ScreamaeraX: Modeling Laryngeal Surgical Robots Using Differential Dynamic Logic

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Computer Science

By:

Natalie McClain

Project Advisors:

Prof. Rose Bohrer
Prof. Loris Fichera

Date: April 2023

Abstract

Surgical robots are a benefit to both doctors and patients, but carry risks in their usage. Modeling them as cyber-physical systems allows us to formally prove that they will not violate safety conditions. The Super-elastic Continuum Robot for Endoscopic Articulation and Manipulation (SCREAM) is a laryngeal endoscopic robot designed for better reachability within the larynx. We created mathematical models with differential equations of a SCREAM's motion in progressively more complex ways. We used Differential Dynamic Logic ($\mathsf{d}\mathcal{L}$) to prove the safety of these models with a proof assistant. In this, the safety condition was not stabbing the larynx of a patient. We created four models, one in two-dimensions and three in three-dimensions, each one with a closer to accurate model of motion than the previous.

Acknowledgements

I would like to thank my advisors, Professors Rose Bohrer and Loris Fichera, for their support and feedback throughout the project. Professor Bohrer provided familiarity with software verification and experience with the topics as well as keeping me focused on the project. Professor Fichera's experience with robotics was invaluable, especially as I got into more complex movements of the robot. I'd also like to thank Boots McClain and the little team of other solo-MQP students who helped keep me on track the last several months.

Contents

List of Tables

List of Figures

1 Introduction

The use of an endoscopic instrument to treat a benign or pre-malignant tumor in the folds of the larynx presents a more appealing option for patients than alternatives [1]. The procedure is less invasive than traditional methods, and can often be completed in an outpatient setting while the patient is awake, removing the need to recover from general anesthesia. Previous Major Qualifying Projects have designed such a robot to better access the laryngeal folds, and use a laser to remove the tumors [2].

For this project, we sought to model this robot as a Cyber-Physical System. By modeling the movement of this robot and it's environment, we can prove safety statements. Because individuals' larynges are different, and very complex shapes, the models are forced to use a simplified shape [3].

To create these models, we used KeYmaera X, an axiomatic Tactical Theorem Prover for Hybrid Systems [4]. This program gave us a framework to define the problem, and assisted in proving the model. For this project we developed three models, with increasing complexity. Chapters 2 and 3 will explain KeYmaera X and Differential Dynamic Logic ($\mathsf{d}\mathcal{L}$) in further detail, as well as further detailing how this method has been used with surgical robots previously. In chapter 4, we will explain the preparation undergone in the early stages of this project. Chapter 5 will explain the models and how we proved them, and finally chapter 6 will explain what further work can be done in this field.

2 Background: Differential Dynamic Logic and KeYmaera X

Cyber-physical systems (CPS) are, as the name suggests, systems that are both cyber and physical. In this case, cyber describes computation, control, and communication while physical describes motion or any other physical properties. They are used to model everything from planes to robots to autonomous vehicles. Previous work has been done to prove how CPS can be used to verify safety across the abstraction gaps between a model and real physics [5]. Through the use of *sandboxing*, a potentially unsafe model can be caught if it violates safety conditions by making a nondeterministic choice, and instead will follow a deterministic path that is known to be safe. Cyber-physical systems are often modeled with hybrid programs, which we will be focusing on from now on.

For this project, we used $\mathsf{d}\mathcal{L}$ to create our models [4]. We describe $\mathsf{d}\mathcal{L}$'s syntax and semantics, i.e., notations and their meanings. The semantics are state-based: each state ω maps variables x to real numbers $\omega(x)$ in $\in \mathbf{R}$. The syntax consists of terms, hybrid programs, and formulas. Terms are polynomials with real

values and a numeric meaning in each state. Hybrid programs may nondeterministically change the state when run. Formulas are either true or false in each state. Hybrid programs and formulas may contain each other.

2.1 Hybrid Programs

We are using the definitions from Chemical Case Studies in KeYmaera X [6]. Hybrid programs α, β are defined by

$$\alpha, \beta ::= ?P \mid x := e \mid x' = f(x) \,\&\, Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

The hybrid programs (HP) are defined by how they run. From a defined start state, what final states can they reach? They can be deterministic, nondeterministic, or terminate early and not reach any final state.

The test program, or $?P$ never modifies the state. In this context, P is a formula, meaning it is either true or false (see above). In this case that it is true, the test ends in the current state. But if it is false, then there is no final state, instead causing an execution failure. The next program, deterministic assignment $x := e$ updates the state by setting the value of x to the current value of e. Next, $x' = f(x)\&Q$, are ordinary differential equations (ODEs), which are a powerful feature of hybrid programs. This is how continuous is modeled within a hybrid system. The sample system. ODEs here models how x evolves in continuous time with regards to $f'(x)$, where $f(x)$ is a term. How long this lasts is nondeterministic, in this problem, Q is the evolution domain constraint. If a formula Q is provided, Q is tested continuously and the evolution must stop before it becomes false. A choice, $\alpha \cup \beta$, nondeterministically runs either α or β, but not both. Composition $\alpha; \beta$ runs both, first α then β in the resulting state. Finally loops, $\alpha*$ run a nondeterministic but finite number of repetitions, zero, one, or many.

These programs fit together into varied ways. For example, nondeterministic choice can be combined with a test to create an if-else statement. The program "if$(P)\alpha$ else β" is equivalent to $\{?P; \alpha\} \cup \{?\neg P; \beta\}$

2.2 Formulas

Formulas in $\mathsf{d}\mathcal{L}$ are limited to:

$$P, Q ::= e \geq \tilde{e} | \neg P | P \wedge Q | P \rightarrow Q | [\alpha] P | \langle \alpha \rangle P$$

Formulas are either true or false about the state ω. The first formula, comparison $e \geq \tilde{e}$ is true when e is greater than or equal to $\tilde{e}$. Logical connectives allow for the definition of all other comparisons $(>, =, != , \leq, <)$, allowing for any comparison to be used in a formula. Negation, $\neg P$, is true when P is false. Conjunction, $P \wedge Q$, is true when both P and Q are true, equivalent to a logical "and". Implication, $P \rightarrow Q$, is true when P being true implies Q being true as well. Combinations of negation and conjunction allow for the definition of all other logic operators. For example $\neg(\neg P \wedge \neg Q)$ defines logical disjunction, or the "or" operator $P \vee Q$.

The final two formulas define $\mathsf{d}\mathcal{L}$, and allow the inclusion of hybrid programs within formulas. Box formula, $[\alpha]P$, is true if in state ω, *every* run of α starting in ω ends in a state where P is true. Diamond formula $\langle \alpha \rangle P$, is true if *some* run of α from ω will result in a state where P is true.

2.3 Defining Systems and Proofs

In order to prove a model, we must first create it. For each model, we define a loop invariant and a safety condition. The invariant must hold true before and after each loop is completed. The safety condition must be true at all times. Because of this, the invariant is a formula that implies the safety condition. The arithmetic to define these conditions is often complex, especially as the functions defining movement become more complex. Previous work has been done to determine criterion for checking invariants for hybrid systems. This criterion resulted in a method of generating invariants, however it relies there existing such an invariant that fits a predetermined template [7]. We did not use this method for defining our invariants.

This project focused on time-triggered style models, which mean that the models follow a pattern of 1. setting initial conditions, 2. control, where changes are defined, 3. the changes are acted upon for a nondeterministic amount of time up to T, 4 the loop invariant is checked, and then the steps 2-4 loop [4].

Some of the proofs required use of differential cuts, invariants and weakening. Differential cuts are similar to a logical cut, and allow the user to define a lemma with regards to the premise, use it to help prove the premise, and get proved itself. The differential cut proves a post-condition then assumes it in the domain constraint, which places a pseudo-restriction on the domain of the differential equation, but ultimately does not change the problem [4].

In our proof assistant, the dIRule, or differential invariant rule, proves a formula P to be true

locally by reducing it to it's differential $(P)'$. It reduces a property of an ODE, $x' = f(x)$ to that of a discrete assignment, $x' := f(x)$. This allows for complex ODEs with nontrivial solutions to be checked easily. Instead of solving the ODE, the proof merely requires deriving the post-condition P and substitution [4].

2.4 KeYmaera X

KeYmaera X is a prover for modeling and proving cyber-physical systems. It analyzes control programs along with physical behavior of the system to verify safety and reachability properties of the systems [8]. It proves truth in every state, or validity [6]. The prover component is interactive and tactic based [9]. At each step, the user selects a proof technique, each of which are implemented as tactics. This allows complex methods to be reduced to simpler components, which are more trustworthy in their simplicity.

3 Related Work

Previous work has been done with both the robot that is the focus of these models, and in utilizing Cyber-physical systems and $\mathsf{d}\mathcal{L}$ with surgical robots.

3.1 SCREAM

The subject of the modeling in this project is SCREAM, or Super-elastic Continuum Robot for Endoscopic Articulation and Manipulation [2], which was developed to better reach the area around the vocal folds within the larynx with an endoscope [1]. This robot has three degrees of motion: translation through the larynx, rotation within the larynx, and bending of the instrument at the end of the endoscope.

The key pieces of information we will use in our models are the degrees of motion, as well as the limits on how far they can move in any direction, for example the probe cannot bend past 90 degrees.

The benefits of endoscopic nasal surgeries include that the treatments do not require a general anesthetic and are often in a doctor's office as opposed to operating room, both of which can lead to lower charges for the patient. These surgeries utilize a laser and fiber optic cables to fire laser pulses at diseased tissue until the cells are killed. However, the ability to reach all of the larynx with existing optical fibers is limited, as the doctor has little control over the direction of the laser while still maintaining a view of the field. The previous project, SCREAM, was designed to overcome these problems by describing a new robotic device to enable bending of the optic fiber to aim the laser.

Previous use of cyber controlled medical systems have had dire consequences. In the mid 1980s, Therac-25, a software-controlled radiation therapy machine greatly overdosed six people resulting in injury and death [10]. Conducting software verification builds safety inherently into the cyber-physical system. Professor Nancy Levenson reflected on the incident 30 years later, stating "Almost all software-related accidents have involved requirements flaws, not coding or implementation errors...we have to focus less on assurance and more on identifying the safety-critical requirements and building safety into these machines from the beginning of development" [10]. Software verification fits well to fulfill this. By identifying what safety means in the context of a system, we can use verification to ensure that these safety-critical requirements are sufficient.

3.2 Previous Verification with a Surgical Robot

Previously, quantified differential dynamic logic (QdL) has been applied to a surgical robot, using a relative of KeYmaera X [11][12]. This research created a model of algorithm and physics of the system, utilized a language to precisely specify it, and then conducted a rigorous proof to show the behavior of the model was exactly that of the specification. They created a hybrid program as described above, but specifically used QdL, which is a generalization of $\mathtt{d}\mathcal{L}$. QdL adds quantified programs, new variable types, and decidability which is important for the proof later on. The advantage of QdL was it allowed them to have as many dimensions as they needed to represent the envelope of the robot, which was the complex shape that defined the footprint of the robot.

In using QdL, previous researchers were able to find a flaw in the algorithm of a surgical robot. They proved it was in general unsafe, and applied QdL to develop a new algorithm which would provide for safe operation. They then created a proof with QdL that guaranteed the safety of this algorithm for all possible inputs. Unlike the robot at the focus of this project, the robot in this paper was one designed to be used by a surgeon in a more invasive surgery involving the patient being unconscious. This robot is used within the patient, and its more complexly shaped environment, compared to the simplified environment that SCREAM acted in, were also substantial differences that resulting in this project not requiring QdL. The movement of this robot also did not involve rotation, unlike SCREAM.

4 Preparation

For the first part of this project we worked through part of a graduate special topics class that utilized KeYmaera X to familiarize ourselves with the concepts and proof assistant. We worked through a number of homework assignments, and created both time- and event-triggered models of simple movement, for both of these a ball bouncing and simple autonomous vehicles. These vehicles helped to familiarize us with designing invariants, and defining systems through mathematical functions. We were able to apply these examples directly to the earliest model with simple translating motion.

5 Models and Results

In this project we modeled the movement of a hand-held robot for in-office laser surgery on a patient's vocal folds. The robot has three degrees of motion - translation, rotation, and bending of the instrument on the end. There are many goals for proof in the context of this robot. The primary goal for this project was to prove safety of robot usage by ensuring it does not stab the walls of the larynx. We proved this through progressively more complex models.

The complexity of the model was based on both how we allow the robot to move, and how the larynx is modeled. We began with a two-dimensional model, with the larynx modeled by a simple rectangle, and the robot only translating in two directions within it. This model will be referred to as the 2d model. It was followed by a shift to three dimensions, with the larynx now modeled as a cylinder. The first three-dimensional model has simplified movement, with rotation and translation in two directions, replacing the hinging motion with another translation. This model is the 3d-translate model. We continued to model the larynx as a cylinder, first with a model replacing that second degree of translation with a hinging motion (3d-hinge model). The most complex model would be a model that utilizes a more complex, and thus realistic, model of a larynx, which would also introduce the challenge of reaching various points in the larynx.

Beyond this Nasal Scope Safety (NSS) model, there are other safety measures that could be modeled. For example, ensuring the laser is not pointed at any point of the larynx for too long as to not cause additional scarring. After proving safety, the next step would be proving the ability to reach the previously difficult-to-reach points in the larynx, which is the advantage of SCREAM [1], and then proving both safety and reachability simultaneously.

All of the models we designed use a time-triggered framework as opposed to an event-triggered framework because in the context of this robot, controlled by a doctor, only being able to receive commands every set time interval was more realistic. This framework allows for a nondeterministic choice to allow for a change in behavior at every control cycle, lasting at most some time $T > 0$.

5.1 Two Dimensional Model

The first step we took in creating the initial model, 2d, was to decide how to define the shape of the larynx. Our initial thought was to define it with value defining lines marking each edge of the rectangle. However we realized it would be simpler to place the corner of the rectangle on the origin, and define just a height and width to the rectangle. These were defined as $RHeight$ and $RWidth$ respectively. The location of the

robot was defined with $xScream$ and $yScream$, representing x and y coordinates. From there, we defined the safety condition as never hitting the "end" of the larynx, represented in this model by $x = RWidth$, or hitting either wall of the larynx, represented by $y = rHeight$ and $y = 0$. The final safety constraint was

$$xScream < RWidth \land yScream < RHeight \land yScream > 0$$

For this model, the invariant is the same as the safety condition. A maximum velocity for both x and y directions is defined initially, as $xVelLimit$ and $yVelLimit$. One of the initial conditions placed on these velocities is that in one control cycle, the robot will not travel across the entire larynx in any direction.

We assumed the acceleration for this model (and all subsequent models) to be negligible, so in each control cycle there is a non-deterministic assignment of velocity for each direction. The possible velocity for each direction was the positive of the initially declared maximum, the negative of that value, or zero. The robot can always safely not move at all in either direction (set both velocities to zero). Because negative velocity is only safe in some cases, the velocity limits, $xVelLimit$ and $yVelLimit$, must be positive. Because we are assuming no acceleration, the calculation of the potential locations at the next control cycle is simple. For each nondeterministic choice for velocity, the new position is calculated for that dimension. If it does not break the safety condition it is acceptable. No test is conducted for setting velocity to zero because not moving is always safe in this model. This movement does not address the possibility of diagonal movement because for the rectangle model, diagonal movement poses no different risk than two one-directional movements. In order for a diagonal movement to be unsafe, the motion is also unsafe on either the x or y axis. The control for the x direction is shown below.

```
?(xScream + xVelLimit * T < length); xVel := xVelLimit;}
++ xVel := -xVelLimit;
++ xVel := 0;} /* assign an x velocity */
```

This model can be proved with just the `auto` tactic because of the invariant and safety condition. See Appendix A for the full model and proof.

5.2 Three Dimensional Translation Model

As we moved to the third dimension for this model, we began to model the larynx as a cylinder. We realized that adding the rotation would make the arithmetic for determining the location of the robot as well as how it moved significantly more complex in a Cartesian coordinate system. As shown below in Figure 1, the

movement for 3d-translate is very easily expressed in terms of polar coordinates - the robot has a distance from a central z-axis and an angle relative to the y-axis - so we designed an intermediate model called 3d-translatePolar to use as a stepping stone to determine movement functions.

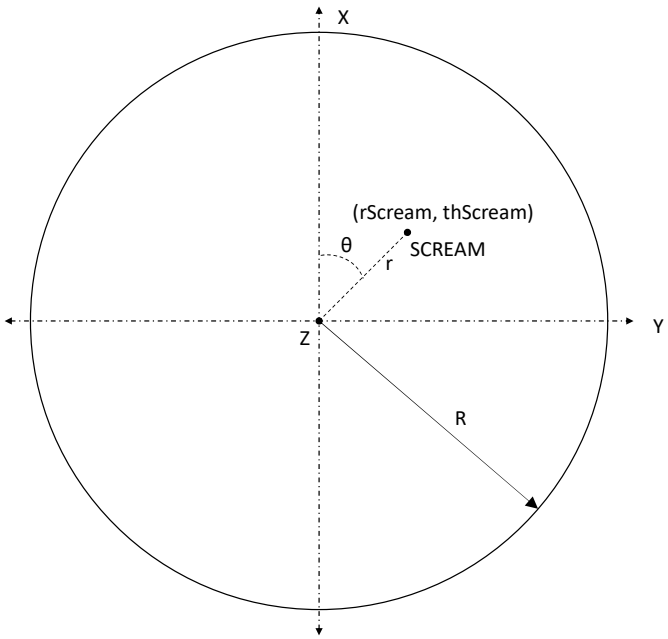

Figure 1: The location of 3d-translate in a cross section of the larynx

5.2.1 Polar Model

This model was similar to the two dimensional model, but now instead of having x and y directions, we used r, θ, and z directions. As in the previous model, we assume acceleration to be negligible. Because it is rotational, we defined the velocities as *speed* for changes in angle, $rVel$ for changes in radius, and $zVel$ for changes in z. Similarities between this and the previous models extend to the control for the z and r values, which is nearly identical to the y and x control previous. Because a change in angle cannot result in a dangerous outcome in a perfectly cylindrical larynx, the control for the angle was quite simple. Either the *speed* had a value of 0 and it did not change, or it had a value of *speedLimit* assigned as an initial condition to be positive. This control is shown below.

```
{speed := speedLimit;
    ++ speed := 0;} /* assign an angular speed (change in angle)*/
```

The continuous dynamics for this model were also relatively simple. A velocity for each coordinate was defined in the control and used in the ODE. The invariant and safety constraint were once again the identical, ensuring $rScream$ remained less than the radius of the larynx, and greater than the negative radius, and that $zScream$ was less than the *length* of the cylinder, ensuring it would not go past the end of the larynx.

```
  /*Continuous Dynamics*/
  t := 0;{
      {rScream' = rVel,
        thScream' = speed, zScream' = zVel,
        t' = 1 & t <= T} /* evolution domain and time-trigger */
    }
  }*@invariant(rScream < radius & rScream > -radius & zScream < length) /* loop
  invariant */
]
(rScream < radius & rScream > -radius & zScream < length) /* safety condition */
```

The initial conditions were defined such that the dimensions of the cylinder were positive and the robot began at the origin, at the center of one end of the cylinder. We defined limits for the velocities, making them all positive and that in a full time step T they would not go past any face of the cylinder. This intermediate model was able to be proved with a loop invariant and `auto`. The complete model and proof can be found in Appendix B.

5.2.2 Cartesian Model

With the intermediate model completed, we could move on to the model with the same motion, but instead of defining the location of SCREAM based on polar coordinates, it would be defined by Cartesian coordinates. This model used a similar control to the previous model, but instead of having a defined $rScream$, we defined a function symbol called *norm* that would calculate the distance from the z-axis given and x and y position.

```
  Definitions
      Real norm(Real x, Real y) = (x^2 + y^2)^(1/2);
  End.

  {{?(norm(xScream, yScream) + rVelLimit * T < radius); rVel := rVelLimit;}
      ++ {?(norm(xScream, yScream) - rVelLimit * T > -radius); rVel := -
```

```
rVelLimit;}
     ++ rVel := 0;} /* assign a radius velocity (change in radius)*/
```

Because the motion was still inherently polar, and was defined as such, it made the ODE more complicated than previous. We used partial derivatives of the functions to convert polar to Cartesian to adjust. The continuous dynamics for $xScream$ and $yScream$ are shown below.

```
xScream' = -yScream * (speed/norm(xScream, yScream)) + (xScream*rVel)/norm(
xScream, yScream),
yScream' = xScream * (speed/norm(xScream, yScream)) + (yScream*rVel)/norm(
xScream, yScream)
```

The invariant and safety condition were again identical, and also similar to the previous with the usage of the *norm* function again.

```
zScream < length & norm(xScream, yScream) < radius
```

Proving this model however, was more complicated than previous. While most of the branches of the proof proved automatically, the two cases with an increasing radius and change in the z value were beyond the scope of the `auto` tactic, so a differential cut was used to characterize the movement. This additional statement (below) clarified that the maximum change in radius from the current point in the remaining time in the control cycle could not result in a movement past the edge of the cylinder.

```
(xScream^2+yScream^2)^(1/2)+rVelLimit()*(T()-t) < radius()
```

With that the prover was able to complete the proof using the `auto` tactic.

5.3 Three Dimensional Hinge Model

The movement of the final model can be seen in Figure 2. The two angles, one measured from the z-axis and one from the y-axis, can be seen and are labeled as θ and ϕ respectively.

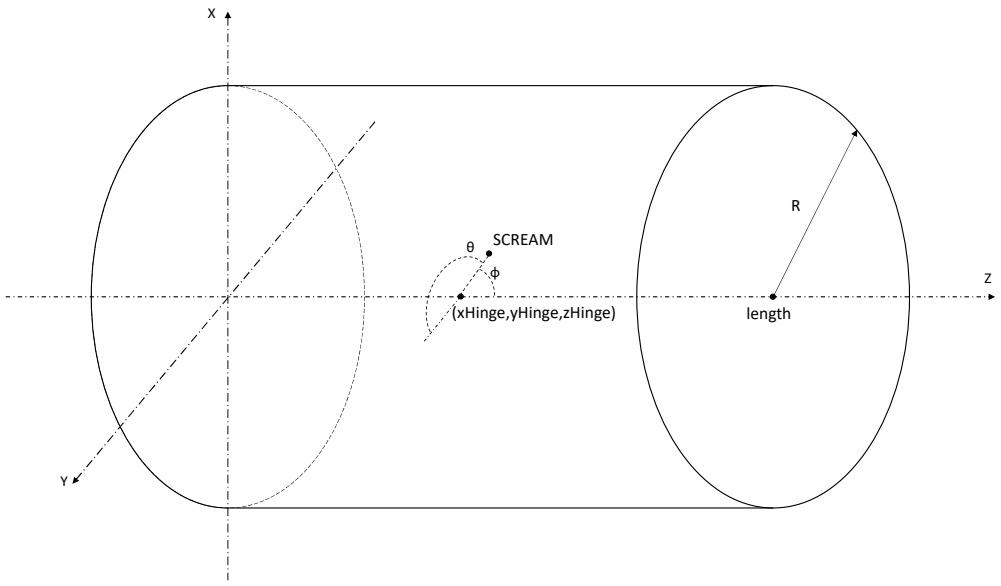

Figure 2: The movement of 3d-hinge

Allowing for rotation in two directions presented its own share of problems. Because the hinging motion is in only one direction, and that direction is relative to the relative y-axis that is twisting with the other rotation, the two rotations must be combined. We did this using rotation matrices. The twisting motion of this model is about the z-axis, so the rotation matrix can be represented by

$$R_{twist} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The hinge motion is about the y axis, so its rotation matrix is represented as such

$$R_{hinge} = \begin{bmatrix} cos\phi & 0 & -sin\phi \\ 0 & 1 & 0 \\ sin\phi & 0 & cos\phi \end{bmatrix}$$

To combine the rotations, we simply multiply them

$$R_{twist \times hinge} = \begin{bmatrix} cos\theta cos\phi & -sin\theta & -cos\theta sin\phi \\ sin\theta cos\phi & cos\theta & -sin\theta sin\phi \\ sin\phi & 0 & cos\phi \end{bmatrix}$$

To find the location of the end of the instrument based on the known location of the hinge point, we multiplied $R_{twist \times hinge}$ by the vector $v = \langle r, 0, 0 \rangle$ where r is the length of the hinging arm. Completing this multiplication left us with the matrix

$$R_{final} = \begin{bmatrix} -rcos\theta sin\phi \\ -rsin\theta sin\phi \\ rcos\phi \end{bmatrix}$$

These three values, when added to the x, y and z coordinates of the hinge point, give the coordinates of the end of the hinge. To compute the change in sine and cosine of each angle, we used similar equations to the previous model. The change in the sine of a given angle is equal to the cosine of the angle multiplied by the angular velocity, and the change in cosine is equal to the negative of the sine multiplied by the angular velocity. Using these equations, we represented the safety condition as

```
xHinge - r*ctheta*sphi < R &
yHinge - r*stheta*sphi < R &
zHinge + r*cphi < length
```

To define the location and state of the robot in this model, we maintained variables with the Cartesian coordinates of the hinge point, as well as the sine and cosine of each angle, and the velocities of each direction and angle. Because square roots can make the proof more complex, we opted for a more conservative norm, the L_1-norm, which is the sum of the absolute values of the x and y distances.

In an effort to simplify the model, the control was changed to use tests to abbreviate the previously bulky velocity assignments. Rotation simply assigned any value to each of *thetaVel* and *phiVel*, then checked to ensure it was non-negative and less than the respective *limit*. The simplest of the translation assignments can be seen with *zvel* below

```
{zvel :=*;
  ?(-zVelLimit <= zvel & zvel <= zVelLimit);
  ?(zHinge + r +  zvel* T < length);}
```

We assigned both x and y velocities at the same time as a part of the control. After checking to confirm the velocities assigned were within the correct range, the L_1-norm is computed using both new velocities to ensure that this movement will not move the robot within a range where either rotational motion could cause harm. The full control can be found in Appendix D.

For the continuous dynamics, the translation is the simpler case than rotation. The changes in $xHinge$, $yHinge$ and $zHinge$ are their respective velocities assigned previously. However, because we are not storing the angles themselves but rather their sines and cosines, their changes are based on product of the velocities and their mathematical derivative.

For this model we added some invariants to the continuous dynamics to avoid having to add them in as differential cuts during the proof. They included $sin^2 + cos^2 = 1$ for each angle, ensuring the robot remained on a circle around the center, and that $t >= 0$.

```
@invariant(t >= 0, cphi^2 + sphi^2 = 1, ctheta^2 + stheta^2 = 1)
```

The invariant for this model was different from the safety condition, while the safety condition calculated the actual location of the end of the instrument, the invariant just ensures the the hinge point stays at least the instrument's radius r from the walls of the larynx, as well as once again ensuring $sin^2 + cos^2 = 1$ for each angle.

The proof for this model was mostly a process of breaking the problem into simpler pieces by unfolding and with box ands. These could be separated into rotational motion, x- and y-axis motion, and z-axis motion. The two rotational motions were proved using a diRule that proved once again, $sin^2 + cos^2 = 1$ for each angle, and that the x and y positions were equal to the old ones plus the velocity multiplied by t, also ensuring that $t >= 0$. Each absolute value had to be expanded which lead to several branches. These branches could be solved with the `auto` tactic once the irrelevant information was hidden, as that information caused the prover to take too long and fail on `auto`. The complete proof for this model is in Appendix D

6 Conclusion and Future Work

As we reach the end of this project, we reflect on what we completed and what we were unable to complete in the time frame. For this project, we set the initial goal to complete four models of progressing difficulty - a two dimensional and three three-dimensional models with different movement. We were not able to complete

the stretch goal, which would have included an approximation of bending motion more complex but more accurate than the hinge. We were able to demonstrate how logical verification can be used to prove safety of a real system.

In the future, this stretch goal could be completed and proved. Additionally, a more realistic model of the larynx, at least one with changing diameters, could be used to ensure non-stabbing safety. There are also other risks to the patient, including the laser at the end of the instrument being aimed at one point in the patient for too long, causing burns. Future models could take that into account.

Outside of safety, there are other aspects of this robot that could be modeled. The goal of previous SCREAM projects has been to increase the ability of the instrument to reach points in the larynx [2]. With more complex laryngeal models, that would be helpful to determine how successful this robot is.

The farthest potential for future work would be combining what these models with the robot itself. Implementing what is learned and determined in the models into the actual instrument, and that instrument being used on patients in real medical settings [5].

Through this project, we were able to model the movement of a robot that has the potential to help patients, and show that it could be used safely, albeit in simplified context. We determined what restrictions were necessary for the robot to be unable to cause harm, and proved them through use of differential dynamic logic.

Appendices

A 2d Model and Proof

```
/* Exported from KeYmaera X v5.0 */

Theorem "2D"

Definitions
  Real length;
  Real height;

  Real xVelLimit;
  Real yVelLimit;
  Real T;
End.

ProgramVariables
  Real xScream;
  Real yScream;
  Real xVel;
  Real yVel;

  Real t;
End.

Problem
  /* INITIAL CONDITIONS */
  (height > 0 & length > 0 & yScream < height & yScream > 0 & xScream = 0 & (
    xVelLimit * T) < length & (yVelLimit * T) < height & xVelLimit > 0 & yVelLimit
    > 0)
  ->
    [
    {
      /* CONTROL */
```

```
      {
        {{?(xScream + xVelLimit * T < length); xVel := xVelLimit;}
        ++ xVel := -xVelLimit;
        ++ xVel := 0;} /* assign an x velocity */

        {{?(yScream + yVelLimit * T < height); yVel := yVelLimit;}
        ++ {?(yScream - yVelLimit * T > 0); yVel := -yVelLimit;}
        ++ yVel := 0;} /* assign a y velocity */
      }
      /* CONTINUOUS DYNAMICS */
      t := 0;
      {
        {xScream' = xVel,
          yScream' = yVel, t' = 1 & t <= T} /* evolution domain and time-trigger
    */
      }
    }*@invariant(xScream < length & yScream < height & yScream > 0) /* loop
    invariant */
  ]
  (xScream < length & yScream < height & yScream > 0) /* safety condition */
End.

Tactic "2D: Proof"
auto
End.

End.
```

B 3d-translatePolar Model and Proof

```
    /* Exported from KeYmaera X v5.0 */

Theorem "3dTranslationPolar"

Definitions
  Real length;
  Real radius;

  Real speedLimit;
  Real rVelLimit;
  Real zVelLimit;

  Real T;
End.

ProgramVariables
  Real thScream;
  Real rScream;
  Real zScream;

  Real speed;
  Real rVel;
  Real zVel;

  Real t;
End.

Problem
    /* INITIAL CONDITIONS */
  (rScream = 0 & thScream = 0  &zScream = 0 & radius > 0 & length > 0 &
  (zVelLimit * T) < length & (rVelLimit * T) < radius &
  rVelLimit > 0 & speedLimit > 0 & zVelLimit > 0)
  ->
```

```
    [
    {
      /* CONTROL */
      {
        {{?(zScream + zVelLimit * T < length); zVel := zVelLimit;}
        ++ zVel := -zVelLimit;
        ++ zVel := 0;} /* assign a z velocity */

        {speed := speedLimit;
        ++ speed := 0;} /* assign an angular velocity (change in angle)*/

        {{?(rScream + rVelLimit * T < radius); rVel := rVelLimit;}
        ++ {?(rScream - rVelLimit * T > -radius); rVel := -rVelLimit;}
        ++ rVel := 0;} /* assign a radius velocity (change in radius)*/
      }
      /* CONTINUOUS DYNAMICS */
      t := 0;
      {
        {rScream' = rVel,
          thScream' = speed, zScream' = zVel,
          t' = 1 & t <= T} /* evolution domain and time-trigger */
      }
    }*@invariant(rScream < radius & rScream > -radius & zScream < length) /* loop
    invariant */
  ]
  (rScream < radius & rScream > -radius & zScream < length) /* safety condition */
End.

Tactic "3dTranslationPolar: Proof"
pending("implyR('R==\"rScream=0&thScream=0&radius()>0&length()>0&zVelLimit()*T() <
     length()&rVelLimit()*T() < radius()&rVelLimit()>0&speedLimit()>0&zVelLimit()
    >0->[{{{?zScream+zVelLimit()*T() < length();zVel:=zVelLimit();++zVel:=-
    zVelLimit();++zVel:=0;}{speed:=speedLimit();++speed:=0;}{?rScream+rVelLimit()*T
    () < radius();rVel:=rVelLimit();++?rScream-rVelLimit()*T()>=0;rVel:=-rVelLimit
    ();++rVel:=0;}}t:=0;{rScream'=rVel,thScream'=speed,zScream'=zVel,t'=1&t<=T()
```

```
    }}*](rScream < radius()&zScream < length())\") ; loop(\"rScream < radius()&
    zScream < length()\", 'R==\"[{{{?zScream+zVelLimit()*T() < length();zVel:=
    zVelLimit();++zVel:=-zVelLimit();++zVel:=0;}{speed:=speedLimit();++speed:=0;}{?
    rScream+rVelLimit()*T() < radius();rVel:=rVelLimit();++?rScream-rVelLimit()*T()
    >=0;rVel:=-rVelLimit();++rVel:=0;}}t:=0;{rScream'=rVel,thScream'=speed,zScream
    '=zVel,t'=1&t<=T()}}*](rScream < radius()&zScream < length())\") ; <(
  \"Init\": todo,
  \"Post\": auto,
  \"Step\": auto
)");
auto
End.


End.
```

C 3d-translate Model and Proof

```
/* Exported from KeYmaera X v5.0 */

Theorem "3dTranslation"

Definitions
  Real length;
  Real radius;

  Real zVelLimit;
  Real rVelLimit;
  Real speedLimit;
  Real T;

  Real norm(Real x, Real y) = (x^2 + y^2)^(1/2);
End.

ProgramVariables
  Real xScream;
  Real yScream;
  Real zScream;

  Real speed;
  Real rVel;
  Real zVel;

  Real t;
End.

Problem
    /* INITIAL CONDITIONS */
  (xScream = 0 & yScream = 0 & zScream = 0 & radius > 0 & length > 0 & T >= 0 &
    speed > 0 & (zVelLimit * T) < length & (rVelLimit * T) < radius & zVelLimit > 0
     & rVelLimit > 0)
```

```
  ->
    [
    {
      /* CONTROL */
      {
        {{?(zScream + zVelLimit * T < length); zVel := zVelLimit;}
        ++ zVel := -zVelLimit;
        ++ zVel := 0;} /* assign a z velocity */

        {speed := *;
        {?(speed >= 0 & speed <= speedLimit);};} /* assign an angular velocity (
    change in angle)*/

        {{?(norm(xScream, yScream) + rVelLimit * T < radius); rVel := rVelLimit;}
        ++ {?(norm(xScream, yScream) - rVelLimit * T > -radius); rVel := -
    rVelLimit;}
        ++ rVel := 0;} /* assign a radius velocity (change in radius)*/
      }
      /* CONTINUOUS DYNAMICS */
      t := 0;
      {
        {xScream' = -yScream * (speed/norm(xScream, yScream)) + (xScream*rVel)/
    norm(xScream, yScream),
         yScream' = xScream * (speed/norm(xScream, yScream)) + (yScream*rVel)/norm
    (xScream, yScream),
         zScream' = zVel, t' = 1 & t <= T} /* evolution domain and time-trigger */
      }
    }*@invariant(zScream < length & norm(xScream, yScream) < radius) /* loop
    invariant */
  ]
  (zScream < length & norm(xScream, yScream) < radius) /* safety condition */
End.

Tactic "3dTranslation: Proof"
expand("norm");
```

```
implyR('R=="xScream=0&yScream=0&zScream=0&radius()>0&length()>0&T()>=0&speed>0&
    zVelLimit()*T() < length()&rVelLimit()*T() < radius()&zVelLimit()>0&rVelLimit()
    >0->[{{{?zScream+zVelLimit()*T() < length();zVel:=zVelLimit();++zVel:=-
    zVelLimit();++zVel:=0;}{speed:=*;?speed>=0&speed<=speedLimit();}{?(xScream^2+
    yScream^2)^(1/2)+rVelLimit()*T() < radius();rVel:=rVelLimit();++?(xScream^2+
    yScream^2)^(1/2)-rVelLimit()*T()>-radius();rVel:=-rVelLimit();++rVel:=0;}}t
    :=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
    xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
    +yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=zVel,t'=1&t<=T()}}*](zScream
     < length()&(xScream^2+yScream^2)^(1/2) < radius())");
loop("zScream < length()&(xScream^2+yScream^2)^(1/2) < radius()", 'R=="[{{{?
    zScream+zVelLimit()*T() < length();zVel:=zVelLimit();++zVel:=-zVelLimit();++
    zVel:=0;}{speed:=*;?speed>=0&speed<=speedLimit();}{?(xScream^2+yScream^2)^(1/2)
    +rVelLimit()*T() < radius();rVel:=rVelLimit();++?(xScream^2+yScream^2)^(1/2)-
    rVelLimit()*T()>-radius();rVel:=-rVelLimit();++rVel:=0;}}t:=0;{xScream'=-
    yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(xScream^2+yScream^2)
    ^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))+yScream*rVel/(
    xScream^2+yScream^2)^(1/2),zScream'=zVel,t'=1&t<=T()}}*](zScream < length()&(
    xScream^2+yScream^2)^(1/2) < radius())"); <(
  "Init":
     auto,
  "Post":
     auto,
  "Step":
     unfold; <(
       "[?(xScream^2+yScream^2)^(1/2)+rVelLimit()*T() < radius();rVel:=rVelLimit()
    ;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
    xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
    +yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=zVelLimit(),t'=1&t<=T()}](
    zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
         boxAnd('R=="[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+
    xScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(
    xScream^2+yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),
    zScream'=zVelLimit(),t'=1&t<=T()}](zScream < length()&(xScream^2+yScream^2)
    ^(1/2) < radius())");
```

```
    andR('R=="[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*
rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=
zVelLimit(),t'=1&t<=T()}]zScream < length()&[{xScream'=-yScream*(speed/(xScream
^2+yScream^2)^(1/2))+xScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=
xScream*(speed/(xScream^2+yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+
yScream^2)^(1/2),zScream'=zVelLimit(),t'=1&t<=T()}](xScream^2+yScream^2)^(1/2)
< radius()"); <(
      "[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*
rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=
zVelLimit(),t'=1&t<=T()}]zScream < length()":
        dC("(xScream^2+yScream^2)^(1/2)+rVelLimit()*(T()-t) < radius()", 'R=="
[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVelLimit()/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=zVelLimit(),t'=1&t<=T
()}]zScream < length()"); <(
          "Use":
            auto,
          "Show":
            auto
        ),
      "[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*
rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=
zVelLimit(),t'=1&t<=T()}](xScream^2+yScream^2)^(1/2) < radius()":
        auto
    ),
  "[?(xScream^2+yScream^2)^(1/2)+rVelLimit()*T() < radius();rVel:=rVelLimit()
;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=-zVelLimit(),t'=1&t<=T()}](
zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
    auto,
```

```
  "[?(xScream^2+yScream^2)^(1/2)+rVelLimit()*T() < radius();rVel:=rVelLimit()
;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=0,t'=1&t<=T()}](zScream <
length()&(xScream^2+yScream^2)^(1/2) < radius())":
    boxAnd('R=="[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+
xScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(
xScream^2+yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),
zScream'=0,t'=1&t<=T()}](zScream < length()&(xScream^2+yScream^2)^(1/2) <
radius())");
    andR('R=="[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*
rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=0,t
'=1&t<=T()}]zScream < length()&[{xScream'=-yScream*(speed/(xScream^2+yScream^2)
^(1/2))+xScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed
/(xScream^2+yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),
zScream'=0,t'=1&t<=T()}](xScream^2+yScream^2)^(1/2) < radius()"); <(
      "[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*
rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=0,t
'=1&t<=T()}]zScream < length()":
        ODE('R=="[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+
xScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(
xScream^2+yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),
zScream'=0,t'=1&t<=T()}]zScream < length()"),
      "[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*
rVelLimit()/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=0,t
'=1&t<=T()}](xScream^2+yScream^2)^(1/2) < radius()":
        dC("(xScream^2+yScream^2)^(1/2)+rVelLimit()*(T()-t) < radius()", 'R=="
[{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVelLimit()/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVelLimit()/(xScream^2+yScream^2)^(1/2),zScream'=0,t'=1&t<=T()}](
xScream^2+yScream^2)^(1/2) < radius()"); <(
          "Use":
```

```
            auto,
          "Show":
            auto
        )
    ),
  "[?(xScream^2+yScream^2)^(1/2)-rVelLimit()*T()>-radius();rVel:=-rVelLimit()
;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=0,t'=1&t<=T()}](zScream <
length()&(xScream^2+yScream^2)^(1/2) < radius())":
    auto,
  "[?(xScream^2+yScream^2)^(1/2)-rVelLimit()*T()>-radius();rVel:=-rVelLimit()
;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=-zVelLimit(),t'=1&t<=T()}](
zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
    auto,
  "[?(xScream^2+yScream^2)^(1/2)-rVelLimit()*T()>-radius();rVel:=-rVelLimit()
;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+xScream*rVel/(
xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+yScream^2)^(1/2))
+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=zVelLimit(),t'=1&t<=T()}](
zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
    auto,
  "[rVel:=0;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+
xScream*rVel/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=-zVelLimit
(),t'=1&t<=T()}](zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
    auto,
  "[rVel:=0;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+
xScream*rVel/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
yScream^2)^(1/2))+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=0,t'=1&t<=T
()}](zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
    auto,
  "[rVel:=0;][t:=0;{xScream'=-yScream*(speed/(xScream^2+yScream^2)^(1/2))+
xScream*rVel/(xScream^2+yScream^2)^(1/2),yScream'=xScream*(speed/(xScream^2+
```

```
    yScream^2)^(1/2))+yScream*rVel/(xScream^2+yScream^2)^(1/2),zScream'=zVelLimit()
    ,t'=1&t<=T()}](zScream < length()&(xScream^2+yScream^2)^(1/2) < radius())":
        auto
    )
)
End.


End.
```

D 3d-Hinge Model and Proof

```
    /* Exported from KeYmaera X v5.0 */

Theorem "3dHinge"

Definitions
  Real length;
  Real radius; /*radius of larynx*/

  Real r; /*length of hinge portion*/

  Real xVelLimit;
  Real yVelLimit;
  Real zVelLimit;

  Real thetaVelLimit;
  Real phiVelLimit;

  Real T;

  Real norm(Real x, Real y) = (x^2 + y^2);
  Real l1norm(Real x, Real y) = abs(x) + abs(y);
End.

ProgramVariables
  Real xHinge;
  Real yHinge;
  Real zHinge;

  Real xVel;
  Real yVel;
  Real zVel;

  Real stheta;
```

```
  Real ctheta;
  Real sphi;
  Real cphi;

  Real thetaVel; /*twist*/
  Real phiVel; /*hinge angle*/

  Real t;
End.

Problem
    /* INITIAL CONDITIONS */
  (xHinge = 0 & yHinge = 0 & zHinge = 0 &
  r > 0 & r < radius & radius > 0 & length > 0 & T >= 0 & r < length &
  (xVelLimit * T) < radius-r & (yVelLimit * T) < radius-r & (zVelLimit * T) <
    length-r &
  zVelLimit > 0 & xVelLimit > 0 & yVelLimit > 0 &
  stheta = 0 & sphi = 0 & ctheta = 1 & cphi = 1
  /*TODO: add initial conditions for angle velocities*/
  )
  ->
    [
    {
      /* CONTROL */
      {
        /*Rotation*/

        {thetaVel := *;
        {?(thetaVel >= 0 & thetaVel <= thetaVelLimit);}}; /* assign an angular
    velocity (twisting)*/
        {{phiVel := *;
        {?(phiVel >= 0 & phiVel <= phiVelLimit);}}  /*hinge motion*/
        }

        /*Translation*/
```

```
        {{zVel :=*;
         ?(-zVelLimit <= zVel & zVel <= zVelLimit);
         ?(zHinge + r +  zVel* T < length);}
         /* assign a z velocity */
        {xVel :=*; yVel:=*;
        ?(-yVelLimit <= yVel & yVel <= yVelLimit);
        ?(-xVelLimit <= xVel & xVel <= xVelLimit);
        ?(abs(xHinge + xVel * T) + abs(yHinge + yVel * T) < radius-r);
         /*assign x and y velocities*/
        } }
     }
      /* CONTINUOUS DYNAMICS */
      t := 0;
      {
        {zHinge' = zVel, xHinge' = xVel, yHinge' = yVel,
        ctheta' = -stheta*thetaVel, stheta' = ctheta*thetaVel,
        cphi' = -sphi*phiVel, sphi' = cphi*phiVel,
        t' = 1 & t <= T & cphi >= 0}@invariant(
            t >= 0,
            cphi^2 + sphi^2 = 1,
            ctheta^2 + stheta^2 = 1) /* evolution domain and time-trigger */
      }
    }*@invariant(zHinge + r < length & l1norm(xHinge, yHinge) < (radius-r) & cphi
    ^2 + sphi^2 = 1 & ctheta^2 + stheta^2 = 1) /* loop invariant */
  ]
  (xHinge - r*ctheta*sphi < radius &
   yHinge - r*stheta*sphi < radius &
   zHinge + r*cphi < length) /* safety condition */
End.

Tactic "3dHinge: Proof"
implyR('radius=="xHinge=0&yHinge=0&zHinge=0&r()>0&r() < radius()&radius()>0&length
    ()>0&T()>=0&r() < length()&xVelLimit()*T() < radius()-r()&yVelLimit()*T() <
    radius()-r()&zVelLimit()*T() < length()-r()&zVelLimit()>0&xVelLimit()>0&
```

```
    yVelLimit()>0&stheta=0&sphi=0&ctheta=1&cphi=1->[{{{thetaVel:=*;?thetaVel>=0&
    thetaVel<=thetaVelLimit();}{phiVel:=*;?phiVel>=0&phiVel<=phiVelLimit();}{zVel
    :=*;?-zVelLimit()<=zVel&zVel<=zVelLimit();?zHinge+r()+zVel*T() < length();}xVel
    :=*;yVel:=*;?-yVelLimit()<=yVel&yVel<=yVelLimit();?-xVelLimit()<=xVel&xVel<=
    xVelLimit();?abs(xHinge+xVel*T())+abs(yHinge+yVel*T()) < radius()-r();}t:=0;{
    zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*thetaVel,stheta'=ctheta*
    thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&t<=T()&cphi>=0}}*](xHinge-r
    ()*ctheta*sphi < radius()&yHinge-r()*stheta*sphi < radius()&zHinge+r()*cphi <
    length())");
loop("zHinge+r() < length()&abs(xHinge)+abs(yHinge) < radius()-r()&cphi^2+sphi
    ^2=1&ctheta^2+stheta^2=1", 'radius=="[{{{thetaVel:=*;?thetaVel>=0&thetaVel<=
    thetaVelLimit();}{phiVel:=*;?phiVel>=0&phiVel<=phiVelLimit();}{zVel:=*;?-
    zVelLimit()<=zVel&zVel<=zVelLimit();?zHinge+r()+zVel*T() < length();}xVel:=*;
    yVel:=*;?-yVelLimit()<=yVel&yVel<=yVelLimit();?-xVelLimit()<=xVel&xVel<=
    xVelLimit();?abs(xHinge+xVel*T())+abs(yHinge+yVel*T()) < radius()-r();}t:=0;{
    zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*thetaVel,stheta'=ctheta*
    thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&t<=T()&cphi>=0}}*](xHinge-r
    ()*ctheta*sphi < radius()&yHinge-r()*stheta*sphi < radius()&zHinge+r()*cphi <
    length())"); <(
  "Init":
    auto,
  "Post":
    auto,
  "Step":
    unfold;
    boxAnd('radius=="[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*
    thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&t<=T
    ()&cphi>=0}](zHinge+r() < length()&abs(xHinge)+abs(yHinge) < radius()-r()&cphi
    ^2+sphi^2=1&ctheta^2+stheta^2=1)");
    unfold; <(
      "[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*thetaVel,stheta'=
    ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&t<=T()&cphi>=0}]
    zHinge+r() < length()":
        absExp('L=="abs(xHinge)+abs(yHinge) < radius()-r()");
```

```
    dC("zHinge=old(zHinge)+zVel*t&t>=0", 'radius=="[{zHinge'=zVel,xHinge'=xVel
,yHinge'=yVel,ctheta'=-stheta*thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*
phiVel,sphi'=cphi*phiVel,t'=1&t<=T()&cphi>=0}]zHinge+r() < length()"); <(
      "Use":
        dW('radius=="[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*
thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&(t<=
T()&cphi>=0)&zHinge=zHinge_0+zVel*t&t>=0}]zHinge+r() < length()");
        auto,
      "Show":
        dIRule('radius=="[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-
stheta*thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t
'=1&t<=T()&cphi>=0}](zHinge=zHinge_0+zVel*t&t>=0)"); <(
          "dI Init":
            auto,
          "dI Step":
            auto
        )
    ),
  "[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*thetaVel,stheta'=
ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&t<=T()&cphi>=0}](abs(
xHinge)+abs(yHinge) < radius()-r()&cphi^2+sphi^2=1&ctheta^2+stheta^2=1)":
    dC("cphi^2+sphi^2=1&ctheta^2+stheta^2=1&t>=0&xHinge=old(xHinge)+xVel*t&
yHinge=old(yHinge)+yVel*t", 'radius=="[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,
ctheta'=-stheta*thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*
phiVel,t'=1&t<=T()&cphi>=0}](abs(xHinge)+abs(yHinge) < radius()-r()&cphi^2+sphi
^2=1&ctheta^2+stheta^2=1)"); <(
      "Use":
        dW('radius=="[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-stheta*
thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t'=1&(t<=
T()&cphi>=0)&cphi^2+sphi^2=1&ctheta^2+stheta^2=1&t>=0&xHinge=xHinge_0+xVel*t&
yHinge=yHinge_0+yVel*t}](abs(xHinge)+abs(yHinge) < radius()-r()&cphi^2+sphi
^2=1&ctheta^2+stheta^2=1)");
        unfold;
        expandAll;
        unfold;
```

```
          unfold;
          andR('radius=="abs__4+abs__3 < radius()-r()&cphi^2+sphi^2=1&ctheta^2+
stheta^2=1"); <(
            "abs__4+abs__3 < radius()-r()":
              orL('L=="yHinge_0+yVel*T()>=0&abs_=yHinge_0+yVel*T()|yHinge_0+yVel
*T() < 0&abs_=-(yHinge_0+yVel*T())"); <(
                "yHinge_0+yVel*T()>=0&abs_=yHinge_0+yVel*T()":
                  orL('L=="xHinge_0+xVel*T()>=0&abs__0=xHinge_0+xVel*T()|
xHinge_0+xVel*T() < 0&abs__0=-(xHinge_0+xVel*T())"); <(
                    "xHinge_0+xVel*T()>=0&abs__0=xHinge_0+xVel*T()":
                      orL('L=="yHinge_0>=0&abs__1=yHinge_0|yHinge_0 < 0&abs__1=-
yHinge_0"); <(
                        "yHinge_0>=0&abs__1=yHinge_0":
                          orL('L=="yHinge>=0&abs__3=yHinge|yHinge < 0&abs__3=-
yHinge"); <(
                            "yHinge>=0&abs__3=yHinge":
                              orL('L=="xHinge>=0&abs__4=xHinge|xHinge < 0&abs__4
=-xHinge"); <(
                                "xHinge>=0&abs__4=xHinge":
                                  orL('L=="xHinge_0>=0&abs__2=xHinge_0|xHinge_0
< 0&abs__2=-xHinge_0"); <(
                                    "xHinge_0>=0&abs__2=xHinge_0":
                                      hideL('L=="cphi>=0");
                                      hideL('L=="cphi^2+sphi^2=1");
                                      hideL('L=="ctheta^2+stheta^2=1");
                                      hideL('L=="zVelLimit()*T() < length()-r()"
);
                                      hideL('L=="zVelLimit()>0");
                                      hideL('L=="length()>0");
                                      hideL('L=="thetaVel>=0");
                                      hideL('L=="thetaVel<=thetaVelLimit()");
                                      hideL('L=="phiVel>=0");
                                      hideL('L=="phiVel<=phiVelLimit()");
                                      auto,
                                    "xHinge_0 < 0&abs__2=-xHinge_0":
```

```
                                    hideL('L=="cphi>=0");
                                    hideL('L=="cphi^2+sphi^2=1");
                                    hideL('L=="ctheta^2+stheta^2=1");
                                    hideL('L=="zVelLimit()*T() < length()-r()"
);
                                    hideL('L=="zVelLimit()>0");
                                    hideL('L=="thetaVel>=0");
                                    hideL('L=="thetaVel<=thetaVelLimit()");
                                    hideL('L=="phiVel>=0");
                                    auto
                                ),
                              "xHinge < 0&abs__4=-xHinge":
                                hideL('L=="cphi>=0");
                                hideL('L=="cphi^2+sphi^2=1");
                                hideL('L=="ctheta^2+stheta^2=1");
                                hideL('L=="length()>0");
                                hideL('L=="r() < length()");
                                hideL('L=="zVelLimit()*T() < length()-r()");
                                hideL('L=="zVelLimit()>0");
                                hideL('L=="thetaVel>=0");
                                hideL('L=="thetaVel<=thetaVelLimit()");
                                hideL('L=="phiVel>=0");
                                hideL('L=="phiVel<=phiVelLimit()");
                                hideL('L=="-zVelLimit()<=zVel");
                                hideL('L=="zVel<=zVelLimit()");
                                auto
                            ),
                          "yHinge < 0&abs__3=-yHinge":
                            hideL('L=="cphi>=0");
                            hideL('L=="cphi^2+sphi^2=1");
                            hideL('L=="ctheta^2+stheta^2=1");
                            hideL('L=="length()>0");
                            hideL('L=="r() < length()");
                            hideL('L=="zVelLimit()*T() < length()-r()");
                            hideL('L=="zVelLimit()>0");
```

```
            hideL('L=="thetaVel>=0");
            hideL('L=="thetaVel<=thetaVelLimit()");
            hideL('L=="phiVel>=0");
            hideL('L=="phiVel<=phiVelLimit()");
            hideL('L=="-zVelLimit()<=zVel");
            hideL('L=="zVel<=zVelLimit()");
            auto
        ),
      "yHinge_0 < 0&abs__1=-yHinge_0":
        hideL('L=="cphi>=0");
        hideL('L=="cphi^2+sphi^2=1");
        hideL('L=="ctheta^2+stheta^2=1");
        hideL('L=="r() < length()");
        hideL('L=="zVelLimit()*T() < length()-r()");
        hideL('L=="zVelLimit()>0");
        hideL('L=="thetaVel>=0");
        hideL('L=="thetaVel<=thetaVelLimit()");
        hideL('L=="phiVel>=0");
        hideL('L=="phiVel<=phiVelLimit()");
        hideL('L=="-zVelLimit()<=zVel");
        hideL('L=="zVel<=zVelLimit()");
        auto
    ),
  "xHinge_0+xVel*T() < 0&abs__0=-(xHinge_0+xVel*T())":
    hideL('L=="cphi>=0");
    hideL('L=="cphi^2+sphi^2=1");
    hideL('L=="ctheta^2+stheta^2=1");
    hideL('L=="length()>0");
    hideL('L=="r() < length()");
    hideL('L=="zVelLimit()*T() < length()-r()");
    hideL('L=="zVelLimit()>0");
    hideL('L=="thetaVel>=0");
    hideL('L=="thetaVel<=thetaVelLimit()");
    hideL('L=="phiVel>=0");
    hideL('L=="phiVel<=phiVelLimit()");
```

```
                        hideL('L=="-zVelLimit()<=zVel");
                        hideL('L=="zVel<=zVelLimit()");
                        auto
                    ),
                  "yHinge_0+yVel*T() < 0&abs_=-(yHinge_0+yVel*T())":
                    hideL('L=="cphi>=0");
                    hideL('L=="cphi^2+sphi^2=1");
                    hideL('L=="ctheta^2+stheta^2=1");
                    hideL('L=="length()>0");
                    hideL('L=="zVelLimit()*T() < length()-r()");
                    hideL('L=="zVelLimit()>0");
                    hideL('L=="thetaVel>=0");
                    hideL('L=="thetaVel<=thetaVelLimit()");
                    hideL('L=="phiVel>=0");
                    hideL('L=="phiVel<=phiVelLimit()");
                    hideL('L=="-zVelLimit()<=zVel");
                    hideL('L=="zVel<=zVelLimit()");
                    auto
                ),
              "cphi^2+sphi^2=1&ctheta^2+stheta^2=1":
                auto
            ),
          "Show":
            dIRule('radius=="[{zHinge'=zVel,xHinge'=xVel,yHinge'=yVel,ctheta'=-
    stheta*thetaVel,stheta'=ctheta*thetaVel,cphi'=-sphi*phiVel,sphi'=cphi*phiVel,t
    '=1&t<=T()&cphi>=0}](cphi^2+sphi^2=1&ctheta^2+stheta^2=1&t>=0&xHinge=xHinge_0+
    xVel*t&yHinge=yHinge_0+yVel*t)"); <(
              "dI Init":
                auto,
              "dI Step":
                auto
            )
        )
    )
)
```

```
End.


End.
```

References

[1] A. J. Chiluisa, N. E. Pacheco, H. S. Do, R. M. Tougas, E. V. Minch, R. Mihaleva, Y. Shen, Y. Liu, T. L. Carroll, and L. Fichera, "Light in the larynx: a miniaturized robotic optical fiber for in-office laser surgery of the vocal folds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*, pp. 427–434, IEEE, 2022.

[2] R. Mihaleva, E. Minch, R. Tougas, and J. H. Do, "SCREAM 4: Flexible steerable laser probe for office-based laryngeal interventions," tech. rep., Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA, USA, April 2022.

[3] L. Bailly, T. Cochereau, L. Orgéas, N. Henrich Bernardoni, S. Rolland du Roscoat, A. McLeer-Florin, Y. Robert, X. Laval, T. Laurencin, P. Chaffanjon, B. Fayard, and E. Boller, "3D multiscale imaging of human vocal folds using synchrotron X-ray microtomography in phase retrieval mode," *Scientific Reports*, vol. 8, p. 14003, Sept. 2018.

[4] A. Platzer, *Logical foundations of cyber-physical systems*. Springer, 2018.

[5] R. Bohrer, Y. K. Tan, S. Mitsch, M. O. Myreen, and A. Platzer, "VeriPhy: Verified controller executables from verified cyber-physical system models," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, ACM, 2018.

[6] R. Bohrer, "Chemical case studies in KeYmaera X," in *Formal Methods for Industrial Critical Systems* (J. F. Groote and M. Huisman, eds.), (Cham), pp. 103–120, Springer International Publishing, 2022.

[7] J. Liu, N. Zhan, and H. Zhao, "Computing semi-algebraic invariants for polynomial dynamical systems," in *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011* (S. Chakraborty, A. Jerraya, S. K. Baruah, and S. Fischmeister, eds.), pp. 97–106, ACM, 2011.

[8] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *CADE* (A. P. Felty and A. Middeldorp, eds.), vol. 9195 of *LNCS*, pp. 527–538, Springer, 2015.

[9] N. Fulton, S. Mitsch, R. Bohrer, and A. Platzer, "Bellerophon: Tactical theorem proving for hybrid systems," in *ITP* (M. Ayala-Rincón and C. A. Muñoz, eds.), vol. 10499 of *LNCS*, pp. 207–224, Springer, 2017.

[10] N. G. Leveson, "The Therac-25: 30 years later," *Computer*, vol. 50, no. 11, pp. 8–11, 2017.

[11] A. Platzer, "A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems," *Logical Methods in Computer Science*, vol. 8, no. 4, pp. 1–44, 2012. Special issue for selected papers from CSL'10.

[12] Y. Kouskoulas, D. W. Renshaw, A. Platzer, and P. Kazanzides, "Certifying the safe design of a virtual fixture control algorithm for a surgical robot," in *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA* (C. Belta and F. Ivancic, eds.), pp. 263–272, ACM, 2013.