Lattice（下）-- 轨迹采样+轨迹评估+碰撞检测

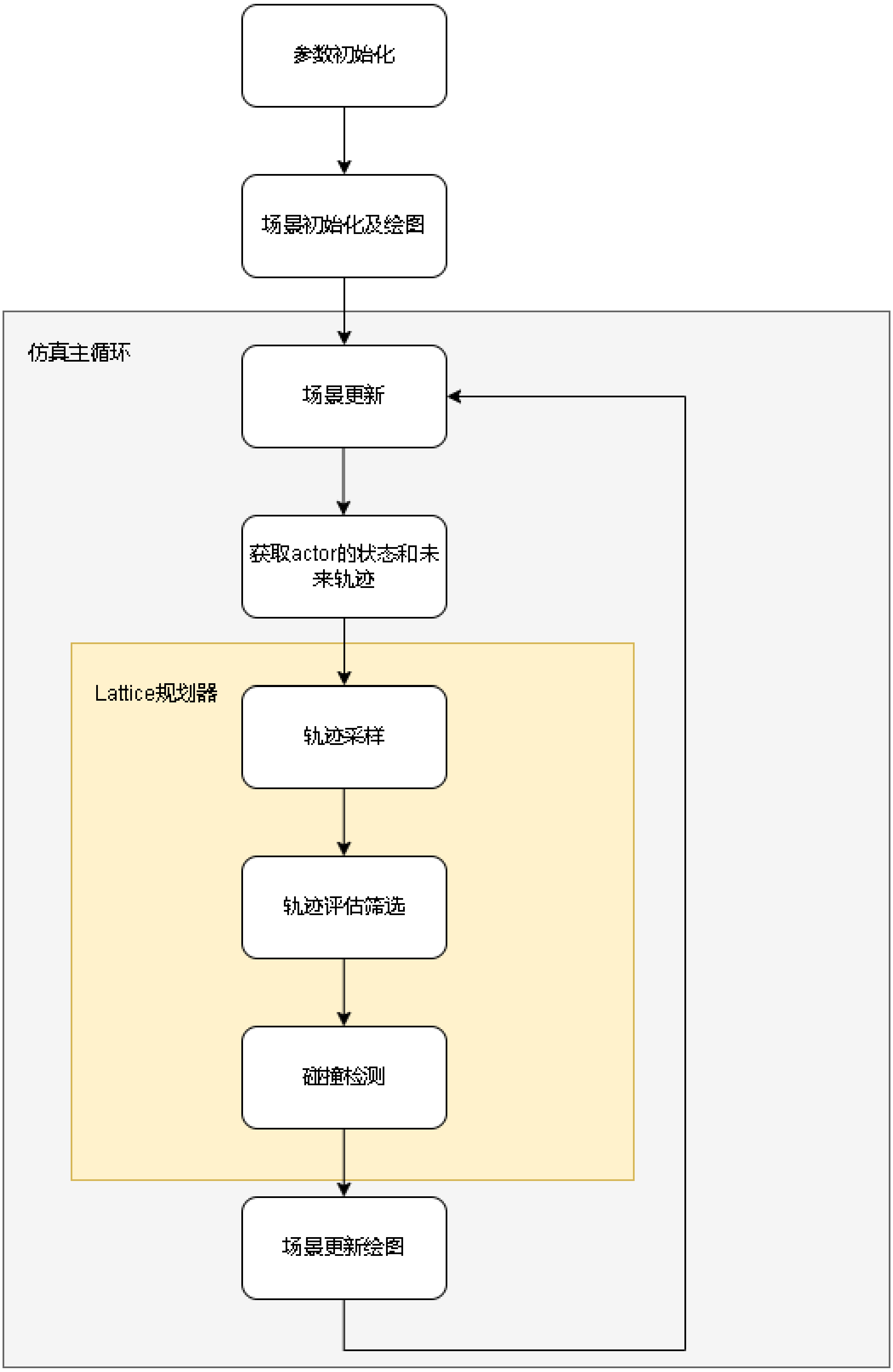
1 仿真环境

1.1 仿真程序框架

Lattice（上）篇已经讲解了参考线，frenet坐标，多项式拟合。

有了这些基础知识，就能够开始搭建一个基本的Lattice规划器了。这一篇来搭建这个规划器。

lattice规划器仿真流程：



完整代码

```
% lattice_planner_demo.m

clear;
% 车辆参数
carLen   = 4.7;
carWidth = 1.8;
carRear = 1.175;
% 场景
scenario = ScenarioEnv;
hold on;
```

```
scenario.show;
% 参考线
laneWidth = scenario.laneWidth;
refPath  = scenario.refPath;
connector = TrajectoryGeneratorFrenet(refPath);
scenario.SampleTime = connector.TimeResolution;
% 动态胶囊参数
Geometry.Length = carLen; % in meters
Geometry.Radius = carWidth/2; % in meters
Geometry.FixedTransform = -carRear; % in meters
% 规划参数
replanRate = 10; % Hz
scenario.replanRate = replanRate;
timeHorizons = 1:3; % in seconds
maxHorizon = max(timeHorizons); % in seconds
scenario.maxHorizon = maxHorizon;
latDevWeight    =  1;
timeWeight      = -1;
speedWeight     =  1;
maxAcceleration =  15; % in meters/second^2
maxCurvature    =   1; % 1/meters, or radians/meter
minVelocity     =   0; % in meters/second
speedLimit = 11; % in meters/second
safetyGap = 10; % in meters
% 初始化
numActors = 5;
% 非ego车辆当前位姿
actorPoses = repelem(struct('States',[]),numActors,1);
% 非ego车辆未来轨迹
futureTrajectory = repelem(struct('Trajectory',[]),numActors,1);
% ego起点
egoState = frenet2global(refPath,[0 0 0 -0.5*laneWidth 0 0]);
% 主循环
while true
    % 场景更新
    scenario.update(futureTrajectory);
    % 获取场景内Actor的信息
    [curActorState,futureTrajectory] = scenario.getActorInfo();
    % 轨迹终点状态采样
    [allTS, allDT, numTS] = SamplingEndcontions(refPath, laneWidth, egoState,
curActorState, safetyGap, speedLimit, timeHorizons);
    % 轨迹评估
    costTS = EvaluateTSCost(allTS,allDT,laneWidth,speedLimit,speedWeight,
latDevWeight, timeWeight);
    % global转frenet
    egoFrenetState = refPath.global2frenet(egoState);
    % 轨迹生成
    [frenetTraj,globalTraj] = connector.connect(egoFrenetState,allTS,allDT);
    % 轨迹筛选
    isValid =
EvaluateTrajectory(globalTraj,maxAcceleration,maxCurvature,minVelocity);
    % 碰撞检测--设置actor的未来轨迹
    for i = 1:5
        actorPoses(i).States = futureTrajectory(i).Trajectory(:,1:3);
    end
    % 按照代价对轨迹进行排序
    [cost, idx] = sort(costTS);
    % 开始轨迹
```

```
    optimalTrajectory = [];
    for i = 1:numel(idx)
        % 根据筛选的结果选取有效的轨迹
        if isValid(idx(i))
            % 设置ego的未来轨迹
            egoPoses.States = globalTraj(idx(i)).Trajectory(:,1:3);
            % 碰撞检测
            isColliding = checkTrajCollision(egoPoses, actorPoses, Geometry);
            if all(~isColliding)
                % 无碰撞，则找到最优的全局路径
                optimalTrajectory = globalTraj(idx(i)).Trajectory;
                break;
            end
        end
    end
    % 更新绘图
    scenario.updateShow(egoState, curActorState,globalTraj);
    if isempty(optimalTrajectory)
        % 如果没有找到轨迹，则报错
        error('No valid trajectory has been found.');
    else
        % 更新ego位置
        egoState = optimalTrajectory(2,:);
    end
    % 立即绘图
    drawnow;
    % 延时
    pause(0.1);
end
```

1.2 场景类

ScenarioEnv是场景类，用于场景仿真

主要实现功能：

1 保存actors的运动轨迹

2 不断更新actors的当前位置和未来轨迹

3 给ego和actors绘图并更新绘图

主要的属性和方法如下：

```
classdef ScenarioEnv < handle

    properties(Access=public)
        actors                    % actors信息
        refPath                   % 参考线
        laneWidth                 % 车道宽
        egop                      % ego绘图句柄
        carp                      % actors绘图句柄
        trajp                     % 轨迹绘图句柄
        SampleTime                % 采样时间
        replanRate                % 重规划频率
        maxHorizon                % 滚动窗口长度
        fig                       % 图窗figure句柄
        ax                        % 图窗axes句柄
        futureTrajectory          % actors的未来轨迹
        curState                  % actors的当前状态
    end
    methods
        function updateShow(obj, egoState, curActorState, globalTraj)% 更新绘图 ...
        function [curState, futureTrajectory] = getActorInfo(obj)% 获取actors的信息 ...
        function update(obj, futureTrajectory) %actors位置和未来轨迹更新 ...
        function show(obj) % 绘图 ...
        function obj = ScenarioEnv()% 构造函数 ...
    end


end
```

场景类的完整代码

```
classdef ScenarioEnv < handle

    properties(Access=public)
        actors                  % actors信息
        refPath                 % 参考线
        laneWidth               % 车道宽
        egop                    % ego绘图句柄
        carp                    % actors绘图句柄
        trajp                   % 轨迹绘图句柄
        trajopt                 % 最优轨迹绘图句柄
        SampleTime              % 采样时间
        replanRate              % 重规划频率
        maxHorizon              % 滚动窗口长度
        fig                     % 图窗figure句柄
        ax                      % 图窗axes句柄
        futureTrajectory        % actors的未来轨迹
        curState                % actors的当前状态
    end
    methods
        function updateShow(obj, egoState, curActorState, 
globalTraj,optimalTrajectory)% 更新绘图
            curpos = egoState(1:3);
            xy = [-2.5 -2.5 2.5 2.5
                1   -1  -1   1];
            M = [cos(curpos(3)), -sin(curpos(3));sin(curpos(3)) cos(curpos(3))];
            xy = M*xy+curpos(1:2)';
            set(obj.egop,'xdata',xy(1,:),'ydata',xy(2,:));
            for idx = 1:5
                curpos = curActorState(idx,1:3);
                xy = [-2.5 -2.5 2.5 2.5
                    1   -1  -1   1];
```

```
                M = [cos(curpos(3)), -sin(curpos(3));sin(curpos(3))
cos(curpos(3))];
                xy = M*xy+curpos(1:2)';
                set(obj.carp(idx),'xdata',xy(1,:),'ydata',xy(2,:));
            end

            for idx = 1:12
                if idx > length(globalTraj)
                    break;
                end
                set(obj.trajp(idx), 'xdata',globalTraj(idx).Trajectory(:,1),
'ydata',globalTraj(idx).Trajectory(:,2));
            end
            set(obj.trajopt, 'xdata',optimalTrajectory(:,1),
'ydata',optimalTrajectory(:,2));
        end
        function [curState, futureTrajectory] = getActorInfo(obj)% 获取actors的信息
            curState = obj.curState;
            futureTrajectory = obj.futureTrajectory;
        end
        function update(obj, futureTrajectory) %actors位置和未来轨迹更新
            % 获取非ego的位姿和未来轨迹
            numActor = numel(futureTrajectory);
            curState = zeros(numActor,6);
            minUpdateSteps = (1/obj.replanRate)/obj.SampleTime;
            maxNumStates = obj.maxHorizon/obj.SampleTime;
            statesNeeded = max(maxNumStates-
size(futureTrajectory(1).Trajectory,1),minUpdateSteps);
            for i = 1:statesNeeded
                for k = 1:5
                    obj.actors{k}.s = obj.actors{k}.s +
obj.SampleTime*obj.actors{k}.speed(1);
                end
                p1 = obj.actors{1}.refPath.interpolate(obj.actors{1}.s);
                poses1.Position = [p1(1:2),0];
                poses1.Velocity = obj.actors{1}.speed(1)*
[cos(p1(3)),sin(p1(3)),0];
                poses1.Yaw = p1(3)*180/pi;
                poses1.AngularVelocity =
[0,0,p1(4)*obj.actors{1}.speed(1)*180/pi];

                p1 = obj.actors{2}.refPath.interpolate(obj.actors{2}.s);
                poses2.Position = [p1(1:2),0];
                poses2.Velocity = obj.actors{1}.speed(1)*
[cos(p1(3)),sin(p1(3)),0];
                poses2.Yaw = p1(3)*180/pi;
                poses2.AngularVelocity =
[0,0,p1(4)*obj.actors{1}.speed(1)*180/pi];

                p1 = obj.actors{3}.refPath.interpolate(obj.actors{3}.s);
                poses3.Position = [p1(1:2),0];
                poses3.Velocity = obj.actors{1}.speed(1)*
[cos(p1(3)),sin(p1(3)),0];
                poses3.Yaw = p1(3)*180/pi;
                poses3.AngularVelocity =
[0,0,p1(4)*obj.actors{1}.speed(1)*180/pi];

                p1 = obj.actors{4}.refPath.interpolate(obj.actors{4}.s);
```

```
                poses4.Position = [p1(1:2),0];
                poses4.Velocity = obj.actors{1}.speed(1)*
[cos(p1(3)),sin(p1(3)),0];
                poses4.Yaw = p1(3)*180/pi;
                poses4.AngularVelocity =
[0,0,p1(4)*obj.actors{1}.speed(1)*180/pi];

                p1 = obj.actors{5}.refPath.interpolate(obj.actors{5}.s);
                poses5.Position = [p1(1:2),0];
                poses5.Velocity = obj.actors{1}.speed(1)*
[cos(p1(3)),sin(p1(3)),0];
                poses5.Yaw = p1(3)*180/pi;
                poses5.AngularVelocity =
[0,0,p1(4)*obj.actors{1}.speed(1)*180/pi];
                poses = [poses1,poses1,poses2,poses3,poses4,poses5];
                for j = 1:numActor
                    actIdx = j+1;
                    xy = poses(actIdx).Position(1:2);
                    v  = norm(poses(actIdx).Velocity,2);
                    th =
atan2(poses(actIdx).Velocity(2),poses(actIdx).Velocity(1));
                    k = poses(actIdx).AngularVelocity(3)/v/180*pi;
                    futureTrajectory(j).Trajectory(i,:) = [xy th k v 0];
                end
            end
            % Reorder the states
            for i = 1:numActor
                futureTrajectory(i).Trajectory =
circshift(futureTrajectory(i).Trajectory,-statesNeeded,1);
                curState(i,:) = futureTrajectory(i).Trajectory(1,:);
            end
            obj.futureTrajectory = futureTrajectory;
            obj.curState = curState;
        end
        function show(obj) % 绘图
            % 绘制场景
            hold on;
            ss = 0:1:obj.refPath.Length;
            len = length(ss);
            pp = obj.refPath.interpolate(ss);
            plot(pp(:,1),pp(:,2), '--');
            for i = 2:5
                frestate = [0, 0, 0, -obj.laneWidth/2+obj.laneWidth*(i-3), 0, 0];
                frestates = repmat(frestate, len, 1);
                frestates(:,1) = ss';
                pp1 = obj.refPath.frenet2global(frestates);
                plot(pp1(:,1),pp1(:,2), 'k','LineWidth',1);
            end
            axis equal;
            obj.fig = gcf;
            obj.ax = gca;
            obj.egop = patch(nan,nan,'r');
            obj.carp = zeros(5,1);
            for i = 1:5
                obj.carp(i) = patch(nan,nan,'g');
            end
            obj.trajp = zeros(12,1);
            for i = 1:12
```

```
                obj.trajp(i) = plot(nan,nan, 'marker','.');
            end
            title('Lattice Planner Demo');
            obj.trajopt = plot(nan,nan, 'marker','.', 'color','g','linewidth',3);
        end
        function obj = ScenarioEnv()% 构造函数
            waypoints = [0 50; 150 50; 300 75; 310 75; 400 0; 300 -50; 290 -50; 0
-50]; % in meters
            obj.laneWidth   = 3.6;
            obj.refPath = FrenetReferencePath(waypoints);

            % Add actors
            car1.Position = [34.7 49.3 0];
            car1.waypoints = [34.7 49.3 0;
                60.1 48.2 0;
                84.2 47.9 0;
                119 49.3 0;
                148.1 51.4 0;
                189.6 58.7 0;
                230.6 68 0;
                272.6 74.7 0;
                301.4 77.5 0;
                316.7 76.8 0;
                332.4 75.2 0;
                348.9 72.2 0;
                366.2 65.1 0;
                379.6 55.6 0];

            car1.speed = [10;10;10;10;10;10;10;10;10;10;10;10;10;10];

            car2.Position = [17.6 46.7 0];
            car2.waypoints = [17.6 46.7 0;
                43.4 45.5 0;
                71.3 43.8 0;
                102.3 43.5 0;
                123.5 45.5 0;
                143.6 47.4 0;
                162.4 50 0;
                198.5 61 0;
                241.1 70.1 0;
                272.3 74.1 0;
                292 76.6 0;
                312.8 77.2 0;
                350.3 75.2 0;
                362.5 70.4 0;
                375.9 63.3 0;
                390.7 49.9 0;
                401.3 33 0];
            car2.speed = [9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9];

            car3.Position = [62.6 51.9 0];
            car3.waypoints = [62.6 51.9 0;
                87.4 51.3 0;
                117.7 52.2 0;
                147.6 55 0;
                174.9 59.7 0;
                203.3 65.8 0;
                265 69.7 0;
```

```
                288.3 73.1 0;
                314.5 73.1 0;
                334.9 70.8 0;
                360 59.9 0];
            car3.speed = [6;6;6;6;6;6;6;6;6;6;6];

            car4.Position = [101.7 41.1 0];
            car4.waypoints = [101.7 41.1 0;
                124.6 42 0;
                148.5 43.9 0;
                171.9 48.2 0;
                197.1 52.8 0;
                222.3 58.5 0;
                252.4 64.4 0;
                281.4 68.5 0;
                307.7 69.5 0;
                329.9 68.2 0;
                352.7 62.8 0];
            car4.speed = [7;7;7;7;7;7;7;7;7;7;7];

            car5.Position = [251.3 75.6 0];
            car5.waypoints = [251.3 75.6 0;
                255.7 76.7 0];
            car5.speed = [0.01;0.01];
            car1.refPath = FrenetReferencePath(car1.waypoints(:,1:2));
            car2.refPath = FrenetReferencePath(car2.waypoints(:,1:2));
            car3.refPath = FrenetReferencePath(car3.waypoints(:,1:2));
            car4.refPath = FrenetReferencePath(car4.waypoints(:,1:2));
            car5.refPath = FrenetReferencePath(car5.waypoints(:,1:2));
            car1.s = 0;
            car2.s = 0;
            car3.s = 0;
            car4.s = 0;
            car5.s = 0;
            obj.actors = {car1,car2,car3,car4,car5};
        end
    end


end
```

2 轨迹采样

Lattice规划器就是通过采样很多条可能的轨迹，然后进行筛选评估寻优得到最优的轨迹的。

前面说过给定初始状态和终点状态，可以用五次（或者四次，纵向一般用四次）多项式来连接初始状态和终点状态，从而得到整条轨迹的数据。因此轨迹采样问题实际上就是给定起点状态，采样不同的终点状态。

终点状态采样在`SamplingEndcontions.m`中实现

```
function [allTS, allDT, numTS] = SamplingEndcontions(refPath, laneWidth,
egoState, curActorState, safetyGap, speedLimit, timeHorizons)
    % Generate cruise control states.
    [termStatesCC,timesCC] = SamplingBasicCruiseControl(...
```

```
        refPath,laneWidth,egoState,speedLimit,timeHorizons);

    % Generate lane change states.
    [termStatesLC,timesLC] = SamplingBasicLaneChange(...
        refPath,laneWidth,egoState,timeHorizons);

    % Generate vehicle following states.
    [termStatesF,timesF] = SamplingBasicLeadVehicleFollow(...
        refPath,laneWidth,safetyGap,egoState,curActorState,timeHorizons);

    % Combine the terminal states and times.
    allTS = [termStatesCC; termStatesLC; termStatesF];
    allDT = [timesCC; timesLC; timesF];
    numTS = [numel(timesCC); numel(timesLC); numel(timesF)];
end
```

实际上按照决策来分，一共采样了3类轨迹：巡航，变道，跟车，最后将它们汇总起来。

2.1 巡航轨迹采样

决策特性：保持固定速度

终点速度设置为巡航速度

终点位置不约束（此时纵向使用四次多项式拟合）

终点横向位移通过预测本车的未来车道来计算。

注意，这个特性很重要，当本车处在变道过程中的时候转成巡航状态，此时仍然能够完成这个变道过程

具体步骤：

step1：获取当前状态的frenet状态

step2：预测末状态所在的车道

step3：根据step2得到的末状态的车道，确定末状态横向位移

这样就能确定巡航决策的末状态。

```
function [terminalStates, times] = SamplingBasicCruiseControl(refPath, laneWidth,
egoState, targetVelocity, dt)
    % Convert ego state to Frenet coordinates
    frenetState = global2frenet(refPath, egoState);

    % Determine current and future lanes
    futureLane = PredictLane(frenetState, laneWidth, dt);

    % Convert future lanes to lateral offsets
    lateralOffsets = (2-futureLane+.5)*laneWidth;

    % Return terminal states
    terminalStates      = zeros(numel(dt),6);
    terminalStates(:,1) = nan;
    terminalStates(:,2) = targetVelocity;
    terminalStates(:,4) = lateralOffsets;
    times = dt(:);
end
```

2.2 变道轨迹采样

决策特性：变换一个车道

终点速度和当前速度保持一致

终点位置不约束（此时纵向使用四次多项式拟合）

终点横向位移通过相邻车道中心计算出来

具体步骤：

step1：获取当前状态的frenet状态。

step2：获取当前所在的车道。

step3：校验当前车道的左右相邻车道是否可用。

step4：获得变道决策的末状态横向位移。

```
function [terminalStates, times] = SamplingBasicLaneChange(refPath, laneWidth,
egoState, dt)
    if egoState(5) == 0
        terminalStates = [];
        times = [];
    else
        % Convert ego state to Frenet coordinates
        frenetState = global2frenet(refPath, egoState);

        % Get current lane
        curLane = PredictLane(frenetState, laneWidth, 0);

        % Determine if future lanes are available
        adjacentLanes = curLane+[-1 1];
        validLanes = adjacentLanes > 0 & adjacentLanes <= 4;

        % Calculate lateral deviation for adjacent lanes
        lateralOffset = (2-adjacentLanes(validLanes)+.5)*laneWidth;
        numLane = nnz(validLanes);

        % Calculate terminal states
        terminalStates = zeros(numLane*numel(dt),6);
        terminalStates(:,1) = nan;
        terminalStates(:,2) = egoState(5);
        terminalStates(:,4) = repelem(lateralOffset(:),numel(dt),1);
        times = repmat(dt(:),numLane,1);
    end
end
```

2.3 跟车轨迹采样

决策特性：跟随本车道前车。

终点速度和前车保持一致

终点位置与前车保持一定距离

终点加速度为0

横向状态和前车保持一致。

具体步骤:

step1: 获取当前状态的frenet状态。

step2: 获取当前所在的车道。

step3: 得到演员车的frenet状态

step4: 预测演员车未来的车道

step5: 找到未来和在ego同车道前方的演员车

step6: 根据ego同车道

```
function [terminalStates, times] = SamplingBasicLeadVehicleFollow(refPath,
laneWidth, safetyGap, egoState, actorState, dt)
    % Convert ego state to Frenet coordinates
    frenetStateEgo = global2frenet(refPath, egoState);

    % Get current lane of ego vehicle
    curEgoLane = PredictLane(frenetStateEgo, laneWidth, 0);

    % Get current and predicted lanes for each actor
    frenetStateActors = global2frenet(refPath, actorState);

    predictedActorLanes = zeros(numel(dt),size(actorState,1));
    for i = 1:size(actorState,1)
        predictedActorLanes(:,i) =
PredictLane(frenetStateActors(i,:),laneWidth,dt);
    end
    % For each time horizon, find the closest car in the same lane as
    % ego vehicle
    terminalStates = zeros(numel(dt),6);
    validTS = false(numel(dt),1);
    for i = 1:numel(dt)
        % Find vehicles in same lane t seconds into the future
        laneMatch = curEgoLane == predictedActorLanes(i,:)';

        % Determine if they are ahead of the ego vehicle
        leadVehicle = frenetStateEgo(1) < frenetStateActors(:,1);

        % Of these, find the vehicle closest to the ego vehicle (assume
        % constant longitudinal velocity)
        future_S = frenetStateActors(:,1) + frenetStateActors(:,2)*dt(i);
        future_S(~leadVehicle | ~laneMatch) = inf;
        [actor_S1, idx] = min(future_S);

        % Check if any car meets the conditions
        if actor_S1 ~= inf
            % If distance is greater than safety gap, set the terminal
            % state behind this lead vehicle
            if frenetStateEgo(1)+safetyGap < actor_S1
                ego_S1 = actor_S1-safetyGap;
                terminalStates(i,:) = [ego_S1 frenetStateActors(idx,2) 0
frenetStateActors(idx,4:5) 0];
                validTS(i) = true;
```

```
            end
        end
    end
    % Remove any bad terminal states
    terminalStates(~validTS,:) = [];
    times = dt(validTS(:));
    times = times(:);
end
```

2.4 车道预测

车道预测实现在`PredictLane.m`中

车道预测的作用是预测ego或者演员车在未来dt时刻的横向位移，从而计算出未来dt时刻所在的车道号。

首先，如果dt == 0，则只需要计算当前所在的车道即可。

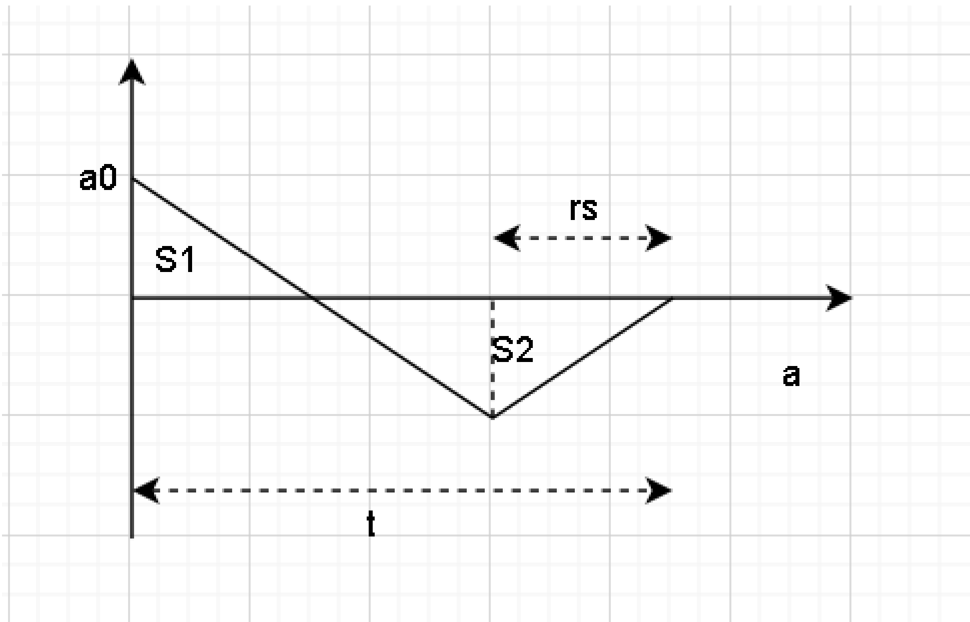
如果dt≠0，那么又分2种情况：

1 a0 = 0,初始横向加速度为0

此时假设它之后的横向运动是匀减速到0，这样计算出平均加速度，然后计算出横向位移

2 a0≠0,初始横向加速度不为0

此时假设它之后的横向运动是：横向加速度先减后加，最后到0，同时横向速度也变到0；



如图：初始加速度a0，按照一定的加速度变化率-da，降低到t-rs时刻，再以加速度变化率da上升到0

这样就保证t时刻加速度为0，同时，要保证t时刻速度也为0；初始速度为v0，那么v0 + S1 = S2

列出公式：

$$v_0+\frac{1}{2}a_0(t-2r_s)=r_s^2*d_a$$
$$\text{其中}\, d_a*(t-2r_s)=a_0$$
$$\text{消去}\, d_a$$
$$\frac{(-2a_0)r_s^2+(4v_0+4a_0t)r_s-a_0t^2-2v_0t}{4r_s-2t}=0$$
$$\text{消去}\, \frac{-2}{4r_s-2t}$$
$$a_0r_s^2-(2v_0+2a_0t)r_s+\frac{1}{2}a_0t^2+v_0t=0$$

求解这个二次方程即可得到rs，然后就能计算出最终的横向位移了。

```
function laneNum = PredictLane(frenetState, laneWidth, dt)
    laneBounds = [inf (2:-1:-2)*laneWidth -inf];
    laneNum = zeros(numel(dt),1);

    for i = 1:numel(dt)
        if dt(i) == 0
        %  计算当前的车道号即可
            dLaneEgo = laneBounds-frenetState(4);
            laneNum(i) = min(find(dLaneEgo(2:(end-1)) >= 0 & dLaneEgo(3:(end)) <
0,1),4);
        else
            % Retrieve current velocity/acceleration/time
            t  = dt(i);
            a0 = frenetState(6);
            v0 = frenetState(5);
            % Solve for the constant change in acceleration and time of
            % application that arrest the ego vehicle's lateral
            % velocity and acceleration over a given number of seconds.
            % 如果当前的横向加速度为0
            % 则考虑未来dt时间内，会匀减速减到0
            % 这时候，可以计算出dt时间后的横向位移
            if a0 == 0
                avgAcc = -v0/t;
                Ldiff = v0*t + avgAcc/2*t^2;
            else
                % 如果当前的横向加速度不为0
                % 则需要根据终点加速度和终点速度为0这一条件来计算加速度变化率切换的时间点
                % 是一个二次方程，用求根公式求解
                a = a0;
                b = (-2*v0-2*a0*t);
                c = (v0*t+a0/2*t^2);

                % Possible time switches
                r = (-b+(sqrt(b^2-4*a*c).*[-1 1]))/(2*a);

                % Select the option that occurs in the future
                rS = r(r>0 & r <= t);

                % Calculate the constant change in acceleration
                da0 = a0/(t-2*rS);

                % Predict total distance traveled over t seconds
```

```
                Ldiff = v0*t + a0/2*t^2 + da0/6*t^3 - da0/6*(t-rS)^3;
            end
            % Find distance between predicted offset and each lane
            dLaneEgo = laneBounds-(frenetState(4)+Ldiff);

            % Determine future lane
            laneNum(i) = min(find(dLaneEgo(2:(end-1)) >= 0 & dLaneEgo(3:(end)) <
0,1),4);
        end
    end
end
```

3 轨迹评估

轨迹采样完成之后，就进行轨迹评估。

轨迹评估包括轨迹代价评估和轨迹筛选。

3.1 轨迹代价评估

这里面轨迹评估比较简单，只对终点状态进行评估，因此轨迹代价评估在轨迹点生成之前。

评估的维度包括：

1 车道中心偏离代价， 当前终点状态到最近的车道中心的距离

2 时间代价

3 终点速度代价

```
function costs = EvaluateTSCost(terminalStates, times, laneWidth, speedLimit,
speedWeight, latWeight, timeWeight)
    % Find lateral deviation from nearest lane center
    laneCenters = (1.5:-1:-1.5)*laneWidth;
    latDeviation = abs(laneCenters-terminalStates(:,4));

    % Calculate lateral deviation cost
    latCost = min(latDeviation,[],2)*latWeight;

    % Calculate trajectory time cost
    timeCost = times*timeWeight;

    % Calculate terminal speed vs desired speed cost
    speedCost = abs(terminalStates(:,2)-speedLimit)*speedWeight;

    % Return cumulative cost
    costs = latCost+timeCost+speedCost;
end
```

3.2 轨迹筛选

轨迹筛选在轨迹点生成之后，也就是轨迹的笛卡尔坐标点计算出来以后。

校验每一条轨迹是否满足约束，不满足约束则标记为无效。

每条轨迹需要校验以下约束：

1 最大加速度约束

2 最大曲率约束

3 最小速度约束

```
function isValid = EvaluateTrajectory(globalTrajectory, maxAcceleration,
maxCurvature, minVelocity)
    isValid = true(numel(globalTrajectory),1);
    for i = 1:numel(globalTrajectory)
        % Acceleration constraint
        accViolated  = any(abs(globalTrajectory(i).Trajectory(:,6)) >
abs(maxAcceleration));

        % Curvature constraint
        curvViolated = any(abs(globalTrajectory(i).Trajectory(:,4)) >
abs(maxCurvature));

        % Velocity constraint
        velViolated  = any(globalTrajectory(i).Trajectory(:,5) < minVelocity);

        isValid(i) = ~accViolated && ~curvViolated && ~velViolated;
    end
end
```

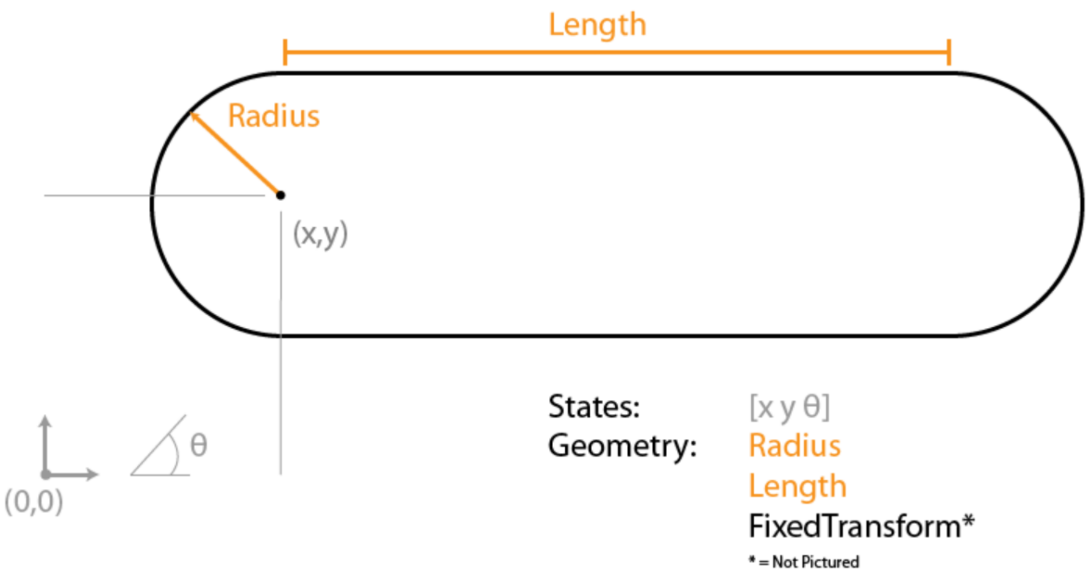
4 碰撞检测

碰撞检测通过动态胶囊来实现.

胶囊作为车辆或者障碍物的碰撞边界的好处是计算速度快，计算2个胶囊是否碰撞只需要计算2条线段的距离即可。

4.1 胶囊边界

车辆边界用一个胶囊来包围：

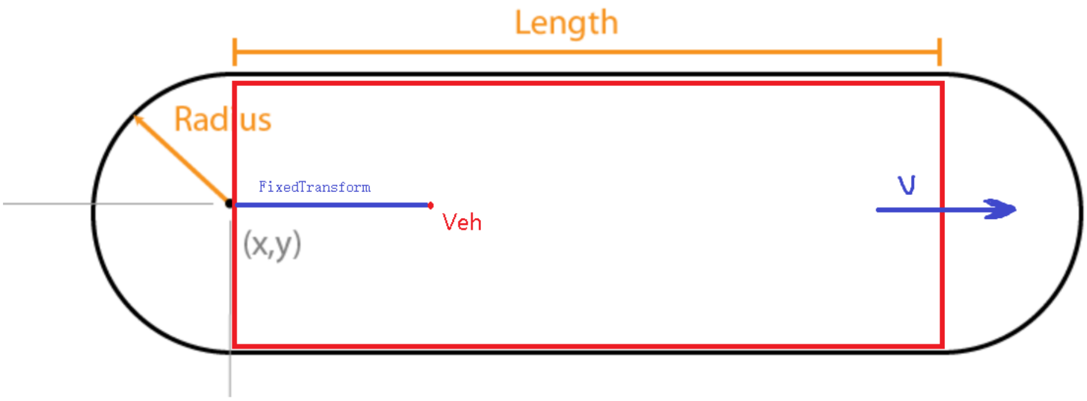


胶囊的几何形状通过3个参数来定义，1 长度 2 半径 3 长度方向的偏移。

胶囊中的线段通过起点坐标，方向，长度来定义。

这样，我们知道车辆的位姿以后，就能够确定胶囊中线段，并且胶囊方向和车辆方向一致。

车辆位姿和胶囊一一对应。



4.2 胶囊碰撞分配

前面说到一个轨迹点的边界用1个胶囊来表示，那么一段轨迹的边界就用一个胶囊集合来表示（轨迹点之间时间间隔相同）。那么检测2段轨迹是否碰撞，只需要检测2个胶囊集合是否会碰撞。

程序中碰撞检测代码：

重点是checkOneTrajCollision，检测ego的轨迹和单个演员车的轨迹是否碰撞

它的步骤是，首先分别计算出ego轨迹的胶囊集合和演员车轨迹的胶囊集合，然后调用checkCollisionCapsule来判断2个胶囊集合是否会碰撞。

```
function isColliding = checkTrajCollision(egoPoses, actorPoses, Geometry)
isColliding = true;
% 遍历每一个演员车，检查ego的轨迹和演员车的轨迹是否碰撞
for i = 1:length(actorPoses)
    if checkOneTrajCollision(egoPoses, actorPoses(i), Geometry)
        return;
    end
end
isColliding = false;
end

% 检测ego的轨迹和单个演员车的轨迹是否碰撞
function isColliding = checkOneTrajCollision(egoPoses, actorPoses, Geometry)
% Geometry.Length = carLen; % in meters
% Geometry.Radius = carWidth/2; % in meters
% Geometry.FixedTransform = -carRear; % in meters
len1 = size(egoPoses.States,1);
len2 = size(actorPoses.States,1);
p1 = zeros(2, 31);
v1 = zeros(2, 31);
p2 = zeros(2, 31);
v2 = zeros(2, 31);
D1 = Geometry.Length;
R1 = Geometry.Radius;
D2 = D1;
R2 = R1;
for i = 1:31
    if i > len1
```

```
        v1(:,i) = v1(:,i-1);
        p1(:,i) = p1(:,i-1);
    else
        v1(:,i) = [cos(egoPoses.States(i,3)); sin(egoPoses.States(i,3))];
        p1(:,i) = egoPoses.States(i,1:2)'+Geometry.FixedTransform*v1(:,i);
    end
end
for i = 1:31
    if i > len2
        v2(:,i) = v2(:,i-1);
        p2(:,i) = p2(:,i-1);
    else
        v2(:,i) = [cos(actorPoses.States(i,3)); sin(actorPoses.States(i,3))];
        p2(:,i) = actorPoses.States(i,1:2)'+Geometry.FixedTransform*v2(:,i);
    end
end
[isCollidings, ~] = checkCollisionCapsule(p1, v1, D1, R1, p2, v2, D2, R2);
isColliding = any(isCollidings);
end
```

`checkCollisionCapsule.m`

checkCollisionCapsule函数分为2个阶段

1 进行碰撞检测分配，就是对2个胶囊集合，分配出胶囊两两之间进行碰撞的组合。

碰撞检测分配有2个模式：

简化模式：

假设待检测的2个胶囊集合分别为$A_i, i=[1,...,N]$和$B_i, i=[1,...,M]$，并且B集合的胶囊个数是A集合的倍数关系，即M是N的整数倍。此时分配的两两检测组合关系是：

$$
\begin{gathered}
A_1 - B_1 \\
A_2 - B_2 \\
\ldots \\
A_N - B_N \\
A_1 - B_{N+1} \\
A_2 - B_{N+2} \\
\ldots \\
A_N - B_{2*N} \\
\ldots \\
A_N - B_M
\end{gathered}
$$

完全模式：

完全模式的两两检测组合关系是：

$$
\begin{gathered}
A_1 - B_1 \\
A_2 - B_1 \\
\dots \\
A_N - B_1 \\
A_1 - B_2 \\
A_2 - B_2 \\
\dots \\
A_N - B_2 \\
\dots \\
A_N - B_M
\end{gathered}
$$

完全模式计算的结果更加准确，简化模式计算的速度更快。

```
function [collisionFound, distance] = checkCollisionCapsule(p1, v1, D1, R1, p2,
v2, D2, R2, exhaustive)
    numSeg1 = size(p1,2);
    numSeg2 = size(p2,2);
    dim = size(p1,1);

    if numSeg1 == 0
        collisionFound = false(0,1);
        distance = [];
        return;
    end

    if numSeg2 == 0
        collisionFound = false(numSeg1,1);
        if nargout == 2
            distance = inf(numSeg1,1);
        else
            distance = [];
        end
        return;
    end

    if nargin > 8 && exhaustive == true
        % The first set of line-segments will be checked exhaustively
        % against the second set

        % Calculate values needed to vectorize operations
        numChecks = numSeg1*numSeg2;

        % Replicate vectors
        v = repmat(v1.*repelem(D1,dim,1),1,numSeg2);
        u = repelem(v2.*repelem(D2,dim,1),1,numSeg1);

        % Calculate distance between beginning of line-segment pairs
        w0 = repelem(p2,1,numSeg1)-repmat(p1,1,numSeg2); % Vector from lineSeg1-
base to lineSeg2-base

        % Calculate distance thresholds for each pair of radii
        combinedRSquared = (repmat(R1(:)',1,numChecks/numel(R1)) +
repelem(R2(:)',1,numChecks/numel(R2))).^2;

        % Calculate height dimension of output
        outputHeight = numSeg1;
```

```
    else
        % Each line-segment in xy1's ith column will be checked against
        % the set of line segments in xy2, corresponding to columns
        % i:size(xy1,2):end

        % Calculate the number of times set1 must be checked against set2
        numObj = numSeg2/numSeg1;
        validateattributes(numObj,{'numeric'},
{'integer','positive'},'collisionCheckCapsuleVectorized','LineSegRatio');
        outputHeight = size(p1,2);

        % Replicate vectors
        v = repmat(v1.*D1,1,numObj);
        u = v2.*repelem(D2,dim,1);

        % Calculate distance between beginning of line-segment pairs
        w0 = p2-repmat(p1,1,numObj);

        % Calculate distance thresholds for each pair of radii
        combinedRSquared = (repmat(R1(:)',1,numObj*numSeg1/numel(R1))+R2).^2;

        numChecks = size(p1,2)*numObj;
    end

    a = sum(u.*u); % Unit Vector
    b = sum(v.*u);
    c = sum(v.*v); % Unit Vector
    d = sum(u.*w0);
    e = sum(v.*w0);

    % Find parametric values for nearest points on each line segment
    [tC, sC] = findNearestPoints(numChecks, a, b, c, d, e);

    % Calculate closest distance between pairs of line segments
    squareDist = sum((w0 + repmat(sC,dim,1).*u - repmat(tC,dim,1).*v).^2);

    % If distance is below the bounding radii of the two capsules, a
    % collision has occurred
    collisionFound = reshape(squareDist(:) <= combinedRSquared(:), outputHeight,
[]);

    if nargout == 2
        distance = reshape(sqrt(max(squareDist(:),0))-sqrt(combinedRSquared(:)),
outputHeight, []);
        distance(distance<0) = nan;
    end
end
```

2 第2阶段就是循环寻找2个胶囊的最短距离，下一节介绍。

4.3 胶囊碰撞检测

2个胶囊的碰撞检测可以通过计算2个胶囊的线段的最短距离，然后和2个胶囊的半径和进行比较来完成。

这部分实现在findNearestPoints函数。

这里介绍寻找2线段最短距离的算法思想：

首先用参数方程来描述2个线段，然后计算2条线段是否相交。

如果相交，则距离为0.

如果不相交，则分别计算端点到另一条线段的最短距离，4个距离中取最小值即可。

matlab的实现中则是更加巧妙，它根据线段和线段之间的位置关系进行了细分，然后计算最短距离。

下面分析代码，首先看函数findNearestPoints的参数

```
function [tC, sC] = findNearestPoints(numChecks, a,b,c,d,e)
```

这里a b c d e分别是什么，需要看它是如何被调用的

在checkCollisionCapsule函数中

p1 v1 D1 R1分别表示胶囊1的起点位置，方向向量，长度，半径

p2 v2 D2 R2分别表示胶囊1的起点位置，方向向量，长度，半径

```
function [collisionFound, distance] = checkCollisionCapsule(p1, v1, D1, R1, p2, v2, D2, R2, exhaustive)
```

findNearestPoints调用时的代码

```
    v = repmat(v1.*D1,1,numObj);
    u = v2.*repelem(D2,dim,1);
    % Calculate distance between beginning of line-segment pairs
    w0 = p2-repmat(p1,1,numObj);
    ...
    a = sum(u.*u); % Unit Vector
    b = sum(v.*u);
    c = sum(v.*v); % Unit Vector
    d = sum(u.*w0);
    e = sum(v.*w0);
```

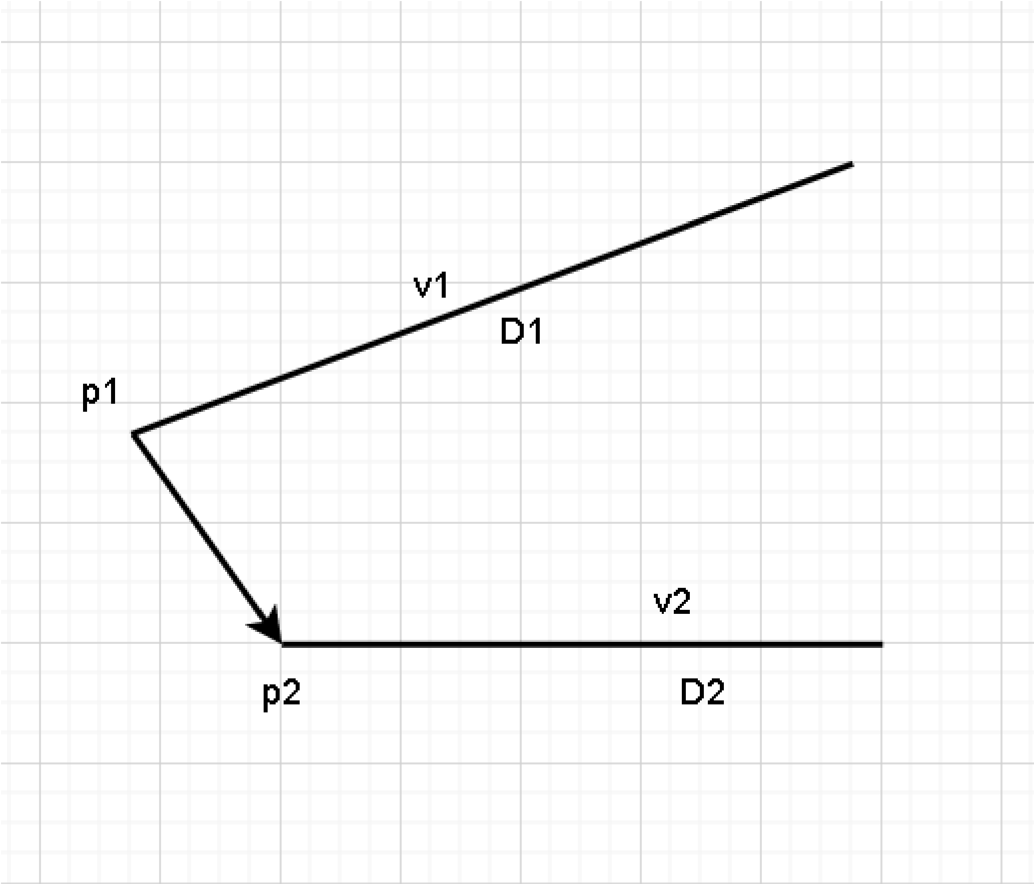
这里需要解释一下为什么要这么做。

下图是2条线段的表示：

w0 = p2-p1,表示线段1起点到线段2起点的向量。

v = v1*D1 表示线段1起点到终点的向量

u = v2*D2表示线段2起点到终点的向量



那么线段1上的点用参数方程表示:

```
p1 + v*t1
```

线段2上的点用参数方程表示:

```
p2 + u*t2
```

二者的交点方程:

```
p1 +v*t1 = u*t2 + p2
即:  vt1 - ut2 = w0
```

上面方程是一个向量方程，需要化成一般的线性方程才行

1) 左右同时点乘一个u

```
u.*v t1 - u.*u t2 = u.*w0
即 b*t1 - a*t2 = d
```

2) 左右同时点乘一个v

```
v.*v t1 - v.*u t2 = v.*w0
c*t1 - b*t2 = e
```

联列:

```
b*t1 -a*t2 = d
c*t1 -b*t2 = e
```

上面二元一次方程组的系数都是a b c d e中的数。因此，解释了findNearestPoints函数的参数为什么要这么取。

下面再来看最短距离怎么求。

先求解上面二元一次方程组的根：

```
t1 = (ae-bd)/(ac-b^2) = tC/denom
t2 = (be-cd)/(ac-b^2) = sC/denom
```

接下来就是分情况讨论了。

首先赋值 sD = denom

1）如果sD < sqrtEps,意味着2条线段所在的直线平行或者共线。

2）如果sC < 0 ,则交点在线段2起点以外

3）如果sC > sD,则交点在线段2终点以外

4）其他情况,交点就在线段2上

针对上面每一种情况又需要对交点在线段1的情况来分情况讨论。

由于需要讨论的情况比较多，这里选取2种情况详细讨论，其余的情况可以自行研究。

这段程序写的非常精妙，需要花很多时间去研读，如果只是应用的话可以直接复用这段程序。

1.A) 上面第一种情况的前提下：tC < 0线段2起点在线段1上的投影点在线段1起点之外的情况

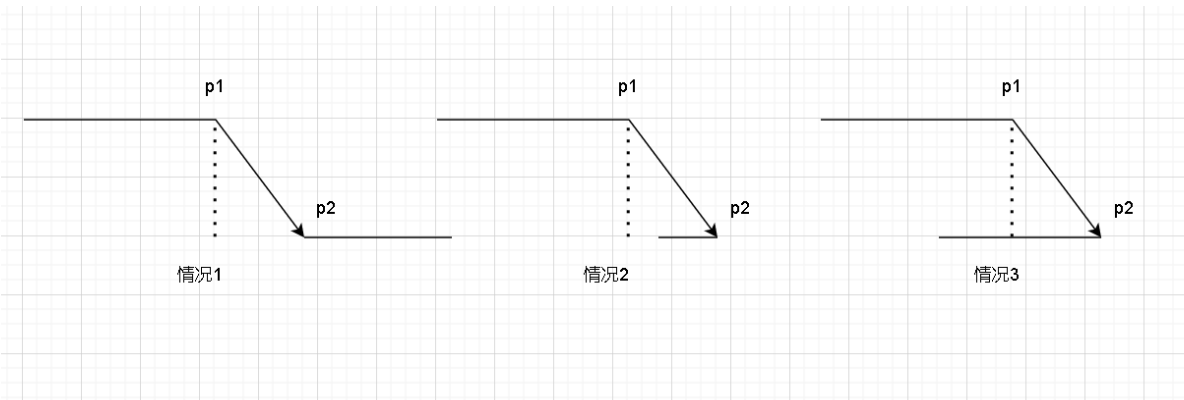
取sC = 0, sD = 1; tC = e, tD = c;

tC = v*w0 即线段2起点在线段1的投影，tC < 0，说明投影点在线段1的起点之外。

此时，tC = 0, 就是线段1取点

-d = -u*w0，是-w0在u上的投影，亦即线段1起点在线段2上的投影

又分3种情况



情况1 sC = 0, sD = denom ,:

t1 = tC/dD = 0,

t2 = sC/sD = 0

即线段1起点到线段2起点距离为最近距离

情况2 sC = sD = denom:

t1 = tC/dD = 0;

t2 = sC/sD = 1;

即线段1起点到线段2终点的距离为最近距离

情况3 sC = -d,sD = a :

最后结果:

t1 = tC/dD = 0,

t2 = sC/sD = -d/a = -(u*w0) / (u * u)

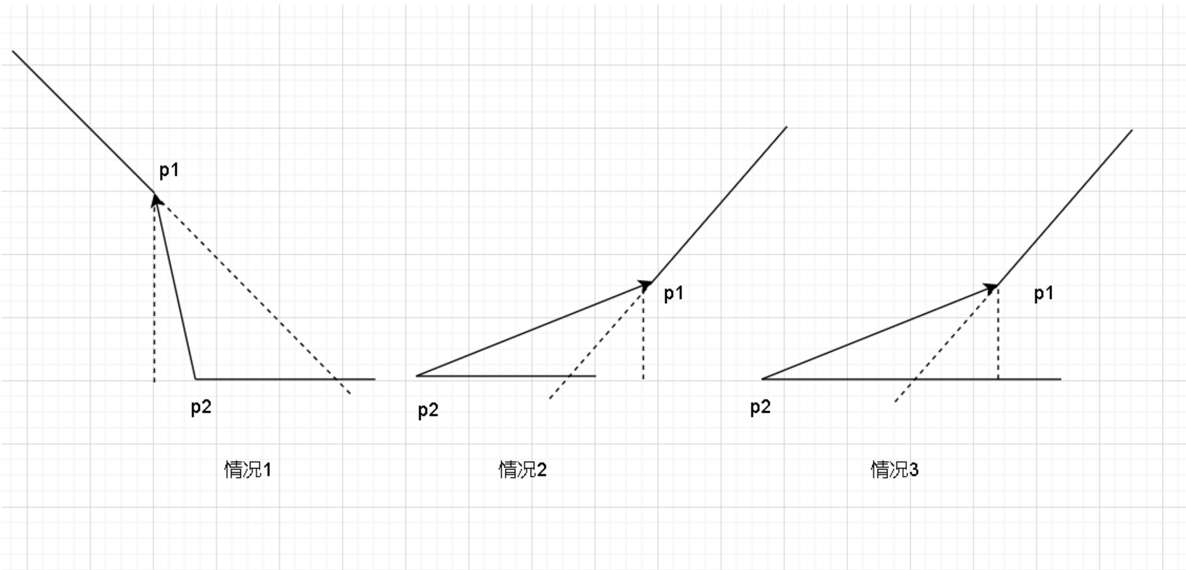
即线段1起点到线段2所在直线的距离为最近距离。

4.A) 上面第4种情况的前提下：tC < 0 线段交点在线段1起点外面的情况

此时，取tC = 0，就是线段1取起点

-d = -u*w0，是-w0在u上的投影，亦即线段1起点在线段2上的投影

又分3种情况



情况1 sC = 0, sD = denom ,:

t1 = tC/dD = 0,

t2 = sC/sD = 0

即线段1起点到线段2起点距离为最近距离。

情况2 sC = sD = denom:

t1 = tC/dD = 0;

t2 = sC/sD = 1;

即线段1起点到线段2终点的距离为最近距离。

情况3 sC = -d,sD = a :

最后结果:

t1 = tC/dD = 0,

t2 = sC/sD = -d/a = -(u*w0) / (u * u)

即线段1起点到线段2所在直线的距离。

剩下的情况就不一一详细讨论了。

总之最后求出了线段间的最短距离，然后再计算2个胶囊的半径之和，就能判定2个胶囊是否碰撞了。

```
function [tC, sC] = findNearestPoints(numChecks, a,b,c,d,e)
%findNearestPoints Constrains the parametric variables to the length
%of each line segment to find the nearest points on each line

    % Calculate numerator and denominator for the parameters of the line-segments
    denom = a.*c - b.*b;
    tC = a.*e-b.*d;
    sC = b.*e-c.*d;

    % Precalc our floating-point check rather than repeatedly calculate
    sqrtEps = sqrt(eps);

    for i = 1:numChecks
        % Constrain parameter S of line 2 so that it falls within the
        % bounds of the line segment and recalculate T
        sD = denom(i);
        if sD < sqrtEps
            % Degenerate case where lines are collinear or parallel
            sC(i) = 0;
            sD = 1;
            tC(i) = e(i);
            tD = c(i);
        elseif sC(i) < 0
            sC(i) = 0;
            tC(i) = e(i);
            tD = c(i);
        elseif sC(i) > sD
            sC(i) = sD;
            tC(i) = e(i)+b(i);
            tD = c(i);
        else
            tD = denom(i);
        end

        % Constrain the parameter T such that it falls within the bounds of
        % line-seg1 while minimizing the distance between line 1 and line 2
        if tC(i) < 0
            tC(i) = 0;
            if -d(i) < 0
                sC(i) = 0;
            elseif -d(i) > a(i)
                sC(i) = sD;
            else
                sC(i) = -d(i);
                sD = a(i);
            end
        elseif tC(i) >= tD
            tC(i) = tD;
            if (-d(i) + b(i)) < 0
                sC(i) = 0;
            elseif (-d(i) + b(i) > a(i))
                sC(i) = sD;
            else
                sC(i) = -d(i) + b(i);
                sD = a(i);
```

```
            end
        end

        % Recalculate the parameters using the updated numerator and denominator
        if sC(i) < sqrtEps
            sC(i) = 0;
        else
            sC(i) = sC(i)/sD;
        end
        if tC(i) < sqrtEps
            tC(i) = 0;
        else
            tC(i) = tC(i)/tD;
        end
    end
end
```