful modeling of our robot and the application of formal methods have led us to a design of a single-boundary control algorithm that satisfies our original requirements, accurately represents lag, predictively enforces safety, and is *composable* so that it can be applied to multiple boundaries in sequence, and still ensure a safe system.

We can now complete the model of our redesigned control algorithm by taking the single-boundary control program and embedding it in two loops, one nested inside the other. The outer loop models the controller's interaction with the physical world, describing force input from the surgeon, and the evolution of continuous time after the control algorithm has provided force feedback and enforced safety. The nested inner loop models the control algorithm being applied successively to each boundary; each iteration of the inner loop computes and applies force feedback for, and enforces the safety of, a single arbitrarily oriented boundary. The nested inner loop iterates over all of the boundaries. The final model is given in Table 2.

Safety Proof.

After our careful modeling effort, we were able to mechanize the safety proof in KeYmaeraD.[2]

THEOREM 2. *The SBS controller in Table 2 is safe for an arbitrary number of arbitrarily oriented and positioned boundaries in three dimensions, i.e., the following* QdℒL *formula is valid:*

$$(G > 0) \wedge (e > 0) \wedge (D > 0) \wedge (k \geq 0) \wedge (next(end) = end) \wedge$$
$$(\forall i : B, |\hat{n}(i)|^2 = 1) \wedge (\forall (h\ g : B), (next(h) \neq end) \rightarrow$$
$$(h \neq g) \rightarrow (next(h) \neq next(g))) \wedge$$
$$[i := first; (i := next(i))^*]((i \neq end) \rightarrow safe_i) \rightarrow$$
$$[ctrl_2][i := first; (i := next(i))^*]((i \neq end) \rightarrow safe_i)$$

The proof is structured the same way as the model: with two inductive steps, one nested inside the other. Both inductive reasoning steps have the three branches: one representing their base cases, one for the inductive step, and the last for the postcondition. The middle "inductive step" of the first application of induction leads to the second application of loop induction. (This is the location of the nesting.) The second "inductive step" of the second application of inductive reasoning breaks out into one hundred and forty different branches, each representing slightly different conditions of force feedback and safety enforcement.

To see where the branches come from, we can examine the final model in Table 2. For each time step, the system: computes a damping coefficient to provide force feedback (ten cases); independently computes a damping coefficient to ensure safe operation enforcement of virtual fixture boundaries (seven cases discussed in Fig. 5); and then compares the damping coefficients and makes a decision whether to apply safety damping depending on their relative magnitudes. Each of these actions is shown in the model, and the product of these cases, $10 \times 7 \times 2 = 140$, gives us the total number of major subcases in the proof.

The main idea of the quantified proof is to show that for each branch in the inner inductive step, the application of damping for a given boundary, either by leaving the original damping in place, or in applying additional damping to provide force feedback or ensure safety, successfully ensures safety of that boundary. The proof must also show that when additional multiplicative damping factors are added, that additional damping does not impair the safety of the other boundaries that have been controlled-for so far.

The final, completed safety proof has 156,024 proof steps. The proof steps are described by a manually created proof script, that allows the user to describe the structure of the proof. The script is like a program that describes how to conduct the proof, detailing individual steps in parts of the proof, and automating the proving process in other parts, re-using proof strategies as appropriate. On a laptop, the 2D version of the model takes around two minutes to machine-check the proof, confirming its truth, while the full 3D version takes 70 minutes to complete, mostly waiting to automatically solve the arithmetic for certain unoptimized branches of the proof tree.

7. FUTURE WORK

One avenue of future work might be to explore how to address more complicated boundaries. The current approach could be extended by designing a control algorithm that dynamically adds and removes boundaries depending on where you are in the allowed operating volume. Boundaries can be safely removed at any time, and boundaries can be safely added at the beginning of a time step, providing that the tool tip is in a safe location with respect to the new boundary when it is added. One can conceive of extensions to the current algorithm that would simulate surfaces with convex topologies and cusps, by selectively adding and removing planar boundaries at appropriate times to stay inside one convex connected component at a time. This would be an interesting verification challenge.

8. REFERENCES

[1] J. Abbott, P. Marayong, and A. Okamura. Haptic virtual fixtures for robot-assisted manipulation. In S. Thrun, R. Brooks, and H. Durrant-Whyte, editors, *Robotics Research*, volume 28 of *Springer Tracts in Advanced Robotics*, pages 49–64. Springer, 2007.

[2] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30:179–198, 2007.

[2]The final models and proofs are available online at `http://symbolaris.com/pub/medrobot-examples.zip`

$$
\begin{array}{l}
\text{ctrl}_2 \equiv \Big(f_{xp} := *; f_{yp} := *; f_{zp} := *; k := 1; i := \text{first}; \\
\quad \big(?i \neq \text{end}; \\
\qquad \text{xtr}(i) := *; u(i) := 1; \\
\qquad d_0(i) := (p_x - r_x(i))n_x(i) + (p_y - r_y(i))n_y(i) + \\
\qquad\quad (p_z - r_z(i))n_z(i); \\
\qquad f_{np}(i) := f_{xp}n_x(i) + f_{yp}n_y(i) + f_{zp}n_z(i); \\
\qquad f_n(i) := f_x n_x(i) + f_y n_y(i) + f_z n_z(i); \\
\qquad ((?(d_0(i) \geq d \wedge (3 \geq 3)); \text{// force feedback section} \\
\qquad\quad u(i) := *; (?u(i)1 = 1)) \cup \\
\qquad (?((0 \leq d_0(i)) \wedge (d_0(i) \leq d)); \\
\qquad ((?f_{np}(i) \geq 0 \wedge f_n(i) \geq 0); \\
\qquad\quad u(i) := *; (?u(i)1 = 1)) \cup \\
\qquad ((?f_{np}(i) > 0 \wedge f_n(i) \leq 0 \wedge \\
\qquad (\text{xtr}(i) f_{np}(i) = -1 f_n(i)) \wedge (\text{xtr}(i) \leq (1/2)e)); \\
\qquad\quad u(i) := *; (?u(i)1 = 1)) \cup \\
\qquad ((?f_{np}(i) > 0 \wedge f_n(i) \leq 0 \wedge \\
\qquad (\text{xtr}(i) f_{np}(i) = -1 f_n(i)) \wedge (\text{xtr}(i) \geq (1/2)e)); \\
\qquad\quad u(i) := *; (?u(i)d = d_0(i))) \cup \\
\qquad ((?f_{np}(i) = 0 \wedge f_n(i) \leq 0); \\
\qquad\quad u(i) := *; (?u(i)d = d_0(i))) \cup \\
\qquad ((?f_{np}(i) < 0 \wedge f_n(i) \geq 0 \wedge \\
\qquad (\text{xtr}(i) f_{np}(i) = -1 f_n(i)) \wedge (\text{xtr}(i) \leq (1/2)e)); \\
\qquad\quad u(i) := *; (?u(i)d = d_0(i))) \cup \\
\qquad ((?f_{np}(i) < 0 \wedge f_n(i) \geq 0 \wedge \\
\qquad (\text{xtr}(i) f_{np}(i) = -1 f_n(i)) \wedge (\text{xtr}(i) \geq (1/2)e)); \\
\qquad\quad u(i) := *; (?u(i)1 = 1)) \cup \\
\qquad ((?f_{np}(i) = 0 \wedge f_n(i) \geq 0); \\
\qquad\quad u(i) := *; (?u(i)1 = 1)) \cup \\
\qquad ((?f_{np}(i) \leq 0 \wedge f_n(i) \leq 0); \\
\qquad\quad u(i) := *; (?u(i)d = d_0(i)))) \cup \\
\qquad (?((d_0(i) \leq 0) \wedge (3 \geq 3)); \\
\qquad\quad u(i) := *; (?u(i)1 = 0))); \\
\qquad \text{dist}(i) := (d_0(i) + Gu(i)(f_n(i)e + f_{np}(i)e^2(1/2))); \\
\qquad \text{disc}(i) := ((Gu(i)f_n(i))^2 - 2Gu(i)f_{np}(i)d_0(i)); \\
\qquad ((?(f_{np}(i) \leq 0 \wedge \text{dist}(i) \geq 0); \text{// safety section} \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G = 1); \text{kt} := u(i)) \cup \\
\qquad (?(f_{np}(i) \leq 0 \wedge \text{dist}(i) \leq 0); \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G(f_n(i)e + (1/2)f_{np}(i)(e^2)) = 1); \\
\qquad\quad \text{kt} := (-1)d_0(i)\text{tmp}) \cup \\
\qquad ((?f_{np}(i) \geq 0 \wedge f_n(i) \leq 0 \wedge \text{disc}(i) \leq 0); \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G = 1); \text{kt} := u(i)) \cup \\
\qquad ((?f_{np}(i) \geq 0 \wedge f_n(i) \leq 0 \wedge \text{disc}(i) \geq 0 \\
\qquad\quad \wedge f_n(i) + f_{np}(i)e \geq 0); \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G(f_n(i)^2) = 1); \\
\qquad\quad \text{kt} := 2f_{np}(i)d_0(i)\text{tmp}) \cup \\
\qquad ((?f_{np}(i) \geq 0 \wedge f_n(i) \leq 0 \wedge \text{disc}(i) \geq 0 \wedge \\
\qquad f_n(i) + f_{np}(i)e \leq 0 \wedge \text{dist}(i) \leq 0); \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G(f_n(i)e + (1/2)f_{np}(i)(e^2)) = 1); \\
\qquad\quad \text{kt} := (-1)d_0(i)\text{tmp}) \cup \\
\qquad ((?f_{np}(i) \geq 0 \wedge f_n(i) \leq 0 \wedge \text{disc}(i) \geq 0 \wedge \\
\qquad f_n(i) + f_{np}(i)e \leq 0 \wedge \text{dist}(i) \geq 0); \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G = 1); \text{kt} := u(i)) \cup \\
\qquad ((?f_{np}(i) \geq 0 \wedge f_n(i) \geq 0); \\
\qquad\quad \text{tmp} := *; (?\text{tmp}G = 1); \text{kt} := u(i))); \\
\qquad ((?\text{kt} \geq k; k := k) \cup (?\text{kt} < k; k := \text{kt})); i := \text{next}(i) \\
\quad \big)^*; (?i = \text{end}); t := 0; \\
\quad (p_x' = Gf_x k, p_y' = Gf_y k, p_z' = Gf_z k, \\
\quad f_x' = f_{xp}, f_y' = f_{yp}, f_z' = f_{zp}, t' = 1 \& t \leq e) \Big)^*
\end{array}
$$

Table 2: A complete time-triggered model of a redesigned control algorithm that enforces the safety of an arbitrary number of arbitrarily oriented and positioned virtual fixture boundaries, in three dimensions. This model is realistic, provides directional force feedback, and is proven to be safe.

[3] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *CAV*, pages 379–395, 2011.

[4] A. Kapoor, M. Li, and R. Taylor. Constrained control for surgical assistant robots. In *ICRA*, volume 1, pages 231–236, 2006.

[5] P. Kazanzides. Virtual fixture computation. Note on combining the effects of multiple virtual fixtures, Dec. 2011.

[6] P. Kazanzides, Y. Kouskoulas, A. Deguet, and Z. Shao. Proving the correctness of concurrent robot software. In *ICRA*, pages 4718–4723. IEEE, 2012.

[7] P. Kazanzides, J. Zuhars, B. Mittelstadt, and R. H. Taylor. Force sensing and control for a surgical robot. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 612–617, May 1992.

[8] Y. Kouskoulas, F. Ming, Z. Shao, and P. Kazanzides. Certifying the concurrent state table implementation in a surgical robotic system (extended version). Technical report, Yale University, 2011.

[9] M. Li, M. Ishii, and R. Taylor. Spatial motion constraints using virtual fixtures generated by anatomy. *Robotics, IEEE Transactions on*, 23(1):4–19, 2007.

[10] S. Park, R. Howe, and D. Torchiana. Virtual fixtures for robotic cardiac surgery. In W. Niessen and M. Viergever, editors, *MICCAI*, volume 2208 of *LNCS*, pages 1419–1420. Springer, 2001.

[11] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.

[12] A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

[13] A. Platzer. Quantified differential dynamic logic for distributed hybrid systems. In A. Dawar and H. Veith, editors, *CSL*, volume 6247 of *LNCS*, pages 469–483. Springer, 2010.

[14] A. Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4):1–44, 2012.

[15] D. W. Renshaw, S. M. Loos, and A. Platzer. Distributed theorem proving for distributed hybrid systems. In S. Qin and Z. Qiu, editors, *ICFEM*, volume 6991 of *LNCS*, pages 356–371. Springer, 2011.

[16] T. Xia, C. Baird, G. Jallo, K. Hayes, N. Nakajima, N. Hata, and P. Kazanzides. An integrated system for planning, navigation and robotic assistance for skull base surgery. *International Journal of Medical Robotics*, 4(4):321–330, 2008.