Simulation of Constant-Distance Robot Following using Radio Proximity Beacons

Jeremy Elson
University of Southern California
CSCI547 – Final Project Report

December 10, 1999

Abstract

This paper reports the first step in an attempt to examine the relationship between simulators and reality. Whereas some simulation efforts attempt to model physical phenomena in the greatest possible detail, others have been successful in only adding pseudo-random noise sufficient to swamp inaccuracies that come from an intentionally simplistic model. In this vein, we develop a crude, noisy radio model with the hope that simulations of robot behaviors using it can be later validated against reality. The behavior is simple: a follower that attempts to keep a robot at a constant distance from its leader.

1 Introduction

This paper reports the first step in an attempt to examine the relationship between simulators and reality. Whereas some simulation efforts attempt to model physical phenomena in the greatest possible detail, others have been successful by adding pseudo-random noise sufficient to swamp inaccuracies that come from an intentionally simplistic model. For example, the *ns* network simulator attempts very detailed simulation of network links, queues, and radio propagation. In contrast, the *arena* robot simulator simulates sensors that are physically dependent on physical phenomena by using a trivial model and adding random noise. *arena* works well because the noise component of the simulated sensor readings swamps errors due to insufficiencies of the model.

It is not yet entirely clear why such a detailed simulation is required for *ns*, but not for *arena*. One possible explanation is that network controllers work in environments where "sensor error" is not expected—the environment is sensed with perfect fidelity on an extremely short timescale. Robot controllers, on the other hand, have inherent to their design the assumption that the environment is not perceivable without error. They make control decisions on a much longer timescale, and always with the assumption that the sensors may be lying. *arena*'s method works because controllers robust enough to work in the face of *arena*'s random error are also likely to work when faced with the vagaries of the real world.

Whatever the reason for the difference in simulation philosophy, the dichotomy has become more prominent due to projects that simulate simultaneously in both domains; e.g., robots that are both physical actuators and entities on a network. The purpose of this study is to determine whether or not a network simulated with *arena*-like simplicity can be used when the network is operating in the domain of a robot controller.

2 The Experiment

For this study, we selected a simple behavior of "constant-distance following." The simulation was performed using the *arena* robot simulator, capable of simulating a number of Pioneer-like robots in a user-specified world. In the simulation, two robots repeatedly circle the central corridor shown in Figure 1. One robot is marked as a leader and tries to circle the corridor while periodically emitting a radio beacon. The other robot (the follower) traverses the same known path through the environment while using the radio beacon to keep a constant distance from the leader.

The design of the controllers was decomposed into two subtasks. First, each robot must be capable of autonomously and reliably navigating infinite circles around the corridor of the environment in Figure 1. Next, the follower must also attempt to infer its distance from the leader using the radio beacon in order to follow it at a constant distance. The design of each of these tasks is described below.

3 Corridor Traversal

Our corridor traversal algorithm assumes that the robot is traveling clockwise around a cyclic corridor. It can not detect such a corridor; it assumes that its initial position and direction are somewhere on its desired path. The shape of the corridor is assumed to be convex (modulo slight deviations such as closed doors), but has no other constraints. Although our sample environment in Figure 1 is a simple rectangular corridor, our controller will work equally well in buildings whose corners are not rectilinear.

Our controller for implementing this task is purely reactive: it contains no state and, indeed, does very little processing at all. It makes a continuous stream of instantaneous control decisions based on its current sonar state. Each of the Pioneer's seven sonars are first classified into one of three states: NEAR (an object sensed at 2m or less); MID (an object sensed between 2-5m);

Figure 1: The environment used in the simulation: a model of the Salvatori Computer Science Center.

and FAR (an object is sensed at 5m or more).

The essence of the controller is shown in Figure 2, where the desired steady-state of each of the robots' seven front- and side-looking sonars are shown. To maintain this state, the controller has two independent, reactive control pathways: one for speed control and one for direction control. Although implemented as a simple series of "if–then" clauses, these control elements are written in the "style" of classic behavior-based controllers. That is, they each have multiple and potentially conflicting goals that have a defined priority in their ability to affect the final control decision.

3.1 Turning Behaviors

The algorithm that governs turning is shown in Program Fragment 1. The first turning rule states that we should follow the corridor after the right-hand wall has suddenly disappeared—i.e., is no longer visible to the right side-looking sonar. In this case, where we assume we need to turn quickly to make a corner, we make a sharp (4 radian/sec) right turn.

In the case when we see a collision in one of our two left-front sensors, we make a shallow turn to the right. We also make this shallow right turn if we have drifted far enough away from the right wall that Sonar 5 can no longer see it. Note that this is less drastic action than the

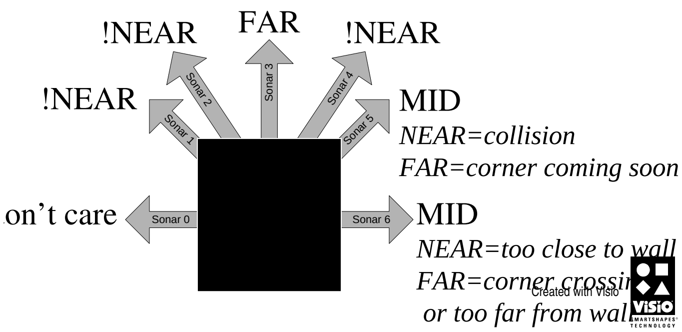


Figure 2: Desired steady state of all seven sonar sensors for proper direction control in a clockwise traversal of a cyclic corridor. The sonars are numbered clockwise from 0 to 6.

Program Fragment 1 Turn controller for clockwise travel around a cyclic corridor.

```
if (state[6] == FAR) {
  turnRate = 4.0;  // Sharp right turn to follow the wall
} else if (state[5] == FAR || state[1] == NEAR || state[2] == NEAR) {
  turnRate = 2.0;  // Shallow right to avoid a left collision,
                   // if we're drifting away from the wall
} else if (state[3] != FAR || state[4] == NEAR || state[5] == NEAR
           || state[6] == NEAR) {
  turnRate = -2.0;  // Shallow left for right or dead on collision,
}                   // or getting too close to wall
```

sharp turn taken if Sonar 6 loses sight of the wall.

Finally, we make a shallow turn to the left if we see a collision with something dead ahead or to our right (Sonars 3, 4 and 5, respectively), or if Sonar 6 indicates we are getting too close to the right wall. The dead-ahead looking sonar assumes that any detected object is a collision. In contrast, the diagonal sonars only assume a collision in the NEAR state; they normally pick up the corridor walls in the MID state, even when the robot is correctly positioned.

3.2 Speed Behaviors

Program Fragment 2 shows the code used for controlling the robot's speed. In the absence of any special conditions, it commands the robot to travel at speed `nominalSpeed`. This speed is normally 7.0 m/s for a leader robot, and is variable for a following robot, as will be described later.

The first special case considered is a collision. If a robot detects a very nearby object with any of its three front-looking sonars (Sonars 3, 4, and 5), it immediately backs up slowly[1] with a speed of -1.0 m/s.

Next, the robot attempts to guess when it may have a sharp turn in its future. If Sonar 5 loses sight of the wall, the controller guesses that a corner is approaching, and slows down to $^2/_3$ of its nominal speed.

Finally, if the robot thinks it is required to perform a sharp turn to the right at a corner, it further slows to $^1/_3$ of its nominal speed.

3.3 Corridor Following Results

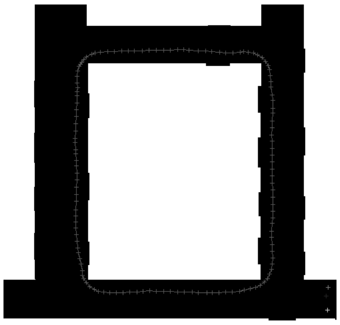
This simple controller is surprisingly robust and reliable. Figure 3 shows plots of a robot's traverse through our SAL model. It makes tight turns quite reliably, follows a reasonably straight path down the corridor despite the irregular sonar readings caused by doorways, and never loses track of the wall. The tight turns and rare loss of path around corners appears to be largely due to the robot's tendency to slow down as it approaches corners. Even after thousands of laps, the controller continues to function without failure.

Figure 4 shows the same traverse in the presence of three stationary obstacles. Again, the controller is quite robust, avoiding the vast majority of obstacles. However, it is admittedly

[1] Earlier versions of the controller had the robot stop completely, but this proved to not be very robust: when faced with an obstacle, the robot would just stop forever instead of making any effort to find a way around.

Program Fragment 2 Speed controller for clockwise travel around a cyclic corridor.

```
if (state[2] == NEAR || state[3] == NEAR || state[4] == NEAR)
  speed = -1.0;  // Go backwards to avoid collisions
else if (state[6] == FAR)
  speed = 2.5;  // Slow down a lot to go around a sharp turn
else if (state[5] == FAR)
  speed = 5.0;  // Slow down a little if the sharp turn is coming
else
  speed = nominalSpeed;  // 7.0
```


a)

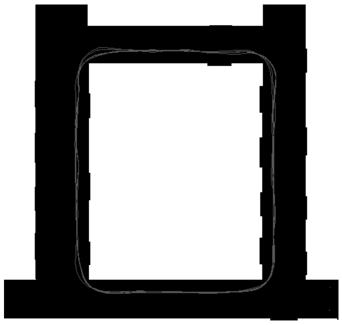
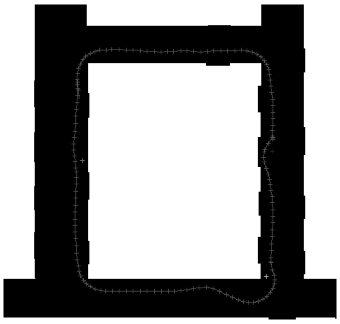

b)

Figure 3: a) *(left)* Plot of a single robot's traverse once around the corridor. Hash marks are at constant time intervals; the robot can be seen slowing down as it approaches a sharp turn. b) *(right)* A plot of many laps around the corridor. The robot's actions are relatively consistent.


a)

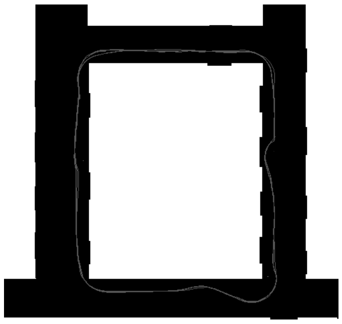

b)

Figure 4: a) *(left)* Plot of a single robot's traverse once around the corridor in the presence of three stationary obstacles. The robot avoids them in ways that look surprisingly "intelligent." b) *(right)* Many laps around the corridor.

possible to place obstacles in certain pathological ways such that the robot gets stuck and makes no further progress.

4 Constant Distance Following

With a good corridor following behavior, the constant distance following problem actually becomes much easier. Constrained by their corridor following, both robots can be thought of as following a simple one-dimensional path. The follower's problem has now been reduced to estimating its distance to the leader and adjusting its relative speed accordingly. The problem essentially has only one degree of freedom: speed control.

The difficulty now lies in the distance estimation. In this experiment, the follower estimates its distance from the leader by listening to a radio beacon emitted by the leader. The beacon comes from a simulated radio whose nominal transmission radius, 5m, is also the desired following distance. We model the simplest possible characteristics of Radiometrix RPC radios, where the probability of a successful packet transmission drops off sharply as the distance between the sender and receiver passes the radio's nominal transmission radius. Figure 5 shows an idealized picture of this property.

Given this kind of radio, it seems relatively straightforward to build a controller for constant distance following. The leader sends a continuous stream of packets at a constant frequency. The follower then listens to this packet stream. Because the follower knows the leader's sending frequency, it can easily determine its packet loss rate averaged over some window of recent transmissions. A high loss rate means the leader is too far away; a low loss rate means the leader is too close. The follower can then adjust its nominal speed accordingly. Our controller that

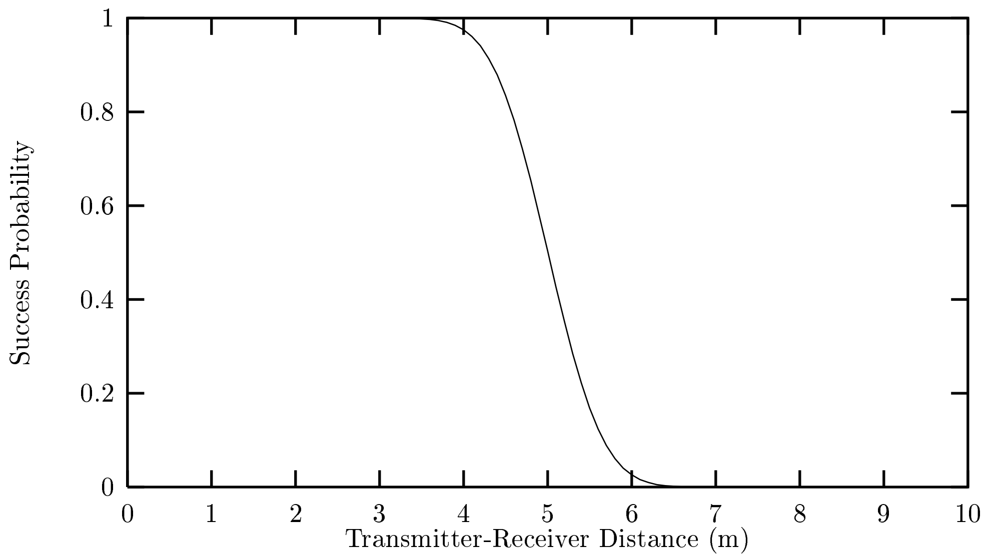


Figure 5: Idealized radio model for a radio with a nominal transmission radius of 5m.

implements this policy is shown in Program Fragment 3.

4.1 The Radio Model

This brings us to the heart of our study. The success of the controller described above is, of course, critically dependent on the actual radio propagation model. A radio system that differs significantly in its behavior from the ideal shown in Figure 5 will break a controller that assumes the ideal. If we want a radio-based controller designed in simulation to work in the real world, the simulated radio must have the same behavior as a real one, *to whatever level of detail is used by the controller.* It is this point, in my opinion, that makes imprecise simulations possible in robotics and not in pure networking research: network controllers extract much more detailed information from the world.

Radio propogation is notoriously hard to model in many domains. Indoor models are particularly bad, where multipath interference, reflections, shadowing, and other effects are caused by a very complex environment. The success rate of a packet is extremely difficult to predict unless the transmitter-receiver distance is well within the transmitter's nominal transmission radius. As this radius is reached, the success of a transmission decreases; however, there can be islands of good reception in a sea of bad loss.

We attempt to simulate this property using an "$(r+\text{percent-error})^2$" radio propagation model. In this model, the success of a packet is 1 if the distance from the transmitter to the receiver is below the sender's transmission radius, 0 otherwise. However, a random error is added to or subtracted from the distance before the thresholding decision is made. The error is uniformly selected from a range that is proportional to the distance; for example, $r' = r \pm 25\%$.

We studied our controller using four different percent-error settings in our simple radio model, as illustrated in Figure 6. Zero error corresponds to a "better-than-ideal" model, where the success probability drops instantaneously from 1 to 0. This could theoretically allow a follower to get a fix on its desired distance to the leader with arbitrary position in a simulator, but could

Program Fragment 3 Fragment for constant distance following from a fixed range radio beacon. Note that the follower is assumed to know the leader's nominal speed. However, it does not know the leader's actual current speed, which varies from nominal due to turns and collisions. The `getLossRate` function returns the loss rate over a 10-packet window.

```
if (following) {
  getLossRate(&shortLossRate);
  if (shortLossRate > 80)
    nominalSpeed = 9.0;
  else if (shortLossRate < 20)
    nominalSpeed = 5.0;
  else
    nominalSpeed = 7.0;
} else {
  // leader
  nominalSpeed = 7.0;
}
```

never be achieved in the real world due to the vagaries of real radio propagation. Five and 15 percent errors correspond more closely to the idealized real-world model shown in Figure 5 and are probably similar to real outdoor propagation conditions. 50 percent error is a pathological case where any given packet's success is unlikely to correlate well with the transmitter-receiver distance, as in an indoor environment.

4.2 Constant Distance Following Results

The complete controller as described in previous sections was run for each of the four radio models shown in Figure 6. For each model, a leader and follower robot were both started at opposite sides of the SAL corridor. In each case, the follower gradually caught up to the leader, and eventually settled into its steady state of continuous speed corrections in an attempt to maintain constant distance.

After this equilibrium was reached, we recorded the exact robot positions as reported by the simulator at 0.1 second intervals through 20 laps through the SAL corridor. At each time interval, we computed the difference between the desired following distance of 5m and the actual (simulated) distance between the robots. The cumulative error frequencies are shown in Figure 7.

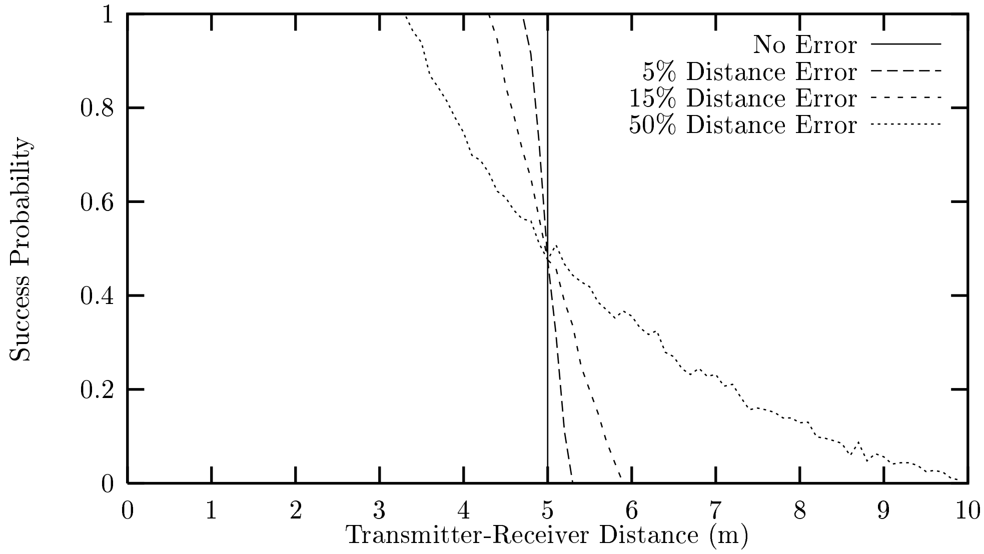


Figure 6: The "$(r + \text{percent-error})^2$" radio transmission model used for the robotic simulation. Four settings for the percentage error were used: 0%, 5%, 15% and 50% (from vertical to diagonal, respectively).

5 Conclusions

The most surprising result is that the performance of the controller is essentially the same for all four radio models. As described earlier, the 50 percent error model was designed to model indoor radio propagation, while lower errors were meant to model outdoor radio propagation. If we assume that our simplistic radio models are adequate, our simulation results suggest that the controller is robust enough to operate in the harsh indoor environment as well as the more favorable outdoor environment.

A critical test is to run the same controller on real robots, both indoors and outdoors. It is this future test that has the potential to either lend weight to or disprove our original theory that the simple radio model we selected is sufficient for robotics simulations. If the real robots fail indoors but succeed outdoors, we have proof that our radio model was not sufficiently predictive to be useful in controller design. On the other hand, if the tests in the real world are as uniform as in our simulator, it suggests that the radio models might be adequate. Of course, in this latter case, we have not *proven* anything: it could just be that this particular controller was so reliable as to work everywhere, and the real-world performance of some other controller might not be predicted as well by our simple radio model.

If the real-world test does succeed as uniformly as this simulation suggests, we have probably learned more about this particular controller than we have about the simulator or radio model. The controller attempts to "see through" transient errors by averaging results over many readings; the (simulated) packet rate is fast enough that a series of transmission attempts are measured before each control decision. A success in the real world suggests that sensors with data rates high enough to be polled many times per control cycle can be more reliable than controllers based on slower sensors.

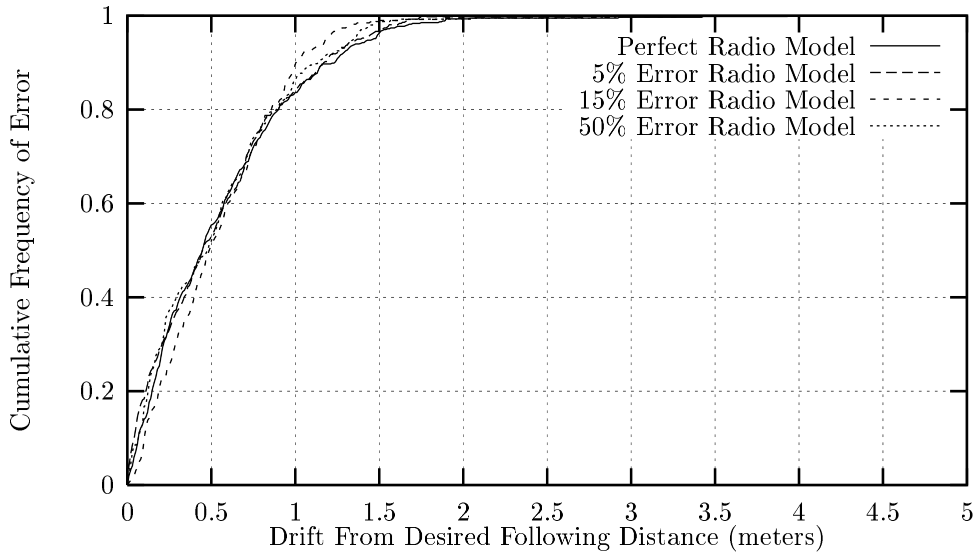


Figure 7: Cumulative distribution of errors in following distance for the four radio models.

6 Future Work

In the near term, we plan to implement the constant following distance algorithm on real robots by running the same controller on a real Pioneer-2. Although the robotic part of the controller should be easy to migrate, we will implement a real radio beaconing system based on actual Radiometrix RPC radios.

Depending on our success and level of interest, we may further extend the comparisons between the simulator and reality by trying different radio models and implementing additional controllers that use radio information.

7 Acknowledgments

The author would like to thank Lewis Girod, Deborah Estrin, and Gaurav Sukhatme for helpful feedback provided during the course of this research. Additional thanks go to Dr. Sukhatme for teaching an enjoyable, interesting, and highly educational class.