F priedas. Globaliųjų bei lokaliųjų judėjimo maršrutų planavimo kodai programinio paketo *MATLAB* aplinkoje

F.1. Globaliojo judėjimo maršruto planavimo kodo fragmentas
F.1. Code fragment of the global path planning

Pirminiai autonominės kelių transporto priemonės parametrai:

```
global l_g; global l_f; global v; global SR; global w;  %(R - steering
ratio, l_g - [m], l_f - [m], v - m/s, w (vairo sukimo greitis) - rad/s)
l_g = 1.4;  l_f = 1.3;  v = 3; SR = 17.6; w = 2.65;
% Minimalus ATP apsisukimo spindulys [m]
r = 5.1;
% Maksimalus imanomas vairuojamuju ratu pasukimo kampas [rad]:
d_max = atan((l_f + l_g) / r);
% Neutrali vairuojamuju ratu padetis [rad]:
d_0 = 0;
```

Dubinso maršrutų skaičiavimo metodo taikymas:

```
%Pirmasis perplanavimas: Pirmosios ir antrosios tiesiu krypties vektoriai
q_1_skaitiklis = [x(2) y(2)] - [x(1) y(1)]; q_1_vardiklis = sqrt(
(q_1_skaitiklis(1))^2 + (q_1_skaitiklis(2))^2); q_1 = [
(q_1_skaitiklis(1)/q_1_vardiklis) (q_1_skaitiklis(2)/q_1_vardiklis) ];
q_i1_skaitiklis = [x(3) y(3)] - [x(2) y(2)]; q_i1_vardiklis = sqrt(
(q_i1_skaitiklis(1))^2 + (q_i1_skaitiklis(2))^2); q_i1 = [
(q_i1_skaitiklis(1)/q_i1_vardiklis) (q_i1_skaitiklis(2)/q_i1_vardiklis) ];
% Kampas tarp krypties vektoriu [rad]:
q_1T = -(transpose(q_1)); a_1 = acos(dot(q_1T,q_i1));
% Apskritimo centras trajektorijos planavimui:
qp_1_skaitiklis = q_1 - q_i1; qp_1_vardiklis = sqrt(
(qp_1_skaitiklis(1))^2 + (qp_1_skaitiklis(2))^2); qp_1 = [
(qp_1_skaitiklis(1)/qp_1_vardiklis) (qp_1_skaitiklis(2)/qp_1_vardiklis) ];
C_1 = [x(2) y(2)] - (r/sin((a_1)/2))* qp_1;
% Apskritimas trajektorijos planavimui:
xCenter1 = C_1(1); yCenter1 = C_1(2); theta = 0 : 0.01 : 2*pi; xc1 = r *
cos(theta) + xCenter1; yc1 = r * sin(theta) + yCenter1;
% Atstumo pirmojo posukio planavimui skaiciavimas:
k_11 = (r/(tan((a_1)/2)))*q_1; k_12 = -(r/(tan((a_1)/2)))*q_i1; H_11 =
[x(2) y(2)] - k_11; H_12 = [x(2) y(2)] - k_12;
```

```
%Antrasis perplanavimas: treciosios atkarpos krypties vektoriai
q_i2_skaitiklis = [x(4) y(4)] - [x(3) y(3)]; q_i2_vardiklis = sqrt(
(q_i2_skaitiklis(1))^2 + (q_i2_skaitiklis(2))^2); q_i2 = [
(q_i2_skaitiklis(1)/q_i2_vardiklis) (q_i2_skaitiklis(2)/q_i2_vardiklis) ];
% Kampas tarp krypties vektoriu [rad]:
q_i1T = -(transpose(q_i1)); a_2 = acos(dot(q_i1T,q_i2));
% Apskritimo centras trajektorijos planavimui:
qp_2_skaitiklis = q_i1 - q_i2; qp_2_vardiklis = sqrt(
(qp_2_skaitiklis(1))^2 + (qp_2_skaitiklis(2))^2); qp_2 = [
(qp_2_skaitiklis(1)/qp_2_vardiklis) (qp_2_skaitiklis(2)/qp_2_vardiklis) ];
C_2 = [x(3) y(3)] - (r/sin((a_2)/2))* qp_2;
% Apskritimas trajektorijos planavimui:
xCenter2 = C_2(1); yCenter2 = C_2(2); xc2 = r * cos(theta) + xCenter2; yc2
= r * sin(theta) + yCenter2;
% Atstumo antrojo posukio planavimui skaiciavimas:
k_21 = (r/(tan((a_2)/2)))*q_i1; k_22 = -(r/(tan((a_2)/2)))*q_i2; H_21 =
[x(3) y(3)] - k_21; H_22 = [x(3) y(3)] - k_22;
```

```
%Treciasis perplanavimas: ketirtosios atkarpos krypties vektoriai
q_i3_skaitiklis = [x(5) y(5)] - [x(4) y(4)]; q_i3_vardiklis = sqrt(
(q_i3_skaitiklis(1))^2 + (q_i3_skaitiklis(2))^2); q_i3 = [
(q_i3_skaitiklis(1)/q_i3_vardiklis) (q_i3_skaitiklis(2)/q_i3_vardiklis) ];
% Kampas tarp krypties vektoriu [rad]:
q_i2T = -(transpose(q_i2)); a_3 = acos(dot(q_i2T,q_i3));
% Apskritimo centras trajektorijos planavimui:
qp_3_skaitiklis = q_i2 - q_i3; qp_3_vardiklis = sqrt(
(qp_3_skaitiklis(1))^2 + (qp_3_skaitiklis(2))^2); qp_3 = [
(qp_3_skaitiklis(1)/qp_3_vardiklis) (qp_3_skaitiklis(2)/qp_3_vardiklis) ];
C_3 = [x(4) y(4)] - (r/sin((a_3)/2))* qp_3;
% Apskritimas trajektorijos planavimui:
xCenter3 = C_3(1); yCenter3 = C_3(2); xc3 = r * cos(theta) + xCenter3; yc3
= r * sin(theta) + yCenter3;
% Atstumo treciojo posukio planavimui skaiciavimas:
k_31 = (r/(tan((a_3)/2)))*q_i2; k_32 = -(r/(tan((a_3)/2)))*q_i3; H_31 =
[x(4) y(4)] - k_31; H_32 = [x(4) y(4)] - k_32;
```

Genetinio algoritmo taikymas:

```
options = optimoptions('ga','PopulationSize',100,'Maxgenerations',100,
'PlotFcn', {@gaplotbestf, @gaplotbestindiv, @gaplotstopping});
nvars = 1;  % Optimizuojamu kintamuju skaicius
LB = 5.1; % Atitinka minimalu ATP apsisukimo spinduli
UB = 154; % Atitinka maksimalu ATP apsisukimo spinduli
```

```
% Pirmojo planavimo optimizavimas:
% Optimizavimo funkcijai reikalingu parametru reiksmes:
global U_i1; global t_1;
S_11 = find(D_i < D_2); % Tasko, kuriame vertinami nuokrypiai parinkimas
U_i1 = u_i(S_11(end)); t_1 = t(S_11(end));

% Parametrai apribojimo skaiciavimui:
global x_1; global y_1; global x_2; global y_2; global q_1_1; global
q_1_2; global q_1_a;
x_1 = x(1); y_1 = y(1); x_2 = x(2); y_2 = y(2); q_1_1 = q_1(1); q_1_2 =
q_1(2);

if q_1_vardiklis >= q_i1_vardiklis
   q_1_a = q_i1_vardiklis;
else
   q_1_a = q_1_vardiklis;
end

% Optimizavimas
Obj_1 = @Optima_1;
ConsFCN_1 = @Apribojimas_1;
r_1 = ga(Obj_1,nvars,[],[],[],[],LB,UB,ConsFCN_1,options); % Ieskomas
realus reikalingas apsisukimo spindulys r_1

% Apskritimo centro perskaiciavimas ivertinant optimizuota posukio
spinduli:
C_1 = [x(2) y(2)] - (r_1/sin((a_1)/2))* qp_1;
xCenter1 = C_1(1); yCenter1 = C_1(2); theta = 0 : 0.01 : 2*pi; xc1 = r_1 *
cos(theta) + xCenter1; yc1 = r_1 * sin(theta) + yCenter1;

% Atstumu posukio perplanavimui perskaiciavimas:
k_11 = (r_1/(tan((a_1)/2)))*q_1; k_12 = -(r_1/(tan((a_1)/2)))*q_i1; H_11 =
[x(2) y(2)] - k_11; H_12 = [x(2) y(2)] - k_12;

% Atstumu, kuriuos nuvaziavus pasikeicia vairuojumu ratu pasukimo kampas
perskaiciavimas [m]:
D_1 = sqrt ((H_11(1) - x(1))^2 + (H_11(2) - x(1))^2)-(((((atan((l_g +
l_f)/r_1)) * v * SR ) / w))/2); D_2 = D_1 + ((pi-a_1) * r_1);

% Salygos ratu atsukimui:
sal_2_1 = (D_2+(D_2 - D_1));
t_r2_1 = abs((((atan((l_f + l_g) / r_1)) + d_0) * SR)) / w; t_r3_1 =
```

```
abs(((d_0 + (atan((l_f + l_g) / r_1)))* SR) / w);

% Antrojo planavimo optimizavimas:
global U_i2; global t_2; global U_i11;
S_21 = find(D_i < D_4); % Tasko, kuriame vertinami nuorkrypiai parinkimas
U_i2 = u_i(S_21(end)); t_2 = t(S_21(end)) - t(S_11(end)+1); U_i11 =
u_i(S_11(end)+1);

% Parametrai apribojimu skaiciavimui:
global x_3; global y_3; global q_i1_1; global q_i1_2; global q_2_a;
x_3 = x(3); y_3 = y(3); q_i1_1 = q_i1(1); q_i1_2 = q_i1(2);

if q_i1_vardiklis >= q_i2_vardiklis
   q_2_a = q_i2_vardiklis;
else
   q_2_a = q_i1_vardiklis;
end

% Optimizavimas
Obj_2 = @Optima_2;
ConsFCN_2 = @Apribojimas_2;
r_2 = ga(Obj_2,nvars,[],[],[],[],LB,UB,ConsFCN_2,options); % Ieskomas
realus reikalingas apsisukimo spindulys r_2

% Apskritimo centro perskaiciavimas ivertinant optimizuota posukio
spinduli:
C_2 = [x(3) y(3)] - (r_2/sin((a_2)/2))* qp_2;
xCenter2 = C_2(1); yCenter2 = C_2(2); xc2 = r_2 * cos(theta) + xCenter2;
yc2 = r_2 * sin(theta) + yCenter2;

% Atstumu posukio perplanavimui perskaiciavimas:
k_21 = (r_2/(tan((a_2)/2)))*q_i1; k_22 = -(r_2/(tan((a_2)/2)))*q_i2; H_21
= [x(3) y(3)] - k_21; H_22 = [x(3) y(3)] - k_22;

% Atstumu, kuriuos nuvaziavus pasikeicia vairuojumu ratu pasukimo kampas
perskaiciavimas [m]:
D_3 = D_2 + (sqrt ((H_21(1) - H_12(1))^2 + (H_21(2) - H_12(2))^2)); D_4 =
D_3 + ((pi-a_2) * r_2);

% Salygos ratu atsukimui:
sal_4_1 = (D_4+(D_4-D_3));
t_r4_1 = abs((((atan((l_f + l_g) / r_2)) + d_0) * SR)) / w; t_r5_1 =
abs(((d_0 + (atan((l_f + l_g) / r_2)))* SR) / w);
```

```
% Treciojo planavimo optimizavimas:
global U_i3; global t_3; global U_i22;
S_31 = find(D_i < D_6); % Tasko, kuriame vertinami nuorkrypiai parinkimas
U_i3 = u_i(S_31(end)); t_3 = t(S_31(end)) - t(S_21(end)+1); U_i22 =
u_i(S_21(end)+1);

% Parametrai apribojimu skaiciavimui:
global x_4; global y_4; global q_i2_1; global q_i2_2; global q_3_a;
x_4 = x(4); y_4 = y(4); q_i2_1 = q_i2(1); q_i2_2 = q_i2(2);
if q_i2_vardiklis >= q_i3_vardiklis
   q_3_a = q_i3_vardiklis;
else
   q_3_a = q_i2_vardiklis;
end
% Optimizavimas
Obj_3 = @Optima_3;
ConsFCN_3 = @Apribojimas_3;
r_3 = ga(Obj_3,nvars,[],[],[],[],LB,UB,ConsFCN_3,options); % Ieskomas
realus reikalingas apsisukimo spindulys r_3

% Apskritimo centro perskaiciavimas ivertinant optimizuota posukio
spinduli:
C_3 = [x(4) y(4)] - (r_3/sin((a_3)/2))* qp_3;
xCenter3 = C_3(1); yCenter3 = C_3(2); xc3 = r_3 * cos(theta) + xCenter3;
yc3 = r_3 * sin(theta) + yCenter3;

% Atstumu posukio perplanavimui perskaiciavimas:
k_31 = (r_3/(tan((a_3)/2)))*q_i2; k_32 = -(r_3/(tan((a_3)/2)))*q_i3; H_31
= [x(4) y(4)] - k_31; H_32 = [x(4) y(4)] - k_32;

% Atstumu, kuriuos nuvaziavus pasikeicia vairuojumu ratu pasukimo kampas
perskaiciavimas [m]:
D_5 = D_4 + (sqrt ((H_31(1) - H_22(1))^2 + (H_31(2) - H_22(2))^2)); D_6 =
D_5 + ((pi-a_3) * r_3);

% Salygos ratu atsukimui:
sal_6_1 = (D_6+(D_6-D_5));
t_r6_1 = abs((((atan((l_f + l_g) / r_3)) + d_0) * SR)) / w; t_r7_1 =
abs(((d_0 + (atan((l_f + l_g) / r_3)))* SR) / w);

D_0_1 = D_6 + (sqrt ((x(5) - H_32(1))^2 + (y(5) - H_32(2))^2));
```

Optimizavimo funkcijos:

```
% Pirmasis optimizavimas
  function f_1 = Optima_1(r_1)
           global U_i1; global t_1; global l_g; global l_f; global v;
           f_1 = (((v/(1+l_g^2*tan(atan((l_f + l_g) /
r_1))^2/(l_g+l_f)^2)^(1/2)/(l_g+l_f)*tan(atan((l_f + l_g) / r_1))*t_1) -
U_i1)^2); end
    function [c,ceq] = Apribojimas_1(r_1)
            global x_2; global y_2; global a_1; global q_1_1; global
q_1_2; global q_1_a;
            c = ((sqrt (((x_2) - ((x_2)-((r_1/(tan((a_1)/2)))*q_1_1)))^2 +
((y_2) - ((y_2)-((r_1/(tan((a_1)/2)))*q_1_2)))^2))) - (((0.5)*q_1_a));
            ceq = [ ]; end
  % Antrasis optimizavimas
  function f_2 = Optima_2(r_2)
           global U_i2; global t_2; global l_g; global l_f; global v;
global U_i11;
           f_2 = ((((v/(1+l_g^2*tan(atan((l_f + l_g) /
r_2))^2/(l_g+l_f)^2)^(1/2)/(l_g+l_f)*tan(atan((l_f + l_g) / r_2))*t_2) +
U_i11) - U_i2)^2); end
  function [c,ceq] = Apribojimas_2(r_2)
            global x_3; global y_3; global a_2; global q_i1_1; global
q_i1_2; global q_2_a;
            c = ((sqrt (((x_3) - ((x_3)-((r_2/(tan((a_2)/2)))*q_i1_1)))^2
+ ((y_3) - ((y_3)-((r_2/(tan((a_2)/2)))*q_i1_2)))^2))) - (((0.5)*q_2_a));
            ceq = [ ]; end
  % Treciasis optimizavimas
  function f_3 = Optima_3(r_3)
           global U_i3; global t_3; global l_g; global l_f; global v;
global U_i22;
           f_3 = ((((v/(1+l_g^2*tan(atan((l_f + l_g) /
r_3))^2/(l_g+l_f)^2)^(1/2)/(l_g+l_f)*tan(atan((l_f + l_g) / r_3))*t_3) +
U_i22) - U_i3)^2); end
  function [c,ceq] = Apribojimas_3(r_3)
            global x_4; global y_4; global a_3; global q_i2_1; global
q_i2_2; global q_3_a;
            c = ((sqrt (((x_4) - ((x_4)-((r_3/(tan((a_3)/2)))*q_i2_1)))^2
+ ((y_4) - ((y_4)-((r_3/(tan((a_3)/2)))*q_i2_2)))^2))) - (((0.5)*q_3_a));
            ceq = [ ]; end
  end
```

F.2. Lokaliojo judėjimo maršruto planavimo kodo fragmentas
F.2. Code fragment of the local path planning

Pirminiai autonominės kelių transporto priemonės parametrai:

```
% Autonomine transporto priemone:
W_a = 1.725; % Autonominio automobilio plotis, m
V_a = 3; % Autonominio automobilio greitis, m/s
L_a = 4.45; % Autonominio automobilio ilgis
l_f = 1.3; l_g = 1.4; % Autonominio automobilio atstumo nuo mases centro
iki priekines ir galines asiu skaiciavimai;

% Kliutis:
W_k = 2; % Kliuties plotis, m
L_k = 2; % Kliuties ilgis, m

% Kiti parametrai:
t_d = 0.9; % Priimamas sistemos veikimo delay, s
SD = 1.5; % Saugus atstumas, m
```

Judėjimo aplinka:

```
r_1 = 20; % Posukio trajektorijos spindulys, m
% Atstumas, kuriam esant pradedamas lenkimo manevras:
S = r_1 + (V_a * t_d);

% Aprasomas globalusis posukio spindulys:
C_x = S; % Pastovaus spindulio posukio xentro X koordinate
C_y = r_1; % Pastovaus spindulio posukio xentro Y koordinate (kaire arba
desine puse, + kaire puse; - desine puse)

% Laisvas atstumas (judejiams isoriniu arba vidiniu ziedu):
vidinis_ziedas = 1;
isorinis_ziedas = 0;
theta_turn = (pi/2); % Posukio trajektorijos kampas
theta_turn_dec = pi/12; % Posukio trajektorijos kampo mazinimas

if C_y > 0 % Posukio spindulio koordinaciu isreiskimo salyga, priklausomai
nuo to yra atliekamas posukis i kaire ar desine puse
theta_1 = theta_turn_dec : 0.01 : theta_turn; % Posukio trajektorijos
kampo kitimas
C_x_1 = fliplr(r_1 * cos(-theta_1) + C_x); C_y_1 = fliplr(r_1 * sin(-
theta_1) + C_y); % Posukio trajektorijos koordinates
else
```

```
theta_1 = theta_turn_dec : 0.01 : theta_turn; % Posukio trajektorijos
kampo kitimas
C_x_1 = fliplr(r_1 * cos(-theta_1) + C_x); C_y_1 = -(fliplr(abs(r_1) *
sin(-theta_1) - C_y)); % Posukio trajektorijos koordinates
end

% Ivedami posukio atkarpas dalinantys taskai:

V_L = length(C_x_1);
V_L_1 = round(V_L*(1/6));
V_L_2 = round(V_L*(2/6));
V_L_3 = round(V_L*(3/6));
V_L_4 = round(V_L*(4/6));
V_L_5 = round(V_L*(5/6));

Padetis_k = V_L_3; % Kliuties padetis
Kreives_p = (3/6); % Vertinimo taskas pagal kliuties padeti

% Kliutis, su kuria negalima susidurti: (reikalingi ivertinti kliuties
stovejimo kampa)
mid_x = C_x_1(Padetis_k); mid_y = C_y_1(Padetis_k); mid_theta_1 =
theta_1(Padetis_k); % Kliuties centro koordinate
obstacle(1,:) = [mid_x - (L_k/2), mid_y - (W_k/2), L_k, W_k]; % [kliuties
pradzia X koordinate, kliuties pradines Y koordinates, kliuties ilgis,
kliuties aukstis]

rectangle('Position',obstacle,'FaceColor',[0 0 0.5 0.5]); % Vaizduojama
kliutis
x_max = mid_x;
y_max = mid_y;
```

Lokaliojo maršruto planavimas

```
% Judejimo pradzios koordinates
q_start.coord = [0 0];

EPS = 1;

numNodes = S/2; % Mazgu skaicius

q_start.cost = 0;
q_start.parent = 0;
```

```
% Galutines Bezier koordinates:

if vidinis_ziedas >= isorinis_ziedas
IS_k = (sqrt((W_k)^2 + (L_k)^2))/2; % Kliuties istrizaines ilgis
r_k = (r_1 - 2*SD - 2*IS_k - (2*W_a) - vidinis_ziedas); % Atstumas nuo
globaliojo posukio centro iki pageidaujamo saugaus judejimo Bezier tasko
else
IS_k = (sqrt((W_k)^2 + (L_k)^2))/2;
r_k = 1.5*(r_1 + 2*SD + 2*IS_k + (2*W_a) + isorinis_ziedas);
end

if vidinis_ziedas >= isorinis_ziedas
if theta_turn_dec == 0
if C_y > 0
x_r = C_x + r_k * cos(mid_theta_1+( (3*pi) /2 )); % Saugaus bezier tasko
salia kliutis X koordinate
y_r = C_y + r_k * sin(mid_theta_1+ ( (3*pi) /2 )); % Saugaus bezier tasko
salia kliutis Y koordinate
else
x_r = C_x + r_k * cos(mid_theta_1+( (3*pi) /2 ));
y_r = C_y - r_k * sin(-mid_theta_1+ ( (3*pi) /2 ));

end
else
if C_y > 0
x_r = C_x + r_k * cos(mid_theta_1+ ( (3*pi) /2 ) - theta_turn_dec); %
Saugaus bezier tasko salia kliutis X koordinate
y_r = C_y + r_k * sin(mid_theta_1+ ( (3*pi) /2 ) - theta_turn_dec); %
Saugaus bezier tasko salia kliutis Y koordinate
else
x_r = C_x + r_k * cos(mid_theta_1+ ( (3*pi) /2 ) - theta_turn_dec);
y_r = C_y - r_k * sin(-mid_theta_1+ ( (3*pi) /2 ) + theta_turn_dec);
end
end

else
if theta_turn_dec == 0
if C_y > 0
x_r = C_x + r_k * cos(mid_theta_1+( (3*pi) /2 )); % Saugaus bezier tasko
salia kliutis X koordinate
y_r = C_y + r_k * sin(mid_theta_1+ ( (3*pi) /2 )); % Saugaus bezier tasko
salia kliutis Y koordinate
else
x_r = C_x + r_k * cos(mid_theta_1+( (3*pi) /2 ));
```

```
y_r = C_y - r_k * sin(-mid_theta_1+ ( (3*pi) /2 ));
end

else
if C_y > 0
x_r = C_x + r_k * cos(mid_theta_1+( (3*pi) /2 ) - theta_turn_dec); %
Saugaus bezier tasko salia kliutis X koordinate
y_r = C_y + r_k * sin(mid_theta_1+ ( (3*pi) /2 ) - theta_turn_dec); %
Saugaus bezier tasko salia kliutis Y koordinate
else
x_r = C_x + r_k * cos(mid_theta_1+( (3*pi) /2 ) - theta_turn_dec);
y_r = C_y - r_k * sin(mid_theta_1+ ( (3*pi) /2 ) - theta_turn_dec);
end
end
end

q_goal.coord = [x_r y_r];
q_goal.cost = 0;

% Greitosios paieskos atsitiktines tieses

nodes(1) = q_start;
figure(1)
axis([-1 abs(x_max*1.5) -1 abs(y_max*1.5)])
grid on;
axis equal;
for i = 1:1
rectangle('Position',obstacle,'FaceColor',[0 0 0.5 0.5]);
end

hold on

for i = 1:1:numNodes

    if vidinis_ziedas >= isorinis_ziedas
    q_rand = [floor(rand(1)*x_max) floor(rand(1)*y_max)];
    else
    q_rand = [floor(rand(1)*x_max) floor(rand(1)*-y_max)];
    end

plot(q_rand(1), q_rand(2), 'x', 'Color',  [0 0.4470 0.7410])

    for j = 1:1:length(nodes)
        if nodes(j).coord == q_goal.coord
```

```
            break
        end
    end

    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist(n.coord, q_rand);
        ndist = [ndist tmp];
    end
    [val, idx] = min(ndist);
    q_near = nodes(idx);

    q_new.coord = steer(q_rand, q_near.coord, val, EPS);

    if noCollision(q_rand, q_near.coord, obstacle(1,:)) && ...
            noCollision(q_rand, q_near.coord, obstacle(1,:)) && ...
            noCollision(q_rand, q_near.coord, obstacle(1,:))
        line([q_near.coord(1), q_new.coord(1)], [q_near.coord(2),
q_new.coord(2)], 'Color', 'k', 'LineWidth', 2);
        drawnow
        hold on
        q_new.cost = dist(q_new.coord, q_near.coord) + q_near.cost;

        q_nearest = [];
        r = 10;
        neighbor_count = 1;
        for j = 1:1:length(nodes)
            if noCollision(nodes(j).coord, q_new.coord, obstacle) &&
dist(nodes(j).coord, q_new.coord) <= r
                q_nearest(neighbor_count).coord = nodes(j).coord;
                q_nearest(neighbor_count).cost = nodes(j).cost;
                neighbor_count = neighbor_count+1;
            end
        end

        q_min = q_near;
        C_min = q_new.cost;
```

```
        for k = 1:1:length(q_nearest)
            if noCollision(q_nearest(k).coord, q_new.coord, obstacle) &&
q_nearest(k).cost + dist(q_nearest(k).coord, q_new.coord) < C_min
```

```
                q_min = q_nearest(k);
                C_min = q_nearest(k).cost + dist(q_nearest(k).coord,
q_new.coord);
                line([q_min.coord(1), q_new.coord(1)], [q_min.coord(2),
q_new.coord(2)], 'Color', 'g');
                hold on
            end
        end

        for j = 1:1:length(nodes)
            if nodes(j).coord == q_min.coord
                q_new.parent = j;
            end
        end

        nodes = [nodes q_new];
    end
end

D = [];
for j = 1:1:length(nodes)
    tmpdist = dist(nodes(j).coord, q_goal.coord);
    D = [D tmpdist];
end

[val, idx] = min(D);
q_final = nodes(idx);
q_goal.parent = idx;
q_end = q_goal;
nodes = [nodes q_goal];
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
nodes(start).coord(2)], 'Color', 'r', 'LineWidth', 2);
    hold on
    q_end = nodes(start);
end
```

```
IS_k_1 = (sqrt((W_k)^2 + (L_k)^2))/2;
r_k_1 = (r_1 - 2*SD - 2*IS_k_1 - (2*W_a) - vidinis_ziedas);
x_r_1 = C_x + r_k_1 * cos(mid_theta_1+( (3*pi) /2 ));
y_r_1 = C_y + r_k_1 * sin(mid_theta_1+ ( (3*pi) /2 ));
```

```
y_r_2 = C_y - r_k_1 * sin(-theta_1(Padetis_k)+ ( (3*pi) /2 ));

% Navigaciniai taskai Bezier kreivei:
if Padetis_k > V_L_3
P_0 = q_start.coord;
P_2 = [C_x_1(1), C_y_1(1)];
P_1 = [(q_start.coord(1) + ((P_2(1)-q_start.coord(1))/2)), 0];
P_3 = [C_x_1(1), q_final.coord(2)];
P_4 = [q_goal.coord(1),q_goal.coord(2)] ;
else
P_0 = q_start.coord;
P_2 = [C_x_1(1)*(2/3), C_y_1(1)];
P_1 = [(q_start.coord(1) + ((P_2(1)-q_start.coord(1))/2)), 0];
P_3 = [C_x_1(1), q_final.coord(2)];
P_4 = [q_goal.coord(1),q_goal.coord(2)] ;
end

% 5 taskas
if  theta_turn_dec == 0
if vidinis_ziedas >= isorinis_ziedas
P_5 = [C_x_1(end), q_goal.coord(2)];
else

if    C_y > 0
P_5 = [C_x_1(end), y_r_1];
else
P_5 = [C_x_1(end), y_r_2];
end
end
else

if C_y > 0
P_5 = [C_x_1(end), C_y_1(end)];
else
P_5 = [C_x_1(end) + (cos((pi/2) - theta_turn_dec) ), C_y_1(end)];
end
end


% 6 taskas
if C_y > 0
if  theta_turn_dec == 0
if vidinis_ziedas >= isorinis_ziedas
P_6 = [C_x_1(end), q_goal.coord(2)+ (S/2)];
```

```
else
P_6 = [C_x_1(end), y_r_1 + (S/2)];
end

else
P_6 = [C_x_1(end) + (cos((pi/2) - theta_turn_dec) * ((S/2))), C_y_1(end) +
(sin((pi/2) - theta_turn_dec) * ((S/2)))];
end
else

if  theta_turn_dec == 0
if vidinis_ziedas >= isorinis_ziedas
P_6 = [C_x_1(end), q_goal.coord(2)-(S/2)];
else
P_6 = [C_x_1(end), y_r_2-(S/2)];
end
else
P_6 = [C_x_1(end) + (cos((pi/2) - theta_turn_dec) * ((S/2))), C_y_1(end) -
(sin((pi/2) - theta_turn_dec) * ((S/2)))];
end
end

% 7 taskas
if C_y > 0
if  theta_turn_dec == 0
if vidinis_ziedas >= isorinis_ziedas
P_7 = [C_x_1(end), q_goal.coord(2)+S];
else
P_7 = [C_x_1(end), y_r_1+S];
end
else
P_7 = [C_x_1(end) + (cos((pi/2) - theta_turn_dec) * (S)), C_y_1(end) +
(sin((pi/2) - theta_turn_dec) * (S))];
end

else
if  theta_turn_dec == 0
    if vidinis_ziedas >= isorinis_ziedas
P_7 = [C_x_1(end), q_goal.coord(2)-S];
    else
P_7 = [C_x_1(end), y_r_2-S];
    end

else
```

```
P_7 = [C_x_1(end) + (cos((pi/2) - theta_turn_dec) * (S)), C_y_1(end) -
(sin((pi/2) - theta_turn_dec) * (S))];
end
end

% Bezier Kreive
n = 8; % 7 eiles bezier kreive
[p] = [P_0; P_1; P_2; P_3; P_4; P_5; P_6; P_7];
n1=n-1;
for    i=0:1:n1
sigma(i+1)=factorial(n1)/(factorial(i)*factorial(n1-i));
end
l=[];
UB=[];
for u=0:0.002:1
for d=1:n
UB(d)=sigma(d)*((1-u)^(n-d))*(u^(d-1));
end
l=cat(1,l,UB);
end
P=l*p;
figure(1)
line(P(:,1),P(:,2), 'color', 'b', 'LineWidth', 3)
line(p(:,1),p(:,2),'color', 'k','LineStyle','--')
hold on
% Bezier kreives greiciai ir pagreiciai:
u = 0:0.002:1; % Laikas pagal Bezier metoda isvestiniu skaiciavimui

% Bezier tikrinimas:
B_X = (P_0(1).*(1-u).^7) + (7.*u.*P_1(1).*(1-u).^6) +
(21.*(u.^2).*P_2(1).*(1-u).^5) + (35.*(u.^3).*P_3(1).*(1-u).^4) +
(35.*(u.^4).*P_4(1).*(1-u).^3) + (21.*(u.^5).*P_5(1).*(1-u).^2) +
(7.*(u.^6).*P_6(1).*(1-u)) + (u.^7.*P_7(1));
B_Y = (P_0(2).*(1-u).^7) + (7.*u.*P_1(2).*(1-u).^6) +
(21.*(u.^2).*P_2(2).*(1-u).^5) + (35.*(u.^3).*P_3(2).*(1-u).^4) +
(35.*(u.^4).*P_4(2).*(1-u).^3) + (21.*(u.^5).*P_5(2).*(1-u).^2) +
(7.*(u.^6).*P_6(2).*(1-u)) + (u.^7.*P_7(2));


% Bezier kreives greiciai, pirma isvestine:
P_vel_x = (7.*((1-u).^6).*(P_1(1) - P_0(1))) + (42.*u.*((1-u).^5).*(P_2(1)
- P_1(1))) + (105.*(u.^2).*((1-u).^4).*(P_3(1) - P_2(1))) +
(140.*(u.^3).*((1-u).^3).*(P_4(1) - P_3(1))) + (105.*(u.^4).*((1-
u).^2).*(P_5(1) - P_4(1))) + (42.*(u.^5).*((1-u)).*(P_6(1) - P_5(1))) +
```

```
(7.*(u.^6).*(P_7(1) - P_6(1)));
P_vel_y = (7.*((1-u).^6).*(P_1(2) - P_0(2))) + (42.*u.*((1-u).^5).*(P_2(2)
- P_1(2))) + (105.*(u.^2).*((1-u).^4).*(P_3(2) - P_2(2))) +
(140.*(u.^3).*((1-u).^3).*(P_4(2) - P_3(2))) + (105.*(u.^4).*((1-
u).^2).*(P_5(2) - P_4(2))) + (42.*(u.^5).*((1-u)).*(P_6(2) - P_5(2))) +
(7.*(u.^6).*(P_7(2) - P_6(2)));

% Bezier kreives pagreiciai, antra isvestine:
P_acc_x = (42.*((1-u).^5).*(P_2(1) - 2.*P_1(1) + P_0(1))) + (210.*u.*((1-
u).^4).*(P_3(1) - 2.*P_2(1) + P_1(1))) + (420.*(u.^2).*((1-u).^3).*(P_4(1)
- 2.*P_3(1) + P_2(1))) + (420.*(u.^3).*((1-u).^2).*(P_5(1) - 2.*P_4(1) +
P_3(1))) + (210.*(u.^4).*((1-u)).*(P_6(1) - 2.*P_5(1) + P_4(1))) +
(42.*(u.^5).*(P_7(1) - 2.*P_6(1) + P_5(1)));
P_acc_y = (42.*((1-u).^5).*(P_2(2) - 2.*P_1(2) + P_0(2))) + (210.*u.*((1-
u).^4).*(P_3(2) - 2.*P_2(2) + P_1(2))) + (420.*(u.^2).*((1-u).^3).*(P_4(2)
- 2.*P_3(2) + P_2(2))) + (420.*(u.^3).*((1-u).^2).*(P_5(2) - 2.*P_4(2) +
P_3(2))) + (210.*(u.^4).*((1-u)).*(P_6(2) - 2.*P_5(2) + P_4(2))) +
(42.*(u.^5).*(P_7(2) - 2.*P_6(2) + P_5(2)));

% Trajektorijos kreivis:
K = ((P_vel_x .* P_acc_y) - (P_vel_y .* P_acc_x)) ./ ( (P_vel_x).^2 +
(P_vel_y).^2 ).^1.5 ;

% Trajektorijos spindulys:
R = 1 ./ K;

% Vairuojamuju ratu pasukimo kampas:
d_1 = (atan((l_f+l_g) ./ R));

P_X_values = P(:,1); P_Y_values = P(:,2); % Pakartojamos vektoriaus
reiksmes

difference = sqrt(((P_X_values(2:end) - P_X_values(1:end-1)).^2) +
((P_Y_values(2:end) - P_Y_values(1:end-1)).^2)); % Skaiciuojamas atstumas
tarp tasku
ARC_L = sum(difference); % Bendras Bezier kreives ilgis

difference_time = difference ./ V_a ; % Suskaiciuojamas kiekvienas laiko
zingsnis atskirai
time_f = [0; difference_time]; % Galutinis laiko zingsniu vektorius
t_sum = cumsum(time_f);
steer = transpose(d_1);
S_sum = (t_sum * V_a)+6;
sig1 = [t_sum steer];
```

```
% Patikrinimas:
Bezier_T = ARC_L * Kreives_p; % Atstumas, kuri nuvaziavus reikia tikrinti
ar nepazeidzia salygos
B_sum = cumsum(difference);
Bezier_T_1 = find(B_sum >= (Bezier_T-2));

if C_y > 0
if vidinis_ziedas >= isorinis_ziedas
X_K_1 = obstacle(1); % Kliuties kampo X koordinate
Y_K_1 = obstacle(2) + obstacle(4); % Kliuties kampo y koordinate
else
X_K_1 = obstacle(1)+obstacle(3); % Kliuties kampo X koordinate
Y_K_1 = obstacle(2); % Kliuties kampo y koordinate
end
else
if vidinis_ziedas >= isorinis_ziedas
X_K_1 = obstacle(1); % Kliuties kampo X koordinate
Y_K_1 = obstacle(2); % Kliuties kampo y koordinate
else
X_K_1 = obstacle(1)+obstacle(3); % Kliuties kampo X koordinate
Y_K_1 = obstacle(2)+obstacle(4); % Kliuties kampo y koordinate
end
end

B_T_X = P_X_values(Bezier_T_1(1)-1); B_T_Y = P_Y_values(Bezier_T_1(1)-1);
% Tikrinimo tasko koordinates

Salyga_2 = sqrt(((B_T_X - X_K_1).^2) + ((B_T_Y - Y_K_1).^2)); % Atstumas
nuvaziavus tokia pat kreiviu dali tarp automobilio centro ir kliuties
```