Metrics for Performance Benchmarking of Kinematic Algorithms

Tunwu Li
SID: 2253543

Xianzuo Du
SID: 2235587

December 13, 2022

Abstract——Performance benchmarking is an effective way to compete solutions under different experimental conditions. In this paper, we present four software-independent metrics to quantitatively compare the performance of two kinematic algorithms by following the same path. We validate the ability of the metrics to reflect the performance of the kinematic algorithms through variable-based experiments.

I. Introduction

Accurate control of a robot's motion is dependent on the establishment of a kinematic model. The accuracy of the model, i.e. the accuracy of the description of a robot's motion, is the basis for tracking and controlling a robot. Therefore, a competent kinematic model should have properties that truly reflect the position and orientation of a robot. Furthermore, taking into account the kinematic properties of a robot during the tracking control phase helps generate an optimal trajectory that satisfies safety constraints.

A. Motivation

However, the overabundance of options when it comes to choosing a suitable kinematic algorithm for collecting features of a path creates a certain amount of stress [1]. Researchers experiment with their ideas under different conditions, making it difficult to compare the results presented in different papers, and replication is even more difficult. Also, there are few accepted standards for measuring kinematics, which prompted us to create a set of metrics.

B. Aim & Hypothesis Statement

Our aim is to develop a set of performance benchmarks for evaluating and quantitatively comparing the performance of two types of kinematic algorithms. We propose four metrics: the trend of the rotation angle over time, the error of the robot's stop coordinates, the quality of the map, and the error of the smoothness of the trajectory. With the above metrics, we hope to attain the following hypotheses:

- The metrics reflect the features of the route travelled by the robot through line following, such as the angle of the route turn, the figure enclosed by the route and its area, and the distance from the starting point where the robot stops.
- The metrics allow a quantitative assessment of the differences between the two kinematics, for example, the amount of error between the two groups of measurements and the true value and the magnitude of the error.

II. Implementation

A. Pre-processing of line sensors' data

After reading the values of the left sensor, the middle sensor and the right sensor (w_0, w_1, w_2), we proceed as follows:

- Normalisation. It allows the difference between the left and right sensors to be limited to the range $[-1, 1]$, thus eliminating the undesirable effect of odd data. The equation for normalisation is expressed as follow [2]:

$$w_i = \frac{w_i}{\sum_{i=0}^{2} w_i} \tag{1}$$

- First-order low-pass filtering of the error signal e_{line}:

$$ave_{e_{line}}(t) = \alpha e_{line} + (1-\alpha) ave_{e_{line}}(t-1) \tag{2}$$

  Where $ave_{e_{line}}(t)$ is the filtered output value, weighted by the sampled value at this time and the filtered output value at the previous time. α is the filtering coefficient, here we take 0.3.

B. Kinematic algorithms

We achieve the trajectory projection by reading the left and right wheel encoders. The distance between

wheels $2l = 87\ mm$, the wheel radius $r = 16\ mm$ [3] and the position of the robot at time $t-1$ are known. The position of the robot at time t is then derived from these parameters.

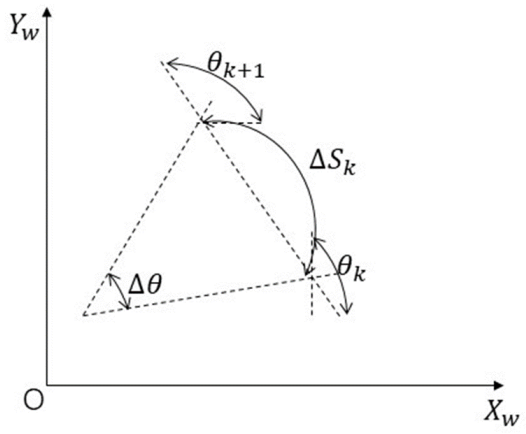


Figure 1: Kinematic modelling

In sampling interval, if the speeds of the left and right wheels are the same, the robot will theoretically travel an arc trajectory. As shown in Figure 1, the forward direction is the positive direction of the X-axis and $\triangle\theta$ is the rotation angle. The length of the arc travelled by the robot, $\triangle S_k$, can be determined by summing the speeds of the left and right wheels.

A number of models can be deduced based on this. We compare two of them, the tangent model and the secant model:

1. Tangent model. The most basic model, assuming that the robot advances $\triangle S_k$ in a straight line in the direction of the previous moment t and then rotates $\triangle\theta$ degree. Its expressions are [4]:

$$\begin{cases} x_{k+1} = x_k + \triangle S_k\ cos\theta_k \\ y_{k+1} = y_k + \triangle S_k\ sin\theta_k \\ \theta_{k+1} = \theta_k + \triangle\theta_k \end{cases} \tag{3}$$

2. Secant model. It assumes that the robot travels along the cut line of the arc, first by turning $\frac{\triangle\theta}{2}$, then going $\triangle S_k$ distance in this direction, and finally by turning $\frac{\triangle\theta}{2}$ again. Its expressions are:

$$\begin{cases} x_{k+1} = x_k + \triangle S_k\ cos(\theta_k + \frac{\triangle\theta_k}{2}) \\ y_{k+1} = y_k + \triangle S_k\ sin(\theta_k + \frac{\triangle\theta_k}{2}) \\ \theta_{k+1} = \theta_k + \triangle\theta_k \end{cases} \tag{4}$$

C. PD for heading control

Proportional derivative (PD) control is a closed-loop control approach consisting of a proportional unit and a differential unit [5]. The combination of a proportional controller and a derivative controller provides accurate control because the overshoot caused by the proportional controller is eliminated by the derivative controller and the steady state error of the derivative controller is reduced by the proportional controller.

When following a line, we expect a smooth trajectory, so we adopt PD control to confine the steering speed to an acceptable range. The flow of the control loop is shown in Figure 2.

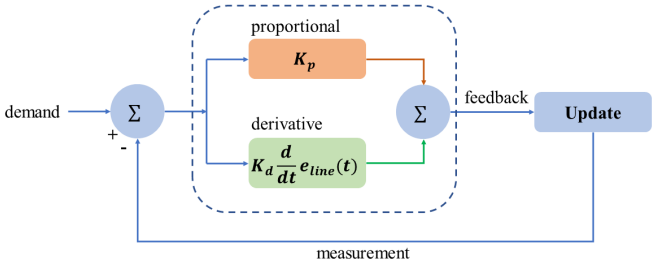


Figure 2: PD control flow diagram

The error between the demand and the real-time measured value of the sensor is first transmitted to the controller. The controller then processes the value and feeds it to the actuator. Finally the actuator (motors) responds. The difference between the left and right sensor is e_{line} and the desired value $demand = 0$, then:

$$demand = 0 = K_p e_{line}(t) + K_d \frac{d}{dt} e_{line}(t) \tag{5}$$

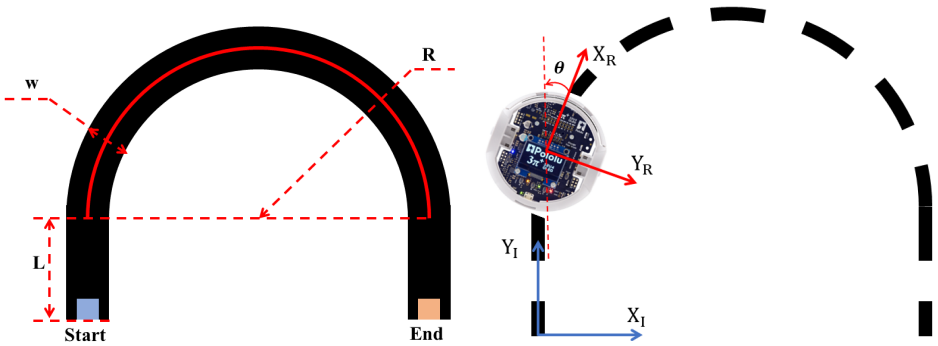
Where K_p and K_d are the proportional gain of the error and the proportional gain of the derivative of the error, respectively. As PD control is not the main subject of this paper, the process of debugging the gains is not described here, but only the debugged values are given of K_p and K_d are -12 and -9.

D. Map for line following

Figure 3 illustrates the map and the layout of the frames. The diameter of the semicircular arc $R = 93.35\ mm$, the width of the semicircular arc $w = 19\ mm$ and the length of the straight line segment $L = 62.5\ mm$. The blue and yellow squares denote the position where the robot starts and finishes line following respectively.

The reason for designing the combination of two straight segments and a semi-circular arc is that the straight and circular segments evaluate the kinematics describing the position and the rotation respectively. Moreover, its reference dimensions are computable.

In Figure 3(b), we define X_I, Y_I to represent the global frame, X_R, Y_R as the local frame of the robot's centre point and θ as the rotation angle. Once the robot is reset, the position is the origin $(0, 0)$, from which the subsequent position is calculated. Therefore, in theory,



(a) Map (b) Layout of frames

Figure 3: Map for line following and layout of frames (`https://www.pololu.com/docs/0J83`)——(a) map and (b) layout of frames

the Y-coordinate Y_{end} of the end position is 0 and the X-coordinate is calculated as below, so the coordinates of the end position should be $(167.7, 0)$.

$$X_{end} = 2R - w = 186.7 - 19 = 167.7\ mm \quad (6)$$

Our pre-processing is summarised by the following pseudo-codes (Algorithm 1).

Algorithm 1

Input: demand
*i*nitialization: Line sensors, motors, encoders, and PD controller
*s*tate = 0

while state == 0 **do**
  record θ, X and Y every 50 ms
  obtain e_{line}
  $ave_{e_{line}}(t) \leftarrow 0.3 \times e_{line} + 0.7 \times ave_{e_{line}}(t-1)$
  steering speed $\leftarrow$ PD controller$(demand, e_{line})$
  motors.follow line(steering speed)
  update θ, X and Y by kinematics
  if all sensors detect white **then**
    state = 1
  end if
end while
return θ, X **and** Y
while state == 1 **do**
  print θ, X and Y
end while

III. Experiment Methodology

We apply variable-based approach to our experiments, where we control the factors that influence the patterns in the position and orientation of $3Pi+$, retaining kinematic as the only independent variable.

A. Experimental procedure

The experimental steps are as follows:

1. Print the map on a piece of A3 paper.
2. Draw the start line to ensure that $3Pi+$ starts at the same position each time.
3. Repeat the test ten times with the tangent model, recording data once for each test.
4. Upload the codes for the secant model and repeat the test ten times, recording data once for each test.

B. Variables

1) Controlled variables

1. Map for line following: combination of a semicircular arc with a radius of 93.35 mm and two straight segments with a length of 62.5 mm.
2. Steering speed: limited by PD control to achieve a smooth trajectory.
3. Time interval for data collection: 50 ms.
4. Ambient light: each test is conducted under the same level of light to avoid bias caused by infrared detection.

2) Independent variable

The only independent variable is the kinematic algorithm: tangent or secant. The aim of this paper is to contrast them quantitatively.

3) Dependent variables

The data collected during the robot's line following process, including:

1. Rotation angles θ.
2. X-coordinates.
3. Y-coordinates.

C. Metrics

We design four metrics to evaluate the performance of the kinematics, in order of priority as follows.

1) Rotation angle over time

The rotation angle θ is a prerequisite for estimating the position of the robot. Therefore, we consider the trend of theta over time t to be the No. 1 metric since a reasonable rotation angle leads to a reliable position estimate.

2) Error of stop coordinates

We adopt the sum of the absolute values of the errors between the X and Y coordinates recorded by the kinematic models and the ground truths $(167.7, 0)$ because it indicates the cumulative error in recording the end point of a path relative to the start point, as calculated by:

$$Err_{cor} = |X_{end} - 167.7| + |Y_{end} - 0| \ mm \tag{7}$$

3) Map completeness

To identify the gap, the area of the plotted graph needs to be matched to the actual area. We define this metric as the ratio of the area M enclosed by the recorded coordinates to the area P of the ground truth [1]:

$$Completeness = \frac{M}{P} \tag{8}$$

4) Bending energy of the circular segment

Bending energy (B_E) is the energy required to bend a straight rod into the desired shapes [6]. We discard the coordinates of the straight segment and calculate the B_E of the trajectory by fitting a binomial curve to the arc segment. The curvature k is first calculated for any point on the trajectory:

$$k(X_i, f(X_i)) = \frac{f''(X_i)}{(1 + (f'(X_i))^2)^{\frac{3}{2}}} \tag{9}$$

Next, B_E is equal to the sum of the squares of the curvatures at each point:

$$B_E = \frac{1}{n} \sum_{i=1}^{n} k^2(X_i, f(X_i)) \tag{10}$$

After obtaining the $B_E s$, we compare them with the B_E of the standard circular segment, the one with the smaller error being the more accurate description.

D. Analysis of data

We experiment with ten sets of each kinematic model (twenty sets in total), obtaining θ, X and Y every 50 ms. First, we obtain the error in the rotation angle by subtracting the ground truth from θ at each moment returned by each of the two models. We consider the last coordinates of each data set as the stopping position of the robot, and its error is equal to the sum of the errors of the X and Y coordinates. Next, the regions enclosed by the X and Y coordinates is plotted, calculating the areas and matching it to the real area of the map. At last, we fit the trajectory of the arc segment by binomials and then sample 20 points to calculate the curvatures and bending energy.

The above four steps correspond to the four metrics we have designed, as already mentioned in the previous subsection. A more detailed analysis of data is demonstrated in Section IV.

IV. Results

A. Rotation angle over time

Note that to visualise the trend in the angle of rotation measured by the two models, we plot the absolute value of θ (Figure 4). The blue and orange curves show the trend of $|\theta|$ over time t measured with the tangent and secant models respectively, with the length of the error bars denoting the magnitude of the error between the measured and true angles, the longer the error bars, the larger the error.

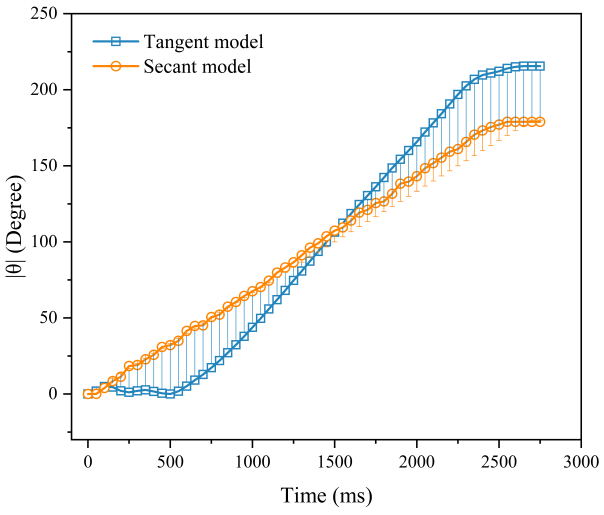


Figure 4: Absolute values of rotation angle over time

Firstly, both models properly assess the overall trend in the absolute value of the rotation angle, increasing from 0 to around 180°. The secant model's measurements were closer to the true value with a maximum error of 13.79° (7.67%), which occurred at

$t = 2350\ ms$. In comparison, the maximum error for the tangent model was 17.43° (9.68%) and the sum of the errors for the secant model was 194.92 less than the tangent model. As a result, the secant model is 40% more accurate than the tangent model in measuring the angle of rotation.

B. Error of stop coordinates

The sums of the errors in the X and Y coordinates measured by the two models over the ten experiments are summarised in Table 1. The average coordinates of the end positions measured by the tangent model are $(170.59, -38.68)$. The X-coordinate differs from the actual value (167.7) by only 2.8 mm (1.67%). However, the error in the Y-coordinate is 38.68 mm, the sum of which is 41.48 mm.

In contrast, the average coordinates of the end positions measured by the secant model are $(182.62, 0.83)$. The difference between the Y-coordinate and the actual value (0) is only 0.83 mm, while the error in the X-coordinate amounts to 14.92 mm (8.90%) and the sum of the errors is 15.75 mm. Thus, the secant model is 62.46% more accurate than the tangent model for the measurement of stop coordinates.

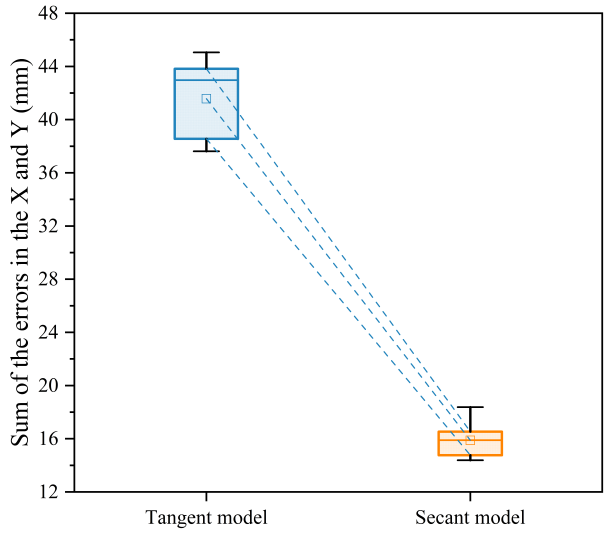


Figure 5: Means and distributions of the stop positions

After that, we apply a boxplot to represent the distribution of the two sets of data. The blue and orange boxes represent the error measured by the tangent model and the secant model respectively. The small square in the centre of the box denotes the mean, the line in the middle represents the median, and the height of the box indicates the fluctuation. As can be seen from Figure 5, the secant model not only has a smaller error, but is also more stable than the tangent model.

C. Map completeness

The regions enclosed by the coordinates recorded by the tangent model and the secant model are shown in Figure 6. We apply the vector method in Python (see Appendix for codes) to calculate the area of these two regions as $15,970\ mm^2$ and $15,381\ mm^2$. The real area of the map is:

$$P = 2LR + \frac{\pi(R - \frac{w}{2})^2}{2} = 21,525\ mm^2 \tag{11}$$

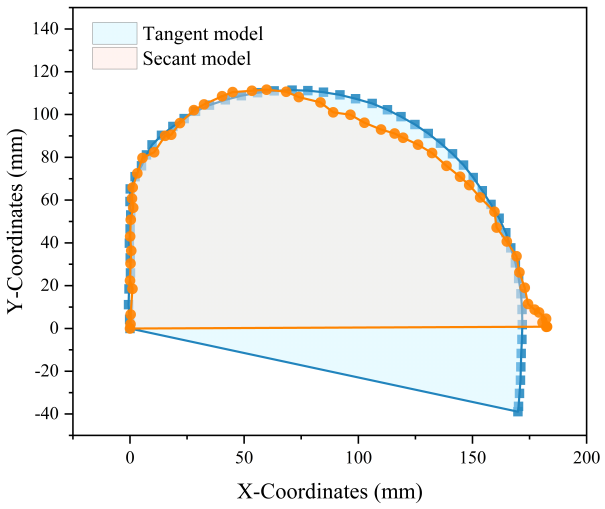


Figure 6: Regions enclosed by recored coordinates

So, the map completeness for the tangent and secant models are 74.19% and 71.46% from Equation 8. The main errors stem from the poor performance in measuring the distance in the Y direction, with variances of about 43 mm (28.14%).

D. Bending energy of the circular segment

We discard the coordinates of the straight line segments and fit only the coordinates of the arc segments, for a segmentation function cannot be formulated in one polynomial. First, a scatter is drawn through the recorded coordinates and we fit them in binomials. The expressions are shown below:

- tangent model:

$$Y_I = -0.0081X_I^2 + 1.18X_I + 71.83 \tag{12}$$

- secant model:

$$Y_I = -0.0076X_I^2 + 1.05X_I + 73.70 \tag{13}$$

The fitted curves are shown in Figure 7, with the blue and orange curves representing the tangent and secant models respectively.

Afterwards, we sample twenty X-coordinates uniformly distributed in the interval $[2, 170]$ and compute

Table 1: Sum of the errors in the X and Y coordinates from the two models

	Trail 1	Trail 2	Trail 3	Trail 4	Trail 5	Trail 6	Trail 7	Trail 8	Trail 9	Trail 10
Tangent model	38.56	43.46	43.11	38.26	37.62	44.11	42.82	43.82	38.9	45.06
Secant model	14.72	16.53	16.32	18.38	14.38	15.77	15.25	16.7	14.76	16.01

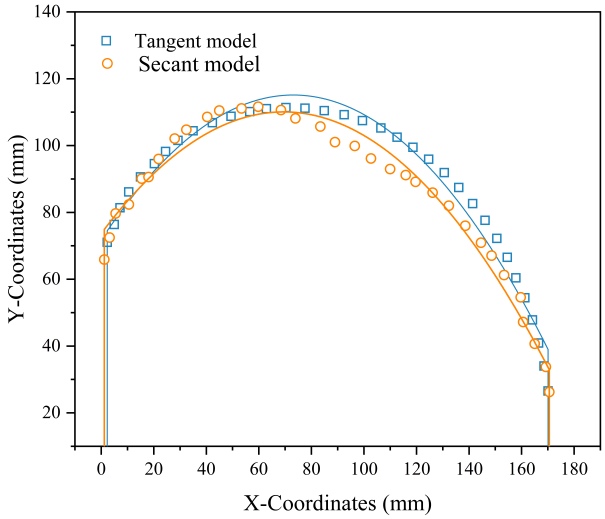


Figure 7: Scatter plot of the circular segments and the fitted curves

the fitted curves of the two models from Equation 9 and Equation 10 for the B_Es in this interval as -9.82×10^{-6} and -1.72×10^{-5}. The B_E of the arc in the map is:

$$B_{E_{map}} = (\frac{1}{R})^2 = 1.15\times10^{-4} \tag{14}$$

Hence, the errors on the B_Es returned by the two models are -9.82×10^{-6} and -1.73×10^{-5} respectively, implying that the tangent model is 43.16% more accurate than the secant model in measuring the circular segment.

V. Discussion

Firstly, the four metrics we proposed substantially reflect the features of the map in the experiment, except for the map completeness due to the deviation in Y distance (25%). The other three metrics accurately describe the angle the path turned, the dimension in X, and the curvature and B_E of the circular segment.

The secant model is superior in measuring the rotation angle and stop position, being 40% and 62.46% more accurate than the tangent model, respectively, and the secant model returns data with less variance in the stop positions. However, the tangent model has a better map completeness and a better fit to the circular segment.

Based on the ranking of the influence of the metrics, we believe that the secant model is preferable because it has a noticeable advantage in rotation angle and stop position, while a slight inferiority in map completeness and B_E.

Both models are flawed in their estimation of the distance in the Y, with errors of around 43 mm (28.14%). We suspect that the reason for it derives from the dependence of the coordinates on the rotation angle and the excessive time interval for collecting the data. 50 ms interval is a reluctance, limited by the storage space of $3Pi+$.

One of the future research directions is to embrace advanced kinematics, such as the arc model, which may measure Y-direction distances precisely. In this paper only one PD was adopted to manage the steering of the $3Pi+$, in the future we will add two PD controllers to maintain a fixed speed for the left and right wheels.

VI. Conclusion

In this paper, we proposed a set of metrics to quantitatively compare kinematic algorithms: trend in rotation angle over time, error in stop coordinates, map completeness, and bending energy of circular segments. We tested our metrics against the tangent and secant models based on the Polou 3Pi+ robot and found that our metrics reflect the disparity in performance of the two models when following the same path and the secant model characterises the map more accurately than the tangent model in this paper's scenario, dominating the top two ranked metrics.

References

[1] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, "Metrics for performance benchmarking of multi-robot exploration," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3407–3414.

[2] P. O'Dowd, "Labsheet 4: Line Following," 2022.

[3] Pololu Corporation, *Pololu 3pi+ 32U4 User' s Guide*, 2022. [Online]. Available: https://www.pololu.com/

[4] P. O'Dowd, "Labsheet 6: Odometry," 2022.

[5] ——, "Labsheet 7: PID Control," 2022.

[6] N. D. M. Ceballos, J. A. Valencia, N. L. Ospina, and A. Barrera, *Quantitative performance metrics for mobile robots navigation.* INTECH Open Access Publisher London, UK, 2010.

Appendix

A. *Python codes for vector method*

```python
import math

class Point():
    def __init__(self,x,y):
        self.x = x
        self.y = y

def GetAreaOfPolyGon(points):

    area = 0
    if(len(points)<3):
         raise Exception("error")

    p1 = points[0]
    for i in range(1,len(points)-1):
        p2 = points[i]
        p3 = points[i + 1]

        vecp1p2 = Point(p2.x-p1.x,p2.y-p1.y)
        vecp2p3 = Point(p3.x-p2.x,p3.y-p2.y)

        vecMult = vecp1p2.x*vecp2p3.y - vecp1p2.y*vecp2p3.x
        sign = 0
        if(vecMult>0):
            sign = 1
        elif(vecMult<0):
            sign = -1

        triArea = GetAreaOfTriangle(p1,p2,p3)*sign
        area += triArea
    return abs(area)

def GetAreaOfTriangle(p1,p2,p3):
    area = 0
    p1p2 = GetLineLength(p1,p2)
    p2p3 = GetLineLength(p2,p3)
    p3p1 = GetLineLength(p3,p1)
    s = (p1p2 + p2p3 + p3p1)/2
    area = s*(s-p1p2)*(s-p2p3)*(s-p3p1)
    area = math.sqrt(area)
    return area

def GetLineLength(p1,p2):
    length = math.pow((p1.x-p2.x),2) + math.pow((p1.y-p2.y),2)
    length = math.sqrt(length)
    return length

def main():

    points = []
    x = []
    y = []

    for index in range(len(x)):
        points.append(Point(x[index],y[index]))

    area = GetAreaOfPolyGon(points)
    print(area)
    print(math.ceil(area))
    ##assert math.ceil(area)==1

if __name__ == '__main__':
    main()
    print("OK")
```