```cpp
#include "../include/PurePursuitFollower.h"
#include "PurePursuitPathGen.h"

#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <math.h>
#include <array>

#define PI 3.14159265

PurePursuitFollower::PurePursuitFollower(double lookahead) {
    this->lookahead = lookahead;
    this->prev_time = timer.millis().convert(second);
}

void PurePursuitFollower::read_from_file(std::string filename) {
    std::ifstream fin;
    fin.open(filename);
    points.clear();
    followPoint temp;
    while (!fin.eof()) {
        fin >> temp.x >> temp.y >> temp.vel;
        points.push_back(temp);
    }
}

void PurePursuitFollower::read(PurePursuitPathGen obj) {
    std::vector<point> temppoints;
    temppoints = obj.get_points();
    followPoint temp;
    printf("READING\n");
    for(int i = 0; i < temppoints.size(); i++) {
        temp.x = temppoints[i].x;
        temp.y = temppoints[i].y;
        temp.vel = temppoints[i].vel;
        points.push_back(temp);
    }
    for(followPoint x: points) {
        printf("%f %f %f\n", x.x, x.y, x.vel);
    }
}


void PurePursuitFollower::calc_closest_point(double x, double y) {
    double min = 1E7;
    for (int i = last_closest_point; i < points.size(); i++) {
        double dist = sqrt(((x - points[i].x) * (x - points[i].x)) + ((y - points[i].y) * (y - points[i].y)));
        if (dist < min) {
            min = dist;
            last_closest_point = i;
            closest_point = points[i];
        }
    }
```

```
}

void PurePursuitFollower::calc_lookahead(double x, double y) {
    std::pair<double, double> d, f;
    double a, b, c, discriminant, t1, t2;
    for (int i = last_closest_point + 1; i < points.size(); i++) {
        d.first = points[i].x - points[i - 1].x;
        d.second = points[i].y - points[i - 1].y;
        f.first = points[i].x - x;
        f.second = points[i].y - y;
        a = d.first * d.first + d.second * d.second;
        b = 2 * (f.first * d.first + f.second * d.second);
        c = (f.first * f.first + f.second * f.second) - lookahead * lookahead;
        discriminant = b * b - (4 * a * c);
        if (discriminant >= 0) {
            discriminant = sqrt(discriminant);
            t1 = (-b - discriminant) / (2 * a);
            t2 = (-b + discriminant) / (2 * a);
            if (t1 >= 0 && t1 <= 1 && t1 + i - 1 > last_fractional_index) {
                lookahead_point.first = points[i - 1].x + (t1 * d.first);
                lookahead_point.second = points[i - 1].y + (t1 * d.second);
                last_lookahead_point = lookahead_point;
                break;
            }
            if (t2 >= 0 && t2 <= 1 && t2 + i - 1 > last_fractional_index) {
                lookahead_point.first = points[i - 1].x + (t2 * d.first);
                lookahead_point.second = points[i - 1].y + (t2 * d.second);
                last_lookahead_point = lookahead_point;
                break;
            }
        }
    }
    lookahead_point = last_lookahead_point;
    last_lookahead_point = lookahead_point;
}

void PurePursuitFollower::calc_curvature_at_point(double x, double y, double theta) {
    double xtemp;
    double a, b, c;
    a = -tan((theta));
    b = 1;
    c = (tan((theta)) * x) - y;
    double temp = (sin((theta)) * (lookahead_point.first - x)) - (cos((theta)) *
(lookahead_point.second - y));
    int sign = (temp > 0) ? 1 : ((temp < 0) ? -1 : 0);
    xtemp = abs((a * lookahead_point.first) + (b * lookahead_point.second) + c) / sqrt((a * a)
+ (b * b));
    this->curvature = ((2 * xtemp) / (lookahead * lookahead));
    this->curvature *= sign;
}

std::array<double, 4> PurePursuitFollower::follow_sim(double x, double y, double theta) {
    calc_closest_point(x, y);
    calc_lookahead(x, y);
    calc_curvature_at_point(x, y, theta);
    std::array<double, 4> vels;
    if (closest_point.x == points[points.size() - 1].x && closest_point.y ==
points[points.size() - 1].y) {
```

```
        vels[0] = 0;
        vels[1] = 0;
        vels[2] = 0;
        vels[3] = 0;
        return vels;
    }
    vels[0] = closest_point.vel;
    vels[1] = vels[0] * curvature;
    vels[2] = 0;
    vels[3] = 0;
    return vels;
}

std::array<double, 4> PurePursuitFollower::follow(double x, double y, double theta) {
    calc_closest_point(x, y);
    calc_lookahead(x, y);
    calc_curvature_at_point(x, y, theta);
    std::array<double, 4> vels;
    if (closest_point.x == points[points.size() - 1].x && closest_point.y ==
points[points.size() - 1].y) {
        vels[0] = 0;
        vels[1] = 0;
        vels[2] = 0;
        vels[3] = 0;
        return vels;
    }
    double time = (timer.millis().convert(second)-prev_time);
    double vel, ang;
    vel = (closest_point.vel);
    vel = prev_vel+(std::clamp(vel-prev_vel, -(time*max_accel), (time*max_accel)));
    printf("vel: %f curvature: %f\n", vel, curvature);
    vel = vel/10;
    ang = vel*curvature;
    ang = ang/(10/(2*PI));
    vels[0] = vel;
    vels[1] = ang;
    vels[2] = vel;
    vels[3] = ang;
    prev_time = timer.millis().convert(second);
    prev_vel = vel;
    return vels;
}
```