Mobile Robotics

Matlab Simulation of Braitenberg Vehicles

Vitor Santos, Nuno Lau
Universidade de Aveiro - 2024

Basic functions

Main geometric transformations, linear and circular paths, and simple robot animation

1-Basic Functions - Homogeneous Transformations

- Use column vectors to represent points: $P = [x\ y]^T$
- Create translation and rotation functions.
- Code suggestions:

```
function T=transl(v)
if numel(v) < 2
    v=[v(1) v(1)];
end

T=[ 1 0 v(1)
    0 1 v(2)
    0 0 1
 ];
```

```
function T=rotat(a)

T=[cos(a)  -sin(a) 0
   sin(a)   cos(a) 0
   0        0      1
   ];
```

2-Basic functions - Linear trajectory

- Create a linear trajectory...
- ... between P1 and P2 with N points
- Returns a structure with 3 fields:
  - **MM.xy** - the N xy points
  - **MM.angle** - orientation vector at each point (actually they are all the same but we intend to establish a methodology)
  - **MM.T** - The associated geometric transformations in case of utility (redundant information)
- Code suggestion:

```
function MM=traject(P1,P2,N)
% MM.xy - the matrix of xy points
% MM.angle - the vector of orentations
% MM.T - The associate geometric transformation
M=[linspace(P1(1), P2(1), N)
   linspace(P1(2), P2(2), N)
   ];

dM=diff(M,1,2); %first diff. along dim. 2
ang=atan2(dM(2,:), dM(1,:));
ang(end+1)=ang(end);

T=zeros(3,3,width(M));
for n=1:size(T,3)
    T(:,:,n)=transl(M(:,n))*rotat(ang(n));
end

MM.xy=M;
MM.angle=ang;
MM.T=T;
```

3-Basic functions - Circular trajectories

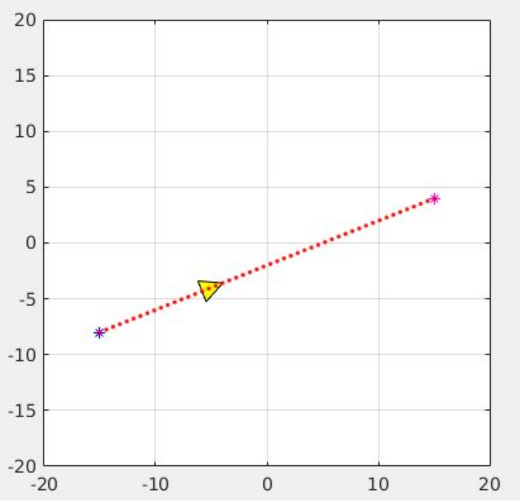
- Create a circular path between two points P1 and P2 with a given radius R and N points.
- Set some options:
  - which center
  - the minimum radius
  - the sense of arc
- The return is in the same format as the linear path.

```
function MM=circtraj(P1,P2,R,N)
% MM.xy - the matrix of xy points
% MM.angle - the vector of orentations
% MM.T - The associate geometric transformation

A = P1; % Point A to be on circumference
B = P2; % Same with point B
d = norm(B-A);
R = max(R,d/2); % ensure R radius >= d/2
if isinf(R); R = 1e6; end %use a very large R

C = (B+A)/2+sqrt(R^2-d^2/4)/d*[0,-1;1,0]*(B-A); % One center
%C = (B+A)/2-sqrt(R^2-d^2/4)/d*[0,-1;1,0]*(B-A); % the other
a = atan2(A(2)-C(2),A(1)-C(1));
b = atan2(B(2)-C(2),B(1)-C(1));
b = mod(b-a,2*pi)+a; % Ensure that arc goes counterclockwise
t = linspace(a,b,N);
M=C+R*[cos(t); sin(t)];

dM=diff(M,1,2); %first difference along dimension 2
ang=atan2(dM(2,:), dM(1,:));
ang(end+1)=ang(end);

T=zeros(3,3,width(M));
for n=1:size(T,3)
    T(:,:,n)=transl(M(:,n))*rotat(ang(n));
end

MM.xy=M;
MM.angle=ang;
MM.T=T;
```

Code inspired in https://www.mathworks.com/matlabcentral/answers/367126-plot-an-arc-on-a-2d-grid-by-given-radius-and-end-points

4-Basic example of animation

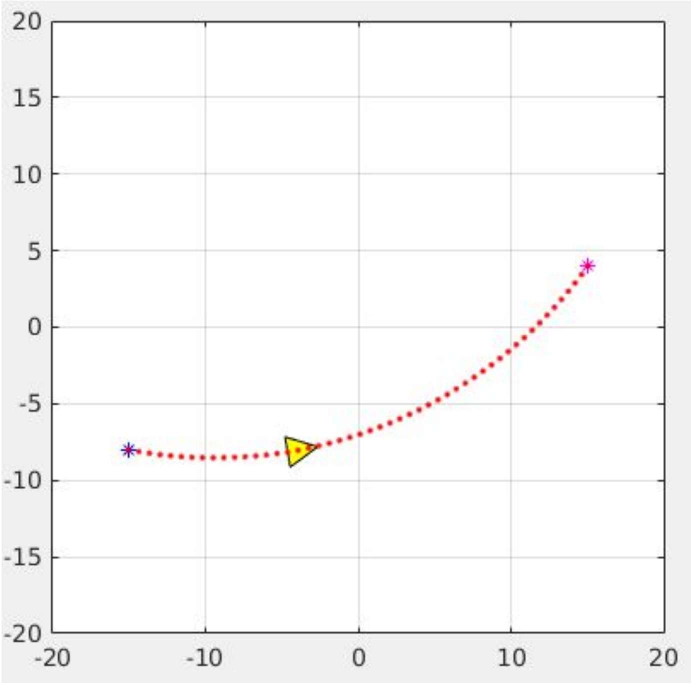
- Create a simple model of the robot
- Create a linear path between 2 points
- represent the trajectory
- Animate robot movement




```
R=[ 0 0  2
    1 -1 0
    ];

Rh=R; Rh(3,:)= 1; %homogeneous version

hR=fill(Rh(1,:), Rh(2,:), 'y' );
axis equal; grid on; hold on

P1=[-15 -8]';
P2=[15 4]';
N=50;

MM=traject(P1,P2,N);

M=MM.xy;
TT=MM.T;

plot(P1(1),P1(2), '*b', P2(1),P2(2), '*m');
plot(M(1,:), M(2,:),'.r')
axis ([-20 20 -20 20]) %adjust if needed

for n=1:N
    T=TT(:,:,n);
    nR=T*Rh;
    hR.XData=nR(1,:); hR.YData=nR(2,:);
    pause(0.05)% adjust for your needs
end
```

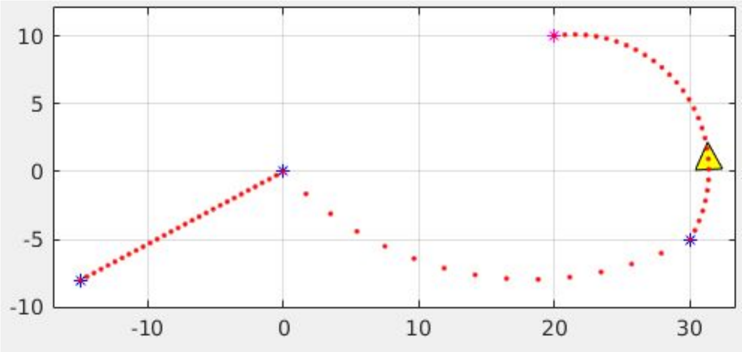
5-Represent and animate a robot in a circular path

- Same points as before
- Use an R=30 radius (illustrated)
- Test the circular path also with R=infinity and confirm the similarity with the linear path!



6- Composite trajectory

- Path with multiple segments
- variable radius
- Variable speeds (steps)



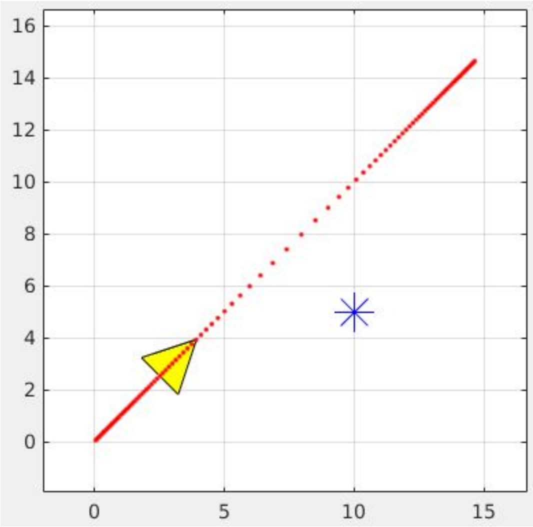

```
P1=[-15 -8]';
P2=[0 0]';
P3=[30 -5]';
P4=[20 10]';

P=[P1 P2 P3 P4];
RR=[Inf, 25, 10];
NN=[30 15 30];

allxy=[]; allangle=[]; allT=[];
for n=1:width(P)-1
    R=RR(n); N=NN(n);
    M=circtraj(P(:,n), P(:,n+1),R,N);
    plot(P(1,n),P(2,n), '*b', P(1,n+1),P(2,n+1), '*m');
    allxy=cat(2,allxy,M.xy);%allxy=[allxy M.xy];
    allangle=cat(2,allangle,M.angle);
    allT=cat(3,allT,M.T);
end

% Set and enlarge axis for good viewport
axis([min(allxy(1,:)) max(allxy(1,:)) ...
     min(allxy(2,:)) max(allxy(2,:))]);
daxl=2*ones(4,1) ;
axl=axis; daxl=[-daxl(1) daxl(2) -daxl(3) daxl(4)];
axis(axl+daxl);

plot(allxy(1,:), allxy(2,:),'.r')
for n=1:width(allxy)
    T=allT(:,:,n);
    nR=T*Rh; hR.XData=nR(1,:); hR.YData=nR(2,:);
    pause(0.05)
end
```

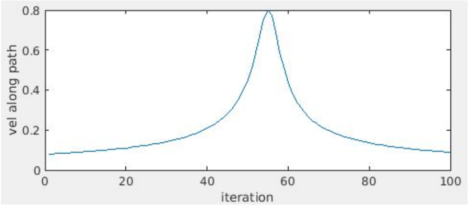
Generate Trajectories to Emulate Braitenberg Vehicles

Trajectories are generated recursively using simple laws of motion and perception.

7- Vehicle 1: linear movement, constant heading

- Starting position and orientation
  - Pi=(0,0), a=π/4
- Data source position
  - S=(10,5)
- Define law of motion with the distance:
  - $v=k/d^2$ (excitatory)
    - v=k*d would be inhibitory and the value of k has a lot of influence (it would have to be limited as it happens in other vehicles ahead)
  - d=‖P-S‖ (distance to source)
  - k - source sensitivity (e.g. k=10)
- Calculate next position recursively
  - P(n+1) = P(n) + v(n)*Δt
  - Δt = 1, for example
- Run for 100 iterations

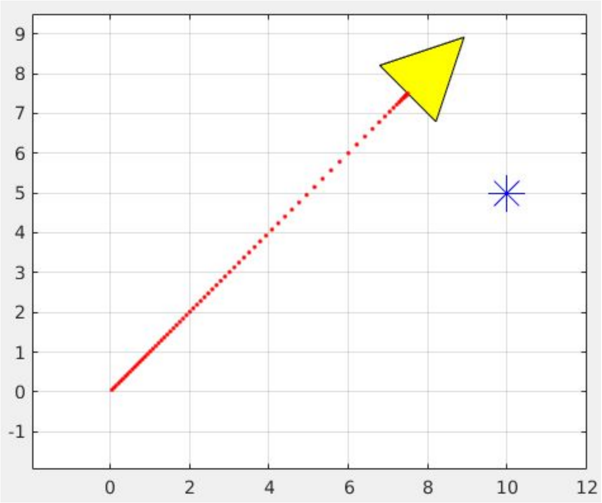




8- Vehicle 1: Considering the directionality of the sensor

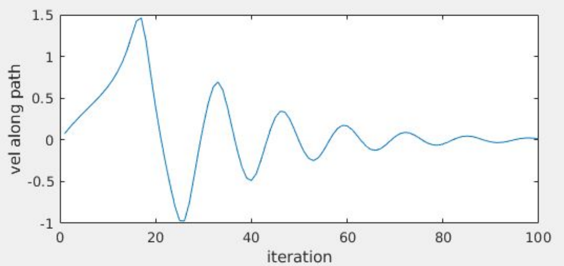
- Same parameters as before
- But take into account the direction of the source
- Only the along-track component is measured (anisotropic sensitivity of the on-board sensor).
- Relative source angle:
  - a2 = ∡(S-P) - a
- Speed adjustment:
  - v=v*cos(a2);

- The robot slows down and ends up stopping (zero speed) when the source appears in quadrature:



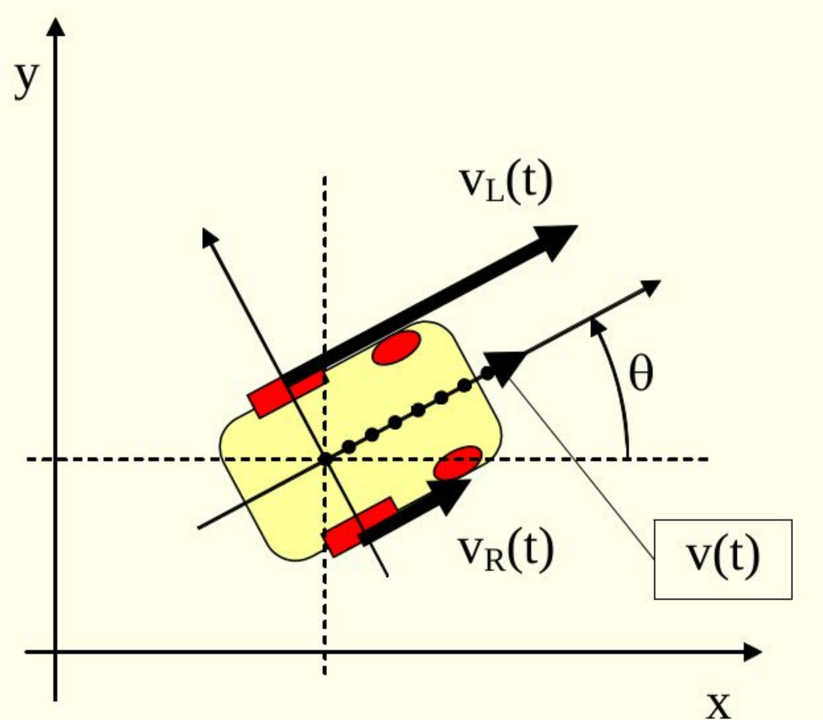
9- Vehicle 1: Adjust the law of motion with short memory

- The same as before
- But now add the previous speed to the current velocity.
- For example, adjust like this:
  - v ← v + ki*vprev;
  - vprev - velocity of previous iteration (initially zero)
  - ki - fraction of the previous velocity to be added (for example, ki=0.9)
  - ki < 1 for stability.
  - With higher ki, the velocity of the robot oscillates around the point closest to the source, generating a kind of feedback control.



Robot kinematics with differential drive

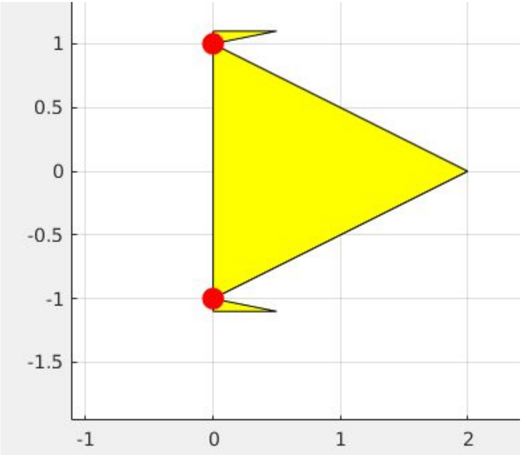
- L - Distance between wheels
- vR - Linear speed of the right wheel
- vL - Left wheel linear velocity
- v=(vL+vR)/2 - linear velocity of the robot
- ω=(vR-vL)/L - angular velocity of the robot
- Δt - sampling time
- Δl = v*Δt - linear displacement in Δt
- Δθ= ω*Δt - angular displacement in Δt
- Discrete update equations:
    - $\theta_{n+1}=\theta_n + \Delta\theta$
    - $x_{n+1}=x_n+\Delta l*\cos\theta_{n+1}$
    - $y_{n+1}=y_n+\Delta l*\sin\theta_{n+1}$



10a - Vehicle 2: Two motors and two sensors (Sn1, Sn2)

- Sensors now have their own placements that affect their perception.
  - Define their default position (Sn1 and Sn2) and the geometric transformation relative to the robot's reference frame (TS1 and TS2) to use later when changing the robot's position.

```
%             Sn1                 Sn2
%              |                   |
R=[ 0 0   0.5 0 2  0  0    0.5  0 %x
    1 1.1 1.1 1 0 -1 -1.1 -1.1 -1 %y
    ];

%Local placement of sensors
TS1=[1 0 0
     0 1 1
     0 0 1];
TS2=[1 0  0
     0 1 -1
     0 0  1];
Sn1=[0  1 1]'; %homog
Sn2=[0 -1 1]'; %homog
```



10b - Vehicle 2 - Simulating perception of Sn1 and Sn2

- The positions of the sensors on the world board must now be calculated after the robot's position and its relative position in the robot's reference system:
  - **P** - robot center position
  - **th** - robot orientation
  - **Sn1c** - Sn1 sensor position in the world frame
  - **Sn2c** - Sn2 sensor position in the world frame
- Thus, their distances to the source can be easily calculated:
  - **d1** - Euclidean distance from sensor Sn1 to data source S
  - **d2** - Euclidean distance from sensor Sn2 to data source S

```
Sn1c=transl(P)*rotat(th)*TS1*Sn1;
Sn2c=transl(P)*rotat(th)*TS2*Sn2;
d1=norm(Sn1c(1:2)-S);
d2=norm(Sn2c(1:2)-S);
```

10c - Vehicle 2A - Calculate velocities and movement

Calculate wheel velocities using the stimuli (distance to source)

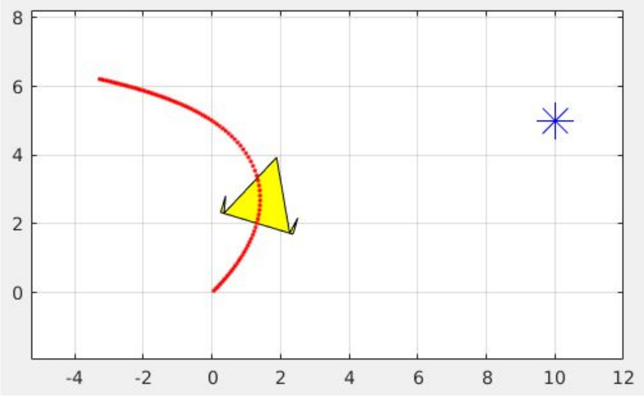
```
vL=k/d1^2;  % left wheel velocity
vR=k/d2^2;  % right wheel velocity
```

- Calculate the kinematics of the robot.
- For each simulation point, do something similar to the one indicated and store all P and th values that are the real trajectory of Vehicle 2A (direct excitatory connections).
- In the examples the value used for L is 2 (L=2)

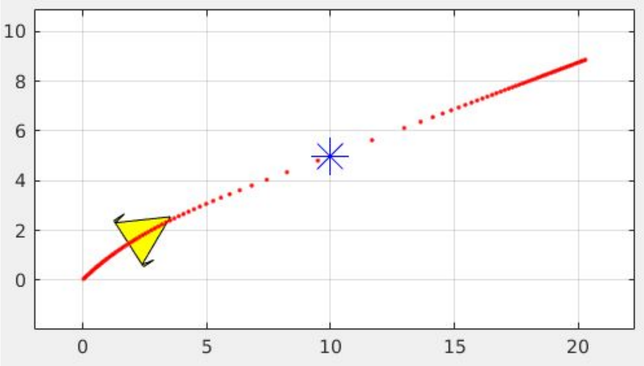
```
w=(vR-vL)/L;                  % angular velocity
v=(vR+vL)/2;                  % linear velocity (mean)

Dl=v*Dt;                      % linear displacement
Dth=w*Dt;                     % angular displacement
th=th+Dth;                    % new angular position
P=P+Dl*[cos(th); sin(th)];    % new xy position (approx.)
```

10d - Vehicle 2 - The “Fear” and “Aggression” behaviors

- Vehicle 2A, with direct excitatory connections avoids the source.

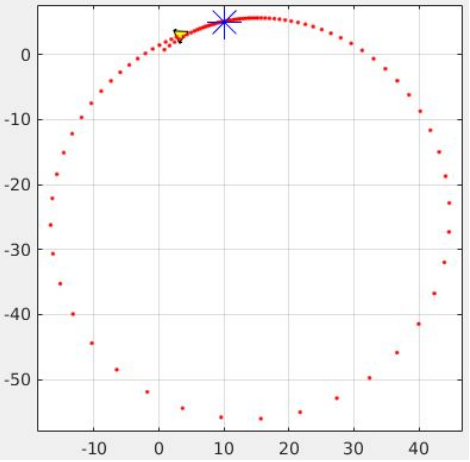


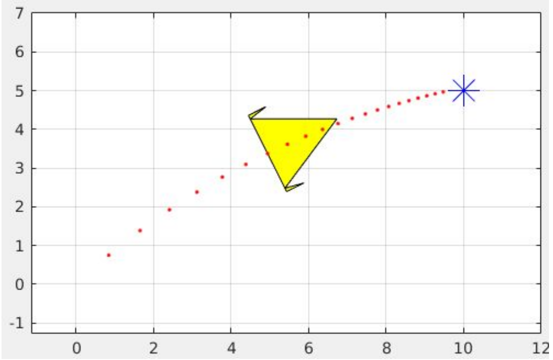
- Vehicle 2B, with excitatory cross connections, is heading towards the source.
- To get a cross connection just swap the left and right speeds:
    - `vT= vL; vL=vR; vR=vT;  % swap`
- To avoid overshooting the source, a condition can be added if the stimulus is above a certain value or, equivalently, when the distance from the source is small (e.g., d < L).



11 - Vehicle 3 - The inhibitory approach

- Vehicle 3 is identical to Vehicle 2 but the connections are inhibitory.
  - This means that the greater the stimulus, the lesser the action and vice versa.
  - To avoid extreme reactions in the absence of stimuli, a lower sensitivity should be used.
  - In the examples **k=0.1** compared to **k=10** in the previous examples
- When using distances as a stimulus, a minimum threshold must be set to prevent the robot from continuing without stopping!
- Two situations are illustrated.
  - One in which the robot passes the source but, as the stimulus has not been saturated, it continues and moves away while returning.
  - In the other, the stimulus was saturated and the robot stops near the source, whatever its starting point (the robot stops if d < L)





11b - Vehicle 3B - The most intense escape from the source

- Vehicle 3B uses inhibitory crosslinks
- Unless the source is directly facing it, it moves away from it intensely, even at low sensitivities!
- Here, too, the lack of stimulation must be saturated (long distances) to avoid exaggerated distancing.
    - Therefore, the speed is cut when the distance exceeds a limit value.