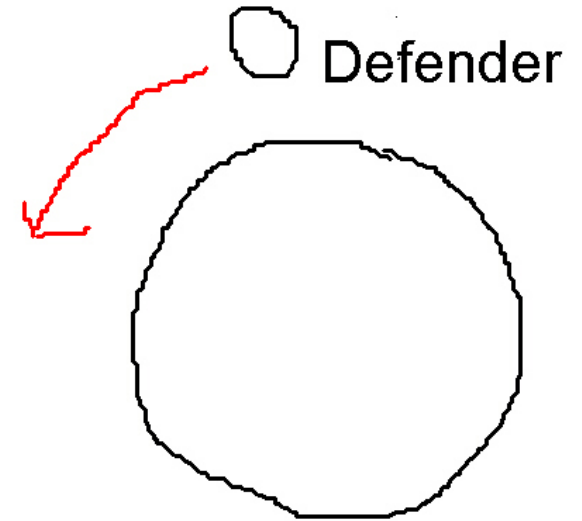


Defender {drive}, bounds [dt, dt+epsilon]

State:

p initially 0, p_o, v_o in R^3



Transitions:

input position(p') // Input from vision system

Eff: $p := p'$

input opponent(p, v) // opponent sensor

Eff: $p_o := p$; $v_o := v$

internal interception // bounds in [0, 0]?

Pre: $|p - p_o| < \epsilon$

Eff: bang

output drive(v') // v' is the desired velocity

Pre: $v' = k(p_o - p) + v_o$

Vision system

State: p, v in R^3 , initially 0; moved : bool, initially false

output position(p') // send position information periodically

Pre: $p_new = \text{physics}(p, v)$ $p' = p_new$

Eff: $p := p_new$

input drive(v') // watch the drive settings

Eff: $v := v'$

output opponent(p, v) // Opponent wakes up randomly

Pre: not moved $_p, v$ within drive limits

Eff: moved := true

Tasks and bounds:

{position}, bounds $[dt - \text{epsilon}, dt + \text{epsilon}]$ // epsilon = 0?

{opponent}, bounds $[0, t]$ // do we want this?

Theorem: interception happens.

Informal proof: suppose the opponent wakes up at time t_0 , with position, velocity = (p_0, v_0) . The defender sensor saves the info in (p_o, v_o) . Within time $[t_0+dt, t_0+dt+\epsilon]$, the defender computes an intercept course from position info recorded from $[t_0, t_0-dt]$. Use physical laws to calculate interception time.

Problems:

1. The defender might be too slow to react. Have to set up initial conditions so this can't happen.
2. The defender can recompute drive velocities whenever it gets new position info.
3. Position errors need to be specified correctly.

2-on-1 drill:

Defender

State: p_1, p_2, p_o, v_o in R^3

input position(p_1', p_2')

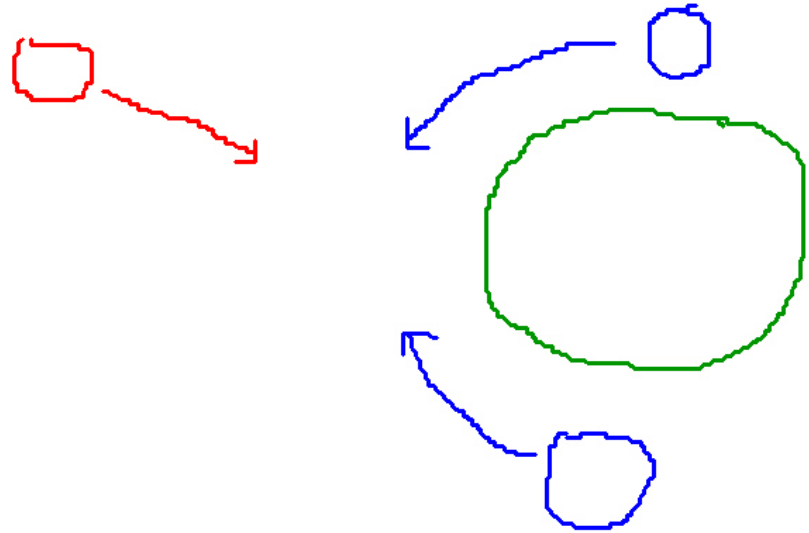
Eff: $p_1, p_2 := p_1', p_2'$

input opponent(p, v)

Eff: $p_o, v_o = p, v$

output drive(v) bounds[dt, dt+epsilon]

Pre: if $|p_1 - p_o| < |p_2 - p_o|$, compute intercept



2-on-1 with communication
State: p1, p2, po, vo in R3
deciding : bool, i : int

input position(p)
Eff: p1 := p

input opponent(p, v)
Eff: po, vo := p, v

output announce1(p)
Pre: p = p1

input announce2(p)
Eff: p2 := p

output init(i)
Pre: not deciding
if $|p1-po| < |p2-po|$ then i=1 else i=2 Eff: deciding := true

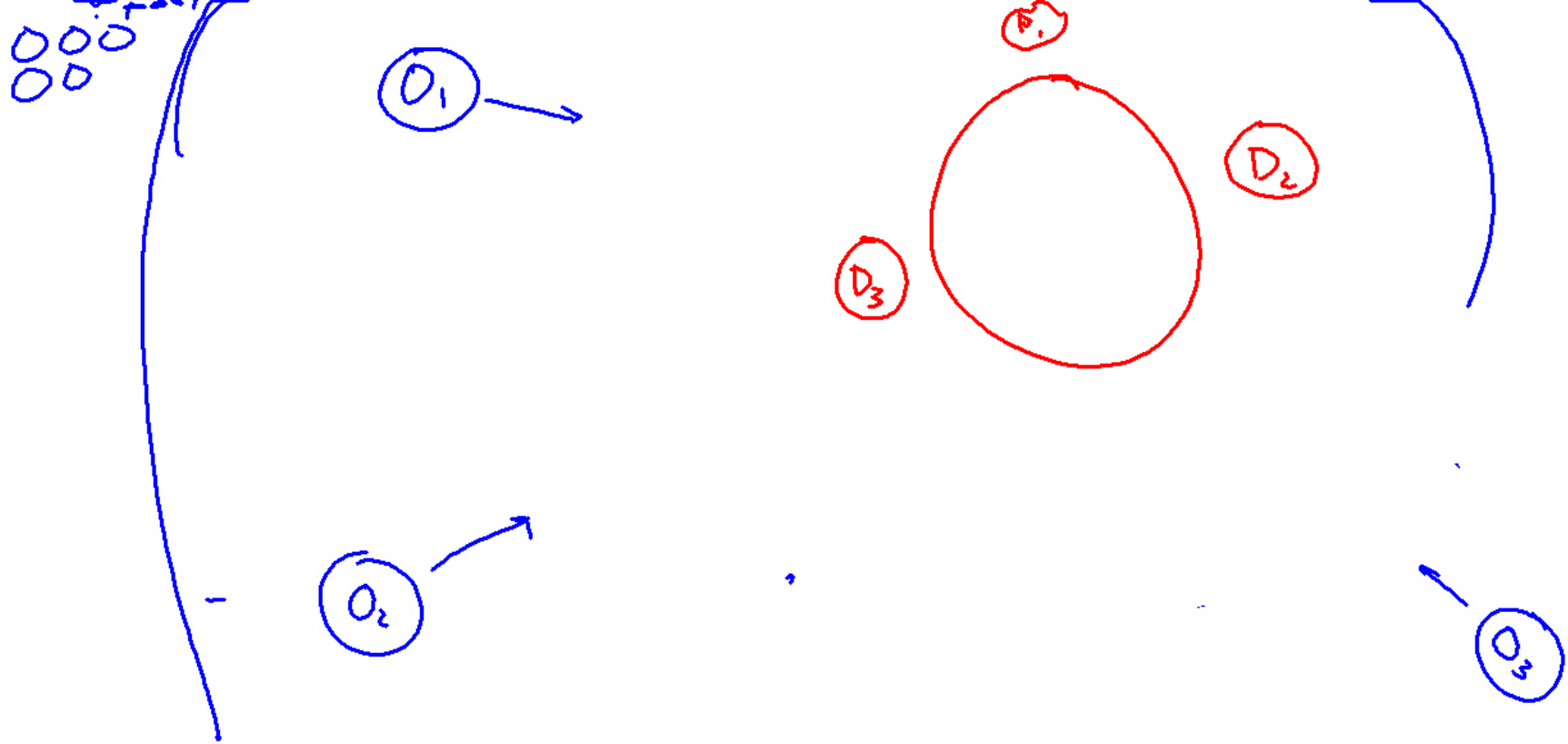
input decide(i')
Eff: deciding := false; i := i'

output drive(v)
Pre: if not deciding _i = 1 then
 ...compute intercept course...
 else
 v=0

/* In this version, the vehicles announce
* their positions to one another. They
* use the consensus protocol to
* decide which will intercept (based
* only on distance.
*/

Theorem: intercept happens

Informal proof: The opponent starts moving at t_0 . An init happens within $[t_0, t_0 + dt + \epsilon]$. A decision is made within $[t_0 + t, t_0 + dt + \dots]$. The intercept course is computed from data that is at most dt old.



Defender(i)

State:

$p[N], v[N] : R^3$

opponents : R^3 list

deciding : bool, **plan** : plan

input position_j(p', v')

Eff: $p[j], v[j] := p', v'$

input opponent_j(p', v')

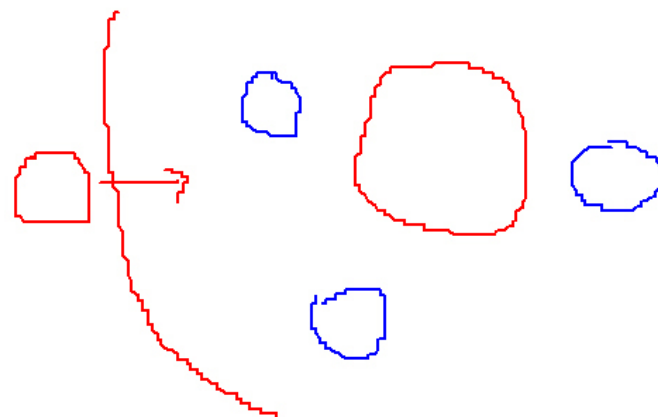
Eff: add (j, p', v') to **opponents**

output init_i(plan)

Pre: not deciding and formulate(plan)

input decide_i(plan')

Eff: **deciding** := false, **plan** := plan'



output control(v)

Pre: if not deciding

& exists $j. (i, j)$ in plan

then

$v = \text{intercept } j$

else

$v = 0$

output announce_i(p, v)

Pre: $p, v = p[i], v[i]$

Algorithm: defenders periodically announce their position/velocity. Sensors periodically sense position and opponents. Whenever a new event happens, consensus is invoked to decide on a new plan. This runs in parallel; once a decision is made, the control loop adjust velocities for interception.

Options: it may be better not to include the plan in consensus. Instead, we could use a 3PC to synchronize all the position information in the defenders, then they would all make the same decision individually. This is a "state transfer" algorithm.

Theorem: all opponents are intercepted.

Roboflag drill mpeg

<http://www.tam.cornell.edu/~earl/LPE>

Go to:

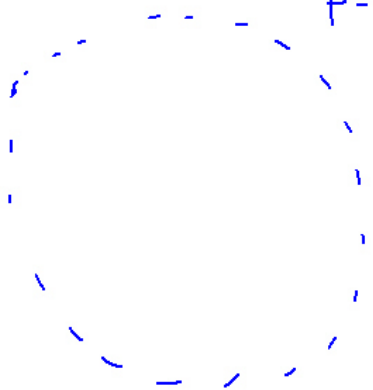
http://www.cs.caltech.edu/~jyh/spool/pursuit_automata.ps

Opponent

$$\textcircled{1} \quad \dot{p}_0 = v_0$$

Continuous-time
version

○ defender
 $\dot{p} = v$



$$e = p - p_0$$

$$\dot{e} = -k e \quad k > 0$$

$$v - v_0 = -k(p - p_0)$$

$$v = k(p_0 - p) + v_0$$

$$\|v_0\| = 1$$

$$\|v_n\| \cdot \tau < d$$

Discrete-time version

$$\checkmark \quad p_0(n+1) := p_0(n) + h \cdot v_0(n)$$

$$p(n+1) := p(n) + h \cdot v(n)$$

$$(h > 0) \quad e(n) = p(n) - p_0(n)$$

$$e(n+1) = e(n) + h \cdot (v(n) - v_0(n))$$

$$v(n) = k(p_0(n) - p(n)) + v_0(n) \quad \uparrow$$

$-ke(n)$

$$e(n+1) = (1 - hk) e(n) \quad \frac{1}{2h} < k < \frac{1}{h}$$

$0 < 1 - hk \leq \frac{1}{2}$

hi

send($L(p_0, V_0)$)
Eff: $p_0 := p_0 + hV_0$

internal $output()$
Eff: compute
if flag then
 send to p_i add to queue
else
 send to p_j

{state
 queue of msgs
 add to queue

output send(L)

Pre: queue is nonempty
 $hd(queue) = i$

Eff: remove from q

bounds $e \in \{e, 2e\}$
 $e \leq h$