# 1 Body

One of the first models of drone behavior we thought of is that drones fly in straight lines, and do not interact with other drones. When a drone reaches the boundary of the surveillance region, that drone "bounces off" (see Figure 1). Note that if the wall is vertical, the bouncing just multiplies the $x$ component of the velocity by $-1$.

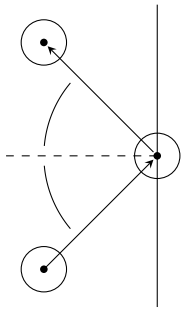

Figure 1: An incoming drone bounces off of a barrier. The circle is its radius of vision.

If the drones do not interact with each other, and are given uniformly random velocities and positions, then they should be distributed uniformly throughout the surveillence domain.

We have implemented a simulation of this model in matlab. A typical moment in time looks like this:

# 2 Code Appendix

Listing 1: Ideal Gas Simulation

```
% model parameters
w = 2.0; % width of container
h = 3.0; % height of container
r = 0.1; % radius of drone vision

eps = 1e-13;
huge = 1e15;

% simulation parameters
n_steps = 100000; % number of events
```

```
n_drones = 100; % number of drones at a time

% set up randomly distributed velocites all with
%       unit magnitude
v = zeros(n_drones,2);
for i = 1:n_drones
        theta = 2*pi*rand();
        v(i,:) = [cos(theta), sin(theta)];
end

% place drones randomly within the domain,
%       but a distance of no less than r away from
%       the walls
x = [r, r] + [w-r,h-r] .* rand(n_drones,2);

% array to save the drones in
X = zeros(n_drones, n_steps, 2);

t = 0;
time_intervals = zeros(1,n_steps); % save event times

for i = 1:n_steps
    % for each drone, find the time until it collides with each wall
        tls = (r - x(:,1))./v(:,1);
        trs = (w - r - x(:,1))./v(:,1);
        tbs = (r - x(:,2))./v(:,2);
        tts = (h - r - x(:,2))./v(:,2);

        % for each wall, find the minimum positive time until a collision
        %       with it, and which drone this time corresponds to.
        [tl, il] = min(tls + (huge*(tls < eps)));
        [tr, ir] = min(trs + (huge*(trs < eps)));
        [tb, ib] = min(tbs + (huge*(tbs < eps)));
        [tt, it] = min(tts + (huge*(tts < eps)));

        % find the minimum overall time until a wall collision,
        %       and which wall will be collided with
    [t_so_far, which_wall] = min([tl, tr, tb, tt]);

        % move the drones forwards and advance time
        x += v * t_so_far;
        t += t_so_far;
        printf('.')

        % if the drone collides with the left or right wall, invert the x velocity
        % if it collides with the top or bottom wall, invert the y velocity
```

```
        if which_wall == 1
                v(il, 1) = -v(il,1);
        end
        if which_wall == 2
                v(ir, 1) = -v(ir,1);
        end
        if which_wall == 3
                v(ib, 2) = -v(ib,2);
        end
        if which_wall == 4
                v(it, 2) = -v(it,2);
        end

        % record the timestamp and the drone positions at this time
        time_intervals(i) = t_so_far;
        X(:,i,:) = x;
end
```