EE 7560 Homework 8

Shaopan Guo (gshaop1@lsu.edu)

April 10, 2021

0.1 Problem Formulation

The motion of the vehicle i for $i = 1, \ldots, N$ is governed by the usual longitudinal dynamics model

$$\begin{cases} \dot{p}_i = & v_i, \\ M\dot{v}_i = & F_{mot_i} - c_2 - c_3 v_i^2 - Mg\sin(\alpha_i), \end{cases} \tag{1}$$

where M is the total mass of the vehicle, α is the road slope, g is the gravitational acceleration, F_{mot} represents the traction force generated by the electric motor, and c_2, c_3 are parameters to be identified.

The torque is assumed to be propotional to the armature current, I_a and - because of the fixed-gear arrangement - the same is true for the traction force generated by the motor at the wheels:

$$F_{mot_i} = c_1 I_{a_i}. \tag{2}$$

The battery current I_{b_i} is defined as

$$I_{b_i} = \gamma I_{a_i}, \tag{3}$$

where γ is the conversion rate. The net electrochemical power (i.e. the one that corresponds to the actual battery charge or discharge) is given by

$$P_{b_i} = I_{b_i} E, \tag{4}$$

where voltage source E is assumed to be constant. Define

$$\dot{x} = \begin{bmatrix} x_{1_i} \\ x_{2_i} \end{bmatrix} = \begin{bmatrix} p_i \\ v_i \end{bmatrix}. \tag{5}$$

Then we can write (1) - (3) into the following state-space form:

$$\begin{cases} \dot{x}_{1_i} = & x_{2_i}, \\ \dot{x}_{2_i} = & -\dfrac{c_3}{M} x_{2_i}^2 + \dfrac{c_1}{M\gamma} I_{b_i} - \dfrac{c_2}{M} - g\sin\alpha_i. \end{cases} \tag{6}$$

It is worthy to mention that the real input is the motor torque, while the design is to control the derivative of the current. Once we have I_{b_i}, the motor torque can be calculated using following equation:

$$T_{mot_i} = \frac{c_1}{\gamma} I_{b_i} r_w,$$

where r_w is the wheel radius.

0.2 Path Planning

Here we use a digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ to describe the map, where the elements in the vertex set $\mathcal{V} = \{\nu_1, \ldots, \nu_N\}$ represent intersections that the electric vehicle may pass through in the journey, the elements in the edge set

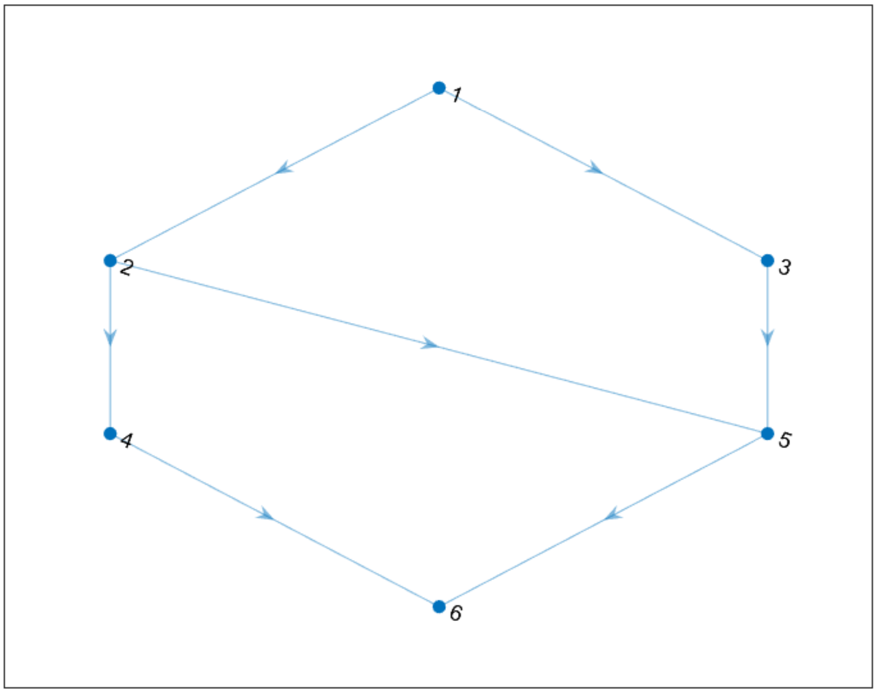

Figure 1: The map.

denote the road segment between two distinct intersections. Without loss of generality, we use ν_1 and ν_N to denote the origin and destination, respectively. An ordered pair (ν_i, ν_j) belonging to the edge set $\mathcal{E}$ means that the electric vehicle can move from intersection ν_i to intersection ν_j. For any $\nu_i \in \mathcal{V}/\{\nu_1, \nu_N\}$, we can always find a sequences of edges starting with ν_1 ending with ν_i and a sequences of edges starting with ν_i ending with ν_N. In other words, our modeling does not include intersections that cannot be reached from the origin or cannot reach the destination. The map used in this homework is generated by the following `Matlab code`:

```
%% Generate a map.
s = [1 1 2 2 3 4 5];
t = [2 3 4 5 5 6 6];
G = digraph(s,t);         % This map is a digraph.
I = full(incidence(G));
plot(G)
```

and shown in Fig. 1. The

We use $\bar{v}_{ij}$ and l_{ij} to represent the maximum velocity limitation and the length of segment (ν_i, ν_j), respectively. To simplify the problem, we assume that the electric vehicle keeps the maximum velocity $\bar{v}_{ij}$ when it moves on the segment (ν_i, ν_j) and the road slop of the segment (ν_i, ν_j) is α_{ij}. Thus the energy required to complete (ν_i, ν_j) can be calculated as

$$w_{ij} = EI_{ij}t_{ij}, \tag{7}$$

where $t_{ij} = \frac{l_{ij}}{\bar{v}_{ij}}$ and $I_{ij} = \frac{c_3\gamma}{c_1}\bar{v}_{ij}^2 + \frac{c_2\gamma}{c_1} + g\gamma\sin(\alpha_{ij})$. In the dynamic programming scheme, w_{ij} is regarded as the reward of (ν_i, ν_j), which can be calculated using the following `Matlab` `function`:

```
function reward = ppReward(vmax,l,alpha,m,Voc,c1,gamma,Crr, Af, Cd, g, rho)

frr = Crr * m * g * cos(alpha);        % rolling resistance
fad = 0.5 * rho * Af * Cd * vmax^2;    % air resistance
fgx = m * g * sin(alpha);              % grade resistance
ft = frr + fad + fgx;                  % traction force

Ia = ft / c1;         % armature current
Ib = gamma * Ia;      % battery current

tf = l / vmax;        % This is length "l", not number 1.
reward = Voc * Ib * tf;
end
```

The length, maximum velocity, and slope of each segment (ν_i, ν_j) is generated randomly using the following `Matlag code`:

```
numInter = size(I,1);
numEdge = size(I,2);
[v0,¬,¬] = converterMPH(70);                  % average velocity limitation: 70mph
rng(1)
vmax = v0 * (1 + rand(numEdge,1));      % generate maximum velocity for each
                                        % road segment.
len0 = 10*1000;                       % average road length: 1km
len = len0 * (1 + rand(numEdge,1));     % generate road lenth for each
                                        % road segment.
alpha = zeros(1,numEdge);

roads = [];
for i=1:numEdge
    roads = [roads road(len(i),vmax(i),alpha(i))];
end
```

where `road` is a class defined as

```
classdef road

    properties
        length        % the length of the road
        velocity      % maximum allowable velocity on this road
        slope         % road slope
    end

    methods
        function obj = road(len,vmax,alpha)
            if nargin < 3, alpha = ""; end
            if nargin < 2, vmax = ""; end
            if nargin < 1, len = ""; end
            obj.length = double(len);
            obj.velocity = double(vmax);
            obj.slope = double(alpha);
        end

        function obj= set.length(obj,len)
            obj.length = double(len);
        end
        function obj= set.velocity(obj,vmax)
```

```
            obj.velocity = double(vmax);
        end
        function obj= set.slope(obj,alpha)
            obj.slope = double(alpha);
        end
    end
end
```

`converterMPH` is a converter:

```
function [mps kph fps]= converterMPH(MPH)
% Convert mph to m/s.

kph = 1.609344 * MPH;
mps = 0.44704 * MPH;
fps = 1.46666667 * MPH;
end
```

Our aim is to find an optimal route starting with ν_1 end with ν_N to minimize the total reward. To achieve this goal, we write the following `Matlab` `function`

```
function [path, cost] = ppDP(rewards,I,state,finalstate)
if state == finalstate
    path = [finalstate];
    cost = 0;
else
    fromk = find(I(state,:)==-1);     % Find paths starting with intersection state
    num_fromk = size(fromk,2);        % number of paths starting with intersection k
    temp_reward = rewards(fromk);
    temp_cost = zeros(1, num_fromk);
    temp_nstate = zeros(1, num_fromk);
    for i = 1:num_fromk
        temp_nstate(i) = find(I(:,fromk(i))==1);
        [¬, cc] = ppDP(rewards,I,temp_nstate(i),finalstate);
        temp_cost(i) = cc + temp_reward(i);
    end
    [MM, II] = min(temp_cost);
    cost = MM;
    [pp, ¬] = ppDP(rewards,I,temp_nstate(II),finalstate);
    path = [state pp];
end
end
```

Meanwhile, we also write a traditional recursive algorithm to verify the correctness of our dynamic programming function:

```
function [path, cost] = ppR(rewards,I,state)
if state == 1
    path = [1];
    cost = 0;
else
    tok = find(I(state,:)==1);     % Find paths ending with intersection k
    num_tok = size(tok,2);     % number of paths ending with intersection k
    temp_reward = rewards(tok);
    temp_cost = zeros(1, num_tok);
    temp_pstate = zeros(1, num_tok);
    for i = 1:num_tok
        temp_pstate(i) = find(I(:,tok(i))==-1);
        [¬, cc] = ppR(rewards,I,temp_pstate(i));
```

```
        temp_cost(i) = cc + temp_reward(i);
    end
    [MM, II] = min(temp_cost);
    cost = MM;
    [pp, ¬] = ppR(rewards,I,temp_pstate(II));
    path = [pp state];
end
end
```

Now we can get the optimal path using the following parameter setting and code:

```
% Electric Vehicle
m = 2300;          % Total mass of the vehicle: 2300kg
Voc = 340;         % Open circuit voltage: 340v
Crr = 0.01;        % Rolling resistance coefficient
g = 9.8;           % Gravitational acceleration: 9.8m/s
c2 = Crr * m * g * cos(0);      % Rolling resistance
rho = 1.23;                     % Air density
Af = 2.1;                       % Vehicle frontal area: 2.1m^2
Cd = 0.38;                      % Aerodynamic drag coefficient
c3 = 0.5 * rho * Af * Cd;
gamma = 1.2;                    % The conversion rate: Ib = gamma*Ia
c1 = 30;                        % The ratio between Fmot and Ia: Fmot = c1 * Ia

rewards = [];
for ii=1:numEdge
    rewards = [rewards ppReward(roads(i).velocity,roads(i).length,...
        roads(i).slope,m,Voc,c1,gamma,Crr, Af, Cd, g, rho)];
end

[pathR, costR] = ppR(rewards,I,numInter);

[path, cost] = ppDP(rewards,I,1, numInter);
```

The results of $ppR(rewards, I, numInter)$ and $ppDP(rewards, I, 1, numInter)$ are same, which are

```
pathR =

     1     2     4     6

path =

     1     2     4     6
```

0.3 Car-following Model

Now we consider a platoon with 4 electric vehicles, which can be defined using a new class `EV`:

```
classdef EV < handle
    %% MEMBERS
    properties
        role                % Leder: role=1; Follower: role = 2
        Af
        g
        rho
        Crr
        Cd
        gamma
```

```
        t
        dt
        tf
        alpha

        vDes
        aMax
        bMax
        bnHat
        Ln

        m       % Mass
        l       % Length

        c1;
        c2;
        c3;

        x       % [X, dX]
        r       % position vector [X]
        v      % velocity vector [dX]

        dx

        u        % Control Input [Ib]
%         Ib        % Battery current
    end

    properties
        r_des
        r_err
        r_err_prev
        r_err_sum

        v_des
        v_err
        v_err_prev
        v_err_sum

        kP_r
        kI_r
        kD_r

        kP_v
        kI_v
        kD_v
    end

    %% METHODS
    methods
        %% CONSTRUCTOR
        function obj = EV(role,params, initStates, initInputs, gains, ...
                simTime, dt, alpha, Ln)
            obj.role = role;
            obj.Af = 2.1;
            obj.g = 9.8;
            obj.rho = 1.23;
            obj.Crr = 0.01;
            obj.Cd = 0.38;
            obj.gamma = 1.2;
            obj.t = 0.0;
            obj.dt = dt;
            obj.tf = simTime;
            obj.alpha = alpha;

```

```
            obj.vDes = 50;
            obj.aMax = 2;
            obj.bMax = -3;
            obj.bnHat = -3.5;
            obj.Ln = Ln;

            obj.m = params('Mass');
            obj.l = params('Length');

            obj.c1 = 30;
            obj.c2 = obj.Crr * obj.m * obj.g * cos(obj.alpha);
            obj.c3 = 0.5 * obj.rho * obj.Af * obj.Cd;

            obj.x = initStates;
            obj.r = obj.x(1);
            obj.v = obj.x(2);

            obj.dx = zeros(2,1);

            obj.u = initInputs;

            obj.r_des = 0.0;
            obj.r_err = 0.0;
            obj.r_err_prev = 0.0;
            obj.r_err_sum = 0.0;

            obj.v_des = 0.0;
            obj.v_err = 0.0;
            obj.v_err_prev = 0.0;
            obj.v_err_sum = 0.0;

            obj.kP_r = gains('P_r');
            obj.kI_r = gains('I_r');
            obj.kD_r = gains('D_r');

            obj.kP_v = gains('P_v');
            obj.kI_v = gains('I_v');
            obj.kD_v = gains('D_v');
        end

        function state = GetStates(obj)
            state = obj.x;
        end

        function obj = EvalEOM(obj)    % Equations Of Motions
            obj.dx(1) = obj.v;
            obj.dx(2) = 1 / obj.m * (-obj.c3 * obj.x(2)^2 ...
                + obj.c1/obj.gamma*obj.u - obj.c2) - obj.g*sin(obj.alpha);
        end

        function obj = UpdateState(obj)
            obj.t = obj.t + obj.dt;

            obj.EvalEOM();
            obj.x = obj.x + obj.dx.*obj.dt;

            obj.r = obj.x(1);
            obj.v = obj.x(2);
        end

        function obj = Ctrl(obj,refSig)
            if obj.role == 1
                i = find(refSig(1,:)>obj.x(1),1);
                if isempty(i)
                    obj.v_des = 0;
```

```
                else
                    obj.v_des = refSig(2,i);
                end
            else
                obj.v_des = gipps(obj.dt,obj.x(2),obj.aMax,obj.vDes, ...
                    obj.bMax,refSig(1),obj.x(1),obj.Ln,refSig(2),obj.bnHat);
%                 obj.kP_v = 100;
            end
                obj.v_err = obj.v_des - obj.x(2);
                obj.u = (obj.kP_v * obj.v_err + ...
                obj.kI_v * obj.v_err_sum + ...
                obj.kD_v * (obj.v_err - obj.v_err_prev)/obj.dt);
%                 obj.u = (((obj.kP_v * obj.v_err + ...
%                 obj.kI_v * obj.v_err_sum + ...
%                 obj.kD_v * (obj.v_err - obj.v_err_prev)/obj.dt) ...
%                 +obj.g*sin(obj.alpha))*obj.m+obj.c2*obj.x(2)^2+obj.c2) ...
%                 *obj.gamma/obj.c1;

                obj.v_err_sum = obj.v_err_sum + obj.v_err;
                obj.v_err_prev = obj.v_err;
        end
    end

end
```

If EV i is the leader vehicle, its desired velocity is the maximum velocity limitation of each segments. Otherwise, its desired velocity can be calculated through Gipps' model:

```
function [v,v1,v2] = gipps(dt,v_,aMax,vDes,bMax,xn_,x_,Ln,vn_,bnHat)
% The Gipps Model
% This function is based on Eq. (2.12) in elefteriadou14
% v is the speed of vehicle n+1 at time t+Delta_t
% dt is the apparent reation time, a constant for  all vehicles
% v_ is the speed of vehicle n+1 at time t
% aMax is the maximum acceleration which the driver of vehicle n+1 whishes
%      whishes to undertake
% vDes is the speed at which the driver of vehicle n+1 whishes to travel
% bMax is the actual most severe deceleration rate that the driver of
%      vehicle n+1 wishes to undertake (b(n+1)<0)
% xn_ is the location of the front of vehicle n at time t
% x_ is the location of the front of vehicle n+1 at time t
% Ln  is the effective size of vehicle n; that is the physical length plus
%     a margin into which the following vehicle is not willing to introduce
%     even at rest
% vn_ is the speed of vehicle n at time t
% bnHat is the most severe deceleration rate that vehicle n+1 estimates for
%       vehicle n

v1 = v_ + 2.5*aMax*dt*(1 - v_/vDes)*sqrt(0.025 + v_/vDes);
v2 = bMax*dt + sqrt(bMax^2*dt^2 - bMax*(2*(xn_ - Ln - x_)-v_*dt-vn_^2/bnHat));
v = min([v1, v2]);
end
```

The controller we used is a simple proportional controller. We set $P = 3$ for the leader vehicle and $P = 100$ for the follower vehicle:

```
%% INIT. PARAMS
ev1_params = containers.Map({'Mass', 'Length'}, {2300, 5});

ev1_initStates = [0,...                % position
    0]';                               % velocity
```

```

Ln = 7.62;
%     the effective size of vehicle n; that is the physical length plus
%     a margin into which the following vehicle is not willing to introduce
%     even at rest

ev2_initStates = [-Ln,...                 % position
    0]';                                  % velocity
ev3_initStates = [-2*Ln,...                 % position
    0]';                                  % velocity
ev4_initStates = [-3*Ln,...                 % position
    0]';

ev1_initInputs = [0];

ev1_gains = containers.Map({'P_r','I_r','D_r', ...
    'P_v','I_v','D_v'}, ...
    {0.0, 0.0, 0.0, ...
    3.0, 0.0, 0.0});
ev2_gains = containers.Map({'P_r','I_r','D_r', ...
    'P_v','I_v','D_v'}, ...
    {0.0, 0.0, 0.0, ...
    100.0, 0.0, 0.0});

simulationTime = 1800;
dt = 1;
alpha = 0;
role1 = 1;        % Leader
role2 = 2;        % Follower

ev1 = EV(role1,ev1_params, ev1_initStates, ev1_initInputs, ev1_gains, ...
    simulationTime,dt,alpha,Ln);
ev2 = EV(role2,ev1_params, ev2_initStates, ev1_initInputs, ev2_gains, ...
    simulationTime,dt,alpha,Ln);
ev3 = EV(role2,ev1_params, ev3_initStates, ev1_initInputs, ev2_gains, ...
    simulationTime,dt,alpha,Ln);
ev4 = EV(role2,ev1_params, ev4_initStates, ev1_initInputs, ev2_gains, ...
    simulationTime,dt,alpha,Ln);
```

Results are shown in Fig. 2 and Fig. 3. Fig. 2 shows the position and velocity of the leader vehicle. From this figure, we can see that the leader vehicle passes 3 intersections before it reaches the destination. On each segment between two distinct intersections, its velocity tracks the maximum allowable velocity of the road. Fig. 3 shows the spacing and velocity difference between the leader vehicle and each follower vehicle. In this figure, the vehicles achieve the platoon behavior: they try to keep a safe inter-vehicles distance and the same velocity. From the sub-figure titled spacing, we can also find that the vehicles will keep a longer spacing to keep traffic safe if their velocity increase.

The code associated with this homework can be download using the following link:

https://github.com/ShaopanGuo/Dynamic-Programming-Path-Planning.git

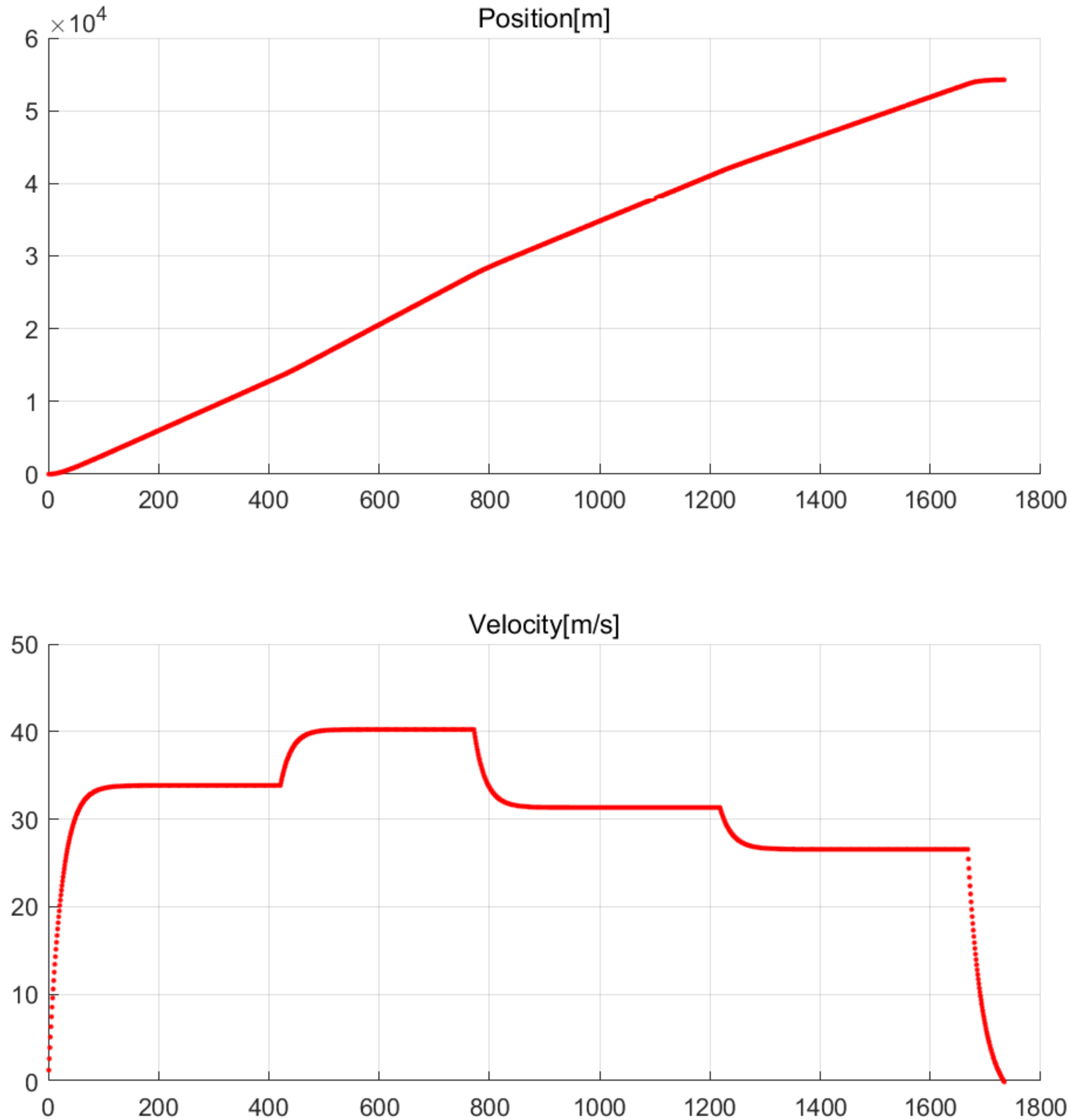

Figure 2: The position and velocity of the leader vehicle.

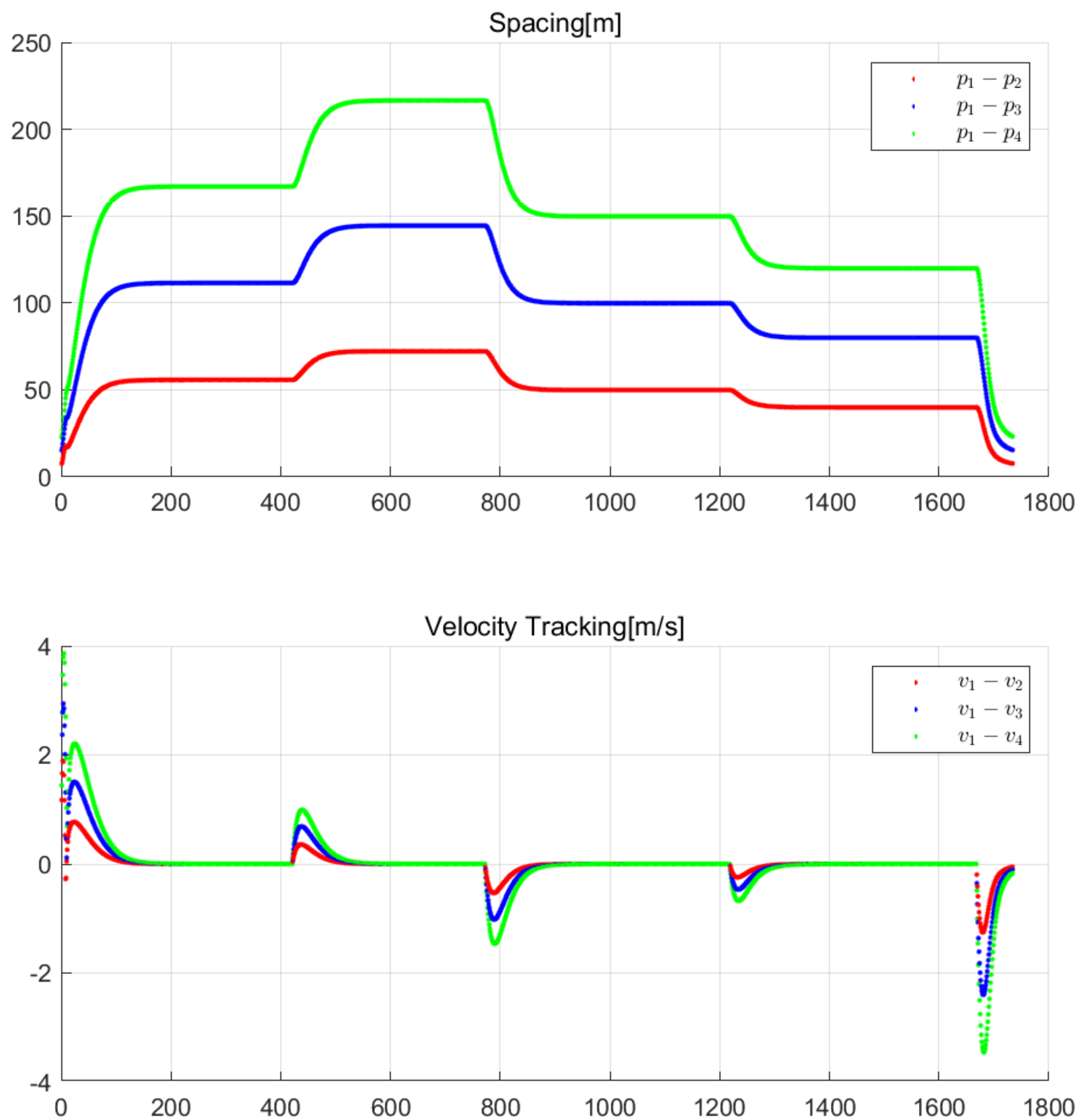

Figure 3: The platoon behavior.