

```

1 #include "../include/PurePursuitFollower.h"
2 #include "PurePursuitPathGen.h"
3
4 #include <vector>
5 #include <string>
6 #include <fstream>
7 #include <iostream>
8 #include <algorithm>
9 #include <math.h>
10 #include <array>
11
12 #define PI 3.14159265
13
14 PurePursuitFollower::PurePursuitFollower(double lookahead) {
15     this->lookahead = lookahead;
16     this->prev_time = timer.millis().convert(second);
17 }
18
19 void PurePursuitFollower::read_from_file(std::string filename) {
20     std::ifstream fin;
21     fin.open(filename);
22     points.clear();
23     followPoint temp;
24     while (!fin.eof()) {
25         fin >> temp.x >> temp.y >> temp.vel;
26         points.push_back(temp);
27     }
28 }
29
30 void PurePursuitFollower::read(PurePursuitPathGen obj) {
31     std::vector<point> temppoints;
32     temppoints = obj.get_points();
33     followPoint temp;
34     printf("READING\n");
35     for(int i = 0; i < temppoints.size(); i++) {
36         temp.x = temppoints[i].x;
37         temp.y = temppoints[i].y;
38         temp.vel = temppoints[i].vel;
39         points.push_back(temp);
40     }
41     for(followPoint x: points) {
42         printf("%f %f %f\n", x.x, x.y, x.vel);
43     }
44 }
45
46
47 void PurePursuitFollower::calc_closest_point(double x, double y) {
48     double min = 1E7;
49     for (int i = last_closest_point; i < points.size(); i++) {
50         double dist = sqrt(((x - points[i].x) * (x - points[i].x)) + ((y - points[i].y) * (y -
points[i].y)));
51         if (dist < min) {
52             min = dist;
53             last_closest_point = i;
54             closest_point = points[i];
55         }
56     }

```

```

57 }
58
59 void PurePursuitFollower::calc_lookahead(double x, double y) {
60     std::pair<double, double> d, f;
61     double a, b, c, discriminant, t1, t2;
62     for (int i = last_closest_point + 1; i < points.size(); i++) {
63         d.first = points[i].x - points[i - 1].x;
64         d.second = points[i].y - points[i - 1].y;
65         f.first = points[i].x - x;
66         f.second = points[i].y - y;
67         a = d.first * d.first + d.second * d.second;
68         b = 2 * (f.first * d.first + f.second * d.second);
69         c = (f.first * f.first + f.second * f.second) - lookahead * lookahead;
70         discriminant = b * b - (4 * a * c);
71         if (discriminant >= 0) {
72             discriminant = sqrt(discriminant);
73             t1 = (-b - discriminant) / (2 * a);
74             t2 = (-b + discriminant) / (2 * a);
75             if (t1 >= 0 && t1 <= 1 && t1 + i - 1 > last_fractional_index) {
76                 lookahead_point.first = points[i - 1].x + (t1 * d.first);
77                 lookahead_point.second = points[i - 1].y + (t1 * d.second);
78                 last_lookahead_point = lookahead_point;
79                 break;
80             }
81             if (t2 >= 0 && t2 <= 1 && t2 + i - 1 > last_fractional_index) {
82                 lookahead_point.first = points[i - 1].x + (t2 * d.first);
83                 lookahead_point.second = points[i - 1].y + (t2 * d.second);
84                 last_lookahead_point = lookahead_point;
85                 break;
86             }
87         }
88     }
89     lookahead_point = last_lookahead_point;
90     last_lookahead_point = lookahead_point;
91 }
92
93 void PurePursuitFollower::calc_curvature_at_point(double x, double y, double theta) {
94     double xtemp;
95     double a, b, c;
96     a = -tan((theta));
97     b = 1;
98     c = (tan((theta)) * x) - y;
99     double temp = (sin((theta)) * (lookahead_point.first - x)) - (cos((theta)) *
100 (lookahead_point.second - y));
101     int sign = (temp > 0) ? 1 : ((temp < 0) ? -1 : 0);
102     xtemp = abs((a * lookahead_point.first) + (b * lookahead_point.second) + c) / sqrt((a * a
103 + (b * b)));
104     this->curvature = ((2 * xtemp) / (lookahead * lookahead));
105     this->curvature *= sign;
106 }
107
108 std::array<double, 4> PurePursuitFollower::follow_sim(double x, double y, double theta) {
109     calc_closest_point(x, y);
110     calc_lookahead(x, y);
111     calc_curvature_at_point(x, y, theta);
112     std::array<double, 4> vels;
113     if (closest_point.x == points[points.size() - 1].x && closest_point.y ==
114 points[points.size() - 1].y) {

```

```

112         vels[0] = 0;
113         vels[1] = 0;
114         vels[2] = 0;
115         vels[3] = 0;
116         return vels;
117     }
118     vels[0] = closest_point.vel;
119     vels[1] = vels[0] * curvature;
120     vels[2] = 0;
121     vels[3] = 0;
122     return vels;
123 }
124
125 std::array<double, 4> PurePursuitFollower::follow(double x, double y, double theta) {
126     calc_closest_point(x, y);
127     calc_lookahead(x, y);
128     calc_curvature_at_point(x, y, theta);
129     std::array<double, 4> vels;
130     if (closest_point.x == points[points.size() - 1].x && closest_point.y ==
points[points.size() - 1].y) {
131         vels[0] = 0;
132         vels[1] = 0;
133         vels[2] = 0;
134         vels[3] = 0;
135         return vels;
136     }
137     double time = (timer.millis().convert(second)-prev_time);
138     double vel, ang;
139     vel = (closest_point.vel);
140     vel = prev_vel+(std::clamp(vel-prev_vel, -(time*max_accel), (time*max_accel)));
141     printf("vel: %f curvature: %f\n", vel, curvature);
142     vel = vel/10;
143     ang = vel*curvature;
144     ang = ang/(10/(2*PI));
145     vels[0] = vel;
146     vels[1] = ang;
147     vels[2] = vel;
148     vels[3] = ang;
149     prev_time = timer.millis().convert(second);
150     prev_vel = vel;
151     return vels;
152 }

```