FCPS Final Project Term Paper: Verifying Safety for a Hopping, Straight-Legged Bipeded Robot

David Bayani
dbayani@andrew.cmu.edu

1 Abstract:

We present a simple bipedded (i.e., two-legged) robot and verify that it is safe over the course of some non-trivial leg motion in two dimensions. Specifically, we show that our robot is safely able to hop indefinitely. To the author's knowledge, this may be one of the first applications of computer-based formal verification to ensure safety for an entire legged robot, control system and all.

2 Introduction:

Producing walking robots is a classic challenge in robotics. Walking offers robots increased mobility over terrain compared to wheeled motions, but comes at the cost of increased sophistication and potentially unsafe behavior (namely, falling and damaging the robot). Bipeded locomotion is among the most challenging form of walking motion, given the minimal base of support it provides the robot while walking.

Walking motions in robotics have been studied for decades, however most approaches focus on issues of robotics controls and efficiency of walking motions. That is, most theoretical work has been to establish safe and efficient robot's controllers given the understanding of the robot's kinematics. To the author's knowledge, all these methods of analysis have been done manually with minimal computer assistance. Efforts to verify the safety of proposed robot controllers have primarily been conducted using simulation or actual implementation of the system.

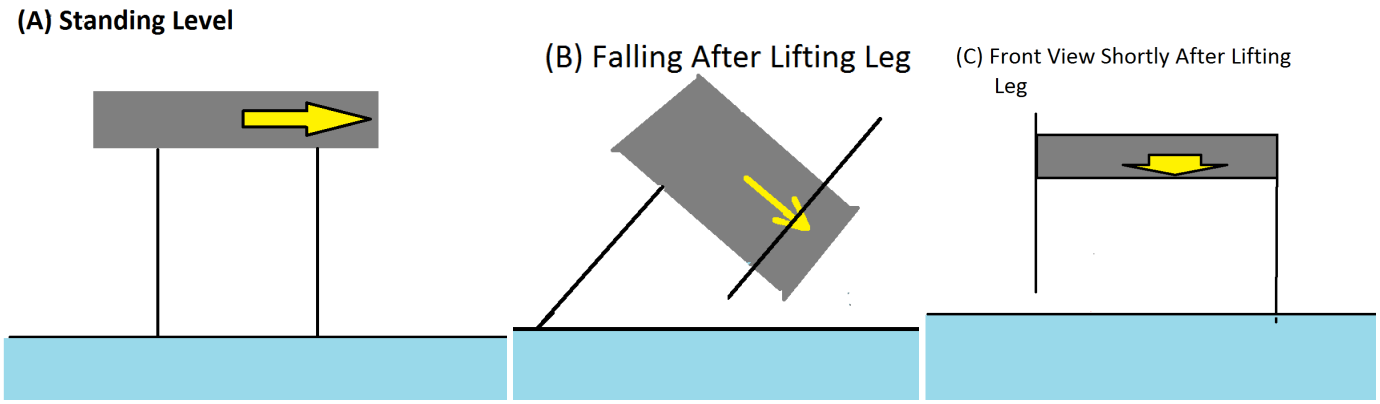
In this paper, we overview and prove safety properties for a model of a biped robot. Our analysis is able to encompass the controller, the robot, and the model of the surrounding environment giving it (in some sense) larger scope than controller-focused methods alone. To the author's knowledge, this may be the first completely formal, computer-based verification of a walking system.

3 General Description of the Robot We Aim to Model, and Post-condition We aim To Prove

In this section we overview the reasoning behind components of our model and why assumptions present may or may not be reasonable. We also describe how extensions may be made to the model presented, in order to frame our model as a component of a range of reasonable walking situations.

In order to create a model of a bipeded robot, we first consider what sort of biped it is that we want to consider; that is, we first decide what the thing-in-the-world is that we then want to model. Our robot will consist of three parts: a main body, and two independent legs. We will consider the body to be some two dimensional rectangle (oriented perpendicularly to the plane of sight) onto which the legs attach to the side. We will model the legs as one-dimensional line segments that are orthogonal to the plane of the robot's body (in two dimensions, this corresponds to the legs meeting the body at right-angles).

We consider a bipeded robot that has a simple leg motions - relative to the plane of the body, the robot's legs may move forward, backward, up or down. Further, we consider legs to be single units that go through this motion. Using legs that conform to this motion contrasts jointed-legs, which achieve up-down and forward-back motion thorough the coordination of several rotational-motions. We determined that this latter model, while more common in nature, added too many complexities to be approachable given our time-table.



4 Safety Property Postcondition to Prove

We want to ensure the following about our robot as it operates:

1. The robot's body does not tilt beyond some small, reasonable amount. This might ensure that some sensitive cargo on the robot does not become damaged. Further, this might make real-world implementations of the robot more robust by ensuring that they do not tilt beyond some range of recoverable orientations (i.e., ranges of orientation where the robot can reliably return to an up-right position via actuation). Thus, this requirement may be seen as a strict cargo-safety requirement and as a pre-emptive robot safety requirement (i.e., we consider the robot as unsafe if it enters a state that is "too risky", here being that it is tilted too far).

2. No part of the robot's main body touches the ground. This is a bare-minimum safety property, covering cases when the robot falls or drops, impacting the ground.

3. Neither the robot's feet nor the center of the robot come within minPointDistance() of each other. In real-life, this might reflect the fact that the legs and center of the robot take up some space, and thus we want to prevent them from colliding together.

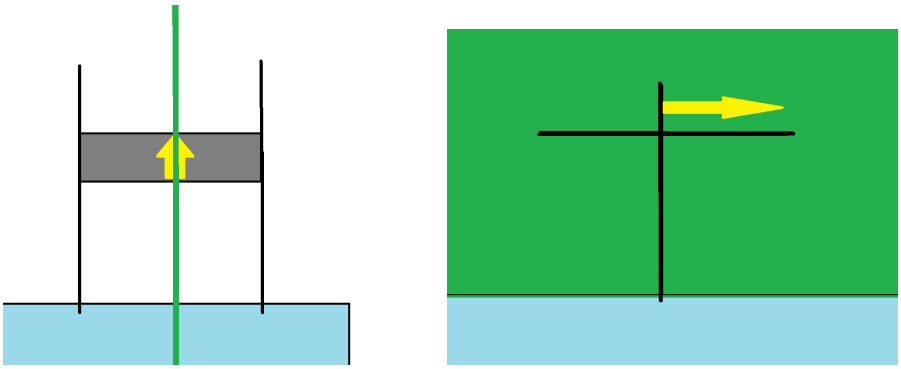
5 Assumptions, Limits of Model, and How to Extend What Will Be Shown

5.1 Restriction to Two Dimensions

We consider the motion of the robot as restriction to an x-y axis. That is, we consider the robot 's motion as projected onto a plane in-line with the side of the robot with legs. Further, we suppose that the robot cannot turn about a vertical axis in any way - that is, the same sides of the robot are always parallel or perpendicular to the plane we see. Further still, we suppose that our robot is restricted to falling only forward or backward in this plane- no matter how the robot falls, its feet will always have a constant z-coordinate.

While the above assumptions may appear strong, they are necessary in order to sensically consider the robot's walking motion in 2 dimensions; allowing any non-trivial vertical-axis rotations by the robot would require a general 3-dimensional model, as would falling sideways from the plane.

Note that if one were to extend this model to 3-dimensions, a proof of our preconditions would require covering cases isomorphic to our 2-dimensional model. Projecting the robot's motion onto a plane through the center of the body and between the legs results in a viewing-plane where the robot appears to make no vertical-axis rotations (or more generally, rotations whose axises are lines in the line or vision). We conjecture that given such a plane and a reasonable bound on the amount of tilt off this plane by the robot, one could reduce most of the remaining safety proof to two dimensions. Thus, though our two dimensional assumptions may seem like an over simplification of a three-dimensional machine, they may provide a solid ground-work for an approach handle higher dimensions. Regardless of the truth of this conjecture, it is clear that handling two dimensions well is necessary to achieve safety proofs of 3 or more dimensions.



On the left, how the plane we are using to view the robot in two dimensions crosses the body in three dimensions. On the right, the plane we are actually using to view the robot.

5.2 Distribution of Mass in the Robot

We assume that the mass of the robot is evenly distributed around the center of the robot. We thus model the robot's mass as being a point mass located at the center of the robot's body. This is a less than ideal assumption, since it fails to model how moving the robot's legs (the legs having mass and thus weight) effect stability. Despite this assumption, our model may still be realistic in cases where the robot's body is far more massive than its legs; our particular model of robot, where all movement mechanisms are located at the body, seems like a reasonable good candidate for such a situation.

An alternative assumption we strongly considered was using three-masses on the robot - one at the center of the body and one at the center of each leg. While this would be a good step toward a more realistic model, we found that accounting for changing centers of mass in the dynamics would likely make verification significantly less approachable for the time-line of this project. We are considering exploring the possibility further for the May 5th presentation, however.

5.3 Environment the robot is walking in - a flat plane

We consider our robot to be walking in a flat plane. This is the natural environment to first consider when verifying the safety of some walking motion, as few general-purpose walking motions exhibit safer-behavior in rough and/or tilted planes.

Generalizing beyond the flatness requirement is straight forward. Currently, we implicitly use a vector normal with the plane in order to determine the direction of gravity in respect to the plane; in other words, when the robot falls, it follows a vector that is vertical, which happens to be orthogonal to the plan of motion. To consider walking in planes with some tilt, the gravity vector would be given an orientation in respect to the plane representing the ground. This would cause the robot to fall in some sideways fashion, which is equivalent to falling straight down on a tilted plane. We suggest this extension - altering a gravity vector as opposed to explicitly changing the equation for the ground - because the latter model makes detecting when and how a foot contacts the ground far less trivial and would require throwing out much of the model framework we develop here.

5.4 Interaction With Ground - Friction

We model the ground as having infinite friction. When one of the robot's feet touch the ground and some normal force is applied by the plane, any lateral motion by the foot stops. We emphasize that normal force must be applied in order for lateral motion to halt. This is consistent with usual, basic models of friction, where friction is calculated as $\mu n = \mu m|g|$ - the normal force times some friction constant μ, which here we take as ∞. It is debatable whether, here, if $n = 0$ then μn should be infinity or zero - here, we take it to be zero.

Once work is established for approaches with no lateral motion for feet on the ground, it may be generalized to model finite frictions by including appropriate cut-offs between when friction stops lateral motion and when friction slows lateral motion. We decided that in order to make a

proof approachable in the time-frame, we would model only one of these forms of friction-related dynamics, since modeling both would require more than double the work.

5.5 Event Triggered

As a start, we considered our robot to be event-triggered. That is, we allow control-reevaluation at least as often as when feet reached their limit or motion. For example, reevaluation occurs whenever one of the feet can no longer be moved up. Signaling when the limits of controllable motion have been reached is a realistic feedback that is implemented in many systems. Ideally, modeling this would also be coupled with some time-delay; the signal is sent to the controller, by the controller takes some epsilon time to process it. In order to simplify what was already a complex (and computational-resource exhausting) model, we did not include time delays in processing event-signals generated by limits of motion being reached.

5.6 Finite Length Legs

We consider the robot to have finite-length legs. While simple conceptually, finite-length legs add non-trivial complexity to the model, as they requires that we model a limit to the ability to extend or move the feet in some direction. For a reasonable approximation to reality, we model the feet as hitting a limit of their motion then stopping - an instantaneous change from having positive speed to being stopped. Notice that since the legs have no mass in this model, they exert no force on impact and, due to lack of inertia, can be started or stopped instantaneously.

We describe exactly how we model this discontinuous cut-off momentarily.

6 Formal Modeling

6.1 Mathematical model of robot's body and feet

We model the robot's body and legs using two constants and sixteen to twenty variables depending on the situation. We use two constants to specify the length of the robot's legs (legLength()) and the size of the robot's body (bodySize()). We use two variables to specify a unit-magnitude "forward vector", a vector that lies on the plane of the robot's body and indicates which direction is forward; this is shown as the yellow arrow in this paper's pictures. We will use F_x and F_y to refer to the x and y components of this forward vector, respectively. We use two variables to indicate the current center of the robot's body, which will be represented by C_x and C_y for the x and y coordinates respectively.

The remaining twelve to sixteen variables are used to specify the location of the robot's leg, their acceleration , and their velocities. When a leg is not touching the ground, the position of a leg is represented by two proportions that indicate how far forward along the body the leg is (θ_{F1} for leg1, θ_{F2} for leg2), and how high the leg is lifted (θ_{Up1} for leg1, θ_{Up2} for leg2). Both of these proportions have a corresponding velocity and acceleration; the controller determines the acceleration. The velocities and accelerations respectively are $vel\theta_{F1}$, $acc\theta_{F1}$, $vel\theta_{Up1}$, and $acc\theta_{Up1}$ for leg1 and $vel\theta_{F2}$, $acc\theta_{F2}$, $vel\theta_{Up2}$, and $acc\theta_{Up2}$ for leg 2. When a leg is touching the ground (and is carrying a load so that the surface applies a normal force to the leg), we use two variables to indicate the x and y coordinates of the foot, which will remain unchanged over the course of the dynamics that follow it due to friction and gravity - we call these xInitialLeg1 and yInitialLeg1 for leg 1 and xIntialLeg2 and yInitialLeg2 for leg 2. It is because of these initial-leg-position variables that we gave a range on the number of variables relevant - not all the dynamics use the same set of variables that indicate initial foot positions.

Given the above variables, the below equations give the position of the robot's feet in respect to the origin of the plane:

$$xFootLeg1() = (2 * \theta_{F1} - 1) * bodySize() * F_x + (1 - \theta_{Up1}) * legLength() * F_y + C_x$$
$$yFootLeg1() = (2 * \theta_{F1} - 1) * bodySize() * F_y + (1 - \theta_{Up1}) * legLength() * (-F_x) + C_y$$
$$xFootLeg2() = (2 * \theta_{F2} - 1) * bodySize() * F_x + (1 - \theta_{Up2}) * legLength() * F_y + C_x$$
$$yFootLeg2() = (2 * \theta_{F2} - 1) * bodySize() * F_y + (1 - \theta_{Up2}) * legLength() * (-F_x) + C_y$$

Formulas used to determine the position of the feet of the legs

---

To make sense of the above equations, first consider where the robot's feet are in respect to the the center of the robot. Leg 1 is located along the line orthogonal to the plane of the robot's body and passing through the point $(2 * \theta_{F1} - 1) * bodySize() * (F_x, F_y)$ (using vector operations); in other words, along the body, leg 1 is $(2 * \theta_{F1} - 1)$ of the way forward. The factor of two comes from our convention of considering the center of the robot to be halfway forward ($\theta_F = \frac{1}{2}$), which would result in leg1 aligning with the center of the robot. The foot of leg 1 is located $(1 - \theta_{Up1}) * legLength() * (F_y, -F_x)$ down from where leg 1 attaches to the robot's body; the proportion of leg 1 below the robot is $(1 - \theta_{Up1})$. Notice that the vector $(F_y, -F_x)$ is normal to the body of the robot and orthogonal to the forward vector (F_x, F_y). Thus, in respect to the center of

the robot, the foot of leg 1 is located at

$$(2 * \theta_{F1} - 1) * bodySize() * (F_x, F_y) + (1 - \theta_{Up1}) * legLength() * (F_y, -F_x).$$

In order to adjust for the center of the plane, we simply add the position of the center of the robot to yield:

$$(xFootLeg1(),\ yFootLeg1()) =$$
$$(2 * \theta_{F1} - 1) * bodySize() * (F_x, F_y) + (1 - \theta_{Up1}) * legLength() * (F_y, -F_x) + (C_x, C_y)$$

Similar reasoning applies to leg 2.

6.2 Modeling Discontinuous Changes in Motion - Friction and Limits of Leg Extension

We model friction and limits of leg extension by hybrid-program tests and ODE domain constraints.

Prior to executing the dynamics, we ensure that if the legs are at the boundary of their allowed motion, their velocity and acceleration in the direction of the boundary are zero. In order to guarantee that legs do not move past their boundaries during the execution of the dynamics, we include the leg-boundary conditions as part of the ODE domain constrains - namely, we require that $0 \leq \theta_{F1} \leq 1,\ 0 \leq \theta_{F2} \leq 1,\ 0 \leq \theta_{Up1} \leq 1,$ and $0 \leq \theta_{Up2} \leq 1$ hold over the evolution of the ODEs. After our ODEs execute, a loop surrounding the entirety of our model returns control to the commands that force leg velocities and leg accelerations to respect the boundaries.

6.3 Modeling Violating the Postconditions

Our explicit expression for the postcondition is :

$$\begin{gathered}(F_y \leq maxTilt())\& \\ ((yBodyCorner1() > 0)\&(yBodyCorner2() > 0)\&(yPositionBody > 0))\& \\ (((xFootLeg1() - C_x)^2 + (yFootLeg1() - C_y)^2) > minPointDistance())\& \\ (((xFootLeg2() - C_x)^2 + (yFootLeg2() - C_y)^2) > minPointDistance())\& \\ (((xFootLeg1() - xFootLeg2())^2 + (yFootLeg1() - yFootLeg2())^2) > minPointDistance())\end{gathered}$$

Postcondition to Our Model

Here, $F_y \leq maxTilt()$ is the condition stating that the robot should not tilt too far. One may show that it is equivalent to requiring the dot product of the plane's norm and the norm of the robot's body to be at least $1 - maxTilt()$; this is useful if one wants to extend our model to planes that are not flat.

In the section of our model covering the cases of the dynamics, we include a branch that allows execution to end if, on that iteration of the loop, some of the postconditions were are violated (or were on the border of being violated). This allows the postcondition to be tested in any case were it is violated on entry to the loop. We have copied the content of this branch in our model below:

$$?(\ \ ((yBodyCorner1() \leq 0)|(yBodyCorner2() \leq 0)|(yPositionBody \leq 0))|$$
$$(((xFootLeg1() - C_x)^2 + (yFootLeg1() - C_y)^2) \leq minPointDistance())|$$
$$(((xFootLeg2() - C_x)^2 + (yFootLeg2() - C_y)^2) \leq minPointDistance())|$$
$$(((xFootLeg1() - xFootLeg2())^2 + (yFootLeg1() - yFootLeg2())^2) \leq minPointDistance())\ \ )$$

Hybrid Program Statement 1 : Conditions present in the branch of our model that ensure that strong violations of the postconditions are detected.

In addition to the above, we put in every ODE domain constraint the following formula:

$$((yBodyCorner1() \geq 0)\&(yBodyCorner2() \geq 0)\&(yPositionBody \geq 0))\&$$
$$(((xFootLeg1() - C_x)^2 + (yFootLeg1() - C_y)^2) \geq minPointDistance())\&$$
$$(((xFootLeg2() - C_x)^2 + (yFootLeg2() - C_y)^2) \geq minPointDistance())\&$$
$$(((xFootLeg1() - xFootLeg2())^2 + (yFootLeg1() - yFootLeg2())^2) \geq minPointDistance())$$

Formula 3: subformula of ODE domain constraints

Note that if it were not for including Hybrid Program Statement 1, we would risk creating a model that had no transitions in certain cases where the post-conditions are explicitly violated by including the above ODE domain constraints. In those cases, the program without Hybrid Program Statement 1 would vacuously satisfy the postcondition despite explicitly violating it in reality.

Formula 3 is a subformula of the postcondition that has been weakened to include its boundary; in general, it is necessary for ODE domain constraints to contain the boundaries of formulas over variable in order to avoid mathematical curiosities such as control that is infinitely dense in time and trying to perform calculus on inappropriate topologies. Beyond the purposes of reasonable modeling, including the boundary prevents us from creating a model that is safe by construction; that is, one where unsafe behavior is prevented by trivially filtering out any states that violate it. Here, by the properties of ODEs in dL, if an ODE can reach an unsafe state, then it must be able to first reach a state on the boundary of being safe and unsafe. Since we are using the box modality to run our program, in order for our model to satisfy the post-condition, every run of the program must satisfy the postcondition. Thus, if there is some run of the ODE that reaches the boundary of the safety condition, we will not satisfy safety conditions and thus fail the postcondition. By doing this, we may assume over the course of ODE executions that Formula 3 remains true, and we may do so without breaking our model or making it useless. The conditions in Formula 3 are taken advantage of in the dynamics in order to preserve well-definedness.

6.4 Dynamics of Having No Legs on the Ground

When the robot has no feet on the ground (say because it picked both of them up quickly), the body of the robot enters free fall. During free fall, the feet of the robot move freely until they contact the ground. The domain constraint on our ODE requires that both feet have non-negative height over the entirety of fall. This constraint implies that free-fall dynamics will continue no longer than when the first foot contacts the ground, because falling straight down past this time would imply the foot would have negative height. This behavior is exactly what we want, because the moment a foot touches the ground, different dynamics other than free-fall take effect; these alternative dynamics are described in the following subsections.

As one can see by referring to our ODE for free-fall, we do not change C_x during free fall, which ignores momentum. We decided that this simplification was acceptable while proving safety, since

our safety conditions are invariant under choices of C_x. Beyond C_x not changing during the course of this ODE, notice that the orientation of the robot does not change during free fall - this is expected given that the only mass present in the system is at the center of the robot, so there is no change to the center of mass nor is there any rotational momentum around the center of mass.

6.5 Dynamics of Having One Leg on the Ground

When one leg is on the ground (we will call it legDown), the body of the robot falls while pivoting on legDown. This follows a rotation about the point where legDown contacts the ground. The rate at which the robot rotates around legDown is determined by the torque and moment of inertia about this center of rotation (we ignore lateral momentum), both of which are a exclusively a function of the distribution of mass for a falling system. In our case, the only mass is located at the center of the robot, allowing use to use equations for point-mass calculations of moment of inertia and torque due to gravity:

Moment of inertia: $I = mr^2 = m((C_x - xInitialLegDown)^2 + (C_y - yInitialLegDown)^2)$

In the below calculation of Torque, ζ is the angle the the gravity pulls in respect to the lever arm of the robot falling. It is shown as the red arc next to the green arrow that represent the pull of gravity below. Elementary geometry shows that the angles highlighted in red are equal, and thus $(sin(\zeta))(r) = (sin(\zeta))(\sqrt{(C_x - xInitialLegDown)^2 + (C_y - yInitialLegDown)^2})$ reduces to be the length (and direction) of the yellow line, which is just $C_x - xInitialLegDown$

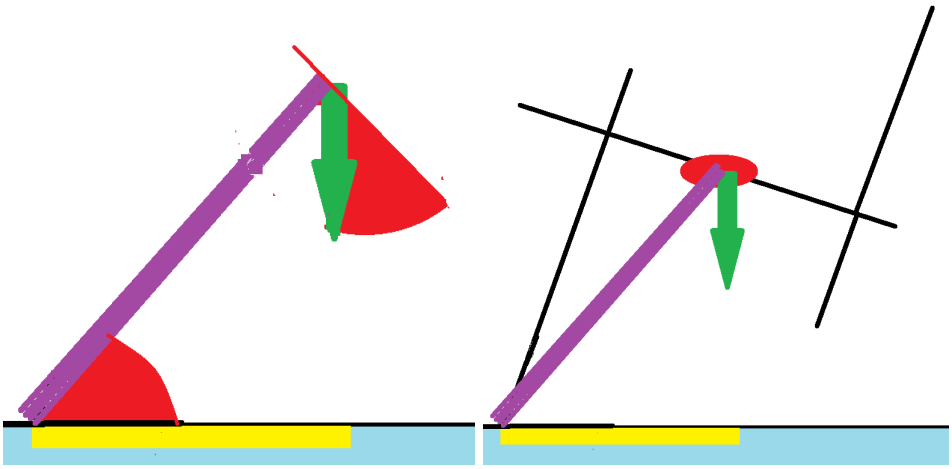


Illustration of the geometry behind the lever-arm calculations (left) present in the dynamics of a robot balancing on one foot (right).

Torque: $T = -gm(sin(\zeta))(r) = -gm(C_x - xInitialLegDown)$

Using this and the fact that $T = I\alpha$ (α being angular acceleration), we get that:
$\alpha = \frac{T}{I} =$
$\frac{-gm(C_x - xInitialLegDown)}{m((C_x - xInitialLegDown)^2 + (C_y - yInitialLegDown)^2)} =$

$\frac{-g(C_x - xInitialLegDown)}{((C_x - xInitialLegDown)^2 + (C_y - yInitialLegDown)^2)}$
Notice here that for well-definedness, it is critical that we require Formula 3 (described earlier) to hold in the ODE domain constraints. Formula 3 ensures that the center of the robot and legDown do not meet, ensuring that the denominator of the above is non-zero.

Given the rotational acceleration α, we are able to find how quickly F_x and F_y change. Again, (F_x, F_y) is a unit-vector reflecting the orientation of the robot's body, and thus correspond to $(cos(\beta)\ ,\ sin(\beta))$ where β is the angle of the robot's body in respect to the plane. We know that the angle of the robot changes at rate ω, the rotational velocity, which gives that:
$F_x' = (cos(\beta))' = (\beta)'(-sin(\beta)) = -\omega(sin(\beta)) = -\omega F_y$
$F_y' = (sin(\beta))' = (\beta)'(cos(\beta)) = \omega(cos(\beta)) = \omega F_y$
Where $\omega' = \alpha$ and the initial value of ω is a function of previous dynamics.

As the robot falls, it is still able to move its feet. However, as a result of infinite friction and supporting weight of the robot, legDown is in a fixed position (so long as it is not lifted with vertical acceleration greater than g). Thus, as the robot falls, moving legDown results in the center of the robot moving. By using the equations initially provided for the x and y coordinates of the foot of a robot, we can solve for the center of the robot. The center must be known while the ODE evolves, however, due to α depending on the location of the center as the robot falls. Thus, we have the following:

$(C_x)' =$
$(\ xFootLegDown() - ((2 * \theta_{FDown} - 1) * bodySize() * F_x + (1 - \theta_{UpDown}) * legLength() * F_y)\ )' =$
$2 * vel\theta_{FDown} * F_x - vel\theta_{UpDown} * F_y + (2 * \theta_{FDown} - 1) * (-\omega * F_y) + (1 - \theta_{UpDown}) * (\omega * F_x)$

$(C_y)' =$
$(\ yFootLegDown() - (\ (2 * \theta_{FDown} + -1) * bodySize() * F_y + (1 - \theta_{UpDown}) * legLength() * (-F_x)\ )\ )' =$
$2 * vel\theta_{FDown} * F_y + vel\theta_{UpDown} * F_x - (1 - \theta_{UpDown}) * (-\omega * F_y) + (2 * \theta FDown - 1) * (\omega * F_x)$

Note that as long as these dynamics are in effect, $xFootLegDown()$ and $yFootLegDown()$ are constants equal to xInitialLegDown and yInitialLegDown , respectively.

These dynamics apply so long as one foot remains on the ground, namely as long as legDown has its foot on the ground. This remains true so long as
$acc\theta_{UpDown} * |F_x| * legLength() \leq g$
in other words, as long as the foot is not lifted up faster than the rate of gravity pulls the robot down; this condition is part of the ODE domain constraint.

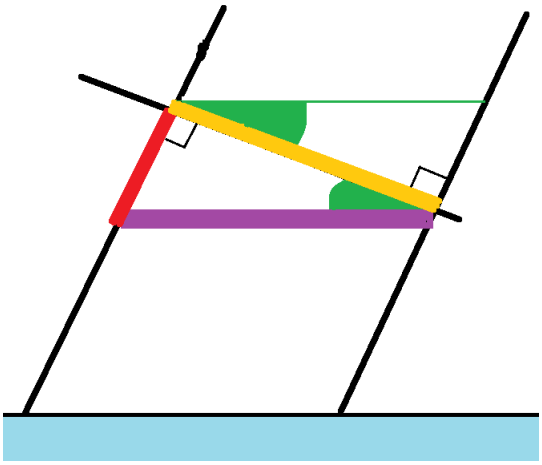
6.6 Dynamics of Having Two Legs on the Ground and the Center of Mass Fully Supported

When two legs are firmly on the ground and fully supported, the robot (in our 2-dimensional projection) stands stably still. This occurs when both feet are on the ground and on either side of the center of mass, summarized by:

$$(xInitialLeg1 - C_x) * (xInitialLeg2 - C_x) <= 0$$

Notice the use of $xInitialLeg1$ and $xInitialLeg2$ above as opposed to $xFootLeg1$ or $xFootLeg2$ since the legs, being on the ground **and** both supporting the weight of the robot, experience infinite friction and cannot move along the ground.

Again, notice that the legs cannot move along the ground and that the robot is stable. Recalling that we require the legs to meet the body at right angles , we see that the position and orientation of the robot's body is completely determined by the position and motion of the feet.



illustrating the feet fixed on the ground and the effect the that difference in the portion of the legs below the robot has on the orientation of the robot's body.

In particular, basic geometry tells us that the two green angles must be congruent. Notice that if β is the upper of the two green angles, then $sin(\beta) = F_y$ and $cos(\beta) = F_x$. Further notice that the length of the red line is equal to $|(length\ of\ leg\ 1\ under\ robot) - (length\ of\ leg\ 2\ under\ robot)| = |legLength()(1 - \theta_{Up1}) - legLength()(1 - \theta_{Up2})| = legLength()|\theta_{Up2} - \theta_{Up1}|$. Since the legs are two parallel lines and since the purple line is parallel with the ground, it must be that purple line is the same length as the distance between the feet, which is just $|xInitialLeg1 - xInitialLeg2|$. Finally, we see that the yellow line is simply the difference between how far the two legs have been moved forward, which is just

$$|(distance\ from\ the\ center\ of\ the\ robot\ leg\ 1) - (distance\ from\ the\ center\ of\ the\ robot\ leg\ 2)| = |(2\theta_{F1} - 1)bodySize() - (2\theta_{F2} - 1)bodySize()| = 2(bodySize())|\theta_{F1} - \theta_{F2}|.$$

From the above, then, we see that

$$F_y = sin(\beta) = \frac{signed\ length\ red\ line}{signed\ length\ purple\ line} = \frac{legLength()(\theta_{Up2} - \theta_{Up1})}{xInitialLeg2 - xInitialLeg1}$$
$$F_x = cos(\beta) = \frac{signed\ length\ yellow\ line}{signed\ length\ purple\ line} = \frac{2(bodySize())(\theta_{F2} - \theta_{F1})}{xInitialLeg2 - xInitialLeg1}$$

In the above, we refer to "singed length" - by this we simply choose a convention of always subtracting the positions of foot 2 from the positions of foot 1 in order to determine direction - this consistent consideration ensures that we properly extract orientation in the same way that is does when calculating "rise over run" from grade-school algebra

For well definedness, it is critical that $xInitialLeg2 - xInitialLeg1 \neq 0$. This is not an issue, however, since we require in the ODE domain constraint that
$(((xFootLeg1() - xFootLeg2())^2 + (yFootLeg1() - yFootLeg2())^2) \geq minPointDistance())$
If both feet were on the ground and at the same x-coordinate, this equation would be violated, since $minPointDistance() > 0$.

As one may have already noticed from our discussion of the line-lengths, this is an over-determined system; given information about any two of the red, yellow, or purple line, the geometry of this

situation allows us to determine what the third member must be. This over-determinacy reflects the fact that some combinations of leg position, etc, are impossible if we model correctly. It is this necessary to prevent the model from allowing the feet to move in a way which violates these constraints, which can be done sufficiently by ensuring:
$(xInitialLeg2 - xInitialLeg1)^2 = legLength()^2(\theta_{Up2} - \theta_{Up1})^2 + 4(bodySize())^2(\theta_{F1} - \theta_{F2})^2$
which follows from the Pythagorean theorem. We put this equation in the ODE domain constraint.

Using the above, we have determined enough information to solve of the center of the robot:
$C_x = xInitialLeg1 - ((2 * \theta_{F1} - 1) * bodySize() * F_x + (1 - \theta_{Up1}) * legLength() * F_y)$
$C_y = yInitialLeg1 - (\ (2 * \theta_{F1} + -1) * bodySize() * F_y + (1 - \theta_{Up1}) * legLength() * (-F_x)\ )$
Similarly, we could have used leg 2 to perform this calculation. Note that none of the dynamics depend on the center of the body nor the angle of the body - our dynamics here simply move the feet. Thus, to simplify our model, we do not include differential calculations for the rate of change of the center nor the rate of change of the orientation, but simply reference their closed-form solutions in the ODE domain constraints and discretely assign to the relevant variables after the ODE finishes evolving.

Similar to what was done in the case of one foot on the ground, we limit the evolution of the ODE in this case to be for as long as neither of the legs are lifted at a rate faster than how gravity would cause the robot to fall. Unlike in the one-legged case, however, the rate that the robot falls is not simply g here - it is the tangential acceleration at that position when rotating around the other foot. For foot 1, this is determined by

$$((xInitialLeg1 - xInitialLeg2) * ((g() * (xInitialLeg2 - C_x))/((xInitialLeg2 - C_x)^2 + (yInitialLeg2 - C_y)^2)) \geq acc\theta_{Up1} * |F_x| * legLength()$$

The left hand-side is the rotational acceleration (α, calculated before) times the radius of rotation (the distance between the feet, $xInitialLeg1 - xInitialLeg2$, with the order of subtraction being crucial), giving the tangential acceleration at the foot of leg 1. The right hand side is the upward acceleration of foot 1, as determined previously.

7 Leg Motion and Proof of Its Safety

To demonstrate our model and prove some interesting properties, we proved our generic postcondition for a robot that exhibits non-trivial leg motion.

7.1 Leg Motion : Hop and Spread

We considered an event-triggered controller that did the following:

- If both legs are fully extended down and touching the ground, lift them with an upward acceleration $2(\frac{g}{legLength()})$. Move the left leg (we will suppose it was leg 2) backward with an acceleration of 1 and the right-leg (we will suppose it was leg 1) forward with an acceleration of 1
- If both legs are fully extended up, lower them at a rate of $\frac{g}{legLength()}$ and do not change their forward accelerations
- If both legs are touching the ground and are not fully extended down, lower both legs with an acceleration of 1.
- If none of the above conditions are met, do not alter the current setting for the accelerations of moving the robot's legs.

We started the robot with both legs fully extended down touching the ground. Leg 2 was positioned $\frac{1}{4}$ along the body, while leg 1 was positioned $\frac{3}{4}$ along the body. The desired effect of the controller is to get the robot to do multiple hop-and-splits while staying safe; that is, we wanted the robot to hop up and down, and while doing so, spread its legs apart. This is a non-trivial leg motion, a making a guarantee of safety interesting.

7.2 Proof overview

Our model consists of a main loop whose content can be considered in three segments (in order): (1) the controller, (2) discrete aspects of physics that enforce finiteness of the legs and friction , (3) the continous aspects of physics, such as moving legs and falling. The least trivial aspects of proving that the model satisfies the postconditions is proving that the continuous physics and that iterations of our model (i.e., running the loop) are safe; unlike the controller or the discrete aspects of physics, neither the loop nor all of our ODEs can be translated into first-order formulas (in the case of our ODEs, this is because they do not have solutions in the language over real-closed fields). Given KeYmaera X's ability to run quantifier elimination on first-order formulas over reals-closed fields, the components of our model that can be translated into first-order formulas generally require little manual attention. Thus, in the sections that follow, we overview how we proved (or plan to prove) the sections of our model that are not necessarily machine-solvable.

7.3 Proving the Loop

In order to prove the loop that surrounds all our model's content, it was necessary to use a loop invariant, since the variables bound by our model effect the truth-value of our postcondition. The content of our loop invariant contained the following specifications:

- Bounds and general information about constant quantities, such as $g > 0$. This is required since the inductive step of the loop-invariant proof removes the initial conditions, which otherwise specify this content.
- Bounds and general information on variables that change over the course of the model's execution, but that are ensured by virtue of the model itself (i.e., are true regardless of what the controller does). This includes requirements made by the finiteness of the legs ($\theta_{F1},\ \theta_{F2},\ \theta_{Up1},\ \theta_{Up2} \in [0,1]$) , requirements that the body, corners of the body, and feet of the robot not be below ground (that their height be non-negative) and that the forward-vector have unit-magnitude.
- Behavior of the model coming as a consequence of control decisions. This includes the following:
  - The both feet have the same proportions of upward position, velocity, and acceleration ($\theta_{Up1} = \theta_{Up2},\ vel\theta_{Up1} = vel\theta_{Up2},\ acc\theta_{Up1} = acc\theta_{Up2}$). If the robot is completely level, this corresponds to both legs having the same height, vertical velocity, and vertical acceleration.
  - That the angular velocity of the robot's body be zero ($\omega = 0$) and that the robot's body be completely level ($F_x = 1,\ F_y = 0$).
  - That leg 1 is $\frac{3}{4}$ of the way along the body or further, and leg 2 is $\frac{1}{4}$ the way along the body or less ($\theta_{F1} \geq \frac{3}{4}$ and $\theta_{F2} \leq \frac{1}{4}$). Further, the leg 1 only moves further right, and leg 2 only moves further left, as reflected in their accelerations and velocities ($vel\theta_{F1} \geq 0,\ vel\theta_{F2} \leq 0,\ acc\theta_{F1} \geq 0, acc\theta_{F2} \leq 0$).
  - Bounds on how far the body of the robot will ever fall, $C_y \geq \frac{legLength()}{8}$.
  - Other specific relationships between accelerations and velocities that are a result of control decisions. We refer the reader to the section of our loop invariant (provided in the model delivered) labeled "strengthened loop invariant necessary for proof" for many of them.

7.4 Proof for Falling Dynamics

The falling dynamics we considered featured an ODE that specified that the height of the robot's body decreased during free-fall and that the legs of the robot were allowed to move freely. This ODE is solvable and has a solution that is expressible in our logical language - in fact, the solutions are second degree polynomials in respect to time. We solve the ODE to reduce it to a first-order formula, then run quantifier elimination on the result.

7.5 Proof of Two-Foot-on-Ground Dynamics

As explained previously, due to a combination of gravity and infinite ground friction, the dynamics involved for when the robot has two feet on the ground are completely determined by the position and motion of the legs. The ODE describing the motion of the legs is solvable and expressible in our logical language, as it was in the case of our falling dynamics. We solved the ODE as we did before, and an application of quantifier elimination achieves a proof of this case.

7.6 Proof of One-Foot-on-Ground Dynamics

Unlike the dynamics for free-fall or when both feet were on the ground, the dynamics for when one foot is on the ground do not have solutions expressible in our logical language (and perhaps lacks a general solution for all terms - we did not investigate solutions to the ODE further than noting that they would not help us). The inexpressibility in our language of any potential solution can be seen in the differential equation for F_x and F_y, where $F_x = (-\omega)(F_y)$ and $F_y = (\omega)(F_x)$; this equation specifies rotational dynamics, which requires the transcendental functions *sin* and *cosin* to express. Transcendental functions are not part of our logical language, which works over real-closed fields. To tackle this ODE, then, it was necessary to use a differential invariant, particularly one strong enough to imply the loop invariant so that the inductive step of proving the loop would go through.

It is actually the case, in our model, that if the controller and dynamics behave as expected, then the one-leg-on-ground dynamics should only ever run for zero time units. This is because we require that, over the entire duration of our dynamics, no part of the body of the robot nor any part of the legs goes below the ground - no part has negative height. We expect that a robot that follows the controls and physics specified would have both legs at the same height at all times. Thus, if we are following the one-leg-on-ground dynamics, it must be that one of the feet has a height of zero, implying that the other leg's foot has a height of zero as well. Further, as expressed in the proposed loop invariant, we expect neither leg to be at the center of the robot nor to approach the center of the robot (the leg on the left only goes left and the leg on the right only goes right). Thus, if must be that the weight of the robot is not fully supported over either of the legs, implying that the robot would fall and rotate around whichever leg is on the ground. We see, then, that if both feet already have height zero and the robot falls for more than zero time-units, then one of the robot's feet must go below ground, having negative height. Since our ODE domain constraints prevent feet from having negative height, it must be that the ODE runs for zero time units if at all.

In order to prove this formally, one must prove that each of the key elements described above are true over the course of the ODE's execution. To do this, we first confirmed that our understanding of where the robot's legs were relative to the center of the robot matched our understanding: (1) that both feet were extended below the robot by the same amount and at the same velocity and accelerations (2) that the legs stay strictly on their respective, opposite sides of the robot (3) that both legs moved away from the center of the robot at the same non-negative rate. Proving (3) may have been unnecessary, but regardless of its necessity, we found it helpful to align the formal model with more of our intuitions. To clarify slightly, points (1), (2), and (3) proved properties of the leg's proportions in respect to the body (θ_{Up1}, $\theta_{F1}, vel\theta_{Up1}$, etc) , **not** the coordinates of the robot's components. In order to speak about the coordinates of the robot's body components, it would be necessary to gleam information about the orientation of the robot at a given time; if the legs were at opposite ends of the robot, the actual coordinates of the legs' locations are different when the robot is standing up-right versus if it has fallen sideways. From here, we showed three more collections of formulas simultaneously: (4) that the length of the lever arm (the length from the foot being pivoted on to the center of the robot) is non-zero and has negative sign (the sign indicates direction of rotation) (5) that the leg not being pivoted on had height zero (6) all the conditions specified in the loop invariant.

7.7 Proving that the Postconditions were not Violated Prior to the ODEs

As we described when overviewing our robot's dynamics, in order to assume stronger ODE domain constraints without weakening the guarantees of our model, we included a case that would result in the postcondition being violated when none of the ODE domain constraints would be satisfied. Proving that the model behaved properly in this case was trivial - we decomposed the hybrid program containing the relevant test and were able to prove that case of the loop's inductive step by appeal to quantifier elimination. This showed that, given the proposed loop invariant held (the inductive hypothesis), the postcondition would definitely hold prior to our dynamics.

7.8 Sanity Check Performed on Initial Conditions and Loop Invariant

Due to the size of our model's preconditions and loop invariant (particularly after by-hand expansion, as explained in the following section), a quick sanity check was performed to ensure that a contradiction was not inadvertently assumed. A contradiction might accidentally be assumed if minor errors occur when inputting the assumptions, which happens with greater likelyhood when the specification of the assumptions are long. Since our preconditions and loop invariants were first-order formulas over real-closed fields, we used quantifier elimination to check whether our assumptions could be used to deduce false. Since quantifier elimination is a sound and complete proof procedure over real-closed fields, false would be deducible if and only if our assumptions contained a (subtle) contradiction. Much to our relief, quantifier elimination found that false was not provable in any of the cases we tried.

7.9 Challenges to Proving Our Model and Current Status of Proof

Due to the relatively large size of our model, numerous machine / technical issues occurred that delayed attempts to conduct proofs. Eventually, we settled on an approach that seemed to bypass these technical issues and allowed us to proceed in showing the crux of a safety argument. Specifically, to reduce the size of the model considered, we manually separated the model into the components required in a proof:

- We showed that the preconditions to our model satisfied the proposed loop invariant
- We showed that the proposed loop invariant implies the desired postcondition
- We attempted to prove the non-trivial aspects of the loop inductive step by manually decomposing the hybrid program statements prior to the dynamics and then proving each case of the dynamics separately. This is exactly what a fully automated approach to the proof would have done.

We have provided proposed proofs for all parts of our dynamics and for proving the components of the loop mentioned above. We are currently waiting for several of the proofs of our dynamics to complete or report failure. These results should be available for our presentations on May 5th.