

## EE 7560 Homework 8

Shaopan Guo (gshaop1@lsu.edu)

April 10, 2021

### 0.1 Problem Formulation

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

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

where  $M$  is the total mass of the vehicle,  $\alpha$  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 proportional 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:

$$F_{mot_i} = c_1 I_{a_i}. \quad (2)$$

The battery current  $I_{b_i}$  is defined as

$$I_{b_i} = \gamma I_{a_i}, \quad (3)$$

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

$$P_{b_i} = I_{b_i} E, \quad (4)$$

where voltage source  $E$  is assumed to be constant. Define

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

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

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

It is worthy to mention that the real input is the motor torque, while the design is to control 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  $\mathcal{V} = \{\nu_1, \dots, \nu_N\}$  represent intersections that the electric vehicle may pass through in the journey, the elements in the edge set

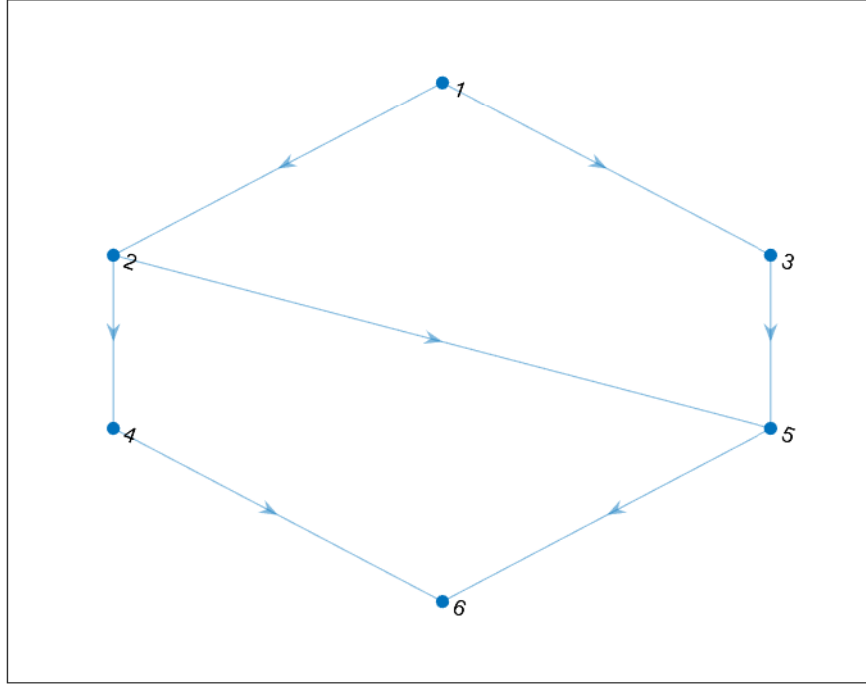


Figure 1: The map.

denote the road segment between two distinct intersections. Without loss of generality, we use  $\nu_1$  and  $\nu_N$  to denote the origin and destination, respectively. An ordered pair  $(\nu_i, \nu_j)$  belonging to the edge set  $\mathcal{E}$  means that the electric vehicle can move from intersection  $\nu_i$  to intersection  $\nu_j$ . For any  $\nu_i \in \mathcal{V}/\{\nu_1, \nu_N\}$ , we can always find a sequences of edges starting with  $\nu_1$  ending with  $\nu_i$  and a sequences of edges starting with  $\nu_i$  ending with  $\nu_N$ . 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:

```

1 %% Generate a map.
2 s = [1 1 2 2 3 4 5];
3 t = [2 3 4 5 5 6 6];
4 G = digraph(s,t);           % This map is a digraph.
5 I = full(incidence(G));
6 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  $(\nu_i, \nu_j)$ , respectively. To simplify the problem, we assume that the electric vehicle keeps the maximum velocity  $\bar{v}_{ij}$  when it moves on the segment  $(\nu_i, \nu_j)$  and the road slop of the segment  $(\nu_i, \nu_j)$  is  $\alpha_{ij}$ . Thus the energy required to complete  $(\nu_i, \nu_j)$  can be calculated as

$$w_{ij} = EI_{ij}t_{ij}, \quad (7)$$

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  $(\nu_i, \nu_j)$ , which can be calculated using the following Matlab [function](#):

```

1 function reward = ppReward(vmax,l,alpha,m,Voc,c1,gamma,Crr, Af, Cd, g, rho)
2
3 frr = Crr * m * g * cos(alpha);           % rolling resistance
4 fad = 0.5 * rho * Af * Cd * vmax^2;       % air resistance
5 fgx = m * g * sin(alpha);                 % grade resistance
6 ft = frr + fad + fgx;                     % traction force
7
8 Ia = ft / c1;                             % armature current
9 Ib = gamma * Ia;                          % battery current
10
11 tf = 1 / vmax;                           % This is length "l", not number 1.
12 reward = Voc * Ib * tf;
13 end

```

The length, maximum velocity, and slope of each segment  $(\nu_i, \nu_j)$  is generated randomly using the following Matlab code:

```

1 numInter = size(I,1);
2 numEdge = size(I,2);
3 [v0,~,~] = converterMPH(70);              % average velocity limitation: 70mph
4 rng(1)
5 vmax = v0 * (1 + rand(numEdge,1));        % generate maximum velocity for each
6                                           % road segment.
7 len0 = 10*1000;                           % average road length: 1km
8 len = len0 * (1 + rand(numEdge,1));       % generate road lenh for each
9                                           % road segment.
10 alpha = zeros(1,numEdge);
11
12 roads = [];
13 for i=1:numEdge
14     roads = [roads road(len(i),vmax(i),alpha(i))];
15 end

```

where road is a class defined as

```

1 classdef road
2
3     properties
4         length           % the length of the road
5         velocity         % maximum allowable velocity on this road
6         slope            % road slope
7     end
8
9     methods
10        function obj = road(len,vmax,alpha)
11            if nargin < 3, alpha = ""; end
12            if nargin < 2, vmax = ""; end
13            if nargin < 1, len = ""; end
14            obj.length = double(len);
15            obj.velocity = double(vmax);
16            obj.slope = double(alpha);
17        end
18
19        function obj = set.length(obj,len)
20            obj.length = double(len);
21        end
22        function obj = set.velocity(obj,vmax)

```

```

23         obj.velocity = double(vmax);
24     end
25     function obj= set.slope(obj,alpha)
26         obj.slope = double(alpha);
27     end
28 end
29 end

```

converterMPH is a converter:

```

1 function [mps kph fps]= converterMPH(MPH)
2 % Convert mph to m/s.
3
4 kph = 1.609344 * MPH;
5 mps = 0.44704 * MPH;
6 fps = 1.46666667 * MPH;
7 end

```

Our aim is to find an optimal route starting with  $\nu_1$  end with  $\nu_N$  to minimize the total reward. To achieve this goal, we write the following Matlab `function`

```

1 function [path, cost] = ppDP(rewards,I,state,finalstate)
2 if state == finalstate
3     path = [finalstate];
4     cost = 0;
5 else
6     fromk = find(I(state,:) == -1); % Find paths starting with intersection state
7     num_fromk = size(fromk,2); % number of paths starting with intersection k
8     temp_reward = rewards(fromk);
9     temp_cost = zeros(1, num_fromk);
10    temp_nstate = zeros(1, num_fromk);
11    for i = 1:num_fromk
12        temp_nstate(i) = find(I(:,fromk(i)) == 1);
13        [r, cc] = ppDP(rewards,I,temp_nstate(i),finalstate);
14        temp_cost(i) = cc + temp_reward(i);
15    end
16    [MM, II] = min(temp_cost);
17    cost = MM;
18    [pp, r] = ppDP(rewards,I,temp_nstate(II),finalstate);
19    path = [state pp];
20 end
21 end

```

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

```

1 function [path, cost] = ppR(rewards,I,state)
2 if state == 1
3     path = [1];
4     cost = 0;
5 else
6     tok = find(I(state,:) == 1); % Find paths ending with intersection k
7     num_tok = size(tok,2); % number of paths ending with intersection k
8     temp_reward = rewards(tok);
9     temp_cost = zeros(1, num_tok);
10    temp_pstate = zeros(1, num_tok);
11    for i = 1:num_tok
12        temp_pstate(i) = find(I(:,tok(i)) == -1);
13        [r, cc] = ppR(rewards,I,temp_pstate(i));

```

```

14     temp_cost(i) = cc + temp_reward(i);
15 end
16 [MM, II] = min(temp_cost);
17 cost = MM;
18 [pp, ~] = ppR(rewards,I,temp_pstate(II));
19 path = [pp state];
20 end
21 end

```

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

```

1 % Electric Vehicle
2 m = 2300; % Total mass of the vehicle: 2300kg
3 Voc = 340; % Open circuit voltage: 340v
4 Crr = 0.01; % Rolling resistance coefficient
5 g = 9.8; % Gravitational acceleration: 9.8m/s
6 c2 = Crr * m * g * cos(0); % Rolling resistance
7 rho = 1.23; % Air density
8 Af = 2.1; % Vehicle frontal area: 2.1m^2
9 Cd = 0.38; % Aerodynamic drag coefficient
10 c3 = 0.5 * rho * Af * Cd;
11 gamma = 1.2; % The conversion rate: Ib = gamma*Ia
12 c1 = 30; % The ratio between Fmot and Ia: Fmot = c1 * Ia
13
14 rewards = [];
15 for ii=1:numEdge
16     rewards = [rewards ppReward(roads(i).velocity,roads(i).length,...
17         roads(i).slope,m,Voc,c1,gamma,Crr, Af, Cd, g, rho)];
18 end
19
20 [pathR, costR] = ppR(rewards,I,numInter);
21
22 [path, cost] = ppDP(rewards,I,1, numInter);

```

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

```

1 pathR =
2
3     1     2     4     6
4
5 path =
6
7     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:

```

1 classdef EV < handle
2     %% MEMBERS
3     properties
4         role % Leder: role=1; Follower: role = 2
5         Af
6         g
7         rho
8         Crr
9         Cd
10        gamma

```

```

11     t
12     dt
13     tf
14     alpha
15
16     vDes
17     aMax
18     bMax
19     bnHat
20     Ln
21
22     m      % Mass
23     l      % Length
24
25     c1;
26     c2;
27     c3;
28
29     x      % [X, dX]
30     r      % position vector [X]
31     v      % velocity vector [dX]
32
33     dx
34
35     u      % Control Input [Ib]
36 %      Ib      % Battery current
37 end
38
39 properties
40     r_des
41     r_err
42     r_err_prev
43     r_err_sum
44
45     v_des
46     v_err
47     v_err_prev
48     v_err_sum
49
50     kP_r
51     kI_r
52     kD_r
53
54     kP_v
55     kI_v
56     kD_v
57 end
58
59 %% METHODS
60 methods
61     %% CONSTRUCTOR
62     function obj = EV(role,params, initStates, initInputs, gains, ...
63         simTime, dt, alpha, Ln)
64         obj.role = role;
65         obj.Af = 2.1;
66         obj.g = 9.8;
67         obj.rho = 1.23;
68         obj.Crr = 0.01;
69         obj.Cd = 0.38;
70         obj.gamma = 1.2;
71         obj.t = 0.0;
72         obj.dt = dt;
73         obj.tf = simTime;
74         obj.alpha = alpha;
75

```

```

76         obj.vDes = 50;
77         obj.aMax = 2;
78         obj.bMax = -3;
79         obj.bnHat = -3.5;
80         obj.Ln = Ln;
81
82         obj.m = params('Mass');
83         obj.l = params('Length');
84
85         obj.c1 = 30;
86         obj.c2 = obj.Crr * obj.m * obj.g * cos(obj.alpha);
87         obj.c3 = 0.5 * obj.rho * obj.Af * obj.Cd;
88
89         obj.x = initState;
90         obj.r = obj.x(1);
91         obj.v = obj.x(2);
92
93         obj.dx = zeros(2,1);
94
95         obj.u = initInputs;
96
97         obj.r_des = 0.0;
98         obj.r_err = 0.0;
99         obj.r_err_prev = 0.0;
100        obj.r_err_sum = 0.0;
101
102        obj.v_des = 0.0;
103        obj.v_err = 0.0;
104        obj.v_err_prev = 0.0;
105        obj.v_err_sum = 0.0;
106
107        obj.kP_r = gains('P_r');
108        obj.kI_r = gains('I_r');
109        obj.kD_r = gains('D_r');
110
111        obj.kP_v = gains('P_v');
112        obj.kI_v = gains('I_v');
113        obj.kD_v = gains('D_v');
114    end
115
116    function state = GetStates(obj)
117        state = obj.x;
118    end
119
120    function obj = EvalEOM(obj) % Equations Of Motions
121        obj.dx(1) = obj.v;
122        obj.dx(2) = 1 / obj.m * (-obj.c3 * obj.x(2)^2 ...
123            + obj.c1/obj.gamma*obj.u - obj.c2) - obj.g*sin(obj.alpha);
124    end
125
126    function obj = UpdateState(obj)
127        obj.t = obj.t + obj.dt;
128
129        obj.EvalEOM();
130        obj.x = obj.x + obj.dx.*obj.dt;
131
132        obj.r = obj.x(1);
133        obj.v = obj.x(2);
134    end
135
136    function obj = Ctrl(obj,refSig)
137        if obj.role == 1
138            i = find(refSig(1,:)>obj.x(1),1);
139            if isempty(i)
140                obj.v_des = 0;

```

```

141         else
142             obj.v_des = refSig(2,i);
143         end
144     else
145         obj.v_des = gipps(obj.dt,obj.x(2),obj.aMax,obj.vDes, ...
146             obj.bMax,refSig(1),obj.x(1),obj.Ln,refSig(2),obj.bnHat);
147         % obj.kP_v = 100;
148     end
149     obj.v_err = obj.v_des - obj.x(2);
150     obj.u = (obj.kP_v * obj.v_err + ...
151         obj.kI_v * obj.v_err_sum + ...
152         obj.kD_v * (obj.v_err - obj.v_err_prev)/obj.dt);
153     % obj.u = ((obj.kP_v * obj.v_err + ...
154     % obj.kI_v * obj.v_err_sum + ...
155     % obj.kD_v * (obj.v_err - obj.v_err_prev)/obj.dt) ...
156     % +obj.g*sin(obj.alpha))*obj.m+obj.c2*obj.x(2)^2+obj.c2) ...
157     % *obj.gamma/obj.c1;
158
159     obj.v_err_sum = obj.v_err_sum + obj.v_err;
160     obj.v_err_prev = obj.v_err;
161 end
162 end
163
164 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:

```

1 function [v,v1,v2] = gipps(dt,v_,aMax,vDes,bMax,xn_,x_,Ln,vn_,bnHat)
2 % The Gipps Model
3 % This function is based on Eq. (2.12) in elefteriadou14
4 % v is the speed of vehicle n+1 at time t+Delta_t
5 % dt is the apparent reation time, a constant for all vehicles
6 % v_ is the speed of vehicle n+1 at time t
7 % aMax is the maximum acceleration which the driver of vehicle n+1 wishes
8 % wishes to undertake
9 % vDes is the speed at which the driver of vehicle n+1 wishes to travel
10 % bMax is the actual most severe deceleration rate that the driver of
11 % vehicle n+1 wishes to undertake (b(n+1)<0)
12 % xn_ is the location of the front of vehicle n at time t
13 % x_ is the location of the front of vehicle n+1 at time t
14 % Ln is the effective size of vehicle n; that is the physical length plus
15 % a margin into which the following vehicle is not willing to introduce
16 % even at rest
17 % vn_ is the speed of vehicle n at time t
18 % bnHat is the most severe deceleration rate that vehicle n+1 estimates for
19 % vehicle n
20
21 v1 = v_ + 2.5*aMax*dt*(1 - v_/vDes)*sqrt(0.025 + v_/vDes);
22 v2 = bMax*dt + sqrt(bMax^2*dt^2 - bMax*(2*(xn_ - Ln - x_)-v_*dt-vn_^2/bnHat));
23 v = min([v1, v2]);
24 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:

```

1 %% INIT. PARAMS
2 evl_params = containers.Map({'Mass', 'Length'}, {2300, 5});
3
4 evl_initStates = [0,... % position
5     0]'; % velocity

```



```

6
7 Ln = 7.62;
8 % the effective size of vehicle n; that is the physical length plus
9 % a margin into which the following vehicle is not willing to introduce
10 % even at rest
11
12 ev2_initStates = [-Ln,...           % position
13                  0]';              % velocity
14 ev3_initStates = [-2*Ln,...        % position
15                  0]';              % velocity
16 ev4_initStates = [-3*Ln,...        % position
17                  0]';
18
19 ev1_initInputs = [0];
20
21 ev1_gains = containers.Map({'P_r','I_r','D_r', ...
22                             'P_v','I_v','D_v'}, ...
23                             {0.0, 0.0, 0.0, ...
24                              3.0, 0.0, 0.0});
25 ev2_gains = containers.Map({'P_r','I_r','D_r', ...
26                             'P_v','I_v','D_v'}, ...
27                             {0.0, 0.0, 0.0, ...
28                              100.0, 0.0, 0.0});
29
30 simulationTime = 1800;
31 dt = 1;
32 alpha = 0;
33 role1 = 1;           % Leader
34 role2 = 2;           % Follower
35
36 ev1 = EV(role1, ev1_params, ev1_initStates, ev1_initInputs, ev1_gains, ...
37           simulationTime, dt, alpha, Ln);
38 ev2 = EV(role2, ev1_params, ev2_initStates, ev1_initInputs, ev2_gains, ...
39           simulationTime, dt, alpha, Ln);
40 ev3 = EV(role2, ev1_params, ev3_initStates, ev1_initInputs, ev2_gains, ...
41           simulationTime, dt, alpha, Ln);
42 ev4 = EV(role2, ev1_params, ev4_initStates, ev1_initInputs, ev2_gains, ...
43           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>

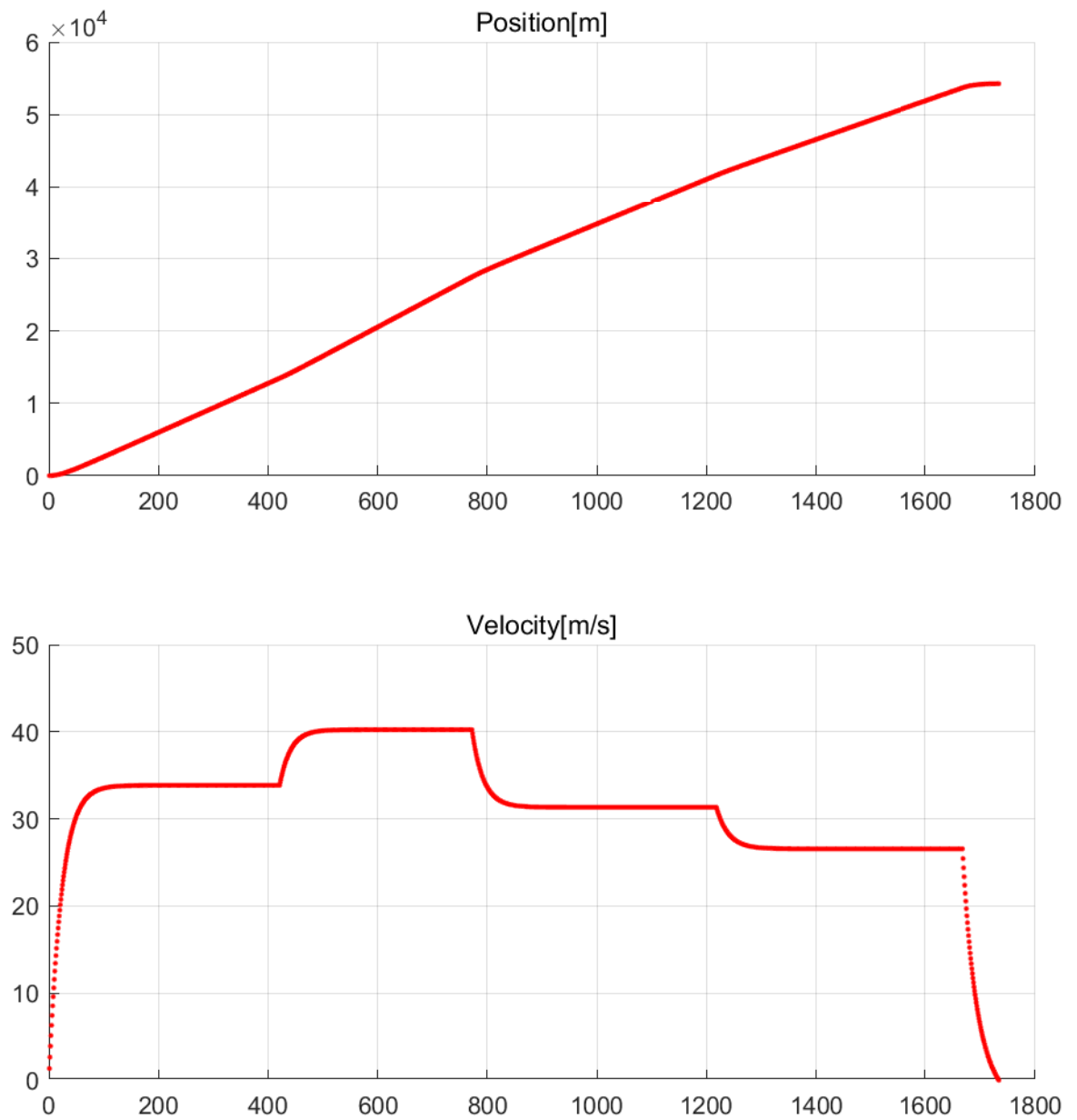


Figure 2: The position and velocity of the leader vehicle.

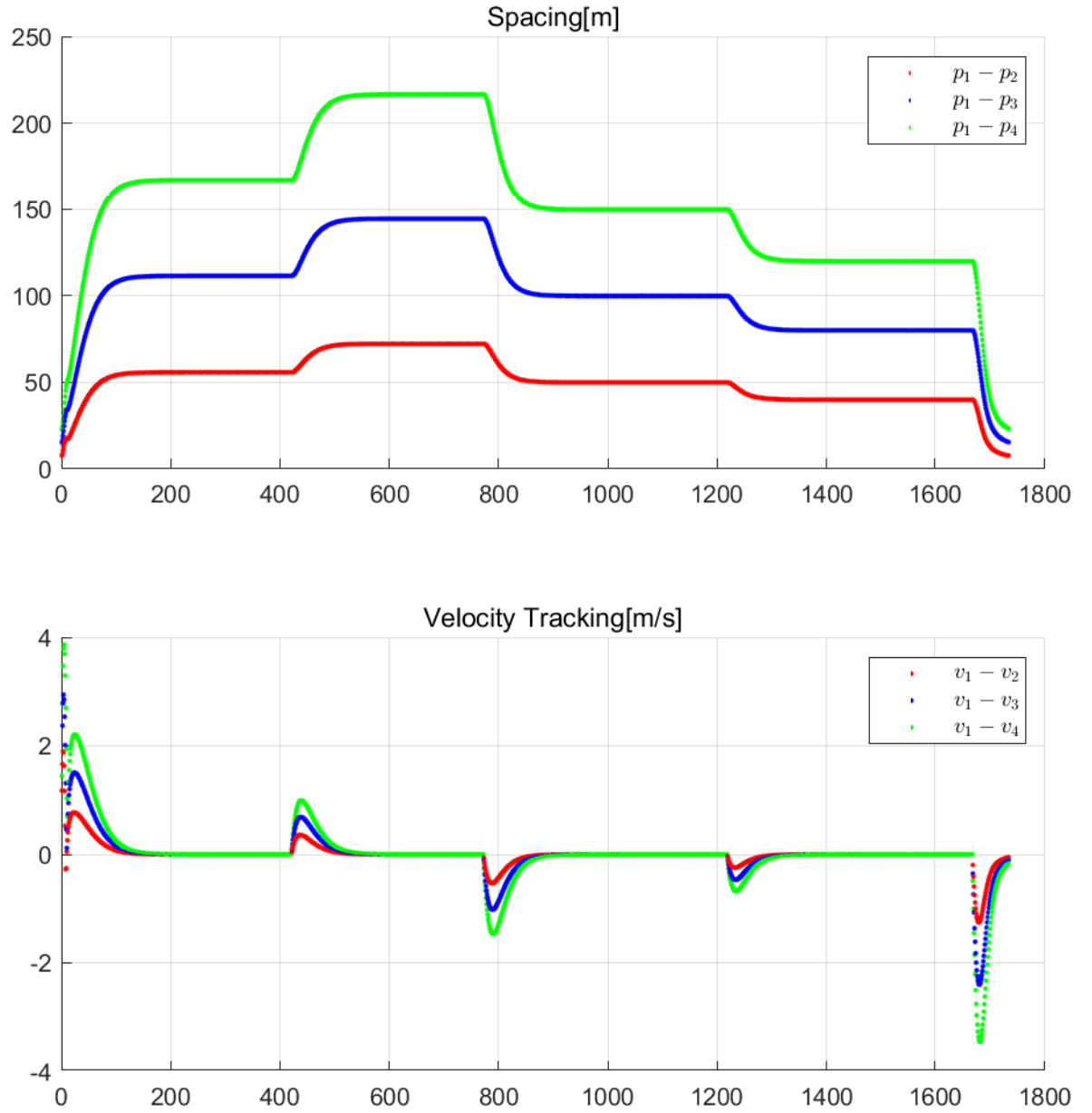


Figure 3: The platoon behavior.