Lab 3: Robots on Racetracks
15-424/15-624/15-824 Foundations of Cyber-Physical Systems
Course TA: Brandon Bohrer (bbohrer@cs)

Betabot Due Date: 3/03/16, worth 20 points
Checkpoint Due Date: 3/7/16, worth 10 points
Veribot Due Date: 3/09/16, worth 70 points

In this lab we will practice two essential techniques:

1. Reasoning about ODEs with differential invariants when solving them is difficult
2. Reasoning about arithmetic with lower bounds when exact solutions are difficult

By putting these techniques together, we will be able to verify a robot on a racetrack!

1 Practice with Differential Invariants, Diff Cuts, and Weakening in KeYmaera X

To get some practice with differential invariants, prove the following $\mathtt{d}\mathcal{L}$ formula using KeYmaera X. You **must not** use master/auto! You also may not use the ODE Solve rule. Instead, use the Differential Invariant, Differential Cut, and Differential Weakening rules. Save your resulting model and proof archive as `L3Q1.kyx` and `L3Q1.kya` for BetaBot/VeriBot, respectively.

$$\vdash (x \geq 0 \land y \geq 0 \land z \geq 0) \rightarrow [x' = y, y' = z, z' = x^2]\, x \geq 0$$

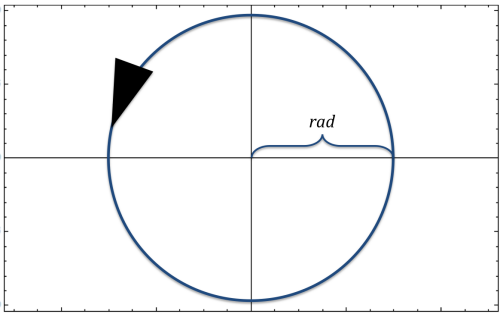
Since we have already given you the model, you don't need to hand in anything for this problem on BetaBot, but the proof is due on the Checkpoint date, not just the VeriBot date.

In the remaining questions, you will design controllers to drive your robots around a circular racetrack centered at the origin and with radius *rad*. The properties you prove are safety properties (e.g. stay on the track and don't hit the obstacle), but you should design your controller to satisfy a basic efficiency condition as well (i.e. it should eventually come near the obstacle). So, for example, a robot that reaches the obstacle slower than necessary is fine, but a robot that doesn't reach the obstacle because it just sits still is useless.

Because using polar coordinates would make the lab trivial (and cause you to fail Lab 4), you are **required to use Cartesian coordinates**, which also generalize better to the next lab.
In this question, your robot must be able to drive *counter-clockwise* on the track **without leaving it**. In other words, the robot must always be on the circle with radius *rad* centered at the origin. The racetrack is completely empty, so the robot may choose to travel at any speed. Because you are using Cartesian coordinates, velocity here means *linear* velocity / ground speed, not angular velocity.

2 Keep The Robot on the Racetrack

Design a hybrid program to keep your robot on the track. This initial hybrid program has no controller! Your robot should move at an arbitrary, constant speed with arbitrary, constant radius. The goal of this



problem is to focus on modeling physics correctly to ensure your robot stays on the track (and with correct velocity). Create a formula which, if proved, would guarantee that your robot never drifts off the circle. You will need to determine appropriate initial conditions to make this property provable; however, try to prove the strongest property you can by making the initial conditions as general as possible.

a) Use KeYmaera X to prove that your controller keeps the robot on the track using differential invariants. Submit your model and proof archive as `L3Q2.kyx` and `L3Q2.kya` for BetaBot/VeriBot, respectively.

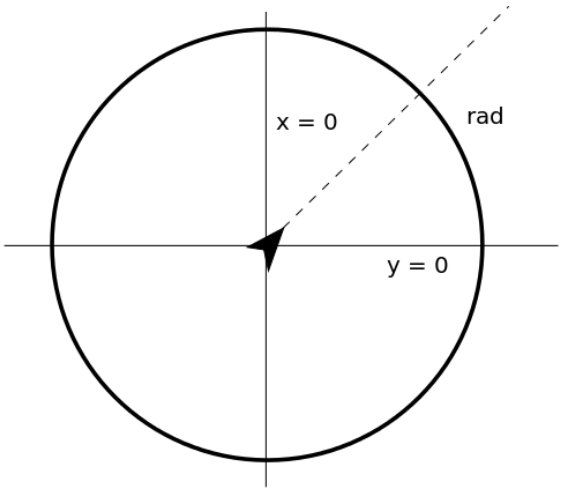
b) **Question:** Solve the differential equation you used to model the physical dynamics of the system. Could you rewrite an equivalent hybrid program using this solution and the assignment operator instead of a continuous evolution? Why or why not? Submit a *brief* answer in `L3.txt`. Also *briefly* explain the intuition behind your differential equations (**BetaBots**).

3 Keep An Accelerating Robot on the Loop with a Loop

Augment your model so the robot can change its speed and use KeYmaera X to prove that your controller keeps the robot on the track. Your HP must include a loop so that the acceleration controller may execute more than once. Submit your model and proof as `L3Q3.kyx` (BetaBot) and `L3Q3.kya` (VeriBot). Because you don't have any obstacles to avoid yet, you can set the acceleration to whatever you want. The goal is just to make sure you can still model the physics correctly when you add acceleration. This should not be much harder than the previous problem, but is easy to get wrong if you are not careful.

4 Straight-line Static Obstacle Avoidance Near Volcanos

Let's take a brief detour from the racetrack in order to prepare for the next problem. In this problem, the robot starts at the origin, in the infield of the racetrack. The racetrack is now made of lava, so if your robot touches it, they die. (for convenience, touching the lava exactly is ok so long as you don't go past it). Design a controller that computes + sets the braking acceleration such that the robot stops before it hits the racetrack, then prove that it is safe. If you wish, you may assume the robot is capable of arbitrarily high braking acceleration for this problem, mainly because this problem will still practice the proof techniques we want you to practice either way. But adding a bound on acceleration might fall in line with the following problems better. Note there is no loop necessary for this problem, it's sufficient to let the ODE run once.



a) Design a hybrid program to model the control and movement of the robot (in `L3Q4.kyx`, for BetaBot). Create a formula which, if proved, would guarantee that the robot never reaches the lava under appropriate starting conditions. If it helps for your proof, you can assume for simplicity that the robot moves toward Quadrant I ($v_x, v_y \geq 0$), but you may not assume it is moving along the x or y axis.

   Depending on your proof approach, your proof might be equally simple without this assumption.

b) Prove this property by showing that the distance between the robot and origin is always less than the radius of the lava. You can either try to prove this directly, or using a lower bound, which might be easier depending on the approach you took. If you did the proof using a lower bound, then in `L3.txt`, write down an expression θ which is always a lower bound on the Eulidean distance, but is also a reasonable notion of distance (e.g. satisfies triangle inequality, is never negative). If you didn't need a lower bound, say so in `L3.txt` (for the sake of clarity).

c) Use KeYmaera X to prove your safety property. If using a lower bound, you can the distance from the origin is always less than θ, then show θ is a lower bound on `rad` to complete the argument. Submit the .kyx file you proved and the proof as `L3Q4.kyx` and `L3Q4.kya` for BetaBot/VeriBot, respectively.

5 Racetrack Debris (curved-path static obstacle avoidance)

In this problem you will combine everything you learned in the previous problems to verify safety of a robot driving on a circular track. A frustrated student left their computer on the track while waiting for KeYmaera X to find a safety proof. Your goal is to develop a robot that moves counter-clockwise around the track but stops before running over the student's computer (because you feel their pain). Of course, you can't start your robot at 100 mph just feet behind the computer and expect the system to still be safe, so you may define some initial conditions on the distance between your robot and the relative position of the computer. If these initial conditions are satisfied, your controller must be able to bring your robot to a stop before it hits the computer. The bounds on acceleration and braking for your controlled robot are $A > 0$ and $-B < 0$. *Your controller must be time-triggered.*

The computer is at the leftmost position on the track because that's closest to the nearest place where the student could get coffee. Following the previous problems, **you should start with a very conservative controller with very loose lower bounds**. You should only attempt to make the controller less conservative once it works, and you **should not** expect to make your bounds exact.

a) Design a hybrid program to keep your robot on the track and stop before the position of the obstacle. Create a formula which, if proved, would guarantee that your controller satisfies these safety properties.

b) Use KeYmaera X to prove the safety properties. Submit the .kyx file you proved and the proof archive as `L3Q5.kyx` for BetaBot and `L3Q5.kya` for VeriBot.

c) In `L3.txt`, briefly explain your control decision **(Betabots)** and any tricks used in your proof **(Veribots)**.

d) **(Veribots)** Suppose you've proven this controller to be safe. Based on your model and your proof, should we believe the same controller would work on a circle in which the obstacle was dropped at an arbitrary spot? In `L3.txt` write whether the controller ought to work and briefly argue why or why not.

6 Racetrack Debris (Extra credit, Veribots only)

Meta-arguments like in Q5.d aren't formally proved in KeYmaera X. For extra credit (*only for Veribots*), don't assume that the student's computer was dropped at the leftmost position of the racetrack. The befuddled student might have dropped it anywhere on the racetrack!

Submit `L3Q6.kyx` and `L3Q6.kya` for the Veribots deadline. In `L3.txt` (VeriBots), briefly explain how proving this generalisation was different from the previous question.

7 Submission Checklist

Submit a zip file on autolab. The Makefile distributed with the assignment can make this for you. **Remember to include a file named andrewids.txt with your names in the following format, otherwise I will not know that you both submitted:**

```
aplatzer bbohrer
```

Additionally, when working in groups, *only one person should submit, otherwise I will end up grading you twice.*

- `andrewids.txt` (All submissions, only if working in pairs).
- `L3Q1.kyx` (Actually we gave you this file so you don't really have to submit it ever.)
- `L3Q1.kya` (Checkpoint only)
- `L3Q2.kyx` (BetaBot only)
- `L3Q2.kya` (VeriBot only)
- `L3Q3.kyx` (BetaBot only)
- `L3Q3.kya` (VeriBot only)
- `L3Q4.kyx` (BetaBot only)

- `L3Q4.kya` (VeriBot only)
- `L3Q5.kyx` (BetaBot only)
- `L3Q5.kya` (VeriBot only)
- `L3Q6.kyx` (VeriBot only, extra credit)
- `L3Q6.kya` (VeriBot only, extra credit)
- `L3.txt` (All submissions)