Case 2: Two cars with kinematic constraints avoiding each other

There are two important parts to the GVO algorithm. Prediction and Correction. There are a few steps before implementing prediction and correction. The following step by step report depicts our understanding of the GVO algorithm.

Let's call the two cars A and B, with the variables in the code ‘A’ and ‘B’ are the x,y, theta states of the respective cars for time ‘t’ and input ‘u’. The input for the kinematic simple car is the input pair (Us, Uphi) where Us is the velocity of the vehicle (ideally ranging from -1 to 1) and Uphi is the steering for the vehicle (ideally ranging from -pi/2 < Uphi< pi/2)

*Step 1: Data for problem setup*

1. A sample set of discrete inputs from the set of all continuous possible inputs your car can take.
2. For our problem we have taken this input set to be Us = 1 and Uphi ranging from -45 to 45 degrees. The reason for Us being always 1 will be explained in step 6.
3. An initial position A and B for the cars comprising of (x0, y0 and theta0) in the global frame of reference.
4. A goal position for the cars, indicating the position where they have to reach comprising of (xgoal, ygoal) in the global frame of reference.
5. An initial Ustar for the cars which are the inputs for which they will reach their respective goal positions without any dynamic or static obstacles in their path.
6. A radius for each car which is the radius of uncertainty outside which we want to keep the obstacles.
7. A prediction time with a step size. This is the time for which the car will predict its own and the obstacles trajectories to check for possible collision for a certain input. We took this value to be 5 seconds with a step size of 0.05 seconds.
8. A logic array ‘free’ for each car. This array is of the same length as your input sample vector which is initially assigned with all its cells being value ‘1’.

*Code for Step 1:*

t= 0:0.05:5; % prediction time (t limit)

Ainit= [0;0;0]; ustarA= [1,0]; Binit= [2;2;0]; ustarB= [0.8, 0]; %initial conditions for tsim=0

Agoal= [6;6]; Bgoal= [7;5]; %goal positions

rB= 2; rA= rB; % radius of uncertainty

uA= [ones(1,91);linspace(-45,45,91)]; uB= [ones(1,91);linspace(-45,45,91)]; %sample possible actions

freeA= ones(1,91); freeB= ones(1,91); % variable free for finding u dash

*Step 2: Prediction*

In this step, a loop is run for the same number of times as the row length of Usample matrix and for each input pair (Us and Uphi) it is predicted if the vehicles will collide in the next five seconds. This is done using the following steps:

* + - 1. The distance function is formulated by D2= (xA-xB)2+(yA-yB)2. This gives the velocity obstacle for the two agents.
      2. The minimum time is then found when velocity obstacle between the two agents is 0.
      3. Now at this minimum time, the distance between the two agents is found. This distance would be the most minimum distance that the two agents have between them outside the velocity obstacle.
      4. This distance is then checked whether it is lesser than the sum of the uncertainty radii of the two agents. If it is, then that particular input is an unacceptable one and if it is not, then the input is acceptable.
      5. The acceptance of the input is recorded in the logic array ‘free’ by changing the value of the cell corresponding to the input pair’s column value is assigned to be ‘0’.
      6. This ensures we now have a ‘free’ array giving us positions in the Usample matrix that are acceptable or unacceptable. (1 is acceptable and 0 is not).

*Code for Step 2:*

for i=1:n

[A,Adot]= car\_A(Ainit, uA(:,i), t);

[B,Bdot]= car\_B(Binit, uB(:,i), t);

for l=1:numel(t)

D(l)= sqrt(((A(1,l)- B(1,l))^2)+((A(2,l) - B(2,l))^2)); % distance square

%distance square dot for A

DsqrdotA(l)= (2\*(A(1,l)-B(1,l))\*(Adot(1,l)-Bdot(1,l)))+(2\*(A(2,l)-B(2,l))\*(Adot(2,l)-Bdot(2,l)));

%distance square dot for B

DsqrdotB(l)= (-2\*(A(1,l)-B(1,l))\*(Adot(1,l)-Bdot(1,l)))+(-2\*(A(2,l)-B(2,l))\*(Adot(2,l)-Bdot(2,l)));

if -tolerance<DsqrdotA(l)<tolerance %checking for near zeros in D square dot for A

dA= D(l);

if(dA<rA+rB)

freeA(i)=0;

end

end

if -tolerance<DsqrdotB(l)<tolerance %checking for near zeros in D square dot for B

dB= D(l);

if(dB<rA+rB)

freeB(i)=0;

end

end

end

end

Note: The ‘tolerance’ is a value above and below which D square dot can lie. Since it might never actually attain a zero-value due to numerical differentiation.

*Step 3: Updating sample set of U*

Now out of all possible discrete inputs, the inputs that are accepted need to be created as separate set. This set of U is called Udash. This set is assigned by taking advantage of our updated logic array ‘free’.

Udash is populated by picking U values from the columns corresponding to the positions at which the ‘free’ array has a value of 1.

*Code for Step 3:*

count1=1; count2= 1;

for k=1:n

if freeA(k)==1

udashA(:,count1)= uA(:,k); %eliminating all infeasible actions for A

count1=count1+1;

end

if freeB(k)==1

udashB(:,count2)= uB(:,k); %eliminating all infeasible actions for B

count2=count2+1;

end

end

*Step 4: Finding a feasible/optimal input for that time step.*

1. At this point, a single input pair Uoptimal needs to be chosen for the agents respectively which will lead them towards their safe navigation at that particular time step (The time step referred to here is not the prediction time step, but the simulation time step ‘tsim’, chosen to be 20 seconds with a step size of 1).
2. This is done by finding the argmin for the function f(Udash)= abs(Udash-Ustar)
3. Physically, this gives the input pair that is acceptable as well as gives least change in input from the ideal input Ustar.

*Code for Step 4:*

for q1=1:count1-1

%difference in udash and ustar for car A

udiffA(q1)= sqrt(((ustarA(1) -udashA(1,q1))^2) + ((ustarA(2) - udashA(2,q1))^2));

end

for q2=1:count2-1

%difference in udash and ustar for car B

udiffB(q2)= sqrt(((ustarB(1) -udashB(1,q2))^2) + ((ustarB(2) - udashB(2,q2))^2));

end

[uminA,I1]= min(udiffA); [uminB,I2]= min(udiffB); % finding minimum value and index for udash-ustar

uoptA= [1,udashA(2,I1)]; uoptB= [1, udashB(2,I2)]; % best action for A and B

*Step 5: Moving the car for the chosen optimal input*

By invoking the kinematic car function for Uoptimal the coordinates are found for a time of 1sec.

*Code for Step 5:*

[Aact(:,tsim), Adotact]= car\_A(Ainit,uoptA,tsim); %vehicle kinematics function car A

[Bact(:,tsim), Bdotact]= car\_B(Binit,uoptB,tsim); %vehicle kinematics function car B

xactA= Aact(1,tsim); yactA= Aact(2,tsim); % actual XY coordinates for car A for a feasible uoptA

xactB= Bact(1,tsim); yactB= Bact(2,tsim); % actual XY coordinates for car B for a feasible uoptB

*Step 6: Updating for next simulation time step*

The initial states and ideal input Ustar for the next time step’s prediction loop for Cars A and B are necessary to predict the motion for the next 5 seconds. This is done by:

Reassigning the initial position for the next time step as the last known position of the car.

Find the slope between the last known position of car and goal position given by

Slope= yG-ycurrent/xG-xcurrent

for each car and their respective goals.

The theta required for reaching goal is given by tan-1(slope)

For this theta, a new Uphi is calculated. Now the reason for choose Us to be always 1, mentioned in our Step1)b) of this report is because both Us and Uphi are dependent on theta which requires one of these values to be constant and obviously steering cannot be constant.

This new Uphi is then used to update Ustar (ideal input) for the next time step.

*Code for Step 6:*

%updating Ainit and Binit for next timestep

Ainit= Aact(:,tsim); Binit= Bact(:,tsim);

% theta required for reaching goal- Agoal and Bgoal respectively

thetanewA= atand((Agoal(2)-yactA)/(Agoal(1)-xactA));

thetanewB= atand((Bgoal(2)-yactB)/(Bgoal(1)-xactB));

% new ustar that is the ideal input to goal

ustarA= [1; atand(thetanewA)];

ustarB= [1; atand(thetanewB)];

*Step 7: End simulation when vehicle reaches goal.*

*Code for Step 7:*

if Aact(1,tsim)== Agoal(1) && Aact(2,tsim)== Agoal(2)

break

end