NAIL139 AI for Computer Games
Lecture 4: Local Navigation

Adam Dingle

November 7, 2025

Local Navigation: Overview

- How can an agent decide how to move, using local information only?
- Steering behaviors
  - Seek, flee, arrive, align, pursue, evade, ...
- Velocity obstacles
  - Avoid running into nearby moving objects/agents

Simple vehicle model (point mass approximation)

Vehicle could be a spaceship, boat, car...

- Attributes
  - position: vector (m)
  - velocity: vector (m/s)
- Constants
  - MAX_VELOCITY: scalar (m/s)
  - MAX_ACCEL: scalar (m/s^2)
- On every frame, steering behavior outputs
  - acceleration: vector (m/s^2)

Vehicle model with orientation

- Attributes
  - position: vector (m)
  - velocity: vector (m/s)
  - orientation: scalar (radians)
- Constants
  - MAX_VELOCITY: scalar (m/s)
  - MAX_ACCEL: scalar (m/s^2)
  - MAX_ANGULAR_VELOCITY: scalar (radians/s)
- On every frame, steering behavior outputs
  - acceleration: scalar (m/s^2)
  - angular velocity: scalar (radians/s)

Simple vehicle model (point mass approximation)

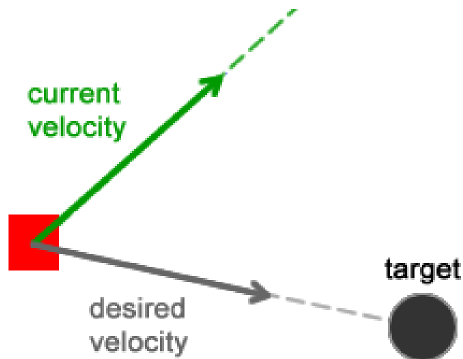
- Attributes
  - position: vector (m)
  - velocity: vector (m/s)
- Constants
  - MAX_VELOCITY: scalar (m/s)
  - MAX_ACCEL: scalar (m/s^2)
- On every frame, steering behavior outputs
  - acceleration: vector (m/s^2)

```
# Physics for simple vehicle model
func tick(accel: vector, elapsed: float):
    accel = accel.limit_length(MAX_ACCEL)
    velocity += accel * elapsed
    velocity = velocity.limit_length(MAX_VELOCITY)
    position += velocity * elapsed
```

Steering behaviors

- Flocks, Herds, and Schools (Reynolds, 1987)
- Steering Behaviors For Autonomous Characters (Reynolds, 1999)
- Simple behaviors
  - Seek and Flee
  - Arrive
  - Pursue and Evade
  - Wander
  - Path Following
  - ...
- Combined behaviors
  - Flocking
  - Leader Following
  - ...

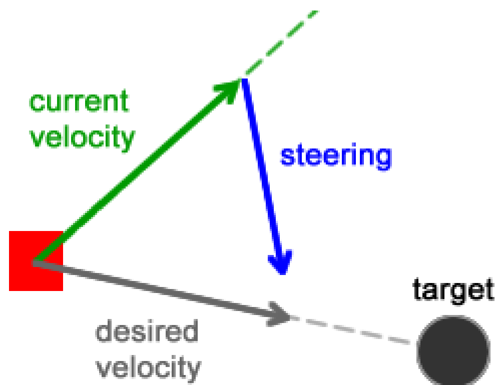
Seek toward a static target



Can we just accelerate toward the target?

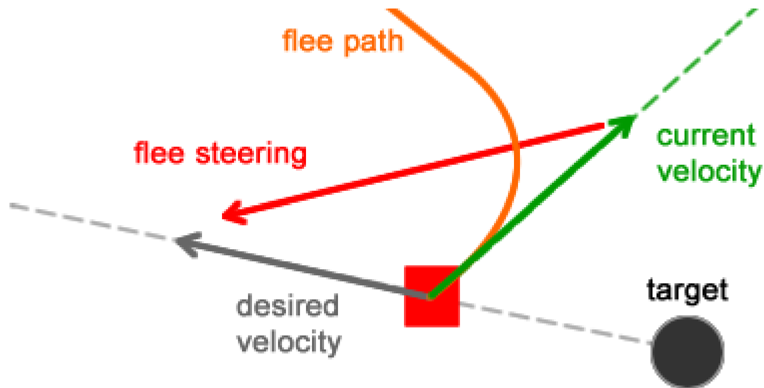
```
# Accelerate directly toward target
func try_seek(target_pos: vector):
    return normalize(target_pos - position) * MAX_ACCEL
```

Seek, the right way




```
# Seek toward a static target
func seek(target_pos: vector):
    desired_vel = normalize(target_pos - position) * MAX_VELOCITY
    return normalize(desired_vel - velocity) * MAX_ACCEL
```

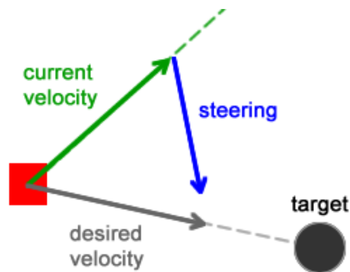
Flee from a static target

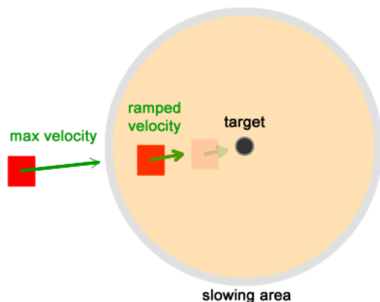



```
# Flee from a static target
func flee(from_pos):
    desired_vel = normalize(position - from_pos) * MAX_VELOCITY
    return normalize(desired_vel - velocity) * MAX_ACCEL
```

Arrive at a static target

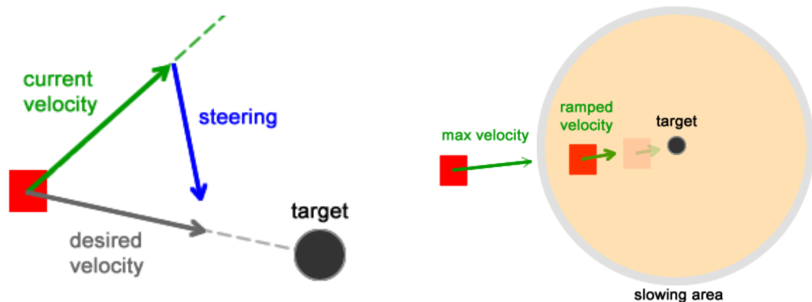
- Like Seek, but agent slows down as it approaches the target
- Slowing distance d is the distance to decelerate to a full stop
- Let v_{max}, a_{max} be MAX_VELOCITY, MAX_ACCEL
- How can we calculate d as a function of v_{max} and a_{max}?



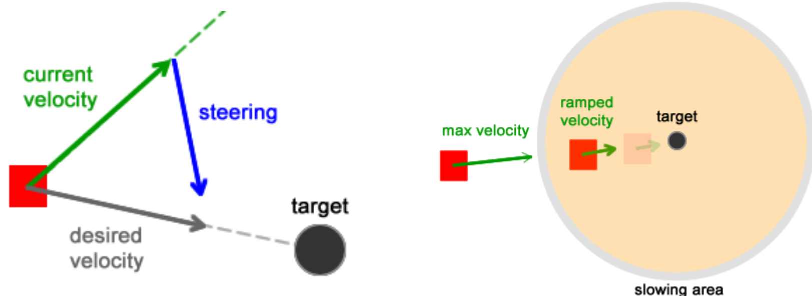


Arrive at a static target

- Like Seek, but agent slows down as it approaches the target
- Slowing distance d is the distance to decelerate to a full stop
- Let v_{max}, a_{max} be MAX_VELOCITY, MAX_ACCEL
- Time to decelerate $= v_{max}/a_{max}$
- Average speed during deceleration $= v_{max}/2$
- So d $= (v_{max}/a_{max})(v_{max}/2) = v_{max}^2/(2a_{max})$



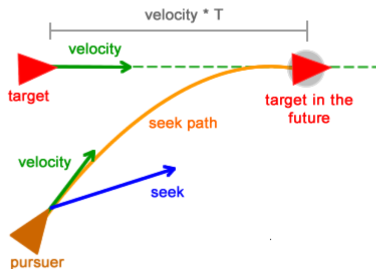
Arrive at a static target



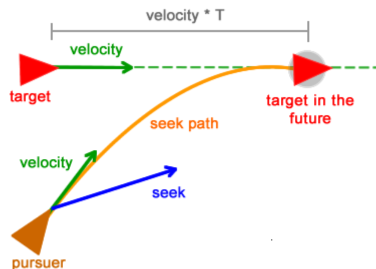

```
# Arrive at a static target
func arrive(target_pos: vector):
    slowing_distance = MAX_SPEED ** 2.0 / (2 * MAX_ACCEL)
    distance = (target_pos - position).length()
    d = minf(distance / slowing_distance, 1.0)
    clipped_speed = d * MAX_SPEED
    desired_vel = clipped_speed * normalize(target_pos - position)
    return normalize(desired_vel - velocity) * MAX_ACCEL
```

Pursue a moving target

- Like Seek, but toward a moving target
- Agent predicts the location of the target in the future
- Prediction based on target velocity & time T to reach target
  - How to calculate T?
  - Possible approximation: time to reach current target pos at max velocity

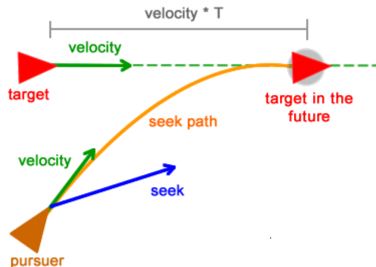


Pursue a moving target




```
# Pursue a moving target
func pursue(target_pos: vector, target_vel: vector):
    time_to_target = (target_pos - position).length() / MAX_SPEED
    predicted_pos = target_pos + target_vel * time_to_target
    return seek(predicted_pos)
```

Evade a moving target




```
# Evade a moving target
func evade(target_pos: vector, target_vel: vector):
    time_to_target = (target_pos - position).length() / MAX_SPEED
    predicted_pos = target_pos + target_vel * time_to_target
    return flee(predicted_pos)
```

Warning: this does not seem to be very effective!

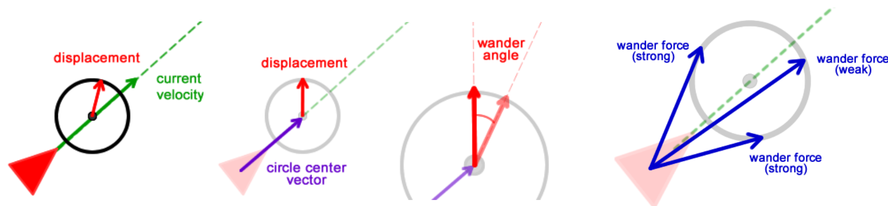
Wander

To wander around, can we just accelerate in a random direction on each game tick?

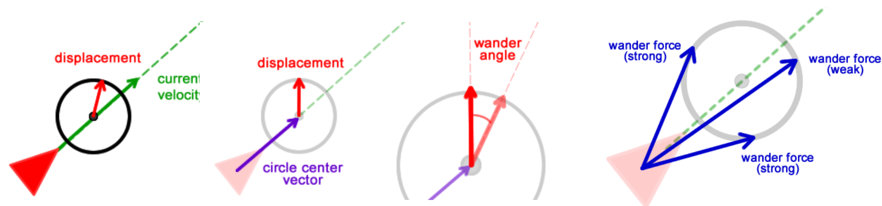
```
func try_wander():
    angle = randf_range(0, 2 * PI)
    return Vector2.from_angle(angle) * MAX_ACCEL
```

Wander: a more natural-looking approach

- At each time step, a random offset is added to the wander direction
- The wander direction determines a point on a circle in front of the agent
- The agent accelerates toward that point



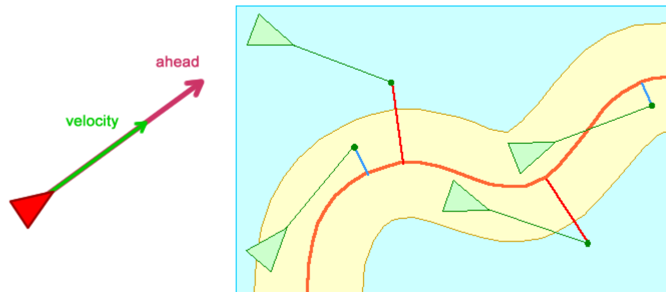
Wander: a more natural-looking approach




```
func wander():
    wander_angle += randf_range(- WANDER_RATE, WANDER_RATE)
    v = (normalize(velocity) * WANDER_DIST +
        Vector2.from_angle(wander_angle) * WANDER_RADIUS)
    return normalize(v) * MAX_ACCEL
```

Path Following

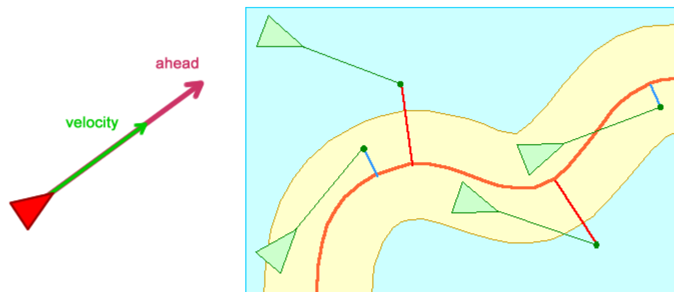
- Agent projects its future position P at a certain fixed distance ahead, then finds the point on the path nearest to P
- Agent then seeks toward that point




```
func path_follow(curve):
  future = position + velocity * PATH_LOOKAHEAD
  goal = curve.get_closest_point(future)
  return seek(goal)
```

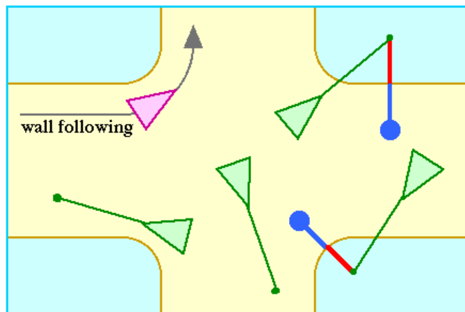
Path Following: possible variations

1. Do not seek if the projected position P is within a permitted distance from the path (i.e. in the yellow area below)
2. Reflect the projected position P to the other side of the path, then seek there



Wall Following and Containment

- To follow a wall, follow a path that is a fixed distance from the wall
- To stay contained inside a polygonal area:
  - Agent projects its future position P at a fixed distance D ahead
  - If P is inside the polygon, accelerate forward
  - Otherwise find the closest point W on the polygon, then seek toward W + C(W - P) for some constant C
  - Setting C = D = stopping distance seems to work well



Flocking/Swarming

A combined steering behavior, inspired by nature

Separation

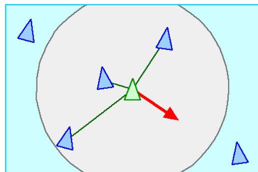
- Steer away from nearby flockmates, with separation force inversely proportional to distance

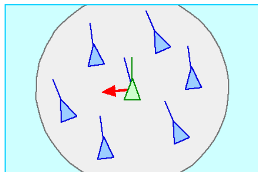
Alignment

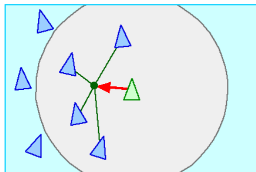
- Steer toward average velocity of nearby flockmates

Cohesion

- Steer toward average position of nearby flockmates

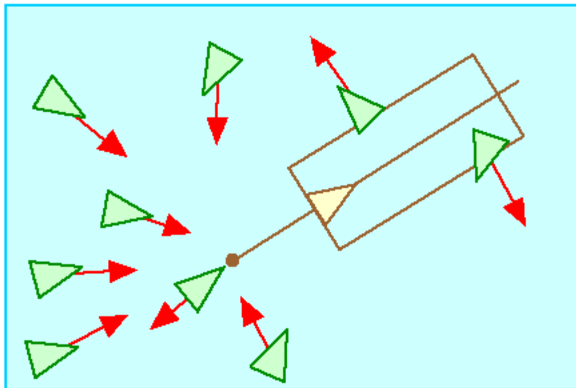






Leader Following

- Agents are steered to follow a leader
- Steering force consists of:
  - Arrival – the target is slightly behind leader
  - Separation – to prevent collisions with other followers
  - A follower in a rectangular region in front of the leader will steer away from the leader's path

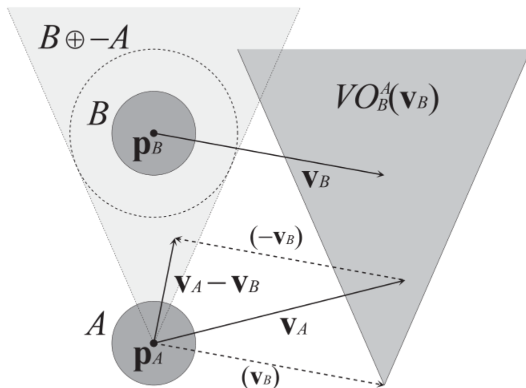


Velocity Obstacles

- Goal: avoid collisions more reliably than we can achieve with steering behaviors
- Invented in robotics (and elsewhere)
- Fiorini and Shiller, Motion planning in dynamic environments using velocity obstacles (1998)

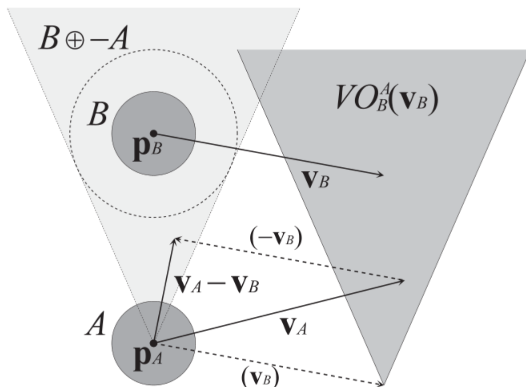
Velocity Obstacles

- Agents A, B are at positions p_A, p_B
- B is travelling at a fixed velocity v_B
- Which velocities v_A will allow A to avoid colliding with B?



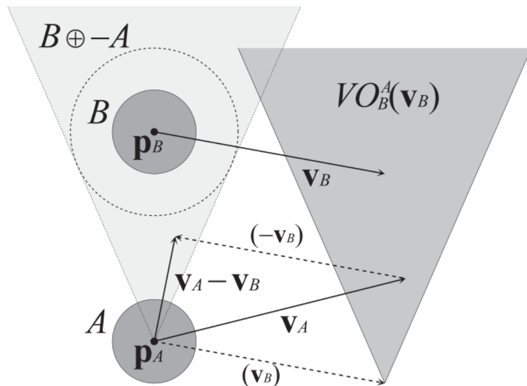
Velocity Obstacles

- $B \oplus -A$ is the set of positions where A would collide with B
- $S \oplus T = \{s + t \mid s \in S, t \in T\}$ (Minkowski sum)
- If A and B are circles, $B \oplus -A$ is a circle whose radius is the sum of the radii of A and B



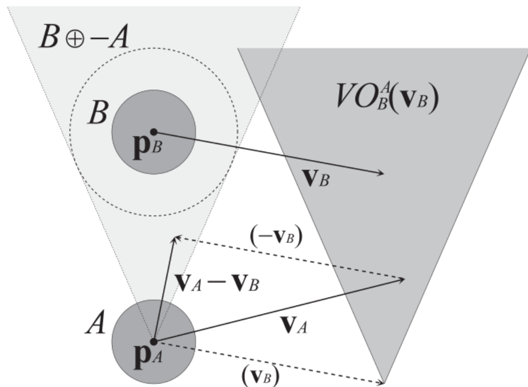
Velocity Obstacles

- The left cone shows velocities v_A for which A would not collide with B if B were not moving
- But B is moving at v_B, so we must consider the relative velocity $v_A - v_B$



Velocity Obstacles

- $\lambda(p, v) = \{p + tv \mid t >= 0\}$
  - a ray starting at p, heading in direction v
- $VO_B^A(v_B) = \{v_A \mid \lambda(p_A, v_A - v_B) \cap (B \oplus -A) \neq \emptyset\}$
  - velocity obstacle of B to A
  - set of velocities v_A for which A will collide with B!



Velocity Obstacles

- VO_B^A is a cone with apex at v_B
- If $v_A \in VO_B^A$, A will collide with B
- If v_A is the apex velocity v_B, they will not collide
- If v_A is in the left or right half-plane outside VO_B^A, A will pass B on the left or right