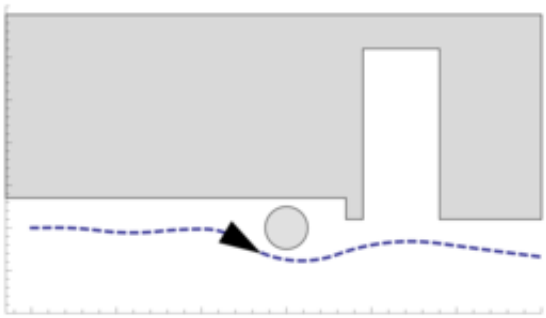
Lab 4: Static and Dynamic Obstacles
15-424/15-624/15-824 Logical Foundations of Cyber-Physical Systems
TA: Katherine Cordwell (kcordwel@cs.cmu.edu)

Betabot Due Date: Extended to Saturday, November 2nd, **11:59 PM with NO late days**, worth 20 points
Veribot Due Date: Extended to Saturday, November 9th, 11:59PM (2 late days, max 6 per semester), worth 80 points

Lab Resources: `https://lfcps.org/course/lfcps19/lab4.zip`

Getting Started

In this lab you will design a controller which may move freely (non-deterministically) on a plane, rather than just around a fixed circular track. The robot should be able to move anywhere, but it must always avoid a single, static (not moving) obstacle.



Modeling the free motion of a robot in 2D can be thought of as an extension of Lab 3, but now with discrete control of steering as well as acceleration. When steering is changed, you might think of it as the robot switching from one circular track to another, but the new track must be tangent to the old one at the position of the robot so that the robot can maintain its position, direction and velocity. The new track's radius and whether the track is on the left or the right of the robot (in other words, whether the robot is traveling clockwise or counter-clockwise around the track) may change at each discrete transition. To help you visualize what is happening, we have created this YouTube video: `http://youtu.be/C_pyRQT6bBw`

Note: There are many design choices for you to make in this assignment. The template files contains suggestive variable names that may be helpful but you are free to choose/add your own variables. If you do so, please add comments explaining what they do.

1 Modeling Robot Movement

For the first portion of this lab, model the robot's movement *without considering obstacles or time-triggers*. The robot must always be on *some* circular track, but it may choose which circular track to follow at every control point. For simplicity, you may assume that straight line motion for your robot is approximated by choosing a sufficiently large track radius.

Constraints

- The robot should have a non-deterministic controller. That is, if there are multiple valid control decisions, then your model should strive to allow as many of them as possible, because this makes the model more general.

- You **may not** have discrete changes in the position, direction, or linear velocity of the robot (or any other variable which would implicitly cause such a discrete change), because that would not be faithful to physics.
- The bounds on acceleration and braking for your controlled robot are $A > 0$ and $-B < 0$ respectively.
- You should discretely control the track radius and acceleration of the robot. You should allow switching between clockwise and counter-clockwise motion by switching between positive and negative track radius.
- The robot should always have a non-zero turning radius (i.e., it should not spin in place).

Suggestions and hints

- Using *guarded* non-deterministic assignments like `var:=*; ?(P(var))` will be very useful in this problem. You may use it to choose the radius *trackr* of the conceptual "track" that the robot is moving on.
- As with earlier labs, make sure that your robot always has some control choice, i.e., the guards are exhaustive.
- You may find the functions `max(x,y)` and `abs(x)` of use in this assignment. While these exist in KeYmaera X, they do not always perform reliably, especially with `QE`. We recommend that you get rid of these functions in your proofs if you get stuck.

Submission Instructions

1. (Betabot). Fill in the missing parts of the provided template to model the situation described above. Write a corresponding safety property (i.e., the robot stays on the chosen track for every control cycle). Submit this file as L4Q1.kyx.
2. (Veribot). Use KeYmaera X to prove that your model is safe. Export the proof as L4Q1.kyx.

2 Static Obstacle and Time-triggering

In this part of the lab you will add a static obstacle and a time-triggered controller to your model.

Constraints

- All constraints from Part 1 still apply.
- The robot should still have a non-deterministic controller (i.e., it should be able to drive anywhere that does not cause it to come too close to the obstacle).
- The controller should be time-triggered.
- Your robot now has a shape, which can be over-approximated by a circle of radius $r > 0$.
- The obstacle can also be over-approximated by a circle of radius $rogr > 0$.

Suggestions and Hints

- All the hints from Part 1 are still relevant.
- We recommend that you start with a model that uses braking to enforce safety but leaves steering entirely non-deterministic. In other words, your safety argument should not care about your steering decisions at all.

  As with Lab 3, this is a tradeoff between efficiency and ease of proof. A controller that uses both braking and steering to provide safety would be more efficient, but much harder to verify due to the difficulty of reasoning about curved paths.

  Only move on to more complex controllers once you are confident of the proof for this (simpler) controller.
- You may also simplify your model by using an infinitesimal point (x, y) to represent the position of your robot and another infinitesimal point $(rogx, rogy)$ for the position of the obstacle **provided** you ensure that the two never get within a symbolic *buffer* distance of each other.

Submission Instructions

1. (Betabot). Fill in the missing parts of the provided template to model the hybrid program above and verify that it is safe. Submit this file as L4Q2.kyx.
2. (Veribot). Use KeYmaera X to prove that your hybrid program is safe. Export the proof as L4Q2.kyx.

3 Dynamic Obstacle

This exercise is identical to the above, except that the obstacle is now a *rogue-bot* which drives around with constant velocity rather than a stationary obstacle. Here are some extra rules for this problem:

- Your robot should never turn with a turning radius of magnitude less than $minr > 0$ (i.e., it cannot turn too sharply).
- The rogue-bot drives at constant speed $rogv \geq 0$ in a fixed direction.

Submission Instructions

- (Veribot). Find a controller that you can convince yourselves is safe and model it. You only have to submit the **model** in the L4Q3.kyx file. **You do not need a proof, unless you want extra credit!**
- (Veribot). **Bonus:** Prove it and export the proof as L4Q3.kyx. **Indicate that you have proved the model in the header.**
- (Veribot). **Question:** What if your robot has a top speed that is less than $rogv$? Or, if instead of moving on an unbounded 2D plane your robot is constrained to a bounded space? In these cases, it can be very difficult to ensure that your robot never collides with the obstacle. For example, if the rogue bot was driving at high speed directly towards your robot, then your robot might not even have enough time to get out of the way!

  In lab4.txt, propose and discuss a few possible safety properties for these two new scenarios which, if proved, would guarantee that your robot still exhibits reasonable behavior even in the presence of unreasonable obstacles. What are the pros and cons of each of your proposed safety properties?

4 Submission Checklist

If you are working with a partner, then you must submit a file called andrewids.txt containing both of your Andrew IDs. To make the grading infrastructure happy, please put them on a single line separated by a space, e.g.:

```
kcordwel aplatzer
```

Make sure you submit this file when working in a group. Otherwise, one of you will not get credit because there is no record of your submission. Additionally, ONLY one of you should submit on Autolab so that we do not end up grading your submissions twice.

For all submissions, remember to check the Autograder's output on Autolab to ensure that your files were submitted in the right format, parse correctly, etc. If you are working in a group, please also ensure that the Autograder correctly reports your Andrew IDs in its output, e.g.:

```
==> Group Andrew IDs: kcordwel, aplatzer
```

Use the provided templates, and *do not forget to fill in the section at the top.* It gives us important information when grading your submission!

1. **Initial submission (Betabot)**. The Betabot zip file should contain:
   - `L4Q1.kyx`
   - `L4Q2.kyx`
   - `andrewids.txt` (only if working in pairs)
2. **Final submission (Veribot)**. The Veribot zip file should contain:
   - `L4Q1.kyx` (proof required)
   - `L4Q2.kyx` (proof required)
   - `L4Q3.kyx` (proof is extra credit)
   - `lab4.txt`
   - `andrewids.txt` (only if working in pairs).