(define signal follow-person
  (behavior see-person?
    (rt-vector (* person-heading rotate gain)
      (let ((range-error (- person-range desired-range)))
        (+ (* range-error range-p-gain)
          (* (integral range-error) range-i-gain))))))

The robot successfully tracks you and follows you. However, when you stop suddenly, it keeps driving and rear-ends you. Explain each of the following in one sentence of less than 20 words. Assume that the tracking system is working perfectly and that the control loop is operating at high frequency so that the problem is not lag-related.

A. Why does the robot collide with you even though it contains a feedback loop to keep you at a specified distance? (10 points)

B. How can you modify the system to prevent this failure mode without adding any additional sensors or sensory processing? You need not write any code for this (although you may if you want). It is sufficient to explain what needs to be done in one English sentence. (10 points)

C. Explain how your proposed modifications will hurt other aspects of the robots' performance (if they will at all) (10 points).
Consider the following freespace follower:

```
(define-signal follow-freespace
  (behavior #t
    (rt-vector (* (- left-distance right-distance) rotate-gain)
      (let ((distance-error (- distance-ahead goal-distance)))
        (cond ((< distance-error 5)
          (* distance-error low-translate-gain))
          ((and (> distance-error 5) (< distance-error 20))
            (* distance-error medium-translate-gain))
          (else
            maximum-translate-velocity)))))
```
**Question 3 (20 points)**

Suppose you try to write a wandering behavior as follows:

```scheme
(define-signal wedged?
  (and (< distance-ahead 20)
       (= left-distance right-distance)))

(define-signal unwedge
  (behavior wedged?
    (rt-vector 200 0)))

(define-signal wander
  (behavior-or unwedge follow-freespace))
```

This program seems to work well except that when it drives up to a wall, it goes postal: it jerks to one side, then turns back, then jerks again, on and on until you kill it. Assume `follow-freespace` behavior has been fixed, so the error must be in `unwedge` or in the way the two behaviors are combined. Explain in one sentence of 20 words or less:

A. Why it does this stupid thing (hint: what's the name of this phenomenon?) (10 points)

B. How to fix it (10 points)
Consider the freespace follower from question 2 and assume that the translation control bug has been fixed. Again, you find that it mostly works well, but that it doesn't corner well. When it comes to a corner, it:

- Starts turning slowly, but then overshoots the corner
- Comes to the far wall and has to stop driving forward
- However, it keeps turning until it's finally unblocked (see figure, dashed line).

You try to fix this by raising the rotate gain. While this causes it to corner well, it oscillates when going down a straight corridor (see figure, dotted line). Explain how to make it corner well without oscillating badly or getting blocked (solid line in the figure). Limit your explanation to one sentence of less than 20 words. If you prefer, you may give the appropriate code for the rotation signal.
You're building a robot for the design competition. This year's event is the Faculty Safari, in which you build robots to roam the halls of the CS department, hunting and subduing faculty members.

Your robot is an RWI base like the ones you've been using in class that runs a sonar-based wandering program. You've added a turret built from a Futaba model airplane servo motor. Futabas are integrated servo-motor packages that contain a motor, a drive-train, a position sensor, and a P controller implemented using an analog circuit. The Futaba takes an analog signal that specifies the direction the servo should point in. The analog P controller generates the appropriate drive signals for the motor to turn it to the right position. Futabas are cheap, fairly strong, and quite fast. However, they have bad drive trains. Their cheap plastic gear boxes are noisy and inefficient. They are also inaccurate because their bearings contain enough friction to make them stop short of their set-points. For example, when moving from 0 degrees to 90 degrees, a Futaba might stop at 85 degrees because the error is small enough that the output of the analog P-controller is insufficient to overcome friction.

Your robot has a turret-mounted directional heat sensor that reports the direction (in the turret’s coordinate system) of the strongest heat source, and a turret-mounted air tazer that fires a pair of high-voltage electrodes at the rapidly fleeing faculty member.

You use the wandering program above to drive around in search for faculty, and you add to it the following behavior to shoot them:

```scheme
(define-signal zap-ian
  (behavior #t
    (turret-motor-vector
      ;; How to turn
      (integrate (if see-heat? (* heat-bearing rotate-gain) 0))

      ;; When to fire
      (and see-heat? (= heat-bearing 0)))))
```
Assume that:

- **turret-motor-vector** takes a rotational velocity and a fire control signal (a Boolean), respectively, as arguments.
- **heat-bearing** gives the direction of the heat source, assuming there is one, relative to the turret's current direction. Thus, if the heat source is dead-ahead, it reads zero, otherwise it returns a positive or negative number giving the number of degrees error in the current turret orientation. If there is no heat source, it returns a random number.

The integrator is used because the turret's Futaba needs a position signal, but the heat-bearing signal only gives a position error. In theory, the rotate-gain could be 1 and the turret would always point to the target within one cycle of the control loop. However, there is enough lag in the system that this leads to oscillation. So a small rotate gain is used to ensure stability.

Answer each of the following questions in one sentence of 20 words or less.

A. It almost never seems to shoot, even when you stand there like a sitting duck. Why? (10 points)

B. When a person walks into the area, the turret seems to notice them, turn more or less toward them, and stop. However, after a little while, the turret will suddenly "twitch" to one side of the target and stop. After a while, it twitches again to the other side. It seems to continue this behavior indefinitely: it sits still for a while, then suddenly moves to the opposite side, over and over again. Explain why. What is this phenomenon called? (10 points)

C. Explain how to stop the twitching. You may give code, or a single English sentence. (10 points)
Question 6

Give the equivalent C, C++, or Scheme code for the following program. You need not give the same code that the GRL compiler would generate, simply a program that has identical behavior.

(define-signal wedged?
  (> (true-time (< center-distance 30))
      500))

(define-signal unwedge
  (behavior wedged?
    (rt-vector 2000 0)))

(define-signal follow-freespace
  (behavior #t
    (rt-vector (* (- left-distance right-distance) rotate-gain)
                (* (- center-distance stopping-distance) translate-distance))))

(define-signal base-controller
  (drive-base unwedge follow-freespace))

Note that we have not given definitions for most of the signals here. If the definition isn’t given for a signal, just assume that its current value is always in a global variable with the same name as the signal. So if writing in C, the current value of the signal center-distance would always magically be in the C variable centerDistance. (Pretend, for example, that there’s a separate thread computing and recomputing it asynchronously).

You should also assume that:

- The translational and rotational velocities of the base are set by calling the procedures set-translate-velocity! and set-rotate-velocity!. (If you’re writing C code, then they would be set_translate_velocity and set_translate_velocity).
- You can get the current time (in milliseconds) by calling the procedure ms-clock (or ms_clock, if you’re writing C).