Nature Inspired Computing

K.Dimopoulos

Steering Vehicles

Reynolds vehicle agents

In the late 1980s, computer scientist Craig Reynolds developed algorithmic steering behaviors for animated characters. These behaviors allowed individual elements to navigate their digital environments in a lifelike manner, with strategies for fleeing, wandering, arriving, pursuing, evading, and more.

Later, in his 1999 paper “Steering Behaviors for Autonomous Characters,” Reynolds uses the word vehicle to describe his autonomous agents.

By building a system of multiple vehicles that steer themselves according to simple, locally based rules, surprising levels of complexity emerge. The most famous example is Reynolds’s boids model for flocking or swarming behavior, which I’ll demonstrate later on

Why Vehicles?

In his book Vehicles: Experiments in Synthetic Psychology (Bradford Books, 1986), Italian neuroscientist and cyberneticist Valentino Braitenberg describes a series of hypothetical vehicles with simple internal structures, writing, “This is an exercise in fictional science, or science fiction, if you like that better.”

Braitenberg argues that his extraordinarily simple mechanical vehicles manifest behaviors such as fear, aggression, love, foresight, and optimism. Reynolds took his inspiration from Braitenberg

Reynolds describes the motion of idealized vehicles—idealized because he wasn’t concerned with their actual engineering, but rather started with the assumption that they work and respond to the rules defined. These vehicles have three layers

The 3 layers of an idealized vehicle

1. **Action Selection.**
    - A vehicle has a goal (or goals) and can select an action (or a combination of actions) based on that goal. The vehicle takes a look at its environment and calculates an action based on a desire. Goals and associated actions such as: seek a target, avoid an obstacle, and follow a path.
2. **Steering.**
    - Once an action has been selected, the vehicle has to calculate its next move. For us, the next move will be a force; more specifically, a steering force. A simple steering force formula is: steering force = desired velocity - current velocity.
3. **Locomotion.**
    - For the most part, we're going to ignore this third layer. Nevertheless, this isn't to say that you should ignore locomotion entirely. You will find great value in thinking about the locomotive design of your vehicle and how you choose to animate it. Could you add spinning wheels or oscillating paddles or shuffling legs?

Seek behaviour

Desired Speed and Steering Force

For the vehicle to go to the target it needs to correct its velocity to the desired velocity.

The Desired velocity is a velocity that takes the vehicle from its current position to the target:

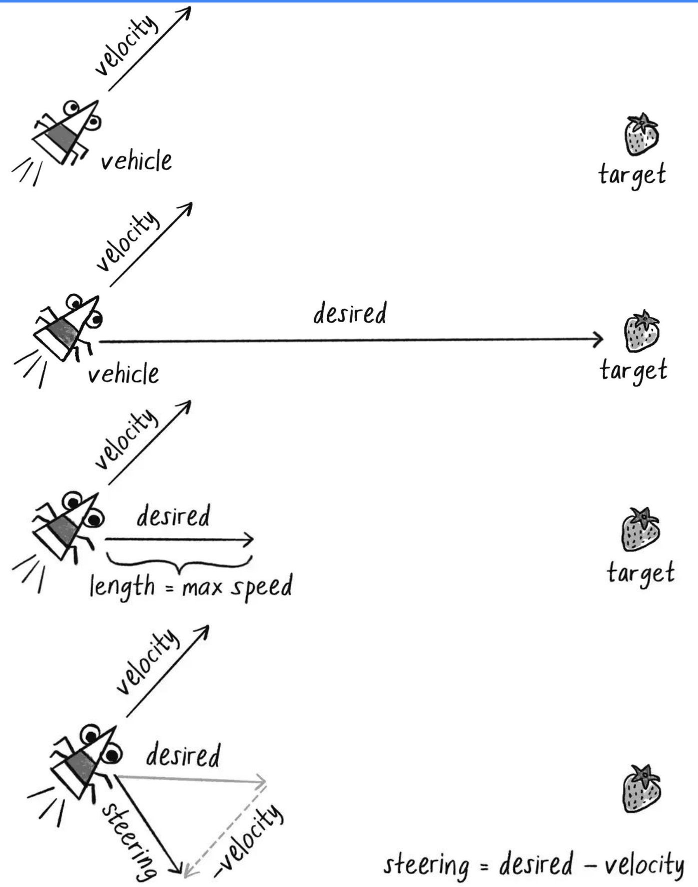
desired velocity = target - current position

However there are physical limits on the velocity that a vehicle can move thus the desired velocity is limited by max velocity

For the vehicle to go to the target it needs to correct its velocity to the desired velocity. To do that it needs to apply a force equal to the difference of the two velocities:

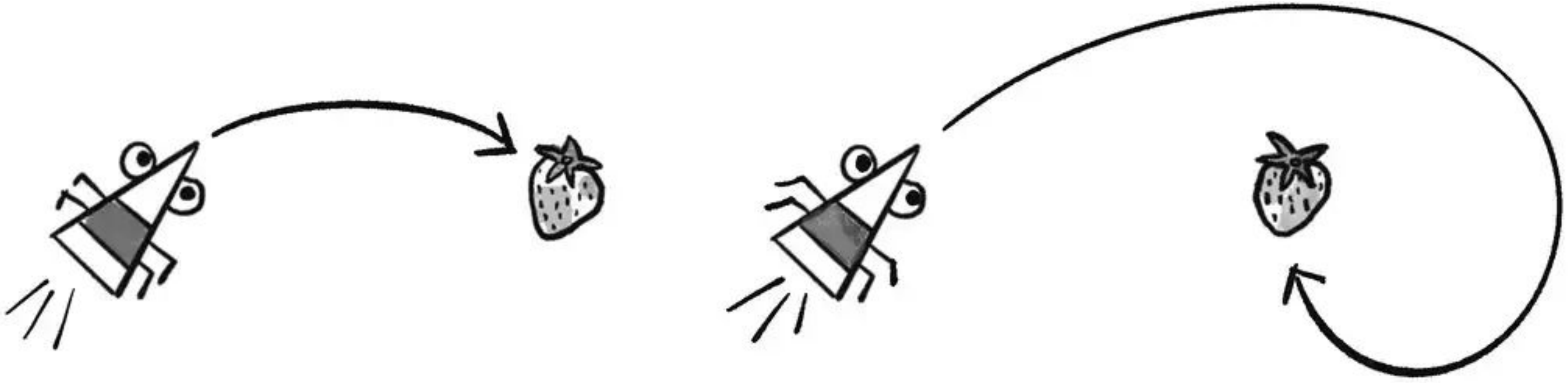
steering force = desired velocity - current velocity

However there are physical limits that limit the maximum speed that the vehicle can move by. Thus the desired velocity will be limited by the max speed that the vehicle can move



Maximum Steering Force

What sort of vehicle is this? Is it a super sleek race car with amazing handling? Or a giant Mack truck that needs a lot of advance notice to turn? A graceful panda, or a lumbering elephant? Steering ability can be controlled by limiting the magnitude of the steering force. Let’s call that limit the “maximum force” (or maxforce for short).



Seeking a target

Let’s put everything together now. Our vehicle will have:

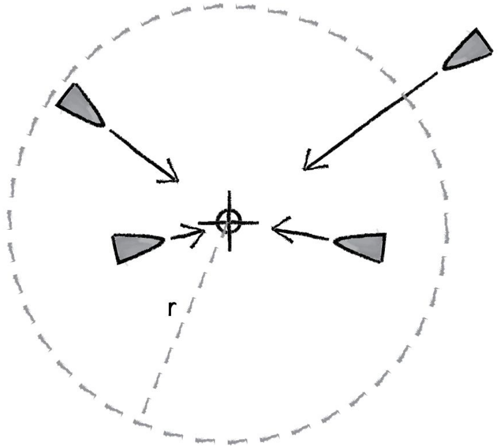
- location, velocity, acceleration, maxSeed, maxForce.
- Have a seek() behaviour that:
  - Calculates desired velocity based on its location and that of the mouse (limited to maxSpeed)
  - Calculate a steer force (limited to maxForce)
  - Returns or apply the steer force

Arrive behaviour

Arrive steering behavior

Lets make our vehicle slow down as it approaches its target (limited environment perception):

Let’s imagine a circle around the target with a given radius. If the vehicle is within that circle, it slows down—at the edge of the circle, its desired speed is maximum speed, and at the target itself, its desired speed is 0.



Turning the arrive to an agent perception

The arrive behavior is a great demonstration of an autonomous agent’s perception of the environment—including its own state.

We can imagine the circle being around the vehicle instead of the target. Then if the target is within that circle, the vehicle slows down—at the edge of the circle, its desired speed is maximum speed, and at the target itself, its desired speed is 0.

Random steering

Wandering Definition (by Reynolds)

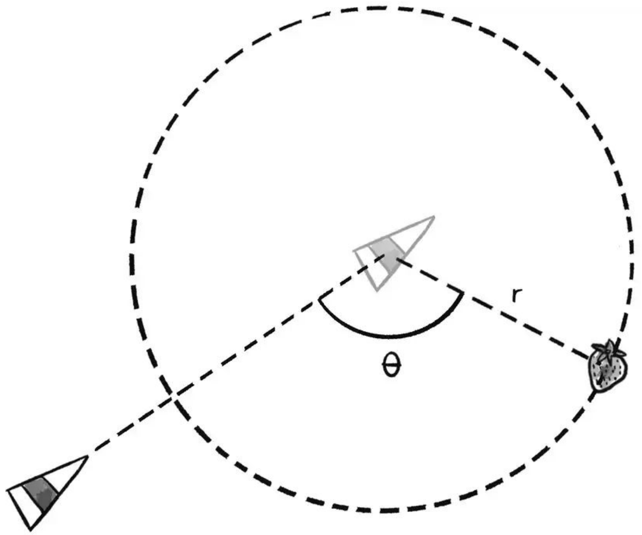
"Wandering is a type of random steering which has some long-term order: the steering direction on one frame is related to the steering direction on the next frame. This produces more interesting motion than, for example, simply generating a random steering direction each frame."

Wandering according to Reynolds

First, the vehicle predicts its future position as a fixed distance in front of it (in the direction of its current velocity).

Then it draws a circle with radius r centered on that position and picks a random point along the circumference of the circle.

That point, which moves randomly around the circle for each frame of animation, is the vehicle’s target, so its desired velocity points in that direction.



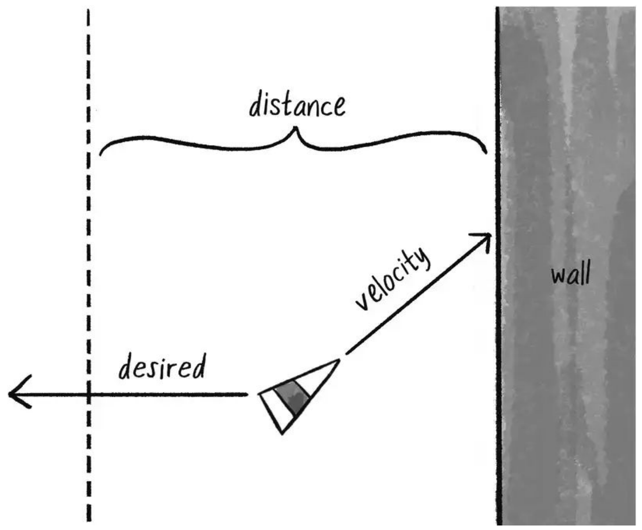
Wall following

Avoid walls

To avoid walls we can define the behaviour with the following rule:

If a vehicle comes within a distance d of a wall, that vehicle desires to move at maximum speed in the opposite direction of the wall.

The desired velocity is vertical to the wall (rotated by 90 degrees) in the opposite direction)



Follow walls

To follow a wall simply enhance the behaviour with an additional rule:

If a vehicle comes within a distance dmin of a wall, that vehicle desires to move at maximum speed in the opposite direction of the wall.

If a vehicle distance from the wall becomes bigger than dmax, that vehicle desires to move at maximum speed in the direction of the wall.

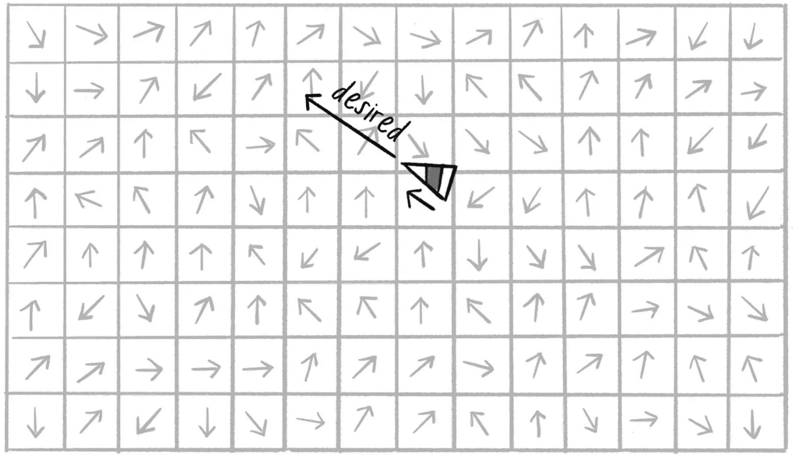
Where dmax < dmin

Flow fields

Flow Fields

Another one of Reynolds's steering behaviors is flow-field following. Think of the canvas as a grid. In each cell of the grid lives an arrow pointing in a certain direction—you know, a vector. As a vehicle moves around the canvas, it asks, "Hey, what arrow is beneath me? That's my desired velocity!"

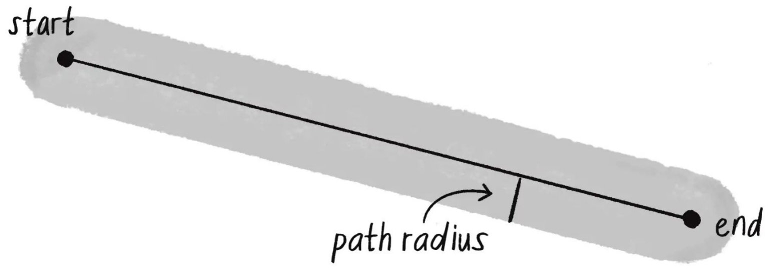
Reynolds's own flow-field example involves the vehicle looking ahead to its future position and following the vector at that spot. For simplicity's sake, you can instead have the vehicle follow the vector at its current position.



Path following

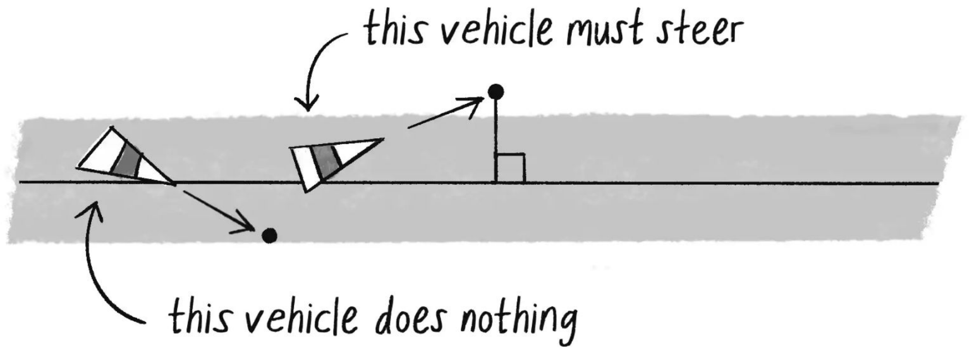
Path definition

A path is a sequence of points connected with lines. The path also has a thickness defined by a radius around the line.



Path Following algorithm

- At each frame the vehicle predicts its future position at the next frame (by adding its velocity to its position).
- Then the vertical distance from the point to the path is calculated.
- If that distance is bigger than the path radius, the vehicle predicts that at the next frame it will be outside the path, and it needs to correct its velocity.
- To correct the velocity, the vehicle picks a point ahead on the path, and uses its seek behaviour.



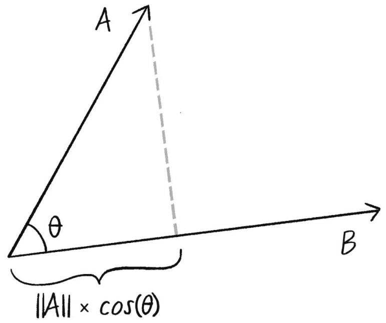
Considerations

How far ahead do we pick a point?

- No correct answer, it really is up to us

How do we calculate the normal from a point to a line?

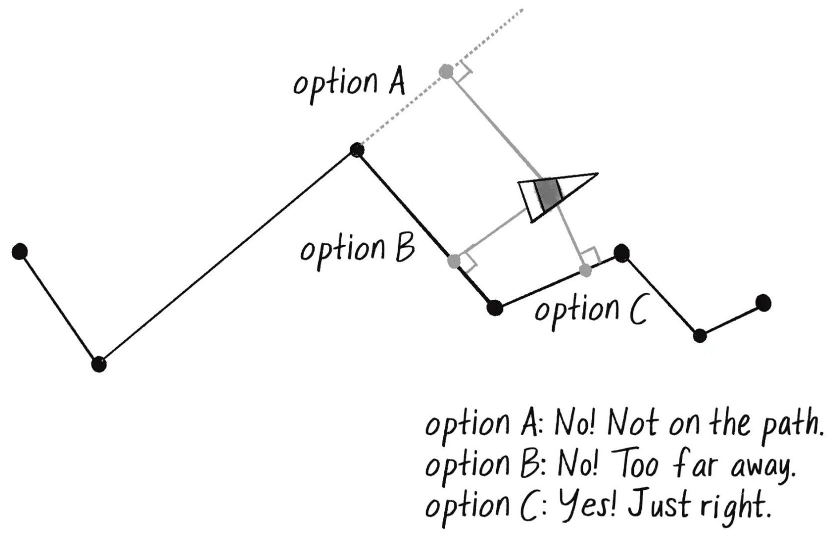
- We use the scalar projection:
    - based on the dot product between the vector that point from the starting point of the line to the position of the vehicle, and the unit vector that points at the direction of the path. Then the distance from our position to that point is the normal



Multiple segments in the path

To find the target with just one line segment, I had to compute the normal to that line segment. Now that I have a series of line segments, I also have a series of normal points to be computed—one for each segment. Which one does the vehicle choose? The solution Reynolds proposed is to pick the normal point that is (a) closest and (b) on the path.

If you have a point and an infinitely long line, you'll always have a normal point that touches the line. But if you have a point and a finite line segment, you won't necessarily find a normal that's on the line segment. If this happens for any of the segments, I can disqualify those normals. Once I'm left with just those normals that are on the path, I pick the one that's shortest.


option A
option B
option C
option A: No! Not on the path.
option B: No! Too far away.
option C: Yes! Just right.